fournova

COMMAND LINE CHEAT SHEET

presented by Tower - the best Git client for Mac and Windows

DIRECTORIES

`$ pwd`

Display path of current working directory

`$ cd <directory>`

Change directory to <directory>

`$ cd ..`

Navigate to parent directory

`$ ls`

List directory contents

`$ ls -la`

List detailed directory contents, including hidden files

`$ mkdir <directory>`

Create new directory named <directory>

OUTPUT

`$ cat <file>`

Output the contents of <file>

`$ less <file>`

Output the contents of <file> using the less command (which supports pagination etc.)

`$ head <file>`

Output the first 10 lines of <file>

`$ <cmd> > <file>`

Direct the output of <cmd> into <file>

`$ <cmd> >> <file>`

Append the output of <cmd> to <file>

`$ <cmd1> | <cmd2>`

Direct the output of <cmd1> to <cmd2>

`$ clear`

Clear the command line window

FILES

`$ rm <file>`

Delete <file>

`$ rm -r <directory>`

Delete <directory>

`$ rm -f <file>`

Force-delete <file> (add -r to force-delete a directory)

`$ mv <file-old> <file-new>`

Rename <file-old> to <file-new>

`$ mv <file> <directory>`

Move <file> to <directory> (possibly overwriting an existing file)

`$ cp <file> <directory>`

Copy <file> to <directory> (possibly overwriting an existing file)

`$ cp -r <directory1> <directory2>`

Copy <directory1> and its contents to <directory2> (possibly overwriting files in an existing directory)

`$ touch <file>`

Update file access & modification time (and create <file> if it doesn't exist)

PERMISSIONS

`$ chmod 755 <file>`

Change permissions of <file> to 755

`$ chmod -R 600 <directory>`

Change permissions of <directory> (and its contents) to 600

`$ chown <user>:<group> <file>`

Change ownership of <file> to <user> and <group> (add -R to include a directory's contents)

SEARCH

`$ find <dir> -name "<file>"`

Find all files named <file> inside <dir> (use wildcards [*] to search for parts of filenames, e.g. "file.*")

`$ grep "<text>" <file>`

Output all occurrences of <text> inside <file> (add -i for case-insensitivity)

`$ grep -rl "<text>" <dir>`

Search for all files containing <text> inside <dir>

NETWORK

`$ ping <host>`

Ping <host> and display status

`$ whois <domain>`

Output whois information for <domain>

`$ curl -O <url/to/file>`

Download <file> (via HTTP[S] or FTP)

`$ ssh <username>@<host>`

Establish an SSH connection to <host> with user <username>

`$ scp <file> <user>@<host>:/remote/path`

Copy <file> to a remote <host>

PROCESSES

`$ ps ax`

Output currently running processes

`$ top`

Display live information about currently running processes

`$ kill <pid>`

Quit process with ID <pid>

fournova

COMMAND LINE CHEAT SHEET

presented by Tower - the best Git client for Mac and Windows

GETTING HELP

On the command line, help is always at hand: you can either type `man <command>` or `<command> --help` to receive detailed documentation about the command in question.

FILE PERMISSIONS

On Unix systems, file permissions are set using three digits: the first one representing the permissions for the owning user, the second one for its group, and the third one for anyone else.

Add up the desired access rights for each digit as following:

4 – access/read (r)
2 – modify/write (w)
1 – execute (x)

For example, `755` means "rwx" for owner and "rx" for both group and anyone. `740` represents "rwx" for owner, "r" for group and no rights for other users.

COMBINING COMMANDS

If you plan to run a series of commands after another, it might be useful to combine them instead of waiting for each command to finish before typing the next one. To do so, simply separate the commands with a semicolon (;) on the same line.

Additionally, it is possble to execute a command only if its predecessor produces a certain result. Code placed after the `&&` operator will only be run if the previous command completes successfully, while the opposite `||` operator only continues if the previous command fails. The following command will create the folder "videos" only if the `cd` command fails (and the folder therefore doesn't exist):

```
$ cd ~/videos || mkdir ~/videos
```

THE "CTRL" KEY

Various keyboard shortcuts can assist you when entering text: Hitting `CTRL+A` moves the caret to the beginning and `CTRL+E` to the end of the line.

In a similar fashion, `CTRL+K` deletes all characters after and `CTRL+U` all characters in front of the caret.

Pressing `CTRL+L` clears the screen (similarly to the `clear` command). If you should ever want to abort a running command, `CTRL+C` will cancel it.

THE "TAB" KEY

Whenever entering paths and file names, the `TAB` key comes in very handy. It autocompletes what you've written, reducing typos quite efficiently. E.g. when you want to switch to a different directory, you can either type every component of the path by hand:

```
$ cd ~/projects/acmedesign/docs/
```

...or use the `TAB` key (try this yourself):

```
$ cd ~/pr[TAB]ojects/
  ac[TAB]medesign/d[TAB]ocs/
```

In case your typed characters are ambiguous (because "ac" could point to the "acmedesign" or the "actionscript" folder), the command line won't be able to autocomplete. In that case, you can hit `TAB` twice to view all possible matches and then type a few more characters.

THE ARROW KEYS

The command line keeps a history of the most recent commands you executed. By pressing the `ARROW UP` key, you can step through the last called commands (starting with the most recent). `ARROW DOWN` will move forward in history towards the most recent call.

Bonus tip: Calling the `history` command prints a list of all recent commands.

HOME FOLDER

File and directory paths can get long and awkward. If you're addressing a path inside of your home folder though, you can make things easier by using the `~` character. So instead of writing `cd /Users/your-username/projects/`, a simple `cd ~/projects/` will do.

And in case you should forget your user name, `whoami` will remind you.

OUTPUT WITH "LESS"

The `less` command can display *and paginate* output. This means that it only displays one page full of content and then waits for your explicit instructions. You'll know you have `less` in front of you if the last line of your screen either shows the file's name or just a colon (:).

Apart from the arrow keys, hitting `SPACE` will scroll one page forward, `b` will scroll one page backward, and `q` will quit the `less` program.

DIRECTING OUTPUT

The output of a command does not necessarily have to be printed to the command line. Instead, you can decide to direct it to somewhere else.

Using the `>` operator, for example, output can be directed to a file. The following command will save the running processes to a text file in your home folder:

```
$ ps ax > ~/processes.txt
```

It is also possible to pass output to another command using the `|` (pipe) operator, which makes it very easy to create complex operations. E.g., this chain of commands will list the current directory's contents, search the list for PDF files and display the results with the `less` command:

```
$ ls | grep ".pdf" | less
```

Git Cheat Sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via `HTTP` or `SSH`.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to `HEAD`.
`git reflog`	Show a log of changes to the local repository's `HEAD`. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Additional Options +

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.