GIT

What is Git?

Git is a widely used modern **version control system** for tracking changes in computer files. The term version control system suggests a **system that records all the changes made to a file or set of data**, so a specific version can be considered whenever needed. This feature makes the process of collaboration so feasible with all team members, making it considerably more comfortable to work over a big project.

Git makes it possible for several people involved in the project to work together and track each other's progress over time. In software development, the tool helps in Source Code Management. Git favors not only programmers but also non-technical users by keeping track of their project files.

* Git is used to tracking changes in the source code
* It allows multiple developers to work together
* It supports non-linear development through its thousands of parallel branches
* Any changes made to the source code can be tracked by others
* There is regular communication between the developers
* Every developer has an entire copy of the code on their local systems

While working on Git, we actively use two repositories:

* Local repository: The local repository is present on our computer and consists of all the files and folders. This Repository is used to make changes locally, review history, and commit when offline.
* Remote repository: The remote repository refers to the server repository that may be present anywhere. This repository is used by all the team members to exchange the changes made.

Both repositories have their own set of commands. There are separate Git Commands that work on different types of repositories.

Commands

Create repositories (git init)

**git init**

* The command git init is used to create an empty Git repository.
* After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins

Make changes (add, commit, status)

**git add**

* Add command is used after checking the status of the files, to add those files to the staging area.
* Before running the commit command, "git add" is used to add any new or modified files.

**git commit**

* The commit command makes sure that the changes are saved to the local repository.
* The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

**git status**

* The git status command tells the current state of the repository.
* The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

Parallel development (branch, merge)

**git branch**

* The git branch command is used to determine what branch the local repository is on.
* The command enables adding and deleting a branch.

# Create a new branch  
  git branch <branch\_name>

# List all remote or local branches  
  git branch –a

# Delete a branch  
  git branch -d <branch\_name>

**git merge**

* The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch.
* It is used to merge the changes in the staging branch to the stable branch

git merge <branch\_name>

Sync repositories (push, pull)

**git push**

* The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.
* The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

git push -u origin master

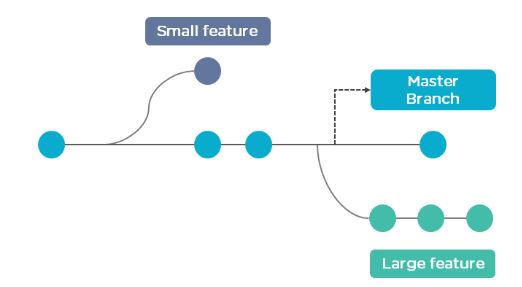
**git pull**

* The git pull command is used to fetch and merge changes from the remote repository to the local repository.
* The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

git pull <branch\_name> <remote URL>

Branches

Branch in Git is used to keep your changes until they are ready. You can do your work on a branch while the main branch (master) remains stable. After you are done with your work, you can merge it with the main office.



The above diagram shows there is a master branch. There are two separate branches called “small feature” and “large feature.” Once you are finished working with the two separate branches, you can merge them and create a master branch.

Tags

Git Tags are specific reference points in the Git history. Git tags are used to capture the specific point in the history that is further used to point to a released version. A tag does not change like a branch. They don’t have a further history of commits after being created.  Most people use this feature to mark some release points like (v1.0,…v4.0, and so on). In simple words, Git Tags are used to give some meaningful name to a particular in the git project repository.

There are two different types of Git tags:

1. Annotated tags

The annotated tags are saved as a full object in the database of Git. These types of tags store some extra metadata information such as the name of the tagger, tagger email id, and date. Annotated tags stores with a tagging message. It is best practice suggested in git is to store git tags in the form of annotated tags over lightweight. Using the annotated tags, you can store all the associated meta-data in the database.

To create an annotated tag, open the terminal application by pressing Ctrl+Alt+t and run the following command:

*$*git tag -a 'Release\_1\_0' -m 'Tagged basic string operation code' HEAD

In the above command, we have tagged the current HEAD by using the git tag command. The user provides a tag name ‘Release\_1\_0’ with the -a option, and the tag message is provided with the –m option.

1. Lightweight tags

This type of tags is used for ‘bookmarks’ to a commit; Lightweight tags are just a name or a specific pointer to a commit. Lightweight tags are useful for quick link creation to relevant commits.

The following command is used to create lightweight tags:

$ git tag <tag\_name>

Commits

Commits are the core building block units of a Git project timeline. Commits can be thought of as snapshots or milestones along the timeline of a Git project. Commits are created with the **git commit** command to capture the state of a project at that point in time. Git Snapshots are always committed to the local repository. Git doesn’t force you to interact with the central repository until you’re ready. Just as the staging area is a buffer between the working directory and the project history, each developer’s local repository is a buffer between their contributions and the central repository.

Hooks

Git hooks are scripts that run automatically every time a particular event occurs in a Git repository. They let you customize Git’s internal behavior and trigger customizable actions at key points in the development life cycle. Hooks can reside in either local or server-side repositories, and they are only executed in response to actions in that repository

Stash

In Git, the stash operation takes your modified tracked files, stages changes, and saves them on a stack of unfinished changes that you can reapply at any time. If you want to switch branches for customer escalation, but you don’t want to commit what you’ve been working on yet; you can stash the changes. To push a new stash onto your stack, run the **git stash** command. Now you can safely switch the branch and work elsewhere. We can view a list of stashed changes by using the **git stash list** command.

Sources:

<https://www.simplilearn.com/tutorials/git-tutorial>

<https://linuxhint.com/use-git-tags/>

<https://www.atlassian.com/git/tutorials/saving-changes/git-commit>

<https://www.atlassian.com/git/tutorials/git-hooks>

<https://www.tutorialspoint.com/git/git_stash_operation.htm>