git Basics

PREPARED BY

Aishwarya Verma

What is git

About GIT

- Git is a **Distributed Version Control System**.
- Git was created by, **Linus Torvalds** on 7th **April 2005**, on Linux **Kernel**.
- Git is a free software for tracking changes in any set of files.
- Usually it is used for coordinating work among programmers for collaboratively developing source code during software development.

Advantages of using git

- Easily recover files.
- Track who has introduced an issue and when.
- Rollback to previously working stat.

History of Version Control System (VCS)

1. **Local VCS**
   - Use of local database to keep track of files saved in local hard disk.
   - Pros
     - Can track files and rollback.
   - Cons
     - If hard disk is lost, everything is lost.
     - Collaboration issues.

2. **Centralized VCS**
   - Files tracked are saved in a central server
   - Pros
     - Files can be pulled and changes can be pushed to the server
   - Cons
     - If server is damaged, rollback to previous state is difficult.

What is git

3. Distributed VCS

- Files tracked are saved in a central server, with all the history details.
- When files are pulled, all the file tracking details are also saved in working computer. i.e., full backup is pulled.
- Programmers can collaborate effectively.
- Feature to ignore files which are not to be tracked.
- If the server gets damaged, we still have the full backup with tracking history of the project, which was pulled from server previously.
- Rollback to previous changes is very easy.

GIT Features

- It captures snapshots, not the differences.
- **.git** file, is the one which contains all the project history.
- Almost every operation is local (no internet required)
- i.e., all operations are performed locally and after that changes can be pushed to a remote repository (centralized server).
- The goals include speed, data integrity, and support for distributed, non-linear workflows.
- Git has integrity i.e., all the changes are tracked and need approval for merging.
- It uses SHA1 checksum to validate the files.
- Git generally only adds data.

Git Hosting Platforms

- Github, Bitbucket, Gitlab are a few popular hosting platform of git repositories

What is git

GIT Installation and Setup

- Official link to download git software - https://git-scm.com/downloads
- Git installation comes with -
  - git command line tool
  - git bash (a terminal program)
  - git GUI (Graphical User Interface), which uses command line tool in backend.
- After git installation, to check if it is installed successfully
  - make sure to check the checkbox -> Launch Git Bash
- Git commands can be used via windows power-shell also.
- But using git commands without installing git, will lead to errors.
- When the git bash terminal opens, you can use the below command -

```
$ git
```

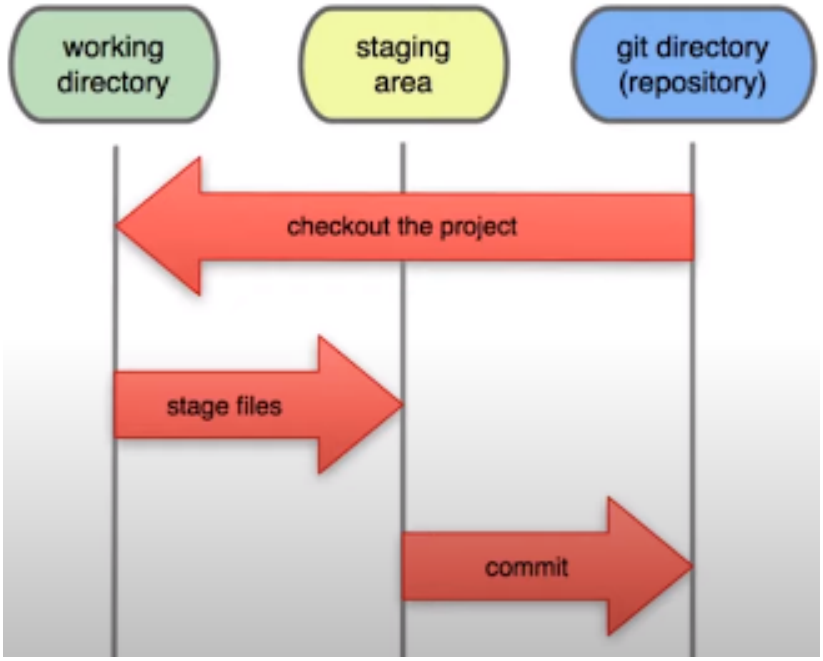
- It gives overview of git and about its usage.
- To increase font size in git bash, use --> Ctrl++

Git - Three stage architecture

1. Working Directory
   - The path of the project folder currently working on and on which version control is needed.
2. Staging Area
   - It has the files which the programmer wants to commit to the new version in remote repository
3. Git Repository
   - Here the staged files are committed to the git repository(repo), which stores the new version of the files having the changes done.

What is git

Git - Three stage architecture



Git - File status lifecycle

- Repository (repo)
  - The folder or directory, in which the files related to our project is being tracked using git, also where the '.git' file is stored.
- Un-Tracked
  - The files of the which are not added to .git, i.e., files under repo which are not being tracked.
  - To specifically put some files of the repo untracked, we can put them under .gitignore file.
- Un-Modified
  - The files of the repo, which are being tracked, and are not yet modified.
- Modified
  - The files of the repo, which are being tracked and are modified.
- Staged
  - The files of the repo, that are modified and staged for next commit.
  - Once the staged files are committed, the status of the file changes to unmodified.

Basic Linux commands

1. pwd

- Shows the path of present working directory

```
$ pwd
```

2. cd

- Change directory path

```
$ cd Desktop
$ cd /c
```

3. ls

- lists the files/contents present in current directory

```
$ ls
```

4. touch

- It is used to create a file without any content, i.e., it is empty.
- This command can be used when the user doesn't have data to store at the time of file creation.

```
$ touch error.log
```

5. rm

- This command deletes the file with the name specified.
- If the file name is ".git" and it is deleted using this command, the project folder will no more be tracked and so it will no more be a git repo.

```
$ rm -rf .git
```

Git Commands

1. git

- Gives overview of git and about its usage

```
$ git
```

2. git config

- use it to change or display the git configuration setup
- below command lists all the configuration parameters

```
$ git config --list
```

- use below command, to change configuration details

```
$  git config --global user.name "Harry"

$ git config --global user.email "harry@codewithharry.com"
```

- to display configuration details, use

```
$ git config user.name

$ git config user.email
```

- below command can be used to change configurations of any editor

```
$ git config --global core.editor vim
```

- use config to add alias in git

```
$ git config --global alias.st status
```

--> this will add "st" as alias/short-hand notation for status

--> now we can use "git status" as "git st"

Git Commands

2. git config - alias continued

- use config to add alias in git

```
$ git config --global alias.ci commit
```

--> use "git ci" and it works same as "git commit"

```
$ git config --global alias.unstage 'restore --staged--'
```

--> use "git unstage filename.txt" and it works same as "git restore --staged filename.txt"

```
$ git config --global alias.last 'log -p -1'
```

--> use "git last" and it works same as "git log -p -1"

3. git status

- It displays the status of the current directory.
- If the current directory/repository(repo) is enabled with git tracking, it shows the status of the files under the directory.

```
$ git status
```

--> by default, the branch on git repo will be master

- if the current directory/ repository (repo) is not enabled with git tracking, it gives fatal error, showing that the current directory is not a git repo.
- it will not show status of empty folders created under project repo.
- it is the most common command and must be used after every git operation, to avoid any errors.

Git Commands

4. git init

- use it to make your current directory a git repo
- this command initializes current directory/folder as git repo and starts tracking the directory.

```
$ git init
```

5. git add

- use it to enable tracking on files in the current directory
- it will move the files to the staging area, and the files become ready to commit files.
- it is also used to resolve merge conflicts.

```
$ git add --a

$ git add .
```

--> to add all untracked/modified files to the staging area

```
$ git add filename.txt
```

--> to add a particular file to the staging area

6. git commit

- use it to commit the changes (staged files) to the project repo (local)
- to take a snapshot of the changed file.

```
$ git commit
```

--> it will open the vim editor, where you can use it to edit the commit message

Git Commands

6. git commit - continued

```
$ git commit -m "sensible commit message, changes in prog.py"
```

--> commit the changes with the commit message

```
$ git commit -a -m "Direct Commit"
```

--> to commit without the staging, i.e., skipping the staging area
--> i.e., it will commit the changes directly, without adding them to staging area.
--> Using this, you can skip stage area with only tracked file.
--> You can't skip stage area if your file is not tracked.
--> So, first track the file then directly commit the file with appropriate message.

```
$ git commit --amend
```

--> this command is used to change the last commit message,
--> and also to add the current changes in the git repo to be added with the last commit.

- this command will open a vim editor, so below are some quick usage commands:

  1. use "**i**" to edit the message
  2. use "**esc**" to switch to read mode
  3. use "**:wq**" to save changes to the commit and quit vim editor

Git Commands

7. git log

- use it to get the history of commits performed on the project repo.

```
$ git log -p
```

--> this will show all the logs with the differences made via each commit.

```
$ git log -p -<number>
$ git log -p -3
```

--> any number after "-p" will show that many number of commits.
--> here we gave "-3", so it will show us last 3 commits with difference details.

```
$ git log --stat
```

--> gives short summary of the changes in each commit

```
$ git log --pretty=oneline
```

--> gives each commit details/logs/summary in one line

```
$ git log --pretty=short
```

--> gives each commit details in short
--> like it shows commit id, author and commit message

```
$ git log --pretty=full
```

--> gives each commit details in full, with some more details
--> like it shows commit id, author (the one who has created the file), committer (the one who has made changes to the file) and the commit message.

Git Commands

7. git log - continued

```
$ git log --pretty=format:"%h -- %an"
```

--> this will display the logs as per the format specified.
--> Here, "%h" is hash of commit id, and "%an" is the author name.
--> And many more can be found in git documentation - on searching for git pretty format examples.

```
$ git log --since=2.days
$ git log --since=2.weeks
$ git log --since=2.months
$ git log --since=2.years
```

--> shows all the commit details done in last 2 days (or specified timelines) for the respective git repo.

- ":" appears in last as pointer/cursor
- type "**q**" after ":" --> to exit/quit from the logs

8. rm -rf .git

```
$ rm -rf .git
```

--> a linux command to remove(delete) file

- this command is used to delete **.git** file from your git repo.
- use it when you want to stop tracking your project repo.
- as **.git** stores all the commits and snapshots of tracking, removing the file also removes all the tracking details of your project repo.
- After executing this command, your project repo will no more be a git repo. Deadly command 😵, think before use!!

Git Commands

9. git clone

- to clone a repo from a git hosting website. ex- github/gitlab/etc.

```
$ git clone <full url/link of repo to be cloned>
```

--> the branch cloned to local directory is by default the master branch.

```
$ git clone -b <branchname> <remote-repo-url> <targetFolderName>
```

--> it will clone the git repo from a specific branch
--> and it will create a folder with specified targetFolderName and save the cloned git repo in the target folder.

10. git diff

```
$ git diff
```

--> it compares the files in staging area with the modified files in working directory and shows the differences between the files.
--> if there are no modified files in the working directory and all files are staged, then this command displays nothing.
--> This is because, there is no staged file, which is modifed.

```
$ git diff --staged
```

--> it shows all the changes/differences of the files in the repo, which were made after the last commit to the repo.
--> i.e., this command will compare the previous/last commit with the current staging area

Git Commands

11. ".gitignore" file

- to create ".gitignore" file in your project code, use below command-

```
$ touch .gitignore
```

- now add the file/folder names into the **.gitignore** file, which you do not wish to track under your project repo.
- now you can check **git status** of your project repo, and then you will find only **.gitignore** shows up as untracked, from the unwanted files list.
- now use the command **git add .** (or) **git add --a** , to track your .gitignore file
- and commit the file using **git commit -m "Added .gitignore file"**

- if you want to add file/folder names matching a pattern, which are to be ignored, that also can be done.

--> to ignore many files with extension ".log", then in **.gitignore file** enter: ***.log**

--> to add a folder in **.gitignore file**, enter:

- **someFolder/** --> this will ignore all the folders with name "someFolder", which may be present any where in your git repo.
- **/randomFolder/** --> this will ignore the folders with name "randomFolder", which are **not** nested into (or) inside any other folder within the git repo.
- **static/justFolder** --> this will ignore the folders with name "justFolder", which is present under the folder named "static".

--> **newFile.json** --> this will ignore the file with name "newFile.json".

Git Commands

11. ".gitignore" file -- continued

- if the files specified in the **.gitignore file**, are modified, then the status of the modification will be shown in git status.
- but this must not happen as we do not want to track those files.
- so to overcome this, we have to explicitly untrack the particular file, by using the next command.

12. Delete/Remove a file using git

```
$ git rm third.txt
```

--> this command not only deletes the specified file, but also adds changes to the staging area (check by git status), which can be directly committed.

```
$ git rm --cached newFile.json
```

--> this command will **not delete** the file,
--> instead it will remove the file from being tracked, i.e., git will no more track the file.

13. Moving and Renaming Files in Git

```
$ git mv first.txt first_renamed.txt
```

- basic cut and paste with new name operation
- this command is used to move and rename the file using git
- it also adds changes to the staging area (check by git status), which can be directly committed.

Git Commands

14. git restore

```
$ git restore --staged
```

--> use this command to unstage files in git.

```
$ git restore newFile.txt
```

--> this command will discard all the recent changes made to the specified file and make it as it was in last commit.

```
$ git restore --staged newFile.json
```

--> this command will remove the specified file from the staging area, and make it unstaged.

15. git checkout

- this command is used to unmodify files in git.
- also to do some git branch related operations.

```
$ git checkout -- newFile.json
```

--> this will bring the previous changes in the specified file. (previous committed content)
--> it is useful when you have made changes, but want to get back to previous content in the file.

```
$ git checkout -f
```

--> this command will rollback to previous commit changes of your project repo, replacing all current modifications.

- **Note:** the modified content can not be restored, after bringing the previous changes in the file, if your file(s) is(are) not committed.

Git Commands

15. git checkout -- continued

```
$ git checkout -b <branchName>
```

--> this command will create a new branch, with the specified name in the project repo.
--> it also changes the current working branch to the specified branch.

```
$ git checkout <branchName>
```

--> It will change the current working branch to the specified branch
--> And respective branch changes will appear in the working directory (project repo)

16. git remote

```
$ git remote add origin git@github.com:gitHubAccount/repoName.git
```

--> this command will add a path to existing repo from any of the git hosting website like github/gitlab, etc., to your local machine (currently working on).
--> here, "**origin**" is the name of the url/website of the remote server.

```
$ git remote
```

--> it shows the names of the remote server added.

```
$ git remote -v
```

--> shows the name of remote server with website/url details.

Git Commands

17. git branch

```
$ git branch
```

--> this command will display the branches in the current git repo.

```
$ git branch -v
```

--> this will show the branches and the last commit details of the branches.

```
$ git branch --merged
```

--> this will show already merged branches (generally merged with the main/master branch)

```
$ git branch --no-merged
```

--> this will show branches that are not merged with the main branch.

```
$ git branch -c <branchName>
```

--> this command will create a new branch in the working directory (repo) with the specified name
--> and it will be replica of current branch in your repo (till last commit)

- Branching Workflow -- Broadly classified into 2 types of branches:
  1. Long running (requirement related) Branches
  2. Topic (Issue related) Branches

Git Commands

17. git branch -- continued

- use below commands to delete the branch, which is usually done when the feature/issue is closed.

```
$ git branch -d <branchName>
```

--> it will give error if the specified branch is not merged to the main branch

```
$ git branch -D <branchName>
```

--> it will give **Not** give any error and delete the specified branch, even if the specified branch is not merged to the main branch.

18. git push

```
$ git push -u origin master
```

--> this command will push the changes done in the local project git repo (currently working) to the remote repo under the master branch.
--> it will show error if sufficient right access is not present.
--> "origin" is the target(remote) repo and "master" is the target branch.

```
$ git push origin <branchName>
```

--> this will push the changes to the origin(repo) by creating a new branch specified (if non-existing) from the local branch.

- **tip**: please switch to the branch (working directory) you are currently working on and want to push the changes.
- So that push to the remote repo to the specified brach will be done without any error.

Git Commands

18. git push -- continued

```
$ git push origin <Current Working branchName>:<new branch>
```

--> this will push the files with all changes from the current branch to the remote repo under the newly created branch with the new branch name specified.

- **Note**: here, the new branch will be created only under the remote repo and not in the local working directory.

```
$ git push -d origin <branchName>
```

--> this command will **delete** the branch specified, from the remote repo (and not from the local repo).

19. git merge

```
$ git merge <branchName>
```

--> this command will merge current working directory with the specified branch.

20. git pull

```
$ git pull origin master
```

--> use this command to check if latest code is being pulled from the server, as specified from the remote master branch.

to be continued ... 😎