Git and Github Master Class

What is VCS?

VCS stands for **Version Control System**. It is a tool that helps manage and track changes to source code or other files over time. VCS is essential for software development and other projects where maintaining a history of changes, collaboration, and versioning is critical.

Popular VCS Tools:

- **Git**: The most widely used DVCS; supports branching and distributed workflows.
- **Subversion (SVN)**: A CVCS used in enterprise applications.
- **Mercurial**: Another DVCS, simpler than Git in some aspects.
- **Perforce**: A CVCS often used for large-scale enterprise projects.

Popular VCS Tools:

- Git: The most widely used DVCS; supports branching and distributed workflows.
- Subversion (SVN): A CVCS used in enterprise applications.
- Mercurial: Another DVCS, simpler than Git in some aspects.
- Perforce: A CVCS often used for large-scale enterprise projects.

Why do we need VCS?

1. **Tracking Changes**: It records changes to files over time, enabling developers to see who changed what and when.

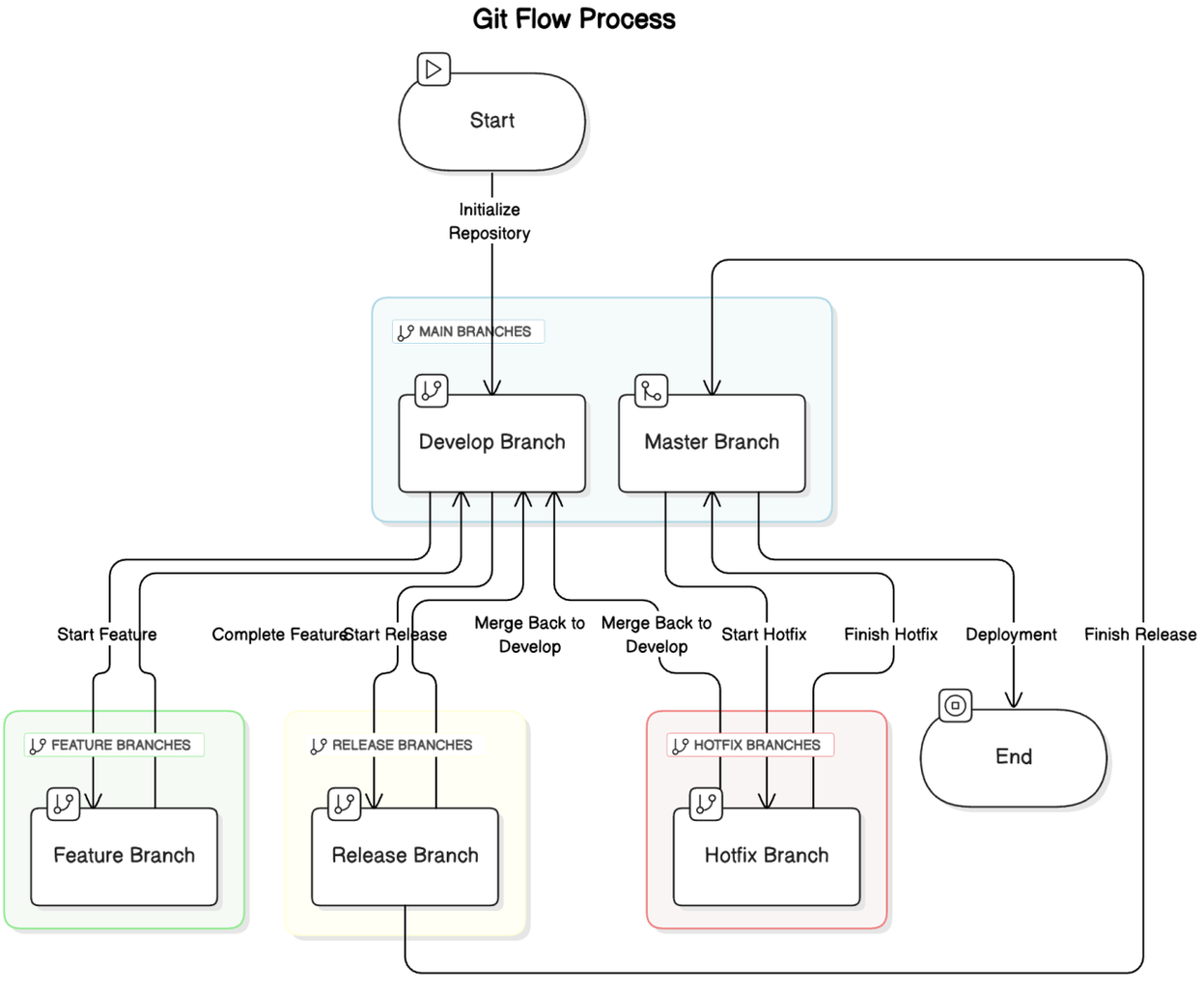
2. **Collaboration**: Multiple people can work on the same project simultaneously without overwriting each other's work.
3. **Branching and Merging**: Developers can create separate branches for different features or experiments and later merge them into the main project.
4. **Version History**: It keeps a history of all changes, making it easy to revert to previous versions if needed.
5. **Conflict Resolution**: Helps manage and resolve conflicts when multiple developers make changes to the same file.
6. **Backup and Recovery**: Acts as a **backup** for the project.

Introduction to Git

Git was created by Linus Torvalds

Git is a **version control system (VCS)** designed to track changes in source code and collaborate efficiently with others. It is widely used for software development due to its speed, flexibility, and support for non-linear workflows (e.g., branching and merging).

This what a typical git workflow in a startup or enterprise application code could look like.



Git CheatSheet

Command	Description
`git init`	Initialize a new Git repository
`git clone <url>`	Clone a repository from a URL
`git status`	Show the status of the working directory
`git add <file>`	Stage a file for commit
`git add .`	Stage all changes in the current directory
`git commit -m "message"`	Commit staged changes with a message
`git push`	Push commits to a remote repository
`git pull`	Fetch and merge changes from a remote repo
`git branch`	List branches
`git branch <name>`	Create a new branch
`git checkout <branch>`	Switch to a specific branch
`git merge <branch>`	Merge a branch into the current branch
`git log`	View commit history
`git diff`	Show differences between working files
`git reset <file>`	Unstage a file
`git stash`	Save changes without committing
`git stash pop`	Reapply stashed changes
`git remote add <name> <url>`	Add a remote repository
`git fetch`	Download objects and refs from another repo
`git rebase <branch>`	Reapply commits on top of another branch

start tracking the project
i.e start version controlling of that project (It initialize git in project but does not track individual file).

It shows which are tracked and which are not.

Now, individual files are tracked.

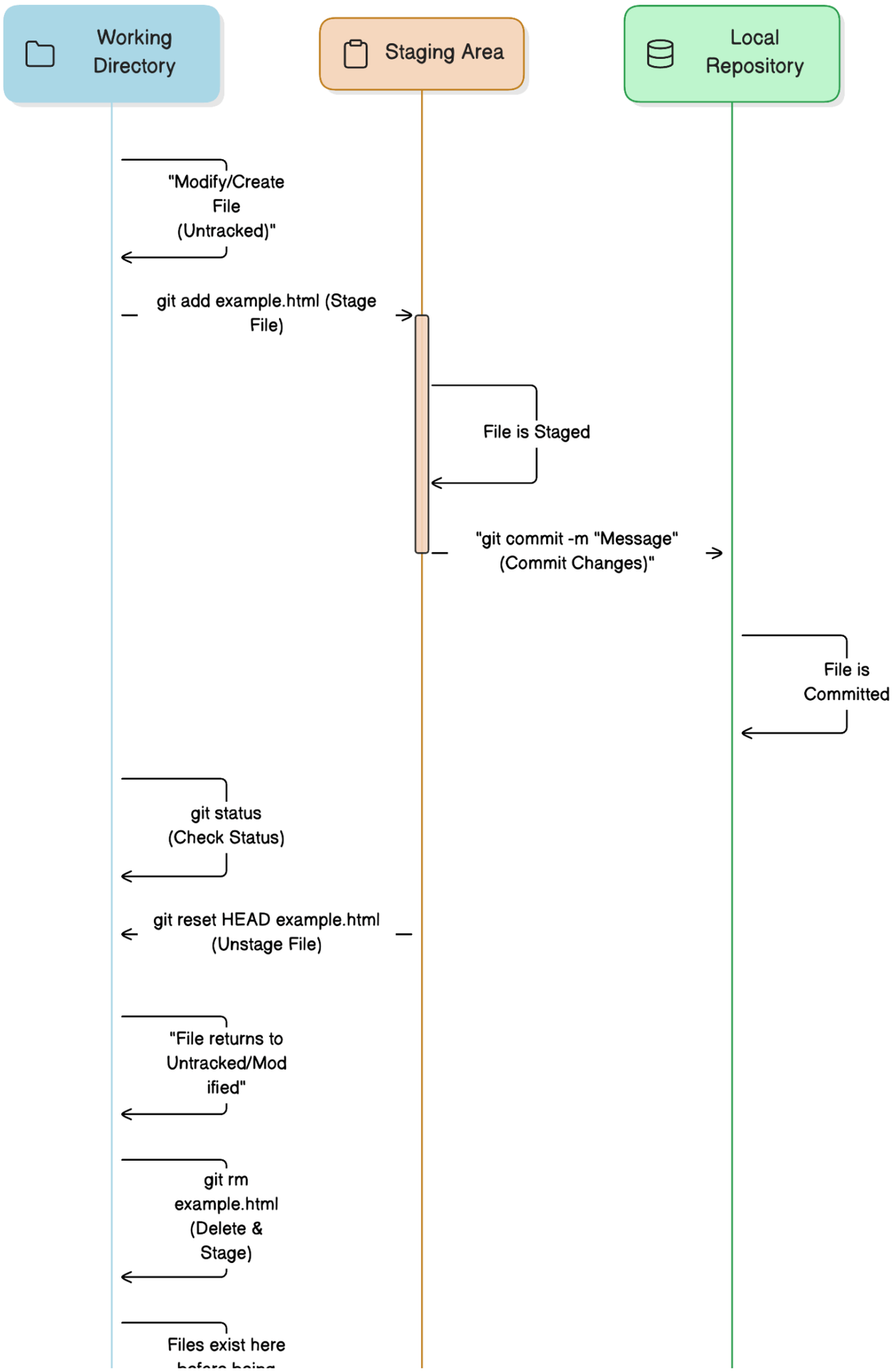
All files are tracked.

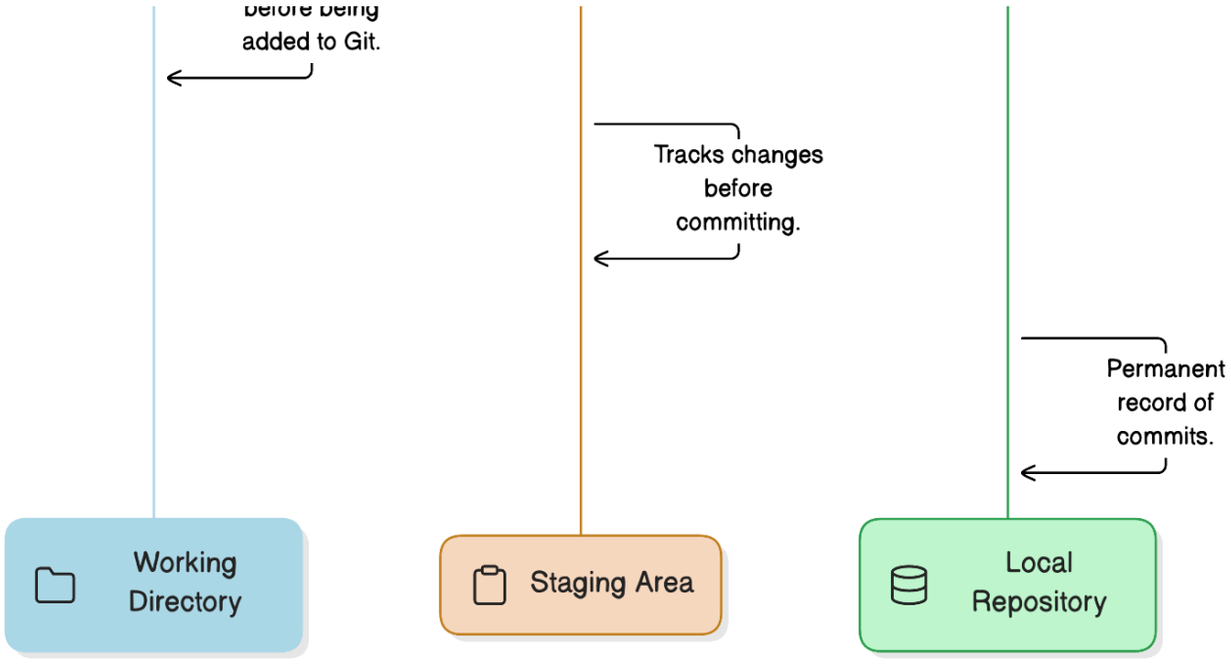
Commit is like checkpoint.

Push commits to Github.

It Gives difference between prev commit and curret situation.

Git Workflow





What is `.git/` Directory?

As soon as we run `git init` command, we get following message in the console:

```
Initialized empty Git repository in /home/Coding/piyushgarg-dev/projects
```

Let's look at what is in the `.git` folder

It is a hidden folder that is created inside your Folder after you run git init.

```
.git
├── config
├── HEAD
├── hooks
│   └── prepare-commit-msg.msample
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

- `config` is a text file that contains your git configuration for the current repo.
- `HEAD` contains the current head of the repo.
- `hooks` contain any scripts that can be run before/after git does anything.
- `objects` contains the git objects, ie the data about the files, commits etc in your repo. We will go in depth into this in this blog.
- `refs` as we previously mentioned, stores references(pointers)

In objects folder, folders are created with first two letters of hashId of commit. Reading the file (file name is hashID) inside the folder gives us all the changes made in that commit but in compressed format.

To see changes in that commit we can use
git cat-file -p 621ccc9194006093bac84cf2dfd393a77bc4a73a(HASH Id)

But git recommends use git diff to see changes, don't go inside .git folder.

-> There is a file named "index" in .git hidden folder. Which keeps track of tracked and untracked files. But "index" file is compressed and difficult to read. Hence git status is directly used.

-> All the git commands are actually readng and writing over .git files.
This means your project is tracking inside your own project in .git folder.

-> Whenever you commit, hash Id of that commit is generated (A long string).
To see that hashId use git log.

-> All the commits are in the form of linked list and head is pointing towards latest commit.
Remember every commit has reference of its prev commit (Linked List).

-> If you want to see difference between any two commits , you can use
git diff 621ccc9194006093b e9b37d8bcdbe029e (No need to write full hash id just some starting letters of whole string.)

-> How HashId is made?
It just reads your files of project and perform hashing on code using SHA1 algorithm and we get HASH ID.

-> Now you want to head of commit and you want to convert commits (which are after the commit you are shifting to) into changes (i.e statging area) . use : git reset hu1hete7b (i.e some letters of hashId of commit)
But if you want to discard the commits (which are after the commit you are shifting to) and delete them
use : git reset --hard hu1hete7b.

What is Github ?
It works as central space which can act as s source of truth.
All the collaborative workers of a project push and pull from github.

Now to understand create New Repository in Github and run commands on your terminal
-> git remode add origin //link//
->git push -u origin main

Now you code is pushed in github and repository is created.