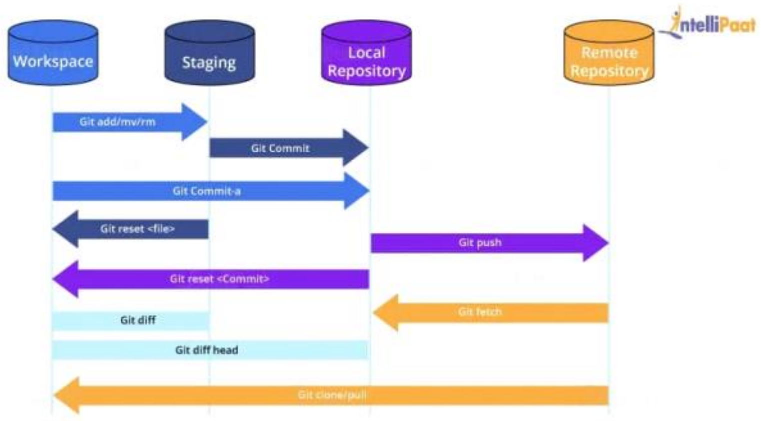
# Git-Basics

- **Introduction**
  - Documentation - [Git - Documentation (git-scm.com)]() & [https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud]()
  - SideNote: Readme.md is a readme file that can help provide information on the repository. It is created in mark down language. Pointers of how to can be found @ [How to make the perfect Readme.md on GitHub | by Sayan Mondal | The Startup | Medium]()
- **General**
  - Version Control : The ability to keep track of changes to a single file or object.
  - Current Version Control Software enable tracking of changes as well as provide ability to have multiple resources work on the same set of files.
  - Examples - subversion, CVS, Github, GitLab, BitBucket, VSS etc.
  - VCS = Version Control System
  - Two types :
    - Central VCS - No local copy, resources work directly with central copy (eg. Subversion)
    - Distributed VCS - There is a local copy. Resources work with local copy and this local copy is sync'd with central copy. (e.g. Git)
- **Git**
  - Git is a utility set / software that enables versioning and central repository for collaboration.
  - Github, Gitlab & Bitbucket are cloud services examples that implement Git utility set (of course with variations). Detailed comparison is available @ [Git Storage: Comparing GitHub, GitLab, and Bitbucket (seesparkbox.com)]()
  - Communication to these services can be made through http, ssh, git or file protocols.
- **Git Architecture (high level)**
-


- **Terminology**:
  - Repository - The place where the code or files are located. (called repo for short). Usually end with .git extension.
  - Local Repo - Local copy of files (also called working directory)
  - Remote Repo - Remote / central copy of code or files. Like Github or GitLab or other locations. (some times referred to as origin)
  - Commit - The process of adding the file with changes back to Repo. (usually with comments on what changed)
  - Commit ID - Each commit is tracked with a unique ID, which is a SHA1 hash. The first 4 digits should be enough to identify a commit ID.
  - Clone  - Make a copy of  (usually done when trying to first copy the central repo to local repo).
  - Branch - Logical label of a point in repository (can be thought of a logical partition of the folder ).
    - NOTE: File created while in that branch will not be visible outside the branch, unless they are merged or rebased
  - HEAD - Latest commit point.
  - HEAD~1 - Last but one (next to last) commit point.
  - Web Hook - A feature provided by the services leveraging Git to provide push notifications based on updates in the repository. Usually used for triggering CI/ CD workflows automatically when a repo is updated.
  - Untracked file or folder - Files/ folders that are flagged as changed but not added to staging . Hence cannot be  pushed to repo.
  - .gitignore - A text file in the repo that says to ignore files/ directories for tracking or comitting . File needs to be manually created / added.
- **Git Commands** (check git manual for complete list)

| Command | Purpose |
|---|---|
| $ git config | • Helps manage local or global configuration based on switch used.<br>• Helps add shortcuts (alias)<br>    • $ git config --global alias lga "log --graph --oneline"<br>        ○ Will allow - lga to be used instead of using the complete switch above. |
| $ git  init | • To initilize git in a particular director<br>• Creates a .git folder that contains the tracking |
| $ git add <file or director> | • Informs git to track the changes for specified file or directory (pushed to staging)<br>• Every time a change is made, for the file to be pushed it needs to be added to staging. |

| | |
|---|---|
| $ git add -u | • Adds all updated files to staging |
| $ git add -A | • Add all added / news files to staging |
| $ git status | • Tells files that are being tracked or not tracked.<br>• Provides info if a file is modified or not. |
| $ git diff | • Tells difference between two commits<br>• E.g.<br>    • $ git diff HEAD~1 .. HEAD (changes between the last 2 commits) |
| $ git log | • Shows log.<br>• There are multiple switches available like --graph, -- oneline etc.<br>• Other variation is $git shortlog (same as git log but with FORMAT = short)<br>• Just like git log, multiple switches exist for shortlog too. |
| $ git checkout <file> | • Reverts changes made to a file |
| $ git clean | • Removes newly added / modified files<br>• Switch<br>    • -n : notifies files cleaned<br>    • -f : force cleans |
| $ git commit | • Commits changes to the repository (local repo)<br>• Switch:<br>    • -m "message" : Provides inline commit comment (else the comment needs to be updated in editor) |
| $ git clone <repo Url> | • Copies the contents of the repo to the current directory.<br>• No need to run git init, if the directory was not tracked earlier. Will be auto tracked post cloning. |
| $git remote | • Provides details of the remote repository.<br>• Variation:<br>    • $git remote add origin <url for git rep> [Links local repo to remote repo] |
| $ git push | • Pushes committed changes in local repo to remote repo.<br>• Needs authentication. (UID/PWD for https / git). SSH Key for SSH connection.<br>• Variation:<br>    • $ git push origin <local branch> [pushes specific local branch to remote repo, creates branch if it does'nt exist]<br>    • $ git push origin <local branch>:<remote branch> [pushes specified local branch to specified remote branch]<br>    • $ git push origin : <remote branch> [with no local branch , with delete the remote branch] |
| $ git pull | • Syncs remote repo with local repo.<br>• $git merge origin/master or $git fetch dos the same.<br>• $ git pull <source repo identified > < source repo branch><br>    • $ git pull origin master [all changes from provided source gets synced to local ] |
| $ git tag | • Tags added to current branch (usually used for versioning)<br>    • -a <tag name > [adds tag<br>    • -s <tag name > [Signed tag -phass phrase required]<br>    • -v <tag name > [verifies tags]<br>• By default tags will not be pushed.<br>    • $ git push --tags [command to push with tags] |
| $ git branch | • Lists branches & shows current branch with a "*"<br>• Variations:<br>    • $ git branch -r [Lists remote branch details] |
| $ git branch <branch name> | • Creates a branch with specified name<br>• Variations:<br>    • $ git branch <branch name> <SHA1 Hash> [Creates branch from that particular reference point]<br>    • $ git branch -m <old name> <new name> [ renames branches]<br>    • $ git branch -d <branch name> [delete branch]<br>    • $ git branch -D <branch name> [Force delete branch] |
| $ git checkout <branch name> | • Switches to the specified branch.<br>• Variations:<br>    • $ git checkout -b <branch> [will create and switch to branch] |
| $ git merge <branch name> | • Merges changes in specified branch with the current branch (usually done when in master to merge child branch with master)<br>• Variation:<br>    • $ git merge origin/master . [merges remote modifications to local ] |
| $ git branch --set-upstream <local branch> <remote branch> | • Links the branches between local & remote repo |
| $ git stash | • Moves pending commit activities to a holding area<br>    • i.e during this stage, even if there are changes that need committed, they will not be showing up in $ git status .<br>    • Usually done if we want to switch branches without having to commit the work done.<br>• Further read : https://opensource.com/article/21/4/git-stash |

| | |
|---|---|
| | • Variations:<br>  • $git stash pop [ remove from stash ]<br>  • $git stash list [lists entries in stash ]<br>  • $ git stash apply [apply changes back to stash ]<br>  • $ git stash drop [removes the stash entry ]<br>  • $ git stash branch <branch name> [ Creates a new branch , checks out the branch and applies the stash to new branch , basically creating the stash in new branch instead of current branch] |
| $ git rebase | • Concept of cleaning up the commit log entries. Helps log history clean.<br>• Gets the updated master branch commits , making the branch look like it has the latest updates.<br>• It resets the commit head.<br>• Further read : https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase |
| $ git revert<br>$ git reset | • Undo changes .<br>Further read : https://www.atlassian.com/git/tutorials/undoing-changes<br>• Example:<br>  • git reset --hard a1e8fb5 [the commit history is reset to that specified commit, basically rolling back upto that commit point and removing the commit entries in log. Keeps log clean]<br>  • $ git revert HEAD [Git will create a new commit with the inverse of the last commit. This adds a new commit to the current branch history. So it reverses the specified commit but also makes a log saying it reverted it. Log will look a little clumsy ] |

- **Other Info:**
  - GitHub also has workflow facilities that allows actions / workflow to be triggered based on certain action in the repository.
  - Note: Resent developments have extended the features of the Git tools beyond source code management and are extending the capabilities like Continuous Integration etc.
    - Some examples of Continuous Integration Tools: https://benmatselby.dev/post/build-tool-comparison/
      - Jenkins
      - GitLab CI
      - Git Hub Actions
      - Bitbucket Pipelines
      - AWS Code Pipeline
      - Azure DevOps (Azure Pipelines)