entr(1)

Run arbitrary commands when files change

overview | download 5.8 | man page

Examples

Rebuild project if sources change

```
ls | entr make
```

Rebuild project and run tests if the build was successful

```
ls | entr -s 'make && make test'
```

hint: ag and ack offer many advantages over utilities such as `find(1)` or `ls(1)` in that they recognize files by their contents and are smart enough to skip directories such as `.git`

Status Filters

`entr` allows custom status messages to be displayed by writing a status filters in `awk(1)`. Select a different script by setting the environment variable `ENTR_STATUS_SCRIPT`.

Restarting Services

`entr` adheres to the principle of separation of concerns, yet the reload (`-r`) option was added to solve a common use case that would otherwise require some careful scripting:

```
ls *.rb | entr -r ruby main.rb
```

This will,

1. immediately start the server
2. block until any of the listed files change
3. terminate the background process
4. wait for the server to exit before restarting

The restart option starts the utility as a background process that does not have access to `STDIN`. In this case keyboard input may be provided using a FIFO

```
mkfifo fifo
cat >fifo
```

Then start the `entr` in another console

```
ls main.go | entr -r -s '<fifo go run main.go'
```

Restarting Services with Timeout

In some cases the child process fails to respond when `entr` attempts to restart it. In this case it's helpful to send an alternate signal or kill the child process if it does not respond.

The timeout(1) utility can solve this by sending `SIGKILL` if the utility does not terminate after the specified period

```
ls app | entr -r timeout -k 5 0 ./app
```

`timeout` can also be used to send an alternate signal, such as `SIGINT`

```
ls app | entr -r timeout -s 2 0 ./app
```

Watching for New Files

The directory watch option (`-d`) was added to react to events when a new file is added to a directory. Since `entr` relies on standard input piped from other Unix tools, an external shell loop must be used to rescan the file system. One way to implement this feature would be to simply require the users to list directories, but `entr` will infer the directories if they aren't listed explicitly

```
while true; do
  ls -d src/*.py | entr -d ./setup.py
done
```

Other Implementation Details

Some architectural limitations are for good reasons, but it's not easy to see why a particular restriction applies.

First, the `-r` flag cannot be used with an interactive task:

1. Closing `STDIN` on the child allows `entr` to accept keyboard input.

2. If `entr` were to close it's own file descriptor to `STDIN` there is no reliable and immediate way to determine when the child has terminated in order to restore keyboard input.
3. In restart mode signals need to propagate reliably, so we use a new process group. If the utility reads STDIN under this new process group, the kernel will suspend it. Finding your process in the `T` state is confusing, so we closes STDIN to raise an error instead.

Processing Files

The `/_` shortcut for entr was intended as a means of saving typing in the case where you are monitoring one file

```
echo schema.sql | entr psql -f /_
```

For any other use it is incorrect because the operating system may consolidate events for files that were changed in close succession, but `entr` only prints one filename. This approach would also be incomplete because there is no way of detecting changes that occurred if `entr` isn't running.

Over time I have been asked to extend and enhance the capabilities of the `/_` shortcut; this is usually to in an effort to

1. emulate rsync(1) — send only files that changed
2. emulate make(1) — build only files that changed

`entr` will detect if something changed in your tree, but other tools should handle building source or copying files. Here is an example using a `Makefile`

```
.SUFFIXES: .md .html

SOURCES != ls input/*.md | awk 'sub("\.md$$", ".html")'

.md.html:
    markdown2html $< > $@

all: ${SOURCES}
```

Then monitor using

```
while sleep 0.1; do
  ls *.md | entr -d make
done
```

`make` uses the modification times of files to determine what needs to be updated. You can also do this comparison in shell, which may be more adaptable to your di-

rectory structure

```
#!/bin/sh
for input in $(ls input/*.md); do
  output="output/$(basename $input .md).html"

  [[ $input -nt $output ]] && {
      (set -x; markdown2html $input > $output)
  }
done
```

Tuning

The limits page includes instructions for raising the maximum number of watches on BSD, MacOS and Linux.

Last updated on April 30, 2026