automake

COLLABORATORS

	TITLE : automake		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 15, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

automake

1.1 automake.guide

```
                GNU Automake
************

   This file documents the GNU Automake package for creating GNU
Standards-compliant Makefiles from template files.  This edition
documents version 1.1e.
```

1.2 automake.guide/Introduction

```
Introduction
************

   Automake is a tool for automatically generating `Makefile.in's from
files called `Makefile.am'.  The `Makefile.am' is basically a series of
`make' macro definitions (with rules being thrown in occasionally).
The generated `Makefile.in's are compliant with the GNU Makefile
standards.

   The GNU Makefile Standards Document (see Makefile Conventions) is
long, complicated, and subject to change.  The goal of Automake is to
remove the burden of Makefile maintenance from the back of the
individual GNU maintainer (and put it on the back of the Automake
maintainer).
```

```
   The typical Automake input files is simply a series of macro
definitions.  Each such file is processed to create a `Makefile.in'.
There should generally be one `Makefile.am' per directory of a project.

   Automake does constrain a project in certain ways; for instance it
assumes that the project uses Autoconf (see The Autoconf Manual), and
enforces certain restrictions on the `configure.in' contents.

   `Automake' requires `perl' in order to generate the `Makefile.in's.
However, the distributions created by Automake are fully GNU
standards-compliant, and do not require `perl' in order to be built.

   Mail suggestions and bug reports for Automake to
bug-gnu-utils@prep.ai.mit.edu.
```

1.3 automake.guide/Invoking Automake

```
                Creating a `Makefile.in'
************************

   To create all the `Makefile.in's for a package, run the `automake'
program in the top level directory, with no arguments.  `automake' will
automatically find each appropriate `Makefile.am' (by scanning
`configure.in'; see
                configure
                ) and generate the corresponding
`Makefile.in'.

   You can optionally give `automake' an argument; `.am' is appended to
the argument and the result is used as the name of the input file.
This feature is generally only used to automatically rebuild an
out-of-date `Makefile.in'.  Note that `automake' must always be run
from the topmost directory of a project, even if being used to
regenerate the `Makefile.in' in some subdirectory.  This is necessary
because `automake' must scan `configure.in', and because `automake'
uses the knowledge that a `Makefile.in' is in a subdirectory to change
its behavior in some cases.

   `automake' accepts the following options:

`-a'
`--add-missing'
     Automake requires certain common files to exist in certain
     situations; for instance `config.guess' is required if
     `configure.in' runs `AC_CANONICAL_HOST'.  Automake is distributed
     with several of these files; this option will cause the missing
     ones to be automatically added to the package, whenever possible.
     In general if Automake tells you a file is missing, try using this
     option.

`--amdir=DIR'
     Look for Automake data files in directory DIR instead of in the
     installation directory.  This is typically used for debugging.
```

'--build-dir=DIR'
Tell Automake where the build directory is. This option is used when including dependencies into a 'Makefile.in' generated by 'make dist'; it should not be used otherwise.

'--foreign'
An alias for '--strictness=foreign'.

'--gnits'
An alias for '--strictness=gnits'.

'--gnu'
An alias for '--strictness=gnu'.

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

'-i'
'--include-deps'
Include all automatically generated dependency information (see

Dependencies
) in the generated 'Makefile.in'. This is generally
done when making a distribution; see See
Dist
.

'-o DIR'
'--output-dir=DIR'
Put the generated 'Makefile.in' in the directory DIR. Ordinarily each 'Makefile.in' is created in the directory of the corresponding 'Makefile.am'. This option is used when making distributions.

'--srcdir-name=DIR'
Tell Automake the name of the source directory used in the current build. This option is used when including dependencies into a 'Makefile.in' generated by 'make dist'; it should not be used otherwise.

'-s LEVEL'
'--strictness=LEVEL'
Set the global strictness to LEVEL; this can be overridden in each 'Makefile.am' if required. See
Generalities
for more information.

'-v'
'--verbose'
Cause Automake to print information about which files are being read or created.

'--version'
Print the version number of Automake and exit.

1.4 automake.guide/Generalities

```
                General ideas
*************

   There are a few basic ideas that will help understand how Automake
works.
```

1.5 automake.guide/General Operation

```
General Operation
=================

   Automake essentially works by reading a `Makefile.am' and generating
a `Makefile.in'.

   The macro definitions and targets in the `Makefile.am' are copied
into the generated file.  This allows you to add essentially arbitrary
code into the generated `Makefile.in'.  For instance the Automake
distribution includes a non-standard `cvs-dist' target, which the
Automake maintainer uses to make distributions from his source control
system.

   Note that GNU make extensions are not recognized by Automake.  Using
such extensions in a `Makefile.am' will lead to errors or confusing
behavior.

   Automake tries to group comments with adjoining targets (or variable
definitions) in an intelligent way.

   A target defined in `Makefile.am' generally overrides any such
```

target of a similar name that would be automatically generated by 'automake'. Although this is a supported feature, it is generally best to avoid making use of it, as sometimes the generated rules are very particular.

When examining a variable definition, Automake will recursively examine variables referenced in the definition. Eg if Automake is looking at the content of 'foo_SOURCES' in this snippet

```
     xs = a.c b.c
     foo_SOURCES = c.c $(xs)
```

it would use the files 'a.c', 'b.c', and 'c.c' as the contents of 'foo_SOURCES'.

Automake also allows a form of comment which is *not* copied into the output; all lines beginning with '##' are completely ignored by Automake.

It is customary to make the first line of 'Makefile.am' read:

```
     ## Process this file with automake to produce Makefile.in
```

1.6 automake.guide/Depth

Depth
=====

'automake' supports three kinds of directory hierarchy: "flat", "shallow", and "deep".

A "flat" package is one in which all the files are in a single directory. The 'Makefile.am' for such a package by definition lacks a 'SUBDIRS' macro. An example of such a package is 'termutils'.

A "deep" package is one in which all the source lies in subdirectories; the top level directory contains mainly configuration information. GNU cpio is a good example of such a package, as is GNU 'tar'. The top level 'Makefile.am' for a deep package will contain a 'SUBDIRS' macro, but no other macros to define objects which are built.

A "shallow" package is one in which the primary source resides in the top-level directory, while various parts (typically libraries) reside in subdirectories. 'automake' is one such package (as is GNU 'make', which does not currently use 'automake').

1.7 automake.guide/Strictness

Strictness
==========

While Automake is intended to be used by maintainers of GNU packages, it does make some effort to accommodate those who wish to use it, but do not want to use all the GNU conventions.

To this end, Automake supports three levels of "strictness" - the strictness indicating how stringently Automake should check standards conformance.

The valid strictness levels are:

'foreign'
: Automake will check for only those things which are absolutely required for proper operations. For instance, whereas GNU standards dictate the existence of a 'NEWS' file, it will not be required in this mode. The name comes from the fact that Automake is intended to be used for GNU programs; these relaxed rules are not the standard mode of operation.

'gnu'
: Automake will check - as much as possible - for compliance to the GNU standards for packages. This is the default.

'gnits'
: Automake will check for compliance to the as-yet-unwritten GNITS standards. These are based on the GNU standards, but are even more detailed. Unless you are a GNITS standards contributor, it is recommended that you avoid this option until such time as the GNITS standard is actually published.

1.8 automake.guide/Uniform

The Uniform Naming Scheme
=========================

Automake variables generally follow a uniform naming scheme that makes it easy to decide how programs (and other derived objects) are built, and how they are installed. This scheme also supports 'configure' time determination of what should be built.

At 'make' time, certain variables are used to determine which objects are to be built. These variables are called "primary" variables. For instance, the primary variable 'PROGRAMS' holds a list of programs which are to be compiled and linked.

A different set of variables is used to decide where the built objects should be installed. These variables are named after the primary variables, but have a prefix indicating which standard directory should be used as the installation directory. The standard directory names are given in the GNU standards (see Directory Variables). 'automake' extends this list with 'pkglibdir', 'pkgincludedir', and 'pkgdatadir'; these are the same as the non-'pkg' versions, but with '@PACKAGE@' appended.

For each primary, there is one additional variable named by prepending 'EXTRA_' to the primary name. This variable is used to list objects which may or may not be built, depending on what 'configure' decides. This variable is required because Automake must know the entire list of objects to be built in order to generate a 'Makefile.in' that will work in all cases.

For instance, 'cpio' decides at configure time which programs are built. Some of the programs are installed in 'bindir', and some are installed in 'sbindir':

```
     EXTRA_PROGRAMS = mt rmt
     bin_PROGRAMS = cpio pax
     sbin_PROGRAMS = @PROGRAMS@
```

Defining a primary variable is an error.

Note that the common 'dir' suffix is left off when constructing the variable names; thus one writes 'bin_PROGRAMS' and not 'bindir_PROGRAMS'.

Not every sort of object can be installed in every directory. Automake will flag those attempts it finds in error. Automake will also diagnose obvious misspellings in directory names.

Sometimes the standard directories - even as augmented by Automake - are not enough. In particular it is sometimes useful, for clarity, to install objects in a subdirectory of some predefined directory. To this end, Automake allows you to extend the list of possible installation directories. A given prefix (eg 'zar') is valid if a variable of the same name with 'dir' appended is defined (eg 'zardir').

For instance, until HTML support is part of Automake, you could use this to install raw HTML documentation:

```
     htmldir = $(prefix)/html
     html_DATA = automake.html
```

The special prefix 'noinst' indicates that the objects in question should not be installed at all.

The special prefix 'check' indicates that the objects in question should not be built until the 'make check' command is run.

Possible primary names are 'PROGRAMS', 'LIBRARIES', 'LISP', 'SCRIPTS', 'DATA', 'HEADERS', 'MANS', and 'TEXINFOS'.

1.9 automake.guide/Canonicalization

```
How derived variables are named
===============================
```

Sometimes a Makefile variable name is derived from some text the user supplies. For instance program names are rewritten into Makefile macro names. Automake canonicalizes this text, so that it does not have to follow Makefile variable naming rules. All characters in the name except for letters, numbers, and the underscore are turned into underscores when making macro references. Eg, if your program is named 'sniff-glue', the derived variable name would be 'sniff_glue_SOURCES', not 'sniff-glue_SOURCES'.

1.10 automake.guide/configure

```
                Scanning 'configure.in'
***********************
```

Automake scans the package's 'configure.in' to determine certain information about the package. Some 'autoconf' macros are required and some variables must be defined in 'configure.in'. Automake will also use information from 'configure.in' to further tailor its output.

1.11 automake.guide/Requirements

```
                Configuration requirements
==========================
```

The simplest way to meet the basic Automake requirements is to use the macro 'AM_INIT_AUTOMAKE' (FIXME: xref). But if you prefer, you can do the required steps by hand:

* Define the variables 'PACKAGE' and 'VERSION' with 'AC_SUBST'. 'PACKAGE' should be the name of the package as it appears when bundled for distribution. For instance, Automake defines 'PACKAGE' to be 'automake'. 'VERSION' should be the version number of the release that is being developed. We recommend that you make 'configure.in' the only place in your package where the version

```
     number is defined; this makes releases simpler.

     Automake doesn't do any interpretation of 'PACKAGE' or 'VERSION',
     except in 'Gnits' mode (FIXME xref).

   * Use the macro 'AC_ARG_PROGRAM' if a program or script is installed.

   * Use 'AC_PROG_MAKE_SET' if the package is not flat.

   * Use 'AM_PROG_INSTALL' if any scripts (see
                Scripts
                ) are installed
     by the package.  Otherwise, use 'AC_PROG_INSTALL'.

   Here are the other macros which Automake requires but which are not
run by 'AM_INIT_AUTOMAKE':

'AC_OUTPUT'
     Automake uses this to determine which files to create.  Listed
     files named 'Makefile' are treated as 'Makefile's.  Other listed
     files are treated differently.  Currently the only difference is
     that a 'Makefile' is removed by 'make distclean', while other files
     are removed by 'make clean'.
```

1.12 automake.guide/Optional

```
                Other things Automake recognizes
================================

   Automake will also recognize the use of certain macros and tailor the
generated 'Makefile.in' appropriately.  Currently recognized macros and
their effects are:

'AC_CONFIG_HEADER'
     Automake will generate rules to automatically regenerate the config
     header.  If you do use this macro, you must create the file
     'stamp-h.in' in your source directory.  It can be empty.  Also, the
     'AC_OUTPUT' command in 'configure.in' must create 'stamp-h', eg:
          AC_OUTPUT(Makefile,
          [test -z "$CONFIG_HEADERS" || echo timestamp > stamp-h])
     Note that Automake does not currently currently check to make sure
     the 'AC_OUTPUT' command is correct.  Hopefully a future version of
     'autoconf' will let Automake handle this automatically.

'AC_CONFIG_AUX_DIR'
     Automake will look for various helper scripts, such as
     'mkinstalldirs', in the directory named in this macro invocation.
     If not seen, the scripts are looked for in their "standard"
     locations (either the top source directory, or in the source
     directory corresponding to the current 'Makefile.am', whichever is
     appropriate).  FIXME: give complete list of things looked for in
     this directory
```

```
'AC_PATH_XTRA'
     Automake will insert definitions for the variables defined by
     'AC_PATH_XTRA' into each 'Makefile.in' that builds a C program or
     library.

'AC_CANONICAL_HOST'
'AC_CANONICAL_SYSTEM'
'AC_CHECK_TOOL'
     Automake will ensure that 'config.guess' and 'config.sub' exist.

'AC_FUNC_ALLOCA'
'AC_FUNC_GETLOADAVG'
'AC_FUNC_MEMCMP'
'AC_STRUCT_ST_BLOCKS'
'AM_FUNC_FNMATCH'
'AM_FUNC_STRTOD'
'AC_REPLACE_FUNCS'
'AC_REPLACE_GNU_GETOPT'
'AM_WITH_REGEX'
     Automake will ensure that the appropriate source files are part of
     the distribution, and will ensure that the appropriate
     dependencies are generated for these objects.  See
                A Library
                 for
     more information.

'LIBOBJS'
     Automake will detect statements which put '.o' files into
     'LIBOBJS', and will treat these additional files as if they were
     discovered via 'AC_REPLACE_FUNCS'.

'AC_PROG_RANLIB'
     This is required if any libraries are built in the package.

'AC_PROG_CXX'
     This is required if any C++ source is included.

'AC_PROG_YACC'
     If a Yacc source file is seen, then you must either use this macro
     or declare the variable 'YACC' in 'configure.in'.  The former is
     preferred.

'AC_DECL_YYTEXT'
     This macro is required if there is Yacc source in the package.

'AC_PROG_LEX'
     If a Lex source file is seen, then this macro must be used.

'ALL_LINGUAS'
     If Automake sees that this variable is set in 'configure.in', it
     will check the 'po' directory to ensure that all the named '.po'
     files exist, and that all the '.po' files that exist are named.

'AM_C_PROTOTYPES'
     This is required when using automatic de-ANSI-fication, see See

                ANSI
```

```
                 .

`ud_GNU_GETTEXT'
     This macro is required for packages which use GNU gettext (see

                 gettext
                 ).  It is distributed with gettext.  Automake uses this
     macro to ensure that the package meets some of gettext's
     requirements.

`AM_MAINTAINER_MODE'
     This macro adds a `--enable-maintainer-mode' option to
     `configure'.  If this is used, `automake' will cause
     "maintainer-only" rules to be turned off by default in the
     generated `Makefile.in's.  This macro is disallowed in `Gnits'
     mode.  FIXME xref.
```

1.13 automake.guide/Invoking aclocal

```
Auto-generating aclocal.m4
==========================

   The `aclocal' program will automatically generate `aclocal.m4' files
based on the contents of `configure.in'.

   ... explain why on earth you'd want to do this

   `aclocal' accepts the following options:

`--acdir=DIR'
     Look for the macro files in DIR instead of the installation
     directory.  This is typically used for debugging.

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

`--output=FILE'
     Cause the output to be put into FILE instead of `aclocal.m4'.

`--verbose'
     Print the names of the files it examines.

`--version'
     Print the version number of Automake and exit.
```

1.14 automake.guide/Macros

```
Autoconf macros supplied with Automake
======================================
`AM_FUNC_FNMATCH'
     If the `fnmatch' function is not available, or does not work
     correctly (like the one on SunOS 5.4), add `fnmatch.o' to output
     variable `LIBOBJS'.

`AM_FUNC_STRTOD'
     If the `strtod' function is not available, or does not work
     correctly (like the one on SunOS 5.4), add `strtod.o' to output
     variable `LIBOBJS'.

`AM_C_PROTOTYPES'
`AM_TIOCGWINSZ_NEEDS_IOCTL'
`AM_INIT_AUTOMAKE'
`AM_MAINTAINER_MODE'
`AM_PATH_LISPDIR'
`AM_PROG_CC_STDC'
`AM_PROG_INSTALL'
`AM_SANITY_CHECK_CC'
`AM_SYS_POSIX_TERMIOS'
`AM_TYPE_PTRDIFF_T'
`AM_WITH_DMALLOC'
`AM_WITH_REGEX'
```

1.15 automake.guide/Extending aclocal

```
Writing your own aclocal macros
===============================

   ... explain format of macro files ... explain how to get your own
macros installed (using acinstall) ... explain situations where this is
actually useful (eg gettext)
```

1.16 automake.guide/Top level

```
The top-level `Makefile.am'
***************************

   In non-flat packages, the top level `Makefile.am' must tell Automake
which subdirectories are to be built.  This is done via the `SUBDIRS'
variable.

   The `SUBDIRS' macro holds a list of subdirectories in which building
of various sorts can occur.  Many targets (eg `all') in the generated
`Makefile' will run both locally and in all specified subdirectories.
Note that the directories listed in `SUBDIRS' are not required to
contain `Makefile.am's; only `Makefile's (after configuration).  This
allows inclusion of libraries from packages which do not use Automake
(such as `gettext').

   In a deep package, the top-level `Makefile.am' is often very short.
```

For instance, here is the 'Makefile.am' from the textutils distribution:

```
SUBDIRS = lib src doc man
EXTRA_DIST = @README_ALPHA@
```

'SUBDIRS' can contain configure substitutions (eg '@DIRS@'); Automake itself does not actually examine the contents of this variable.

If 'SUBDIRS' is defined, then your 'configure.in' must include 'AC_PROG_MAKE_SET'.

The use of 'SUBDIRS' is not restricted to just the top-level 'Makefile.am'. Automake can be used to construct packages of arbitrary depth.

1.17 automake.guide/Programs

```
                Building Programs and Libraries
*******************************
```

A large part of Automake's functionality is dedicated to making it easy to build C programs and libraries.

1.18 automake.guide/A Program

```
Building a program
==================

   In a directory containing source that gets built into a program (as
opposed to a library), the ‘PROGRAMS’ primary is used.  Programs can be
installed in ‘bindir’, ‘sbindir’, ‘libexecdir’, ‘pkglibdir’, or not at
all.

   For instance:

     bin_PROGRAMS = hello

   In this simple case, the resulting ‘Makefile.in’ will contain code
to generate a program named ‘hello’.  The variable ‘hello_SOURCES’ is
used to specify which source files get built into an executable:

     hello_SOURCES = hello.c

   This causes ‘hello.c’ to be compiled into ‘hello.o’, and then linked
to produce ‘hello’.

   If ‘prog_SOURCES’ is needed, but not specified, then it defaults to
the single file ‘prog.c’.  In the example above, the definition of
‘hello_SOURCES’ is actually redundant.

   Multiple programs can be built in a single directory.  Multiple
programs can share a single source file.  The source file must be
listed in each ‘_SOURCES’ definition.

   Header files listed in a ‘_SOURCES’ definition will be included in
the distribution but otherwise ignored.  In case it isn’t obvious, you
should not include the header file generated by ‘configure’ in an
‘_SOURCES’ variable; this file should not be distributed.  Lex (‘.l’)
and yacc (‘.y’) files can also be listed; support for these should work
but is still preliminary.

   Sometimes it is useful to determine the programs that are to be
built at configure time.  For instance, GNU ‘cpio’ only builds ‘mt’ and
‘rmt’ under special circumstances.

   In this case, you must notify ‘Automake’ of all the programs that
can possibly be built, but at the same time cause the generated
‘Makefile.in’ to use the programs specified by ‘configure’.  This is
done by having ‘configure’ substitute values into each ‘_PROGRAMS’
definition, while listing all optionally built programs in
‘EXTRA_PROGRAMS’.

   If you need to link against libraries that are not found by
‘configure’, you can use ‘LDADD’ to do so.  This variable actually can
be used to add any options to the linker command line.

   Sometimes, multiple programs are built in one directory but do not
share the same link-time requirements.  In this case, you can use the
‘PROG_LDADD’ variable (where PROG is the name of the program as it
```

appears in some '_PROGRAMS' variable, and usually written in lowercase) to override the global 'LDADD'. (If this variable exists for a given program, then that program is not linked using 'LDADD'.)

For instance, in GNU cpio, 'pax', 'cpio', and 'mt' are linked against the library 'libcpio.a'. However, 'rmt' is built in the same directory, and has no such link requirement. Also, 'mt' and 'rmt' are only built on certain architectures. Here is what cpio's 'src/Makefile.am' looks like (abridged):

```
     bin_PROGRAMS = cpio pax @MT@
     libexec_PROGRAMS = @RMT@
     EXTRA_PROGRAMS = mt rmt

     LDADD = ../lib/libcpio.a @INTLLIBS@
     rmt_LDADD =

     cpio_SOURCES = ...
     pax_SOURCES = ...
     mt_SOURCES = ...
     rmt_SOURCES = ...
```

It is also occasionally useful to have a program depend on some other target which is not actually part of that program. This can be done using the 'prog_DEPENDENCIES' variable. Each program depends on the contents of such a variable, but no further interpretation is done.

If 'prog_DEPENDENCIES' is not supplied, it is computed by Automake. The automatically-assigned value is the contents of 'prog_LDADD' with all the '-l' and '-L' options removed. Be warned that 'configure' substitutions are preserved; this can lead to bad dependencies if you are not careful.

1.19 automake.guide/A Library

```
Building a library
==================
```

Building a library is much like building a program. In this case, the name of the primary is 'LIBRARIES'. Libraries can be installed in 'libdir' or 'pkglibdir'.

Each '_LIBRARIES' variable is a list of the base names of libraries to be built. For instance to create a library named 'libcpio.a', but not install it, you would write:

```
     noinst_LIBRARIES = cpio
```

The sources that go into a library are determined exactly as they are for programs, via the '_SOURCES' variables. Note that programs and libraries share a namespace, so one cannot have a program ('lob') and a library ('liblob.a') with the same name in one directory.

Extra objects can be added to a library using the 'library_LIBADD'

variable. This should be used for objects determined by 'configure'.
Again from cpio:

```
     cpio_LIBADD = @LIBOBJS@ @ALLOCA@
```

1.20 automake.guide/LIBOBJS

Special handling for LIBOBJS and ALLOCA
=======================================

Automake explicitly recognizes the use of '@LIBOBJS@' and '@ALLOCA@', and uses this information, plus the list of 'LIBOBJS' files derived from 'configure.in' to automatically include the appropriate source files in the distribution (see
Dist
). These source files are
also automatically handled in the dependency-tracking scheme, see See

Dependencies
.

'@LIBOBJS@' and '@ALLOCA@' are specially recognized in any '_LDADD' or '_LIBADD' variable.

1.21 automake.guide/Program variables

Variables used when building a program
======================================

Occasionally it is useful to know which 'Makefile' variables Automake uses for compilations; for instance you might need to do your own compilation in some special cases.

Some variables are inherited from Autoconf; these are 'CC', 'CFLAGS', 'CPPFLAGS', 'DEFS', 'LDFLAGS', and 'LIBS'.

There are some additional variables which Automake itself defines:

'INCLUDES'
     A list of '-I' options. This can be set in your 'Makefile.am' if you have special directories you want to look in.

'COMPILE'
     This is the command used to actually compile a C source file. The filename is appended to form the complete command line.

'LINK'
     This is the command used to actually link a C program.

1.22 automake.guide/Yacc and Lex

```
Yacc and Lex support
====================

   Automake has somewhat idiosyncratic support for Yacc and Lex.
FIXME: describe it here.
```

1.23 automake.guide/C++

```
C++ and other languages
=======================

   Automake includes full support for C++, and rudimentary support for
other languages.  Support for other languages will be improved based on
demand.

   Any package including C++ code must use `AC_PROG_CXX' in its
`configure.in'.

   A few additional variables are defined when a C++ source file is
seen:

`CXX'
     The name of the C++ compiler.

`CXXFLAGS'
     Any flags to pass to the C++ compiler.

`CXXCOMPILE'
     The command used to actually compile a C++ source file.  The file
     name is appended to form the complete command line.

`CXXLINK'
     The command used to actually link a C++ program.
```

1.24 automake.guide/ANSI

```
                Automatic de-ANSI-fication
==========================

   Although the GNU standards prohibit it, some GNU programs are
written in ANSI C; see FIXME.  This is possible because each source
file can be "de-ANSI-fied" before the actual compilation takes place.

   If the `Makefile.am' variable `AUTOMAKE_OPTIONS' (See
                Options
                )
contains the option `ansi2knr' then code to handle de-ANSI-fication is
inserted into the generated `Makefile.in'.
```

This causes each source file to be treated as ANSI C. If an ANSI C compiler is available, it is used.

This support requires the source files 'ansi2knr.c' and 'ansi2knr.1' to be in the same directory as the ANSI C source; these files are distributed with Automake. Also, the package 'configure.in' must call the macro 'AM_C_PROTOTYPES'.

Automake also handles finding the 'ansi2knr' support files in some other directory in the current package. This is done by prepending the relative path to the appropriate directory to the 'ansi2knr' option. For instance, suppose the package has ANSI C code in the 'src' and 'lib' subdirs. The files 'ansi2knr.c' and 'ansi2knr.1' appear in 'lib'. Then this could appear in 'src/Makefile.am':

```
     AUTOMAKE_OPTIONS = ../lib/ansi2knr
```

Note that the directory holding the 'ansi2knr' support files must be built before all other directories using these files. Automake does not currently check that this is the case.

1.25 automake.guide/Dependencies

```
Automatic dependency tracking
=============================
```

As a developer it is often painful to continually update the 'Makefile.in' whenever the include-file dependencies change in a project. 'automake' supplies a way to automatically track dependency changes, and distribute the dependencies in the generated 'Makefile.in'.

Currently this support requires the use of GNU 'make' and 'gcc'. It might become possible in the future to supply a different dependency generating program, if there is enough demand.

This mode is enabled by default if any C program or library is defined in the current directory.

When you decide to make a distribution, the 'dist' target will re-run 'automake' with the '--include-deps' option. This causes the previously generated dependencies to be inserted into the generated 'Makefile.in', and thus into the distribution. '--include-deps' also turns off inclusion of the dependency generation code.

This mode can be suppressed by putting 'no-dependencies' in the variable 'AUTOMAKE_OPTIONS'.

If you unpack a distribution made by 'make dist', and you want to turn on the dependency-tracking code again, simply run 'automake' with no arguments.

1.26 automake.guide/Other objects

```
                Other Derived Objects
*********************

   Automake can handle derived objects which are not C programs.
Sometimes the support for actually building such objects must be
explicitly supplied, but Automake will still automatically handle
installation and distribution.



                 Scripts
                  Executable scripts

                 Headers
                  Header files

                 Data
                  Architecture-independent data files

                 Sources
                  Derived sources
```

1.27 automake.guide/Scripts

```
                Executable Scripts
==================

   It is possible to define and install programs which are scripts.
Such programs are listed using the `SCRIPTS' primary name.  `automake'
doesn't define any dependencies for scripts; the `Makefile.am' should
include the appropriate rules.

   `automake' does not assume that scripts are derived objects; such
objects must be deleted by hand; see See
                Clean
                 for more information.

   `automake' itself is a script that is generated at configure time
from `automake.in'.  Here is how this is handled:

     bin_SCRIPTS = automake

   Since `automake' appears in the `AC_OUTPUT' macro, a target for it
is automatically generated.

   Script objects can be installed in `bindir', `sbindir',
`libexecdir', or `pkgdatadir'.
```

1.28 automake.guide/Headers

```
                Header files
============

   Header files are specified by the `HEADERS' family of variables.
Generally header files are not installed, so the `noinst_HEADERS'
variable will be the most used.

   All header files must be listed somewhere; missing ones will not
appear in the distribution.  Often it is clearest to list uninstalled
headers with the rest of the sources for a program.  See
                A Program
                .
Headers listed in a `_SOURCES' variable need not be listed in any
`_HEADERS' variable.

   Headers can be installed in `includedir', `oldincludedir', or
`pkgincludedir'.
```

1.29 automake.guide/Data

```
Architecture-independent data files
===================================

   Automake supports the installation of miscellaneous data files using
the `DATA' family of variables.

   Such data can be installed in the directories `datadir',
`sysconfdir', `sharedstatedir', `localstatedir', or `pkgdatadir'.

   All such data files are included in the distribution.

   Here is how `automake' installs its auxiliary data files:

     pkgdata_DATA = clean-kr.am clean.am compile-kr.am compile-vars.am \
     compile.am data.am depend.am dist-subd-top.am dist-subd-vars.am \
     dist-subd.am dist-vars.am dist.am footer.am header-vars.am header.am \
     libscripts.am libprograms.am libraries-vars.am libraries.am library.am \
     mans-vars.am mans.am packagedata.am program.am programs.am remake-hdr.am \
     remake-subd.am remake.am scripts.am subdirs.am tags.am tags-subd.am \
     texinfos-vars.am texinfos.am hack-make.sed nl-remove.sed
```

1.30 automake.guide/Sources

```
Built sources
=============

   Occasionally a file which would otherwise be called "source" (eg a C
```

'.h' file) is actually derived from some other file. Such files should
be listed in the 'BUILT_SOURCES' variable.

Files listed in 'BUILT_SOURCES' are built before any automatic
dependency tracking is done. Built sources are included in a
distribution.

1.31 automake.guide/Other GNU Tools

```
                Other GNU Tools
***************
```

Since Automake is primarily intended to generate 'Makefile.in's for
use in GNU programs, it tries hard to interoperatoe with other GNU
tools.

```
                 Emacs Lisp
                  Emacs Lisp

                 gettext
                  Gettext
```

1.32 automake.guide/Emacs Lisp

```
Emacs Lisp
==========
```

Automake provides some support for Emacs Lisp. The 'LISP' primary
is used to hold a list of '.el' files. Possible prefixes for this
primary are 'lisp_' and 'noinst_'. Note that if 'lisp_LISP' is
defined, then 'configure.in' must run 'AM_PATH_LISPDIR' (fixme xref).

By default Automake will byte-compile all Emacs Lisp source files
using the Emacs found by 'AM_PATH_LISPDIR'. If you wish to avoid
byte-compiling, simply define the variable 'ELCFILES' to be empty.

1.33 automake.guide/gettext

```
Gettext
=======
```

If 'ud_GNU_GETTEXT' is seen in 'configure.in', then Automake turns
on support for GNU gettext, a message catalog system for
internationalization (see GNU Gettext).

The ‘gettext’ support in Automake requires the addition of two subdirectories to the package, ‘intl’ and ‘po’. Automake ensure that these directories exist and are mentioned in ‘SUBDIRS’.

Furthermore, Automake checks that the definition of ‘ALL_LINGUAS’ in ‘configure.in’ corresponds to all the valid ‘.po’ files, and nothing more.

1.34 automake.guide/Documentation

```
                Building documentation
**********************
```

Currently Automake provides support for Texinfo and man pages.

```
                 Texinfo
                  Texinfo

                 Man pages
                  Man pages
```

1.35 automake.guide/Texinfo

```
Texinfo
=======
```

If the current directory contains Texinfo source, you must declare it with the ‘TEXINFOS’ primary. Generally Texinfo files are converted into info, and thus the ‘info_TEXINFOS’ macro is most commonly used here. Note that any Texinfo source file must end in the ‘.texi’ or ‘.texinfo’ extension.

If the ‘.texi’ file ‘@include’s ‘version.texi’, then that file will be automatically generated. ‘version.texi’ defines three Texinfo macros you can reference: ‘EDITION’, ‘VERSION’, and ‘UPDATED’. The first two hold the version number of your package (but are kept separate for clarity); the last is the date the primary file was last modified. The ‘version.texi’ support requires the ‘mdate-sh’ program; this program is supplied with Automake.

Sometimes an info file actually depends on more than one ‘.texi’ file. For instance, in the ‘xdvik’ distribution, ‘kpathsea.texi’ includes the files ‘install.texi’, ‘copying.texi’, and ‘freedom.texi’. You can tell Automake about these dependencies using the ‘texi_TEXINFOS’ variable. Here is how ‘xdvik’ could do it:

```
     info_TEXINFOS = kpathsea.texi
     kpathsea_TEXINFOS = install.texi copying.texi freedom.texi
```

By default, Automake requires the file 'texinfo.tex' to appear in the same directory as the Texinfo source. However, if you used 'AC_CONFIG_AUX_DIR' in 'configure.in', then 'texinfo.tex' is looked for there. Automake supplies 'texinfo.tex'.

Automake generates an 'install-info' target; some people apparently use this. By default, info pages are installed by 'make install'. This can be prevented via the 'no-installinfo' option.

1.36 automake.guide/Man pages

Man pages
=========

A package can also include man pages. (Though see the GNU standards on this matter, See Man Pages.) Man pages are declared using the 'MANS' primary. Generally the 'man_MANS' macro is used. Man pages are automatically installed in the correct subdirectory of 'mandir', based on the file extension.

By default, man pages are installed by 'make install'. However, since the GNU project does not require man pages, many maintainers do not expend effort to keep the man pages up to date. In these cases, the 'no-installman' option will prevent the man pages from being installed by default. The user can still explicitly install them via 'make install-man'.

Here is how the documentation is handled in GNU 'cpio' (which includes both Texinfo documentation and man pages):

```
     info_TEXINFOS = cpio.texi
     man_MANS = cpio.1 mt.1
```

Texinfo source, info pages and man pages are all considered to be source for the purposes of making a distribution.

1.37 automake.guide/Install

What Gets Installed
****

Naturally, Automake handles the details of actually installing your program once it has been built. All 'PROGRAMS', 'SCRIPTS', 'LIBRARIES', 'LISP', 'DATA' and 'HEADERS' are automatically installed in the appropriate places.

Automake also handles installing any specified info and man pages.

Automake generates separate 'install-data' and 'install-exec' targets, in case the installer is installing on multiple machines which share directory structure - these targets allow the machine-independent parts to be installed only once. The 'install' target depends on both of these targets.

Automake also generates an 'uninstall' target, and an 'installdirs' target.

It is possible to extend this mechanism by defining an 'install-exec-local' or 'install-data-local' target. If these targets exist, they will be run at 'make install' time.

1.38 automake.guide/Clean

What Gets Cleaned
****

The GNU Makefile Standards specify a number of different clean rules. Generally the files that can cleaned are determined automatically by Automake. Of course, Automake also recognizes some variables that can be defined to specify additional files to clean. These variables are 'MOSTLYCLEANFILES', 'CLEANFILES', 'DISTCLEANFILES', and 'MAINTAINERCLEANFILES'.

1.39 automake.guide/Dist

What Goes in a Distribution
****

The 'dist' target in the generated 'Makefile.in' can be used to generate a gzip'd 'tar' file for distribution. The tar file is named based on the PACKAGE and VERSION variables; more precisely it is named 'PACKAGE-VERSION.tar.gz'.

For the most part, the files to distribute are automatically found by Automake: all source files are automatically included in a distribution, as are all 'Makefile.am's and 'Makefile.in's. Automake also has a built-in list of commonly used files which, if present in the current directory, are automatically included. This list is printed by 'automake --help'. Also, files which are read by 'configure' (ie, the source files corresponding to the files specified in the 'AC_OUTPUT' invocation) are automatically distributed.

Still, sometimes there are files which must be distributed, but which are not covered in the automatic rules. These files should be listed in the 'EXTRA_DIST' variable.

Occasionally it is useful to be able to change the distribution before it is packaged up. If the 'dist-hook' target exists, it is run after the distribution directory is filled, but before the actual tar (or shar) file is created. One way to use this is for distributing file in subdirectories for which a new 'Makefile.am' is overkill:

```
     dist-hook:
             mkdir $(distdir)/random
             cp -p random/a1 random/a2 $(distdir)/random
```

Automake also generates a 'distcheck' target which can be help to ensure that a given distribution will actually work. 'distcheck' makes a distribution, and then tries to do a 'VPATH' build.

1.40 automake.guide/Tests

Support for test suites
****

Automake supports a two forms of test suite.

If the variable 'TESTS' is defined, its value is taken to be a list of programs to run in order to do the testing. The programs can either be derived objects or source objects; the generated rule will look both in SRCDIR and '.'. The number of failures will be printed at the end of the run. The variable 'TESTS_ENVIRONMENT' can be used to set environment variables for the test run; the environment variable 'srcdir' is set in the rule.

If 'dejagnu' appears in 'AUTOMAKE_OPTIONS', then the a 'dejagnu'-based test suite is assumed. The value of the variable 'DEJATOOL' is passed as the '--tool' argument to 'runtest'; it defaults to the name of the package. The variables 'EXPECT', 'RUNTEST' and 'RUNTESTFLAGS' can also be overridden to provide project-specific values. For instance, you will need to do this if you are testing a compiler toolchain, because the default values do not take into account host and target names.

In either case, the testing is done via 'make check'.

1.41 automake.guide/Options

Changing Automake's Behavior
****

Various features of Automake can be controlled by options in the 'Makefile.am'. Such options are listed in a special variable named 'AUTOMAKE_OPTIONS'. Currently understood options are:

'gnits'
'gnu'

```
‘foreign’
     The same as the corresponding ‘--strictness’ option.

‘no-installman’
     The generated ‘Makefile.in’ will not cause man pages to be
     installed by default.  However, an ‘install-man’ target will still
     be available for optional installation.  This option is disallowed
     at ‘GNU’ strictness and above.

‘no-installinfo’
     The generated ‘Makefile.in’ will not cause info pages to be built
     or installed by default.  However, ‘info’ and ‘install-info’
     targets will still be available.  This option is disallowed at
     ‘GNU’ strictness and above.

‘ansi2knr’
‘path/ansi2knr’
     Turn on automatic de-ANSI-fication.  See
                  ANSI
                  .  If preceeded by a
     path, the generated ‘Makefile.in’ will look in the specified
     directory to find the ‘ansi2knr’ program.  Generally the path
     should be a relative path to another directory in the same
     distribution (though Automake currently does not check this).  It
     is up to you to make sure that the specified directory is built
     before the current directory; if ‘ansi2knr’ does not exist then
     the build will fail.

‘dejagnu’
     Cause ‘dejagnu’-specific rules to be generated.  See
                  Tests
                  .

‘dist-shar’
     Generate a ‘dist-shar’ target as well as the ordinary ‘dist’
     target.  This new target will create a shar archive of the
     distribution.

‘dist-zip’
     Generate a ‘dist-zip’ target as well as the ordinary ‘dist’
     target.  This new target will create a zip archive of the
     distribution.

‘dist-tarZ’
     Generate a ‘dist-tarZ’ target as well as the ordinary ‘dist’
     target.  This new target will create a compressed tar archive of
     the distribution; a traditional ‘tar’ and ‘compress’ will be
     assumed.  Warning: if you are actually using ‘GNU tar’, then the
     generated archive might contain nonportable constructs.

‘no-dependencies’
     This is similar to using ‘--include-deps’ on the command line, but
     is useful for those situations where you don’t have the necessary
     bits to make automatic dependency tracking work See
                  Dependencies
                  .
     In this case the effect is to effectively disable automatic
     dependency tracking.

VERSION
     A version number (eg `0.30') can be specified.  If Automake is not
     newer than the version specified, creation of the `Makefile.in'
     will be suppressed.

   Unrecognized options are diagnosed by `automake'.
```

1.42 automake.guide/Miscellaneous

```
                Miscellaneous Rules
*******************

   There are a few rules and variables that didn't fit anywhere else.
```

1.43 automake.guide/Tags

```
Interfacing to `etags'
======================

   `automake' will generate rules to generate `TAGS' files for use with
GNU Emacs under some circumstances.

   If any C source code or headers are present, then a `tags' target
will be generated for the directory.

   At the topmost directory of a multi-directory package, a `tags'
target file will be generated which, when run, will generate a `TAGS'
file that includes by reference all `TAGS' files from subdirectories.

   Also, if the variable `ETAGS_ARGS' is defined, a `tags' target will
be generated.  This variable is intended for use in directories which
contain taggable source that `etags' does not understand.

   Here is how Automake generates tags for its source, and for nodes in
its Texinfo file:
     ETAGS_ARGS = automake.in --lang=none \
      --regex='/^@node[ \t]+\([^,]+\)/\1/' automake.texi
```

   If you add filenames to ETAGS_ARGS, you will probably also want to
set TAGS_DEPENDENCIES. The contents of this variable are added
directly to the dependencies for the 'tags' target.

   Automake will also generate an 'ID' target which will run 'mkid' on
the source. This is only supported on a directory-by-directory basis.

1.44 automake.guide/Suffixes

Handling new file extensions
============================

   It is sometimes useful to introduce a new implicit rule to handle a
file type that Automake does not know about. If this is done, you must
notify GNU Make of the new suffixes. This can be done by putting a list
of new suffixes in the 'SUFFIXES' variable.

1.45 automake.guide/Built

Built sources
=============

   FIXME write this

1.46 automake.guide/Extending

When Automake Isn't Enough
****

   Sometimes 'automake' isn't enough. Then you just lose.

   Actually, 'automake's implicit copying semantics means that many
problems can be worked around by simply adding some 'make' targets and
rules to 'Makefile.in'. 'automake' will ignore these additions.

   There are some caveats to doing this. Although you can overload a
target already used by 'automake', it is often inadvisable,
particularly in the topmost directory of a non-flat package. However,
various useful targets have a '-local' version you can specify in your
'Makefile.in'. Automake will supplement the standard target with these
user-supplied targets.

   The targets that support a local version are 'all', 'info', 'dvi', 'check', 'install-data', 'install-exec', and 'uninstall'. Note that there are no 'uninstall-exec-local' or 'uninstall-data-local' targets; just use 'uninstall-local'. It doesn't make sense to uninstall just data or just executables.

For instance, here is how to install a file in '/etc':

```
  install-data-local:
          $(INSTALL_DATA) $(srcdir)/afile /etc/afile
```

Some targets also have a way to run another target, called a "hook", after their work is done. The hook is named after the principal target, with '-hook' appended. The targets allowing hooks are 'install-data', 'install-exec', 'dist', and 'distcheck'.

For instance, here is how to create a hard link to an installed program:

```
  install-exec-hook:
          ln $(bindir)/program $(bindir)/proglink
```

1.47 automake.guide/Distributing

Distributing 'Makefile.in's
****

Automake places no restrictions on the distribution of the resulting 'Makefile.in's. We still encourage software authors to distribute their work under terms like those of the GPL, but doing so is not required to use Automake.

Some of the files that can be automatically installed via the '--add-missing' switch do fall under the GPL; examine each file to see.

1.48 automake.guide/Examples

Some example packages
****

Here are some examples of how Automake can be used.

```
                  Automake's own use

                 Textutils
                  A deep hierarchy
```

1.49 automake.guide/Hello

```
The simplest GNU program
========================

   `hello' is renowned for its classic simplicity and versatility.
What better place to begin a tour?  The below shows what could be used
as the Hello distribution's `Makefile.am'.

     bin_PROGRAMS = hello
     hello_SOURCES = hello.c version.c getopt.c getopt1.c getopt.h
     hello_LDADD = @ALLOCA@
     info_TEXINFOS = hello.texi
     hello_TEXINFOS = gpl.texi

     EXTRA_DIST = testdata

     check-local: hello
             @echo expect no output from diff
             ./hello > test.out
             diff -c $(srcdir)/testdata test.out
             rm -f test.out

   Of course, Automake also requires some minor changes to
`configure.in'.  The new `configure.in' would read:

     dnl Process this file with autoconf to produce a configure script.
     AC_INIT(hello.c)
     AM_INIT_AUTOMAKE(hello, 1.3)
     AC_PROG_CC
     AC_PROG_CPP
     AC_PROG_INSTALL
     AC_STDC_HEADERS
     AC_HAVE_HEADERS(string.h fcntl.h sys/file.h)
     AC_ALLOCA
     AC_OUTPUT(Makefile)

   If Hello were really going to use Automake, the `version.c' file
would probably be deleted, or changed so as to be automatically
generated.
```

1.50 automake.guide/Tricky

```
A tricker example
=================

   Here is another, trickier example.  It shows how to generate two
programs (`ctags' and `etags') from the same source file (`etags.c').
The difficult part is that each compilation of `etags.c' requires
different `cpp' flags.

     bin_PROGRAMS = etags ctags
     ctags_SOURCES =
     ctags_LDADD = ctags.o
     ctags_DEPENDENCIES = ctags.o

     etags.o:
             $(COMPILE) -DETAGS_REGEXPS etags.c

     ctags.o:
             $(COMPILE) -DCTAGS -o ctags.o etags.c

   Note that `ctags_SOURCES' is defined to be empty - that way no
implicit value is substituted.  The implicit value, however, is used to
generate `etags' from `etags.o'.

   `ctags_LDADD' is used to get `ctags.o' into the link line, while
`ctags_DEPENDENCIES' exists to make sure that `ctags.o' gets built in
the first place.

   This is a somewhat pathological example.
```

1.51 automake.guide/Automake

```
Automake uses itself
====================

   Automake, of course, uses itself to generate its `Makefile.in'.
Since Automake is a shallow package, it has more than one
`Makefile.am'.  Here is the top-level `Makefile.am':

     ## Process this file with automake to create Makefile.in

     AUTOMAKE_OPTIONS = gnits
     MAINT_CHARSET = latin1
     PERL = @PERL@

     SUBDIRS = tests

     bin_SCRIPTS = automake
     info_TEXINFOS = automake.texi

     pkgdata_DATA = clean-kr.am clean.am compile-kr.am compile-vars.am \
     compile.am data.am depend.am \
     dist-vars.am footer.am header.am header-vars.am \
     kr-vars.am libraries-vars.am \
     libraries.am library.am mans-vars.am \
     program.am programs.am remake-hdr.am \
     remake-subd.am remake.am scripts.am subdirs.am tags.am tags-subd.am \
     tags-clean.am \
     texi-version.am texinfos-vars.am texinfos.am \
     libraries-clean.am programs-clean.am data-clean.am \
     COPYING INSTALL texinfo.tex \
     ansi2knr.c ansi2knr.1 \
     aclocal.m4

     ## These must all be executable when installed.
     pkgdata_SCRIPTS = config.guess config.sub install-sh mdate-sh mkinstalldirs

     # The following requires a fixed version of the Emacs 19.30 etags.
     ETAGS_ARGS = automake.in --lang=none \
      --regex='/^@node[ \t]+\([^,]+\)/\1/' automake.texi

     ## `test -x' is not portable.  So we use Perl instead.  If Perl
     ## doesn't exist, then this test is meaningless anyway.
     # Check to make sure some installed files are executable.
     installcheck-local:
      $(PERL) -e "exit ! -x '$(pkgdatadir)/config.guess';"
      $(PERL) -e "exit ! -x '$(pkgdatadir)/config.sub';"
      $(PERL) -e "exit ! -x '$(pkgdatadir)/install-sh';"
      $(PERL) -e "exit ! -x '$(pkgdatadir)/mdate-sh';"
      $(PERL) -e "exit ! -x '$(pkgdatadir)/mkinstalldirs';"

     # Some simple checks:
     # * syntax check with perl4 and perl5.
     # * make sure the scripts don't use 'true'
     # * expect no instances of '${...}'
     # These are only really guaranteed to work on my machine.
     maintainer-check: automake check
      $(PERL) -c -w automake
      @if grep '^[^#].*true' $(srcdir)/[a-z]*.am; then \
        echo "can't use 'true' in GNU Makefile" 1>&2; \
        exit 1;       \
      else :; fi
      @if test `fgrep '$${' $(srcdir)/[a-z]*.am | wc -l` -ne 0; then \
        echo "found too many uses of '\$${'" 1>&2; \
        exit 1;       \
      fi
      if $(SHELL) -c 'perl4.036 -v' >/dev/null 2>&1; then \
        perl4.036 -c -w automake; \
      else :; fi

     # Tag before making distribution.  Also, don't make a distribution if
     # checks fail.  Also, make sure the NEWS file is up-to-date.
     cvs-dist: maintainer-check
      @if sed 1q NEWS | grep -e "$(VERSION)" > /dev/null; then :; else \
        echo "NEWS not updated; not releasing" 1>&2; \
        exit 1;       \
      fi
      cvs tag `echo "Release-$(VERSION)" | sed 's/\./-/g'`
      $(MAKE) dist
```

As you can see, Automake defines many of its own rules, to make the

```
maintainer's job easier.  For instance the `cvs-dist' rule
automatically tags the current version in the CVS repository, and then
makes a standard distribution.

   Automake consists primarily of one program, `automake', and a number
of auxiliary scripts.  Automake also installs a number of programs
which are possibly installed via the `--add-missing' option; these
scripts are listed in the `pkgdata_SCRIPTS' variable.

   Automake also has a `tests' subdirectory, as indicated in the
`SUBDIRS' variable above.  Here is `tests/Makefile.am':

     ## Process this file with automake to create Makefile.in

     AUTOMAKE_OPTIONS = gnits

     TESTS = mdate.test vtexi.test acoutput.test instexec.test checkall.test \
     acoutnoq.test acouttbs.test libobj.test proginst.test acoutqnl.test \
     confincl.test spelling.test prefix.test badprog.test depend.test

     EXTRA_DIST = defs

   This is where all the tests are really run.  `defs' is an
initialization file used by each test script; it is explicitly mentioned
because `automake' has no way of automatically finding it.
```

1.52 automake.guide/Textutils

```
A deep hierarchy
================

   The GNU textutils are a collection of programs for manipulating text
files.  They are distributed as a deep package.  The textutils have only
recently been modified to use Automake; the examples come from a
prerelease.

   Here is the top-level `Makefile.am':

     SUBDIRS = lib src doc man

   In the `lib' directory, a library is built which is used by each
textutil.  Here is `lib/Makefile.am':

     noinst_LIBRARIES = tu

     EXTRA_DIST = rx.c regex.c

     tu_SOURCES = error.h getline.h getopt.h linebuffer.h \
     long-options.h md5.h regex.h rx.h xstrtod.h xstrtol.h xstrtoul.h \
     error.c full-write.c getline.c getopt.c getopt1.c \
     linebuffer.c long-options.c md5.c memchr.c safe-read.c \
     xmalloc.c xstrtod.c xstrtol.c xstrtoul.c

     tu_LIBADD = @REGEXOBJ@ @LIBOBJS@ @ALLOCA@
```

The ‘src’ directory contains the source for all the textutils - 23 programs in all. The ‘Makefile.am’ for this directory also includes some simple checking code, and constructs a ‘version.c’ file on the fly:

```
bin_PROGRAMS = cat cksum comm csplit cut expand fmt fold head join md5sum \
nl od paste pr sort split sum tac tail tr unexpand uniq wc

noinst_HEADERS = system.h version.h
DISTCLEANFILES = stamp-v version.c

INCLUDES = -I$(top_srcdir)/lib

LDADD = version.o ../lib/libtu.a

$(PROGRAMS): version.o ../lib/libtu.a

AUTOMAKE_OPTIONS = ansi2knr

version.c: stamp-v
stamp-v: Makefile
 rm -f t-version.c
 echo '#include <config.h>' > t-version.c
 echo '#include "version.h"' >> t-version.c
 echo 'const char *version_string = "'GNU @PACKAGE@ @VERSION@'";' \
   >> t-version.c
 if cmp -s version.c t-version.c; then \
   rm t-version.c;     \
 else          \
   mv t-version.c version.c;   \
 fi
 echo timestamp > $@

check: md5sum
 ./md5sum \
  --string="" \
  --string="a" \
  --string="abc" \
  --string="message digest" \
  --string="abcdefghijklmnopqrstuvwxyz" \
  --string="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789" \
  --string ↩
     ="1234567890123456789012345678901234567890123456789012345678901234567
      \
       | diff -c $(srcdir)/md5-test.rfc -
```

The ‘doc’ directory builds the info documentation for the textutils:

```
info_TEXINFOS = textutils.texi
```

And, last, the ‘man’ directory installs the man pages for all the textutils:

```
man_MANS = cat.1 cksum.1 comm.1 csplit.1 cut.1 expand.1 fmt.1 fold.1 head.1 \
join.1 md5sum.1 nl.1 od.1 paste.1 pr.1 sort.1 split.1 sum.1 tac.1 tail.1 \
tr.1 unexpand.1 uniq.1 wc.1
```

You can now see how easy it is to handle even a largish project using Automake.

1.53 automake.guide/Future

```
Some ideas for the future
*************************
```

Here are some things that might happen in the future:

* HTML support.

* The output will be cleaned up. For instance, only variables which are actually used will appear in the generated 'Makefile.in'.

* There will be support for automatically recoding a distribution. The intent is to allow a maintainer to use whatever character set is most convenient locally, but for all distributions to be Unicode or ISO 10646 with the UTF-8 encoding.

1.54 automake.guide/Variables

```
                Index of Variables
******************
```

1.55 automake.guide/Configure variables

```
          Index of Configure Variables and Macros
***************************************
```

```
  Optional

AC_PROG_INSTALL
  Requirements

AC_PROG_LEX
  Optional

AC_PROG_MAKE_SET
  Requirements

AC_PROG_RANLIB
  Optional

AC_PROG_YACC
  Optional

AC_REPLACE_FUNCS
  Optional

AC_REPLACE_GNU_GETOPT
  Optional

AC_STRUCT_ST_BLOCKS
  Optional

ALL_LINGUAS
  Optional

AM_C_PROTOTYPES <1>
  ANSI

AM_C_PROTOTYPES
  Optional

AM_FUNC_FNMATCH
  Optional

AM_FUNC_STRTOD
  Optional

AM_INIT_AUTOMAKE
  Requirements

AM_PROG_INSTALL
  Requirements

AM_WITH_REGEX
  Optional

jm_MAINTAINER_MODE
  Optional

LIBOBJS
  Optional

PACKAGE <1>
```

1.56 automake.guide/Targets

```
                Index of Targets
****************
```

```
  Extending

uninstall
  Install
```