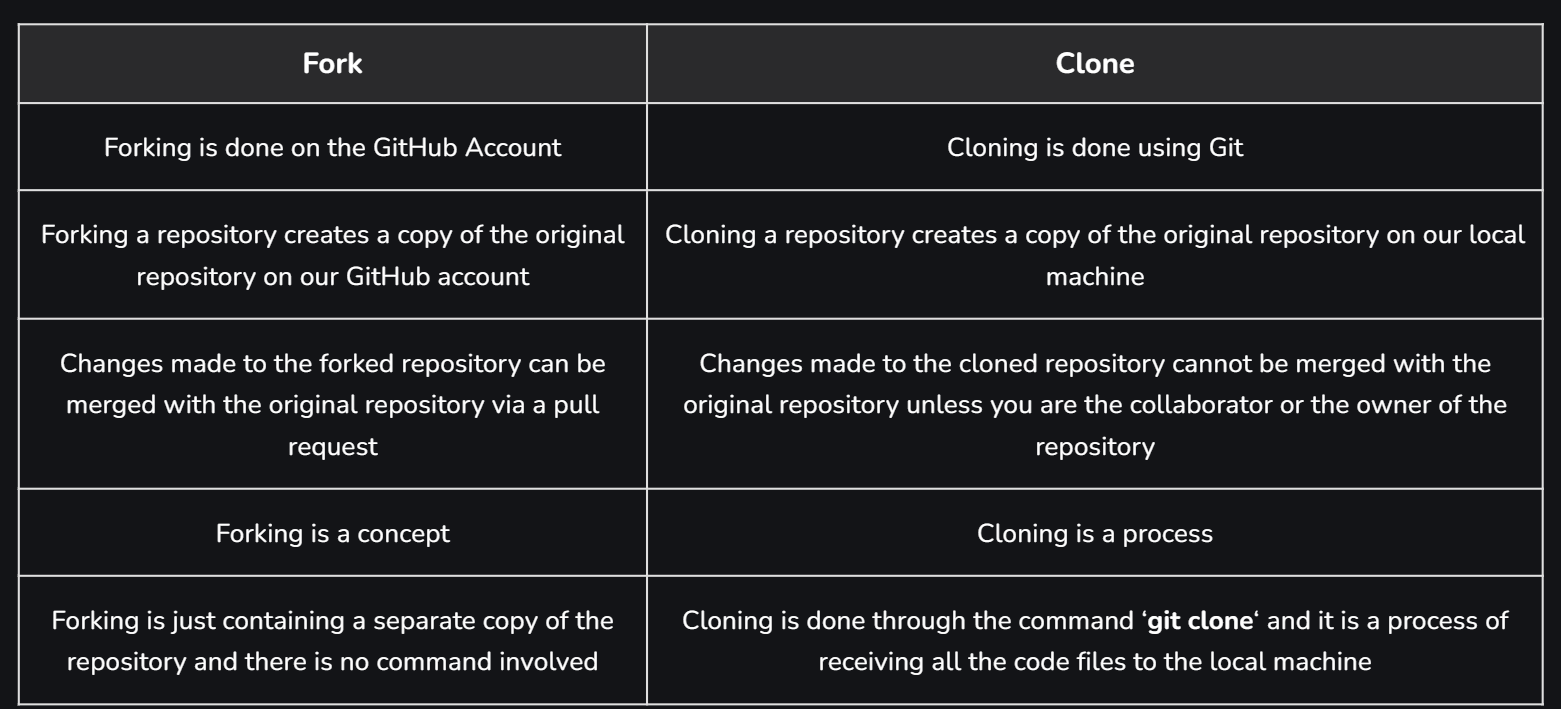
Git uses

Git vs github

| **Feature** | **Git** | **GitHub** |
| --- | --- | --- |
| Type | DVCS | Cloud service |
| Functionality | Version control | Hosting + tools |
| Local vs. Cloud | Local(without internet) | Cloud-based (internet is needed) |
| Collaboration | Limited | Extensive through pull requests, issues |
| Cost | Free (open-source) | Free and paid plans |

Git clone vs git fork

* **git clone** is used to create a local working copy of a remote repository without affecting the original
* **git fork** is used to create a copy of a remote repository on GitHub under your own account and establishes a link between the original and your forked copy. (changes reflect)



Git pull vs git fetch

* Downloads updates from a remote repository to your local repository.

Git pull= git fetch+ git merge

Git checkout vs git switch

* Git checkout: Creates branches, switches, restores files
* Git switch : Switching branches.

Git merge vs git rebase

[What's the difference between 'git merge' and 'git rebase'? - Stack Overflow](https://stackoverflow.com/questions/16666089/whats-the-difference-between-Git-merge-and-git-rebase)

Git merge: creates new commit, not linear history (large projects)

Git rebase: no commit is created, creates on top of the branch, linear history

Git revert vs git reset

A screenshot of a computer screen

Description automatically generated

It's worth noting that **git reset** does not add any new commits. Instead, it alters the existing commit history by moving the HEAD (and possibly the branch pointer) to a specified commit.

Git restore vs git reset

| **Feature** | **Git restore** | **Git reset** |
| --- | --- | --- |
| Functionality | Recover files/directories | Reset branch to commit |
| Target | Specific files/directories | Entire branch |
| Working Directory Impact | No (default), yes with --worktree | Yes |
| Stage Area Impact | No (default), yes with --staged | Yes |
| Use Cases | Recover specific files, unintended edits | Revert entire branch |

Git diff vs git show vs git status

| **Command** | **Purpose in a nutshell** | **Output in plain English** |
| --- | --- | --- |
| **git diff** | Shows differences between files | Highlights the changes between two versions of a file |
| **git show HEAD** | Displays information about the latest commit | Presents details like the commit hash, author name, date and time, commit title and message, changeset summary and detailed file modifications |
| **git status** | Provides a summary of the repository state | Indicates whether there are any uncommitted changes, staged changes or untracked files in your working directory and whether you are ahead or behind the remote branch |
| **git log** | Shows a history of commits | Lists all commits made to the repository, with information about each one including author name, date and message |

A screenshot of a chat

Description automatically generated

Reset: soft, hard, mixed

Git reset operates in three main modes: soft, mixed, and hard.

* **Soft reset** (git reset --soft) will move the HEAD to a specified commit but does not alter the index or working directory. All the changes will stay in the index, and you can re-commit them.
* **Mixed reset** (git reset --mixed) is the default mode when you run git reset. This moves HEAD to a specified commit and resets the index to match it, but leaves the working directory unchanged. This effectively unstages all changes.
* **Hard reset** (git reset --hard) moves HEAD to a specified commit, resets the index to match it, and also resets the working directory to match it. This discards all changes in the working directory and index.

Git stash

Generate ssh key:

ssh-keygen -t rsa -b 4096 -C "test@email.com"

add key in github account

git clone sshpath

**Git Stash Commands:**

1. git stash – to stash changes
2. git stash save "<Stashing Message>“- save with message
3. git stash list – to get list of stashes
4. git stash apply – to apply changes to working area, without deleting the stash(recent stash)
   1. git stash apply <stash id>
5. git stash show – to get changes of the stash
   1. git stash show -p - edited content and files changed {-p - partial stash}
6. git stash pop – to apply changes to working area, by deleting the stash
   1. git stash pop <stash id>
7. git stash drop – unstash (it deletes the most recent stash) (files can be recovered from backup)
   1. git stash drop <stash id>
   2. git stash drop stash@{1}
8. git stash clear – delete all available stashes (for branch, cannot be undo)
9. git stash branch <Branch Name> - creates new branch for making changes without distributing that stash

git stash pop = git stash apply + git stash drop

git pull = git fetch + git merge

git status --prettyprint -- graph

git status --oneline

git log --oneline

git log –prettyprint

**Git Cherry-pick :**

git cherry-pick is a powerful command that allows you to apply changes from a specific commit to the current branch.

It essentially “picks” a commit from one branch and applies it to another, creating a new commit that includes the same changes.

Steps:

* Get the commit using ‘git log’ and switch to target branch
* Run ‘git cherry-pick <commit\_id>’ (commit id is in different branch)
* If there are conflicts, Git will pause and allow you to resolve them manually.   
  Once you’ve resolved the conflicts,
  + run git add .
  + and then git cherry-pick –continue
* git cherry-pick –-abort - to abort the cherry-pick
* git cherry-pick --no-commit abc123 – don’t create new commit

to change the message of cherry you need to edit the commit and -m use new commit and -f forces the cherry-pick even if there are conflicts

[Git Cheat Sheet (2024) - All Git Commands (geeksforgeeks.org)](https://www.geeksforgeeks.org/git-cheat-sheet/)

[30+ Commonly Asked GIT Interview Questions (2024) - InterviewBit](https://www.interviewbit.com/git-interview-questions/)

* git branch --merged helps to get the list of the branches that have been merged into the current branch.
* Note: git branch --no-merged lists the branches that have not been merged to the current branch



The **git fetch --prune** command is a powerful tool for maintaining a clean and efficient Git repository by ensuring your local tracking branches are synchronized with the remote repository and removing any stale branches that no longer exist on the remote.

Git reflog - **Keeping Track of Your Git Journey**

The **git reflog** command in Git is like a **"history book"** for your references. It keeps track of all the changes made to your HEAD, branches, and other Git references, allowing you to rewind and restore previous states if needed.

**git blame** is a command used in Git version control software to trace changes in a file and determine who and when changes were made. This command is particularly useful when you're trying to find out who introduced an error or a bug, or simply to understand the history of the modifications to a specific piece of code.

**git blame <file>**

A screenshot of a computer

Description automatically generated

A commit represents a snapshot of the changes made to files in a repository at a specific point in time

Git large file system (Git LFS)

When you use git push --force, you force-push your changes to the remote repository regardless of whether others have updated it since your last fetch. This can lead to the unintentional loss of other developers' work.

In contrast, git push --force-with-lease is a safer alternative. It checks whether the remote branch you are pushing to has been updated by others since your last fetch. If the remote branch has been updated, the push is rejected, preventing you from overwriting other developers' changes unintentionally.

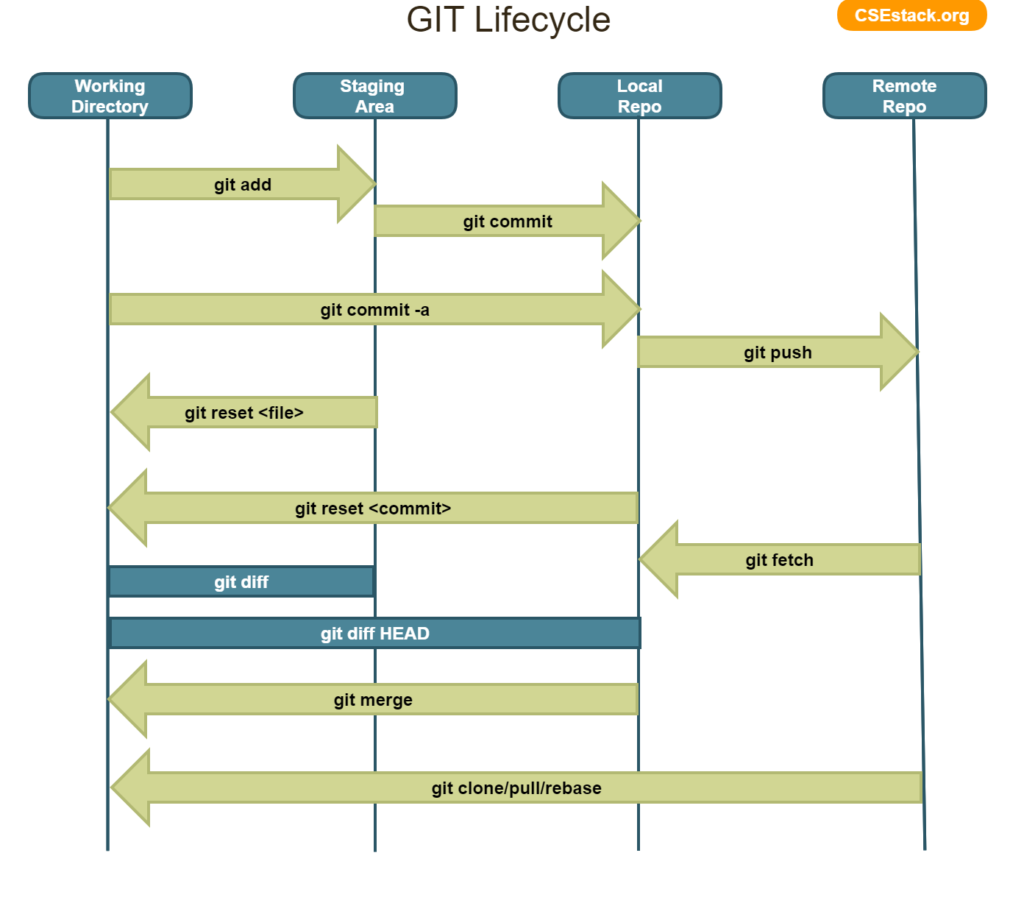
`HEAD` refers to the last commit on the current branch, `working tree` is the set of files in your directory, and `index` (or staging area) is a staging area for commits.

git log --graph

**Understanding Detached HEAD in Git:**

* **Definition:** A state where the HEAD pointer is not attached to any branch.
* **Occurrence:** Can occur when checking out a specific commit, tag, or file version.
  + git checkout <commit-hash>
* **Significance:** Allows you to explore different parts of your Git history without affecting the active branch.

**Undo changes -** <https://youtu.be/3dk3s4LK-Wg?si=dM53Cr1rCoCK4s09>



C# - [C#.NET Tutorials For Beginners - Dot Net Tutorials](https://dotnettutorials.net/course/csharp-dot-net-tutorials/)

Design Patterns: <https://www.geeksforgeeks.org/software-design-patterns/>

Iterator Pattern

