NAME

watchexec − Execute commands when watched files change

SYNOPSIS

watchexec [**−w**|**−−watch**] [**−c**|**−−clear**] [**−o**|**−−on−busy−update**] [**−r**|**−−restart**] [**−s**|**−−signal**] [**−k**|**−−kill**] [**−−stop−signal**] [**−−stop−timeout**] [**−−debounce**] [**−−stdin−quit**] [**−−no−vcs−ignore**] [**−−no−project−ignore**] [**−−no−global−ignore**] [**−−no−default−ignore**] [**−p**|**−−postpone**] [**−−delay−run**] [**−−poll**] [**−−shell**] [**−n**] [**−−no−environment**] [**−−emit−events−to**] [**−E**|**−−env**] [**−−no−process−group**] [**−N**|**−−notify**] [**−−project−origin**] [**−−workdir**] [**−e**|**−−exts**] [**−f**|**−−filter**] [**−−filter−file**] [**−i**|**−−ignore**] [**−−ignore−file**] [**−−fs−events**] [**−−no−meta**] [**−−print−events**] [**−v**|**−−verbose**]... [**−−log−file**] [**−−man−page**] [**−−completions**] [**−h**|**−−help**] [**−V**|**−−version**] [*COMMAND*]

DESCRIPTION

Execute commands when watched files change.

Recursively monitors the current directory for changes, executing the command when a filesystem change is detected (among other event sources). By default, watchexec uses efficient kernel−level mechanisms to watch for changes.

At startup, the specified <COMMAND> is run once, and watchexec begins monitoring for changes.

Examples:

Rebuild a project when source files change:

$ watchexec make

Watch all HTML, CSS, and JavaScript files for changes:

$ watchexec −e html,css,js make

Run tests when source files change, clearing the screen each time:

$ watchexec −c make test

Launch and restart a node.js server:

$ watchexec −r node app.js

Watch lib and src directories for changes, rebuilding each time:

$ watchexec −w lib −w src make

OPTIONS

−w, **−−watch**=*PATH*

Watch a specific file or directory

By default, Watchexec watches the current directory.

When watching a single file, it's often better to watch the containing directory instead, and filter on the filename. Some editors may replace the file with a new one when saving, and some platforms may not detect that or further changes.

Upon starting, Watchexec resolves a "project origin" from the watched paths. See the help for '−−project−origin' for more information.

This option can be specified multiple times to watch multiple files or directories.

−c, **−−clear**=*MODE*

Clear screen before running command

If this doesn't completely clear the screen, try '−−clear=reset'.

−o, **−−on−busy−update**=*MODE*
What to do when receiving events while the command is running

Default is to **queue** up events and run the command once again when the previous run has finished. You can also use **do−nothing**, which ignores events while the command is running and may be useful to avoid spurious changes made by that command, or **restart**, which terminates the running command and starts a new one. Finally, there's **signal**, which only sends a signal; this can be useful with programs that can reload their configuration without a full restart.

The signal can be specified with the '−−signal' option.

Note that this option is scheduled to change its default to **do−nothing** in the next major release. File an issue if you have any concerns.

−r, **−−restart**
Restart the process if it's still running

This is a shorthand for '−−on−busy−update=restart'.

−s, **−−signal**=*SIGNAL*
Send a signal to the process when it's still running

Specify a signal to send to the process when it's still running. This implies '−−on−busy−update=signal'; otherwise the signal used when that mode is **restart** is controlled by '−−stop−signal'.

See the long documentation for '−−stop−signal' for syntax.

−k, **−−kill**
Hidden legacy shorthand for '−−signal=kill'

−−stop−signal=*SIGNAL*
Signal to send to stop the command

This is used by **restart** and **signal** modes of '−−on−busy−update' (unless '−−signal' is provided). The restart behaviour is to send the signal, wait for the command to exit, and if it hasn't exited after some time (see '−−timeout−stop'), forcefully terminate it.

The default on unix is **SIGTERM**.

Input is parsed as a full signal name (like "SIGTERM"), a short signal name (like "TERM"), or a signal number (like "15"). All input is case−insensitive.

On Windows this option is technically supported but only supports the "KILL" event, as Watchexec cannot yet deliver other events. Windows doesn't have signals as such; instead it has termination (here called "KILL" or "STOP") and "CTRL+C", "CTRL+BREAK", and "CTRL+CLOSE" events. For portability the unix signals "SIGKILL", "SIGINT", "SIGTERM", and "SIGHUP" are respectively mapped to these.

−−stop−timeout=*TIMEOUT*
Time to wait for the command to exit gracefully

This is used by the **restart** mode of '−−on−busy−update'. After the graceful stop signal is sent, Watchexec will wait for the command to exit. If it hasn't exited after this time, it is forcefully terminated.

Takes a unit−less value in seconds, or a time span value such as "5min 20s".

The default is 60 seconds. Set to 0 to immediately force−kill the command.

−−debounce=*TIMEOUT*
Time to wait for new events before taking action

When an event is received, Watchexec will wait for up to this amount of time before handling it (such as running the command). This is essential as what you might perceive as a single change may actually emit many events, and without this behaviour, Watchexec would run much too often. Additionally, it's not infrequent that file writes are not atomic, and each write may emit an event, so this is a good way to avoid running a command while a file is partially written.

An alternative use is to set a high value (like "30min" or longer), to save power or bandwidth on intensive tasks, like an ad−hoc backup script. In those use cases, note that every accumulated event will build up in memory.

Takes a unit−less value in milliseconds, or a time span value such as "5sec 20ms".

The default is 50 milliseconds. Setting to 0 is highly discouraged.

−−stdin−quit
Exit when stdin closes

This watches the stdin file descriptor for EOF, and exits Watchexec gracefully when it is closed. This is used by some process managers to avoid leaving zombie processes around.

−−no−vcs−ignore
Don't load gitignores

Among other VCS exclude files, like for Mercurial, Subversion, Bazaar, DARCS, Fossil. Note that Watchexec will detect which of these is in use, if any, and only load the relevant files. Both global (like '~/.gitignore') and local (like '.gitignore') files are considered.

This option is useful if you want to watch files that are ignored by Git.

−−no−project−ignore
Don't load project−local ignores

This disables loading of project−local ignore files, like '.gitignore' or '.ignore' in the watched project. This is contrasted with '−−no−vcs−ignore', which disables loading of Git and other VCS ignore files, and with '−−no−global−ignore', which disables loading of global or user ignore files, like '~/.gitignore' or '~/.config/watchexec/ignore'.

Supported project ignore files:

− Git: .gitignore at project root and child directories, .git/info/exclude, and the file pointed to by ‘core.excludesFile‘ in .git/config. − Mercurial: .hgignore at project root and child directories. − Bazaar: .bzrignore at project root. − Darcs: _darcs/prefs/boring − Fossil: .fossil−settings/ignore−glob − Ripgrep/Watchexec/generic: .ignore at project root and child directories.

VCS ignore files (Git, Mercurial, Bazaar, Darcs, Fossil) are only used if the corresponding VCS is discovered to be in use for the project/origin. For example, a .bzrignore in a Git repository will be discarded.

Note that this was previously called '−−no−ignore', but that's now deprecated and its use is discouraged, as it may be repurposed in the future.

−−no−global−ignore
Don't load global ignores

This disables loading of global or user ignore files, like '~/.gitignore', '~/.config/watchexec/ignore', or '%APPDATA%\Bazzar\2.0\ignore'. Contrast with '−−no−vcs−ignore' and '−−no−project−ignore'.

Supported global ignore files

− Git (if core.excludesFile is set): the file at that path − Git (otherwise): the first found of $XDG_CONFIG_HOME/git/ignore, %APPDATA%/.gitignore, %USERPROFILE%/.gitignore, $HOME/.config/git/ignore, $HOME/.gitignore. − Bazaar: the first found of %APPDATA%/Bazzar/2.0/ignore, $HOME/.bazaar/ignore. − Watchexec: the first found of $XDG_CONFIG_HOME/watchexec/ignore, %APPDATA%/watchexec/ignore, %USERPROFILE%/.watchexec/ignore, $HOME/.watchexec/ignore.

Like for project files, Git and Bazaar global files will only be used for the corresponding VCS as used in the project.

−−no−default−ignore
Don't use internal default ignores

Watchexec has a set of default ignore patterns, such as editor swap files, ‘*.pyc‘, ‘*.pyo‘, ‘.DS_Store‘, ‘.bzr‘, ‘_darcs‘, ‘.fossil−settings‘, ‘.git‘, ‘.hg‘, ‘.pijul‘, ‘.svn‘, and Watchexec log files.

−p, **−−postpone**
Wait until first change before running command

By default, Watchexec will run the command once immediately. With this option, it will instead wait until an event is detected before running the command as normal.

−−delay−run=*DURATION*
Sleep before running the command

This option will cause Watchexec to sleep for the specified amount of time before running the command, after an event is detected. This is like using "sleep 5 && command" in a shell, but portable and slightly more efficient.

Takes a unit−less value in seconds, or a time span value such as "2min 5s".

−−poll=*INTERVAL*
Poll for filesystem changes

By default, and where available, Watchexec uses the operating system's native file system watching capabilities. This option disables that and instead uses a polling mechanism, which is less efficient but can work around issues with some file systems (like network shares) or edge cases.

Optionally takes a unit−less value in milliseconds, or a time span value such as "2s 500ms", to use as the polling interval. If not specified, the default is 30 seconds.

Aliased as '−−force−poll'.

−−shell=*SHELL*
Use a different shell

By default, Watchexec will use 'sh' on unix and 'cmd' (CMD.EXE) on Windows. With this, you

can override that and use a different shell, for example one with more features or one which has your custom aliases and functions.

If the value has spaces, it is parsed as a command line, and the first word used as the shell program, with the rest as arguments to the shell.

The command is run with the '−c' flag (except for 'cmd' and 'powershell' on Windows, where the '/C' option is used).

Note that the default shell will change at the next major release: the value of '$SHELL' will be respected, falling back to 'sh' on unix and to PowerShell on Windows.

The special value 'none' can be used to disable shell use entirely. In that case, the command provided to Watchexec will be parsed, with the first word being the executable and the rest being the arguments, and executed directly. Note that this parsing is rudimentary, and may not work as expected in all cases.

Using 'none' is a little more efficient and can enable a stricter interpretation of the input, but it also means that you can't use shell features like globbing, redirection, or pipes.

Examples:

Use without shell:

$ watchexec −n −− zsh −x −o shwordsplit scr

Use with powershell:

$ watchexec −−shell=powershell −− test−connection localhost

Use with cmd:

$ watchexec −−shell=cmd −− dir

Use with a different unix shell:

$ watchexec −−shell=bash −− 'echo $BASH_VERSION'

Use with a unix shell and options:

$ watchexec −−shell='zsh −x −o shwordsplit' −− scr

−n Don't use a shell

This is a shorthand for '−−shell=none'.

−−no−environment
Shorthand for '−−emit−events=none'

This is the old way to disable event emission into the environment. See '−−emit−events' for more.

−−emit−events−to=*MODE*
Configure event emission

Watchexec emits event information when running a command, which can be used by the command to target specific changed files.

One thing to take care of is assuming inherent behaviour where there is only chance. Notably, it could appear as if the ‘RENAMED‘ variable contains both the original and the new path being renamed. In previous versions, it would even appear on some platforms as if the original always came before the new. However, none of this was true. It's impossible to reliably and portably know which changed path is the old or new, "half" renames may appear (only the original, only the new), "unknown" renames may appear (change was a rename, but whether it was the old or new isn't known), rename events might split across two debouncing boundaries, and so on.

This option controls where that information is emitted. It defaults to 'environment', which sets environment variables with the paths of the affected files, for filesystem events:

$WATCHEXEC_COMMON_PATH is set to the longest common path of all of the below variables, and so should be prepended to each path to obtain the full/real path. Then:

− $WATCHEXEC_CREATED_PATH is set when files/folders were created − $WATCHEXEC_REMOVED_PATH is set when files/folders were removed − $WATCHEXEC_RENAMED_PATH is set when files/folders were renamed − $WATCHEXEC_WRITTEN_PATH is set when files/folders were modified − $WATCHEXEC_META_CHANGED_PATH is set when files/folders' metadata were modified − $WATCHEXEC_OTHERWISE_CHANGED_PATH is set for every other kind of pathed event

Multiple paths are separated by the system path separator, ';' on Windows and ':' on unix. Within each variable, paths are deduplicated and sorted in binary order (i.e. neither Unicode nor locale aware).

This is the legacy mode and will be deprecated and removed in the future. The environment of a process is a very restricted space, while also limited in what it can usefully represent. Large numbers of files will either cause the environment to be truncated, or may error or crash the process entirely.

Two new modes are available: 'stdin' writes absolute paths to the stdin of the command, one per line, each prefixed with ‘create:‘, ‘remove:‘, ‘rename:‘, ‘modify:‘, or ‘other:‘, then closes the handle; 'file' writes the same thing to a temporary file, and its path is given with the $WATCHEXEC_EVENTS_FILE environment variable.

There are also two JSON modes, which are based on JSON objects and can represent the full set of events Watchexec handles. Here's an example of a folder being created on Linux:

```json
{
 "tags": [
  {
   "kind": "path",
   "absolute": "/home/user/your/new−folder",
   "filetype": "dir"
  },
  {
   "kind": "fs",
   "simple": "create",
   "full": "Create(Folder)"
  },
  {
   "kind": "source",
   "source": "filesystem",
  }
```

```
  ],
  "metadata": {
    "notify−backend": "inotify"
  } } ‘‘‘
```

The fields are as follows:

− ‘tags‘, structured event data. − ‘tags[].kind‘, which can be:
* 'path', along with:
  + ‘absolute‘, an absolute path.
  + ‘filetype‘, a file type if known ('dir', 'file', 'symlink', 'other').
* 'fs':
  + ‘simple‘, the "simple" event type ('access', 'create', 'modify', 'remove', or 'other').
  + ‘full‘, the "full" event type, which is too complex to fully describe here, but looks like 'General(Precise(Specific))'.
* 'source', along with:
  + ‘source‘, the source of the event ('filesystem', 'keyboard', 'mouse', 'os', 'time', 'internal').
* 'keyboard', along with:
  + ‘keycode‘. Currently only the value 'eof' is supported.
* 'process', for events caused by processes:
  + ‘pid‘, the process ID.
* 'signal', for signals sent to Watchexec:
  + ‘name‘, the normalised signal name ('hangup', 'interrupt', 'quit', 'terminate', 'user1', 'user2').
* 'completion', for when a command ends:
  + ‘disposition‘, the exit disposition ('success', 'error', 'signal', 'stop', 'exception', 'continued').
  + ‘code‘, the exit, stop, or exception code.
  + ‘signal‘, the signal name or number if the exit was caused by a signal.

The 'json−stdin' mode will emit JSON events to the standard input of the command, one per line, then close stdin. The 'json−file' mode will create a temporary file, write the events to it, and provide the path to the file with the $WATCHEXEC_EVENTS_FILE environment variable.

Finally, the special 'none' mode will disable event emission entirely.

−E, **−−env**=*KEY=VALUE*
Add env vars to the command

This is a convenience option for setting environment variables for the command, without setting them for the Watchexec process itself.

Use key=value syntax. Multiple variables can be set by repeating the option.

−−no−process−group
Don't use a process group

By default, Watchexec will run the command in a process group, so that signals and terminations are sent to all processes in the group. Sometimes that's not what you want, and you can disable the behaviour with this option.

−N, **−−notify**
Alert when commands start and end

With this, Watchexec will emit a desktop notification when a command starts and ends, on supported platforms. On unsupported platforms, it may silently do nothing, or log a warning.

−−project−origin=*DIRECTORY*
Set the project origin

Watchexec will attempt to discover the project's "origin" (or "root") by searching for a variety of markers, like files or directory patterns. It does its best but sometimes gets it it wrong, and you can override that with this option.

The project origin is used to determine the path of certain ignore files, which VCS is being used, the meaning of a leading '/' in filtering patterns, and maybe more in the future.

When set, Watchexec will also not bother searching, which can be significantly faster.

−−workdir=*DIRECTORY*
Set the working directory

By default, the working directory of the command is the working directory of Watchexec. You can change that with this option. Note that paths may be less intuitive to use with this.

−e, **−−exts**=*EXTENSIONS*
Filename extensions to filter to

This is a quick filter to only emit events for files with the given extensions. Extensions can be given with or without the leading dot (e.g. 'js' or '.js'). Multiple extensions can be given by repeating the option or by separating them with commas.

−f, **−−filter**=*PATTERN*
Filename patterns to filter to

Provide a glob−like filter pattern, and only events for files matching the pattern will be emitted. Multiple patterns can be given by repeating the option. Events that are not from files (e.g. signals, keyboard events) will pass through untouched.

−−filter−file=*PATH*
Files to load filters from

Provide a path to a file containing filters, one per line. Empty lines and lines starting with '#' are ignored. Uses the same pattern format as the '−−filter' option.

This can also be used via the $WATCHEXEC_FILTER_FILES environment variable.

−i, **−−ignore**=*PATTERN*
Filename patterns to filter out

Provide a glob−like filter pattern, and events for files matching the pattern will be excluded. Multiple patterns can be given by repeating the option. Events that are not from files (e.g. signals, keyboard events) will pass through untouched.

−−ignore−file=*PATH*
Files to load ignores from

Provide a path to a file containing ignores, one per line. Empty lines and lines starting with '#' are ignored. Uses the same pattern format as the '−−ignore' option.

This can also be used via the $WATCHEXEC_IGNORE_FILES environment variable.

−−fs−events=*EVENTS*
Filesystem events to filter to

This is a quick filter to only emit events for the given types of filesystem changes. Choose from

'access', 'create', 'remove', 'rename', 'modify', 'metadata'. Multiple types can be given by repeating the option or by separating them with commas. By default, this is all types except for 'access'.

This may apply filtering at the kernel level when possible, which can be more efficient, but may be more confusing when reading the logs.

−−no−meta
Don't emit fs events for metadata changes

This is a shorthand for '−−fs−events create,remove,rename,modify'. Using it alongside the '−−fs−events' option is non−sensical and not allowed.

−−print−events
Print events that trigger actions

This prints the events that triggered the action when handling it (after debouncing), in a human readable form. This is useful for debugging filters.

Use '−v' when you need more diagnostic information.

−v, **−−verbose**
Set diagnostic log level

This enables diagnostic logging, which is useful for investigating bugs or gaining more insight into faulty filters or "missing" events. Use multiple times to increase verbosity.

Goes up to '−vvvv'. When submitting bug reports, default to a '−vvv' log level.

You may want to use with '−−log−file' to avoid polluting your terminal.

Setting $RUST_LOG also works, and takes precendence, but is not recommended. However, using $RUST_LOG is the only way to get logs from before these options are parsed.

−−log−file=*PATH*
Write diagnostic logs to a file

This writes diagnostic logs to a file, instead of the terminal, in JSON format. If a log level was not already specified, this will set it to '−vvv'.

If a path is not provided, the default is the working directory. Note that with '−−ignore−nothing', the write events to the log will likely get picked up by Watchexec, causing a loop; prefer setting a path outside of the watched directory.

If the path provided is a directory, a file will be created in that directory. The file name will be the current date and time, in the format 'watchexec.YYYY−MM−DDTHH−MM−SSZ.log'.

−−manpage
Show the manual page

This shows the manual page for Watchexec, if the output is a terminal and the 'man' program is available. If not, the manual page is printed to stdout in ROFF format (suitable for writing to a watchexec.1 file).

−−completions=*COMPLETIONS*
Generate a shell completions script

Provides a completions script or configuration for the given shell. If Watchexec is not distributed with pre−generated completions, you can use this to generate them yourself.

Supported shells: bash, elvish, fish, nu, powershell, zsh.

−h, **−−help**
Print help (see a summary with '−h')

−V, **−−version**
Print version

[*COMMAND*]
Command to run on changes

It's run when events pass filters and the debounce period (and once at startup unless '−−postpone' is given). If you pass flags to the command, you should separate it with −− though that is not strictly required.

Examples:

$ watchexec −w src npm run build

$ watchexec −w src −− rsync −a src dest

Take care when using globs or other shell expansions in the command. Your shell may expand them before ever passing them to Watchexec, and the results may not be what you expect. Compare:

$ watchexec echo src/*.rs

$ watchexec echo 'src/*.rs'

$ watchexec −−shell=none echo 'src/*.rs'

Behaviour depends on the value of '−−shell': for all except 'none', every part of the command is joined together into one string with a single ascii space character, and given to the shell as described in the help for '−−shell'. For 'none', each distinct element the command is passed as per the execvp(3) convention: first argument is the program, as a path or searched for in the 'PATH' environment variable, rest are arguments.

EXTRA

Use @argfile as first argument to load arguments from the file 'argfile' (one argument per line) which will be inserted in place of the @argfile (further arguments on the CLI will override or add onto those in the file).

VERSION

v1.21.1 branch:clap4 commit_hash:9ac37000 build_time:2023−03−04 02:43:21 +13:00 build_env:rustc 1.67.1 (d5a82bbd2 2023−02−07),stable−x86_64−unknown−linux−gnu

AUTHORS

FÃ©lix Saparelli <felix@passcod.name>, Matt Green <mattgreenrocks@gmail.com>