**Git:** is aprimarily a **version control system** in software development.A **tool to manage source code history.**

It serves **2 main purposes:**

* **Code Backup**
* **Code Versioning**

**Git** VCS mainly runs at the command line of the local machine. Git employs a powerful branching system.

Git keeps track of the files and modifications to those files in a repository but only when we tell it to do so. [ which file to track and when to take a “snapshot” of any modification]

**Git Servers**: GitHub, Bitbucket, GitLab, etc. To share your repositories will require Git Server. **GitHub:** is a **code hosting platform** for **git repositories.**

Is a website that allows us to publish git repositories online. Which can be useful for many reasons.

* Provides backup of file.
* Visual interface for navigating the repositories.
* People can navigate make repo collaboration.
* Provides light weight issue tracking system.

[\*Note: Every commit takes a snapshot of the changes that commit includes both a description of changes and line by line details of those changes].

There are 3 main states that a file can reside in local repository:

* Committed – data is safely in the local repo.
* Modified – changed but not committed.
* Staged – marked modified file in current version to go into next snapshot.

**Git client/git bash:** Git bash/client comes as installer for operating systems.

* Once we have git bash we will get command line utility, using it we can use git commands to work with repositories.

**First time configuration to use git**:

* git config - - global user.name “username” > to set git bash user account.
* git config - - global user.email “email id” > to set email account for user.
* git config -list > to check the configuration

**Working with files:** to determine which files are in which is the status is the git status command, means which file is already been committed and which haven’t.

* git status --> to check the status of the files tracked or not.

the below msg indicate – when all files here already been committed and pushed.

on branch master

nothing to commit (working directory clean)

the below msg indicate – if we add a new file to the project and the file didn’t exists before.

on branch master

untracked files

* git init --> this creates a new subdirectory named (.git) that contains all the necessary repository files.

At this **point nothing** is in the **project** are not **tracked yet**. By **adding and committing** will **start versioning** the existing files you should **start by tracking** those files and **do initial commit**.

**Adding and Committing:** to **start versioning existing files** we should **start by tracking** that file do an **initial commit**.

[Note: When we create git repository, it will create master branch by default.]

* git add <filename>--> add a specific file to index/stage local repository or git to track.
* git add . --> add file to index/stage in local repository.
* Git add -A --> add all files in current directory.
* Git add -p --> choosing what changes to add goes through all files ( with ‘y’ ‘n’ options).

[ Note: “**index**” holds a **snapshot** of the **content** of the **working tree**.]

* git commit -m “message”-- > will start tracking the files.
* git log > show the all commits.
* git log –pretty=oneline > shows in commit history in one line.
* git remote add origin <url of github server> > to add origin
* git push -u origin master (-u upstream linking) push for first time.

[Note: time stamp of the commit is displayed but not when we had pushed.]

**SSH configuration using ssh Keys**

* ssh-keygen -t rsa -C <emailid>

remember the passphrase provided

first copy the public key.

Next copy and paste in the ssh key in github account settings.

[Note: if we configure ssh key, there no need to use upstream link (-u) ]

* ssh -T [git@github.com](mailto:git@github.com) >to check ssh keys working properly
* git clone <ssh url of repository> > clones the remote repository to local machine.

**Ignoring files and folders by git:**

We do by specifying that specific filenames in the .gitinore file and put it into root folder of the git project.

Sample.txt > igores specific file.

\*.txt > ignores all files with given extension.

Test/ > ignores complete folder.

**Branching and Merging:**

**Master branch** of the **repository** should **always contain working and stable code.**

Branches are used for keeping bug fixes and feature work separate from the master branch (production) branch. When the change is ready, they merge with these branches into master branch.

**Branching** allows to work in a **separate copy** of your **code without affecting** the **master branch.**

When we **create a branch**, a **complete clone copy** of your **repository** is **created** under new name. then we modify the code in this new branch independently including committing file and such on.

Once the new features have been fully integrated and the code is stable then we will merge into master branch.

* git branch <branchname> >to create a branch.
* git checkout <branchname> >to switch to branch.
* git checkout -b <branchname> >to create and switch to branch.
* git branch -v > list out branches.
* git merge <branchname> > to merge branch with master.\*we must be master branch.
* git push -u origin master “comments” > to push local mater branch to remote.
* git push -u origin <branchname> > to push local branch to remote.
* git delete -D <branchname> > to delete local branch.
* git push origin – delete <branchname> > to delete remote branch.
* git rebase <branchname> > similar to merge but it is pull + merge.
* git rebase -i <develop branch> > we can fix up commit history.

**Advanced branch commands:**

* git branch -a > to view all current branched for the repository, (including local & remote)
* git -a –merged > to view branches that have been merged into current. (local & remote).
* git -a –no—merged > to view all branches haven’t merged.
* git – list > lists all branches

**Fixing mistakes and back tracking:**

* git reset HEAD > switch to the version of the code.
* git reset HEAD -- <filename> > most resent commit of specific file.
* git reset HEAD^ -- <filename> > switch to the version of code before most recent commit.

**Search for word in repo:**

* git grep ‘wordsearch’ > search for part of string in a directory.
* git grep -n ‘wordsearch’ > search for part of string prints line number

**Seeing who write what?**

* git blame ‘<filename>’ > shows alteration history of file and author.
* git blame ‘<filename>’ -i > with author and commit ID.

**Logging:**

* git log > list all commits in a repository.
* git log -P > list of all commits showing commit messages.
* git log -S ‘word’ > list of commits with particular string.
* git log – author ‘name’ > list all commits by author.
* git log – oneline > oneline display.
* git log –since = yesterday > shows commit since yesterday.
* git log –grep “item” –author “name” > search item by author.

**Remote and Backup:** create account and create a new repository on GitHub. **Copy the link of the repository** to **add it to the origin of the local git repository** where the code is stored.

* git remote add origin https//github.com//………….. .git
* git push origin master

**Advanced add commands:**

* git add <filename>
* git add -A > add files in current directory
* git add . > all file changes in current directory
* git add -p > choosing which changes to add. [ this will go through all your changes and you can say ‘y’ or ‘n’ to the changes.

**Advanced commits commands:**

* git commit -m “message”> typically used for shorter messages
* git commit <filename> -m “message” > add files and commit in one shot.
* git commit -am “message” > add file and commit staged file
* git commit - - amend “new message” > change the most recent commit message.

**Undoing the changes locally.**

[Note: The below **checkout** command undo only if we have not performed **add .** and **commit**]

* git checkout <filename> > will undo one file
* git checkout <commitId>
* git checkout . > will undo all fines in working directory.

[Note: The below **reset** command undo after we perform **add .** and **commit**]

* git reset - -soft HEAD -1 > undo only last commit/we can to till last 5 commits.
* Git reset - -mixed HEAD -1 > reset index but not working tree
* Git reset - - Hard Head -1 > reset the index and working tree.
* git reset - - soft < commit id> > to undo changes to commits.

Then do add . and git commit -m “message”

**Undoing the changes in Remote repo.**

[Note: never rewrite history in remote repo. Instead go ahead of latest commit add a new commit ahead after doing undoing.]

* Git revert < commit id> -m “ message”

**To save local changes:**

* git stash save <name> > saves local changes with given name.
* Git stash save -u > to stash untracked files.
* git stash save – include-untracked > same as above.
* git stash list > lists all stashes.
* git stash apply stash@{1} > takes given stash id applies to repo.
* git stash pop stash@{1} > is similar to apply but it deletes the stash from after it Is applied.
* git stash branch <bname> stash@{1} > creates new branch with latest stash and deletes stash.
* git stash clear > deletes all stashed made in repo and may be possible to revert.
* git stash drop > deletes all stashes cannot be reverted.
* git stash drop stash@{1} > deletes only give stash.

**Tags:** Are used to version the code and also used for tag- based deployments code version.

* Tags are used after **commits**, tag refers to HEAD. / Tags are given to commits
* Release purpose, based on tagging, we pull code in Jenkins / Tags mark a release point.

**Types of tags:**

* Lightweight
* Annotated (expressive message)
* git tag <version name> > creates a lightweight tag.
* git tag -a <versionname> -m “message” > create annotated tag
* git tag <tagname> commit id > create tag for older or given commit.
* git show <tagename> > to check particular tag.
* git tag -l > lists all tags.
* git push origin <tagename> > to push single to remote repo.
* git push origin –tags > to all tag at a time
* git tag -d <tagename> > to delete single tag.
* git tag -d $(git tag -l) > to delete all tags.
* git push origin -d <tagname> > to delete tag in remote repo.
* git push origin -d $(git tag -l) > to delete all tags.

**@@@@@Important points:**

**git log --oneline**

**git clone “url”** --> downloads an existing git repository to the local computer.

**git fetch origin:** really only downloads new data from a repository but doesn’t integrate any of this new data into working folder ( never manipulate of destroys anything.

**git pull origin master:** will download new data from repository and manipulate the data in working folder.

**git merge and git rebase :** used to integrate changes from one branch to another branch but there is a slight difference the way of integration. Rebase rewrites committing history and results in the straight-line committing history.

**Very important git command to resolve push:**

**Git pull origin master – - allow-unrelated-histories** 🡪 if it rejects push

**Use escape and :wq** ----> to exit from commit screen.

**git rebase -i** -- > combine a sequence of commit together into simple one. You might use this to organize a messy commit history.

Local repository consists of three trees maintained by git.

* Working directory.
* Index (stage)
* Head (which points to the last commit you’ve made)
* pwd > to check present working directory
* mkdir <dirname> > to create a directory
* rm -Rf > to delete directory.
* cd <dirname>/ > change to the directory.
* touch sample.txt > linx command
* notepad sample.txt > to open fine in notepad.