Autoconf

Creating Automatic Configuration Scripts
Edition 2.52, for Autoconf version 2.52
17 July 2001

David MacKenzie and Ben Elliston

Copyright © 1992, 1993, 1994, 1995, 1996, 1998, 1999, 2000, 2001 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

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 9
3.4 Using `autoconf` to Create `configure` 10
3.5 Using `autoreconf` to Update `configure` Scripts 13

4 Initialization and Output Files 17

4.1 Notices in `configure` 17
4.2 Finding `configure` Input 17
4.3 Outputting Files 18
4.4 Taking Configuration Actions 19
4.5 Creating Configuration Files 20
4.6 Substitutions in Makefiles 20
4.6.1 Preset Output Variables 21
4.6.2 Installation Directory Variables 22
4.6.3 Build Directories 24
4.6.4 Automatic Remaking 25
4.7 Configuration Header Files 26
4.7.1 Configuration Header Templates 27
4.7.2 Using `autoheader` to Create '`config.h.in`' 27
4.7.3 Autoheader Macros 29
4.8 Running Arbitrary Configuration Commands 30
4.9 Creating Configuration Links 30
4.10 Configuring Other Packages in Subdirectories 31
4.11 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 35
5.2.1 Particular Program Checks 35
5.2.2 Generic Program and File Checks 36
5.3 Files 38
5.4 Library Files 38
5.5 Library Functions 39
5.5.1 Portability of Classical Functions 39
5.5.2 Particular Function Checks 39
5.5.3 Generic Function Checks 43
5.6 Header Files 45
5.6.1 Particular Header Checks 45
5.6.2 Generic Header Checks 48
5.7 Declarations 49
5.7.1 Particular Declaration Checks 49
5.7.2 Generic Declaration Checks 49
5.8 Structures 50
5.8.1 Particular Structure Checks 50
5.8.2 Generic Structure Checks 51
5.9 Types 52
5.9.1 Particular Type Checks 52
5.9.2 Generic Type Checks 52
5.10 Compilers and Preprocessors 53
5.10.1 Generic Compiler Characteristics 53
5.10.2 C Compiler Characteristics 53
5.10.3 C++ Compiler Characteristics 56
5.10.4 Fortran 77 Compiler Characteristics 57
5.11 System Services 60
5.12 UNIX Variants 61

6 Writing Tests 63

6.1 Examining Declarations 63
6.2 Examining Syntax 64
6.3 Examining Libraries 64
6.4 Checking Run Time Behavior 65
6.4.1 Running Test Programs 66
6.4.2 Guidelines for Test Programs 66
6.4.3 Test Functions 67
6.5 Systemology 67
6.6 Multiple Cases 68
6.7 Language Choice 68

7 Results of Tests 71
7.1 Defining C Preprocessor Symbols 71
7.2 Setting Output Variables 72
7.3 Caching Results 73
7.3.1 Cache Variable Names 74
7.3.2 Cache Files 75
7.3.3 Cache Checkpointing 76
7.4 Printing Messages 76

8 Programming in M4 79
8.1 M4 Quotation 79
8.1.1 Active Characters 79
8.1.2 One Macro Call 80
8.1.3 Quotation and Nested Macros 80
8.1.4 Quadrigraphs 82
8.1.5 Quotation Rule Of Thumb 82
8.2 Programming in M4sugar 84
8.2.1 Redefined M4 Macros 84
8.2.2 Forbidden Patterns 84

9 Writing Autoconf Macros 85
9.1 Macro Definitions 85
9.2 Macro Names 85
9.3 Reporting Messages 86
9.4 Dependencies Between Macros 87
9.4.1 Prerequisite Macros 87
9.4.2 Suggested Ordering 88
9.5 Obsoleting Macros 89
9.6 Coding Style 89

10 Portable Shell Programming 93
10.1 Shellology 93
10.2 Here-Documents 94
10.3 File Descriptors 95
10.4 File System Conventions 96
10.5 Shell Substitutions 98
10.6 Assignments 100
10.7 Special Shell Variables 101
10.8 Limitations of Shell Builtins 102
10.9 Limitations of Usual Tools 107
10.10 Limitations of Make 112

11 Manual Configuration 115
11.1 Specifying the System Type 115
11.2 Getting the Canonical System Type 116
11.3 Using the System Type 117

12 Site Configuration 119

12.1 Working With External Software 119
12.2 Choosing Package Options 120
12.3 Making Your Help Strings Look Pretty 121
12.4 Configuring Site Details 121
12.5 Transforming Program Names When Installing 122
12.5.1 Transformation Options 122
12.5.2 Transformation Examples 122
12.5.3 Transformation Rules 123
12.6 Setting Site Defaults 124

13 Running `configure` Scripts 127

13.1 Basic Installation 127
13.2 Compilers and Options 128
13.3 Compiling For Multiple Architectures 128
13.4 Installation Names 128
13.5 Optional Features 128
13.6 Specifying the System Type 129
13.7 Sharing Defaults 129
13.8 Environment Variables 129
13.9 `configure` Invocation 130

14 Recreating a Configuration 131

15 Obsolete Constructs 133

15.1 Obsolete '`config.status`' Invocation 133
15.2 '`acconfig.h`' 134
15.3 Using `autoupdate` to Modernize '`configure.ac`' 134
15.4 Obsolete Macros 135
15.5 Upgrading From Version 1 145
15.5.1 Changed File Names 145
15.5.2 Changed Makefiles 145
15.5.3 Changed Macros 145
15.5.4 Changed Results 146
15.5.5 Changed Macro Writing 147
15.6 Upgrading From Version 2.13 147
15.6.1 Changed Quotation 147
15.6.2 New Macros 148

16 Questions About Autoconf 151

16.1 Distributing `configure` Scripts 151
16.2 Why Require GNU M4? 151
16.3 How Can I Bootstrap? 151
16.4 Why Not Imake? 152

17 History of Autoconf 155

17.1 Genesis 155
17.2 Exodus 155
17.3 Leviticus 156
17.4 Numbers 156
17.5 Deuteronomy 157

Environment Variable Index 159

Output Variable Index 161

Preprocessor Symbol Index 163

Autoconf Macro Index 165

M4 Macro Index 169

Concept Index 171

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 [Preprocessor Symbol Index], page 163).

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 145, for information about upgrading from version 1. See Chapter 17 [History], page 155, for the story of Autoconf's development. See Chapter 16 [Questions], page 151, 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://sources.redhat.com/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 runtime 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`').

Automake may require that additional tools be present on the *developer's* machine. For example, the `Makefile.in` that the developer works with may not be portable (e.g. it might use special features of your compiler to automatically generate dependency information). Running `make dist`, however, produces a '`hello-1.0.tar.gz`' package (or whatever the program/version is) with a `Makefile.in` that will work on any system.

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 to 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, one in each subdirectory of the package (see Section 4.6 [Makefile Substitutions], page 20);
- optionally, a C header file, the name of which is configurable, containing `#define` directives (see Section 4.7 [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 131);
- 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 75);
- 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 63, 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 93. 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 produce 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 4.3 [Output], page 18). 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 4.7 [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 9, 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 is only installed if you already have Perl installed. 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.

'--autoconf-dir=*dir*'
'-A *dir*' Override the location where the installed Autoconf data files are looked for. You can also set the AC_MACRODIR environment variable to a directory; this option overrides the environment variable.

This option is rarely needed and dangerous; it is only used when one plays with different versions of Autoconf simultaneously.

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 85, 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.

'`--autoconf-dir=`*dir*'
'`-A` *dir*' Override the location where the installed Autoconf data files are looked for. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

This option is rarely needed and dangerous; it is only used when one plays with different versions of Autoconf simultaneously.

'`--localdir=`*dir*'
'`-l` *dir*' Look for the package file '`aclocal.m4`' in directory *dir* instead of in the current directory.

'`--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 86, 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. `autoconf -W` *category* will actually behave as if you had run:

```
autoconf --warnings=syntax,$WARNINGS,category
```

If you want to disable `autoconf`'s defaults and `WARNINGS`, but (for example) enable the warnings about obsolete constructs, you would use '`-W none,obsolete`'.

`autoconf` 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_TRY_RUN([true])])

AC_DEFUN([OUTER],
[INNER])

AC_INIT
OUTER
```

you get:

```
$ autoconf -Wcross
configure.ac:8: warning: AC_TRY_RUN called without default \
to allow cross compiling
$ autoconf -Wcross,error
configure.ac:8: error: AC_TRY_RUN called without default \
to allow cross compiling
acgeneral.m4:3044: AC_TRY_RUN is expanded from...
configure.ac:2: INNER is expanded from...
configure.ac:5: OUTER is expanded from...
configure.ac:8: the top level
```

'`--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 below 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`'.

The *format* of '`--trace`' 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.

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:$l";'
$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 its 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

If you have a lot of Autoconf-generated configure scripts, the autoreconf program can save you some work. It runs autoconf (and autoheader, where appropriate) repeatedly to remake the Autoconf configure scripts and configuration header templates in the directory tree rooted at the current directory. By default, it only remakes those files that are older than their 'configure.ac' or (if present) 'aclocal.m4'. Since autoheader does not change the timestamp of its output file if the file wouldn't be changing, this is not necessarily the minimum amount of work. If you install a new version of Autoconf, you can make autoreconf remake *all* of the files by giving it the '--force' option.

If you give autoreconf the '--autoconf-dir=*dir*' or '--localdir=*dir*' options, it passes them down to autoconf and autoheader (with relative paths adjusted properly).

autoreconf does not support having, in the same directory tree, both directories that are parts of a larger package (sharing 'aclocal.m4' and 'acconfig.h') and directories that are independent packages (each with their own 'aclocal.m4' and 'acconfig.h'). It assumes

that they are all part of the same package if you use '`--localdir`', or that each directory is a separate package if you don't use it. This restriction may be removed in the future.

See Section 4.6.4 [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`' Copy missing auxiliary files. This option is similar to the option `--add-missing` in `automake`.

'`--symlink`'
'`-s`' Instead of copying missing auxiliary files, install symbolic links.

'`--localdir=`*dir*'
'`-l` *dir*' Have `autoconf` and `autoheader` look for the package files '`aclocal.m4`' and (`autoheader` only) '`acconfig.h`' (but not '*file*`.top`' and '*file*`.bot`') in directory *dir* instead of in the directory containing each '`configure.ac`'.

'`--autoconf-dir=`*dir*'
'`-A` *dir*' Override the location where the installed Autoconf data files are looked for. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

This option is rarely needed and dangerous; it is only used when one plays with different versions of Autoconf simultaneously.

'`--m4dir=`*dir*'
'`-M` *dir*' Specify location of additional macro files ('`m4`' by default).

Additionally, the following options are recognized and passed to `automake`:

'`--cygnus`'
Assume program is part of Cygnus-style tree.

'`--foreign`'
Set strictness to foreign.

'`--gnits`' Set strictness to gnits.

'`--gnu`' Set strictness to gnu.

'`--include-deps`'
Include generated dependencies in '`Makefile.in`'.

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 initialization and the creation of output files.

4.1 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 do not create `configure`. For example:

```
AC_PREREQ(2.52)
```

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
```

4.2 Finding `configure` Input

Every `configure` script must call `AC_INIT` before doing anything else. The only other required macro is `AC_OUTPUT` (see Section 4.3 [Output], page 18).

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

Process any command-line arguments and perform various initializations and verifications. Set the name of the *package* and its *version*. The optional argument *bug-report-address* should be the email to which users should send bug reports.

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 130, 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`'.

4.3 Outputting Files

Every Autoconf-generated `configure` script must finish by calling `AC_OUTPUT`. It is the macro that generates '`config.status`', which will create the '`Makefile`'s and any other files resulting from configuration. The only other required macro is `AC_INIT` (see Section 4.2 [Input], page 17).

AC_OUTPUT Macro

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

'`config.status`' will take all the configuration actions: all the output files (see Section 4.5 [Configuration Files], page 20, macro `AC_CONFIG_FILES`), header files (see Section 4.7 [Configuration Headers], page 26, macro `AC_CONFIG_HEADERS`), commands (see Section 4.8 [Configuration Commands], page 30, macro `AC_CONFIG_COMMANDS`), links (see Section 4.9 [Configuration Links], page 30, macro `AC_CONFIG_LINKS`), subdirectories to configure (see Section 4.10 [Subdirectories], page 31, macro `AC_CONFIG_SUBDIRS`) are honored.

Historically, the usage of `AC_OUTPUT` was somewhat different. See Section 15.4 [Obsolete Macros], page 135, for a description of the arguments that `AC_OUTPUT` used to support.

If you run `make` on 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 variable `MAKE`, define output variable `SET_MAKE` to be empty. Otherwise, define `SET_MAKE` to contain '`MAKE=make`'. Calls `AC_SUBST` for `SET_MAKE`.

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

```
@SET_MAKE@
```

4.4 Taking 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.

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.

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*.

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

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 macro `AC_CONFIG_FILES` and `AC_CONFIG_HEADERS` use specials *tag*s: they may have the form '*output*' or '*output*:*inputs*'. The file *output* is instantiated from its templates, *inputs* if specified, 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 131, 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.

4.5 Creating Configuration Files

Be sure to read the previous section, Section 4.4 [Configuration Actions], page 19.

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 4.4 [Configuration Actions], page 19. See Section 4.6 [Makefile Substitutions], page 20, for more information on using output variables. See Section 7.2 [Setting Output Variables], page 72, 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.

4.6 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 72, 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.

4.6.1 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 [Output Variable Index], page 161, for a complete list of output variables. See Section 4.6.2 [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 72, `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 4.7 [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 72, 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.

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 77 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 77 features.

srcdir Variable

The directory that contains the source code for that `Makefile`.

top_srcdir Variable

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

4.6.2 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 on purpose 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 to her late desires.

'`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 but in Makefiles. For instance, instead of trying to evaluate `datadir` in '`configure`' and hardcoding 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 '`.sh`', and uses this Makefile snippet:

```
.sh:
        rm -f $@ $@.tmp
        sed 's,@datadir\@,$(pkgdatadir),g' $< >$@.tmp
        chmod +x $@.tmp
        mv $@.tmp $@
```

Three things 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 probably the variables you use, such as '`$(pkgdatadir)`', will contain some.

4.6.3 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 in 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
```

4.6.4 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 10.9 [Limitations of Usual Tools], page 107), 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 4.3 [Output], page 18, for more information about `AC_OUTPUT`.

See Chapter 14 [config.status Invocation], page 131, for more examples of handling configuration-related dependencies.

4.7 Configuration Header Files

When a package tests more than a few 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 4.4 [Configuration Actions], page 19. 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 some changes in 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 4.4 [Configuration Actions], page 19, for more details on *header*.

4.7.1 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
```

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.

Since it is a tedious task to keep a template header up to date, you may use autoheader to generate it, see Section 4.7.2 [autoheader Invocation], page 27.

4.7.2 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 using its

third argument for each symbol (see Section 7.1 [Defining Symbols], page 71). An additional constraint is that the first argument of `AC_DEFINE` must be a literal. Note that all symbols defined by Autoconf's built-in 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.

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

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

'`--autoconf-dir=`*dir*'
'`-A` *dir*'
: Override the location where the installed Autoconf data files are looked for. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

  This option is rarely needed and dangerous; it is only used when one plays with different versions of Autoconf simultaneously.

'`--localdir=`*dir*'
'`-l` *dir*'
: Look for the package files '`aclocal.m4`' and '`acconfig.h`' (but not '*file*`.top`' and '*file*`.bot`') in directory *dir* instead of in the current directory.

'`--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*

4.7.3 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 71. 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 the 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.

4.8 Running Arbitrary Configuration Commands

You execute arbitrary commands either 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 135, 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 to the *tag*. Since typically the *cmds* create a file, *tag* should naturally be the name of that file. This macro is one of the instantiating macros, see Section 4.4 [Configuration Actions], page 19.

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`'. A typical use is computing values derived from variables built during the execution of `configure`:

```
AC_CONFIG_COMMANDS_PRE(
[LTLIBOBJS=`echo $LIBOBJS | sed 's/\.o/\.lo/g'`
AC_SUBST(LTLIBOBJS)])
```

AC_CONFIG_COMMANDS_POST (*cmds*) Macro

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

4.9 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 another directory than its sources.

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. 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 4.4 [Configuration Actions], page 19.

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.

4.10 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, no error is reported, so a `configure` script can configure whichever parts of a large source tree are present. 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.ac`' 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 (e.g., to adjust a relative path for the cache file or source directory). 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.

4.11 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 besides '`/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*; otherwise leave the prefix specified in '`Makefile.in`' unchanged. 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 63).

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 73).

Some of these macros set output variables. See Section 4.6 [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 71, 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 *argument*s 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 45).

Most generic macros provide the following default set of includes:

```
#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 Autoconf will automatically 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 `mawk`, `gawk`, `nawk`, and `awk`, in that order, and set output variable `AWK` to the first one that is found. It tries `mawk` first because that is reported to be the fastest implementation.

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 4.3 [Output], page 18). 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 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 63). 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 72, `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* exists on the `PATH`. If it 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 116). 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 116). 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 77 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 6.7 [Language Choice], page 68), try to ensure that the C, C++, or Fortran 77 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 '`-l`*library*' to `LIBS` and define '`HAVE_LIB`*library*' (in all capitals). This macro is intended to support building of `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_TRY_LINK_FUNC` first with no libraries, then for each library listed in *search-libs*.

Add '`-l`*library*' 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 Classical 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

`unlink` The POSIX spec says that `unlink` causes the given files to be removed only after there are no more open file handles for it. Not all OS's support this behaviour 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.

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 is available and works (unlike the one on Solaris 2.4), define `HAVE_FNMATCH`.

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.

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. 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 ('`getloadavg.c`') of '`getloadavg`', 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 `NLIST_STRUCT`.
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

If `getpgrp` takes no argument (the POSIX.1 version), define `GETPGRP_VOID`. Otherwise, it is the BSD version, which takes a process ID as an argument. 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` and `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` works correctly ('`malloc (0)`' returns a valid pointer), define `HAVE_MALLOC`.

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_MKTIME Macro

If the `mktime` function is not available, or does not work correctly, require an `AC_LIBOBJ` replacement for '`mktime`'.

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_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.1 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 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`. If its implementation correctly returns a `char *`, define `HAVE_WORKING_STRERROR_R`. On at least DEC UNIX 4.0[A-D] and HP-UX B.10.20, `strerror_r` returns `int`. Actually, this tests only whether it returns a scalar or an array, but that should be enough. This is used by the common '`error.c`'.

AC_FUNC_STRFTIME Macro

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

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.)

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 63).

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 6.7 [Language Choice], page 68, for more information about selecting the language for checks.)

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

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.

Use the first three of the following macros to specify a function to be replaced, and the last one (`AC_REPLACE_FUNCS`) to check for and replace the function if needed.

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` 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_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 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` et al. 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_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 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 put 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.1, 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.1 compatible, then instead of including it, define the POSIX.1 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.1 systems. If there is no '`unistd.h`', it is definitely not a POSIX.1 system. However, some non-POSIX.1 systems do have '`unistd.h`'.

The way to check if the system supports POSIX.1 is:

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

#ifdef _POSIX_VERSION
/* Code for POSIX.1 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` or `struct timezone` 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
```

5.6.2 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 63).

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

If the system header file *header-file* is usable, 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.

The meaning of "usable" depends upon the content of *includes*:

if *includes* is empty

check whether

header-file

can be *preprocessed* without error.

if *include* is set

Check whether

includes
`#include <`*header-file*`>`

can be *compiled* without error. You may use `AC_CHECK_HEADER` (and `AC_CHECK_HEADERS`) to check whether two headers are compatible.

You may pass any kind of dummy content for *includes*, such as a single space, a comment, to check whether *header-file* compiles with success.

AC_CHECK_HEADERS (*header-file...*, [*action-if-found*] , [*action-if-not-found*] , [*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.

Be sure to read the documentation of `AC_CHECK_HEADER` to understand the influence of *includes*.

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 49) or, for more complex tests, you may use `AC_TRY_COMPILE` (see Section 6.2 [Examining Syntax], page 64).

5.7.1 Particular Declaration Checks

The following macros check for certain declarations.

AC_DECL_SYS_SIGLIST Macro

Define `SYS_SIGLIST_DECLARED` if the variable `sys_siglist` is declared in a system header file, either '`signal.h`' or '`unistd.h`'.

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*]) 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*]) 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
char *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 macro (see Section 5.8.2 [Generic Structures], page 51) or, for more complex tests, you may use `AC_TRY_COMPILE` (see Section 6.2 [Examining Syntax], page 64).

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*]) 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*]) 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).

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_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*]) 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*]) 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 135.

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, they fail. If the executables can't be run, and cross-compilation is not enabled, they fail too. See Chapter 11 [Manual Configuration], page 115, for more on support for cross compiling.

5.10.1 Generic Compiler Characteristics

AC_CHECK_SIZEOF (*type*, [*unused*] , [*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*, make 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.

5.10.2 C Compiler Characteristics

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 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_CC_STDC Macro

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.

If you use this macro, you should check after calling it 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.

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.

If the current language is C (see Section 6.7 [Language Choice], page 68), many of the specific test macros use the value of `CPP` indirectly by calling `AC_TRY_CPP`, `AC_CHECK_HEADER`, `AC_EGREP_HEADER`, or `AC_EGREP_CPP`.

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.

The following macros check for C compiler or machine architecture features. To check for characteristics not listed here, use `AC_TRY_COMPILE` (see Section 6.2 [Examining Syntax], page 64) or `AC_TRY_RUN` (see Section 6.4 [Run Time], page 65)

AC_C_BIGENDIAN Macro

If words are stored with the most significant byte first (like Motorola and SPARC, but not Intel and VAX, CPUs), define `WORDS_BIGENDIAN`.

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_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 defined `__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 the `long double` type, define `HAVE_LONG_DOUBLE`. Some C compilers that do not define `__STDC__` do support the `long double` type; some compilers that define `__STDC__` do not support `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

Check to see if function prototypes are understood by the compiler. If so, define '`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 *));
```

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.

5.10.3 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.

If the current language is C++ (see Section 6.7 [Language Choice], page 68), many of the specific test macros use the value of `CXXCPP` indirectly by calling `AC_TRY_CPP`, `AC_CHECK_HEADER`, `AC_EGREP_HEADER`, or `AC_EGREP_CPP`.

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++.

5.10.4 Fortran 77 Compiler Characteristics

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 cf77 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 -O2`' for `g77` (or '`-O2`' where `g77` does not accept '`-g`'). Otherwise, set `FFLAGS` to '`-g`' for all other Fortran 77 compilers.

AC_PROG_F77_C_O Macro

Test if the Fortran 77 compiler accepts the options '`-c`' and '`-o`' simultaneously, and define `F77_NO_MINUS_C_MINUS_O` if it does not.

The following macros check for Fortran 77 compiler characteristics. To check for characteristics not listed here, use `AC_TRY_COMPILE` (see Section 6.2 [Examining Syntax], page 64) or `AC_TRY_RUN` (see Section 6.4 [Run Time], page 65), making sure to first set the current language to Fortran 77 `AC_LANG(Fortran 77)` (see Section 6.7 [Language Choice], page 68).

AC_F77_LIBRARY_LDFLAGS Macro

Determine the linker flags (e.g. '`-L`' and '`-l`') for the *Fortran 77 intrinsic and run-time libraries* that are required to successfully link a Fortran 77 program or shared library. The output variable `FLIBS` is set to these flags.

This macro is intended to be used in those situations when it is necessary to mix, e.g. C++ and Fortran 77 source code into 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 77 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 77 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, the macro `AC_F77_LIBRARY_LDFLAGS` was created to determine these Fortran 77 libraries.

The macro `AC_F77_DUMMY_MAIN` or `AC_F77_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

With many compilers, the Fortran libraries detected by `AC_F77_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 e.g. `MAIN__` to run the user's program. The `AC_F77_DUMMY_MAIN` or `AC_F77_MAIN` macro figures out how to deal with this interaction.

When using Fortran for purely numerical functions (no I/O, etcetera), users often prefer to provide their 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` detects whether any such routine is *required* for linking, and what its name is; the shell variable `F77_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` 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
```

Note that `AC_F77_DUMMY_MAIN` is called automatically from `AC_F77_WRAPPERS`; there is generally no need to call it explicitly unless one wants to change the default actions.

AC_F77_MAIN Macro

As discussed above for `AC_F77_DUMMY_MAIN`, many Fortran libraries allow you to provide an entry point called e.g. `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` macro detects whether it is *possible* to utilize such an alternate main function, and defines `F77_MAIN` to the name of the function. (If no alternate main function name is found, `F77_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` instead of `main`.

AC_F77_WRAPPERS Macro

Defines C macros `F77_FUNC(name,NAME)` and `F77_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 77 compiler.

Fortran 77 is case-insensitive, and in order to achieve this the Fortran 77 compiler converts all identifiers into a canonical case and format. To call a Fortran 77 subroutine from C or to write a C function that is callable from Fortran 77, the C program must explicitly use identifiers in the format expected by the Fortran 77 compiler. In order to do this, one simply wraps all C identifiers in one of the macros provided by `AC_F77_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*).

Although Autoconf tries to be intelligent about detecting the name-mangling scheme of the Fortran 77 compiler, there may be Fortran 77 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` 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 77 identifier contains an underscore (e.g. `foo_bar`), you should use `F77_FUNC_` instead of `F77_FUNC` (with the same arguments). This is because some Fortran 77 compilers mangle names differently if they contain an underscore.

AC_F77_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 77 linker (see also `AC_F77_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++.

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 them 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)`.

[1] large-file support, `http://www.sas.com/standards/large.file/x_open.20Mar96.html`.

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 POSIX termios headers and functions are available on the system. If so, set the shell variable `am_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_ISC_POSIX Macro

If on a POSIXized ISC UNIX, define `_POSIX_SOURCE` and add '`-posix`' (for the GNU C compiler) or '`-Xp`' (for other C compilers) to output variable `CC`. This allows the use of POSIX facilities. Must be called after `AC_PROG_CC` and before any other macros that run the C compiler.

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.

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 C compiler system. They do not cache the results of their tests for future use (see Section 7.3 [Caching Results], page 73), 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 C 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 85, for how to do that.

6.1 Examining Declarations

The macro `AC_TRY_CPP` is used to check whether particular header files exist. You can check for one at a time, or more than one if you need several header files to all exist for some purpose.

AC_TRY_CPP (*includes*, [*action-if-true*] , [*action-if-false*]) Macro

includes is C or C++ `#include` statements and declarations, on which shell variable, back quote, and backslash substitutions are performed. (Actually, it can be any C program, but other statements are probably not useful.) If the preprocessor produces no error messages while processing it, run shell commands *action-if-true*. Otherwise run shell commands *action-if-false*.

This macro uses `CPPFLAGS`, but not `CFLAGS`, because '`-g`', '`-O`', etc. are not valid options to many C preprocessors.

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.

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 `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

To check for C preprocessor symbols, either defined by header files or predefined by the C preprocessor, use `AC_EGREP_CPP`. Here is an example of the latter:

```
AC_EGREP_CPP(yes,
[#ifdef _AIX
  yes
#endif
], is_aix=yes, is_aix=no)
```

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 `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

This macro calls `AC_PROG_CPP` or `AC_PROG_CXXCPP` (depending on which language is current, see Section 6.7 [Language Choice], page 68), if it hasn't been called already.

6.2 Examining Syntax

To check for a syntax feature of the C, C++ or Fortran 77 compiler, such as whether it recognizes a certain keyword, use `AC_TRY_COMPILE` to try to compile a small program that uses that feature. You can also use it to check for structures and structure members that are not present on all systems.

AC_TRY_COMPILE (*includes, function-body,* [*action-if-found*] , [*action-if-not-found*]) Macro

Create a C, C++ or Fortran 77 test program (depending on which language is current, see Section 6.7 [Language Choice], page 68), to see whether a function whose body consists of *function-body* can be compiled.

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 77). This macro also uses `CFLAGS` or `CXXFLAGS` if either C or C++ is the currently selected language, as well as `CPPFLAGS`, when compiling. If Fortran 77 is the currently selected language then `FFLAGS` will be used when compiling.

If the file compiles successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*.

This macro does not try to link; use `AC_TRY_LINK` if you need to do that (see Section 6.3 [Examining Libraries], page 64).

6.3 Examining Libraries

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 runtime behavior if needed. On the other hand, it can be slower than scanning the libraries once.

A few systems have linkers that do not return a failure exit status when there are unresolved functions in the link. This bug makes the configuration scripts produced by Autoconf unusable on those systems. However, some of them can be given options that make the exit status correct. This is a problem that Autoconf does not currently handle automatically. If users encounter this problem, they might be able to solve it by setting `LDFLAGS` in the environment to pass whatever options the linker needs (for example, '`-Wl,-dn`' on MIPS RISC/OS).

`AC_TRY_LINK` 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_TRY_LINK (*includes*, *function-body*, [*action-if-found*] , [*action-if-not-found*]) Macro

Depending on the current language (see Section 6.7 [Language Choice], page 68), create a test program to see whether a function whose body consists of *function-body* can be compiled and linked.

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 77). This macro also uses `CFLAGS` or `CXXFLAGS` if either C or C++ is the currently selected language, as well as `CPPFLAGS`, when compiling. If Fortran 77 is the currently selected language then `FFLAGS` will be used when compiling. However, both `LDFLAGS` and `LIBS` will be used during linking in all cases.

If the file compiles and links successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*.

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

Depending on the current language (see Section 6.7 [Language Choice], page 68), create a test program to see whether a program whose body consists of a prototype of and a call to *function* can be compiled and linked.

If the file compiles and links successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*.

6.4 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_TRY_RUN`. Avoid running

test programs if possible, because this prevents people from configuring your package for cross-compiling.

6.4.1 Running Test Programs

Use the following macro if you need to test run-time behavior of the system while configuring.

AC_TRY_RUN (*program*, [*action-if-true*] , [*action-if-false*] , [*action-if-cross-compiling*]) Macro

program is the text of a C program, on which shell variable and back quote substitutions are performed. If it 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 exit status of the program is available in the shell variable '`$?`'. This macro uses `CFLAGS` or `CXXFLAGS`, `CPPFLAGS`, `LDFLAGS`, and `LIBS` when compiling.

If the C 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.

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_TRY_RUN`. `autoconf` prints a warning message when creating `configure` each time it encounters a call to `AC_TRY_RUN` 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 115). Alternatively, set up a test results cache file with the correct values for the host system (see Section 7.3 [Caching Results], page 73).

To provide a default for calls of `AC_TRY_RUN` that are embedded in other macros, including a few of the ones that come with Autoconf, you can call `AC_PROG_CC` before running them. Then, if the shell variable `cross_compiling` is set to '`yes`', use an alternate method to get the results instead of calling the macros.

6.4.2 Guidelines for Test Programs

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.

6.4.3 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
char *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, make 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`).

6.5 Systemology

This section aims at presenting some systems and pointers to documentation. It may help you addressing particular problems reported by users.

QNX 4.25 QNX is a realtime operating system running on Intel architecture meant to be scalable from the small embedded systems to hundred processor supercomputer. It claims to be POSIX certified. More information is available on the QNX home page[1] , including the QNX man pages[2] .

[1] QNX home page, `www.qnx.com`.

[2] QNX man pages, `http://support.qnx.com/support/docs/qnx4/`.

6.6 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.

```
AC_MSG_CHECKING([how to get file system type])
fstype=no
# The order of these tests is important.
AC_TRY_CPP([#include <sys/statvfs.h>
#include <sys/fstyp.h>],
           [AC_DEFINE(FSTYPE_STATVFS) fstype=SVR4])
if test $fstype = no; then
  AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/fstyp.h>],
             [AC_DEFINE(FSTYPE_USG_STATFS) fstype=SVR3])
fi
if test $fstype = no; then
  AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/vmount.h>],
             [AC_DEFINE(FSTYPE_AIX_STATFS) fstype=AIX])
fi
# (more cases omitted here)
AC_MSG_RESULT([$fstype])
```

6.7 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.

'`C++`' Do compilation tests using `CXX` and `CXXCPP` and use extension '`.C`' for test programs.

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

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_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 87) with an argument of either `AC_PROG_CPP` or `AC_PROG_CXXCPP`, depending on which language is current.

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 4.7 [Configuration Headers], page 26, for more information about this kind of output.

AC_DEFINE (*variable*, [*value*] , [*description*]) Macro

Define C preprocessor variable *variable*. If *value* is given, set *variable* to that value (verbatim), otherwise set it to 1. *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")
```

AC_DEFINE_UNQUOTED (*variable*, [*value*] , [*description*]) 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 the 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 4.6.1 [Preset Output Variables], page 21, 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 different 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 4.6.4 [Automatic Remaking], page 25, and see Chapter 14 [config.status Invocation], page 131), then the configuration can be inconsistent, composed of results depending upon two different compilers.

Such variables are named *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.
- *variable* is kept in the cache including if it was not specified on the '`./configure`' command line. 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.
- *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 131) 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 75). 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 74, 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 85.

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 4.10 [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 115).

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,, (exit 1); exit)
AM_PATH_GTKMM(0.9.5,, (exit 1); exit)
... 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 86.

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_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 brokenness of 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 are the special characters 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", although 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 unneeded 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 possible quotings: 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 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|@`' '`$`'

'`@%:@`' '`#`'

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 '`[^@<:@]`', if properly quoted, will appear as '`[^[]`' in the `configure` script.

8.1.5 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 See Section 8.1.4 [Quadrigraphs], page 82, 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 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".

8.2.1 Redefined M4 Macros

All the M4 native macros are moved in the '`m4_`' pseudo-namespace, e.g., M4sugar renames `define` as `m4_define` etc. There is one exception: `dnl` kept its original name, and no `m4_dnl` is defined.

M4sugar redefines some M4 macros, and made them slightly incompatible with their native equivalent.

m4_defn (*macro*) Macro
Contrary to the M4 builtin, this macro fails if *macro* is not defined. See `m4_undefine`.

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_popdef (*macro*) Macro
Contrary to the M4 builtin, this macro fails if *macro* is not defined. See `m4_undefine`.

8.2.2 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 over quotation).

M4sugar forbids all the tokens matching '`^m4_`' and '`^dnl$`'.

m4_pattern_forbid (*pattern*) Macro
Declare 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 vy '`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.

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 `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 87).

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])
# --------------------------------------
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 74, for more information on them).

The first word of the name after 'AC_' usually tells the category of 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 76.

AC_DIAGNOSE (*category, message*) Macro

Report *message* as a warning (or as an error if requested by the user) if it falls into the *category*. You are encouraged to use standard categories, which currently include:

'all'	messages that don't fall into one of the following category. 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 "$body_temparature_in_celsius" -gt "38" &&
  dance_floor=occupied])
AC_DEFUN([NEWTON_JOHN],
[test "$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 substitute 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 79.

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 85), 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_TRY_RUN 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:

```
patsubst([$1], [$"])
```

use

```
patsubst([$1], [$""])
```

so that Emacsen do not open a 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 unneeded. 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 85, 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 `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.
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. 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 Sequent DYNIX) will ignore the line, because they interpret '`#! /`' as a 4-byte magic number.

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 10.9 [Limitations of Usual Tools], page 107.

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.

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 workaround 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:

```
cat ${FOO=`bar`}
```

- beware that single builtin substitutions are not performed by a sub shell, hence their effect applies to the current shell! See Section 10.5 [Shell Substitutions], page 98, 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.

`/usr/xpg4/bin/sh` on Solaris
: The POSIX-compliant Bourne shell on a Solaris system is `/usr/xpg4/bin/sh` and is part of an extra optional package. There is no extra charge for this package, but it is also not part of a minimal OS install and therefore some folks may not have it.

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 user's existant 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. Users can generally speed things up by using a faster 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 the shell can take 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 several times the same file descriptor, 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 subshells. This might result in undesired 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...

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 10.8 [Limitations of Builtins], page 102).

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 `$ac_path_separator` instead (or

the `PATH_SEPARATOR` output variable). `autoconf` 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[1] 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_HEADER(config.h)
  AC_CONFIG_FILES([source.c foo.bar])
  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_HEADER(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 behaviour 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.

[1] doschk, `ftp://ftp.gnu.org/gnu/non-gnu/doschk/doschk-1.1.tar.gz`.

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 backquoted 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 backquoted expressions (pfew!).

`$@`

One of the most famous shell-portability issues is related to '`"$@"`': when there are no positional arguments, it is supposed to be equivalent to nothing. But some shells, for instance under Digital Unix 4.0 and 5.0, will then replace it with an empty argument. To be portable, use '`${1+"$@"}`'.

`${`*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 as Ash 0.2, trying to optimize, does not fork a sub-shell 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 -n "`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)
```

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 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 98, items '`${`*var*`:-`*value*`}`' and '`${`*var*`=`*value*`}`' for the rationale.

10.7 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 10.8 [Limitations of Builtins], page 102) and a fallback value is needed. We list these values below.

`CDPATH` When this variable is set `cd` is verbose, so idioms such as '`abs=`cd $rel && pwd``' break because `abs` receives the path twice.

Setting `CDPATH` to the empty value is not enough for most shells. A simple colon is enough except for `zsh`, which prefers a leading dot:

```
zsh-3.1.6 % mkdir foo && (CDPATH=: cd foo)
/tmp/foo
zsh-3.1.6 % (CDPATH=:. cd foo)
/tmp/foo
zsh-3.1.6 % (CDPATH=.: cd foo)
zsh-3.1.6 %
```

(of course we could just `unset CDPATH`, since it also behaves properly if set to the empty string).

Life wouldn't be so much fun if `bash` and `zsh` had the same behavior:

```
bash-2.02 % (CDPATH=:. cd foo)
bash-2.02 % (CDPATH=.: cd foo)
/tmp/foo
```

Therefore, a portable solution to neutralize '`CDPATH`' is

```
CDPATH=${ZSH_VERSION+.}:
```

Note that since `zsh` supports `unset`, you may unset '`CDPATH`' using '`:`' as a fallback, see Section 10.8 [Limitations of Builtins], page 102.

`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.

`LANG`
`LC_ALL`
`LC_TIME`
`LC_CTYPE`
`LANGUAGE`
`LC_COLLATE`
`LC_NUMERIC`
`LC_MESSAGES`

These must not be set unconditionally because not all systems understand e.g. '`LANG=C`' (notably SCO). Fixing `LC_MESSAGES` prevents Solaris `sh` from translating var values in `set`! Non-C `LC_CTYPE` values break the ctype check. Fixing `LC_COLLATE` makes scripts more portable in some cases. For example, it causes the regular expression '`[a-z]`' to match only lower-case letters on ASCII platforms. However, '`[a-z]`' does not work in general even when `LC_COLLATE` is fixed; for example, it does not work for EBCDIC platforms. For maximum portability, you should use regular expressions like '`[abcdefghijklmnopqrstuvwxyz]`' that list characters explicitly instead of relying on ranges.

If one of these variables is set, you should try to unset it, using '`C`' as a fall back value. see Section 10.8 [Limitations of Builtins], page 102, builtin `unset`, for more details.

`NULLCMD` When executing the command '`>foo`', `zsh` executes '`$NULLCMD >foo`'. The Bourne shell considers `NULLCMD` is '`:`', 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.

`status` This variable is an alias to '`$?`' for `zsh` (at least 3.1.6), hence read-only. Do not use it.

`PATH_SEPARATOR`

On DJGPP systems, the `PATH_SEPARATOR` 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 autoconf to detect the regular DOS path separator '`;`', so it can be safely substituted in files that may not support '`;`' as path separator. So either unset this variable or set it to '`;`'.

`RANDOM` Many shells provide `RANDOM`, a variable that returns a different integer when 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`.

10.8 Limitations of Shell Builtins

No, no, we are serious: some shells do have limitations! :)

You should always keep in mind that any built-in 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.

`!` You can't use `!`, you'll have to rewrite your code.

`break` The use of '`break 2`', etcetera, is safe.

`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
```

`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 4.6.1 [Preset Output Variables], page 21, `ECHO_N` etc. for a means to simulate '`-c`'.

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, 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 status. 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 *environment variable* a shell 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 DU 4.0, forget to `export` the environment variables they receive. As a result, two variables are coexisting: 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 98), use:

```
for arg in ${1+"$@"}; do
  echo "$arg"
done
```

`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
```

`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 parameters, 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
```

`shift` Not only is `shift`ing 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.

`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.

`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 frequent 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.2 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 `CDPATH` or `LANG`, 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 CDPATH || CDPATH=:
```

See Section 10.7 [Special Shell Variables], page 101, for some neutralizing values. Also, see Section 10.8 [Limitations of Builtins], page 102, documentation of `export`, for the case of environment variables.

10.9 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`'.

`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` 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`).

`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 `dirname`, but it is reasonably easy to emulate, e.g.:

```
dir=`expr "x$file" : 'x\(.*\)/[^/]*' \|
          '.'      : '.'
```

But there are a few subtilities, e.g., under UN*X, should '`//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.2 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.
>
> I have heard rumors that this special treatment of '`//`' may be dropped in future versions of POSIX, but for now it's still the standard.

`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 the latest POSIX draft, 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, they are not supported on Solaris.

The POSIX.2-1992 standard is ambiguous as to whether ‘`expr a : b`’ (and ‘`expr 'a' : '\(b\)'`’) output ‘`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`’.

You may believe that one means 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 work around consists in testing `expr` and use a variable set to `expr` or to `false` according to the result.

`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.

`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 (eg., IRIX 6.5 and Solaris 2.5.1). Anyway, Stardent Vistra SVR4 `grep` lacks ‘`-e`’... Instead, use alternation and `egrep`.

`ln` Don't rely on `ln` having a '`-f`' option. Symbolic links are not available on old systems, use '`ln`' as a fall back.

For versions of the DJGPP before 2.04, `ln` emulates soft links for 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.

`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.

Moving directories across mount points is not portable, use `cp` and `rm`.

`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 not portable. Anchors ('`^`' and '`$`') inside groups are not portable.

Nested groups are extremely portable, but there is at least one `sed` (System V/68 Base Operating System R3V7.1) that does not support it.

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 helping enforcing indenting 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".

`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 remind 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` 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.

10.10 Limitations of Make

Make itself suffers a great number of limitations, only a few of which being listed here. First of all, remember that since commands are executed by the shell, all its weaknesses are inherited...

Leading underscore in macro names

Some Make 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
```

`VPATH` Don't use it! For instance any assignment to `VPATH` causes Sun `make` to only execute the first set of double-colon rules.

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[1]:

'`--build=`*build-type*'
: the type of system on which the package is being configured and compiled.

'`--host=`*host-type*'
: the type of system on which the package will run.

'`--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.

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. Also, 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. Note, however, that it won't guess the build-system type, since this

[1] 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
```

may require running test programs. 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, 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.

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 4.3 [Output], page 18, 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`.

For temporary backward-compatibility, when '`--host`' is specified by '`--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.

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`.

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 4.9 [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 in '`configure.ac`', use:

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

You can also use the host system type to find cross-compilation tools. See Section 5.2.2 [Generic Programs], page 36, 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 it in '`[`' and '`]`' in order to produce the leading spaces.

You should format your *help-string* with the macro `AC_HELP_STRING` (see Section 12.3 [Pretty Help Strings], page 121).

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*] , Macro
[*action-if-not-given*])

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 119).

You should format your *help-string* with the macro `AC_HELP_STRING` (see Section 12.3 [Pretty Help Strings], page 121).

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 119) and `AC_ARG_ENABLE` (see Section 12.2 [Package Options], page 120) 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 `AC_HELP_STRING` macro.

AC_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 119) or `AC_ARG_ENABLE` (see Section 12.2 [Package Options], page 120). The following example will make this clearer.

```
AC_DEFUN(TEST_MACRO,
[AC_ARG_WITH(foo,
             AC_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 `AC_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 `AC_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],
             AC_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 that differs from the host type (specified with '`--host`'), 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':

```
transform = @program_transform_name@
install: all
        $(INSTALL_PROGRAM) myprog $(bindir)/`echo myprog | \
                                               sed '$(transform)'`

uninstall:
        rm -f $(bindir)/`echo myprog | sed '$(transform)'`
```

If you have more than one program to install, you can do it in a loop:

```
PROGRAMS = cp ls rm
install:
        for p in $(PROGRAMS); do \
          $(INSTALL_PROGRAM) $$p $(bindir)/`echo $$p | \
                                             sed '$(transform)'`; \
        done

uninstall:
        for p in $(PROGRAMS); do \
          rm -f $(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 75). 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, so 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 variables by setting them in the environment. You can do that on the command line like this:

```
./configure CC=c89 CFLAGS=-O2 LIBS=-lposix
```

See Section 13.8 [Environment Variables], page 129, 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 host the package will run on. Usually `configure` can figure that out, but if it prints a message saying it cannot guess the host 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 host 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*'. In this case, you should also specify the build platform with '`--build=`*type*', because, in this case, it may not be possible to guess the build platform (it sometimes involves compiling and running simple test programs, and this can't be done if the compiler is a cross compiler).

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 Environment 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 *option*s 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.

'`--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 4.4 [Configuration Actions], page 19, 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 4.6.4 [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 '`/bin/sh`'.

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 4.6.4 [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 133, for details.

15 Obsolete Constructs

Autoconf changes, and throughout the years some constructs are obsoleted. Most of the changes involve the macros, but 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 131, 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 131, 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 4.7.3 [Autoheader Macros], page 29), but in older releases a file, 'acconfig.h', contained the list of needed templates. autoheader copies 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 jurasik 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] --'
```

Use 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 85, for a description of the new scheme. Although the old names still work (see Section 15.4 [Obsolete Macros], page 135, 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.

'`--autoconf-dir=`*dir*'
'`-A` *dir*' Override the location where the installed Autoconf data files are looked for. You can also set the `AC_MACRODIR` environment variable to a directory; this option overrides the environment variable.

This option is rarely needed and dangerous; it is only used when one plays with different versions of Autoconf simultaneously.

'`--localdir=`*dir*'
'`-l` *dir*' Look for the package file '`aclocal.m4`' in directory *dir* instead of in the current directory.

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 85, for a description of the new naming scheme. Below, 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 116, 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. Second, 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 52, 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([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_LINK` (see Section 6.3 [Examining Libraries], page 64), 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 76).

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_YYTEXT Macro

Does nothing, now integrated in AC_PROG_LEX.

AC_DIR_HEADER Macro

Like calling AC_FUNC_CLOSEDIR_VOID andAC_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 (Sequent UNIX), 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_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 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, while `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_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_STRUCT_ST_BLKSIZE`

AC_ST_BLOCKS Macro
`AC_STRUCT_ST_BLOCKS`

AC_ST_RDEV Macro
`AC_STRUCT_ST_RDEV`

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`

AC_TEST_PROGRAM Macro
`AC_TRY_RUN`

AC_TIMEZONE Macro
`AC_STRUCT_TIMEZONE`

AC_TIME_WITH_SYS_TIME Macro
`AC_HEADER_TIME`

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 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 4.7.3 [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 135, 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 134.

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 it prints 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 76. Those macros work best in conjunction with cache variables. See Section 7.3 [Caching Results], page 73.

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 74.

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 72.

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 85.

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 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 145) 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 name of these macros! Therefore, when m4 find in 'aclocal.m4' something like 'AC_DEFUN(AM_TYPE_PTRDIFF_T, ...)', 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 by 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
$
```

16 Questions About Autoconf

Several questions about Autoconf come up occasionally. Here some of them are addressed.

16.1 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.

16.2 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:

```
builtin
indir
patsubst
__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.

16.3 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).

16.4 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 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...

17.1 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.

17.2 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.

17.3 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.

17.4 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.

17.5 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.)

Environment Variable Index

This is an alphabetical list of the environment variables that Autoconf checks.

A

AC_MACRODIR 9, 10, 14, 28, 135

C

CDPATH 101
CONFIG_COMMANDS 133
CONFIG_FILES 133
CONFIG_HEADERS 133
CONFIG_LINKS 133
CONFIG_SHELL 132
CONFIG_SITE 124
CONFIG_STATUS 132

I

IFS 101

L

LANG 102
LANGUAGE 102
LC_ALL 102
LC_COLLATE 102
LC_CTYPE 102
LC_MESSAGES 102
LC_NUMERIC 102
LC_TIME 102

N

NULLCMD 102

P

PATH_SEPARATOR 102

R

RANDOM 102

S

SIMPLE_BACKUP_SUFFIX 134
status 102

W

WARNINGS 11, 28

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 72, for more information on how this is done.

A

ALLOCA 39
AWK 35

B

bindir 23
build 116
build_alias 116
build_cpu 116
build_os 116
build_vendor 116

C

CC 54, 56, 60, 61
CFLAGS 21, 54
configure_input 21
CPP 54
CPPFLAGS 21
cross_compiling 115
CXX 56
CXXCPP 57
CXXFLAGS 21, 56

D

datadir 23
DEFS 21

E

ECHO_C 22
ECHO_N 22
ECHO_T 22
exec_prefix 23
EXEEXT 53, 137

F

F77 57
FFLAGS 22, 57
FLIBS 57

G

GETGROUPS_LIBS 41
GETLOADAVG_LIBS 41

H

host 116
host_alias 116
host_cpu 116
host_os 116
host_vendor 116

I

includedir 23
infodir 23
INSTALL 35
INSTALL_DATA 35
INSTALL_PROGRAM 35
INSTALL_SCRIPT 35

K

KMEM_GROUP 41

L

LDFLAGS 22
LEX 35
LEX_OUTPUT_ROOT 35
LEXLIB 35
libdir 23
libexecdir 23
LIBOBJS 41, 42, 44, 45, 51
LIBS 22, 142, 144
LN_S 36
localstatedir 23

M

mandir 23

N

NEED_SETGID 41

O

OBJEXT 53, 140
oldincludedir 23

P

POW_LIB 43
prefix 23
program_transform_name 122

R

RANLIB 36

S

sbindir 23
SET_MAKE 19
sharedstatedir 23
srcdir 22
subdirs 31
sysconfdir 23

T

target 116
target_alias 116
target_cpu 116
target_os 116
target_vendor 116
top_srcdir 22

X

X_CFLAGS 60
X_EXTRA_LIBS 60
X_LIBS 60
X_PRE_LIBS 60

Y

YACC 36

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__ 55
_ALL_SOURCE 61
_FILE_OFFSET_BITS 60
_LARGE_FILES 60
_LARGEFILE_SOURCE 41
_MINIX 61
_POSIX_1_SOURCE 61
_POSIX_SOURCE 61
_POSIX_VERSION 47

C

C_ALLOCA 39
C_GETLOADAVG 41
CLOSEDIR_VOID 40
const 55

D

DGUX 41
DIRENT 137

F

F77_DUMMY_MAIN 58
F77_FUNC 59
F77_FUNC_ 59
F77_MAIN 58
F77_NO_MINUS_C_MINUS_O 57

G

GETGROUPS_T 52
GETLODAVG_PRIVILEGED 41
GETPGRP_VOID 42
gid_t 52
GWINSZ_IN_SYS_IOCTL 48

H

HAVE_ALLOCA_H 39
HAVE_CONFIG_H 26
HAVE_DECL_*symbol* 50
HAVE_DIRENT_H 45
HAVE_DOPRNT 43
HAVE_*function* 44
HAVE_GETMNTENT 41
HAVE_*header* 49
HAVE_LONG_DOUBLE 55
HAVE_LONG_FILE_NAMES 61
HAVE_LSTAT_EMPTY_STRING_BUG 43
HAVE_MMAP 42
HAVE_NDIR_H 45
HAVE_OBSTACK 42
HAVE_RESTARTABLE_SYSCALLS 143
HAVE_ST_BLKSIZE 50
HAVE_ST_BLOCKS 51
HAVE_ST_RDEV 51
HAVE_STAT_EMPTY_STRING_BUG 43
HAVE_STRCOLL 43
HAVE_STRERROR_R 43
HAVE_STRFTIME 43
HAVE_STRINGIZE 56
HAVE_STRUCT_STAT_ST_BLKSIZE 50
HAVE_STRUCT_STAT_ST_BLOCKS 51
HAVE_STRUCT_STAT_ST_RDEV 51
HAVE_SYS_DIR_H 45
HAVE_SYS_NDIR_H 45
HAVE_SYS_WAIT_H 47
HAVE_TM_ZONE 51
HAVE_TZNAME 51
HAVE_UTIME_NULL 43
HAVE_VFORK_H 41
HAVE_VPRINTF 43
HAVE_WAIT3 138
HAVE_WORKING_FORK 41
HAVE_WORKING_STRERROR_R 43
HAVE_WORKING_VFORK 41

I

inline 55
INT_16_BITS 139

L

LONG_64_BITS 140
LSTAT_FOLLOWS_SLASHED_SYMLINK 42

M

MAJOR_IN_MKDEV 46
MAJOR_IN_SYSMACROS 46
mode_t 52

N

NDIR 137
NEED_MEMORY_H 140
NEED_SETGID 41
NLIST_NAME_UNION 41
NLIST_STRUCT 41
NO_MINUS_C_MINUS_O 54

O

off_t 52

P

PARAMS 56
pid_t 52
PROTOTYPES 56

R

RETSIGTYPE 52

S

SELECT_TYPE_ARG1 42
SELECT_TYPE_ARG234 42
SELECT_TYPE_ARG5 42
SETPGRP_VOID 42
SETVBUF_REVERSED 43
size_t 52
STDC_HEADERS 46
SVR4 41
SYS_SIGLIST_DECLARED 49
SYSDIR 137
SYSNDIR 137

T

TIME_WITH_SYS_TIME 48
TM_IN_SYS_TIME 51

U

uid_t 52
UMAX 41
UMAX4_3 41
USG 143

V

vfork 41
volatile 55

W

WORDS_BIGENDIAN 55

X

X_DISPLAY_MISSING 60

Y

YYTEXT_POINTER 35

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 29
AH_TEMPLATE 29
AH_TOP 29
AH_VERBATIM 29
AIX 61
ALLOCA 135
ARG_ARRAY 135
ARG_ENABLE 120
ARG_PROGRAM 122
ARG_VAR 73
ARG_WITH 119
AU_DEFUN 89

B

BEFORE 88
BOTTOM 29

C

C_BIGENDIAN 55
C_CHAR_UNSIGNED 55
C_CONST 55
C_CROSS 135
C_INLINE 55
C_LONG_DOUBLE 55
C_PROTOTYPES 56
C_STRINGIZE 56
C_VOLATILE 55
CACHE_CHECK 74
CACHE_LOAD 76
CACHE_SAVE 76
CACHE_VAL 73
CANONICAL_BUILD 116
CANONICAL_HOST 116
CANONICAL_SYSTEM 135
CANONICAL_TARGET 116
CHAR_UNSIGNED 136
CHECK_DECL 49
CHECK_DECLS 50
CHECK_FILE 38
CHECK_FILES 38
CHECK_FUNC 44
CHECK_FUNCS 44
CHECK_HEADER 48
CHECK_HEADERS 49
CHECK_LIB 38
CHECK_MEMBER 51
CHECK_MEMBERS 51
CHECK_PROG 37
CHECK_PROGS 37
CHECK_SIZEOF 53
CHECK_TOOL 37
CHECK_TOOLS 37
CHECK_TYPE 52, 136
CHECK_TYPES 53
CHECKING 136
COMPILE_CHECK 136
CONFIG_AUX_DIR 18
CONFIG_COMMANDS 30
CONFIG_FILES 20
CONFIG_HEADERS 26
CONFIG_LINKS 30
CONFIG_SRCDIR 18
CONFIG_SUBDIRS 31
CONST 136
COPYRIGHT 17
CROSS_CHECK 137
CYGWIN 137

D

DECL_SYS_SIGLIST 49
DECL_YYTEXT 137
DEFINE 71
DEFINE_UNQUOTED 71
DEFUN 85, 89
DIAGNOSE 86
DIR_HEADER 137
DYNIX_SEQ 137

E

EGREP_CPP 64
EGREP_HEADER 63
EMXOS2 137
ENABLE 121
ERROR 137
EXEEXT 137

F

F77_DUMMY_MAIN 58
F77_FUNC 59
F77_LIBRARY_LDFLAGS 57
F77_MAIN 58
F77_WRAPPERS 59
FATAL 87
FIND_X 137
FIND_XTRA 138
FUNC_ALLOCA 39
FUNC_CHECK 138
FUNC_CHOWN 40
FUNC_CLOSEDIR_VOID 40
FUNC_ERROR_AT_LINE 40
FUNC_FNMATCH 40
FUNC_FORK 41
FUNC_FSEEKO 41
FUNC_GETGROUPS 41
FUNC_GETLOADAVG 41
FUNC_GETMNTENT 41
FUNC_GETPGRP 42
FUNC_LSTAT 43
FUNC_LSTAT_FOLLOWS_SLASHED_SYMLINK 42
FUNC_MALLOC 42
FUNC_MEMCMP 42
FUNC_MKTIME 42
FUNC_MMAP 42
FUNC_OBSTACK 42
FUNC_SELECT_ARGTYPES 42
FUNC_SETPGRP 42
FUNC_SETVBUF_REVERSED 43
FUNC_STAT 43
FUNC_STRCOLL 43
FUNC_STRERROR_R 43
FUNC_STRFTIME 43
FUNC_STRTOD 43
FUNC_UTIME_NULL 43
FUNC_VPRINTF 43
FUNC_WAIT3 138

G

GCC_TRADITIONAL 138
GETGROUPS_T 138
GETLOADAVG 138

H

HAVE_FUNCS 138
HAVE_HEADERS 138
HAVE_LIBRARY 138
HAVE_POUNDBANG 138
HEADER_CHECK 138
HEADER_DIRENT 45
HEADER_EGREP 138
HEADER_MAJOR 46
HEADER_STAT 46
HEADER_STDC 46
HEADER_SYS_WAIT 47
HEADER_TIME 48
HEADER_TIOCGWINSZ 48
HELP_STRING 121

I

INIT 17, 139
INLINE 139
INT_16_BITS 139
IRIX_SUN 139
ISC_POSIX 61

L

LANG_C 139
LANG_CPLUSPLUS 139
LANG_FORTRAN77 139
LANG_POP 69
LANG_PUSH 69
LANG_RESTORE 139
LANG_SAVE 139
LIBOBJ 44
LIBSOURCE 44
LIBSOURCES 45
LINK_FILES 139
LN_S 140
LONG_64_BITS 140
LONG_DOUBLE 140
LONG_FILE_NAMES 140

M

MAJOR_HEADER 140
MEMORY_H 140
MINGW32 140
MINIX 61
MINUS_C_MINUS_O 140
MMAP 140
MODE_T 140
MSG_CHECKING 77
MSG_ERROR 77
MSG_NOTICE 77
MSG_RESULT 77
MSG_WARN 77

O

OBJEXT 140
OBSOLETE 140
OFF_T 141
OUTPUT 18, 141
OUTPUT_COMMANDS 141
OUTPUT_COMMANDS_POST 30
OUTPUT_COMMANDS_PRE 30

P

PATH_PROG 37
PATH_PROGS 38
PATH_TOOL 38
PATH_X 60
PATH_XTRA 60
PID_T 141
PREFIX 141
PREFIX_DEFAULT 32
PREFIX_PROGRAM 32
PREREQ 17
PROG_AWK 35
PROG_CC 54
PROG_CC_C_O 54
PROG_CC_STDC 54
PROG_CPP 54
PROG_CXX 56
PROG_CXXCPP 57
PROG_F77_C_O 57
PROG_FORTRAN 57
PROG_GCC_TRADITIONAL 56
PROG_INSTALL 35
PROG_LEX 35
PROG_LN_S 36
PROG_MAKE_SET 19
PROG_RANLIB 36
PROG_YACC 36
PROGRAM_CHECK 142
PROGRAM_EGREP 142
PROGRAM_PATH 142
PROGRAMS_CHECK 141
PROGRAMS_PATH 141

R

REMOTE_TAPE 142
REPLACE_FUNCS 45
REQUIRE 87
REQUIRE_CPP 69
RESTARTABLE_SYSCALLS 142
RETSIGTYPE 142
REVISION 17
RSH 142

S

SCO_INTL 142
SEARCH_LIBS 39
SET_MAKE 142
SETVBUF_REVERSED 142
SIZE_T 142
SIZEOF_TYPE 142
ST_BLKSIZE 143
ST_BLOCKS 143
ST_RDEV 143
STAT_MACROS_BROKEN 46, 142
STDC_HEADERS 142
STRCOLL 142
STRUCT_ST_BLKSIZE 50
STRUCT_ST_BLOCKS 51
STRUCT_ST_RDEV 51
STRUCT_TIMEZONE 51
STRUCT_TM 51
SUBST 72
SUBST_FILE 72

SYS_INTERPRETER 60
SYS_LARGEFILE 60
SYS_LONG_FILE_NAMES 61
SYS_POSIX_TERMIOS 61
SYS_RESTARTABLE_SYSCALLS 143
SYS_SIGLIST_DECLARED 143

T

TEMPLATE 29
TEST_CPP 143
TEST_PROGRAM 143
TIME_WITH_SYS_TIME 143
TIMEZONE 143
TOP 29
TRY_COMPILE 64
TRY_CPP 63
TRY_LINK 65
TRY_LINK_FUNC 65
TRY_RUN 66
TYPE_GETGROUPS 52
TYPE_MODE_T 52
TYPE_OFF_T 52
TYPE_PID_T 52
TYPE_SIGNAL 52
TYPE_SIZE_T 52
TYPE_UID_T 52

U

UID_T 143
UNISTD_H 143
USG 143
UTIME_NULL 144

V

VALIDATE_CACHED_SYSTEM_TUPLE 144
VERBATIM 29
VERBOSE 144
VFORK 144
VPRINTF 144

W

WAIT3 144
WARN 144
WARNING 87
WITH 120
WORDS_BIGENDIAN 144

X

XENIX_DIR 144

Y

YYTEXT_POINTER 144

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_`'.

D

`defn` 84

P

`pattern_allow` 84
`pattern_forbid` 84

U

`undefine` 84

Concept Index

This is an alphabetical list of the files, tools, and concepts introduced in this document.

!

! 103

$

$(*commands*) 100
${*var*:-*value*} 98
${*var*=*expanded-value*} 99
${*var*=*literal*} 98

/

`/usr/xpg4/bin/sh` on Solaris 94

:

: 107

@

'`@%:@`' 82
'`@:>@`' 82
'`@<:@`' 82
'`@S|@`' 82

`

`*commands*` 99

"

'`"$@"`' 98

A

'`acconfig.h`' 134
'`aclocal.m4`' 5
Ash 93
`autoconf` 10
`autoheader` 27
Automake 3
`autoreconf` 13
`autoscan` 9
`autoupdate` 134
`awk` 107

B

Back trace 11
Bash 93
`break` 103

C

Cache 73
Cache variable 74
Cache, enabling 130
`case` 103
`cat` 108
`cmp` 108
Command Substitution 99
'`config.h`' 26
'`config.h.bot`' 134
'`config.h.in`' 27
'`config.h.top`' 134
`config.status` 131
Configuration Header 26
Configuration Header Template 27
`configure` 5, 127
'`configure.ac`' 5
'`configure.in`' 5
Copyright Notice 17
`cp` 108

D

Declaration, checking 49
`diff` 109
`dirname` 109
`dnl` 85, 89

E

`echo` 103
`egrep` 109
Endianness 55
`exit` 103
`export` 104
`expr` 109, 110
`expr` ('|') 109

F

`false` ... 104
File, checking ... 38
`for` ... 104
Function, checking ... 39

G

`grep` ... 110

H

Header, checking ... 45

I

`if` ... 104
`ifnames` ... 9
Includes, default ... 33
Instantiation ... 18

L

Language ... 68
Library, checking ... 38
Libtool ... 4
Links ... 30
`ln` ... 111

M

M4sugar ... 84
Macro invocation stack ... 11
Messages, from `autoconf` ... 86
Messages, from `configure` ... 76
`mv` ... 111

O

obstack ... 42

P

POSIX termios headers ... 61
Previous Variable ... 72
Programs, checking ... 35

Q

QNX 4.25 ... 67
quadrigraphs ... 82
quotation ... 7, 79

R

Revision ... 17

S

`sed` ... 111
`sed` (‘`t`’) ... 111
`set` ... 105
`shift` ... 105
Structure, checking ... 50
Symbolic links ... 111

T

termios POSIX headers ... 61
`test` ... 105
`touch` ... 112
`trap` ... 106
`true` ... 107

U

`undefined macro: _m4_divert_diversion` ... 148
`unset` ... 107

V

Variable, Precious ... 72
Version ... 17
`VPATH` ... 113

Z

Zsh ... 94