Basic Commands

- **Git Config :**
  - git config -- global user.name NAME = set user name globally
  - git config --global user.email EMAIL = set user email globally
  - git config user.name OR git config user.email = check saved info

Creating repo

- git init = creates a git repository in the directory currently in

---

Staging

- git status = to check status , if staged or unstaged
- git add FILE_NAME = to add a file to staging area
- git rm --cached FILE_NAME = to remove a file from staging area
- git add . = to add all files in project to staging area

Commiting

- git commit -m "Specific Changes Made" = commits the staging area giving them a specific id
- git log = shows all the commits with details
- git log --oneline = shows all the commits in one line each
- **SPECIAL log** : this will log the info in a nice format (Try it once ☺)

  git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit

  - this can be used as an alias

  Git Stash

  - git stash = clears the changes to the initial state (last commit) & creates a unique id for the current state
  - git stash apply = brings back the current state
  - using git stash multiple times creates a list of stashes of all states with multiple ids
  - git stash list = shows all the stash (States) with their **ID**
  - git stash apply ID = **ID** will be the number , which state you want to go back to
  - git stash push -m "Your message" = used to give description to stash
  - git stash drop ID = used to remove a stash saved
  - git stash pop ID = applies the specific stash and removes it from history
  - git stash clear = removes all the stash history

Gitignore

- a .gitignore file can be created , in which you can specify all the folders/files that should not be staged and commited
- For example : node_modules/ .css.map etc.
- It's Good to create a gitignore at the start of Project
- a good gitignore generator for reference :
  - Gitignore.io

Reverting & Reset

- use git log --oneline to see the **commit_ID** to change to
- Checkout commit :
  - git checkout commit_ID = to just check the commit id entered , see it in read only ... changes will not be saved
  - git checkout master = to come back to original commit (As checkout removes us from master branch)
- Revert commit :
  - git revert commit_ID = to remove the changes of the provided commit (will add a new revert commit and remove the changes of the specific commit)
- Reset Commit :
  - git reset commit_ID = will remove all the commits after the provided id , but the files in local directory will not be touched (therefore you can still commit to original state after doing changes as needed) ... might take you to vim editor (type ":wq" then "Enter" to exit)
  - git reset commit_ID --hard = will remove all the commits after the provided id and even delete all the files and lines from local directory too

---

Branches

- Used to test a new feature or code , by creating a branch .. then merging it to master only if needed
- can be used for multiple developers working on same project .. create different branch for each developer adding their own feature then merging at the end
- git branch branch_name = to create a new branch
- git branch -a = to list all the branches
- git checkout branch_name = to shift to the other branch
- git branch -d branch_name = to delete the branch only when it has been merged

- git branch -D branch_name = to delete the branch (even if not merged to master)
- git checkout -b branch_name = to create and shift to a new branch at once

---

Merging branches

- after completing changes in a branch and commiting them
- come back to master and run
- git merge branch_name = this will merge the branch to master (all commits show in master) = automatic
- git merge --squash branch_name = this will merge the branch to master (only the commit after merge is shown in master) = manual

Conflicts

- If Branch's Base (First Commit) is Master's Head (Last Commit) = No Conflict
- If Master had commits after creating Branch = Conflicts Might Come
- to solve this , edit the files manually , Solve The Conflicts then ..
- run git add . and then git commit -m "Message" and the changes will be made

Git **Rebase** & Git **Merge**

- Using Git Merge Shows that the Branches Were Added to master , i.e the tree is not inline for all commits
- whereas Git Rebase keeps changing the base, and makes the commit inline , feels like the branch was never there
- RUN git rebase master on your branch
    1. Takes the base of master , matches it with every commit of your branch
    2. If The Master is already your base , no need of step 3, 4, 5
    3. solve the conflicts , then git add .
    4. run git rebase --continue
    5. Repeat 2, 3 steps for every commit - conflict
    6. Now The Master's Head is Branch's Base
    7. Move to Master
    8. run git rebase branch_name
    9. Now All the commits of Branch are added above your Master commits
- **NOTE!** : It is specified in the git docs that rebase should not be used in public repos (collaboration) as it can cause major errors and conflicts, it can be used in private repos.

Github

- Creating new & Cloning Repo
    - create a new repo on Github and copy the URL

- now push your code to it with
- git push git_url master = pushing code of master branch (to push all branches replace *master* with *--all*)
- creating an alias to not always type URL
- git remote add origin git_url = origin can be name of anything else, but origin is the word most commonly used
- git push origin master = to push code to using alias
- git push -u origin master = pushes and starts tracking the branch (u don't need to specify it again , ex. if pulling)
- git clone git_url = will copy the repo to current directory and also add the origin alias by default
- git remote -v = to check all the aliases made
- adding id and password in push\pull :
  - replacing the origin in git push origin master
  - git push https://username:password@repo_url.git master
  - if password contains @ replace it with **%40**
  - **NOTE** : this can store your password in plain text
  - to avoid this you can remove the password and enter it later
  - git push https://username@repo_url.git master

- Collaborating
  - Most of the collaboration features are already available on Github, Example
  - git pull git_url = to pull changes from remote to local repo
  - create a branch and make your changes
  - git push origin branch_name = to push the specific branch to remote
  - create a **Compare & Pull Request** when you want are ready for the branch to be merged (with a message)
  - the reviewer of the repo will accept the changes and merge it (and specify a merge commit message)
  - pull the project every time before editing to see the changes
  - git branch -r = helps us to see the remote branches & the connections

---

- Forking (Contributing)
  - to contribute to an open source project
  - click on fork , which will copy the repo to your account
  - make changes by pulling the repo, then push it (this will happen on your account)
  - then go to the owner account's repo and create a pull request there
  - the owner can compare the changes and accept your changes
  - which will end up merging your changes to their project