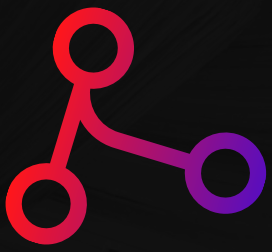

ABSOLUTE GIT CHEATSHEET

Learning Git
is easier than you think

Muhammad Ahsan Ayaz

Frontend Architect, Author,
Google Developers Expert

CLONING GIT REPO

Create a new local repository

```
git init PROJECT_NAME
```

Clone a repository

```
git clone YOUR_GIT_URL
```

Clone a repo inside a specific directory

```
git clone GIT_URL FOLDER_NAME
```

WORKING WITH BRANCHES

List all local branches

```
git branch
```

List all remote branches

```
git branch --remote
```

List both local and remote branches

```
git branch -av
```

Switch to an existing branch

```
git checkout MY_BRANCH
```

Create a new branch

```
git checkout -b MY_BRANCH
```

Delete a branch locally

```
git checkout -d MY_BRANCH
```

Delete a remote branch

```
git push origin --delete MY_BRANCH
```

Rename a branch

```
git branch -m NEW_NAME
```

Switch to the previous branch

```
git checkout -
```

Check out a single file from another branch

```
git checkout BRANCH_NAME -- FILE_NAME
```

Show all existing tags on the current branch

```
git tag
```

Add a tag to latest commit

```
git tag v1.2.0              # example
```

Delete a tag

```
git tag -d v1.2.0           # example
```

CONFIGURING GIT

Set user name for commits & tags

```
git config --global user.name “YOUR_NAME“
```

Set user email for commits & tags

```
git config --global user.email “YOUR_EMAIL“
```

Set upstream automatically for new branches

```
git config --global push.autoSetupRemote true
```

This is really handy to avoid the error/warning on console that says:

```
~/Documents/code/devtools git:(minor/test_branch) (0.051s)
git push
fatal: The current branch minor/test_branch has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin minor/test_branch

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

Enable colorization of Git output

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

Note: You can avoid the `--global` flag in all the above commands to set these up within the current git repository only

Edit the global config in an editor

```
git config --global --edit
```

MAKING CHANGES

Check active code changes

```
git status
```

Stage a file, ready for commit

```
git add FILE_PATH
```

Stage all files in the current folder

```
git add .
```

Unstage file, keeping file changes

```
git reset FILE_PATH
```

Reset everything to the last commit

```
git reset --hard
```

Diff of changes (not staged)

```
git diff
```

This shows only the changes which are **not** staged at the moment

Diff of staged code (not committed)

```
git diff --staged
```

This shows only the changes which are staged but **not** committed at the moment

Discard changes in a file (not staged)

```
git restore FILE_PATH
```

Commit all staged files

```
git commit -m “commit message“
```

Add and commit the ‘tracked’ files

```
git commit -am “commit message“
```

Note: usually newly created files are not tracked. With git commit -am, they’re not part of the commit

Amend the last commit

```
git commit --amend
```

This results in a prompt where you can edit the commit message

Reset author of the last commit

```
git commit --amend --reset-author
```

Rewrite last commit with this new message

```
git commit --amend -m “new message“
```

Delete the file from project and stage removal for commit

```
git rm FILE_PATH
```

Rename a file and stage

```
git mv EXISTING_PATH NEW_PATH
```

IGNORING FILES

Create a .gitignore file at the root of your git project

```
# Ignores the logs folder and all its files
logs/*

# "!" means don't ignore
!logs/.gitkeep

# Ignore Mac system files
.DS_store

# Ignore node_modules folder
node_modules

# Ignore SASS config files
.sass-cache

# Ignore all files with specific extensions
*.log
*.tmp

# Ignore build directories
dist/

# Ignore OS-specific files
.DS_Store
Thumbs.db

# Ignore IDE/editor specific files
.idea/
.vscode/

# Ignore environment variable files
.env
```

WORKING WITH REMOTES

List all remotes linked with local repo

```
git remote
```

Add (link) a new git remote

```
git remote add REMOTE_NAME REMOTE_URL
```

List all remotes with their URLs

```
git remote -v
```

Remove (unlink) a git remote

```
git remote rm REMOTE_NAME
```

Rename a (linked) git remote

```
git remote rename OLD_NAME NEW_NAME
```

WORKING WITH LOGS

Show the commit history of active git branch

```
git logs
```

Show the reflogs (concise view of git commits)

```
git reflog
```

Show commits that changed a file, even across renames

```
git log --follow FILE_PATH
```

Show commits on branchA that are not on branchB

```
git log BRANCH_B...BRANCH_A
```

Show diff of what is in branchA and is not in branchB

```
git diff BRANCH_B...BRANCH_A
```

Show the last commit and the changes done in it

```
git show
```

Search change in the logs by content

```
git log -S SEARCH_TERM
git log -S 'nodemailer' # example
```

Show logs that include changes of a particular file over time

```
git log -p FILE_PATH
git log -p ./app/contact/api.ts    #example
```

Print out a cool visualization of the git log

```
git log --pretty=oneline --graph
--decorate --all
```

MERGING BRANCHES

Merge branchA into branchB

```
git checkout BRANCH_B
git merge BRANCH_A
```

Rebase branchA into branchB

```
git checkout BRANCH_B
git rebase BRANCH_A
```

Rebase branchA into branchB interactively

```
git checkout BRANCH_B
git rebase -i BRANCH_A
```

Merge a remote branch into your local branch

```
git merge REMOTE_NAME/BRANCH_NAME
git merge origin/feat-1      # example
```

Merge a specific commit from another branch to your current branch

```
git checkout YOU_BRANCH
git cherry-pick COMMIT_ID
```

Revert a git commit on your current branch

```
git revert COMMIT_ID
```

WORKING WITH STASH

Move all your current changes to stash

```
git add .
git stash
```

This is really helpful when you want to pull the changes from remote without losing your active changes

See the list of stash file changes in the stack-order (latest first)

```
git stash list
```

Apply the changes from the most recent stash (on the top of stash)

```
git stash pop
```

Discard the most recent stash

```
git stash drop
```

Remove all the stash items from the list

```
git stash clear
```

THANK YOU FOR READING!

SHARE THIS FREE EBOOK TO SPREAD THE GIT LOVE ❤️

Muhammad Ahsan Ayaz

Frontend Architect, Author,
Google Developers Expert

https://bio.link/codewithahsan