Autoconf

Creating Automatic Configuration Scripts
for version 2.59, 5 November 2003

David MacKenzie
Ben Elliston
Akim Demaille

This manual is for GNU Autoconf (version 2.59, 5 November 2003), a package for creating scripts to configure source code packages using templates and an M4 macro package.

Table of Contents

1 Introduction 1

2 The GNU Build System 3
2.1 Automake 3
2.2 Libtool 3
2.3 Pointers 4

3 Making configure Scripts 5
3.1 Writing 'configure.ac' 6
3.1.1 A Shell Script Compiler 6
3.1.2 The Autoconf Language 7
3.1.3 Standard 'configure.ac' Layout 8
3.2 Using autoscan to Create 'configure.ac' 9
3.3 Using ifnames to List Conditionals 10
3.4 Using autoconf to Create configure 10
3.5 Using autoreconf to Update configure Scripts ... 12

4 Initialization and Output Files 15
4.1 Initializing configure 15
4.2 Notices in configure 15
4.3 Finding configure Input 16
4.4 Outputting Files 17
4.5 Performing Configuration Actions 18
4.6 Creating Configuration Files 19
4.7 Substitutions in Makefiles 20
4.7.1 Preset Output Variables 20
4.7.2 Installation Directory Variables 22
4.7.3 Build Directories 25
4.7.4 Automatic Remaking 25
4.8 Configuration Header Files 26
4.8.1 Configuration Header Templates 27
4.8.2 Using autoheader to Create 'config.h.in' ... 28
4.8.3 Autoheader Macros 29
4.9 Running Arbitrary Configuration Commands 30
4.10 Creating Configuration Links 30
4.11 Configuring Other Packages in Subdirectories ... 31
4.12 Default Prefix 32

5 Existing Tests 33

5.1 Common Behavior 33
5.1.1 Standard Symbols 33
5.1.2 Default Includes 33
5.2 Alternative Programs 34
5.2.1 Particular Program Checks 35
5.2.2 Generic Program and File Checks 37
5.3 Files 38
5.4 Library Files 38
5.5 Library Functions 39
5.5.1 Portability of C Functions 39
5.5.2 Particular Function Checks 41
5.5.3 Generic Function Checks 46
5.6 Header Files 48
5.6.1 Portability of Headers 48
5.6.2 Particular Header Checks 49
5.6.3 Generic Header Checks 53
5.7 Declarations 54
5.7.1 Particular Declaration Checks 54
5.7.2 Generic Declaration Checks 54
5.8 Structures 55
5.8.1 Particular Structure Checks 55
5.8.2 Generic Structure Checks 56
5.9 Types 56
5.9.1 Particular Type Checks 56
5.9.2 Generic Type Checks 57
5.10 Compilers and Preprocessors 58
5.10.1 Specific Compiler Characteristics 58
5.10.2 Generic Compiler Characteristics 58
5.10.3 C Compiler Characteristics 59
5.10.4 C++ Compiler Characteristics 63
5.10.5 Fortran Compiler Characteristics 63
5.11 System Services 68
5.12 UNIX Variants 69

6 Writing Tests 71

6.1 Language Choice 71
6.2 Writing Test Programs 72
6.2.1 Guidelines for Test Programs 72
6.2.2 Test Functions 73
6.2.3 Generating Sources 74
6.3 Running the Preprocessor 76
6.4 Running the Compiler 77
6.5 Running the Linker 77
6.6 Checking Run Time Behavior 78
6.7 Systemology 79
6.8 Multiple Cases 79

7 Results of Tests 81
7.1 Defining C Preprocessor Symbols 81
7.2 Setting Output Variables 82
7.3 Caching Results 83
7.3.1 Cache Variable Names 85
7.3.2 Cache Files 85
7.3.3 Cache Checkpointing 86
7.4 Printing Messages 86

8 Programming in M4 89
8.1 M4 Quotation 89
8.1.1 Active Characters 89
8.1.2 One Macro Call 90
8.1.3 Quotation and Nested Macros 90
8.1.4 `changequote` is Evil 92
8.1.5 Quadrigraphs 93
8.1.6 Quotation Rule Of Thumb 94
8.2 Using `autom4te` 95
8.2.1 Invoking `autom4te` 95
8.2.2 Customizing `autom4te` 99
8.3 Programming in M4sugar 100
8.3.1 Redefined M4 Macros 100
8.3.2 Evaluation Macros 101
8.3.3 Forbidden Patterns 101
8.4 Programming in M4sh 102

9 Writing Autoconf Macros 103
9.1 Macro Definitions 103
9.2 Macro Names 103
9.3 Reporting Messages 104
9.4 Dependencies Between Macros 105
9.4.1 Prerequisite Macros 105
9.4.2 Suggested Ordering 106
9.5 Obsoleting Macros 107
9.6 Coding Style 107

10 Portable Shell Programming 111
10.1 Shellology 111
10.2 Here-Documents 113
10.3 File Descriptors 114
10.4 File System Conventions 115
10.5 Shell Substitutions 116
10.6 Assignments 119
10.7 Parentheses in Shell Scripts 120
10.8 Special Shell Variables 120
10.9 Limitations of Shell Builtins 124
10.10 Limitations of Usual Tools 131
10.11 Limitations of Make 138

11 Manual Configuration 149

11.1 Specifying the System Type 149
11.2 Getting the Canonical System Type 150
11.3 Using the System Type 150

12 Site Configuration 153

12.1 Working With External Software 153
12.2 Choosing Package Options 154
12.3 Making Your Help Strings Look Pretty 155
12.4 Configuring Site Details 155
12.5 Transforming Program Names When Installing 156
12.5.1 Transformation Options 156
12.5.2 Transformation Examples 156
12.5.3 Transformation Rules 157
12.6 Setting Site Defaults 157

13 Running `configure` Scripts 159

13.1 Basic Installation 159
13.2 Compilers and Options 160
13.3 Compiling For Multiple Architectures 160
13.4 Installation Names 160
13.5 Optional Features 160
13.6 Specifying the System Type 161
13.7 Sharing Defaults 161
13.8 Defining Variables 161
13.9 `configure` Invocation 161

14 Recreating a Configuration 163

15 Obsolete Constructs 165

15.1 Obsolete '`config.status`' Invocation 165
15.2 '`acconfig.h`' 166
15.3 Using `autoupdate` to Modernize '`configure.ac`' 166
15.4 Obsolete Macros 167
15.5 Upgrading From Version 1 177
15.5.1 Changed File Names 177
15.5.2 Changed Makefiles 177
15.5.3 Changed Macros 178
15.5.4 Changed Results 178
15.5.5 Changed Macro Writing 179
15.6 Upgrading From Version 2.13 180
15.6.1 Changed Quotation 180
15.6.2 New Macros 181
15.6.3 Hosts and Cross-Compilation 182
15.6.4 `AC_LIBOBJ` vs.`LIBOBJS` 183
15.6.5 `AC_FOO_IFELSE` vs.`AC_TRY_FOO` 184

16 Generating Test Suites with Autotest 187
16.1 Using an Autotest Test Suite 187
16.1.1 `testsuite` Scripts 187
16.1.2 Autotest Logs 189
16.2 Writing '`testsuite.at`' 189
16.3 Running `testsuite` Scripts 191
16.4 Making `testsuite` Scripts 192

17 Frequent Autoconf Questions, with answers 195
17.1 Distributing `configure` Scripts 195
17.2 Why Require GNU M4? 195
17.3 How Can I Bootstrap? 195
17.4 Why Not Imake? 196
17.5 How Do I `#define` Installation Directories? 197
17.6 What is '`autom4te.cache`'? 198
17.7 Header Present But Cannot Be Compiled 198

18 History of Autoconf 201
18.1 Genesis 201
18.2 Exodus 201
18.3 Leviticus 202
18.4 Numbers 202
18.5 Deuteronomy 203

Appendix A Copying This Manual 205
A.1 GNU Free Documentation License 205
A.1.1 ADDENDUM: How to use this License for your documents 211

Appendix B Indices 213
B.1 Environment Variable Index 213
B.2 Output Variable Index 213
B.3 Preprocessor Symbol Index 215
B.4 Autoconf Macro Index 216
B.5 M4 Macro Index 220
B.6 Autotest Macro Index 220
B.7 Program and Function Index 221
B.8 Concept Index 223

1 Introduction

> A physicist, an engineer, and a computer scientist were discussing the nature of God. “Surely a Physicist,” said the physicist, “because early in the Creation, God made Light; and you know, Maxwell’s equations, the dual nature of electromagnetic waves, the relativistic consequences...” “An Engineer!,” said the engineer, “because before making Light, God split the Chaos into Land and Water; it takes a hell of an engineer to handle that big amount of mud, and orderly separation of solids from liquids...” The computer scientist shouted: “And the Chaos, where do you think it was coming from, hmm?”
>
> —Anonymous

Autoconf is a tool for producing shell scripts that automatically configure software source code packages to adapt to many kinds of UNIX-like systems. The configuration scripts produced by Autoconf are independent of Autoconf when they are run, so their users do not need to have Autoconf.

The configuration scripts produced by Autoconf require no manual user intervention when run; they do not normally even need an argument specifying the system type. Instead, they individually test for the presence of each feature that the software package they are for might need. (Before each check, they print a one-line message stating what they are checking for, so the user doesn’t get too bored while waiting for the script to finish.) As a result, they deal well with systems that are hybrids or customized from the more common UNIX variants. There is no need to maintain files that list the features supported by each release of each variant of UNIX.

For each software package that Autoconf is used with, it creates a configuration script from a template file that lists the system features that the package needs or can use. After the shell code to recognize and respond to a system feature has been written, Autoconf allows it to be shared by many software packages that can use (or need) that feature. If it later turns out that the shell code needs adjustment for some reason, it needs to be changed in only one place; all of the configuration scripts can be regenerated automatically to take advantage of the updated code.

The Metaconfig package is similar in purpose to Autoconf, but the scripts it produces require manual user intervention, which is quite inconvenient when configuring large source trees. Unlike Metaconfig scripts, Autoconf scripts can support cross-compiling, if some care is taken in writing them.

Autoconf does not solve all problems related to making portable software packages—for a more complete solution, it should be used in concert with other GNU build tools like Automake and Libtool. These other tools take on jobs like the creation of a portable, recursive ‘`Makefile`’ with all of the standard targets, linking of shared libraries, and so on. See Chapter 2 [The GNU Build System], page 3, for more information.

Autoconf imposes some restrictions on the names of macros used with `#if` in C programs (see Section B.3 [Preprocessor Symbol Index], page 215).

Autoconf requires GNU M4 in order to generate the scripts. It uses features that some UNIX versions of M4, including GNU M4 1.3, do not have. You must use version 1.4 or later of GNU M4.

See Section 15.5 [Autoconf 1], page 177, for information about upgrading from version 1. See Chapter 18 [History], page 201, for the story of Autoconf's development. See Chapter 17 [FAQ], page 195, for answers to some common questions about Autoconf.

See the Autoconf web page[1] for up-to-date information, details on the mailing lists, pointers to a list of known bugs, etc.

Mail suggestions to the Autoconf mailing list (`autoconf@gnu.org`).

Bug reports should be preferably submitted to the Autoconf Gnats database[2], or sent to the Autoconf Bugs mailing list (`bug-autoconf@gnu.org`). If possible, first check that your bug is not already solved in current development versions, and that it has not been reported yet. Be sure to include all the needed information and a short '`configure.ac`' that demonstrates the problem.

Autoconf's development tree is accessible via CVS; see the Autoconf web page for details. There is also a CVSweb interface to the Autoconf development tree[3]. Patches relative to the current CVS version can be sent for review to the Autoconf Patches mailing list (`autoconf-patches@gnu.org`).

Because of its mission, Autoconf includes only a set of often-used macros that have already demonstrated their usefulness. Nevertheless, if you wish to share your macros, or find existing ones, see the Autoconf Macro Archive[4], which is kindly run by Peter Simons (`simons@computer.org`).

---

[1] Autoconf web page, `http://www.gnu.org/software/autoconf/autoconf.html`.

[2] Autoconf Gnats database, `http://bugs.gnu.org/cgi-bin/gnatsweb.pl?database=autoconf`.

[3] CVSweb interface to the Autoconf development tree, `http://subversions.gnu.org/cgi-bin/cvsweb/autoconf/`.

[4] Autoconf Macro Archive, `http://www.gnu.org/software/ac-archive/`.

2 The GNU Build System

Autoconf solves an important problem—reliable discovery of system-specific build and run-time information—but this is only one piece of the puzzle for the development of portable software. To this end, the GNU project has developed a suite of integrated utilities to finish the job Autoconf started: the GNU build system, whose most important components are Autoconf, Automake, and Libtool. In this chapter, we introduce you to those tools, point you to sources of more information, and try to convince you to use the entire GNU build system for your software.

2.1 Automake

The ubiquity of `make` means that a '`Makefile`' is almost the only viable way to distribute automatic build rules for software, but one quickly runs into `make`'s numerous limitations. Its lack of support for automatic dependency tracking, recursive builds in subdirectories, reliable timestamps (e.g., for network filesystems), and so on, mean that developers must painfully (and often incorrectly) reinvent the wheel for each project. Portability is non-trivial, thanks to the quirks of `make` on many systems. On top of all this is the manual labor required to implement the many standard targets that users have come to expect (`make install`, `make distclean`, `make uninstall`, etc.). Since you are, of course, using Autoconf, you also have to insert repetitive code in your `Makefile.in` to recognize `@CC@`, `@CFLAGS@`, and other substitutions provided by `configure`. Into this mess steps *Automake*.

Automake allows you to specify your build needs in a `Makefile.am` file with a vastly simpler and more powerful syntax than that of a plain `Makefile`, and then generates a portable `Makefile.in` for use with Autoconf. For example, the `Makefile.am` to build and install a simple "Hello world" program might look like:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

The resulting `Makefile.in` (~400 lines) automatically supports all the standard targets, the substitutions provided by Autoconf, automatic dependency tracking, `VPATH` building, and so on. `make` will build the `hello` program, and `make install` will install it in '`/usr/local/bin`' (or whatever prefix was given to `configure`, if not '`/usr/local`').

The benefits of Automake increase for larger packages (especially ones with subdirectories), but even for small programs the added convenience and portability can be substantial. And that's not all. . . .

2.2 Libtool

Very often, one wants to build not only programs, but libraries, so that other programs can benefit from the fruits of your labor. Ideally, one would like to produce *shared* (dynamically linked) libraries, which can be used by multiple programs without duplication on disk or in memory and can be updated independently of the linked programs. Producing shared libraries portably, however, is the stuff of nightmares—each system has its own incompatible tools, compiler flags, and magic incantations. Fortunately, GNU provides a solution: *Libtool*.

Libtool handles all the requirements of building shared libraries for you, and at this time seems to be the *only* way to do so with any portability. It also handles many other headaches, such as: the interaction of `Makefile` rules with the variable suffixes of shared libraries, linking reliably with shared libraries before they are installed by the superuser, and supplying a consistent versioning system (so that different versions of a library can be installed or upgraded without breaking binary compatibility). Although Libtool, like Autoconf, can be used on its own, it is most simply utilized in conjunction with Automake—there, Libtool is used automatically whenever shared libraries are needed, and you need not know its syntax.

2.3 Pointers

Developers who are used to the simplicity of `make` for small projects on a single system might be daunted at the prospect of learning to use Automake and Autoconf. As your software is distributed to more and more users, however, you will otherwise quickly find yourself putting lots of effort into reinventing the services that the GNU build tools provide, and making the same mistakes that they once made and overcame. (Besides, since you're already learning Autoconf, Automake will be a piece of cake.)

There are a number of places that you can go to for more information on the GNU build tools.

- Web

  The home pages for Autoconf[1], Automake[2], and Libtool[3].

- Automake Manual

  See section "Automake" in GNU *Automake*, for more information on Automake.

- Books

  The book GNU *Autoconf, Automake and Libtool*[4] describes the complete GNU build environment. You can also find the entire book on-line at "The Goat Book" home page[5].

- Tutorials and Examples

  The Autoconf Developer Page[6] maintains links to a number of Autoconf/Automake tutorials online, and also links to the Autoconf Macro Archive[7].

---

[1] Autoconf, `http://www.gnu.org/software/autoconf/`.

[2] Automake, `http://www.gnu.org/software/automake/`.

[3] Libtool, `http://www.gnu.org/software/libtool/`.

[4] GNU *Autoconf, Automake and Libtool*, by G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor. New Riders, 2000, ISBN 1578701902.

[5] "The Goat Book" home page, `http://sources.redhat.com/autobook/`.

[6] Autoconf Developer Page, `http://sources.redhat.com/autoconf/`.

[7] Autoconf Macro Archive, `http://www.gnu.org/software/ac-archive/`.

3 Making configure Scripts

The configuration scripts that Autoconf produces are by convention called `configure`. When run, `configure` creates several files, replacing configuration parameters in them with appropriate values. The files that `configure` creates are:

- one or more '`Makefile`' files, usually one in each subdirectory of the package (see Section 4.7 [Makefile Substitutions], page 20);
- optionally, a C header file, the name of which is configurable, containing `#define` directives (see Section 4.8 [Configuration Headers], page 26);
- a shell script called '`config.status`' that, when run, will recreate the files listed above (see Chapter 14 [config.status Invocation], page 163);
- an optional shell script normally called '`config.cache`' (created when using '`configure --config-cache`') that saves the results of running many of the tests (see Section 7.3.2 [Cache Files], page 85);
- a file called '`config.log`' containing any messages produced by compilers, to help debugging if `configure` makes a mistake.

To create a `configure` script with Autoconf, you need to write an Autoconf input file '`configure.ac`' (or '`configure.in`') and run `autoconf` on it. If you write your own feature tests to supplement those that come with Autoconf, you might also write files called '`aclocal.m4`' and '`acsite.m4`'. If you use a C header file to contain `#define` directives, you might also run `autoheader`, and you will distribute the generated file '`config.h.in`' with the package.

Here is a diagram showing how the files that can be used in configuration are produced. Programs that are executed are suffixed by '`*`'. Optional files are enclosed in square brackets ('`[]`'). `autoconf` and `autoheader` also read the installed Autoconf macro files (by reading '`autoconf.m4`').

Files used in preparing a software package for distribution:

```
your source files --> [autoscan*] --> [configure.scan] --> configure.ac

configure.ac --.
               |   .------> autoconf* -----> configure
[aclocal.m4] --+---+
               |   `-----> [autoheader*] --> [config.h.in]
[acsite.m4] ---'

Makefile.in -------------------------------> Makefile.in
```

Files used in configuring a software package:

```
                       .-------------> [config.cache]
configure* ------------+-------------> config.log
                       |
[config.h.in] -.       v            .-> [config.h] -.
               +--> config.status* -+               +--> make*
Makefile.in ---'                    `-> Makefile ---'
```

3.1 Writing 'configure.ac'

To produce a `configure` script for a software package, create a file called '`configure.ac`' that contains invocations of the Autoconf macros that test the system features your package needs or can use. Autoconf macros already exist to check for many features; see Chapter 5 [Existing Tests], page 33, for their descriptions. For most other features, you can use Autoconf template macros to produce custom checks; see Chapter 6 [Writing Tests], page 71, for information about them. For especially tricky or specialized features, '`configure.ac`' might need to contain some hand-crafted shell commands; see Chapter 10 [Portable Shell], page 111. The `autoscan` program can give you a good start in writing '`configure.ac`' (see Section 3.2 [autoscan Invocation], page 9, for more information).

Previous versions of Autoconf promoted the name '`configure.in`', which is somewhat ambiguous (the tool needed to process this file is not described by its extension), and introduces a slight confusion with '`config.h.in`' and so on (for which '`.in`' means "to be processed by `configure`"). Using '`configure.ac`' is now preferred.

3.1.1 A Shell Script Compiler

Just as for any other computer language, in order to properly program '`configure.ac`' in Autoconf you must understand *what* problem the language tries to address and *how* it does so.

The problem Autoconf addresses is that the world is a mess. After all, you are using Autoconf in order to have your package compile easily on all sorts of different systems, some of them being extremely hostile. Autoconf itself bears the price for these differences: `configure` must run on all those systems, and thus `configure` must limit itself to their lowest common denominator of features.

Naturally, you might then think of shell scripts; who needs `autoconf`? A set of properly written shell functions is enough to make it easy to write `configure` scripts by hand. Sigh! Unfortunately, shell functions do not belong to the least common denominator; therefore, where you would like to define a function and use it ten times, you would instead need to copy its body ten times.

So, what is really needed is some kind of compiler, `autoconf`, that takes an Autoconf program, '`configure.ac`', and transforms it into a portable shell script, `configure`.

How does `autoconf` perform this task?

There are two obvious possibilities: creating a brand new language or extending an existing one. The former option is very attractive: all sorts of optimizations could easily be implemented in the compiler and many rigorous checks could be performed on the Autoconf program (e.g., rejecting any non-portable construct). Alternatively, you can extend an existing language, such as the `sh` (Bourne shell) language.

Autoconf does the latter: it is a layer on top of `sh`. It was therefore most convenient to implement `autoconf` as a macro expander: a program that repeatedly performs *macro expansions* on text input, replacing macro calls with macro bodies and producing a pure `sh` script in the end. Instead of implementing a dedicated Autoconf macro expander, it is natural to use an existing general-purpose macro language, such as M4, and implement the extensions as a set of M4 macros.

3.1.2 The Autoconf Language

The Autoconf language is very different from many other computer languages because it treats actual code the same as plain text. Whereas in C, for instance, data and instructions have very different syntactic status, in Autoconf their status is rigorously the same. Therefore, we need a means to distinguish literal strings from text to be expanded: quotation.

When calling macros that take arguments, there must not be any blank space between the macro name and the open parenthesis. Arguments should be enclosed within the M4 quote characters '`[`' and '`]`', and be separated by commas. Any leading spaces in arguments are ignored, unless they are quoted. You may safely leave out the quotes when the argument is simple text, but *always* quote complex arguments such as other macro calls. This rule applies recursively for every macro call, including macros called from other macros.

For instance:

```
AC_CHECK_HEADER([stdio.h],
                [AC_DEFINE([HAVE_STDIO_H])],
                [AC_MSG_ERROR([Sorry, can't do anything for you])])
```

is quoted properly. You may safely simplify its quotation to:

```
AC_CHECK_HEADER(stdio.h,
                [AC_DEFINE(HAVE_STDIO_H)],
                [AC_MSG_ERROR([Sorry, can't do anything for you])])
```

Notice that the argument of `AC_MSG_ERROR` is still quoted; otherwise, its comma would have been interpreted as an argument separator.

The following example is wrong and dangerous, as it is underquoted:

```
AC_CHECK_HEADER(stdio.h,
                AC_DEFINE(HAVE_STDIO_H),
                AC_MSG_ERROR([Sorry, can't do anything for you]))
```

In other cases, you may have to use text that also resembles a macro call. You must quote that text even when it is not passed as a macro argument:

```
echo "Hard rock was here!  --[AC_DC]"
```

which will result in

```
echo "Hard rock was here!  --AC_DC"
```

When you use the same text in a macro argument, you must therefore have an extra quotation level (since one is stripped away by the macro substitution). In general, then, it is a good idea to *use double quoting for all literal string arguments*:

```
AC_MSG_WARN([[AC_DC stinks  --Iron Maiden]])
```

You are now able to understand one of the constructs of Autoconf that has been continually misunderstood... The rule of thumb is that *whenever you expect macro expansion, expect quote expansion*; i.e., expect one level of quotes to be lost. For instance:

```
AC_COMPILE_IFELSE([char b[10];],, [AC_MSG_ERROR([you lose])])
```

is incorrect: here, the first argument of `AC_COMPILE_IFELSE` is '`char b[10];`' and will be expanded once, which results in '`char b10;`'. (There was an idiom common in Autoconf's past to address this issue via the M4 `changequote` primitive, but do not use it!) Let's take a closer look: the author meant the first argument to be understood as a literal, and therefore it must be quoted twice:

```
AC_COMPILE_IFELSE([[char b[10];]],, [AC_MSG_ERROR([you lose])])
```

Voilà, you actually produce '`char b[10];`' this time!

The careful reader will notice that, according to these guidelines, the "properly" quoted `AC_CHECK_HEADER` example above is actually lacking three pairs of quotes! Nevertheless, for the sake of readability, double quotation of literals is used only where needed in this manual.

Some macros take optional arguments, which this documentation represents as [*arg*] (not to be confused with the quote characters). You may just leave them empty, or use '`[]`' to make the emptiness of the argument explicit, or you may simply omit the trailing commas. The three lines below are equivalent:

```
AC_CHECK_HEADERS(stdio.h, [], [], [])
AC_CHECK_HEADERS(stdio.h,,,)
AC_CHECK_HEADERS(stdio.h)
```

It is best to put each macro call on its own line in '`configure.ac`'. Most of the macros don't add extra newlines; they rely on the newline after the macro call to terminate the commands. This approach makes the generated `configure` script a little easier to read by not inserting lots of blank lines. It is generally safe to set shell variables on the same line as a macro call, because the shell allows assignments without intervening newlines.

You can include comments in '`configure.ac`' files by starting them with the '`#`'. For example, it is helpful to begin '`configure.ac`' files with a line like this:

```
# Process this file with autoconf to produce a configure script.
```

3.1.3 Standard '`configure.ac`' Layout

The order in which '`configure.ac`' calls the Autoconf macros is not important, with a few exceptions. Every '`configure.ac`' must contain a call to `AC_INIT` before the checks, and a call to `AC_OUTPUT` at the end (see Section 4.4 [Output], page 17). Additionally, some macros rely on other macros having been called first, because they check previously set values of some variables to decide what to do. These macros are noted in the individual descriptions (see Chapter 5 [Existing Tests], page 33), and they also warn you when `configure` is created if they are called out of order.

To encourage consistency, here is a suggested order for calling the Autoconf macros. Generally speaking, the things near the end of this list are those that could depend on things earlier in it. For example, library functions could be affected by types and libraries.

```
Autoconf requirements
AC_INIT(package, version, bug-report-address)
information on the package
checks for programs
checks for libraries
checks for header files
checks for types
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services
AC_CONFIG_FILES([file...])
AC_OUTPUT
```

3.2 Using autoscan to Create 'configure.ac'

The `autoscan` program can help you create and/or maintain a '`configure.ac`' file for a software package. `autoscan` examines source files in the directory tree rooted at a directory given as a command line argument, or the current directory if none is given. It searches the source files for common portability problems and creates a file '`configure.scan`' which is a preliminary '`configure.ac`' for that package, and checks a possibly existing '`configure.ac`' for completeness.

When using `autoscan` to create a '`configure.ac`', you should manually examine '`configure.scan`' before renaming it to '`configure.ac`'; it will probably need some adjustments. Occasionally, `autoscan` outputs a macro in the wrong order relative to another macro, so that `autoconf` produces a warning; you need to move such macros manually. Also, if you want the package to use a configuration header file, you must add a call to `AC_CONFIG_HEADERS` (see Section 4.8 [Configuration Headers], page 26). You might also have to change or add some `#if` directives to your program in order to make it work with Autoconf (see Section 3.3 [ifnames Invocation], page 10, for information about a program that can help with that job).

When using `autoscan` to maintain a '`configure.ac`', simply consider adding its suggestions. The file '`autoscan.log`' will contain detailed information on why a macro is requested.

`autoscan` uses several data files (installed along with Autoconf) to determine which macros to output when it finds particular symbols in a package's source files. These data files all have the same format: each line consists of a symbol, whitespace, and the Autoconf macro to output if that symbol is encountered. Lines starting with '`#`' are comments.

`autoscan` accepts the following options:

'`--help`'
'`-h`' Print a summary of the command line options and exit.

'`--version`'
'`-V`' Print the version number of Autoconf and exit.

'`--verbose`'
'`-v`' Print the names of the files it examines and the potentially interesting symbols it finds in them. This output can be voluminous.

'--include=*dir*'
'-I *dir*' Append *dir* to the include path. Multiple invocations accumulate.

'--prepend-include=*dir*'
'-B *dir*' Prepend *dir* to the include path. Multiple invocations accumulate.

3.3 Using ifnames to List Conditionals

`ifnames` can help you write 'configure.ac' for a software package. It prints the identifiers that the package already uses in C preprocessor conditionals. If a package has already been set up to have some portability, `ifnames` can thus help you figure out what its `configure` needs to check for. It may help fill in some gaps in a 'configure.ac' generated by `autoscan` (see Section 3.2 [autoscan Invocation], page 9).

`ifnames` scans all of the C source files named on the command line (or the standard input, if none are given) and writes to the standard output a sorted list of all the identifiers that appear in those files in `#if`, `#elif`, `#ifdef`, or `#ifndef` directives. It prints each identifier on a line, followed by a space-separated list of the files in which that identifier occurs.

`ifnames` accepts the following options:

'--help'
'-h' Print a summary of the command line options and exit.

'--version'
'-V' Print the version number of Autoconf and exit.

3.4 Using autoconf to Create configure

To create `configure` from 'configure.ac', run the `autoconf` program with no arguments. `autoconf` processes 'configure.ac' with the M4 macro processor, using the Autoconf macros. If you give `autoconf` an argument, it reads that file instead of 'configure.ac' and writes the configuration script to the standard output instead of to `configure`. If you give `autoconf` the argument '-', it reads from the standard input instead of 'configure.ac' and writes the configuration script to the standard output.

The Autoconf macros are defined in several files. Some of the files are distributed with Autoconf; `autoconf` reads them first. Then it looks for the optional file 'acsite.m4' in the directory that contains the distributed Autoconf macro files, and for the optional file 'aclocal.m4' in the current directory. Those files can contain your site's or the package's own Autoconf macro definitions (see Chapter 9 [Writing Autoconf Macros], page 103, for more information). If a macro is defined in more than one of the files that `autoconf` reads, the last definition it reads overrides the earlier ones.

`autoconf` accepts the following options:

'--help'
'-h' Print a summary of the command line options and exit.

'--version'
'-V' Print the version number of Autoconf and exit.

'--verbose'
'-v' Report processing steps.

'`--debug`'
'`-d`' Don't remove the temporary files.

'`--force`'
'`-f`' Remake '`configure`' even if newer than its input files.

'`--include=`*dir*'
'`-I` *dir*' Append *dir* to the include path. Multiple invocations accumulate.

'`--prepend-include=`*dir*'
'`-B` *dir*' Prepend *dir* to the include path. Multiple invocations accumulate.

'`--output=`*file*'
'`-o` *file*' Save output (script or trace) to *file*. The file '`-`' stands for the standard output.

'`--warnings=`*category*'
'`-W` *category*'

Report the warnings related to *category* (which can actually be a comma separated list). See Section 9.3 [Reporting Messages], page 104, macro `AC_DIAGNOSE`, for a comprehensive list of categories. Special values include:

'`all`' report all the warnings

'`none`' report none

'`error`' treats warnings as errors

'`no-`*category*'
disable warnings falling into *category*

Warnings about '`syntax`' are enabled by default, and the environment variable `WARNINGS`, a comma separated list of categories, is honored. Passing '`-W` *category*' will actually behave as if you had passed '`--warnings=syntax,$WARNINGS,`*category*'. If you want to disable the defaults and `WARNINGS`, but (for example) enable the warnings about obsolete constructs, you would use '`-W none,obsolete`'.

Because `autoconf` uses `autom4te` behind the scenes, it displays a back trace for errors, but not for warnings; if you want them, just pass '`-W error`'. See Section 8.2.1 [autom4te Invocation], page 95, for some examples.

'`--trace=`*macro*`[:`*format*`]`'
'`-t` *macro*`[:`*format*`]`'

Do not create the `configure` script, but list the calls to *macro* according to the *format*. Multiple '`--trace`' arguments can be used to list several macros. Multiple '`--trace`' arguments for a single macro are not cumulative; instead, you should just make *format* as long as needed.

The *format* is a regular string, with newlines if desired, and several special escape codes. It defaults to '`$f:$l:$n:$%`'; see Section 8.2.1 [autom4te Invocation], page 95, for details on the *format*.

'`--initialization`'
'`-i`' By default, '`--trace`' does not trace the initialization of the Autoconf macros (typically the `AC_DEFUN` definitions). This results in a noticeable speedup, but can be disabled by this option.

It is often necessary to check the content of a 'configure.ac' file, but parsing it yourself is extremely fragile and error-prone. It is suggested that you rely upon '--trace' to scan 'configure.ac'. For instance, to find the list of variables that are substituted, use:

```
$ autoconf -t AC_SUBST
configure.ac:2:AC_SUBST:ECHO_C
configure.ac:2:AC_SUBST:ECHO_N
configure.ac:2:AC_SUBST:ECHO_T
More traces deleted
```

The example below highlights the difference between '$@', '$*', and $%.

```
$ cat configure.ac
AC_DEFINE(This, is, [an
[example]])
$ autoconf -t 'AC_DEFINE:@: $@
*: $*
$: $%'
@: [This],[is],[an
[example]]
*: This,is,an
[example]
$: This:is:an [example]
```

The *format* gives you a lot of freedom:

```
$ autoconf -t 'AC_SUBST:$$ac_subst{"$1"} = "$f:$1";'
$ac_subst{"ECHO_C"} = "configure.ac:2";
$ac_subst{"ECHO_N"} = "configure.ac:2";
$ac_subst{"ECHO_T"} = "configure.ac:2";
More traces deleted
```

A long *separator* can be used to improve the readability of complex structures, and to ease their parsing (for instance when no single character is suitable as a separator):

```
$ autoconf -t 'AM_MISSING_PROG:${|:::::|}*'
ACLOCAL|:::::|aclocal|:::::|$missing_dir
AUTOCONF|:::::|autoconf|:::::|$missing_dir
AUTOMAKE|:::::|automake|:::::|$missing_dir
More traces deleted
```

3.5 Using autoreconf to Update configure Scripts

Installing the various components of the GNU Build System can be tedious: running autopoint for Gettext, automake for 'Makefile.in' etc. in each directory. It may be needed either because some tools such as automake have been updated on your system, or because some of the sources such as 'configure.ac' have been updated, or finally, simply in order to install the GNU Build System in a fresh tree.

autoreconf runs autoconf, autoheader, aclocal, automake, libtoolize, and autopoint (when appropriate) repeatedly to update the GNU Build System in the specified directories and their subdirectories (see Section 4.11 [Subdirectories], page 31). By default, it only remakes those files that are older than their sources.

If you install a new version of some tool, you can make `autoreconf` remake *all* of the files by giving it the '`--force`' option.

See Section 4.7.4 [Automatic Remaking], page 25, for '`Makefile`' rules to automatically remake `configure` scripts when their source files change. That method handles the timestamps of configuration header templates properly, but does not pass '`--autoconf-dir=`*`dir`*' or '`--localdir=`*`dir`*'.

`autoreconf` accepts the following options:

'`--help`'
'`-h`' Print a summary of the command line options and exit.

'`--version`'
'`-V`' Print the version number of Autoconf and exit.

'`--verbose`'
: Print the name of each directory where `autoreconf` runs `autoconf` (and `autoheader`, if appropriate).

'`--debug`'
'`-d`' Don't remove the temporary files.

'`--force`'
'`-f`' Remake even '`configure`' scripts and configuration headers that are newer than their input files ('`configure.ac`' and, if present, '`aclocal.m4`').

'`--install`'
'`-i`' Install the missing auxiliary files in the package. By default, files are copied; this can be changed with '`--symlink`'.

This option triggers calls to '`automake --add-missing`', '`libtoolize`', '`autopoint`', etc.

'`--symlink`'
'`-s`' When used with '`--install`', install symbolic links to the missing auxiliary files instead of copying them.

'`--make`'
'`-m`' When the directories were configured, update the configuration by running '`./config.status --recheck && ./config.status`', and then run '`make`'.

'`--include=`*`dir`*'
'`-I` *`dir`*' Append *dir* to the include path. Multiple invocations accumulate.

'`--prepend-include=`*`dir`*'
'`-B` *`dir`*' Prepend *dir* to the include path. Multiple invocations accumulate.

'`--warnings=`*`category`*'
'`-W` *`category`*'
: Report the warnings related to *category* (which can actually be a comma separated list).

'`cross`' related to cross compilation issues.

'`obsolete`'
: report the uses of obsolete constructs.

'`portability`'
: portability issues

'`syntax`'
: dubious syntactic constructs.

'`all`'
: report all the warnings

'`none`'
: report none

'`error`'
: treats warnings as errors

'`no-`*`category`*'
: disable warnings falling into *category*

Warnings about '`syntax`' are enabled by default, and the environment variable `WARNINGS`, a comma separated list of categories, is honored. Passing '`-W` *`category`*' will actually behave as if you had passed '`--warnings=syntax,$WARNINGS,`*`category`*'. If you want to disable the defaults and `WARNINGS`, but (for example) enable the warnings about obsolete constructs, you would use '`-W none,obsolete`'.

4 Initialization and Output Files

Autoconf-generated `configure` scripts need some information about how to initialize, such as how to find the package's source files and about the output files to produce. The following sections describe the initialization and the creation of output files.

4.1 Initializing `configure`

Every `configure` script must call `AC_INIT` before doing anything else. The only other required macro is `AC_OUTPUT` (see Section 4.4 [Output], page 17).

AC_INIT (*package, version,* [*bug-report*], [*tarname*]) [Macro]

Process any command-line arguments and perform various initializations and verifications.

Set the name of the *package* and its *version*. These are typically used in '`--version`' support, including that of `configure`. The optional argument *bug-report* should be the email to which users should send bug reports. The package *tarname* differs from *package*: the latter designates the full package name (e.g., '`GNU Autoconf`'), while the former is meant for distribution tar ball names (e.g., '`autoconf`'). It defaults to *package* with '`GNU `' stripped, lower-cased, and all characters other than alphanumerics and underscores are changed to '`-`'.

It is preferable that the arguments of `AC_INIT` be static, i.e., there should not be any shell computation, but they can be computed by M4.

The following M4 macros (e.g., `AC_PACKAGE_NAME`), output variables (e.g., `PACKAGE_NAME`), and preprocessor symbols (e.g., `PACKAGE_NAME`) are defined by `AC_INIT`:

`AC_PACKAGE_NAME`, `PACKAGE_NAME`
: Exactly *package*.

`AC_PACKAGE_TARNAME`, `PACKAGE_TARNAME`
: Exactly *tarname*.

`AC_PACKAGE_VERSION`, `PACKAGE_VERSION`
: Exactly *version*.

`AC_PACKAGE_STRING`, `PACKAGE_STRING`
: Exactly '*`package version`*'.

`AC_PACKAGE_BUGREPORT`, `PACKAGE_BUGREPORT`
: Exactly *bug-report*.

4.2 Notices in `configure`

The following macros manage version numbers for `configure` scripts. Using them is optional.

AC_PREREQ (*version*) [Macro]

Ensure that a recent enough version of Autoconf is being used. If the version of Autoconf being used to create `configure` is earlier than *version*, print an error message to the standard error output and exit with failure (exit status is 63). For example:

```
AC_PREREQ(2.59)
```

This macro is the only macro that may be used before `AC_INIT`, but for consistency, you are invited not to do so.

AC_COPYRIGHT (*copyright-notice*) [Macro]

State that, in addition to the Free Software Foundation's copyright on the Autoconf macros, parts of your `configure` are covered by the *copyright-notice*.

The *copyright-notice* will show up in both the head of `configure` and in '`configure --version`'.

AC_REVISION (*revision-info*) [Macro]

Copy revision stamp *revision-info* into the `configure` script, with any dollar signs or double-quotes removed. This macro lets you put a revision stamp from '`configure.ac`' into `configure` without RCS or CVS changing it when you check in `configure`. That way, you can determine easily which revision of '`configure.ac`' a particular `configure` corresponds to.

For example, this line in '`configure.ac`':

```
AC_REVISION($Revision: 1.30 $)
```

produces this in `configure`:

```
#! /bin/sh
# From configure.ac Revision: 1.30
```

4.3 Finding `configure` Input

AC_CONFIG_SRCDIR (*unique-file-in-source-dir*) [Macro]

unique-file-in-source-dir is some file that is in the package's source directory; `configure` checks for this file's existence to make sure that the directory that it is told contains the source code in fact does. Occasionally people accidentally specify the wrong directory with '`--srcdir`'; this is a safety check. See Section 13.9 [configure Invocation], page 161, for more information.

Packages that do manual configuration or use the `install` program might need to tell `configure` where to find some other shell scripts by calling `AC_CONFIG_AUX_DIR`, though the default places it looks are correct for most cases.

AC_CONFIG_AUX_DIR (*dir*) [Macro]

Use the auxiliary build tools (e.g., '`install-sh`', '`config.sub`', '`config.guess`', Cygnus `configure`, Automake and Libtool scripts etc.) that are in directory *dir*. These are auxiliary files used in configuration. *dir* can be either absolute or relative to '*`srcdir`*'. The default is '*`srcdir`*' or '*`srcdir/..`*' or '*`srcdir/../..`*', whichever is the first that contains '`install-sh`'. The other files are not checked for, so that using `AC_PROG_INSTALL` does not automatically require distributing the other auxiliary files. It checks for '`install.sh`' also, but that name is obsolete because some `make` have a rule that creates '`install`' from it if there is no '`Makefile`'.

Similarly, packages that use `aclocal` should declare where local macros can be found using `AC_CONFIG_MACRO_DIR`.

AC_CONFIG_MACRO_DIR (*dir*) [Macro]

Future versions of `autopoint`, `libtoolize`, `aclocal` and `autoreconf` will use directory *dir* as the location of additional local Autoconf macros. Be sure to call this macro directly from '`configure.ac`' so that tools that install macros for `aclocal` can find the declaration before '`--trace`' can be called safely.

4.4 Outputting Files

Every Autoconf script, e.g., '`configure.ac`', should finish by calling `AC_OUTPUT`. That is the macro that generates and runs '`config.status`', which will create the '`Makefile`'s and any other files resulting from configuration. This is the only required macro besides `AC_INIT` (see Section 4.3 [Input], page 16).

AC_OUTPUT [Macro]

Generate '`config.status`' and launch it. Call this macro once, at the end of '`configure.ac`'.

'`config.status`' will perform all the configuration actions: all the output files (see Section 4.6 [Configuration Files], page 19, macro `AC_CONFIG_FILES`), header files (see Section 4.8 [Configuration Headers], page 26, macro `AC_CONFIG_HEADERS`), commands (see Section 4.9 [Configuration Commands], page 30, macro `AC_CONFIG_COMMANDS`), links (see Section 4.10 [Configuration Links], page 30, macro `AC_CONFIG_LINKS`), subdirectories to configure (see Section 4.11 [Subdirectories], page 31, macro `AC_CONFIG_SUBDIRS`) are honored.

The location of your `AC_OUTPUT` invocation is the exact point where configuration actions are taken: any code afterwards will be executed by `configure` once `config.status` was run. If you want to bind actions to `config.status` itself (independently of whether `configure` is being run), see Section 4.9 [Running Arbitrary Configuration Commands], page 30.

Historically, the usage of `AC_OUTPUT` was somewhat different. See Section 15.4 [Obsolete Macros], page 167, for a description of the arguments that `AC_OUTPUT` used to support.

If you run `make` in subdirectories, you should run it using the `make` variable `MAKE`. Most versions of `make` set `MAKE` to the name of the `make` program plus any options it was given. (But many do not include in it the values of any variables set on the command line, so those are not passed on automatically.) Some old versions of `make` do not set this variable. The following macro allows you to use it even with those versions.

AC_PROG_MAKE_SET [Macro]

If `make` predefines the Make variable `MAKE`, define output variable `SET_MAKE` to be empty. Otherwise, define `SET_MAKE` to contain '`MAKE=make`'. Calls `AC_SUBST` for `SET_MAKE`.

If you use this macro, place a line like this in each '`Makefile.in`' that runs `MAKE` on other directories:

```
@SET_MAKE@
```

4.5 Performing Configuration Actions

'configure' is designed so that it appears to do everything itself, but there is actually a hidden slave: 'config.status'. 'configure' is in charge of examining your system, but it is 'config.status' that actually takes the proper actions based on the results of 'configure'. The most typical task of 'config.status' is to *instantiate* files.

This section describes the common behavior of the four standard instantiating macros: AC_CONFIG_FILES, AC_CONFIG_HEADERS, AC_CONFIG_COMMANDS and AC_CONFIG_LINKS. They all have this prototype:

```
AC_CONFIG_FOOS(tag..., [commands], [init-cmds])
```

where the arguments are:

tag... A whitespace-separated list of tags, which are typically the names of the files to instantiate.

You are encouraged to use literals as *tags*. In particular, you should avoid

```
... && my_foos="$my_foos fooo"
... && my_foos="$my_foos foooo"
AC_CONFIG_FOOS($my_foos)
```

and use this instead:

```
... && AC_CONFIG_FOOS(fooo)
... && AC_CONFIG_FOOS(foooo)
```

The macros AC_CONFIG_FILES and AC_CONFIG_HEADERS use special *tags*: they may have the form '*output*' or '*output*:*inputs*'. The file *output* is instantiated from its templates, *inputs* (defaulting to '*output*.in').

For instance 'AC_CONFIG_FILES(Makefile:boiler/top.mk:boiler/bot.mk)' asks for the creation of 'Makefile' that will be the expansion of the output variables in the concatenation of 'boiler/top.mk' and 'boiler/bot.mk'.

The special value '-' might be used to denote the standard output when used in *output*, or the standard input when used in the *inputs*. You most probably don't need to use this in 'configure.ac', but it is convenient when using the command line interface of './config.status', see Chapter 14 [config.status Invocation], page 163, for more details.

The *inputs* may be absolute or relative filenames. In the latter case they are first looked for in the build tree, and then in the source tree.

commands

Shell commands output literally into 'config.status', and associated with a tag that the user can use to tell 'config.status' which the commands to run. The commands are run each time a *tag* request is given to 'config.status', typically each time the file '*tag*' is created.

The variables set during the execution of configure are *not* available here: you first need to set them via the *init-cmds*. Nonetheless the following variables are precomputed:

srcdir The path from the top build directory to the top source directory. This is what configure's option '--srcdir' sets.

`ac_top_srcdir`
: The path from the current build directory to the top source directory.

`ac_top_builddir`
: The path from the current build directory to the top build directory. It can be empty, or else ends with a slash, so that you may concatenate it.

`ac_srcdir`
: The path from the current build directory to the corresponding source directory.

The *current* directory refers to the directory (or pseudo-directory) containing the input part of *tags*. For instance, running

```
AC_CONFIG_COMMANDS([deep/dir/out:in/in.in], [...], [...])
```

with '`--srcdir=../package`' produces the following values:

```
# Argument of --srcdir
srcdir='../package'
# Reversing deep/dir
ac_top_builddir='../../'
# Concatenation of $ac_top_builddir and srcdir
ac_top_srcdir='../../../package'
# Concatenation of $ac_top_srcdir and deep/dir
ac_srcdir='../../../package/deep/dir'
```

independently of '`in/in.in`'.

init-cmds
: Shell commands output *unquoted* near the beginning of '`config.status`', and executed each time '`config.status`' runs (regardless of the tag). Because they are unquoted, for example, '`$var`' will be output as the value of `var`. *init-cmds* is typically used by '`configure`' to give '`config.status`' some variables it needs to run the *commands*.

  You should be extremely cautious in your variable names: all the *init-cmds* share the same name space and may overwrite each other in unpredictable ways. Sorry. . . .

All these macros can be called multiple times, with different *tags*, of course!

4.6 Creating Configuration Files

Be sure to read the previous section, Section 4.5 [Configuration Actions], page 18.

AC_CONFIG_FILES (*file*. . . , [*cmds*], [*init-cmds*]) [Macro]

Make `AC_OUTPUT` create each '*`file`*' by copying an input file (by default '*`file`*`.in`'), substituting the output variable values. This macro is one of the instantiating macros; see Section 4.5 [Configuration Actions], page 18. See Section 4.7 [Makefile Substitutions], page 20, for more information on using output variables. See Section 7.2 [Setting Output Variables], page 82, for more information on creating them. This macro creates the directory that the file is in if it doesn't exist. Usually, '`Makefile`'s are created this way, but other files, such as '`.gdbinit`', can be specified as well.

Typical calls to AC_CONFIG_FILES look like this:

```
AC_CONFIG_FILES([Makefile src/Makefile man/Makefile X/Imakefile])
AC_CONFIG_FILES([autoconf], [chmod +x autoconf])
```

You can override an input file name by appending to *file* a colon-separated list of input files. Examples:

```
AC_CONFIG_FILES([Makefile:boiler/top.mk:boiler/bot.mk]
                [lib/Makefile:boiler/lib.mk])
```

Doing this allows you to keep your file names acceptable to MS-DOS, or to prepend and/or append boilerplate to the file.

4.7 Substitutions in Makefiles

Each subdirectory in a distribution that contains something to be compiled or installed should come with a file 'Makefile.in', from which configure will create a 'Makefile' in that directory. To create a 'Makefile', configure performs a simple variable substitution, replacing occurrences of '@*variable*@' in 'Makefile.in' with the value that configure has determined for that variable. Variables that are substituted into output files in this way are called *output variables*. They are ordinary shell variables that are set in configure. To make configure substitute a particular variable into the output files, the macro AC_SUBST must be called with that variable name as an argument. Any occurrences of '@*variable*@' for other variables are left unchanged. See Section 7.2 [Setting Output Variables], page 82, for more information on creating output variables with AC_SUBST.

A software package that uses a configure script should be distributed with a file 'Makefile.in', but no 'Makefile'; that way, the user has to properly configure the package for the local system before compiling it.

See section "Makefile Conventions" in *The GNU Coding Standards*, for more information on what to put in 'Makefile's.

4.7.1 Preset Output Variables

Some output variables are preset by the Autoconf macros. Some of the Autoconf macros set additional output variables, which are mentioned in the descriptions for those macros. See Section B.2 [Output Variable Index], page 213, for a complete list of output variables. See Section 4.7.2 [Installation Directory Variables], page 22, for the list of the preset ones related to installation directories. Below are listed the other preset ones. They all are precious variables (see Section 7.2 [Setting Output Variables], page 82, AC_ARG_VAR).

CFLAGS [Variable]

Debugging and optimization options for the C compiler. If it is not set in the environment when configure runs, the default value is set when you call AC_PROG_CC (or empty if you don't). configure uses this variable when compiling programs to test for C features.

configure_input [Variable]

A comment saying that the file was generated automatically by configure and giving the name of the input file. AC_OUTPUT adds a comment line containing this variable to the top of every 'Makefile' it creates. For other files, you should reference this variable in a comment at the top of each input file. For example, an input shell script should begin like this:

```
#! /bin/sh
# @configure_input@
```

The presence of that line also reminds people editing the file that it needs to be processed by `configure` in order to be used.

CPPFLAGS [Variable]

Header file search directory ('`-I`*dir*') and any other miscellaneous options for the C and C++ preprocessors and compilers. If it is not set in the environment when `configure` runs, the default value is empty. `configure` uses this variable when compiling or preprocessing programs to test for C and C++ features.

CXXFLAGS [Variable]

Debugging and optimization options for the C++ compiler. If it is not set in the environment when `configure` runs, the default value is set when you call `AC_PROG_CXX` (or empty if you don't). `configure` uses this variable when compiling programs to test for C++ features.

DEFS [Variable]

'`-D`' options to pass to the C compiler. If `AC_CONFIG_HEADERS` is called, `configure` replaces '`@DEFS@`' with '`-DHAVE_CONFIG_H`' instead (see Section 4.8 [Configuration Headers], page 26). This variable is not defined while `configure` is performing its tests, only when creating the output files. See Section 7.2 [Setting Output Variables], page 82, for how to check the results of previous tests.

ECHO_C [Variable]
ECHO_N [Variable]
ECHO_T [Variable]

How does one suppress the trailing newline from `echo` for question-answer message pairs? These variables provide a way:

```
echo $ECHO_N "And the winner is... $ECHO_C"
sleep 100000000000
echo "${ECHO_T}dead."
```

Some old and uncommon `echo` implementations offer no means to achieve this, in which case `ECHO_T` is set to tab. You might not want to use it.

FCFLAGS [Variable]

Debugging and optimization options for the Fortran compiler. If it is not set in the environment when `configure` runs, the default value is set when you call `AC_PROG_FC` (or empty if you don't). `configure` uses this variable when compiling programs to test for Fortran features.

FFLAGS [Variable]

Debugging and optimization options for the Fortran 77 compiler. If it is not set in the environment when `configure` runs, the default value is set when you call `AC_PROG_F77` (or empty if you don't). `configure` uses this variable when compiling programs to test for Fortran 77 features.

LDFLAGS [Variable]

Stripping ('-s'), path ('-L'), and any other miscellaneous options for the linker. Don't use this variable to pass library names ('-l') to the linker, use `LIBS` instead. If it is not set in the environment when `configure` runs, the default value is empty. `configure` uses this variable when linking programs to test for C, C++, and Fortran features.

LIBS [Variable]

'-l' options to pass to the linker. The default value is empty, but some Autoconf macros may prepend extra libraries to this variable if those libraries are found and provide necessary functions, see Section 5.4 [Libraries], page 38. `configure` uses this variable when linking programs to test for C, C++, and Fortran features.

builddir [Variable]

Rigorously equal to '.'. Added for symmetry only.

abs_builddir [Variable]

Absolute path of `builddir`.

top_builddir [Variable]

The relative path to the top-level of the current build tree. In the top-level directory, this is the same as `builddir`.

abs_top_builddir [Variable]

Absolute path of `top_builddir`.

srcdir [Variable]

The relative path to the directory that contains the source code for that 'Makefile'.

abs_srcdir [Variable]

Absolute path of `srcdir`.

top_srcdir [Variable]

The relative path to the top-level source code directory for the package. In the top-level directory, this is the same as `srcdir`.

abs_top_srcdir [Variable]

Absolute path of `top_srcdir`.

4.7.2 Installation Directory Variables

The following variables specify the directories where the package will be installed, see section "Variables for Installation Directories" in *The* GNU *Coding Standards*, for more information. See the end of this section for details on when and how to use these variables.

bindir [Variable]

The directory for installing executables that users run.

datadir [Variable]

The directory for installing read-only architecture-independent data.

exec_prefix [Variable]

The installation prefix for architecture-dependent files. By default it's the same as *prefix*. You should avoid installing anything directly to *exec_prefix*. However, the default value for directories containing architecture-dependent files should be relative to *exec_prefix*.

includedir [Variable]

The directory for installing C header files.

infodir [Variable]

The directory for installing documentation in Info format.

libdir [Variable]

The directory for installing object code libraries.

libexecdir [Variable]

The directory for installing executables that other programs run.

localstatedir [Variable]

The directory for installing modifiable single-machine data.

mandir [Variable]

The top-level directory for installing documentation in man format.

oldincludedir [Variable]

The directory for installing C header files for non-GCC compilers.

prefix [Variable]

The common installation prefix for all files. If *exec_prefix* is defined to a different value, *prefix* is used only for architecture-independent files.

sbindir [Variable]

The directory for installing executables that system administrators run.

sharedstatedir [Variable]

The directory for installing modifiable architecture-independent data.

sysconfdir [Variable]

The directory for installing read-only single-machine data.

Most of these variables have values that rely on `prefix` or `exec_prefix`. It is deliberate that the directory output variables keep them unexpanded: typically '`@datadir@`' will be replaced by '`${prefix}/share`', not '`/usr/local/share`'.

This behavior is mandated by the GNU coding standards, so that when the user runs:

'`make`' she can still specify a different prefix from the one specified to `configure`, in which case, if needed, the package shall hard code dependencies corresponding to the make-specified prefix.

'make install'
: she can specify a different installation location, in which case the package *must* still depend on the location which was compiled in (i.e., never recompile when 'make install' is run). This is an extremely important feature, as many people may decide to install all the files of a package grouped together, and then install links from the final locations to there.

In order to support these features, it is essential that `datadir` remains being defined as '`${prefix}/share`' to depend upon the current value of `prefix`.

A corollary is that you should not use these variables except in Makefiles. For instance, instead of trying to evaluate `datadir` in '`configure`' and hard-coding it in Makefiles using e.g., '`AC_DEFINE_UNQUOTED(DATADIR, "$datadir")`', you should add '`-DDATADIR="$(datadir)"`' to your `CPPFLAGS`.

Similarly you should not rely on `AC_OUTPUT_FILES` to replace `datadir` and friends in your shell scripts and other files, rather let `make` manage their replacement. For instance Autoconf ships templates of its shell scripts ending with '`.in`', and uses a Makefile snippet similar to:

```
edit = sed \
        -e 's,@datadir\@,$(pkgdatadir),g' \
        -e 's,@prefix\@,$(prefix),g'

autoconf: Makefile $(srcdir)/autoconf.in
        rm -f autoconf autoconf.tmp
        $(edit) $(srcdir)/autoconf.in >autoconf.tmp
        chmod +x autoconf.tmp
        mv autoconf.tmp autoconf

autoheader: Makefile $(srcdir)/autoheader.in
        rm -f autoheader autoheader.tmp
        $(edit) $(srcdir)/autoconf.in >autoheader.tmp
        chmod +x autoheader.tmp
        mv autoheader.tmp autoheader
```

Some details are noteworthy:

'`@datadir\@`'
: The backslash prevents `configure` from replacing '`@datadir@`' in the sed expression itself.

'`$(pkgdatadir)`'
: Don't use '`@pkgdatadir@`'! Use the matching makefile variable instead.

'`,`'
: Don't use '`/`' in the sed expression(s) since most likely the variables you use, such as '`$(pkgdatadir)`', will contain some.

'`Dependency on 'Makefile'`'
: Since `edit` uses values that depend on the configuration specific values (`prefix` etc.) and not only on `VERSION` and so forth, the output depends on '`Makefile`', not '`configure.ac`'.

'`Separated dependencies and Single Suffix Rules`'
: You can't use them! The above snippet cannot be (portably) rewritten as:

```
autoconf autoheader: Makefile
.in:
        rm -f $@ $@.tmp
        $(edit) $< >$@.tmp
        chmod +x $@.tmp
        mv $@.tmp $@
```

See Section 10.11 [Limitations of Make], page 138, for details.

``$(srcdir)''
: Be sure to specify the path to the sources, otherwise the package won't support separated builds.

4.7.3 Build Directories

You can support compiling a software package for several architectures simultaneously from the same copy of the source code. The object files for each architecture are kept in their own directory.

To support doing this, `make` uses the `VPATH` variable to find the files that are in the source directory. GNU Make and most other recent `make` programs can do this. Older `make` programs do not support `VPATH`; when using them, the source code must be in the same directory as the object files.

To support `VPATH`, each '`Makefile.in`' should contain two lines that look like:

```
srcdir = @srcdir@
VPATH = @srcdir@
```

Do not set `VPATH` to the value of another variable, for example '`VPATH = $(srcdir)`', because some versions of `make` do not do variable substitutions on the value of `VPATH`.

`configure` substitutes the correct value for `srcdir` when it produces '`Makefile`'.

Do not use the `make` variable `$<`, which expands to the file name of the file in the source directory (found with `VPATH`), except in implicit rules. (An implicit rule is one such as '`.c.o`', which tells how to create a '`.o`' file from a '`.c`' file.) Some versions of `make` do not set `$<` in explicit rules; they expand it to an empty value.

Instead, '`Makefile`' command lines should always refer to source files by prefixing them with '`$(srcdir)/`'. For example:

```
time.info: time.texinfo
        $(MAKEINFO) $(srcdir)/time.texinfo
```

4.7.4 Automatic Remaking

You can put rules like the following in the top-level '`Makefile.in`' for a package to automatically update the configuration information when you change the configuration files. This example includes all of the optional files, such as '`aclocal.m4`' and those related to configuration header files. Omit from the '`Makefile.in`' rules for any of these files that your package does not use.

The '`$(srcdir)/`' prefix is included because of limitations in the `VPATH` mechanism.

The '`stamp-`' files are necessary because the timestamps of '`config.h.in`' and '`config.h`' will not be changed if remaking them does not change their contents. This feature avoids unnecessary recompilation. You should include the file '`stamp-h.in`' your package's distribution, so make will consider 'config.h.in' up to date. Don't use touch (see Section 10.10 [Limitations of Usual Tools], page 131), rather use echo (using date would cause needless differences, hence CVS conflicts etc.).

```
$(srcdir)/configure: configure.ac aclocal.m4
        cd $(srcdir) && autoconf

# autoheader might not change config.h.in, so touch a stamp file.
$(srcdir)/config.h.in: stamp-h.in
$(srcdir)/stamp-h.in: configure.ac aclocal.m4
        cd $(srcdir) && autoheader
        echo timestamp > $(srcdir)/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in config.status
        ./config.status

Makefile: Makefile.in config.status
        ./config.status

config.status: configure
        ./config.status --recheck
```

(Be careful if you copy these lines directly into your Makefile, as you will need to convert the indented lines to start with the tab character.)

In addition, you should use 'AC_CONFIG_FILES([stamp-h], [echo timestamp > stamp-h])' so 'config.status' will ensure that 'config.h' is considered up to date. See Section 4.4 [Output], page 17, for more information about AC_OUTPUT.

See Chapter 14 [config.status Invocation], page 163, for more examples of handling configuration-related dependencies.

4.8 Configuration Header Files

When a package contains more than a few tests that define C preprocessor symbols, the command lines to pass '-D' options to the compiler can get quite long. This causes two problems. One is that the make output is hard to visually scan for errors. More seriously, the command lines can exceed the length limits of some operating systems. As an alternative to passing '-D' options to the compiler, configure scripts can create a C header file containing '#define' directives. The AC_CONFIG_HEADERS macro selects this kind of output. It should be called right after AC_INIT.

The package should '#include' the configuration header file before any other header files, to prevent inconsistencies in declarations (for example, if it redefines const). Use '#include <config.h>' instead of '#include "config.h"', and pass the C compiler a '-I.' option (or '-I..'; whichever directory contains 'config.h'). That way, even if the source directory is configured itself (perhaps to make a distribution), other build directories can also be configured without finding the 'config.h' from the source directory.

AC_CONFIG_HEADERS (*header ...*, [*cmds*], [*init-cmds*]) [Macro]

This macro is one of the instantiating macros; see Section 4.5 [Configuration Actions], page 18. Make `AC_OUTPUT` create the file(s) in the whitespace-separated list *header* containing C preprocessor `#define` statements, and replace '`@DEFS@`' in generated files with '`-DHAVE_CONFIG_H`' instead of the value of `DEFS`. The usual name for *header* is '`config.h`'.

If *header* already exists and its contents are identical to what `AC_OUTPUT` would put in it, it is left alone. Doing this allows making some changes in the configuration without needlessly causing object files that depend on the header file to be recompiled.

Usually the input file is named '*header*`.in`'; however, you can override the input file name by appending to *header* a colon-separated list of input files. Examples:

```
AC_CONFIG_HEADERS([config.h:config.hin])
AC_CONFIG_HEADERS([defines.h:defs.pre:defines.h.in:defs.post])
```

Doing this allows you to keep your file names acceptable to MS-DOS, or to prepend and/or append boilerplate to the file.

See Section 4.5 [Configuration Actions], page 18, for more details on *header*.

4.8.1 Configuration Header Templates

Your distribution should contain a template file that looks as you want the final header file to look, including comments, with `#undef` statements which are used as hooks. For example, suppose your '`configure.ac`' makes these calls:

```
AC_CONFIG_HEADERS([conf.h])
AC_CHECK_HEADERS([unistd.h])
```

Then you could have code like the following in '`conf.h.in`'. On systems that have '`unistd.h`', `configure` will '`#define`' '`HAVE_UNISTD_H`' to 1. On other systems, the whole line will be commented out (in case the system predefines that symbol).

```
/* Define as 1 if you have unistd.h.  */
#undef HAVE_UNISTD_H
```

Pay attention that '`#undef`' is in the first column, and there is nothing behind '`HAVE_UNISTD_H`', not even white spaces. You can then decode the configuration header using the preprocessor directives:

```
#include <conf.h>

#if HAVE_UNISTD_H
# include <unistd.h>
#else
/* We are in trouble.  */
#endif
```

The use of old form templates, with '`#define`' instead of '`#undef`' is strongly discouraged. Similarly with old templates with comments on the same line as the '`#undef`'. Anyway, putting comments in preprocessor macros has never been a good idea.

Since it is a tedious task to keep a template header up to date, you may use `autoheader` to generate it, see Section 4.8.2 [autoheader Invocation], page 28.

4.8.2 Using autoheader to Create 'config.h.in'

The `autoheader` program can create a template file of C '`#define`' statements for `configure` to use. If '`configure.ac`' invokes `AC_CONFIG_HEADERS(file)`, `autoheader` creates '*file*`.in`'; if multiple file arguments are given, the first one is used. Otherwise, `autoheader` creates '`config.h.in`'.

In order to do its job, `autoheader` needs you to document all of the symbols that you might use; i.e., there must be at least one `AC_DEFINE` or one `AC_DEFINE_UNQUOTED` call with a third argument for each symbol (see Section 7.1 [Defining Symbols], page 81). An additional constraint is that the first argument of `AC_DEFINE` must be a literal. Note that all symbols defined by Autoconf's builtin tests are already documented properly; you only need to document those that you define yourself.

You might wonder why `autoheader` is needed: after all, why would `configure` need to "patch" a '`config.h.in`' to produce a '`config.h`' instead of just creating '`config.h`' from scratch? Well, when everything rocks, the answer is just that we are wasting our time maintaining `autoheader`: generating '`config.h`' directly is all that is needed. When things go wrong, however, you'll be thankful for the existence of `autoheader`.

The fact that the symbols are documented is important in order to *check* that '`config.h`' makes sense. The fact that there is a well-defined list of symbols that should be `#define`'d (or not) is also important for people who are porting packages to environments where `configure` cannot be run: they just have to *fill in the blanks*.

But let's come back to the point: `autoheader`'s invocation...

If you give `autoheader` an argument, it uses that file instead of '`configure.ac`' and writes the header file to the standard output instead of to '`config.h.in`'. If you give `autoheader` an argument of '`-`', it reads the standard input instead of '`configure.ac`' and writes the header file to the standard output.

`autoheader` accepts the following options:

'`--help`'
'`-h`' Print a summary of the command line options and exit.

'`--version`'
'`-V`' Print the version number of Autoconf and exit.

'`--verbose`'
'`-v`' Report processing steps.

'`--debug`'
'`-d`' Don't remove the temporary files.

'`--force`'
'`-f`' Remake the template file even if newer than its input files.

'`--include=dir`'
'`-I dir`' Append *dir* to include path. Multiple invocations accumulate.

'`--prepend-include=dir`'
'`-B dir`' Prepend *dir* to include path. Multiple invocations accumulate.

'--warnings=*category*'
'-W *category*'
: Report the warnings related to *category* (which can actually be a comma separated list). Current categories include:

  'obsolete'
  : report the uses of obsolete constructs

  'all' report all the warnings

  'none' report none

  'error' treats warnings as errors

  'no-*category*'
  : disable warnings falling into *category*

4.8.3 Autoheader Macros

`autoheader` scans 'configure.ac' and figures out which C preprocessor symbols it might define. It knows how to generate templates for symbols defined by `AC_CHECK_HEADERS`, `AC_CHECK_FUNCS` etc., but if you `AC_DEFINE` any additional symbol, you must define a template for it. If there are missing templates, `autoheader` fails with an error message.

The simplest way to create a template for a *symbol* is to supply the *description* argument to an '`AC_DEFINE(symbol)`'; see Section 7.1 [Defining Symbols], page 81. You may also use one of the following macros.

AH_VERBATIM (*key, template*) [Macro]

Tell `autoheader` to include the *template* as-is in the header template file. This *template* is associated with the *key*, which is used to sort all the different templates and guarantee their uniqueness. It should be a symbol that can be `AC_DEFINE`'d.

For example:

```
AH_VERBATIM([_GNU_SOURCE],
[/* Enable GNU extensions on systems that have them.  */
#ifndef _GNU_SOURCE
# define _GNU_SOURCE
#endif])
```

AH_TEMPLATE (*key, description*) [Macro]

Tell `autoheader` to generate a template for *key*. This macro generates standard templates just like `AC_DEFINE` when a *description* is given.

For example:

```
AH_TEMPLATE([CRAY_STACKSEG_END],
            [Define to one of _getb67, GETB67, getb67
             for Cray-2 and Cray-YMP systems.  This
             function is required for alloca.c support
             on those systems.])
```

will generate the following template, with the description properly justified.

```
/* Define to one of _getb67, GETB67, getb67 for Cray-2 and
   Cray-YMP systems.  This function is required for alloca.c
   support on those systems.  */
#undef CRAY_STACKSEG_END
```

AH_TOP (*text*) [Macro]

Include *text* at the top of the header template file.

AH_BOTTOM (*text*) [Macro]

Include *text* at the bottom of the header template file.

4.9 Running Arbitrary Configuration Commands

You can execute arbitrary commands before, during, and after 'config.status' is run. The three following macros accumulate the commands to run when they are called multiple times. AC_CONFIG_COMMANDS replaces the obsolete macro AC_OUTPUT_COMMANDS; see Section 15.4 [Obsolete Macros], page 167, for details.

AC_CONFIG_COMMANDS (*tag*. . ., [*cmds*], [*init-cmds*]) [Macro]

Specify additional shell commands to run at the end of 'config.status', and shell commands to initialize any variables from configure. Associate the commands with *tag*. Since typically the *cmds* create a file, *tag* should naturally be the name of that file. If needed, the directory hosting *tag* is created. This macro is one of the instantiating macros; see Section 4.5 [Configuration Actions], page 18.

Here is an unrealistic example:

```
fubar=42
AC_CONFIG_COMMANDS([fubar],
                   [echo this is extra $fubar, and so on.],
                   [fubar=$fubar])
```

Here is a better one:

```
AC_CONFIG_COMMANDS([time-stamp], [date >time-stamp])
```

AC_CONFIG_COMMANDS_PRE (*cmds*) [Macro]

Execute the *cmds* right before creating 'config.status'.

AC_CONFIG_COMMANDS_POST (*cmds*) [Macro]

Execute the *cmds* right after creating 'config.status'.

4.10 Creating Configuration Links

You may find it convenient to create links whose destinations depend upon results of tests. One can use AC_CONFIG_COMMANDS but the creation of relative symbolic links can be delicate when the package is built in a directory different from the source directory.

AC_CONFIG_LINKS (*dest:source*. . ., [*cmds*], [*init-cmds*]) [Macro]

Make AC_OUTPUT link each of the existing files *source* to the corresponding link name *dest*. Makes a symbolic link if possible, otherwise a hard link if possible, otherwise a copy. The *dest* and *source* names should be relative to the top level source or build directory. This macro is one of the instantiating macros; see Section 4.5 [Configuration Actions], page 18.

For example, this call:

```
AC_CONFIG_LINKS(host.h:config/$machine.h
                object.h:config/$obj_format.h)
```

creates in the current directory '`host.h`' as a link to '*srcdir*`/config/$machine.h`', and '`object.h`' as a link to '*srcdir*`/config/$obj_format.h`'.

The tempting value '`.`' for *dest* is invalid: it makes it impossible for '`config.status`' to guess the links to establish.

One can then run:

```
./config.status host.h object.h
```

to create the links.

4.11 Configuring Other Packages in Subdirectories

In most situations, calling `AC_OUTPUT` is sufficient to produce '`Makefile`'s in subdirectories. However, `configure` scripts that control more than one independent package can use `AC_CONFIG_SUBDIRS` to run `configure` scripts for other packages in subdirectories.

AC_CONFIG_SUBDIRS (*dir* ...) [Macro]

Make `AC_OUTPUT` run `configure` in each subdirectory *dir* in the given whitespace-separated list. Each *dir* should be a literal, i.e., please do not use:

```
if test "$package_foo_enabled" = yes; then
  $my_subdirs="$my_subdirs foo"
fi
AC_CONFIG_SUBDIRS($my_subdirs)
```

because this prevents '`./configure --help=recursive`' from displaying the options of the package `foo`. Rather, you should write:

```
if test "$package_foo_enabled" = yes; then
  AC_CONFIG_SUBDIRS(foo)
fi
```

If a given *dir* is not found, an error is reported: if the subdirectory is optional, write:

```
if test -d $srcdir/foo; then
  AC_CONFIG_SUBDIRS(foo)
fi
```

If a given *dir* contains `configure.gnu`, it is run instead of `configure`. This is for packages that might use a non-Autoconf script `Configure`, which can't be called through a wrapper `configure` since it would be the same file on case-insensitive filesystems. Likewise, if a *dir* contains '`configure.in`' but no `configure`, the Cygnus `configure` script found by `AC_CONFIG_AUX_DIR` is used.

The subdirectory `configure` scripts are given the same command line options that were given to this `configure` script, with minor changes if needed, which include:

- adjusting a relative path for the cache file;
- adjusting a relative path for the source directory;

   – propagating the current value of `$prefix`, including if it was defaulted, and if the default values of the top level and of the subdirectory '`configure`' differ.

This macro also sets the output variable `subdirs` to the list of directories '*dir* ...'. '`Makefile`' rules can use this variable to determine which subdirectories to recurse into.

This macro may be called multiple times.

4.12 Default Prefix

By default, `configure` sets the prefix for files it installs to '`/usr/local`'. The user of `configure` can select a different prefix using the '`--prefix`' and '`--exec-prefix`' options. There are two ways to change the default: when creating `configure`, and when running it.

Some software packages might want to install in a directory other than '`/usr/local`' by default. To accomplish that, use the `AC_PREFIX_DEFAULT` macro.

AC_PREFIX_DEFAULT (*prefix*) [Macro]

Set the default installation prefix to *prefix* instead of '`/usr/local`'.

It may be convenient for users to have `configure` guess the installation prefix from the location of a related program that they have already installed. If you wish to do that, you can call `AC_PREFIX_PROGRAM`.

AC_PREFIX_PROGRAM (*program*) [Macro]

If the user did not specify an installation prefix (using the '`--prefix`' option), guess a value for it by looking for *program* in `PATH`, the way the shell does. If *program* is found, set the prefix to the parent of the directory containing *program*, else default the prefix as described above ('`/usr/local`' or `AC_PREFIX_DEFAULT`). For example, if *program* is `gcc` and the `PATH` contains '`/usr/local/gnu/bin/gcc`', set the prefix to '`/usr/local/gnu`'.

5 Existing Tests

These macros test for particular system features that packages might need or want to use. If you need to test for a kind of feature that none of these macros check for, you can probably do it by calling primitive test macros with appropriate arguments (see Chapter 6 [Writing Tests], page 71).

These tests print messages telling the user which feature they're checking for, and what they find. They cache their results for future `configure` runs (see Section 7.3 [Caching Results], page 83).

Some of these macros set output variables. See Section 4.7 [Makefile Substitutions], page 20, for how to get their values. The phrase "define *name*" is used below as a shorthand to mean "define C preprocessor symbol *name* to the value 1". See Section 7.1 [Defining Symbols], page 81, for how to get those symbol definitions into your program.

5.1 Common Behavior

Much effort has been expended to make Autoconf easy to learn. The most obvious way to reach this goal is simply to enforce standard interfaces and behaviors, avoiding exceptions as much as possible. Because of history and inertia, unfortunately, there are still too many exceptions in Autoconf; nevertheless, this section describes some of the common rules.

5.1.1 Standard Symbols

All the generic macros that `AC_DEFINE` a symbol as a result of their test transform their *arguments* to a standard alphabet. First, *argument* is converted to upper case and any asterisks ('`*`') are each converted to '`P`'. Any remaining characters that are not alphanumeric are converted to underscores.

For instance,

```
AC_CHECK_TYPES(struct $Expensive*)
```

will define the symbol '`HAVE_STRUCT__EXPENSIVEP`' if the check succeeds.

5.1.2 Default Includes

Several tests depend upon a set of header files. Since these headers are not universally available, tests actually have to provide a set of protected includes, such as:

```
#if TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
#  include <sys/time.h>
# else
#  include <time.h>
# endif
#endif
```

Unless you know exactly what you are doing, you should avoid using unconditional includes, and check the existence of the headers you include beforehand (see Section 5.6 [Header Files], page 48).

Most generic macros use the following macro to provide the default set of includes:

AC_DEFAULT_INCLUDES ([*include-directives*]) [Macro]

Expand to *include-directives* if defined, otherwise to:

```
#include <stdio.h>
#if HAVE_SYS_TYPES_H
# include <sys/types.h>
#endif
#if HAVE_SYS_STAT_H
# include <sys/stat.h>
#endif
#if STDC_HEADERS
# include <stdlib.h>
# include <stddef.h>
#else
# if HAVE_STDLIB_H
#  include <stdlib.h>
# endif
#endif
#if HAVE_STRING_H
# if !STDC_HEADERS && HAVE_MEMORY_H
#  include <memory.h>
# endif
# include <string.h>
#endif
#if HAVE_STRINGS_H
# include <strings.h>
#endif
#if HAVE_INTTYPES_H
# include <inttypes.h>
#else
# if HAVE_STDINT_H
#  include <stdint.h>
# endif
#endif
#if HAVE_UNISTD_H
# include <unistd.h>
#endif
```

If the default includes are used, then check for the presence of these headers and their compatibility, i.e., you don't need to run `AC_HEADERS_STDC`, nor check for '`stdlib.h`' etc.

These headers are checked for in the same order as they are included. For instance, on some systems '`string.h`' and '`strings.h`' both exist, but conflict. Then `HAVE_STRING_H` will be defined, but `HAVE_STRINGS_H` won't.

5.2 Alternative Programs

These macros check for the presence or behavior of particular programs. They are used to choose between several alternative programs and to decide what to do once one has been chosen. If there is no macro specifically defined to check for a program you need, and you don't need to check for any special properties of it, then you can use one of the general program-check macros.

5.2.1 Particular Program Checks

These macros check for particular programs—whether they exist, and in some cases whether they support certain features.

AC_PROG_AWK [Macro]

Check for `gawk`, `mawk`, `nawk`, and `awk`, in that order, and set output variable `AWK` to the first one that is found. It tries `gawk` first because that is reported to be the best implementation.

AC_PROG_EGREP [Macro]

Check for `grep -E` and `egrep`, in that order, and set output variable `EGREP` to the first one that is found.

AC_PROG_FGREP [Macro]

Check for `grep -F` and `fgrep`, in that order, and set output variable `FGREP` to the first one that is found.

AC_PROG_INSTALL [Macro]

Set output variable `INSTALL` to the path of a BSD-compatible `install` program, if one is found in the current `PATH`. Otherwise, set `INSTALL` to '*dir*`/install-sh -c`', checking the directories specified to `AC_CONFIG_AUX_DIR` (or its default directories) to determine *dir* (see Section 4.4 [Output], page 17). Also set the variables `INSTALL_PROGRAM` and `INSTALL_SCRIPT` to '`${INSTALL}`' and `INSTALL_DATA` to '`${INSTALL} -m 644`'.

This macro screens out various instances of `install` known not to work. It prefers to find a C program rather than a shell script, for speed. Instead of '`install-sh`', it can also use '`install.sh`', but that name is obsolete because some `make` programs have a rule that creates '`install`' from it if there is no '`Makefile`'.

Autoconf comes with a copy of '`install-sh`' that you can use. If you use `AC_PROG_INSTALL`, you must include either '`install-sh`' or '`install.sh`' in your distribution, or `configure` will produce an error message saying it can't find them—even if the system you're on has a good `install` program. This check is a safety measure to prevent you from accidentally leaving that file out, which would prevent your package from installing on systems that don't have a BSD-compatible `install` program.

If you need to use your own installation program because it has features not found in standard `install` programs, there is no reason to use `AC_PROG_INSTALL`; just put the file name of your program into your '`Makefile.in`' files.

AC_PROG_LEX [Macro]

If `flex` is found, set output variable `LEX` to '`flex`' and `LEXLIB` to '`-lfl`', if that library is in a standard place. Otherwise set `LEX` to '`lex`' and `LEXLIB` to '`-ll`'.

Define YYTEXT_POINTER if yytext is a 'char *' instead of a 'char []'. Also set output variable LEX_OUTPUT_ROOT to the base of the file name that the lexer generates; usually 'lex.yy', but sometimes something else. These results vary according to whether lex or flex is being used.

You are encouraged to use Flex in your sources, since it is both more pleasant to use than plain Lex and the C source it produces is portable. In order to ensure portability, however, you must either provide a function yywrap or, if you don't use it (e.g., your scanner has no '#include'-like feature), simply include a '%noyywrap' statement in the scanner's source. Once this done, the scanner is portable (unless *you* felt free to use nonportable constructs) and does not depend on any library. In this case, and in this case only, it is suggested that you use this Autoconf snippet:

```
AC_PROG_LEX
if test "$LEX" != flex; then
  LEX="$SHELL $missing_dir/missing flex"
  AC_SUBST(LEX_OUTPUT_ROOT, lex.yy)
  AC_SUBST(LEXLIB, '')
fi
```

The shell script missing can be found in the Automake distribution.

To ensure backward compatibility, Automake's AM_PROG_LEX invokes (indirectly) this macro twice, which will cause an annoying but benign "AC_PROG_LEX invoked multiple times" warning. Future versions of Automake will fix this issue; meanwhile, just ignore this message.

AC_PROG_LN_S [Macro]

If 'ln -s' works on the current file system (the operating system and file system support symbolic links), set the output variable LN_S to 'ln -s'; otherwise, if 'ln' works, set LN_S to 'ln', and otherwise set it to 'cp -p'.

If you make a link in a directory other than the current directory, its meaning depends on whether 'ln' or 'ln -s' is used. To safely create links using '$(LN_S)', either find out which form is used and adjust the arguments, or always invoke ln in the directory where the link is to be created.

In other words, it does not work to do:

```
$(LN_S) foo /x/bar
```

Instead, do:

```
(cd /x && $(LN_S) foo bar)
```

AC_PROG_RANLIB [Macro]

Set output variable RANLIB to 'ranlib' if ranlib is found, and otherwise to ':' (do nothing).

AC_PROG_YACC [Macro]

If bison is found, set output variable YACC to 'bison -y'. Otherwise, if byacc is found, set YACC to 'byacc'. Otherwise set YACC to 'yacc'.

5.2.2 Generic Program and File Checks

These macros are used to find programs not covered by the "particular" test macros. If you need to check the behavior of a program as well as find out whether it is present, you have to write your own test for it (see Chapter 6 [Writing Tests], page 71). By default, these macros use the environment variable `PATH`. If you need to check for a program that might not be in the user's `PATH`, you can pass a modified path to use instead, like this:

```
AC_PATH_PROG([INETD], [inetd], [/usr/libexec/inetd],
             [$PATH:/usr/libexec:/usr/sbin:/usr/etc:etc])
```

You are strongly encouraged to declare the *variable* passed to `AC_CHECK_PROG` etc. as precious, See Section 7.2 [Setting Output Variables], page 82, `AC_ARG_VAR`, for more details.

AC_CHECK_PROG (*variable, prog-to-check-for, value-if-found,* [*value-if-not-found*], [*path*], [*reject*]) [Macro]

Check whether program *prog-to-check-for* exists in `PATH`. If it is found, set *variable* to *value-if-found*, otherwise to *value-if-not-found*, if given. Always pass over *reject* (an absolute file name) even if it is the first found in the search path; in that case, set *variable* using the absolute file name of the *prog-to-check-for* found that is not *reject*. If *variable* was already set, do nothing. Calls `AC_SUBST` for *variable*.

AC_CHECK_PROGS (*variable, progs-to-check-for,* [*value-if-not-found*], [*path*]) [Macro]

Check for each program in the whitespace-separated list *progs-to-check-for* existing in the `PATH`. If one is found, set *variable* to the name of that program. Otherwise, continue checking the next program in the list. If none of the programs in the list are found, set *variable* to *value-if-not-found*; if *value-if-not-found* is not specified, the value of *variable* is not changed. Calls `AC_SUBST` for *variable*.

AC_CHECK_TOOL (*variable, prog-to-check-for,* [*value-if-not-found*], [*path*]) [Macro]

Like `AC_CHECK_PROG`, but first looks for *prog-to-check-for* with a prefix of the host type as determined by `AC_CANONICAL_HOST`, followed by a dash (see Section 11.2 [Canonicalizing], page 150). For example, if the user runs '`configure --host=i386-gnu`', then this call:

```
AC_CHECK_TOOL(RANLIB, ranlib, :)
```

sets `RANLIB` to '`i386-gnu-ranlib`' if that program exists in `PATH`, or otherwise to '`ranlib`' if that program exists in `PATH`, or to '`:`' if neither program exists.

AC_CHECK_TOOLS (*variable, progs-to-check-for,* [*value-if-not-found*], [*path*]) [Macro]

Like `AC_CHECK_TOOL`, each of the tools in the list *progs-to-check-for* are checked with a prefix of the host type as determined by `AC_CANONICAL_HOST`, followed by a dash (see Section 11.2 [Canonicalizing], page 150). If none of the tools can be found with a prefix, then the first one without a prefix is used. If a tool is found, set *variable* to the name of that program. If none of the tools in the list are found, set *variable* to *value-if-not-found*; if *value-if-not-found* is not specified, the value of *variable* is not changed. Calls `AC_SUBST` for *variable*.

AC_PATH_PROG (*variable, prog-to-check-for,* [*value-if-not-found*], [*path*]) [Macro]

Like `AC_CHECK_PROG`, but set *variable* to the entire path of *prog-to-check-for* if found.

AC_PATH_PROGS (*variable, progs-to-check-for,* [*value-if-not-found*], [*path*]) [Macro]
Like `AC_CHECK_PROGS`, but if any of *progs-to-check-for* are found, set *variable* to the entire path of the program found.

AC_PATH_TOOL (*variable, prog-to-check-for,* [*value-if-not-found*], [*path*]) [Macro]
Like `AC_CHECK_TOOL`, but set *variable* to the entire path of the program if it is found.

5.3 Files

You might also need to check for the existence of files. Before using these macros, ask yourself whether a run-time test might not be a better solution. Be aware that, like most Autoconf macros, they test a feature of the host machine, and therefore, they die when cross-compiling.

AC_CHECK_FILE (*file,* [*action-if-found*], [*action-if-not-found*]) [Macro]
Check whether file *file* exists on the native system. If it is found, execute *action-if-found*, otherwise do *action-if-not-found*, if given.

AC_CHECK_FILES (*files,* [*action-if-found*], [*action-if-not-found*]) [Macro]
Executes `AC_CHECK_FILE` once for each file listed in *files*. Additionally, defines '`HAVE_file`' (see Section 5.1.1 [Standard Symbols], page 33) for each file found.

5.4 Library Files

The following macros check for the presence of certain C, C++, or Fortran library archive files.

AC_CHECK_LIB (*library, function,* [*action-if-found*], [*action-if-not-found*], [*other-libraries*]) [Macro]
Depending on the current language(see Section 6.1 [Language Choice], page 71), try to ensure that the C, C++, or Fortran function *function* is available by checking whether a test program can be linked with the library *library* to get the function. *library* is the base name of the library; e.g., to check for '`-lmp`', use '`mp`' as the *library* argument.

action-if-found is a list of shell commands to run if the link with the library succeeds; *action-if-not-found* is a list of shell commands to run if the link fails. If *action-if-found* is not specified, the default action will prepend '`-llibrary`' to `LIBS` and define '`HAVE_LIBlibrary`' (in all capitals). This macro is intended to support building `LIBS` in a right-to-left (least-dependent to most-dependent) fashion such that library dependencies are satisfied as a natural side-effect of consecutive tests. Some linkers are very sensitive to library ordering so the order in which `LIBS` is generated is important to reliable detection of libraries.

If linking with *library* results in unresolved symbols that would be resolved by linking with additional libraries, give those libraries as the *other-libraries* argument, separated by spaces: e.g., '`-lXt -lX11`'. Otherwise, this macro will fail to detect that *library* is present, because linking the test program will always fail with unresolved symbols. The *other-libraries* argument should be limited to cases where it is desirable to test for one library in the presence of another that is not already in `LIBS`.

AC_SEARCH_LIBS (*function*, *search-libs*, [*action-if-found*], [*action-if-not-found*], [*other-libraries*]) [Macro]

Search for a library defining *function* if it's not already available. This equates to calling '`AC_LINK_IFELSE([AC_LANG_CALL([], [function])])`' first with no libraries, then for each library listed in *search-libs*.

Add '`-llibrary`' to `LIBS` for the first library found to contain *function*, and run *action-if-found*. If the function is not found, run *action-if-not-found*.

If linking with *library* results in unresolved symbols that would be resolved by linking with additional libraries, give those libraries as the *other-libraries* argument, separated by spaces: e.g., '`-lXt -lX11`'. Otherwise, this macro will fail to detect that *function* is present, because linking the test program will always fail with unresolved symbols.

5.5 Library Functions

The following macros check for particular C library functions. If there is no macro specifically defined to check for a function you need, and you don't need to check for any special properties of it, then you can use one of the general function-check macros.

5.5.1 Portability of C Functions

Most usual functions can either be missing, or be buggy, or be limited on some architectures. This section tries to make an inventory of these portability issues. By definition, this list will always require additions. Please help us keeping it as complete as possible.

`exit` Did you know that, on some older hosts, `exit` returns `int`? This is because `exit` predates `void`, and there was a long tradition of it returning `int`.

`putenv` POSIX specifies that `putenv` puts the given string directly in `environ`, but some systems make a copy of it instead (eg. glibc 2.0, or BSD). And when a copy is made, `unsetenv` might not free it, causing a memory leak (eg. FreeBSD 4).

POSIX specifies that `putenv("FOO")` removes '`FOO`' from the environment, but on some systems (eg. FreeBSD 4) this is not the case and instead `unsetenv` must be used.

On MINGW, a call `putenv("FOO=")` removes '`FOO`' from the environment, rather than inserting it with an empty value.

`signal` handler

Normally `signal` takes a handler function with a return type of `void`, but some old systems required `int` instead. Any actual `int` value returned is not used, this is only a difference in the function prototype demanded.

All systems we know of in current use take `void`. Presumably `int` was to support K&R C, where of course `void` is not available. `AC_TYPE_SIGNAL` (see Section 5.9.1 [Particular Types], page 56) can be used to establish the correct type in all cases.

`snprintf` The ISO C99 standard says that if the output array isn't big enough and if no other errors occur, `snprintf` and `vsnprintf` truncate the output and return the number of bytes that ought to have been produced. Some older systems return the truncated length (e.g., GNU C Library 2.0.x or IRIX 6.5), some a negative value (e.g., earlier GNU C Library versions), and some the buffer length without truncation (e.g., 32-bit Solaris 7). Also, some buggy older systems ignore the length and overrun the buffer (e.g., 64-bit Solaris 7).

`sprintf`
: The ISO C standard says `sprintf` and `vsprintf` return the number of bytes written, but on some old systems (SunOS 4 for instance) they return the buffer pointer instead.

`sscanf`
: On various old systems, e.g., HP-UX 9, `sscanf` requires that its input string be writable (though it doesn't actually change it). This can be a problem when using `gcc` since it normally puts constant strings in read-only memory (see section "Incompatibilities" in *Using and Porting the* GNU *Compiler Collection*). Apparently in some cases even having format strings read-only can be a problem.

`strnlen`
: AIX 4.3 provides a broken version which produces the following results:

```
strnlen ("foobar", 0) = 0
strnlen ("foobar", 1) = 3
strnlen ("foobar", 2) = 2
strnlen ("foobar", 3) = 1
strnlen ("foobar", 4) = 0
strnlen ("foobar", 5) = 6
strnlen ("foobar", 6) = 6
strnlen ("foobar", 7) = 6
strnlen ("foobar", 8) = 6
strnlen ("foobar", 9) = 6
```

`sysconf`
: `_SC_PAGESIZE` is standard, but some older systems (eg. HP-UX 9) have `_SC_PAGE_SIZE` instead. This can be tested with `#ifdef`.

`unlink`
: The POSIX spec says that `unlink` causes the given file to be removed only after there are no more open file handles for it. Not all OS's support this behavior though. So even on systems that provide `unlink`, you cannot portably assume it is OK to call it on files that are open. For example, on Windows 9x and ME, such a call would fail; on DOS it could even lead to file system corruption, as the file might end up being written to after the OS has removed it.

`unsetenv`
: On MINGW, `unsetenv` is not available, but a variable '`FOO`' can be removed with a call `putenv("FOO=")`, as described under `putenv` above.

`va_copy`
: The ISO C99 standard provides `va_copy` for copying `va_list` variables. It may be available in older environments too, though possibly as `__va_copy` (e.g., `gcc` in strict C89 mode). These can be tested with `#ifdef`. A fallback to `memcpy (&dst, &src, sizeof(va_list))` will give maximum portability.

`va_list`
: `va_list` is not necessarily just a pointer. It can be a `struct` (e.g., `gcc` on Alpha), which means `NULL` is not portable. Or it can be an array (e.g., `gcc` in some PowerPC configurations), which means as a function parameter it can be effectively call-by-reference and library routines might modify the value back in the caller (e.g., `vsnprintf` in the GNU C Library 2.1).

Signed >> Normally the C >> right shift of a signed type replicates the high bit, giving a so-called "arithmetic" shift. But care should be taken since the ISO C standard doesn't require that behavior. On those few processors without a native arithmetic shift (for instance Cray vector systems) zero bits may be shifted in, the same as a shift of an unsigned type.

5.5.2 Particular Function Checks

These macros check for particular C functions—whether they exist, and in some cases how they respond when given certain arguments.

AC_FUNC_ALLOCA [Macro]

Check how to get `alloca`. Tries to get a builtin version by checking for 'alloca.h' or the predefined C preprocessor macros `__GNUC__` and `_AIX`. If this macro finds 'alloca.h', it defines `HAVE_ALLOCA_H`.

If those attempts fail, it looks for the function in the standard C library. If any of those methods succeed, it defines `HAVE_ALLOCA`. Otherwise, it sets the output variable `ALLOCA` to 'alloca.o' and defines `C_ALLOCA` (so programs can periodically call 'alloca(0)' to garbage collect). This variable is separate from `LIBOBJS` so multiple programs can share the value of `ALLOCA` without needing to create an actual library, in case only some of them use the code in `LIBOBJS`.

This macro does not try to get `alloca` from the System V R3 'libPW' or the System V R4 'libucb' because those libraries contain some incompatible functions that cause trouble. Some versions do not even contain `alloca` or contain a buggy version. If you still want to use their `alloca`, use `ar` to extract 'alloca.o' from them instead of compiling 'alloca.c'.

Source files that use `alloca` should start with a piece of code like the following, to declare it properly. In some versions of AIX, the declaration of `alloca` must precede everything else except for comments and preprocessor directives. The `#pragma` directive is indented so that pre-ANSI C compilers will ignore it, rather than choke on it.

```
/* AIX requires this to be the first thing in the file.  */
#ifndef __GNUC__
# if HAVE_ALLOCA_H
#  include <alloca.h>
# else
#  ifdef _AIX
 #pragma alloca
#  else
#   ifndef alloca /* predefined by HP cc +Olibcalls */
char *alloca ();
#   endif
#  endif
# endif
#endif
```

AC_FUNC_CHOWN [Macro]

If the `chown` function is available and works (in particular, it should accept '`-1`' for `uid` and `gid`), define `HAVE_CHOWN`.

AC_FUNC_CLOSEDIR_VOID [Macro]

If the `closedir` function does not return a meaningful value, define `CLOSEDIR_VOID`. Otherwise, callers ought to check its return value for an error indicator.

AC_FUNC_ERROR_AT_LINE [Macro]

If the `error_at_line` function is not found, require an `AC_LIBOBJ` replacement of '`error`'.

AC_FUNC_FNMATCH [Macro]

If the `fnmatch` function conforms to POSIX, define `HAVE_FNMATCH`. Detect common implementation bugs, for example, the bugs in Solaris 2.4.

Note that for historical reasons, contrary to the other specific `AC_FUNC` macros, `AC_FUNC_FNMATCH` does not replace a broken/missing `fnmatch`. See `AC_REPLACE_FNMATCH` below.

AC_FUNC_FNMATCH_GNU [Macro]

Behave like `AC_REPLACE_FNMATCH` (*replace*) but also test whether `fnmatch` supports GNU extensions. Detect common implementation bugs, for example, the bugs in the GNU C Library 2.1.

AC_FUNC_FORK [Macro]

This macro checks for the `fork` and `vfork` functions. If a working `fork` is found, define `HAVE_WORKING_FORK`. This macro checks whether `fork` is just a stub by trying to run it.

If '`vfork.h`' is found, define `HAVE_VFORK_H`. If a working `vfork` is found, define `HAVE_WORKING_VFORK`. Otherwise, define `vfork` to be `fork` for backward compatibility with previous versions of `autoconf`. This macro checks for several known errors in implementations of `vfork` and considers the system to not have a working `vfork` if it detects any of them. It is not considered to be an implementation error if a child's invocation of `signal` modifies the parent's signal handler, since child processes rarely change their signal handlers.

Since this macro defines `vfork` only for backward compatibility with previous versions of `autoconf` you're encouraged to define it yourself in new code:

```
#if !HAVE_WORKING_VFORK
# define vfork fork
#endif
```

AC_FUNC_FSEEKO [Macro]

If the `fseeko` function is available, define `HAVE_FSEEKO`. Define `_LARGEFILE_SOURCE` if necessary to make the prototype visible on some systems (e.g. glibc 2.2). Otherwise linkage problems may occur when compiling with `AC_SYS_LARGEFILE` on largefile-sensitive systems where `off_t` does not default to a 64bit entity.

AC_FUNC_GETGROUPS [Macro]

If the `getgroups` function is available and works (unlike on Ultrix 4.3, where '`getgroups (0, 0)`' always fails), define `HAVE_GETGROUPS`. Set `GETGROUPS_LIBS` to any libraries needed to get that function. This macro runs `AC_TYPE_GETGROUPS`.

AC_FUNC_GETLOADAVG [Macro]

Check how to get the system load averages. To perform its tests properly, this macro needs the file '`getloadavg.c`'; therefore, be sure to set the `AC_LIBOBJ` replacement directory properly (see Section 5.5.3 [Generic Functions], page 46, `AC_CONFIG_LIBOBJ_DIR`).

If the system has the `getloadavg` function, define `HAVE_GETLOADAVG`, and set `GETLOADAVG_LIBS` to any libraries needed to get that function. Also add `GETLOADAVG_LIBS` to `LIBS`. Otherwise, require an `AC_LIBOBJ` replacement for '`getloadavg`' with source code in '*`dir`*`/getloadavg.c`', and possibly define several other C preprocessor macros and output variables:

1. Define `C_GETLOADAVG`.
2. Define `SVR4`, `DGUX`, `UMAX`, or `UMAX4_3` if on those systems.
3. If '`nlist.h`' is found, define `HAVE_NLIST_H`.
4. If '`struct nlist`' has an '`n_un.n_name`' member, define `HAVE_STRUCT_NLIST_N_UN_N_NAME`. The obsolete symbol `NLIST_NAME_UNION` is still defined, but do not depend upon it.
5. Programs may need to be installed setgid (or setuid) for `getloadavg` to work. In this case, define `GETLOADAVG_PRIVILEGED`, set the output variable `NEED_SETGID` to '`true`' (and otherwise to '`false`'), and set `KMEM_GROUP` to the name of the group that should own the installed program.

AC_FUNC_GETMNTENT [Macro]

Check for `getmntent` in the '`sun`', '`seq`', and '`gen`' libraries, for IRIX 4, PTX, and Unixware, respectively. Then, if `getmntent` is available, define `HAVE_GETMNTENT`.

AC_FUNC_GETPGRP [Macro]

Define `GETPGRP_VOID` if it is an error to pass 0 to `getpgrp`; this is the POSIX behavior. On older BSD systems, you must pass 0 to `getpgrp`, as it takes an argument and behaves like POSIX's `getpgid`.

```
#if GETPGRP_VOID
  pid = getpgrp ();
#else
  pid = getpgrp (0);
#endif
```

This macro does not check whether `getpgrp` exists at all; if you need to work in that situation, first call `AC_CHECK_FUNC` for `getpgrp`.

AC_FUNC_LSTAT_FOLLOWS_SLASHED_SYMLINK [Macro]

If '`link`' is a symbolic link, then `lstat` should treat '`link/`' the same as '`link/.`'. However, many older `lstat` implementations incorrectly ignore trailing slashes.

It is safe to assume that if `lstat` incorrectly ignores trailing slashes, then other symbolic-link-aware functions like `unlink` also incorrectly ignore trailing slashes.

If lstat behaves properly, define LSTAT_FOLLOWS_SLASHED_SYMLINK, otherwise require an AC_LIBOBJ replacement of lstat.

AC_FUNC_MALLOC [Macro]

If the malloc function is compatible with the GNU C library malloc (i.e., 'malloc (0)' returns a valid pointer), define HAVE_MALLOC to 1. Otherwise define HAVE_MALLOC to 0, ask for an AC_LIBOBJ replacement for 'malloc', and define malloc to rpl_malloc so that the native malloc is not used in the main project.

Typically, the replacement file 'malloc.c' should look like (note the '#undef malloc'):

```
#if HAVE_CONFIG_H
# include <config.h>
#endif
#undef malloc

#include <sys/types.h>

void *malloc ();

/* Allocate an N-byte block of memory from the heap.
   If N is zero, allocate a 1-byte block.  */

void *
rpl_malloc (size_t n)
{
  if (n == 0)
    n = 1;
  return malloc (n);
}
```

AC_FUNC_MEMCMP [Macro]

If the memcmp function is not available, or does not work on 8-bit data (like the one on SunOS 4.1.3), or fails when comparing 16 bytes or more and with at least one buffer not starting on a 4-byte boundary (such as the one on NeXT x86 OpenStep), require an AC_LIBOBJ replacement for 'memcmp'.

AC_FUNC_MBRTOWC [Macro]

Define HAVE_MBRTOWC to 1 if the function mbrtowc and the type mbstate_t are properly declared.

AC_FUNC_MKTIME [Macro]

If the mktime function is not available, or does not work correctly, require an AC_LIBOBJ replacement for 'mktime'. For the purposes of this test, mktime should conform to the POSIX standard and should be the inverse of localtime.

AC_FUNC_MMAP [Macro]

If the mmap function exists and works correctly, define HAVE_MMAP. Only checks private fixed mapping of already-mapped memory.

AC_FUNC_OBSTACK [Macro]

If the obstacks are found, define `HAVE_OBSTACK`, else require an `AC_LIBOBJ` replacement for '`obstack`'.

AC_FUNC_REALLOC [Macro]

If the `realloc` function is compatible with the GNU C library `realloc` (i.e., '`realloc (0, 0)`' returns a valid pointer), define `HAVE_REALLOC` to 1. Otherwise define `HAVE_REALLOC` to 0, ask for an `AC_LIBOBJ` replacement for '`realloc`', and define `realloc` to `rpl_realloc` so that the native `realloc` is not used in the main project. See `AC_FUNC_MALLOC` for details.

AC_FUNC_SELECT_ARGTYPES [Macro]

Determines the correct type to be passed for each of the `select` function's arguments, and defines those types in `SELECT_TYPE_ARG1`, `SELECT_TYPE_ARG234`, and `SELECT_TYPE_ARG5` respectively. `SELECT_TYPE_ARG1` defaults to '`int`', `SELECT_TYPE_ARG234` defaults to '`int *`', and `SELECT_TYPE_ARG5` defaults to '`struct timeval *`'.

AC_FUNC_SETPGRP [Macro]

If `setpgrp` takes no argument (the POSIX version), define `SETPGRP_VOID`. Otherwise, it is the BSD version, which takes two process IDs as arguments. This macro does not check whether `setpgrp` exists at all; if you need to work in that situation, first call `AC_CHECK_FUNC` for `setpgrp`.

AC_FUNC_STAT [Macro]
AC_FUNC_LSTAT [Macro]

Determine whether `stat` or `lstat` have the bug that it succeeds when given the zero-length file name as argument. The `stat` and `lstat` from SunOS 4.1.4 and the Hurd (as of 1998-11-01) do this.

If it does, then define `HAVE_STAT_EMPTY_STRING_BUG` (or `HAVE_LSTAT_EMPTY_STRING_BUG`) and ask for an `AC_LIBOBJ` replacement of it.

AC_FUNC_SETVBUF_REVERSED [Macro]

If `setvbuf` takes the buffering type as its second argument and the buffer pointer as the third, instead of the other way around, define `SETVBUF_REVERSED`.

AC_FUNC_STRCOLL [Macro]

If the `strcoll` function exists and works correctly, define `HAVE_STRCOLL`. This does a bit more than '`AC_CHECK_FUNCS(strcoll)`', because some systems have incorrect definitions of `strcoll` that should not be used.

AC_FUNC_STRTOD [Macro]

If the `strtod` function does not exist or doesn't work correctly, ask for an `AC_LIBOBJ` replacement of '`strtod`'. In this case, because '`strtod.c`' is likely to need '`pow`', set the output variable `POW_LIB` to the extra library needed.

AC_FUNC_STRERROR_R [Macro]

If `strerror_r` is available, define `HAVE_STRERROR_R`, and if it is declared, define `HAVE_DECL_STRERROR_R`. If it returns a `char *` message, define `STRERROR_R_CHAR_P`; otherwise it returns an `int` error number. The Thread-Safe Functions option of

POSIX requires `strerror_r` to return `int`, but many systems (including, for example, version 2.2.4 of the GNU C Library) return a `char *` value that is not necessarily equal to the buffer argument.

AC_FUNC_STRFTIME [Macro]

Check for `strftime` in the '`intl`' library, for SCO UNIX. Then, if `strftime` is available, define `HAVE_STRFTIME`.

AC_FUNC_STRNLEN [Macro]

If the `strnlen` function is not available, or is buggy (like the one from AIX 4.3), require an `AC_LIBOBJ` replacement for it.

AC_FUNC_UTIME_NULL [Macro]

If '`utime(`*`file`*`, NULL)`' sets *file*'s timestamp to the present, define `HAVE_UTIME_NULL`.

AC_FUNC_VPRINTF [Macro]

If `vprintf` is found, define `HAVE_VPRINTF`. Otherwise, if `_doprnt` is found, define `HAVE_DOPRNT`. (If `vprintf` is available, you may assume that `vfprintf` and `vsprintf` are also available.)

AC_REPLACE_FNMATCH [Macro]

If the `fnmatch` function does not conform to POSIX (see `AC_FUNC_FNMATCH`), ask for its `AC_LIBOBJ` replacement.

The files '`fnmatch.c`', '`fnmatch_loop.c`', and '`fnmatch_.h`' in the `AC_LIBOBJ` replacement directory are assumed to contain a copy of the source code of GNU `fnmatch`. If necessary, this source code is compiled as an `AC_LIBOBJ` replacement, and the '`fnmatch_.h`' file is linked to '`fnmatch.h`' so that it can be included in place of the system `<fnmatch.h>`.

5.5.3 Generic Function Checks

These macros are used to find functions not covered by the "particular" test macros. If the functions might be in libraries other than the default C library, first call `AC_CHECK_LIB` for those libraries. If you need to check the behavior of a function as well as find out whether it is present, you have to write your own test for it (see Chapter 6 [Writing Tests], page 71).

AC_CHECK_FUNC (*function*, [*action-if-found*], [*action-if-not-found*]) [Macro]

If C function *function* is available, run shell commands *action-if-found*, otherwise *action-if-not-found*. If you just want to define a symbol if the function is available, consider using `AC_CHECK_FUNCS` instead. This macro checks for functions with C linkage even when `AC_LANG(C++)` has been called, since C is more standardized than C++. (see Section 6.1 [Language Choice], page 71, for more information about selecting the language for checks.)

AC_CHECK_FUNCS (*function*. . . , [*action-if-found*], [*action-if-not-found*]) [Macro]

For each *function* in the whitespace-separated argument list, define `HAVE_`*`function`* (in all capitals) if it is available. If *action-if-found* is given, it is additional shell code to execute when one of the functions is found. You can give it a value of '`break`' to break out of the loop on the first match. If *action-if-not-found* is given, it is executed when one of the functions is not found.

Autoconf follows a philosophy that was formed over the years by those who have struggled for portability: isolate the portability issues in specific files, and then program as if you were in a POSIX environment. Some functions may be missing or unfixable, and your package must be ready to replace them.

AC_LIBOBJ (*function*) [Macro]

Specify that '*function*.c' must be included in the executables to replace a missing or broken implementation of *function*.

Technically, it adds '*function*.$ac_objext' to the output variable `LIBOBJS` if it is not already in, and calls `AC_LIBSOURCE` for '*function*.c'. You should not directly change `LIBOBJS`, since this is not traceable.

AC_LIBSOURCE (*file*) [Macro]

Specify that *file* might be needed to compile the project. If you need to know what files might be needed by a 'configure.ac', you should trace `AC_LIBSOURCE`. *file* must be a literal.

This macro is called automatically from `AC_LIBOBJ`, but you must call it explicitly if you pass a shell variable to `AC_LIBOBJ`. In that case, since shell variables cannot be traced statically, you must pass to `AC_LIBSOURCE` any possible files that the shell variable might cause `AC_LIBOBJ` to need. For example, if you want to pass a variable `$foo_or_bar` to `AC_LIBOBJ` that holds either `"foo"` or `"bar"`, you should do:

```
AC_LIBSOURCE(foo.c)
AC_LIBSOURCE(bar.c)
AC_LIBOBJ($foo_or_bar)
```

There is usually a way to avoid this, however, and you are encouraged to simply call `AC_LIBOBJ` with literal arguments.

Note that this macro replaces the obsolete `AC_LIBOBJ_DECL`, with slightly different semantics: the old macro took the function name, e.g., `foo`, as its argument rather than the file name.

AC_LIBSOURCES (*files*) [Macro]

Like `AC_LIBSOURCE`, but accepts one or more *files* in a comma-separated M4 list. Thus, the above example might be rewritten:

```
AC_LIBSOURCES([foo.c, bar.c])
AC_LIBOBJ($foo_or_bar)
```

AC_CONFIG_LIBOBJ_DIR (*directory*) [Macro]

Specify that `AC_LIBOBJ` replacement files are to be found in *directory*, a relative path starting from the top level of the source tree. The replacement directory defaults to '.', the top level directory, and the most typical value is 'lib', corresponding to 'AC_CONFIG_LIBOBJ_DIR(lib)'.

`configure` might need to know the replacement directory for the following reasons: (i) some checks use the replacement files, (ii) some macros bypass broken system headers by installing links to the replacement headers, etc.

It is common to merely check for the existence of a function, and ask for its `AC_LIBOBJ` replacement if missing. The following macro is a convenient shorthand.

AC_REPLACE_FUNCS (*function*...) [Macro]

Like `AC_CHECK_FUNCS`, but uses '`AC_LIBOBJ(function)`' as *action-if-not-found*. You can declare your replacement function by enclosing the prototype in '`#if !HAVE_function`'. If the system has the function, it probably declares it in a header file you should be including, so you shouldn't redeclare it lest your declaration conflict.

5.6 Header Files

The following macros check for the presence of certain C header files. If there is no macro specifically defined to check for a header file you need, and you don't need to check for any special properties of it, then you can use one of the general header-file check macros.

5.6.1 Portability of Headers

This section tries to collect knowledge about common headers, and the problems they cause. By definition, this list will always require additions. Please help us keeping it as complete as possible.

'`inttypes.h`' vs. '`stdint.h`'
: Paul Eggert notes that: ISO C 1999 says that '`inttypes.h`' includes '`stdint.h`', so there's no need to include '`stdint.h`' separately in a standard environment. Many implementations have '`inttypes.h`' but not '`stdint.h`' (e.g., Solaris 7), but I don't know of any implementation that has '`stdint.h`' but not '`inttypes.h`'. Nor do I know of any free software that includes '`stdint.h`'; '`stdint.h`' seems to be a creation of the committee.

'`linux/irda.h`'
: It requires '`linux/types.h`' and '`sys/socket.h`'.

'`linux/random.h`'
: It requires '`linux/types.h`'.

'`net/if.h`'
: On Darwin, this file requires that '`sys/socket.h`' be included beforehand. One should run:

```
AC_CHECK_HEADERS([sys/socket.h])
AC_CHECK_HEADERS([net/if.h], [], [],
[#include <stdio.h>
#if STDC_HEADERS
# include <stdlib.h>
# include <stddef.h>
#else
# if HAVE_STDLIB_H
#  include <stdlib.h>
# endif
#endif
#if HAVE_SYS_SOCKET_H
# include <sys/socket.h>
#endif
])
```

'netinet/if_ether.h'
: On Darwin, this file requires that 'stdio.h' and 'sys/socket.h' be included beforehand. One should run:

```
AC_CHECK_HEADERS([sys/socket.h])
AC_CHECK_HEADERS([netinet/if_ether.h], [], [],
[#include <stdio.h>
#if STDC_HEADERS
# include <stdlib.h>
# include <stddef.h>
#else
# if HAVE_STDLIB_H
#  include <stdlib.h>
# endif
#endif
#if HAVE_SYS_SOCKET_H
# include <sys/socket.h>
#endif
])
```

'stdint.h'
: See above, item 'inttypes.h' vs. 'stdint.h'.

'stdlib.h'
: On many systems (e.g., Darwin), 'stdio.h' is a prerequisite.

'sys/mount.h'
: On FreeBSD 4.8 on ia32 and using gcc version 2.95.4, 'sys/params.h' is a prerequisite.

'sys/socket.h'
: On Darwin, 'stdlib.h' is a prerequisite.

'sys/ucred.h'
: On HP Tru64 5.1, 'sys/types.h' is a prerequisite.

'X11/extensions/scrnsaver.h'
: Using XFree86, this header requires 'X11/Xlib.h', which is probably so required that you might not even consider looking for it.

```
AC_CHECK_HEADERS([X11/extensions/scrnsaver.h], [], [],
[[#include <X11/Xlib.h>
]])
```

5.6.2 Particular Header Checks

These macros check for particular system header files—whether they exist, and in some cases whether they declare certain symbols.

AC_HEADER_DIRENT [Macro]

Check for the following header files. For the first one that is found and defines 'DIR', define the listed C preprocessor macro:

'dirent.h' HAVE_DIRENT_H

'sys/ndir.h'	HAVE_SYS_NDIR_H
'sys/dir.h'	HAVE_SYS_DIR_H
'ndir.h'	HAVE_NDIR_H

The directory-library declarations in your source code should look something like the following:

```
#if HAVE_DIRENT_H
# include <dirent.h>
# define NAMLEN(dirent) strlen((dirent)->d_name)
#else
# define dirent direct
# define NAMLEN(dirent) (dirent)->d_namlen
# if HAVE_SYS_NDIR_H
#  include <sys/ndir.h>
# endif
# if HAVE_SYS_DIR_H
#  include <sys/dir.h>
# endif
# if HAVE_NDIR_H
#  include <ndir.h>
# endif
#endif
```

Using the above declarations, the program would declare variables to be of type `struct dirent`, not `struct direct`, and would access the length of a directory entry name by passing a pointer to a `struct dirent` to the `NAMLEN` macro.

This macro also checks for the SCO Xenix 'dir' and 'x' libraries.

AC_HEADER_MAJOR [Macro]

If 'sys/types.h' does not define `major`, `minor`, and `makedev`, but 'sys/mkdev.h' does, define `MAJOR_IN_MKDEV`; otherwise, if 'sys/sysmacros.h' does, define `MAJOR_IN_SYSMACROS`.

AC_HEADER_STAT [Macro]

If the macros `S_ISDIR`, `S_ISREG`, etc. defined in 'sys/stat.h' do not work properly (returning false positives), define `STAT_MACROS_BROKEN`. This is the case on Tektronix UTekV, Amdahl UTS and Motorola System V/88.

AC_HEADER_STDBOOL [Macro]

If 'stdbool.h' exists and is conformant to C99, define `HAVE_STDBOOL_H` to 1; if the type `_Bool` is defined, define `HAVE__BOOL` to 1. To fulfill the C99 requirements, your 'system.h' should contain the following code:

```
#if HAVE_STDBOOL_H
# include <stdbool.h>
#else
# if ! HAVE__BOOL
#  ifdef __cplusplus
typedef bool _Bool;
#  else
```

```
typedef unsigned char _Bool;
#  endif
# endif
# define bool _Bool
# define false 0
# define true 1
# define __bool_true_false_are_defined 1
#endif
```

AC_HEADER_STDC [Macro]

Define `STDC_HEADERS` if the system has ANSI C header files. Specifically, this macro checks for 'stdlib.h', 'stdarg.h', 'string.h', and 'float.h'; if the system has those, it probably has the rest of the ANSI C header files. This macro also checks whether 'string.h' declares `memchr` (and thus presumably the other `mem` functions), whether 'stdlib.h' declare `free` (and thus presumably `malloc` and other related functions), and whether the 'ctype.h' macros work on characters with the high bit set, as ANSI C requires.

Use `STDC_HEADERS` instead of `__STDC__` to determine whether the system has ANSI-compliant header files (and probably C library functions) because many systems that have GCC do not have ANSI C header files.

On systems without ANSI C headers, there is so much variation that it is probably easier to declare the functions you use than to figure out exactly what the system header files declare. Some systems contain a mix of functions from ANSI and BSD; some are mostly ANSI but lack 'memmove'; some define the BSD functions as macros in 'string.h' or 'strings.h'; some have only the BSD functions but 'string.h'; some declare the memory functions in 'memory.h', some in 'string.h'; etc. It is probably sufficient to check for one string function and one memory function; if the library has the ANSI versions of those then it probably has most of the others. If you put the following in 'configure.ac':

```
AC_HEADER_STDC
AC_CHECK_FUNCS(strchr memcpy)
```

then, in your code, you can use declarations like this:

```
#if STDC_HEADERS
# include <string.h>
#else
# if !HAVE_STRCHR
#  define strchr index
#  define strrchr rindex
# endif
char *strchr (), *strrchr ();
# if !HAVE_MEMCPY
#  define memcpy(d, s, n) bcopy ((s), (d), (n))
#  define memmove(d, s, n) bcopy ((s), (d), (n))
# endif
#endif
```

If you use a function like `memchr`, `memset`, `strtok`, or `strspn`, which have no BSD equivalent, then macros won't suffice; you must provide an implementation of each function. An easy way to incorporate your implementations only when needed (since the ones in system C libraries may be hand optimized) is to, taking `memchr` for example, put it in '`memchr.c`' and use '`AC_REPLACE_FUNCS(memchr)`'.

AC_HEADER_SYS_WAIT [Macro]

If '`sys/wait.h`' exists and is compatible with POSIX, define `HAVE_SYS_WAIT_H`. Incompatibility can occur if '`sys/wait.h`' does not exist, or if it uses the old BSD `union wait` instead of `int` to store a status value. If '`sys/wait.h`' is not POSIX compatible, then instead of including it, define the POSIX macros with their usual interpretations. Here is an example:

```
#include <sys/types.h>
#if HAVE_SYS_WAIT_H
# include <sys/wait.h>
#endif
#ifndef WEXITSTATUS
# define WEXITSTATUS(stat_val) ((unsigned)(stat_val) >> 8)
#endif
#ifndef WIFEXITED
# define WIFEXITED(stat_val) (((stat_val) & 255) == 0)
#endif
```

`_POSIX_VERSION` is defined when '`unistd.h`' is included on POSIX systems. If there is no '`unistd.h`', it is definitely not a POSIX system. However, some non-POSIX systems do have '`unistd.h`'.

The way to check if the system supports POSIX is:

```
#if HAVE_UNISTD_H
# include <sys/types.h>
# include <unistd.h>
#endif

#ifdef _POSIX_VERSION
/* Code for POSIX systems.  */
#endif
```

AC_HEADER_TIME [Macro]

If a program may include both '`time.h`' and '`sys/time.h`', define `TIME_WITH_SYS_TIME`. On some older systems, '`sys/time.h`' includes '`time.h`', but '`time.h`' is not protected against multiple inclusion, so programs should not explicitly include both files. This macro is useful in programs that use, for example, `struct timeval` as well as `struct tm`. It is best used in conjunction with `HAVE_SYS_TIME_H`, which can be checked for using `AC_CHECK_HEADERS(sys/time.h)`.

```
#if TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
#  include <sys/time.h>
# else
#  include <time.h>
# endif
#endif
```

AC_HEADER_TIOCGWINSZ [Macro]

If the use of `TIOCGWINSZ` requires '`<sys/ioctl.h>`', then define `GWINSZ_IN_SYS_IOCTL`. Otherwise `TIOCGWINSZ` can be found in '`<termios.h>`'.

Use:

```
#if HAVE_TERMIOS_H
# include <termios.h>
#endif

#if GWINSZ_IN_SYS_IOCTL
# include <sys/ioctl.h>
#endif
```

5.6.3 Generic Header Checks

These macros are used to find system header files not covered by the "particular" test macros. If you need to check the contents of a header as well as find out whether it is present, you have to write your own test for it (see Chapter 6 [Writing Tests], page 71).

AC_CHECK_HEADER (*header-file*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

If the system header file *header-file* is compilable, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*. If you just want to define a symbol if the header file is available, consider using `AC_CHECK_HEADERS` instead.

For compatibility issues with older versions of Autoconf, please read below.

AC_CHECK_HEADERS (*header-file*. . . , [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

For each given system header file *header-file* in the whitespace-separated argument list that exists, define `HAVE_header-file` (in all capitals). If *action-if-found* is given, it is additional shell code to execute when one of the header files is found. You can give it a value of '`break`' to break out of the loop on the first match. If *action-if-not-found* is given, it is executed when one of the header files is not found.

For compatibility issues with older versions of Autoconf, please read below.

Previous versions of Autoconf merely checked whether the header was accepted by the preprocessor. This was changed because the old test was inappropriate for typical uses. Headers are typically used to compile, not merely to preprocess, and the old behavior sometimes accepted headers that clashed at compile-time. If you need to check whether a header is preprocessable, you can use `AC_PREPROC_IFELSE` (see Section 6.3 [Running the Preprocessor], page 76).

This scheme, which improves the robustness of the test, also requires that you make sure that headers that must be included before the *header-file* be part of the *includes*, (see Section 5.1.2 [Default Includes], page 33). If looking for 'bar.h', which requires that 'foo.h' be included before if it exists, we suggest the following scheme:

```
AC_CHECK_HEADERS([foo.h])
AC_CHECK_HEADERS([bar.h], [], [],
[#if HAVE_FOO_H
# include <foo.h>
# endif
])
```

5.7 Declarations

The following macros check for the declaration of variables and functions. If there is no macro specifically defined to check for a symbol you need, then you can use the general macros (see Section 5.7.2 [Generic Declarations], page 54) or, for more complex tests, you may use `AC_COMPILE_IFELSE` (see Section 6.4 [Running the Compiler], page 77).

5.7.1 Particular Declaration Checks

There are no specific macros for declarations.

5.7.2 Generic Declaration Checks

These macros are used to find declarations not covered by the "particular" test macros.

AC_CHECK_DECL (*symbol*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

If *symbol* (a function or a variable) is not declared in *includes* and a declaration is needed, run the shell commands *action-if-not-found*, otherwise *action-if-found*. If no *includes* are specified, the default includes are used (see Section 5.1.2 [Default Includes], page 33).

This macro actually tests whether it is valid to use *symbol* as an r-value, not if it is really declared, because it is much safer to avoid introducing extra declarations when they are not needed.

AC_CHECK_DECLS (*symbols*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

For each of the *symbols* (*comma*-separated list), define `HAVE_DECL_symbol` (in all capitals) to '1' if *symbol* is declared, otherwise to '0'. If *action-if-not-found* is given, it is additional shell code to execute when one of the function declarations is needed, otherwise *action-if-found* is executed.

This macro uses an m4 list as first argument:

```
AC_CHECK_DECLS(strdup)
AC_CHECK_DECLS([strlen])
AC_CHECK_DECLS([malloc, realloc, calloc, free])
```

Unlike the other '`AC_CHECK_*S`' macros, when a *symbol* is not declared, `HAVE_DECL_`*`symbol`* is defined to '`0`' instead of leaving `HAVE_DECL_`*`symbol`* undeclared. When you are *sure* that the check was performed, use `HAVE_DECL_`*`symbol`* just like any other result of Autoconf:

```
#if !HAVE_DECL_SYMBOL
extern char *symbol;
#endif
```

If the test may have not been performed, however, because it is safer *not* to declare a symbol than to use a declaration that conflicts with the system's one, you should use:

```
#if defined HAVE_DECL_MALLOC && !HAVE_DECL_MALLOC
void *malloc (size_t *s);
#endif
```

You fall into the second category only in extreme situations: either your files may be used without being configured, or they are used during the configuration. In most cases the traditional approach is enough.

5.8 Structures

The following macros check for the presence of certain members in C structures. If there is no macro specifically defined to check for a member you need, then you can use the general structure-member macros (see Section 5.8.2 [Generic Structures], page 56) or, for more complex tests, you may use `AC_COMPILE_IFELSE` (see Section 6.4 [Running the Compiler], page 77).

5.8.1 Particular Structure Checks

The following macros check for certain structures or structure members.

AC_STRUCT_ST_BLKSIZE [Macro]

If `struct stat` contains an `st_blksize` member, define `HAVE_STRUCT_STAT_ST_BLKSIZE`. The former name, `HAVE_ST_BLKSIZE` is to be avoided, as its support will cease in the future. This macro is obsoleted, and should be replaced by

```
AC_CHECK_MEMBERS([struct stat.st_blksize])
```

AC_STRUCT_ST_BLOCKS [Macro]

If `struct stat` contains an `st_blocks` member, define `HAVE_STRUCT_STAT_ST_BLOCKS`. Otherwise, require an `AC_LIBOBJ` replacement of '`fileblocks`'. The former name, `HAVE_ST_BLOCKS` is to be avoided, as its support will cease in the future.

AC_STRUCT_ST_RDEV [Macro]

If `struct stat` contains an `st_rdev` member, define `HAVE_STRUCT_STAT_ST_RDEV`. The former name for this macro, `HAVE_ST_RDEV`, is to be avoided as it will cease to be supported in the future. Actually, even the new macro is obsolete and should be replaced by:

```
AC_CHECK_MEMBERS([struct stat.st_rdev])
```

AC_STRUCT_TM [Macro]

If 'time.h' does not define `struct tm`, define `TM_IN_SYS_TIME`, which means that including 'sys/time.h' had better define `struct tm`.

AC_STRUCT_TIMEZONE [Macro]

Figure out how to get the current timezone. If `struct tm` has a `tm_zone` member, define `HAVE_STRUCT_TM_TM_ZONE` (and the obsoleted `HAVE_TM_ZONE`). Otherwise, if the external array `tzname` is found, define `HAVE_TZNAME`.

5.8.2 Generic Structure Checks

These macros are used to find structure members not covered by the "particular" test macros.

AC_CHECK_MEMBER (*aggregate.member*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

Check whether *member* is a member of the aggregate *aggregate*. If no *includes* are specified, the default includes are used (see Section 5.1.2 [Default Includes], page 33).

```
AC_CHECK_MEMBER(struct passwd.pw_gecos,,
                [AC_MSG_ERROR([We need 'passwd.pw_gecos'!])],
                [#include <pwd.h>])
```

You can use this macro for sub-members:

```
AC_CHECK_MEMBER(struct top.middle.bot)
```

AC_CHECK_MEMBERS (*members*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

Check for the existence of each '*aggregate.member*' of *members* using the previous macro. When *member* belongs to *aggregate*, define `HAVE_`*aggregate*`_`*member* (in all capitals, with spaces and dots replaced by underscores). If *action-if-found* is given, it is executed for each of the found members. If *action-if-not-found* is given, it is executed for each of the members that could not be found.

This macro uses m4 lists:

```
AC_CHECK_MEMBERS([struct stat.st_rdev, struct stat.st_blksize])
```

5.9 Types

The following macros check for C types, either builtin or typedefs. If there is no macro specifically defined to check for a type you need, and you don't need to check for any special properties of it, then you can use a general type-check macro.

5.9.1 Particular Type Checks

These macros check for particular C types in 'sys/types.h', 'stdlib.h' and others, if they exist.

AC_TYPE_GETGROUPS [Macro]

Define `GETGROUPS_T` to be whichever of `gid_t` or `int` is the base type of the array argument to `getgroups`.

AC_TYPE_MBSTATE_T [Macro]

Define `HAVE_MBSTATE_T` if `<wchar.h>` declares the `mbstate_t` type. Also, define `mbstate_t` to be a type if `<wchar.h>` does not declare it.

AC_TYPE_MODE_T [Macro]

Equivalent to '`AC_CHECK_TYPE(mode_t, int)`'.

AC_TYPE_OFF_T [Macro]

Equivalent to '`AC_CHECK_TYPE(off_t, long)`'.

AC_TYPE_PID_T [Macro]

Equivalent to '`AC_CHECK_TYPE(pid_t, int)`'.

AC_TYPE_SIGNAL [Macro]

If '`signal.h`' declares `signal` as returning a pointer to a function returning `void`, define `RETSIGTYPE` to be `void`; otherwise, define it to be `int`.

Define signal handlers as returning type `RETSIGTYPE`:

```
RETSIGTYPE
hup_handler ()
{
...
}
```

AC_TYPE_SIZE_T [Macro]

Equivalent to '`AC_CHECK_TYPE(size_t, unsigned)`'.

AC_TYPE_UID_T [Macro]

If `uid_t` is not defined, define `uid_t` to be `int` and `gid_t` to be `int`.

5.9.2 Generic Type Checks

These macros are used to check for types not covered by the "particular" test macros.

AC_CHECK_TYPE (*type*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

Check whether *type* is defined. It may be a compiler builtin type or defined by the *includes* (see Section 5.1.2 [Default Includes], page 33).

AC_CHECK_TYPES (*types*, [*action-if-found*], [*action-if-not-found*], [*includes* = '`default-includes`']) [Macro]

For each *type* of the *types* that is defined, define `HAVE_type` (in all capitals). If no *includes* are specified, the default includes are used (see Section 5.1.2 [Default Includes], page 33). If *action-if-found* is given, it is additional shell code to execute when one of the types is found. If *action-if-not-found* is given, it is executed when one of the types is not found.

This macro uses m4 lists:

```
AC_CHECK_TYPES(ptrdiff_t)
AC_CHECK_TYPES([unsigned long long, uintmax_t])
```

Autoconf, up to 2.13, used to provide to another version of `AC_CHECK_TYPE`, broken by design. In order to keep backward compatibility, a simple heuristics, quite safe but not totally, is implemented. In case of doubt, read the documentation of the former `AC_CHECK_TYPE`, see Section 15.4 [Obsolete Macros], page 167.

5.10 Compilers and Preprocessors

All the tests for compilers (`AC_PROG_CC`, `AC_PROG_CXX`, `AC_PROG_F77`) define the output variable `EXEEXT` based on the output of the compiler, typically to the empty string if Unix and '`.exe`' if Win32 or OS/2.

They also define the output variable `OBJEXT` based on the output of the compiler, after '`.c`' files have been excluded, typically to '`o`' if Unix, '`obj`' if Win32.

If the compiler being used does not produce executables, the tests fail. If the executables can't be run, and cross-compilation is not enabled, they fail too. See Chapter 11 [Manual Configuration], page 149, for more on support for cross compiling.

5.10.1 Specific Compiler Characteristics

Some compilers exhibit different behaviors.

Static/Dynamic Expressions

Autoconf relies on a trick to extract one bit of information from the C compiler: using negative array sizes. For instance the following excerpt of a C source demonstrates how to test whether '`int`'s are 4 bytes long:

```
int
main (void)
{
  static int test_array [sizeof (int) == 4 ? 1 : -1];
  test_array [0] = 0
  return 0;
}
```

To our knowledge, there is a single compiler that does not support this trick: the HP C compilers (the real one, not only the "bundled") on HP-UX 11.00:

```
$ cc -c -Ae +O2 +Onolimit conftest.c
cc: "conftest.c": error 1879: Variable-length arrays cannot \
    have static storage.
```

Autoconf works around this problem by casting `sizeof (int)` to `long` before comparing it.

5.10.2 Generic Compiler Characteristics

AC_CHECK_SIZEOF (*type*, [*unused*], [*includes* = '`default-includes`']) [Macro]

Define `SIZEOF_type` (see Section 5.1.1 [Standard Symbols], page 33) to be the size in bytes of *type*. If '`type`' is unknown, it gets a size of 0. If no *includes* are specified, the default includes are used (see Section 5.1.2 [Default Includes], page 33). If you provide *include*, be sure to include '`stdio.h`' which is required for this macro to run.

This macro now works even when cross-compiling. The *unused* argument was used when cross-compiling.

For example, the call

```
AC_CHECK_SIZEOF(int *)
```

defines `SIZEOF_INT_P` to be 8 on DEC Alpha AXP systems.

AC_LANG_WERROR [Macro]

Normally Autoconf ignores warnings generated by the compiler, linker, and preprocessor. If this macro is used, warnings will be treated as fatal errors instead for the current language. This macro is useful when the results of configuration will be used where warnings are unacceptable; for instance, if parts of a program are built with the GCC '`-Werror`' option. If the whole program will be built using '`-Werror`' it is often simpler to put '`-Werror`' in the compiler flags (`CFLAGS` etc.).

5.10.3 C Compiler Characteristics

The following macros provide ways to find and exercise a C Compiler. There are a few constructs that ought to be avoided, but do not deserve being checked for, since they can easily be worked around.

Don't use lines containing solitary backslashes

They tickle a bug in the HP-UX C compiler (checked on HP-UX 10.20, 11.00, and 11i). Running the compiler on the following source,

```
#ifdef __STDC__
/\
* A comment with backslash-newlines in it. %{ %} *\
\
/
char str[] = "\\
" A string with backslash-newlines in it %{ %} \\
"";
char apostrophe = '\\
\
'\
';
#endif
```

yields

```
error cpp: "foo.c", line 13: error 4048: Non-terminating comment at end of f
error cpp: "foo.c", line 13: error 4033: Missing #endif at end of file.
```

Removing the lines with solitary backslashes solves the problem.

Don't compile several files at once if output matters to you

Some compilers, such as the HP's, reports the name of the file it is compiling *when* they are several. For instance:

```
$ cc a.c b.c
a.c:
b.c:
```

This can cause problems if you observe the output of the compiler to detect failures. Invoking '`cc -c a.c -o a.o; cc -c b.c -o b.o; cc a.o b.o -o c`' solves the issue.

Don't rely on correct #line support

On Solaris 8, c89 (Sun WorkShop 6 update 2 C 5.3 Patch 111679-08 2002/05/09)) rejects #line directives whose line numbers are greater than 32767. In addition, nothing in POSIX makes this invalid. That is the reason why Autoconf stopped issuing #line directives.

AC_PROG_CC ([*compiler-search-list*]) [Macro]

Determine a C compiler to use. If CC is not already set in the environment, check for gcc and cc, then for other C compilers. Set output variable CC to the name of the compiler found.

This macro may, however, be invoked with an optional first argument which, if specified, must be a space separated list of C compilers to search for. This just gives the user an opportunity to specify an alternative search list for the C compiler. For example, if you didn't like the default order, then you could invoke AC_PROG_CC like this:

```
AC_PROG_CC(cl egcs gcc cc)
```

If the C compiler is not in ANSI C mode by default, try to add an option to output variable CC to make it so. This macro tries various options that select ANSI C on some system or another. It considers the compiler to be in ANSI C mode if it handles function prototypes correctly.

After calling this macro you can check whether the C compiler has been set to accept ANSI C; if not, the shell variable ac_cv_prog_cc_stdc is set to 'no'. If you wrote your source code in ANSI C, you can make an un-ANSIfied copy of it by using the program ansi2knr, which comes with Automake. See also under AC_C_PROTOTYPES below.

If using the GNU C compiler, set shell variable GCC to 'yes'. If output variable CFLAGS was not already set, set it to '-g -O2' for the GNU C compiler ('-O2' on systems where GCC does not accept '-g'), or '-g' for other compilers.

AC_PROG_CC_C_O [Macro]

If the C compiler does not accept the '-c' and '-o' options simultaneously, define NO_MINUS_C_MINUS_O. This macro actually tests both the compiler found by AC_PROG_CC, and, if different, the first cc in the path. The test fails if one fails. This macro was created for GNU Make to choose the default C compilation rule.

AC_PROG_CPP [Macro]

Set output variable CPP to a command that runs the C preprocessor. If '$CC -E' doesn't work, '/lib/cpp' is used. It is only portable to run CPP on files with a '.c' extension.

Some preprocessors don't indicate missing include files by the error status. For such preprocessors an internal variable is set that causes other macros to check the standard error from the preprocessor and consider the test failed if any warnings have been reported. For most preprocessors, though, warnings do not cause include-file tests to fail unless AC_PROG_CPP_WERROR is also specified.

AC_PROG_CPP_WERROR [Macro]

This acts like AC_PROG_CPP, except it treats warnings from the preprocessor as errors even if the preprocessor exit status indicates success. This is useful for avoiding headers that generate mandatory warnings, such as deprecation notices.

The following macros check for C compiler or machine architecture features. To check for characteristics not listed here, use `AC_COMPILE_IFELSE` (see Section 6.4 [Running the Compiler], page 77) or `AC_RUN_IFELSE` (see Section 6.6 [Run Time], page 78).

AC_C_BACKSLASH_A [Macro]

Define '`HAVE_C_BACKSLASH_A`' to 1 if the C compiler understands '`\a`'.

AC_C_BIGENDIAN ([*action-if-true*], [*action-if-false*], [*action-if-unknown*]) [Macro]

If words are stored with the most significant byte first (like Motorola and SPARC CPUs), execute *action-if-true*. If words are stored with the least significant byte first (like Intel and VAX CPUs), execute *action-if-false*.

This macro runs a test-case if endianness cannot be determined from the system header files. When cross-compiling, the test-case is not run but grep'ed for some magic values. *action-if-unknown* is executed if the latter case fails to determine the byte sex of the host system.

The default for *action-if-true* is to define '`WORDS_BIGENDIAN`'. The default for *action-if-false* is to do nothing. And finally, the default for *action-if-unknown* is to abort configure and tell the installer which variable he should preset to bypass this test.

AC_C_CONST [Macro]

If the C compiler does not fully support the ANSI C qualifier `const`, define `const` to be empty. Some C compilers that do not define `__STDC__` do support `const`; some compilers that define `__STDC__` do not completely support `const`. Programs can simply use `const` as if every C compiler supported it; for those that don't, the '`Makefile`' or configuration header file will define it as empty.

Occasionally installers use a C++ compiler to compile C code, typically because they lack a C compiler. This causes problems with `const`, because C and C++ treat `const` differently. For example:

```
const int foo;
```

is valid in C but not in C++. These differences unfortunately cannot be papered over by defining `const` to be empty.

If `autoconf` detects this situation, it leaves `const` alone, as this generally yields better results in practice. However, using a C++ compiler to compile C code is not recommended or supported, and installers who run into trouble in this area should get a C compiler like GCC to compile their C code.

AC_C_RESTRICT [Macro]

If the C compiler recognizes the `restrict` keyword, don't do anything. If it recognizes only a variant spelling (`__restrict`, `__restrict__`, or `_Restrict`), then define `restrict` to that. Otherwise, define `restrict` to be empty. Thus, programs may simply use `restrict` as if every C compiler supported it; for those that do not, the '`Makefile`' or configuration header defines it away.

Although support in C++ for the `restrict` keyword is not required, several C++ compilers do accept the keyword. This macro works for them, too.

AC_C_VOLATILE [Macro]

If the C compiler does not understand the keyword `volatile`, define `volatile` to be empty. Programs can simply use `volatile` as if every C compiler supported it; for those that do not, the '`Makefile`' or configuration header will define it as empty.

If the correctness of your program depends on the semantics of `volatile`, simply defining it to be empty does, in a sense, break your code. However, given that the compiler does not support `volatile`, you are at its mercy anyway. At least your program will compile, when it wouldn't before.

In general, the `volatile` keyword is a feature of ANSI C, so you might expect that `volatile` is available only when `__STDC__` is defined. However, Ultrix 4.3's native compiler does support volatile, but does not define `__STDC__`.

AC_C_INLINE [Macro]

If the C compiler supports the keyword `inline`, do nothing. Otherwise define `inline` to `__inline__` or `__inline` if it accepts one of those, otherwise define `inline` to be empty.

AC_C_CHAR_UNSIGNED [Macro]

If the C type `char` is unsigned, define `__CHAR_UNSIGNED__`, unless the C compiler predefines it.

AC_C_LONG_DOUBLE [Macro]

If the C compiler supports a working `long double` type with more range or precision than the `double` type, define `HAVE_LONG_DOUBLE`.

AC_C_STRINGIZE [Macro]

If the C preprocessor supports the stringizing operator, define `HAVE_STRINGIZE`. The stringizing operator is '`#`' and is found in macros such as this:

```
#define x(y) #y
```

AC_C_PROTOTYPES [Macro]

If function prototypes are understood by the compiler (as determined by `AC_PROG_CC`), define `PROTOTYPES` and `__PROTOTYPES`. In the case the compiler does not handle prototypes, you should use `ansi2knr`, which comes with the Automake distribution, to unprotoize function definitions. For function prototypes, you should first define `PARAMS`:

```
#ifndef PARAMS
# if PROTOTYPES
#  define PARAMS(protos) protos
# else /* no PROTOTYPES */
#  define PARAMS(protos) ()
# endif /* no PROTOTYPES */
#endif
```

then use it this way:

```
size_t my_strlen PARAMS ((const char *));
```

This macro also defines `__PROTOTYPES`; this is for the benefit of header files that cannot use macros that infringe on user name space.

AC_PROG_GCC_TRADITIONAL [Macro]

Add '-traditional' to output variable CC if using the GNU C compiler and ioctl does not work properly without '-traditional'. That usually happens when the fixed header files have not been installed on an old system. Since recent versions of the GNU C compiler fix the header files automatically when installed, this is becoming a less prevalent problem.

5.10.4 C++ Compiler Characteristics

AC_PROG_CXX ([*compiler-search-list*]) [Macro]

Determine a C++ compiler to use. Check if the environment variable CXX or CCC (in that order) is set; if so, then set output variable CXX to its value.

Otherwise, if the macro is invoked without an argument, then search for a C++ compiler under the likely names (first g++ and c++ then other names). If none of those checks succeed, then as a last resort set CXX to g++.

This macro may, however, be invoked with an optional first argument which, if specified, must be a space separated list of C++ compilers to search for. This just gives the user an opportunity to specify an alternative search list for the C++ compiler. For example, if you didn't like the default order, then you could invoke AC_PROG_CXX like this:

```
AC_PROG_CXX(cl KCC CC cxx cc++ xlC aCC c++ g++ egcs gcc)
```

If using the GNU C++ compiler, set shell variable GXX to 'yes'. If output variable CXXFLAGS was not already set, set it to '-g -O2' for the GNU C++ compiler ('-O2' on systems where G++ does not accept '-g'), or '-g' for other compilers.

AC_PROG_CXXCPP [Macro]

Set output variable CXXCPP to a command that runs the C++ preprocessor. If '$CXX -E' doesn't work, '/lib/cpp' is used. It is only portable to run CXXCPP on files with a '.c', '.C', or '.cc' extension.

Some preprocessors don't indicate missing include files by the error status. For such preprocessors an internal variable is set that causes other macros to check the standard error from the preprocessor and consider the test failed if any warnings have been reported. However, it is not known whether such broken preprocessors exist for C++.

5.10.5 Fortran Compiler Characteristics

The Autoconf Fortran support is divided into two categories: legacy Fortran 77 macros (F77), and modern Fortran macros (FC). The former are intended for traditional Fortran 77 code, and have output variables like F77, FFLAGS, and FLIBS. The latter are for newer programs that can (or must) compile under the newer Fortran standards, and have output variables like FC, FCFLAGS, and FCLIBS.

Except for two new macros AC_FC_SRCEXT and AC_FC_FREEFORM (see below), the FC and F77 macros behave almost identically, and so they are documented together in this section.

AC_PROG_F77 ([*compiler-search-list*]) [Macro]

Determine a Fortran 77 compiler to use. If F77 is not already set in the environment, then check for g77 and f77, and then some other names. Set the output variable F77 to the name of the compiler found.

This macro may, however, be invoked with an optional first argument which, if specified, must be a space separated list of Fortran 77 compilers to search for. This just gives the user an opportunity to specify an alternative search list for the Fortran 77 compiler. For example, if you didn't like the default order, then you could invoke `AC_PROG_F77` like this:

```
AC_PROG_F77(fl32 f77 fort77 xlf g77 f90 xlf90)
```

If using `g77` (the GNU Fortran 77 compiler), then `AC_PROG_F77` will set the shell variable `G77` to '`yes`'. If the output variable `FFLAGS` was not already set in the environment, then set it to '`-g -02`' for `g77` (or '`-O2`' where `g77` does not accept '`-g`'). Otherwise, set `FFLAGS` to '`-g`' for all other Fortran 77 compilers.

AC_PROG_FC ([*compiler-search-list*], [*dialect*]) [Macro]

Determine a Fortran compiler to use. If `FC` is not already set in the environment, then `dialect` is a hint to indicate what Fortran dialect to search for; the default is to search for the newest available dialect. Set the output variable `FC` to the name of the compiler found.

By default, newer dialects are preferred over older dialects, but if `dialect` is specified then older dialects are preferred starting with the specified dialect. `dialect` can currently be one of Fortran 77, Fortran 90, or Fortran 95. However, this is only a hint of which compiler *name* to prefer (e.g. `f90` or `f95`), and no attempt is made to guarantee that a particular language standard is actually supported. Thus, it is preferable that you avoid the `dialect` option, and use AC_PROG_FC only for code compatible with the latest Fortran standard.

This macro may, alternatively, be invoked with an optional first argument which, if specified, must be a space separated list of Fortran compilers to search for, just as in `AC_PROG_F77`.

If the output variable `FCFLAGS` was not already set in the environment, then set it to '`-g -02`' for GNU `g77` (or '`-O2`' where `g77` does not accept '`-g`'). Otherwise, set `FCFLAGS` to '`-g`' for all other Fortran compilers.

AC_PROG_F77_C_O [Macro]
AC_PROG_FC_C_O [Macro]

Test if the Fortran compiler accepts the options '`-c`' and '`-o`' simultaneously, and define `F77_NO_MINUS_C_MINUS_O` or `FC_NO_MINUS_C_MINUS_O`, respectively, if it does not.

The following macros check for Fortran compiler characteristics. To check for characteristics not listed here, use `AC_COMPILE_IFELSE` (see Section 6.4 [Running the Compiler], page 77) or `AC_RUN_IFELSE` (see Section 6.6 [Run Time], page 78), making sure to first set the current language to Fortran 77 or Fortran via `AC_LANG(Fortran 77)` or `AC_LANG(Fortran)` (see Section 6.1 [Language Choice], page 71).

AC_F77_LIBRARY_LDFLAGS [Macro]
AC_FC_LIBRARY_LDFLAGS [Macro]

Determine the linker flags (e.g., '`-L`' and '`-l`') for the *Fortran intrinsic and run-time libraries* that are required to successfully link a Fortran program or shared library.

The output variable `FLIBS` or `FCLIBS` is set to these flags (which should be include after `LIBS` when linking).

This macro is intended to be used in those situations when it is necessary to mix, e.g., C++ and Fortran source code in a single program or shared library (see section "Mixing Fortran 77 With C and C++" in GNU *Automake*).

For example, if object files from a C++ and Fortran compiler must be linked together, then the C++ compiler/linker must be used for linking (since special C++-ish things need to happen at link time like calling global constructors, instantiating templates, enabling exception support, etc.).

However, the Fortran intrinsic and run-time libraries must be linked in as well, but the C++ compiler/linker doesn't know by default how to add these Fortran 77 libraries. Hence, this macro was created to determine these Fortran libraries.

The macros `AC_F77_DUMMY_MAIN`/`AC_FC_DUMMY_MAIN` or `AC_F77_MAIN`/`AC_FC_MAIN` will probably also be necessary to link C/C++ with Fortran; see below.

AC_F77_DUMMY_MAIN ([*action-if-found*], [*action-if-not-found*]) [Macro]
AC_FC_DUMMY_MAIN ([*action-if-found*], [*action-if-not-found*]) [Macro]

With many compilers, the Fortran libraries detected by `AC_F77_LIBRARY_LDFLAGS` or `AC_FC_LIBRARY_LDFLAGS` provide their own `main` entry function that initializes things like Fortran I/O, and which then calls a user-provided entry function named (say) `MAIN__` to run the user's program. The `AC_F77_DUMMY_MAIN`/`AC_FC_DUMMY_MAIN` or `AC_F77_MAIN`/`AC_FC_MAIN` macro figures out how to deal with this interaction.

When using Fortran for purely numerical functions (no I/O, etc.) often one prefers to provide one's own `main` and skip the Fortran library initializations. In this case, however, one may still need to provide a dummy `MAIN__` routine in order to prevent linking errors on some systems. `AC_F77_DUMMY_MAIN` or `AC_FC_DUMMY_MAIN` detects whether any such routine is *required* for linking, and what its name is; the shell variable `F77_DUMMY_MAIN` or `FC_DUMMY_MAIN` holds this name, `unknown` when no solution was found, and `none` when no such dummy main is needed.

By default, *action-if-found* defines `F77_DUMMY_MAIN` or `FC_DUMMY_MAIN` to the name of this routine (e.g., `MAIN__`) *if* it is required. [*action-if-not-found*] defaults to exiting with an error.

In order to link with Fortran routines, the user's C/C++ program should then include the following code to define the dummy main if it is needed:

```
#ifdef F77_DUMMY_MAIN
#  ifdef __cplusplus
     extern "C"
#  endif
   int F77_DUMMY_MAIN() { return 1; }
#endif
```

(Replace `F77` with `FC` for Fortran instead of Fortran 77.)

Note that this macro is called automatically from `AC_F77_WRAPPERS` or `AC_FC_WRAPPERS`; there is generally no need to call it explicitly unless one wants to change the default actions.

AC_F77_MAIN [Macro]
AC_FC_MAIN [Macro]

As discussed above, many Fortran libraries allow you to provide an entry point called (say) `MAIN__` instead of the usual `main`, which is then called by a `main` function in the Fortran libraries that initializes things like Fortran I/O. The `AC_F77_MAIN/AC_FC_MAIN` macro detects whether it is *possible* to utilize such an alternate main function, and defines `F77_MAIN/FC_MAIN` to the name of the function. (If no alternate main function name is found, `F77_MAIN/FC_MAIN` is simply defined to `main`.)

Thus, when calling Fortran routines from C that perform things like I/O, one should use this macro and name the "main" function `F77_MAIN/FC_MAIN` instead of `main`.

AC_F77_WRAPPERS [Macro]
AC_FC_WRAPPERS [Macro]

Defines C macros `F77_FUNC(name,NAME)/FC_FUNC(name,NAME)` and `F77_FUNC_(name,NAME)/FC_FUNC_(name,NAME)` to properly mangle the names of C/C++ identifiers, and identifiers with underscores, respectively, so that they match the name-mangling scheme used by the Fortran compiler.

Fortran is case-insensitive, and in order to achieve this the Fortran compiler converts all identifiers into a canonical case and format. To call a Fortran subroutine from C or to write a C function that is callable from Fortran, the C program must explicitly use identifiers in the format expected by the Fortran compiler. In order to do this, one simply wraps all C identifiers in one of the macros provided by `AC_F77_WRAPPERS` or `AC_FC_WRAPPERS`. For example, suppose you have the following Fortran 77 subroutine:

```
      subroutine foobar(x,y)
      double precision x, y
      y = 3.14159 * x
      return
      end
```

You would then declare its prototype in C or C++ as:

```
#define FOOBAR_F77 F77_FUNC(foobar,FOOBAR)
#ifdef __cplusplus
extern "C"  /* prevent C++ name mangling */
#endif
void FOOBAR_F77(double *x, double *y);
```

Note that we pass both the lowercase and uppercase versions of the function name to `F77_FUNC` so that it can select the right one. Note also that all parameters to Fortran 77 routines are passed as pointers (see section "Mixing Fortran 77 With C and C++" in GNU *Automake*).

(Replace `F77` with `FC` for Fortran instead of Fortran 77.)

Although Autoconf tries to be intelligent about detecting the name-mangling scheme of the Fortran compiler, there may be Fortran compilers that it doesn't support yet. In this case, the above code will generate a compile-time error, but some other behavior (e.g., disabling Fortran-related features) can be induced by checking whether the `F77_FUNC/FC_FUNC` macro is defined.

Now, to call that routine from a C program, we would do something like:

```
{
    double x = 2.7183, y;
    FOOBAR_F77(&x, &y);
}
```

If the Fortran identifier contains an underscore (e.g., `foo_bar`), you should use `F77_FUNC_`/`FC_FUNC_` instead of `F77_FUNC`/`FC_FUNC` (with the same arguments). This is because some Fortran compilers mangle names differently if they contain an underscore.

AC_F77_FUNC (*name*, [*shellvar*]) [Macro]
AC_FC_FUNC (*name*, [*shellvar*]) [Macro]

Given an identifier *name*, set the shell variable *shellvar* to hold the mangled version *name* according to the rules of the Fortran linker (see also `AC_F77_WRAPPERS` or `AC_FC_WRAPPERS`). *shellvar* is optional; if it is not supplied, the shell variable will be simply *name*. The purpose of this macro is to give the caller a way to access the name-mangling information other than through the C preprocessor as above, for example, to call Fortran routines from some language other than C/C++.

AC_FC_SRCEXT (*ext*, [*action-if-success*], [*action-if-failure*]) [Macro]

By default, the `FC` macros perform their tests using a '`.f`' extension for source-code files. Some compilers, however, only enable newer language features for appropriately named files, e.g. Fortran 90 features only for '`.f90`' files. On the other hand, some other compilers expect all source files to end in '`.f`' and require special flags to support other filename extensions. The `AC_FC_SRCEXT` macro deals with both of these issues.

The `AC_FC_SRCEXT` tries to get the `FC` compiler to accept files ending with the extension .*ext* (i.e. *ext* does *not* contain the dot). If any special compiler flags are needed for this, it stores them in the output variable `FCFLAGS_`*ext*. This extension and these flags are then used for all subsequent `FC` tests (until `AC_FC_SRCEXT` is called again).

For example, you would use `AC_FC_SRCEXT(f90)` to employ the '`.f90`' extension in future tests, and it would set a `FCFLAGS_f90` output variable with any extra flags that are needed to compile such files.

The `FCFLAGS_`*ext* can *not* be simply absorbed into `FCFLAGS`, for two reasons based on the limitations of some compilers. First, only one `FCFLAGS_`*ext* can be used at a time, so files with different extensions must be compiled separately. Second, `FCFLAGS_`*ext* must appear *immediately* before the source-code filename when compiling. So, continuing the example above, you might compile a '`foo.f90`' file in your Makefile with the command:

```
foo.o: foo.f90
     $(FC) -c $(FCFLAGS) $(FCFLAGS_f90) foo.f90
```

If `AC_FC_SRCEXT` succeeds in compiling files with the *ext* extension, it calls [*action-if-success*] (defaults to nothing). If it fails, and cannot find a way to make the `FC` compiler accept such files, it calls [*action-if-failure*] (defaults to exiting with an error message).

AC_FC_FREEFORM ([*action-if-success*], [*action-if-failure*]) [Macro]

The `AC_FC_FREEFORM` tries to ensure that the Fortran compiler (`$FC`) allows free-format source code (as opposed to the older fixed-format style from Fortran 77). If necessary, it may add some additional flags to `FCFLAGS`.

This macro is most important if you are using the default '`.f`' extension, since many compilers interpret this extension as indicating fixed-format source unless an additional flag is supplied. If you specify a different extension with `AC_FC_SRCEXT`, such as '`.f90`' or '`.f95`', then `AC_FC_FREEFORM` will ordinarily succeed without modifying `FCFLAGS`.

If `AC_FC_FREEFORM` succeeds in compiling free-form source, it calls [*action-if-success*] (defaults to nothing). If it fails, it calls [*action-if-failure*] (defaults to exiting with an error message).

5.11 System Services

The following macros check for operating system services or capabilities.

AC_PATH_X [Macro]

Try to locate the X Window System include files and libraries. If the user gave the command line options '`--x-includes=dir`' and '`--x-libraries=dir`', use those directories. If either or both were not given, get the missing values by running `xmkmf` on a trivial '`Imakefile`' and examining the '`Makefile`' that it produces. If that fails (such as if `xmkmf` is not present), look for the files in several directories where they often reside. If either method is successful, set the shell variables `x_includes` and `x_libraries` to their locations, unless they are in directories the compiler searches by default.

If both methods fail, or the user gave the command line option '`--without-x`', set the shell variable `no_x` to '`yes`'; otherwise set it to the empty string.

AC_PATH_XTRA [Macro]

An enhanced version of `AC_PATH_X`. It adds the C compiler flags that X needs to output variable `X_CFLAGS`, and the X linker flags to `X_LIBS`. Define `X_DISPLAY_MISSING` if X is not available.

This macro also checks for special libraries that some systems need in order to compile X programs. It adds any that the system needs to output variable `X_EXTRA_LIBS`. And it checks for special X11R6 libraries that need to be linked with before '`-lX11`', and adds any found to the output variable `X_PRE_LIBS`.

AC_SYS_INTERPRETER [Macro]

Check whether the system supports starting scripts with a line of the form '`#! /bin/csh`' to select the interpreter to use for the script. After running this macro, shell code in '`configure.ac`' can check the shell variable `interpval`; it will be set to '`yes`' if the system supports '`#!`', '`no`' if not.

AC_SYS_LARGEFILE [Macro]

Arrange for large-file support[1]. On some hosts, one must use special compiler options to build programs that can access large files. Append any such options to the output variable `CC`. Define `_FILE_OFFSET_BITS` and `_LARGE_FILES` if necessary.

Large-file support can be disabled by configuring with the '`--disable-largefile`' option.

If you use this macro, check that your program works even when `off_t` is longer than `long`, since this is common when large-file support is enabled. For example, it is not correct to print an arbitrary `off_t` value `X` with `printf ("%ld", (long) X)`.

The LFS introduced the `fseeko` and `ftello` functions to replace their C counterparts `fseek` and `ftell` that do not use `off_t`. Take care to use `AC_FUNC_FSEEKO` to make their prototypes available when using them and large-file support is enabled.

AC_SYS_LONG_FILE_NAMES [Macro]

If the system supports file names longer than 14 characters, define `HAVE_LONG_FILE_NAMES`.

AC_SYS_POSIX_TERMIOS [Macro]

Check to see if the POSIX termios headers and functions are available on the system. If so, set the shell variable `ac_cv_sys_posix_termios` to '`yes`'. If not, set the variable to '`no`'.

5.12 UNIX Variants

The following macros check for certain operating systems that need special treatment for some programs, due to exceptional oddities in their header files or libraries. These macros are warts; they will be replaced by a more systematic approach, based on the functions they make available or the environments they provide.

AC_AIX [Macro]

If on AIX, define `_ALL_SOURCE`. Allows the use of some BSD functions. Should be called before any macros that run the C compiler.

AC_GNU_SOURCE [Macro]

If using the GNU C library, define `_GNU_SOURCE`. Allows the use of some GNU functions. Should be called before any macros that run the C compiler.

AC_ISC_POSIX [Macro]

For INTERACTIVE UNIX (ISC), add '`-lcposix`' to output variable `LIBS` if necessary for POSIX facilities. Call this after `AC_PROG_CC` and before any other macros that use POSIX interfaces. INTERACTIVE UNIX is no longer sold, and Sun says that they will drop support for it on 2006-07-23, so this macro is becoming obsolescent.

AC_MINIX [Macro]

If on Minix, define `_MINIX` and `_POSIX_SOURCE` and define `_POSIX_1_SOURCE` to be 2. This allows the use of POSIX facilities. Should be called before any macros that run the C compiler.

---

[1] large-file support, `http://www.unix-systems.org/version2/whatsnew/lfs20mar.html`.

6 Writing Tests

If the existing feature tests don't do something you need, you have to write new ones. These macros are the building blocks. They provide ways for other macros to check whether various kinds of features are available and report the results.

This chapter contains some suggestions and some of the reasons why the existing tests are written the way they are. You can also learn a lot about how to write Autoconf tests by looking at the existing ones. If something goes wrong in one or more of the Autoconf tests, this information can help you understand the assumptions behind them, which might help you figure out how to best solve the problem.

These macros check the output of the compiler system of the current language (see Section 6.1 [Language Choice], page 71). They do not cache the results of their tests for future use (see Section 7.3 [Caching Results], page 83), because they don't know enough about the information they are checking for to generate a cache variable name. They also do not print any messages, for the same reason. The checks for particular kinds of features call these macros and do cache their results and print messages about what they're checking for.

When you write a feature test that could be applicable to more than one software package, the best thing to do is encapsulate it in a new macro. See Chapter 9 [Writing Autoconf Macros], page 103, for how to do that.

6.1 Language Choice

Autoconf-generated `configure` scripts check for the C compiler and its features by default. Packages that use other programming languages (maybe more than one, e.g., C and C++) need to test features of the compilers for the respective languages. The following macros determine which programming language is used in the subsequent tests in '`configure.ac`'.

AC_LANG (*language*) [Macro]

Do compilation tests using the compiler, preprocessor, and file extensions for the specified *language*.

Supported languages are:

'`C`' Do compilation tests using `CC` and `CPP` and use extension '`.c`' for test programs. Use compilation flags: `CPPFLAGS` with `CPP`, and both `CPPFLAGS` and `CFLAGS` with `CC`.

'`C++`' Do compilation tests using `CXX` and `CXXCPP` and use extension '`.C`' for test programs. Use compilation flags: `CPPFLAGS` with `CXXPP`, and both `CPPFLAGS` and `CXXFLAGS` with `CXX`.

'`Fortran 77`'
Do compilation tests using `F77` and use extension '`.f`' for test programs. Use compilation flags: `FFLAGS`.

'`Fortran`' Do compilation tests using `FC` and use extension '`.f`' (or whatever has been set by `AC_FC_SRCEXT`) for test programs. Use compilation flags: `FCFLAGS`.

AC_LANG_PUSH (*language*) [Macro]

Remember the current language (as set by `AC_LANG`) on a stack, and then select the *language*. Use this macro and `AC_LANG_POP` in macros that need to temporarily switch to a particular language.

AC_LANG_POP ([*language*]) [Macro]

Select the language that is saved on the top of the stack, as set by `AC_LANG_PUSH`, and remove it from the stack.

If given, *language* specifies the language we just *quit*. It is a good idea to specify it when it's known (which should be the case...), since Autoconf will detect inconsistencies.

```
AC_LANG_PUSH(Fortran 77)
# Perform some tests on Fortran 77.
# ...
AC_LANG_POP(Fortran 77)
```

AC_LANG_ASSERT (*language*) [Macro]

Check statically that the current language is *language*. You should use this in your language specific macros to avoid that they be called with an inappropriate language.

This macro runs only at `autoconf` time, and incurs no cost at `configure` time. Sadly enough and because Autoconf is a two layer language[1], the macros `AC_LANG_PUSH/AC_LANG_POP` cannot be "optimizing", therefore as much as possible you ought to avoid using them to wrap your code, rather, require from the user to run the macro with a correct current language, and check it with `AC_LANG_ASSERT`. And anyway, that may help the user understand she is running a Fortran macro while expecting a result about her Fortran 77 compiler...

AC_REQUIRE_CPP [Macro]

Ensure that whichever preprocessor would currently be used for tests has been found. Calls `AC_REQUIRE` (see Section 9.4.1 [Prerequisite Macros], page 105) with an argument of either `AC_PROG_CPP` or `AC_PROG_CXXCPP`, depending on which language is current.

6.2 Writing Test Programs

Autoconf tests follow is common scheme: feeding some program with some input, and most of the time, feeding a compiler with some source file. This section is dedicated to these source samples.

6.2.1 Guidelines for Test Programs

The most important rule to follow when writing testing samples is:

Look for realism.

This motto means that testing samples must be written with the same strictness as real programs are written. In particular, you should avoid "shortcuts" and simplifications.

[1] Because M4 is not aware of Sh code, especially conditionals, some optimizations that look nice statically may produce incorrect results at runtime.

Don't just play with the preprocessor if you want to prepare a compilation. For instance, using `cpp` to check if a header is functional might let your `configure` accept a header which will cause some *compiler* error. Do not hesitate checking header with other headers included before, especially required headers.

Make sure the symbols you use are properly defined, i.e., refrain for simply declaring a function yourself instead of including the proper header.

Test programs should not write anything to the standard output. They should return 0 if the test succeeds, nonzero otherwise, so that success can be distinguished easily from a core dump or other failure; segmentation violations and other failures produce a nonzero exit status. Test programs should `exit`, not `return`, from `main`, because on some systems (old Suns, at least) the argument to `return` in `main` is ignored.

Test programs can use `#if` or `#ifdef` to check the values of preprocessor macros defined by tests that have already run. For example, if you call `AC_HEADER_STDC`, then later on in '`configure.ac`' you can have a test program that includes an ANSI C header file conditionally:

```
#if STDC_HEADERS
# include <stdlib.h>
#endif
```

If a test program needs to use or create a data file, give it a name that starts with '`conftest`', such as '`conftest.data`'. The `configure` script cleans up by running '`rm -rf conftest*`' after running test programs and if the script is interrupted.

6.2.2 Test Functions

Function declarations in test programs should have a prototype conditionalized for C++. In practice, though, test programs rarely need functions that take arguments.

```
#ifdef __cplusplus
foo (int i)
#else
foo (i) int i;
#endif
```

Functions that test programs declare should also be conditionalized for C++, which requires '`extern "C"`' prototypes. Make sure to not include any header files containing clashing prototypes.

```
#ifdef __cplusplus
extern "C" void *malloc (size_t);
#else
void *malloc ();
#endif
```

If a test program calls a function with invalid parameters (just to see whether it exists), organize the program to ensure that it never invokes that function. You can do this by calling it in another function that is never invoked. You can't do it by putting it after a call to `exit`, because GCC version 2 knows that `exit` never returns and optimizes out any code that follows it in the same block.

If you include any header files, be sure to call the functions relevant to them with the correct number of arguments, even if they are just 0, to avoid compilation errors due to prototypes. GCC version 2 has internal prototypes for several functions that it automatically inlines; for example, `memcpy`. To avoid errors when checking for them, either pass them the correct number of arguments or redeclare them with a different return type (such as `char`).

6.2.3 Generating Sources

Autoconf provides a set of macros that can be used to generate test source files. They are written to be language generic, i.e., they actually depend on the current language (see Section 6.1 [Language Choice], page 71) to "format" the output properly.

AC_LANG_CONFTEST (*source*) [Macro]

Save the *source* text in the current test source file: '`conftest.`*`extension`*' where the *extension* depends on the current language.

Note that the *source* is evaluated exactly once, like regular Autoconf macro arguments, and therefore (i) you may pass a macro invocation, (ii) if not, be sure to double quote if needed.

AC_LANG_SOURCE (*source*) [Macro]

Expands into the *source*, with the definition of all the `AC_DEFINE` performed so far.

For instance executing (observe the double quotation!):

```
AC_INIT(Autoconf Documentation, 2.59, bug-autoconf@gnu.org)
AC_DEFINE([HELLO_WORLD], ["Hello, World\n"])
AC_LANG_CONFTEST(
   [AC_LANG_SOURCE([[const char hw[] = "Hello, World\n";]])])
gcc -E -dD conftest.c -o -
```

results in:

```
# 1 "conftest.c"
# 1169 "configure"

# 1 "confdefs.h" 1

#define PACKAGE_NAME "Autoconf Documentation"
#define PACKAGE_TARNAME "autoconf-documentation"
#define PACKAGE_VERSION "2.59"
#define PACKAGE_STRING "Autoconf Documentation 2.59"
#define PACKAGE_BUGREPORT "bug-autoconf@gnu.org"
#define HELLO_WORLD "Hello, World\n"
# 1170 "configure" 2

const char hw[] = "Hello, World\n";
```

AC_LANG_PROGRAM (*prologue, body*) [Macro]

Expands into a source file which consists of the *prologue*, and then *body* as body of the main function (e.g., `main` in C). Since it uses `AC_LANG_SOURCE`, the feature of the latter are available.

For instance:

```
AC_INIT(Autoconf Documentation, 2.59, bug-autoconf@gnu.org)
AC_DEFINE([HELLO_WORLD], ["Hello, World\n"])
AC_LANG_CONFTEST(
[AC_LANG_PROGRAM([[const char hw[] = "Hello, World\n";]],
                 [[fputs (hw, stdout);]])])
gcc -E -dD conftest.c -o -
```

results in:

```
# 1 "conftest.c"
# 1169 "configure"

# 1 "confdefs.h" 1

#define PACKAGE_NAME "Autoconf Documentation"
#define PACKAGE_TARNAME "autoconf-documentation"
#define PACKAGE_VERSION "2.59"
#define PACKAGE_STRING "Autoconf Documentation 2.59"
#define PACKAGE_BUGREPORT "bug-autoconf@gnu.org"
#define HELLO_WORLD "Hello, World\n"
# 1170 "configure" 2

const char hw[] = "Hello, World\n";
int
main ()
{
fputs (hw, stdout);
  ;
  return 0;
}
```

AC_LANG_CALL (*prologue, function*) [Macro]

Expands into a source file which consists of the *prologue*, and then a call to the *function* as body of the main function (e.g., `main` in C). Since it uses `AC_LANG_PROGRAMS`, the feature of the latter are available.

This function will probably be replaced in the future by a version which would enable specifying the arguments. The use of this macro is not encouraged, as it violates strongly the typing system.

AC_LANG_FUNC_LINK_TRY (*function*) [Macro]

Expands into a source file which consists of a pseudo use of the *function* as body of the main function (e.g., `main` in C): a simple (function pointer) assignment. Since it uses `AC_LANG_PROGRAMS`, the feature of the latter are available.

As `AC_LANG_CALL`, this macro is documented only for completeness. It is considered to be severely broken, and in the future will be removed in favor of actual function calls (with properly typed arguments).

6.3 Running the Preprocessor

Sometimes one might need to run the preprocessor on some source file. *Usually it is a bad idea*, as you typically need to *compile* your project, not merely run the preprocessor on it; therefore you certainly want to run the compiler, not the preprocessor. Resist to the temptation of following the easiest path.

Nevertheless, if you need to run the preprocessor, then use `AC_PREPROC_IFELSE`.

AC_PREPROC_IFELSE (*input*, [*action-if-true*], [*action-if-false*]) [Macro]

Run the preprocessor of the current language (see Section 6.1 [Language Choice], page 71) on the *input*, run the shell commands *action-if-true* on success, *action-if-false* otherwise. The *input* can be made by `AC_LANG_PROGRAM` and friends.

This macro uses `CPPFLAGS`, but not `CFLAGS`, because '`-g`', '`-O`', etc. are not valid options to many C preprocessors.

It is customary to report unexpected failures with `AC_MSG_FAILURE`.

For instance:

```
AC_INIT(Autoconf Documentation, 2.59, bug-autoconf@gnu.org)
AC_DEFINE([HELLO_WORLD], ["Hello, World\n"])
AC_PREPROC_IFELSE(
   [AC_LANG_PROGRAM([[const char hw[] = "Hello, World\n";]],
                    [[fputs (hw, stdout);]])],
   [AC_MSG_RESULT([OK])],
   [AC_MSG_FAILURE([unexpected preprocessor failure])])
```

results in:

```
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking how to run the C preprocessor... gcc -E
OK
```

The macro `AC_TRY_CPP` (see Section 15.4 [Obsolete Macros], page 167) used to play the role of `AC_PREPROC_IFELSE`, but double quotes its argument, making it impossible to use it to elaborate sources. You are encouraged to get rid of your old use of the macro `AC_TRY_CPP` in favor of `AC_PREPROC_IFELSE`, but, in the first place, are you sure you need to run the *preprocessor* and not the compiler?

AC_EGREP_HEADER (*pattern*, *header-file*, *action-if-found*, [*action-if-not-found*]) [Macro]

If the output of running the preprocessor on the system header file *header-file* matches the extended regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

AC_EGREP_CPP (*pattern, program,* [*action-if-found*], [*action-if-not-found*]) [Macro]

program is the text of a C or C++ program, on which shell variable, back quote, and backslash substitutions are performed. If the output of running the preprocessor on *program* matches the extended regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

6.4 Running the Compiler

To check for a syntax feature of the current language's (see Section 6.1 [Language Choice], page 71) compiler, such as whether it recognizes a certain keyword, or simply to try some library feature, use `AC_COMPILE_IFELSE` to try to compile a small program that uses that feature.

AC_COMPILE_IFELSE (*input,* [*action-if-found*], [*action-if-not-found*]) [Macro]

Run the compiler and compilation flags of the current language (see Section 6.1 [Language Choice], page 71) on the *input*, run the shell commands *action-if-true* on success, *action-if-false* otherwise. The *input* can be made by `AC_LANG_PROGRAM` and friends.

It is customary to report unexpected failures with `AC_MSG_FAILURE`. This macro does not try to link; use `AC_LINK_IFELSE` if you need to do that (see Section 6.5 [Running the Linker], page 77).

6.5 Running the Linker

To check for a library, a function, or a global variable, Autoconf `configure` scripts try to compile and link a small program that uses it. This is unlike Metaconfig, which by default uses `nm` or `ar` on the C library to try to figure out which functions are available. Trying to link with the function is usually a more reliable approach because it avoids dealing with the variations in the options and output formats of `nm` and `ar` and in the location of the standard libraries. It also allows configuring for cross-compilation or checking a function's run-time behavior if needed. On the other hand, it can be slower than scanning the libraries once, but accuracy is more important than speed.

`AC_LINK_IFELSE` is used to compile test programs to test for functions and global variables. It is also used by `AC_CHECK_LIB` to check for libraries (see Section 5.4 [Libraries], page 38), by adding the library being checked for to `LIBS` temporarily and trying to link a small program.

AC_LINK_IFELSE (*input,* [*action-if-found*], [*action-if-not-found*]) [Macro]

Run the compiler (and compilation flags) and the linker of the current language (see Section 6.1 [Language Choice], page 71) on the *input*, run the shell commands *action-if-true* on success, *action-if-false* otherwise. The *input* can be made by `AC_LANG_PROGRAM` and friends.

`LDFLAGS` and `LIBS` are used for linking, in addition to the current compilation flags.

It is customary to report unexpected failures with `AC_MSG_FAILURE`. This macro does not try to execute the program; use `AC_RUN_IFELSE` if you need to do that (see Section 6.6 [Run Time], page 78).

6.6 Checking Run Time Behavior

Sometimes you need to find out how a system performs at run time, such as whether a given function has a certain capability or bug. If you can, make such checks when your program runs instead of when it is configured. You can check for things like the machine's endianness when your program initializes itself.

If you really need to test for a run-time behavior while configuring, you can write a test program to determine the result, and compile and run it using `AC_RUN_IFELSE`. Avoid running test programs if possible, because this prevents people from configuring your package for cross-compiling.

AC_RUN_IFELSE (*input*, [*action-if-found*], [*action-if-not-found*], [*action-if-cross-compiling*]) [Macro]

If *program* compiles and links successfully and returns an exit status of 0 when executed, run shell commands *action-if-true*. Otherwise, run shell commands *action-if-false*.

The *input* can be made by `AC_LANG_PROGRAM` and friends. `LDFLAGS` and `LIBS` are used for linking, in addition to the compilation flags of the current language (see Section 6.1 [Language Choice], page 71).

If the compiler being used does not produce executables that run on the system where `configure` is being run, then the test program is not run. If the optional shell commands *action-if-cross-compiling* are given, they are run instead. Otherwise, `configure` prints an error message and exits.

In the *action-if-false* section, the exit status of the program is available in the shell variable '`$?`', but be very careful to limit yourself to positive values smaller than 127; bigger values should be saved into a file by the *program*. Note also that you have simply no guarantee that this exit status is issued by the *program*, or by the failure of its compilation. In other words, use this feature if sadist only, it was reestablished because the Autoconf maintainers grew tired of receiving "bug reports".

It is customary to report unexpected failures with `AC_MSG_FAILURE`.

Try to provide a pessimistic default value to use when cross-compiling makes run-time tests impossible. You do this by passing the optional last argument to `AC_RUN_IFELSE`. `autoconf` prints a warning message when creating `configure` each time it encounters a call to `AC_RUN_IFELSE` with no *action-if-cross-compiling* argument given. You may ignore the warning, though users will not be able to configure your package for cross-compiling. A few of the macros distributed with Autoconf produce this warning message.

To configure for cross-compiling you can also choose a value for those parameters based on the canonical system name (see Chapter 11 [Manual Configuration], page 149). Alternatively, set up a test results cache file with the correct values for the host system (see Section 7.3 [Caching Results], page 83).

To provide a default for calls of `AC_RUN_IFELSE` that are embedded in other macros, including a few of the ones that come with Autoconf, you can test whether the shell variable `cross_compiling` is set to '`yes`', and then use an alternate method to get the results instead of calling the macros.

6.7 Systemology

This section aims at presenting some systems and pointers to documentation. It may help you addressing particular problems reported by users.

The Rosetta Stone for Unix[2] contains a lot of interesting crossed information on various Unices.

Darwin
: Darwin is also known as Mac OS X. Beware that the file system *can* be case-preserving, but case insensitive. This can cause nasty problems, since for instance the installation attempt for a package having an '`INSTALL`' file can result in '`make install`' report that nothing was to be done!

  That's all dependent on whether the file system is a UFS (case sensitive) or HFS+ (case preserving). By default Apple wants you to install the OS on HFS+. Unfortunately, there are some pieces of software which really need to be built on UFS. We may want to rebuild Darwin to have both UFS and HFS+ available (and put the /local/build tree on the UFS).

QNX 4.25
: QNX is a realtime operating system running on Intel architecture meant to be scalable from the small embedded systems to the hundred processor supercomputer. It claims to be POSIX certified. More information is available on the QNX home page[3], including the QNX man pages[4].

Tru64
: The documentation of several versions of Tru64[5] is available in different formats.

Unix version 7
: Documentation is available in the V7 Manual[6].

6.8 Multiple Cases

Some operations are accomplished in several possible ways, depending on the UNIX variant. Checking for them essentially requires a "case statement". Autoconf does not directly provide one; however, it is easy to simulate by using a shell variable to keep track of whether a way to perform the operation has been found yet.

Here is an example that uses the shell variable `fstype` to keep track of whether the remaining cases need to be checked.

---

[2] Rosetta Stone for Unix, `http://bhami.com/rosetta.html`.

[3] QNX home page, `www.qnx.com`.

[4] QNX man pages, `http://support.qnx.com/support/docs/qnx4/`.

[5] documentation of several versions of Tru64, `http://www.tru64unix.compaq.com/docs/base_doc/DOCUMENTATION/`.

[6] V7 Manual, `http://plan9.bell-labs.com/7thEdMan/index.html`.

```
AC_MSG_CHECKING([how to get file system type])
fstype=no
# The order of these tests is important.
AC_COMPILE_IFELSE([AC_LANG_PROGRAM([[#include <sys/statvfs.h>
#include <sys/fstyp.h>]])],
                  [AC_DEFINE(FSTYPE_STATVFS) fstype=SVR4])
if test $fstype = no; then
  AC_COMPILE_IFELSE([AC_LANG_PROGRAM([[#include <sys/statfs.h>
#include <sys/fstyp.h>]])],
                  [AC_DEFINE(FSTYPE_USG_STATFS) fstype=SVR3])
fi
if test $fstype = no; then
  AC_COMPILE_IFELSE([AC_LANG_PROGRAM([[#include <sys/statfs.h>
#include <sys/vmount.h>]])]),
                  [AC_DEFINE(FSTYPE_AIX_STATFS) fstype=AIX])
fi
# (more cases omitted here)
AC_MSG_RESULT([$fstype])
```

7 Results of Tests

Once `configure` has determined whether a feature exists, what can it do to record that information? There are four sorts of things it can do: define a C preprocessor symbol, set a variable in the output files, save the result in a cache file for future `configure` runs, and print a message letting the user know the result of the test.

7.1 Defining C Preprocessor Symbols

A common action to take in response to a feature test is to define a C preprocessor symbol indicating the results of the test. That is done by calling `AC_DEFINE` or `AC_DEFINE_UNQUOTED`.

By default, `AC_OUTPUT` places the symbols defined by these macros into the output variable `DEFS`, which contains an option '`-D`*symbol*`=`*value*' for each symbol defined. Unlike in Autoconf version 1, there is no variable `DEFS` defined while `configure` is running. To check whether Autoconf macros have already defined a certain C preprocessor symbol, test the value of the appropriate cache variable, as in this example:

```
AC_CHECK_FUNC(vprintf, [AC_DEFINE(HAVE_VPRINTF)])
if test "$ac_cv_func_vprintf" != yes; then
  AC_CHECK_FUNC(_doprnt, [AC_DEFINE(HAVE_DOPRNT)])
fi
```

If `AC_CONFIG_HEADERS` has been called, then instead of creating `DEFS`, `AC_OUTPUT` creates a header file by substituting the correct values into `#define` statements in a template file. See Section 4.8 [Configuration Headers], page 26, for more information about this kind of output.

AC_DEFINE (*variable, value,* [*description*]) [Macro]
AC_DEFINE (*variable*) [Macro]

Define the C preprocessor variable *variable* to *value* (verbatim). *value* should not contain literal newlines, and if you are not using `AC_CONFIG_HEADERS` it should not contain any '`#`' characters, as `make` tends to eat them. To use a shell variable (which you need to do in order to define a value containing the M4 quote characters '`[`' or '`]`'), use `AC_DEFINE_UNQUOTED` instead. *description* is only useful if you are using `AC_CONFIG_HEADERS`. In this case, *description* is put into the generated '`config.h.in`' as the comment before the macro define. The following example defines the C preprocessor variable `EQUATION` to be the string constant '`"$a > $b"`':

```
AC_DEFINE(EQUATION, "$a > $b")
```

If neither *value* nor *description* are given, then *value* defaults to 1 instead of to the empty string. This is for backwards compatibility with older versions of Autoconf, but this usage is obsolescent and may be withdrawn in future versions of Autoconf.

AC_DEFINE_UNQUOTED (*variable, value,* [*description*]) [Macro]
AC_DEFINE_UNQUOTED (*variable*) [Macro]

Like `AC_DEFINE`, but three shell expansions are performed—once—on *variable* and *value*: variable expansion ('`$`'), command substitution ('`` ` ``'), and backslash escaping ('`\`'). Single and double quote characters in the value have no special meaning. Use this macro instead of `AC_DEFINE` when *variable* or *value* is a shell variable. Examples:

```
AC_DEFINE_UNQUOTED(config_machfile, "$machfile")
AC_DEFINE_UNQUOTED(GETGROUPS_T, $ac_cv_type_getgroups)
AC_DEFINE_UNQUOTED($ac_tr_hdr)
```

Due to a syntactical bizarreness of the Bourne shell, do not use semicolons to separate `AC_DEFINE` or `AC_DEFINE_UNQUOTED` calls from other macro calls or shell code; that can cause syntax errors in the resulting `configure` script. Use either spaces or newlines. That is, do this:

```
AC_CHECK_HEADER(elf.h, [AC_DEFINE(SVR4) LIBS="$LIBS -lelf"])
```

or this:

```
AC_CHECK_HEADER(elf.h,
  [AC_DEFINE(SVR4)
   LIBS="$LIBS -lelf"])
```

instead of this:

```
AC_CHECK_HEADER(elf.h, [AC_DEFINE(SVR4); LIBS="$LIBS -lelf"])
```

7.2 Setting Output Variables

Another way to record the results of tests is to set *output variables*, which are shell variables whose values are substituted into files that `configure` outputs. The two macros below create new output variables. See Section 4.7.1 [Preset Output Variables], page 20, for a list of output variables that are always available.

AC_SUBST (*variable*, [*value*]) [Macro]

Create an output variable from a shell variable. Make `AC_OUTPUT` substitute the variable *variable* into output files (typically one or more '`Makefile`'s). This means that `AC_OUTPUT` will replace instances of '`@variable@`' in input files with the value that the shell variable *variable* has when `AC_OUTPUT` is called. This value of *variable* should not contain literal newlines.

If *value* is given, in addition assign it to *variable*.

AC_SUBST_FILE (*variable*) [Macro]

Another way to create an output variable from a shell variable. Make `AC_OUTPUT` insert (without substitutions) the contents of the file named by shell variable *variable* into output files. This means that `AC_OUTPUT` will replace instances of '`@variable@`' in output files (such as '`Makefile.in`') with the contents of the file that the shell variable *variable* names when `AC_OUTPUT` is called. Set the variable to '`/dev/null`' for cases that do not have a file to insert.

This macro is useful for inserting '`Makefile`' fragments containing special dependencies or other `make` directives for particular host or target types into '`Makefile`'s. For example, '`configure.ac`' could contain:

```
AC_SUBST_FILE(host_frag)
host_frag=$srcdir/conf/sun4.mh
```

and then a '`Makefile.in`' could contain:

```
@host_frag@
```

Running `configure` in varying environments can be extremely dangerous. If for instance the user runs '`CC=bizarre-cc ./configure`', then the cache, '`config.h`', and many other output files will depend upon `bizarre-cc` being the C compiler. If for some reason the user runs `./configure` again, or if it is run via '`./config.status --recheck`', (See Section 4.7.4 [Automatic Remaking], page 25, and see Chapter 14 [config.status Invocation], page 163), then the configuration can be inconsistent, composed of results depending upon two different compilers.

Environment variables that affect this situation, such as '`CC`' above, are called *precious variables*, and can be declared as such by `AC_ARG_VAR`.

AC_ARG_VAR (*variable, description*) [Macro]

Declare *variable* is a precious variable, and include its *description* in the variable section of '`./configure --help`'.

Being precious means that

- *variable* is `AC_SUBST`'d.
- The value of *variable* when `configure` was launched is saved in the cache, including if it was not specified on the command line but via the environment. Indeed, while `configure` can notice the definition of `CC` in '`./configure CC=bizarre-cc`', it is impossible to notice it in '`CC=bizarre-cc ./configure`', which, unfortunately, is what most users do.

  We emphasize that it is the *initial* value of *variable* which is saved, not that found during the execution of `configure`. Indeed, specifying '`./configure FOO=foo`' and letting '`./configure`' guess that `FOO` is `foo` can be two very different runs.
- *variable* is checked for consistency between two `configure` runs. For instance:

  ```
  $ ./configure --silent --config-cache
  $ CC=cc ./configure --silent --config-cache
  configure: error: `CC' was not set in the previous run
  configure: error: changes in the environment can compromise \
  the build
  configure: error: run `make distclean' and/or \
  `rm config.cache' and start over
  ```

  and similarly if the variable is unset, or if its content is changed.
- *variable* is kept during automatic reconfiguration (see Chapter 14 [config.status Invocation], page 163) as if it had been passed as a command line argument, including when no cache is used:

  ```
  $ CC=/usr/bin/cc ./configure undeclared_var=raboof --silent
  $ ./config.status --recheck
  running /bin/sh ./configure undeclared_var=raboof --silent \
    CC=/usr/bin/cc  --no-create --no-recursion
  ```

7.3 Caching Results

To avoid checking for the same features repeatedly in various `configure` scripts (or in repeated runs of one script), `configure` can optionally save the results of many checks in a *cache file* (see Section 7.3.2 [Cache Files], page 85). If a `configure` script runs with caching enabled and finds a cache file, it reads the results of previous runs from the cache and avoids rerunning those checks. As a result, `configure` can then run much faster than if it had to perform all of the checks every time.

AC_CACHE_VAL (*cache-id, commands-to-set-it*) [Macro]

Ensure that the results of the check identified by *cache-id* are available. If the results of the check were in the cache file that was read, and `configure` was not given the '`--quiet`' or '`--silent`' option, print a message saying that the result was cached; otherwise, run the shell commands *commands-to-set-it*. If the shell commands are run to determine the value, the value will be saved in the cache file just before `configure` creates its output files. See Section 7.3.1 [Cache Variable Names], page 85, for how to choose the name of the *cache-id* variable.

The *commands-to-set-it must have no side effects* except for setting the variable *cache-id*, see below.

AC_CACHE_CHECK (*message, cache-id, commands-to-set-it*) [Macro]

A wrapper for `AC_CACHE_VAL` that takes care of printing the messages. This macro provides a convenient shorthand for the most common way to use these macros. It calls `AC_MSG_CHECKING` for *message*, then `AC_CACHE_VAL` with the *cache-id* and *commands* arguments, and `AC_MSG_RESULT` with *cache-id*.

The *commands-to-set-it must have no side effects* except for setting the variable *cache-id*, see below.

It is very common to find buggy macros using `AC_CACHE_VAL` or `AC_CACHE_CHECK`, because people are tempted to call `AC_DEFINE` in the *commands-to-set-it*. Instead, the code that *follows* the call to `AC_CACHE_VAL` should call `AC_DEFINE`, by examining the value of the cache variable. For instance, the following macro is broken:

```
AC_DEFUN([AC_SHELL_TRUE],
[AC_CACHE_CHECK([whether true(1) works], [ac_cv_shell_true_works],
                [ac_cv_shell_true_works=no
                 true && ac_cv_shell_true_works=yes
                 if test $ac_cv_shell_true_works = yes; then
                   AC_DEFINE([TRUE_WORKS], 1
                             [Define if 'true(1)' works properly.])
                 fi])
])
```

This fails if the cache is enabled: the second time this macro is run, `TRUE_WORKS` *will not be defined.* The proper implementation is:

```
AC_DEFUN([AC_SHELL_TRUE],
[AC_CACHE_CHECK([whether true(1) works], [ac_cv_shell_true_works],
                [ac_cv_shell_true_works=no
                 true && ac_cv_shell_true_works=yes])
 if test $ac_cv_shell_true_works = yes; then
   AC_DEFINE([TRUE_WORKS], 1
             [Define if 'true(1)' works properly.])
 fi
])
```

Also, *commands-to-set-it* should not print any messages, for example with `AC_MSG_CHECKING`; do that before calling `AC_CACHE_VAL`, so the messages are printed regardless of whether the results of the check are retrieved from the cache or determined by running the shell commands.

7.3.1 Cache Variable Names

The names of cache variables should have the following format:

    package-prefix_cv_value-type_specific-value_[additional-options]

for example, '`ac_cv_header_stat_broken`' or '`ac_cv_prog_gcc_traditional`'. The parts of the variable name are:

package-prefix
: An abbreviation for your package or organization; the same prefix you begin local Autoconf macros with, except lowercase by convention. For cache values used by the distributed Autoconf macros, this value is '`ac`'.

`_cv_`
: Indicates that this shell variable is a cache value. This string *must* be present in the variable name, including the leading underscore.

value-type
: A convention for classifying cache values, to produce a rational naming system. The values used in Autoconf are listed in Section 9.2 [Macro Names], page 103.

specific-value
: Which member of the class of cache values this test applies to. For example, which function ('`alloca`'), program ('`gcc`'), or output variable ('`INSTALL`').

additional-options
: Any particular behavior of the specific member that this test applies to. For example, '`broken`' or '`set`'. This part of the name may be omitted if it does not apply.

The values assigned to cache variables may not contain newlines. Usually, their values will be Boolean ('`yes`' or '`no`') or the names of files or functions; so this is not an important restriction.

7.3.2 Cache Files

A cache file is a shell script that caches the results of configure tests run on one system so they can be shared between configure scripts and configure runs. It is not useful on other systems. If its contents are invalid for some reason, the user may delete or edit it.

By default, `configure` uses no cache file (technically, it uses '`--cache-file=/dev/null`'), to avoid problems caused by accidental use of stale cache files.

To enable caching, `configure` accepts '`--config-cache`' (or '`-C`') to cache results in the file '`config.cache`'. Alternatively, '`--cache-file=`*`file`*' specifies that *file* be the cache file. The cache file is created if it does not exist already. When `configure` calls `configure` scripts in subdirectories, it uses the '`--cache-file`' argument so that they share the same cache. See Section 4.11 [Subdirectories], page 31, for information on configuring subdirectories with the `AC_CONFIG_SUBDIRS` macro.

'config.status' only pays attention to the cache file if it is given the '--recheck' option, which makes it rerun configure.

It is wrong to try to distribute cache files for particular system types. There is too much room for error in doing that, and too much administrative overhead in maintaining them. For any features that can't be guessed automatically, use the standard method of the canonical system type and linking files (see Chapter 11 [Manual Configuration], page 149).

The site initialization script can specify a site-wide cache file to use, instead of the usual per-program cache. In this case, the cache file will gradually accumulate information whenever someone runs a new configure script. (Running configure merges the new cache results with the existing cache file.) This may cause problems, however, if the system configuration (e.g., the installed libraries or compilers) changes and the stale cache file is not deleted.

7.3.3 Cache Checkpointing

If your configure script, or a macro called from 'configure.ac', happens to abort the configure process, it may be useful to checkpoint the cache a few times at key points using AC_CACHE_SAVE. Doing so will reduce the amount of time it takes to re-run the configure script with (hopefully) the error that caused the previous abort corrected.

AC_CACHE_LOAD [Macro]

Loads values from existing cache file, or creates a new cache file if a cache file is not found. Called automatically from AC_INIT.

AC_CACHE_SAVE [Macro]

Flushes all cached values to the cache file. Called automatically from AC_OUTPUT, but it can be quite useful to call AC_CACHE_SAVE at key points in 'configure.ac'.

For instance:

```
 ... AC_INIT, etc. ...
# Checks for programs.
AC_PROG_CC
AC_PROG_GCC_TRADITIONAL
 ... more program checks ...
AC_CACHE_SAVE

# Checks for libraries.
AC_CHECK_LIB(nsl, gethostbyname)
AC_CHECK_LIB(socket, connect)
 ... more lib checks ...
AC_CACHE_SAVE

# Might abort...
AM_PATH_GTK(1.0.2,, [AC_MSG_ERROR([GTK not in path])])
AM_PATH_GTKMM(0.9.5,, [AC_MSG_ERROR([GTK not in path])])
 ... AC_OUTPUT, etc. ...
```

7.4 Printing Messages

configure scripts need to give users running them several kinds of information. The following macros print messages in ways appropriate for each kind. The arguments to all of them get enclosed in shell double quotes, so the shell performs variable and back-quote substitution on them.

These macros are all wrappers around the `echo` shell command. `configure` scripts should rarely need to run `echo` directly to print messages for the user. Using these macros makes it easy to change how and when each kind of message is printed; such changes need only be made to the macro definitions and all of the callers will change automatically.

To diagnose static issues, i.e., when `autoconf` is run, see Section 9.3 [Reporting Messages], page 104.

AC_MSG_CHECKING (*feature-description*) [Macro]

Notify the user that `configure` is checking for a particular feature. This macro prints a message that starts with '`checking `' and ends with '`...`' and no newline. It must be followed by a call to `AC_MSG_RESULT` to print the result of the check and the newline. The *feature-description* should be something like '`whether the Fortran compiler accepts C++ comments`' or '`for c89`'.

This macro prints nothing if `configure` is run with the '`--quiet`' or '`--silent`' option.

AC_MSG_RESULT (*result-description*) [Macro]

Notify the user of the results of a check. *result-description* is almost always the value of the cache variable for the check, typically '`yes`', '`no`', or a file name. This macro should follow a call to `AC_MSG_CHECKING`, and the *result-description* should be the completion of the message printed by the call to `AC_MSG_CHECKING`.

This macro prints nothing if `configure` is run with the '`--quiet`' or '`--silent`' option.

AC_MSG_NOTICE (*message*) [Macro]

Deliver the *message* to the user. It is useful mainly to print a general description of the overall purpose of a group of feature checks, e.g.,

```
AC_MSG_NOTICE([checking if stack overflow is detectable])
```

This macro prints nothing if `configure` is run with the '`--quiet`' or '`--silent`' option.

AC_MSG_ERROR (*error-description*, [*exit-status*]) [Macro]

Notify the user of an error that prevents `configure` from completing. This macro prints an error message to the standard error output and exits `configure` with *exit-status* (1 by default). *error-description* should be something like '`invalid value $HOME for \$HOME`'.

The *error-description* should start with a lower-case letter, and "cannot" is preferred to "can't".

AC_MSG_FAILURE (*error-description*, [*exit-status*]) [Macro]

This `AC_MSG_ERROR` wrapper notifies the user of an error that prevents `configure` from completing *and* that additional details are provided in '`config.log`'. This is typically used when abnormal results are found during a compilation.

AC_MSG_WARN (*problem-description*) [Macro]

Notify the `configure` user of a possible problem. This macro prints the message to the standard error output; `configure` continues running afterward, so macros that call `AC_MSG_WARN` should provide a default (back-up) behavior for the situations they warn about. *problem-description* should be something like '`ln -s seems to make hard links`'.

8 Programming in M4

Autoconf is written on top of two layers: *M4sugar*, which provides convenient macros for pure M4 programming, and *M4sh*, which provides macros dedicated to shell script generation.

As of this version of Autoconf, these two layers are still experimental, and their interface might change in the future. As a matter of fact, *anything that is not documented must not be used.*

8.1 M4 Quotation

The most common problem with existing macros is an improper quotation. This section, which users of Autoconf can skip, but which macro writers *must* read, first justifies the quotation scheme that was chosen for Autoconf and then ends with a rule of thumb. Understanding the former helps one to follow the latter.

8.1.1 Active Characters

To fully understand where proper quotation is important, you first need to know what the special characters are in Autoconf: '`#`' introduces a comment inside which no macro expansion is performed, '`,`' separates arguments, '`[`' and '`]`' are the quotes themselves, and finally '`(`' and '`)`' (which M4 tries to match by pairs).

In order to understand the delicate case of macro calls, we first have to present some obvious failures. Below they are "obvious-ified", but when you find them in real life, they are usually in disguise.

Comments, introduced by a hash and running up to the newline, are opaque tokens to the top level: active characters are turned off, and there is no macro expansion:

```
# define([def], ine)
⇒# define([def], ine)
```

Each time there can be a macro expansion, there is a quotation expansion, i.e., one level of quotes is stripped:

```
int tab[10];
⇒int tab10;
[int tab[10];]
⇒int tab[10];
```

Without this in mind, the reader will try hopelessly to use her macro `array`:

```
define([array], [int tab[10];])
array
⇒int tab10;
[array]
⇒array
```

How can you correctly output the intended results[1]?

---

[1] Using `defn`.

8.1.2 One Macro Call

Let's proceed on the interaction between active characters and macros with this small macro, which just returns its first argument:

```
define([car], [$1])
```

The two pairs of quotes above are not part of the arguments of `define`; rather, they are understood by the top level when it tries to find the arguments of `define`. Therefore, it is equivalent to write:

```
define(car, $1)
```

But, while it is acceptable for a '`configure.ac`' to avoid unnecessary quotes, it is bad practice for Autoconf macros which must both be more robust and also advocate perfect style.

At the top level, there are only two possibilities: either you quote or you don't:

```
car(foo, bar, baz)
⇒foo
[car(foo, bar, baz)]
⇒car(foo, bar, baz)
```

Let's pay attention to the special characters:

```
car(#)
error EOF in argument list
```

The closing parenthesis is hidden in the comment; with a hypothetical quoting, the top level understood it this way:

```
car([#)]
```

Proper quotation, of course, fixes the problem:

```
car([#])
⇒#
```

The reader will easily understand the following examples:

```
car(foo, bar)
⇒foo
car([foo, bar])
⇒foo, bar
car((foo, bar))
⇒(foo, bar)
car([(foo], [bar)])
⇒(foo
car([], [])
⇒
car([[]], [[]])
⇒[]
```

With this in mind, we can explore the cases where macros invoke macros. . . .

8.1.3 Quotation and Nested Macros

The examples below use the following macros:

```
define([car], [$1])
define([active], [ACT, IVE])
define([array], [int tab[10]])
```

Each additional embedded macro call introduces other possible interesting quotations:

```
car(active)
⇒ACT
car([active])
⇒ACT, IVE
car([[active]])
⇒active
```

In the first case, the top level looks for the arguments of `car`, and finds '`active`'. Because M4 evaluates its arguments before applying the macro, '`active`' is expanded, which results in:

```
car(ACT, IVE)
⇒ACT
```

In the second case, the top level gives '`active`' as first and only argument of `car`, which results in:

```
active
⇒ACT, IVE
```

i.e., the argument is evaluated *after* the macro that invokes it. In the third case, `car` receives '`[active]`', which results in:

```
[active]
⇒active
```

exactly as we already saw above.

The example above, applied to a more realistic example, gives:

```
car(int tab[10];)
⇒int tab10;
car([int tab[10];])
⇒int tab10;
car([[int tab[10];]])
⇒int tab[10];
```

Huh? The first case is easily understood, but why is the second wrong, and the third right? To understand that, you must know that after M4 expands a macro, the resulting text is immediately subjected to macro expansion and quote removal. This means that the quote removal occurs twice—first before the argument is passed to the `car` macro, and second after the `car` macro expands to the first argument.

As the author of the Autoconf macro `car`, you then consider it to be incorrect that your users have to double-quote the arguments of `car`, so you "fix" your macro. Let's call it `qar` for quoted car:

```
define([qar], [[$1]])
```

and check that `qar` is properly fixed:

```
qar([int tab[10];])
⇒int tab[10];
```

Ahhh! That's much better.

But note what you've done: now that the arguments are literal strings, if the user wants to use the results of expansions as arguments, she has to use an *unquoted* macro call:

```
qar(active)
⇒ACT
```

where she wanted to reproduce what she used to do with `car`:

```
car([active])
⇒ACT, IVE
```

Worse yet: she wants to use a macro that produces a set of `cpp` macros:

```
define([my_includes], [#include <stdio.h>])
car([my_includes])
⇒#include <stdio.h>
qar(my_includes)
error EOF in argument list
```

This macro, `qar`, because it double quotes its arguments, forces its users to leave their macro calls unquoted, which is dangerous. Commas and other active symbols are interpreted by M4 before they are given to the macro, often not in the way the users expect. Also, because `qar` behaves differently from the other macros, it's an exception that should be avoided in Autoconf.

8.1.4 `changequote` is Evil

The temptation is often high to bypass proper quotation, in particular when it's late at night. Then, many experienced Autoconf hackers finally surrender to the dark side of the force and use the ultimate weapon: `changequote`.

The M4 builtin `changequote` belongs to a set of primitives that allow one to adjust the syntax of the language to adjust it to one's needs. For instance, by default M4 uses '`` ` ``' and '`'`' as quotes, but in the context of shell programming (and actually of most programming languages), that's about the worst choice one can make: because of strings and back-quoted expressions in shell code (such as '`'this'`' and '`` `that` ``'), because of literal characters in usual programming languages (as in '`'0'`'), there are many unbalanced '`` ` ``' and '`'`'. Proper M4 quotation then becomes a nightmare, if not impossible. In order to make M4 useful in such a context, its designers have equipped it with `changequote`, which makes it possible to choose another pair of quotes. M4sugar, M4sh, Autoconf, and Autotest all have chosen to use '`[`' and '`]`'. Not especially because they are unlikely characters, but *because they are characters unlikely to be unbalanced.*

There are other magic primitives, such as `changecom` to specify what syntactic forms are comments (it is common to see '`changecom(<!--, -->)`' when M4 is used to produce HTML pages), `changeword` and `changesyntax` to change other syntactic details (such as the character to denote the n-th argument, '`$`' by default, the parenthesis around arguments etc.).

These primitives are really meant to make M4 more useful for specific domains: they should be considered like command line options: '`--quotes`', '`--comments`', '`--words`', and `--syntax`. Nevertheless, they are implemented as M4 builtins, as it makes M4 libraries self contained (no need for additional options).

There lies the problem. . . .

The problem is that it is then tempting to use them in the middle of an M4 script, as opposed to its initialization. This, if not carefully thought out, can lead to disastrous effects: *you are changing the language in the middle of the execution.* Changing and restoring the syntax is often not enough: if you happened to invoke macros in between, these macros will be lost, as the current syntax will probably not be the one they were implemented with.

8.1.5 Quadrigraphs

When writing an Autoconf macro you may occasionally need to generate special characters that are difficult to express with the standard Autoconf quoting rules. For example, you may need to output the regular expression '`[^[]`', which matches any character other than '`[`'. This expression contains unbalanced brackets so it cannot be put easily into an M4 macro.

You can work around this problem by using one of the following *quadrigraphs*:

'`@<:@`' '`[`'

'`@:>@`' '`]`'

'`@S|@`' '`$`'

'`@%:@`' '`#`'

'`@&t@`' Expands to nothing.

Quadrigraphs are replaced at a late stage of the translation process, after `m4` is run, so they do not get in the way of M4 quoting. For example, the string '`^@<:@`', independently of its quotation, will appear as '`^[`' in the output.

The empty quadrigraph can be used:

- to mark trailing spaces explicitly

  Trailing spaces are smashed by `autom4te`. This is a feature.
- to produce other quadrigraphs

  For instance '`@<@&t@:@`' produces '`@<:@`'.
- to escape *occurrences* of forbidden patterns

  For instance you might want to mention `AC_FOO` in a comment, while still being sure that `autom4te` will still catch unexpanded '`AC_*`'. Then write '`AC@&t@_FOO`'.

The name '`@&t@`' was suggested by Paul Eggert:

> I should give some credit to the '`@&t@`' pun. The '`&`' is my own invention, but the '`t`' came from the source code of the ALGOL68C compiler, written by Steve Bourne (of Bourne shell fame), and which used '`mt`' to denote the empty string. In C, it would have looked like something like:
>
> ```
> char const mt[] = "";
> ```
>
> but of course the source code was written in Algol 68.
>
> I don't know where he got '`mt`' from: it could have been his own invention, and I suppose it could have been a common pun around the Cambridge University computer lab at the time.

8.1.6 Quotation Rule Of Thumb

To conclude, the quotation rule of thumb is:

One pair of quotes per pair of parentheses.

Never over-quote, never under-quote, in particular in the definition of macros. In the few places where the macros need to use brackets (usually in C program text or regular expressions), properly quote *the arguments*!

It is common to read Autoconf programs with snippets like:

```
AC_TRY_LINK(
changequote(<<, >>)dnl
<<#include <time.h>
#ifndef tzname /* For SGI.  */
extern char *tzname[]; /* RS6000 and others reject char **tzname.  */
#endif>>,
changequote([, ])dnl
[atoi (*tzname);], ac_cv_var_tzname=yes, ac_cv_var_tzname=no)
```

which is incredibly useless since `AC_TRY_LINK` is *already* double quoting, so you just need:

```
AC_TRY_LINK(
[#include <time.h>
#ifndef tzname /* For SGI.  */
extern char *tzname[]; /* RS6000 and others reject char **tzname.  */
#endif],
            [atoi (*tzname);],
            [ac_cv_var_tzname=yes],
            [ac_cv_var_tzname=no])
```

The M4-fluent reader will note that these two examples are rigorously equivalent, since M4 swallows both the '`changequote(<<, >>)`' and '`<<`' '`>>`' when it *collects* the arguments: these quotes are not part of the arguments!

Simplified, the example above is just doing this:

```
changequote(<<, >>)dnl
<<[]>>
changequote([, ])dnl
```

instead of simply:

```
[[]]
```

With macros that do not double quote their arguments (which is the rule), double-quote the (risky) literals:

```
AC_LINK_IFELSE([AC_LANG_PROGRAM(
[[#include <time.h>
#ifndef tzname /* For SGI.  */
extern char *tzname[]; /* RS6000 and others reject char **tzname.  */
#endif]],
                                [atoi (*tzname);])],
               [ac_cv_var_tzname=yes],
               [ac_cv_var_tzname=no])
```

See Section 8.1.5 [Quadrigraphs], page 93, for what to do if you run into a hopeless case where quoting does not suffice.

When you create a `configure` script using newly written macros, examine it carefully to check whether you need to add more quotes in your macros. If one or more words have disappeared in the M4 output, you need more quotes. When in doubt, quote.

However, it's also possible to put on too many layers of quotes. If this happens, the resulting `configure` script will contain unexpanded macros. The `autoconf` program checks for this problem by doing '`grep AC_ configure`'.

8.2 Using `autom4te`

The Autoconf suite, including M4sugar, M4sh, and Autotest, in addition to Autoconf per se, heavily rely on M4. All these different uses revealed common needs factored into a layer over `m4`: `autom4te`[2].

`autom4te` is a preprocessor that is like `m4`. It supports M4 extensions designed for use in tools like Autoconf.

8.2.1 Invoking `autom4te`

The command line arguments are modeled after M4's:

```
autom4te options files
```

where the *files* are directly passed to `m4`. In addition to the regular expansion, it handles the replacement of the quadrigraphs (see Section 8.1.5 [Quadrigraphs], page 93), and of '`__oline__`', the current line in the output. It supports an extended syntax for the *files*:

'`file.m4f`'
: This file is an M4 frozen file. Note that *all the previous files are ignored*. See the option '`--melt`' for the rationale.

'`file?`'
: If found in the library path, the *file* is included for expansion, otherwise it is ignored instead of triggering a failure.

Of course, it supports the Autoconf common subset of options:

'`--help`'
'`-h`'
: Print a summary of the command line options and exit.

'`--version`'
'`-V`'
: Print the version number of Autoconf and exit.

'`--verbose`'
'`-v`'
: Report processing steps.

'`--debug`'
'`-d`'
: Don't remove the temporary files and be even more verbose.

'`--include=dir`'
'`-I dir`'
: Also look for input files in *dir*. Multiple invocations accumulate.

[2] Yet another great name from Lars J. Aas.

'--output=*file*'
'-o *file*' Save output (script or trace) to *file*. The file '-' stands for the standard output.

As an extension of m4, it includes the following options:

'--warnings=*category*'
'-W *category*'
Report the warnings related to *category* (which can actually be a comma separated list). See Section 9.3 [Reporting Messages], page 104, macro AC_DIAGNOSE, for a comprehensive list of categories. Special values include:

'all' report all the warnings

'none' report none

'error' treats warnings as errors

'no-*category*'
disable warnings falling into *category*

Warnings about 'syntax' are enabled by default, and the environment variable WARNINGS, a comma separated list of categories, is honored. autom4te -W *category* will actually behave as if you had run:

```
autom4te --warnings=syntax,$WARNINGS,category
```

If you want to disable autom4te's defaults and WARNINGS, but (for example) enable the warnings about obsolete constructs, you would use '-W none,obsolete'.

autom4te displays a back trace for errors, but not for warnings; if you want them, just pass '-W error'. For instance, on this 'configure.ac':

```
AC_DEFUN([INNER],
[AC_RUN_IFELSE([AC_LANG_PROGRAM([exit (0)])])])

AC_DEFUN([OUTER],
[INNER])

AC_INIT
OUTER
```

you get:

```
$ autom4te -l autoconf -Wcross
configure.ac:8: warning: AC_RUN_IFELSE called without default \
to allow cross compiling
$ autom4te -l autoconf -Wcross,error -f
configure.ac:8: error: AC_RUN_IFELSE called without default \
to allow cross compiling
acgeneral.m4:3044: AC_RUN_IFELSE is expanded from...
configure.ac:2: INNER is expanded from...
configure.ac:5: OUTER is expanded from...
configure.ac:8: the top level
```

'--melt'
'-m' Do not use frozen files. Any argument *file*.m4f will be replaced with *file*.m4. This helps tracing the macros which are executed only when the files are frozen, typically m4_define. For instance, running:

```
autom4te --melt 1.m4 2.m4f 3.m4 4.m4f input.m4
```

is roughly equivalent to running:

```
m4 1.m4 2.m4 3.m4 4.m4 input.m4
```

while

```
autom4te 1.m4 2.m4f 3.m4 4.m4f input.m4
```

is equivalent to:

```
m4 --reload-state=4.m4f input.m4
```

'--freeze'
'-f' Produce a frozen state file. autom4te freezing is stricter than M4's: it must produce no warnings, and no output other than empty lines (a line with whitespace is *not* empty) and comments (starting with '#'). Please, note that contrary to m4, this options takes no argument:

```
autom4te 1.m4 2.m4 3.m4 --freeze --output=3.m4f
```

corresponds to

```
m4 1.m4 2.m4 3.m4 --freeze-state=3.m4f
```

'--mode=*octal-mode*'
'-m *octal-mode*'
Set the mode of the non-traces output to *octal-mode*; by default '0666'.

As another additional feature over m4, autom4te caches its results. GNU M4 is able to produce a regular output and traces at the same time. Traces are heavily used in the GNU Build System: autoheader uses them to build 'config.h.in', autoreconf to determine what GNU Build System components are used, automake to "parse" 'configure.ac' etc. To save the long runs of m4, traces are cached while performing regular expansion, and conversely. This cache is (actually, the caches are) stored in the directory 'autom4te.cache'. *It can safely be removed* at any moment (especially if for some reason autom4te considers it is trashed).

'--cache=*directory*'
'-C *directory*'
Specify the name of the directory where the result should be cached. Passing an empty value disables caching. Be sure to pass a relative path name, as for the time being, global caches are not supported.

'--no-cache'
Don't cache the results.

'--force'
'-f' If a cache is used, consider it obsolete (but update it anyway).

Because traces are so important to the GNU Build System, autom4te provides high level tracing features as compared to M4, and helps exploiting the cache:

'--trace=*macro*[:*format*]'
'-t *macro*[:*format*]'

Trace the invocations of *macro* according to the *format*. Multiple '--trace' arguments can be used to list several macros. Multiple '--trace' arguments for a single macro are not cumulative; instead, you should just make *format* as long as needed.

The *format* is a regular string, with newlines if desired, and several special escape codes. It defaults to '$f:$l:$n:$%'. It can use the following special escapes:

'$$' The character '$'.

'$f' The filename from which *macro* is called.

'$l' The line number from which *macro* is called.

'$d' The depth of the *macro* call. This is an M4 technical detail that you probably don't want to know about.

'$n' The name of the *macro*.

'$*num*' The *num*th argument of the call to *macro*.

'$@'
'$*sep*@'
'${*separator*}@'

All the arguments passed to *macro*, separated by the character *sep* or the string *separator* (',' by default). Each argument is quoted, i.e., enclosed in a pair of square brackets.

'$*'
'$*sep**'
'${*separator*}*'

As above, but the arguments are not quoted.

'$%'
'$*sep*%'
'${*separator*}%'

As above, but the arguments are not quoted, all new line characters in the arguments are smashed, and the default separator is ':'.

The escape '$%' produces single-line trace outputs (unless you put newlines in the 'separator'), while '$@' and '$*' do not.

See Section 3.4 [autoconf Invocation], page 10, for examples of trace uses.

'--preselect=*macro*'
'-p *macro*'

Cache the traces of *macro*, but do not enable traces. This is especially important to save CPU cycles in the future. For instance, when invoked, `autoconf` preselects all the macros that `autoheader`, `automake`, `autoreconf` etc. will trace, so that running `m4` is not needed to trace them: the cache suffices. This results in a huge speed-up.

Finally, `autom4te` introduces the concept of *Autom4te libraries*. They consists in a powerful yet extremely simple feature: sets of combined command line arguments:

'`--language=language`'
'`-l =language`'
: Use the *language* Autom4te library. Current languages include:

  `M4sugar` create M4sugar output.

  `M4sh` create M4sh executable shell scripts.

  `Autotest` create Autotest executable test suites.

  `Autoconf` create Autoconf executable configure scripts.

  `Autoconf-without-aclocal-m4`
  : create Autoconf executable configure scripts without reading '`aclocal.m4`'.

'`--prepend-include=dir`'
'`-B dir`'
: Prepend directory *dir* to the search path. This is used to include the language-specific files before any third-party macros.

As an example, if Autoconf is installed in its default location, '`/usr/local`', running '`autom4te -l m4sugar foo.m4`' is strictly equivalent to running '`autom4te --prepend-include /usr/local/share/autoconf m4sugar/m4sugar.m4f --warnings syntax foo.m4`'. Recursive expansion applies: running '`autom4te -l m4sh foo.m4`' is the same as '`autom4te --language M4sugar m4sugar/m4sh.m4f foo.m4`', i.e., '`autom4te --prepend-include /usr/local/share/autoconf m4sugar/m4sugar.m4f m4sugar/m4sh.m4f --mode 777 foo.m4`'. The definition of the languages is stored in '`autom4te.cfg`'.

8.2.2 Customizing autom4te

One can customize `autom4te` via '`~/.autom4te.cfg`' (i.e., as found in the user home directory), and '`./.autom4te.cfg`' (i.e., as found in the directory from which `autom4te` is run). The order is first reading '`autom4te.cfg`', then '`~/.autom4te.cfg`', then '`./.autom4te.cfg`', and finally the command line arguments.

In these text files, comments are introduced with `#`, and empty lines are ignored. Customization is performed on a per-language basis, wrapped in between a '`begin-language: "language"`', '`end-language: "language"`' pair.

Customizing a language stands for appending options (see Section 8.2.1 [autom4te Invocation], page 95) to the current definition of the language. Options, and more generally arguments, are introduced by '`args: arguments`'. You may use the traditional shell syntax to quote the *arguments*.

As an example, to disable Autoconf caches ('`autom4te.cache`') globally, include the following lines in '`~/.autom4te.cfg`':

```
## ------------------ ##
## User Preferences.  ##
## ------------------ ##
```

```
begin-language: "Autoconf"
args: --no-cache
end-language: "Autoconf"
```

8.3 Programming in M4sugar

M4 by itself provides only a small, but sufficient, set of all-purpose macros. M4sugar introduces additional generic macros. Its name was coined by Lars J. Aas: "Readability And Greater Understanding Stands 4 M4sugar".

8.3.1 Redefined M4 Macros

With a few exceptions, all the M4 native macros are moved in the '`m4_`' pseudo-namespace, e.g., M4sugar renames `define` as `m4_define` etc.

Some M4 macros are redefined, and are slightly incompatible with their native equivalent.

dnl [Macro]
: This macro kept its original name: no `m4_dnl` is defined.

m4_defn (*macro*) [Macro]
: Contrary to the M4 builtin, this macro fails if *macro* is not defined. See `m4_undefine`.

m4_exit (*exit-status*) [Macro]
: This macro corresponds to `m4exit`.

m4_if (*comment*) [Macro]
m4_if (*string-1, string-2, equal,* [*not-equal*]) [Macro]
m4_if (*string-1, string-2, equal, . . .*) [Macro]
: This macro corresponds to `ifelse`.

m4_undefine (*macro*) [Macro]
: Contrary to the M4 builtin, this macro fails if *macro* is not defined. Use

```
m4_ifdef([macro], [m4_undefine([macro])])
```

to recover the behavior of the builtin.

m4_bpatsubst (*string, regexp,* [*replacement*]) [Macro]
: This macro corresponds to `patsubst`. The name `m4_patsubst` is kept for future versions of M4sh, on top of GNU M4 which will provide extended regular expression syntax via `epatsubst`.

m4_popdef (*macro*) [Macro]
: Contrary to the M4 builtin, this macro fails if *macro* is not defined. See `m4_undefine`.

m4_bregexp (*string, regexp,* [*replacement*]) [Macro]
: This macro corresponds to `regexp`. The name `m4_regexp` is kept for future versions of M4sh, on top of GNU M4 which will provide extended regular expression syntax via `eregexp`.

m4_wrap (*text*) [Macro]
: This macro corresponds to `m4wrap`.

You are encouraged to end *text* with '`[]`', so that there are no risks that two consecutive invocations of `m4_wrap` result in an unexpected pasting of tokens, as in

```
m4_define([foo], [Foo])
m4_define([bar], [Bar])
m4_define([foobar], [FOOBAR])
m4_wrap([bar])
m4_wrap([foo])
⇒FOOBAR
```

8.3.2 Evaluation Macros

The following macros give some control over the order of the evaluation by adding or removing levels of quotes. They are meant for hard-core M4 programmers.

m4_dquote (*arg1, ...*) [Macro]
Return the arguments as a quoted list of quoted arguments.

m4_quote (*arg1, ...*) [Macro]
Return the arguments as a single entity, i.e., wrap them into a pair of quotes.

The following example aims at emphasizing the difference between (i), not using these macros, (ii), using `m4_quote`, and (iii), using `m4_dquote`.

```
$ cat example.m4
# Overquote, so that quotes are visible.
m4_define([show], [$[]1 = [$1], $[]@ = [$@]])
m4_divert(0)dnl
show(a, b)
show(m4_quote(a, b))
show(m4_dquote(a, b))
$ autom4te -l m4sugar example.m4
$1 = a, $@ = [a],[b]
$1 = a,b, $@ = [a,b]
$1 = [a],[b], $@ = [[a],[b]]
```

8.3.3 Forbidden Patterns

M4sugar provides a means to define suspicious patterns, patterns describing tokens which should not be found in the output. For instance, if an Autoconf '`configure`' script includes tokens such as '`AC_DEFINE`', or '`dnl`', then most probably something went wrong (typically a macro was not evaluated because of overquotation).

M4sugar forbids all the tokens matching '`^m4_`' and '`^dnl$`'.

m4_pattern_forbid (*pattern*) [Macro]
Declare that no token matching *pattern* must be found in the output. Comments are not checked; this can be a problem if, for instance, you have some macro left unexpanded after an '`#include`'. No consensus is currently found in the Autoconf community, as some people consider it should be valid to name macros in comments (which doesn't makes sense to the author of this documentation, as '`#`'-comments should document the output, not the input, documented by '`dnl`' comments).

Of course, you might encounter exceptions to these generic rules, for instance you might have to refer to '`$m4_flags`'.

m4_pattern_allow (*pattern*) [Macro]

Any token matching *pattern* is allowed, including if it matches an `m4_pattern_forbid` pattern.

8.4 Programming in M4sh

M4sh, pronounced "mash", is aiming at producing portable Bourne shell scripts. This name was coined by Lars J. Aas, who notes that, according to the Webster's Revised Unabridged Dictionary (1913):

> Mash \Mash\, n. [Akin to G. meisch, maisch, meische, maische, mash, wash, and prob. to AS. miscian to mix. See "Mix".]
>
> 1. A mass of mixed ingredients reduced to a soft pulpy state by beating or pressure....
> 2. A mixture of meal or bran and water fed to animals.
> 3. A mess; trouble. [Obs.] –Beau. & Fl.

For the time being, it is not mature enough to be widely used.

M4sh provides portable alternatives for some common shell constructs that unfortunately are not portable in practice.

AS_DIRNAME (*pathname*) [Macro]

Return the directory portion of *pathname*, using the algorithm required by POSIX. See Section 10.10 [Limitations of Usual Tools], page 131, for more details about what this returns and why it is more portable than the `dirname` command.

AS_IF (*test*, [*RUN-IF-TRUE*], [*RUN-IF-FALSE*]) [Macro]

Run shell code TEST. If TEST exits with a zero status then run shell code RUN-IF-TRUE, else run shell code RUN-IF-FALSE, with simplifications if either RUN-IF-TRUE or RUN-IF-FALSE is empty.

AS_MKDIR_P (*filename*) [Macro]

Make the directory *filename*, including intervening directories as necessary. This is equivalent to '`mkdir -p` *`filename`*', except that it is portable to older versions of `mkdir` that lack support for the '`-p`' option.

AS_SET_CATFILE (*var*, *dir*, *file*) [Macro]

Set the shell variable *var* to *dir*/*file*, but optimizing the common cases (*dir* or *file* is '`.`', *file* is absolute etc.).

9 Writing Autoconf Macros

When you write a feature test that could be applicable to more than one software package, the best thing to do is encapsulate it in a new macro. Here are some instructions and guidelines for writing Autoconf macros.

9.1 Macro Definitions

Autoconf macros are defined using the `AC_DEFUN` macro, which is similar to the M4 builtin `m4_define` macro. In addition to defining a macro, `AC_DEFUN` adds to it some code that is used to constrain the order in which macros are called (see Section 9.4.1 [Prerequisite Macros], page 105).

An Autoconf macro definition looks like this:

```
AC_DEFUN(macro-name, macro-body)
```

You can refer to any arguments passed to the macro as '`$1`', '`$2`', etc. See section "How to define new macros" in GNU *m4*, for more complete information on writing M4 macros.

Be sure to properly quote both the *macro-body and* the *macro-name* to avoid any problems if the macro happens to have been previously defined.

Each macro should have a header comment that gives its prototype, and a brief description. When arguments have default values, display them in the prototype. For example:

```
# AC_MSG_ERROR(ERROR, [EXIT-STATUS = 1])
# --------------------------------------
m4_define([AC_MSG_ERROR],
[{ _AC_ECHO([configure: error: $1], 2); exit m4_default([$2], 1); }])
```

Comments about the macro should be left in the header comment. Most other comments will make their way into '`configure`', so just keep using '`#`' to introduce comments.

If you have some very special comments about pure M4 code, comments that make no sense in '`configure`' and in the header comment, then use the builtin `dnl`: it causes M4 to discard the text through the next newline.

Keep in mind that `dnl` is rarely needed to introduce comments; `dnl` is more useful to get rid of the newlines following macros that produce no output, such as `AC_REQUIRE`.

9.2 Macro Names

All of the Autoconf macros have all-uppercase names starting with '`AC_`' to prevent them from accidentally conflicting with other text. All shell variables that they use for internal purposes have mostly-lowercase names starting with '`ac_`'. To ensure that your macros don't conflict with present or future Autoconf macros, you should prefix your own macro names and any shell variables they use with some other sequence. Possibilities include your initials, or an abbreviation for the name of your organization or software package.

Most of the Autoconf macros' names follow a structured naming convention that indicates the kind of feature check by the name. The macro names consist of several words, separated by underscores, going from most general to most specific. The names of their cache variables use the same convention (see Section 7.3.1 [Cache Variable Names], page 85, for more information on them).

The first word of the name after '`AC_`' usually tells the category of the feature being tested. Here are the categories used in Autoconf for specific test macros, the kind of macro that you are more likely to write. They are also used for cache variables, in all-lowercase. Use them where applicable; where they're not, invent your own categories.

`C` C language builtin features.

`DECL` Declarations of C variables in header files.

`FUNC` Functions in libraries.

`GROUP` UNIX group owners of files.

`HEADER` Header files.

`LIB` C libraries.

`PATH` The full path names to files, including programs.

`PROG` The base names of programs.

`MEMBER` Members of aggregates.

`SYS` Operating system features.

`TYPE` C builtin or declared types.

`VAR` C variables in libraries.

After the category comes the name of the particular feature being tested. Any further words in the macro name indicate particular aspects of the feature. For example, `AC_FUNC_UTIME_NULL` checks the behavior of the `utime` function when called with a `NULL` pointer.

An internal macro should have a name that starts with an underscore; Autoconf internals should therefore start with '`_AC_`'. Additionally, a macro that is an internal subroutine of another macro should have a name that starts with an underscore and the name of that other macro, followed by one or more words saying what the internal macro does. For example, `AC_PATH_X` has internal macros `_AC_PATH_X_XMKMF` and `_AC_PATH_X_DIRECT`.

9.3 Reporting Messages

When macros statically diagnose abnormal situations, benign or fatal, they should report them using these macros. For dynamic issues, i.e., when `configure` is run, see Section 7.4 [Printing Messages], page 86.

AC_DIAGNOSE (*category, message*) [Macro]

Report *message* as a warning (or as an error if requested by the user) if warnings of the *category* are turned on. You are encouraged to use standard categories, which currently include:

'`all`' messages that don't fall into one of the following categories. Use of an empty *category* is equivalent.

'`cross`' related to cross compilation issues.

'`obsolete`'
use of an obsolete construct.

'`syntax`' dubious syntactic constructs, incorrectly ordered macro calls.

AC_WARNING (*message*) [Macro]

Equivalent to 'AC_DIAGNOSE([syntax], *message*)', but you are strongly encouraged to use a finer grained category.

AC_FATAL (*message*) [Macro]

Report a severe error *message*, and have autoconf die.

When the user runs 'autoconf -W error', warnings from AC_DIAGNOSE and AC_WARNING are reported as error, see Section 3.4 [autoconf Invocation], page 10.

9.4 Dependencies Between Macros

Some Autoconf macros depend on other macros having been called first in order to work correctly. Autoconf provides a way to ensure that certain macros are called if needed and a way to warn the user if macros are called in an order that might cause incorrect operation.

9.4.1 Prerequisite Macros

A macro that you write might need to use values that have previously been computed by other macros. For example, AC_DECL_YYTEXT examines the output of flex or lex, so it depends on AC_PROG_LEX having been called first to set the shell variable LEX.

Rather than forcing the user of the macros to keep track of the dependencies between them, you can use the AC_REQUIRE macro to do it automatically. AC_REQUIRE can ensure that a macro is only called if it is needed, and only called once.

AC_REQUIRE (*macro-name*) [Macro]

If the M4 macro *macro-name* has not already been called, call it (without any arguments). Make sure to quote *macro-name* with square brackets. *macro-name* must have been defined using AC_DEFUN or else contain a call to AC_PROVIDE to indicate that it has been called.

AC_REQUIRE must be used inside an AC_DEFUN'd macro; it must not be called from the top level.

AC_REQUIRE is often misunderstood. It really implements dependencies between macros in the sense that if one macro depends upon another, the latter will be expanded *before* the body of the former. In particular, 'AC_REQUIRE(FOO)' is not replaced with the body of FOO. For instance, this definition of macros:

```
AC_DEFUN([TRAVOLTA],
[test "$body_temperature_in_celsius" -gt "38" &&
  dance_floor=occupied])
AC_DEFUN([NEWTON_JOHN],
[test "$hair_style" = "curly" &&
  dance_floor=occupied])

AC_DEFUN([RESERVE_DANCE_FLOOR],
[if date | grep '^Sat.*pm' >/dev/null 2>&1; then
  AC_REQUIRE([TRAVOLTA])
  AC_REQUIRE([NEWTON_JOHN])
fi])
```

with this 'configure.ac'

```
AC_INIT
RESERVE_DANCE_FLOOR
if test "$dance_floor" = occupied; then
  AC_MSG_ERROR([cannot pick up here, let's move])
fi
```

will not leave you with a better chance to meet a kindred soul at other times than Saturday night since it expands into:

```
test "$body_temperature_in_Celsius" -gt "38" &&
  dance_floor=occupied
test "$hair_style" = "curly" &&
  dance_floor=occupied
fi
if date | grep '^Sat.*pm' >/dev/null 2>&1; then


fi
```

This behavior was chosen on purpose: (i) it prevents messages in required macros from interrupting the messages in the requiring macros; (ii) it avoids bad surprises when shell conditionals are used, as in:

```
if ...; then
  AC_REQUIRE([SOME_CHECK])
fi
...
SOME_CHECK
```

You are encouraged to put all `AC_REQUIRE`s at the beginning of a macro. You can use `dnl` to avoid the empty lines they leave.

9.4.2 Suggested Ordering

Some macros should be run before another macro if both are called, but neither *requires* that the other be called. For example, a macro that changes the behavior of the C compiler should be called before any macros that run the C compiler. Many of these dependencies are noted in the documentation.

Autoconf provides the `AC_BEFORE` macro to warn users when macros with this kind of dependency appear out of order in a '`configure.ac`' file. The warning occurs when creating `configure` from '`configure.ac`', not when running `configure`.

For example, `AC_PROG_CPP` checks whether the C compiler can run the C preprocessor when given the '`-E`' option. It should therefore be called after any macros that change which C compiler is being used, such as `AC_PROG_CC`. So `AC_PROG_CC` contains:

```
AC_BEFORE([$0], [AC_PROG_CPP])dnl
```

This warns the user if a call to `AC_PROG_CPP` has already occurred when `AC_PROG_CC` is called.

AC_BEFORE (*this-macro-name, called-macro-name*) [Macro]

Make M4 print a warning message to the standard error output if *called-macro-name* has already been called. *this-macro-name* should be the name of the macro that is calling `AC_BEFORE`. The macro *called-macro-name* must have been defined using `AC_DEFUN` or else contain a call to `AC_PROVIDE` to indicate that it has been called.

9.5 Obsoleting Macros

Configuration and portability technology has evolved over the years. Often better ways of solving a particular problem are developed, or ad-hoc approaches are systematized. This process has occurred in many parts of Autoconf. One result is that some of the macros are now considered *obsolete*; they still work, but are no longer considered the best thing to do, hence they should be replaced with more modern macros. Ideally, `autoupdate` should replace the old macro calls with their modern implementation.

Autoconf provides a simple means to obsolete a macro.

AU_DEFUN (*old-macro*, *implementation*, [*message*]) [Macro]

Define *old-macro* as *implementation*. The only difference with `AC_DEFUN` is that the user will be warned that *old-macro* is now obsolete.

If she then uses `autoupdate`, the call to *old-macro* will be replaced by the modern *implementation*. The additional *message* is then printed.

9.6 Coding Style

The Autoconf macros follow a strict coding style. You are encouraged to follow this style, especially if you intend to distribute your macro, either by contributing it to Autoconf itself, or via other means.

The first requirement is to pay great attention to the quotation. For more details, see Section 3.1.2 [Autoconf Language], page 7, and Section 8.1 [M4 Quotation], page 89.

Do not try to invent new interfaces. It is likely that there is a macro in Autoconf that resembles the macro you are defining: try to stick to this existing interface (order of arguments, default values, etc.). We *are* conscious that some of these interfaces are not perfect; nevertheless, when harmless, homogeneity should be preferred over creativity.

Be careful about clashes both between M4 symbols and between shell variables.

If you stick to the suggested M4 naming scheme (see Section 9.2 [Macro Names], page 103), you are unlikely to generate conflicts. Nevertheless, when you need to set a special value, *avoid using a regular macro name*; rather, use an “impossible” name. For instance, up to version 2.13, the macro `AC_SUBST` used to remember what *symbol*s were already defined by setting `AC_SUBST_symbol`, which is a regular macro name. But since there is a macro named `AC_SUBST_FILE`, it was just impossible to ‘`AC_SUBST(FILE)`’! In this case, `AC_SUBST(symbol)` or `_AC_SUBST(symbol)` should have been used (yes, with the parentheses)... or better yet, high-level macros such as `AC_EXPAND_ONCE`.

No Autoconf macro should ever enter the user-variable name space; i.e., except for the variables that are the actual result of running the macro, all shell variables should start with `ac_`. In addition, small macros or any macro that is likely to be embedded in other macros should be careful not to use obvious names.

Do not use `dnl` to introduce comments: most of the comments you are likely to write are either header comments which are not output anyway, or comments that should make their way into ‘`configure`’. There are exceptional cases where you do want to comment special M4 constructs, in which case `dnl` is right, but keep in mind that it is unlikely.

M4 ignores the leading spaces before each argument, use this feature to indent in such a way that arguments are (more or less) aligned with the opening parenthesis of the macro being called. For instance, instead of

```
AC_CACHE_CHECK(for EMX OS/2 environment,
ac_cv_emxos2,
[AC_COMPILE_IFELSE([AC_LANG_PROGRAM(, [return __EMX__;])],
[ac_cv_emxos2=yes], [ac_cv_emxos2=no])])
```

write

```
AC_CACHE_CHECK([for EMX OS/2 environment], [ac_cv_emxos2],
[AC_COMPILE_IFELSE([AC_LANG_PROGRAM([], [return __EMX__;])],
                   [ac_cv_emxos2=yes],
                   [ac_cv_emxos2=no])])
```

or even

```
AC_CACHE_CHECK([for EMX OS/2 environment],
               [ac_cv_emxos2],
               [AC_COMPILE_IFELSE([AC_LANG_PROGRAM([],
                                                    [return __EMX__;])],
                                  [ac_cv_emxos2=yes],
                                  [ac_cv_emxos2=no])])
```

When using `AC_RUN_IFELSE` or any macro that cannot work when cross-compiling, provide a pessimistic value (typically '`no`').

Feel free to use various tricks to prevent auxiliary tools, such as syntax-highlighting editors, from behaving improperly. For instance, instead of:

```
m4_bpatsubst([$1], [$"])
```

use

```
m4_bpatsubst([$1], [$""])
```

so that Emacsen do not open an endless "string" at the first quote. For the same reasons, avoid:

```
test $[#] != 0
```

and use:

```
test $[@%:@] != 0
```

Otherwise, the closing bracket would be hidden inside a '`#`'-comment, breaking the bracket-matching highlighting from Emacsen. Note the preferred style to escape from M4: '`$[1]`', '`$[@]`', etc. Do not escape when it is unnecessary. Common examples of useless quotation are '`[$]$1`' (write '`$$1`'), '`[$]var`' (use '`$var`'), etc. If you add portability issues to the picture, you'll prefer '`${1+"$[@]"}`' to '`"[$]@"`', and you'll prefer do something better than hacking Autoconf `:-)`.

When using `sed`, don't use '`-e`' except for indenting purpose. With the `s` command, the preferred separator is '`/`' unless '`/`' itself is used in the command, in which case you should use '`,`'.

See Section 9.1 [Macro Definitions], page 103, for details on how to define a macro. If a macro doesn't use `AC_REQUIRE` and it is expected to never be the object of an `AC_REQUIRE` directive, then use `m4_define`. In case of doubt, use `AC_DEFUN`. All the `AC_REQUIRE` statements should be at the beginning of the macro, `dnl`'ed.

You should not rely on the number of arguments: instead of checking whether an argument is missing, test that it is not empty. It provides both a simpler and a more predictable interface to the user, and saves room for further arguments.

Unless the macro is short, try to leave the closing '`])`' at the beginning of a line, followed by a comment that repeats the name of the macro being defined. This introduces an additional newline in `configure`; normally, that is not a problem, but if you want to remove it you can use '`[]dnl`' on the last line. You can similarly use '`[]dnl`' after a macro call to remove its newline. '`[]dnl`' is recommended instead of '`dnl`' to ensure that M4 does not interpret the '`dnl`' as being attached to the preceding text or macro output. For example, instead of:

```
AC_DEFUN([AC_PATH_X],
[AC_MSG_CHECKING([for X])
AC_REQUIRE_CPP()
# ...omitted...
  AC_MSG_RESULT([libraries $x_libraries, headers $x_includes])
fi])
```

you would write:

```
AC_DEFUN([AC_PATH_X],
[AC_REQUIRE_CPP()[]dnl
AC_MSG_CHECKING([for X])
# ...omitted...
  AC_MSG_RESULT([libraries $x_libraries, headers $x_includes])
fi[]dnl
])# AC_PATH_X
```

If the macro is long, try to split it into logical chunks. Typically, macros that check for a bug in a function and prepare its `AC_LIBOBJ` replacement should have an auxiliary macro to perform this setup. Do not hesitate to introduce auxiliary macros to factor your code.

In order to highlight the recommended coding style, here is a macro written the old way:

```
dnl Check for EMX on OS/2.
dnl _AC_EMXOS2
AC_DEFUN(_AC_EMXOS2,
[AC_CACHE_CHECK(for EMX OS/2 environment, ac_cv_emxos2,
[AC_COMPILE_IFELSE([AC_LANG_PROGRAM(, return __EMX__;)],
ac_cv_emxos2=yes, ac_cv_emxos2=no)])
test "$ac_cv_emxos2" = yes && EMXOS2=yes])
```

and the new way:

```
# _AC_EMXOS2
# ----------
# Check for EMX on OS/2.
m4_define([_AC_EMXOS2],
[AC_CACHE_CHECK([for EMX OS/2 environment], [ac_cv_emxos2],
[AC_COMPILE_IFELSE([AC_LANG_PROGRAM([], [return __EMX__;])],
                   [ac_cv_emxos2=yes],
                   [ac_cv_emxos2=no])])
test "$ac_cv_emxos2" = yes && EMXOS2=yes[]dnl
```

```
])# _AC_EMXOS2
```

10 Portable Shell Programming

When writing your own checks, there are some shell-script programming techniques you should avoid in order to make your code portable. The Bourne shell and upward-compatible shells like the Korn shell and Bash have evolved over the years, but to prevent trouble, do not take advantage of features that were added after UNIX version 7, circa 1977 (see Section 6.7 [Systemology], page 79).

You should not use shell functions, aliases, negated character classes, or other features that are not found in all Bourne-compatible shells; restrict yourself to the lowest common denominator. Even `unset` is not supported by all shells! Also, include a space after the exclamation point in interpreter specifications, like this:

```
#! /usr/bin/perl
```

If you omit the space before the path, then 4.2BSD based systems (such as DYNIX) will ignore the line, because they interpret '`#! /`' as a 4-byte magic number. Some old systems have quite small limits on the length of the '`#!`' line too, for instance 32 bytes (not including the newline) on SunOS 4.

The set of external programs you should run in a `configure` script is fairly small. See section "Utilities in Makefiles" in GNU *Coding Standards*, for the list. This restriction allows users to start out with a fairly small set of programs and build the rest, avoiding too many interdependencies between packages.

Some of these external utilities have a portable subset of features; see Section 10.10 [Limitations of Usual Tools], page 131.

There are other sources of documentation about shells. See for instance the Shell FAQs[1].

10.1 Shellology

There are several families of shells, most prominently the Bourne family and the C shell family which are deeply incompatible. If you want to write portable shell scripts, avoid members of the C shell family. The the Shell difference FAQ[2] includes a small history of Unix shells, and a comparison between several of them.

Below we describe some of the members of the Bourne shell family.

Ash
: `ash` is often used on GNU/Linux and BSD systems as a light-weight Bourne-compatible shell. Ash 0.2 has some bugs that are fixed in the 0.3.x series, but portable shell scripts should work around them, since version 0.2 is still shipped with many GNU/Linux distributions.

  To be compatible with Ash 0.2:

  - don't use '`$?`' after expanding empty or unset variables:

    ```
    foo=
    false
    $foo
    echo "Don't use it: $?"
    ```

  - don't use command substitution within variable expansion:

[1] the Shell FAQs, `http://www.faqs.org/faqs/unix-faq/shell/`.

[2] the Shell difference FAQ, `http://www.faqs.org/faqs/unix-faq/shell/shell-differences/`.

```
cat ${FOO=`bar`}
```

– beware that single builtin substitutions are not performed by a subshell, hence their effect applies to the current shell! See Section 10.5 [Shell Substitutions], page 116, item "Command Substitution".

Bash
: To detect whether you are running `bash`, test if `BASH_VERSION` is set. To disable its extensions and require POSIX compatibility, run '`set -o posix`'. See section "Bash POSIX Mode" in *The* GNU *Bash Reference Manual*, for details.

Bash 2.05 and later
: Versions 2.05 and later of `bash` use a different format for the output of the `set` builtin, designed to make evaluating its output easier. However, this output is not compatible with earlier versions of `bash` (or with many other shells, probably). So if you use `bash` 2.05 or higher to execute `configure`, you'll need to use `bash` 2.05 for all other build tasks as well.

Ksh
: The Korn shell is compatible with the Bourne family and it mostly conforms to POSIX. It has two major variants commonly called '`ksh88`' and '`ksh93`', named after the years of initial release. It is usually called `ksh`, but Solaris systems have three variants: `/usr/bin/ksh` is '`ksh88`', `/usr/xpg4/bin/sh` is a POSIX-compliant variant of '`ksh88`', and `/usr/dt/bin/dtksh` is '`ksh93`'. `/usr/bin/ksh` is standard on Solaris; the other variants are parts of optional packages. There is no extra charge for these packages, but they are not part of a minimal OS install and therefore some installations may not have it. A public-domain clone of the Korn shell called '`pdksh`' is also widely available: it has most of the '`ksh88`' features along with a few of its own.

Zsh
: To detect whether you are running `zsh`, test if `ZSH_VERSION` is set. By default `zsh` is *not* compatible with the Bourne shell: you have to run '`emulate sh`' and set `NULLCMD` to '`:`'. See section "Compatibility" in *The Z Shell Manual*, for details.

  Zsh 3.0.8 is the native `/bin/sh` on Mac OS X 10.0.3.

The following discussion between Russ Allbery and Robert Lipe is worth reading:

Russ Allbery:

> The GNU assumption that `/bin/sh` is the one and only shell leads to a permanent deadlock. Vendors don't want to break users' existing shell scripts, and there are some corner cases in the Bourne shell that are not completely compatible with a POSIX shell. Thus, vendors who have taken this route will *never* (OK... "never say never") replace the Bourne shell (as `/bin/sh`) with a POSIX shell.

Robert Lipe:

> This is exactly the problem. While most (at least most System V's) do have a Bourne shell that accepts shell functions most vendor `/bin/sh` programs are not the POSIX shell.
>
> So while most modern systems do have a shell *somewhere* that meets the POSIX standard, the challenge is to find it.

10.2 Here-Documents

Don't rely on '\' being preserved just because it has no special meaning together with the next symbol. In the native `/bin/sh` on OpenBSD 2.7 '\"' expands to '"' in here-documents with unquoted delimiter. As a general rule, if '\\' expands to '\' use '\\' to get '\'.

With OpenBSD 2.7's `/bin/sh`

```
$ cat <<EOF
> \" \\
> EOF
" \
```

and with Bash:

```
bash-2.04$ cat <<EOF
> \" \\
> EOF
\" \
```

Many older shells (including the Bourne shell) implement here-documents inefficiently. And some shells mishandle large here-documents: for example, Solaris 8 `dtksh`, which is derived from `ksh` M-12/28/93d, mishandles variable expansion that occurs on 1024-byte buffer boundaries within a here-document. Users can generally fix these problems by using a faster or more reliable shell, e.g., by using the command '`bash ./configure`' rather than plain '`./configure`'.

Some shells can be extremely inefficient when there are a lot of here-documents inside a single statement. For instance if your '`configure.ac`' includes something like:

```
if <cross_compiling>; then
  assume this and that
else
  check this
  check that
  check something else
  ...
  on and on forever
  ...
fi
```

A shell parses the whole `if`/`fi` construct, creating temporary files for each here document in it. Some shells create links for such here-documents on every `fork`, so that the clean-up code they had installed correctly removes them. It is creating the links that can take the shell forever.

Moving the tests out of the `if`/`fi`, or creating multiple `if`/`fi` constructs, would improve the performance significantly. Anyway, this kind of construct is not exactly the typical use of Autoconf. In fact, it's even not recommended, because M4 macros can't look into shell conditionals, so we may fail to expand a macro when it was expanded before in a conditional path, and the condition turned out to be false at run-time, and we end up not executing the macro at all.

10.3 File Descriptors

Some file descriptors shall not be used, since some systems, admittedly arcane, use them for special purpose:

3 — some systems may open it to '`/dev/tty`'.
4 — used on the Kubota Titan.

Don't redirect the same file descriptor several times, as you are doomed to failure under Ultrix.

```
ULTRIX V4.4 (Rev. 69) System #31: Thu Aug 10 19:42:23 GMT 1995
UWS V4.4 (Rev. 11)
$ eval 'echo matter >fullness' >void
illegal io
$ eval '(echo matter >fullness)' >void
illegal io
$ (eval '(echo matter >fullness)') >void
Ambiguous output redirect.
```

In each case the expected result is of course '`fullness`' containing '`matter`' and '`void`' being empty.

Don't try to redirect the standard error of a command substitution: it must be done *inside* the command substitution: when running '`: `cd /zorglub` 2>/dev/null`' expect the error message to escape, while '`: `cd /zorglub 2>/dev/null``' works properly.

It is worth noting that Zsh (but not Ash nor Bash) makes it possible in assignments though: '`foo=`cd /zorglub` 2>/dev/null`'.

Most shells, if not all (including Bash, Zsh, Ash), output traces on stderr, even for sub-shells. This might result in undesirable content if you meant to capture the standard-error output of the inner command:

```
$ ash -x -c '(eval "echo foo >&2") 2>stderr'
$ cat stderr
+ eval echo foo >&2
+ echo foo
foo
$ bash -x -c '(eval "echo foo >&2") 2>stderr'
$ cat stderr
+ eval 'echo foo >&2'
++ echo foo
foo
$ zsh -x -c '(eval "echo foo >&2") 2>stderr'
# Traces on startup files deleted here.
$ cat stderr
+zsh:1> eval echo foo >&2
+zsh:1> echo foo
foo
```

You'll appreciate the various levels of detail. . . .

One workaround is to grep out uninteresting lines, hoping not to remove good ones. . . .

Don't try to move/delete open files, such as in '`exec >foo; mv foo bar`'; see Section 10.9 [Limitations of Builtins], page 124, `mv` for more details.

10.4 File System Conventions

While `autoconf` and friends will usually be run on some Unix variety, it can and will be used on other systems, most notably DOS variants. This impacts several assumptions regarding file and path names.

For example, the following code:

```
case $foo_dir in
  /*) # Absolute
     ;;
  *)
     foo_dir=$dots$foo_dir ;;
esac
```

will fail to properly detect absolute paths on those systems, because they can use a drivespec, and will usually use a backslash as directory separator. The canonical way to check for absolute paths is:

```
case $foo_dir in
  [\\/]* | ?:[\\/]* ) # Absolute
     ;;
  *)
     foo_dir=$dots$foo_dir ;;
esac
```

Make sure you quote the brackets if appropriate and keep the backslash as first character (see Section 10.9 [Limitations of Builtins], page 124).

Also, because the colon is used as part of a drivespec, these systems don't use it as path separator. When creating or accessing paths, use the `PATH_SEPARATOR` output variable instead. `configure` sets this to the appropriate value ('`:`' or '`;`') when it starts up.

File names need extra care as well. While DOS-based environments that are Unixy enough to run `autoconf` (such as DJGPP) will usually be able to handle long file names properly, there are still limitations that can seriously break packages. Several of these issues can be easily detected by the doschk[3] package.

A short overview follows; problems are marked with SFN/LFN to indicate where they apply: SFN means the issues are only relevant to plain DOS, not to DOS boxes under Windows, while LFN identifies problems that exist even under Windows.

No multiple dots (SFN)
: DOS cannot handle multiple dots in filenames. This is an especially important thing to remember when building a portable configure script, as `autoconf` uses a .in suffix for template files.

  This is perfectly OK on Unices:

  ```
  AC_CONFIG_HEADERS([config.h])
  AC_CONFIG_FILES([source.c foo.bar])
  ```

[3] doschk, `ftp://ftp.gnu.org/gnu/non-gnu/doschk/doschk-1.1.tar.gz`.

```
AC_OUTPUT
```

but it causes problems on DOS, as it requires 'config.h.in', 'source.c.in' and 'foo.bar.in'. To make your package more portable to DOS-based environments, you should use this instead:

```
AC_CONFIG_HEADERS([config.h:config.hin])
AC_CONFIG_FILES([source.c:source.cin foo.bar:foobar.in])
AC_OUTPUT
```

No leading dot (SFN)

DOS cannot handle filenames that start with a dot. This is usually not a very important issue for autoconf.

Case insensitivity (LFN)

DOS is case insensitive, so you cannot, for example, have both a file called 'INSTALL' and a directory called 'install'. This also affects make; if there's a file called 'INSTALL' in the directory, 'make install' will do nothing (unless the 'install' target is marked as PHONY).

The 8+3 limit (SFN)

Because the DOS file system only stores the first 8 characters of the filename and the first 3 of the extension, those must be unique. That means that 'foobar-part1.c', 'foobar-part2.c' and 'foobar-prettybird.c' all resolve to the same filename ('FOOBAR-P.C'). The same goes for 'foo.bar' and 'foo.bartender'.

Note: This is not usually a problem under Windows, as it uses numeric tails in the short version of filenames to make them unique. However, a registry setting can turn this behavior off. While this makes it possible to share file trees containing long file names between SFN and LFN environments, it also means the above problem applies there as well.

Invalid characters

Some characters are invalid in DOS filenames, and should therefore be avoided. In a LFN environment, these are '/', '\', '?', '*', ':', '<', '>', '|' and '"'. In a SFN environment, other characters are also invalid. These include '+', ',', '[' and ']'.

10.5 Shell Substitutions

Contrary to a persistent urban legend, the Bourne shell does not systematically split variables and back-quoted expressions, in particular on the right-hand side of assignments and in the argument of case. For instance, the following code:

```
case "$given_srcdir" in
.)  top_srcdir="`echo "$dots" | sed 's,/$,,'`"
*)  top_srcdir="$dots$given_srcdir" ;;
esac
```

is more readable when written as:

```
case $given_srcdir in
.)  top_srcdir=`echo "$dots" | sed 's,/$,,'`
```

```
*)  top_srcdir=$dots$given_srcdir ;;
esac
```

and in fact it is even *more* portable: in the first case of the first attempt, the computation of `top_srcdir` is not portable, since not all shells properly understand `"`...`"`...`"`...``"`. Worse yet, not all shells understand `"`...\"...\"...`"` the same way. There is just no portable way to use double-quoted strings inside double-quoted back-quoted expressions (pfew!).

`$@` One of the most famous shell-portability issues is related to '`"$@"`'. When there are no positional arguments, POSIX says that '`"$@"`' is supposed to be equivalent to nothing, but the original Unix Version 7 Bourne shell treated it as equivalent to '`""`' instead, and this behavior survives in later implementations like Digital Unix 5.0.

The traditional way to work around this portability problem is to use '`${1+"$@"}`'. Unfortunately this method does not work with Zsh (3.x and 4.x), which is used on Mac OS X. When emulating the Bourne shell, Zsh performs word splitting on '`${1+"$@"}`':

```
zsh $ emulate sh
zsh $ for i in "$@"; do echo $i; done
Hello World
!
zsh $ for i in ${1+"$@"}; do echo $i; done
Hello
World
!
```

Zsh handles plain '`"$@"`' properly, but we can't use plain '`"$@"`' because of the portability problems mentioned above. One workaround relies on Zsh's "global aliases" to convert '`${1+"$@"}`' into '`"$@"`' by itself:

```
test "${ZSH_VERSION+set}" = set && alias -g '${1+"$@"}'='"$@"'
```

A more conservative workaround is to avoid '`"$@"`' if it is possible that there may be no positional arguments. For example, instead of:

```
cat conftest.c "$@"
```

you can use this instead:

```
case $# in
0) cat conftest.c;;
*) cat conftest.c "$@";;
esac
```

`${var:-value}`
Old BSD shells, including the Ultrix `sh`, don't accept the colon for any shell substitution, and complain and die.

`${var=literal}`
Be sure to quote:

```
: ${var='Some words'}
```

otherwise some shells, such as on Digital Unix V 5.0, will die because of a "bad substitution".

Solaris' /bin/sh has a frightening bug in its interpretation of this. Imagine you need set a variable to a string containing '}'. This '}' character confuses Solaris' /bin/sh when the affected variable was already set. This bug can be exercised by running:

```
$ unset foo
$ foo=${foo='}'}
$ echo $foo
}
$ foo=${foo='}'   # no error; this hints to what the bug is
$ echo $foo
}
$ foo=${foo='}'}
$ echo $foo
}}
 ^ ugh!
```

It seems that '}' is interpreted as matching '${', even though it is enclosed in single quotes. The problem doesn't happen using double quotes.

`${var=expanded-value}`
On Ultrix, running

```
default="yu,yaa"
: ${var="$default"}
```

will set *var* to 'M-yM-uM-,M-yM-aM-a', i.e., the 8th bit of each char will be set. You won't observe the phenomenon using a simple 'echo $var' since apparently the shell resets the 8th bit when it expands $var. Here are two means to make this shell confess its sins:

```
$ cat -v <<EOF
$var
EOF
```

and

```
$ set | grep '^var=' | cat -v
```

One classic incarnation of this bug is:

```
default="a b c"
: ${list="$default"}
for c in $list; do
  echo $c
done
```

You'll get 'a b c' on a single line. Why? Because there are no spaces in '$list': there are 'M- ', i.e., spaces with the 8th bit set, hence no IFS splitting is performed!!!

One piece of good news is that Ultrix works fine with ': ${list=$default}'; i.e., if you *don't* quote. The bad news is then that QNX 4.25 then sets *list* to the *last* item of *default*!

The portable way out consists in using a double assignment, to switch the 8th bit twice on Ultrix:

```
list=${list="$default"}
```

. . . but beware of the '}' bug from Solaris (see above). For safety, use:

```
test "${var+set}" = set || var={value}
```

`commands`

While in general it makes no sense, do not substitute a single builtin with side effects, becauase Ash 0.2, trying to optimize, does not fork a subshell to perform the command.

For instance, if you wanted to check that `cd` is silent, do not use '`test -z "`cd /`"`' because the following can happen:

```
$ pwd
/tmp
$ test -z "`cd /`" && pwd
/
```

The result of '`foo=`exit 1``' is left as an exercise to the reader.

`$(`*`commands`*`)`

This construct is meant to replace '`` `commands` ``'; they can be nested while this is impossible to do portably with back quotes. Unfortunately it is not yet widely supported. Most notably, even recent releases of Solaris don't support it:

```
$ showrev -c /bin/sh | grep version
Command version: SunOS 5.8 Generic 109324-02 February 2001
$ echo $(echo blah)
syntax error: `(' unexpected
```

nor does IRIX 6.5's Bourne shell:

```
$ uname -a
IRIX firebird-image 6.5 07151432 IP22
$ echo $(echo blah)
$(echo blah)
```

If you do use '`$(commands)`', make sure that the commands do not start with a parenthesis, as that would cause confusion with a different notation '`$((expression))`' that in modern shells is an arithmetic expression not a command. To avoid the confusion, insert a space between the two opening parentheses.

10.6 Assignments

When setting several variables in a row, be aware that the order of the evaluation is undefined. For instance '`foo=1 foo=2; echo $foo`' gives '1' with sh on Solaris, but '2' with Bash. You must use '`;`' to enforce the order: '`foo=1; foo=2; echo $foo`'.

Don't rely on the following to find '`subdir/program`':

```
PATH=subdir$PATH_SEPARATOR$PATH program
```

as this does not work with Zsh 3.0.6. Use something like this instead:

```
(PATH=subdir$PATH_SEPARATOR$PATH; export PATH; exec program)
```

Don't rely on the exit status of an assignment: Ash 0.2 does not change the status and propagates that of the last statement:

```
$ false || foo=bar; echo $?
1
$ false || foo=`:`; echo $?
0
```

and to make things even worse, QNX 4.25 just sets the exit status to 0 in any case:

```
$ foo=`exit 1`; echo $?
0
```

To assign default values, follow this algorithm:

1. If the default value is a literal and does not contain any closing brace, use:
   ```
   : ${var='my literal'}
   ```
2. If the default value contains no closing brace, has to be expanded, and the variable being initialized will never be IFS-split (i.e., it's not a list), then use:
   ```
   : ${var="$default"}
   ```
3. If the default value contains no closing brace, has to be expanded, and the variable being initialized will be IFS-split (i.e., it's a list), then use:
   ```
   var=${var="$default"}
   ```
4. If the default value contains a closing brace, then use:
   ```
   test "${var+set}" = set || var='${indirection}'
   ```

In most cases '`var=${var="$default"}`' is fine, but in case of doubt, just use the latter. See Section 10.5 [Shell Substitutions], page 116, items '`${var:-value}`' and '`${var=value}`' for the rationale.

10.7 Parentheses in Shell Scripts

Beware of two opening parentheses in a row, as some shell implementations mishandle them. For example, '`pdksh`' 5.2.14 misparses the following code:

```
if ((true) || false); then
  echo ok
fi
```

To work around this problem, insert a space between the two opening parentheses. There is a similar problem and workaround with '`$((`'; see Section 10.5 [Shell Substitutions], page 116.

POSIX requires support for `case` patterns with opening parentheses like this:

```
case $filename in
(*.c) echo "C source code";;
esac
```

but the `(` in this example is not portable to many older Bourne shell implementations. It can be omitted safely.

10.8 Special Shell Variables

Some shell variables should not be used, since they can have a deep influence on the behavior of the shell. In order to recover a sane behavior from the shell, some variables should be unset, but `unset` is not portable (see Section 10.9 [Limitations of Builtins], page 124) and a fallback value is needed. We list these values below.

`CDPATH` When this variable is set it specifies a list of directories to search when invoking `cd` with a relative filename. POSIX 1003.1-2001 says that if a nonempty directory name from `CDPATH` is used successfully, `cd` prints the resulting absolute filename. Unfortunately this output can break idioms like '`abs=`cd src && pwd``' because `abs` receives the path twice. Also, many shells do not conform to this part of POSIX; for example, `zsh` prints the result only if a directory name other than '`.`' was chosen from `CDPATH`.

In practice the shells that have this problem also support `unset`, so you can work around the problem as follows:

```
(unset CDPATH) >/dev/null 2>&1 && unset CDPATH
```

Autoconf-generated scripts automatically unset `CDPATH` if possible, so you need not worry about this problem in those scripts.

`IFS` Don't set the first character of `IFS` to backslash. Indeed, Bourne shells use the first character (backslash) when joining the components in '`"$@"`' and some shells then re-interpret (!) the backslash escapes, so you can end up with backspace and other strange characters.

The proper value for `IFS` (in regular code, not when performing splits) is '⟨SPC⟩⟨TAB⟩⟨RET⟩'. The first character is especially important, as it is used to join the arguments in '`@*`'.

`LANG`
`LC_ALL`
`LC_COLLATE`
`LC_CTYPE`
`LC_MESSAGES`
`LC_MONETARY`
`LC_NUMERIC`
`LC_TIME`

Autoconf-generated scripts normally set all these variables to '`C`' because so much configuration code assumes the C locale and POSIX requires that locale environment variables be set to '`C`' if the C locale is desired. However, some older, nonstandard systems (notably SCO) break if locale environment variables are set to '`C`', so when running on these systems Autoconf-generated scripts unset the variables instead.

`LANGUAGE`

`LANGUAGE` is not specified by POSIX, but it is a GNU extension that overrides `LC_ALL` in some cases, so Autoconf-generated scripts set it too.

`LC_ADDRESS`
`LC_IDENTIFICATION`
`LC_MEASUREMENT`
`LC_NAME`
`LC_PAPER`
`LC_TELEPHONE`

These locale environment variables are GNU extensions. They are treated like their POSIX brethren (`LC_COLLATE`, etc.) as described above.

`LINENO` Most modern shells provide the current line number in `LINENO`. Its value is the line number of the beginning of the current command. Autoconf attempts to execute `configure` with a modern shell. If no such shell is available, it attempts to implement `LINENO` with a Sed prepass that replaces each instance of the string `$LINENO` (not followed by an alphanumeric character) with the line's number.

You should not rely on `LINENO` within `eval`, as the behavior differs in practice. Also, the possibility of the Sed prepass means that you should not rely on `$LINENO` when quoted, when in here-documents, or when in long commands that cross line boundaries. Subshells should be OK, though. In the following example, lines 1, 6, and 9 are portable, but the other instances of `LINENO` are not:

```
$ cat lineno
echo 1. $LINENO
cat <<EOF
3. $LINENO
4. $LINENO
EOF
( echo 6. $LINENO )
eval 'echo 7. $LINENO'
echo 8. '$LINENO'
echo 9. $LINENO '
10.' $LINENO
$ bash-2.05 lineno
1. 1
3. 2
4. 2
6. 6
7. 1
8. $LINENO
9. 9
10. 9
$ zsh-3.0.6 lineno
1. 1
3. 2
4. 2
6. 6
7. 7
8. $LINENO
9. 9
10. 9
```

```
$ pdksh-5.2.14 lineno
1. 1
3. 2
4. 2
6. 6
7. 0
8. $LINENO
9. 9
10. 9
$ sed '=' <lineno |
>   sed '
>     N
>     s,$,-,
>     : loop
>     s,^\([0-9]*\)\(.*\)[$]LINENO\([^a-zA-Z0-9_]\),\1\2\1\3,
>     t loop
>     s,-$,,
>     s,^[0-9]*\n,,
>   ' |
>   sh
1. 1
3. 3
4. 4
6. 6
7. 7
8. 8
9. 9
10. 10
```

`NULLCMD` When executing the command '`>foo`', `zsh` executes '`$NULLCMD >foo`'. The Bourne shell considers `NULLCMD` to be '`:`', while `zsh`, even in Bourne shell compatibility mode, sets `NULLCMD` to '`cat`'. If you forgot to set `NULLCMD`, your script might be suspended waiting for data on its standard input.

`ENV`
`MAIL`
`MAILPATH`
`PS1`
`PS2`
`PS4` These variables should not matter for shell scripts, since they are supposed to affect only interactive shells. However, at least one shell (the pre-3.0 UWIN `ksh`) gets confused about whether it is interactive, which means that (for example) a `PS1` with a side effect can unexpectedly modify '`$?`'. To work around this bug, Autoconf-generated scripts do something like this:

```
(unset ENV) >/dev/null 2>&1 && unset ENV MAIL MAILPATH
PS1='$ '
PS2='> '
PS4='+ '
```

`PWD` POSIX 1003.1-2001 requires that `cd` and `pwd` must update the `PWD` environment variable to point to the logical path to the current directory, but traditional shells do not support this. This can cause confusion if one shell instance maintains `PWD` but a subsidiary and different shell does not know about `PWD` and executes `cd`; in this case `PWD` will point to the wrong directory. Use '`pwd`' rather than '`$PWD`'.

`status` This variable is an alias to '`$?`' for `zsh` (at least 3.1.6), hence read-only. Do not use it.

`PATH_SEPARATOR`
If it is not set, `configure` will detect the appropriate path separator for the build system and set the `PATH_SEPARATOR` output variable accordingly.

On DJGPP systems, the `PATH_SEPARATOR` environment variable can be set to either '`:`' or '`;`' to control the path separator `bash` uses to set up certain environment variables (such as `PATH`). Since this only works inside `bash`, you want `configure` to detect the regular DOS path separator ('`;`'), so it can be safely substituted in files that may not support '`;`' as path separator. So it is recommended to either unset this variable or set it to '`;`'.

`RANDOM` Many shells provide `RANDOM`, a variable that returns a different integer each time it is used. Most of the time, its value does not change when it is not used, but on IRIX 6.5 the value changes all the time. This can be observed by using `set`.

10.9 Limitations of Shell Builtins

No, no, we are serious: some shells do have limitations! :)

You should always keep in mind that any builtin or command may support options, and therefore have a very different behavior with arguments starting with a dash. For instance, the innocent '`echo "$word"`' can give unexpected results when `word` starts with a dash. It is often possible to avoid this problem using '`echo "x$word"`', taking the '`x`' into account later in the pipe.

`.` Use `.` only with regular files (use '`test -f`'). Bash 2.03, for instance, chokes on '`. /dev/null`'. Also, remember that `.` uses `PATH` if its argument contains no slashes, so if you want to use `.` on a file '`foo`' in the current directory, you must use '`. ./foo`'.

`!` You can't use `!`; you'll have to rewrite your code.

`break` The use of '`break 2`' etc. is safe.

`cd` POSIX 1003.1-2001 requires that `cd` must support the '`-L`' ("logical") and '`-P`' ("physical") options, with '`-L`' being the default. However, traditional shells do not support these options, and their `cd` command has the '`-P`' behavior.

Portable scripts should assume neither option is supported, and should assume neither behavior is the default. This can be a bit tricky, since the POSIX default behavior means that, for example, '`ls ..`' and '`cd ..`' may refer to different directories if the current logical directory is a symbolic link. It is safe to use `cd` *`dir`* if *dir* contains no '`..`' components. Also, Autoconf-generated scripts check for this problem when computing variables like `ac_top_srcdir` (see Section 4.5 [Configuration Actions], page 18), so it is safe to `cd` to these variables.

Also please see the discussion of the `pwd` command.

`case` You don't need to quote the argument; no splitting is performed.

You don't need the final ';;', but you should use it.

Because of a bug in its `fnmatch`, `bash` fails to properly handle backslashes in character classes:

```
bash-2.02$ case /tmp in [/\\]*) echo OK;; esac
bash-2.02$
```

This is extremely unfortunate, since you are likely to use this code to handle UNIX or MS-DOS absolute paths. To work around this bug, always put the backslash first:

```
bash-2.02$ case '\TMP' in [\\/]*) echo OK;; esac
OK
bash-2.02$ case /tmp in [\\/]*) echo OK;; esac
OK
```

Some shells, such as Ash 0.3.8, are confused by an empty `case/esac`:

```
ash-0.3.8 $ case foo in esac;
error Syntax error: ";" unexpected (expecting ")")
```

Many shells still do not support parenthesized cases, which is a pity for those of us using tools that rely on balanced parentheses. For instance, Solaris 8's Bourne shell:

```
$ case foo in (foo) echo foo;; esac
error syntax error: `(' unexpected
```

`echo` The simple `echo` is probably the most surprising source of portability troubles. It is not possible to use '`echo`' portably unless both options and escape sequences are omitted. New applications which are not aiming at portability should use '`printf`' instead of '`echo`'.

Don't expect any option. See Section 4.7.1 [Preset Output Variables], page 20, `ECHO_N` etc. for a means to simulate '`-n`'.

Do not use backslashes in the arguments, as there is no consensus on their handling. On '`echo '\n' | wc -l`', the `sh` of Digital Unix 4.0 and MIPS RISC/OS 4.52, answer 2, but the Solaris' `sh`, Bash, and Zsh (in `sh` emulation mode) report 1. Please note that the problem is truly `echo`: all the shells understand '`'\n'`' as the string composed of a backslash and an '`n`'.

Because of these problems, do not pass a string containing arbitrary characters to `echo`. For example, '`echo "$foo"`' is safe if you know that *foo*'s value cannot contain backslashes and cannot start with '`-`', but otherwise you should use a here-document like this:

```
cat <<EOF
$foo
EOF
```

`exit` The default value of `exit` is supposed to be `$?`; unfortunately, some shells, such as the DJGPP port of Bash 2.04, just perform '`exit 0`'.

```
bash-2.04$ foo=`exit 1` || echo fail
fail
bash-2.04$ foo=`(exit 1)` || echo fail
fail
bash-2.04$ foo=`(exit 1); exit` || echo fail
bash-2.04$
```

Using '`exit $?`' restores the expected behavior.

Some shell scripts, such as those generated by `autoconf`, use a trap to clean up before exiting. If the last shell command exited with nonzero status, the trap also exits with nonzero status so that the invoker can tell that an error occurred.

Unfortunately, in some shells, such as Solaris 8 `sh`, an exit trap ignores the `exit` command's argument. In these shells, a trap cannot determine whether it was invoked by plain `exit` or by `exit 1`. Instead of calling `exit` directly, use the `AC_MSG_ERROR` macro that has a workaround for this problem.

`export` The builtin `export` dubs a shell variable *environment variable*. Each update of exported variables corresponds to an update of the environment variables. Conversely, each environment variable received by the shell when it is launched should be imported as a shell variable marked as exported.

Alas, many shells, such as Solaris 2.5, IRIX 6.3, IRIX 5.2, AIX 4.1.5, and Digital UNIX 4.0, forget to `export` the environment variables they receive. As a result, two variables coexist: the environment variable and the shell variable. The following code demonstrates this failure:

```
#! /bin/sh
echo $FOO
FOO=bar
echo $FOO
exec /bin/sh $0
```

when run with '`FOO=foo`' in the environment, these shells will print alternately '`foo`' and '`bar`', although it should only print '`foo`' and then a sequence of '`bar`'s.

Therefore you should `export` again each environment variable that you update.

`false` Don't expect `false` to exit with status 1: in the native Bourne shell of Solaris 8 it exits with status 255.

`for` To loop over positional arguments, use:

```
for arg
do
  echo "$arg"
done
```

You may *not* leave the `do` on the same line as `for`, since some shells improperly grok:

```
for arg; do
  echo "$arg"
done
```

If you want to explicitly refer to the positional arguments, given the '$@' bug (see Section 10.5 [Shell Substitutions], page 116), use:

```
for arg in ${1+"$@"}; do
  echo "$arg"
done
```

But keep in mind that Zsh, even in Bourne shell emulation mode, performs word splitting on '${1+"$@"}'; see Section 10.5 [Shell Substitutions], page 116, item '$@', for more.

`if` Using '!' is not portable. Instead of:

```
if ! cmp -s file file.new; then
  mv file.new file
fi
```

use:

```
if cmp -s file file.new; then :; else
  mv file.new file
fi
```

There are shells that do not reset the exit status from an `if`:

```
$ if (exit 42); then true; fi; echo $?
42
```

whereas a proper shell should have printed '0'. This is especially bad in Makefiles since it produces false failures. This is why properly written Makefiles, such as Automake's, have such hairy constructs:

```
if test -f "$file"; then
  install "$file" "$dest"
else
  :
fi
```

`printf` A format string starting with a '-' can cause problems. `bash` (eg. 2.05b) will interpret it as an options string and give an error. And '--' to mark the end of options is not good in the NetBSD Almquist shell (eg. 0.4.6) which will take that literally as the format string. Putting the '-' in a '%c' or '%s' is probably the easiest way to avoid doubt,

```
printf %s -foo
```

`pwd` With modern shells, plain `pwd` outputs a "logical" directory name, some of whose components may be symbolic links. These directory names are in contrast to "physical" directory names, whose components are all directories.

POSIX 1003.1-2001 requires that `pwd` must support the '-L' ("logical") and '-P' ("physical") options, with '-L' being the default. However, traditional shells do not support these options, and their `pwd` command has the '-P' behavior.

Portable scripts should assume neither option is supported, and should assume neither behavior is the default. Also, on many hosts '/bin/pwd' is equivalent to 'pwd -P', but POSIX does not require this behavior and portable scripts should not rely on it.

Typically it's best to use plain pwd. On modern hosts this outputs logical directory names, which have the following advantages:

- Logical names are what the user specified.
- Physical names may not be portable from one installation host to another due to network filesystem gymnastics.
- On modern hosts 'pwd -P' may fail due to lack of permissions to some parent directory, but plain pwd cannot fail for this reason.

Also please see the discussion of the cd command.

set
: This builtin faces the usual problem with arguments starting with a dash. Modern shells such as Bash or Zsh understand '--' to specify the end of the options (any argument after '--' is a parameter, even '-x' for instance), but most shells simply stop the option processing as soon as a non-option argument is found. Therefore, use 'dummy' or simply 'x' to end the option processing, and use shift to pop it out:

```
set x $my_list; shift
```

Some shells have the "opposite" problem of not recognizing all options (e.g., 'set -e -x' assigns '-x' to the command line). It is better to elide these:

```
set -ex
```

shift
: Not only is shifting a bad idea when there is nothing left to shift, but in addition it is not portable: the shell of MIPS RISC/OS 4.52 refuses to do it.

source
: This command is not portable, as POSIX does not require it; use . instead.

test
: The test program is the way to perform many file and string tests. It is often invoked by the alternate name '[', but using that name in Autoconf code is asking for trouble since it is an M4 quote character.

If you need to make multiple checks using test, combine them with the shell operators '&&' and '||' instead of using the test operators '-a' and '-o'. On System V, the precedence of '-a' and '-o' is wrong relative to the unary operators; consequently, POSIX does not specify them, so using them is nonportable. If you combine '&&' and '||' in the same statement, keep in mind that they have equal precedence.

You may use '!' with test, but not with if: 'test ! -r foo || exit 1'.

test (files)
: To enable configure scripts to support cross-compilation, they shouldn't do anything that tests features of the build system instead of the host system. But occasionally you may find it necessary to check whether some arbitrary file exists. To do so, use 'test -f' or 'test -r'. Do not use 'test -x', because 4.3BSD does not have it. Do not use 'test -e' either, because Solaris 2.5 does not have it. To test for symbolic links on systems that have them, use 'test

'-h' rather than 'test -L'; either form conforms to POSIX 1003.1-2001, but older shells like Solaris 8 /bin/sh support only '-h'.

test (strings)
: Avoid 'test "*string*"', in particular if *string* might start with a dash, since test might interpret its argument as an option (e.g., '*string* = "-n"').

Contrary to a common belief, 'test -n *string*' and 'test -z *string*' **are** portable. Nevertheless many shells (such as Solaris 2.5, AIX 3.2, UNICOS 10.0.0.6, Digital Unix 4 etc.) have bizarre precedence and may be confused if *string* looks like an operator:

```
$ test -n =
test: argument expected
```

If there are risks, use 'test "x*string*" = x' or 'test "x*string*" != x' instead.

It is common to find variations of the following idiom:

```
test -n "`echo $ac_feature | sed 's/[-a-zA-Z0-9_]//g'`" &&
  action
```

to take an action when a token matches a given pattern. Such constructs should always be avoided by using:

```
echo "$ac_feature" | grep '[^-a-zA-Z0-9_]' >/dev/null 2>&1 &&
  action
```

Use case where possible since it is faster, being a shell builtin:

```
case $ac_feature in
  *[!-a-zA-Z0-9_]*) action;;
esac
```

Alas, negated character classes are probably not portable, although no shell is known to not support the POSIX syntax '[!...]' (when in interactive mode, zsh is confused by the '[!...]' syntax and looks for an event in its history because of '!'). Many shells do not support the alternative syntax '[^...]' (Solaris, Digital Unix, etc.).

One solution can be:

```
expr "$ac_feature" : '.*[^-a-zA-Z0-9_]' >/dev/null &&
  action
```

or better yet

```
expr "x$ac_feature" : '.*[^-a-zA-Z0-9_]' >/dev/null &&
  action
```

'expr "X*foo*" : "X*bar*"' is more robust than 'echo "X*foo*" | grep "^X*bar*"', because it avoids problems when '*foo*' contains backslashes.

trap
: It is safe to trap at least the signals 1, 2, 13, and 15. You can also trap 0, i.e., have the trap run when the script ends (either via an explicit exit, or the end of the script).

Although POSIX is not absolutely clear on this point, it is widely admitted that when entering the trap '$?' should be set to the exit status of the last command run before the trap. The ambiguity can be summarized as: "when the trap is launched by an `exit`, what is the *last* command run: that before `exit`, or `exit` itself?"

Bash considers `exit` to be the last command, while Zsh and Solaris 8 `sh` consider that when the trap is run it is *still* in the `exit`, hence it is the previous exit status that the trap receives:

```
$ cat trap.sh
trap 'echo $?' 0
(exit 42); exit 0
$ zsh trap.sh
42
$ bash trap.sh
0
```

The portable solution is then simple: when you want to '`exit 42`', run '`(exit 42); exit 42`', the first `exit` being used to set the exit status to 42 for Zsh, and the second to trigger the trap and pass 42 as exit status for Bash.

The shell in FreeBSD 4.0 has the following bug: '`$?`' is reset to 0 by empty lines if the code is inside `trap`.

```
$ trap 'false

echo $?' 0
$ exit
0
```

Fortunately, this bug only affects `trap`.

`true` Don't worry: as far as we know `true` is portable. Nevertheless, it's not always a builtin (e.g., Bash 1.x), and the portable shell community tends to prefer using `:`. This has a funny side effect: when asked whether `false` is more portable than `true` Alexandre Oliva answered:

> In a sense, yes, because if it doesn't exist, the shell will produce an exit status of failure, which is correct for `false`, but not for `true`.

`unset` You cannot assume the support of `unset`. Nevertheless, because it is extremely useful to disable embarrassing variables such as `PS1`, you can test for its existence and use it *provided* you give a neutralizing value when `unset` is not supported:

```
if (unset FOO) >/dev/null 2>&1; then
  unset=unset
else
  unset=false
fi
$unset PS1 || PS1='$ '
```

See Section 10.8 [Special Shell Variables], page 120, for some neutralizing values. Also, see Section 10.9 [Limitations of Builtins], page 124, documentation of `export`, for the case of environment variables.

10.10 Limitations of Usual Tools

The small set of tools you can expect to find on any machine can still include some limitations you should be aware of.

`awk` Don't leave white spaces before the parentheses in user functions calls; GNU awk will reject it:

```
$ gawk 'function die () { print "Aaaaarg!"  }
        BEGIN { die () }'
gawk: cmd. line:2:        BEGIN { die () }
gawk: cmd. line:2:                          ^ parse error
$ gawk 'function die () { print "Aaaaarg!"  }
        BEGIN { die() }'
Aaaaarg!
```

If you want your program to be deterministic, don't depend on `for` on arrays:

```
$ cat for.awk
END {
  arr["foo"] = 1
  arr["bar"] = 1
  for (i in arr)
    print i
}
$ gawk -f for.awk </dev/null
foo
bar
$ nawk -f for.awk </dev/null
bar
foo
```

Some AWK, such as HPUX 11.0's native one, have regex engines fragile to inner anchors:

```
$ echo xfoo | $AWK '/foo|^bar/ { print }'
$ echo bar | $AWK '/foo|^bar/ { print }'
bar
$ echo xfoo | $AWK '/^bar|foo/ { print }'
xfoo
$ echo bar | $AWK '/^bar|foo/ { print }'
bar
```

Either do not depend on such patterns (i.e., use '`/^(.*foo|bar)/`', or use a simple test to reject such AWK.

`cat` Don't rely on any option. The option '`-v`', which displays non-printing characters, *seems* portable, though.

`cc` When a compilation such as '`cc foo.c -o foo`' fails, some compilers (such as CDS on Reliant UNIX) leave a '`foo.o`'.

HP-UX `cc` doesn't accept '`.S`' files to preprocess and assemble. '`cc -c foo.S`' will appear to succeed, but in fact does nothing.

The default executable, produced by '`cc foo.c`', can be

- '`a.out`' — usual Unix convention.
- '`b.out`' — i960 compilers (including `gcc`).
- '`a.exe`' — DJGPP port of `gcc`.
- '`a_out.exe`' — GNV `cc` wrapper for DEC C on OpenVMS.
- '`foo.exe`' — various MS-DOS compilers.

`cmp` `cmp` performs a raw data comparison of two files, while `diff` compares two text files. Therefore, if you might compare DOS files, even if only checking whether two files are different, use `diff` to avoid spurious differences due to differences of newline encoding.

`cp` Traditionally, file timestamps had 1-second resolution, and '`cp -p`' copied the timestamps exactly. However, many modern filesystems have timestamps with 1-nanosecond resolution. Unfortunately, '`cp -p`' implementations truncate timestamps when copying files, so this can result in the destination file appearing to be older than the source. The exact amount of truncation depends on the resolution of the system calls that `cp` uses; traditionally this was `utime`, which has 1-second resolution, but some newer `cp` implementations use `utimes`, which has 1-microsecond resolution. These newer implementations include GNU coreutils 5.0.91 or later, and Solaris 8 (sparc) patch 109933-02 or later. Unfortunately as of September 2003 there is still no system call to set time stamps to the full nanosecond resolution.

SunOS `cp` does not support '`-f`', although its `mv` does. It's possible to deduce why `mv` and `cp` are different with respect to '`-f`'. `mv` prompts by default before overwriting a read-only file. `cp` does not. Therefore, `mv` requires a '`-f`' option, but `cp` does not. `mv` and `cp` behave differently with respect to read-only files because the simplest form of `cp` cannot overwrite a read-only file, but the simplest form of `mv` can. This is because `cp` opens the target for write access, whereas `mv` simply calls `link` (or, in newer systems, `rename`).

Bob Proulx notes that '`cp -p`' always *tries* to copy ownerships. But whether it actually does copy ownerships or not is a system dependent policy decision implemented by the kernel. If the kernel allows it then it happens. If the kernel does not allow it then it does not happen. It is not something `cp` itself has control over.

In SysV any user can chown files to any other user, and SysV also had a non-sticky '`/tmp`'. That undoubtedly derives from the heritage of SysV in a business environment without hostile users. BSD changed this to be a more secure model where only root can `chown` files and a sticky '`/tmp`' is used. That undoubtedly derives from the heritage of BSD in a campus environment.

Linux by default follows BSD, but it can be configured to allow `chown`. HP-UX as an alternate example follows SysV, but it can be configured to use the modern security model and disallow `chown`. Since it is an administrator configurable parameter you can't use the name of the kernel as an indicator of the behavior.

`date` Some versions of `date` do not recognize special % directives, and unfortunately, instead of complaining, they just pass them through, and exit with success:

```
$ uname -a
OSF1 medusa.sis.pasteur.fr V5.1 732 alpha
$ date "+%s"
%s
```

`diff` Option '-u' is nonportable.

Some implementations, such as Tru64's, fail when comparing to '/dev/null'. Use an empty file instead.

`dirname` Not all hosts have a working `dirname`, and you should instead use `AS_DIRNAME` (see Section 8.4 [Programming in M4sh], page 102). For example:

```
dir=`dirname "$file"`       # This is not portable.
dir=`AS_DIRNAME(["$file"])` # This is more portable.
```

This handles a few subtleties in the standard way required by POSIX. For example, under UN*X, should 'dirname //1' give '/'? Paul Eggert answers:

> No, under some older flavors of Unix, leading '//' is a special path name: it refers to a "super-root" and is used to access other machines' files. Leading '///', '////', etc. are equivalent to '/'; but leading '//' is special. I think this tradition started with Apollo Domain/OS, an OS that is still in use on some older hosts.
>
> POSIX allows but does not require the special treatment for '//'. It says that the behavior of dirname on path names of the form '//([^/]+/*)?' is implementation defined. In these cases, GNU `dirname` returns '/', but it's more portable to return '//' as this works even on those older flavors of Unix.

`egrep` POSIX 1003.1-2001 no longer requires `egrep`, but many older hosts do not yet support the POSIX replacement `grep -E`. To work around this problem, invoke `AC_PROG_EGREP` and then use `$EGREP`.

The empty alternative is not portable, use '?' instead. For instance with Digital Unix v5.0:

```
> printf "foo\n|foo\n" | $EGREP '^(|foo|bar)$'
|foo
> printf "bar\nbar|\n" | $EGREP '^(foo|bar|)$'
bar|
> printf "foo\nfoo|\n|bar\nbar\n" | $EGREP '^(foo||bar)$'
foo
|bar
```

`$EGREP` also suffers the limitations of `grep`.

`expr` No `expr` keyword starts with 'x', so use 'expr x"*word*" : 'x*regex*'' to keep `expr` from misinterpreting *word*.

Don't use `length`, `substr`, `match` and `index`.

`expr` ('|') You can use '|'. Although POSIX does require that 'expr ''' return the empty string, it does not specify the result when you '|' together the empty string (or zero) with the empty string. For example:

```
expr '' \| ''
```

GNU/Linux and POSIX.2-1992 return the empty string for this case, but traditional UNIX returns '0' (Solaris is one such example). In POSIX.1-2001, the specification has been changed to match traditional UNIX's behavior (which is bizarre, but it's too late to fix this). Please note that the same problem does arise when the empty string results from a computation, as in:

```
expr bar : foo \| foo : bar
```

Avoid this portability problem by avoiding the empty string.

`expr` (':')
Don't use '`\?`', '`\+`' and '`\|`' in patterns, as they are not supported on Solaris.

The POSIX standard is ambiguous as to whether '`expr 'a' : '\(b\)''`' outputs '0' or the empty string. In practice, it outputs the empty string on most platforms, but portable scripts should not assume this. For instance, the QNX 4.25 native `expr` returns '0'.

One might think that a way to get a uniform behavior would be to use the empty string as a default value:

```
expr a : '\(b\)' \| ''
```

Unfortunately this behaves exactly as the original expression; see the '`expr (':')`' entry for more information.

Older `expr` implementations (e.g., SunOS 4 `expr` and Solaris 8 `/usr/ucb/expr`) have a silly length limit that causes `expr` to fail if the matched substring is longer than 120 bytes. In this case, you might want to fall back on '`echo|sed`' if `expr` fails.

Don't leave, there is some more!

The QNX 4.25 `expr`, in addition of preferring '0' to the empty string, has a funny behavior in its exit status: it's always 1 when parentheses are used!

```
$ val=`expr 'a' : 'a'`; echo "$?: $val"
0: 1
$ val=`expr 'a' : 'b'`; echo "$?: $val"
1: 0

$ val=`expr 'a' : '\(a\)'`; echo "?: $val"
1: a
$ val=`expr 'a' : '\(b\)'`; echo "?: $val"
1: 0
```

In practice this can be a big problem if you are ready to catch failures of `expr` programs with some other method (such as using `sed`), since you may get twice the result. For instance

```
$ expr 'a' : '\(a\)' || echo 'a' | sed 's/^\(a\)$/\1/'
```

will output 'a' on most hosts, but 'aa' on QNX 4.25. A simple workaround consists in testing `expr` and use a variable set to `expr` or to `false` according to the result.

`fgrep`
POSIX 1003.1-2001 no longer requires `fgrep`, but many older hosts do not yet support the POSIX replacement `grep -F`. To work around this problem, invoke `AC_PROG_FGREP` and then use `$FGREP`.

`find` The option '-maxdepth' seems to be GNU specific. Tru64 v5.1, NetBSD 1.5 and Solaris 2.5 `find` commands do not understand it.

The replacement of '{}' is guaranteed only if the argument is exactly `{}`, not if it's only a part of an argument. For instance on DU, and HP-UX 10.20 and HP-UX 11:

```
$ touch foo
$ find . -name foo -exec echo "{}-{}" \;
{}-{}
```

while GNU `find` reports './foo-./foo'.

`grep` Don't use 'grep -s' to suppress output, because 'grep -s' on System V does not suppress output, only error messages. Instead, redirect the standard output and standard error (in case the file doesn't exist) of `grep` to '/dev/null'. Check the exit status of `grep` to determine whether it found a match.

Don't use multiple regexps with '-e', as some `grep` will only honor the last pattern (e.g., IRIX 6.5 and Solaris 2.5.1). Anyway, Stardent Vistra SVR4 `grep` lacks '-e'. . . Instead, use extended regular expressions and alternation.

Don't rely on '-w', as Irix 6.5.16m's `grep` does not support it.

`ln` Don't rely on `ln` having a '-f' option. Symbolic links are not available on old systems; use '$(LN_S)' as a portable substitute.

For versions of the DJGPP before 2.04, `ln` emulates soft links to executables by generating a stub that in turn calls the real program. This feature also works with nonexistent files like in the Unix spec. So 'ln -s file link' will generate 'link.exe', which will attempt to call 'file.exe' if run. But this feature only works for executables, so 'cp -p' is used instead for these systems. DJGPP versions 2.04 and later have full symlink support.

`ls` The portable options are '-acdilrtu'. Modern practice is for '-l' to output both owner and group, but traditional `ls` omits the group.

Modern practice is for all diagnostics to go to standard error, but traditional 'ls foo' prints the message 'foo not found' to standard output if 'foo' does not exist. Be careful when writing shell commands like 'sources='ls *.c 2>/dev/null`', since with traditional `ls` this is equivalent to 'sources="*.c not found"' if there are no '.c' files.

`mkdir` None of `mkdir`'s options are portable. Instead of 'mkdir -p *filename*', you should use use `AS_MKDIR_P(`*filename*`)` (see Section 8.4 [Programming in M4sh], page 102).

`mv` The only portable options are '-f' and '-i'.

Moving individual files between file systems is portable (it was in V6), but it is not always atomic: when doing 'mv new existing', there's a critical section where neither the old nor the new version of 'existing' actually exists.

Be aware that moving files from '/tmp' can sometimes cause undesirable (but perfectly valid) warnings, even if you created these files. On some systems, creating the file in '/tmp' is setting a guid `wheel` which you may not be part of. So the file is copied, and then the `chgrp` fails:

```
$ touch /tmp/foo
$ mv /tmp/foo .
error mv: ./foo: set owner/group (was: 3830/0): Operation not permitted
$ echo $?
0
$ ls foo
foo
```

This behavior conforms to POSIX:

> If the duplication of the file characteristics fails for any reason, mv shall write a diagnostic message to standard error, but this failure shall not cause mv to modify its exit status."

Moving directories across mount points is not portable, use `cp` and `rm`.

Moving/Deleting open files isn't portable. The following can't be done on DOS/WIN32:

```
exec > foo
mv foo bar
```

nor can

```
exec > foo
rm -f foo
```

`sed` Patterns should not include the separator (unless escaped), even as part of a character class. In conformance with POSIX, the Cray `sed` will reject '`s/[^/]*$//`': use '`s,[^/]*$,,`'.

Sed scripts should not use branch labels longer than 8 characters and should not contain comments.

Don't include extra '`;`', as some `sed`, such as NetBSD 1.4.2's, try to interpret the second as a command:

```
$ echo a | sed 's/x/x/;;s/x/x/'
sed: 1: "s/x/x/;;s/x/x/": invalid command code ;
```

Input should have reasonably long lines, since some `sed` have an input buffer limited to 4000 bytes.

Alternation, '`\|`', is common but POSIX does not require its support, so it should be avoided in portable scripts. Solaris 8 `sed` does not support alternation; e.g., '`sed '/a\|b/d'`' deletes only lines that contain the literal string '`a|b`'.

Anchors ('`^`' and '`$`') inside groups are not portable.

Nested parenthesization in patterns (e.g., '`\(\(a*\)b*)\)`') is quite portable to modern hosts, but is not supported by some older `sed` implementations like SVR3.

Of course the option '`-e`' is portable, but it is not needed. No valid Sed program can start with a dash, so it does not help disambiguating. Its sole usefulness is to help enforcing indentation as in:

```
sed -e instruction-1 \
    -e instruction-2
```

as opposed to

```
sed instruction-1;instruction-2
```

Contrary to yet another urban legend, you may portably use '&' in the replacement part of the s command to mean "what was matched". All descendants of Bell Lab's V7 sed (at least; we don't have first hand experience with older seds) have supported it.

POSIX requires that you must not have any white space between '!' and the following command. It is OK to have blanks between the address and the '!'. For instance, on Solaris 8:

```
$ echo "foo" | sed -n '/bar/ ! p'
error Unrecognized command: /bar/ ! p
$ echo "foo" | sed -n '/bar/! p'
error Unrecognized command: /bar/! p
$ echo "foo" | sed -n '/bar/ !p'
foo
```

sed ('t') Some old systems have sed that "forget" to reset their 't' flag when starting a new cycle. For instance on MIPS RISC/OS, and on IRIX 5.3, if you run the following sed script (the line numbers are not actual part of the texts):

```
s/keep me/kept/g  # a
t end             # b
s/.*/deleted/g    # c
: end             # d
```

on

```
delete me         # 1
delete me         # 2
keep me           # 3
delete me         # 4
```

you get

```
deleted
delete me
kept
deleted
```

instead of

```
deleted
deleted
kept
deleted
```

Why? When processing 1, a matches, therefore sets the t flag, b jumps to d, and the output is produced. When processing line 2, the t flag is still set (this is the bug). Line a fails to match, but sed is not supposed to clear the t flag when a substitution fails. Line b sees that the flag is set, therefore it clears it, and jumps to d, hence you get 'delete me' instead of 'deleted'. When processing 3, t is clear, a matches, so the flag is set, hence b clears the flags and jumps. Finally, since the flag is clear, 4 is processed properly.

There are two things one should remember about 't' in `sed`. Firstly, always remember that 't' jumps if *some* substitution succeeded, not only the immediately preceding substitution. Therefore, always use a fake '`t clear; : clear`' to reset the t flag where indeed.

Secondly, you cannot rely on `sed` to clear the flag at each new cycle.

One portable implementation of the script above is:

```
t clear
: clear
s/keep me/kept/g
t end
s/.*/deleted/g
: end
```

`touch` If you specify the desired timestamp (e.g., with the '`-r`' option), `touch` typically uses the `utime` or `utimes` system call, which can result in the same kind of timestamp truncation problems that '`cp -p`' has.

On some old BSD systems, `touch` or any command that results in an empty file does not update the timestamps, so use a command like `echo` as a workaround.

GNU `touch` 3.16r (and presumably all before that) fails to work on SunOS 4.1.3 when the empty file is on an NFS-mounted 4.2 volume.

10.11 Limitations of Make

`make` itself suffers a great number of limitations, only a few of which are listed here. First of all, remember that since commands are executed by the shell, all its weaknesses are inherited. . . .

`$<` POSIX says that the '`$<`' construct in makefiles can be used only in inference rules and in the '`.DEFAULT`' rule; its meaning in ordinary rules is unspecified. Solaris 8's `make` for instance will replace it with the argument.

Leading underscore in macro names

Some `make`s don't support leading underscores in macro names, such as on NEWS-OS 4.2R.

```
$ cat Makefile
_am_include = #
_am_quote =
all:; @echo this is test
$ make
Make: Must be a separator on rules line 2.  Stop.
$ cat Makefile2
am_include = #
am_quote =
all:; @echo this is test
$ make -f Makefile2
this is test
```

Trailing backslash in macro

On some versions of HP-UX, `make` will read multiple newlines following a backslash, continuing to the next non-empty line. For example,

```
FOO = one \

BAR = two

test:
        : FOO is "$(FOO)"
        : BAR is "$(BAR)"
```

shows `FOO` equal to `one BAR = two`. Other `make`s sensibly let a backslash continue only to the immediately following line.

Escaped newline in comments

According to POSIX, '`Makefile`' comments start with `#` and continue until an unescaped newline is reached.

```
% cat Makefile
# A = foo \
      bar \
      baz

all:
        @echo ok
% make   # GNU make
ok
```

However in Real World this is not always the case. Some implementations discards anything from `#` up to the end of line, ignoring any trailing backslash.

```
% pmake  # BSD make
"Makefile", line 3: Need an operator
Fatal errors encountered -- cannot continue
```

Therefore, if you want to comment out a multi-line definition, prefix each line with `#`, not only the first.

```
# A = foo \
#     bar \
#     baz
```

`make macro=value` and sub-`make`s.

A command-line variable definition such as `foo=bar` overrides any definition of `foo` in the '`Makefile`'. Some `make` implementations (such as GNU `make`) will propagate this override to sub-invocations of `make`. Some other implementation will not pass the substitution along to sub-`make`s.

```
% cat Makefile
foo = foo
one:
        @echo $(foo)
        $(MAKE) two
```

```
two:
        @echo $(foo)
% make foo=bar            # GNU make 3.79.1
bar
make two
make[1]: Entering directory '/home/adl'
bar
make[1]: Leaving directory '/home/adl'
% pmake foo=bar           # BSD make
bar
pmake two
foo
```

You have a few possibilities if you do want the `foo=bar` override to propagate to sub-`make`s. One is to use the `-e` option, which causes all environment variables to have precedence over the '`Makefile`' macro definitions, and declare foo as an environment variable:

```
% env foo=bar make -e
```

The `-e` option is propagated to sub-`make`s automatically, and since the environment is inherited between `make` invocations, the `foo` macro will be overridden in sub-`make`s as expected.

This syntax (`foo=bar make -e`) is portable only when used outside a '`Makefile`', for instance from a script or from the command line. When run inside a `make` rule, GNU `make` 3.80 and prior versions forget to propagate the `-e` option to sub-`make`s.

Moreover, using `-e` could have unexpected side-effects if your environment contains some other macros usually defined by the Makefile. (See also the note about `make -e` and `SHELL` below.)

Another way to propagate overrides to sub-`make`s is to do it manually, from your '`Makefile`':

```
foo = foo
one:
        @echo $(foo)
        $(MAKE) foo=$(foo) two
two:
        @echo $(foo)
```

You need to foresee all macros that a user might want to override if you do that.

The `SHELL` macro

POSIX-compliant `make`s internally use the `$(SHELL)` macro to spawn shell processes and execute '`Makefile`' rules. This is a builtin macro supplied by `make`, but it can be modified from the '`Makefile`' or a command-line argument.

Not all `make`s will define this `SHELL` macro. OSF/Tru64 `make` is an example; this implementation will always use `/bin/sh`. So it's a good idea to always define `SHELL` in your '`Makefile`'s. If you use Autoconf, do

```
SHELL = @SHELL@
```

POSIX-compliant makes should never acquire the value of $(SHELL) from the environment, even when `make -e` is used (otherwise, think about what would happen to your rules if `SHELL=/bin/tcsh`).

However not all `make` implementations will make this exception. For instance it's not surprising that OSF/Tru64 `make` doesn't protect `SHELL`, since it doesn't use it.

```
% cat Makefile
SHELL = /bin/sh
FOO = foo
all:
        @echo $(SHELL)
        @echo $(FOO)
% env SHELL=/bin/tcsh FOO=bar make -e   # OSF1 V4.0 Make
/bin/tcsh
bar
% env SHELL=/bin/tcsh FOO=bar gmake -e  # GNU make
/bin/sh
bar
```

Comments in rules

Never put comments in a rule.

Some `make` treat anything starting with a tab as a command for the current rule, even if the tab is immediately followed by a `#`. The `make` from Tru64 Unix V5.1 is one of them. The following '`Makefile`' will run `# foo` through the shell.

```
all:
        # foo
```

The '`obj/`' subdirectory.

Never name one of your subdirectories '`obj/`' if you don't like surprises.

If an '`obj/`' directory exists, BSD `make` will enter it before reading '`Makefile`'. Hence the '`Makefile`' in the current directory will not be read.

```
% cat Makefile
all:
        echo Hello
% cat obj/Makefile
all:
        echo World
% make      # GNU make
echo Hello
Hello
% pmake     # BSD make
echo World
World
```

`make -k`

Do not rely on the exit status of `make -k`. Some implementations reflect whether they encountered an error in their exit status; other implementations always succeed.

```
% cat Makefile
all:
        false
% make -k; echo exit status: $?    # GNU make
false
make: *** [all] Error 1
exit status: 2
% pmake -k; echo exit status: $?   # BSD make
false
*** Error code 1 (continuing)
exit status: 0
```

`VPATH`

There is no `VPATH` support specified in POSIX. Many `make`s have a form of `VPATH` support, but its implementation is not consistent amongst `make`s.

Maybe the best suggestion to give to people who need the `VPATH` feature is to choose a `make` implementation and stick to it. Since the resulting '`Makefile`'s are not portable anyway, better choose a portable `make` (hint, hint).

Here are a couple of known issues with some `VPATH` implementations.

`VPATH` and double-colon rules

Any assignment to `VPATH` causes Sun `make` to only execute the first set of double-colon rules. (This comment has been here since 1994 and the context has been lost. It's probably about SunOS 4. If you can reproduce this, please send us a test case for illustration.)

`$<` not supported in explicit rules

As said elsewhere, using `$<` in explicit rules is not portable. The prerequisite file must be named explicitly in the rule. If you want to find the prerequisite via a `VPATH` search, you have to code the whole thing manually. For instance, using the following pattern:

```
VPATH = ../src
foo.o: foo.c
        cc -c `test -f foo.c || echo ../src/`foo.c -o foo.o
```

Automatic rule rewriting

Some `make` implementations, such as SunOS `make`, will search prerequisites in `VPATH` and rewrite all their occurrences in the rule appropriately.

For instance

```
VPATH = ../src
foo.o: foo.c
        cc -c foo.c -o foo.o
```

would execute `cc -c ../src/foo.c -o foo.o` if '`foo.c`' was found in '`../src`'. That sounds great.

However, for the sake of other `make` implementations, we can't rely on this, and we have to search `VPATH` manually:

```
VPATH = ../src
foo.o: foo.c
        cc -c `test -f foo.c || echo ../src/`foo.c -o foo.o
```

However the "prerequisite rewriting" still applies here. So if '`foo.c`' is in '`../src`', SunOS `make` will execute

```
cc -c `test -f ../src/foo.c || echo ../src/`foo.c -
o foo.o
```

which reduces to

```
cc -c foo.c -o foo.o
```

and thus fails. Oops.

One workaround is to make sure that foo.c never appears as a plain word in the rule. For instance these three rules would be safe.

```
VPATH = ../src
foo.o: foo.c
        cc -c `test -f ./foo.c || echo ../src/`foo.c -
o foo.o
foo2.o: foo2.c
        cc -c `test -f 'foo2.c' || echo ../src/`foo2.c -
o foo2.o
foo3.o: foo3.c
        cc -c `test -f "foo3.c" || echo ../src/`foo3.c -
o foo3.o
```

Things get worse when your prerequisites are in a macro.

```
VPATH = ../src
HEADERS = foo.h foo2.h foo3.h
install-HEADERS: $(HEADERS)
        for i in $(HEADERS); do \
          $(INSTALL) -m 644 `test -f $$i || echo ../src/`$$i \
            $(DESTDIR)$(includedir)/$$i; \
        done
```

The above `install-HEADERS` rule is not SunOS-proof because `for i in $(HEADERS);` will be expanded as `for i in foo.h foo2.h foo3.h;` where `foo.h` and `foo2.h` are plain words and are hence subject to `VPATH` adjustments.

If the three files are in '`../src`', the rule is run as:

```
for i in ../src/foo.h ../src/foo2.h foo3.h; do \
  install -m 644 `test -f $i || echo ../src/`$i \
     /usr/local/include/$i; \
done
```

where the two first `install` calls will fail. For instance, consider the `foo.h` installation:

```
install -m 644 `test -f ../src/foo.h || echo ../src/`../src/foo.h
  /usr/local/include/../src/foo.h;
```

It reduces to:

```
install -m 644 ../src/foo.h /usr/local/include/../src/foo.h;
```

Note that the manual `VPATH` search did not cause any problems here; however this command installs '`foo.h`' in an incorrect directory.

Trying to quote `$(HEADERS)` in some way, as we did for `foo.c` a few '`Makefile`'s ago, does not help:

```
install-HEADERS: $(HEADERS)
        headers='$(HEADERS)'; for i in $$headers; do \
          $(INSTALL) -m 644 `test -f $$i || echo ../src/`$$i \
            $(DESTDIR)$(includedir)/$$i; \
        done
```

Indeed, `headers='$(HEADERS)'` expands to `headers='foo.h foo2.h foo3.h'` where `foo2.h` is still a plain word. (Aside: the `headers='$(HEADERS)'; for i in $$headers;` idiom is a good idea if `$(HEADERS)` can be empty, because some shell produce a syntax error on `for i in;`.)

One workaround is to strip this unwanted '`../src/`' prefix manually:

```
VPATH = ../src
HEADERS = foo.h foo2.h foo3.h
install-HEADERS: $(HEADERS)
        headers='$(HEADERS)'; for i in $$headers; do \
          i=`expr "$$i" : '../src/\(.*\)'`;
          $(INSTALL) -m 644 `test -f $$i || echo ../src/`$$i \
            $(DESTDIR)$(includedir)/$$i; \
        done
```

Automake does something similar.

OSF/Tru64 `make` creates prerequisite directories magically

When a prerequisite is a sub-directory of `VPATH`, Tru64 `make` will create it in the current directory.

```
% mkdir -p foo/bar build
% cd build
% cat >Makefile <<END
VPATH = ..
all: foo/bar
END
% make
mkdir foo
mkdir foo/bar
```

This can yield unexpected results if a rule uses a manual `VPATH` search as presented before.

```
VPATH = ..
all : foo/bar
        command `test -d foo/bar || echo ../`foo/bar
```

The above `command` will be run on the empty '`foo/bar`' directory that was created in the current directory.

target lookup

GNU `make` uses a rather complex algorithm to decide when it should use files found via a `VPATH` search. See section "How Directory Searches are Performed" in *The* GNU *Make Manual*.

If a target needs to be rebuilt, GNU `make` discards the filename found during the `VPATH` search for this target, and builds the file locally using the filename given in the '`Makefile`'. If a target does not need to be rebuilt, GNU `make` uses the filename found during the `VPATH` search.

Other `make` implementations, like NetBSD `make`, are easier to describe: the filename found during the `VPATH` search will be used whether the target needs to be rebuilt or not. Therefore new files are created locally, but existing files are updated at their `VPATH` location.

OpenBSD and FreeBSD `make`s, however, will never perform a `VPATH` search for a dependency which has an explicit rule. This is extremely annoying.

When attempting a `VPATH` build for an autoconfiscated package (e.g,, `mkdir build && cd build && ../configure`), this means the GNU `make` will build everything locally in the '`build`' directory, while BSD `make` will build new files locally and update existing files in the source directory.

```
% cat Makefile
VPATH = ..
all: foo.x bar.x
foo.x bar.x: newer.x
        @echo Building $@
% touch ../bar.x
% touch ../newer.x
% make        # GNU make
Building foo.x
Building bar.x
% pmake       # NetBSD make
Building foo.x
Building ../bar.x
% fmake       # FreeBSD make, OpenBSD make
Building foo.x
Building bar.x
% tmake       # Tru64 make
Building foo.x
```

```
Building bar.x
% touch ../bar.x
% make        # GNU make
Building foo.x
% pmake       # NetBSD make
Building foo.x
% fmake       # FreeBSD make, OpenBSD make
Building foo.x
Building bar.x
% tmake       # Tru64 make
Building foo.x
Building bar.x
```

Note how NetBSD `make` updates '`../bar.x`' in its VPATH location, and how FreeBSD, OpenBSD, and Tru64 `make` always update '`bar.x`', even when '`../bar.x`' is up to date.

Another point worth mentioning is that once GNU `make` has decided to ignore a `VPATH` filename (e.g., it ignored '`../bar.x`' in the above example) it will continue to ignore it when the target occurs as a prerequisite of another rule.

The following example shows that GNU `make` does not look up '`bar.x`' in `VPATH` before performing the `.x.y` rule, because it ignored the `VPATH` result of '`bar.x`' while running the `bar.x: newer.x` rule.

```
% cat Makefile
VPATH = ..
all: bar.y
bar.x: newer.x
        @echo Building $@
.SUFFIXES: .x .y
.x.y:
        cp $< $@
% touch ../bar.x
% touch ../newer.x
% make        # GNU make
Building bar.x
cp bar.x bar.y
cp: cannot stat 'bar.x': No such file or directory
make: *** [bar.y] Error 1
% pmake       # NetBSD make
Building ../bar.x
cp ../bar.x bar.y
% rm bar.y
% fmake       # FreeBSD make, OpenBSD make
echo Building bar.x
cp bar.x bar.y
cp: cannot stat 'bar.x': No such file or directory
*** Error code 1
```

```
% tmake       # Tru64 make
Building bar.x
cp: bar.x: No such file or directory
*** Exit 1
```

Note that if you drop away the command from the `bar.x: newer.x` rule, GNU `make` will magically start to work: it knows that `bar.x` hasn't been updated, therefore it doesn't discard the result from `VPATH` ('`../bar.x`') in succeeding uses. Tru64 will also work, but FreeBSD and OpenBSD still don't.

```
% cat Makefile
VPATH = ..
all: bar.y
bar.x: newer.x
.SUFFIXES: .x .y
.x.y:
        cp $< $@
% touch ../bar.x
% touch ../newer.x
% make        # GNU make
cp ../bar.x bar.y
% rm bar.y
% pmake       # NetBSD make
cp ../bar.x bar.y
% rm bar.y
% fmake       # FreeBSD make, OpenBSD make
cp bar.x bar.y
cp: cannot stat 'bar.x': No such file or directory
*** Error code 1
% tmake       # True64 make
cp ../bar.x bar.y
```

It seems the sole solution that would please every `make` implementation is to never rely on `VPATH` searches for targets. In other words, `VPATH` should be reserved to unbuilt sources.

Single Suffix Rules and Separated Dependencies

A *Single Suffix Rule* is basically a usual suffix (inference) rule ('`.from.to:`'), but which *destination* suffix is empty ('`.from:`').

Separated dependencies simply refers to listing the prerequisite of a target, without defining a rule. Usually one can list on the one hand side, the rules, and on the other hand side, the dependencies.

Solaris `make` does not support separated dependencies for targets defined by single suffix rules:

```
$ cat Makefile
.SUFFIXES: .in
foo: foo.in
.in:
```

```
        cp $< $ $ touch foo.in
$ make
$ ls
Makefile  foo.in
```

while GNU Make does:

```
$ gmake
cp foo.in foo
$ ls
Makefile  foo       foo.in
```

Note it works without the '`foo: foo.in`' dependency.

```
$ cat Makefile
.SUFFIXES: .in
.in:
        cp $< $ $ make foo
cp foo.in foo
```

and it works with double suffix inference rules:

```
$ cat Makefile
foo.out: foo.in
.SUFFIXES: .in .out
.in.out:
        cp $< $ $ make
cp foo.in foo.out
```

As a result, in such a case, you have to write target rules.

Timestamp Resolution

Traditionally, file timestamps had 1-second resolution, and `make` used those timestamps to determine whether one file was newer than the other. However, many modern filesystems have timestamps with 1-nanosecond resolution. Some `make` implementations look at the entire timestamp; others ignore the fractional part, which can lead to incorrect results. Normally this is not a problem, but in some extreme cases you may need to use tricks like '`sleep 1`' to work around timestamp truncation bugs.

Commands like '`cp -p`' and '`touch -r`' typically do not copy file timestamps to their full resolutions (see Section 10.10 [Limitations of Usual Tools], page 131). Hence you should be wary of rules like this:

```
dest: src
        cp -p src dest
```

as '`dest`' will often appear to be older than '`src`' after the timestamp is truncated, and this can cause `make` to do needless rework the next time it is invoked. To work around this problem, you can use a timestamp file, e.g.:

```
dest-stamp: src
        cp -p src dest
        date >dest-stamp
```

11 Manual Configuration

A few kinds of features can't be guessed automatically by running test programs. For example, the details of the object-file format, or special options that need to be passed to the compiler or linker. You can check for such features using ad-hoc means, such as having `configure` check the output of the `uname` program, or looking for libraries that are unique to particular systems. However, Autoconf provides a uniform method for handling unguessable features.

11.1 Specifying the System Type

Like other GNU `configure` scripts, Autoconf-generated `configure` scripts can make decisions based on a canonical name for the system type, which has the form: '*cpu-vendor-os*', where *os* can be '*system*' or '*kernel-system*'

`configure` can usually guess the canonical name for the type of system it's running on. To do so it runs a script called `config.guess`, which infers the name using the `uname` command or symbols predefined by the C preprocessor.

Alternately, the user can specify the system type with command line arguments to `configure`. Doing so is necessary when cross-compiling. In the most complex case of cross-compiling, three system types are involved. The options to specify them are:

'`--build=`*build-type*'
: the type of system on which the package is being configured and compiled. It defaults to the result of running `config.guess`.

'`--host=`*host-type*'
: the type of system on which the package will run. By default it is the same as the build machine. Specifying it enables the cross-compilation mode.

'`--target=`*target-type*'
: the type of system for which any compiler tools in the package will produce code (rarely needed). By default, it is the same as host.

If you mean to override the result of `config.guess`, use '`--build`', not '`--host`', since the latter enables cross-compilation. For historical reasons, passing '`--host`' also changes the build type. Therefore, whenever you specify `--host`, be sure to specify `--build` too. This will be fixed in the future.

```
./configure --build=i686-pc-linux-gnu --host=m68k-coff
```

will enter cross-compilation mode, but `configure` will fail if it can't run the code generated by the specified compiler if you configure as follows:

```
./configure CC=m68k-coff-gcc
```

`configure` recognizes short aliases for many system types; for example, '`decstation`' can be used instead of '`mips-dec-ultrix4.2`'. `configure` runs a script called `config.sub` to canonicalize system type aliases.

This section deliberately omits the description of the obsolete interface; see Section 15.6.3 [Hosts and Cross-Compilation], page 182.

11.2 Getting the Canonical System Type

The following macros make the system type available to `configure` scripts.

The variables '`build_alias`', '`host_alias`', and '`target_alias`' are always exactly the arguments of '`--build`', '`--host`', and '`--target`'; in particular, they are left empty if the user did not use them, even if the corresponding `AC_CANONICAL` macro was run. Any configure script may use these variables anywhere. These are the variables that should be used when in interaction with the user.

If you need to recognize some special environments based on their system type, run the following macros to get canonical system names. These variables are not set before the macro call.

If you use these macros, you must distribute `config.guess` and `config.sub` along with your source code. See Section 4.4 [Output], page 17, for information about the `AC_CONFIG_AUX_DIR` macro which you can use to control in which directory `configure` looks for those scripts.

AC_CANONICAL_BUILD [Macro]

> Compute the canonical build-system type variable, `build`, and its three individual parts `build_cpu`, `build_vendor`, and `build_os`.
>
> If '`--build`' was specified, then `build` is the canonicalization of `build_alias` by `config.sub`, otherwise it is determined by the shell script `config.guess`.

AC_CANONICAL_HOST [Macro]

> Compute the canonical host-system type variable, `host`, and its three individual parts `host_cpu`, `host_vendor`, and `host_os`.
>
> If '`--host`' was specified, then `host` is the canonicalization of `host_alias` by `config.sub`, otherwise it defaults to `build`.

AC_CANONICAL_TARGET [Macro]

> Compute the canonical target-system type variable, `target`, and its three individual parts `target_cpu`, `target_vendor`, and `target_os`.
>
> If '`--target`' was specified, then `target` is the canonicalization of `target_alias` by `config.sub`, otherwise it defaults to `host`.

Note that there can be artifacts due to the backward compatibility code. See See Section 15.6.3 [Hosts and Cross-Compilation], page 182, for more.

11.3 Using the System Type

How do you use a canonical system type? Usually, you use it in one or more `case` statements in '`configure.ac`' to select system-specific C files. Then, using `AC_CONFIG_LINKS`, link those files which have names based on the system name, to generic names, such as '`host.h`' or '`target.c`' (see Section 4.10 [Configuration Links], page 30). The `case` statement patterns can use shell wild cards to group several cases together, like in this fragment:

```
case $target in
i386-*-mach* | i386-*-gnu*)
             obj_format=aout emulation=mach bfd_gas=yes ;;
i960-*-bout) obj_format=bout ;;
```

```
esac
```

and later in ‘configure.ac’, use:

```
AC_CONFIG_LINKS(host.h:config/$machine.h
                object.h:config/$obj_format.h)
```

Note that the above example uses $target because it’s taken from a tool which can be built on some architecture ($build), run on another ($host), but yet handle data for a third architecture ($target). Such tools are usually part of a compiler suite, they generate code for a specific $target.

However $target should be meaningless for most packages. If you want to base a decision on the system where your program will be run, make sure you use the $host variable, as in the following excerpt:

```
case $host in
  *-*-msdos* | *-*-go32* | *-*-mingw32* | *-*-cygwin* | *-*-windows*)
    MUMBLE_INIT="mumble.ini"
    ;;
  *)
    MUMBLE_INIT=".mumbleinit"
    ;;
esac
AC_SUBST([MUMBLE_INIT])
```

You can also use the host system type to find cross-compilation tools. See Section 5.2.2 [Generic Programs], page 37, for information about the AC_CHECK_TOOL macro which does that.

12 Site Configuration

`configure` scripts support several kinds of local configuration decisions. There are ways for users to specify where external software packages are, include or exclude optional features, install programs under modified names, and set default values for `configure` options.

12.1 Working With External Software

Some packages require, or can optionally use, other software packages that are already installed. The user can give `configure` command line options to specify which such external software to use. The options have one of these forms:

```
--with-package[=arg]
--without-package
```

For example, '`--with-gnu-ld`' means work with the GNU linker instead of some other linker. '`--with-x`' means work with The X Window System.

The user can give an argument by following the package name with '`=`' and the argument. Giving an argument of '`no`' is for packages that are used by default; it says to *not* use the package. An argument that is neither '`yes`' nor '`no`' could include a name or number of a version of the other package, to specify more precisely which other package this program is supposed to work with. If no argument is given, it defaults to '`yes`'. '`--without-`*package*' is equivalent to '`--with-`*package*`=no`'.

`configure` scripts do not complain about '`--with-`*package*' options that they do not support. This behavior permits configuring a source tree containing multiple packages with a top-level `configure` script when the packages support different options, without spurious error messages about options that some of the packages support. An unfortunate side effect is that option spelling errors are not diagnosed. No better approach to this problem has been suggested so far.

For each external software package that may be used, '`configure.ac`' should call `AC_ARG_WITH` to detect whether the `configure` user asked to use it. Whether each package is used or not by default, and which arguments are valid, is up to you.

AC_ARG_WITH (*package, help-string,* [*action-if-given*], [*action-if-not-given*]) [Macro]

If the user gave `configure` the option '`--with-`*package*' or '`--without-`*package*', run shell commands *action-if-given*. If neither option was given, run shell commands *action-if-not-given*. The name *package* indicates another software package that this program should work with. It should consist only of alphanumeric characters and dashes.

The option's argument is available to the shell commands *action-if-given* in the shell variable `withval`, which is actually just the value of the shell variable `with_`*package*, with any '`-`' characters changed into '`_`'. You may use that variable instead, if you wish.

The argument *help-string* is a description of the option that looks like this:

```
  --with-readline         support fancy command line editing
```

help-string may be more than one line long, if more detail is needed. Just make sure the columns line up in '`configure --help`'. Avoid tabs in the help string. You'll need to enclose the help string in '`[`' and '`]`' in order to produce the leading spaces.

You should format your *help-string* with the macro `AS_HELP_STRING` (see Section 12.3 [Pretty Help Strings], page 155).

AC_WITH (*package, action-if-given,* [*action-if-not-given*]) [Macro]

This is an obsolete version of `AC_ARG_WITH` that does not support providing a help string.

12.2 Choosing Package Options

If a software package has optional compile-time features, the user can give `configure` command line options to specify whether to compile them. The options have one of these forms:

`--enable-`*`feature`*`[=`*`arg`*`]`
`--disable-`*`feature`*

These options allow users to choose which optional features to build and install. '`--enable-`*`feature`*' options should never make a feature behave differently or cause one feature to replace another. They should only cause parts of the program to be built rather than left out.

The user can give an argument by following the feature name with '`=`' and the argument. Giving an argument of '`no`' requests that the feature *not* be made available. A feature with an argument looks like '`--enable-debug=stabs`'. If no argument is given, it defaults to '`yes`'. '`--disable-`*`feature`*' is equivalent to '`--enable-`*`feature`*`=no`'.

`configure` scripts do not complain about '`--enable-`*`feature`*' options that they do not support. This behavior permits configuring a source tree containing multiple packages with a top-level `configure` script when the packages support different options, without spurious error messages about options that some of the packages support. An unfortunate side effect is that option spelling errors are not diagnosed. No better approach to this problem has been suggested so far.

For each optional feature, '`configure.ac`' should call `AC_ARG_ENABLE` to detect whether the `configure` user asked to include it. Whether each feature is included or not by default, and which arguments are valid, is up to you.

AC_ARG_ENABLE (*feature, help-string,* [*action-if-given*], [*action-if-not-given*]) [Macro]

If the user gave `configure` the option '`--enable-`*`feature`*' or '`--disable-`*`feature`*', run shell commands *action-if-given*. If neither option was given, run shell commands *action-if-not-given*. The name *feature* indicates an optional user-level facility. It should consist only of alphanumeric characters and dashes.

The option's argument is available to the shell commands *action-if-given* in the shell variable `enableval`, which is actually just the value of the shell variable `enable_`*`feature`*, with any '`-`' characters changed into '`_`'. You may use that variable instead, if you wish. The *help-string* argument is like that of `AC_ARG_WITH` (see Section 12.1 [External Software], page 153).

You should format your *help-string* with the macro `AS_HELP_STRING` (see Section 12.3 [Pretty Help Strings], page 155).

AC_ENABLE (*feature, action-if-given,* [*action-if-not-given*]) [Macro]

This is an obsolete version of `AC_ARG_ENABLE` that does not support providing a help string.

12.3 Making Your Help Strings Look Pretty

Properly formatting the '`help strings`' which are used in `AC_ARG_WITH` (see Section 12.1 [External Software], page 153) and `AC_ARG_ENABLE` (see Section 12.2 [Package Options], page 154) can be challenging. Specifically, you want your own '`help strings`' to line up in the appropriate columns of '`configure --help`' just like the standard Autoconf '`help strings`' do. This is the purpose of the `AS_HELP_STRING` macro.

AS_HELP_STRING (*left-hand-side, right-hand-side*) [Macro]

Expands into an help string that looks pretty when the user executes '`configure --help`'. It is typically used in `AC_ARG_WITH` (see Section 12.1 [External Software], page 153) or `AC_ARG_ENABLE` (see Section 12.2 [Package Options], page 154). The following example will make this clearer.

```
AC_DEFUN([TEST_MACRO],
[AC_ARG_WITH([foo],
             AS_HELP_STRING([--with-foo],
                            [use foo (default is NO)]),
             [ac_cv_use_foo=$withval], [ac_cv_use_foo=no])
AC_CACHE_CHECK([whether to use foo],
               [ac_cv_use_foo], [ac_cv_use_foo=no])])
```

Please note that the call to `AS_HELP_STRING` is **unquoted**. Then the last few lines of '`configure --help`' will appear like this:

```
--enable and --with options recognized:
  --with-foo              use foo (default is NO)
```

The `AS_HELP_STRING` macro is particularly helpful when the *left-hand-side* and/or *right-hand-side* are composed of macro arguments, as shown in the following example.

```
AC_DEFUN(MY_ARG_WITH,
[AC_ARG_WITH([$1],
             AS_HELP_STRING([--with-$1], [use $1 (default is $2)]),
             ac_cv_use_$1=$withval, ac_cv_use_$1=no),
AC_CACHE_CHECK(whether to use $1, ac_cv_use_$1, ac_cv_use_$1=$2)])
```

12.4 Configuring Site Details

Some software packages require complex site-specific information. Some examples are host names to use for certain services, company names, and email addresses to contact. Since some configuration scripts generated by Metaconfig ask for such information interactively, people sometimes wonder how to get that information in Autoconf-generated configuration scripts, which aren't interactive.

Such site configuration information should be put in a file that is edited *only by users*, not by programs. The location of the file can either be based on the `prefix` variable, or be a standard location such as the user's home directory. It could even be specified by an environment variable. The programs should examine that file at run time, rather than at compile time. Run-time configuration is more convenient for users and makes the configuration process simpler than getting the information while configuring. See section "Variables for Installation Directories" in GNU *Coding Standards*, for more information on where to put data files.

12.5 Transforming Program Names When Installing

Autoconf supports changing the names of programs when installing them. In order to use these transformations, 'configure.ac' must call the macro AC_ARG_PROGRAM.

AC_ARG_PROGRAM [Macro]

Place in output variable program_transform_name a sequence of sed commands for changing the names of installed programs.

If any of the options described below are given to configure, program names are transformed accordingly. Otherwise, if AC_CANONICAL_TARGET has been called and a '--target' value is given, the target type followed by a dash is used as a prefix. Otherwise, no program name transformation is done.

12.5.1 Transformation Options

You can specify name transformations by giving configure these command line options:

'--program-prefix=*prefix*'
: prepend *prefix* to the names;

'--program-suffix=*suffix*'
: append *suffix* to the names;

'--program-transform-name=*expression*'
: perform sed substitution *expression* on the names.

12.5.2 Transformation Examples

These transformations are useful with programs that can be part of a cross-compilation development environment. For example, a cross-assembler running on a Sun 4 configured with '--target=i960-vxworks' is normally installed as 'i960-vxworks-as', rather than 'as', which could be confused with a native Sun 4 assembler.

You can force a program name to begin with 'g', if you don't want GNU programs installed on your system to shadow other programs with the same name. For example, if you configure GNU diff with '--program-prefix=g', then when you run 'make install' it is installed as '/usr/local/bin/gdiff'.

As a more sophisticated example, you could use

```
--program-transform-name='s/^/g/; s/^gg/g/; s/^gless/less/'
```

to prepend 'g' to most of the program names in a source tree, excepting those like gdb that already have one and those like less and lesskey that aren't GNU programs. (That is assuming that you have a source tree containing those programs that is set up to use this feature.)

One way to install multiple versions of some programs simultaneously is to append a version number to the name of one or both. For example, if you want to keep Autoconf version 1 around for awhile, you can configure Autoconf version 2 using '--program-suffix=2' to install the programs as '/usr/local/bin/autoconf2', '/usr/local/bin/autoheader2', etc. Nevertheless, pay attention that only the binaries are renamed, therefore you'd have problems with the library files which might overlap.

12.5.3 Transformation Rules

Here is how to use the variable `program_transform_name` in a 'Makefile.in':

```
PROGRAMS = cp ls rm
transform = @program_transform_name@
install:
        for p in $(PROGRAMS); do \
          $(INSTALL_PROGRAM) $$p $(DESTDIR)$(bindir)/`echo $$p | \
                                              sed '$(transform)'`; \
        done

uninstall:
        for p in $(PROGRAMS); do \
          rm -f $(DESTDIR)$(bindir)/`echo $$p | sed '$(transform)'`; \
        done
```

It is guaranteed that `program_transform_name` is never empty, and that there are no useless separators. Therefore you may safely embed `program_transform_name` within a sed program using ';':

```
transform = @program_transform_name@
transform_exe = s/$(EXEEXT)$$//;$(transform);s/$$/$(EXEEXT)/
```

Whether to do the transformations on documentation files (Texinfo or `man`) is a tricky question; there seems to be no perfect answer, due to the several reasons for name transforming. Documentation is not usually particular to a specific architecture, and Texinfo files do not conflict with system documentation. But they might conflict with earlier versions of the same files, and `man` pages sometimes do conflict with system documentation. As a compromise, it is probably best to do name transformations on `man` pages but not on Texinfo manuals.

12.6 Setting Site Defaults

Autoconf-generated `configure` scripts allow your site to provide default values for some configuration values. You do this by creating site- and system-wide initialization files.

If the environment variable `CONFIG_SITE` is set, `configure` uses its value as the name of a shell script to read. Otherwise, it reads the shell script '*prefix*/share/config.site' if it exists, then '*prefix*/etc/config.site' if it exists. Thus, settings in machine-specific files override those in machine-independent ones in case of conflict.

Site files can be arbitrary shell scripts, but only certain kinds of code are really appropriate to be in them. Because `configure` reads any cache file after it has read any site files, a site file can define a default cache file to be shared between all Autoconf-generated `configure` scripts run on that system (see Section 7.3.2 [Cache Files], page 85). If you set a default cache file in a site file, it is a good idea to also set the output variable `CC` in that site file, because the cache file is only valid for a particular compiler, but many systems have several available.

You can examine or override the value set by a command line option to `configure` in a site file; options set shell variables that have the same names as the options, with any dashes turned into underscores. The exceptions are that '`--without-`' and '`--disable-`' options are like giving the corresponding '`--with-`' or '`--enable-`' option and the value '`no`'. Thus, '`--cache-file=localcache`' sets the variable `cache_file` to the value '`localcache`'; '`--enable-warnings=no`' or '`--disable-warnings`' sets the variable `enable_warnings` to the value '`no`'; '`--prefix=/usr`' sets the variable `prefix` to the value '`/usr`'; etc.

Site files are also good places to set default values for other output variables, such as `CFLAGS`, if you need to give them non-default values: anything you would normally do, repetitively, on the command line. If you use non-default values for *prefix* or *exec_prefix* (wherever you locate the site file), you can set them in the site file if you specify it with the `CONFIG_SITE` environment variable.

You can set some cache values in the site file itself. Doing this is useful if you are cross-compiling, where it is impossible to check features that require running a test program. You could "prime the cache" by setting those values correctly for that system in '*prefix*`/etc/config.site`'. To find out the names of the cache variables you need to set, look for shell variables with '`_cv_`' in their names in the affected `configure` scripts, or in the Autoconf M4 source code for those macros.

The cache file is careful to not override any variables set in the site files. Similarly, you should not override command-line options in the site files. Your code should check that variables such as `prefix` and `cache_file` have their default values (as set near the top of `configure`) before changing them.

Here is a sample file '`/usr/share/local/gnu/share/config.site`'. The command '`configure --prefix=/usr/share/local/gnu`' would read this file (if `CONFIG_SITE` is not set to a different file).

```
# config.site for configure
#
# Change some defaults.
test "$prefix" = NONE && prefix=/usr/share/local/gnu
test "$exec_prefix" = NONE && exec_prefix=/usr/local/gnu
test "$sharedstatedir" = '$prefix/com' && sharedstatedir=/var
test "$localstatedir" = '$prefix/var' && localstatedir=/var

# Give Autoconf 2.x generated configure scripts a shared default
# cache file for feature test results, architecture-specific.
if test "$cache_file" = /dev/null; then
  cache_file="$prefix/var/config.cache"
  # A cache file is only valid for one C compiler.
  CC=gcc
fi
```

13 Running configure Scripts

Below are instructions on how to configure a package that uses a `configure` script, suitable for inclusion as an 'INSTALL' file in the package. A plain-text version of 'INSTALL' which you may use comes with Autoconf.

13.1 Basic Installation

These are generic installation instructions.

The `configure` shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a 'Makefile' in each directory of the package. It may also create one or more '.h' files containing system-dependent definitions. Finally, it creates a shell script 'config.status' that you can run in the future to recreate the current configuration, and a file 'config.log' containing compiler output (useful mainly for debugging `configure`).

It can also use an optional file (typically called 'config.cache' and enabled with '--cache-file=config.cache' or simply '-C') that saves the results of its tests to speed up reconfiguring. (Caching is disabled by default to prevent problems with accidental use of stale cache files.)

If you need to do unusual things to compile the package, please try to figure out how `configure` could check whether to do them, and mail diffs or instructions to the address given in the 'README' so they can be considered for the next release. If you are using the cache, and at some point 'config.cache' contains results you don't want to keep, you may remove or edit it.

The file 'configure.ac' (or 'configure.in') is used to create 'configure' by a program called `autoconf`. You only need 'configure.ac' if you want to change it or regenerate 'configure' using a newer version of `autoconf`.

The simplest way to compile this package is:

1. `cd` to the directory containing the package's source code and type './configure' to configure the package for your system. If you're using `csh` on an old version of System V, you might need to type 'sh ./configure' instead to prevent `csh` from trying to execute `configure` itself.

   Running `configure` takes awhile. While running, it prints some messages telling which features it is checking for.
2. Type 'make' to compile the package.
3. Optionally, type 'make check' to run any self-tests that come with the package.
4. Type 'make install' to install the programs and any data files and documentation.
5. You can remove the program binaries and object files from the source code directory by typing 'make clean'. To also remove the files that `configure` created (so you can compile the package for a different kind of computer), type 'make distclean'. There is also a 'make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

13.2 Compilers and Options

Some systems require unusual options for compilation or linking that the `configure` script does not know about. Run '`./configure --help`' for details on some of the pertinent environment variables.

You can give `configure` initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

```
./configure CC=c89 CFLAGS=-O2 LIBS=-lposix
```

See Section 13.8 [Defining Variables], page 161, for more details.

13.3 Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of `make` that supports the `VPATH` variable, such as GNU `make`. `cd` to the directory where you want the object files and executables to go and run the `configure` script. `configure` automatically checks for the source code in the directory that `configure` is in and in '`..`'.

If you have to use a `make` that does not support the `VPATH` variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use '`make distclean`' before reconfiguring for another architecture.

13.4 Installation Names

By default, '`make install`' will install the package's files in '`/usr/local/bin`', '`/usr/local/man`', etc. You can specify an installation prefix other than '`/usr/local`' by giving `configure` the option '`--prefix=`*path*'.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give `configure` the option '`--exec-prefix=`*path*', the package will use *path* as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '`--bindir=`*path*' to specify different values for particular kinds of files. Run '`configure --help`' for a list of the directories you can set and what kinds of files go in them.

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving `configure` the option '`--program-prefix=`*PREFIX*' or '`--program-suffix=`*SUFFIX*'.

13.5 Optional Features

Some packages pay attention to '`--enable-`*feature*' options to `configure`, where *feature* indicates an optional part of the package. They may also pay attention to '`--with-`*package*' options, where *package* is something like '`gnu-as`' or '`x`' (for the X Window System). The '`README`' should mention any '`--enable-`' and '`--with-`' options that the package recognizes.

For packages that use the X Window System, `configure` can usually find the X include and library files automatically, but if it doesn't, you can use the `configure` options '`--x-includes=dir`' and '`--x-libraries=dir`' to specify their locations.

13.6 Specifying the System Type

There may be some features `configure` cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the *same* architectures, `configure` can figure that out, but if it prints a message saying it cannot guess the machine type, give it the '`--build=type`' option. *type* can either be a short name for the system type, such as '`sun4`', or a canonical name which has the form:

```
cpu-company-system
```

where *system* can have one of these forms:

```
os kernel-os
```

See the file '`config.sub`' for the possible values of each field. If '`config.sub`' isn't included in this package, then this package doesn't need to know the machine type.

If you are *building* compiler tools for cross-compiling, you should use the '`--target=type`' option to select the type of system they will produce code for.

If you want to *use* a cross compiler, that generates code for a platform different from the build platform, you should specify the *host* platform (i.e., that on which the generated programs will eventually be run) with '`--host=type`'.

13.7 Sharing Defaults

If you want to set default values for `configure` scripts to share, you can create a site shell script called '`config.site`' that gives default values for variables like `CC`, `cache_file`, and `prefix`. `configure` looks for '`prefix/share/config.site`' if it exists, then '`prefix/etc/config.site`' if it exists. Or, you can set the `CONFIG_SITE` environment variable to the location of the site script. A warning: not all `configure` scripts look for a site script.

13.8 Defining Variables

Variables not defined in a site shell script can be set in the environment passed to `configure`. However, some packages may run configure again during the build, and the customized values of these variables may be lost. In order to avoid this problem, you should set them in the `configure` command line, using '`VAR=value`'. For example:

```
./configure CC=/usr/local2/bin/gcc
```

will cause the specified gcc to be used as the C compiler (unless it is overridden in the site shell script).

13.9 `configure` Invocation

`configure` recognizes the following options to control how it operates.

'`--help`'
'`-h`' Print a summary of the options to `configure`, and exit.

'--version'
'-V' Print the version of Autoconf used to generate the `configure` script, and exit.

'--cache-file=*file*'
Enable the cache: use and save the results of the tests in *file*, traditionally '`config.cache`'. *file* defaults to '`/dev/null`' to disable caching.

'--config-cache'
'-C' Alias for '`--cache-file=config.cache`'.

'--quiet'
'--silent'
'-q' Do not print messages saying which checks are being made. To suppress all normal output, redirect it to '`/dev/null`' (any error messages will still be shown).

'--srcdir=*dir*'
Look for the package's source code in directory *dir*. Usually `configure` can determine that directory automatically.

`configure` also accepts some other, not widely useful, options. Run '`configure --help`' for more details.

14 Recreating a Configuration

The `configure` script creates a file named '`config.status`', which actually configures, *instantiates*, the template files. It also records the configuration options that were specified when the package was last configured in case reconfiguring is needed.

Synopsis:

```
./config.status option... [file...]
```

It configures the *files*; if none are specified, all the templates are instantiated. The files must be specified without their dependencies, as in

```
./config.status foobar
```

not

```
./config.status foobar:foo.in:bar.in
```

The supported *options* are:

'`--help`'
'`-h`' Print a summary of the command line options, the list of the template files, and exit.

'`--version`'
'`-V`' Print the version number of Autoconf and exit.

'`--silent`'
'`--quiet`'
'`-q`' Do not print progress messages.

'`--debug`'
'`-d`' Don't remove the temporary files.

'`--file=file[:template]`'
: Require that *file* be instantiated as if '`AC_CONFIG_FILES(file:template)`' was used. Both *file* and *template* may be '`-`' in which case the standard output and/or standard input, respectively, is used. If a *template* filename is relative, it is first looked for in the build tree, and then in the source tree. See Section 4.5 [Configuration Actions], page 18, for more details.

  This option and the following ones provide one way for separately distributed packages to share the values computed by `configure`. Doing so can be useful if some of the packages need a superset of the features that one of them, perhaps a common library, does. These options allow a '`config.status`' file to create files other than the ones that its '`configure.ac`' specifies, so it can be used for a different package.

'`--header=file[:template]`'
: Same as '`--file`' above, but with '`AC_CONFIG_HEADERS`'.

'`--recheck`'
: Ask '`config.status`' to update itself and exit (no instantiation). This option is useful if you change `configure`, so that the results of some tests might be different from the previous run. The '`--recheck`' option re-runs `configure` with

the same arguments you used before, plus the '--no-create' option, which prevents configure from running 'config.status' and creating 'Makefile' and other files, and the '--no-recursion' option, which prevents configure from running other configure scripts in subdirectories. (This is so other 'Makefile' rules can run 'config.status' when it changes; see Section 4.7.4 [Automatic Remaking], page 25, for an example).

'config.status' checks several optional environment variables that can alter its behavior:

CONFIG_SHELL [Variable]

The shell with which to run configure for the '--recheck' option. It must be Bourne-compatible. The default is a shell that supports LINENO if available, and '/bin/sh' otherwise.

CONFIG_STATUS [Variable]

The file name to use for the shell script that records the configuration. The default is './config.status'. This variable is useful when one package uses parts of another and the configure scripts shouldn't be merged because they are maintained separately.

You can use './config.status' in your Makefiles. For example, in the dependencies given above (see Section 4.7.4 [Automatic Remaking], page 25), 'config.status' is run twice when 'configure.ac' has changed. If that bothers you, you can make each run only regenerate the files for that rule:

```
config.h: stamp-h
stamp-h: config.h.in config.status
        ./config.status config.h
        echo > stamp-h

Makefile: Makefile.in config.status
        ./config.status Makefile
```

The calling convention of 'config.status' has changed; see Section 15.1 [Obsolete config.status Use], page 165, for details.

15 Obsolete Constructs

Autoconf changes, and throughout the years some constructs have been obsoleted. Most of the changes involve the macros, but in some cases the tools themselves, or even some concepts, are now considered obsolete.

You may completely skip this chapter if you are new to Autoconf. Its intention is mainly to help maintainers updating their packages by understanding how to move to more modern constructs.

15.1 Obsolete 'config.status' Invocation

'config.status' now supports arguments to specify the files to instantiate; see Chapter 14 [config.status Invocation], page 163, for more details. Before, environment variables had to be used.

CONFIG_COMMANDS [Variable]

The tags of the commands to execute. The default is the arguments given to AC_OUTPUT and AC_CONFIG_COMMANDS in 'configure.ac'.

CONFIG_FILES [Variable]

The files in which to perform '@*variable*@' substitutions. The default is the arguments given to AC_OUTPUT and AC_CONFIG_FILES in 'configure.ac'.

CONFIG_HEADERS [Variable]

The files in which to substitute C #define statements. The default is the arguments given to AC_CONFIG_HEADERS; if that macro was not called, 'config.status' ignores this variable.

CONFIG_LINKS [Variable]

The symbolic links to establish. The default is the arguments given to AC_CONFIG_LINKS; if that macro was not called, 'config.status' ignores this variable.

In Chapter 14 [config.status Invocation], page 163, using this old interface, the example would be:

```
config.h: stamp-h
stamp-h: config.h.in config.status
        CONFIG_COMMANDS= CONFIG_LINKS= CONFIG_FILES= \
          CONFIG_HEADERS=config.h ./config.status
        echo > stamp-h

Makefile: Makefile.in config.status
        CONFIG_COMMANDS= CONFIG_LINKS= CONFIG_HEADERS= \
          CONFIG_FILES=Makefile ./config.status
```

(If 'configure.ac' does not call AC_CONFIG_HEADERS, there is no need to set CONFIG_HEADERS in the make rules. Equally for CONFIG_COMMANDS etc.)

15.2 'acconfig.h'

In order to produce 'config.h.in', autoheader needs to build or to find templates for each symbol. Modern releases of Autoconf use AH_VERBATIM and AH_TEMPLATE (see Section 4.8.3 [Autoheader Macros], page 29), but in older releases a file, 'acconfig.h', contained the list of needed templates. autoheader copied comments and #define and #undef statements from 'acconfig.h' in the current directory, if present. This file used to be mandatory if you AC_DEFINE any additional symbols.

Modern releases of Autoconf also provide AH_TOP and AH_BOTTOM if you need to prepend/append some information to 'config.h.in'. Ancient versions of Autoconf had a similar feature: if './acconfig.h' contains the string '@TOP@', autoheader copies the lines before the line containing '@TOP@' into the top of the file that it generates. Similarly, if './acconfig.h' contains the string '@BOTTOM@', autoheader copies the lines after that line to the end of the file it generates. Either or both of those strings may be omitted. An even older alternate way to produce the same effect in ancient versions of Autoconf is to create the files '*file*.top' (typically 'config.h.top') and/or '*file*.bot' in the current directory. If they exist, autoheader copies them to the beginning and end, respectively, of its output.

In former versions of Autoconf, the files used in preparing a software package for distribution were:

```
configure.ac --.   .------> autoconf* -----> configure
               +---+
[aclocal.m4] --+   `---.
[acsite.m4] ---'       |
                       +--> [autoheader*] -> [config.h.in]
[acconfig.h] ----.     |
                 +-----'
[config.h.top] --+
[config.h.bot] --'
```

Using only the AH_ macros, 'configure.ac' should be self-contained, and should not depend upon 'acconfig.h' etc.

15.3 Using autoupdate to Modernize 'configure.ac'

The autoupdate program updates a 'configure.ac' file that calls Autoconf macros by their old names to use the current macro names. In version 2 of Autoconf, most of the macros were renamed to use a more uniform and descriptive naming scheme. See Section 9.2 [Macro Names], page 103, for a description of the new scheme. Although the old names still work (see Section 15.4 [Obsolete Macros], page 167, for a list of the old macros and the corresponding new names), you can make your 'configure.ac' files more readable and make it easier to use the current Autoconf documentation if you update them to use the new macro names.

If given no arguments, autoupdate updates 'configure.ac', backing up the original version with the suffix '~' (or the value of the environment variable SIMPLE_BACKUP_SUFFIX, if that is set). If you give autoupdate an argument, it reads that file instead of 'configure.ac' and writes the updated file to the standard output.

autoupdate accepts the following options:

'--help'
'-h' Print a summary of the command line options and exit.

'--version'
'-V' Print the version number of Autoconf and exit.

'--verbose'
'-v' Report processing steps.

'--debug'
'-d' Don't remove the temporary files.

'--force'
'-f' Force the update even if the file has not changed. Disregard the cache.

'--include=*dir*'
'-I *dir*' Also look for input files in *dir*. Multiple invocations accumulate. Directories are browsed from last to first.

15.4 Obsolete Macros

Several macros are obsoleted in Autoconf, for various reasons (typically they failed to quote properly, couldn't be extended for more recent issues etc.). They are still supported, but deprecated: their use should be avoided.

During the jump from Autoconf version 1 to version 2, most of the macros were renamed to use a more uniform and descriptive naming scheme, but their signature did not change. See Section 9.2 [Macro Names], page 103, for a description of the new naming scheme. Below, if there is just the mapping from old names to new names for these macros, the reader is invited to refer to the definition of the new macro for the signature and the description.

AC_ALLOCA [Macro]

`AC_FUNC_ALLOCA`

AC_ARG_ARRAY [Macro]

removed because of limited usefulness

AC_C_CROSS [Macro]

This macro is obsolete; it does nothing.

AC_CANONICAL_SYSTEM [Macro]

Determine the system type and set output variables to the names of the canonical system types. See Section 11.2 [Canonicalizing], page 150, for details about the variables this macro sets.

The user is encouraged to use either `AC_CANONICAL_BUILD`, or `AC_CANONICAL_HOST`, or `AC_CANONICAL_TARGET`, depending on the needs. Using `AC_CANONICAL_TARGET` is enough to run the two other macros.

AC_CHAR_UNSIGNED [Macro]

`AC_C_CHAR_UNSIGNED`

AC_CHECK_TYPE (*type, default*) [Macro]

Autoconf, up to 2.13, used to provide this version of `AC_CHECK_TYPE`, deprecated because of its flaws. Firstly, although it is a member of the `CHECK` clan, singular subfamily, it does more than just checking. Secondly, missing types are not `typedef`'d, they are `#define`'d, which can lead to incompatible code in the case of pointer types.

This use of `AC_CHECK_TYPE` is obsolete and discouraged; see Section 5.9.2 [Generic Types], page 57, for the description of the current macro.

If the type *type* is not defined, define it to be the C (or C++) builtin type *default*, e.g., '`short`' or '`unsigned`'.

This macro is equivalent to:

```
AC_CHECK_TYPE([type],,
              [AC_DEFINE_UNQUOTED([type], [default],
                                  [Define to `default' if
                                   <sys/types.h> does not define.])])
```

In order to keep backward compatibility, the two versions of `AC_CHECK_TYPE` are implemented, selected by a simple heuristics:

1. If there are three or four arguments, the modern version is used.
2. If the second argument appears to be a C or C++ type, then the obsolete version is used. This happens if the argument is a C or C++ *builtin* type or a C identifier ending in '`_t`', optionally followed by one of '`[(* `' and then by a string of zero or more characters taken from the set '`[]()* _a-zA-Z0-9`'.
3. If the second argument is spelled with the alphabet of valid C and C++ types, the user is warned and the modern version is used.
4. Otherwise, the modern version is used.

You are encouraged either to use a valid builtin type, or to use the equivalent modern code (see above), or better yet, to use `AC_CHECK_TYPES` together with

```
#if !HAVE_LOFF_T
typedef loff_t off_t;
#endif
```

AC_CHECKING (*feature-description*) [Macro]

Same as '`AC_MSG_NOTICE([checking feature-description...]`'.

AC_COMPILE_CHECK (*echo-text, includes, function-body, action-if-found,* [*action-if-not-found*]) [Macro]

This is an obsolete version of `AC_TRY_COMPILE` itself replaced by `AC_COMPILE_IFELSE` (see Section 6.4 [Running the Compiler], page 77), with the addition that it prints '`checking for echo-text`' to the standard output first, if *echo-text* is non-empty. Use `AC_MSG_CHECKING` and `AC_MSG_RESULT` instead to print messages (see Section 7.4 [Printing Messages], page 86).

AC_CONST [Macro]

`AC_C_CONST`

AC_CROSS_CHECK [Macro]

Same as `AC_C_CROSS`, which is obsolete too, and does nothing `:-)`.

AC_CYGWIN [Macro]

Check for the Cygwin environment in which case the shell variable `CYGWIN` is set to '`yes`'. Don't use this macro, the dignified means to check the nature of the host is using `AC_CANONICAL_HOST`. As a matter of fact this macro is defined as:

```
AC_REQUIRE([AC_CANONICAL_HOST])[]dnl
case $host_os in
  *cygwin* ) CYGWIN=yes;;
         * ) CYGWIN=no;;
esac
```

Beware that the variable `CYGWIN` has a very special meaning when running CygWin32, and should not be changed. That's yet another reason not to use this macro.

AC_DECL_SYS_SIGLIST [Macro]

Same as:

```
AC_CHECK_DECLS([sys_siglist],,,
[#include <signal.h>
/* NetBSD declares sys_siglist in unistd.h.  */
#if HAVE_UNISTD_H
# include <unistd.h>
#endif
])
```

AC_DECL_YYTEXT [Macro]

Does nothing, now integrated in `AC_PROG_LEX`.

AC_DIR_HEADER [Macro]

Like calling `AC_FUNC_CLOSEDIR_VOID` and`AC_HEADER_DIRENT`, but defines a different set of C preprocessor macros to indicate which header file is found:

Header	Old Symbol	New Symbol
'`dirent.h`'	`DIRENT`	`HAVE_DIRENT_H`
'`sys/ndir.h`'	`SYSNDIR`	`HAVE_SYS_NDIR_H`
'`sys/dir.h`'	`SYSDIR`	`HAVE_SYS_DIR_H`
'`ndir.h`'	`NDIR`	`HAVE_NDIR_H`

AC_DYNIX_SEQ [Macro]

If on DYNIX/ptx, add '`-lseq`' to output variable `LIBS`. This macro used to be defined as

```
AC_CHECK_LIB(seq, getmntent, LIBS="-lseq $LIBS")
```

now it is just `AC_FUNC_GETMNTENT`.

AC_EXEEXT [Macro]

Defined the output variable `EXEEXT` based on the output of the compiler, which is now done automatically. Typically set to empty string if Unix and '`.exe`' if Win32 or OS/2.

AC_EMXOS2 [Macro]

Similar to `AC_CYGWIN` but checks for the EMX environment on OS/2 and sets `EMXOS2`.

AC_ERROR [Macro]
`AC_MSG_ERROR`

AC_FIND_X [Macro]
`AC_PATH_X`

AC_FIND_XTRA [Macro]
`AC_PATH_XTRA`

AC_FUNC_CHECK [Macro]
`AC_CHECK_FUNC`

AC_FUNC_WAIT3 [Macro]
If `wait3` is found and fills in the contents of its third argument (a '`struct rusage *`'), which HP-UX does not do, define `HAVE_WAIT3`.

These days portable programs should use `waitpid`, not `wait3`, as `wait3` is being removed from the Open Group standards, and will not appear in the next revision of POSIX.

AC_GCC_TRADITIONAL [Macro]
`AC_PROG_GCC_TRADITIONAL`

AC_GETGROUPS_T [Macro]
`AC_TYPE_GETGROUPS`

AC_GETLOADAVG [Macro]
`AC_FUNC_GETLOADAVG`

AC_HAVE_FUNCS [Macro]
`AC_CHECK_FUNCS`

AC_HAVE_HEADERS [Macro]
`AC_CHECK_HEADERS`

AC_HAVE_LIBRARY (*library*, [*action-if-found*], [*action-if-not-found*], [*other-libraries*]) [Macro]
This macro is equivalent to calling `AC_CHECK_LIB` with a *function* argument of `main`. In addition, *library* can be written as any of '`foo`', '`-lfoo`', or '`libfoo.a`'. In all of those cases, the compiler is passed '`-lfoo`'. However, *library* cannot be a shell variable; it must be a literal name.

AC_HAVE_POUNDBANG [Macro]
`AC_SYS_INTERPRETER` (different calling convention)

AC_HEADER_CHECK [Macro]
`AC_CHECK_HEADER`

AC_HEADER_EGREP [Macro]
`AC_EGREP_HEADER`

AC_HELP_STRING [Macro]
`AS_HELP_STRING`

AC_INIT (*unique-file-in-source-dir*) [Macro]

Formerly `AC_INIT` used to have a single argument, and was equivalent to:

```
AC_INIT
AC_CONFIG_SRCDIR(unique-file-in-source-dir)
```

AC_INLINE [Macro]

`AC_C_INLINE`

AC_INT_16_BITS [Macro]

If the C type `int` is 16 bits wide, define `INT_16_BITS`. Use '`AC_CHECK_SIZEOF(int)`' instead.

AC_IRIX_SUN [Macro]

If on IRIX (Silicon Graphics UNIX), add '`-lsun`' to output `LIBS`. If you were using it to get `getmntent`, use `AC_FUNC_GETMNTENT` instead. If you used it for the NIS versions of the password and group functions, use '`AC_CHECK_LIB(sun, getpwnam)`'. Up to Autoconf 2.13, it used to be

```
AC_CHECK_LIB(sun, getmntent, LIBS="-lsun $LIBS")
```

now it is defined as

```
AC_FUNC_GETMNTENT
AC_CHECK_LIB(sun, getpwnam)
```

AC_LANG_C [Macro]

Same as '`AC_LANG(C)`'.

AC_LANG_CPLUSPLUS [Macro]

Same as '`AC_LANG(C++)`'.

AC_LANG_FORTRAN77 [Macro]

Same as '`AC_LANG(Fortran 77)`'.

AC_LANG_RESTORE [Macro]

Select the *language* that is saved on the top of the stack, as set by `AC_LANG_SAVE`, remove it from the stack, and call `AC_LANG(language)`.

AC_LANG_SAVE [Macro]

Remember the current language (as set by `AC_LANG`) on a stack. The current language does not change. `AC_LANG_PUSH` is preferred.

AC_LINK_FILES (*source. . .*, *dest. . .*) [Macro]

This is an obsolete version of `AC_CONFIG_LINKS`. An updated version of:

```
AC_LINK_FILES(config/$machine.h config/$obj_format.h,
              host.h            object.h)
```

is:

```
AC_CONFIG_LINKS(host.h:config/$machine.h
                object.h:config/$obj_format.h)
```

AC_LN_S [Macro]

`AC_PROG_LN_S`

AC_LONG_64_BITS [Macro]

Define `LONG_64_BITS` if the C type `long int` is 64 bits wide. Use the generic macro '`AC_CHECK_SIZEOF([long int])`' instead.

AC_LONG_DOUBLE [Macro]

`AC_C_LONG_DOUBLE`

AC_LONG_FILE_NAMES [Macro]

`AC_SYS_LONG_FILE_NAMES`

AC_MAJOR_HEADER [Macro]

`AC_HEADER_MAJOR`

AC_MEMORY_H [Macro]

Used to define `NEED_MEMORY_H` if the `mem` functions were defined in '`memory.h`'. Today it is equivalent to '`AC_CHECK_HEADERS(memory.h)`'. Adjust your code to depend upon `HAVE_MEMORY_H`, not `NEED_MEMORY_H`; see Section 5.1.1 [Standard Symbols], page 33.

AC_MINGW32 [Macro]

Similar to `AC_CYGWIN` but checks for the MingW32 compiler environment and sets `MINGW32`.

AC_MINUS_C_MINUS_O [Macro]

`AC_PROG_CC_C_O`

AC_MMAP [Macro]

`AC_FUNC_MMAP`

AC_MODE_T [Macro]

`AC_TYPE_MODE_T`

AC_OBJEXT [Macro]

Defined the output variable `OBJEXT` based on the output of the compiler, after .c files have been excluded. Typically set to '`o`' if Unix, '`obj`' if Win32. Now the compiler checking macros handle this automatically.

AC_OBSOLETE (*this-macro-name*, [*suggestion*]) [Macro]

Make M4 print a message to the standard error output warning that *this-macro-name* is obsolete, and giving the file and line number where it was called. *this-macro-name* should be the name of the macro that is calling `AC_OBSOLETE`. If *suggestion* is given, it is printed at the end of the warning message; for example, it can be a suggestion for what to use instead of *this-macro-name*.

For instance

```
AC_OBSOLETE([$0], [; use AC_CHECK_HEADERS(unistd.h) instead])dnl
```

You are encouraged to use `AU_DEFUN` instead, since it gives better services to the user.

AC_OFF_T [Macro]

`AC_TYPE_OFF_T`

AC_OUTPUT ([*file*]..., [*extra-cmds*], [*init-cmds*]) [Macro]

The use of `AC_OUTPUT` with argument is deprecated. This obsoleted interface is equivalent to:

```
AC_CONFIG_FILES(file...)
AC_CONFIG_COMMANDS([default],
                   extra-cmds, init-cmds)
AC_OUTPUT
```

AC_OUTPUT_COMMANDS (*extra-cmds*, [*init-cmds*]) [Macro]

Specify additional shell commands to run at the end of '`config.status`', and shell commands to initialize any variables from `configure`. This macro may be called multiple times. It is obsolete, replaced by `AC_CONFIG_COMMANDS`.

Here is an unrealistic example:

```
fubar=27
AC_OUTPUT_COMMANDS([echo this is extra $fubar, and so on.],
                   [fubar=$fubar])
AC_OUTPUT_COMMANDS([echo this is another, extra, bit],
                   [echo init bit])
```

Aside from the fact that `AC_CONFIG_COMMANDS` requires an additional key, an important difference is that `AC_OUTPUT_COMMANDS` is quoting its arguments twice, unlike `AC_CONFIG_COMMANDS`. This means that `AC_CONFIG_COMMANDS` can safely be given macro calls as arguments:

```
AC_CONFIG_COMMANDS(foo, [my_FOO()])
```

Conversely, where one level of quoting was enough for literal strings with `AC_OUTPUT_COMMANDS`, you need two with `AC_CONFIG_COMMANDS`. The following lines are equivalent:

```
AC_OUTPUT_COMMANDS([echo "Square brackets: []"])
AC_CONFIG_COMMANDS([default], [[echo "Square brackets: []"]])
```

AC_PID_T [Macro]

`AC_TYPE_PID_T`

AC_PREFIX [Macro]

`AC_PREFIX_PROGRAM`

AC_PROG_CC_STDC [Macro]

This macro has been integrated into `AC_PROG_CC`.

AC_PROGRAMS_CHECK [Macro]

`AC_CHECK_PROGS`

AC_PROGRAMS_PATH [Macro]

`AC_PATH_PROGS`

AC_PROGRAM_CHECK [Macro]

`AC_CHECK_PROG`

AC_PROGRAM_EGREP [Macro]

`AC_EGREP_CPP`

AC_PROGRAM_PATH [Macro]
: `AC_PATH_PROG`

AC_REMOTE_TAPE [Macro]
: removed because of limited usefulness

AC_RESTARTABLE_SYSCALLS [Macro]
: `AC_SYS_RESTARTABLE_SYSCALLS`

AC_RETSIGTYPE [Macro]
: `AC_TYPE_SIGNAL`

AC_RSH [Macro]
: removed because of limited usefulness

AC_SCO_INTL [Macro]
: If on SCO UNIX, add '`-lintl`' to output variable `LIBS`. This macro used to

```
AC_CHECK_LIB(intl, strftime, LIBS="-lintl $LIBS")
```

Now it just calls `AC_FUNC_STRFTIME` instead.

AC_SETVBUF_REVERSED [Macro]
: `AC_FUNC_SETVBUF_REVERSED`

AC_SET_MAKE [Macro]
: `AC_PROG_MAKE_SET`

AC_SIZEOF_TYPE [Macro]
: `AC_CHECK_SIZEOF`

AC_SIZE_T [Macro]
: `AC_TYPE_SIZE_T`

AC_STAT_MACROS_BROKEN [Macro]
: `AC_HEADER_STAT`

AC_STDC_HEADERS [Macro]
: `AC_HEADER_STDC`

AC_STRCOLL [Macro]
: `AC_FUNC_STRCOLL`

AC_ST_BLKSIZE [Macro]
: `AC_CHECK_MEMBERS`

AC_ST_BLOCKS [Macro]
: `AC_STRUCT_ST_BLOCKS`

AC_ST_RDEV [Macro]
: `AC_CHECK_MEMBERS`

AC_SYS_RESTARTABLE_SYSCALLS [Macro]

If the system automatically restarts a system call that is interrupted by a signal, define `HAVE_RESTARTABLE_SYSCALLS`. This macro does not check if system calls are restarted in general–it tests whether a signal handler installed with `signal` (but not `sigaction`) causes system calls to be restarted. It does not test if system calls can be restarted when interrupted by signals that have no handler.

These days portable programs should use `sigaction` with `SA_RESTART` if they want restartable system calls. They should not rely on `HAVE_RESTARTABLE_SYSCALLS`, since nowadays whether a system call is restartable is a dynamic issue, not a configuration-time issue.

AC_SYS_SIGLIST_DECLARED [Macro]

`AC_DECL_SYS_SIGLIST`

AC_TEST_CPP [Macro]

`AC_TRY_CPP`, replaced by `AC_PREPROC_IFELSE`.

AC_TEST_PROGRAM [Macro]

`AC_TRY_RUN`, replaced by `AC_RUN_IFELSE`.

AC_TIMEZONE [Macro]

`AC_STRUCT_TIMEZONE`

AC_TIME_WITH_SYS_TIME [Macro]

`AC_HEADER_TIME`

AC_TRY_COMPILE (*includes, function-body,* [*action-if-found*], [*action-if-not-found*]) [Macro]

Same as '`AC_COMPILE_IFELSE([AC_LANG_SOURCE([[`*`includes`*`]], [[`*`function-body`*`]])], [`*`action-if-true`*`], [`*`action-if-false`*`])`' (see Section 6.4 [Running the Compiler], page 77).

This macro double quotes both *includes* and *function-body*.

For C and C++, *includes* is any `#include` statements needed by the code in *function-body* (*includes* will be ignored if the currently selected language is Fortran or Fortran 77). The compiler and compilation flags are determined by the current language (see Section 6.1 [Language Choice], page 71).

AC_TRY_CPP (*input,* [*action-if-true*], [*action-if-false*]) [Macro]

Same as '`AC_PREPROC_IFELSE([AC_LANG_SOURCE([[`*`input`*`]])], [`*`action-if-true`*`], [`*`action-if-false`*`])`' (see Section 6.3 [Running the Preprocessor], page 76).

This macro double quotes the *input*.

AC_TRY_LINK (*includes, function-body,* [*action-if-found*], [*action-if-not-found*]) [Macro]

Same as '`AC_LINK_IFELSE([AC_LANG_SOURCE([[`*`includes`*`]], [[`*`function-body`*`]])], [`*`action-if-true`*`], [`*`action-if-false`*`])`' (see Section 6.4 [Running the Compiler], page 77).

This macro double quotes both *includes* and *function-body*.

Depending on the current language (see Section 6.1 [Language Choice], page 71), create a test program to see whether a function whose body consists of *function-body* can be compiled and linked. If the file compiles and links successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*.

This macro double quotes both *includes* and *function-body*.

For C and C++, *includes* is any `#include` statements needed by the code in *function-body* (*includes* will be ignored if the currently selected language is Fortran or Fortran 77). The compiler and compilation flags are determined by the current language (see Section 6.1 [Language Choice], page 71), and in addition `LDFLAGS` and `LIBS` are used for linking.

AC_TRY_LINK_FUNC (*function, [action-if-found], [action-if-not-found]*) [Macro]

This macro is equivalent to '`AC_LINK_IFELSE([AC_LANG_CALL([[includes]], [[function-body]])], [action-if-true], [action-if-false])`'.

AC_TRY_RUN (*program, [action-if-true], [action-if-false], [action-if-cross-compiling]*) [Macro]

Same as '`AC_RUN_IFELSE([AC_LANG_SOURCE([[program]], [action-if-true], [action-if-false], [action-if-cross-compiling])`' (see Section 6.6 [Run Time], page 78).

AC_UID_T [Macro]

`AC_TYPE_UID_T`

AC_UNISTD_H [Macro]

Same as '`AC_CHECK_HEADERS(unistd.h)`'.

AC_USG [Macro]

Define `USG` if the BSD string functions are defined in '`strings.h`'. You should no longer depend upon `USG`, but on `HAVE_STRING_H`; see Section 5.1.1 [Standard Symbols], page 33.

AC_UTIME_NULL [Macro]

`AC_FUNC_UTIME_NULL`

AC_VALIDATE_CACHED_SYSTEM_TUPLE (*[cmd]*) [Macro]

If the cache file is inconsistent with the current host, target and build system types, it used to execute *cmd* or print a default error message. This is now handled by default.

AC_VERBOSE (*result-description*) [Macro]

`AC_MSG_RESULT`.

AC_VFORK [Macro]

`AC_FUNC_VFORK`

AC_VPRINTF [Macro]

`AC_FUNC_VPRINTF`

AC_WAIT3 [Macro]

`AC_FUNC_WAIT3`

AC_WARN [Macro]

`AC_MSG_WARN`

AC_WORDS_BIGENDIAN [Macro]

`AC_C_BIGENDIAN`

AC_XENIX_DIR [Macro]

This macro used to add '-lx' to output variable LIBS if on Xenix. Also, if 'dirent.h' is being checked for, added '-ldir' to LIBS. Now it is merely an alias of AC_HEADER_DIRENT instead, plus some code to detect whether running XENIX on which you should not depend:

```
AC_MSG_CHECKING([for Xenix])
AC_EGREP_CPP(yes,
[#if defined M_XENIX && !defined M_UNIX
  yes
#endif],
             [AC_MSG_RESULT([yes]); XENIX=yes],
             [AC_MSG_RESULT([no]); XENIX=])
```

AC_YYTEXT_POINTER [Macro]

`AC_DECL_YYTEXT`

15.5 Upgrading From Version 1

Autoconf version 2 is mostly backward compatible with version 1. However, it introduces better ways to do some things, and doesn't support some of the ugly things in version 1. So, depending on how sophisticated your 'configure.ac' files are, you might have to do some manual work in order to upgrade to version 2. This chapter points out some problems to watch for when upgrading. Also, perhaps your configure scripts could benefit from some of the new features in version 2; the changes are summarized in the file 'NEWS' in the Autoconf distribution.

15.5.1 Changed File Names

If you have an 'aclocal.m4' installed with Autoconf (as opposed to in a particular package's source directory), you must rename it to 'acsite.m4'. See Section 3.4 [autoconf Invocation], page 10.

If you distribute 'install.sh' with your package, rename it to 'install-sh' so make builtin rules won't inadvertently create a file called 'install' from it. AC_PROG_INSTALL looks for the script under both names, but it is best to use the new name.

If you were using 'config.h.top', 'config.h.bot', or 'acconfig.h', you still can, but you will have less clutter if you use the AH_ macros. See Section 4.8.3 [Autoheader Macros], page 29.

15.5.2 Changed Makefiles

Add '@CFLAGS@', '@CPPFLAGS@', and '@LDFLAGS@' in your 'Makefile.in' files, so they can take advantage of the values of those variables in the environment when configure is run. Doing this isn't necessary, but it's a convenience for users.

Also add '@configure_input@' in a comment to each input file for AC_OUTPUT, so that the output files will contain a comment saying they were produced by configure. Automatically selecting the right comment syntax for all the kinds of files that people call AC_OUTPUT on became too much work.

Add 'config.log' and 'config.cache' to the list of files you remove in distclean targets.

If you have the following in 'Makefile.in':

```
prefix = /usr/local
exec_prefix = $(prefix)
```

you must change it to:

```
prefix = @prefix@
exec_prefix = @exec_prefix@
```

The old behavior of replacing those variables without '@' characters around them has been removed.

15.5.3 Changed Macros

Many of the macros were renamed in Autoconf version 2. You can still use the old names, but the new ones are clearer, and it's easier to find the documentation for them. See Section 15.4 [Obsolete Macros], page 167, for a table showing the new names for the old macros. Use the autoupdate program to convert your 'configure.ac' to using the new macro names. See Section 15.3 [autoupdate Invocation], page 166.

Some macros have been superseded by similar ones that do the job better, but are not call-compatible. If you get warnings about calling obsolete macros while running autoconf, you may safely ignore them, but your configure script will generally work better if you follow the advice that is printed about what to replace the obsolete macros with. In particular, the mechanism for reporting the results of tests has changed. If you were using echo or AC_VERBOSE (perhaps via AC_COMPILE_CHECK), your configure script's output will look better if you switch to AC_MSG_CHECKING and AC_MSG_RESULT. See Section 7.4 [Printing Messages], page 86. Those macros work best in conjunction with cache variables. See Section 7.3 [Caching Results], page 83.

15.5.4 Changed Results

If you were checking the results of previous tests by examining the shell variable DEFS, you need to switch to checking the values of the cache variables for those tests. DEFS no longer exists while configure is running; it is only created when generating output files. This difference from version 1 is because properly quoting the contents of that variable turned out to be too cumbersome and inefficient to do every time AC_DEFINE is called. See Section 7.3.1 [Cache Variable Names], page 85.

For example, here is a 'configure.ac' fragment written for Autoconf version 1:

```
AC_HAVE_FUNCS(syslog)
case "$DEFS" in
*-DHAVE_SYSLOG*) ;;
*) # syslog is not in the default libraries.  See if it's in some other.
  saved_LIBS="$LIBS"
  for lib in bsd socket inet; do
```

```
        AC_CHECKING(for syslog in -l$lib)
        LIBS="$saved_LIBS -l$lib"
        AC_HAVE_FUNCS(syslog)
        case "$DEFS" in
        *-DHAVE_SYSLOG*) break ;;
        *) ;;
        esac
        LIBS="$saved_LIBS"
      done ;;
    esac
```

Here is a way to write it for version 2:

```
AC_CHECK_FUNCS(syslog)
if test $ac_cv_func_syslog = no; then
  # syslog is not in the default libraries.  See if it's in some other.
  for lib in bsd socket inet; do
    AC_CHECK_LIB($lib, syslog, [AC_DEFINE(HAVE_SYSLOG)
      LIBS="$LIBS -l$lib"; break])
  done
fi
```

If you were working around bugs in `AC_DEFINE_UNQUOTED` by adding backslashes before quotes, you need to remove them. It now works predictably, and does not treat quotes (except back quotes) specially. See Section 7.2 [Setting Output Variables], page 82.

All of the Boolean shell variables set by Autoconf macros now use '`yes`' for the true value. Most of them use '`no`' for false, though for backward compatibility some use the empty string instead. If you were relying on a shell variable being set to something like 1 or '`t`' for true, you need to change your tests.

15.5.5 Changed Macro Writing

When defining your own macros, you should now use `AC_DEFUN` instead of `define`. `AC_DEFUN` automatically calls `AC_PROVIDE` and ensures that macros called via `AC_REQUIRE` do not interrupt other macros, to prevent nested '`checking...`' messages on the screen. There's no actual harm in continuing to use the older way, but it's less convenient and attractive. See Section 9.1 [Macro Definitions], page 103.

You probably looked at the macros that came with Autoconf as a guide for how to do things. It would be a good idea to take a look at the new versions of them, as the style is somewhat improved and they take advantage of some new features.

If you were doing tricky things with undocumented Autoconf internals (macros, variables, diversions), check whether you need to change anything to account for changes that have been made. Perhaps you can even use an officially supported technique in version 2 instead of kludging. Or perhaps not.

To speed up your locally written feature tests, add caching to them. See whether any of your tests are of general enough usefulness to encapsulate them into macros that you can share.

15.6 Upgrading From Version 2.13

The introduction of the previous section (see Section 15.5 [Autoconf 1], page 177) perfectly suits this section. . . .

> Autoconf version 2.50 is mostly backward compatible with version 2.13. However, it introduces better ways to do some things, and doesn't support some of the ugly things in version 2.13. So, depending on how sophisticated your 'configure.ac' files are, you might have to do some manual work in order to upgrade to version 2.50. This chapter points out some problems to watch for when upgrading. Also, perhaps your configure scripts could benefit from some of the new features in version 2.50; the changes are summarized in the file 'NEWS' in the Autoconf distribution.

15.6.1 Changed Quotation

The most important changes are invisible to you: the implementation of most macros have completely changed. This allowed more factorization of the code, better error messages, a higher uniformity of the user's interface etc. Unfortunately, as a side effect, some construct which used to (miraculously) work might break starting with Autoconf 2.50. The most common culprit is bad quotation.

For instance, in the following example, the message is not properly quoted:

```
AC_INIT
AC_CHECK_HEADERS(foo.h,,
AC_MSG_ERROR(cannot find foo.h, bailing out))
AC_OUTPUT
```

Autoconf 2.13 simply ignores it:

```
$ autoconf-2.13; ./configure --silent
creating cache ./config.cache
configure: error: cannot find foo.h
$
```

while Autoconf 2.50 will produce a broken 'configure':

```
$ autoconf-2.50; ./configure --silent
configure: error: cannot find foo.h
./configure: exit: bad non-numeric arg 'bailing'
./configure: exit: bad non-numeric arg 'bailing'
$
```

The message needs to be quoted, and the AC_MSG_ERROR invocation too!

```
AC_INIT
AC_CHECK_HEADERS(foo.h,,
                 [AC_MSG_ERROR([cannot find foo.h, bailing out])])
AC_OUTPUT
```

Many many (and many more) Autoconf macros were lacking proper quotation, including no less than. . . AC_DEFUN itself!

```
$ cat configure.in
AC_DEFUN([AC_PROG_INSTALL],
[# My own much better version
```

```
])
AC_INIT
AC_PROG_INSTALL
AC_OUTPUT
$ autoconf-2.13
autoconf: Undefined macros:
***BUG in Autoconf--please report*** AC_FD_MSG
***BUG in Autoconf--please report*** AC_EPI
configure.in:1:AC_DEFUN([AC_PROG_INSTALL],
configure.in:5:AC_PROG_INSTALL
$ autoconf-2.50
$
```

15.6.2 New Macros

Because Autoconf has been dormant for years, Automake provided Autoconf-like macros for a while. Autoconf 2.50 now provides better versions of these macros, integrated in the `AC_` namespace, instead of `AM_`. But in order to ease the upgrading via `autoupdate`, bindings to such `AM_` macros are provided.

Unfortunately Automake did not quote the names of these macros! Therefore, when `m4` finds something like '`AC_DEFUN(AM_TYPE_PTRDIFF_T, ...)`' in '`aclocal.m4`', `AM_TYPE_PTRDIFF_T` is expanded, replaced with its Autoconf definition.

Fortunately Autoconf catches pre-`AC_INIT` expansions, and will complain, in its own words:

```
$ cat configure.in
AC_INIT
AM_TYPE_PTRDIFF_T
$ aclocal-1.4
$ autoconf
./aclocal.m4:17: error: m4_defn: undefined macro: _m4_divert_diversion
actypes.m4:289: AM_TYPE_PTRDIFF_T is expanded from...
./aclocal.m4:17: the top level
$
```

Future versions of Automake will simply no longer define most of these macros, and will properly quote the names of the remaining macros. But you don't have to wait for it to happen to do the right thing right now: do not depend upon macros from Automake as it is simply not its job to provide macros (but the one it requires itself):

```
$ cat configure.in
AC_INIT
AM_TYPE_PTRDIFF_T
$ rm aclocal.m4
$ autoupdate
autoupdate: 'configure.in' is updated
$ cat configure.in
AC_INIT
AC_CHECK_TYPES([ptrdiff_t])
$ aclocal-1.4
```

```
$ autoconf
$
```

15.6.3 Hosts and Cross-Compilation

Based on the experience of compiler writers, and after long public debates, many aspects of the cross-compilation chain have changed:

- the relationship between the build, host, and target architecture types,
- the command line interface for specifying them to `configure`,
- the variables defined in `configure`,
- the enabling of cross-compilation mode.

The relationship between build, host, and target have been cleaned up: the chain of default is now simply: target defaults to host, host to build, and build to the result of `config.guess`. Nevertheless, in order to ease the transition from 2.13 to 2.50, the following transition scheme is implemented. *Do not rely on it*, as it will be completely disabled in a couple of releases (we cannot keep it, as it proves to cause more problems than it cures).

They all default to the result of running `config.guess`, unless you specify either '`--build`' or '`--host`'. In this case, the default becomes the system type you specified. If you specify both, and they're different, `configure` will enter cross compilation mode, so it won't run any tests that require execution.

Hint: if you mean to override the result of `config.guess`, prefer '`--build`' over '`--host`'. In the future, '`--host`' will not override the name of the build system type. Whenever you specify `--host`, be sure to specify `--build` too.

For backward compatibility, `configure` will accept a system type as an option by itself. Such an option will override the defaults for build, host, and target system types. The following configure statement will configure a cross toolchain that will run on NetBSD/alpha but generate code for GNU Hurd/sparc, which is also the build platform.

```
./configure --host=alpha-netbsd sparc-gnu
```

In Autoconf 2.13 and before, the variables `build`, `host`, and `target` had a different semantics before and after the invocation of `AC_CANONICAL_BUILD` etc. Now, the argument of '`--build`' is strictly copied into `build_alias`, and is left empty otherwise. After the `AC_CANONICAL_BUILD`, `build` is set to the canonicalized build type. To ease the transition, before, its contents is the same as that of `build_alias`. Do *not* rely on this broken feature.

For consistency with the backward compatibility scheme exposed above, when '`--host`' is specified but '`--build`' isn't, the build system will be assumed to be the same as '`--host`', and '`build_alias`' will be set to that value. Eventually, this historically incorrect behavior will go away.

The former scheme to enable cross-compilation proved to cause more harm than good, in particular, it used to be triggered too easily, leaving regular end users puzzled in front of cryptic error messages. `configure` could even enter cross-compilation mode only because the compiler was not functional. This is mainly because `configure` used to try to detect cross-compilation, instead of waiting for an explicit flag from the user.

Now, `configure` enters cross-compilation mode if and only if '`--host`' is passed.

That's the short documentation. To ease the transition between 2.13 and its successors, a more complicated scheme is implemented. *Do not rely on the following*, as it will be removed in the near future.

If you specify '`--host`', but not '`--build`', when `configure` performs the first compiler test it will try to run an executable produced by the compiler. If the execution fails, it will enter cross-compilation mode. This is fragile. Moreover, by the time the compiler test is performed, it may be too late to modify the build-system type: other tests may have already been performed. Therefore, whenever you specify `--host`, be sure to specify `--build` too.

```
./configure --build=i686-pc-linux-gnu --host=m68k-coff
```

will enter cross-compilation mode. The former interface, which consisted in setting the compiler to a cross-compiler without informing `configure` is obsolete. For instance, `configure` will fail if it can't run the code generated by the specified compiler if you configure as follows:

```
./configure CC=m68k-coff-gcc
```

15.6.4 AC_LIBOBJ vs. LIBOBJS

Up to Autoconf 2.13, the replacement of functions was triggered via the variable `LIBOBJS`. Since Autoconf 2.50, the macro `AC_LIBOBJ` should be used instead (see Section 5.5.3 [Generic Functions], page 46). Starting at Autoconf 2.53, the use of `LIBOBJS` is an error.

This change is mandated by the unification of the GNU Build System components. In particular, the various fragile techniques used to parse a '`configure.ac`' are all replaced with the use of traces. As a consequence, any action must be traceable, which obsoletes critical variable assignments. Fortunately, `LIBOBJS` was the only problem, and it can even be handled gracefully (read, "without your having to change something").

There were two typical uses of `LIBOBJS`: asking for a replacement function, and adjusting `LIBOBJS` for Automake and/or Libtool.

As for function replacement, the fix is immediate: use `AC_LIBOBJ`. For instance:

```
LIBOBJS="$LIBOBJS fnmatch.o"
LIBOBJS="$LIBOBJS malloc.$ac_objext"
```

should be replaced with:

```
AC_LIBOBJ([fnmatch])
AC_LIBOBJ([malloc])
```

When asked for automatic de-ANSI-fication, Automake needs `LIBOBJS`'ed filenames to have '`$U`' appended to the base names. Libtool requires the definition of `LTLIBOBJS`, whose suffixes are mapped to '`.lo`'. People used to run snippets such as:

```
# This is necessary so that .o files in LIBOBJS are also built via
# the ANSI2KNR-filtering rules.
LIBOBJS=`echo "$LIBOBJS" | sed 's/\.o /\$U.o /g;s/\.o$/\$U.o/'`
LTLIBOBJS=`echo "$LIBOBJS" | sed 's/\.o/\.lo/g'`
```

```
AC_SUBST(LTLIBOBJS)
```

Note that this code is *wrong*, because '.o' is not the only possible extension[1]! It should have read:

```
# This is necessary so that .o files in LIBOBJS are also built via
# the ANSI2KNR-filtering rules.
LIB@&t@OBJS=`echo "$LIB@&t@OBJS" |
             sed 's,\.[[^.]]* ,$U&,g;s,\.[[^.]]*$,$U&,'`
LTLIBOBJS=`echo "$LIB@&t@OBJS" |
           sed 's,\.[[^.]]* ,.lo ,g;s,\.[[^.]]*$,.lo,'`
AC_SUBST(LTLIBOBJS)
```

You no longer have to use this: `AC_OUTPUT` normalizes `LIBOBJS` and `LTLIBOBJS` (hence it works with any version of Automake and Libtool). Just remove these lines (`autoupdate` cannot handle this task, since this is not a macro).

Note that `U` must not be used in your Makefiles.

15.6.5 AC_FOO_IFELSE vs. AC_TRY_FOO

Since Autoconf 2.50, internal codes uses `AC_PREPROC_IFELSE`, `AC_COMPILE_IFELSE`, `AC_LINK_IFELSE`, and `AC_RUN_IFELSE` on one hand and `AC_LANG_SOURCES`, and `AC_LANG_PROGRAM` on the other hand instead of the deprecated `AC_TRY_CPP`, `AC_TRY_COMPILE`, `AC_TRY_LINK`, and `AC_TRY_RUN`. The motivations where:

- a more consistent interface: `AC_TRY_COMPILE` etc. were double quoting their arguments;
- the combinatoric explosion is solved by decomposing on the one hand the generation of sources, and on the other hand executing the program;
- this scheme helps supporting more languages than plain C and C++.

In addition to the change of syntax, the philosphy has changed too: while emphasis was put on speed at the expense of accuracy, today's Autoconf promotes accuracy of the testing framework at, ahem..., the expense of speed.

As a perfect example of what is *not* to be done, here is how to find out whether a header file contains a particular declaration, such as a typedef, a structure, a structure member, or a function. Use `AC_EGREP_HEADER` instead of running `grep` directly on the header file; on some systems the symbol might be defined in another header file that the file you are checking '`#include`'s.

As a (bad) example, here is how you should not check for C preprocessor symbols, either defined by header files or predefined by the C preprocessor: using `AC_EGREP_CPP`:

```
AC_EGREP_CPP(yes,
[#ifdef _AIX
  yes
#endif
], is_aix=yes, is_aix=no)
```

The above example, properly written would (i) use `AC_LANG_PROGRAM`, and (ii) run the compiler:

[1] Yet another reason why assigning `LIBOBJS` directly is discouraged.

```
AC_COMPILE_IFELSE([AC_LANG_PROGRAM(
[[#if !defined _AIX
# error _AIX not defined
#endif
]])],
                   [is_aix=yes],
                   [is_aix=no])
```

16 Generating Test Suites with Autotest

> **N.B.: This section describes an experimental feature which will be part of Autoconf in a forthcoming release. Although we believe Autotest is stabilizing, this documentation describes an interface which might change in the future: do not depend upon Autotest without subscribing to the Autoconf mailing lists.**

It is paradoxical that portable projects depend on nonportable tools to run their test suite. Autoconf by itself is the paragon of this problem: although it aims at perfectly portability, up to 2.13, its test suite was using DejaGNU, a rich and complex testing framework, but which is far from being standard on Unix systems. Worse yet, it was likely to be missing on the most fragile platforms, the very platforms that are most likely to torture Autoconf and exhibit deficiencies.

To circumvent this problem many package maintainers have developed their own testing framework, based on simple shell scripts whose sole output are their exit status: the test succeeded, or failed. In addition, most of these tests share some common patterns, what results in lots of duplicated code, tedious maintenance etc.

Following exactly the same reasoning that yielded to the inception of Autoconf, Autotest provides a test suite generation frame work, based on M4 macros, building a portable shell script. The suite itself is equipped with automatic logging and tracing facilities which greatly diminish the interaction with bug reporters, and simple timing reports.

Autoconf itself has been using Autotest for years, and we do attest that it has considerably improved the strength of the test suite, and the quality of bug reports. Other projects are known to use some generation of Autotest, such as Bison, Free Recode, Free Wdiff, GNU Tar, each of them having different needs, what slowly polishes Autotest as a general testing framework.

Nonetheless, compared to DejaGNU, Autotest is inadequate for interactive tool testing, which is probably its main limitation.

16.1 Using an Autotest Test Suite

16.1.1 `testsuite` Scripts

Generating testing or validation suites using Autotest is rather easy. The whole validation suite is held in a file to be processed through `autom4te`, itself using GNU M4 under the scene, to produce a stand-alone Bourne shell script which then gets distributed. Neither `autom4te` nor GNU M4 are not needed anymore at the installer end.

Each test of the validation suite should be part of some test group. A *test group* is a sequence of interwoven tests that ought to be executed together, usually because one test in the group creates data files than a later test in the same group needs to read. Complex test groups make later debugging more tedious. It is much better keeping keep only a few tests per test group, and if you can put only one test per test group, this is just ideal.

For all but the simplest packages, some file such as '`testsuite.at`' does not fully hold all test sources, as these are often easier to maintain in separate files. Each of these separate files holds a single test group, or a sequence of test groups all addressing some common functionality in the package. In such cases, file '`testsuite.at`' only initializes the whole validation suite, and sometimes do elementary health checking, before listing include statements for all other test files. The special file '`package.m4`', containing the identification of the package, is automatically included if found.

A convenient alternative consists in moving all the global issues (local Autotest macros, elementary health checking, and `AT_INIT` invocation) into the file `local.at`, and making '`testsuite.at`' be a simple list of `m4_include` of sub test suites. In such case, generating the whole test suite or pieces of it is only a matter of choosing the `autom4te` command line arguments.

The validation scripts that Autotest produces are by convention called `testsuite`. When run, `testsuite` executes each test group in turn, producing only one summary line per test to say if that particular test succeeded or failed. At end of all tests, summarizing counters get printed. One debugging directory is left for each test group which failed, if any: such directories are named '`testsuite.dir/`*nn*', where *nn* is the sequence number of the test group, and they include:

- a debugging script named '`run`' which reruns the test in *debug mode* (see Section 16.3 [testsuite Invocation], page 191). The automatic generation of debugging scripts has the purpose of easing the chase for bugs.
- all the files created with `AT_DATA`
- a log of the run, named '`testsuite.log`'

In the ideal situation, none of the tests fail, and consequently, no debugging directory is left out of validation.

It often happens in practice that individual tests in the validation suite need to get information coming out of the configuration process. Some of this information, common for all validation suites, is provided through the file '`atconfig`', automatically created by `AC_CONFIG_TESTDIR`. For configuration informations which your testing environment specifically needs, you might prepare an optional file named '`atlocal.in`', instantiated by `AC_CONFIG_FILES`. The configuration process produces '`atconfig`' and '`atlocal`' out of these two input files, and these two produced files are automatically read by the '`testsuite`' script.

Here is a diagram showing the relationship between files.

Files used in preparing a software package for distribution:

```
                [package.m4] -->.
                                 \
subfile-1.at ->.  [local.at] ---->+
    ...         \                  \
subfile-i.at ---->-- testsuite.at -->-- autom4te* -->testsuite
    ...         /
subfile-n.at ->'
```

Files used in configuring a software package:

```
                                    .--> atconfig
                                   /
[atlocal.in] -->  config.status* --<
                                   \
                                    `--> [atlocal]
```

Files created during the test suite execution:

```
atconfig -->.                    .--> testsuite.log
             \                  /
              >-- testsuite* --<
             /                  \
[atlocal] ->'                    `--> [testsuite.dir]
```

16.1.2 Autotest Logs

When run, the test suite creates a log file named after itself, e.g., a test suite named `testsuite` creates '`testsuite.log`'. It contains a lot of information, usually more than maintainers actually need, but therefore most of the time it contains all that is needed:

command line arguments

A very bad Unix habit which is unfortunately wide spread consists of setting environment variables before the command, such as in '`CC=my-home-grown-cc ./testsuite`'. This results in the test suite not knowing this change, hence (i) it can't report it to you, and (ii) it cannot preserve the value of `CC` for subsequent runs. Autoconf faced exactly the same problem, and solved it by asking users to pass the variable definitions as command line arguments. Autotest requires this rule too, but has no means to enforce it; the log then contains a trace of the variables the user changed.

'`ChangeLog`' excerpts

The topmost lines of all the '`ChangeLog`'s found in the source hierarchy. This is especially useful when bugs are reported against development versions of the package, since the version string does not provide sufficient information to know the exact state of the sources the user compiled. Of course this relies on the use of a '`ChangeLog`'.

build machine

Running a test suite in a cross-compile environment is not an easy task, since it would mean having the test suite run on a machine *build*, while running programs on a machine *host*. It is much simpler to run both the test suite and the programs on *host*, but then, from the point of view of the test suite, there remains a single environment, *host* = *build*. The log contains relevant information on the state of the build machine, including some important environment variables.

tested programs

The absolute path and answers to '`--version`' of the tested programs (see Section 16.2 [Writing testsuite.at], page 189, `AT_TESTED`).

configuration log

The contents of '`config.log`', as created by `configure`, are appended. It contains the configuration flags and a detailed report on the configuration itself.

16.2 Writing '`testsuite.at`'

The '`testsuite.at`' is a Bourne shell script making use of special Autotest M4 macros. It often contains a call to `AT_INIT` nears its beginning followed by one call to `m4_include` per source file for tests. Each such included file, or the remainder of '`testsuite.at`' if include files are not used, contain a sequence of test groups. Each test group begins with one call to `AT_SETUP`, it contains an arbitrary number of shell commands or calls to `AT_CHECK`, and it completes with one call to `AT_CLEANUP`.

AT_INIT ([*name*]) [Macro]

Initialize Autotest. Giving a *name* to the test suite is encouraged if your package includes several test suites. In any case, the test suite always displays the package name and version. It also inherits the package bug report address.

AT_TESTED (*executables*) [Macro]

Log the path and answer to '`--version`' of each program in space-separated list *executables*. Several invocations register new executables, in other words, don't fear registering one program several times.

Autotest test suites rely on the `PATH` to find the tested program. This saves from generating the absolute paths to the various tools, and makes it possible to test installed programs. Therefore, knowing what programs are being exercised is crucial to understand some problems in the test suite itself, or its occasional misuses. It is a good idea to also subscribe foreign programs you depend upon, to ease incompatibility diagnostics.

AT_SETUP (*test-group-name*) [Macro]

This macro starts a group of related tests, all to be executed in the same subshell. It accepts a single argument, which holds a few words (no more than about 30 or 40 characters) quickly describing the purpose of the test group being started.

AT_KEYWORDS (*keywords*) [Macro]

Associate the space-separated list of *keywords* to the enclosing test group. This makes it possible to run "slices" of the test suite. For instance if some of your test groups exercise some '`foo`' feature, then using '`AT_KEYWORDS(foo)`' lets you run '`./testsuite -k foo`' to run exclusively these test groups. The *title* of the test group is automatically recorded to `AT_KEYWORDS`.

Several invocations within a test group accumulate new keywords. In other words, don't fear registering several times the same keyword in a test group.

AT_XFAIL_IF (*shell-condition*) [Macro]

Determine whether the test is expected to fail because it is a known bug (for unsupported features, you should skip the test). *shell-condition* is a shell expression such as a `test` command; you can instantiate this macro many times from within the same test group, and one of the conditions will be enough to turn the test into an expected failure.

AT_CLEANUP [Macro]

End the current test group.

AT_DATA (*file, contents*) [Macro]

Initialize an input data *file* with given *contents*. Of course, the *contents* have to be properly quoted between square brackets to protect against included commas or spurious M4 expansion. The contents ought to end with an end of line.

AT_CHECK (*commands*, [*status* = '0'], [*stdout* = ''], [*stderr* = ''], [*run-if-fail*], [*run-if-pass*]) [Macro]

Execute a test by performing given shell *commands*. These commands should normally exit with *status*, while producing expected *stdout* and *stderr* contents. If *commands* exit with status 77, then the whole test group is skipped. Otherwise, if this test fails, run shell commands *run-if-fail* or, if this test passes, run shell commands *run-if-pass*.

The *commands must not* redirect the standard output, nor the standard error.

If *status*, or *stdout*, or *stderr* is '`ignore`', then the corresponding value is not checked.

The special value '`expout`' for *stdout* means the expected output of the *commands* is the content of the file '`expout`'. If *stdout* is '`stdout`', then the standard output of the *commands* is available for further tests in the file '`stdout`'. Similarly for *stderr* with '`expout`' and '`stderr`'.

16.3 Running `testsuite` Scripts

Autotest test suites support the following arguments:

'`--help`'
'`-h`' Display the list of options and exit successfully.

'`--version`'
'`-V`' Display the version of the test suite and exit successfully.

'`--clean`'
'`-c`' Remove all the files the test suite might have created and exit. Meant for `clean` Makefile targets.

'`--list`'
'`-l`' List all the tests (or only the selection), including their possible keywords.

By default all the tests are performed (or described with '`--list`') in the default environment first silently, then verbosely, but the environment, set of tests, and verbosity level can be tuned:

'*`variable=value`*'
Set the environment *variable* to *value*. Do not run '`FOO=foo ./testsuite`' as debugging scripts would then run in a different environment.

The variable `AUTOTEST_PATH` specifies the testing path to prepend to `PATH`. It handles specially relative paths (not starting with '`/`'): they are considered to be relative to the top level of the package being built. All the directories are made absolute, first starting from the top level *build* tree, then from the *source* tree. For instance '`./testsuite AUTOTEST_PATH=tests:bin`' for a '`/src/foo-1.0`' source package built in '`/tmp/foo`' results in '`/tmp/foo/tests:/tmp/foo/bin`' and then '`/src/foo-1.0/tests:/src/foo-1.0/bin`' being prepended to `PATH`.

'*`number`*'
'*`number-number`*'
'*`number-`*'
'*`-number`*' Add the corresponding test groups, with obvious semantics, to the selection.

'--keywords=*keywords*'
'-k *keywords*'
: Add to the selection the test groups which title or keywords (arguments to `AT_SETUP` or `AT_KEYWORDS`) match *all* the keywords of the comma separated list *keywords*.

  Running './testsuite -k autoupdate,FUNC' will select all the tests tagged with 'autoupdate' *and* 'FUNC' (as in 'AC_CHECK_FUNC', 'AC_FUNC_FNMATCH' etc.) while './testsuite -k autoupdate -k FUNC' runs all the tests tagged with 'autoupdate' *or* 'FUNC'.

'--errexit'
'-e'
: If any test fails, immediately abort testing. It implies '--debug': post test group clean up, debugging script generation, and logging are inhibited. This option is meant for the full test suite, it is not really useful for generated debugging scripts.

'--verbose'
'-v'
: Force more verbosity in the detailed output of what is being done. This is the default for debugging scripts.

'--debug'
'-d'
: Do not remove the files after a test group was performed —but they are still removed *before*, therefore using this option is sane when running several test groups. Do not create debugging scripts. Do not log (in order to preserve supposedly existing full log file). This is the default for debugging scripts, but it can also be useful to debug the testsuite itself.

'--trace'
'-x'
: Trigger shell tracing of the test groups.

16.4 Making testsuite Scripts

For putting Autotest into movement, you need some configuration and Makefile machinery. We recommend, at least if your package uses deep or shallow hierarchies, that you use 'tests/' as the name of the directory holding all your tests and their 'Makefile'. Here is a check list of things to do.

– Make sure to create the file 'package.m4', which defines the identity of the package. It must define `AT_PACKAGE_STRING`, the full signature of the package, and `AT_PACKAGE_BUGREPORT`, the address to which bug reports should be sent. For sake of completeness, we suggest that you also define `AT_PACKAGE_NAME`, `AT_PACKAGE_TARNAME`, and `AT_PACKAGE_VERSION`. See Section 4.1 [Initializing configure], page 15, for a description of these variables. We suggest the following Makefile excerpt:

```
$(srcdir)/package.m4: $(top_srcdir)/configure.ac
        {                                      \
          echo '# Signature of the current package.'; \
          echo 'm4_define([AT_PACKAGE_NAME],      [@PACKAGE_NAME@])'; \
          echo 'm4_define([AT_PACKAGE_TARNAME],   [@PACKAGE_TARNAME@])'; \
          echo 'm4_define([AT_PACKAGE_VERSION],   [@PACKAGE_VERSION@])'; \
          echo 'm4_define([AT_PACKAGE_STRING],    [@PACKAGE_STRING@])'; \
          echo 'm4_define([AT_PACKAGE_BUGREPORT], [@PACKAGE_BUGREPORT@])'; \
        } >$(srcdir)/package.m4
```

Be sure to distribute '`package.m4`' and to put it into the source hierarchy: the test suite ought to be shipped!

– Invoke `AC_CONFIG_TESTDIR`.

AC_CONFIG_TESTDIR (*directory*, [*test-path* = '*directory*']) [Macro]

An Autotest test suite is to be configured in *directory*. This macro requires the instantiation of '*directory*`/atconfig`' from '*directory*`/atconfig.in`', and sets the default `AUTOTEST_PATH` to *test-path* (see Section 16.3 [testsuite Invocation], page 191).

– Still within '`configure.ac`', as appropriate, ensure that some `AC_CONFIG_FILES` command includes substitution for '`tests/atlocal`'.

– The '`tests/Makefile.in`' should be modified so the validation in your package is triggered by '`make check`'. An example is provided below.

With Automake, here is a minimal example about how to link '`make check`' with a validation suite.

```
EXTRA_DIST = testsuite.at testsuite
TESTSUITE = $(srcdir)/testsuite
check-local: atconfig atlocal $(TESTSUITE)
        $(SHELL) $(TESTSUITE)

AUTOTEST = $(AUTOM4TE) --language=autotest
$(TESTSUITE): $(srcdir)/testsuite.at
        $(AUTOTEST) -I $(srcdir) $@.at -o $@.tmp
        mv $@.tmp $@
```

You might want to list explicitly the dependencies, i.e., the list of the files '`testsuite.at`' includes.

With strict Autoconf, you might need to add lines inspired from the following:

```
subdir = tests

atconfig: $(top_builddir)/config.status
        cd $(top_builddir) && \
           $(SHELL) ./config.status $(subdir)/$@

atlocal: $(srcdir)/atlocal.in $(top_builddir)/config.status
        cd $(top_builddir) && \
           $(SHELL) ./config.status $(subdir)/$@
```

and manage to have '`atconfig.in`' and `$(EXTRA_DIST)` distributed.

17 Frequent Autoconf Questions, with answers

Several questions about Autoconf come up occasionally. Here some of them are addressed.

17.1 Distributing `configure` Scripts

> What are the restrictions on distributing `configure` scripts that Autoconf generates? How does that affect my programs that use them?

There are no restrictions on how the configuration scripts that Autoconf produces may be distributed or used. In Autoconf version 1, they were covered by the GNU General Public License. We still encourage software authors to distribute their work under terms like those of the GPL, but doing so is not required to use Autoconf.

Of the other files that might be used with `configure`, '`config.h.in`' is under whatever copyright you use for your '`configure.ac`'. '`config.sub`' and '`config.guess`' have an exception to the GPL when they are used with an Autoconf-generated `configure` script, which permits you to distribute them under the same terms as the rest of your package. '`install-sh`' is from the X Consortium and is not copyrighted.

17.2 Why Require GNU M4?

> Why does Autoconf require GNU M4?

Many M4 implementations have hard-coded limitations on the size and number of macros that Autoconf exceeds. They also lack several builtin macros that it would be difficult to get along without in a sophisticated application like Autoconf, including:

```
m4_builtin
m4_indir
m4_bpatsubst
__file__
__line__
```

Autoconf requires version 1.4 or above of GNU M4 because it uses frozen state files.

Since only software maintainers need to use Autoconf, and since GNU M4 is simple to configure and install, it seems reasonable to require GNU M4 to be installed also. Many maintainers of GNU and other free software already have most of the GNU utilities installed, since they prefer them.

17.3 How Can I Bootstrap?

> If Autoconf requires GNU M4 and GNU M4 has an Autoconf `configure` script, how do I bootstrap? It seems like a chicken and egg problem!

This is a misunderstanding. Although GNU M4 does come with a `configure` script produced by Autoconf, Autoconf is not required in order to run the script and install GNU M4. Autoconf is only required if you want to change the M4 `configure` script, which few people have to do (mainly its maintainer).

17.4 Why Not Imake?

> Why not use Imake instead of `configure` scripts?

Several people have written addressing this question, so I include adaptations of their explanations here.

The following answer is based on one written by Richard Pixley:

> Autoconf generated scripts frequently work on machines that it has never been set up to handle before. That is, it does a good job of inferring a configuration for a new system. Imake cannot do this.
>
> Imake uses a common database of host specific data. For X11, this makes sense because the distribution is made as a collection of tools, by one central authority who has control over the database.
>
> GNU tools are not released this way. Each GNU tool has a maintainer; these maintainers are scattered across the world. Using a common database would be a maintenance nightmare. Autoconf may appear to be this kind of database, but in fact it is not. Instead of listing host dependencies, it lists program requirements.
>
> If you view the GNU suite as a collection of native tools, then the problems are similar. But the GNU development tools can be configured as cross tools in almost any host+target permutation. All of these configurations can be installed concurrently. They can even be configured to share host independent files across hosts. Imake doesn't address these issues.
>
> Imake templates are a form of standardization. The GNU coding standards address the same issues without necessarily imposing the same restrictions.

Here is some further explanation, written by Per Bothner:

> One of the advantages of Imake is that it easy to generate large Makefiles using `cpp`'s '`#include`' and macro mechanisms. However, `cpp` is not programmable: it has limited conditional facilities, and no looping. And `cpp` cannot inspect its environment.
>
> All of these problems are solved by using `sh` instead of `cpp`. The shell is fully programmable, has macro substitution, can execute (or source) other shell scripts, and can inspect its environment.

Paul Eggert elaborates more:

> With Autoconf, installers need not assume that Imake itself is already installed and working well. This may not seem like much of an advantage to people who are accustomed to Imake. But on many hosts Imake is not installed or the default installation is not working well, and requiring Imake to install a package hinders the acceptance of that package on those hosts. For example, the Imake template and configuration files might not be installed properly on a host, or the Imake build procedure might wrongly assume that all source files are in one big directory tree, or the Imake configuration might assume one compiler whereas the package or the installer needs to use another, or there might be a version mismatch between the Imake expected by the package and the Imake supported by the host. These problems are much rarer with Autoconf, where each package comes with its own independent configuration processor.

> Also, Imake often suffers from unexpected interactions between `make` and the installer's C preprocessor. The fundamental problem here is that the C preprocessor was designed to preprocess C programs, not '`Makefile`'s. This is much less of a problem with Autoconf, which uses the general-purpose preprocessor M4, and where the package's author (rather than the installer) does the preprocessing in a standard way.

Finally, Mark Eichin notes:

> Imake isn't all that extensible, either. In order to add new features to Imake, you need to provide your own project template, and duplicate most of the features of the existing one. This means that for a sophisticated project, using the vendor-provided Imake templates fails to provide any leverage—since they don't cover anything that your own project needs (unless it is an X11 program).
>
> On the other side, though:
>
> The one advantage that Imake has over `configure`: '`Imakefile`'s tend to be much shorter (likewise, less redundant) than '`Makefile.in`'s. There is a fix to this, however—at least for the Kerberos V5 tree, we've modified things to call in common '`post.in`' and '`pre.in`' '`Makefile`' fragments for the entire tree. This means that a lot of common things don't have to be duplicated, even though they normally are in `configure` setups.

17.5 How Do I `#define` Installation Directories?

> My program needs library files, installed in `datadir` and similar. If I use
>
> ```
> AC_DEFINE_UNQUOTED([DATADIR], [$datadir],
>                    [Define to the read-only architecture-independent
>                     data directory.])
> ```
>
> I get
>
> ```
> #define DATADIR "${prefix}/share"
> ```

As already explained, this behavior is on purpose, mandated by the GNU Coding Standards, see Section 4.7.2 [Installation Directory Variables], page 22. There are several means to achieve a similar goal:

- Do not use `AC_DEFINE` but use your '`Makefile`' to pass the actual value of `datadir` via compilation flags, see Section 4.7.2 [Installation Directory Variables], page 22, for the details.
- This solution can be simplified when compiling a program: you may either extend the `CPPFLAGS`:

  ```
  CPPFLAGS = -DDATADIR=\"$(datadir)\" @CPPFLAGS@
  ```

  or create a dedicated header file:

  ```
  DISTCLEANFILES = datadir.h
  datadir.h: Makefile
          echo '#define DATADIR "$(datadir)"' >$@
  ```

– Use `AC_DEFINE` but have `configure` compute the literal value of `datadir` and others. Many people have wrapped macros to automate this task. For instance, the macro `AC_DEFINE_DIR` from the Autoconf Macro Archive[1].

  This solution does not conform to the GNU Coding Standards.

– Note that all the previous solutions hard wire the absolute path to these directories in the executables, which is not a good property. You may try to compute the paths relatively to `prefix`, and try to find `prefix` at runtime, this way your package is relocatable. Some macros are already available to address this issue: see `adl_COMPUTE_RELATIVE_PATHS` and `adl_COMPUTE_STANDARD_RELATIVE_PATHS` on the Autoconf Macro Archive[2].

17.6 What is 'autom4te.cache'?

What is this directory '`autom4te.cache`'? Can I safely remove it?

In the GNU Build System, '`configure.ac`' plays a central role and is read by many tools: `autoconf` to create '`configure`', `autoheader` to create '`config.h.in`', `automake` to create '`Makefile.in`', `autoscan` to check the completeness of '`configure.ac`', `autoreconf` to check the GNU Build System components that are used. To "read '`configure.ac`'" actually means to compile it with M4, which can be a very long process for complex '`configure.ac`'.

This is why all these tools, instead of running directly M4, invoke `autom4te` (see Section 8.2.1 [autom4te Invocation], page 95) which, while answering to a specific demand, stores additional information in '`autom4te.cache`' for future runs. For instance, if you run `autoconf`, behind the scenes, `autom4te` will also store information for the other tools, so that when you invoke `autoheader` or `automake` etc., re-processing '`configure.ac`' is not needed. The speed up is frequently of 30, and is increasing with the size of '`configure.ac`'.

But it is and remains being simply a cache: you can safely remove it.

Can I permanently get rid of it?

The creation of this cache can be disabled from '`~/.autom4te.cfg`', see Section 8.2.2 [Customizing autom4te], page 99, for more details. You should be aware that disabling the cache slows down the Autoconf test suite by 40%. The more GNU Build System components are used, the more the cache is useful; for instance running '`autoreconf -f`' on the Coreutils is twice slower without the cache *although '`--force`' implies that the cache is not fully exploited*, and eight times slower than without '`--force`'.

17.7 Header Present But Cannot Be Compiled

The most important guideline to bear in mind when checking for features is to mimic as much as possible the intended use. Unfortunately, old versions of `AC_CHECK_HEADER` and `AC_CHECK_HEADERS` failed to follow this idea, and called the preprocessor, instead of the compiler, to check for headers. As a result, incompatibilities between headers went unnoticed during configuration, and maintainers finally had to deal with this issue elsewhere.

As of Autoconf 2.56 both checks are performed, and `configure` complains loudly if the compiler and the preprocessor do not agree. For the time being the result used is that of

[1] Autoconf Macro Archive, `http://www.gnu.org/software/ac-archive/`.
[2] Autoconf Macro Archive, `http://www.gnu.org/software/ac-archive/`.

the preprocessor, to give maintainers time to adjust their 'configure.ac', but in the near future, only the compiler will be considered.

Consider the following example:

```
$ cat number.h
typedef int number;
$ cat pi.h
const number pi = 3;
$ cat configure.ac
AC_INIT
AC_CHECK_HEADERS(pi.h)
$ autoconf -Wall
$ ./configure
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking how to run the C preprocessor... gcc -E
checking for egrep... grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking pi.h usability... no
checking pi.h presence... yes
configure: WARNING: pi.h: present but cannot be compiled
configure: WARNING: pi.h: check for missing prerequisite headers?
configure: WARNING: pi.h: proceeding with the preprocessor's result
configure: WARNING:     ## ------------------------------------ ##
configure: WARNING:     ## Report this to bug-autoconf@gnu.org. ##
configure: WARNING:     ## ------------------------------------ ##
checking for pi.h... yes
```

The proper way the handle this case is using the fourth argument (see Section 5.6.3 [Generic Headers], page 53):

```
$ cat configure.ac
AC_INIT
```

```
AC_CHECK_HEADERS(number.h pi.h,,,
[[#if HAVE_NUMBER_H
# include <number.h>
#endif
]])
$ autoconf -Wall
$ ./configure
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for number.h... yes
checking for pi.h... yes
```

See Section 5.6.2 [Particular Headers], page 49, for a list of headers with their prerequisite.

18 History of Autoconf

You may be wondering, Why was Autoconf originally written? How did it get into its present form? (Why does it look like gorilla spit?) If you're not wondering, then this chapter contains no information useful to you, and you might as well skip it. If you *are* wondering, then let there be light. . . .

18.1 Genesis

In June 1991 I was maintaining many of the GNU utilities for the Free Software Foundation. As they were ported to more platforms and more programs were added, the number of '`-D`' options that users had to select in the '`Makefile`' (around 20) became burdensome. Especially for me—I had to test each new release on a bunch of different systems. So I wrote a little shell script to guess some of the correct settings for the fileutils package, and released it as part of fileutils 2.0. That `configure` script worked well enough that the next month I adapted it (by hand) to create similar `configure` scripts for several other GNU utilities packages. Brian Berliner also adapted one of my scripts for his CVS revision control system.

Later that summer, I learned that Richard Stallman and Richard Pixley were developing similar scripts to use in the GNU compiler tools; so I adapted my `configure` scripts to support their evolving interface: using the file name '`Makefile.in`' as the templates; adding '`+srcdir`', the first option (of many); and creating '`config.status`' files.

18.2 Exodus

As I got feedback from users, I incorporated many improvements, using Emacs to search and replace, cut and paste, similar changes in each of the scripts. As I adapted more GNU utilities packages to use `configure` scripts, updating them all by hand became impractical. Rich Murphey, the maintainer of the GNU graphics utilities, sent me mail saying that the `configure` scripts were great, and asking if I had a tool for generating them that I could send him. No, I thought, but I should! So I started to work out how to generate them. And the journey from the slavery of hand-written `configure` scripts to the abundance and ease of Autoconf began.

Cygnus `configure`, which was being developed at around that time, is table driven; it is meant to deal mainly with a discrete number of system types with a small number of mainly unguessable features (such as details of the object file format). The automatic configuration system that Brian Fox had developed for Bash takes a similar approach. For general use, it seems to me a hopeless cause to try to maintain an up-to-date database of which features each variant of each operating system has. It's easier and more reliable to check for most features on the fly—especially on hybrid systems that people have hacked on locally or that have patches from vendors installed.

I considered using an architecture similar to that of Cygnus `configure`, where there is a single `configure` script that reads pieces of '`configure.in`' when run. But I didn't want to have to distribute all of the feature tests with every package, so I settled on having a different `configure` made from each '`configure.in`' by a preprocessor. That approach also offered more control and flexibility.

I looked briefly into using the Metaconfig package, by Larry Wall, Harlan Stenn, and Raphael Manfredi, but I decided not to for several reasons. The `Configure` scripts it produces are interactive, which I find quite inconvenient; I didn't like the ways it checked for some features (such as library functions); I didn't know that it was still being maintained, and the `Configure` scripts I had seen didn't work on many modern systems (such as System V R4 and NeXT); it wasn't very flexible in what it could do in response to a feature's presence or absence; I found it confusing to learn; and it was too big and complex for my needs (I didn't realize then how much Autoconf would eventually have to grow).

I considered using Perl to generate my style of `configure` scripts, but decided that M4 was better suited to the job of simple textual substitutions: it gets in the way less, because output is implicit. Plus, everyone already has it. (Initially I didn't rely on the GNU extensions to M4.) Also, some of my friends at the University of Maryland had recently been putting M4 front ends on several programs, including `tvtwm`, and I was interested in trying out a new language.

18.3 Leviticus

Since my `configure` scripts determine the system's capabilities automatically, with no interactive user intervention, I decided to call the program that generates them Autoconfig. But with a version number tacked on, that name would be too long for old UNIX file systems, so I shortened it to Autoconf.

In the fall of 1991 I called together a group of fellow questers after the Holy Grail of portability (er, that is, alpha testers) to give me feedback as I encapsulated pieces of my handwritten scripts in M4 macros and continued to add features and improve the techniques used in the checks. Prominent among the testers were François Pinard, who came up with the idea of making an Autoconf shell script to run M4 and check for unresolved macro calls; Richard Pixley, who suggested running the compiler instead of searching the file system to find include files and symbols, for more accurate results; Karl Berry, who got Autoconf to configure TeX and added the macro index to the documentation; and Ian Lance Taylor, who added support for creating a C header file as an alternative to putting '`-D`' options in a '`Makefile`', so he could use Autoconf for his UUCP package. The alpha testers cheerfully adjusted their files again and again as the names and calling conventions of the Autoconf macros changed from release to release. They all contributed many specific checks, great ideas, and bug fixes.

18.4 Numbers

In July 1992, after months of alpha testing, I released Autoconf 1.0, and converted many GNU packages to use it. I was surprised by how positive the reaction to it was. More people started using it than I could keep track of, including people working on software that wasn't part of the GNU Project (such as TCL, FSP, and Kerberos V5). Autoconf continued to improve rapidly, as many people using the `configure` scripts reported problems they encountered.

Autoconf turned out to be a good torture test for M4 implementations. UNIX M4 started to dump core because of the length of the macros that Autoconf defined, and several bugs showed up in GNU M4 as well. Eventually, we realized that we needed to use some features that only GNU M4 has. 4.3BSD M4, in particular, has an impoverished set of builtin macros; the System V version is better, but still doesn't provide everything we need.

More development occurred as people put Autoconf under more stresses (and to uses I hadn't anticipated). Karl Berry added checks for X11. david zuhn contributed C++ support. François Pinard made it diagnose invalid arguments. Jim Blandy bravely coerced it into configuring GNU Emacs, laying the groundwork for several later improvements. Roland McGrath got it to configure the GNU C Library, wrote the `autoheader` script to automate the creation of C header file templates, and added a '`--verbose`' option to `configure`. Noah Friedman added the '`--autoconf-dir`' option and `AC_MACRODIR` environment variable. (He also coined the term *autoconfiscate* to mean "adapt a software package to use Autoconf".) Roland and Noah improved the quoting protection in `AC_DEFINE` and fixed many bugs, especially when I got sick of dealing with portability problems from February through June, 1993.

18.5 Deuteronomy

A long wish list for major features had accumulated, and the effect of several years of patching by various people had left some residual cruft. In April 1994, while working for Cygnus Support, I began a major revision of Autoconf. I added most of the features of the Cygnus `configure` that Autoconf had lacked, largely by adapting the relevant parts of Cygnus `configure` with the help of david zuhn and Ken Raeburn. These features include support for using '`config.sub`', '`config.guess`', '`--host`', and '`--target`'; making links to files; and running `configure` scripts in subdirectories. Adding these features enabled Ken to convert GNU `as`, and Rob Savoye to convert DejaGNU, to using Autoconf.

I added more features in response to other peoples' requests. Many people had asked for `configure` scripts to share the results of the checks between runs, because (particularly when configuring a large source tree, like Cygnus does) they were frustratingly slow. Mike Haertel suggested adding site-specific initialization scripts. People distributing software that had to unpack on MS-DOS asked for a way to override the '`.in`' extension on the file names, which produced file names like '`config.h.in`' containing two dots. Jim Avera did an extensive examination of the problems with quoting in `AC_DEFINE` and `AC_SUBST`; his insights led to significant improvements. Richard Stallman asked that compiler output be sent to '`config.log`' instead of '`/dev/null`', to help people debug the Emacs `configure` script.

I made some other changes because of my dissatisfaction with the quality of the program. I made the messages showing results of the checks less ambiguous, always printing a result. I regularized the names of the macros and cleaned up coding style inconsistencies. I added some auxiliary utilities that I had developed to help convert source code packages to use Autoconf. With the help of François Pinard, I made the macros not interrupt each others' messages. (That feature revealed some performance bottlenecks in GNU M4, which he hastily corrected!) I reorganized the documentation around problems people want to solve. And I began a test suite, because experience had shown that Autoconf has a pronounced tendency to regress when we change it.

Again, several alpha testers gave invaluable feedback, especially François Pinard, Jim Meyering, Karl Berry, Rob Savoye, Ken Raeburn, and Mark Eichin.

Finally, version 2.0 was ready. And there was much rejoicing. (And I have free time again. I think. Yeah, right.)

Appendix A Copying This Manual

A.1 GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B Indices

B.1 Environment Variable Index

This is an alphabetical list of the environment variables that Autoconf checks.

C

CDPATH 121
CONFIG_COMMANDS 165
CONFIG_FILES 165
CONFIG_HEADERS 165
CONFIG_LINKS 165
CONFIG_SHELL 164
CONFIG_SITE 157
CONFIG_STATUS 164

E

ENV 123

I

IFS 121

L

LANG 121
LANGUAGE 121
LC_ADDRESS 121
LC_ALL 121
LC_COLLATE 121
LC_CTYPE 121
LC_IDENTIFICATION 121
LC_MEASUREMENT 121
LC_MESSAGES 121
LC_MONETARY 121
LC_NAME 121
LC_NUMERIC 121
LC_PAPER 121
LC_TELEPHONE 121
LC_TIME 121
LINENO 122

M

MAIL 123
MAILPATH 123

N

NULLCMD 123

P

PATH_SEPARATOR 124
PS1 123
PS2 123
PS4 123
PWD 124

R

RANDOM 124

S

SIMPLE_BACKUP_SUFFIX 166
status 124

W

WARNINGS 11, 13, 29, 96

B.2 Output Variable Index

This is an alphabetical list of the variables that Autoconf can substitute into files that it creates, typically one or more 'Makefile's. See Section 7.2 [Setting Output Variables], page 82, for more information on how this is done.

A

abs_builddir 22
abs_srcdir 22
abs_top_builddir 22
abs_top_srcdir 22
ALLOCA 41
AWK 35

B

bindir 22
build 150
build_alias 150
build_cpu 150
build_os 150
build_vendor 150
builddir 22

C

CC 60, 63, 69
CFLAGS 20, 60
configure_input 20
CPP 60
CPPFLAGS 21
cross_compiling 149
CXX 63
CXXCPP 63
CXXFLAGS 21, 63

D

datadir 22
DEFS 21

E

ECHO_C 21
ECHO_N 21
ECHO_T 21
EGREP 35
exec_prefix 23
EXEEXT 58, 169

F

F77 63
FC 64
FCFLAGS 21, 64
FCLIBS 64
FFLAGS 21, 63
FGREP 35
FLIBS 64

G

GETGROUPS_LIBS 43
GETLOADAVG_LIBS 43

H

host 150
host_alias 150
host_cpu 150
host_os 150
host_vendor 150

I

includedir 23
infodir 23
INSTALL 35
INSTALL_DATA 35
INSTALL_PROGRAM 35
INSTALL_SCRIPT 35

K

KMEM_GROUP 43

L

LDFLAGS 22
LEX 35
LEX_OUTPUT_ROOT 35
LEXLIB 35
libdir 23
libexecdir 23
LIBOBJS 43, 44, 47, 48, 55
LIBS 22, 69, 174, 177
LN_S 36
localstatedir 23

M

mandir 23

N

NEED_SETGID 43

O

OBJEXT 58, 172
oldincludedir 23

P

PACKAGE_BUGREPORT 15
PACKAGE_NAME 15
PACKAGE_STRING 15
PACKAGE_TARNAME 15
PACKAGE_VERSION 15
POW_LIB 45
prefix 23
program_transform_name 156

R

RANLIB 36

S

sbindir 23
SET_MAKE 17
sharedstatedir 23
srcdir 22
subdirs 31
sysconfdir 23

T

target 150
target_alias 150
target_cpu 150
target_os 150
target_vendor 150
top_builddir 22
top_srcdir 22

U

U 183

X

X_CFLAGS 68
X_EXTRA_LIBS 68
X_LIBS 68
X_PRE_LIBS 68

Y

YACC 36

B.3 Preprocessor Symbol Index

This is an alphabetical list of the C preprocessor symbols that the Autoconf macros define. To work with Autoconf, C source code needs to use these names in `#if` directives.

_

__CHAR_UNSIGNED__ 62
__PROTOTYPES 62
_ALL_SOURCE 69
_FILE_OFFSET_BITS 69
_GNU_SOURCE 69
_LARGE_FILES 69
_LARGEFILE_SOURCE 42
_MINIX 69
_POSIX_1_SOURCE 69
_POSIX_SOURCE 69
_POSIX_VERSION 52

C

C_ALLOCA 41
C_GETLOADAVG 43
CLOSEDIR_VOID 42
const 61

D

DGUX 43
DIRENT 169

F

F77_DUMMY_MAIN 65
F77_FUNC 66
F77_FUNC_ 66
F77_MAIN 66
F77_NO_MINUS_C_MINUS_O 64
FC_FUNC 66
FC_FUNC_ 66
FC_MAIN 66
FC_NO_MINUS_C_MINUS_O 64

G

GETGROUPS_T 56
GETLODAVG_PRIVILEGED 43
GETPGRP_VOID 43
gid_t 57
GWINSZ_IN_SYS_IOCTL 53

H

HAVE__BOOL 50
HAVE_ALLOCA_H 41
HAVE_CONFIG_H 27
HAVE_DECL_STRERROR_R 45
HAVE_DECL_*symbol* 54
HAVE_DIRENT_H 49
HAVE_DOPRNT 46
HAVE_*function* 46
HAVE_GETMNTENT 43
HAVE_*header* 53
HAVE_LONG_DOUBLE 62
HAVE_LONG_FILE_NAMES 69
HAVE_LSTAT_EMPTY_STRING_BUG 45
HAVE_MALLOC 44
HAVE_MBRTOWC 44
HAVE_MMAP 44
HAVE_NDIR_H 49
HAVE_NLIST_H 43
HAVE_OBSTACK 45
HAVE_REALLOC 45
HAVE_RESTARTABLE_SYSCALLS 175
HAVE_ST_BLKSIZE 55
HAVE_ST_BLOCKS 55
HAVE_ST_RDEV 55
HAVE_STAT_EMPTY_STRING_BUG 45
HAVE_STDBOOL_H 50
HAVE_STRCOLL 45
HAVE_STRERROR_R 45
HAVE_STRFTIME 46

HAVE_STRINGIZE ... 62
HAVE_STRNLEN ... 46
HAVE_STRUCT_STAT_ST_BLKSIZE ... 55
HAVE_STRUCT_STAT_ST_BLOCKS ... 55
HAVE_STRUCT_STAT_ST_RDEV ... 55
HAVE_SYS_DIR_H ... 49
HAVE_SYS_NDIR_H ... 49
HAVE_SYS_WAIT_H ... 52
HAVE_TM_ZONE ... 56
HAVE_TZNAME ... 56
HAVE_UTIME_NULL ... 46
HAVE_VFORK_H ... 42
HAVE_VPRINTF ... 46
HAVE_WAIT3 ... 170
HAVE_WORKING_FORK ... 42
HAVE_WORKING_VFORK ... 42

I

inline ... 62
INT_16_BITS ... 171

L

LONG_64_BITS ... 172
LSTAT_FOLLOWS_SLASHED_SYMLINK ... 43

M

MAJOR_IN_MKDEV ... 50
MAJOR_IN_SYSMACROS ... 50
malloc ... 44
mbstate_t ... 57
mode_t ... 57

N

NDIR ... 169
NEED_MEMORY_H ... 172
NEED_SETGID ... 43
NLIST_NAME_UNION ... 43
NO_MINUS_C_MINUS_O ... 60

O

off_t ... 57

P

PACKAGE_BUGREPORT ... 15
PACKAGE_NAME ... 15
PACKAGE_STRING ... 15
PACKAGE_TARNAME ... 15
PACKAGE_VERSION ... 15
PARAMS ... 62
pid_t ... 57
PROTOTYPES ... 62

R

realloc ... 45
restrict ... 61
RETSIGTYPE ... 57

S

SELECT_TYPE_ARG1 ... 45
SELECT_TYPE_ARG234 ... 45
SELECT_TYPE_ARG5 ... 45
SETPGRP_VOID ... 45
SETVBUF_REVERSED ... 45
size_t ... 57
STDC_HEADERS ... 51
STRERROR_R_CHAR_P ... 45
SVR4 ... 43
SYS_SIGLIST_DECLARED ... 169
SYSDIR ... 169
SYSNDIR ... 169

T

TIME_WITH_SYS_TIME ... 52
TM_IN_SYS_TIME ... 56

U

uid_t ... 57
UMAX ... 43
UMAX4_3 ... 43
USG ... 176

V

vfork ... 42
volatile ... 62

W

WORDS_BIGENDIAN ... 61

X

X_DISPLAY_MISSING ... 68

Y

YYTEXT_POINTER ... 35

B.4 Autoconf Macro Index

This is an alphabetical list of the Autoconf macros. To make the list easier to use, the macros are listed without their preceding 'AC_'.

A

AH_BOTTOM 30
AH_TEMPLATE 29
AH_TOP 30
AH_VERBATIM 29
AIX 69
ALLOCA 167
ARG_ARRAY 167
ARG_ENABLE 154
ARG_PROGRAM 156
ARG_VAR 83
ARG_WITH 153
AU_DEFUN 107

B

BEFORE 106

C

C_BIGENDIAN 61
C_CHAR_UNSIGNED 62
C_CONST 61
C_CROSS 167
C_INLINE 62
C_LONG_DOUBLE 62
C_PROTOTYPES 62
C_RESTRICT 61
C_STRINGIZE 62
C_VOLATILE 62
CACHE_CHECK 84
CACHE_LOAD 86
CACHE_SAVE 86
CACHE_VAL 84
CANONICAL_BUILD 150
CANONICAL_HOST 150
CANONICAL_SYSTEM 167
CANONICAL_TARGET 150
CHAR_UNSIGNED 167
CHECK_DECL 54
CHECK_DECLS 54
CHECK_FILE 38
CHECK_FILES 38
CHECK_FUNC 46
CHECK_FUNCS 46
CHECK_HEADER 53
CHECK_HEADERS 53
CHECK_LIB 38
CHECK_MEMBER 56
CHECK_MEMBERS 56
CHECK_PROG 37
CHECK_PROGS 37
CHECK_SIZEOF 58
CHECK_TOOL 37
CHECK_TOOLS 37
CHECK_TYPE 57, 168
CHECK_TYPES 57
CHECKING 168
COMPILE_CHECK 168
COMPILE_IFELSE 77
CONFIG_AUX_DIR 16
CONFIG_COMMANDS 30
CONFIG_FILES 19
CONFIG_HEADERS 27
CONFIG_LIBOBJ_DIR 47
CONFIG_LINKS 30
CONFIG_MACRO_DIR 17
CONFIG_SRCDIR 16
CONFIG_SUBDIRS 31
CONFIG_TESTDIR 193
CONST 168
COPYRIGHT 16
CROSS_CHECK 168
CYGWIN 169

D

DECL_SYS_SIGLIST 169
DECL_YYTEXT 169
DEFAULT_INCLUDES 34
DEFINE 81
DEFINE_UNQUOTED 81
DEFUN 103, 107
DIAGNOSE 104
DIR_HEADER 169
DYNIX_SEQ 169

E

EGREP_CPP 77
EGREP_HEADER 76
EMXOS2 169
ENABLE 155
ERROR 170
EXEEXT 169

F

F77_DUMMY_MAIN 65
F77_FUNC 67
F77_LIBRARY_LDFLAGS 64
F77_MAIN 66
F77_WRAPPERS 66
FATAL 105

FC_FREEFORM 68
FC_FUNC 67
FC_LIBRARY_LDFLAGS 64
FC_MAIN 66
FC_SRCEXT 67
FC_WRAPPERS 66
FIND_X 170
FIND_XTRA 170
FUNC_ALLOCA 41
FUNC_CHECK 170
FUNC_CHOWN 42
FUNC_CLOSEDIR_VOID 42
FUNC_ERROR_AT_LINE 42
FUNC_FNMATCH 42
FUNC_FNMATCH_GNU 42
FUNC_FORK 42
FUNC_FSEEKO 42
FUNC_GETGROUPS 43
FUNC_GETLOADAVG 43
FUNC_GETMNTENT 43
FUNC_GETPGRP 43
FUNC_LSTAT 45
FUNC_LSTAT_FOLLOWS_SLASHED_SYMLINK 43
FUNC_MALLOC 44
FUNC_MBRTOWC 44
FUNC_MEMCMP 44
FUNC_MKTIME 44
FUNC_MMAP 44
FUNC_OBSTACK 45
FUNC_REALLOC 45
FUNC_SELECT_ARGTYPES 45
FUNC_SETPGRP 45
FUNC_SETVBUF_REVERSED 45
FUNC_STAT 45
FUNC_STRCOLL 45
FUNC_STRERROR_R 45
FUNC_STRFTIME 46
FUNC_STRNLEN 46
FUNC_STRTOD 45
FUNC_UTIME_NULL 46
FUNC_VPRINTF 46
FUNC_WAIT3 170

G

GCC_TRADITIONAL 170
GETGROUPS_T 170
GETLOADAVG 170
GNU_SOURCE 69

H

HAVE_C_BACKSLASH_A 61
HAVE_FUNCS 170
HAVE_HEADERS 170
HAVE_LIBRARY 170
HAVE_POUNDBANG 170
HEADER_CHECK 170
HEADER_DIRENT 49
HEADER_EGREP 170
HEADER_MAJOR 50
HEADER_STAT 50
HEADER_STDBOOL 50
HEADER_STDC 51
HEADER_SYS_WAIT 52
HEADER_TIME 52
HEADER_TIOCGWINSZ 53
HELP_STRING 155, 170

I

INIT 15, 171
INLINE 171
INT_16_BITS 171
IRIX_SUN 171
ISC_POSIX 69

L

LANG_ASSERT 72
LANG_C 171
LANG_CALL 75
LANG_CONFTEST 74
LANG_CPLUSPLUS 171
LANG_FORTRAN77 171
LANG_FUNC_LINK_TRY 75
LANG_POP 72
LANG_PROGRAM 74
LANG_PUSH 72
LANG_RESTORE 171
LANG_SAVE 171
LANG_SOURCE 74
LANG_WERROR 59
LIBOBJ 47
LIBSOURCE 47
LIBSOURCES 47
LINK_FILES 171
LINK_IFELSE 77
LN_S 171
LONG_64_BITS 172
LONG_DOUBLE 172
LONG_FILE_NAMES 172

M

MAJOR_HEADER 172
MEMORY_H 172
MINGW32 172
MINIX 69
MINUS_C_MINUS_O 172
MMAP 172
MODE_T 172
MSG_CHECKING 87
MSG_ERROR 87
MSG_FAILURE 87
MSG_NOTICE 87

MSG_RESULT 87
MSG_WARN 88

O

OBJEXT 172
OBSOLETE 172
OFF_T 172
OUTPUT 17, 173
OUTPUT_COMMANDS 173
OUTPUT_COMMANDS_POST 30
OUTPUT_COMMANDS_PRE 30

P

PACKAGE_BUGREPORT 15
PACKAGE_NAME 15
PACKAGE_STRING 15
PACKAGE_TARNAME 15
PACKAGE_VERSION 15
PATH_PROG 37
PATH_PROGS 38
PATH_TOOL 38
PATH_X 68
PATH_XTRA 68
PID_T 173
PREFIX 173
PREFIX_DEFAULT 32
PREFIX_PROGRAM 32
PREPROC_IFELSE 76
PREREQ 15
PROG_AWK 35
PROG_CC 60
PROG_CC_C_O 60
PROG_CC_STDC 173
PROG_CPP 60
PROG_CPP_WERROR 60
PROG_CXX 63
PROG_CXXCPP 63
PROG_EGREP 35
PROG_F77 63
PROG_F77_C_O 64
PROG_FC 64
PROG_FC_C_O 64
PROG_FGREP 35
PROG_GCC_TRADITIONAL 63
PROG_INSTALL 35
PROG_LEX 35
PROG_LN_S 36
PROG_MAKE_SET 17
PROG_RANLIB 36
PROG_YACC 36
PROGRAM_CHECK 173
PROGRAM_EGREP 173
PROGRAM_PATH 174
PROGRAMS_CHECK 173
PROGRAMS_PATH 173

R

REMOTE_TAPE 174
REPLACE_FNMATCH 46
REPLACE_FUNCS 48
REQUIRE 105
REQUIRE_CPP 72
RESTARTABLE_SYSCALLS 174
RETSIGTYPE 174
REVISION 16
RSH 174
RUN_IFELSE 78

S

SCO_INTL 174
SEARCH_LIBS 39
SET_MAKE 174
SETVBUF_REVERSED 174
SIZE_T 174
SIZEOF_TYPE 174
ST_BLKSIZE 174
ST_BLOCKS 174
ST_RDEV 174
STAT_MACROS_BROKEN 50, 174
STDC_HEADERS 174
STRCOLL 174
STRUCT_ST_BLKSIZE 55
STRUCT_ST_BLOCKS 55
STRUCT_ST_RDEV 55
STRUCT_TIMEZONE 56
STRUCT_TM 56
SUBST 82
SUBST_FILE 82
SYS_INTERPRETER 68
SYS_LARGEFILE 69
SYS_LONG_FILE_NAMES 69
SYS_POSIX_TERMIOS 69
SYS_RESTARTABLE_SYSCALLS 175
SYS_SIGLIST_DECLARED 175

T

TEST_CPP 175
TEST_PROGRAM 175
TIME_WITH_SYS_TIME 175
TIMEZONE 175
TRY_COMPILE 175
TRY_CPP 175
TRY_LINK 175
TRY_LINK_FUNC 176
TRY_RUN 176
TYPE_GETGROUPS 56
TYPE_MBSTATE_T 57
TYPE_MODE_T 57
TYPE_OFF_T 57
TYPE_PID_T 57
TYPE_SIGNAL 57
TYPE_SIZE_T 57

TYPE_UID_T 57

U

UID_T 176
UNISTD_H 176
USG 176
UTIME_NULL 176

V

VALIDATE_CACHED_SYSTEM_TUPLE 176
VERBOSE 176
VFORK 176
VPRINTF 176

W

WAIT3 176
WARN 177
WARNING 105
WITH 154
WORDS_BIGENDIAN 177

X

XENIX_DIR 177

Y

YYTEXT_POINTER 177

B.5 M4 Macro Index

This is an alphabetical list of the M4, M4sugar, and M4sh macros. To make the list easier to use, the macros are listed without their preceding 'm4_' or 'AS_'.

B

bpatsubst 100
bregexp 100

D

defn 100
DIRNAME 102
dnl 100
dquote 101

E

exit 100

I

if 100
IF 102

M

MKDIR_P 102

P

pattern_allow 102
pattern_forbid 101
popdef 100

Q

quote 101

S

SET_CATFILE 102

U

undefine 100

W

wrap 100

B.6 Autotest Macro Index

This is an alphabetical list of the Autotest macros. To make the list easier to use, the macros are listed without their preceding 'AT_'.

C

CHECK . . . 191
CLEANUP . . . 190

D

DATA . . . 190

I

INIT . . . 190

K

KEYWORDS . . . 190

S

SETUP . . . 190

T

TESTED . . . 190

X

XFAIL_IF . . . 190

B.7 Program and Function Index

This is an alphabetical list of the programs and functions which portability is discussed in this document.

!

! . . . 124

.

. . . . 124

/

/usr/bin/ksh on Solaris . . . 112
/usr/dt/bin/dtksh on Solaris . . . 112
/usr/xpg4/bin/sh on Solaris . . . 112

A

alloca . . . 41
'alloca.h' . . . 41
awk . . . 131

B

break . . . 124

C

case . . . 125
cat . . . 131
cd . . . 124
chown . . . 42
closedir . . . 42
cmp . . . 132
cp . . . 132
'ctype.h' . . . 51

D

date . . . 132
diff . . . 133
'dirent.h' . . . 49
dirname . . . 133

E

echo . . . 125
egrep . . . 133
error_at_line . . . 42
exit . . . 39
exit . . . 126
export . . . 126
expr . . . 133, 134
expr ('|') . . . 133

F

false . . . 126
fgrep . . . 134
'float.h' . . . 51
fnmatch . . . 42, 46
'fnmatch.h' . . . 46
for . . . 126
fork . . . 42
fseeko . . . 42

G

getgroups 43
getloadavg 43
getmntent 43
getpgid 43
getpgrp 43
grep 135

I

if 127
'inttypes.h' 48

K

Korn shell 112
Ksh 112
'ksh88' 112
'ksh93' 112

L

'linux/irda.h' 48
'linux/random.h' 48
ln 135
ls 135
lstat 43, 45

M

malloc 44
mbrtowc 44
memcmp 44
mkdir 135
mktime 44
mmap 44
mv 135

N

'ndir.h' 49
'net/if.h' 48
'netinet/if_ether.h' 49
'nlist.h' 43

P

'pdksh' 112
printf 127
putenv 39
pwd 127

R

realloc 45

S

sed 136
sed ('t') 137
select 45
set 128
setpgrp 45
setvbuf 45
shift 128
signal 39
'signal.h' 57
snprintf 39
source 128
sprintf 40
sscanf 40
stat 45
'stdarg.h' 51
'stdbool.h' 50
'stdint.h' 48
'stdlib.h' 49, 51, 56
strcoll 45
strerror_r 45
strftime 46
'string.h' 51
'strings.h' 51
strnlen 40, 46
strtod 45
'sys/dir.h' 49
'sys/ioctl.h' 53
'sys/mkdev.h' 50
'sys/mount.h' 49
'sys/ndir.h' 49
'sys/socket.h' 49
'sys/stat.h' 50
'sys/sysmacros.h' 50
'sys/time.h' 52, 56
'sys/types.h' 56
'sys/ucred.h' 49
'sys/wait.h' 52
sysconf 40
'system.h' 50

T

'termios.h' 53
test 128
'time.h' 52, 56
touch 138
trap 129
true 130

U

'unistd.h' 52
unlink 40
unset 130
unsetenv 40
utime 46

V

va_copy 40
va_list 40
vfork 42
'vfork.h' 42
vprintf 46
vsnprintf 39
vsprintf 40

W

'wchar.h' 57

X

'X11/extensions/scrnsaver.h' 49

B.8 Concept Index

This is an alphabetical list of the files, tools, and concepts introduced in this document.

"

'"$@"' 117

$

$(*commands*) 119
$<, explicit rules, and VPATH 142
${*var=expanded-value*} 118
${*var=literal*} 117
$U 183

@

'@&t@' 93
'@S|@' 93

_

_m4_divert_diversion 181

\`

\`*commands*\` 119

A

'acconfig.h' 166
'aclocal.m4' 5
Ash 111
autoconf 10
autoheader 28
Autom4te Library 99
'autom4te.cache' 97
'autom4te.cfg' 99
Automake 3
automatic rule rewriting and VPATH 142
autoreconf 12
autoscan 9
Autotest 187
AUTOTEST_PATH 191
autoupdate 166

B

Back trace 11, 96
Bash 112
Bash 2.05 and later 112
BSD make and 'obj/' 141

C

Cache 83
Cache variable 85
Cache, enabling 162
Command Substitution 119
Comments in 'Makefile' rules 141
'config.h' 26
'config.h.bot' 166
'config.h.in' 27
'config.h.top' 166
config.status 163
config.sub 149
Configuration Header 26
Configuration Header Template 27
configure 5, 159
'configure.ac' 5
'configure.in' 5
Copyright Notice 16

D

Darwin 79
Declaration, checking 54
dnl 103, 107
double-colon rules and VPATH 142

E

Endianness 61
explicit rules, $<, and VPATH 142

F

FDL, GNU Free Documentation License 205
File, checking 38
Function, checking 41

H

Header, checking 48

I

ifnames 10
Includes, default 33
Instantiation 17

L

Language 71
Library, checking 38
Libtool 3
Links 30
Listing directories 135

M

M4sugar 100
Macro invocation stack 11, 96
make -k 141
make and SHELL 140
'Makefile' rules and comments 141
Making directories 135
Messages, from autoconf 104
Messages, from configure 86
Moving open files 135

O

'obj/', subdirectory 141
obstack 45

P

'package.m4' 192
POSIX termios headers 69
prerequisite directories and VPATH 144
Previous Variable 83
Programs, checking 35

Q

QNX 4.25 79
quadrigraphs 93
quotation 7, 89

R

Revision 16
Rule, Single Suffix Inference 147

S

Separated Dependencies 147
SHELL and make 140
Single Suffix Inference Rule 147
Structure, checking 55
Symbolic links 135

T

termios POSIX headers 69
test group 187
testsuite 187, 191
timestamp resolution 132, 138, 148
Tru64 79

U

undefined macro 181
Unix version 7 79

V

V7 79
Variable, Precious 83
Version 15
VPATH 142
VPATH and automatic rule rewriting 142
VPATH and double-colon rules 142
VPATH and prerequisite directories 144
VPATH, explicit rules, and $< 142
VPATH, resolving target pathnames 145

Z

Zsh 112