Title:

Git

Git

1. Git and GitHub

- Git and GitHub : Git is a version control system that allows us to track changes in our files and collaborate with others. It is used to manage the history of our code and to merge change from the different branches.

- Git and GitHub are different : Git is a version control system that is used to track changes to our files. It is a free and open-source software that is available for Windows, macOS, and Linux. Remember, GIT is a software and can be installed on our computer.

  GitHub is a web-based hosting service for Git repositories. GitHub is an online platform that allow us to store and share our code with others. It is a popular platform for developers to collaborate on projets and to share code. Also REMEMBER that it is not that GitHub is the only provider of Git repositories, but it is one of the most popular one.

- About Version Control System : Version control systems are used to manage the history of our code. They allow us to track changes to our files and to collaborate with others. VCS are essential for software development. Consider VCS as checkpoints in game. We can move to any time in the game and we can always go back to the previous checkpoints. This is the same concept in software development.

  Before Git became mainstream, version control systems were used by developers to manage their code.

they were called SCCS (Source Code Control System). SCCS was a proprietary software that was used to manage the history of code. It was expensive and not very user-friendly. Git was created to replace SCCS and to make version control more accessible and user-friendly. Some common version control systems are Subversion (SVN), CVS, and Perforce.

It's time to : (a.) Install Git on our system
(b.) Create an account on GitHub.

Check your Git version by running the cmd below :

```
$ git -v
$ git --version
```

2. Terminology

Git and People who use it talk in a different terminology. For Example they don't call it a folder, they call it a repository. they don't call it alternative timeline, they call it branch.

- Repository : A Repository is a collection of files and directories that are Stored together. Its a way to Store and manage our code. A repository is like a folder on our Computer, but it is more than just a folder. we can think of a repository as a container that holds all our code.

  Remember, there is a difference between a Software on our System vs tracking a particular folder on our System. Git is the Software that can continue tracks our folder if we initialize that particular folder as a git repository.

  At any point we can run the following command to see the current State of our repository :

  ```
  $ git status
  ```

- Configuration Settings : Whenever we checkpoint our changes, git will add Some information about us Such as our username, and email to the commit. there is a git config file that Stores all the settings that we have changed.

  Lets Setup our email and username in this config file. the username and the email is required to Setup the file which we have used on our GitHub account.

  ```
  $ git Config --global user.email "<the_email>"
  $ git config --global user.username "<the_name>"
  ```

Now, we can check our config setting :

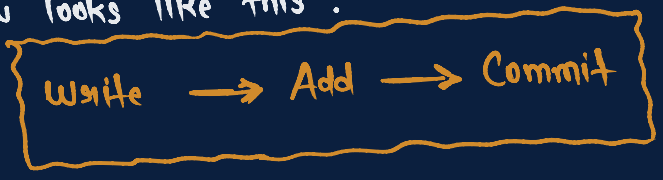
```
$ git config --list
```

- Create a Repository : Creating a repository is a process of creating a new folder on our system and initializing it as a git repository.

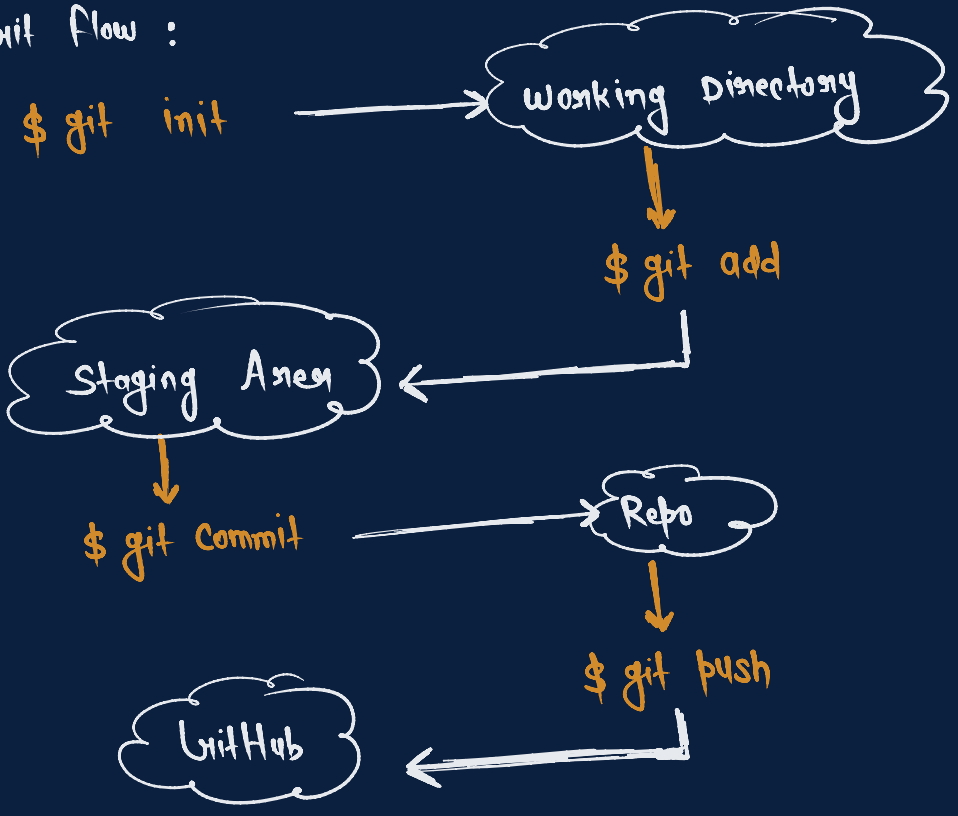
```
$ git status
$ git init
```

- Commit : Commit is a way to save our changes to our repository. It is a way to record our changes and make them permanent. We can think of a commit as a snapshot of our code at a particular point in time. When we commit our changes, we are telling git to save them in permanent way.

Usual flow looks like this :



- Complete Git Flow :



- **Stage** : Stage is a way to tell git to track a particular file or folder.

```
$ git init
$ git add <file1> <file2>
$ git add .
$ git status
```

- **Commit** : Here we are committing the changes to the repo. The message should be short and descriptive.

```
$ git commit -m "<message>"
```

- **Log** : We can also see the history of our repository.

```
$ git log
```

- **gitignore** : Gitignore is a file that tells git which files and folders to ignore. Its a way to prevent git from tracking certain files or folders.

  Files which can be excluded like :- node_modules, .env, .vscode, ... etc

3. Git behind the scene

- Git Snapshots: A git snapshot is a point in time in the history of our code. It represents a specific version of our code, including all the files and folders that were present at that time. Each snapshot is identified by a unique hash code (SHA), which is a string of characters that represents the contents of the snapshot.

  A snapshot is not an image, it's just a representation of the code at a specific point in time. Everything is stored as an object and each object is identified by a unique hash code.

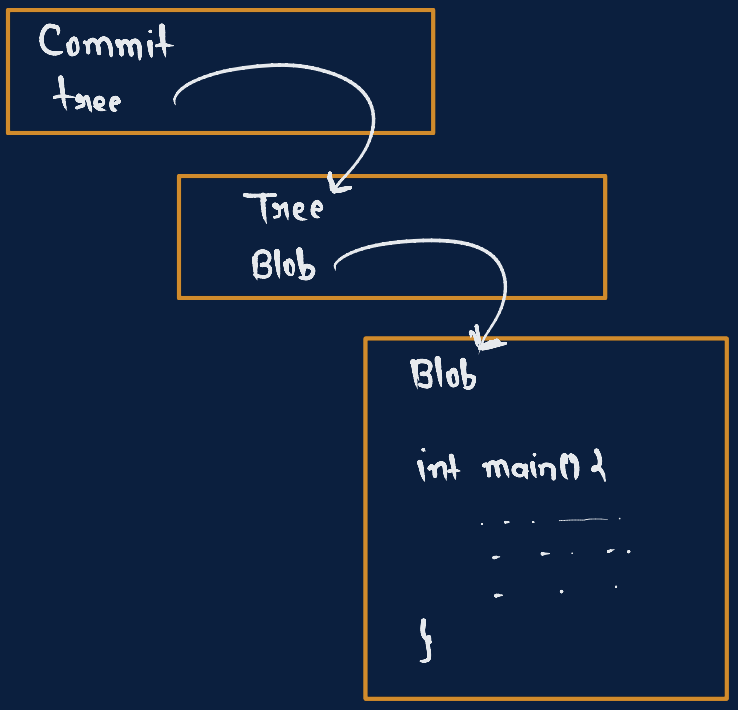
- 3 - Musketeers of Git : the three musketeers of git are :
  * Commit Object
  * tree object
  * Blob Object

  * **Commit Object** : Each commit in the project is stored in ".git" folder in the form of commit object. A commit object contains the following information :
    - tree object
    - Parent commit object
    - Author
    - Commiter
    - Commit message

  * **tree object** : tree object is a container for all the files and folders in the project. It contains the following information:
    - File mode
    - File Name
    - File Hash
    - Parent tree object

  Everything is stored as key-value pairs in the tree object. The key is the file name and the value is the file hash.

* **Blob Object** : Blob object is present in the tree object and contains the actual file content. This is the place where file content is stored.



* Helpful Command to explore git internally :

(1.) Grab tree id to get the tree object

```
$ git show -s --pretty=raw <commit-hash>
```

(2.) Get the tree object

```
$ git ls-tree <tree-id>
```

(3.) Get the blob object

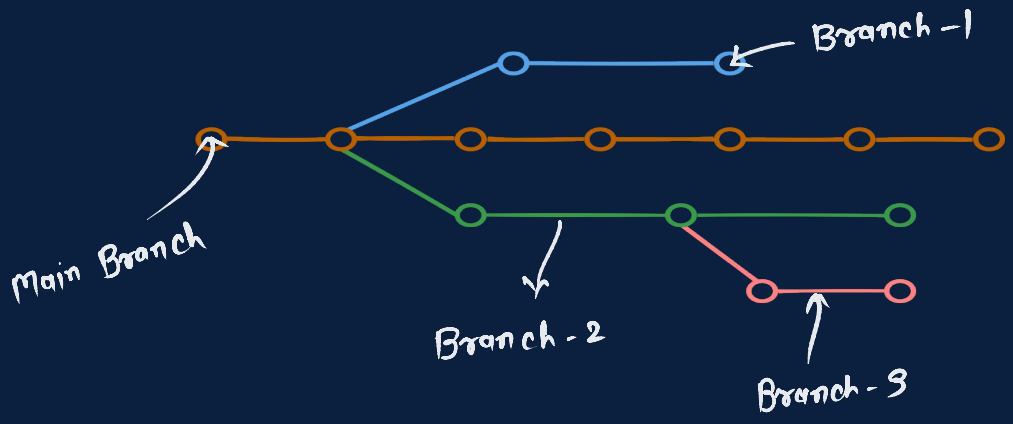
```
$ git show <blob-id>
```

(4.) Get the commit object

```
$ git cat-file -p <commit-id>
```

4. Branches in Git

Branches are a way to work on different versions of a project at the same time. They allow us to create a separate line of development that can be worked on independently of the main branch. This is useful when we want to make changes to a project without affecting the main branch or when we want to work on a new feature or bug fix.



- HEAD : The HEAD is a pointer to the current branch that we are working on.

- Here the important commands to work with branches :

```
$ git branch
$ git branch <branch-name>
$ git switch <branch-name>
$ git switch -c <branch-name>
$ git checkout <branch-name>
$ git branch -m <old-b-n> <new-b-n>
$ git branch -d <branch-name>
```

- Merging Branches : Merging is about bringing changes from one branch to another. In Git we have two types of mergers :

  * Fast-forward Merges → If branch have not diverged
  * 3-way merges → If branches have diverged.

```
$ git merge <branch-name>
```

Manage the merge conflict.

5. Diff, Stash and Tags

- **Git Diff** : This is an informative command that shows the differences between two commits.

  * How to Read the Diff output
    - a/ → The original file (Before changes)
    - b/ → The updated file (After changes)
    - --- → Begining of the original file
    - +++ → Begining of the updated file
    - @@ → Shows the line numbers and position of changes.

  * Here are some important Commands :

```
$ git diff
$ git diff --staged
$ git diff <branch 1> <branch 2>
$ git diff <branch1..branch2>
$ git diff <commit-hash-1> <commit-hash-2>
```

- **Git Stash** : Stash is a way to save our changes in a temporary location. It's useful when switching branches without losing work. We can then come back on the file later and apply the changes.

```
$ git stash
$ git stash save "name-the-stash"
$ git stash list
$ git stash apply
$ git stash apply stash@{number}
$ git stash pop ← Apply and drop
$ git stash drop
$ git stash apply stash@{no} <branch-name>
$ git stash clear
```

- Git Tags : Tags are a way to mark a specific point in our repository. They are useful when we want to remember a specific version of our code or when we want to refer to a specific commit. Tags are like sticky notes that we can attach to our commits.

```
$ git tag <tag-name>
$ git tag -a <tag-name> -m "Release_1.0.msg"
    ↳ This will be attached to the current commit.
$ git tag
$ git tag <tag-name> <commit-hash>
$ git push origin <tag-name>
$ git tag -d <tag-name>
$ git push origin :<tag-name>
    ↳ Delete tag on remote repository.
```

✳ Conclusion : Though not used as frequently as add, commit, or push, they are incredibly helpful in debugging, Context Switching, and release management.

6. Rebase in Git

Git rebase is a powerful Git feature used to change the base of a branch. It effectively allows us to move a branch to a new starting point, usually a different commit, by "replaying" the commit from the original base onto the new base.

Note : Ensure you are on the branch you want to rebase

```
$ git checkout <the-branch>
$ git rebase main
⇒ Resolve the conflict, if appears
$ git add <resolved-files>
$ git rebase --continue
```

- Git Reflog : Git reflog is a command that shows us the history of our commits.

```
$ git reflog
$ git reflog <commit-hash>
$ git reset --hard <commit-hash>
$ git reset --hard HEAD@{number}
```

7. GitHub

GitHub is a web-based git repository hosting service. It is a popular platform for developers to collaborate on projects and to share code. GitHub provides a user-friendly interface for managing and tracking changes to our code, as well as a platform for hosting and sharing our projects with others.

→ Alternatives :

- Gitlab
- Bitbucket
- Azure Repos
- Gitea

* Important Commands to work with GitHub :

```
$ git config --global user.email "email"
$ git config --global user.name "username"
$ git config --list
$ git init
$ git add <file-name>
$ git commit -m "message"
$ git remote -v
$ git remote add <origin or else> "link"
$ git push origin <branch-name>
$ git remote add -u "remote-url"
$ git push -u origin main
$ git push
$ git fetch <remote-name>
$ git pull origin main
$ git clone "url"
```