

12 Terminal Commands Every Web Developer Should Know About

August 10th 2017 [The Terminal](#) | [Unix](#) | [Tools](#)

The terminal is one of the most important productivity tools in a developer's arsenal. Mastering it can have a very positive effect on your workflow, as many everyday tasks get reduced to writing a simple command and hitting Enter.

In this article we've prepared for you a collection of Unix commands that will help you get the most out of your terminal. Some of them are built in, others are free tools that are time-tested and can be installed in less than a minute.

Curl

Curl is a command line tool for making requests over HTTP(s), FTP and dozens of other protocols you may have not heard about. It can download files, check response headers, and freely access remote data.

In web development curl is often used for testing connections and working with RESTful APIs.

```
# Fetch the headers of a URL.
curl -I http://google.com
HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Location: http://www.google.com/?gfe_rd=cr&ei=0fCKWe6HCZTd8AfCoIWYBQ
Content-Length: 258
Date: Wed, 09 Aug 2017 11:24:01 GMT

# Make a GET request to a remote API.
curl http://numbersapi.com/random/trivia
29 is the number of days it takes Saturn to orbit the Sun.
```

Curl commands can get much more complicated than this. There are tons of options for controlling headers, cookies, authentication, and more. You can read more about curl in the excellent free book [Everything curl](#).

Tree

Tree is a tiny command line utility that shows you a visual representation of the files in a directory. It works recursively, going over each level of nesting and drawing a formatted tree of all the contents. This way you can quickly glance over and find the files you are looking for.

```
tree
.
├── css
│   ├── bootstrap.css
│   └── bootstrap.min.css
├── fonts
│   ├── glyphsicons-halflings-regular.eot
│   ├── glyphsicons-halflings-regular.svg
│   ├── glyphsicons-halflings-regular.ttf
│   ├── glyphsicons-halflings-regular.woff
│   └── glyphsicons-halflings-regular.woff2
└── js
    ├── bootstrap.js
    └── bootstrap.min.js
```

There is also the option to filter the results using a simple regEx-like pattern:

```
tree -P '*.min.*'
.
├── css
│   └── bootstrap.min.css
├── fonts
└── js
    └── bootstrap.min.js
```

Tmux

According to its Wiki, Tmux is a *terminal multiplexer*, which translated in human language would mean that it's a tool for connecting multiple terminals to a single terminal session.

```
Erlang, the programming language | Current Stable Version (17.4): erlang.org/download_release/, github.com/erlang/otp | Docs: erlang.org/erldoc, learnyousomeerlang.com | Communi
13:22      | +++ hamid ~nithpeunaffiliated/hamid
13:23      | entity: 6-16, start of second column
13:23      | not percentage but number of each in a very large project
13:23      | ah there it is
13:23      | thank you :)
13:24      | 122 gen_server 36 gen_event 10 gen_fsm
13:24      | np, read that part like 2 days ago
13:26      | hm, any clue if I can just say that gen_server is the most widespread behaviour and cite that history paper?
13:26      | I mean, it is a bit hard to find in the text
13:26      | I think that would be reasonable, if this is a write-up for a project or whatever.
13:26      | alright
13:26      | it's my thesis
13:27      | You could consider counting in a few large modern projects (e.g. riak & deps?) to verify.
13:27      | that'd be too much work tho :P
13:28      | Pff, grep for -behaviour lines. Students these days!
13:28      | ;o)
13:28      | deadline is in 3 weeks and I'm about 1/3 through
13:28      | Just kidding, forge ahead and good luck with the writeup.
13:28      | I guess I might give it a try... if I find some spare time
13:28      | emauton: thanks
[16:52] [mhi^(+Zi)] [8:fn/#erlang(+cgnt)]
#erlang
cyllene irc                                0:irssi*                                16:52

12      | case VTERM_PROP_ALTSCREEN:
11      | break;
10
9      | case VTERM_PROP_CURSORVISIBLE:
8      | term->cursor.visible = val->boolean;
7      | invalidate_terminal(term, term->cursor.row, term->cursor.row + 1);
6      | break;
5
4      | case VTERM_PROP_TITLE: {
3      | Error err;
2      | dict_set_value(term->buf->b_vars,
1      | cstr_as_string("term_title"),
0      | STRING_081(cstr_as_string(val->string)) &fail);
2015-04-22 10:05 Thomas Adam M [master] {origin/master} {origin/HEAD} Merge
2015-04-21 22:42 nicm | o Look up indexes as number before name, makes
2015-04-21 22:38 nicm | o Always format real layout even when zoomed.
2015-04-21 22:32 nicm | o Do not die on USR1 if any of the socket paren
2015-04-21 22:21 nicm | o Simplify error messages when socket connect f
2015-04-21 21:31 nicm | o The free callback could end up being fired be
2015-04-21 21:24 nicm | o Fix moving windows to nonexistent indexes whe
2015-04-21 15:34 nicm | o Bind mouse dragging so that it is passed thro
2015-04-21 15:21 nicm | o Don't eat the mouse event that triggers a dra
2015-04-21 15:18 nicm | o Put mouse_any_flag back, don't know where it
2015-04-21 15:18 nicm | o Pass mouse events through to commands for if-
2015-04-21 15:16 nicm | o cmd_mouse_pane can return NULL, check for tha
2015-04-20 15:41 nicm | o Remove unused-but-set variables from Thom
```

A Tmux Terminal With 3 Split Screens

It lets you switch between programs in one terminal, add split screen panes, and attach multiple terminals to the same session, keeping them in sync. Tmux is especially useful when working on a remote server, as it lets you create new tabs without having to log in again.

Disk usage - du

The du command generates reports on the space usage of files and directories. It is very easy to use and can work recursively, going through each subdirectory and returning the individual size of every file.

A common use case for du is when one of your drives is running out of space and you don't know why. Using this command you can quickly see how much storage each folder is taking, thus finding the biggest memory hoarder.

```
# Running this will show the space usage of each folder in the current directory.  
# The -h option makes the report easier to read.  
# -s prevents recursiveness and shows the total size of a folder.  
# The star wildcard (*) will run du on each file/folder in current directory.
```

```
du -sh *
```

```
1.2G    Desktop  
4.0K    Documents  
40G     Downloads  
4.0K    Music  
4.9M    Pictures  
844K    Public  
4.0K    Templates  
6.9M    Videos
```

There is also a similar command called [df](#) (Disk Free) which returns various information about the available disk space (the opposite of du).

Git

Git is by far the most popular version control system right now. It is one of the defining tools of modern web dev and we just couldn't leave it out of our list.

There are plenty of third-party apps and tools available but most people prefer to access git natively through the terminal. The git CLI is really powerful and can handle even the most tangled project history.

If you want to learn more about git, we recommend checking out our tutorial [Learn Git in 30 Minutes](#).

Tar

Tar is the default Unix tool for working with file archives. It allows you to quickly bundle multiple files into one package, making it easier to store and move them later on.

```
tar -cf archive.tar file1 file2 file3
```

Using the `-x` option it can also extract existing `.tar` archives.

```
tar -xf archive.tar
```

Note that most other formats such as `.zip` and `.rar` cannot be opened by `tar` and require other command utilities such as [unzip](#).

Many modern Unix systems run an expanded version of `tar` (GNU `tar`) that can also perform file size compression:

```
# Create compressed gzip archive.
tar -czf file.tar.gz inputfile1 inputfile2

# Extract .gz archive.
tar -xzf file.tar.gz
```

If your OS doesn't have that version of `tar`, you can use [gzip](#), [zcat](#) or [compress](#) to reduce the size of file archives.

[md5sum](#)

Unix has several built in hashing commands including [md5sum](#), [sha1sum](#) and others. These command line tools have various applications in programming, but most importantly they can be used for checking the integrity of files.

For example, if you have downloaded an `.iso` file from an untrusted source, there is some chance that the file contains harmful scripts. To make sure the `.iso` is safe, you can generate an `md5` or other hash from it.

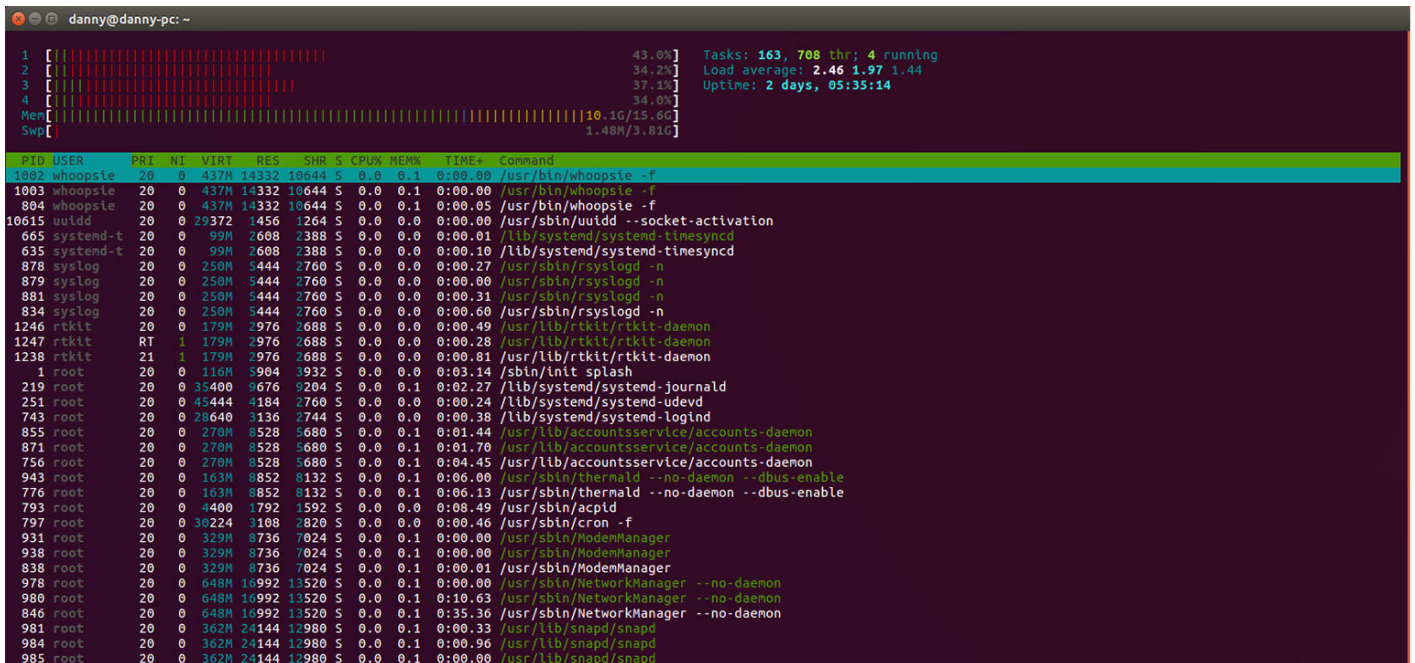
```
md5sum ubuntu-16.04.3-desktop-amd64.iso

0d9fe8e1ea408a5895cbbe3431989295  ubuntu-16.04.3-desktop-amd64.iso
```

You can then compare the generated string to the one provided from the original author (e.g. [UbuntuHashes](#)).

Htop

Htop is a more powerful alternative to the built-in `top` task manager. It provides an advanced interface with many options for monitoring and controlling system processes.



The htop task manager in action.

Although it runs in the terminal, htop has very good support for mouse controls. This makes it much easier to navigate the menus, select processes, and organize the tasks thought sorting and filtering.

Ln

Links in Unix are similar to shortcuts in Windows, allowing you to get quick access to certain files. Links are created via the `ln` command and can be two types: hard or symbolic. Each kind has different properties and is used for different things ([read more](#)).

Here is an example of one of the many ways you can use links. Let's say we have a directory on our desktop called *Scripts*. It contains neatly organized bash scripts that we commonly use. Each time we want to call one of our scripts we would have to do this:

```
~/Desktop/Scripts/git-scripts/git-cleanup
```

Obviously, this isn't very convenient as we have to write the absolute path every time. Instead we can create a symlink from our *Scripts* folder to */usr/local/bin*, which will make the scripts executable from all directories.

```
sudo ln -s ~/Desktop/Scripts/git-scripts/git-cleanup /usr/local/bin/
```

With the created symlink we can now call our script by simply writing its name in any opened terminal.

```
git-cleanup
```

SSH

With the `ssh` command users can quickly connect to a remote host and log into its Unix shell. This makes it possible to conveniently issue commands on the server directly from your local machine's terminal.

To establish a connection you simply need to specify the correct ip address or url. The first time you connect to a new server there will be some form of authentication.

```
ssh username@remote_host
```

If you want to quickly execute a command on the server without logging in, you can simply add a command after the url. The command will run on the server and the result from it will be returned.

```
ssh username@remote_host ls /var/www  
  
some-website.com  
some-other-website.com
```

There is a lot you can do with SSH like creating proxies and tunnels, securing your connection with private keys, transferring files and more. You can read more in [this guide](#).

Grep

Grep is the standard Unix utility for finding strings inside text. It takes an input in the form of a file or direct stream, runs its content through a regular expression, and returns all the matching lines.

This command comes in handy when working with large files that need to be filtered. Below we use `grep` in combination with the `date` command to search through a large log file and generate a new file containing only errors from today.

```
// Search for today's date (in format yyyy-mm-dd) and write the results to a new file.  
grep "$(date +%Y-%m-%d)" all-errors-ever.log > today-errors.log
```

Another great command for working with strings is [sed](#). It is more powerful (and more complicated) than `grep` and can perform almost any string-related task including adding, removing or replacing strings.

Alias

Many Unix commands, including some featured in this article, tend to get pretty long after you add all the options to them. To make them easier to remember, you can create short aliases with the *alias* bash built-in command:

```
# Create an alias for starting a local web server.  
alias server="python -m SimpleHTTPServer 9000"  
  
# Instead of typing the whole command simply use the alias.  
server  
  
Serving HTTP on 0.0.0.0 port 9000 ...
```

The alias will be available as long as you keep that terminal open. To make it permanent you can add the alias command to your `.bashrc` file.