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, which copies the behavior of earlier versions when only *latex* was available.

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*). 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. With 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 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/or 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 (e.g., with 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*, but **are currently non-functional**. More flexibility can be obtained in current versions, since the command strings for running **latex* can now be configured to run multiple commands. This also extends the possibility of postprocessing generated 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, macOS, and other UNIX systems. See the section "Configuration/Initialization (rc) Files", and then the later sections "How to Set Variables in Initialization Files", "Format of Command Specifications", "List of Configuration Variables Usable in Initialization Files", "Custom Dependencies", and "Advanced Configuration"

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. *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 the documentation, '**latex*' means any of the supported engines, i.e., currently *latex*, *lualatex*, *pdflatex*, *xelatex*. Mention of a specific one of these normally refers that specific engines. Earlier versions of this documentation weren't so consistent. Which of these is used to compile a document, depends on the configuration and the command line arguments.

Note 2: **In addition to the options in the list below,** *latexmk* recognizes almost all the options recognized by the **latex* programs in their current TeXLive and MiKTeX implementations. Some of the options for these programs trigger special action or behavior by *latexmk*, in which case they have specific explanations in this document; in this case they may or may not be passed to *latex as well.

Run *latexmk* with the **-showextraoptions** to get a list of these other options that *latexmk* accepts but that are simply passed through to **latex*. For details, see the explanation of the **-showextraoptions** option.

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".

There are certain restrictions on what characters can be in a filename; certain characters are either prohibited or problematic for the *latex* etc programs. These characters are: "$", "%", "\", "~", the double quote character, and the control characters null, tab, form feed, carriage return, line feed, and delete. In addition "&" is prohibited when it is the first character of a filename.

Latexmk gives a fatal error when it detects any of the above characters in the TeX filename(s) specified on the command line. *However* before testing for illegal characters, *latexmk* removes matching pairs of double quotes from a filename. This matches the behavior of *latex* etc, and deals with problems that occasionally result from filenames that have been incorrectly quoted on the command line.

In addition, under Microsoft Windows, the forward slash character "\" is a directory separator, so *latexmk* replaces it by a forward slash "/", which is also a legal directory separator in Windows, and is accepted by *latex* etc.

-auxdir=FOO or **-aux-directory=FOO**

Sets the directory for auxiliary output files of *latex (.aux, .log etc). These are all the generated files, with the exception of final output files (.dvi, .ps, .pdf, .synctex.gz, .synctex). See the **-outdir**/**-output-directory** option for directories for the main output files, and the **-out2dir** option for the final output files.

If the directory specified for the **-aux**/**-aux-directory** option is blank, then the default is used, which is to be the same as the output directory.

If you also use the **-cd** option, and the specified auxiliary output directory is a relative path, then the path is interpreted relative to the document directory.

See the section AUXILIARY AND OUTPUT DIRECTORIES for more details.

-bi <intensity>

How dark to make the banner message (poscript file only). A decimal number between 0 and 1. 0 is black and 1 is white. The default is 0.95.

See the option **-d** for more details.

-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. Also, always treat .bbl files as precious, i.e., do not delete them in a cleanup operation.

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.

This property can also be configured by setting the *$bibtex_use* variable to 0 in a configuration file.

-bibtex-cond

When the source file uses a bbl file for the bibliography and *bibtex* is used to generate the bibliography, run *bibtex* as needed to regenerate the bbl files only if the relevant bib file(s) exist. Thus when the bib file(s) are not available, *bibtex* is not run, thereby avoiding overwriting of the bbl file. Also, always treat .bbl files as precious, i.e., do not delete them in a cleanup operation.

This is the default setting. It can also be configured by setting the *$bibtex_use* variable to 1 in a configuration file.

The reason for using this setting is first to allow automatic switching between the use and non-use of

bibtex depending on the existence or not of a bib file. In addition, when submitting articles to a scientific journal, it is common to submit only .tex and .bbl files (plus graphics files), but not a .bib file. Hence it is often useful to treat .bbl files as true source files, that should be preserved under a clean up operation.

This property can also be configured by setting the *$bibtex_use* variable to 1 in a configuration file.

Note that when **biber** is used, and a bib file doesn't exist, this option does not prevent *biber* from being run, with the bbl file then being incorrect. See the documentation on *$bibtex_use* for more details. However, a bbl file is treated as precious in a clean up operation.

-bibtex-cond1

The same as **-bibtex-cond** except that .bbl files are only treated as precious if one or more bibfiles fails to exist.

Thus if all the bib files exist, *bibtex* is run to generate .bbl files as needed, and then it is appropriate to delete the bbl files in a cleanup operation since they can be re-generated.

This property can also be configured by setting the *$bibtex_use* variable to 1.5 in a configuration file.

Note that when **biber** is used, and a bib file doesn't exist, this option does not prevent *biber* from being run, with the bbl file then being incorrect. See the documentation on *$bibtex_use* for more details. However, a bbl file is treated as precious in a clean up operation.

-bibtexfudge or **-bibfudge**

Turn on the change-directory fudge for bibtex. See documentation of *$bibtex_fudge* for details.

-bibtexfudge- or **-bibfudge-**

Turn off the change-directory fudge for bibtex. See documentation of *$bibtex_fudge* for details.

-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.

See the option **-d** for more details.

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

-bs <scale>

A decimal number that specifies how large the banner message will be produced in a postscript file. 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.

See the option **-d** for more details.

-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* and *@generated_exts* configuration variables are removed.

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

Treatment of .bbl files: If *$bibtex_use* is set to 0 or 1, bbl files are always treated as non-regeneratable. If *$bibtex_use* is set to 1.5, bbl files are counted as non-regeneratable conditionally: If the bib file exists, then bbl files are regeneratable, and are deleted in a clean up. But if *$bibtex_use* is 1.5 and a bib file doesn't exist, then the bbl files are treated as non-regeneratable and hence are not deleted.

In contrast, if *$bibtex_use* is set to 2, bbl files are always treated as regeneratable, and are deleted in a cleanup.

Treatment of files generated by custom dependencies: 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 followed by a make.

See the **-c** option for the specification of whether or not .bbl files are treated as non-regeneratable or 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.

-cc "Continuous compile" without previewer. Equivalent to **-pvc -view=none**.

-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.

This option works by setting the **$do_cd** configuration variable to one; you can set that variable if you want to configure *latexmk* to have the effect of the **-cd** option without specifying it on the command line. See the documentation for that variable.

-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* 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.

This option works by setting the **$do_cd** configuration variable to zero. See the documentation for that variable for more information.

-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. This isn't implemented for pdf files except if produced by ps2pdf. The size and intensity of the banner message 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** or **-pdfps** 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-escape=<string>
Set the kind of escaping used for spaces in the dependency list. The possible values are "none", "unix", "nmake", corresponding respectively to no escaping, escaping with a "\" suitable for standard Unix make, and escaping with "^", suitable for Microsoft's nmake.

-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.

-dir-report
For each .tex file processed, list the settings for aux and out directories, after they have been normalized from the settings specified during initialization. See the description of the variable **$aux_out_dir_report** for more details.

-dir-report-
Do not report the settings for aux and out directories. (Default)

-dir-report-only
After all initialization is complete, give the settings for the aux and out directories, and then halt. This option is primarily used for debugging configuration issues.

-dvi Generate dvi version of document using latex. (And turn off any incompatible requests.)

-dvilua
Generate dvi version of document using lualatex instead of latex. (And turn off any incompatible requests.)

-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.)

-dvilualatex="COMMAND"
This sets the string specifying the command to run *dvi lualatex*. It behaves like the **-pdflatex** option, but sets the variable *$dvilualatex*.

Note: This option when provided with the COMMAND argument only sets the command for invoking dvilualatex; it does not turn on the use of dvilualatex. That is done by other options or in an initialization file.

-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.

(*Note*: the above examples show are to show how to use the **-e** to specify initialization code to be executed. But the particular effect can be achieved also by the use of the **-latex** option with less problems in dealing with quoting.)

-emulate-aux-dir
Emulate the use of an aux directory instead of leaving it to the *latex programs to do it. (MiKTeX supports -aux-directory, but TeXLive doesn't.)

See the section AUXILIARY AND OUTPUT DIRECTORIES for more details.

-emulate-aux-dir-
Turn off emulation to implement an aux directory and leave it to the *latex program to handle the case that the aux directory is different from the output directory. Note that if you use TeXLive, which doesn't support -aux-directory, latexmk will automatically switch aux_dir emulation on after the first run of *latex, because it will find the .log file in the wrong place.

-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** (which *latexmk* passes to **latex*).

-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 Go mode: Force *latexmk* to run each rule at least once, even under situations whre *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 the configuration and wish to reprocess all files.

Equivalently: all rules are initially to be regarded as out of date. This option has the disadvantage that if, for example, there are very many custom dependencies for making graphics files, all of them will be run, which could be very time consuming. See also the **-gt** *option, which only forces a rerun of *latex.*

-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.

-gt Require at least one run of *latex. I.e., process as normal, except that at least one run of the *latex rule is to be made, even when *latexmk*'s normal methods lead it to conclude that a run of *latex is not needed.

Equivalently, treat the *latex rule as initially out-of-date. Note that *latexmk* may separately detect that other rules are out-of-date by its normal methods. If these are normally run before *latex, then this will still be done

This option is particularly useful when some changes in configuration have been made or under certain error conditions, where *latexmk* would not correctly detect the need to run *latex, but where the run is needed.

-h or **-help**
Print help information.

-hnt Generate hnt (HINT) version of document using hilatex. (And turn off any incompatible requests.)

-jobname=STRING
Set the basename of output files(s) to STRING, instead of the default, which is the basename of the specified TeX file. (At present, STRING should not contain spaces.)

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

There is one enhancement, that the STRING may contain the placeholder '%A'. This will be substituted by the basename of the TeX file. The primary purpose is when multiple files are specified on the command line to *latexmk*, and you wish to use a jobname with a different file-dependent value for each file. For example, suppose you had .tex files test1.tex and test2.tex, and you wished to compare the results of compilation by **latex* and those with *xelatex*. Then under a unix-type operating system you could use the command line

```
latexmk -pdf -jobname=%A-pdflatex *.tex
latexmk -pdfxe -jobname=%A-xelatex *.tex
```

Then the .aux, .log, and .pdf files from the use of *pdflatex* would have basenames test1-pdflatex and test2-pdflatex, while from *xelatex*, the basenames would be test1-xelatex and test2-xelatex.

Under MS-Windows with cmd.exe, you would need to double the percent sign, so that the percent character is passed to latexmk rather than being used to substitute an environment variable:

```
latexmk -pdf -jobname=%%A-pdflatex *.tex
latexmk -pdfxe -jobname=%%A-xelatex *.tex
```

-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 This sets the generation of dvi files by *latex*, and turns off the generation of pdf and ps files.

Note: to set the command *used when latex is specified, see the* **-latex="COMMAND"** *option.*

-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).

Note: This option when provided with the COMMAND argument only sets the command for invoking latex; it does not turn on the use of latex. That is done by other options or in an initialization file.

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

-logfilewarninglist
-logfilewarnings After a run of **latex*, give a list of warnings about undefined citations and references (unless silent mode is on).

See also the *$silence_logfile_warnings* configuration variable.

-logfilewarninglist-
-logfilewarnings- After a run of **latex*, do not give a list of warnings about undefined citations and references. (Default)

See also the *$silence_logfile_warnings* configuration variable.

-lualatex
Use *lualatex*. That is, use *lualatex* to process the source file(s) to pdf. The generation of dvi, hnt, postscript and xdv files is turned off.

This option is equivalent to using the following set of options

-pdflua -dvi- -ps-

(*Note*: Note that the method of implementation of this option, but not its intended effect, differ from some earlier versions of *latexmk*.)

-lualatex="COMMAND"
This sets the string specifying the command to run *lualatex*. It behaves like the **-pdflatex** option, but sets the variable *$lualatex*.

Note: This option when provided with the COMMAND argument only sets the command for invoking lualatex; it does not turn on the use of lualatex. That is done by other options or in an initialization file.

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

-makeindexfudge
Turn on the change-directory fudge for makeindex. See documentation of *$makeindex_fudge* for details.

-makeindexfudge-
Turn off the change-directory fudge for makeindex. See documentation of *$makeindex_fudge* for details.

$min_sleep_time [0.01]
This is the minimum nonzero value allowed for *$sleep_time*.

-MSWinBackSlash
This option only has an effect when *latexmk* is running under MS-Windows. This is that when *latexmk* runs a command under MS-Windows, the Windows standard directory separator "\" is used to separate directory components in a file name. Internally, *latexmk* uses "/" for the directory separator character, which is the character used by Unix-like systems.

This is the default behavior. However the default may have been overridden by a configuration file (latexmkrc file) which sets *$MSWin_back_slash=0*.

-MSWinBackSlash-
This option only has an effect when *latexmk* is running under MS-Windows. This is that when *latexmk* runs a command under MS-Windows, the substitution of "\" for the separator character between directory components of a file name is *not* done. Instead the forward slash "/" is used, the same as on Unix-like systems. This is acceptable in most situations under MS-Windows, provided that

filenames are properly quoted, as *latexmk* does by default.

See the documentation for the configuration variable *$MSWin_back_slash* for more details.

-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. Equivalent to the **-bibtex-** option.

-nobibtexfudge or **-nobibfudge**
Turn off the change-directory fudge for bibtex. See documentation of *$bibtex_fudge* for details.

-noemulate-aux-dir
Turn aux_dir emulation off. Same as *-emulate-aux-dir-*.

-nomakeindexfudge
Turn off the change-directory fudge for makeindex. See documentation of *$makeindex_fudge* for details.

-noparallel
(Experimental: subject to change in future versions.) The same as **-parallel-**: Disable parallel processing. (This is the default behavior.) This option sets the variable *$use_parallel* to 0.

See the description of the variable *$use_parallel* for details

-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. But **-norc** is an exception to this rule: it is acted on first, no matter where it occurs on the command line.

-outdir=FOO or **-output-directory=FOO**

Sets the directory for the output files of *latex.

If the aux directory is not set or is the same as the output directory, then all output files of *latex are sent to the output directory.

If the aux directory is set, e.g., by the option **-auxdir**, and is not equal to the output directory, then only the primary output files (.dvi, .ps, .pdf, .synctex, .synctex.gz) are sent to the output directory.

Other generated files are sent to the aux directory.

See the section AUXILIARY AND OUTPUT DIRECTORIES for more details.

-out2dir=FOO

(*Experimental new feature.*)

Sets the directory for the final output files of a whole round of compilations.

The use of this directory solves, among other things, the problem that when multiple runs of *latex and other programs are needed, files like the main pdf file from pdflatex, etc will be changed multiple times. A viewer like SumatraPDF that reloads the file whenever it detects changes will show a distracting sequence of intermediate states of the pdf file, rather than just the final version after all the repeated runs of *latex etc have been done. Instead, when a distinct final-output directory is set, by the **-out2dir** option or the equivalent *$out2_dir* variable is set, the viewer will only see a changed pdf (etc) file after full sequence of repeated runs of *latex etc has finished.

By default the final output directory is the same as the output directory (as specified by the **-outdir** option or the setting of the variable *$out_dir* configuration variable).

-output-format=FORMAT

This option is one that is allowed for *latex*, *lualatex*, and *pdflatex*. But it is not passed to these programs. Instead *latexmk* emulates it in a way suitable for the context of *latexmk* and its workflows.

-If FORMAT is dvi, then dvi output is turned on, and postscript, pdf and xdv output are turned off. This is equivalent to using the options **-dvi -ps- -pdf- -xdv-**.

If FORMAT is pdf, then pdf output is turned on, and dvi, postscript and xdv output are turned off. This is equivalent to using the options **-pdf -ps- -dvi- -xdv-**.

If FORMAT is anything else, *latexmk* gives an error.

-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.

-parallel

(Experimental: subject to change in future versions.) Enable parallel processing using fork(). This option sets the variable *$use_parallel* to 1.

See the description of the variable *$use_parallel* for details

-parallel-

(Experimental: subject to change in future versions.) Disable parallel processing. (This is the default behavior.) This option sets the variable *$use_parallel* to 0.

See the description of the variable *$use_parallel* for details

-parallel=n

(Experimental: subject to change in future versions.) Same as **-parallel** but limits concurrency to n simultaneous processes that are processing the document. (There are additional oversight processes.) Using n=0 or n=1 gives no parellel processing, while n<0 gives unlimited parallelism. This option sets the variable *$use_parallel* to 1, and sets *$parallel_jobs* to the value n.

See the description of the variable *$use_parallel* for details

-pdf Generate pdf version of document using *pdflatex*. (And turn off any incompatible requests.)

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

-pdfdvi

Generate dvi file and then pdf version of document from the dvi file, by default using dvipdf. (And turn off any incompatible requests.)

The program used to compile the document to dvi is *latex* by default, but this can be changed to *dvilulatex* by the use of the **-dvilua** option or by setting *$dvi_mode* to 2.

-pdflua

Generate pdf version of document using *lualatex*. (And turn off any incompatible requests.)

-pdfps

Generate dvi file, ps file from the dvi file, and then pdf file from the ps file. (And turn off any incompatible requests.)

The program used to compile the document to dvi is *latex* by default, but this can be changed to *dvilulatex* by the use of the **-dvilua** option or by setting *$dvi_mode* to 2.

-pdfxe

Generate pdf version of document using *xelatex*. (And turn off any incompatible requests.)

Note that to optimize processing time, *latexmk* uses *xelatex* to generate an .xdv file rather than a pdf file directly. Only after possibly multiple runs to generate a fully up-to-date .xdv file does *latexmk* then call *xdvipdfmx* to generate the final .pdf file.

(*Note:* The reason why *latexmk* arranges for *xelatex* to make an .xdv file instead of the *xelatex*'s default of a .pdf file is as follows: When the document includes large graphics files, especially .png files, the production of a .pdf file can be quite time consuming, even when the creation of the .xdv file

by *xelatex* is fast. So the use of the intermediate .xdv file can result in substantial gains in processing time, since the .pdf file is produced once rather than on every run of *xelatex*.)

-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.)

If after all options have been processed, pdf generation is still turned off, then generation of a dvi file will be turned on, and then the program used to compiled a document will be *latex* (or, more precisely, whatever program is configured to be used in the *$latex* configuration variable).

-pdflatex

Set the generation of pdf files by *pdflatex*. (And turn off any incompatible requests.)

Note: to set the command *used when pdflatex is specified, see the* **-pdflatex="COMMAND"** *option.*

-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 *$pdflatex* configuration variable. (The option **-pdflatex** in fact sets the variable *$pdflatex*.) 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).

Note: This option when provided with the COMMAND argument only sets the command for invoking pdflatex; it does not turn on the use of pdflatex. That is done by other options or in an initialization file.

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

-pdflualatex="COMMAND"

Equivalent to **-lualatex="COMMAND"**.

-pdfxelatex="COMMAND"

Equivalent to **-xelatex="COMMAND"**.

-pretex=CODE

Given that CODE is some TeX code, this options sets that code to be executed before inputting source file. This only works if the command for invoking the relevant **latex* is suitably configured. See the documentation of the variable *$pre_tex_code*, and the substitution strings %P and %U for more details. This option works by setting the variable *$pre_tex_code*.

See also the **-usepretex** option.

An example:

```
latexmk -pretex='\AtBeginDocument{Message\par}' -usepretex foo.tex
```

But this is better written

```
latexmk -usepretex='\AtBeginDocument{Message\par}' foo.tex
```

If you already have a suitable command configured, you only need

```
latexmk -pretex='\AtBeginDocument{Message\par}' foo.tex
```

-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. (And turn off any incompatible requests.)

-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, that will select the kind of file to be previewed (.pdf, .ps or .dvi). Otherwise the viewer views the "highest" kind of output file that is made, with the ordering being .pdf, .ps, .dvi (high to low). This option is incompatible with the **-p** and **-pvc** options, so it turns them off.

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

-pvc "Continuous preview mode", i.e., compile document, run a file previewer, and then regularly check for changed source files and recompile as needed. (See the Description above). Which files are 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 when you use this option, you should 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, as this is normally best for continuous preview mode. If you really want force mode, use the options in the order **-pvc -f**.

If you wish to do a "continuous compile", but don't want a previewer to be run, then do **-pvc -view=none**.

With a good previewer the display will be automatically updated when the viewed file changes. (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**.

-pvctimeout
Do timeout in pvc mode after period of inactivity, which is 30 min. by default. Inactivity means a period when *latexmk* has detected no file changes and hence has not taken any actions like compiling the document.

-pvctimeout-
Don't do timeout in pvc mode after inactivity.

-pvctimeoutmins=<time>
Set period of inactivity in *minutes* for pvc timeout.

-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.

-rc-report
After initialization, give a list of the RC files read. (Default)

-rc-report-
After initialization, do not give a list of the RC files read.

-recorder
Give the -recorder option with **latex*. 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*. This is the default setting of *latexmk*, unless overridden in an initialization file.

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

-recorder-
Do not supply the -recorder option with **latex*.

-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* options that *latexmk* recognizes, but that it simply passes through to the programs **latex* 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* that is being used, that program will probably give a warning or an error.) These options are very numerous, but are not listed in this documentation because they have no direct effect on *latexmk*'s actions.

There are a few options (e.g., **-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 certain options for **latex* (e.g., **-recorder**) that trigger special actions or behavior by *latexmk* itself. Depending on the action, they may also be passed in some form to the called **latex* program, and/or may affect other programs as well. These options do have entries in this documentation. Among these options are: **-jobname=STRING**, **-aux-directory=dir**, **-output-directory=DIR**, **-quiet**, and **-recorder**.

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

Notes that may help when debugging problems: The method that *latexmk* to pass any of the accepted extra options to **latex* is that when it constructs the command line for running **latex*, it inserts the options where the placeholder %O appears in the strings specifying the command line. (I.e., in the strings given by the configuration variables *$pdflatex* etc.) Note that *latexmk* uses this same method to add other options to the command line, and that the strings in *$pdflatex* etc may contain other options. So depending on the situation and the configuration, the use of an extra option on the command line *latexmk* might not achieve the desired effect; e.g., it could be overridden by a later option in the command line.

-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*, and similarly for its friends.

See also the **-logfilewarninglist** and **-logfilewarninglist-** options.

Also reduce the number of informational messages that *latexmk* itself 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*, *$dvilualatex_silent_switch*, *$latex_silent_switch*, *$lualatex_silent_switch* *$makeindex_silent_switch*, *$pdflatex_silent_switch*, and *$xelatex_silent_switch*

-stdtexcmds

Sets the commands for *latex*, etc, so that they are the standard ones. This is useful to override special configurations.

The result is that *$latex = 'latex %O %S'*, and similarly for *$pdflatex*, *$lualatex*, and *$xelatex*. (The option **-no-pdf** needed for *$xelatex* is provided automatically, given that %O appears in the definition.)

-time Show time used. (On MS Windows, what is shown is clock time; on other systems CPU time.) See also the configuration variable *$show_time*.

-time- Do not show time used. See also the configuration variable *$show_time*.

-use-make

When after a run of **latex*, there are warnings about missing files (e.g., as requested by the LaTeX \input, \include, and \includgraphics commands), *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 as a last resort *latexmk* will try to use 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, lualatex, xelatex*).

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.)

-usepretex
Sets the command lines for *latex*, etc, so that they use the code that is defined by the variable *$pre_tex_code* or that is set by the option **-pretex=CODE** to execute the specified TeX code before the source file is read. This option overrides any previous definition of the command lines.

The result is that *$latex = 'latex %O %P'*, and similarly for *$pdflatex*, *$lualatex*, and *$xelatex*. (The option **-no-pdf** needed for *$xelatex* is provided automatically, given that %O appears in the definition.)

-usepretex=CODE
Equivalent to **-pretex=CODE -usepretex**. Example

```
latexmk -usepretex='\AtBeginDocument{Message\par}' foo.tex
```

-v or **-version**
Print version number of *latexmk*.

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

-view=default, **-view=dvi**, **-view=hnt**, **-view=ps**, **-view=pdf**, **-view=none**
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 low-to-high order .dvi, .hnt, .ps, .pdf).

Note the possibility **-view=none** where no viewer is opened at all. One example of is use is in conjunction with the **-pvc** option, when you want *latexmk* to do a compilation automatically whenever source file(s) change, but do not want a previewer to be opened.

-Werror
This causes *latexmk* to return a non-zero status code if any of the files processed gives a warning about problems with citations or references (i.e., undefined citations or references or about multiply defined references). This is **after** *latexmk* has completed all the runs it needs to try and resolve references and citations. Thus **-Werror** causes *latexmk* to treat such warnings as errors, but only when they occur on the last run of *latex and only after processing is complete. Also can be set by the configuration variable *$warnings_as_errors*.

-xdv Generate xdv version of document using xelatex. (And turn off any incompatible requests.)

-xelatex
Use *xelatex*. That is, use *xelatex* to process the source file(s). This will cause generation of a pdf (but indirectly through a xdv file). (And turn off any incompatible requests.)

This option is equivalent to using the following option

```
-pdfxe
```

[*Note*: Note that the method of implementation of this option, but not its intended primary effect,

differ from some earlier versions of *latexmk*. *Latexmk* first uses *xelatex* to make an .xdv file, and does all the extra runs needed (including those of *bibtex*, etc). Only after that does it make the pdf file from the .xdv file, using *xdvipdfmx*. See the documentation for the **-pdfxe** for why this is done.]

-xelatex="COMMAND"

This sets the string specifying the command to run *xelatex*. It sets the variable *$xelatex*.

Warning: It is important to ensure that the **-no-pdf** is used when *xelatex* is invoked, since *latexmk* expects *xelatex* to produce an .xdv file, not a .pdf file. If you provide %O in the command specification, this will be done automatically. See the documentation for the **-pdfxe** option for why *latexmk* makes a .xdv file rather than a .pdf file when *xelatex* is used.

An example of the use of the **-xelatex** option:

```
latexmk -pdfxe -xelatex="xelatex --shell-escape %O %S"  foo.tex
```

Note: This option when provided with the COMMAND argument only sets the command for invoking xelatex; it does not turn on the use of lualatex. That is done by other options or in an initialization file.

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*

DEALING WITH ERRORS, 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. Generally, remember that *latexmk* does its work by running other programs. Your first priority in dealing with errors should be to examine what went wrong with the individual programs. Then you need to correct the causes of errors in the runs of these programs. (Often these come from errors in the source document, but they could also be about missing LaTeX packages, etc.)

c. If *latexmk* doesn't run the programs the way you would like, then you need 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!)

The remainder of these notes consists of ideas for dealing with more difficult situations.

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

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

f. There's a useful trick that can be used when you use *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.

g. See also the section "Advanced Configuration: Some extra resources".

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

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

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

AUXILIARY AND OUTPUT DIRECTORIES

Running *latex and the associated programs generate a number of files, it is often convenient to arrange for the generated files to be in a different directory than the source file(s) of a document. For our purposes here, we identify two classes of generated file.

One class is what one may term the final output files, for example, the .pdf file generated by running pdflatex, or the .dvi file from latex. Also in this class is the ps file generated by applying dvips to a .dvi file. There are also .synctec or .synctex.gz files that can be used by programs that display .pdf files and the like to relate positions in them to positions in source files.

The second class of file is composed of all other generated files: These include notably .aux files that are used for implementing cross referencing, and are both generated on one run and read on a later run. Many packages generate yet more such intermediate files, as well as programs like bibtex, makeindex, etc. There are also .log files from *latex and corresponding files from other programs.

Let us use the term "output directory" for the directory that receives the final output files, and "aux directory" for the directory for the other generated files. If no special options are provided to the *latex programs, these directories default to the current directory, and then the generated files aren't segregated. If the two directories are the same, as is the simplest situation, then all generated files are written to the same directory, and one often simply refers to the output directory, without mentioning a separate aux directory.

Support for them is provided for them in the *latex programs: by the single option **-output-directory** for the TeXLive implementations, and by the options **-aux-directory** and **-output-directory** for the MiKTeX implementations. Special support like this is needed for two reasons: First is that there are many packages that write files and it needs to be arranged that these are automatically written to the appropriate directory without any rewriting of the packages' code. Second is that the files are often read in again on subsequent runs of *latex, and it is necessary that the program knows where to find the files.

A complication is that the TeXLive implementation does not allow for separate aux and output directories. Latexmk deals with this by being able to emulating a separate aux directory: In this method it invokes *latex with just an **-output-directory** option, with the directory set not to the desired output directory, but to the aux directory. After running *latex, it moves the relevant final output file(s) to the intended output directory. Emulation can be turned on by setting the configuration variable *$emulate_aux* to one in a configuration file or by using latexmk's **-emulate-aux-dir** option. The emulation method works equally well if MiKTeX is used.

Latexmk also turns emulation on if it is found to be needed, as follows. Suppose emulation is initially off, but the aux and output directories are different. Then latexmk invokes *latex with an -aux-directory option and after the run finds that it hasn't been obeyed, notably because the .log file is in the output directory rather than the aux directory. Latexmk then sets emulation on, and retries. Conceivably, it could move all the appropriate generated files from the output directory to the aux directory; but there is such a large variety of possibilities for these files that this is hard to identify all of them reliably except for simple cases.

Note that the emulation issue only arises when the user has arranged for the the aux and output directories to be different. When instead they are equal, e.g., because the user only set the *$out_dir* variable, then latexmk invokes *latex with only an **-output-directory** option, which works as intended with both TeXLive and MiKTeX.

In addition, latexmk arranges the invocations of any auxiliary programs like bibtex and makeindex so that they will read and write the relevant files from and to the aux directory. Programs like dvips, dvipdf, ps2pdf, and xdvipdfmx are invoked so that they read from the appropriate places and write their output to the output directory.

Files considered as final output files, i.e., those that belong in the output directory rather than the aux directory: These have the extensions .dvi, .ps., .pdf, .synctex, and .synctex.gz. A special case, because of compatibility issues, is of .fls files: See below.

Note that xelatex when invoked with its **-no-pdf** option, as latexmk does, generates an .xdv file, which would appear to have the same status as a .dvi file generated by latex. Nevertheless, latexmk treats .xdv as an intermediate file that is found in the aux directory. This is to match MiKTeX's treatment of the **-aux-directory** option. As further justification, one can say that under modern conditions an .xdv file is (almost) always an intermediate file. Historically, the situation with .dvi files from latex was different, and currently dvi previewers do exist.

Variables and options for directories: The variables for setting the aux and output directories are *$aux_dir* and *$out_dir*, with corresponding command line options **-auxdir** (or **-aux-directory**) and **-outdir** (or **-output-directory**). When a value for these is blank (which is the default value), it implies the use of a default: For the aux directory, the default is to set it equal to the output directory. For the output directory, the default is to be the current directory.

For the turning on and off of the emulation mode, there is the configuration variable *$emulate_aux* and the options **-emulate-aux-dir**, **-emulate-aux-dir-**, **-noemulate-aux-dir**.

Interaction with -cd option: When the **-cd** option is used (or the equivalent setting of *$do_cd* variable), then latexmk changes the working directory to the document directory before invoking *latex. If the aux and/or output directories are given by relative paths, e.g., by **-outdir=output** for a directory named "output", then the directories are relative to the document directory, rather than relative to the working directory that was in effect when latexmk was invoked. This matches the behavior of *latex as invoked with the provided command line directory argument(s) after the change of working directory to the document directory.

Automatic creation of aux and output directories: Unlike *latex, if latexmk finds the requested directory/ies don't exist, it creates it/them, thereby avoiding errors when *latex is invoked.

If the document uses the \include macro to read a .tex file from a subdirectory, *latex will attempt to write an extra aux file to the corresponding subdirectory of the aux directory. If the subdirectory doesn't exist, then *latex will complain that it can't write the aux file. After the run of *latex, latexmk detects this situation, creates the necessary directory, and reruns *latex with the error situation corrected.

Choice of aux and output directories: Often the aux and output directories are given as subdirectories of the document directory, e.g., by **-outdir=output**. But it is possible to provide, for example, an absolute path or a path relative to a parent directory, e.g., "/tmp/foo" or "../output". Be aware that in general this can cause problems, notably 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.

By default, latexmk evades this issue: Before running bibtex and makeindex, latexmk changes working directory to the aux directory, with appropriate settings of search paths. The use or non-use of this trick is governed by the variables *$bibtex_fudge* and *$makeindex_fudge*. Unfortunately, the trick sometimes makes bibtex and makeindex unable to find files.

If necessary the trick can be turned off. But this is incompatible with an aux directory like, "/tmp/foo" of "../output"). If you really have to deal with this situation, and only if you have to deal with it, then you need to disable the security measures (and assume any risks). One way of doing this is to temporarily set an operating system environment variable openout_any to "a" (as in "all"), to override the default "paranoid" setting.

Certain names of aux and output directories not allowed on Microsoft Windows: It is natural to want to use the name "aux" for the aux directory, e.g., by using the option **-auxdir=aux**. But on Microsoft operating systems "aux" is one of the names that is not allowed for a file or directory. I find it useful to standardize on a name like "auxdir" (e.g., by **-auxdir=auxdir**); this works independently of operating system.

Location of .fls file: Much of the dependency information that latexmk uses comes from the .fls file generated when *latex is invoked with the **-recorder** option, which latexmk does by default. It may seem rational that this is written to the aux directory. But in fact versions of MiKTeX prior to Oct. 2020 wrote it to the output directory. Later versions do write it to the aux directory. To deal with this, latexmk does two things: First, if latexmk finds that the .fls file has only been generated in the "wrong" directory, then latexmk copies it to the

expected directory, after which latexmk's operation continues correctly independently of the behavior of *latex. Second it allows its idea of the "correct" (or expected) directory to be configured by the variable *$fls_uses_aux_dir*. This defaults to zero, to correspond to MiKTeX's current behavior.

ALLOWING FOR CHANGE OF OUTPUT FILE TYPE

When one of the latex engines is run, the usual situation is that *latex* produces a .dvi file, while *pdflatex* and *lualatex* produce a .pdf file. For *xelatex* the default is to produce a .pdf file, but to optimize processing time *latexmk* runs *xelatex* its **-no-pdf** option so that it produces an .xdv file. Further processing by *latexmk* takes this as a starting point.

However, the actual output file may differ from the normal expectation; and then *latexmk* can adjust its processing to accommodate this situation. The difference in output file type can happen for two reasons: One is that for *latex*, *pdflatex* and *lualatex* the document itself can override the defaults. The other is that there may be a configuration, or misconfiguration, such that the program that *latexmk* invokes to compile the document is not the expected one, or is given options incompatible with what *latexmk* initially expects.

Under *latex* and *pdflatex*, control of the output format by the document is done by setting the \pdfoutput macro. Under *lualatex*, the \outputmode macro is used instead.

One example of an important use-case for document control of the output format is a document that uses the psfrag package to insert graphical elements in the output file. The psfrag package achieves its effects by inserting postscript code in the output of the compilation of the document. This entails the use of compilation to a .dvi file, followed by the use of conversion to a postscript file (either directly, as by *dvips* or implicitly, as an intermediate step by *dvipdf*). Then it is useful to force output to be of the .dvi format by inserting \pdfoutput=0 in the preamble of the document.

Another example is where the document uses graphics file of the .pdf, .jpg, and png types. With the default setting for the graphicx package, these can be processed in compilation to .pdf but not with compilation to .dvi. In this case, it is useful to insert \pdfoutput=1 in the preamble of the document to force compilation to .pdf output format.

In all of these cases, it is needed that *latexmk* has to adjust its processing to deal with a mismatch between the actual output format (out of .pdf, .dvi, .xdv) and the initially expected output, if possible. *Latexmk* does this provided the following conditions are met.

The first is that *latexmk*'s *$allow_switch* configuration variable is set to a non-zero value as it is by default. If this variable is zero, a mismatch of filetypes in the compilation results in an error.

The second condition for *latexmk* to be able to handle a change of output type is that no explicit requests for .dvi or .ps output files are made. Explicit requests are by the **-dvi** and **-ps**, **-print=dvi**, **-print=ps**, **-view=dvi**, and **-view=ps** options, and by corresponding settings of the *$dvi_mode*, *$postscript_mode*, *$print_type*, and *$view* configuration variables. The print-type and view-type restrictions only apply when printing and viewing are explicitly requested, respectively. For this purpose, the use of the **-pdfdvi** and **-pdfps** options (and the corresponding setting of the *$pdf_mode* variable) does not count as an explicit request for the .dvi and .ps files; they are merely regarded as a request for making a .pdf file together with an initial proposal for the processing route to make it.

Note that when accommodating a change in output file type, there is involved a substantial change in the network of rules that *latexmk* uses in its actions. The second condition applied to accommodate a change is to avoid situations where the change in the rule network is too radical to be readily handled automatically.

CONFIGURATION/INITIALIZATION (RC) FILES

In this section is explained which configuration files are read by *latexmk*. Subsequent sections "How to Set Variables in Initialization Files", "Format of Command Specifications", "List of Configuration Variables Usable in Initialization Files", "Custom Dependencies", and "Advanced Configuration" give details on what can be configured and how.

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 the following directories for a system RC file, which may be named either "LatexMk" or "latexmkrc". The directories are searched in the following order, and *latexmk* uses the first such file it finds (if any): "/etc", "/opt/local/share/latexmk", "/usr/local/share/latexmk", "/usr/local/lib/latexmk".

On a MS-Windows system it looks just in "C:\latexmk".

On a cygwin system (i.e., a MS-Windows system in which Perl is that of cygwin), *latexmk* looks in the directory "/cygdrive/c/latexmk", in addition to the set used for UNIX.

On an msys system (i.e., a MS-Windows system in which Perl is that of msys), *latexmk* looks in the directory "/c/latexmk", in addition to the set used for UNIX.

If the environment variable LATEXMKRCSYS is set, its value is used as the name of the system RC file, instead of any of the above.

2) The user's RC file, if it exists. This can be in one of two places. The traditional one is ".latexmkrc" in the user's home directory. The other possibility is "latexmk/latexmkrc" in the user's XDG configuration home directory. The actual file read is the first of "$XDG_CONFIG_HOME/latexmk/latexmkrc" or "$HOME/.latexmkrc" which exists. (See https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html for details on the XDG Base Directory Specification.)

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 macOS). 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, in which case *latexmk* does not look for an RC file in it.]

$XDG_CONFIG_HOME is the value of the environment variable XDG_CONFIG_HOME if it exists. If this environment variable does not exist, but $HOME is non-blank, then $XDG_CONFIG_HOME is set to the default value of $HOME/.config. Otherwise $XDG_CONFIG_HOME is blank, and *latexmk* does not look for an RC file under it.

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 https://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". (See the earlier section "Configuration/Initialization (rc) Files" for the files where the configurations are done.) Syntax for setting these variables is of the following forms:

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

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. (Note that some of the possibilities listed here do not apply to the *$kpsewhich* variable; see its documentation.)

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 in the invocation of *latexmk*, it results in the replacement of %O by "-interaction=batchmode".)

The available placeholders are:

%A basename of the main tex file. Unlike %R, this is unaffected by the setting of a jobname by the *-jobname* option or the *$jobname* configuration value.

%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

%P If the variable *$pre_tex_code* is non-empty, then %P is substituted by the contents of *$pre_tex_code* followed by *\input{SOURCE}*, where SOURCE stands for the name of the source file. Appropriate quoting is done. This enables TeX code to be passed to one of the **latex* engines to be executed before the source file is read.

If the variable *$pre_tex_code* is the empty string, then %P is equivalent to %S.

%R root filename.

By default this is the basename of the main tex file. However the value can be changed by the use of the *-jobname* option or the *$jobname* configuration variable. This is then the basename for files like the .aux and .log files produced by running *latex, as well for the main .dvi, .pdf, .ps and/or .xdvi files.

%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.

%U If the variable *$pre_tex_code* is non-empty, then its value is substituted for %U (appropriately quoted). Otherwise it is replaced by a null string.

%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 '\'. Note that if after initialization, *$out_dir* is set, but *$aux_dir* is not set (i.e., it is blank), then *latexmk* sets *$aux_dir* to the same value *$out_dir*.

%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 "%%".

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 some older 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*.

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, *latexmk* 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: Under MS-Windows forward slashes are equivalent to backslashes in a filename under almost all circumstances, 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 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;
}
```

For some of the more exotic possibilities that then become available, see the section "Advanced configuration: Some extra resources and advanced tricks". Also see some of the examples in the directory *example_rcfiles* in the *latexmk* distribution.

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.

You can also use a Perl subroutine instead of a script -- see above. This is generally the most flexible and portable solution.

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 macOS), 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

In this section are specified the variables whose values can be adjusted to configure *latexmk*. (See the earlier section "Configuration/Initialization (rc) Files" for the files where the configurations are done.)

Default values are indicated in brackets. Note that for variables that are boolean in character, concerning whether *latexmk* does or does not behave in a certain way, a non-zero value, normally 1, indicates true, i.e., the behavior occurs, while a zero value indicates a false value, i.e., the behavior does not occur.

$allow_subdir_creation [1]

Specify action to take when message(s) in the .log file indicate a failure of an attempt by a *latex compilation to write a file to a subdirectory of the output directory because the subdirectory didn't exist.

If the value if *$allow_subdir_creation* is 0, no action is taken. If it is 1, then the appropriate subdirectory is created and a rerun of *latex is triggered, but only if the file being written is an .aux file. (This happens, for example, if the document includes a file from a subdirectory of the document directory, by the \include command. If the value of *$allow_subdir_creation* is 2, then the subdirectory creation is done independently of which type of file is in question.

$allow_switch [1]

This controls what happens when the output extension of latex, pdflatex, lualatex or xelatex differs from what is expected. (The possible extensions are .dvi, .pdf, .xdv.) This can happen with the use of the \pdfoutput macro in a document compiled under *latex* or *pdflatex*, or with the use of the \outputmode macro under *lualatex*. It can also happen with certain kinds of incorrect configuration.

In such a case, *latexmk* can appropriately adjust its network of rules. The adjustment is made if *$allow_switch* is on, and if no request for a dvi or ps file has been made.

See the section ALLOWING FOR CHANGE OF OUTPUT EXTENSION.

$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, since *pdflatex* provides no way of specifying a chosen name for the output file.)

This use of a temporary file solves a problem that the making of these files can occupy a substantial time. If a viewer (notably *gv*) sees that the file has changed, it may read the new file before the program writing the file has not yet finished its work, which can cause havoc.

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.

$analyze_input_log_always [1]

After a run of latex (etc), always analyze .log for input files in the <...> and (...) constructions. Otherwise, only do the analysis when fls file doesn't exist or is out of date.

Under normal circumstances, the data in the fls file is reliable, and the test of the log file gets lots of false positives; usually $analyze_input_log_always is best set to zero. But the test of the log file is needed at least in the following situation: When a user needs to persuade latexmk that a certain file is a source file, and latexmk doesn't otherwise find it. Then the user can write code that causes a line with (...) to be written to log file. One important case is for lualatex, which doesn't always generate lines in the .fls file for input lua files. (The situation with lualatex is HIGHLY version dependent, e.g., there was a big change between TeXLive 2016 and TeXLive 2017.)

To keep backward compatibility with older versions of latexmk, the default is to set $analyze_input_log_always to 1.

$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 aux directory, i.e., the directory in which auxiliary files (aux, log, etc) are to be written by a run of *latex.

If this variable is not set, but *$out_dir* is set, then latexmk takes the aux directory to equal the output directory which is the directory to which final output files are to be written.

If neither variable is set, then the current directory when *latex is invoked is used both for the aux and output directories.

If the aux and output directories are distinct, then the aux directory contains all generated files with the exception of "final output files", which are defined to be .dvi, .ps, .pdf, .synctex, and .synctex.gz files.

See the section AUXILIARY AND OUTPUT DIRECTORIES for more details.

$aux_out_dir_report [0]

If this variable is set to 1, then prior to the processing of each primary .tex file, list the settings for aux and output directories, after they have been normalized from the settings specified during initialization.

This report gives a reminder of where to look for generated files.

The report is done per primary .tex file, because of possible directory changes for each file (when the **-cd** option is used). In the simplest cases, the directory names are the same as originally specified. But in general some clean up/normalization is performed; this helps performance and cleans up output to the screen.

If this variable is set to 2, then halt after reporting the settings for the aux and out directories, rather than continuing with processing of tex files. This setting is primarily used for debugging configuration issues. See the **-dir-report-only** option.

$bad_warning_is_error [0]

Whether to treat bad warnings reported by *latex in log file as errors. The specifications of the warning messages are in *@bad_warnings*.

@bad_warnings

Array of regular expressions specifying messages in log file that are officially treated as warnings rather than errors by *latex, but which a user may treat as errors: See *$bad_warning_is_error*.

Currently the default set of these warnings is those about \end occurring inside constructs.

$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.

$biber ["biber %O %S"]

The biber processing program.

$biber_silent_switch ["--onlylog"]

Switch(es) for the biber processing program when silent mode is on.

@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. Generally, it is simplest always to use forward slashes instead of backward slashes.

Important note: This variable is now mostly obsolete in the current version of *latexmk*, since it now uses 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*.

%biblatex_ignore_pattern

Patterns in .xml and .aux files to ignore when the biblatex package is used, to avoid an extra unnecessary run of **latex* after a change in the bibliography. For details see the description of the variable *$biblatex_optimize*.

The default settings are

```
$biblatex_ignore_pattern{xml} = 'package="biblatex" .*active="[01]"';
$biblatex_ignore_pattern{aux} = '^\\abx@aux@read@bbl(@mdfivesum.*|rerun)';
```

These settings are tested to be correct for version 3.22a of biblatex.

$biblatex_optimize [1]

When *latexmk* detects that the biblatex package is being used (for making bibliographic citations), this variable determines whether to make a correction to dependency detection, to avoid an unnecessary extra run of **latex* after a change in the bibliography.

Without this adjustment, there would be an extra run of **latex* whenever there is a bibliography-associated change. The problem is that to be able to provide the user with information about whether an extra run of *biber*/*bibtex* and/or of **latex* is needed, biblatex writes certain dependency information into the .aux and the .run.xml files. This information is not used by *latexmk*, since it has more general and more reliable ways of determining which programs to run. Without special precautions, the dependency information written by biblatex would have the side effect that *latexmk* makes an extra run of **latex* to get stable files. The extra run is harmless, but annoying (and takes time).

To avoid the extra run, *latexmk* configures itself to ignore the relevant lines in the .aux and .run.xml files. This is done by putting appropriate entries in the variable *%hash_calc_ignore_pattern*. The entries are copied from those in the *%biblatex_ignore_pattern* configuration variable. The adjustment to *%hash_calc_ignore_pattern* is only done when it is detected that the biblatex package is used and the variable *$biblatex_optimize* is at its default value. But if there is a relevant entry already in *%hash_calc_ignore_pattern*, the entry is not updated; this avoids interference with a (non-default) setting of this variable by the user.

$bibtex ["bibtex %O %S"]

The BibTeX processing program.

$bibtex_fudge [1]

When using bibtex, whether to change directory to $aux_dir before running bibtex.

The need arises as follows:

a. With bibtex before about 2019, if the filename given to it contains a path component, there was a bug that bibtex would not find extra aux files, as produced by the \include command in TeX.

b. With all moderately recent versions of bibtex, bibtex may refuse to write its bbl and blg files, for security reasons, for certain cases of the path component of the filename given to it.

However, there are also rare cases where the change-directory method prevents bibtex from finding certain bib or bst files. Then *$bibtex_fudge* needs to be set to 0.

$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, and also controls whether or not .bbl files are deleted in a cleanup operation.

The possible values of *$bibtex_use* are:
0: never use *bibtex* or *biber*; never delete .bbl files in a cleanup.
1: only use *bibtex* or *biber* if the bib file(s) exist; never delete .bbl files in a cleanup.
1.5: only use *bibtex* or *biber* if the bib files exist; conditionally delete .bbl files in a cleanup (i.e., delete them only when the bib files all exist).
2: run *bibtex* or *biber* whenever it appears necessary to update the bbl file(s), without testing for the existence of the bib files; always delete .bbl files in a cleanup.

Note: When *biber* is being used, conditional use of *biber* can be problematic. From *latexmk*'s point of view the problem is that because of how *biber* works, a full knowledge of its source files can only be obtained after running *biber*. In contrast, for *bibtex*, full information on which bib files are used is obtained from the .aux file(s) after a run of *latex. But for *biber*, the corresponding information is somewhat incomplete; this the information obtained in the .bcf file that is generated by the biblatex package during a run of *latex.

$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 the fls file (or failing that, in log file) as being generated. It will also include files made from these first generation generated files.

This operation is somewhat dangerous, and can have unintended consequences, since the files to be deleted are determined from a file created by *latex, which can contain erroneous information. Therefore this variable is turned off by default, and then files to be deleted are restricted to those explicitly specified by patterns configured in the variables *clean_ext*, *clean_full_ext*, and *@generated_exts*. Standard cases (e.g., .log files) appear in *latexmk*'s initial value for the array *@generated_exts*.

$cleanup_mode [0]
If nonzero, specifies cleanup mode: 1 for full cleanup, 2 for cleanup except for .dvi, .ps and .pdf files, 3 for cleanup except for dep and aux files. (There is also extra cleaning as specified by the *$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 pythontex-files-%R/*";
```

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, FOO-figures*.log, and pythontex-files-FOO/*, where FOO stands for the basename of the file being processed (as in FOO.tex).

Most of the files to be deleted are relative to the directory specified by *$aux_dir*. Note that if *$out_dir* but not *$aux_dir* is set, then in its initialization, *latexmk* sets *$aux_dir* equal to *$out_dir*. A

normal situation is therefore that *$aux_dir* equals *$out_dir*, which is the only case directly supported by TeXLive, unlike MiKTeX. Note that even with TeXLive latexmk does now support different values for the directories -- see the explanation of the *$emulate_aux* variable.

If *$out_dir* and *$aux_dir* different, latexmk actually deletes any files of the specified names in both *$aux_dir* and *$out_dir*; this is because under certain error conditions, the files may be put in *$out_dir* instead of *$aux_dir*. This also handles the case of deleting any fls file, since that file is in *$out_dir*.

The filenames specified for a clean-up operation can refer not only to regular files but also to directories. Directories are only deleted if they are empty. An example of an application is to pythontex, which creates files in a particular directory. You can arrange to remove both the files and the directory by setting

```
$clean_ext = "pythontex-files-%R/* pythontex-files-%R";
```

See also the (array) variable *@generated_exts*. In the past, this variable had certain uses beyond that of *$clean_ext*. But now, they accomplish the same things. In fact, after initialization including the processing of command line options, *latexmk* simply appends the list of extensions in *$clean_ext* to the array *@generated_exts*.

$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*.

The files specified by *$clean_full_ext* to be deleted are relative to the directory specified by *$out_dir*.

$compiling_cmd [""], $failure_cmd [""], $warning_cmd [""], $success_cmd [""]

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

The commands are executed at the following points: *$compiling_cmd* at the start of compilation, *$success_cmd* at the end of a completely successful compilation, *$failure_cmd* at the end of an unsuccessful compilation, *$warning_cmd* at the of an otherwise successful compilation that gives warnings about undefined citations or references or about multiply defined references. If any of above variables is undefined or blank (the default situation), then the corresponding command is not executed.

However, when *$warning_cmd* is not set, then in the case of a compilation with warnings about references or citations, but with no other error, one or other of *$success_cmd* or *$failure_cmd* is used (if it is set) according to the setting of *$warnings_as_errors*.

An example of a simple 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\"";
$warning_cmd   = "xdotool search --name \"%D\" ".
                 "set_window --name \"%D CITE/REF ISSUE\"";
$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.

More complicated situations can best be handled by defining a Perl subroutine to invoke the necessary commands, and using the "internal" keyword in the definitions to get the subroutine to be invoked. (See the section "Format of Command Specifications" for how to do this.)

Naturally, the above settings that invoke the *xdotool* program are only applicable when the X-Window system is used for the relevant window(s). For other cases, you will have to find what software solutions are available.

@cus_dep_list [()]

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

@default_excluded_files [()]

When *latexmk* is invoked with no files specified on the command line, then, by default, it will process all files in the current directory with the extension .tex. (In general, it will process the files specified in the *@default_files* variable.)

But sometimes you want to exclude particular files from this default list. In that case you can specify the excluded files in the array *@default_excluded_files*. For example if you wanted to process all .tex files with the exception of common.tex, which is a not a standard alone LaTeX file but a file input by some or all of the others, you could do

```
@default_files = ("*.tex");

@default_excluded_files = ("common.tex");
```

If you have a variable or large number of files to be processed, this method saves you from having to list them in detail in *@default_files* and having to update the list every time you change the set of files to be processed.

Notes: 1. This variable has no effect except when no files are specified on the *latexmk* command line. 2. Wildcards are allowed in *@default_excluded_files*.

@default_files [("*.tex")]

Default list of files to be processed.

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 can list the files to be processed by setting *@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.

If you want *latexmk* to process all .tex files with a few exceptions, see the *@default_excluded_files* array variable.

$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.

$deps_escape ["none"]

This variable determines which kind of escaping of space characters to use in dependency lists. The possible values are "none", "unix", "nmake", corresponding respectively to no escaping, escaping with a "\" suitable for standard Unix make, and escaping with "^", suitable for Microsoft's nmake.

Currently the only character escaped is a space, since that is particularly common in file names and directory names. There are other characters that would need escaping if a dependency list is to be used as-is by a make program; but those characters (e.g., "$") commonly cause difficulties when used for .tex documents. Moreover, the detailed rules for which characters need to be escaped depends on the version of make.

$deps_file ["-"]

Name of file to receive list(s) of dependencies at the end of a run, to be used if *$dependents_list* is set. If the filename is "-", then the dependency list is set to stdout (i.e., normally the screen).

$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* 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.

$dvilualatex ["dvilualatex %O %S"]

Specifies the command line to invoke the dvilualatex 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.,

```
$dvilualatex = "dvilualatex --src-specials %O %S";
```

To do a coordinated setting of all of *$dvilualatex*, *$hilatex*, *$latex*, *$pdflatex*, *$lualatex*, and *$xelatex*, see the section "Advanced Configuration".

$dvi_mode [See below for default]

If one, generate a dvi version of the document by use of latex. Equivalent to the **-dvi** option.

If 2, generate a dvi version of the document by use of dvilualatex. Equivalent to the **-dvilua** 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 and $dvi_mode is zero, then $dvi_mode is set to 1.

$dvilualatex_silent_switch ["-interaction=batchmode"]

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

See details of the *$latex_silent_switch* for other information that equally applies to *$dvilualatex_silent_switch*.

$dvi_previewer ["start xdvi %O %S" under UNIX]

The command to invoke a dvi-previewer. [Under MS-Windows the default is "start"; then *latexmk* arranges to use the MS-Windows *start* program, which 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. [Under MS-Windows the default is "start"; then *latexmk* arranges to use the MS-Windows *start* program, which will cause to be run whatever command the system has associated with .dvi files.]

$dvipdf ["dvipdf -dALLOWPSTRANSPARENCY %O %S %D" under UNIX]

Command to convert .dvi to .pdf file.

Under Windows, the default is "dvips -P pdf -o - %S | ps2pdf -sDEVICE=pdfwrite -dALLOWPSTRANSPARENCY %O - %D". The reason for this difference is that the implementation of Ghostscript provided by Windows distributions of TeX only provides the *ps2pdf* program, but not *dvipdf*, unlike the case with typical UNIX Ghostscript packages.

A common reconfiguration is to use the dvipdfm command, which needs its arguments in a different order:

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

$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 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* variable -- see below -- will be included in the options substituted for "%O".

$dvips_landscape ["dvips -tlandscape %O -o %D %S"]

The program to used 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.

$emulate_aux [1]

Whether to emulate the use of aux directory when $aux_dir and $out_dir are different, rather than using the **-aux-directory** option for the *latex programs. (MiKTeX supports **-aux-directory**, but TeXLive doesn't.)

If you use a version of *latex that doesn't support **-aux-directory**, e.g., TeXLive, latexmk will automatically switch aux_dir emulation on after the first run of *latex, because it will find the .log file in the wrong place. But it is better to set *$emulate_aux* to 1 in an rc file, or equivalently to use the **-emulate-aux-dir** option. This emulation mode works equally well with MiKTeX.

Aux directory emulation means that when *latex is invoked, the *output* directory provided to *latex is set to be the desired aux directory. After that, any files that need to be in the output directory will be moved there by latexmk. (These are the files with extensions .dvi, .ps, .pdf, .synctex, .synctex.gz, and, depending on the setting of the *$fls_uses_out_dir* variable, also the .fls file.)

$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.

@file_not_found

This an array of Perl regular expressions that are patterns to find messages in the .log file from a run of *latex that indicate that a file was looked for and not found. To see the current default set, you should look at the definition of *@file_not_found* in the latexmk.pl file.

In the regular expression, the string for the name of the missing file should be enclosed in parentheses. That carries the implication that after latexmk gets a successful match to the pattern, the variable $1 is set to the filename, which is then picked up by latexmk.

If you happen to encounter a package that gives a missing file message of a different form than one that matches one of the built-in patterns, you can add another pattern to the array. An example would be

```
push @file_not_found, 'ˆNo file\\s+(.+)\\s*$';
```

The regular expression itself is

```
ˆMissing file\s+(.+)\s*$
```

But the corresponding string specification in the push statement has to have the backslashes doubled.

This regular expression matches a line that starts with 'No file', then has one or more white space characters, then any number of characters forming the filename, then possible white space, and finally the end of the line. (See documentation on Perl regular expressions for details.)

$filetime_causality_threshold [5]

The use of this variable is as follows: At a number of places, latexmk needs to determine whether a particular file has been produced during a just-concluded run of some rule/program or is leftover from a previous run. (An example is the production of a .bcf file by the biblatex package during a run of *latex to provide bibliographic information to the biber program. If a .bcf file is not produced during a current run of *latex, but is leftover from a previous run, then latexmk has to conclude that the .tex document has changed so that biber is no longer to be used.)

Latexmk's criterion that a file has been produced during a run is that the modification time of the file is more recent than the system time at the beginning of the run. Bascially, if the modification time is earlier than this, then it is a leftover from a previous run. However, a naive use of the criterion can, among other things, run afoul of the granularity of how file times are stored in some file systems, which means it is possible that the filesystem's reported time for a file might be a second or more earlier than the actual modification time, the exact difference being quite random.

The variable *$filetime_causality_threshold* allows an appopriate sloppiness in latexmk's use of file modification time. It can be quite generous; it should merely be less than the time scale on which a human user makes changes to source files for a document (or to configuration files, etc).

$fls_uses_out_dir [0]

This variable determines whether or not the .fls file should be in the output directory instead of the natural directory, which is the aux directory. If the variable is nonzero, the .fls file is to be in the output directory. See the section AUXILIARY AND OUTPUT DIRECTORIES for more details about these directories. The rationale for the existence of the variable *$fls_uses_aux_dir* is explained there.

In all cases, if *latexmk* finds that an .fls file has been generated in the opposite directory to the one specified by *$fls_uses_out_dir*, it copies the file to the other directory (aux or output directory as appropriate). The file is copied rather than simply moved, to avoid potential clashes with other software that assumes the .fls file is generated in the directory it was written to by *latex. Thus the effect an incorrect setting of *$fls_uses_out_dir* is only to cause a superfluous copy of the .fls file to be generated.

$force_mode [0]

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

@generated_exts [('aux', 'bcf', 'fls', 'idx', 'ind', 'lof', 'lot', 'out', 'toc', 'blg', 'ilg', 'log', 'xdv')]

This contains a list of extensions for files that are generated during processing, and that should be deleted during a main clean up operation, as invoked by the command line option **-c**. (The use of **-C** or **-gg** gives this clean up and more.)

The default values are extensions for standard files generated by *latex, bibtex, and the like. (Note that the clean up also deletes the fdb_latexmk file, but that's separately coded into latexmk, currently.)

After initialization of latexmk and the processing of its command line, the items in *clean_ext* are appended to *@generated_exts*. So these two variables have the same meaning (contrary to older versions of latexmk).

The items in *@generated_exts* are normally extensions of files, whose base name is the same as the main tex file. But it is also possible to specify patterns including that basename --- see the explanation of the variable *$clean_ext*.

In addition to specifying files to be deleted in a clean up, latexmk uses the same specification to assist its examination of changes in source files: Under some situations it needs to find those changes in files (since a previous run) that are expected to be due to the user editing a file. This contrasts with the cases of files that are generated by some program run by latexmk and that differ from the results of the previous run. This use of *@generated_exts* is normally unimportant, given the usual accuracy of latexmk's other ways of determining these generated files.

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 this variable equals zero, process files as usual, only running rules when they are determined to be out-of-date. This is equivalent to the **-g-** option.

If this variable equals one, force *latexmk* to run each rule at least once, even under situations whre *latexmk* would normally decide that no changes in the source files have occurred since the previous run. This behavior is useful, for example, if you change the configuration and wish to reprocess all files. This is equivalent to the **-g** option.

If this variable equals two, perform a full clean up, as if the **-C** had been given, and then process normally from a clean initial situation. This is equivalent to the **-gg** option for "super go mode" or "clean make".

If this variable equals three, require at least one run of *latex. I.e., process as normal, except that initially, the *latex rule that is in use is set to be out-of-date, independently of the state of the files. This is equivalent to the **-gt** 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 expression. (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.

To ignore all changes in files of a given type, use a pattern that matches all lines, e.g.,

```
$hash_calc_ignore_pattern{'tkz'} = '^';
```

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 that is read later. To do this, you use Perl's delete function, e.g.,

```
delete $hash_calc_ignore_pattern{'eps'};
```

$hilatex ["hilatex %O %S"]
specifies the command line for the hilatex program.

$hnt_mode [0]
Whether to generate a hnt version of the document by use of hilatex. Can be turned on by the use of the **-hnt** option.

$jobname [""]

This specifies the jobname, i.e., the basename that is used for generated files (.aux, .log, .dvi, .ps, .pdf, etc). If this variable is a null string, then the basename is the basename of the main tex file. (At present, the string in *$jobname* should not contain spaces.)

The placeholder '%A' is permitted. This will be substituted by the basename of the TeX file. The primary purpose is when a variety of tex files are to be processed, and you want to use a different jobname for each but one that is distinct for each. Thus if you wanted to compare compilations of a set of files on different operating systems, with distinct filenames for all the cases, you could set

```
$jobname = "%A-$^O";
```

in an initialization file. (Here *$^O* is a variable provided by perl that contains perl's name for the operating system.)

Suppose you had .tex files test1.tex and test2.tex. Then when you run

```
latexmk -pdf *.tex
```

both files will be compiled. The .aux, .log, and .pdf files will have basenames test1-MSWin32 ante test2-MSWin32 on a MS-Windows system, test1-darwin and test2-darwin on an macOS system, and a variety of similar cases on Linux systems.

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

(*For advanced users:* Because of the different way in which *latexmk* uses the command specified in *$kpsewhich*, some of the possibilities listed in the FORMAT OF COMMAND SPECIFICATIONS do not apply. The *internal* and *start* keywords are not available. A simple command specification with possible options and then "%S" is all that is guaranteed to work. Note that for other commands, "%S" is substituted by a single source file. In contrast, for *$kpsewhich*, "%S" may be substituted by a long list of space-separated filenames, each of which is quoted. The result on STDOUT of running the command is then piped to *latexmk*.)

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.

$kpsewhich_show [0]
Whether to show diagnostics about invocations of *kpsewhich*: the command line use to invoke it and the results. These diagnostics are shown if *$kpsewhich_show* is non-zero or if diagnostics mode is

on. (But in the second case, lots of other diagnostics are also shown.) Without these diagnostics there is nothing visible in *latexmk*'s screen output about invocations of *kpsewhich*.

$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"]

Specifies the command line for 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 %O %S";
```

To do a coordinated setting of all of *$dvilualatex*, *$hilatex*, *$latex*, *$pdflatex*, *$lualatex*, and *$xelatex*, see the section "Advanced Configuration".

$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.

$lualatex ["lualatex %O %S"]

Specifies the command line for the LaTeX processing program that is to be used when the *lualatex* program is called for (e.g., by the option **-lualatex**.

To do a coordinated setting of all of *$dvilualatex*, *$hilatex*, *$latex*, *$pdflatex*, *$lualatex*, and *$xelatex*, see the section "Advanced Configuration".

$lualatex_silent_switch ["-interaction=batchmode"]

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

See details of the *$latex_silent_switch* for other information that equally applies to *$lualatex_silent_switch*.

$make ["make"]

The make processing program.

$makeindex ["makeindex %O -o %D %S"]

The index processing program.

$makeindex_fudge [1]

When using makeindex, whether to change directory to $aux_dir before running makeindex. Set to 1 if $aux_dir is not an explicit subdirectory of current directory, otherwise makeindex will refuse to write its output and log files, for security reasons.

$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* before deciding that there may be an infinite loop and that it needs to bail out, rather than rerunning **latex* 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* 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. With the default value of 1 for this variable, when a command is executed under MS-Windows, *latexmk* substitutes "\" 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 almost all programs and for almost all filenames under MS-Windows, both "\" and "/" are acceptable as the directory separator character, provided at least that filenames are properly quoted. But it is possible that programs exist that only accept "\" on the command line, since that is the standard directory separator for MS-Windows. So for safety *latexmk* makes the substitution from "/" to "\", by default.

However there are also programs on MS-Windows for which a back slash "\" is interpreted differently than as a directory separator; for these the directory separator should be "/". Programs with this behavior include all the *latex programs in the TeXLive implementation (but not the MiKTeX implementation). Hence if you use TeXLive on MS-Windows, then *$MSWin_back_slash* should be set to zero.

$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 [""]

If non-blank, this variable specifies the output directory.

This is the directory in which the main output files are written (dvi, ps, pdf, synctex, synctex.gz). In addition, if the aux directory equals the output directory, as is the case by default, then other generated files are in effect written to the output directory.

If *$out_dir* is blank, the output directory is the current directory at the invocation of *latex; this is equivalent to setting *$out_dir* to '.'.

See the section AUXILIARY AND OUTPUT DIRECTORIES for more details.

$out2_dir [""]

(*Experimental new feature.*)

If non-blank, this variable specifies the final-output directory, i.e., the directory for the final output files. If this variable is blank (its default value), the final-output directory is the same as the output directory.

See the description of the option **-out2dir** for an explanation of the rationale for the idea of separate output and final-output directories.

If the final-output directory is different from the output directory, then after a full round of compilations of the document, the relevant set of files is copied here from the output directory. The files copied are specified by the *@out2_exts* variable, and by default are those with extensions 'hnt', 'pdf', 'ps', 'synctex', 'synctex.gz', and a basename the same as for the main *latex compilation.

@out2_exts [('synctex', 'synctex.gz')]

This array variable is used by latexmk as follows: After it is determined what kinds of final-output file are to be made, their extensions are appended to the array. (Examples: 'pdf' if *$pdf_mode* is non-zero, 'dvi if *$dvi_mode* is non-zero.)

Then these extensions are those for the files that are to be copied to the final-output directory. The basename of the files is that for the main *latex compilation (corresponding to the value specified by the placeholder %R).

The default value of the array is restricted to the synctex files, since those would not be determined automatically by latexmk's current methods.

More general names may be specified in the same way as for the *@generated_exts* variable, by inclusion of %R in a pattern. For example, suppose you have requested generation of a pdf file, but have added a post-processing step to make a 2-up version of the pdf file, with a name obtained by appending '-2up.pdf' to the basename. Then you would want to usee

```
push @out2_exts, '%R-2up.pdf';
```

$parallel_jobs [-1]

(Experimental: subject to change in future versions.) Sets the maximum number of simultaneous processes that are processing the document. (There are additional oversight processes.) Using a value 0 or 1 gives no parellel processing, while a negative value gives unlimited parallelism. But this value has no effect unless parallelism is turned on by the variable *$use_parallel*.

See the description of the variable *$use_parallel* for further details

$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*, using the command specified by the *$pdflatex* variable. If equal to 2, generate a pdf version of the document from the ps file, by using the command specified by the *$ps2pdf* variable. If equal to 3, generate a pdf version of the document from the dvi file, by using the command specified by the *$dvipdf* variable. If equal to 4, generate a pdf version of the document using *lualatex*, using the command specified by the *$lualatex* variable. If equal to 5, generate a pdf version (and an xdv version) of the document using *xelatex*, using the commands specified by the *$xelatex* and *xdvipdfmx* variables.

In *$pdf_mode*=2, it is ensured that .dvi and .ps files are also made. In *$pdf_mode*=3, it is ensured that a .dvi file is also made. But this may be overridden by the document.

$pdflatex ["pdflatex %O %S"]

Specifies the command line for the LaTeX processing program in a version that makes a pdf file instead of a dvi file.

An example use of this variable is to add certain options to the command line for the program, e.g.,

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

(In some earlier versions of *latexmk*, you needed to use an assignment to *$pdflatex* to allow the use of *lualatex* or *xelatex* instead of *pdflatex*. There are now separate configuration variables for the use of *lualatex* or *xelatex*. See *$lualatex* and *$xelatex*.)

To do a coordinated setting of all of *$dvilualatex*, *$hilatex*, *$latex*, *$pdflatex*, *$lualatex*, and *$xelatex*, see the section "Advanced Configuration".

$pdflatex_silent_switch ["-interaction=batchmode"]

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

See details of the *$latex_silent_switch* for other information that equally applies to *$pdflatex_silent_switch*.

$pdf_previewer [Operating-system dependent]

The command to invoke a pdf-previewer.

On most operating systems (MS-Windows, Linux, macOS, cygwin and msys), the default is set so that the previewer opened is the the operating system has associated with pdf files. For example, on macOS the default is "open %S".

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. By default, this is done automatically by latexmk (by whatever method is appropriate to the operating system). But sometimes this behavior is not what is needed. In that case, precede the command string by the keyword "nostart".

$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, this method works nicely with the xpdf program. With recent versions (4.05 or later) you can use the following settings:

```
$pdf_previewer = "Start xpdf -remote %R 'openFile('%S')' singlePageMode";
$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, Linux, and macOS with their default settings of *$pscmd*.

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.

$pre_tex_code ['']

Sets TeX code to be executed before inputting the source file. This works if the relevant one of *$latex*, etc contains a suitable command line with a %P or %U substitution. For example you could do

```
$latex = 'latex %O %P';
$pre_tex_code = '\AtBeginDocument{An initial message\par}';
```

To set all of *$latex, $pdflatex, $lualatex,* and *$xelatex* you could use the subroutine *alt_tex_cmds*:

```
&alt_tex_cmds;
$pre_tex_code = '\AtBeginDocument{An initial message\par}';
```

$preview_continuous_mode [0]

If nonzero, use "preview continuous mode": Compile document, run a previewer, and then regularly check for changed source files and recompile document as needed to keep the viewed file up-to-date. Equivalent to the **-pvc** option.

Which previewer (if any) is run depends on the other settings, see the command line options **-view=**, and the variable *$view*. To do a "continuous compile" without running a previewer, set *$view='none'* together with *$preview_continuous_mode=1*.

$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 the command specified in the *$lpr* variable. 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 -ww -u $ENV{USER}" under macOS, and "ps -f -u $ENV{USER}" under other operating systems (including Linux). In these specifications "$ENV{USER}" is substituted by the username.

$ps2pdf ["ps2pdf -dALLOWPSTRANSPARENCY %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 actually 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_timeout [0]
If this variable is nonzero, there will be a timeout in pvc mode after a period of inactivity. Inactivity means a period when *latexmk* has detected no file changes and hence has not taken any actions like compiling the document. The period of inactivity is in the variable $pvc_timeout_mins.

$pvc_timeout_mins [30]
The period of inactivity, in minutes, after which pvc mode times out. This is used if *$pvc_timeout* is nonzero.

$pvc_view_file_via_temporary [1]
The same as *$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 macOS) and under MS-Windows. It allows the use of filenames containing special characters, notably spaces. (But note that many versions of **latex* 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.)

$rc_report [1]
After initialization, whether to give a list of the RC files read.

$recorder [1]

Whether to use the **-recorder** option to **latex*. 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*.

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

Note about the name of the .fls file: Most implementations of **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 **latex*.

[Comment to technically savvy readers: **latex* doesn't actually need the modified search path. 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 macOS) 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, macOS, Unix collection.)

$show_time [0]

Whether to show time used, both the total and for individual steps.

Note: On MS Windows, this is clock time. On other OSs it is the CPU time used (by latexmk and the child processes it invokes). The OS-dependence is because of a limitation of Windows. If you wish to force the use of clock instead of CPU time, you can set

```
$times_are_clock = 1;
```

$silence_logfile_warnings [0]

Whether after a run of *latex to summarize warnings in the log file about undefined citations and references. Setting $silence_logfile_warnings=0 gives the summary of warnings (provided silent mode isn't also set), and this is useful to locate undefined citations and references without searching through the much more verbose log file or the screen output of *latex. But the summary can also be excessively annoying. The default is not to give these warnings. The command line options **-silence_logfile_warning_list** and **-silence_logfile_warning_list-** also set this variable.

Note that multiple occurrences for the same undefined object on the same page and same line will be compressed to a single warning.

$silent [0]

Whether to run silently. Setting $silent to 1 has the same effect as the **-quiet** of **-silent** options on the command line.

$sleep_time [2]

The time to sleep (in seconds) between checking for source-file changes when running with the **-pvc** option. If non-zero, it is subject to a minimum value give by the *$min_sleep_time* variable. But a zero value is also allowed.

A value of exactly 0 gives no delay between checks for source-file changes; it typically results in 100% CPU usage, which may not be desirable.

In old versions of latexmk, the default value of *$sleep_time* of 2 was set to give a reasonable compromise between responsiveness in -pvc mode and the amount of CPU usage. On modern computers with fast multi-core CPUs, a smaller value, e.g., 0.1 can give good results, especially when working with small documents whose compilation may take well under a second.

$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 is set to be an appropriate value for your system, as reported by the tmpdir() function in Perl's File::Spec package. For more details, you should consult the documentation for that package, as well as the related documentation for individual operating systems, notably for the Perl packages File::Spec::Unix and File::Spec::Win32. (However, note that the macOS documentation in File::Spec::Mac is many years out-of-date; the current behavior of macOS is the same as on other Unix systems.)

For use by latexmk, you can change the value of $tmpdir from the default simply by setting $tmpdir in a latexmkrc file.

$use_make_for_missing_files [0]

Whether to use *make* to try and make files that are missing after a run of **latex*, 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.

$use_parallel [0]

(Experimental: subject to change in future versions.) Whether to allow parallelism.

Two levels of parallelism are used: (a) Multiple top-level .tex files are compiled simultaneously. (b) Within each document, independent pre-primary rules (cusdep conversions, bibtex, makeindex) are run concurrently before the *latex run. Output from parallel jobs is buffered and printed atomically.

The limit on how many parallel jobs are used is given by the value of the variable *$parallel_jobs*.

$user_deleted_file_treated_as_changed [0]

Whether when testing for changed files, a user file that changes status from existing to non-existing should be regarded as changed.

The default value is 0, which implies that if a user file (as opposed to a generated file) has been deleted since the previous run, then no recompilation should be done. The reasoning is that a rerun would simply produce an error.

If the value is 1, then disappearance of a user file is treated as triggering a rerun, but only in non-preview-continuous mode.

If the value is 2, then disappearance of a user file is treated as triggering a rerun, always.

$view ["default"]

Which kind of file is to be previewed if a previewer is used. The possible values are "default", "dvi", "hnt", "ps", "pdf", "none". The value of "default" means that the "highest" of the kinds of file generated is to be used (among .dvi, .hnt, .ps and .pdf).

$warnings_as_errors [0]

Normally *latexmk* copies the behavior of *latex* in treating undefined references and citations and multiply defined references as conditions that give a warning but not an error. The variable *$warnings_as_errors* controls whether this behavior is modified.

When the variable is non-zero, *latexmk* at the end of its run will return a non-zero status code to the operating system if any of the files processed gives a warning about problems with citations or references (i.e., undefined citations or references or multiply defined references). This is **after** *latexmk* has completed all the runs it needs to try and resolve references and citations. Thus *$warnings_as_errors* being nonzero causes *latexmk* to treat such warnings as errors, but only when they occur on the last run of *latex and only after processing is complete. A non-zero value *$warnings_as_errors* can be set by the command-line option **-Werror**.

The default behavior is normally satisfactory in the usual edit-compile-edit cycle. But, for example, *latexmk* can also be used as part of a build process for some bigger project, e.g., for creating documentation in the build of a software application. Then it is often sensible to treat citation and reference warnings as errors that require the overall build process to be aborted. Of course, since multiple runs of *latex are generally needed to resolve references and citations, what matters is *not* the warnings on the first run, but the warnings on the *last* run; *latexmk* takes this into account appropriately.

In addition, a non-zero value for *$warnings_as_errors* changes the use of the commands *$failure_cmd*, *$warning_cmd*, and *$success_cmd* after a compliation. If there are citation or reference warnings, but no other errors, the behavior is as follows. If *$warning_cmd* is set, it is used. If it is not set, then then if *$warnings_as_errors* is non-zero and *$failure_cmd* is set, then *$failure_cmd*.

Otherwise *$success_cmd* is used, if it is set. (The foregoing explanation is rather complicated, because *latexmk* has to deal with the case that one or more of the commands isn't set.)

$xdv_mode [0]
If one, generate an xdv version of the document by use of xelatex.

$xdvipdfmx ["xdvipdfmx -E -o %D %O %S"]

The program to make a pdf file from an xdv file (used in conjunction with *xelatex* when *$pdf_mode*=5).

$xdvipdfmx_silent_switch ["-q"]
Switch(es) for the *xdvipdfmx* program when silent mode is on.

$xelatex ["xelatex %O %S"]
Specifies the command line for the LaTeX processing program of when the *xelatex* program is called for. See the documentation of the **-xelatex** option for some special properties of *latexmk*'s use of *xelatex*.

Note about xelatex: *latexmk* uses *xelatex* to make an .xdv rather than .pdf file, with the .pdf file being created in a separate step. This is enforced by the use of the **-no-pdf** option. If %O is part of the command for invoking *xelatex*, then *latexmk* will insert the **-no-pdf** option automatically, otherwise you must provide the option yourself. See the documentation for the **-pdfxe** option for why *latexmk* makes a .xdv file rather than a .pdf file when *xelatex* is used.

To do a coordinated setting of all of *$dvilualatex*, *$hilatex*, *$latex*, *$pdflatex*, *$lualatex*, and *$xelatex*, see the section "Advanced Configuration".

$xelatex_silent_switch ["-interaction=batchmode"]
Switch(es) for the *xelatex* program (specified in the variable *$xelatex*) when silent mode is on.

See details of the *$latex_silent_switch* for other information that equally applies to *$xelatex_silent_switch*.

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.

Defining a custom dependency:
The old method of configuring *latexmk* to use a custom dependency was to directly manipulate the *@cus_dep_list* array that contains information defining the custom dependencies. (See the section "Old Method of Defining Custom Dependencies" for details.) This method still works, but is no longer preferred.

A better method is to use the 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 arguments are as follows:

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. Generally, the appropriate value of *must* is zero.

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.

Naturally *add_cus_dep* adds a custom dependency with the specified from and to extensions. If a custom dependency has been previously defined (e.g., in an rcfile that was read earlier), then it is replaced by the new one.

The subroutine *remove_cus_dep* removes the specified custom dependency. The subroutine *show_cus_dep* causes a list of the currently defined custom dependencies to be sent to the screen output.

How custom dependencies are used:

An instance of a custom dependency rule is created whenever *latexmk* detects that a run of **latex* 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. If it does, then a corresponding instance of the custom dependency is created, after which the rule is invoked 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 item in the specification of each custom-dependency rule, labeled "must" above, specifies how the rule should be applied when the source file fails to exist.

When *latex* reports that an input file (e.g., a graphics file) does not exist, *latexmk* tries to find a source file and a custom dependency that can be used to make it. If it succeeds, then it creates an instance of the custom dependency and invokes it to make the missing file, after which the next pass of *latex* etc will be able to read the newly created file.

Note for advanced usage: The operating system's environment variable TEXINPUTS can be used to specify a search path for finding files by *latex* etc. Correspondingly, when a missing file is reported, *latexmk* looks in the directories specified in TEXINPUTS as well as in the current directory, to find a source file from which an instance of a custom dependency can be used to make the missing file.

Function to implement custom dependency, traditional method:

The function that implements a custom dependency gets the information on the files to be processed in two ways. The first is through its one argument; the argument contains the base name of the source and

destination files. The second way is described later.

A simple and typical example of code in an initialization rcfile using the first method 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*, *lualatex* or *xelatex* 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.)

Note 3: The subroutines implementing custom dependencies in the examples given just have a single line invoking an external program. That's the usual situation. But since the subroutines are in the Perl language, you can implement much more complicated processing if you need it.

Removing custom dependencies, and when you might need to do this:

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. A situation where this would be desirable is where there are multiple custom dependencies with the same from-extension or the same to-extension. In that case, *latexmk* might choose a different one from the one you want for a specific project. As an example, to remove any "fig" to "eps" rule you would use:

```
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.

Function implementing custom dependency, alternative methods:

So far the examples for functions to implement custom dependencies have used the argument of the function to specify the base name of converted file. This method has been available since very old versions of *latexmk*, and many examples can be found, e.g., on the web.

However in later versions of *latexmk* the internal structure of the implementation of its "rules" for the steps of processing, including custom dependencies, became much more powerful. The function implementing a custom dependency is executed within a special context where a number of extra variables and subroutines are defined. Publicly documented ones, intended to be long-term stable, are listed below, under the heading "Variables and subroutines for processing a rule".

Examples of their use is given in the following examples, concerning multiple index files and glossaries.

The only index-file conversion built-in to *latexmk* is from an ".idx" file written on one run of **latex* to an ".ind" file to be read in on a subsequent run. But with the index.sty package, for example, 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 get *latexmk* to convert files with the extension *.ndx* to *.nnd*. The most elementary method is to define a custom dependency as follows:

```
add_cus_dep( 'ndx', 'nnd', 0, 'ndx2nnd' );
sub ndx2nnd {
   return system( "makeindex -o \"$_[0].nnd\" \"$_[0].ndx\"" );
}
push @generated_exts, 'ndx', 'nnd';
```

Notice the added line compared with earlier examples. The extra line gets the extensions "ndx" and "nnd" added to the list of extensions for generated files; then the extra index files will be deleted by clean-up operations

But if you have yet more indexes with yet different extensions, e.g., "adx" and "and", then you will need a separate function for each pair of extensions. This is quite annoying. You can use the *Run_subst* function to simplify the definitions to use a single function:

```
add_cus_dep( 'ndx', 'nnd', 0, 'dx2nd' );
add_cus_dep( 'adx', 'and', 0, 'dx2nd' );
sub dx2nd {
```

```
    return Run_subst( "makeindex -o %D %S" );
}
push @generated_exts, 'ndx', 'nnd', 'adx', 'and';
```

You could also instead use

```
add_cus_dep( 'ndx', 'nnd', 0, 'dx2nd' );
add_cus_dep( 'adx', 'and', 0, 'dx2nd' );
sub dx2nd {
    return Run_subst( $makeindex );
}
push @generated_exts, 'ndx', 'nnd', 'adx', 'and';
```

This last example uses the command specification in *$makeindex*, and so any customization you have made for the standard index also applies to your extra indexes.

Similar techniques can be applied for glossaries.

Those of you with experience with Makefiles, may get concerned that the *.ndx* file is written during a run of **latex* 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 is one of the issues that *latexmk* is programmed to overcome. It examines the contents of the files (by use of a 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 the *.aux* file, etc) that changes on each new run, then you will have a problem. For real experts: See the *%hash_calc_ignore_pattern* if you have to deal with such problems.

Old Method of Defining Custom Dependencies:

In much older 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 {
    return system( "fig2dev -Lps \"$_[0].fig\" \"$_[0].eps\"" );
}
```

This method still works, and is almost equivalent to the code given earlier that used the *add_cus_dep* subroutine. However, the old method doesn't delete any previous custom-dependency for the same conversion. So the new method is preferable.

ADVANCED CONFIGURATION: SOME EXTRA RESOURCES AND ADVANCED TRICKS

For most purposes, simple configuration for *latexmk* along the lines of the examples given is sufficient. But sometimes you need something harder. In this section, I indicate some extra possibilities. Generally to use these, you need to be fluent in the Perl language, since this is what is used in the rc files.

In this section, I include first, a description of a number of variables and subroutines that provide, among other things, access to latexmk's internal data structures for handling dependencies. Then I describe the hook mechanism whereby at certain points in the processing, latexmk can call user-defined subroutines.

See also the section DEALING WITH ERRORS, PROBLEMS, ETC. See also the examples in the directory *example_rcfiles* in the *latexmk* distributions. Even if none of the examples apply to your case, they may give you useful ideas

Variables and subroutines for processing a rule

A step in the processing is called a rule. One possibility to implement the processing of a rule is by a Perl subroutine. This is always the case for custom dependencies. Also, for any other rule, you can use a subroutine by prefixing the command specification by the word "internal" -- see the section FORMAT OF COMMAND SPECIFICATIONS.

When you use a subroutine for processing a rule, all the possibilities of Perl programming are available, of course. In addition, some of *latexmk*'s internal variables and subroutines are available. The ones listed below are intended to be available to (advanced) users, and their specifications will generally have stability under upgrades. Generally, the variables should be treated as read-only: Changing their values can have bad consequences, since it is liable to mess up the consistency of what *latexmk* is doing.

$compile_status

This variable is set by *latexmk* according to the status of the overall compilation of a document. Its possible values are:

-2: before any compilation,
-1: compilation in progress,
0: last compilation was successful,
1: last compilation gave warnings but no errors, and warnings_as_errors was *not* set,
2: last compilation gave warnings and no errors, but warnings_as_errors was set,
3: last compilation gave errors.

This variable is useful for a subroutine run used in the hook class compile_end, which is invoked just after compilation of a document has been completed. In this case, the values of the $compile_status variable will be in the range 0 to 3, and the subroutine can adjust its behavior according to this value.

$rule This variable has the name of the rule, as known to *latexmk*. Note that the exact contents of this variable for a given rule may be dependent on the version of *latexmk*

$$Pbase

This gives the basename for the rule. Generally, it determines the names of generated files. E.g., for a run of *latex, the name of the .log file is the aux directory concatenated with the basename and then '.log'.

For a *latex rule, the basename is without a directory component. For other rules, it includes the directory component (if any is used).

This (annoying) difference is associated with the different ways in which the commands invoked by latexmk work when the command line includes a name for a source file that includes a directory component. For the *latex commands, the directory of the source file is irrelevant to the directory

component the generated files, which instead is determined by the values in the -aux-directory and/or -output-directory options.

In contrast, many other programs (e.g., biber, bibtex) put their generated files in the same directory as the source file, merely with a changed extension.

Note the double dollar signs: In Perl terms, the variable *$Pbase* is a *reference* to a variable that contains the basename. The second dollar sign derefences the reference to give the actual value. (A reference is is used rather like a pointer, and the 'P' (for 'pointer') at the start of the variable name is a convention used in latexmk to indicate that the variable is a reference variable.)

$$Pdest

This gives the name of the main output file if any. Note the double dollar signs.

$$Psource

This gives the name of the primary source file. Note the double dollar signs.

add_hook(<stack_name>, <subroutine>)

See the section 'Hooks' for more details.

This adds the subroutine specified in the second argument to latexmk's stack of hooks specified by the stack name. It returns 1 on success, and zero otherwise (e.g., if the specified hook stack doesn't exist).

The subroutine can be specified by a reference to the subroutine, as in

```
add_hook( 'after_xlatex_analysis', mmz_analyze )
```

Given that the subroutine mmz_analyze has been defined in the rc file.

The subroutine can be specified by a string whose value is the name of the subroutine, e.g.,

```
add_hook( 'after_xlatex_analysis', 'mmz_analyze' )
```

In simple cases, the subroutine can be an anonymous subroutine defined in the call to add_hooks,

```
add_hook( 'after_main_pdf', sub{ print "TEST\n"; return 0; } );
```

Observe that on success, the subroutine should return 0 (like a call to Perl's system subroutine), so normally this should be coded explicitly. If a hook subroutine returns a non-zero value, latexmk treats that as an error condition.

ensure_path(var, values ...)

The first parameter is the name of one of the system's environment variables for search paths. The remaining parameters are values that should be in the variable. For each of the value parameters, if it isn't already in the variable, then it is prepended to the variable; in that case the environment variable is created if it doesn't already exist. For separating values, the character appropriate the the operating system is used -- see the configuration variable *$search_path_separator*.

Example:

```
ensure_path( 'TEXINPUTS', './custom_cls_sty_files//' );
```

(In this example, the trailing '//' is documented by TeX systems to mean that **latex* search for files in the specified directory and in all subdirectories.)

Technically *ensure_path* works by setting Perl's variable *$ENV{var}*, where *var* is the name of the target variable. The changed value is then passed as an environment variable to any invoked programs.

pushd(path), popd()

These subroutines are used when it is needed to temporarily change the working directory, as in

```
pushd( 'some_directory' );
... Processing done with 'some_directory' as the working directory
popd()
```

They perform exactly the same function as the commands of the same names in operating system command shells like bash on Unix, and cmd.exe on the Windows.

rdb_add_generated(file, ...)

This subroutine is to be used in the context of a rule, that is, from within a subroutine that is carrying out processing of a rule. Such is the case for the subroutine implementing a custom dependency, or the subroutine invoked by using the "internal" keyword in the command specification like that in the variable *$latex*.

Its arguments are a sequence of filenames which are generated during the running of the rule. The names might arise from an analysis of the results of the run, e.g., in a log file, or from knowledge of properties of the specific rule. Calling *rdb_add_generated* with these filenames ensures that these files are flagged as generated by the rule in *latexmk*'s internal data structures. Basically, no action is taken if the files have already been flagged as generated.

A main purpose of using this subroutine is for the situation when a generated file is also the source file for some rule, so that *latexmk* can correctly link the dependency information in its network of rules.

Note: Unlike some other subroutines in this section, there is no argument for a rule for rdb_add_generated*. Instead, the subroutine is to be invoked during the processing of a rule when* latexmk *has set up an appropriate context (i.e., appropriate variables). In contrast, subroutines with a rule argument can be used also outside a rule context.*

rdb_ensure_file($rule, file)

This subroutine ensures that the given file is among the source files for the specified rule. It is typically used when, during the processing of a rule, it is known that a particular extra file is among the dependencies that *latexmk* should know, but its default methods don't find the dependency. Almost always the first argument is the name of the rule currently being processed, so it is then appropriate to specify it by *$rule*.

For examples of its use, see some of the files in the directory *example_rcfiles* of *latexmk*'s distribution. Currently the cases that use this subroutine are *bib2gls-latexmkrc*, *exceltex_latexmkrc* and *texinfo-latexmkrc*. These illustrate typical cases where *latexmk*'s normal processing fails to detect

certain extra source files.

Note that rdb_ensure_file *only has one filename argument, unlike other subroutines in this section. If you want to apply its action to multiple files, you will need one call to* rdb_ensure_file *for each file.*

rdb_ensure_files_here(file, ...)
Like subroutine *rdb_ensure_files*, except that (a) it assumes the context is of a rule, and the files are to be added to the source list for that rule; (b) multiple files are allowed.

rdb_remove_files($rule, file, ...)
This subroutine removes one or more files from the dependency list for the given rule.

rdb_remove_generated(file, ...)
This subroutine is to be used in the context of a rule, that is, from within a subroutine that is carrying out processing of a rule. It performs the opposite action to *rdb_add_generated*. Its effect is to ensure that the given filenames are not listed in *latexmk*'s internal data structures as being generated by the rule.

rdb_list_source($rule)
This subroutine returns the list of source files (i.e., the dependency list) for the given rule.

rdb_set_source($rule, file, ...)

rdb_set_source($rule, @files)
This subroutine sets the dependency list for the given rule to be the specified files. Files that are already in the list have unchanged information. Files that were not in the list are added to it. Files in the previous dependency list that are not in the newly specified list of files are removed from the dependency list.

Run_subst(command_spec)
This subroutine runs the command specified by *command_spec*. The specification is a string in the format listed in the section "Format of Command Specifications". An important action of the *Run_subst* is to make substitutions of placeholders, e.g., %S and %D for source and destination files; these get substituted before the command is run. In addition, the command after substitution is printed to the screen unless *latexmk* is running in silent mode.

test_gen_file_time (<file>)
This subroutine is used in the context of a rule. It returns true or false according to whether or not a file of the given name both exists and was generated in the latest run of the rule. If the subroutine returns false, but the file exists, then the file is a leftover from a previous run.

The test for a file being generated on the current run is whether the modification time of the file is at least as recent as the time that the run of the rule was started. An allowance for the granularity of the values of modification time on file systems is made. See the description of the variable **$filetime_causality_threshold** for details.

In addition, latexmk makes allowance for the possiblity that files are hosted on a different computer than that running latexmk and that the system clock times on the two computers are mismatched. Latexmk automatically detects (and reports) any significant mismatch and corrects for it.

Coordinated Setting of Commands for *latex

To set all of *$dvilualatex*, *$hilatex*, *$latex*, *$pdflatex*, *$lualatex*, and *$xelatex* to a common pattern, you can use one of the following subroutines, std_tex_cmds, alt_tex_cmds, and set_tex_cmds.

To get the standard commands, use

```
&std_tex_cmds;
```

This results in *$latex = 'latex %O %S'*, and similarly for *$dvilualatex*, *$hilatex*, *$pdflatex*, *$lualatex*, and *$xelatex*. Note the ampersand in the invocation; this indicates to Perl that a subroutine is being called. (The use of this subroutine enables you to override previous redefinitions of the *$latex*, etc variables, which might have occurred in an earlier-read rc file.)

To be able to use the string provided by the -pretex option (if any), you can use

```
&alt_tex_cmds;
```

This results in *$latex = 'latex %O %P'*, etc. Again note the ampersand in the invocation; this indicates to Perl that a subroutine is being called.

A more general way of specifying the variables is using

```
set_tex_cmds( 'CMD_SPEC' );
```

Here CMD_SPEC is the command line without the program name. This results in *$latex = 'latex CMD_SPEC'*, and similarly for *$pdflatex*, etc. (An ampersand preceding the subroutine name is not necessary here, since the parentheses show Perl that a subroutine is being invoked.)

An example that provides the *--interaction=batchmode* option to the *latex commands would be

```
set_tex_cmds( '--interaction=batchmode %O %S' );
```

This results in *$latex = 'latex --interaction=batchmode %O %S '*, etc. Note that when '%O' appears after the added option, as here, options provided on the command line to *latexmk* can override the supplied one.

A more general command line can be set up by using the placeholder '%C' in CMD_SPEC. The '%C' is substituted by the basic name of the command, i.e., whichever of 'latex', 'pdflatex', etc is appropriate. (More than one occurrence of '%C' is allowed.) For example to use the development/pre-release versions of latex, etc, which have names, 'latex-dev', 'pdflatex-dev', etc, you could use

```
set_tex_cmds( '%C-dev %O %S' );
```

This results in *$latex = 'latex-dev %O %S'*, etc. (The pre-release programs latex-dev etc are provided in current distributions of TeXLive and MiKTeX.)

Hooks

Latexmk provides a way to arrange for user-defined subroutines to be called at certain points in the processing. These can be used to configure appropriate behavior and actions beyond latexmk's normal behavior. For a good example of how they can be used to accommodate latexmk's behavior to particular packages, see the file memoize_latexmkrc in the example_rcfiles subdirectory of the latexmk distribution. (In a standard TeXLive installation, that subdirectory is to be found in texmf-dist/doc/support/latexmk/)

The hook mechanism is complementary to the method of redefining command strings like **$pdflatex** etc. The

two methods have overlapping domains of usefulness.

Note that the hook mechanism is newly made public in v. 4.84 of latexmk. It is subject to change and improvement as experience is gained.

The hooks are arranged in named hook stacks, and a hook subroutine is added to a given stack by latexmk's add_hook subroutine (documented above). The currently available stacks are as follows, listed in the approximate order in which they are encountered in processing a document:

compile_begin
The subroutines in this hook stack are called just before the start of processing of a document, at the same place that the *$compiling_cmd* is invoked.

before_xlatex
The subroutines in this hook stack are called just before a **latex* program is run.

after_xlatex
The subroutines in this hook stack are called after a *latex program is run. Before the subroutines are called, *latexmk* has done some immediate postprocessing, e.g., to move the generated .pdf file from the aux directory to the output directory when *$emulate_aux* is set to 1.

after_xlatex_analysis
The subroutines in this hook stack are called after latexmk has done its dependency analysis after a **latex* programs is run. Subroutines in this stack provide a useful way of adding items to the dependency information associated with particular packages and that *latexmk* doesn't automatically deal with.

after_main_pdf
The subroutines in this hook stack are called after one of the rules that creates the document's .pdf file. (This covers any of pdflatex, lualatex, dvipdf, ps2pdf, xdvipdfmx.)

compile_end
The subroutines in this hook stack are called just after processing of a document is completed, independently of the success or failure of the processing. These subroutines are called at the same place that commands specified in *$warning_cmd*, *$failure_cmd*, and *$success_cmd* are invoked, but without the invocation being conditional on whether the compilation succeeded or not.

Instead a subroutine used for this hook can adjust its behavior depending on the value of a global variable *$compile_status*. See the documentation for that variable for details.

cleanup
The subroutines in this hook stack are called whenever latexmk is about to do a cleanup operation. They can be used, for example, to tailor the deleted files to the pecularities of particular packages when latexmk's general mechanisms for specifying files to be deleted are too inflexible.

These subroutines are called before latexmk does any of its own file deletion; thus the hook subroutines have access to all the generated files that give package-specific information.

cleanup_extra_full
The subroutines in this hook stack are called in addition to the ones in the cleanup stack, whenever a full cleanup operation is to be done (i.e., one that includes the pdf, ps etc files). They are called immediately

after those in the cleanup stack, but still before latexmk does any of its own file deletion.

(Any other stacks defined in latexmk.pl but not listed above are to be regarded as experimental and subject to change.)

Each subroutine should return 0 on success and a non-zero value on failure. This matches the convention used for running programs, e.g., by Perl's system subroutine, and the matching convention used for subroutines for custom dependencies in latexmk.

For most of the hook stacks, the subroutines are called in the context of a rule, with variables like **$rule** defined. However, some hook stacks, like the cleanup ones, are called from outside any rule; and latexmk adjusts the relevant variables to refer to the overall task (i.e., of processing a particular main .tex file).

Advanced configuration: 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 macOS 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*.

Escaping of characters in dependency lists: There are certain special characters that need to be escaped when names of files and directories containing them appear in a dependency list used by a make program. Generally, such special characters are best avoided.

By default, latexmk does no escaping of this kind, and the user will have to arrange to deal with the issue separately, if the relevant special characters are used. Note that the rules for escaping depend on which make program is used, and on its version.

One special case is of spaces, since those are particularly prevalent, notably in standard choices of name for a user's home directory. So latexmk does provide an option to escape spaces. See the option **-deps_escape=...** and the variable **$deps_escape** for details.

NON_ASCII CHARACTERS IN FILENAMES, RC FILES, ETC

Modern operating systems and file systems allow non-ASCII characters in the names of files and directories that encompass the full Unicode range. Mostly, *latexmk* deals with these correctly. However, there are some situations in which there are problems, notably on Microsoft Windows. Prior to version 4.77, *latexmk* had problems with non-ASCII filenames on Windows, even though there were no corresponding problems on macOS and Linux. These problems are corrected in the present version.

DETAILS TO BE FILLED IN

SEE ALSO

latex(1), bibtex(1), lualatex(1), pdflatex(1), xelatex(1).

BUGS (SELECTED)

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 too easily.)

AUTHOR

Current version, by John Collins (Version 4.90). Report bugs etc to his e-mail (jcc8 at psu.edu).

Released version can be obtained from CTAN: <https://www.ctan.org/pkg/latexmk/>, and from the author's website <https://www.cantab.net/users/johncollins/latexmk/>.
Modifications and enhancements by Evan McLean (Version 2.0)
Original script called "go" by David J. Musliner (RCS Version 3.2)