
git
COMMANDS

❖ SETTING UP A GIT REPOSITORY:

- A Git repository is a virtual storage of your project. It allows you to save versions of your code, which you can access when needed.
- To create a new repo, you'll use the **git init** command. It is a one-time command you use during the initial setup of a new repo.

Initializing a new repository:

$mkdir git-projects

$git init <project_directory>

$git init aws-project

$ls -a

NOTE: The **.git** contains all of the information that Git needs to track changes to your codebase, and it's an essential part of the Git workflow.

Now create a sample text file:

$vim sample

Add some lines…Save a file

$ls

GIT STATUS:

- It displays the state of the working directory and the staging area.

$git status

SAVING CHANGES TO THE REPOSITORY WITH ADD & COMMIT:

GIT ADD:

- It Add file contents to the index / staging area.
- It updates the index using the current content found in the working tree, to prepare the content staged for the next commit.

$git add sample

To add multiple files at a time:

$git add .

$git status

To list staging area / index files:

$git ls-files

GIT COMMIT:

- It records changes to the repository.
- Create a new commit containing the current contents of the index and the given log message describing the changes.
- When we commit, we should always include a message with **-m.**

  $git commit -m “First release of sample file”

  $git status

  The Staging Environment has been committed to our repo, with the message.

GIT COMMIT WITHOUT STAGE:

- sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment.
- The -a option will automatically stage every changed, already tracked file.

  Let's add a small update to sample file:

  $vim sample

  modify the file content

  $git status --short

  NOTE: Short status flags are:

  ?? - Untracked files **M** - Modified files

  A - Files added to stage **D** - Deleted files

$git commit -am “Updated sample file with new line”

$git status

GIT DIFF:

- It helps users understand changes made to files. Which shows the differences between various states of a repository.
- It helps developers see what changes have been made, whether they are between working directory and the staging area, between the staging area and the last commit, or between any two commits.

By default, git diff will show you any uncommitted changes since the last commit.

$git diff

Create a file:

$vim test

hlo...

rnraju4u

$git diff

$git add .

$git commit -m "New file is added"

$git diff

Modify the file:

$vim test

hlo...

rnraju4u**@gmail.com**

$git status

$git diff [Now it will show the difference of committed file & working file]

How to Compare Staged Changes in Git:

$git diff --staged / --cached

GIT RESTORE:

- It restores working tree files
- Restore specified paths in the working tree with some contents from a restore source.
- If a path is tracked but does not exist in the restore source, it will be removed to match the source.

To restore a file in the current directory:

$git restore filename

To restore all files in the current directory:

$git restore **.**

To restore a file in the index to match the version in HEAD:

$git restore –staged filename

$git status

GIT RESET:

- Reset current HEAD to the specified state
- reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

Modifying an existing file:

$vim filename

$git add filename

$git reset HEAD test.txt

$git status

To discard changes in working directory:

$git checkout -- test.txt

$git status

$cat filename

GIT MV:

- Move or rename a file, a directory, or a symlink.
- Moving files in Git involves changing the location or name of files and directories. Git tracks these changes, allowing you to keep a detailed history of your project's structure and modifications.

To rename a file within a git repository:

$git mv oldfilename newfilename

To move a file into directory:

$mkdir web

$git mv filename dirname

$git filename web

$git status

GIT RM:

- Remove files from the working tree and from the index
- Remove file from working and staging area files

To list index files:

$git ls-files

Remove all files from working and staging area files:

$git rm -r .

Remove files only staging area:

$git rm --cached test2

$git ls-files

Remove working directory files:

$ls

$rm test1

GIT CLEAN:

- It is used to remove untracked files from the working tree.
- Cleans the working tree by recursively removing files that are not under version control, starting from the current directory.

To remove a file with dry-run:

$git clean -n

To deletion of untracked files from the current directory:

$git clean -f

Interactive mode or git clean interactive:

$git clean -di

To remove any untracked directories:

$git clean -df

GIT LOG:

- It shows the commit logs.
- The advantage of a version control system is that it records changes. These records allow us to retrieve the data like commits, figuring out bugs, updates. But, all of this history will be useless if we cannot navigate it. At this point, we need the git log command.

To show the whole commit history:

$git log

To list last 3 commits only:

$git log -n 2

To list file-based commits:

$git log filename

To list specific commit:

$git log commit-id

Logs with oneline:

$git log --oneline

Display git log stats:

$git log --stat

Logs with author name:

$git log --author="R N Raju"

Allows viewing your git log as a graph:

$git log --graph

files were modified in each commit & number of lines added / removed:

$git log --patch or -p

List Range of commits <since> .. <until>:

$git log 6734679943..6358979275

GIT SHOW:

- It is used to view expanded details on Git objects such as blobs, trees, tags, and commits.
- It can be used to target specific files at specific revisions.

$git show

To show specific commit details:

$git show <commit id>

GIT BLAME:

- It Show what revision and author last modified each line of a file.
- It is used to examine the contents of a file line by line and see when each line was last modified and who the author of the modifications
- git blame only operates on individual files. A file-path is required for any useful output.

  $git blame filename

GIT-IGNORE:

- There are various types of files we might want the git to ignore before committing, for example, the files that are to do with our user settings or any utility setting, private files like passwords and API keys.
- **".gitignore"** file is used in a git repository to ignore the files and directories which are unnecessary to project this will be ignored by the git once the changes as been committed to the Remote repository.

  Create .gitignore File inside the project folder:

  $vim .gitignore

  ```
  # Log file

  *.log

  # Package Files

  *.jar

  *.war

  *.zip

  *.rar
  ```

  Check the status of the git repository.

  The file(s) added to the **.gitignore** text file will be ignored by the git every time you make any changes and commit.

  $git status