*) GIT & GITHUB

Git was originally designed as Command line tool.

why command line?

→ history
→ new features
→ online help
→ Powerfull.

Git Bash = windows
Terminal = linux / mac

why Source control

1] SC is a type of Backup
2] ongoing backup - versioning / history
3] SC can be used for comparison.
4] collaboration / Teamwork.
5] Branching
6] code Review

Source control options

*] Two main Types
  1] centralized - have central Server for most operations
  2] Decentralized / distributed
    → doesn't Require a central Server
    → Most operations are local
    → [Git, mecurial (Hg)]

free: subversion, cvs

*) why git
  1] open Source
  2] distributed Source control System
  3] very fast
  4] Active community
  5] Scalabale in nature

1] Repositary = collection of version control files
↳ contains the history of changes.

2] Three Stages of Git :

i] working directory = A directory or folder on local your computer that contains all Project files.

ii] Staging area = git index = Holding area, queue
↳ Pre-commit, Holding area up changes for next commit

iii] commit = Git Repositary

4) Remote Repositary → A Remote Repository that contains all the 3 Stages together

5) Master Branch → dfault Branch
↓
timeline that contain your changes

`git version` → shows the current git version

`exit` → to close git Bash

```
git config --global user.name "Rohit"
—— ll —— user.email "l@gmail.com"
—— ll —— --list
```

`git clone link` - clone Repo.

`git status` - ~~tells us we are on which Branch~~

git config --global init.defaultBranch Main

`git ~~init~~ init Pro.name` → will create a new Repository (Remote) in local Laptop

`git pull origin master` → check weather the Repo is up-to-date

`git commit -am "hello"` a = add a stage, m = commit message

git tracked file = any file except working directory

`git ls-files` ⇒ check weather your file is tracked by the git or not.

`git mv test1.txt test2.txt` → Renaming a file using git

(current filename, new filename)

after Rename

`git commit -m "Rename done"`

`rm file.txt` → normal command to delete file

`git rm file.txt` → delete a tracked file.

`git commit -m "file deleted"`

`git rm hell.txt` → staged deletion

`git Reset HEAD hell.txt` → Unstage deletion

`git help log` → viewing help for git log

`git log` → Shows commit history in Reverse chronological order.

`git log --abbrev-commit` → shortens the commit id.

`git log --oneline --graph --decorate` → displays a graphical view of the commit history

`git log <start> <end>` → view Range of commits

`git log --since="3days ago"` → date based log

`git log --<filename>` → view history of a specific file

`git log --follow --<file_path>` → following Renames

`git show <commit_id>` → Shows information about Specific commit

eg → git log --all --graph --decorate --online.

→ creating alias:

```
git config --global alias.hist "log --all --graph --decorate --online"
```

`git hist` — name of the alias command you want to give.

*) Editing the alias

i] open .gitconfig file:

ii] Modify the alias under the [alias] section.

`git add .gitignore` → adding the commit to gitignore file.

Merge tools = PyMerge for windows.

README.md → mark down formatted file

`git stash` → takes your uncommitted changes (both staged & unstaged), saves them away for later use, & then reverts them from your working copy.

`git diff tool` → visually show your changes or differences

<commit_ID> <commit_ID>

staging area vs working directory

`git diff --staged`

`HEAD` → pointer that points the last commit of our history

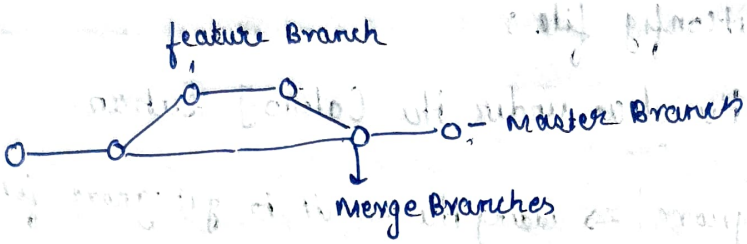
*] Comparing commits

`git log --oneline` → view the commit history

`git fetch origin` → fetch the latest changes from the Remote Repository

`git diff master origin/master` → compare the local `master` branch with the Remote `origin/master` Branch.

`git difftool master origin/master` → visually comparing using git difftool.



*) BRANCHING & MERGE

`git Branch -a` → list all the Branches available

`git Branch Ronny` → creates a new Branch by name Ronny

`git checkout Ronny` → will take the user to Ronny Branch and use it

`*` → shows the Branch we are using

`git Branch -m Ronny Tonny` → changes the name of the Branch

old name, new name

`git Branch -d Ronny` → delete the Branch (Ronny). In this we should be on different Branch to delete Ronny

`git checkout -b title-chang` → will create a Branch before checking it out

`git checkout -b Ronny` → To create a new branch and switch to it at the same time

↑ shortcut

```
git branch Ronny
git checkout Ronny
```

`git checkout Master` → Switch Back to master

1] creating an hotfix Branch

```
git checkout -b hotfix
```

2] merge to hotfix

```
git checkout master
git merge hotfix
```

`git merge --no-ff add-copyright` → To merge the changes from Ronny to master

merge with --no-ff creates a clear history, showing when branches were merged

4) Merge conflict

→ for example there are two branches and making the same changes in both of the Branches then when you try to merge those two branches it will show it as merge conflict

MODIFYING THE SAME AREA IN THE SAME FILE ON BOTH THE BRANCHES, GIT WILL DETECT A CONFLICT.

git conflict markers (<<<<<<, ======, >>>>>>)

`git mergetool` → mergetool by, used to Resolve the conflicts

`git Rebase master` →

Rebase is used to integrate the changes from the one branch into another Branch

`git mergetool` → launch an visual merge tool that is embedded to git

`git Pull master origin` → pull down from master origin

★) GIT STASHING

`git stash apply` → Saves info for later use on the working directory

`git stash list` → list of stash

`git stash drop` → It drops the last stash

git stash command only stash the tracked files

`git stash -u` → stash the untracked file aswell.

`git stash pop` → work as `git stash apply` + `git stash drop`

`git stash save` → Temporarily saves your information

{last stash is always index zero}

`git stash show stash@{2}` → to see the changes in stash with index 2

`git stash drop stash@{1}` → Remove the stash with index 1

`git stash clear` → to Remove all stashes

`git stash branch <Branchname> stash@{index}`

↳ to create a new branch from a stash

`git stash apply stash@{3}` → apply the stash

`git stash branch newchanges` → create a new branch and applied to stash

i] Stash POP: If you want to apply to stash and Remove it in one step, use.

`git stash POP stash@{index}`

ii] Stash Branch: you can also create a new Branch from stash.

`git Rebase -i` ⇒ If you need to clean up your commit history consider the above command.

*) TAGGING

→ A tag is an object Reference for a specific commit, similar to chapter markers in your book.

`git tag mytag` → creates a tag at current commit.

`git tag --list` → view all tags in your Repository

`git show mytag` → display the commit associated with that tag
Show information about that tag

`gittag --delete mytag` → Remove a tag from your Repository

`git tag -a <tagname> -m "message"` → used for more detailed tagging

`git Push origin<tagname>`
`git Push --tags` → To share the tags with others, Push them to Remote Repository.

`git tag -a v-1.0 -m "Release1.0"`
tagname, message

`git show v-1.0` → display the details of annotated tags

Lightweight tags	annotated tags
Simple markers on a commit	includes additional information metadata such as tagger's name, date, & message

`git commit --amend` → Amend the commit message if needed.

```
git diff V-1-0 V-1-2
git difftool V-1-0 V-1-2
```
→ compare difference between the two tags

`git tag -a V-0-8-alpha -f bd35d46 -m "Hello"`

↳ use the -f (force) option to move the tag to the correct commit.

`git Push origin V-0-9-beta` → Push a single tag to github.

`git Push origin --tags` → Push all local tags to github

HEAD → HEAD Refers to the latest commit on the current Branch.

→ you can move head to previous commits using the caret (^) or at (@{}) symbols

1] git reset HEAD^ → moves head back 1 commit

2] git reset HEAD^^ / git reset HEAD~2] → moves head Back 2 commits

3] git reset HEAD@{3} → move head Back 3 commits

*) Types of git Reset

The `git Reset` command is used to undo the changes in your working directory & git back to a specific commit while discarding all the commits made after that one.

1] Soft Reset: `git reset --soft <commit>` moves HEAD to specified commit without changing the staging area or working directory

2] mixed Reset (default): `git Reset commit` moves HEAD to the specified ~~command~~ commit and Reset the staging area, but not in working directory

3] Hard Reset: `git Reset --hard <commit>` moves HEAD to the specified commit and Reset Both the staging area & working directory

git Reflog → shows a log of all actions (reset, checkouts, commits etc) performed in the Repository

git fetch origin → checks local Repo is up to date with the Remote Repo.

4) when to use git Stash

1] Temporary changes → Saves your working Progress without committing it.

2] clean work directory Requirement

3] Quick context Switch

4) when to use git cherry-pick

1] Hotfixes

2] want to apply Specific commit without merging entire Branch

3] Isolated commits

git cherry-pick <commit-hash>

git cherry-pick --continue = continue cherry pick Process