# Git

# Cheat Sheet

Dream Coding
CODE YOUR DREAM

# Git Cheat Sheet

## Set up

### Git Config

**See all the config**

```
git config --list #show all the settings in gitconfig
```

**Open config to edit**

```
git config --global -e #open .gitconfig file
```

**Set default editor**

```
git config --global core.editor "code --wait" # set default text editor for git
```

**User settings**

```
git config --global user.name "name" #set user.name
git config --global user.email "email" #set user.email
git config user.name #check user.name
git config user.email #check user.email
```

**Set Auto CRLF**

```
git config --global core.autocrlf true #for Windows
git config --global core.autocrlf input #for Mac
```

**Git Aliases**

```
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.st status
```

### Help

**Git official site**

https://git-scm.com/docs

**See help**

```
git config --help #detail
git config --h #short
```

## Basic

### Git init

```
git init #initialise git
rm -rf .git #delete .git
```

## Show the working tree status

```
git status #full status
git status -s #short status
```

## Ignoring Files

Add files that should be ignored in **.gitignore** in project directory

For example:

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

## Staging files

```
git add a.txt #stage a.txt file
git add a.txt b.txt #stage a.txt, b.txt files
git add *.txt #stage all files ends with .txt
git add * #stage all files except deleted files and files that begin with a dot
git add . #stage everything
```

## Modifying files

### Removing files

```
rm file.txt #delete file
git add file.txt #add to staging area
git rm file.txt # removes file from working directory and staging area
git rm --cached file.txt #removes from staging area only
git clean -fd #removes all untracked files
```

### Moving files

```
git mv from.txt to.txt
git mv from.text /logs/from.text
```

## Viewing the Staged/Unstaged changes

```
git status #full status
git status -s #short status
git diff #changes in working directory
git diff --staged #changes in staging area
git diff --cached #same as --staged
```

### Visual Diff Tool

**Open .gitconfig and add below**

```
[diff]
    tool = vscode
[difftool "vscode"]
    cmd = code --wait --diff $LOCAL $REMOTE
```

**Run Git Diff tool**

```
git difftool
```

## Commit

```
git commit #commit stagged files
git commit -m "Commit message" #commit stagged files with commit message
git commit -am "Commit message" #commit all files with commit message
```

## Log · History

### See history

```
git log #list of commits
git log --patch #shows the difference introduced in each commit
git log -p #same as --patch
git log --state #abbreviated states for each commit
git log --oneline #oneline
git log --oneline --reverse #oneline, from the oldest to the newest
```

### Formatting

```
git log --pretty=oneline #same as --oneline
git log --pretty=format:"%h - %an %ar %s" #formatting
git log --pretty=format:"%h %s" --graph #show graph
git log --graph --all --pretty=format:'%C(yellow)[%ad]%C(reset) %C(green)[%h]%C(reset) | %C(white)%s %C(bold red){{%an}}%C(rese
git config --global alias.hist "log --graph --all --pretty=format:'%C(yellow)[%ad]%C(reset) %C(green)[%h]%C(reset) | %C(white)%
```

### Filtering

```
git log -2 #shows only the last n commits
git log --author="ellie"
git log --before="2020-09-29"
git log --after="one week ago"
git log --grep="message" #finds in commit messages
git log -S="code" #finds in the code
git log file.txt #logs only for file.txt
```

### History of a file

```
git log file.txt #history of file.txt
git log --state file.txt #shows statistics
git log --patch file.txt #show the changes
```

### HEAD · Hash code

```
git log HEAD
git log HEAD~1
git log hash
```

### Viewing a commit

```
git show HEAD #shows the last commit
git show hash #shows the given commit
git show hash:file.txt
```

### Comparing

```
git diff hash1 hash2 #all changes between two commits
git diff hash1 hash2 file.txt #changes to file.txt only
```

## Tagging

### Creating

```
git tag v1.0.0 #lightweight tag on latest commit
git tag v1.0.0 hash #lightweight tag on the given commit
git show v.0.0 #shows the tag
git tag -a v.1.0.0 -m "message" #annotated tag
```

### Listing

```
git tag #all the tags
git tag -l "v1.0.*" #search certain tags
```

### Deleting

```
git tag -d v1.0.0 #delete the given tag
```

### Syncing with Remote

```
git push origin v1.0.0 #sharing the given tag with remote
git push origin --tags #sharing all the tags
git push origin --delete v1.0.0 #delete a remote tag
```

### Checking out Tags

```
git checkout v1.0.0 #checkout certain tag
git checkout -b branchName v1.0.0 #create a new bracnh with the given tag
```

# Branch

## Creating branch

```
git branch testing #create a new branch called testing
git checkout testing #switches to testing branch
git switch testing #same as the above
git checkout -b testing #create and switch to testing
git switch -C testing #same as the above
```

### Managing Branch

```
git branch #simple listing of all branches
git branch -r #sees the remote branches
git branch --all #list including remote branches
git branch -v #sees the last commit on each branch
git branch --merged #sees merged branches
git branch --no-merged #sees not merged branches
git branch -d testing #deletes the branch
git push origin --delete testing
git branch --move wrong correct #rename
git push --set-upstream origin correct #push new name
```

### Comparing

```
git log branch1..branch2 #all the commits between branch1 and branch2
git diff branch1..branch2 #all the changes between branch1 and branch2
```

## Merge

```
git merge featureA #merges featureA branch into the current one
git merge --squash featureA #suqash merge, only one commit
git merge --no-ff featureA #creates a merge commit
git merge --continue
git merge --abort
git mergetool #opens merge tool
```

#### Merge tool Config

```
[merge]
    tool = vscode
[mergetool]
  keepBackup = false
[mergetool "vscode"]
    cmd = code --wait $MERGED
[mergetool "p4merge"]
    path = "/Applications/p4merge.app/Contents/MacOS/p4merge"
```

## Rebasing

```
git rebase master #rebase current branch onto the master
git rebase --onto master service ui #take commits of the ui branch forked from the service branch and move them to master
```

## Cherry picking

```
git cherry-pick hash #applies the given commit
```

# Stashing

### Saving

```
git stash push -m "message" #make a new stash
git stash #same as above
git stash --keep-index #stash but keep them in the staging area
git stash -u #--include-untracked
```

### Listing

```
git stash list #see all the stashes
git stash show hash #see the given stash
git stash show hash -p #see the given stash with details
```

### Applying

```
git stash apply hash #applies the given stash
git stash apply #applies the latest stash
git stash apply --index #apply the stated changes
git stash pop #apply and drop
git stash branch branchName #apply stash in a new branch
```

### Deleting

```
git stash drop hash #deletes the given stash
git stash clear #deletes all the stashes
```

# Undo

## Local Changes

### Unstaging a staged file

```
git reset HEAD file.txt
```

### Unmodifying a modified file

```
git checkout -- file.txt
```

### Discarding local changes

```
git restore --staged file.txt #unstaging a staged file
git restore file.txt #unmodifying a modified file
git restore . #unmodifying all modified files in the directory
git clean -fd #removes all untracked files
```

### Restoring file from certain commit

```
git restore --source=hash file.txt
git restore --source=HEAD~2 file.txt
```

## Commit

### Amending the last commit

```
git commit --amend
```

## Reset

```
git reset --soft HEAD #removes the commit and keep changes at staging area
git reset --mixed HEAD #removes the commit and keep changes at working directory
git reset --hard HEAD #removes the commit and don't keep the code
```

## Undo action - reflog

```
git reflog
git reset --hard hash
```

## Revert

```
git revert hash #reverts the given commit
git revert HEAD~1
gut revert --no-commit hash #reverts the given commit without revert commit
```

### Interactive Rebasing

```
git rebase -i HEAD~2
git rebase --continue
git rebase --abort
```

## Remote

```
git clone URL #cloning
git remote -v #shows all the remote URLs
git remote add name URL #add a new remote with name
```

#### Inspecting

```
git remote
git remote show
git remote show origin
```

#### Syncing with remotes

```
git fetch #pulls down all the data from remote
git fetch origin #same as the above
git fetch origin master #pulls down only master branch
git pull #fetch and merge
git pull --rebase #use rebase when pulling instead of merge
git push
git push origin master
```

#### Renaming or Removing

```
git remote rename sec second
git remote remove second
```

## Tools

### Basic Debugging

```
git blame file.txt
```

### Bisect

```
git bisect start
git bisect good hash
git bisect good
git bisect bad
git bisect reset
```

### Config

```
[alias]
  s = status
  l = log --graph --all --pretty=format:'%C(yellow)%h%C(cyan)%d%Creset %s %Cgreen(%cr) %C(magenta)<%an>%Creset'
```

```
    up = !git fetch origin master && git rebase origin/master
    co = checkout
    ca = !git add -A && git commit -m
    cad = !git add -A && git commit -m "."
    c = commit
    b = branch
    list = stash list
    save = stash save
    pop = stash pop
    apply = stash apply
    rc = rebase —continue
    get = "!f(){ git fetch && git checkout $1 && git reset --hard origin/$1; };f"
    new = "!f(){ git co -b ellie-$1 origin/master && git pull; };f"
    st = status
    hist = log --graph --all --pretty=format:'%C(yellow)[%ad]%C(reset) %C(green)[%h]%C(reset) | %C(white)%s %C(bold red){{%an}}%C
```