

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIs

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows

<https://windows.github.com>

GitHub for Mac

<https://mac.github.com>

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms

<http://git-scm.com>

SETUP

Configuring user information used across all local repositories

```
git config --global user.name "[firstname lastname]"
```

set a name that is identifiable for credit when review version history

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

set an email address that will be associated with each history marker

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

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

```
git init
```

initialize an existing directory as a Git repository

```
git clone [url]
```

retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

```
git status
```

show modified files in working directory, staged for your next commit

```
git add [file]
```

add a file as it looks now to your next commit (stage)

```
git reset [file]
```

unstage a file while retaining the changes in working directory

```
git diff
```

diff of what is changed but not staged

```
git diff --staged
```

diff of what is staged but not yet committed

```
git commit -m "[descriptive message]"
```

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

```
git branch
```

list your branches. a * will appear next to the currently active branch

```
git branch [branch-name]
```

create a new branch at the current commit

```
git checkout
```

switch to another branch and check it out into your working directory

```
git merge [branch]
```

merge the specified branch's history into the current one

```
git log
```

show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

git log

show the commit history for the currently active branch

git log branchB...branchA

show the commits on branchA that are not on branchB

git log --follow [file]

show the commits that changed file, even across renames

git diff branchB...branchA

show the diff of what is in branchA that is not in branchB

git show [SHA]

show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

git rm [file]

delete the file from project and stage the removal for commit

git mv [existing-path] [new-path]

change an existing file path and stage the move

git log --stat -M

show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or committing of files

```
logs/  
*.notes  
pattern*/
```

Save a file with desired patterns as `gitignore` with either direct string matches or wildcard globs.

git config --global core.excludesfile [file]

system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repository

git remote add [alias] [url]

add a git URL as an alias

git fetch [alias]

fetch down all the branches from that Git remote

git merge [alias]/[branch]

merge a remote branch into your current branch to bring it up to date

git push [alias] [branch]

Transmit local branch commits to the remote repository branch

git pull

fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

git rebase [branch]

apply any commits of current branch ahead of specified one

git reset --hard [commit]

clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

git stash

Save modified and staged changes

git stash list

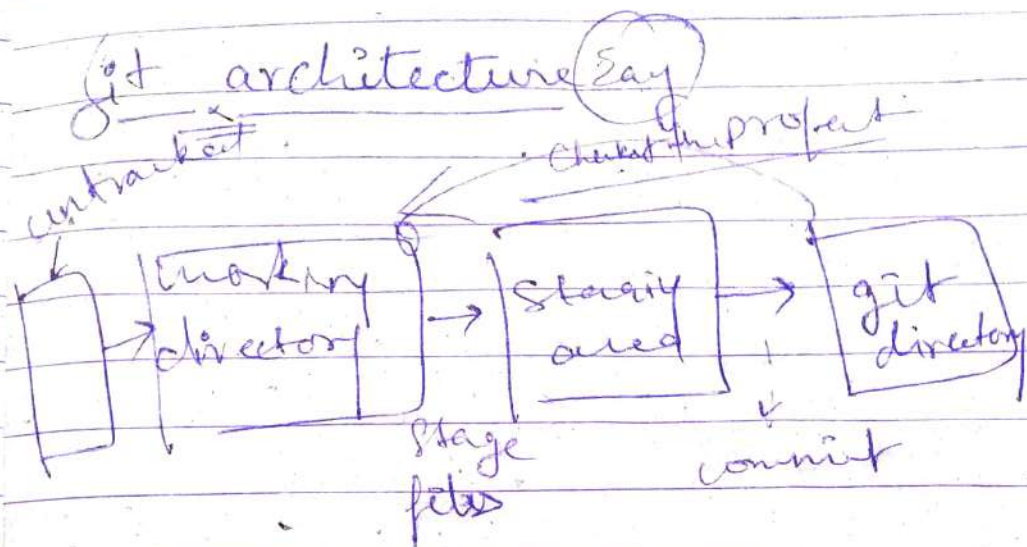
list stack-order of stashed file changes

git stash pop

write working from top of stash stack

git stash drop

discard the changes from top of stash stack



git command

- ① git status
- ② rm -rf .git] → delete git
- ③ git log] → log history
- ④ git add --^{new} file] → stage files all
- ⑤ git commit -m /^{file name} test &] → commit
- ⑥ git diff
- ⑦ git diff --staged
- ⑧ git log -p] → show difference in each
- ⑨ git log -p -1/2/3
- ⑩ git log --stat] → no of commit / summary
- ⑪ git log --pretty = short] pretty commits
- ⑫ git log --since = 2.days
- ⑬ git log --pretty = format : "%h" -- "%ae"
- ⑭ git commit -amend
- ⑮ git checkout --<filename>
- ⑯ git checkout -f] → for all file
- ⑰ git

Git / Practice

git restore --staged filename --to
to unstage.

git checkout -- filename
to the last commit

git checkout -A (or) to last commit
git push origin ~~branchname~~
git push -d origin ~~branchname~~

12:34 pm



wait of time

2:20 pm

git config --global alias.st status [aliasing
name]

git config --global alias.unstage 'restore --staged'

last log -p -1

Branch managers

git branch -v) with last commit

git branch --merged → already merged
git branch --no-merged → not merged

git branch -d develop

-D