Autoconf

Generating Automatic Configuration Scripts
Edition 1.4, for Autoconf version 1.4
May 1993

by David MacKenzie, Roland McGrath, Noah Friedman

1 Introduction

Autoconf is a tool for producing shell scripts that automatically configure software source code packages to adapt to many kinds of UNIX-like systems. For each software package that Autoconf is used with, it creates a configuration script from a template file that lists the operating system features that the package can use.

The configuration scripts produced by Autoconf normally require no manual user intervention when run; they do not even take an argument specifying the system type. Instead, they test for the presence of each feature that might be needed individually (after printing 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, except for occasional quirks.

After the shell code needed to recognize and respond to an operating system feature has been written, Autoconf allows it to be shared between 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 the configuration scripts can be regenerated automatically to take advantage of the updated code.

Autoconf was developed for configuring packages of small utilities; it might not be able to deduce all of the information needed to configure programs with more specialized needs. Larry Wall's Metaconfig package is similar in purpose to Autoconf, but is more general; the scripts it produces are hairier and 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. They should avoid executing test programs, since test programs compiled with a cross-compiler can not be executed on the host system. Also, they shouldn't do anything that tests features of the host system instead of the target system.

Autoconf imposes some restrictions on the names of macros used with `#ifdef` in C programs (see [Preprocessor Symbol Index], page 39).

Autoconf was written by David MacKenzie, with help from François Pinard, Karl Berry, Richard Pixley, Ian Lance Taylor, and Roland McGrath. It was inspired by Brian Fox's automatic configuration system for BASH, by Larry Wall's Metaconfig, and by Richard Stallman, Richard Pixley, and John Gilmore's configuration tools for the GNU compiler and object file utilities.

2 Distributing Autoconf Output

The configuration scripts that Autoconf produces are covered by the GNU General Public License. This is because they consist almost entirely of parts of Autoconf itself, rearranged somewhat, and Autoconf is distributed under the terms of the GPL. However, programs that use Autoconf scripts to configure themselves do not automatically come under the GPL. Distributing an Autoconf configuration script as part of a program is considered to be *mere aggregation* of that work with the Autoconf script. Such programs are not derivative works based on Autoconf; only their configuration scripts are. 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.

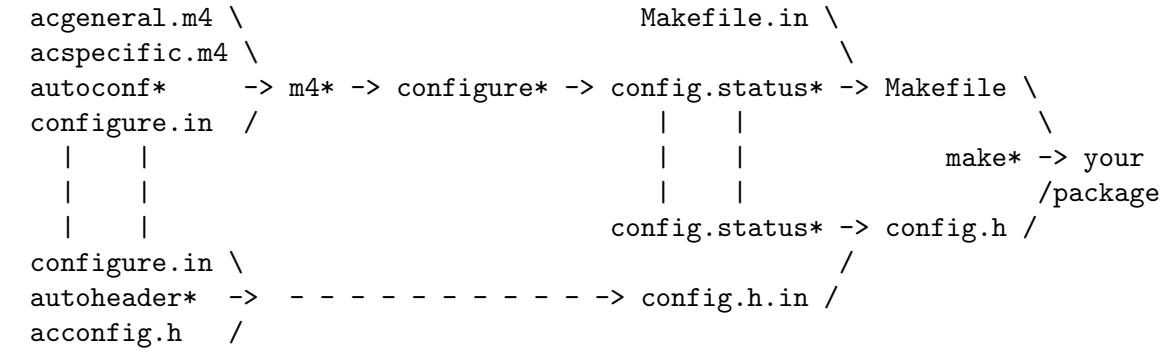
3 Making `configure` Scripts

The configuration scripts that Autoconf produces are by convention called `configure` when they are distributed. When run, they create several files:

- one or more `Makefile` files (one in each subdirectory of the package), from template `Makefile.in` files (see Chapter 7 [Makefiles], page 31);
- optionally, a C header file, the name of which is configurable, containing `#define` statements;
- a shell script called `config.status` that, when run, will recreate the current configuration parameter settings.

To create a `configure` script with Autoconf, you need to write an Autoconf input file and run Autoconf on it to produce the script. And, of course, test the resulting script.

Here is a diagram showing how the files that can be used in configuration are produced:



Executables are suffixed by '`*`', while files appearing twice are linked with lines of '`|`'.

3.1 Writing `configure.in`

To produce a `configure` script for a software package, create a file called `configure.in` 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 4 [Specific Tests], page 9, for their descriptions. For most other features, you can use Autoconf template macros to produce custom checks; see Section 5.2 [General Tests], page 20, for information about them. For especially tricky or specialized features, `configure.in` might need to contain some hand-crafted shell commands. See Chapter 6 [Writing Macros], page 25, for guidelines on writing tests from scratch.

Every `configure.in` must begin with a call to `AC_INIT` and end with a call to `AC_OUTPUT` (see Section 5.1 [Setup], page 19). Other than that, the order in which `configure.in` calls the Autoconf macros is generally not important, except that 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 4 [Specific Tests], page 9).

To encourage consistency, here is a suggested order for calling the Autoconf macros. A few macros need to be called in a different order from the one given here; they are noted in

their individual descriptions (see Chapter 4 [Specific Tests], page 9). (Note that there must not be any space between the macro name and the open parentheses.)

`AC_INIT(`*`file`*`)`
checks for programs
checks for UNIX variants that set `DEFS`
checks for header files
checks for typedefs
checks for functions
checks for structure members
checks for compiler characteristics
checks for operating system services
other checks for UNIX variants
`AC_OUTPUT([`*`file...`*`])`

You can include comments in `configure.in` files by starting them with the `m4` predefined macro `dnl`, which discards text up through the next newline. These comments do not appear in the generated `configure` scripts. For example, it is helpful to begin `configure.in` files with a line like this:

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

See Section 9.1 [Sample configure.in], page 35, for an example of a real `configure.in` script.

3.2 Invoking `autoconf`

To create `configure` from `configure.in`, run the `autoconf` program with no arguments. `autoconf` processes `configure.in` with the `m4` macro processor, using the Autoconf macros. If you give `autoconf` an argument, it reads that file instead of `configure.in` and writes the configuration script to the standard output instead of to `configure`. If you give `autoconf` the argument '`-`', it reads the standard input instead of `configure.in` and writes the configuration script on the standard output.

The Autoconf macros are defined in two or more files. Two of the files are distributed with Autoconf: `acgeneral.m4` (see Chapter 5 [General Purpose Macros], page 19) and `acspecific.m4` (see Chapter 4 [Specific Tests], page 9). `autoconf` also looks for an optional file called `aclocal.m4` both in the directory that contains other installed Autoconf macro files and in the current directory. (If both files exist, it uses both of them.) Those files can contain your site's own locally written Autoconf macro definitions. See Chapter 6 [Writing Macros], page 25, for more information.

You can override the location where `autoconf` looks for the installed macro files by setting the `AC_MACRODIR` environment variable to the appropriate value. You can also use the '`--macrodir`' option (which has higher precedence than the value of `AC_MACRODIR`).

Autoconf requires GNU `m4`. It uses features that some UNIX versions of `m4` do not have; it is also reported to overflow internal limits of some versions of `m4`.

Autoconf does not work well with GNU C library releases before 1.06. The GNU C library contains stubs (which always return an error) for functions that are not available instead of omitting them from the library. As a result, Autoconf scripts are fooled into thinking that those functions are available. This problem does not exist with releases 1.06

and later of the GNU C library, which define C preprocessor macros that the Autoconf macro AC_FUNC_CHECK tests, indicating that certain functions are stubs (see Section 5.2 [General Tests], page 20, for more information on checking for functions).

3.3 Invoking autoheader

You can use the program autoheader to create a template file of C '#define' statements for configure to use. By default, the file that autoheader creates is called config.h.in. autoheader scans configure.in and figures out which C preprocessor symbols it might define. It copies comments and #define and #undef statements from a file called acconfig.h, which comes with Autoconf; it also uses a file called acconfig.h in the current directory, if present. For symbols that AC_HAVE_HEADERS or AC_HAVE_FUNCS define, autoheader generates comments itself rather than copying them from a file, since the possible symbols are effectively limitless.

If you give autoheader an argument, it uses that file instead of configure.in 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.in and writes the header file to the standard output.

You can override the location where autoheader looks for the installed macro and acconfig.h files by setting the AC_MACRODIR environment variable to the appropriate value. You can also use the '--macrodir' option (which has higher precedence than the value of AC_MACRODIR).

4 Specific Tests

These macros test for particular operating system features that packages might need or want to use. If you need to test for a feature that none of these macros check for, you can probably do it by calling one of the general purpose test macros with appropriate arguments (see Section 5.2 [General Tests], page 20).

All of these macros that set `make` variables call `AC_SUBST` on those variables (see Section 5.3 [Setting Variables], page 23, for details about `AC_SUBST`). The phrase "define *name*" is used below as a shorthand to mean either add '`-D`*name*`=1`' to the `make` variable `DEFS`, or put '`#define` *name* `1`' in the configuration header file, depending on whether `AC_CONFIG_HEADER` has been called. See Section 5.3 [Setting Variables], page 23, for more information.

Within each section below, the macros are listed in alphabetical order. The macros are generally named for the `make` variables or C preprocessor macros that they define; those names are based largely on what existing GNU programs use. These macros are defined in the file `acspecific.m4`.

4.1 Alternative Programs

The following macros check for the presence or behavior of particular programs:

`AC_DECLARE_YYTEXT`
: Define `DECLARE_YYTEXT` to declare `yytext` appropriately, depending on whether `lex` or `flex` is being used. This macro calls `AC_PROG_CPP` and `AC_PROG_LEX` if they haven't been called already.

`AC_LN_S`
: If '`ln -s`' works on the current filesystem (the O.S. and filesystem support symbolic links), set shell and `make` variable `LN_S` to '`ln -s`', otherwise set it to '`ln`'.

`AC_MINUS_C_MINUS_O`
: If the C compiler does not accept the '`-c`' and '`-o`' options simultaneously, define `NO_MINUS_C_MINUS_O`.

`AC_PROG_YACC`
: If `bison` is found, set `make` variable `YACC` to '`bison -y`'. Otherwise, if `byacc` is found, set `YACC` to '`byacc`'. Otherwise set `YACC` to '`yacc`'.

`AC_PROG_CPP`
: Set shell and `make` variable `CPP` to a command that runs the C preprocessor. If '`$CC -E`' doesn't work, it uses `/lib/cpp`.

  Many of the specific test macros use the value of `CPP` indirectly by calling `AC_TEST_CPP`, `AC_HEADER_CHECK`, `AC_HEADER_EGREP`, or `AC_PROGRAM_EGREP`. Those macros call this macro first if it hasn't been called already. It should be called after `AC_PROG_CC`.

`AC_PROG_LEX`
: If `flex` is found, set `make` variable `LEX` to '`flex`' and `LEXLIB` to '`-lfl`' (or the full pathname of the '`fl`' library, if it is in a standard place). Otherwise set `LEX` to '`lex`' and `LEXLIB` to '`-ll`'.

`AC_PROG_AWK`
: Check for `mawk`, `gawk`, `nawk`, and `awk`, in that order, and set `make` variable `AWK` to the first one that it finds.

`AC_PROG_CC`
: If `gcc` is found, set `make` variable `CC` to '`gcc`', and set shell variable `GCC` to 1 for use by macros such as `AC_GCC_TRADITIONAL`.

`AC_GCC_TRADITIONAL`
: Add '`-traditional`' to `make` variable `CC` if using the GNU C compiler and `ioctl` does not work properly without '`-traditional`'. This macro calls `AC_PROG_CC` and `AC_PROG_CPP` if they haven't been called already.

`AC_PROG_INSTALL`
: Set `make` variable `INSTALL_PROGRAM` to '`install -c`' and the variable `INSTALL_DATA` to '`install -c -m 644`' if `install` is found, otherwise set both to '`cp`'. Screens out the false matches `/etc/install` and `/usr/sbin/install` (shell scripts found on System V).

`AC_PROG_RANLIB`
: Set `make` variable `RANLIB` to '`ranlib`' if `ranlib` is found, otherwise to '`:`' (do nothing).

`AC_RSH` If a remote shell is available, put '`rtapelib.o`' in `make` variable `RTAPELIB`. Otherwise, also do so if `netdb.h` exists (implying the `rexec` function), and in addition define `HAVE_NETDB_H`. If neither a remote shell nor `rexec` is available, define `NO_REMOTE`.

4.2 Header Files

The following macros check for the presence of certain C header files:

`AC_DIR_HEADER`
: If the system has `dirent.h`, define `DIRENT`; otherwise, if it has `sys/ndir.h`, define `SYSNDIR`; otherwise, if it has `sys/dir.h`, define `SYSDIR`; otherwise, if it has `ndir.h`, define `NDIR`. Also, if the directory library header file contains a declaration of the `closedir` function with a `void` return type, define `VOID_CLOSEDIR`. The directory library declarations in the source code should look something like the following:

```
/* unistd.h defines _POSIX_VERSION on POSIX.1 systems.  */
#if defined(DIRENT) || defined(_POSIX_VERSION)
#include <dirent.h>
#define NLENGTH(dirent) (strlen((dirent)->d_name))
#else /* not (DIRENT or _POSIX_VERSION) */
#define dirent direct
#define NLENGTH(dirent) ((dirent)->d_namlen)
#ifdef SYSNDIR
#include <sys/ndir.h>
#endif /* SYSNDIR */
#ifdef SYSDIR
#include <sys/dir.h>
#endif /* SYSDIR */
#ifdef NDIR
#include <ndir.h>
#endif /* NDIR */
#endif /* not (DIRENT or _POSIX_VERSION) */
```

Using the above declarations, the program would declare variables to be 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 `NLENGTH` macro.

`AC_MAJOR_HEADER`

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_MEMORY_H`

Define `NEED_MEMORY_H` if `memcpy`, `memcmp`, etc. are not declared in `string.h` and `memory.h` exists. This macro is obsolete; instead, use `AC_HAVE_HEADERS(memory.h)`. See the example for `AC_STDC_HEADERS`.

`AC_STDC_HEADERS`

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) 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.

To check whether to use the System V/ANSI C string functions and header file, you can put the following in `configure.in`:

```
AC_STDC_HEADERS
AC_HAVE_HEADERS(string.h memory.h)
```

Then, in the code, use a test like this:

```
#if STDC_HEADERS || HAVE_STRING_H
#include <string.h>
/* An ANSI string.h and pre-ANSI memory.h might conflict.  */
#if !STDC_HEADERS && HAVE_MEMORY_H
#include <memory.h>
#endif /* not STDC_HEADERS and HAVE_MEMORY_H */
#define index strchr
#define rindex strrchr
#define bcopy(s, d, n) memcpy ((d), (s), (n))
#define bcmp(s1, s2, n) memcmp ((s1), (s2), (n))
#define bzero(s, n) memset ((s), 0, (n))
#else /* not STDC_HEADERS and not HAVE_STRING_H */
#include <strings.h>
/* memory.h and strings.h conflict on some systems.  */
#endif /* not STDC_HEADERS and not HAVE_STRING_H */
```

This example asssumes that your code uses the BSD style functions. If you use the System V/ANSI C style functions, you will need to replace the macro definitions with ones that go in the other direction.

`AC_UNISTD_H`

Define `HAVE_UNISTD_H` if the system has `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
```

`_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`.

`AC_USG` Define `USG` if the system does not have `strings.h`, `rindex`, `bzero`, etc. This implies that it has `string.h`, `strrchr`, `memset`, etc.

The symbol `USG` is obsolete. Instead of this macro, use `AC_HAVE_HEADERS(string.h)` and use `HAVE_STRING_H` in your code. See the example for `AC_STDC_HEADERS`.

4.3 Typedefs

The following macros check for predefined C types:

`AC_GETGROUPS_T`

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

`AC_MODE_T`
: If `mode_t` is not defined in `sys/types.h`, define `mode_t` to be `int`.

`AC_PID_T`
: If `pid_t` is not defined in `sys/types.h`, define `pid_t` to be `int`.

`AC_RETSIGTYPE`
: 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_SIZE_T`
: If `size_t` is not defined in `sys/types.h`, define `size_t` to be `unsigned`.

`AC_UID_T`
: If `uid_t` is not defined in `sys/types.h`, define `uid_t` to be `int` and `gid_t` to be `int`.

4.4 Library Functions

The following macros check for particular C library functions:

`AC_ALLOCA`
: 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 that fails, it looks for a function in the standard C library. If that fails, it sets the `make` variable `ALLOCA` to '`alloca.o`'. This variable is separate from `LIBOBJS` so multiple programs can share the value of `ALLOCA` without needing to create an actual library.

  If this macro finds `alloca.h`, it defines `HAVE_ALLOCA_H`.

  This macro does not try to get `alloca` from the SVR3 `libPW` or the SVR4 `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. Note that 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. */
#ifdef __GNUC__
#define alloca __builtin_alloca
#else /* not __GNUC__ */
#if HAVE_ALLOCA_H
#include <alloca.h>
#else /* not HAVE_ALLOCA_H */
#ifdef _AIX
 #pragma alloca
#else /* not _AIX */
char *alloca ();
#endif /* not _AIX */
#endif /* not HAVE_ALLOCA_H */
#endif /* not __GNUC__ */
```

`AC_GETLOADAVG`

Check how to get the system load averages. It tries to get the `getloadavg` function from `/usr/lib/libutils.a`, if present (such as on 4.4BSD), or from `/usr/lib/libgetloadavg.a` or `/usr/local/lib/libgetloadavg.a` (such as is commonly installed on AIX systems). Otherwise, it adds '`getloadavg.o`' to the `make` variable `LIBOBJS` and defines `SVR4`, `DGUX`, `UMAX`, or `UMAX4_3` if on those systems. It then checks for `nlist.h`. If it finds it, it defines `NLIST_STRUCT` and checks whether '`struct nlist`' has an '`n_un`' member; if so, it defines `NLIST_NAME_UNION`. Then it determines whether compiling `getloadavg.c` would define the `LDAV_PRIVILEGED`; this indicates whether the program will need to be installed specially for `getloadavg` to work. If so, it defines `GETLOADAVG_PRIVILEGED`. It always defines the `make` variable `NEED_SETGID`; the value is '`true`' if special installation is required, or '`false`' if not. If `NEED_SETGID` is defined to '`true`', the '`make`' variable `KMEM_GROUP` is also defined to be the special group which should own the installed program.

`AC_SETVBUF_REVERSED`

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`. This is the case on System V before release 3.

`AC_STRCOLL`

Check for a proper declaration of the `strcoll` function. This does a bit more than '`AC_HAVE_FUNCS(strcoll)`', because some systems have incorrect definitions of `strcoll`, which should not be used.

`AC_UTIME_NULL`

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

`AC_VFORK` If `vfork.h` is found, define `HAVE_VFORK_H`. If a working `vfork` is not found, define `vfork` to be `fork`. 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.

`AC_VPRINTF`
: If `vprintf` is found, define `HAVE_VPRINTF`. Otherwise, if `_doprnt` is found, define `HAVE_DOPRNT`.

`AC_WAIT3`
: 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`.

4.5 Structures

The following macros check for certain structures or structure members:

`AC_ST_BLKSIZE`
: If `struct stat` contains an `st_blksize` member, define `HAVE_ST_BLKSIZE`.

`AC_ST_BLOCKS`
: If `struct stat` contains an `st_blocks` member, define `HAVE_ST_BLOCKS`. Otherwise, add '`fileblocks.o`' to the `make` variable `LIBOBJS`.

`AC_ST_RDEV`
: If `struct stat` contains an `st_rdev` member, define `HAVE_ST_RDEV`.

`AC_TIME_WITH_SYS_TIME`
: 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.

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

`AC_STRUCT_TM`
: If `time.h` does not define `struct tm`, define `TM_IN_SYS_TIME`, which means that including `sys/time.h` defines `struct tm`.

`AC_TIMEZONE`
: Figure out how to get the current timezone. If `struct tm` has a `tm_zone` member, define `HAVE_TM_ZONE`. Otherwise, if the external array `tzname` is found, define `HAVE_TZNAME`. This macro calls `AC_STRUCT_TM` if it hasn't been called already.

4.6 Compiler Characteristics

The following macros check for C compiler or machine architecture features:

`AC_ARG_ARRAY`
: If the address of an argument to a C function can not be used like the start of an array, define `NO_ARG_ARRAY`. This ability allows a sequence of arguments with the same type to be accessed as if they were an array of values.

`AC_CROSS_CHECK`
: If the C compiler being used does not produce executables that can run on the system where `configure` is being run, set the shell variable `cross_compiling` to 1. This information can be used by `AC_TEST_PROGRAM` to determine whether to take a default action instead of trying to run a test program (see Section 5.2 [General Tests], page 20).

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

`AC_CONST`
: If the C compiler does not fully support the keyword `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.

`AC_INLINE`
: If the C compiler is a version of GCC that supports the keyword `__inline` but not `inline` (such as some NeXT versions), define `inline` to be `__inline`. This macro calls `AC_PROG_CC` if it hasn't been called already.

`AC_INT_16_BITS`
: If the C type `int` is smaller than the type `long`, define `INT_16_BITS`.

`AC_LONG_DOUBLE`
: 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_WORDS_BIGENDIAN`
: If words are stored with the most significant byte first, define `WORDS_BIGENDIAN`.

4.7 System Services

The following macros check for operating system services:

`AC_HAVE_POUNDBANG(`*action-if-exists* [`,` *action-if-not-exists*]`)`
: Prints '`checking if '#!' works in shell scripts`' to the standard output, then creates sample shell scripts to determine whether using lines of the form `#!/bin/csh` have any effect on what shell is invoked to read the script. *action-if-exists* is a list of shell commands to run if #! works; *action-if-not-exists* is a list of shell commands to run otherwise. There are no default actions.

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

`AC_REMOTE_TAPE`
: If BSD tape drive ioctls are available, define `HAVE_SYS_MTIO_H`, and if sockets are available add `rmt` to `make` variable `PROGS`.

`AC_RESTARTABLE_SYSCALLS`
: If the system automatically restarts a system call that is interrupted by a signal, define `HAVE_RESTARTABLE_SYSCALLS`.

4.8 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:

`AC_AIX` 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_DYNIX_SEQ`
: If on DYNIX/ptx (Sequent UNIX), add '`-lseq`' to `make` variable `LIBS`. Allows use of some BSD system calls and `getmntent`.

`AC_IRIX_SUN`
: If on IRIX (Silicon Graphics UNIX), add '`-lsun`' to `make` variable `LIBS`. Needed to get `getmntent`.

`AC_ISC_POSIX`
: If on a POSIXized ISC UNIX, define `_POSIX_SOURCE` and add '`-posix`' (for the GNU C compiler) or '`-Xp`' (for other C compilers) to `make` 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` 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.

`AC_SCO_INTL`
: If on SCO UNIX, add '`-lintl`' to `make` variable `LIBS`. Used to get `strftime`. It must be called before checking for `strftime`.

`AC_XENIX_DIR`
: If on Xenix, define `VOID_CLOSEDIR` and add '`-lx`' to `make` variable `LIBS`. Also, if `sys/ndir.h` is not being used, add '`-ldir`' to `LIBS`. Needed when using the directory reading functions. This macro must be called after `AC_DIR_HEADER`.

5 General Purpose Macros

These macros provide ways for other macros to control the kind of output that Autoconf produces or to check whether various features are available. They all take arguments. When calling these macros, there must not be any blank space between the macro name and the open parentheses.

Arguments to these macros can be more than one line long if they are enclosed within the `m4` quote characters '`[`' and '`]`'.

Within each section below, the macros are listed in alphabetical order. These macros are defined in the file `acgeneral.m4`.

5.1 Controlling Autoconf Setup

The following macros control the kind of output that Autoconf produces.

`AC_CONFIG_HEADER(`*`header-to-create`*`)`

Create a file *header-to-create* containing C preprocessor `#define` statements instead of setting the `DEFS` variable in a `Makefile`. This macro should be called right after `AC_INIT`. Your distribution should contain a file *`header-to-create`*`.in` that looks as you want the final header file to look, including comments, with default values in the `#define` statements. A default value can be to `#undef` the variable instead of to define it to a value, if your code tests for configuration options using `#ifdef` instead of `#if`.

The usual name for the configuration header file is `config.h`. Some GNU library routines contain

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
```

so if you use those routines, you should add '`-DHAVE_CONFIG_H`' to CFLAGS in `Makefile.in` and call your configuration header file `config.h`. If you use `AC_CONFIG_HEADER`, then `AC_OUTPUT` replaces the string '`@DEFS@`' with '`-DHAVE_CONFIG_H`' instead of with the value of `DEFS` (see Section 5.1 [Setup], page 19).

You can use the program `autoheader` to create *`header-to-create`*`.in` (see Section 3.3 [Invoking autoheader], page 7).

`AC_INIT(`*`unique-file-in-source-dir`*`)`

Process the command-line arguments and find the source code directory. *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 (see Chapter 8 [Running configure Scripts], page 33, for more information).

`AC_PREPARE(`*`unique-file-in-source-dir`*`)`

Find the source code directory and set up shell variables necessary for other Autoconf macros to work. *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 (see Chapter 8 [Running configure Scripts], page 33, for more information). `AC_PREPARE` is the last thing done by `AC_INIT`. Use `AC_PREPARE` instead of `AC_INIT` if you want to do argument parsing yourself; never use both.

`AC_OUTPUT([`*`file`*`...])`
: Create output files (typically one or more `Makefile`s) and `config.status`. If `AC_CONFIG_HEADER` has been called, also create the header file that was named as its argument. The argument is a whitespace-separated list of files to create; if it is omitted, no files are created. `AC_OUTPUT` creates each file *`file`* in the list by copying *`file`*`.in`, substituting the variable values that have been selected by calling `AC_SUBST`. It creates the directory that each file is in if it doesn't exist (but not the parents of that directory). A plausible value for the argument to `AC_OUTPUT` is '`Makefile src/Makefile man/Makefile X/Imakefile`'.

5.2 Checking for Kinds of Features

These macros are templates that, when called with actual parameters, check for various kinds of features. Many of these macros handle two cases: what to do if the given condition is met, and what to do if the condition is not met. In some places you you might want to do something if a condition is true but do nothing if it's false, or vice versa. To omit the true case, pass an empty value for the *action-if-found* argument to the macro. To omit the false case, omit the *action-if-not-found* argument to the macro, including the comma before it.

One shell programming construction that you should not use in the action arguments to these macros is '*`var`*`=${`*`var`*`:-`*`value`*`}`'. Old BSD shells, including the Ultrix `sh`, don't understand the colon, and complain and die. If you omit the colon, it works fine: '*`var`*`=${`*`var`*`-`*`value`*`}`'.

See Chapter 6 [Writing Macros], page 25, for more information on how best to use these macros.

`AC_COMPILE_CHECK(`*`echo-text`*`, `*`includes`*`, `*`function-body`*`, `*`action-if-found`* [`, `*`action-if-not-found`*`])`
: Print '`checking for `*`echo-text`*' to the standard output. Then create a test C program to see whether a function whose body consists of *function-body* can be compiled and linked; *includes* is any `#include` statements needed by the code in *function-body*. If the file compiles and links successfully, run shell commands *action-if-found*, otherwise run *action-if-not-found*. To include double quotes in *function-body* or *includes*, quote them with backslashes.

`AC_FUNC_CHECK(`*`function`*`, `*`action-if-found`* [`, `*`action-if-not-found`*`])`
: If *function* is available, run shell commands *action-if-found*, otherwise *action-if-not-found*.

`AC_HAVE_FUNCS(`*`function`*`...)`
: For each given *function* in the whitespace-separated argument list that is available, define `HAVE_`*`function`* (in all caps). See Chapter 4 [Specific Tests], page 9, for a precise definition of "define" as it is used here.

To check whether a particular library exists, you can use the `AC_HAVE_LIBRARY` macro. If you need to check whether a library other than the default C library actually contains a particular function, temporarily change the shell variable `LIBS`, which contains a list of libraries to use when compiling test files. Here is an example that checks whether the function `rint` is present in the math library:

```
LIBS_save="$LIBS"
LIBS="$LIBS -lm"
AC_HAVE_FUNCS(rint)
LIBS="$LIBS_save"
```

Note that the above code does not decide whether to link the program with '`-lm`'.

`AC_HAVE_HEADERS(`*header-file...*`)`

For each given *header-file* in the whitespace-separated argument list that exists, define `HAVE_`*header-file* (in all caps). See Chapter 4 [Specific Tests], page 9, for a precise definition of "define" as it is used here.

`AC_HAVE_LIBRARY(`*library* [`,` *action-if-found* [`,` *action-if-not-found*]])

Print '`checking for` *library*' to the standard output. Then create a test C program to see whether that program can be linked with the specified library. *action-if-found* is a list of shell commands to run if the link succeeds (which means that the library is present); *action-if-not-found* is a list of shell commands to run if the link fails. If *action-if-found* and *action-if-not-found* are not specified, the default action is to add '`-lfoo`' to `LIBS` and define '`HAVE_LIBfoo`' for library '`foo`'. *library* can be written as any of '`foo`', '`-lfoo`', or '`libfoo.a`'. In all of those cases, the compiler is passed '`-lfoo`'.

`AC_HEADER_CHECK(`*header-file*`,` *action-if-found* [`,` *action-if-not-found*])`

If *header-file* exists, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

`AC_HEADER_EGREP(`*pattern*`,` *header-file*`,` *action-if-found* [`,` *action-if-not-found*])`

If the output of running the C preprocessor on *header-file* contains the `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

`AC_PREFIX(`*program*`)`

If the user did not specify an installation prefix on the command line, 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`.

`AC_PROGRAM_CHECK(`*variable*`,` *prog-to-check-for*`,` *value-if-found*`,` *value-if-not-found*`)`

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*. Calls `AC_SUBST` for *variable*.

`AC_PROGRAM_EGREP(`*pattern*`,` *program*`,` *action-if-found* [`,` *action-if-not-found*]`)`

program is the text of a C program, on which shell variable and backquote substitutions are performed. If the output of running the C preprocessor on *program* contains the `egrep` regular expression *pattern*, execute shell commands *action-if-found*, otherwise execute *action-if-not-found*.

`AC_PROGRAMS_CHECK(`*variable*`,` *progs-to-check-for* `[,` *value-if-not-found*`])n`

Check for each program in the whitespace-separated list *progs-to-check-for* exists in `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* will not be changed. Calls `AC_SUBST` for *variable*.

`AC_REPLACE_FUNCS(`*function-name*`...)`

For each given *function-name* in the whitespace-separated argument list that is not in the C library, add '*function-name*`.o`' to the value of the `make` variable `LIBOBJS`.

`AC_TEST_PROGRAM(`*program*`,` *action-if-true* [`,` *action-if-false*`] [,` *action-if-cross-compiling*`])`

program is the text of a C program, on which shell variable and backquote 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*.

If the optional argument *action-if-cross-compiling* is given and 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. Instead, the shell commands *action-if-cross-compiling* are run. If that argument is given, this macro calls `AC_CROSS_CHECK` if it has not already been called (see Section 4.6 [Compiler Characteristics], page 16).

`AC_TEST_CPP(`*includes*`,` *action-if-true* [`,` *action-if-false*]`)`

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

This macro calls `AC_PROG_CPP` if it hasn't been called already.

`AC_WITH(`*package*`,` *action-if-true* [`,` *action-if-false*]`)`

If the user gave `configure` the option '`--with-`*package*', run shell commands *action-if-true*. Otherwise run shell commands *action-if-false*. The name *package*

should consist only of alphanumeric characters and dashes; typical package names are '`gnu-libc`' and '`x`'.

5.3 Setting Variables

These macros help provide ways for other macros to define shell and `make` variables.

`AC_DEFINE(`*variable* [`,` *value*]`)`

Define C preprocessor variable *variable*. If *value* is given, set *variable* to that value, otherwise set it to 1. To use a *value* containing double quotes, protect them with backslashes.

This macro adds to the shell variable `DEFS`. `AC_OUTPUT` later substitutes the values in `DEFS` into the `Makefile.in` file(s), or if `AC_CONFIG_HEADER` has been called, into the header file named as its argument.

`AC_OUTPUT` creates *header-to-create* from *`header-to-create`*`.in` by substituting the correct values in `#define` statements. For example, suppose your `configure.in` calls `AC_CONFIG_HEADER(conf.h)` and `AC_UNISTD_H`. You could have code like this in `conf.h.in`:

```
/* Define as 1 if you have unistd.h.  */
#define HAVE_UNISTD_H 0
```

On systems that have `unistd.h`, `configure` will change the 0 to a 1. On other systems, it will leave the line unchanged. Alternately, if you prefer to use `#ifdef`, your `conf.h.in` could have code like this:

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

On systems that have `unistd.h`, `configure` will change the second line to read '`#define HAVE_UNISTD_H 1`'. On other systems, it will leave the line unchanged.

If *header-to-create* 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.

`AC_DEFINE_UNQUOTED(`*variable* [`,` *value*]`)`

This is just like `AC_DEFINE`, but it does nothing to quote *value* from various shell and `sed` expansions it will undergo. *value* will be used in many different contexts requiring different quoting, and it is up to you to make sure it works right.

`AC_SUBST(`*variable*`)`

Substitute the variable *variable* when creating the output files (typically one or more `Makefile`s). This means replace instances of '`@`*`variable`*`@`', e.g. in `Makefile.in`, with the current value of the shell variable *variable*. If this macro were not called, the value of *variable* would not be set in the output files, even though `configure` had figured out a value for it.

You can set or add to the value of *variable* in the usual shell way. For example, to add '`-ltermcap`' to the value of the variable `LIBS`:

```
LIBS="$LIBS -ltermcap"
```

5.4 Macro Ordering

These macros provide ways for other macros to make sure that they are called in the correct order.

`AC_BEFORE(`*this-macro-name*`, `*called-macro-name*`)`
: Make `m4` print a warning message on 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 contain a call to `AC_PROVIDE` to indicate that it has been called.

  This macro should be used when one macro makes changes that might affect another macro, so that the other macro should probably not be called first. 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])
  ```

  This warns the user if a call to `AC_PROG_CPP` has already occurred when `AC_PROG_CC` is called.

`AC_PROVIDE(`*macro-name*`)`
: Set a flag recording that *macro-name* has been called. The argument should be the name of the macro that is calling `AC_PROVIDE`. An easy way to get it is from the `m4` builtin variable `$0`, like this:

  ```
  AC_PROVIDE([$0])
  ```

`AC_REQUIRE(`*macro-name*`)`
: 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. The body of *macro-name* must contain a call to `AC_PROVIDE` to indicate that it has been called.

  Macros that need some other macro to be called before they are called can use `AC_REQUIRE` to ensure that it has been, in case the person who made `configure.in` forgot or didn't know to do it. `AC_REQUIRE` and `AC_PROVIDE` together can ensure that a macro is only called if it is needed, and only called once. See Section 6.3 [Dependencies Between Macros], page 26, for more information.

6 Writing Macros

If your package needs to test for some feature that none of the macros supplied with Autoconf handles, you'll need to write one or more new Autoconf macros. Here are some suggestions and some of the rationale behind why the existing macros are written the way they are. You can also learn a lot about how to write Autoconf macros by looking at the existing ones. If something goes wrong in one or more of the Autoconf tests, this information can help you understand why they work the way they do and the assumptions behind them, which might help you figure out how to best solve the problem.

If you add macros that you think would be useful to other people, or find problems with the distributed macros, please send electronic mail to `bug-gnu-utils@prep.ai.mit.edu`, so we can consider them for future releases of Autoconf. Please include the Autoconf version number, which you can get by running '`autoconf --version`'.

6.1 Macro Format

Autoconf macros are defined as arguments to the `m4` builtin command `define`. Their overall structure looks like this:

```
define(macro-name, [macro-body])dnl
```

The square brackets here do not indicate optional text: they should literally be present in the macro definition.

All of the Autoconf macros have names starting with '`AC_`' to prevent them from accidentally conflicting with other text. You should prefix your own macro names with some other sequence, such as your initials or an abbreviation for the name of your organization or software package, to ensure that their names don't conflict with the names of present or future Autoconf macros.

The `m4` builtin `dnl` prevents a newline from being inserted in the output where the macro is defined; without it, the generated `configure` script would begin with dozens of blank lines. `dnl` is also used to introduce comments in `m4`; it causes `m4` to discard the rest of the input line.

You should quote the entire macro body with square brackets to avoid macro expansion problems (see Section 6.2 [Quoting], page 25). 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.

6.2 Quoting

Macros that are called by other macros are evaluated by `m4` several times; each evaluation might require another layer of quotes to prevent unwanted expansions of macros or `m4` builtins, such as '`include`' and '`$1`'. Quotes are also required around macro arguments that contain commas, since commas separate the arguments from each other.

Autoconf (in `acgeneral.m4`) changes the `m4` quote characters from the default '`‘`' and '`’`' to '`[`' and '`]`', because many of the macros use '`‘`' and '`’`', mismatched. However, in a few

places the macros need to use brackets. In those places, they use the `m4` builtin command `changequote` to temporarily disable quoting before the code that uses brackets, like this:

```
changequote(,)dnl
```

Then they turn quoting back on again with another call to `changequote`:

```
changequote([,])dnl
```

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`'.

6.3 Dependencies Between Macros

Some Autoconf macros depend on other macros having been called first in order to work correctly, or in some cases, to work at all. 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.

6.3.1 Prerequisite Macros

A macro that you write might need to use values that have previously been computed by other macros. For example, if you write a new macro that uses the C preprocessor, it depends on `AC_PROG_CPP` having been called first to set the shell variable `CPP` (see Section 4.1 [Alternative Programs], page 9).

Rather than forcing the user of the macros to keep track of all of the dependencies between them, you can use the macros `AC_PROVIDE` and `AC_REQUIRE` to do it automatically. See Section 5.4 [Macro Ordering], page 24, for more information on their syntax.

The new macro that runs the C preprocessor should contain, somewhere before `CPP` is used, the statement

```
AC_REQUIRE([AC_PROG_CPP])
```

and the macro `AC_PROG_CPP` should contain the statement (anywhere in its body)

```
AC_PROVIDE([$0])
```

Then, when the new macro is run, it will invoke `AC_PROG_CPP` if and only if `AC_PROG_CPP` has not already been run.

6.3.2 Suggested Ordering

Some macros should be run before another macro if both are called, but neither requires the other to be called. For example, a macro like `AC_AIX` that changes the behavior of the C compiler (see Section 4.8 [UNIX Variants], page 17) should be called before any macros that run the C compiler. Many of these dependencies are noted in the documentation.

Autoconf provides a way to warn users when macros with this kind of dependency appear out of order in a `configure.in` file. The warning occurs when creating `configure` from `configure.in`, not when running `configure`. It is not a fatal error; `configure` is created as usual.

The AC_BEFORE macro causes m4 to print a warning message on the standard error output when a macro is used before another macro which might change its behavior. The macro which should come first should contain a call to AC_BEFORE and the macro which should come later should contain a call to AC_PROVIDE.

For example, AC_AIX contains

```
AC_BEFORE([$0], [AC_COMPILE_CHECK])
```

and AC_COMPILE_CHECK contains

```
AC_PROVIDE([$0])
```

As a result, if AC_AIX is called after AC_COMPILE_CHECK, it will note that AC_COMPILE_CHECK has already been called and print a warning message.

6.4 Checking for Files

If you need to check whether a file other than a C header file exists, use '`test -f` *filename*'. 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.

Do not use '`test -x`', because 4.3BSD does not have it. Use '`test -f`' or '`test -r`' instead.

6.5 Checking for Symbols

If you need to check whether a symbol is defined in a C header file, you can use AC_HEADER_EGREP if the symbol is not a C preprocessor macro (see Section 5.2 [General Tests], page 20), or compile a small test program that includes the file and references the symbol (see Section 6.6 [Test Programs], page 28). Don't directly `grep` for the symbol in the file, because on some systems it might be defined in another header file that the file you are checking '`#include`'s.

However, if you need to check for a particular UNIX variant which is distinguished by having certain text in a certain file, then use `grep` (or `egrep`). But 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.

To check whether the Autoconf macros have already defined a certain C preprocessor symbol, you can use a `case` statement like this:

```
case "$DEFS" in
  *HAVE_FOO*) ;;
  *) LIBOBJS="$LIBOBJS foo.o" ;;
esac
```

Make sure to enclose the variable name you are checking (usually `DEFS`) in double quotes, because otherwise some old versions of `bash` misinterpret the statement.

6.6 Test Programs

Autoconf checks for many features by compiling small test programs. To find out whether a library function is available, Autoconf tries to compile a small program that uses it. This is unlike Larry Wall's Metaconfig, which 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 and flexible 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 `configure` to check aspects of the function's runtime behavior if needed. On the other hand, it is sometimes slower than scanning the libraries.

If you need to check for a condition other than whether some symbol exists on the system or has a certain value, then you can't use `AC_COMPILE_CHECK` (see Section 5.2 [General Tests], page 20). You have to write a test program by hand. You can compile and run it using `AC_TEST_PROGRAM` (see Section 5.2 [General Tests], page 20).

Try to avoid writing test programs if possible, because using them prevents people from configuring your package for cross-compiling. If it's really best that you test for a run-time behavior, try to provide a default "worst case" value to use when cross-compiling makes run-time tests impossible. You do this by passing the optional last argument to `AC_TEST_PROGRAM`.

6.6.1 Guidelines for Test Programs

Test programs 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 the argument to `return` in `main` is ignored. They should not write anything to the standard output.

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_STDC_HEADERS`, then later on in `configure.in` 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 `conftestdata`. The `configure` script cleans up by running '`rm -f conftest*`' after running test programs and if the script is interrupted.

6.6.2 Tricks for Test Programs

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`).

6.7 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 excerpted from the `configure.in` for GNU `find`. It uses the shell variable `fstype` to keep track of whether the remaining cases need to be checked. There are several more cases which are not shown here but follow the same pattern.

```
echo checking how to get filesystem type
# SVR4.
AC_TEST_CPP([#include <sys/statvfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_STATVFS) fstype=1)
if test -z "$fstype"; then
# SVR3.
AC_TEST_CPP([#include <sys/statfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_USG_STATFS) fstype=1)
fi
if test -z "$fstype"; then
# AIX.
AC_TEST_CPP([#include <sys/statfs.h>
#include <sys/vmount.h>], AC_DEFINE(FSTYPE_AIX_STATFS) fstype=1)
fi
```

7 Makefiles

Each subdirectory in a distribution should come with a file `Makefile.in`, from which `configure` will produce a `Makefile` in that directory. Most of the substitutions that `configure` does are simple: for each configuration variable that the package uses, it just replaces occurrences of '`@variable@`' with the value that `configure` has determined for that variable. Any occurrences of '`@variable@`' for variables that `configure` does not know about are passed through unchanged.

There is no point in checking for the correct value to give a variable that is never used. Every variable that the `configure` script might set a value for should appear in a '`@VARIABLE@`' reference in at least one `Makefile.in`. If `AC_CONFIG_HEADER` is called, `configure` replaces '`@DEFS@`' with '`-DHAVE_CONFIG_H`', since the contents of `DEFS` would be redundant.

See Section "Makefile Conventions" in *The GNU Coding Standards*, for more information on what to put in Makefiles. See Section 9.2 [Sample Makefile.in], page 35, for an example of a real `Makefile.in`.

7.1 Predefined Variables

Some `make` variables are predefined by the Autoconf macros. `AC_SUBST` is called for them automatically (see Section 5.3 [Setting Variables], page 23), so in your `Makefile.in` files you can get their values by enclosing their names in '`@`' characters (see Chapter 7 [Makefiles], page 31). The variables that are defined by the general purpose Autoconf macros are:

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

`DEFS` '`-D`' options to pass to the C compiler. Do not include '`@DEFS@`' in your `Makefile.in` files if you are using `AC_CONFIG_HEADER`.

`LIBS` '`-l`' and '`-L`' options to pass to the linker.

`LIBOBJS` Names of object files (ending in `.o`). Set by `AC_REPLACE_FUNCS` (see Section 5.2 [General Tests], page 20).

7.2 Installation Prefixes

If `configure` has figured out a value for the installation prefix, either by the user supplying one on the command line (see Chapter 8 [Running configure Scripts], page 33) or with `AC_PREFIX`, then it substitutes that value in `Makefile`s that it creates. Wherever a `Makefile.in` contains a line like

```
prefix = /usr/local
```

`configure` substitutes the value it figured out. The word '`prefix`' must not be preceded by any other characters on the line.

There can be separate installation prefixes for architecture-specific files and architecture-independent files see Chapter 8 [Running configure Scripts], page 33). `configure` substitutes the word `exec_prefix` in the same way that it does `prefix`.

7.3 VPATH Substitutions

You might want to compile a software package in a different directory from the one that contains the source code. Doing this allows you to compile the package for several architectures simultaneously from the same copy of the source code and keep multiple sets of object files on disk.

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.in`.

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

7.4 Automatic Remaking

You can put rules like the following in the top-level `Makefile.in` for a package to automatically update the configuration information when you change the configuration files.

The next rule also takes care of making config.h from config.h.in. # If remaking config.h does not change it, its timestamp is untouched. Makefile: Makefile.in config.status $(SHELL) config.status config.status: configure $(SHELL) $(srcdir)/configure –no-create configure: configure.in cd $(srcdir); autoconf config.h.in: configure.in cd $(srcdir); autoheader

8 Running `configure` Scripts

A software package that uses a `configure` script generated by Autoconf 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. Normally, configuring consists of simply doing a `cd` to the package's source code directory and typing:

```
configure
```

If the `PATH` environment variable does not contain the directory '`.`', the command is instead:

```
./configure
```

Users running `csh` on old versions of System V might have to explicitly run `sh` on `configure`:

```
sh configure
```

Running `configure` takes a minute or two. While it is running, it prints some messages that tell what it is doing. If you don't want to see the messages, run `configure` with its standard output redirected to `/dev/null`; for example, '`./configure >/dev/null`'.

To compile the package in a different directory from the one containing the source code, 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 `configure`. `configure` automatically checks for the source code in the directory that `configure` is in and in `..`. If for some reason `configure` is not in the source code directory that you are configuring, then it will report that it can't find the source code. In that case, run `configure` with the option '`--srcdir=`*dir*', where *dir* is the directory that contains the source code.

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*'. Alternately, you can do so by giving a value for the '`prefix`' variable when you run `make`, e.g.,

```
make prefix=/usr/gnu
```

You can specify separate installation prefixes for machine-specific files and machine-independent files. If you give `configure` the option '`--exec-prefix=`*path*' or set the `make` variable '`exec_prefix`' to *path*, the package will use *path* as the prefix for installing programs and libraries. Normally, all files are installed using the same prefix.

Another `configure` option is useful mainly in `Makefile` rules for updating `config.status` and `Makefile`. The '`--no-create`' option figures out the configuration for your system and records it in `config.status`, without actually configuring the package (creating `Makefile`s and perhaps a configuration header file). Later, you can run `./config.status` to actually configure the package. You can also give `config.status` the '`--recheck`' option, which makes it re-run `configure` with the same arguments you used before. This option is useful if you change `configure`.

Some packages pay attention to '`--with-`*package*' options to `configure`, where *package* is something like '`gnu-libc`' or '`x`' (for X windows). The README should mention any '`--with-`' options that the package recognizes.

`configure` ignores any other arguments that you give it.

On systems that require unusual options for compilation or linking that the package's `configure` script does not know about, you can give `configure` initial values for variables by setting them in the environment. In Bourne-compatible shells, you can do that on the command line like this:

```
CC='gcc -traditional' LIBS=-lposix ./configure
```

The `make` variables that you might want to override with environment variables when running `configure` are:

(For these variables, any value given in the environment overrides the value that `configure` would choose:)

`CC` C compiler program. The default is `cc`, or `gcc` if `gcc` is in your `PATH`.

`INSTALL` Program to use to install files. The default is `install` if you have it, `cp` otherwise.

(For these variables, any value given in the environment is added to the value that `configure` chooses:)

`DEFS` Configuration options, in the form '`-Dfoo -Dbar...`'. Do not use this variable in packages that use `AC_CONFIG_HEADER`.

`LIBS` Libraries to link with, in the form '`-lfoo -lbar...`'.

Of course, in the long term, most problems requiring manual intervention should be fixed by updating either the Autoconf macros or the `configure.in` file for that package. See Chapter 3 [Making configure Scripts], page 5, for a discussion of that subject.

9 An Example

Here are sample `configure.in` and `Makefile.in` files, to give a real illustration of using Autoconf. They are from the GNU `cpio` package, which also includes the `mt` and `rmt` programs.

9.1 Sample `configure.in`

Here is `configure.in` from GNU `cpio`. Note the use of the `dnl` macro after `AC_SUBST` to suppress an extra unwanted, though harmless, newline in the generated `configure` script (because the `AC_SUBST` macro does not produce any output where it is called).

```
dnl Process this file with autoconf to produce a configure script.
AC_INIT(cpio.h)
PROGS="cpio"
AC_SUBST(PROGS)dnl
AC_PROG_CC
AC_PROG_CPP
AC_GCC_TRADITIONAL
AC_PROG_INSTALL
AC_AIX
AC_MINIX
AC_ISC_POSIX
AC_RETSIGTYPE
AC_MAJOR_HEADER
AC_REMOTE_TAPE
test -n "$have_mtio" && PROGS="$PROGS mt"
AC_RSH
AC_CONST
AC_UID_T
AC_STDC_HEADERS
AC_UNISTD_H
AC_HAVE_HEADERS(string.h fcntl.h utime.h)
AC_REPLACE_FUNCS(bcopy mkdir strdup)
AC_HAVE_FUNCS(strerror lchown)
AC_VPRINTF
AC_ALLOCA
AC_XENIX_DIR
AC_HAVE_LIBRARY(socket, [LIBS="$LIBS -lsocket"])
AC_HAVE_LIBRARY(nsl, [LIBS="$LIBS -lnsl"])
AC_OUTPUT(Makefile)
```

9.2 Sample `Makefile.in`

Here is `Makefile.in` from GNU `cpio`, with some irrelevant lines omitted, for brevity.

```
#### Start of system configuration section. ####

srcdir = @srcdir@
```

```
VPATH = @srcdir@

CC = @CC@

INSTALL = @INSTALL@
INSTALL_PROGRAM = @INSTALL_PROGRAM@
INSTALL_DATA = @INSTALL_DATA@

DEFS = @DEFS@
LIBS = @LIBS@
RTAPELIB = @RTAPELIB@

CFLAGS = -g
LDFLAGS = -g

prefix = /usr/local
exec_prefix = $(prefix)
binprefix =
manprefix =

bindir = $(exec_prefix)/bin
libdir = /etc
mandir = $(prefix)/man/man1
manext = 1

#### End of system configuration section. ####

SHELL = /bin/sh

SRCS = copyin.c copyout.c copypass.c dstring.c fnmatch.c global.c \
main.c tar.c util.c error.c getopt.c getopt1.c filemode.c version.c \
rtapelib.c dirname.c idcache.c makepath.c xmalloc.c stripslash.c \
userspec.c xstrdup.c bcopy.c mkdir.c strdup.c
OBJS = copyin.o copyout.o copypass.o dstring.o fnmatch.o global.o \
main.o tar.o util.o error.o getopt.o getopt1.o filemode.o version.o \
$(RTAPELIB) dirname.o idcache.o makepath.o xmalloc.o stripslash.o \
userspec.o xstrdup.o @LIBOBJS@ @ALLOCA@
# mt source files not shared with cpio.
MT_SRCS = mt.c argmatch.c
MT_OBJS = mt.o argmatch.o error.o getopt.o getopt1.o \
xmalloc.o $(RTAPELIB) @ALLOCA@
HDRS = cpio.h cpiohdr.h tar.h tarhdr.h dstring.h extern.h filetypes.h \
system.h fnmatch.h getopt.h rmt.h
DISTFILES = $(SRCS) $(HDRS) COPYING COPYING.LIB ChangeLog Makefile.in \
README NEWS INSTALL cpio.1 mt.1 makefile.pc cpio.def cpio.cs \
configure configure.in $(MT_SRCS) rmt.c tcexparg.c alloca.c
```

```
all: @PROGS@

.c.o:
        $(CC) -c $(CFLAGS) $(CPPFLAGS) $(DEFS) -I$(srcdir) $<

install: all $(srcdir)/cpio.1 $(srcdir)/mt.1
        $(INSTALL_PROGRAM) cpio $(bindir)/$(binprefix)cpio
        test ! -f mt || $(INSTALL_PROGRAM) mt $(bindir)/$(binprefix)mt
        -test ! -f rmt || $(INSTALL_PROGRAM) rmt /etc/rmt
        $(INSTALL_DATA) $(srcdir)/cpio.1 $(mandir)/$(manprefix)cpio.$(manext)
        test ! -f mt || \
        $(INSTALL_DATA) $(srcdir)/mt.1 $(mandir)/$(manprefix)mt.$(manext)

cpio: $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

rmt: rmt.o
        $(CC) $(LDFLAGS) -o $@ rmt.o $(LIBS)

mt: $(MT_OBJS)
        $(CC) $(LDFLAGS) -o $@ $(MT_OBJS) $(LIBS)

TAGS: $(SRCS)
        etags $(SRCS)

clean:
        rm -f cpio rmt mt *.o core

mostlyclean: clean

distclean: clean
        rm -f Makefile config.status

realclean: distclean
        rm -f TAGS

dist:
        echo cpio-`sed -e '/version_string/!d' \
        -e 's/[^0-9.]*\([0-9.]*\).*/\1/' -e q version.c` > .fname
        rm -rf `cat .fname`
        mkdir `cat .fname`
        ln $(DISTFILES) `cat .fname`
        tar chZf `cat .fname`.tar.Z `cat .fname`
        rm -rf `cat .fname` .fname
```

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.

shouldnt see this

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_'.

S

T

U

V

W

X

Table of Contents