Cook

A File Construction Tool

Reference Manual

Peter Miller
millerp@canb.auug.org.au

.

This document describes Cook version 2.8
and was prepared 2 February 1999.

NAME

cook – a file construction tool

DESCRIPTION

Put simply, the *cook* program is yet another make-oid.

The *cook* program is a tool for constructing files, and maintaining referential integrity between files. It is given a set of files to create, and recipes of how to create and maintain them. In any non-trivial program there will be prerequisites to performing the actions necessary to creating any file, such as include files. The *cook* program provides a mechanism to define these.

When a program is being developed or maintained, the programmer will typically change one file of several which comprise the program. The *cook* program examines the last-modified times of the files to see when the prerequisites of a file have changed, implying that the file needs to be recreated as it is logically out of date.

The *cook* program also provides a facility for implicit recipes, allowing users to specify how to form a file with a given suffix from a file with a different suffix. For example, to create *filename*.o from *filename*.c

- Cook has a simple but powerful string-based description language with many built-in functions. This allows sophisticated filename specification and manipulation without loss of readability or performance.
- Cook is able to use fingerprints to supplement file modification times. This allows build optimization without contorted rules.
- Cook is able to build your project with multiple parallel threads, with support for rules which must be single threaded. It is possible to distribute parallel builds over your LAN, allowing you to turn your network into a virtual parallel build engine.
- Cook can be configured with an explicit list of primary source files. This allow the dependency graph to be constructed faster by not going down dead ends, and also allows better error messages when the graph can't be constructed. This requires an accurate source file manifest.

ARCHIVE SITE

The latest version of *cook* is available on the Web from:

URL:	http://www.canb.auug.org.au/~millerp/	
File:	cook-2.8.README	# the README from the tar file
File:	cook-2.8.lsm	# LSM format description
File:	cook-2.8.spec	# RedHat package specification
File:	cook-2.8.rm.ps.gz	# PostScript of the Reference Manual
File:	cook-2.8.ug.ps.gz	# PostScript of the User Guide
File:	cook-2.8.tar.gz	# the complete source

This Web page also contains a few other pieces of software written by me. Please have a look if you are interested.

Cook is also carried by `sunsite.unc.edu` in its Linux archives. You will be able to find Cook on any of its mirrors.

URL:	ftp://sunsite.unc.edu/pub/Linux/devel/make/	
File:	cook-2.8.README	# the README from the tar file
File:	cook-2.8.lsm	# LSM format description

File:	cook-2.8.spec	# RedHat package specification
File:	cook-2.8.rm.ps.gz	# PostScript of the Reference Manual
File:	cook-2.8.ug.ps.gz	# PostScript of the User Guide
File:	cook-2.8.tar.gz	# the complete source

This site is extensively mirrored around the world, so look for a copy near you (you will get much better response).

MAILING LIST

A mailing list has been created so that users of *cook* may exchange ideas about how to use the *cook* program. Discussion may include, but is not limited to: bugs, enhancements, and applications. The list is not moderated.

The address of the mailing list is

```
cook-users@canb.auug.org.au
```

Please **do not** send subscribe requests to this address.

To subscribe to this mailing list, send an email message to `majordomo@canb.auug.org.au` with a message body containing the single line

```
subscribe cook-users
```

If you have an email address which is not readily derived from your mail headers (majordomo is only a Perl program, after all) you will need to use a message of the form:

`subscribe cook-users` *address*

where *address* is the email address to which you want messages sent.

The software which handles this mailing list **cannot** send you a copy of the *cook* program.

BUILDING COOK

Full instructions for building the *cook* program may be found in the *BUILDING* file included in this distribution.

COPYRIGHT

cook version 2.8

It should be in the *LICENSE* file included with this distribution.

AUTHOR

Peter Miller E-Mail: millerp@canb.auug.org.au
/\/* WWW: http://www.canb.auug.org.au/~millerp/

NEW IN THIS RELEASE

A number of features have been added to *cook* with this release. The following list is only a summary; for excruciating detail, and also acknowlegements of those who generously sent me feedback, please see the *etc/CHANGES.** files included in this distribution.

Version 2.8

• The remote *host-binding* code has been improved to cope with staggeringly long commands (which tended to make *rsh*(1) barf), and also wierd and wonderfull $SHELL settings.

• The #include directive now accepts more than one file, to be more symmetric with the #include-cooked directive.

• A bug has been fixed where cooktime gave an incorrect error message if setting the file's utimes failed.

• The configure script has been improved for use on non-UNIX systems.

• There is a new builtin [cook] function, a natural companion for the [cando] and [uptodate] functions. See the Cook User Guide for more information.

Version 2.7

• There is a new *cook_bom*(1) command (Bill Of Materials). This may be used to efficiently scan a directory tree for files, so that ingredients lists may be produced automatically. See *cook_bom*(1) for more information.

• There is a new assign-append statement, so you can now use += to append to the value of a variable. See the User Guide for more information.

• There is a new *gate-first* recipe flag, which causes the recipe gate to be evaluated before the ingredients are derived, rather than after.

• The *c_incl*(1) command has a new --interior-files option, so you can tell it about include files that don't exist yet. This is helpful when they are generated, *i.e.* they are interior files of the dependency graph, hence the option name.

• There is a new [interior-files] function, which returns the files interior to the dependency graph (constructed by a recipe), and a complementatry [leaf-files] function, which returns the leaf files of the dependency graph (not constructed by any recipe).

• There is a new ‘‘no-include-cooked-warning’’ flag, if you want to suppress the warnings about derived file dependencies in include-cooked files.

• There is a new *relative_dirname* built-in function, similar to the existing *dirname* function, but it returns ‘‘.’’ for files with no directory part, rather than the absolute path of the current directory.

Version 2.6

• Cook has been ported to Windows-NT using CygWin32. See the BUILDING file for details.

• There are two new functions (*dos-path* and *un-dos-path*) for use when invoking non-CygWin32 WindowsNT programs. See the Cook User Guide for more information.

• Fingerprints now work meaningfully with directories.

• A bug has been fixed in the pattern matching code. It would sometimes cause core dumps.

• A bug involving fingerprints in combination with the search_list has been fixed. Cook would occasionally conclude that a shallow target was up-to-date when a shallow ingredient was edited to be the same as a deeper ingredient.

• A bug has been fixed in cooktime. It would use an inappropriate timezone offset on some systems.

Release 2.5

• A problem which caused some tests to fail on Solaris' tmpfs now has a work-around.

• The ‘‘setenv’’ statement has finally been documented. It's been in the code tfor years, but I could never figure out why folks weren't using it!

• A number of build problems on various systems have been fixed.

Release 2.4

• There is a new form of dependencies. Known as cascaded dependencies, they allow the user to associate additional dependencies with an ingredient. For example, a C source file can be associated with cascaded include dependencies. This means that all files which depend on the C source file, also depend on the included files. The Cook Reference Manual has been updated to include this new functionality.

• There is a new section of the Cook Reference Manual giving suggestions and a template for building large projects.

• There is a new `[expr]` function, to calculate simple arithmetic expressions. See the User Guide for more information.

• There is a new c_incl -no-recursion option, to prevent scanning nested includes. This is of most use when combined with the new cascade dependencies.

• There is a new `[exists-symlink]` function, which may be used to test for the existence of symlinks. The `[exists]` function follows symbolic links, and is not useful when manipulating the links themselves.

Release 2.3

• There are 6 new special variables: graph_leaf_file, graph_leaf_pattern, graph_interior_file, graph_interior_pattern, graph_exterior_file and graph_exterior_pattern. These variables may be used to define the leaves of the derivation graph (the *accept* forms), and non-leave of the graph (the *reject* forms). This can make the graph derivation faster, and greatly improves some error messages. This functionality is of most use when you have an exact source file manifest, *e.g.* from a software configuration management system. See the User Guide for more information.

• The %0 pattern element has been extended to permit the matching of absolute paths.

Release 2.2.2

• There is a new statement type, allowing functions to be invoked as subroutines in any place where a command may be invoked. See the User Guide for more information.

• A number of problems with installing Cook have been fixed. This includes changing -mgm to -mm for the documnetation formatting, and missing include dependencies and missing rules for installing the man pages.

• There is a new ‘‘print’’ builtin function. When combined with the new function call statement, this provides a way of printing information without invoking ‘‘echo’’. See the User Guide for more information.

• Cook now defaults the language to ‘‘en’’ internally if neoither the LANG nor LANGUAGE environment variable was set. This gives better error messages.

Release 2.2.1

• A bug was fixed where a recipe would fail to trigger if some, but not all, of its targets were not present, but the existing targets were up-to-date. This bug was introduced in the inference engine re-write.

Release 2.2

• The *c_incl* utility has had two new languages added. It now understands M4, and also has an ‘‘optimistic’’ language which can scan many assemblers and even some high-level languages. See *c_incl*(1) for more information.

• The *c_incl* utility also has a new `--no-absolute-path` option, to supress scanning and reporting of such files. See *c_incl*(1) for more information.

• There is a new warning added for dependencies on derived ingredients when this information resides solely in derived cookbooks included using the `#include-cooked` facility. This assists in detecting problems which may preclude a successful ‘‘clean’’ build.

• This release adds a number of cookbook functions to the distrubuted cookbooks. These may be used by adding a

```
#include "functions"
```

line to your cookbook. See the Cook User Guide for more information.

• This release fixes a bug where the graph walking phase ignored interrupts until something went wrong.

• This release fixes a bug where *make2cook* did not correctly translate ‘‘%’’ into sematicly equivalent Cook constructs.

Release 2.1

• It is possible to specify that a command is to be executed on a specific machine or machines. This can be useful for restrictively licensed third party software tools.

• The parallel functionality has been extended to implement a virtual parallel machine on a LAN.

• Fingerprinting has been enhanced to be more informative, and to adjust file modification times so that subsequest fingerprint-less runs will not find too much to do.

• The `#line` directive is now available, for better diagnostics of generated cookbooks. The `__FILE__` and `__LINE__` variable are also available.

• There is now a `thread-id` variable, to obtain a thread-unique value for use in generating temporary file names or variable names, *etc*, which are unique to a thread.

• Added the `wordlist` function and the `command-line-goals` variable for compatibility with GNU Make. Updated *make2cook* to understand them.

Release 2.0.1

• An install problem in the generated Makefile, to do with the the manuals, has been fixed.

Release 2.0

Development of this release was generously supported by Endocardial Solutions, Inc.

• Parallel execution is now supported. If you have a multi-processor machine, you can specify the number of parallel processing threads with the -PARallel command line option, or via the *[parallel_jobs]* variable. By using the *[os node]* function, the *[parallel_jobs]* variable can be set appropriately for the host machine automatically by the cookbook. There is a new `single-thread` keyword to support single threading recipes which cannot be paralleized.

• The dependency graph is now constructed differently. This gives exactly the same results, but the order of evaluation of recipes is a little more random. This different graph construction is able to give better error messages, better -Reason information, and allows the introduction of parallel recipe evaluation if you have a multi-processor computer.

• Recipes which use *c_incl*(1) to calculate their dependencies in the ingredients section will need a small modification − they will need to use the `--Absent-Program-Ignore` option. See the User Guide for more information.

• You can now print pair-wise file dependencies by using the -PAirs option.

• You can now print a shell script which approximates the actions cook would take when building the targets by using the -SCript option.

• There is a new ‘‘shallow’’ recipe flag, allowing you to specify that the targets of a recipe are required to be in the top-level directory, not further down the `search_list` path.

• You may now define user-written functions in the cookbook to supplement the built-in functions. Your functions will be called in the same manner as built-in functions. There are new `function` and `return` keywords to support definition of functions.

• The progress indicators produced by the -STar option now have more detail: **+** means that the cook book is being read, ***** means that the graph is being constructed, and **#** means that the graph is being walked.

Release 1.11

• Fixed a bug in the pattern matching which caused %0 (when not at the start of the pattern) to fail to match the empty string.

• The install locations have been changed slightly to conform better to the GNU filesystem standards, and to take advantage of the additional install location options of the configure scripts generated by GNU Autoconf.

Release 1.10

• Error messages have been internationalized. It is now possible to get error messages in your native language, if it is supported.

• The cook command now accepts a -no-include-cooked option, to disable any cooking of the #include-cooked files.

• The cook -TRace option has been renamed -Reason. This is thought to more accurately reflect what it does.

• The cook -Reason output has been changed to cite cookbook file names and line numbers, in order to be more useful. In addition, more reason messages carry location information.

Release 1.9

• There are new ‘‘f77’’ and ‘‘g77’’ cookbooks, to allow Fortran sources, in addition to C sources.

• There is a new [options] function, which expands to the current settings of the command line options. This is useful for recursive cook directory structures. See the Reference Manual for more information.

• There is a new ‘‘recursive’’ cookbook, to assist in constructing recursive cook structures.

• The *find_libs* program now understands about shared libraries.

• A bug which made the builtin [glob] function far to generous has been corrected.

• A bug which caused some expression evaluation errors to be ignored has been corrected.

• The ‘‘set update’’ flag has been re-named the ‘‘set time-adjust’’ flag, to more closely describe what it does. The old name will continue to work indefinitely.

• There is a new ‘‘set time-adjust-back’’ flag, which sets recipe target times to be exactly one (1) second younger than the youngest ingredient. This is usually an adjustment into the recent past.

Release 1.8

• The fingerprint code has been improved to work better with the search_list functionality.

• The diagnostics have been improved when cook ‘‘don’t know how’’. A list of attempted ingredients is included in the error message.

• There is a new *mkdir* recipe flag. This creates recipe target directories before the recipe body is run. See the Reference Manual for more information.

• There is a new *unlink* recipe flag. This unlinks recipe targets before the recipe body is run. See the Reference Manual for more information.

• There is a new *recurse* recipe flag. This overrides the infinite loop recipe heuristic, allowing recipes to recuse upon themselves if one of their ingredients matches one of their targets. See the Reference Manual for more information.

Release 1.7

• The AIX code to handle archive files has been fixed.

• The fingerprint code now works on 64-bit systems.

Release 1.6

• Fixed a bug in the leading-dot removal code, and added an option to make it user-settable. Fixed a bug in the search_path depth code.

Release 1.5

• The *c_incl* program now correctly prints the names of absent include files, causing them to be cooked correctly in a greater number of cases.

• Recipes with no ingredients are now only applied if the target is absent. To still use the previous behaviour, use the "set force" clause on the recipe.

• It is now possible to supplement the last-modified time with a fingerprint, so cook does even fewer unnecesary recompilations than before. Put the statement

```
set fingerprint;
```

somewhere near the top of your *Howto.cook* file for this to be the default for your project.

• There is a new form of include directive:

```
#include-cooked filename...
```

When files are included in this way, *cook* will check to make sure they are up-to-date. If not, they will be cooked, and then *cook* will start again and re-read the cookbook. This is most often used for maintaining include-dependency files.

• **Cook** now configured using a program called *configure*, distributed with the package. The *configure* program is generated by GNU Autoconf. See the *BUILDING* file for more details.

• The semantics of *search_list* have been improved. It is now guaranteed that when ingredients change they result in targets earlier in the *search_list* being updated.

• There is now a *make2cook* translator, to translate *Makefile* files into *Howto.cook* files. Most of the GNU Make extensions are understood. There is no exact semantic mapping between *make* and *cook,* so manual editing is sometimes required. See *make2cook*(1) for more information.

• *Cook* now understands archive member references, in the same format as used by *make*, et al. Archive member references benefit from stat caching and fingerprinting, just as normal files do.

Release 1.4

• The *cook* program is now known to work on more systems. Most changes were aimed at improving portability, or avoiding problems specific to some systems.

• The GNU long option name convention is now understood. Option names for *cook* were always long, so this mostly consists of ignoring the extra leading '-'. The "--foo=bar" convention is also understood for options with arguments.

• Tests which fail now tell you what it was they were testing for. This will give the user some idea of what is happening.

NAME

cook – a file construction tool

SPACE REQUIREMENTS

You will need about 5MB to unpack and build the *cook* package. Your milage may vary.

BEFORE YOU START

There are a few pieces of software you may want to fetch and install before you proceed with your installation of cook.

GNU Gettext

The *cook* package has been internationalized. It can now print error messages in any of the supported languages. In order to do this, the GNU Gettext package must be installed *before* you run the configure script as detailed in the next section. This is because the configure script looks for it. On systems which use the GNU C library, version 2.0 or later, there is no need to explictly do this as GNU Gettext is included. Remember to use the GNU Gettext configure *--with-gnu-gettext* option if your system has native gettext tools.

Please note: if you install this package into */usr/local* (for example) you must ensure that the *./configure* script is told to also look in */usr/local/include* for include files (CFLAGS), and */usr/local/lib* for library files (LDFLAGS). Otherwise the not been installed.

GNU Groff

The documentation for the *cook* package was prepared using the GNU Groff package. This distribution includes full documentation, which may be processed into PostScript or DVI files at install time – if GNU Groff has been installed.

GNU Groff patch

There is a patch for GNU Groff available, which gives *groff*(1) and *gsoelim*(1) a **–B***path* option. This adds an include file search path, similar to *cpp*(1). The Makefile assumes this is available. The patch is at `http://www.canb.auug.org.au/˜millerp/groff-1.11.patch` or its mirrors.

Bison

If your operating system does not have a native *yacc*(1) you will need to fetch and install GNU Bison in order to build the *cook* package.

GCC

You may also want to consider fetching and installing the GNU C Compiler if you have not done so already. This is not essential.

The GNU FTP archives may be found at `prep.ai.mit.edu`, and are mirrored around the world.

SITE CONFIGURATION

The **cook** package is configured using the *configure* program included in this distribution.

The *configure* shell script attempts to guess correct values for various system-dependent variables used during compilation, and creates the *Makefile* and *common/config.h* files. It also creates a shell script *config.status* that you can run in the future to recreate the current configuration.

Normally, you just *cd* to the directory containing *cook*'s source code and type

```
% ./configure
...lots of output...
%
```

If you're using *csh* on an old version of System V, you might need to type

```
% sh configure
```

...lots of output...
%

instead to prevent *csh* from trying to execute *configure* itself.

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* using the quiet option; for example,

```
% ./configure --quiet
%
```

There is a known problem with GCC 2.8.3 and HP/UX. You will need to set `CFLAGS = -O` in the generated Makefile. (The configure script sets it to `CFLAGS = -O2`.) This is because the code optimization breaks the fingerprints. If test 46 fails (see below) this is probably the reason.

To compile the **cook** 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 the *configure* script. *configure* automatically checks for the source code in the directory that *configure* is in and in .. (the parent directory). 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, *configure* will arrange for the *make install* command to install the **cook** package's files in */usr/local/bin*, */usr/local/lib*, */usr/local/share* and */usr/local/man*. There are a number of options which allow you to control the placement of these files.

`--prefix=`*PATH*
: This specifies the path prefix to be used in the installation. Defaults to */usr/local* unless otherwise specified.

`--exec-prefix=`*PATH*
: You can specify separate installation prefixes for architecture-specific files files. Defaults to *${prefix}* unless otherwise specified.

`--bindir=`*PATH*
: This directory contains executable programs. On a network, this directory may be shared between machines with identical hardware and operating systems; it may be mounted read-only. Defaults to *${exec_prefix}/bin* unless otherwise specified.

`--datadir=`*PATH*
: This directory contains installed data, such as the documentation and cookbooks distributed with Cook. On a network, this directory may be shared between all machines; it may be mounted read-only. Defaults to *${prefix}/share/cook* unless otherwise specified. A "cook" directory will be appended if there is none in the specified path.

`--libdir=`*PATH*
: This directory contains installed data, such as the error message catalogues. On a network, this directory may be shared between machines with identical hardware and operating systems; it may be mounted read-only. Defaults to *${exec_prefix}/lib/cook* unless otherwise specified. A "cook" directory will be appended if there is none in the specified path.

`--mandir=`*PATH*
: This directory contains the on-line manual entries. On a network, this directory may be shared between all machines; it may be mounted read-only. Defaults to *${prefix}/man* unless otherwise

specified.

configure ignores most other arguments that you give it; use the `--help` option for a complete list.

On systems that require unusual options for compilation or linking that the *cook* 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`
...lots of output...
$

Here are the *make* variables that you might want to override with environment variables when running *configure*.

Variable: CC
: C compiler program. The default is *cc*.

Variable: CPPFLAGS
: Preprocessor flags, commonly defines and include search paths. Defaults to empty. It is common to use `CFLAGS=-I/usr/local/include` to access other installed packages.

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

Variable: LIBS
: Libraries to link with, in the form `-l`*foo* `-l`*bar*. The *configure* script will append to this, rather than replace it. It is common to use `LIBS=-L/usr/local/lib` to access other installed packages.

If you need to do unusual things to compile the package, the author encourages you to figure out how *configure* could check whether to do them, and mail diffs or instructions to the author so that they can be included in the next release.

BUILDING COOK

All you should need to do is use the

% `make`
...lots of output...
%

command and wait. When this finishes you should see a directory called *bin* containing nine files: *c_incl*, *cook*, *cookfp*, *cooktime*, *find_libs*, *make2cook* and *roffpp*.

cook *cook* program is a file construction tool, and may invoke the following tools in some of its recipes.

cookfp The *cookfp* program is a utility distributed with *cook* which calculates the fingerprints of files. It uses the same algorithm as the fingerprints used by *cook* itself. For more information, see *cook*(1) and *cookfp*(1).

cooktime
The *cooktime* program is a utility distributed with *cook* which allows the time-last-modified and time-last-accessed stamps of files to be set to specific times. For more information, see *cooktime*(1).

c_incl The *c_incl* program is a utility distributed with *cook* which examines C files and determines all the files it includes directly and indirectly. For more information, see *c_incl*(1).

find_libs
The *find_libs* program is a utility distributed with *cook* which tracks down the names of library files, given cc-style library options (-L and -l). For more information, see *find_libs*(1).

make2cook
The *make2cook* program is a utility to help convert Makefiles into cookbooks. An exact 1:1 semantic mapping is not possible, so some addition editing is often required.

roffpp The *roffpp* program is a utility distributed with *cook* which acts as a proprocessor for *roff files, removing source (.so) directives. It accepts include search path command line options just as */lib/cpp* does. For more information, see *roffpp*(1).

You can remove the program binaries and object files from the source directory by using the

```
% make clean
...lots of output...
%
```

command. To remove all of the above files, and also remove the *Makefile* and *common/config.h* and *config.status* files, use the

```
% make distclean
...lots of output...
%
```

command.

The file *etc/configure.in* is used to create *configure* by a GNU program called *autoconf*. You only need to know this if you want to regenerate *configure* using a newer version of *autoconf*.

TESTING COOK

The *cook* program comes with a test suite. To run this test suite, use the command

```
% make sure
...lots of output...
Passed All Tests
%
```

The tests take a few seconds each, with a few very fast, and a couple very slow, but it varies greatly depending on your CPU.

If all went well, the message

```
Passed All Tests
```

should appear at the end of the make.

Known Problems

If test 46 fails, this is often caused by optimization bugs in gcc. Edit the `Makefile` to change `-O2` to `-O`, and delete `common/fp/*.o` to cause them to be re-built. Make and test again.

If you are using Sun's tmpfs file system as your /tmp directory, some tests will fail. This is because the tmpfs file system does not support file locking. Set the COOK_TMP environment variable to somewhere else before running the tests. Something like

```
% setenv COOK_TMP /usr/tmp
%
```

is usually sufficient if you are using C shell, or

```
$ COOK_TMP=/usr/tmp
$ export COOK_TMP
```

$

if you are using Bourne shell. Remember, this must be done before running the tests.

Tests 121 and 122 can sometimes have problems on Solaris, where they give false negatives. If you work out why, please let the author know.

INSTALLING COOK

As explained in the *SITE CONFIGURATION* section, above, the *cook* package is installed under the */usr/local* tree by default. Use the `--prefix=`*PATH* option to *configure* if you want some other path. More specific installation locations are assignable, use the `--help` option to *configure* for details.

All that is required to install the *cook* package is to use the

```
% make install
...lots of output...
%
```

command. Control of the directories used may be found in the first few lines of the *Makefile* file and the other files written by the *configure* script; it is best to reconfigure using the *configure* script, rather than attempting to do this by hand.

PRINTED MANUALS

The easiest way to get copies of the manuals is to get the *cook.2.8.rm.ps.gz* and *cook.2.8.ug.ps.gz* files from the archive site. These are compressed PostScript files of the Reference Manual and User Guide, respectively. The Reference Manual (about 36 pages) contains the README file, the BUILDING file and internationalization notes, as well as all of the manual pages for all of the commands. The User Guide (about 56 pages) tells you how to use the Cook package.

This distribution contains the sources to all of the documentation for *cook*. The author used the GNU groff package and a postscript printer to prepare the documentation. If you do not have this software, you will need to substitute commands appropriate to your site.

If you have the GNU Groff package installed *before* you run the *configure* script, the *Makefile* will contain instructions for constructing the documentation. If you alreday used the *make* command, above, this has already been done. The following command

```
% make groff_all
...lots of output...
%
```

can be used to do this explicitly, if you managed to get to this point without doing it. Please note that there may be some warnings from groff, particularly for the `.txt` files; this is normal.

Once the documents have been formatted, you only need to print them. The following command

```
% lpr lib/en/refman.ps lib/en/user-guide.ps
%
```

will print the English PostScript version of the Reference Manual and the User Guide. Watch the *make* output to see what other versions are available.

GETTING HELP

If you need assistance with the *cook* program, please do not hesitate to contact the author at

```
Peter Miller <millerp@canb.auug.org.au>
```

Any and all feedback is welcome.

When reporting problems, please include the version number given by the

```
% cook -version
```

cook version *2.8.D001*
...warranty disclaimer...
%

command. Please do not send this example; run the program for the exact version number.

In the *common/main.h* file, there is a define of *DEBUG* in comments. If the comments are removed, extensive debugging is turned on. This causes some performance loss, but performs much run-time checking and adds the **-TRACIng** command line option.

When the **-TRACIng** option is followed by one or more file names, it turns on execution traces in those source files. It is best to put this option on the end of the command, so that the names of the files to be traced are not confused with any other filenames or strings on the command line.

WINDOWS-NT

It is possible to build Cook for Windows-NT. I have done this using the Cygnus freeware CygWin32 system, and I believe it has also once been done using the commercial NutCracker system. This document only describes the CygWin32 port.

The Source

You need to FTP the CygWin32 system from Cygnus. It can be found at

http://sourceware.cygnus.com/cygwin/

and then follow the links. The version I used was B19.1, but with the ‘‘most recent’’ DLL update from Sergey Okhapkin, found at

http://www.lexa.ru/sos/

in the ‘‘coolview’’ package. Don’t ask me why it is called coolview. Don’t ask me why Cygus is not coordinating the updates.

Mounting Things

You need to mount a directory onto `/tmp`, or lots of things, and especially *bash*(1), don’t work. If you are in a heavily networked environment, like me, you need to know that using a networked drive for `/tmp` just doesn’t work. I have no idea why. Use

mount C:/temp /tmp

instead. (Or some other local drive.)

Just a tip for all of you who, like me, know UNIX much better than you know Windows-NT: the left-hand mount argument needs to be specified with a drive letter (*e.g.* `C:`*) rather than with a double slash (e.g. not* `//C`*) unless its Windows-NT name starts with* \\.

You need to mount the Cygnus bin directory at `/bin`, otherwise shell scripts that start with `#!/bin/sh` don’t work, among other things. This includes the `./configure` script, and the scripts it writes (*e.g.* `config.status`).

mount `Cygnus-Dir`/H-i386-cygwin32/bin /bin

You need to mount the Cygnus install directory at `/cygnus/b19`, otherwise commands like *bison*(1) can’t find their library files.

mount *Cygnus-Dir* /cygnus/b19

You will want to mount your various network drives onto the same places they appear on your UNIX hosts. This means that your cookbooks will work without change, even if they contain absolute paths. And your users don’t need to learn two names for all the source files.

Don’t forget your home directory. The trick is to set HOME in the cygnus.bat file, before bash starts. (How you do this with one batch file and multiple users I haven’t yet figured out.)

You also need to set the LOGNAME and USER environment variables appropriately, or test 14 will fail.

Mounts persist across CygWin32 sessions. They are stored in a registry file somewhere. You will not need to do all this every time!

Configure

The configure and build step should be the same as for UNIX, as described above. All the problems I encountered were to do with getting the mounts just right. (But expect it to be dog slow compared to Linux or FreeBSD on the same box.)

The configure step is almost the same as for UNIX. I know you are itching to get typing, but read throught to the install section before you configure anything. Please note that "./" doesn't seem to work for shell scripts...

bash$ `sh configure`
...lots of output...
bash$

Build

The build step is exactly the same as for UNIX, and you shouldn't notice any difference...

bash$** make **bash$

Test

All of the tests should pass, you only need to run them to convince yourself the build worked... (a constant surprize to me, I must say!)

bash$ make sure *...lots of output...* **Passed All Tests bash$**

Install

Installing the software works as usual, though you need to make some chioces right at the start (I told you to read this all the way through first). If you want to use the "*/usr/local*" prefix (or any other install prefix) you mount it right at the start. For anything other than the "*/usr/local*" default prefix, you also needed to give a "**--prefix=***blahblah" argument to the configure script, right at the start.*

bash$ *make install ...lots of output...* **bash$**

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

NAME

Internationalization

DESCRIPTION

The Cook package has gone international; it can now speak many languages. This is accomplished by using the GNU Gettext library and utilities. In order to do this, is is necessary to install GNU Gettext prior to configuring, making and installing the Cook package, as described in the *BUILDING* file.

Internationalization

This is the process of identifying all of the error messages in the source code, and providing error message catalogues in a variety of languages. The error message identification was performed by the Cook package's author, and the various GNU translation teams provided the translations. Users of the Cook package do not need to do anything to internationalize it, this has already been done.

Localization

The programs in the Cook package are "localizable" when properly installed; the programs they contain can be made to speak your own native language.

By default, the Cook package will be installed to allow translation of messages. It will automatically detect whether the system provides a usable 'gettext' function.

INSTRUCTIONS FOR USERS

As a user, if your language has been installed for this package, you only have to set the 'LANG' environment variable to the appropriate ISO 639 two-letter code prior to using the programs in the package. For example, let's suppose that you speak German. At the shell prompt, merely execute

```
setenv LANG de
```

(in 'csh'), or

```
LANG=de; export LANG
```

(in 'sh'). This can be done from your *.cshrc* or *.profile* file, setting this automatically each time you login.

An operating system might already offer message localization for many of its programs, while other programs have been installed locally with the full capabilities of GNU Gettext. Using the GNU Gettext extended syntax for the 'LANG' environment variable may break the localization of already available through the operating system. In this case, users should set both the 'LANGUAGE' and 'LANG' environment variables, as programs using GNU Gettext give preference to the 'LANGUAGE' environment variable.

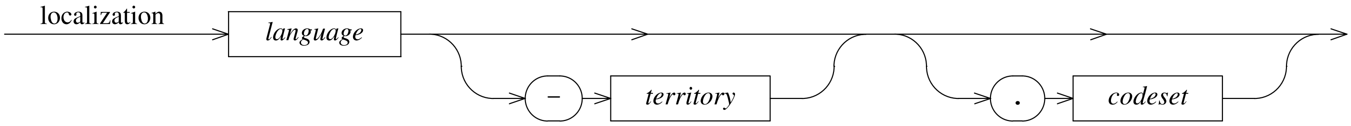
For example, some Swedish users would rather read translations in German when Swedish is not available. This is easily accomplished by setting 'LANGUAGE' to 'sv:de' while leaving 'LANG' set to 'sv'.

DIRECTORY STRUCTURE

All files which may require translation are located below the *lib* directory of the source distribution. It is organized as one directory below *lib* for each localization. Localizations include all documentation as well as the error messages.

Localization Directory Names

Each localization is contained in a sub-directory of the *lib* directory, with a unique name. Each localization sub-directory has a name of the form:



language is one of the 2-letter names from the ISO 639 standard. See *http://www.ics.uci.edu/pub/ietf/-http/related/iso639.txt* for a list.

territory is one of the 2-letter country codes from the ISO 3166 standard. See *ftp://rs.internic.net/-netinfo/iso3166-countrycodes* for a list.

codeset is one of the codeset names defined in RFC 1345. This simplifies the task of moving localizations between charsets, because GNU Recode understands them. See *http://info.internet.isi.edu/1s/in-notes/rfc/files/rfc1345.txt* for a list.

Here are some examples of localization names:

Name	Description
en.ascii	English, ASCII encoding
en_us.ascii	English with US spelling
de.latin1	German, Latin-1 encoding

Localization Directory Contents

Each localization sub-directory in turn contains sub-directories. These are:

Directory	Contents
LC_MESSAGES	The error message (PO) files
building	The BUILDING file
man1	The section 1 manual entries
readme	The README file
refman	The Reference Manual
user-guide	The User Guide

The structure is identical below each of the localization directories. The sub-directories of all localizations will have the same names.

INSTRUCTIONS FOR TRANSLATORS

When translating the error messages, all of the substitutions described in *cook_sub*(5) are also available. Substitution variable names and function names may be abbreviated, in the same way that command line options are abbreviated, but abbreviation should probably be avoided. Substitution names will *never* be internationalized, otherwise the substitutions will stop working, Catch-22.

While Cook was written by an English speaker, the English localization is necessary, to translate the ‘‘terse programmer’’ style error messages into something more user friendly.

Messages which include command line options need to leave the command line options untranslated, because they are not yet internationalized, though they may be one day.

Each LC_MESSAGES directory within each localization contains a number of PO files. There is one for each program in the Cook package, plus one called `common.po` containing messages common to all of them. The MO file for each program is generated by naming the program specific PO file and also the common PO file.

NAME

c_incl - determine dependencies

SYNOPSIS

c_incl [*option*...] *filename*

c_incl -Help

c_incl -VERSion

DESCRIPTION

The *c_incl* program is used to traverse source files looking for include dependencies suitable for `[collect]`ion by cook.

Several input languages are supported, see the options list for details.

OPTIONS

The following options are understood.

-C The source file is a C source file. It is assumed that it will have the dependencies resolved by the *cpp*(1) command. The same include semantics as the *cpp*(1) command will be employed. This is the default. This is short-hand for ‘‘--language=c’’

--Language=*name*

This option may be used to specify the language of the source file. Know names include ‘‘C’’, ‘‘M4’’, ‘‘optimistic’’ and ‘‘roff’’.

The ‘‘optimistic’’ language will take on almost anything. It accepts an `include` keyword in any case, including mixed, with leading white space, but at most one leading punctuation character. It assumes that the filename follows the include keyword and does not contain white space, and does not start or end with punctuation characters (it strips off any it may find). The rest of the line is ignored. The drawback is that it will sometimes recognise commands and other text as unintended include directives, hence the name. This is often used to recognise include directives in a wide variety of assembler input.

-Roff The source file is a *roff source file. It is assumed that it will have the dependencies resolved by the *roffpp*(1) command. The same include semantics as the *roffpp*(1) command will be employed. This is short-hand for ‘‘--language=roff’’

-Verbose

Tell what is happening. **-I***path*
Specify include path, a la *cc*(1).

-I-

Any directories you specify with **-I** options before the **-I-** option are searched only for the case of *#include "file"*; they are not searched for *#include <file>*.

If additional directories are specified with **-I** options after the **-I-**, these directories are searched for all *#include* directives. (Ordinarily all **-I** directories are used this way.)

In addition, the *-I-* option inhibits the use of the current directory (where the current input file came from) as the first search directory for *#include "file"*. There is no way to override this effect of *-I-*. With *-I.* you can specify searching the directory which was current when c_incl was invoked. That is not exactly the same as what the preprocessor does by default, but it is often satisfactory.

The *-I-* option does not inhibit the use of the standard system directories for header files. Thus,

-I- and *-No_System* are independent.

-Absolute_Paths
This option may be used to allow absolute paths in the output. This is usually the default.

-No_Absolute_Paths
This option may be used to exclude absolute paths from the output.

-Absent_Local_Ignore
For files included using a *#include "filename.h"* directive, ignore the file if it cannot be found.

-Absent_Local_Mention
For files included using a *#include "filename.h"* directive, print the file name even if the file cannot be found. This is the default (it probably needs to be built).

-Absent_Local_Error
For files included using a *#include "filename.h"* directive, print a fatal error if the file cannot be found.

-Absent_System_Ignore
For files included with a *#include <filename.h>* directive, ignore the file if it cannot be found. This is the default (it was probably ifdef'ed out).

-Absent_System_Mention
For files included with a *#include <filename.h>* directive, print the file name even if the file cannot be found.

-Absent_System_Error
For files included with a *#include <filename.h>* directive, print a fatal error if the file cannot be found.

-Absent_Program_Ignore
If the file named on the command line cannot be found, behave as if the file were found, but was empty.

-Absent_Program_Error
If the file named on the command line cannot be found, print a fatal error message. This is the default.

-Escape_Newlines
This option may be used to request that newlines in the output are escaped with backslash ("\") characters.

-Help
Give information on how to use *c_incl*.

-VERSion
Tell what version of *c_incl* is being run.

-Interior_Files *filename*...
This option may be used to tell *c_incl* about include files which don't exist yet. This is because they are interior to the dependency graph, but *cook*(1) hasn't finished walking it yet. Often used with Cook's `[interior-files]` function. (**Note:** the *filename* list has an arbitrary number of files; it ends at the next option or end-of-line, so you need to be careful where you put the input filename.)

-No_System
Do not search the */usr/include* directory. By default this is searched last. This option implies the -No_Absolute_Paths option, unless explicitly contradicted.

-CAche
This option may be used to turn caching on. This is the default.

-No_Cache
This option may be used to turn caching off.

-PREfix *string*
This option may be used to print a string before any of the filenames are printed. It will not be printed if no file names are printed.

-SUFfix *string*
This option may be used to print a string after all of the filenames are printed. It will not be printed if no file names are printed.

-Output *filename*
This option may be used to specify the output file. Defaults to the standard output if not set.

-No_Source_Relative_Includes
This option will give a fatal error if a *#include "filename.h"* directive is used. This is necessary when you are using Cook's `search_list` functionality to stitch together a baseline and a private work area.

-RECursion
This option may be used to specify that nested include files are to be scanned, so that their includes may also be discovered. This is the default.

-No_RECursion
This option may be use to specify that nested include files are *not* to be scanned. This option is recommended for use with the Cook `cascade-for` recipes. This option implies -**No_Cache**, unless a **-Cache** option is specified.

-Remove_Leading_Path *path*
This option may be used to remove path prefixes from the included filenames. May be used more than once. This is necessary when you are using Cook's `search_list` functionality to stitch together a baseline and a private work area; usually as "`[prepost "-rlp=" "" [search_list]]`"

-Substitute_Leading_Path *from to*
This option may be used to modify path prefixes from the included filenames. May be used more than once. This is necessary when you are performing heterogeneous builds in the same directory tree. By using an "arch" variable to hold the architecture, and placing each architecture's objects in a separate directory tree, this option may be used as "`-slp [arch] "'[arch]'"`" (The outer quotes protect from Cook, the inner quotes protect from the shell.) If you need more intricate editing, used *sed*(1).

Any other options will generate an error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *c_incl* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

CACHING

The caching mechanism use by the *c_incl* program caches the results of searching files for include files (in a file called *.c_inclrc* in the current directory). The cache is only refreshed when a file changes.

The use of this cache has been shown to dramatically increase the performance of the *c_incl* program. Typically, only a small proportions files in a project change between builds, resulting in a very high cache hit rate.

When using caching, always use the same command line options, otherwise weird and wonderful things will happen.

The *.c_inclrc* file is a binary file. If you wish to rebuild the cache, simply delete this file with the *rm*(1) command. Being a binary file, the *.c_inclrc* file is not portable across machines or operating systems, so you will need to delete it when you move your sources. It is a binary file for performance.

Accesses to the *.c_inclrc* file use file locking, so recipies using *c_incl* need not use the `single-thread` clause.

EXIT STATUS

The *c_incl* command will exit with a status of 1 on any error. The *c_incl* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

c_incl version 2.8

The *c_incl* program comes with ABSOLUTELY NO WARRANTY; for details use the '*c_incl -VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*c_incl -VERSion License*' command.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

NAME

cook – a file construction tool

SYNOPSIS

cook [*option*...][*filename*...]
cook -Help
cook -VERSion

DESCRIPTION

The *cook* program is a tool for constructing files. It is given a set of files to create, and instructions detailing how to construct them. In any non-trivial program there will be prerequisites to performing the actions necessary to creating any file, such as extraction from a source-control system. The *cook* program provides a mechanism to define these.

When a program is being developed or maintained, the programmer will typically change one file of several which comprise the program. The *cook* program examines the last-modified times of the files to see when the prerequisites of a file have changed, implying that the file needs to be recreated as it is logically out of date.

The *cook* program also provides a facility for implicit recipes, allowing users to specify how to form a file with a given suffix from a file with a different suffix. For example, to create *filename***.o** from *filename***.c**

Options and filenames may be arbitrarily mixed on the command line; no processing is done until all options and filenames on the command line have been scanned.

The *cook* program will attempt to create the named files from the recipes given to it. The recipes are contained in a file called *Howto.cook* in the currect directory. This file may, in turn, include other files containing additional recipes.

If no *filename*s are given on the command line the targets of the first recipe defined are cooked.

OPTIONS

The valid options for *cook* are listed below. Any other options (words on the command line beginning with '**-**') will cause a diagnostic message to be issued.

-Action
Execute the commands given in the recipes. This is the default.

-No_Action
Do not execute the commands given in the recipes.

-Book *filename*
Tells cook to used the named cookbook, rather than the default ''Howto.cook'' file.

-CAScade
This option may be used to enable the use of cascaded ingredients. This is the default.

-No_CAScade
This option may be used to disable the use of cascaded ingredients.

-Continue
If cooking a target should fail, continue with other recipes for which the failed target is not an ingredient, directly or indirectly.

-No_Continue
If cooking a target should fail, *cook* will exit. This is the default.

-Errok
When a command is executed, the exit code will be ignored.

-No_Errok
When a command is executed, if the exit code is positive it will be deemed to fail, and thus the recipe containing it to have failed. This is the default.

-FingerPrint
When *cook* examines a file to determine if it has changed, it uses the last-modified time information available in the file system. There are times when this is altered, but the file contents do not actually change. The fingerprinting facility examines the file contents when it appears to have changed, and compares the old fingerprint against the present file contents. (See *cookfp*(1) for a description of the fingerprinting algorithm.) If the fingerprint did not change, the last-modified time in the file system is ignored. Note that this has implications if you are in the habit of using the *touch*(1) command – *cook* will do nothing until you actually change the file.

-No_FingerPrint
Do not use fingerprints to supplement the last-modified time file information. This is the default.

-Force
Always perform the actions of recipes, irrespective of the last-modified times of any of the ingredients. This option is useful if something beyond the scope of the cookbook has been modified; for example, a bug fix in a compiler.

-No_Force
Perform the actions of the recipes if any of the ingredients are logically out of date. This is the default.

-Help
Provide information about how to execute *cook* on *stdout*, and perform no other function.

-Include *filename*
Search the named directory before the standard places for included cookbooks. Each directory so named will be scanned in the order given. The standard places are *$HOME/.cook* then */usr/local/share/cook*.

-Include_Cooked
This option may be used to require the cooking of files named on *#include-cooked* and *#include-cooked-nowarn* include lines in cookbooks. The files named will be included, if present. If the files named need to be updated or created, this will be done, and then the cookbook re-read. This is the default.

-No_Include_Cooked
This option may be used to inhibit the implicit cooking of files named on *#include-cooked* and *#include-cooked-nowarn* include lines in cookbooks. The files will be included, if present, but they will not be updated or created, even if required.

-Include_Cooked_Warning
This option enables the warnings about derived dependencies in derived cookbooks. This is usually the default.

-No_Include_Cooked_Warning
This option disables the warnings about derived dependencies in derived cookbooks.

-List
Causes *cook* to automatically redirect the *stdout* and *stderr* of the session. Output will continue to come to the terminal, unless *cook* is executing in the background. The name of the file will be the name of the cookbook with any suffix removed and "`.list`" appended; this will usually be *Howto.list*. This is the default.

-List *filename*
Causes *cook* to automatically redirect the *stdout* and *stderr* of the session into the named file. Output will continue to come to the terminal, unless *cook* is executing in the background.

-No_List
No automatic redirection of the output of the session will be made.

-No_List *filename*
No automatic redirection of the output of the session will be made, however subsequent **-List** options will default to listing to the named file.

-Meter
After each command is executed, print a summary of the command's CPU usage.

-No_Meter
Do not print a CPU usage summary after each command. This is the default.

-Pairs
This option may be used to generate a list of pair-wise file dependencies, similar to *lorder*(1) output. This may be used to draw file dependency diagrams. It can also be useful when debugging cookbooks.

-PARallel [*number*]
This option may be used to specify the number of parallel executions threads. The number defaults to 4 if no specific number of threads is specified. See also the *parallel_jobs* variable.

Use of this option on single-processor machines needs to be done with great care, as it can bring other processing to a complete halt. Several users doing so simultaneously on a multi-processor machine will have a similar effect. It is also to rapidly run out of virtual memory and temporary disk space if the parallel tasks are complex.

-No_PARallel
This option may be used to specify that a single execution thread is to be used. This is the default.

-Precious
When commands in the body of a recipe fail, do not delete the targets of the recipe.

-No_Precious
When commands in the body of a recipe fail, delete the targets of the recipe. This is the default.

-Reason
Two options are provided for tracing the inferences **cook** makes when attempting to cook a target. The **-Reason** option will cause *cook* will emit copious amounts of information about the inferences it is making when cooking targets. This option may be used when you think **cook** is acting strangely, or are just curious.

-No_Reason
This option may be used to cause *cook* will not emit information about the inferences it is making when cooking targets. This is the default.

-SCript
This option may be used to request a shell script be printed on the standard output. This shell script may be used to construct the files; it captures many of the semantics of the cookbook. This can be useful when a project needs to be distributed, and the recipients do not have *cook(1)* installed. It can also be very useful when debugging cookbooks.

-Silent
Do not echo commands before they are executed.

-No_Silent
Echo commands before they are executed. This is the default.

-STar
Emit progress indicators once a second. These progress indicators include

+	Reading the cookbook
-	Executing a collect function
*	Building the dependency graph
#	Walking the dependency graph

-No_STar
Do not emit progress indicators. This is the default.

-Strip_Dot
Remove leading "./" from filenames before attempting to cook them; applies to all filenames and all recipes. This is the default.

-No_Strip_Dot
Leave leading "./" on filenames while cooking.

-Touch
Update the last-modified times of the target files, rather than execute the actions bound to recipes. This can be useful if you have made a modification to a file that you know will make a system of files logically out of date, but has no significance; for example, adding a comment to a widely used include file.

-No_Touch
Execute the actions bound to recipes, rather than update the last-modified times of the target files. This is the default.

-TErminal
When listing, also send the output stream to the terminal. This is the default.

-No_TErminal
When listing, do not send the output to the terminal.

-Update
This option causes **cook** to check the last-modified time of the targets of recipes, and updates them if necessary, to make sure they are consistent with (younger than) the last-modified times of the ingredients. This results in more system calls, and can slow things down on some systems.

-No_Update
Do not update the file last-modified times after performing the body of a recipe. This is the default.

*name***=***value*
Assign the *value* to the named variable. The value may contain spaces if you can convince the shell to pass them through.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *cook* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *cook* command will exit with a status of 1 on any error. The *cook* command will only exit with a status of 0 if there are no errors.

FILES

The following files are used by **cook**:

Howto.cook
This file contains instructions to *cook* for how to construct files.

/usr/local/share/cook
This directory contains "system" cookbooks for various tools and activities.

.cook.fp This text file is used to remember fingerprints between invokations.

ENVIRONMENT VARIABLES

The following environment variables are used by **cook**:

COOK May be set to contain command-line options, changing the default behaviour of *cook*. May be overridden by the command line.

PAGER Use to paginate the output of the **-Help** and **-VERSion** options. Defaults to *more*(1) if not set.

COOK_AUTOMOUNT_POINTS
A colon-separated list of directories which the automounter may use to mount file systems. Use with extreme care, as this distorts Cook's idea of the shape of the filesystem.

This feature assumes that paths below the automounter's mount directory are echoes of paths without it. *E.g.* When `/home` is the trigger, and `/tmp_mnt/home` is where the on-demand NFS mount is performed, with `/home` appearing to processes to be a symlink.

This is the behavior of the Sun automounter. The AMD automounter is capable of being configured in this way, though it is not typical of the examples in the manual. Nor is it typical of the out-of-the-box Linux AMD configuration in many distributions.

Defauls to "`/tmp_mnt:/a:/.automount`" if not set.

COPYRIGHT

cook version 2.8

The *cook* program comes with ABSOLUTELY NO WARRANTY; for details use the '*cook -VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*cook -VERSion License*' command.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

NAME

cook – bill of materials

SYNOPSIS

cook [*option...*] *dirname* [*outfile*]
cook -Help
cook -VERSion

DESCRIPTION

The *cook* program is used to scan a directory and generate a cookbook fragment contain a bill of materials for that directory. It also includes a recursive reference, via an ‘‘#include-cooked’’ directive, to the bills of materials for nested directories.

Output is sent to the standard output unless an output filename is specified.

OPTIONS

The following options are understood:

-DIRectory *pathname*
This option may be used to specify a directory search path, similar to *cook*(1) `[search_list]` functionality.

-Help
Provide some help with using the *cook* program.

-PREfix *string*
This option may be manipulate the name of the manifest files. Defaults to the empty string if not set.

-SUFfix *string*
This option may be manipulate the name of the manifest files. Defaults to ‘‘`/manifest.cook` if not set.

-VERSion
Print the version of the *cook* program being executed.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *cook* are long, this means ignoring the extra leading ’-’. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *cook* command will exit with a status of 1 on any error. The *cook* command will only exit with a status of 0 if there are no errors.

EXAMPLE

The intended use of this command is to automatically generate a project file manifest in an efficient way. If you have a cookbook of the form

```
all_files_in_. = ;
#include manifest.cook
manifest = [all_files_in_.];

set fingerprint mkdir unlink;

%0manifest.cook: ["if" [in "%0" ""] "then" "." "else" "%0"]
{
        cook_bom
                [addprefix '--dir=' [search_list]]
                [need]
                [target]
                ;
}
```

At the end of this fragment, the `manifest` variable contains a complete list of all files in the directory tree. This variable may then be taken apart with the `match_mask` function to build ingredients lists.

The constructed *manifest.cook* files work for both whole-project and recursive (not recommended) builds.

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the program's name and a brief idea of what it does.*
> Copyright (C) 19yy *name of author*
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19yy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> *signature of Ty Coon,* 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

NAME

cookfp – calculate file fingerprint

SYNOPSIS

cookfp [*option*...][*filename*...]
cookfp -Help
cookfp -VERSion

DESCRIPTION

The *cookfp* program is used to calculate the fingerprints of files. A fingerprint is a hash of the contents of a file. The default fingerprint is cryptographically strong, so the probability of two different files having the same fingerprint is less than 1 in 2**200.

The fingerprint is based on Dan Berstien <djb@silverton.berkeley.edu> public domain fingerprint 0.50 beta package 930809, posted to the alt.sources newsgroup. This program produces identical results; the expected test results were generated using Dan's package.

The fingerprint is a base-64-sanely-encoded fingerprint of the input. Imagine this fingerprint as something universal and permanent. A fingerprint is 76 characters long, containing the following:

1. A Snefru-8 (version 2.5, 8 passes, 512->256) hash. (Derived from the Xerox Secure Hash Function.)
2. An MD5 hash, as per RFC 1321. (Derived from the RSADSI MD5 Message-Digest Algorithm.)
3. A CRC checksum, as in the new cksum utility.
4. Length modulo 2^40.

The output format is not expected to be compatible with anything. However, options are available to produce the purported output of Merkle's snefru program, the purported output of RSADSI's mddriver -x, or the purported output of the POSIX cksum program.

If no files are named as input, the standard input will be used. The special file name ''-'' is understood to mean the standard input.

OPTIONS

The following options are understood:

-Checksum
Print the CRC32 checksum and length of the named file(s).

-Identifier
Print a condensed form of the fingerprint (obtained by performing a CRC32 checksum on the full fingerprint described above - a definite overkill). This is an 8-digit hexadecimal number, useful for generating unique short identifiers out of long names. The first character is forced to be a letter (g-p), so there is no problem in using the output as a variable name.

-Help
Provide some help with using the *cookfp* program.

-Message_Digest
Print the RSA Data Security, Inc. MD5 Message-Digest Algorithm hash of the named file(s).

-Snefru Print the Snefru hash of the named file(s), derived from the Xerox Secure Hash Function.

-VERSion

Print the version of the *cookfp* program being executed.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *cookfp* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *cookfp* command will exit with a status of 1 on any error. The *cookfp* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

Portions of this program are derived from sources from other people, sometimes with liberal copyrights, and sometimes in the public domain. These include:

Dan Bernstien

See *common/fp/README* for details.

Gary S Brown.

See *common/fp/crc32.c* for details.

RSA Data Security, Inc.

See *common/fp/md5.c* for details.

Xerox Corporation

See *common/fp/snefru.c* for details.

In addition to the above copyright holders, there have been numerous authors and contributors, see the named files for details. Files names are relative to the root of the *cook* distribution.

NAME

cooktime – set file times

SYNOPSIS

cooktime [*option*...] *filename*...
cooktime -Help
cooktime -VERSion

DESCRIPTION

The *cooktime* program is used to set the modified time or access time of a file. This can be used to defend against unwanted logical dependencies when making "minor" changes to files.

If no option is specified, the default action is as if "*–Modify now*" was specified.

OPTIONS

The following options are understood.

-Access *date*
This option may be used to set the last-access time of the files. The date is relatively free-format; rember to use quotes to insulate spaces from the shell.

-Modify *date*
This option may be used to set the last-modify time of the files. The date is relatively free-format; rember to use quotes to insulate spaces from the shell.

-Report
When use alone, produces a listing of access times and modify times for the named files. When used with -Access or -Modify, produces a listing of the changes made.

-Help
Give some information on how to use the *cooktime* command.

Any other option will generate a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *cooktime* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *cooktime* command will exit with a status of 1 on any error. The *cooktime* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

cooktime version 2.8

The *cooktime* program comes with ABSOLUTELY NO WARRANTY; for details use the '*cooktime -VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*cooktime -VERSion License*' command.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

NAME

find_libs – find pathnames of libraries

SYNOPSIS

find_libs [**-L***path* ...][**-l***name* ...]

find_libs -Help

find_libs -VERSion

DESCRIPTION

The *find_libs* program is used to find the actual pathname of a library specified on a *cc*(1) command line. This allows *cook*(1) to know these dependencies.

OPTIONS

The following options are understood.

-L*path*

Specify a path to search for libraries on. If more than one is specified, they will be scanned in the order given before the standard */usr/lib* and */lib* places. This is like the same argument to *cc*(1), and the usual find_libs option abbreviation rules do not apply.

-l*name*

Name a library to be searched for. This is like the same argument to *cc*(1), and the usual find_libs option abbreviation rules do not apply.

-Help

Give some information on how to use the *find_libs* command.

-VERSion

Tell the version of the *find_libs* command currently executing.

All other options will result in a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *find_libs* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *find_libs* command will exit with a status of 1 on any error. The *find_libs* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

find_libs version 2.8

The *find_libs* program comes with ABSOLUTELY NO WARRANTY; for details use the '*find_libs*

-VERSion License' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*find_libs -VERSion License*' command.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

NAME

make2cook – translate makefiles into cookbooks

SYNOPSIS

make2cook [*option*...][*infile* [*outfile*]]

make2cook -Help

make2cook -VERSion

DESCRIPTION

The *make2cook* program is used to translate *Makefile*s into cookbooks. This command is provided to ease the transition to using the *cook* command.

If no input file is named, or the special name "-" is used, input will be taken from the standard input. If no output file is named, or the special name "-" is used, output will be taken from the standard output.

SEMANTICS

There is no one-to-one semantic mapping between *make* semantics and *cook* semantics, so the results will probably need some manual editing.

The functionality provided by classic *make (1)* implementations is accurately reproduced. Extensions, such as those offered by GNU Make or BSD make, are not always understood, or are sometimes not reproduced identically.

The following subsections enumerate a few of the things which are understood and not understood. They are probably not complete.

Understood

The *cook* program requires variables to be defined before they are used, whereas *make* will default them to be empty. This is understood, and empty definitions are inserted as required.

Most of the builtin variables of GNU Make are understood.

Most of the builtin rules of classic make, GNU Make and BSD make are reproduced.

For best results there should be a blank line after every rule, so that there can be no confusion where one rule ends and a new one begins.

Builtin variables are defaulted from the environment, if an environment variable of the same name is set.

The GNU Make *override* variable assignment is understood.

The GNU Make "+=" assignment is understood.

The GNU Make ":=" variable assignment is understood.

Traditional make assignments are macros, they are expanded on use, rather than on assignment. The *cook* program has only variables. Assignment statements are re-arranged to ensure the correct results when variables are referenced.

Single and double suffix rules are understood. The .SUFFIXES rules are understood and honoured. Hint: if you want to suppress the builtin-recipes, use a .SUFFIXES rule with no dependencies.

The .PHONY rule is understood, and is translated into a *set forced* flag in appropriate recipes, except files from implicit recipes.

The .PRECIOUS rule is understood, and is translated into a *set precious* flag in the appropriate recipes, except files from implicit recipes.

The .DEFAULT rule is understood, and is translated into an implicit recipe.

The .IGNORE rule is understood, and is translated into a *set errok* statement.

The .SILENT rule is understood, and is translated into a *set silent* statement.

Most GNU Make functions are understood. The *filter* and *filter-out* functions only understand a single pattern. The *sort* function does not remove duplicates (wrap the *stringset* function around it if you need this).

The GNU Make static pattern rules are understood. They are translated into recipe predicates.

The GNU Make and BSD make *include* variants are understood.

The bizarre irregularities surrounding archive files in automatic variables and suffix rules are understood, and translated into consistent readable recipes. The *make* semantics are preserved.

The BSD make *.CURDIR* variable is understood, and translated to an equivalent expression. It cannot be assigned to.

The GNU Make and BSD make conditionals are understood, provided that they bracket whole segments of the makefile, and that these segments are syntactically valid. Cconditionals may also appear within rule body commands. Conditionals are *not* understood within the lines of a *define*.

The GNU Make *define* is understood, but its use as a kind of ‘‘function definition’’ is *not* understood.

The GNU Make *export* and *unexport* directives are understood.

Not Understood

The *cook* program tokenizes its input, whereas make does textual replacement. The shennanigans required to construct a make macro containing a single space are not understood. The translation will result in a *cook* variable which is empty.

References to automatic variables within macro definitions will not work.

The GNU Make *foreach* function is olny partially understood. This has no exact *cook* equivalent.

The GNU Make *origin* function is not understood. This has no *cook* equivalent.

The *archive*((*member*)) notation is not understood. These semantics are not available from *cook*.

The *MAKEFILES* and *MAKELEVEL* variables are not translated, If you wish to reproduce this functionality, you must edit the output.

The *MAKEFLAGS* and *MFLAGS* variables will be translated to use the Cook *options* function, which has a different range of values.

Many variants of make can use builtin rules to make the Makefile if it is absent. *Cook* is unable to cook the cookbook if it is absent.

Wildcards are not understood in rule targets, rule dependencies or include directives. If you want these, you will have to edit the output to use the *[wildcard]* function.

Home directory tildes (~) are not understood in targets and dependencies. If you want this, you will have to edit the output to use the *[home]* function.

The `-l`*`home`* dependency is not understood to mean a library. If you want this, you will have to edit the output to use the *[collect findlibs -l*name*]* function.

The *.EXPORT_ALL_VARIABLES* rule is not understood. This has no *cook* equivalent.

OPTIONS

The following options are understood:

-Help

Provide some help with using the *make2cook* command.

-Environment

This option causes fragments to test for environment variables when performing the default settings for variables. (This corresponds to the make -e option.)

-History_Commands

This option causes *make2cook* to include recipes for *RCS* and *SCCS* in the output.

-Line_Numbers

Insert line number directives into the output, so that it is possible to tell where the lines came from. Most useful when debugging. *make2cook* program.

-No_Internal_Rules

This option may be used to supress all generation of recipes corresponding to make's internal rules. (This corresponds to the make -r option.)

-VERSion

Print the version of the *make2cook* program being executed.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *make2cook* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *make2cook* command will exit with a status of 1 on any error. The *make2cook* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

make2cook version 2.8

The *make2cook* program comes with ABSOLUTELY NO WARRANTY; for details use the '*make2cook -VERSion License*' command. This is free software and you are welcome to redistribute it under certain conditions; for details use the '*make2cook -VERSion License*' command.

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/˜millerp/

NAME

roffpp – replace .so requests within *roff sources

SYNOPSIS

roffpp [*option*...][*infile* [*outfile*]]
roffpp -Help
roffpp -VERSion

DESCRIPTION

The *roffpp* command may be used to copies the input file to the output file, including files named using *.so* directives along the way, and removing the *.so* directives.

This is useful when processing large multi-file documents with filters such as *tbl*(1) or *eqn*(1) which do not understand the *.so* directive. The *.nx* directive is not understood. The *roffpp* program is not a general *roff interpreter, so many constructs will be beyond it, fortunately, most of them have nothing to do with include files. Include files which cannot be found, probably from uninterpreted *roff constructs, if the files really does exist, will simply be passed through unchanged, for *roff to interpret at a later time.

The *roffpp* program also allows the user to specify an include search path. This allows, for example, common files to be kept in a central location.

Only directives of the form

    .so *filename*

are processed. If the directive is introduced using the single quote form, or the dot is not the first character of the line, the directive will be ignored.

Any extra arguments on the line are ignored, and quoting is not understood. All characters are interpreted literally.

Examples of directives which will be ignored include

    'so /usr/lib/tmac/tmac.an
    .if n .so yuck

This list is not exhaustive.

The special file name '–' on the command line means the standard input or standard output, as appropriate. Files which are omitted are also assumed to be the standard input or standard output, as appropriate.

The output attempts to keep file names and line numbers in sync by using the **.lf** directive. The **.lf** directive is also understood as input. This is compatible with *groff*(1) and the other GNU text utilities included in the groff package.

OPTIONS

The following options are understood.

-I*path*

Specify include path, a la *cc*(1). Include paths are searched in the order specified. The include search path defaults to the current directory if and only if the user does not specify any include search paths.

-Help

Give information on how to use *roffpp*.

-VERSion

Tell what version of *roffpp* is being run.

Any other option will generate a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "-help", "-HEL" and "-h" are all interpreted to mean the **-Help** option. The argument "-hlp" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line.

The GNU long option names are understood. Since all option names for *roffpp* are long, this means ignoring the extra leading '-'. The "**--***option***=***value*" convention is also understood.

EXIT STATUS

The *roffpp* command will exit with a status of 1 on any error. The *roffpp* command will only exit with a status of 0 if there are no errors.

COPYRIGHT

AUTHOR

Peter Miller	E-Mail:	millerp@canb.auug.org.au
/\/*	WWW:	http://www.canb.auug.org.au/~millerp/

	The README File	1
	Release Notes	3
	How to Build the Sources	10
	Internationalization	17
c_incl(1)	determine include dependencies	19
cook(1)	a file construction tool	23
cook_bom(1)	bill of materials	29
cook_lic(1)	GNU General Public License	31
cookfp(1)	calculate file fingerprint	37
cooktime(1)	set file times	39
find_libs(1)	find pathnames of libraries	41
make2cook(1)	translate makefiles into cookbooks	43
roffpp(1)	replace .so requests within *roff sources	47

cook_bom(1)	29	cook_bom -	bill of materials
cook_bom(1)	29	cook_	bom - bill of materials
cookfp(1)	37	cookfp -	calculate file fingerprint
c_incl(1)	19		c_incl - determine dependencies
cook(1)	23	cook - a file	construction tool
cook(1)	23		cook - a file construction tool
make2cook(1)	43	make2	cook - translate makefiles into cookbooks
cook_bom(1)	29		cook_bom - bill of materials
make2cook(1)	43	make2cook - translate makefiles into	cookbooks
cookfp(1)	37		cookfp - calculate file fingerprint
cooktime(1)	39		cooktime - set file times
c_incl(1)	19	c_incl - determine	dependencies
c_incl(1)	19	c_incl -	determine dependencies
cook(1)	23	cook - a	file construction tool
cookfp(1)	37	cookfp - calculate	file fingerprint
cooktime(1)	39	cooktime - set	file times
find_libs(1)	41	find_libs -	find pathnames of libraries
find_libs(1)	41		find_libs - find pathnames of libraries
cookfp(1)	37	cookfp - calculate file	fingerprint
c_incl(1)	19	c_	incl - determine dependencies
make2cook(1)	43	make2cook - translate makefiles	into cookbooks
find_libs(1)	41	find_libs - find pathnames of	libraries
find_libs(1)	41	find_	libs - find pathnames of libraries
make2cook(1)	43		make2cook - translate makefiles into cookbooks
make2cook(1)	43	make2cook - translate	makefiles into cookbooks
cook_bom(1)	29	cook_bom - bill of	materials
find_libs(1)	41	find_libs - find	pathnames of libraries
roffpp(1)	47	roffpp -	replace .so requests within *roff sources
roffpp(1)	47	roffpp - replace .so	requests within *roff sources
roffpp(1)	47	roffpp - replace .so requests within *	roff sources
roffpp(1)	47		roffpp - replace .so requests within *roff sources
cooktime(1)	39	cooktime -	set file times
roffpp(1)	47	roffpp - replace .	so requests within *roff sources
roffpp(1)	47	roffpp - replace .so requests within *roff	sources
cooktime(1)	39	cooktime - set file	times
cook(1)	23	cook - a file construction	tool
make2cook(1)	43	make2cook -	translate makefiles into cookbooks
roffpp(1)	47	roffpp - replace .so requests	within *roff sources