NAME

latexmk − generate LaTeX document

SYNOPSIS

latexmk [options] [file ...]

DESCRIPTION

Latexmk completely automates the process of compiling a LaTeX document. Essentially, it is like a specialized relative of the general *make* utility, but one which determines dependencies automatically and has some other very useful features. In its basic mode of operation *latexmk* is given the name of the primary source file for a document, and it issues the appropriate sequence of commands to generate a .dvi, .ps, .pdf and/or hardcopy version of the document.

By default *latexmk* will run the commands necessary to generate a .dvi file.

Latexmk can also be set to run continuously with a suitable previewer. In that case the *latex* program (or one of its relatives), etc, are rerun whenever one of the source files is modified, and the previewer automatically updates the on-screen view of the compiled document.

Latexmk determines which are the source files by examining the log file. (Optionally, it also examines the list of input and output files generated by the **-recorder** option of modern versions of *latex* (and *pdflatex*, *xelatex*, *lualatex*, etc). See the documentation for the **-recorder** option of *latexmk* below.) When *latexmk* is run, it examines properties of the source files, and if any have been changed since the last document generation, *latexmk* will run the various LaTeX processing programs as necessary. In particular, it will repeat the run of *latex* (or a related program)) often enough to resolve all cross references; depending on the macro packages used. With some macro packages and document classes, four, or even more, runs may be needed. If necessary, *latexmk* will also run *bibtex*, *biber*, and/or *makeindex*. In addition, *latexmk* can be configured to generate other necessary files. For example, from an updated figure file it can automatically generate a file in encapsulated postscript or another suitable format for reading by LaTeX.

Latexmk has two different previewing options. In the simple **-pv** option, a dvi, postscript or pdf previewer is automatically run after generating the dvi, postscript or pdf version of the document. The type of file to view is selected according to configuration settings and command line options.

The second previewing option is the powerful **-pvc** option (mnemonic: "preview continuously"). In this case, *latexmk* runs continuously, regularly monitoring all the source files to see if any have changed. Every time a change is detected, *latexmk* runs all the programs necessary to generate a new version of the document. A good previewer (like *gv*) will then automatically update its display. Thus the user can simply edit a file and, when the changes are written to disk, *latexmk* completely automates the cycle of updating the .dvi (and possibly the .ps and .pdf) file, and refreshing the previewer's display. It's not quite WYSIWYG, but usefully close.

For other previewers, the user may have to manually make the previewer update its display, which can be (some versions of *xdvi* and *gsview*) as simple as forcing a redraw of its display.

Latexmk has the ability to print a banner in gray diagonally across each page when making the postscript file. It can also, if needed, call an external program to do other postprocessing on generated dvi and postscript files. (See the options **-dF** and **-pF**, and the documentation for the *$dvi_filter* and *$ps_filter* configuration variables.) These capabilities are leftover from older versions of *latexmk*. More flexibility can be obtained in current versions, since the command strings for running latex, pdflatex, etc can now be configured to run multiple commands. This also extends the possibility of postprocessing generated files to pdf files. files.

Latexmk is highly configurable, both from the command line and in configuration files, so that it can accommodate a wide variety of user needs and system configurations. Default values are set according to the operating system, so *latexmk* often works without special configuration on MS-Windows, cygwin, Linux, OS-X, and other UNIX systems (notably Solaris).

A very annoying complication handled very reliably by *latexmk*, is that LaTeX is a multiple pass system. On each run, LaTeX reads in information generated on a previous run, for things like cross referencing and indexing. In the simplest cases, a second run of LaTeX suffices, and often the log file contains a message about the need for another pass. However, there is a wide variety of add-on macro packages to LaTeX, with a variety of behaviors. The result is to break simple-minded determinations of how many runs are needed and of which programs. In its new version, *latexmk* has a highly general and efficient solution to these issues. The solution involves retaining between runs information on the source files, and a symptom is that *latexmk* generates an extra file (with extension *.fdb_latexmk*, by default) that contains the source file information.

LATEXMK OPTIONS AND ARGUMENTS ON COMMAND LINE

In general the command line to invoke *latexmk* has the form

latexmk [options] [file]

All options can be introduced by single or double "-" characters, e.g., "latexmk -help" or "latexmk --help".

Note 1: In addition to the options in the list below, *latexmk* recognizes almost all the options recognized by the *latex*, *pdflatex* programs (and their relatives) in their current TeXLive and MiKTeX implementations. Some of the options for these programs also trigger special action or behavior by *latexmk*, in which case they have specific explanations in this document. Otherwise, they are just passed through to a called *latex* or *pdflatex* program. Run *latexmk* with the **-showextraoptions** to get a list of the options that *latexmk* accepts and that are simply passed through to *latex* or *pdflatex* (etc). See also the explanation of the **-showextraoptions** option for more information.

Note 2: In this documentation, the program *pdflatex* is often referred to. Users of programs like *lualatex* and *xelatex* should know that from *latexmk*'s point of view, these other programs behave like *pdflatex*, i.e., they make a pdf file from a tex file, etc. So whenever *pdflatex* is mentioned without mention of the other programs, the statements apply equally to *lualatex*, *xelatex*, and any other similar programs. *Latexmk* can be easily configured to use whichever of these programs is needed. See the documentation for the following options: **-pdflatex="COMMAND"**, **-lualatex**, and **-xelatex**, and also see the documentation for the *$pdflatex* configuration variable. At present *latexmk* cannot do automatic detection of which program is to be used.

Definitions of options and arguments

file One or more files can be specified. If no files are specified, *latexmk* will, by default, run on all files in the current working directory with a ".tex" extension. This behavior can be changed: see the description concerning the *@default_files* variable in the section "List of configuration variables usable in initialization files".

If a file is specified without an extension, then the ".tex" extension is automatically added, just as LaTeX does. Thus, if you specify:

latexmk foo

then *latexmk* will operate on the file "foo.tex".

-auxdir=FOO or -aux-directory=FOO

Sets the directory for auxiliary output files of (pdf)latex (.aux, .log etc). This achieves its effect by the **-aux-directory** option of (pdf)latex, which currently is only implemented on the MiKTeX version of (pdf)latex.

See also the **-outdir/-output-directory** options, and the *$aux_dir*, *$out_dir*, and *$search_path_separator* configuration variables of *latexmk*. In particular, see the documentation of *$out_dir* for some complications on what directory names are suitable.

-bibtex When the source file uses bbl files for bibliography, run bibtex or biber as needed to regenerate the bbl files.

This property can also be configured by setting the *$bibtex_use* variable to 2 in a configuration file

-bibtex-

Never run bibtex or biber.

A common use for this option is when a document comes from an external source, complete with its bbl file(s), and the user does not have the corresponding bib files available. In this situation use of the **-bibtex-** option will prevent *latexmk* from trying to run *bibtex* or *biber*, which would result in overwriting of the bbl files.

-bibtex-cond

When the source file uses bbl file(s) for the bibliography, run *bibtex* or *biber* as needed to regenerate the bbl files, but only if the relevant bib file(s) exist. Thus when the bib files are not available, *bibtex* or *biber* is not run, thereby avoiding overwriting of the bbl file(s). This is the default setting.

(Note that it is possible for *latexmk* to decide that the bib file does not exist, even though the bib file does exist and *bibtex* or *biber* finds it. The problem is that the bib file may not be in the current directory but in some search path; the places *latexmk* and *bibtex* or *biber* cause to be searched need not be identical. On modern installations of TeX and related programs this problem should not arise, since *latexmk* uses the *kpsewhich* program to do the search, and *kpsewhich* should use the same search path as *bibtex* and *biber*. If this problem arises, use the **-bibtex** option when invoking *latexmk*.)

-bm <message>

A banner message to print diagonally across each page when converting the dvi file to postscript. The message must be a single argument on the command line so be careful with quoting spaces and such.

Note that if the **-bm** option is specified, the **-ps** option is assumed.

-bi <intensity>

How dark to print the banner message. A decimal number between 0 and 1. 0 is black and 1 is white. The default is 0.95, which is OK unless your toner cartridge is getting low.

-bs <scale>

A decimal number that specifies how large the banner message will be printed. Experimentation is necessary to get the right scale for your message, as a rule of thumb the scale should be about equal to 1100 divided by the number of characters in the message. The default is 220.0 which is just right for 5 character messages.

-commands

List the commands used by *latexmk* for processing files, and then exit.

-c Clean up (remove) all regeneratable files generated by *latex* and *bibtex* or *biber* except dvi, postscript and pdf. These files are a combination of log files, aux files, *latexmk*'s database file of source file information, and those with extensions specified in the *@generated_exts* configuration variable. In addition, files specified by the *$clean_ext* configuration variable are removed.

This cleanup is instead of a regular make. See the **-gg** option if you want to do a cleanup then a make.

If *$bibtex_use* is set to 0 or 1, bbl files are counted as non-regeneratable.

If *$cleanup_includes_cusdep_generated* is nonzero, regeneratable files are considered as including those generated by custom dependencies and are also deleted. Otherwise these files are not deleted.

-C Clean up (remove) all regeneratable files generated by *latex* and *bibtex* or *biber*. This is the same as the **-c** option with the addition of dvi, postscript and pdf files, and those specified in the *$clean_full_ext* configuration variable.

This cleanup is instead of a regular make. See the **-gg** option if you want to do a cleanup than a make.

If *$bibtex_use* is set to 0 or 1, bbl files are counted as non-regeneratable.

If *$cleanup_includes_cusdep_generated* is nonzero, regeneratable files are considered as including those generated by custom dependencies and are also deleted. Otherwise these files are not deleted.

-CA (Obsolete). Now equivalent to the **-C** option. See that option for details.

-cd Change to the directory containing the main source file before processing it. Then all the generated files (aux, log, dvi, pdf, etc) will be relative to the source file.

This option is particularly useful when *latexmk* is invoked from a GUI configured to invoke *latexmk* with a full pathname for the source file.

-cd- Do NOT change to the directory containing the main source file before processing it. Then all the generated files (aux, log, dvi, pdf, etc) will be relative to the current directory rather than the source file.

This is the default behavior and corresponds to the behavior of the *latex* and *pdflatex* programs. However, it is not desirable behavior when *latexmk* is invoked by a GUI configured to invoke *latexmk* with a full pathname for the source file. See the **-cd** option.

-CF Remove the file containing the database of source file information, before doing the other actions requested.

-d Set draft mode. This prints the banner message "DRAFT" across your page when converting the dvi file to postscript. Size and intensity can be modified with the **-bs** and **-bi** options. The **-bm** option will override this option as this is really just a short way of specifying:

```
latexmk -bm DRAFT
```

Note that if the **-d** option is specified, the **-ps** option is assumed.

-deps Show a list of dependent files after processing. This is in the form of a dependency list of the form used by the *make* program, and it is therefore suitable for use in a Makefile. It gives an overall view of the files without listing intermediate files, as well as *latexmk* can determine them.

By default the list of dependent files is sent to stdout (i.e., normally to the screen unless you've redirected *latexmk*'s output). But you can set the filename where the list is sent by the **-deps-out=** option.

See the section "USING *latexmk* WITH *make*" for an example of how to use a dependency list with *make*.

Users familiar with GNU *automake* and *gcc* will find that the **-deps** option is very similar in its purpose and results to the **-M** option to *gcc*. (In fact, *latexmk* also has options **-M**, **-MF**, and **-MP** options that behave like those of *gcc*.)

-dependents
Equivalent to **-deps**.

-deps- Do not show a list of dependent files after processing. (This is the default.)

-dependents-
Equivalent to **-deps-**.

-deps-out=FILENAME
Set the filename to which the list of dependent files is written. If the FILENAME argument is omitted or set to "-", then the output is sent to stdout.

Use of this option also turns on the output of the list of dependent files after processing.

-dF Dvi file filtering. The argument to this option is a filter which will generate a filtered dvi file with the extension ".dviF". All extra processing (e.g. conversion to postscript, preview, printing) will then be performed on this filtered dvi file.

Example usage: To use dviselect to select only the even pages of the dvi file:

    latexmk -dF "dviselect even" foo.tex

-diagnostics
Print detailed diagnostics during a run. This may help for debugging problems or to understand *latexmk*'s behavior in difficult situations.

-dvi Generate dvi version of document.

-dvi- Turn off generation of dvi version of document. (This may get overridden, if some other file is made (e.g., a .ps file) that is generated from the dvi file, or if no generated file at all is requested.)

-e <code>
Execute the specified initialization code before processing. The code is *Perl* code of the same form as is used in *latexmk*'s initialization files -- for more details, see the information on the **-r** option, and the section about "Configuration/initialization (RC) files". The code is typically a sequence of assignment statements separated by semicolons.

The code is executed when the **-e** option is encountered during *latexmk*'s parsing of its command line. See the **-r** option for a way of executing initialization code from a file. An error results in *latexmk* stopping. Multiple instances of the **-r** and **-e** options can be used, and they are executed in the order they appear on the command line.

Some care is needed to deal with proper quoting of special characters in the code on the command line. For example, suppose you want to set the latex command to use its -shell-escape option, then under UNIX/LINUX you could use the line

    latexmk -e '$latex=q/latex %O -shell-escape %S/' file.tex

Note that the single quotes block normal UNIX/LINUX command shells from treating the characters inside the quotes as special. (In this example, the q/.../ construct is a *Perl* idiom equivalent to using single quotes. This avoids the complications of getting a quote character inside an already quoted string in a way that is independent of both the shell and the operating-system.)

The above command line will NOT work under MS-Windows with cmd.exe or command.com or 4nt.exe. For MS-Windows with these command shells you could use

    latexmk -e "$latex=q/latex %O -shell-escape %S/" file.tex or

latexmk -e "$latex='latex %O -shell-escape %S'" file.tex

The last two examples will NOT work with UNIX/LINUX command shells.

-f Force *latexmk* to continue document processing despite errors. Normally, when *latexmk* detects that LaTeX or another program has found an error which will not be resolved by further processing, no further processing is carried out.

Note: "Further processing" means the running of other programs or the rerunning of *latex* (etc) that would be done if no errors had occurred. If instead, or additionally, you want the *latex* (etc) program not to pause for user input after an error, you should arrange this by an option that is passed to the program, e.g., by *latexmk*'s option **-interaction=nonstopmode**.

-f- Turn off the forced processing-past-errors such as is set by the **-f** option. This could be used to override a setting in a configuration file.

-g Force *latexmk* to process document fully, even under situations where *latexmk* would normally decide that no changes in the source files have occurred since the previous run. This option is useful, for example, if you change some options and wish to reprocess the files.

-g- Turn off **-g**.

-gg "Super go mode" or "clean make": clean out generated files as if **-C** had been given, and then do a regular make.

-h, -help
Print help information.

-jobname=STRING
Set the basename of output files(s) to STRING, instead of the default, which is the basename of the specified TeX file.

This is like the same option for current implementations of the *latex*, *pdflatex*, etc, and the passing of this option to these programs is part of *latexmk*'s implementation of **-jobname**.

-l Run in landscape mode, using the landscape mode for the previewers and the dvi to postscript converters. This option is not normally needed nowadays, since current previewers normally determine this information automatically.

-l- Turn off **-l**.

-latex="COMMAND"
This sets the string specifying the command to run latex, and is typically used to add desired options. Since the string normally contains spaces, it should be quoted, e.g.,

latexmk -latex="latex --shell-escape %O %S" foo.tex

The specification of the contents of the string are the same as for the *$latex* configuration variable. Depending on your operating system and the command-line shell you are using, you may need to change the single quotes to double quotes (or something else).

To set the command for running pdflatex (rather than the command for latex) see the **-pdflatex** option.

Note that the effect of this option can also be achieved by using the **-e** option with a suitable line of Perl code to set the *$latex* variable. See the explanation of the **-e** option.

-lualatex
Use lualatex. That is, use lualatex to process the source file(s) to pdf (in place of pdflatex). This option is exactly equivalent to specifying the following sequence of options:

```
-pdflatex="lualatex %O %S" -pdf -dvi- -ps-
```

-M Show list of dependent files after processing. This is equivalent to the **-deps** option.

-MF file
If a list of dependents is made, the **-MF** specifies the file to write it to.

-MP If a list of dependents is made, includes phony target for each source file. If you use the dependents list in a Makefile, the dummy rules work around errors make gives if you remove header files without updating the Makefile to match.

-new-viewer
When in continuous-preview mode, always start a new viewer to view the generated file. By default, *latexmk* will, in continuous-preview mode, test for a previously running previewer for the same file and not start a new one if a previous previewer is running. However, its test sometimes fails (notably if there is an already-running previewer that is viewing a file of the same name as the current file, but in a different directory). This option turns off the default behavior.

-new-viewer-
The inverse of the **-new-viewer** option. It puts *latexmk* in its normal behavior that in preview-continuous mode it checks for an already-running previewer.

-nobibtex
Never run bibtex or biber.

A common use for this option is when a document comes from an external source, complete with its bbl file(s), and the user does not have the corresponding bib files available. In this situation use of the **-nobibtex** option will prevent *latexmk* from trying to run *bibtex* or *biber*, which would result in overwriting of the bbl files.

-norc Turn off the automatic reading of initialization (rc) files.

N.B. Normally the initialization files are read and obeyed, and then command line options are obeyed in the order they are encountered. Then **-norc** is an exception to this rule: it is acted on first, no matter where is occurs on the command line.

-outdir=FOO or -output-directory=FOO

Sets the directory for the output files of (pdf)latex. This achieves its effect by the **-output-directory** option of (pdf)latex, which currently (Dec. 2011) is implemented on the common versions of (pdf)latex, i.e., MiKTeX and TeXLive. It may not be present in other versions.

See also the **-auxdir/-aux-directory** options, and the *$aux_dir*, *$out_dir*, and *$search_path_separator* configuration variables of *latexmk*. In particular, see the documentation of *$out_dir* for some complications on what directory names are suitable.

-p Print out the document. By default the file to be printed is the first in the list postscript, pdf, dvi that is being made. But you can use the **-print=...** option to change the type of file to be printed, and you can configure this in a start up file (by setting the *$print_type* variable).

However, printing is enabled by default only under UNIX/LINUX systems, where the default is to use the lpr command and only on postscript files. In general, the correct behavior for printing very much depends on your system's software. In particular, under MS-Windows you must have suitable program(s) available, and you must have configured the print commands used by *latexmk*. This can be non-trivial. See the documentation on the *$lpr*, *$lpr_dvi*, and *$lpr_pdf* configuration variables to see how to set the commands for printing.

This option is incompatible with the **-pv** and **-pvc** options, so it turns them off.

-pdf Generate pdf version of document using *pdflatex*. (If you wish to use *lualatex* or *xelatex*, you can use whichever of the options **-lualatex** or **-xelatex** applies.) To configure *latexmk* to have such behavior by default, see the section on "Configuration/initialization (rc) files".

-pdfdvi
Generate pdf version of document from the dvi file, by default using dvipdf.

-pdfps Generate pdf version of document from the ps file, by default using ps2pdf.

-pdf- Turn off generation of pdf version of document. (This can be used to override a setting in a configuration file. It may get overridden if some other option requires the generation of a pdf file.)

-pdflatex="COMMAND"
This sets the string specifying the command to run pdflatex, and is typically used to add desired options. Since the string normally contains spaces, it should be quoted, e.g.,

```
latexmk -pdf -pdflatex="pdflatex --shell-escape %O %S"  foo.tex
```

The specification of the contents of the string are the same as for the *$pdflatex* configuration variable. Depending on your operating system and the command-line shell you are using, you may need to change the single quotes to double quotes (or something else).

This option can also be used to set a program to be used instead of the standard *pdflatex* program, e.g.,

```
latexmk -pdf -pdflatex="lualatex %O %S"  foo.tex
```

to use *lualatex* or

```
latexmk -pdf -pdflatex="xelatex %O %S"  foo.tex
```

to use *xelatex*.

To set the command for running *latex* (rather than the command for *pdflatex*) see the **-latex** option.

Note that the effect of this option can also be achieved by using the **-e** option with a suitable line of Perl code to set the *$pdflatex* variable. See the explanation of the **-e** option.

-print=dvi, -print=ps, -print=pdf, -print=auto,
Define which kind of file is printed. This option also ensures that the requisite file is made, and turns on printing.

The (default) case **-print=auto** determines the kind of print file automatically from the set of files that is being made. The first in the list postscript, pdf, dvi that is among the files to be made is the one used for print out.

-ps Generate postscript version of document.

-ps- Turn off generation of postscript version of document. This can be used to override a setting in a configuration file. (It may get overridden by some other option that requires a postscript file, for example a request for printing.)

-pF Postscript file filtering. The argument to this option is a filter which will generate a filtered postscript file with the extension ".psF". All extra processing (e.g. preview, printing) will then be performed on this filtered postscript file.

Example of usage: Use psnup to print two pages on the one page:

latexmk -ps -pF 'psnup -2' foo.tex

or

latexmk -ps -pF "psnup -2" foo.tex

Whether to use single or double quotes round the "psnup -2" will depend on your command interpreter, as used by the particular version of perl and the operating system on your computer.

-pv Run file previewer. If the **-view** option is used, this will select the kind of file to be previewed (dvi, ps or pdf). Otherwise the viewer views the "highest" kind of file selected, by the **-dvi**, **-ps**, **-pdf**, **-pdfps** options, in the order dvi, ps, pdf (low to high). If no file type has been selected, the dvi previewer will be used. This option is incompatible with the **-p** and **-pvc** options, so it turns them off.

-pv- Turn off **-pv**.

-pvc Run a file previewer and continually update the .dvi, .ps, and/or .pdf files whenever changes are made to source files (see the Description above). Which of these files is generated and which is viewed is governed by the other options, and is the same as for the **-pv** option. The preview-continuous option **-pvc** can only work with one file. So in this case you will normally only specify one filename on the command line. It is also incompatible with the **-p** and **-pv** options, so it turns these options off.

The **-pvc** option also turns off force mode (**-f**), as is normally best for continuous preview mode. If you really want force mode, use the options in the order **-pvc -f**.

With a good previewer the display will be automatically updated. (Under *some but not all* versions of UNIX/Linux "gv -watch" does this for postscript files; this can be set by a configuration variable. This would also work for pdf files except for an apparent bug in gv that causes an error when the newly updated pdf file is read.) Many other previewers will need a manual update.

Important note: the acroread program on MS-Windows locks the pdf file, and prevents new versions being written, so it is a bad idea to use acroread to view pdf files in preview-continuous mode. It is better to use a different viewer: SumatraPDF and gsview are good possibilities.

There are some other methods for arranging an update, notably useful for many versions of xdvi and xpdf. These are best set in *latexmk*'s configuration; see below.

Note that if *latexmk* dies or is stopped by the user, the "forked" previewer will continue to run. Successive invocations with the **-pvc** option will not fork new previewers, but *latexmk* will normally use the existing previewer. (At least this will happen when *latexmk* is running under an operating system where it knows how to determine whether an existing previewer is running.)

-pvc- Turn off **-pvc**.

-quiet Same as -silent

-r <rcfile>
Read the specified initialization file ("RC file") before processing.

Be careful about the ordering: (1) Standard initialization files -- see the section below on "Configuration/initialization (RC) files" -- are read first. (2) Then the options on the command line are acted on in the order they are given. Therefore if an initialization file is specified by the **-r** option, it is read during this second step. Thus an initialization file specified with the **-r** option can override both the standard initialization files and *previously* specified options. But all of these can be overridden by *later* options.

The contents of the RC file just comprise a piece of code in the *Perl* programming language (typically a sequence of assignment statements); they are executed when the **-r** option is encountered during *latexmk*'s parsing of its command line. See the **-e** option for a way of giving initialization code directly on *latexmk*'s command line. An error results in *latexmk* stopping. Multiple instances of the **-r** and **-e** options can be used, and they are executed in the order they appear on the command line.

-recorder
Use the -recorder option with *latex* and *pdflatex*. In (most) modern versions of these programs, this results in a file of extension *.fls* containing a list of the files that these programs have read and written. *Latexmk* will then use this file to improve its detection of source files and generated files after a run of *latex* or *pdflatex*.

For further information, see the documentation for the *$recorder* configuration variable.

-recorder-
Do not use the -recorder option with *latex* and *pdflatex*.

-rules Show a list of *latemk*'s rules and dependencies after processing.

-rules- Do not show a list of *latexmk*'s rules and dependencies after processing. (This is the default.)

-showextraoptions
Show the list of extra *latex* and *pdflatex* options that *latexmk* recognizes. These are options for the *latex* and *pdflatex* that *latexmk* recognizes, but simply passes through to these programs when they are run. These options are (currently) a combination of those allowed by the TeXLive and MiKTeX implementations. (If a particular option is given to *latexmk* but is not handled by the particular implementation of *latex* or *pdflatex* that is being used, that program will probably give an error message.) These options are very numerous, but are not listed in this documentation because they have no effect on *latexmk*'s actions.

There are a few options (**-includedirectory=dir**, **-initialize**, **-ini**) that are not recognized, either because they don't fit with *latexmk*'s intended operations, or because they need special processing by *latexmk* that isn't implemented (at least, not yet).

There are also options that are accepted by *latex* etc, but instead trigger actions by *latexmk*: **-help**, **-version**.

Finally, there are certain options for *latex* and *pdflatex* (e.g., **-recorder**) that trigger special actions or behavior by *latexmk* itself as well as being passed in some form to the called *latex* and *pdflatex* program, or that affect other programs as well. These options do have entries in this documentation. These options are: **-jobname=STRING**, **-aux-directory=dir**, **-output-directory=DIR**, **-quiet**, and **-recorder**.

-silent Run commands silently, i.e., with options that reduce the amount of diagnostics generated. For example, with the default settings, the command "latex -interaction=batchmode" is used for latex.

Also reduce the number of informational messages that *latexmk* generates.

To change the options used to make the commands run silently, you need to configure *latexmk* with changed values of its configuration variables, the relevant ones being *$bibtex_silent_switch*, *$biber_silent_switch*, *$dvipdf_silent_switch*, *$dvips_silent_switch*, *$latex_silent_switch*, *$makeindex_silent_switch*, and *$pdflatex_silent_switch*.

-use-make

When after a run of *latex* or *pdflatex*, there are warnings about missing files (e.g., as requested by the LaTeX \input, \include, and \includgraphics), *latexmk* tries to make them by a custom dependency. If no relevant custom dependency with an appropriate source file is found, and if the -use-make option is set, then *latexmk* will try as a resort using the make program to try to make the missing files.

Note that the filename may be specified without an extension, e.g., by \includegraphics{drawing} in a LaTeX file. In that case, *latexmk* will try making drawing.ext with ext set in turn to the possible extensions that are relevant for *latex* (or as appropriate *pdflatex*).

See also the documentation for the *$use_make_for_missing_files* configuration variable.

-use-make-

Do not use the make program to try to make missing files. (Default.)

-v, -version

Print version number of *latexmk*.

-verbose

Opposite of **-silent**. This is the default setting.

-view=default, -view=dvi, -view=ps, -view=pdf

Set the kind of file used when previewing is requested (e.g., by the **-pv** or **-pvc** switches). The default is to view the "highest" kind of requested file (in the order dvi, ps, pdf).

-xelatex

Use xelatex. That is, use xelatex to process the source file(s) to pdf (in place of pdflatex). This option is exactly equivalent to specifying the following sequence of options:

```
-pdflatex="xelatex %O %S" -pdf -dvi- -ps-
```

Compatibility between options

The preview-continuous option **-pvc** can only work with one file. So in this case you will normally only specify one filename on the command line.

Options **-p**, **-pv** and **-pvc** are mutually exclusive. So each of these options turns the others off.

EXAMPLES

% **latexmk thesis**	*# run latex enough times to resolve cross-references*
% **latexmk -pvc -ps thesis**	*# run latex enough times to resolve cross-references, make a postscript file, start a previewer. Then watch for changes in the source file thesis.tex and any files it uses. After any changes rerun latex the appropriate number of times and remake the postscript file. If latex encounters an error, latexmk will keep running, watching for*

source file changes.

% **latexmk -c** *# remove .aux, .log, .bbl, .blg, .dvi, .pdf, .ps & .bbl files*

HOW TO CHANGE THE WAY LATEXMK BEHAVES, DEALING WITH PROBLEMS, ETC

Some possibilities:

a. If you get a strange error, do look carefully at the output that is on the screen and in log files. While there is much that is notoriously verbose in the output of *latex* (and that is added to by *latexmk*), the verbosity is there for a reason: to enable the user to diagnose problems. *Latexmk* does repeat some messages at the end of a run that it thinks would otherwise be easy to miss in the middle of other output.

b. If *latexmk* doesn't do things the way you would like, the first step in finding out what to do is to look in this documentation at the list of command line options and then at the sections on configuration/initialization files. A lot of *latexmk*'s behavior is configurable to deal with particular situations. (But there is a lot of reading.)

c. Further tricks can involve replacing the standard commands that *latexmk* by other commands or scripts.

d. For possible examples of code for use in an RC file, see the directory example_rcfiles in the distribution of *latexmk* (e.g., at http://www.ctan.org/tex-archive/support/latexmk/example_rcfiles). Even if these examples don't do what you want, they may provide suitable inspiration.

e. There's a useful trick that can be used when you are *lualatex* instead of *pdflatex* (and in some related situations). The problem is that *latexmk* won't notice a dependency on a file, bar.baz say, that is input by the lua code in your document instead of by the LaTeX part. (Thus if you change bar.baz and rerun *latexmk*, then *latexmk* will think no files have changed and not rerun *lualatex*, whereas if you had '\input{bar.baz}' in the LaTeX part of the document, *latexmk* would notice the change.) One solution is just to put the following somewhere in the LaTeX part of the document:

```
\typeout{(bar.baz)}
```

This puts a line in the log file that *latexmk* will treat as implying that the file bar.baz was read. (At present I don't know a way of doing this automatically.) Of course, if the file has a different name, change bar.baz to the name of your file.

f. Look on tex.stackexchange, i.e., at http://tex.stackexchange.com/questions/tagged/latexmk Someone may have already solved your problem.

g. Ask a question at tex.stackexchange.com.

h. Or ask me (the author of *latexmk*). My e-mail is at the end of this documentation.

CONFIGURATION/INITIALIZATION (RC) FILES

Latexmk can be customized using initialization files, which are read at startup in the following order:

1) The system RC file, if it exists.
On a UNIX system, *latexmk* searches for following places for its system RC file, in the following order, and reads the first it finds:
"/opt/local/share/latexmk/LatexMk",
"/usr/local/share/latexmk/LatexMk",

"/usr/local/lib/latexmk/LatexMk".
On a MS-WINDOWS system it looks for "C:\latexmk\LatexMk".
On a cygwin system (i.e., a MS-Windows system in which Perl is that of cygwin), *latexmk* reads for the first it finds of
"/cygdrive/c/latexmk/LatexMk",
"/opt/local/share/latexmk/LatexMk",
"/usr/local/share/latexmk/LatexMk",
"/usr/local/lib/latexmk/LatexMk".

2) The user's RC file, "$HOME/.latexmkrc", if it exists. Here $HOME is the user's home directory. [*Latexmk* determines the user's home directory as follows: It is the value of the environment variable HOME, if this variable exists, which normally is the case on UNIX-like systems (including LINUX and OS-X). Otherwise the environment variable USERPROFILE is used, if it exists, which normally is the case on MS-Windows systems. Otherwise a blank string is used instead of $HOME.]

3) The RC file in the current working directory. This file can be named either "latexmkrc" or ".latexmkrc", and the first of these to be found is used, if any.

4) Any RC file(s) specified on the command line with the **-r** option.

Each RC file is a sequence of *Perl* commands. Naturally, a user can use this in creative ways. But for most purposes, one simply uses a sequence of assignment statements that override some of the built-in settings of *Latexmk*. Straightforward cases can be handled without knowledge of the *Perl* language by using the examples in this document as templates. Comment lines are introduced by the "#" character.

Note that command line options are obeyed in the order in which they are written; thus any RC file specified on the command line with the **-r** option can override previous options but can be itself overridden by later options on the command line. There is also the **-e** option, which allows initialization code to be specified in *latexmk*'s command line.

For possible examples of code for in an RC file, see the directory example_rcfiles in the distribution of latexmk (e.g., at http://www.ctan.org/tex-archive/support/latexmk/example_rcfiles).

HOW TO SET VARIABLES IN INITIALIZATION FILES

The important variables that can be configured are described in the section "List of configuration variables usable in initialization files". Syntax for setting these variables is of the following forms:

```
$bibtex = 'bibtex %O %B';
```

for the setting of a string variable,

```
$preview_mode = 1;
```

for the setting of a numeric variable, and

```
@default_files = ('paper', 'paper1');
```

for the setting of an array of strings. It is possible to append an item to an array variable as follows:

```
push @default_files, 'paper2';
```

Note that simple "scalar" variables have names that begin with a $ character and array variables have names that begin with a @ character. Each statement ends with a semicolon.

Strings should be enclosed in single quotes. (You could use double quotes, as in many programming languages. But then the *Perl* programming language brings into play some special rules for interpolating variables into strings. People not fluent in *Perl* will want to avoid these complications.)

You can do much more complicated things, but for this you will need to consult a manual for the *Perl* programming language.

FORMAT OF COMMAND SPECIFICATIONS

Some of the variables set the commands that *latexmk* uses for carrying out its work, for example to generate a dvi file from a tex file or to view a postscript file. This section describes some important features of how the commands are specified.

Placeholders: Supposed you wanted *latexmk* to use the command elatex in place of the regular latex command, and suppose moreover that you wanted to give it the option "--shell-escape". You could do this by the following setting:

$latex = 'elatex --shell-escape %O %S';

The two items starting with the % character are placeholders. These are substituted by appropriate values before the command is run. Thus %S will be replaced by the source file that elatex will be applied to, and %O will be replaced by any options that *latexmk* has decided to use for this command. (E.g., if you used the -silent option it would replace %O by "-interaction=batchmode".)

The available placeholders are:

%B base of filename for current command. E.g., if a postscript file document.ps is being made from the dvi file document.dvi, then the basename is document.

%D destination file (e.g., the name of the postscript file when converting a dvi file to postscript).

%O options

%R root filename. This is the base name for the main tex file.

%S source file (e.g., the name of the dvi file when converting a dvi file to ps).

%T The name of the primary tex file.

%Y Name of directory for auxiliary output files (see the configuration variable *$aux_dir*). A directory separation character ('/') is appended if *$aux_dir* is non-empty and does not end in a suitable character, with suitable characters being those appropriate to UNIX and MS-Windows, i.e., ':', '/' and '\'.

%Z Name of directory for output files (see the configuration variable *$out_dir*). A directory separation character ('/') is appended if *$out_dir* is non-empty and does not end in a suitable character, with suitable characters being those appropriate to UNIX and MS-Windows, i.e., ':', '/' and '\'.

If for some reason you need a literal % character in your string not subject to the above rules, use a pair of these characters. Thus with the command specification $ps_previewer = 'latex -ad=%%Sfile.ad %S', the %%S will become %S when the command is executed, but the %S will be replaced by the source filename, which in this case would be the name of a postscript file to be viewed.

Appropriate quoting will be applied to the filename substitutions, so you mustn't supply them yourself even if the names of your files have spaces in them. (But if your TeX filenames have spaces in them, beware that many versions of the TeX program cannot correctly handle filenames containing spaces.) In case latexmk's quoting does not work correctly on your system, you can turn it off -- see the documentation for the variable *$quote_filenames*.

The distinction between %B and %R needs a bit of care, since they are often the same, but not always. For example on a simple document, the basename of a bibtex run is the same as for the texfile. But in a document with several bibliographies, the bibliography files will have a variety of names. Since bibtex is invoked with the basename of the bibliography file, the setting for the bibtex command should therefore be

$bibtex = 'bibtex %O %B';

Generally, you should use %B rather than %R. Similarly for most purposes, the name %T of the primary texfile is not a useful placeholder.

See the default values in the section "List of configuration variables usable in initialization files" for what is normally the most appropriate usage.

If you omit to supply any placeholders whatever in the specification of a command, *latexmk* will supply what its author thinks are appropriate defaults. This gives compatibility with configuration files for previous versions of *latexmk*, which didn't use placeholders.

"Detaching" a command: Normally when *latexmk* runs a command, it waits for the command to run to completion. This is appropriate for commands like latex, of course. But for previewers, the command should normally run detached, so that *latexmk* gets the previewer running and then returns to its next task (or exits if there is nothing else to do). To achieve this effect of detaching a command, you need to precede the command name with "start ", as in

    $dvi_previewer = 'start xdvi %O %S';

This will be translated to whatever is appropriate for your operating system.

Notes: (1) In some circumstances, *latex* will always run a command detached. This is the case for a previewer in preview continuous mode, since otherwise previewing continuously makes no sense. (2) This precludes the possibility of running a command named start. (3) If the word start occurs more than once at the beginning of the command string, that is equivalent to having just one. (4) Under cygwin, some complications happen, since cygwin amounts to a complicated merging of UNIX and MS-Windows. See the source code for how I've handled the problem.

Command names containing spaces: Under MS-Windows it is common that the name of a command includes spaces, since software is often installed in a subdirectory of "C:\Program Files". Such command names should be enclosed in double quotes, as in

    $lpr_pdf = '"c:/Program Files/Ghostgum/gsview/gsview32.exe" /p %S';
    $pdf_previewer = 'start "c:/Program Files/SumatraPDF/SumatraPDF.exe" %O %S';
    $pdf_previewer = 'start "c:/Program Files/SumatraPDF (x86)/SumatraPDF.exe" %O %S';

(Note about the above example: Forward slashes are equivalent to backslashes in filenames under MS-Windows, provided that the filename is inside double quotes. It is easier to use forward slashes in examples like the one above, since then one does not have to worry about the rules for dealing with forward slashes in strings in the Perl language.)

Command names under Cygwin: If latexmk is executed by Cygwin's Perl, *be particularly certain that pathnames in commands have* **forward** *slashes* not the usual backslashes for the separator of pathname components. See the above examples. Backslashes often get misinterpreted by the Unix shell used by Cygwin's Perl to execute external commands. Forward slashes don't suffer from this problem, and (when quoted, as above) are equally acceptable to MS-Windows.

Using MS-Windows file associations: A useful trick under modern versions of MS-Windows (e.g., WinXP) is to use just the command 'start' by itself:

    $dvi_previewer = 'start %S';

Under recent versions of MS-Windows, this will cause to be run whatever program the system has associated with dvi files. (The same applies for a postscript viewer and a pdf viewer.) But note that this trick is not always suitable for the pdf previwer, if your system has acroread for the default pdf viewer. As explained elsewhere, acroread under MS-Windows does not work well with *latex* and *latexmk*, because acroread locks the pdf file.

Not using a certain command: If a command is not to be run, the command name NONE is used, as in

    $lpr = 'NONE lpr';

This typically is used when an appropriate command does not exist on your system. The string after the "NONE" is effectively a comment.

Options to commands: Setting the name of a command can be used not only for changing the name of the command called, but also to add options to command. Suppose you want *latexmk* to use latex with source specials enabled. Then you might use the following line in an initialization file:

```
$latex = 'latex --src-specials %O %S';
```

Running a subroutine instead of an external command: Use a specification starting with "internal", as in

```
$latex = 'internal mylatex %O %S';
sub mylatex {
   my @args = @_;
   # Possible preprocessing here
   return system 'latex', @args;
}
```

Advanced tricks: Normally one specifies a single command for the commands invoked by *latexmk*. Naturally, if there is some complicated additional processing you need to do in your special situation, you can write a script (or batch file) to do the processing, and then configure *latexmk* to use your script in place of the standard program.

It is also possible to configure *latexmk* to run multiple commands. For example, if when running pdflatex to generate a pdf file from a tex file you need to run another program after pdflatex to perform some extra processing, you could do something like:

```
$pdflatex = 'pdflatex --shell-escape %O %S; pst2pdf_for_latexmk %B';
```

This definition assumes you are using a UNIX-like system (which includes Linux and OS-X), so that the two commands to be run are separated by the semicolon in the middle of the string.

If you are using MS-Windows, you would replace the above line by

```
$pdflatex = 'cmd /c pdflatex --shell-escape %O %S'
          . '&& pst2pdf_for_latexmk %B';
```

Here, the UNIX command separator ; is replaced by &&. In addition, there is a problem that some versions of *Perl* on MS-Windows do not obey the command separator; this problem is overcome by explicitly invoking the MS-Windows command-line processor *cmd.exe*.

LIST OF CONFIGURATION VARIABLES USABLE IN INITIALIZATION FILES

Default values are indicated in brackets.

$always_view_file_via_temporary [0]

Whether ps and pdf files are initially to be made in a temporary directory and then moved to the final location. (This applies to dvips, dvipdf, and ps2pdf operations, and the filtering operators on dvi and ps files. It does not apply to pdflatex, unfortunately.)

This use of a temporary file solves a problem that the making of these files can occupy a substantial time. If a viewer sees that the file has changed, it reads the new file, and this can cause havoc if the program writing the file has not yet finished its work.

See the *$pvc_view_file_via_temporary* variable for a setting that applies only if preview-continuous mode (-pvc option) is used. See *$tmpdir* for the setting of the directory where the temporary file is created.

$auto_rc_use [1]

Whether to automatically read the standard initialization (rc) files, which are the system RC file, the user's RC file, and the RC file in the current directory. The command line option **-norc** can be used to turn this setting off. Each RC file could also turn this setting off, i.e., it could set *$auto_rc_use* to zero to prevent automatic reading of the later RC files.

This variable does not affect the reading of RC files specified on the command line by the **-r** option.

$aux_dir [""]

The directory in which auxiliary files (aux, log, etc) are to be written by a run of (pdf)latex. If this variable is not set, but *$out_dir* is set, then *$aux_dir* is set to *$out_dir*, which is the directory to which general output files are to be written.

Important note: The effect of *$aux_dir*, if different from *$out_dir*, is achieved by giving (pdf)latex the **-aux-directory**. Currently (Dec. 2011) this only works on the MiKTeX version of (pdf)latex.

See also the documentation of *$out_dir* for some complications on what directory names are suitable.

$banner [0]

If nonzero, the banner message is printed across each page when converting the dvi file to postscript. Without modifying the variable *$banner_message*, this is equivalent to specifying the **-d** option.

Note that if **$banner** is nonzero, the **$postscript_mode** is assumed and the postscript file is always generated, even if it is newer than the dvi file.

$banner_intensity [0.95]

Equivalent to the **-bi** option, this is a decimal number between 0 and 1 that specifies how dark to print the banner message. 0 is black, 1 is white. The default is just right if your toner cartridge isn't running too low.

$banner_message ["DRAFT"]

The banner message to print across each page when converting the dvi file to postscript. This is equivalent to the **-bm** option.

$banner_scale [220.0]

A decimal number that specifies how large the banner message will be printed. Experimentation is necessary to get the right scale for your message, as a rule of thumb the scale should be about equal to 1100 divided by the number of characters in the message. The Default is just right for 5 character messages. This is equivalent to the **-bs** option.

@BIBINPUTS

This is an array variable, now mostly obsolete, that specifies directories where *latexmk* should look for .bib files. By default it is set from the BIBINPUTS environment variable of the operating system. If that environment variable is not set, a single element list consisting of the current directory is set. The format of the directory names depends on your operating system, of course. Examples for setting this variable are:

```
@BIBINPUTS = ( ".", "C:\bibfiles" );
@BIBINPUTS = ( ".", "\\server\bibfiles" );
@BIBINPUTS = ( ".", "C:/bibfiles" );
@BIBINPUTS = ( ".", "//server/bibfiles" );
@BIBINPUTS = ( ".", "/usr/local/texmf/bibtex/bib" );
```

Note that under MS Windows, either a forward slash "/" or a backward slash "\" can be used to separate pathname components, so the first two and the second two examples are equivalent. Each backward slash should be doubled to avoid running afoul of *Perl*'s rules for writing strings.

Important note: This variable is now mostly obsolete in the current version of *latexmk*, since it has a better method of searching for files using the kpsewhich command. However, if your system is an unusual one without the kpsewhich command, you may need to set the variable *@BIBINPUTS*.

$biber ["biber %O %S"]
The biber processing program.

$biber_silent_switch ["--onlylog"]
Switch(es) for the biber processing program when silent mode is on.

$bibtex ["bibtex %O %S"]
The BibTeX processing program.

$bibtex_silent_switch ["-terse"]
Switch(es) for the BibTeX processing program when silent mode is on.

$bibtex_use [1]
Under what conditions to run BibTeX or biber. When *latexmk* discovers from the log file that one (or more) BibTeX/biber-generated bibliographies are used, it can run BibTeX or biber whenever it appears necessary to regenerate the bbl file(s) from their source bib database file(s).

But sometimes, the bib file(s) are not available (e.g., for a document obtained from an external archive), but the bbl files are provided. In that case use of BibTeX or biber will result in incorrect overwriting of the precious bbl files. The variable *$bibtex_use* controls whether this happens. Its possible values are: 0: never use BibTeX or biber. 1: only use BibTeX or biber if the bib files exist. 2: run BibTeX or biber whenever it appears necessary to update the bbl files, without testing for the existence of the bib files.

$cleanup_includes_cusdep_generated [0]
If nonzero, specifies that cleanup also deletes files that are generated by custom dependencies. (When doing a clean up, e.g., by use of the **-C** option, custom dependencies are those listed in the *.fdb_latexmk* file from a previous run.)

$cleanup_includes_generated [0]
If nonzero, specifies that cleanup also deletes files that are detected in log file as being generated (see the \openout lines in the log file). It will also include files made from these first generation generated files.

$cleanup_mode [0]
If nonzero, specifies cleanup mode: 1 for full cleanup, 2 for cleanup except for dvi, ps and pdf files, 3 for cleanup except for dep and aux files. (There is also extra cleaning as specified by the *$clean_ext*, *$clean_full_ext* and *@generated_exts* variables.)

This variable is equivalent to specifying one of the **-c** or **-C** options. But there should be no need to set this variable from an RC file.

$clean_ext [""]
Extra extensions of files for *latexmk* to remove when any of the clean-up options (**-c** or **-C**) is selected. The value of this variable is a string containing the extensions separated by spaces.

It is also possible to specify a more general pattern of file to be deleted, by using the place holder %R, as in commands, and it is also possible to use wildcards. Thus setting

```
$clean_ext = "out %R-blx.bib %R-figures*.log";
```

in an initialization file will imply that when a clean-up operation is specified, not only is the standard set of files deleted, but also files of the form FOO.out, FOO-blx.bib, and %R-figures*.log, where FOO stands for the basename of the file being processed (as in FOO.tex).

$clean_full_ext [""]

Extra extensions of files for *latexmk* to remove when the **-C** option is selected, i.e., extensions of files to remove when the .dvi, etc files are to be cleaned-up.

More general patterns are allowed, as for *$clean_ext*.

$compiling_cmd [undefined], $failure_cmd [undefined], $success_cmd [undefined]

These variables specify commands that are executed at certain points of compilations during preview-continuous mode. One motivation for their existance is to allow convenient visual indications of compilation status even when the window receiving the screen output of the compilation is hidden.

The commands are executed at the following points: $compiling_cmd at the start of compilation, $success_cmd at the end of a successful compilation, and $failure_cmd at the end of an unsuccessful compilation. If any of above variables is undefined (the default situation) or blank, then the corresponding command is not executed.

An example of a typical setting of these variables is as follows

```
$compiling_cmd = "xdotool search --name \"%D\" set_window --name \"%D compiling\"";
$success_cmd   = "xdotool search --name \"%D\" set_window --name \"%D OK\"";
$failure_cmd   = "xdotool search --name \"%D\" set_window --name \"%D FAILURE\"";
```

These assume that the program *xdotool* is installed, that the previewer is using an X-Window system for display, and that the title of the window contains the name of the displayed file, as it normally does. When the commands are executed, the placeholder string %D is replaced by the name of the destination file, which is the previewed file. The above commands result in an appropriate string being appended to the filename in the window title: " compiling", " OK", or " FAILURE".

Other placeholders that can be used are %S, %T, and %R, with %S and %T normally being identical. These can be useful for a command changing the title of the edit window. The visual indication in a window title can useful, since the user does not have to keep shifting attention to the (possibly hidden) compilation window to know the status of the compilation.

@cus_dep_list [()]

Custom dependency list -- see section on "Custom Dependencies".

@default_files [("*.tex")]

Default list of files to be processed.

Normally, if no filenames are specified on the command line, *latexmk* processes all tex files specified in the *@default_files* variable, which by default is set to all tex files ("*.tex") in the current directory. This is a convenience: just run *latexmk* and it will process an appropriate set of files. But sometimes you want only some of these files to be processed. In this case you set the *@default_files* in an initialization file (e.g., the file "latexmkrc" in the current directory). Then if no files are specified on the command line then the files you specify by setting *@default_files* are processed.

Three examples:

@default_files = ("paper_current");

@default_files = ("paper1", "paper2.tex");

@default_files = ("*.tex", "*.dtx");

Note that more than file may be given, and that the default extension is ".tex". Wild cards are allowed. The parentheses are because *@default_files* is an array variable, i.e., a sequence of filename specifications is possible.

$dependents_phony [0]
If a list of dependencies is output, this variable determines whether to include a phony target for each source file. If you use the dependents list in a Makefile, the dummy rules work around errors make gives if you remove header files without updating the Makefile to match.

$dependents_list [0]
Whether to display a list(s) of dependencies at the end of a run.

$do_cd [0]
Whether to change working directory to the directory specified for the main source file before processing it. The default behavior is not to do this, which is the same as the behavior of *latex* and *pdflatex* programs. This variable is set by the **-cd** and **-cd-** options on *latexmk*'s command line.

$dvi_filter [empty]
The dvi file filter to be run on the newly produced dvi file before other processing. Equivalent to specifying the **-dF** option.

$dvi_mode [See below for default]
If nonzero, generate a dvi version of the document. Equivalent to the **-dvi** option.

The variable $dvi_mode defaults to 0, but if no explicit requests are made for other types of file (postscript, pdf), then $dvi_mode will be set to 1. In addition, if a request for a file for which a .dvi file is a prerequisite, then $dvi_mode will be set to 1.

$dvi_previewer ["start xdvi %O %S" under UNIX]
The command to invoke a dvi-previewer. [Default is "start" under MS-WINDOWS; under more recent versions of Windows, this will cause to be run whatever command the system has associated with .dvi files.]

Important note: Normally you will want to have a previewer run detached, so that *latexmk* doesn't wait for the previewer to terminate before continuing its work. So normally you should prefix the command by "start ", which flags to *latexmk* that it should do the detaching of the previewer itself (by whatever method is appropriate to the operating system). But sometimes letting *latexmk* do the detaching is not appropriate (for a variety of non-trivial reasons), so you should put the "start " bit in yourself, whenever it is needed.

$dvi_previewer_landscape ["start xdvi %O %S"]
The command to invoke a dvi-previewer in landscape mode. [Default is "start" under MS-WINDOWS; under more recent versions of Windows, this will cause to be run whatever command the system has associated with .dvi files.]

$dvipdf ["dvipdf %O %S %D"]
Command to convert dvi to pdf file. A common reconfiguration is to use the dvipdfm command, which needs its arguments in a different order:

$dvipdf = "dvipdfm %O -o %D %S";

WARNING: The default dvipdf script generates pdf files with bitmapped fonts, which do not look good when viewed by acroread. That script should be modified to give dvips the options "-P pdf" to ensure that type 1 fonts are used in the pdf file.

$dvipdf_silent_switch ["-q"]
Switch(es) for dvipdf program when silent mode is on.

N.B. The standard dvipdf program runs silently, so adding the silent switch has no effect, but is actually innocuous. But if an alternative program is used, e.g., dvipdfmx, then the silent switch has an effect. The default setting is correct for dvipdfm and dvipdfmx.

$dvips ["dvips %O -o %D %S"]
The program to used as a filter to convert a .dvi file to a .ps file. If pdf is going to be generated from pdf, then the value of the $dvips_pdf_switch -- see below -- will be included in the options substituted for "%O".

$dvips_landscape ["dvips -tlandscape %O -o %D %S"]
The program to used as a filter to convert a .dvi file to a .ps file in landscape mode.

$dvips_pdf_switch ["-P pdf"]
Switch(es) for dvips program when pdf file is to be generated from ps file.

$dvips_silent_switch ["-q"]
Switch(es) for dvips program when silent mode is on.

$dvi_update_command [""]
When the dvi previewer is set to be updated by running a command, this is the command that is run. See the information for the variable *$dvi_update_method* for further information, and see information on the variable *$pdf_update_method* for an example for the analogous case of a pdf previewer.

$dvi_update_method [2 under UNIX, 1 under MS-Windows]
How the dvi viewer updates its display when the dvi file has changed. The values here apply equally to the *$pdf_update_method* and to the *$ps_update_method* variables.
0 => update is automatic,
1=> manual update by user, which may only mean a mouse click on the viewer's window or may mean a more serious action.
2 => Send the signal, whose number is in the variable *$dvi_update_signal*. The default value under UNIX is suitable for xdvi.
3 => Viewer cannot do an update, because it locks the file. (As with acroread under MS-Windows.)
4 => run a command to do the update. The command is specified by the variable *$dvi_update_command*.

See information on the variable *$pdf_update_method* for an example of updating by command.

$dvi_update_signal [Under UNIX: SIGUSR1, which is a system-dependent value]
The number of the signal that is sent to the dvi viewer when it is updated by sending a signal -- see the information on the variable *$dvi_update_method*. The default value is the one appropriate for xdvi on a UNIX system.

$failure_cmd [undefined]
See the documentation for $compiling_cmd.

$fdb_ext ["fdb_latexmk"]
The extension of the file which *latexmk* generates to contain a database of information on source files. You will not normally need to change this.

$force_mode [0]
If nonzero, continue processing past minor *latex* errors including unrecognized cross references. Equivalent to specifying the **-f** option.

@generated_exts [(aux , bbl , idx , ind , lof , lot , out , toc , $fdb_ext)]

This contains a list of extensions for files that are generated during a LaTeX run and that are read in by LaTeX in later runs, either directly or indirectly.

This list has two uses: (a) to set the kinds of file to be deleted in a cleanup operation (with the **-c**, **-C**, **-CA**, **-g** and **-gg** options), and (b) in the determination of whether a rerun of (pdf)LaTeX is needed after a run that gives an error.

(Normally, a change of a source file during a run should provoke a rerun. This includes a file generated by LaTeX, e.g., an aux file, that is read in on subsequent runs. But after a run that results in an error, a new run should occur until the user has made a change in the files. But the user may have corrected an error in a source .tex file during the run. So *latexmk* needs to distinguish user-generated and automatically generated files; it determines the automatically generated files as those with extensions in the list in @generated_exts.)

A convenient way to add an extra extension to the list, without losing the already defined ones is to use a push command in the line in an RC file. E.g.,

```
push @generated_exts, "end";
```

adds the extension "end" to the list of predefined generated extensions. (This extension is used by the RevTeX package, for example.)

$go_mode [0]

If nonzero, process files regardless of timestamps, and is then equivalent to the **-g** option.

%hash_calc_ignore_pattern

!!!This variable is for experts only!!!

The general rule *latexmk* uses for determining when an extra run of some program is needed is that one of the source files has changed. But consider for example a latex package that causes an encapsulated postscript file (an "eps" file) to be made that is to be read in on the next run. The file contains a comment line giving its creation date and time. On the next run the time changes, *latex* sees that the eps file has changed, and therefore reruns latex. This causes an infinite loop, that is only terminated because *latexmk* has a limit on the number of runs to guard against pathological situations.

But the changing line has no real effect, since it is a comment. You can instruct *latex* to ignore the offending line as follows:

```
$hash_calc_ignore_pattern{'eps'} = '^%%CreationDate: ';
```

This creates a rule for files with extension *.eps* about lines to ignore. The left-hand side is a *Perl* idiom for setting an item in a hash. Note that the file extension is specified without a period. The value, on the right-hand side, is a string containing a regular expresssion. (See documentation on *Perl* for how they are to be specified in general.) This particular regular expression specifies that lines beginning with "%%CreationDate: " are to be ignored in deciding whether a file of the given extension *.eps* has changed.

There is only one regular expression available for each extension. If you need more one pattern to specify lines to ignore, then you need to combine the patterns into a single regular expression. The simplest method is separate the different simple patterns by a vertical bar character (indicating "alternation" in the jargon of regular expressions). For example,

```
$hash_calc_ignore_pattern{'eps'} = '^%%CreationDate: |^%%Title: ';
```

causes lines starting with either "^%%CreationDate: " or "^%%Title: " to be ignored.

It may happen that a pattern to be ignored is specified in, for example, in a system or user initialization file, and you wish to remove this in a file read later. To do this, you use Perl's delete function, e.g.,

    delete $hash_calc_ignore_pattern{'eps'};

$kpsewhich ["kpsewhich %S"]

The program called to locate a source file when the name alone is not sufficient. Most filenames used by *latexmk* have sufficient path information to be found directly. But sometimes, notably when .bib files are found from the log file of a bibtex or biber run, the name of the file, but not its path is known. The program specified by $kpsewhich is used to find it.

See also the *@BIBINPUTS* variable for another way that *latexmk* also uses to try to locate files; it applies only in the case of .bib files.

$landscape_mode [0]

If nonzero, run in landscape mode, using the landscape mode previewers and dvi to postscript converters. Equivalent to the **-l** option. Normally not needed with current previewers.

$latex ["latex %O %S"]

The LaTeX processing program. Note that as with other programs, you can use this variable not just to change the name of the program used, but also specify options to the program. E.g.,

    $latex = "latex --src-specials";

%latex_input_extensions

This variable specifies the extensions tried by latexmk when it finds that a LaTeX run resulted in an error that a file has not been found, and the file is given without an extension. This typically happens when LaTeX commands of the form \input{file} or \includegraphics{figure}, when the relevant source file does not exist.

In this situation, latexmk searches for custom dependencies to make the missing file(s), but restricts it to the extensions specified by the variable %latex_input_extensions. The default extensions are 'tex' and 'eps'.

(For Perl experts: %latex_input_extensions is a hash whose keys are the extensions. The values are irrelevant.) Two subroutines are provided for manipulating this and the related variable %pdflatex_input_extensions, add_input_ext and remove_input_ext. They are used as in the following examples are possible lines in an initialization file:

    remove_input_ext('latex', 'tex');

removes the extension 'tex' from latex_input_extensions

    add_input_ext('latex', 'asdf');

add the extension 'asdf to latex_input_extensions. (Naturally with such an extension, you should have made an appropriate custom dependency for latexmk, and should also have done the appropriate programming in the LaTeX source file to enable the file to be read. The standard extensions are handled by LaTeX and its graphics/graphicx packages.

$latex_silent_switch ["-interaction=batchmode"]
Switch(es) for the LaTeX processing program when silent mode is on.

If you use MikTeX, you may prefer the results if you configure the options to include -c-style-errors, e.g., by the following line in an initialization file

```
$latex_silent_switch = "-interaction=batchmode -c-style-errors";
```

$lpr ["lpr %O %S" under UNIX/LINUX, "NONE lpr" under MS-WINDOWS]
The command to print postscript files.

Under MS-Windows (unlike UNIX/LINUX), there is no standard program for printing files. But there are ways you can do it. For example, if you have gsview installed, you could use it with the option "/p":

```
$lpr = '"c:/Program Files/Ghostgum/gsview/gsview32.exe" /p';
```

If gsview is installed in a different directory, you will need to make the appropriate change. Note the combination of single and double quotes around the name. The single quotes specify that this is a string to be assigned to the configuration variable *$lpr*. The double quotes are part of the string passed to the operating system to get the command obeyed; this is necessary because one part of the command name ("Program Files") contains a space which would otherwise be misinterpreted.

$lpr_dvi ["NONE lpr_dvi"]
The printing program to print dvi files.

$lpr_pdf ["NONE lpr_pdf"]
The printing program to print pdf files.

Under MS-Windows you could set this to use gsview, if it is installed, e.g.,

```
$lpr = '"c:/Program Files/Ghostgum/gsview/gsview32.exe" /p';
```

If gsview is installed in a different directory, you will need to make the appropriate change. Note the double quotes around the name: this is necessary because one part of the command name ("Program Files") contains a space which would otherwise be misinterpreted.

$make ["make"]
The make processing program.

$makeindex ["makeindex %O -o %D %S"]
The index processing program.

$makeindex_silent_switch ["-q"]
Switch(es) for the index processing program when silent mode is on.

$max_repeat [5]
The maximum number of times *latexmk* will run latex/pdflatex before deciding that there may be an infinite loop and that it needs to bail out, rather than rerunning latex/pdflatex again to resolve cross-references, etc. The default value covers all normal cases.

(Note that the "etc" covers a lot of cases where one run of latex/pdflatex generates files to be read in on a later run.)

$MSWin_back_slash [1]

This configuration variable only has an effect when *latexmk* is running under MS-Windows. It determines whether, when a command is executed under MS-Windows, there should be substituted "\" for the separator character between components of a directory name. Internally, *latexmk* uses "/" for the directory separator character, which is the character used by Unix-like systems.

For many programs under MS-Windows, both "\" and "/" are acceptable as the directory separator character. But some programs only accept "\". So for safety *latexmk* makes a translation, by default. It is conceivable that under certain situations this is undesirable, so the configuration can be changed. (A possible example might be when some of the software is implemented using Cygwin, which provides an Unix-like environment inside MS-Windows.)

$new_viewer_always [0]

This variable applies to *latexmk* **only** in continuous-preview mode. If *$new_viewer_always* is 0, *latexmk* will check for a previously running previewer on the same file, and if one is running will not start a new one. If *$new_viewer_always* is non-zero, this check will be skipped, and *latexmk* will behave as if no viewer is running.

$out_dir [""]

The directory in which output files are to be written by a run of (pdf)latex. See also the variable *$aux_dir*.

The effect of this variable (when non-blank) is achieved by using the **-output-directory** option of (pdf)latex. This exists in the usual current (Dec. 2011) implementations of TeX, i.e., MiKTeX and TeXLive. But it may not be present in other versions.

Commonly, the directory specified for output files is a subdirectory of the current working directory. However, if you specify some other directory, e.g., "/tmp/foo" or "../output", be aware that this could cause problems, e.g., with *makeindex* or *bibtex*. This is because modern versions of these programs, by default, will refuse to work when they find that they are asked to write to a file in a directory that appears not to be the current working directory or one of its subdirectories. This is part of security measures by the whole TeX system that try to prevent malicious or errant TeX documents from incorrectly messing with a user's files. If for *$out_dir* or *$aux_dir* you really do need to specify an absolute pathname (e.g., "/tmp/foo") or a path (e.g., "../output") that includes a higher-level directory, then you need to disable the security measures (and assume any risks). This can be done by temporarily setting the operating system's environment variable openout_any to "a" (as in "all"), to override the default "paranoid" setting.

$pdf_mode [0]

If zero, do NOT generate a pdf version of the document. If equal to 1, generate a pdf version of the document using pdflatex. If equal to 2, generate a pdf version of the document from the ps file, by using the command specified by the *$ps2pdf* variable. If equal to 3, generate a pdf version of the document from the dvi file, by using the command specified by the *$dvipdf* variable.

Equivalent to the **-pdf-**, **-pdf**, **-pdfdvi**, **-pdfps** options.

$pdflatex ["pdflatex %O %S"]

The LaTeX processing program in a version that makes a pdf file instead of a dvi file.

An example of the use of this variable is to arrange for *lualatex*, *xelatex* or some similar program to be used instead of *pdflatex*. Note that *lualatex* and *xelatex* only produce .pdf files (and not .dvi), so to use them you will also need to turn on production of .pdf files, and to turn off the production of .dvi (and .ps) files, either by command line options or by suitable settings in a configuration file.

Thus to use *lualatex*, the following settings are appropriate:

```
$pdflatex = "lualatex %O %S";
$pdf_mode = 1; $postscript_mode = $dvi_mode = 0;
```

To use *xelatex*, the corresponding settings are:

```
$pdflatex = "xelatex %O %S";
$pdf_mode = 1; $postscript_mode = $dvi_mode = 0;
```

Another use of the same variable is to add certain options to the command line for the program, e.g.,

```
$pdflatex = "pdflatex --shell-escape %O %S";
```

%pdflatex_input_extensions

This variable specifies the extensions tried by latexmk when it finds that a pdflatex run resulted in an error that a file has not been found, and the file is given without an extension. This typically happens when LaTeX commands of the form \input{file} or \includegraphics{figure}, when the relevant source file does not exist.

In this situation, latexmk searches for custom dependencies to make the missing file(s), but restricts it to the extensions specified by the variable %pdflatex_input_extensions. The default extensions are 'tex', 'pdf', 'jpg, and 'png'.

(For Perl experts: %pdflatex_input_extensions is a hash whose keys are the extensions. The values are irrelevant.) Two subroutines are provided for manipulating this and the related variable %latex_input_extensions, add_input_ext and remove_input_ext. They are used as in the following examples are possible lines in an initialization file:

```
remove_input_ext( 'pdflatex', 'tex' );
```

removes the extension 'tex' from pdflatex_input_extensions

```
add_input_ext( 'pdflatex', 'asdf' );
```

add the extension 'asdf to pdflatex_input_extensions. (Naturally with such an extension, you should have made an appropriate custom dependency for latexmk, and should also have done the appropriate programming in the LaTeX source file to enable the file to be read. The standard extensions are handled by pdflatex and its graphics/graphicx packages.)

$pdflatex_silent_switch ["-interaction=batchmode"]

Switch(es) for the pdflatex program (specified in the variable *$pdflatex* when silent mode is on.

If you use MikTeX, you may prefer the results if you configure the options to include -c-style-errors, e.g., by the following line in an initialization file

```
$latex_silent_switch = "-interaction=batchmode -c-style-errors";
```

$pdf_previewer ["start acroread %O %S"]

The command to invoke a pdf-previewer.

On MS-WINDOWS, the default is changed to "cmd /c start """; under more recent versions of Windows, this will cause to be run whatever command the system has associated with .pdf files. But this may be undesirable if this association is to *acroread* -- see the notes in the explanation of the **-pvc** option.]

On OS-X the default is changed to "open %S", which results in OS-X starting up (and detaching) the viewer associated with the file. By default, for pdf files this association is to OS-X's preview, which is quite satisfactory.

WARNING: Problem under MS-Windows: if acroread is used as the pdf previewer, and it is actually viewing a pdf file, the pdf file cannot be updated. Thus makes acroread a bad choice of previewer if you use *latexmk*'s previous-continuous mode (option **-pvc**) under MS-windows. This problem does not occur if, for example, SumatraPDF or gsview is used to view pdf files.

Important note: Normally you will want to have a previewer run detached, so that *latexmk* doesn't wait for the previewer to terminate before continuing its work. So normally you should prefix the command by "start ", which flags to *latexmk* that it should do the detaching of the previewer itself (by whatever method is appropriate to the operating system). But sometimes letting *latexmk* do the detaching is not appropriate (for a variety of non-trivial reasons), so you should put the "start " bit in yourself, whenever it is needed.

$pdf_update_command [""]

When the pdf previewer is set to be updated by running a command, this is the command that is run. See the information for the variable *$pdf_update_method*.

$pdf_update_method [1 under UNIX, 3 under MS-Windows]

How the pdf viewer updates its display when the pdf file has changed. See the information on the variable *$dvi_update_method* for the codes. (Note that information needs be changed slightly so that for the value 4, to run a command to do the update, the command is specified by the variable *$pdf_update_command*, and for the value 2, to specify update by signal, the signal is specified by *$pdf_update_signal*.)

Note that acroread under MS-Windows (but not UNIX) locks the pdf file, so the default value is then 3.

Arranging to use a command to get a previewer explicitly updated requires three variables to be set. For example:

```
$pdf_previewer = "start xpdf -remote %R %O %S";
$pdf_update_method = 4;
$pdf_update_command = "xpdf -remote %R -reload";
```

The first setting arranges for the xpdf program to be used in its "remote server mode", with the server name specified as the rootname of the TeX file. The second setting arranges for updating to be done in response to a command, and the third setting sets the update command.

$pdf_update_signal [Under UNIX: SIGHUP, which is a system-dependent value]

The number of the signal that is sent to the pdf viewer when it is updated by sending a signal -- see the information on the variable *$pdf_update_method*. The default value is the one appropriate for gv on a UNIX system.

$pid_position[1 under UNIX, -1 under MS-Windows]

The variable *$pid_position* is used to specify which word in lines of the output from *$pscmd* corresponds to the process ID. The first word in the line is numbered 0. The default value of 1 (2nd word in line) is correct for Solaris 2.6 and Linux. Setting the variable to -1 is used to indicate that

$pscmd is not to be used.

$postscript_mode [0]
If nonzero, generate a postscript version of the document. Equivalent to the **-ps** option.

If some other request is made for which a postscript file is needed, then $postscript_mode will be set to 1.

$preview_continuous_mode [0]
If nonzero, run a previewer to view the document, and continue running *latexmk* to keep .dvi up-to-date. Equivalent to the **-pvc** option. Which previewer is run depends on the other settings, see the command line options **-view=**, and the variable *$view*.

$preview_mode [0]
If nonzero, run a previewer to preview the document. Equivalent to the **-pv** option. Which previewer is run depends on the other settings, see the command line options **-view=**, and the variable *$view*.

$printout_mode [0]
If nonzero, print the document using *lpr*. Equivalent to the **-p** option. This is recommended **not** to be set from an RC file, otherwise you could waste lots of paper.

$print_type = ["auto"]
Type of file to printout: possibilities are "auto", "dvi", "none", "pdf", or "ps". See the option **-print=** for the meaning of the "auto" value.

$pscmd
Command used to get all the processes currently run by the user. The -pvc option uses the command specified by the variable *$pscmd* to determine if there is an already running previewer, and to find the process ID (needed if *latexmk* needs to signal the previewer about file changes).

Each line of the output of this command is assumed to correspond to one process. See the *$pid_position* variable for how the process number is determined.

The default for *pscmd* is "NONE" under MS-Windows and cygwin (i.e., the command is not used), "ps --width 200 -f -u $ENV{USER}" under linux, "ps -ww -u $ENV{USER}" under darwin (Macintosh OS-X), and "ps -f -u $ENV{USER}" under other operating systems (including other flavors of UNIX). In these specifications "$ENV{USER}" is substituted by the username.

$ps2pdf ["ps2pdf %O %S %D"]
Command to convert ps to pdf file.

$ps_filter [empty]
The postscript file filter to be run on the newly produced postscript file before other processing. Equivalent to specifying the **-pF** option.

$ps_previewer ["start gv %O %S", but start %O %S under MS-WINDOWS]
The command to invoke a ps-previewer. (The default under MS-WINDOWS will cause to be run whatever command the system has associated with .ps files.)

Note that gv could be used with the -watch option updates its display whenever the postscript file changes, whereas ghostview does not. However, different versions of gv have slightly different ways of writing this option. You can configure this variable appropriately.

WARNING: Linux systems may have installed one (or more) versions of gv under different names, e.g., ggv, kghostview, etc, but perhaps not one called gv.

Important note: Normally you will want to have a previewer run detached, so that *latexmk* doesn't wait for the previewer to terminate before continuing its work. So normally you should prefix the command by "start ", which flags to *latexmk* that it should do the detaching of the previewer itself (by whatever method is appropriate to the operating system). But sometimes letting *latexmk* do the detaching is not appropriate (for a variety of non-trivial reasons), so you should put the "start " bit in yourself, whenever it is needed.

$ps_previewer_landscape ["start gv -swap %O %S", but start %O %S under MS-WINDOWS]
The command to invoke a ps-previewer in landscape mode.

$ps_update_command [""]
When the postscript previewer is set to be updated by running a command, this is the command that is run. See the information for the variable *$ps_update_method*.

$ps_update_method [0 under UNIX, 1 under MS-Windows]
How the postscript viewer updates its display when the ps file has changed. See the information on the variable *$dvi_update_method* for the codes. (Note that information needs be changed slightly so that for the value 4, to run a command to do the update, the command is specified by the variable *$ps_update_command*, and for the value 2, to specify update by signal, the signal is specified by *$ps_update_signal*.)

$ps_update_signal [Under UNIX: SIGHUP, which is a system-dependent value]
The number of the signal that is sent to the pdf viewer when it is updated by sending a signal -- see *$ps_update_method*. The default value is the one appropriate for gv on a UNIX system.

$pvc_view_file_via_temporary [1]
The same as *$always_view_file_via_temporary*, except that it only applies in preview-continuous mode (-pvc option).

$quote_filenames [1]
This specifies whether substitutions for placeholders in command specifications (as in *$pdflatex*) are surrounded by double quotes. If this variable is 1 (or any other value Perl regards as true), then quoting is done. Otherwise quoting is omitted.

The quoting method used by latexmk is tested to work correctly under UNIX systems (including Linux and Mac OS-X) and under MS-Windows. It allows the use of filenames containing special characters, notably spaces. (But note that many versions of LaTeX and PdfLaTeX cannot correctly deal with TeX files whose names contain spaces. Latexmk's quoting only ensures that such filenames are correctly treated by the operating system in passing arguments to programs.)

$recorder [0]
Whether to use the **-recorder** option to *latex* and *pdflatex*. Use of this option results in a file of extension *.fls* containing a list of the files that these programs have read and written. *Latexmk* will then use this file to improve its detection of source files and generated files after a run of *latex* or *pdflatex*.

It is generally recommended to use this option (or to configure the *$recorder* variable to be on.) But it only works if *(pdf)latex* supports the -recorder option, which is true for most current implementations

Note about the name of the .fls file: Most implementations of *(pdf)latex* produce an .fls file with the same basename as the main document's LaTeX, e.g., for Document.tex, the .fls file is Document.fls. However, some implementations instead produce files named for the program, i.e., latex.fls or pdflatex.fls. In this second case, *latexmk* copies the latex.fls or pdflatex.fls to a file with the basename of the main LaTeX document, e.g., Document.fls.

$search_path_separator [See below for default]

The character separating paths in the environment variables TEXINPUTS, BIBINPUTS, and BSTINPUTS. This variable is mainly used by *latexmk* when the **-outdir**, **-output-directory**, **-auxdir**, and/or **-aux-directory** options are used. In that case *latexmk* needs to communicate appropriately modified search paths to $*bibtex*, *dvipdf*, *dvips*, and *(pdf)latex*.

[Comment to technically savvy readers: *(pdf)latex* doesn't actually need the modified search path, because it corrects it internally. But, surprisingly, *dvipdf* and *dvips* do, because sometimes graphics files get generated in the output or aux directories.]

The default under MSWin and Cygwin is ';' and under UNIX-like operating systems (including Linux and OS-X) is ':'. Normally the defaults give correct behavior. But there can be difficulties if your operating system is of one kind, but some of your software is running under an emulator for the other kind of operating system; in that case you'll need to find out what is needed, and set *$search_path_separator* explicitly. (The same goes, of course, for unusual operating systems that are not in the MSWin, Linux, OS-X, Unix collection.)

$sleep_time [2]

The time to sleep (in seconds) between checking for source file changes when running with the **-pvc** option. This is subject to a minimum of one second delay, except that zero delay is also allowed.

A value of exactly 0 gives no delay, and typically results in 100% CPU usage, which may not be desirable.

$texfile_search [""]

This is an obsolete variable, replaced by the @*default_files* variable.

For backward compatibility, if you choose to set *$texfile_search*, it is a string of space-separated filenames, and then *latexmk* replaces @*default_files* with the filenames in *$texfile_search* to which is added "*.tex".

$success_cmd [undefined]

See the documentation for $compiling_cmd.

$tmpdir [See below for default]

Directory to store temporary files that *latexmk* may generate while running.

The default under MSWindows (including cygwin), is to set *$tmpdir* to the value of the first of whichever of the system environment variables TMPDIR or TEMP exists, otherwise to the current directory. Under other operating systems (expected to be UNIX/Linux, including OS-X), the default is the value of the system environment variable TMPDIR if it exists, otherwise "/tmp".

$use_make_for_missing_files [0]

Whether to use *make* to try and make files that are missing after a run of *latex* or *pdflatex*, and for which a custom dependency has not been found. This is generally useful only when *latexmk* is used as part of a bigger project which is built by using the *make* program.

Note that once a missing file has been made, no further calls to *make* will be made on a subsequent run of *latexmk* to update the file. Handling this problem is the job of a suitably defined Makefile. See the section "USING *latexmk* WITH *make*" for how to do this. The intent of calling *make* from *latexmk* is merely to detect dependencies.

$view ["default"]
Which kind of file is to be previewed if a previewer is used. The possible values are "default", "dvi", "ps", "pdf". The value of "default" means that the "highest" of the kinds of file generated is to be used (among dvi, ps and pdf).

CUSTOM DEPENDENCIES

In any RC file a set of custom dependencies can be set up to convert a file with one extension to a file with another. An example use of this would be to allow *latexmk* to convert a *.fig* file to *.eps* to be included in the *.tex* file.

The old method of configuring *latexmk* was to directly manipulate the **@cus_dep_list** array that contains information defining the custom dependencies. This method still works. But now there are subroutines that allow convenient manipulations of the custom dependency list. These are

```
add_cus_dep( fromextension, toextension, must, subroutine )
remove_cus_dep( fromextension, toextension )
show_cus_dep()
```

The custom dependency is a list of rules, each of which is specified as follow:

from extension:
The extension of the file we are converting from (e.g. "fig"). It is specified without a period.

to extension:
The extension of the file we are converting to (e.g. "eps"). It is specified without a period.

must: If non-zero, the file from which we are converting **must** exist, if it doesn't exist *latexmk* will give an error message and exit unless the **-f** option is specified. If *must* is zero and the file we are converting from doesn't exist, then no action is taken.

function:
The name of the subroutine that *latexmk* should call to perform the file conversion. The first argument to the subroutine is the base name of the file to be converted without any extension. The subroutines are declared in the syntax of *Perl*. The function should return 0 if it was successful and a nonzero number if it failed.

It is invoked whenever *latexmk* detects that a run of latex/pdflatex needs to read a file, like a graphics file, whose extension is the to-extension of a custom dependency. Then *latexmk* examines whether a file exists with the same name, but with the corresponding from-extension, as specified in the custom-dependency rule. If it does, then whenever the destination file (the one with the to-extension) is out-of-date with respect to the corresponding source file.

To make the new destination file, the *Perl* subroutine specified in the rule is invoked, with an argument that is the base name of the files in question. Simple cases just involve a subroutine invoking an external program; this can be done by following the templates below, even by those without knowledge of the *Perl* programming language. Of course, experts could do something much more elaborate.

One other item in each custom-dependency rule labeled "must" above specifies how the rule should be applied when the source file fails to exist.

A simple and typical example of code in an initialization rcfile is

```
add_cus_dep( 'fig', 'eps', 0, 'fig2eps' );
sub fig2eps {
  system( "fig2dev -Leps \"$_[0].fig\" \"$_[0].eps\"" );
}
```

The first line adds a custom dependency that converts a file with extension "fig", as created by the xfig program, to an encapsulated postscript file, with extension "eps". The remaining lines define a subroutine that carries out the conversion. If a rule for converting "fig" to "eps" files already exists (e.g., from a previously read-in initialization file), the *latexmk* will delete this rule before making the new one.

Suppose *latexmk* is using this rule to convert a file "figure.fig" to "figure.eps". Then it will invoke the fig2eps subroutine defined in the above code with a single argument "figure", which is the basename of each of the files (possibly with a path component). This argument is referred to by *Perl* as $_[0]. In the example above, the subroutine uses the *Perl* command system to invoke the program fig2dev. The double quotes around the string are a *Perl* idiom that signify that each string of the form of a variable name, $_[0] in this case, is to be substituted by its value.

If the return value of the subroutine is non-zero, then *latexmk* will assume an error occurred during the execution of the subroutine. In the above example, no explicit return value is given, and instead the return value is the value returned by the last (and only) statement, i.e., the invocation of system, which returns the value 0 on success.

If you use pdflatex instead of latex, then you will probably prefer to convert your graphics files to pdf format, in which case you would replace the above code in an initialization file by

```
add_cus_dep( 'fig', 'pdf, 0, 'fig2pdf' );
sub fig2pdf {
   system( "fig2dev -Lpdf \"$_[0].fig\" \"$_[0].pdf\"" );
}
```

Note 1: In the command lines given in the system commands in the above examples, double quotes have been inserted around the file names (implemented by '\"' in the Perl language). They immunize the running of the program against special characters in filenames. Very often these quotes are not necessary, i.e., they can be omitted. But it is normally safer to keep them in. Even though the rules for quoting vary between operating systems, command shells and individual pieces of software, the quotes in the above examples do not cause problems in the cases I have tested.

Note 2: One case in which the quotes are important is when the files are in a subdirectory and your operating system is Microsoft Windows. Then the separator character for directory components can be either a forward slash '/' or Microsoft's more usual backward slash '\'. Forward slashes are generated by latexmk, to maintain its sanity from software like MiKTeX that mixes both directory separators; but their correct use normally requires quoted filenames. (See a log file from a run of MiKTeX (at least in v. 2.9) for an example of the use of both directory separators.)

If you have some general custom dependencies defined in the system or user initialization file, you may find that for a particular project they are undesirable. So you might want to delete the unneeded ones. For example, you remove any "fig" to "eps" rule by the line

```
remove_cus_dep( 'fig', 'eps' );
```

If you have complicated sets of custom dependencies, you may want to get a listing of the custom dependencies. This is done by using the line

```
show_cus_dep();
```

in an initialization file.

Another example of a custom dependency overcomes a limitation of *latexmk* concerning index files. The only index-file conversion built-in to *latexmk* is from an ".idx" file written on one run of latex/pdflatex to an

".ind" file to be read in on a subsequent run. But with the index.sty package you can create extra indexes with extensions that you configure. *Latexmk* does not know how to deduce the extensions from the information it has. But you can easily write a custom dependency. For example if your latex file uses the command "\newindex{special}{ndx}{nnd}{Special index}" you will need to convert files with the extension *.ndx* to *.nnd*. The following lines in an initialization RC file will cause this to happen:

```
add_cus_dep('ndx', 'nnd', 0, 'makendx2nnd');
sub makendx2nnd {
   system( "makeindex -o \"$_[0].nnd\" \"$_[0].ndx\"" );
}
```

(You will need to modify this code if you use filenames with spaces in them, to provide correct quoting of the filenames.)

Those of you with experience with Makefiles, will undoubtedly be concerned that the *.ndx* file is written during a run of latex/pdflatex and is always later than the *.nnd* last read in. Thus the *.nnd* appears to be perpetually out-of-date. This situation, of circular dependencies, is endemic to latex, and *latexmk* in its current version works correctly with circular dependencies. It examines the contents of the files (by use of an md5 checksum), and only does a remake when the file contents have actually changed.

Of course if you choose to write random data to the *.nnd* (or and *.aux* file, etc) that changes on each new run, then you will have a problem. For real experts: See the *%hash_cal_ignore_pattern* if you have to deal with such problems.

Glossaries can be dealt with similarly.

OLD METHOD OF DEFINING CUSTOM DEPENDENCIES

In previous versions of *latexmk*, the only method of defining custom dependencies was to directly manipulate the table of custom dependencies. This is contained in the **@cus_dep_list** array. It is an array of strings, and each string in the array has four items in it, each separated by a space, the from-extension, the to-extension, the "must" item, and the name of the subroutine for the custom dependency. These were all defined above.

An example of the old method of defining custom dependencies is as follows. It is the code in an RC file to ensure automatic conversion of *.fig* files to *.eps* files:

```
push @cus_dep_list, "fig eps 0 fig2eps";
sub fig2eps {
   system( "fig2dev -Lps \"$_[0].fig\" \"$_[0].eps\"" );
}
```

This method still works, and is equivalent to the earlier code using the add_cus_dep subroutine, except that it doesn't delete any previous custom-dependency for the same conversion. So the new method is preferable.

USING *latexmk* WITH *make*

This section is targeted only at advanced users who use the *make* program for complex projects, as for software development, with the dependencies specified by a Makefile.

Now the basic task of *latexmk* is to run the appropriate programs to make a viewable version of a LaTeX document. However, the usual *make* program is not suited to this purpose for at least two reasons. First is that the use of LaTeX involves circular dependencies (e.g., via .aux files), and these cannot be handled by the standard *make* program. Second is that in a large document the set of source files can change quite frequently, particularly with included graphics files; in this situation keeping a Makefile manually updated is inappropriate and error-prone, especially when the dependencies can be determined automatically. *Latexmk* solves both of these problems robustly.

Thus for many standard LaTeX documents *latexmk* can be used by itself without the *make* program. In a complex project it simply needs to be suitably configured. A standard configuration would be to define custom dependencies to make graphics files from their source files (e.g., as created by the *xfig* program). Custom dependencies are *latexmk*'s equivalent of pattern rules in Makefiles.

Nevertheless there are projects for which a Makefile is appropriate, and it is useful to know how to use *latexmk* from a Makefile. A typical example would be to generate documentation for a software project. Potentially the interaction with the rest of the rules in the Makefile could be quite complicated, for example if some of the source files for a LaTeX document are generated by the project's software.

In this section, I give a couple of examples of how *latexmk* can be usefully invoked from a Makefile. The examples use specific features of current versions of GNU *make*, which is the default on both linux and OS-X systems. They may need modifications for other versions of *make*.

The simplest method is simply to delegate all the relevant tasks to *latexmk*, as is suitable for a straightforward LaTeX document. For this a suitable Makefile is like

```
.PHONY : FORCE_MAKE
all : try.pdf
%.pdf : %.tex FORCE_MAKE
	latexmk -pdf -dvi- -ps- $<
```

(Note: the last line must be introduced by a tab for the Makefile to function correctly!) Naturally, if making try.pdf from its associated LaTeX file try.tex were the only task to be performed, a direct use of *latexmk* without a Makefile would normally be better. The benefit of using a Makefile for a LaTeX document would be in a larger project, where lines such as the above would be only be a small part of a larger Makefile.

The above example has a pattern rule for making a .pdf file from a .tex file, and it is defined to use latexmk in the obvious way. There is a conventional default target named "all", with a prerequisite of try.pdf. So when *make* is invoked, by default it makes try.pdf. The only complication is that there may be many source files beyond try.tex, but these aren't specified in the Makefile, so changes in them will not by themselves cause *latexmk* to be invoked. Instead, the pattern rule is equipped with a "phony" prerequisite FORCE_MAKE; this has the effect of causing the rule to be always out-of-date, so that *latexmk* is always run. It is *latexmk* that decides whether any action is needed, e.g., a rerun of *pdflatex*. Effectively the Makefile delegates all decisions to *latexmk*, while *make* has no knowledge of the list of source files except for primary LaTeX file for the document. If there are, for example, graphics files to be made, these must be made by custom dependencies configured in *latexmk*.

But something better is needed in more complicated situations, for example, when the making of graphics files needs to be specified by rules in the Makefile. To do this, one can use a Makefile like the following:

```
TARGETS = document1.pdf document2.pdf
DEPS_DIR = .deps
LATEXMK = latexmk -recorder -use-make -deps \
    -e 'warn qq(In Makefile, turn off custom dependencies\n);' \
    -e '@cus_dep_list = ();' \
    -e 'show_cus_dep();'
all : $(TARGETS)
$(foreach file,$(TARGETS),$(eval -include $(DEPS_DIR)/$(file)P))
```

```
$(DEPS_DIR) :
	mkdir $@
%.pdf : %.tex
	if [ ! -e $(DEPS_DIR) ]; then mkdir $(DEPS_DIR); fi
	$(LATEXMK) -pdf -dvi- -ps- -deps-out=$(DEPS_DIR)/$@P $<
%.pdf : %.fig
	fig2dev -Lpdf $< $@
```

(Again, the lines containing the commands for the rules should be started with tabs.) This example was inspired by how GNU *automake* handles automatic dependency tracking of C source files.

After each run of latexmk, dependency information is put in a file in the .deps subdirectory. The Makefile causes these dependency files to be read by *make*, which now has the full dependency information for each target .pdf file. To make things less trivial it is specificed that two files document1.pdf and document2.pdf are the targets. The dependency files are .deps/document1.pdfP and .deps/document2.pdfP.

There is now no need for the phony prerequisite for the rule to make .pdf files from .tex files. But I have added a rule to make .pdf files from .fig files produced by the *xfig* program; these are commonly used for graphics insertions in LaTeX documents. *Latexmk* is arranged to output a dependency file after each run. It is given the **-recorder** option, which improves its detection of files generated during a run of *pdflatex*; such files should not be in the dependency list. The **-e** options are used to turn off all custom dependencies, and to document this. Instead the **-use-make** is used to delegate the making of missing files to *make* itself.

Suppose in the LaTeX file there is a command \includegraphics{graph}, and an *xfig* file "graph.fig" exists. On a first run, *pdflatex* reports a missing file, named "graph". *Latexmk* succeeds in making "graph.pdf" by calling "make graph.pdf", and after completion of its work, it lists "fig.pdf" among the dependents of the file *latexmk* is making. Then let "fig.fig" be updated, and then let *make* be run. *Make* first remakes "fig.pdf", and only then reruns *latexmk*.

Thus we now have a method by which all the subsidiary processing is delegated to *make*.

SEE ALSO

latex(1), bibtex(1).

BUGS

Sometimes a viewer (gv) tries to read an updated .ps or .pdf file after its creation is started but before the file is complete. Work around: manually refresh (or reopen) display. Or use one of the other previewers and update methods.

(The following isn't really a bug, but concerns features of previewers.) Preview continuous mode only works perfectly with certain previewers: Xdvi on UNIX/LINUX works for dvi files. Gv on UNIX/LINUX works for both postscript and pdf. Ghostview on UNIX/LINUX needs a manual update (reopen); it views postscript and pdf. Gsview under MS-Windows works for both postscript and pdf, but only reads the updated file when its screen is refreshed. Acroread under UNIX/LINUX views pdf, but the file needs to be closed and reopened to view an updated version. Under MS-Windows, acroread locks its input file and so the pdf file cannot be updated. (Remedy: configure *latexmk* to use *sumatrapdf* instead.)

THANKS TO

Authors of previous versions. Many users with their feedback, and especially David Coppit (username david at node coppit.org) who made many useful suggestions that contributed to version 3, and Herbert Schulz. (Please note that the e-mail addresses are not written in their standard form to avoid being harvested by worms and viruses.)

AUTHOR

Current version, by John Collins (username collins at node phys.psu.edu). (Version 4.39).

Released version can be obtained from CTAN: <http://www.ctan.org/tex-archive/support/latexmk/>, and from the author's website <http://www.phys.psu.edu/~collins/software/latexmk/>.
Modifications and enhancements by Evan McLean (Version 2.0)
Original script called "go" by David J. Musliner (RCS Version 3.2)