Git Cheat Sheet – CIRA Software Engineering Group

Getting & configuring Git

Git is a freely-available, open-source version control software for Linux, Windows, and Mac. To get Git, visit: **`https://git-scm.com/downloads`**.

`git --version`	Check the version of your Git installation

Set and display Git local configuration information

`git config --list`	Display your current local Git configuration
`git config --global user.name "<name>"`	Sets the username associated with your commits
`git config --global user.email "<email>"`	Sets the email you want attached to your commits
`git config --global color.ui auto`	Adds a splash of color to Git to make your life easier
`git config --global core.editor <editor>`	If you are not a fan of your system's default, you can change the editor (e.g., **`atom`**, **`vim`**, **`emacs`**)

Where to start – setting up your repository

Initialize a new local repository, set up a remote, or clone an existing project along with its history.

`git init <repository name>`	Creates a new local repository
`git remote add <remote name> <url>`	Creates a link to a remote repository (e.g., a hosting service like GitHub or GitLab). *<remote name>* is an alias for the *<url>* & is typically set to 'origin'. (`-f` fetches the remote history during the add)
`git clone <url>`	Creates a local copy of the repository on your machine
`git clone -b <branch> <url>`	Only clones a specific branch

Making a change – stage & snapshot your changes

Review your edits, commit a snapshot, and tag your projects progress.

✔ `git status`	List which files have been modified or staged locally
`git show`	Last commit & subsequent file changes information
✔ `git diff <file>`	Show unstaged changes to your file(s)
`git diff --staged`	Show staged, uncommitted changes
`git diff --cached`	Difference between staged changes and the last commit
✔ `git add <file>`	Add a file snapshot to your next commit
✔ `git commit -m "<message>"`	Commit your staged content as a new commit snapshot
`git commit --amend`	Replace and/or edit the last commit before **`git push`**
`git tag`	Lists the repository's tags (`-n` displays notes & messages)
`git tag -a <version number> -m "<message>"`	Flags the code with a version number and message

What was I thinking? – reviewing history & undoing changes

Git gets it – you & your collaborators aren't always perfect. Browse & inspect how the files in the project have changed as well as erase your mistakes.

✔ `git log`	List version history for the currently checked out branch
`git log --tags --decorate=full`	A more human-readable version of `git log`
`git log --follow <file>`	Display the history for a single file
`git blame <file>`	Adds user & log information about file changes
`git diff <commit ID> <file>`	Changes between a previous commit & current file state
`git diff <branch 1>...<branch 2> <file>`	Shows the changes in a file between branches. Note that the **`...`** blends the differences between the two versions together to make it easier to compare.
`git reset <file>`	Removes the *<file>* from the staging area & keeps local, unstaged changes. This is a "mixed" mode reset.
`git reset <commit id>`	Undoes all commits after *<commit id>* while keeping local, unstaged changes
`git reset --hard <commit id>`	"Hard" reset mode undoes changes after the *<commit id>*, clears the staging area, & rewrites history
`git revert <commit id>`	Leaves commit history intact, but restores/copies a previous commit and sets it as the most recent
`git clean`	Removes untracked files from working directory (`-n` displays what will be removed; `-f` cleans the directory)
`git checkout <file>`	Restore a file to the previous commit

Keeping your repository up-to-date – managing remotes

Commands to assist in synchronizing your local repository with your remote. Note: **`<remote name>`** *is the alias you set for the remote (e.g.,* '**`origin`**'*) when you do a* **`git remote add`**.

`git remote -v`	Lists the url & name of the remote fetch & push
`git remote show <remote name>`	Displays the remote url & detailed branch statuses
`git remote set-url <remote name> <url>`	Changes the url for a specific remote name if it has been previously set by `git remote add`
`git fetch <remote name> <branch name>`	Fetches the entire repository history from the remote (adding branch only gets that specific branch)
✔ `git pull <remote name>`	Downloads the current working branch & merges it with local (to rebase rather than merge, add `--rebase`)
✔ `git push -u <remote name> <branch name>`	Sends local branch commits to the remote (`--tags` pushes tags; `--all` uploads changes to every local branch)

Git Cheat Sheet – CIRA Software Engineering Group

Organizing patches & features – branches

Creating branches isolates changes to your coding project while keeping your master branch stable.

✔ `git branch`	Lists branches in the repository (an * proceeds the current/working branch). `-a` shows remote branch names.
`git branch <branch name>`	Creates a new branch
`git branch -d <branch name>`	Deletes the specified branch
✔ `git checkout <branch name>`	Switches the repository to the user defined branch
`git checkout -b <branch name>`	A single step combining creating & checking out a branch

Incorperating changes into the main branch – merging

To merge, or not to merge, that is the question – useful commands to allow you to combine changes in your code with other branches in your repository. Once the 'feature' you added in your branch is stable, you can merge away!

✔ `git merge <branch name>`	Combine branch of the code you want to insert (add `--no-ff` to prevent/resolve a "fast-forward")
`git remote prune <remote name>`	Removes dead wood by cleaning up deleted remote branches in your local repository

Spring cleaning – refactoring or removing files & paths

Need to relocate or completely remove files or directories? Then, we've got Git commands for you! Note: use Git to relocate or remove your file rather than OS-specific commands so that the change is tracked.

`git rm <file or directory>`	Deletes the file or directory from your local working repository & stages the deletion for the next commit
`git rm --cached <file or directory>`	Removes the file or directory from version control, but keeps your local copy
✔ `git mv <source> <destination>`	Changes the name of the file or directory and stages the change for the next commit
`git filter-branch --tree-filter 'rm -f <file>' HEAD`	Removes the snapshots of the file from your commit history. *Warning:* rewriting history will open a huge can of worms – *You've been warned!*

Temporary commits & experimental code fragments – stashing

Putting changes on hold so you can do something else – let's face it, multitasking is next to impossible.

`git stash list`	List all current stash entries (e.g., changesets & stash ids)
`git stash save "<stash name>"`	Save and store your changes on a list
`git stash pop stash@{<number>}`	Applies a stash & removes it from the stash list
`git stash show -p stash@{<number>}`	Displays the differences between the branch & stashed changes

Incorporating snippets of code from other projects – subtrees

A friend has an awesome script in version control that you want to add to your shiny, new repository. ***Don't copy it! Subtree it!*** *Subtrees are repositories within repositories. Here are a few notes:*

- *Always run subtree commands from the top/root directory of your repository.*
- *Try not to modify the subtree's code in the current repository. But, it can be done if necessary.*
- ***DO NOT USE** a leading or trailing "/" for the subtree directory prefix; `dir1/dir2` is safe.*
- *Always use `--squash` to keep the subtree repository history out of your history.*

`git subtree add --prefix=<subtree directory> <remote name> <remote branch name> --squash`	Insert a branch of a remote repository into the specified subtree directory within your current repository. New remotes can be set with `git remote add`.
`git subtree pull --prefix=<subtree directory> <remote name> <remote branch name> --squash`	Get the updated branch for the remote repository for the specified subtree

Ingoring patterns & suppressing tracking – `.gitignore`

Including a well-thought-out ***`.gitignore`*** *file in your repository can save you a lot of time and headaches. The* ***`.gitignore`*** *tells Git what patterns to avoid and prevent you from committing to your repositories. Below are a few examples:*

`*.nc`	Adds a wildcard glob to avoid committing files with the extension `.nc`
`temporary_*`	Adds a wildcard glob for the partial name of a file
`data/`	Excludes the directory `data` from being committed
`data/*.dat`	Excludes files with the extension `.dat` in the directory `data`
`git ls-files -o -i --exclude-standard`	Lists all ignored files in the repository (good to check to make sure something you want version controlled isn't being ignored)
`git add --force <file>`	Adds a snapshot for an otherwise ignored file

Other useful Git resources

- `http://swcarpentry.github.io/git-novice/`
- `https://rogerdudler.github.io/git-guide/`
- `https://www.atlassian.com/git/tutorials/`
- `https://git-scm.com/docs/gittutorial`
- `https://try.github.io/levels/1/challenges/1`
- Your favorite internet search engine or question-and-answer site (e.g., Stack Overflow)

✔	Indicates a Git command that is used frequently
`command`	A Git command
`<text>`	User defined input in the Git command

v.2018.11.27