**Git**

### **The Three States**

Pay attention now — here is the main thing to remember about Git if you want the rest of your learning process to go smoothly. Git has **three main states** that your files can reside in: **modified, staged, and committed:**

* **Modified means that you have changed the file but have not committed it to your database yet.**
* **Staged means that you have marked a modified file in its current version to go into your next commit snapshot.**
* **Committed means that the data is safely stored in your local database.**

**If a particular version of a file is in the Git directory, it’s considered committed.**

**If it has been modified and was added to the staging area, it is staged.**

**And if it was changed since it was checked out but has not been staged, it is modified.**

**to check your configuration settings of git: git config -l**

## **Getting a Git Repository**

You typically obtain a Git repository in one of **two** ways:

1. You can take **a local directory** that is currently not under version control, and **turn it into a Git repository**, or
2. You can **clone an existing Git repository** from elsewhere.

### **Cloning an Existing Repository**

URL -> HTTPS or SSH

**git clone <repo-url> <directory>**

Optional

If we don’t give directory name -> It will create a directory with same name of the repository

Otherwise, it will create a directory with the given name.

**cd dev1**

**ls -la .git/** -> will list all the files

**cat .git/HEAD** -> Default branch name (main or master)

**git log ->** Name, date & **displays committed snapshots**

**git config -l -> to check your configuration settings of git**

**nano ~/.gitconfig**  -> to check .gitconfig file [ **Global** ]

cd <repo>

cd .git -> to check .gitconfig file [ **Local** ]

Cat config

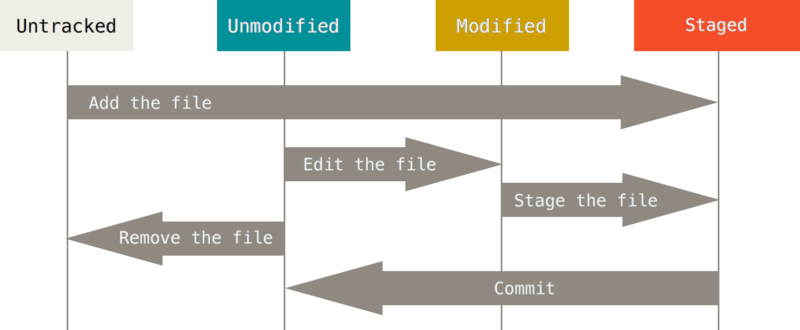
Git Configuration

**git config –local user.name <name>**  #write in .git/config

**git config –global user.name <name>** #write in ~/.git/config, Default

**git config user.name <name>** #write in ~/.git/config

**Git config -l**  # see all config(global, local, etc)



**Fig: Git File Status**

**Tracked Files:** The files were in the last snapshot:

* Modified
* Unmodified
* Staged

**Untracked Files:** The files are not tracked.

Create a hello.md file: **echo 'HEllo World!' > hello.md**

Lists of file:  **ls -la**

Open the file in vs Code: **code ./hello.md**

Check the git Status: **git status**

Add in staged area: **git add hello.md**

Restore from stage to unstaged: **git restore --staged hello.md**

Git Commit:  **git commit -m 'C1:main Add hello.md'**

History of Commit: **git log**

In front “+” & Green -> Add

In front “-” & Red -> Delete

To see details of Commit: **git log -p commit\_ID**

To see the Local branch: **git branch**

To see the Remote branch: **git branch -r**

To push git in Remote: **git push origin main**

**Working Tree:** Which modifications are in the local but not committed yet.

**Working Tree Clean:** After committing no changes.

Pull from remote directory to local directory:

**mkdir dev2**

**cd dev2**

**ls -la**

**git init**

**ls -la**

**git remote -v**

**git remote add origin git@github.com:Evanraisul/js-learn.git**

**git remote -v**

**git pull origin main**

**git log**

New commit with merge the previous work:

**git commit --amend -m 'C2:main Add amend.md'**

Ignore File(s)

Create a file named **.gitignore**. Add the file name you want to ignore.

**$** echo .DS\_Store > .gitignore

**echo file.txt > .gitignore**

**$** git status # see the file is ignored

**Git status**

**Git Branching**

Create new branch: **git branch b1**

Move to branch main to b1: **git checkout b1**

Create and Move to a new branch: **git checkout -b b2**

See git local branch list: **git branch**

See git remote branch list: **git branch -r**

See git all branch list: **git branch -a**

* Create a new branch with a remote branch history:
  + Same name: $ git checkout <bn1> # make sure no ‘bn1’ branch exists locally,
  + Different name: $ git checkout -b <bn1> <remote/bn2>

**Different Name:**

**git branch new**

**git push origin new**

**git branch -D new**

**git branch -la**

**git branch b2 remotes/origin/new**

**(\*) --> means current branch.**

**last commit show in first -> git log**

Delete branch b2: **git branch -D b2 [ from main branch ]**

**Git Stash:**

**(b1) -> change b1.md File**

**(b1) git status** -> modified

**(b1) git add .**

**(b1) git status** -> staged

**(b1) git commit -m 'C5:b1 Modify b1.md file'**

**git checkout b5** -> Switched to b5 Branch

**(b5) git log ->** Commit 5(C5) is not in this log

**(b5)> git add .**

**(b5)> git commit -m 'C6:b5 modify b1.md file'**

**git checkout b1**

**Please commit your changes or stash them before you switch branches.**

**(b5)> What changes I made: git diff**

**(b5)> git stash save 'more change'**

**(b5)> git status** -> Work Tree Clean.

**(b5)> git stash list** -> stash@{0}: On b5: more change

**(b5)> git checkout b1**  ->Switched

**(b1)> git checkout b5**  ->Switched

To get back half-done work: **(b5)> git stash apply**

**(b5)> git diff**

**git stash save 'more change 2'**

**git stash list**

**git stash apply -> Apply last one.**

**git stash save 'more change 3'**

**git stash list**

stash@{0}: On b5: more change 3

stash@{1}: On b5: more change 2

stash@{2}: On b5: more change

**git stash apply stash@{1} -> Apply specific stash**

**git stash list**

**git stash save 'more change 4'**

**git stash list**

**git status**

**git stash drop stash@{1} -> drop a specific stash, default-> 0**

**git stash list**

stash@{0}: On b5: more change 4

stash@{1}: On b5: more change 2

stash@{2}: On b5: more change

**git stash apply**

**git stash save 'more change 5'**

**git stash list**

stash@{0}: On b5: more change 5

stash@{1}: On b5: more change 4

stash@{2}: On b5: more change 2

stash@{3}: On b5: more change

**git stash pop -> Apply & Drop**

**git stash list**

stash@{0}: On b5: more change 4

stash@{1}: On b5: more change 2

stash@{2}: On b5: more change

Cherry-pick

* $ git cherry-pick C1 # apply the C1 commit’s changes
* $ git cherry-pick C1 C3 # apply the C1 & C3 changes
* $ git cherry-pick C1..C5 # apply a range of commits, C1 < C5, C1 **not** included
* $ git cherry-pick C1**^**..C5 # C1 included

**git cherry-pick 3efdf67d8b20a9e2e3154a7a6bcefa636a0470b6**

**7651d7933af94d14a35107b27689096993a5d742**

Unique ID of C4 & C5

Will generate the C4 and C5. but the **Unique ID will be different.**

**Revert?**

- Undo the changes of specific commit(s). Non-destructive, forward moving undo operation.

* $ git revert C1 # undo/reverse C1 changes
* $ git revert C1^..C4 # undo C1 to C4 changes

**git revert 530b7dd8cb8a67757560a917e600518197075754**

To revert a range C1 to C5: **git revert 530b7dd8cb8a67757560a917e600518197075754^..7651d7933af94d14a35107b27689096993a5d742**

**[ In git log top one is the last & bellow one is the first one ]**

**[ So, the loop have to be below to top ]**

**Git Conflicts?**

* Conflicts happen when git needs your help to decide which changes you want to keep (while merging)
* <<<< ... ==== ... >>>

**This branch is out-of-date with the base branch, merge the latest changes from master into this branch.**

gitk

Reset branch\_name branch

Mixed

git add .

git commit -s -m "Start config"

git push -u origin sc -f

—------------------------------------------------------------------

Git checkout master

Git pull

Git checkout sc

Git rebase origin/master

Git push -u -f

—---------------------------------------------------------------------------------------