GIT REPORT

Joseph Sony
Dept. of Computer Science and Engineering
MIT Manipal
Manipal, India
josephsonychirattavelil@gmail.com

***Abstract*— Git is a version control system that is spread out and maintains track of changes to code. It lets many work on the same project at the same time. It enables developers to keep track of all their work, make branches for different features, and mix changes from different people. Git is quick and reliable because every developer has a complete copy of the project's history on their own computer. Version control in software development is now majorly done with this technology, which is employed by platforms like GitHub, GitLab, and Bitbucket.**

***Keywords*—git, version control, GitHub, git commands, git method**

.

ABOUT GIT

GIT is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It was originally created by Linus Torvalds in 2005 for the development of the Linux kernel.

Different Terms Used In GIT

A. *Repository (Repo):*

A storage location containing all project files and their version history, either local or remote (e.g., GitHub).

B. *Working Directory:*

The local files and folders currently being edited..

C. *Staging Area:*

An intermediary space to prepare and select changes from the working directory before committing.

D. *Commit:*

A recorded snapshot of changes in the repository, identified by a unique hash and message.

E. *Branch:*

An independent line of development used for features or fixes without altering the main codebase.

F. *HEAD:*

A reference to the latest commit in the active branch, showing the current position in the history.

G. *Remote:*

A server-hosted copy of the repository for collaboration and backup (commonly named *origin*).

HOW GIT WORKS

Git is a tool that lets developers keep track of changes in their projects and manage them effectively. A local repository is set up on each user's computer, allowing it to sync with a remote repository on platforms such as GitHub. Changes are first saved in the working directory, then they get added to the staging area, and finally, they are committed with a message that describes the update. Git allows you to create branches so you can work on different features separately without messing up the main code. The branches can be combined with the main branch later on. Developers use commands such as push and pull to share and update their work across different repositories. Git makes sure that data stays intact by using unique identifiers for every change, which helps with teamwork and keeps version control reliable.

GIT COMMANDS

1. ***git config --global user.name "<YourName>"***

   Sets your username in Git so that your commits are associated with your name.

2. ***git config --global user.email "<YourEmail>"***

   Defines your email address for Git commits, typically the one linked to your GitHub account.

3. ***git init***

   Initializes a new Git repository in the current directory.

4. ***git add <FileName>***

   Stages specific files for the next commit. Using . instead of a filename adds all modified files.

5. ***git reset <FileName>***
   Removes specific files from the staging area. Using . removes all staged files.

6. ***git status***

   Displays the current state of the working directory and staging area, showing which files are modified, staged, or untracked.

7. ***git commit -m "<CommitMessage>"***

   Commits staged changes with a descriptive message, creating a new snapshot of the repository.

8. ***git push***

   Uploads local commits to a connected remote repository. Not required for local-only projects.

9. ***git push --force origin main***

   The command is used to forcefully overwrite the main branch on the remote repository with the local main branch's history. Normally, GIT prevents pushes that would result in a non-fast-forward merge, meaning if the remote branch has commits that your local branch does not, a regular git push would fail to prevent accidental loss of history. The --force flag overrides this safety mechanism, making the remote main branch's history exactly match your local main branch, regardless of any diverging commits on the remote. This can lead to the loss of commits on the remote repository if those commits were not present in your local branch's history.

10. ***git reflog***

    Shows a record of all recent changes to the HEAD and branch references.

11. ***git fetch***

    Downloads changes from a remote repository without merging them into your current branch.

12. ***git branch <BranchName>***

    This command creates a new branch with the specified name. Branches are useful for working on new features without affecting the main codebase

13. ***git merge <Name>***

    Merges the specified branch into the current one, combining their commit histories.

14. ***git pull***

    Fetches and merges updates from a remote repository into your current branch...

15. ***git checkout <Name>***

    Switches to the specified branch or commit and updates the working directory accordingly.

16. ***git clone <URL> <Link>***

    Copies a remote repository to your local system, optionally specifying a target directory.

17. ***git checkout <Name>~<Number>***

    This checks out a commit that is a certain number of steps before the specified branch or commit.

18. ***git rebase <Target> <Name>***

    This command rebases the given branch <Name> onto the target branch <Target>. It reapplies commits from <Name> on top of <Target>.

19. ***git reset HEAD~<num>***

    This resets your current branch to the state of an earlier commit. The number specifies how many commits to go back.

20. ***git revert HEAD***

    This creates a new commit that undoes the changes made in the last commit, without modifying history.

21. ***git cherry-pick <NamesOfCommits>***

    This applies specific commits from one branch onto the current branch.

22. ***git rebase –i***

    This launches an interactive rebase session starting from the current HEAD, giving you control over recent commits. This allows you to interactively edit, reorder, or drop commits from the last specified number of commits.

GIT METHODS

1. ***Branching***

   Allows developers to create independent lines of development for features, fixes, or experiments without affecting the main project.

2. ***Merging***

   Combines changes from one branch into another, producing a merge commit that unites their histories.

3. ***Resolving Merging Conflicts***

   Merge conflicts occur when changes from different branches affect the same lines of code. GIT will mark these conflicts, and the developer must manually resolve them before completing the merge.

4. ***Rebasing***

   Moves or re-applies commits from one branch onto another to maintain a cleaner, linear history. Should be used carefully with shared branches.

5. ***Pushing***

   Sends committed changes to a remote repository so others can access the latest updates.

6. ***Pulling***

   Pulling brings changes from a remote repository into the local repository. It can be thought of as a combination of `git fetch` and `git merge`.

7. ***Checkout***

Used to switch branches or restore specific versions of files

8. ***Collaborations***

Collaboration in GIT typically involves using remote repositories such as GitHub. Developers clone repositories, work on branches, push changes, and use pull requests for review and integration.

References

[1] Learn Git Branching: https://learngitbranching.js.org/

[2] YouTube – Git Tutorial for Beginners: https://www.youtube.com/watch?v=HVsySz-h9r4

[3] Official Git Documentation: https://git-scm.com/doc

[4] Pro Git Book: https://git-scm.com/book/en/v2