

# Cook

A File Construction Tool

# Reference Manual

Peter Miller  
*[pmiller@opensource.org.au](mailto:pmiller@opensource.org.au)*

This document describes Cook version 2.34  
and was prepared 21 February 2011.

This document describing the Cook program, and the Cook program itself, are  
Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,  
2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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 3 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, see <<http://www.gnu.org/licenses/>>.

**NAME**

cook – a file construction tool

**DESCRIPTION**

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to see when the prerequisites of a file have changed, implying that the file needs to be recreated as it is logically out of date.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 is a replacement for the traditional *make(1)* tool.
- Cook is more powerful than the traditional *make* tool.
- Cook has true variables, not simple macros.
- Cook has user defined functions.
- Cook can build in parallel.
- Cook can distribute builds across your LAN.
- Cook is able to use fingerprints to supplement file modification times. This allows build optimization without contorted rules.
- In addition to walking the dependency graph, Cook can turn the input rules into a shell script, or a web page.
- Cook runs on almost any flavor of UNIX. The source distribution is self configuring using a GNU Autoconf generated configure script.
- There is a *make2cook* utility included in the distribution to help convert makefiles into cookbooks.
- 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 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.
- Cook has special *cascade* dependencies, allowing powerful include dependency specification, amongst other things.

If you are putting together a source-code distribution and planning to write a makefile, consider writing a cookbook instead. Although Cook takes a day or two to learn, it is much more powerful and a bit more intuitive than the traditional *make(1)* tool. And Cook doesn't interpret tab differently to 8 space characters!

**ARCHIVE SITE**

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

URL:	<a href="http://miller.emu.id.au/pmiller/cook/">http://miller.emu.id.au/pmiller/cook/</a>	
File:	<code>cook-2.34.README</code>	# the README from the tar file
File:	<code>cook-2.34.spec</code>	# RedHat package specification
File:	<code>cook-2.34.rm.ps.gz</code>	# PostScript of the Reference Manual
File:	<code>cook-2.34.ug.ps.gz</code>	# PostScript of the User Guide
File:	<code>cook-2.34.tar.gz</code>	# 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.

**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@lists.auug.org.au`

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

To subscribe to this mailing list, visit the *cook*-users mailing list page, and go through the **subscribe** dialogue.

<http://lists.auug.org.au/listinfo/cook-users>,

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 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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 3 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, see <http://www.gnu.org/licenses/>.

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

**AUTHOR**

Peter Miller	E-Mail:	<code>pmiller@opensource.org.au</code>
/\/*	WWW:	<a href="http://miller.emu.id.au/pmiller/">http://miller.emu.id.au/pmiller/</a>

## 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 acknowledgements of those who generously sent me feedback, please see the *etc/CHANGES.\** files included in this distribution.

### Version 2.34 (2010-Sep-25)

- This change set adds a link to the PPA to the download page. This means you can install a pre-built Ubuntu package, using the normal `apt-get install` method.
- This change set fixes a problem where some systems (usually pseudo filesystems) report ENOSYS instead of EACCES like any sane person expects.

### Version 2.33 (2009-Dec-20)

- The email address for the mailing list has changed. See README or the web site for more information.
- There is a new file-size-statistics option, for hinky NFS server update latency.
- A bug has been fixed in the ^C interrupt handler, it no longer deadlocks causing the processes to hang.
- A bug in the architecture handling has been fixed.
- This a number of 64-bit build issues have been fixed.
- The exit status is no longer printed for silent (not echoed) commands.
- The metering details are no longer printed for silent (not echoed) commands.
- The progress stars can now be enabled using the “`set star`” statements within a cookbook.
- A bug has been fixed in the `[collect]` and `[collect_lines]` built-in functions. If a word (or line) was longer than 1024 bytes, a buffer overrun occurred. A variable size buffer is now used.
- There is a new `[strlen]` built-in function, which may be used to obtain the length of a string.
- A bug has been fixed in the `[substitute]` function, it no longer infinite loops when the string to be replaced is the empty string.

### Version 2.32 (2008-Jul-29)

- The `[options]` function now includes the `-nic` option.
- There is now documentation in the `cook(1)` man page for the `cook --page-width` option.

### Version 2.31 (2008-Feb-13)

- A build problem with the `./configure` file has been fixed.
- There is a new “`set no-ctime`” flag, for coping with Aegis penchant for making and breaking hard links.
- The LICENSE file has been updated to match the licensing of the source code.

### Version 2.30 (2007-Aug-21)

- Several build and portability problems have been fixed.
- Several typographical and spelling errors have been fixed in the User Guide.
- The license has been changed to GNU GPL version 3.

### Version 2.29 (2007-Jun-22)

- There is a new variable for specifying the granularity of the file timestamps. Most POSIX systems will support a value of 1. Rather than default to the worst case, the user can now specify the value in seconds with a built-in cook variable.
- There is a new recipe option available called *symlink-ingredients* that has the effect of creating symbolic links for ingredients which are present on the search path, but not in the first directory in the search path. This option creates the necessary symbolic links. This is for use with brain dead tools, like GNU Automake, which don't grok search paths.

### Version 2.28 (2007-Jun-5)

- The `[print]` function has been enhanced so that it is now able to print more than one line, if you include a newline `\n` escape.

- A problem with the Makefile has been fixed.
- This change fixes a problem building the temp file name code which uses `sprintf()`. Basically, the code now uses `snprintf()` which is better and makes the problem go away.

#### Version 2.27 (2007-Mar-13)

- An ANSI C compiler is now required to build Cook.
- A bug has been fixed in the `cook_bom` command's `-prefix` and `-suffix` options.
- The fingerprint code is now more robust when faced with file modification time trickery by users.
- A few things have been improved for using Cook on Cygwin.
- The `c_incl -r` option now understands `.PSPIC` directives, as well as `.so` directives.

#### Version 2.26 (2006-Jan-17)

- A number of build problems have been fixed.
- A bug has been fixed in the tell-position flag. It wasn't actually giving the file name and line number when executing commands if you used the "set tell-position" variants, only the `-tell-position` command line option.
- The email address in the LSM file has been fixed,
- A bug has been fixed in the `cook -fp-update` command, it would segfault in some cases.
- A bug has been fixed in the cookbook include file processing.
- A bug has been fixed in the negative flag setting (comamnd line options and "set" clauses).
- The `find_command` command now copes better with directories it is not allowed to access.
- A Java cookbook has been added to to the distribution.
- A bug has been fixed in the execution of some commands. If any words of the command had spaces in them, it did not pass it to a shell to be executed, but instead constructed a command of a different shape than the user expected.

#### Version 2.25 (2004-Jun-10)

- The `./configure` script now understands the `--with-nlsdir` option, used to specify the install location of the `.mo` files.
- A bug has been fixed on Linux (and it only ever occurred on Linux) where cook would suddenly stop for no reason with exit status 1. Turns out that sometimes `fflush(stderr)` returns an EAGAIN error.
- A bug has been fixed which caused the `cook -script` option to produce invalid shell scripts when a recipe body contained no statements.
- A bug has been fixed in the graph file pair generation, used to generate warnings about dangerous `#include-cooked` contents.
- The metering output now includes elapsed times and percentages.
- There is a new `tell-position` setting, so that when Cook prints a command it is about to run, it includes the file name and line number of the command. This can be useful when debugging cookbooks.
- A bug has been fixed in the output line wrapping. Once again it adapts to the window width.

#### Version 2.24 (2003-Jul-17)

- A major problem with parallel execution and hangs has been fixed. The table indexed by process ID was now growing correctly.
- Some words have been added to the User Guide about the SHELL environment variable, and the effects of errors in the `.profile` file.
- Building RPMs has been improved, and the spec file now uses more modern RPM features.
- Building on Cygwin has been improved.

- Building on AIX has been improved.

#### **Version 2.23 (2003-May-01)**

- Build problem encountered using newer versions of GNU Bison have been fixed.
- For Cook developers, there is now a .ae file on the web site.
- An error in the documentation of the *errok* flag has been fixed.

#### **Version 2.22 (2003-Feb-28)**

- A small problem with fingerprints has been fixed.
- A tutorial has been contributed.
- You can now have international characters in comments.
- A C++ cookbook has been added.
- A test failure on Cygwin has been fixed.
- The [read] and [read\_lines] builtin functions have been added. See the Reference Manual for more information.

#### **Version 2.21 (2002-Aug-26)**

- The *c\_incl(1)* command now accepts the *-stripdot* and *-nostripdot* options. These may be used to control the removal of redundant leading dot directories.
- A bug has been fixed where cascade recipes failed to heed the *stripdot* setting.
- There is a new [stripdot] function, so that you can strip leading dot directories from file names within functions.
- A bug has been fixed in how the builtin functions which manipulate build graphs were called. This fixed a problem with freeing a string which had already been freed.

#### **Version 2.20 (2002-Jun-06)**

- There is a fix for the build problems caused by recent GNU Gettext releases.
- The fingerprint handling is now more robust, particularly when faced with files that move backwards in time.
- There is a fix for the build problems caused by recent Bison releases.

#### **Version 2.19 (2002-Feb-19)**

- Some introduced with recent versions of GNU Bison have been fixed. Bison's include file insulation didn't use YY in the insulating symbol (just to be completely inconsistent) and in another case a namespace clash occurred for a function name.
- The generated Makefile has been improved, along with other small build and install improvements.
- A top-level *fail* statement now halts the parse as soon as it is executed. This will make it more useful for checking build environments.
- Documentation about *cook\_rsh(1)* has been added to the Parallel chapter of the User Guide.

#### **Version 2.18 (2001-Oct-15)**

- A bug has been fixed in the *ingredients-fingerprint* recipe attribute. It was failing to save the fingerprint cache file in some cases, and thus came to incorrect conclusions on following runs.
- The (*exists*) ingredients attribute has been fixed so that it no longer implies behaviour similar to *set shallow*.
- There is a new *cook\_rsh(1)* program, for use with the *host-binding* recipe attribute, which allows you to load balance builds across classes of hosts. See *cook\_rsh(1)* and the Parallel chapter of the User Guide for more information.
- Some build problems have been fixed on various platforms.

- More keywords are now understood for M4 include directives.

#### **Version 2.17 (2001-Apr-25)**

- When using file fingerprints, the way the *.cook.fp* file is written has been changed, so that the timestamp of the containing directory is modified much less often. This is useful in combination with the *cook\_bom(1)* utility.
- A bug has been fixed under Cygwin, where archive members were not being fingerprinted correctly.
- A bug has been fixed in the `[quote]` function. It now quotes all *sh(1)*, *csh(1)* and *bash(1)* special characters correctly.
- A bug has been fixed in the `[uptodate]` function. It now works as advertised.
- There is a new *ingredients-fingerprint* recipe flag. This means that you can now cause a recipe to re-trigger when the ingredients list changes. This is especially useful when a library has a file removed.
- The dependency graph can now have the edge types specified. The “weak” edge type is useful for managing links, and the “exists” edge type is useful for managing version stamps. See the User Guide for more information.

#### **Version 2.16 (2000-Oct-25)**

- The *stringset* function now accepts a “+” operator. While union is implicit, the apparently redundant “+” operator is useful for cancelling the other operators.
- The “reason and fingerprint bug” has been fixed. This caused a mysterious error message to appear sometimes when using the `-reason` option in combination with fingerprints.
- The `%` and `%n` patterns are now allowed to match the empty string, provided they aren’t the first thing in the pattern (otherwise undesirable absolute path problems can occur).
- The *c\_incl(1)* command now accepts “-” as a file name on the command line, meaning standard input.
- Some improvements have been made to the Cygwin support, extending the “.exe” automatic executable suffix coverage to a couple more places.
- A bug in the “c” cookbook has been fixed, which was getting `.h` dependency files wrong.

#### **Version 2.15 (2000-Apr-11)**

- The *C\_incl(1)* problem with absolute paths has been fixed.
- A bug has been fixed which caused problems on Solaris and SGI, where Cook would report a No child processes error.

#### **Version 2.12 (2000-Mar-28)**

- The *c\_incl* program now has a **-quote-filenames** option, which means that you can have filenames with spaces and special characters in them.
- A bug in the *c\_incl* program’s path flattening has been fixed.
- A small Y2K bug has been fixed in the date parsing used by the *cooktime(1)* command.
- A bug which caused the `-parallel` option to lose track of processes when you used `[execute]` in a recipe body has been fixed.
- The restrictions on the placement of the placement of `%0` in a pattern have been dropped; too many people didn’t like it. This does *not* break any cookbooks.
- Cook now copes with the absence of the `HOME` environment variable. This was a problem for CGI scripts.

#### **Version 2.11 (1999-Nov-04)**

- Numerous portability problems have been fixed in the configure and build.
- A bug has been fixed which prevented Cook from working correctly when run by some versions of *cron(8)* and *at(1)*.

- There is a new `cook_bom --ignore` option, allowing you to nominate file patterns that you don't want in the file lists.
- There is a new `[_FUNCTION_]` variable, which contains the name of the executing function, which supplements the existing `[_FILE_]` and `[_LINE_]` variables.
- Functions now have local variables, just put the word `local` on the left-hand-side of the first assignment. Local variables are reentrant and thread-safe.

### Version 2.10 (1999-Sep-06)

- The `[print]` and `[write]` functions now work more sensibly with the `-SCRIPT` option.
- The fingerprint code has been improved. It now does considerably fewer redundant fingerprint calculations, resulting in some very welcome speed improvements.
- The behaviour of the remote shell invocation to cope with `rshd` at the remote end failing to spawn a shell, and it copes with the default shell at the remote end not being the Bourne shell.
- The `-PARallel` behaviour has been improved, so that it now looks for child process who have finished *more than* it looks for recipes to run. This doesn't change the semantics any, but it matches user expectations far better (and results in shorter-lived zombie processes).
- The `set meter` recipe flag works once more. (It stopped working when the parallel modifications were made, and mysteriously forgotten until now.)
- There are some changes made to the fingerprinting code to detect when files under ClearCase move backwards in time (because the underlying file version is "uncovered") meaning that the derived (object) files need to be rebuilt.
- There is a new `[mtime-seconds]` function, similar to the `[mtime]` function, except that it returns seconds since the epoch, rather than a human readable date. More useful to handing to `[expr]`.
- A bug has been fixed on SGI IRIX which failed to cope with not being able to create directories because they already exist.
- Ingredient recipes (ones with no body) may now have a double colon rather than a single colon, even when there is more than one target specified. Some users may find this a more natural syntax for ingredients recipes.
- The `[expr]` function now reports an error when given a number too big to represent, rather than quietly returning wrong answers. The range of representable values depends on your system.
- Cook now works with GNU Regex correctly on Windows-NT.

### Version 2.9 (1999-May-27)

- There is a new "for each" style looping construct. See the User Guide for more information.
- It is now possible to use regular expression patterns, instead of Cook's native patterns. You can set this for a whole cookbook or individual recipes. The default is to use Cook's native patterns. See the *File Name Patterns* chapter of the User Guide for more information.
- A bug which caused *host-binding* and *single-thread* to core dump has been fixed.
- All text file input now copes with CRLF sequences, so mixing NT and Unix builds on the one file server no longer creates problems.
- Fingerprints are now cached per-directory, rather than one huge file for an entire directory tree. This is more useful in recursive build and `[search_list]` situations.
- The `[cando]`, `[cook]` and `[uptodate]` functions now return lists of successful files, rather than a simple true/false result.
- The `[in]` and `[matches]` functions now return the list index (1 based) of the matching word. See the User Guide for more information.
- There is a new `cook -web` option, to print a HTML web page on the standard output, representing the dependency graph. This is useful in documenting the build process, or debugging cookbooks.

- There is a new *cook --fingerprint-update* option which scans the directory tree below the current directory and updates the file fingerprints. This helps when you use another tool (such as RCS or ClearCase) which alters the file but preserves the file's modification time.
- There is a new [write] function for writing text files. This is useful for coping with Windows-NT's absurdly short command lines.

#### Version 2.8 (1999-Feb-01)

- 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 (1998-Dec-30)

- 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 (1998-Nov-09)

- 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 (1998-Sep-02)

- 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 (1998-Jul-21)

- 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 (1998-May-20)

- 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 (1997-Dec-10)

- 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 (1997-Nov-04)

- 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 (1997-Oct-31)

- 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 distributed 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 semantically equivalent Cook constructs.

#### **Release 2.1 (1997-Oct-12)**

- 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 subsequent 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 (1997-Sep-11)**

#### **Version 1.26 (2005-Jan-17)**

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 parallelized.
- 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 (1997-Jun-14)**

- 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 recurse 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 unnecessary 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 mileage 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.

Please note: if you install these packages 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 *./configure* script will incorrectly conclude that they have not been installed.

**ANSI C compiler**

You will need an ANSI C compiler to be able to compile cook. If you don't have one, you may wish to consider installing the GNU C compiler, it's free.

**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 explicitly 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.

**GNU rx**

Cook needs regular expressions to operate correctly. Get a copy from your nearest GNU mirror. On systems which use the GNU C library, version 2.0 or later, there is no need to explicitly do this as GNU rx is included.

**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. You must use GNU Groff version 1.15 or later.

On Solaris, you may need to edit the *Makefile* to change the groff *-man* and *-mm* options to *-mgn* and *-mgm* instead.

**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 `ftp.gnu.org`, 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. 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. 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.

`--with-nsdir=PATH`

This directory contains the install 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 `--libdir` 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 *-lfoo -lbar*. 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.

Variable: NLSDIR

Similar to the *--with-nlsdir* option.

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 preprocessor 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.2.34.rm.ps.gz* and *cook.2.34.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 already 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 <pmiller@opensource.org.au>
```

Any and all feedback is welcome.

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

```
% cook -version
cook version 2.34.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.cygnum.com/cygwin/> and then follow the links. The version I used was B20.1.

### 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-cygwin/bin /bin
```

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 Cygwin sessions. They are stored in a registry file somewhere. You will not need to do all this every time! ore recent Cygwin versions don't need this at all.

### 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 through to the install section before you configure anything.

```
bash$ ./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
...lots of output...
bash$
```

### Test

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

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

If test 12 fails, it probably means you don't have */bin* right.

### Install

Installing the software works as usual, though you need to make some choices 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

*cook* version 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

The *Cook* package 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.

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

### AUTHOR

Peter Miller    E-Mail:    pmiller@opensource.org.au  
/\ /\ \*        WWW:        http://miller.emu.id.au/pmiller/

**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, it 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.

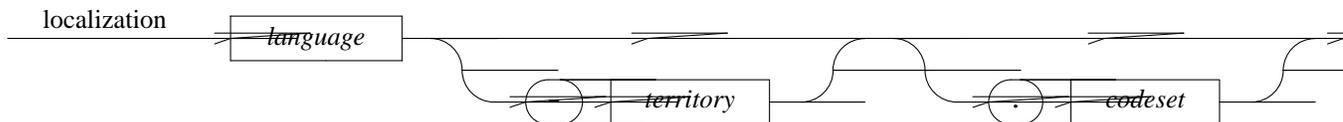
For example, some Swedish users would rather read translations in German when Swedish is not available. This is easily accomplished by setting 'LANGUAGE' to 'sv:de' while leaving 'LANG' set to 'sv'.

**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 or #include-cooked-ing by cook.

The filename “-” is understood to mean the standard input. When you use this file name, caching is ignored.

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.

**-Ipath**

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`.

**-EXclude filename**

This option may be used to nominate include file names which are not to be used.

**-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.

**-Quote\_FileNames**

This option may be used to have *c\_incl* quote filenames. This permits filenames to contain characters which are special to Cook, including spaces.

**-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 used 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]]"`

**-STripdot**

This option may be used to specify that leading redundant dot directories are to be removed from paths before processing. This is the default.

**-No\_STripdot**

This option may be used to specify that leading redundant dot directories need not be removed from paths before processing. (Some path flattening may still occur.)

**-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 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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:    pmiller@opensource.org.au  
^/\\*            WWW:        http://miller.emu.id.au/pmiller/

**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 current directory. This file may, in turn, include other files containing additional recipes.

If no *filenames* 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.

**-CTime** The inode *st\_ctime* data is used to supplement the *st\_mtime* data when determining whether or not files have changed. This is the default. (If you have no idea what this is, don’t mess with it.)

**-No\_CTime**

Do not supplement *st\_mtime* with *st\_ctime*. This may be important when *st\_nlink* changes at critical times, because making and breaking hard links touches *st\_ctime*. (If you have no idea what this is, seriously, don’t mess with it.)

**-Errork**

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

**-No\_Errork**

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.

**-FingerPrint\_Update**

This option may be used to scan the directory tree below the current directory and update the file fingerprints. This helps when you use another tool (such as RCS or ClearCase) which alters the file but preserves the file's modification time.

**-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/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.

**-Page-Length number**

This option may be used to set the length of the page, used when **Cook** needs to paginate output. Defaults to what the *LINES* environment variable tells it, or the terminal emulator tells it if *LINES* isn't set. **-Page-Width number** This option may be used to set the width of the page, used when **Cook** needs to wrap output (*e.g.* when it prints commands being executed). Defaults to what the *COLS* environment variable tells it, or the terminal emulator tells it if *COLS* isn't set. The maximum value for *number* is 32767.

**-PARallel [ number ]**

This option may be used to specify the number of parallel execution 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
- @ Writing fingerprint files.

**-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.

**-SymLink-Ingredients**

The option asks that, when using a search path, that non-top-level recipe ingredients get a top-level symlink to the actual file. This is intended for brain dead tools, like GNU Autoconf, that don't grok search paths.

**-No-SymLink-Ingredients**

Do not create top level symlinks to ingredients. This is the default.

**-Tell\_Position**

This option may be used to cause the position of commands (filename and line number) to be printed along with the command just before it is executed (provided the **-No\_Silent** option is in force).

**-No\_Tell\_Position**

This option may be used to suppress printing the position of commands (filename and line number) along with the command just before it is executed. This is the default.

**-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.

**-TErминаl**

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

**-No\_TErминаl**

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

**-Time\_Adjust**

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. This corresponds to the *time-adjust* recipe flag.

**-No\_Time\_Adjust**

Do not update the file last-modified times after performing the body of a recipe. This is the default. This corresponds to the *no-time-adjust* recipe flag.

**-Web**

This option may be used to request a HTML web page be printed on the standard output. This web page may be used to document the file dependencies; it captures many of the semantics of the cookbook. It can also be very useful when debugging cookbooks.

*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/share/cook*

This directory contains "system" cookbooks for various tools and activities.

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

**ENVIRONMENT VARIABLES**

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

**COOK** May be set to contain command-line options, changing the default behavior 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 file system.

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.

Defaults to “/tmp\_mnt : /a : / .automount” if not set.

## **COPYRIGHT**

*cook* version 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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:    pmiller@opensource.org.au  
^/\\*            WWW:        http://miller.emu.id.au/pmiller/

**NAME**

cook\_bom – bill of materials

**SYNOPSIS**

**cook\_bom** [ *option...* ] *dirname* [ *outfile* ]

**cook\_bom -Help**

**cook\_bom -VERSion**

**DESCRIPTION**

The *cook\_bom* program is used to scan a directory and generate a cookbook fragment containing 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\_bom* program.

**-IGNore** *string*

This option may be used to specify filename patterns to be ignored. It may be given as many times as required.

**-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\_bom* 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\_bom* are long, this means ignoring the extra leading ‘-’. The “--option=value” convention is also understood.

**EXIT STATUS**

The *cook\_bom* command will exit with a status of 1 on any error. The *cook\_bom* 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]]
        "--ignore='*~'"
        [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

*cook\_bom* version 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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

## AUTHOR

Peter Miller    E-Mail:    pmiller@opensource.org.au  
 ^\^\*            WWW:        http://miller.emu.id.au/pmiller/

## GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program -- to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. 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 them 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 prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. 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.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

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

## TERMS AND CONDITIONS

## 0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of

the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

### 1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

### 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your

copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

### 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

### 4. Conveying Verbatim Copies.

You may convey 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; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

### 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to

apply to the other parts of the aggregate.

#### 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in

ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

#### 7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license,

or stated as exceptions; the above requirements apply either way.

#### 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

#### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

#### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

#### 11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s

essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

#### 12. No Surrender of Others’ Freedom.

If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

#### 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License,

section 13, concerning interaction through a network will apply to the combination as such.

#### 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU 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 that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

#### 15. Disclaimer of Warranty.

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.

#### 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS 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.

#### 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

### 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 state 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) *year 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 3 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, see <<http://www.gnu.org/licenses/>>.

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

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

<program> Copyright (C) <year> <name of author>

This program 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, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <<http://www.gnu.org/licenses/>>.

The GNU 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 Lesser General Public License instead of this License. But first, please read <<http://www.gnu.org/philosophy/why-not-lgpl.html>>.

**NAME**

cook – load balancing rsh

**SYNOPSIS**

```
cook [ option... ] architecture command [ argument... ]
cook -Help
```

**DESCRIPTION**

The *cook* program is a wrapper around *rsh(1)* which does simple load balancing. It obtains its load information by running the *rup(1)* command, and selects the most suitable host based on the architecture you specify, and the least load of all hosts of that architecture.

The first command line argument is the architecture name which is used to get the list of possible hosts. From that list the *rup(1)* command is run to determine the host with the lowest load, which is in turn used as the first argument of the eventual *rsh(1)* command.

**COOKBOOKS**

In order to make use of this program, somewhere in your cookbook, you need to add a line which reads

```
parallel_rsh = "cook";
```

If the host chosen is the same as the caller (build host) then this program just exec the command skipping the rsh. So it costs nothing to use this in a one machine network!

For each recipe you want distributed to a remote host, you need to add a host-binding attribute to. Typical usage is where you have a multi-architecture build.

```
%1/%0%.o: %0%.c
    host-binding %1 {
        cc -o [target] -c [resolve %0%.c]; }
```

In the recipe given here, each architecture has its object files placed into a separate architecture-specific directory tree. The architecture name (%1) is used in the host-binding, so that the compiles may be load-balanced to all machines of that architecture.

If you need a command to run on a specific host (say, because that's where a specific application license resides), then simply use the host name in the host-binding attribute, rather than an architecture name.

**DEFINING THE CLASSES**

The */usr/share/cook/host\_lists.pl* file is expected to exist, and to contain variable definitions used to determine if hosts are members of particular architectures.

The */usr/share/cook/host\_lists.pl* file defines a perl HOL "hash of lists" The hash is %ArchNames and it maps names of architectures as user want to see them, to list references as the actual lists are stored.

The names of each architecture could be any form you wish but the convention is to use the GNUish names such as "sparc-sun-solaris2.8".

For each architecture, define one or more lists of machines according to what function each machine set may do. This can be as simple or as elaborate as required. The form of the list variable name can be any valid perl identifier but may as well be like the architecture name with dash changed to underbar and dot removed, and the type added. For example one might define solaris hosts as:

```
@sparc_sun_solaris28_hosts = (
    "mickey", "minny", "scrooge" );
```

And linux hosts as:

```
@i386_linux22_hosts = (
    "goofy", "scrooge" );
```

If there is a need to define different sets of machines for different types of jobs then add a suffix to the names in the *host-binding* directive on each of the recipes, and lists here with the same suffix.

The hash to map argument names to lists is defined like:

```
%ArchNames = (
    "sparc-solaris2.8",      => @sparc_solaris28_hosts,
    "i586-unknown-linux22", => @i386_linux22_hosts, );
```

Of course if users have differing opinions as to what the architecture names should look like, you can define "alias" mappings as well.

```
"sun4-SunOS-5.8",      => @sparc_solaris28_hosts,
```

Or maybe the level is of no importance, then define

```
"sparc-solaris",      => @sparc_solaris28_hosts,
"sparc-solaris2.7",   => @sparc_solaris28_hosts,
```

Also, this list isn't allowed to be empty.

And finally, curtesy of Perl, the last line of the file must read

```
1; for obscure and magical reasons.
```

## SYSLOG LOGGING

Typical commands seen during a build would look like

```
sh -c 'cd /aegis/dd/gumby2.2.C079 && \ sh -ce /aegis/dd/gumby2.2.C079/.6.1; \ echo $? >
/aegis/dd/gumby2.2.C079/.6.2'
```

So we can extract the project/ change from the command quite easily and logging it via syslog would be a trivial addition.

## OPTIONS

This command is not usually given any options.

```
-h      Help - show usage info
-vP     Verbose - report choice
-Tn     Trace value for testing
```

## FILES

/usr/share/cook/exclude.hosts

This file is used to list those host which must not be used by this script. Simply list excuded hosts, one hostname per line. If the file is absent, all hosts reported by rup(1) may be used.

/usr/share/cook/host\_lists.pl

This file defines the classes of hosts for each architecture.

## AUTHOR

Jerry Pendergraft <jerry@endocardial.com>

**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-sanelly-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

*cookfp* version 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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

## AUTHOR

Peter Miller    E-Mail:    pmiller@opensource.org.au  
^/\^\*            WWW:        http://miller.emu.id.au/pmiller/

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

Dan Bernstein

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; remember 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; remember to use quotes to insulate spaces from the shell.

**-Time-Stamp-Granularity seconds**

This option may be used to specify the granularity of the filesystem's timestamps, otherwise a default value of 1 second is used.

**-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 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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.

COOKTIME(1)

COOKTIME(1)

**AUTHOR**

Peter Miller    E-Mail:    pmiller@opensource.org.au  
^/\^\*            WWW:        http://miller.emu.id.au/pmiller/

**NAME**

`find_libs` – find pathnames of libraries

**SYNOPSIS**

`find_libs` [ **-L***path* ... ] [ **-I***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.

**-I***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 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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:    pmiller@opensource.org.au  
^/\^\*            WWW:        http://miller.emu.id.au/pmiller/

**NAME**

make2cook – translate makefiles into cookbooks

**SYNOPSIS**

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

**make2cook -Help**

**make2cook -VERSion**

**DESCRIPTION**

The *make2cook* program is used to translate *Makefiles* 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 error* 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 only 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 *-lhome* dependency is not understood to mean a library. If you want this, you will have to edit the output to use the *[collect findlibs -lname]* 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 suppress 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 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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:    pmiller@opensource.org.au  
^/\\*            WWW:        http://miller.emu.id.au/pmiller/

**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 *.If* directive. The *.If* 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**

*roffpp* version 2.34

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Peter Miller

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

### **AUTHOR**

Peter Miller    E-Mail:    pmiller@opensource.org.au  
^/\\*            WWW:        http://miller.emu.id.au/pmiller/



	The README File . . . . .	1
	Release Notes . . . . .	3
	How to Build the Sources . . . . .	13
	Windows NT . . . . .	18
	Internationalization . . . . .	20
c_incl(1)	determine include dependencies . . . . .	22
cook(1)	a file construction tool . . . . .	26
cook_bom(1)	bill of materials . . . . .	32
cook_lic(1)	GNU General Public License . . . . .	34
cook_rsh(1)	load balancing rsh . . . . .	43
cookfp(1)	calculate file fingerprint . . . . .	45
cooktime(1)	set file times . . . . .	47
find_libs(1)	find pathnames of libraries . . . . .	49
make2cook(1)	translate makefiles into cookbooks . . . . .	51
roffpp(1)	replace .so requests within *roff sources . . . . .	54

cook_rsh(1)	43	cook_rsh - load	balancing rsh
cook_bom(1)	32	cook_bom -	bill of materials
cook_bom(1)	32	cook_	bom - bill of materials
cookfp(1)	45	cookfp -	calculate file fingerprint
c_incl(1)	22		c_incl - determine dependencies
cook(1)	26	cook - a file	construction tool
cook(1)	26		cook - a file construction tool
cook_bom(1)	32		cook_bom - bill of materials
make2cook(1)	51	make2cook - translate makefiles into	cookbooks
cookfp(1)	45		cookfp - calculate file fingerprint
cook_rsh(1)	43		cook_rsh - load balancing rsh
cooktime(1)	47		cooktime - set file times
make2cook(1)	51	make2	cook - translate makefiles into cookbooks
c_incl(1)	22	c_incl - determine	dependencies
c_incl(1)	22	c_incl -	determine dependencies
cook(1)	26	cook - a	file construction tool
cookfp(1)	45	cookfp - calculate	file fingerprint
cooktime(1)	47	cooktime - set	file times
find_libs(1)	49	find_libs -	find pathnames of libraries
find_libs(1)	49	find_libs -	find pathnames of libraries
cookfp(1)	45	cookfp - calculate file	fingerprint
c_incl(1)	22	c_	incl - determine dependencies
make2cook(1)	51	make2cook - translate makefiles	into cookbooks
find_libs(1)	49	find_libs - find pathnames of	libraries
find_libs(1)	49	find_	libs - find pathnames of libraries
cook_rsh(1)	43	cook_rsh -	load balancing rsh
make2cook(1)	51		make2cook - translate makefiles into
			cookbooks
make2cook(1)	51	make2cook - translate	makefiles into cookbooks
cook_bom(1)	32	cook_bom - bill of	materials
find_libs(1)	49	find_libs - find	pathnames of libraries
roffpp(1)	54	roffpp -	replace .so requests within *roff sources
roffpp(1)	54	roffpp - replace .so	requests within *roff sources
roffpp(1)	54		roffpp - replace .so requests within *roff
			sources
roffpp(1)	54	roffpp - replace .so requests within *	roff sources
cook_rsh(1)	43	cook_rsh - load balancing	rsh
cook_rsh(1)	43	cook_	rsh - load balancing rsh
cooktime(1)	47	cooktime -	set file times
roffpp(1)	54	roffpp - replace .	so requests within *roff sources
roffpp(1)	54	roffpp - replace .so requests within *roff	sources
cooktime(1)	47	cooktime - set file	times
cook(1)	26	cook - a file construction	tool
make2cook(1)	51	make2cook -	translate makefiles into cookbooks
roffpp(1)	54	roffpp - replace .so requests	within *roff sources