MAKE(1) User Commands MAKE(1)

NAME

make - GNU make utility to maintain groups of programs

SYNOPSIS

make [OPTION]... [TARGET]...

DESCRIPTION

The make utility will determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them. The manual describes the GNU implementation of make, which was written by Richard Stallman and Roland McGrath, and is currently main? tained by Paul Smith. Our examples show C programs, since they are very common, but you can use make with any programming language whose compiler can be run with a shell command. In fact, make is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change.

To prepare to use make, you must write a file called the makefile that describes the relationships among files in your program, and the states the commands for updating each file. In a program, typically the exe? cutable file is updated from object files, which are in turn made by compiling source files.

Once a suitable makefile exists, each time you change some source files, this simple shell command:

make

suffices to perform all necessary recompilations. The make program uses the makefile description and the last-modification times of the files to decide which of the files need to be updated. For each of those files, it issues the commands recorded in the makefile.

make executes commands in the makefile to update one or more target names, where name is typically a program. If no -f option is present, make will look for the makefiles GNUmakefile, makefile, and Makefile, in that order.

Normally you should call your makefile either makefile or Makefile. (We recommend Makefile because it appears prominently near the begin? ning of a directory listing, right near other important files such as README.) The first name checked, GNUmakefile, is not recommended for most makefiles. You should use this name if you have a makefile that is specific to GNU make, and will not be understood by other versions of make. If makefile is '-', the standard input is read.

make updates a target if it depends on prerequisite files that have

not exist.

OPTIONS

-b, -m

These options are ignored for compatibility with other versions of make.

-B, --always-make

Unconditionally make all targets.

-C dir, --directory=dir

Change to directory dir before reading the makefiles or doing any? thing else. If multiple -C options are specified, each is inter? preted relative to the previous one: -C / -C etc is equivalent to -C /etc. This is typically used with recursive invocations of make.

-d Print debugging information in addition to normal processing. The debugging information says which files are being considered for remaking, which file-times are being compared and with what re? sults, which files actually need to be remade, which implicit rules are considered and which are applied---everything interest? ing about how make decides what to do.

Print debugging information in addition to normal processing. If the FLAGS are omitted, then the behavior is the same as if -d was specified. FLAGS may be a for all debugging output (same as using -d), b for basic debugging, v for more verbose basic debugging, i for showing implicit rules, j for details on invocation of com? mands, and m for debugging while remaking makefiles. Use n to disable all previous debugging flags.

-e, --environment-overrides

Give variables taken from the environment precedence over vari? ables from makefiles.

-f file, --file=file, --makefile=FILE

Use file as a makefile.

-i, --ignore-errors

Ignore all errors in commands executed to remake files.

-I dir, --include-dir=dir

Specifies a directory dir to search for included makefiles. If several -I options are used to specify several directories, the directories are searched in the order specified. Unlike the argu? ments to other flags of make, directories given with -I flags may come directly after the flag: -Idir is allowed, as well as -I dir.

-I flag.

-j [jobs], --jobs[=jobs]

Specifies the number of jobs (commands) to run simultaneously. If there is more than one -j option, the last one is effective. If the -j option is given without an argument, make will not limit the number of jobs that can run simultaneously. When make invokes a sub-make, all instances of make will coordinate to run the spec? ified number of jobs at a time; see the section PARALLEL MAKE AND THE JOBSERVER for details.

--jobserver-fds [R,W]

Internal option make uses to pass the jobserver pipe read and write file descriptor numbers to sub-makes; see the section PARAL? LEL MAKE AND THE JOBSERVER for details

-k, --keep-going

Continue as much as possible after an error. While the target that failed, and those that depend on it, cannot be remade, the other dependencies of these targets can be processed all the same.

-l [load], --load-average[=load]

Specifies that no new jobs (commands) should be started if there are others jobs running and the load average is at least load (a

limit.

-L, --check-symlink-times

Use the latest mtime between symlinks and target.

-n, --just-print, --dry-run, --recon

Print the commands that would be executed, but do not execute them (except in certain circumstances).

-o file, --old-file=file, --assume-old=file

Do not remake the file file even if it is older than its dependen? cies, and do not remake anything on account of changes in file. Essentially the file is treated as very old and its rules are ig? nored.

-O[type], --output-sync[=type]

When running multiple jobs in parallel with -j, ensure the output of each job is collected together rather than interspersed with output from other jobs. If type is not specified or is target the output from the entire recipe for each target is grouped together. If type is line the output from each command line within a recipe is grouped together. If type is recurse output from an entire re? cursive make is grouped together. If type is none output synchro? nization is disabled.

-p, --print-data-base

Print the data base (rules and variable values) that results from reading the makefiles; then execute as usual or as otherwise spec? ified. This also prints the version information given by the -v switch (see below). To print the data base without trying to re? make any files, use make -p -f/dev/null.

-q, --question

``Question mode''. Do not run any commands, or print anything; just return an exit status that is zero if the specified targets are already up to date, nonzero otherwise.

-r, --no-builtin-rules

Eliminate use of the built-in implicit rules. Also clear out the default list of suffixes for suffix rules.

-R, --no-builtin-variables

Don't define any built-in variables.

-s, --silent, --quiet

Silent operation; do not print the commands as they are executed.

-S, --no-keep-going, --stop

Cancel the effect of the -k option. This is never necessary ex?

level make via MAKEFLAGS or if you set -k in MAKEFLAGS in your en? vironment.

-t, --touch

Touch files (mark them up to date without really changing them) instead of running their commands. This is used to pretend that the commands were done, in order to fool future invocations of make.

--trace

Information about the disposition of each target is printed (why the target is being rebuilt and what commands are run to rebuild it).

-v, --version

Print the version of the make program plus a copyright, a list of authors and a notice that there is no warranty.

-w, --print-directory

Print a message containing the working directory before and after other processing. This may be useful for tracking down errors from complicated nests of recursive make commands.

--no-print-directory

-W file, --what-if=file, --new-file=file, --assume-new=file

Pretend that the target file has just been modified. When used with the -n flag, this shows you what would happen if you were to modify that file. Without -n, it is almost the same as running a touch command on the given file before running make, except that the modification time is changed only in the imagination of make.

--warn-undefined-variables

Warn when an undefined variable is referenced.

EXIT STATUS

GNU make exits with a status of zero if all makefiles were successfully parsed and no targets that were built failed. A status of one will be returned if the -q flag was used and make determines that a target needs to be rebuilt. A status of two will be returned if any errors were encountered.

SEE ALSO

The full documentation for make is maintained as a Texinfo manual. If the info and make programs are properly installed at your site, the command

info make

should give you access to the complete manual. Additionally, the manual is also available online at https://www.gnu.org/software/make/manual/html_node/index.html

PARALLEL MAKE AND THE JOBSERVER

Using the -j option, the user can instruct make to execute tasks in parallel. By specifying a numeric argument to -j the user may specify an upper limit of the number of parallel tasks to be run.

When the build environment is such that a top level make invokes sub-makes (for instance, a style in which each sub-directory contains its own Makefile), no individual instance of make knows how many tasks are running in parallel, so keeping the number of tasks under the upper limit would be impossible without communication between all the make instances running. While solutions like having the top level make serve as a central controller are feasible, or using other synchronization mechanisms like shared memory or sockets can be created, the current implementation uses a simple shared pipe.

This pipe is created by the top-level make process, and passed on to all the sub-makes. The top level makeprocesswrites N-1 one-byte tokens into the pipe (The top level make is assumed to reserve one token for itself). Whenever any of the make processes (including the top-level make) needs to run a new task, it reads a byte from the shared pipe.

back to the pipe. Once the task is completed, the make process writes a token back to the pipe (and thus, if the tokens had been exhausted, un? blocking the first make process that was waiting to read a token). Since only N-1 tokens were written into the pipe, no more than N tasks can be running at any given time.

If the job to be run is not a sub-make then make will close the job? server pipe file descriptors before invoking the commands, so that the command can not interfere with the jobserver, and the command does not find any unusual file descriptors.

BUGS

See the chapter ``Problems and Bugs'' in The GNU Make Manual.

AUTHOR

This manual page contributed by Dennis Morse of Stanford University. Further updates contributed by Mike Frysinger. It has been reworked by Roland McGrath. Maintained by Paul Smith.

COPYRIGHT

Free Software Foundation; either version 3 of the License, or (at your option) any later version.

GNU Make 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/.