



Architecture & Deployment

2025-2026 v0.1.0 on branch main Rev: 6d218297357081f922e8a7dd1d5cd8471c27fa79

Command Line Cheatsheet

Useful commands for Bash (**B**ourne-**a**gain **s**hell), the default shell for most Linux distributions and macOS. Most of these commands will also work with more advanced shells like ZSH (**Z** shell).

Table of contents

- I don't remember anything!
- Getting help (man, --help).
- Navigating
 - Where am I? (pwd).
 - What is there? (ls).
 - Move around (cd).
 - Special paths (., ..., ~).
- Reading
 - What's in this file? (cat, head, tail, less).
- Writing
 - Create a directory (mkdir).
 - Create a file (echo, touch).
 - Edit a file (nano).
 - Copy stuff (cp).
 - Move stuff (mv).
 - Delete stuff (rm).
- Finding stuff
 - Find files (find).
 - Find lines in files (grep).
- Environment variables
 - \$PATH
 - Do I have that command? (which).
- Miscellaneous

- Can I write a long command on multiple lines?

I don't remember anything!

Use the `history` command to list all the commands you previously typed:

```
$> history
24 ls
25 cd foo
26 ssh jde@archidep.ch
27 cat file.txt
```

Note that each command is prefixed by a number. Type an exclamation mark followed by that number to retrieve the command:

```
$> !26
$> ssh jde@archidep.ch
```

If you're looking for a specific command, you can filter your command history by piping it into the `grep` command. For example, this will search for all commands containing the word "ssh" in your history:

```
$> history | grep ssh
19 ssh-keygen --help
20 ssh-keygen
26 ssh jde@archidep.ch
```

Getting help (`man` , `--help`)

- `man ls` displays the manual of the `ls` command on Unix systems.
- `help ls` displays the help of the `ls` command in Git Bash on Windows.
- `git --help` displays the help of the `git` command. Many (but not all) commands provide a help page.

Navigating

Where am I? (`pwd`)

Use the `pwd` command, meaning **p**rint **w**orking **d**irectory:

```
$> pwd  
/Users/jde/Downloads
```

What is there? (`ls`)

Use the `ls` command, meaning **l**ist:

Command	Effect
<code>ls</code>	List the files in the current directory (invisible files are hidden)
<code>ls -a</code>	List all files in the current directory, including invisible ones
<code>ls -ahl</code>	List all files in the current directory, also displaying their mode, owner, group, size and last modification date
<code>ls foo</code>	List all files in the <code>foo</code> directory inside the current directory

Move around (`cd`)

Use the `cd` command, meaning **change directory**:

Command	Effect
<code>cd .</code>	Move into the current directory (<u>whewwwwww</u>)
<code>cd foo</code>	Move into the <code>foo</code> directory inside the current directory
<code>cd ./foo</code>	<i>Same as previous</i>
<code>cd foo/bar</code>	Move into the <code>bar</code> directory inside the <code>foo</code> directory inside the current directory
<code>cd ./foo/bar</code>	<i>Same as previous</i>
<code>cd ..</code>	Move into the parent directory (e.g. into <code>/foo</code> if you are in <code>/foo/bar</code>)
<code>cd ../..</code>	Move into the parent directory of the parent directory (e.g. into <code>/</code> if you are in <code>/foo/bar</code>)
<code>cd ~</code>	Move into your home directory
<code>cd</code>	<i>Same as previous</i>
<code>cd /</code>	Move to the root of the file system

Special paths (`.`, `..`, `~`)

Path	Where
<code>.</code>	The current directory (the same as indicated by <code>pwd</code>)
<code>..</code>	The parent directory (e.g. <code>/foo</code> if you are in <code>/foo/bar</code>)
<code>~</code>	Your user's home directory (e.g. <code>/Users/username</code> in macOS)
<code>/</code>	The file system's root directory on Unix systems

Reading

What's in this file? (`cat`, `head`, `tail`, `less`)

Display the **entire contents** of a file in the CLI with the `cat` command (as in concatenate):

```
$> cat file.txt
Hello
World
```

Display the **first or last N lines** (10 by default) of a file with the `head` and `tail` commands, respectively:

```
$> head file.txt
```

```
$> head -n 100 file.txt
```

```
$> tail file.txt
```

```
$> tail -n 50 file.txt
```

Display a large file in interactive mode, allowing you to scroll with the up and down arrow keys (exit this mode by typing the letter `q`, as in **quit**):

```
$> less file.txt
```

Writing

Create a directory (`mkdir`)

`mkdir` stands for **make directory**.

Create one directory in the current directory:

```
$> mkdir foo
```

Create a directory and all missing intermediate directories:

```
$> mkdir -p foo/bar/baz/qux
```

Create a file (`echo` , `touch`)

Create an empty file:

```
$> touch file.txt
```

Create (or overwrite) a file containing one line:

```
$> echo "Hello World!" > file.txt
```

Tip

Do the same with the `tee` command if you need to create a file with administrative privileges:

```
$> echo "Hello World!" | sudo tee file.txt
```

Append one line at the end of a file (also creates the file if it does not exist):

```
$> echo "Hello World!" >> file.txt
```

Tip

Again with the `tee` command if you need to create a file with administrative privileges:

```
$> echo "Hello World!" | sudo tee -a file.txt
```

Edit a file (`nano`)

Edit a file with nano:

```
$> nano /path/to/file.txt
```

Tip

Nano will always display available commands at the bottom of the screen. `Ctrl-X` is the one you will use most often, which saves and exits. Note that **nano will ask you to confirm the file name**. Just press Enter (unless you want to save your changes to another file, to keep the original one).

If you need superuser privileges to access the file, you can use `sudo`:

```
$> sudo nano /path/to/file.txt
```

Copy stuff (`cp`)

Copy a file to another location:

```
$> cp oldname.txt newname.txt
```

```
$> cp foo/oldname.txt bar/baz/newname.txt
```

Recursively copy a directory and all its contents:

```
cp -R olddirectory newdirectory
```

Move stuff (`mv`)

Move a file (or directory):

```
$> mv file.txt /somewhere/else
```

```
$> mv directory /somewhere/else
```

Delete stuff (`rm`)

Delete (or **remove**) a file:

```
$> rm file.txt
```

You can also delete a directory with `rm` by adding the `-r` (recursive) option.

```
$> rm -r directory
```

⚠ Be **EXTREMELY careful** when you type this command. One wrong move and you could permanently lose a lot of files (e.g. if you misspell the name of the directory you want to delete).

Finding stuff

Find files (`find`)

Recursively list all files in the current directory:

```
$> find .
```

Recursively list all JavaScript files in the current directory:

```
$> find . -name "*.js"
```

Find lines in files (`grep`)

Recursively find all files in the current directory containing the word “foo”:

```
$> grep -R foo .
```

Environment variables

Display an environment variable:

```
$> echo $F00
```

```
$> echo $PATH
```

Set an environment variable (it will be gone when you close the CLI):

```
$> export F00=bar
```

```
$> echo $F00
```

To be able to keep this environment variable in all future CLIs, save the `export FOO=bar` line to your `~/.bash_profile` file (create it if it doesn't exist).

`$PATH`

`$PATH` is an environment variable that contains a semicolon-delimited (`:`) list of directories. When you type a command such as `git`, Bash will look for a binary file named `git` in each of these directories one by one and execute the first one it finds.

If it doesn't find such an executable, it will return a `command not found` error.

```
$> echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

You can modify your path by adding `export` statements to your `~/.bash_profile` file (create it if it doesn't exist):

```
# Add the /foo directory to the beginning of $PATH
export PATH="/foo:$PATH"
```

```
# Add the /opt/local/bin directory to the beginning of $PATH
export PATH="/opt/local/bin:$PATH"
```

Do I have that command? (`which`)

Locate an executable in the `$PATH` with the `which` command:

```
$> which git
/usr/local/bin/git
```

```
$> which foo
foo not found
```

Miscellaneous

Can I write a long command on multiple lines?

```
$> echo \  
    "Hello" \  
    "World"  
Hello World
```

[↑](#) Back to top