Git

Enjoy this cheat sheet at its fullest within Dash, the macOS documentation browser.

Create

Clone an existing repository

```
git clone ssh://user@domain.tld/repo.git
```

Clone an existing repository and all its sub-modules recursively

```
git clone --recurse-submodules ssh://user@domain.tld/repo.git
```

Create a new local repository

```
git init
```

Configuration

Set the name attached to all your commits

```
git config [--global] user.name <name>
```

Set the email attached to all your commits

```
git config [--global] user.email <email>
```

Set colorization of command line output for all repos

```
git config --global color.ui auto
```

Print set name (in current repository or globally)

```
git config [--global] user.name
```

Print set email (in current repository or globally)

```
git config [--global] user.email
```

Local Changes

List changed files in your working directory

```
git status
```

List changes to tracked files

```
git diff
```

Add all current changes in file to the next commit

```
git add <file>
```

Add all current changes to the next commit

```
git add .
```

Add changes to the next commit interactively

```
git add -p <file>
```

Rename file and add it to next commit

```
git mv <file> <new file name>
```

Delete file and add its deletion to next commit

```
git rm <file>
```

Commit all local changes in tracked files

```
git commit -a
```

Commit previously staged changes

```
git commit
```

Change the last commit

```
git commit --amend
```

Note: You shouldn't amend published commits!

Commit History

Show all commits

```
git log
```

Show changes over time for a specific file

```
git log -p <file>
```

Show changes over time for a specific committer

```
git log --author=<committer name>
```

Note: `<committer name>` is a pattern, so `Ed` will match `Edward Smith`. Quotes are optional if the pattern doesn't contain spaces.

Search (grep) commit messages for the given string

```
git log --grep=<string>
```

Who changed what and when in file

```
git blame <file>
```

Store changes temporarily

```
git stash
```

Remove and apply stashed changes

```
git stash pop
```

Remove file from all previous commits but keep it locally

```
git rm --cached <file>
```

Branches & Tags

List all existing branches

```
git branch
```

Switch HEAD branch

```
git checkout <branch>
```

Create a new branch based on your current HEAD

```
git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
git branch --track <new-branch> <remote-branch>
```

Delete a local branch

```
git branch -d <branch>
```

Delete a remote branch

```
git push origin --delete <branch>
```

Rename a branch locally

```
git branch -m <old name> <new name>
```

Rename a branch on remote

```
git push <remote> :<old name>
git push <remote> <new name>
```

Tag the current commit

```
git tag <tag-name>
```

Update & Publish

List all currently configured remotes

```
git remote -v
```

Show information about a remote

```
git remote show <remote>
```

Add new remote repository

```
git remote add <remote> <url>
```

Rename a remote

```
git remote rename <old-name> <new-name>
```

Download all changes from remote, but don't merge into HEAD

```
git fetch <remote>
```

Download all changes from remote, but don't merge into HEAD and clean up deleted branches from origin

```
git fetch -p <remote>
```

Download changes and directly merge into HEAD

```
git pull <remote> <branch>
```

Publish local changes on a remote

```
git push <remote> <branch>
```

Track a remote repository

```
git remote add --track <remote-branch> <remote> <url>
```

Publish your tags

```
git push --tags
```

Merge & Rebase

Merge branch into your current HEAD

```
git merge <branch>
```

Rebase your current HEAD onto branch

```
git rebase <branch>
```

Note: You shouldn't rebase published commits!

Abort a rebase

```
git rebase --abort
```

Continue a rebase after resolving conflicts

```
git rebase --continue
```

Resolve conflicts using your configured merge tool

```
git mergetool
```

Manually resolve conflicts using your editor and mark file as resolved

```
git add <resolved-file>
git rm <resolved-file>
```

Undo

Discard all local changes in your working directory

```
git reset --hard HEAD
```

Discard local changes in a specific file

```
git checkout HEAD <file>
```

Revert a commit by providing a new commit with contrary changes

```
git revert <commit>
```

Restore a specific file from a previous commit

```
git checkout <commit> <file>
```

Reset your HEAD pointer to a previous commit

- Discarding local changes:

  ```
  git reset --hard <commit>
  ```

- Preserving all changes as unstaged changes:

  ```
  git reset <commit>
  ```

- Preserving uncommitted local changes:

  ```
  git reset --keep <commit>
  ```

Submodules

Make changes, commit and checkout submodule files

Just go the submodule directory and use git as usual

List all currently configured submodules

```
git submodule
```

or

```
git submodule status
```

Show information about a submodule

```
git remote show <remote>
```

Add a new submodule

Beware of the submodule name you choose: If you use a forward slash (`/`) git will think you want to delete the submodule and want to add all the files in the submodule directory. Please DON'T use a forward slash after the submodule name.

1. Run `git submodule add -b <branch> --name <name> <repository-path-or-url>`

2. Add the `.gitmodule` file and submodule folder to the superproject index
3. Commit both files on the superproject

Remove a submodule

1. `git submodule deinit -f <submodule_path>`
2. `rm -rf .git/modules/<submodule_path>`
3. `git rm -f <submodule_path>`

(Details)

Clone a project with submodules

1. Clone the superproject as usual
2. Run `git submodule init` to init the submodules
3. Run `git submodule update` to have the submodules on a detached HEAD

or

Run `git clone --recurse-submodules ssh://user@domain.tld/repo.git`

See all changes on submodules

```
git diff --submodule
```

Update the submodules to the lastest changes on their respective branches

```
git submodule update --remote
```

Update a specific submodule to the lastest changes on its branch

```
git submodule update --remote <submodule-name>
```

Push changes to the superproject only if all submodules are pushed also

```
git push --recurse-submodules=check
```

Push changes to the submodules and then push the superproject changes

```
git push --recurse-submodules=on-demand
```

Run arbitrary commands on each submodule

```
git submodule foreach '<arbitrary-command-to-run>'
```

Notes

- Based on the cheat sheet from Tower.app. The original can be found here.
- Converted and extended by Jens Kohl.

You can modify and improve this cheat sheet here