Copy of 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.

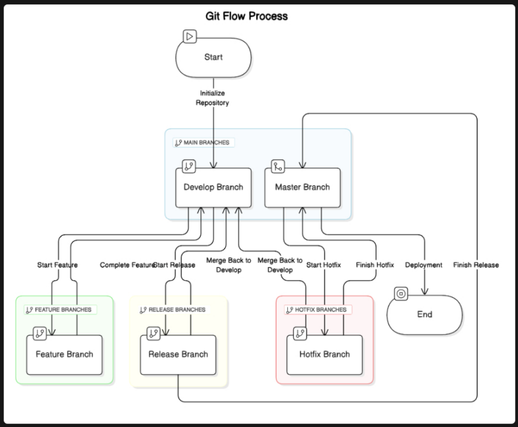
Why do we need VCS?

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 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.

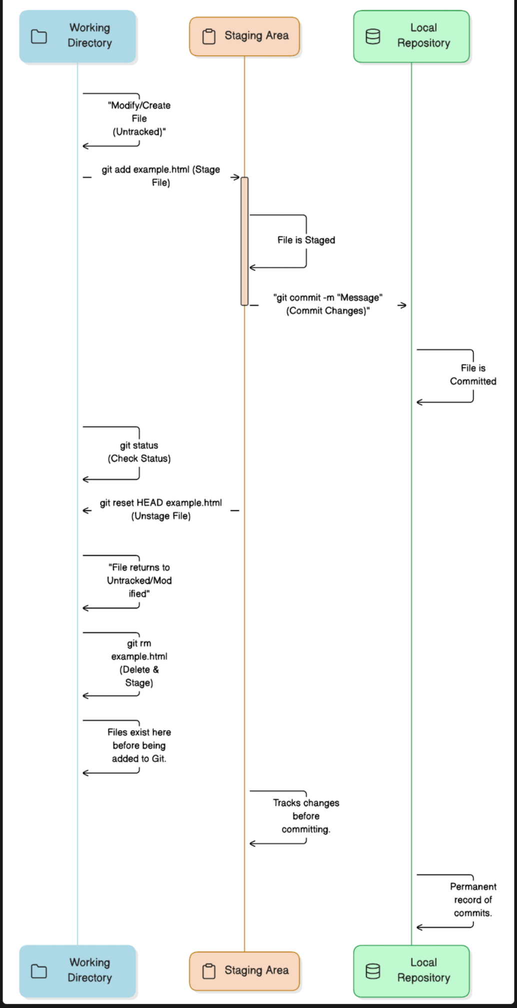


Git CheatSheet

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

Git Workflow


Working Directory
Staging Area
Local Repository
"Modify/Create File (Untracked)"
git add example.html (Stage File)
File is Staged
"git commit -m "Message" (Commit Changes)"
File is Committed
git status (Check Status)
git reset HEAD example.html (Unstage File)
"File returns to Untracked/Modified"
git rm example.html (Delete & Stage)
Files exist here before being added to Git.
Tracks changes before committing.
Permanent record of commits.
Working Directory
Staging Area
Local Repository

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

```
.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)