# Lesson:

# General Command on Git


PW SKILLS

# Topics

- git init
  - git status
  - git add
  - git commit & commit message convention
  - git pull
  - git clone
  - git push
  - git stash
  - git checkout
  - git branch
  - git reset
  - git cherrypick
  - git log
  - git diff
  - git merge
  - git rebase
  - git tag
  - git squash
  - git prune
  - git reflog
  - git clean
  - git help
  - git blame
  - git revert
  - git remote

# git init

The git init command is used to initialize a new Git repository in an existing project. When this command is run, it will create a new hidden subdirectory file name .git that contains all the necessary repository files and subdirectories to manage version control for your project.

To initialize git in your project - navigate to the root folder open the terminal run the below command.

```
Unset
git init
```

After running git init, you'll receive a message indicating that the Git repository has been initialized.
i.e
“Initialized empty Git repository in /path/to/your/directory/.git/”
*** Note** - if any git command is used before initializing the project with git, the given error message will be shown -

```
Unset
# using git command before initializing the project with git init
git status
fatal: not a git repository (or any of the parent directories): .git
```

# git status

After the project is initialized with git, The git status command can be used to check and display the current status of the project. It provides information about which files have been modified, which files are staged for the next commit, and any untracked files in your project. It’s a helpful command to understand the state of your project.

To check the status of your project, run the below command

```
Unset
git status

### output - if no change are there in the project -
git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

### output - if untrack files or new updated files are there –
git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        dropdown&navbar.zip
        dropdown.html
```

```
        navbar.html

nothing added to commit but untracked files present (use "git add" to track)
```

# git add

The git add command is used in order to begin tracking a new file or stage changes in the working directory so that the changes can be included in the next commit. Staging is an essential step in the Git workflow because it allows you to choose which changes to be included in the commit and which ones should be excluded.
Run the below command to keep track of changed files or new files

```
Unset
git add .                                  # to add all the changes or new files
git add newFile.js                         # to add only single file
git add newFile1.js newFile2.js            # to add multiples file
```

# git commit & commit message convention

Commits are a fundamental part of Git and serve as snapshots of the project at a specific point in time. Each commit has a unique identifier (a hash) and includes a message that describes the changes made in that commit. The git commit command is used to save staged changes in your Git repository as a new commit.

```
Unset
### syntax -
git commit -m "message_that_describes_the_commit_made"

### example
git commit -m "initial commit"
```

Commit message convention - it is always important to have a commit with proper convention as it is important for maintaining a clean and understandable version history in the project. Here are some of the points of the standard convention for writing commit messages with examples -

- Types - describes the purpose or category of the commit. Common types include feat, fix, docs, style, refactor, test, and chore
- Short Description (subject line) - A concise, imperative sentence that briefly describes the change. It should be in the present tense, start with a capital letter, and not end with a period.
- Body (optional) - A more detailed description or explanation of the change. This part can provide context, reasoning, and any additional information about the commit. It should be wrapped at around 72 characters per line.

Examples of standard commits -

```
Unset
### Adding a New Features (feat)
feat: Add user login functionality

### Fixing a Bug (fix)
fix: Fix issue in the login page

### Documentation update (docs)
docs: Update swagger README with installation instruction
### Coding Style Changes (style)
style: Format code according to project style guide

### Code Refactoring (refactor)
refactor: Refactor database connection url

### Adding or Modifying Tests (test)
test: Add unit for user login logic

### Routine Tasks or Maintenance (chore)
chore: Update dependencies
```

# git pull –

The git pull command is used to fetch changes from a remote repository (usually the origin) and merge them into your current branch. It essentially combines two operations git fetch and git merge
Fetching changes (git fetch) - Git fetches new changes from the remote repository (usually named "origin" by default) into your local repository without immediately merging them into your current branch.

Merging Changes (git merge) - After fetching changes, Git automatically tries to merge them into your current branch.
Example of git pull

```
Unset
# Fetch and merge changes from the remote "origin" repository into your current branch
git pull

# Fetch and merge changes from a specific remote branch into your current branch
git pull origin feature-branch
```

# git clone -

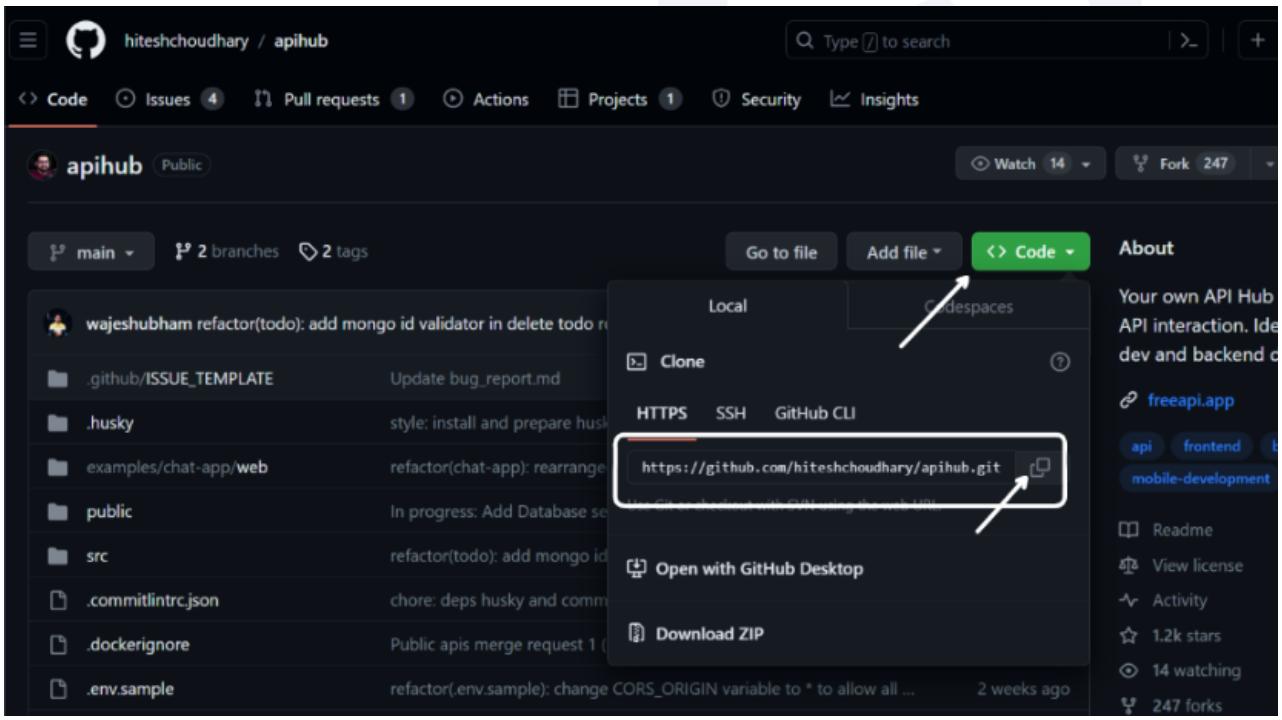
The git clone command is used to make a copy of a remote Git repository on a local machine. It allows us to start working on a project that is hosted on a remote server by downloading all the project's files, commit history, and branches

Example of git clone

```
Unset
# Clone a remote Git repository to your local machine
git clone https://github.com/username/repo-name.git

# This creates a new directory "repo-name" on your computer containing the
project's files and history.
```

To get the actual link to any repository, click on the green button with the text "Code" and the link will be shown.

i.e




# git push -

The git push command in git is used to upload your local repository's commits and associated objects like files, branches, and tags to a remote repository. As a result, it will allow developers to share their work with others or synchronize their work across multiple devices.

Example -

```
Unset
git push
```

# git stash -

The git stash is a useful Git command used to temporarily save changes in your working directory and index without committing them. This is particularly handy when you need to switch to a different branch or perform some other operation that requires a clean working directory.

Example of git stash -

```
Unset
# to save without commit -
git stash

# retrieve stashed changes
git stash apply

# apply a specific stash by index
git stash apply stash@{0}

#remove the latest stage from the stage list
git stash pop

# Listing all the stashes
git stash list

# removing specific stash -
git stash drop stash@{0}
```

The main purpose of git stash is to save changes that are not ready to be committed yet but need to be temporarily stored. This allows you to switch branches or perform other operations without committing to half-finished work.

# git checkout

The git checkout command is used to switch branches, restore files, and even go back to previous commits. It's a versatile command with multiple uses.
Example uses of git checkout -

```
Unset
# To switch to an existing branch
git checkout <branch_name>

# create a new branch and switching to it
# git checkout -b <new_branch_name>

# To create a new commit that undo the changes from a specific commit
git checkout <commit_hash> ^ -- .
git commit -m "Revert to previous commit

# discarding Uncommitted changes in your working directory for a specific file
use -
git checkout -- <file>

# To examine the state of repository at specific commit,  the HEAD can #be detach -
git checkout <commit_hash>

# To create a new orphan branch i.e a branch with no commit history
git checkout --orphan <new_branch_name>

 # To view a specific version of a file from a particular commit or tag
 git checkout <commit_hash> -- <file>
```

# git branch

The git branch command in Git is used to list, create delete, and manage branches with a Git repository. Branches are used to isolate development work and manage different lines of code.

Uses of git branch example -

```
Unset
# List all branches
git branch

# create a new branch
git branch <new_branch_name>
eg - git branch feature/new-feature

#switch branch
git checkout <branch_name>
```

```
# delete branch
git branch -D <branch_name>

# rename branch
git branch -m new-name

# list remote Branch
git branch -a

# list all branches
git branch -a
```

**Note:** We can also create and checkout both in a single command as follows

```
Unset
# create a new branch and checkout to same branch
git checkout -b feature/new-feature
```

# git reset

The git reset is a powerful Git command that allows you to reset the current branch to a specified state. It's used to undo changes, unstaged files, or move the branch to a pointer to a different commit.

There are three primary options for **git reset -**

1. Soft Reset
2. Mixed Reset
3. Hard Reset

```
Unset
# Soft reset - it moves the branch pointer to a new commit. but leaves the
changes in the staging area.
git reset --soft <commit_hash>

# Mixed Reset (which is the default) - it is similar to a soft reset, but it #
also unstaged the changes, moving them from the staging area (index) back # to
the working directory. The changes are now uncommitted and unstaged.
git reset <commit_hash> or git reset --mixed <commit_hash>

# Hard Reset  - A hard reset moves the branch pointer to a new commit and
#discards all changes (uncommitted and staged) in both the working directory #
and the staging area. Be cautious with this option as it permanently #discards
changes.
git reset --hard <commit_hash>
```

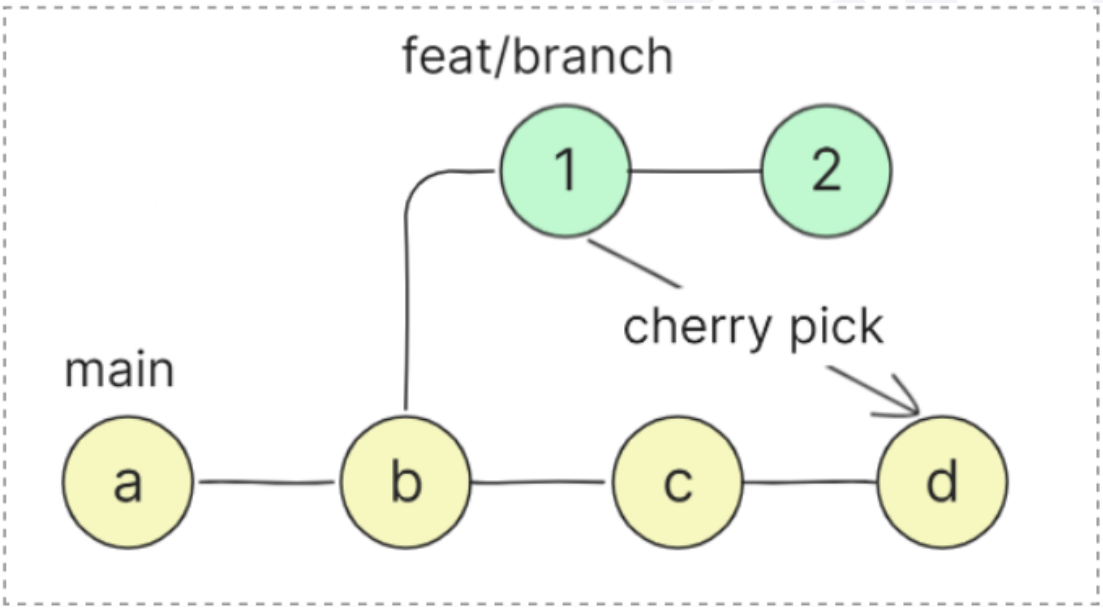
Common uses of git reset -

- Undo the last commit (git reset -- soft HEAD^) - This soft reset moves the branch pointer to the previous commit, effectively "undoing" the last commit while keeping the changes staged for the next commit
- Unstage changes (git reset) - This mixed reset unstaged all changes, moving them from the staging area back to the working directory.
- Move the branch to a specific commit (git reset -- <commit_has>)- This hard reset moves the branch pointer to the specific commit, discarding all changes and effectively resetting the branch to that commit.
- Move the branch Pointer without Discarding Changes (git reset --soft <commit_has>) - This soft reset moves the branch pointer to the specified commit, keeping the changes staged for the next commit.

# git cherrypick

The git cherrypick is a Git command used to apply a specific commit from one branch to another. It allows us to pick a single commit and apply it to your current working branch.
This can be useful when we want to selectively apply changes from one branch to another without merging the entire branch

```
Unset
git cherry-pick <commit_hash>
```




# git log

The git log command in Git is used to display the commit history of a repository. It shows information about past commits, including hashes, authors, dates, and commit messages.

Some of the uses of git log are as follows -

```
Unset
#1. display history commits
git log

#2. display compact commit history
git log --oneline

#3. display details for a single commit
git log -p <commit_hash>

#4. display the last N commits
git log -n <number_of_commits>

#5. display commits for a specific branch
git log <branch_name>

#6. display commits by author
git log --author="<authro_name>"
```

# git diff -

The git diff is a Git command that allows us to compare the differences between various parts of your Git repository. It is a powerful tool for inspecting changes made to your code

Some of the uses of git diff are as follows -

```
Unset
#1. Compare the working directory to the last Commit
git diff

#2. Compare the staged changes to the Last commit
git diff --staged


#3. display details for a single commit
git diff -p <commit_hash> <commit_hash1>

#4. Compare two commits
git diff <commit1> <commit2>

#5. Compare the current branch to another Branch
git diff <branch_name>
```

```
#6. Compare a Specific File
git diff <file_name>
```

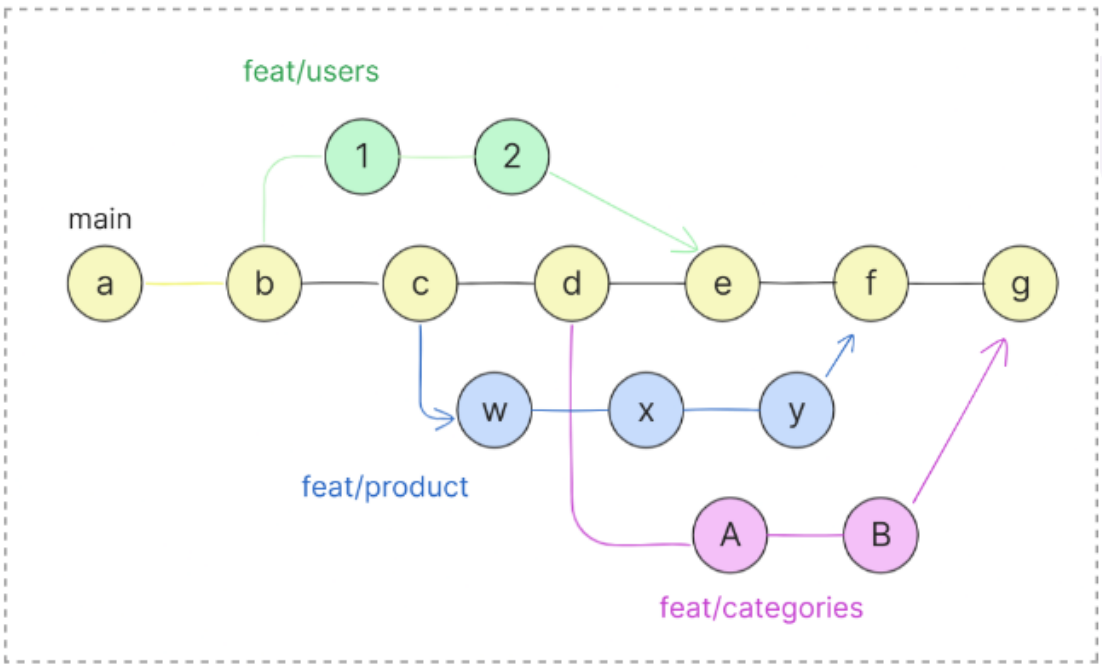

```
#6. Compare a Specific File
git diff <file_name>
```


# git merge

The git merge is a Git command that is used to integrate changes from one branch to another branch. This is typically used to combine the changes made in a features branch back into the main branch.

```
Unset
git merge <branch_name>
```

Git merge illustration diagram -



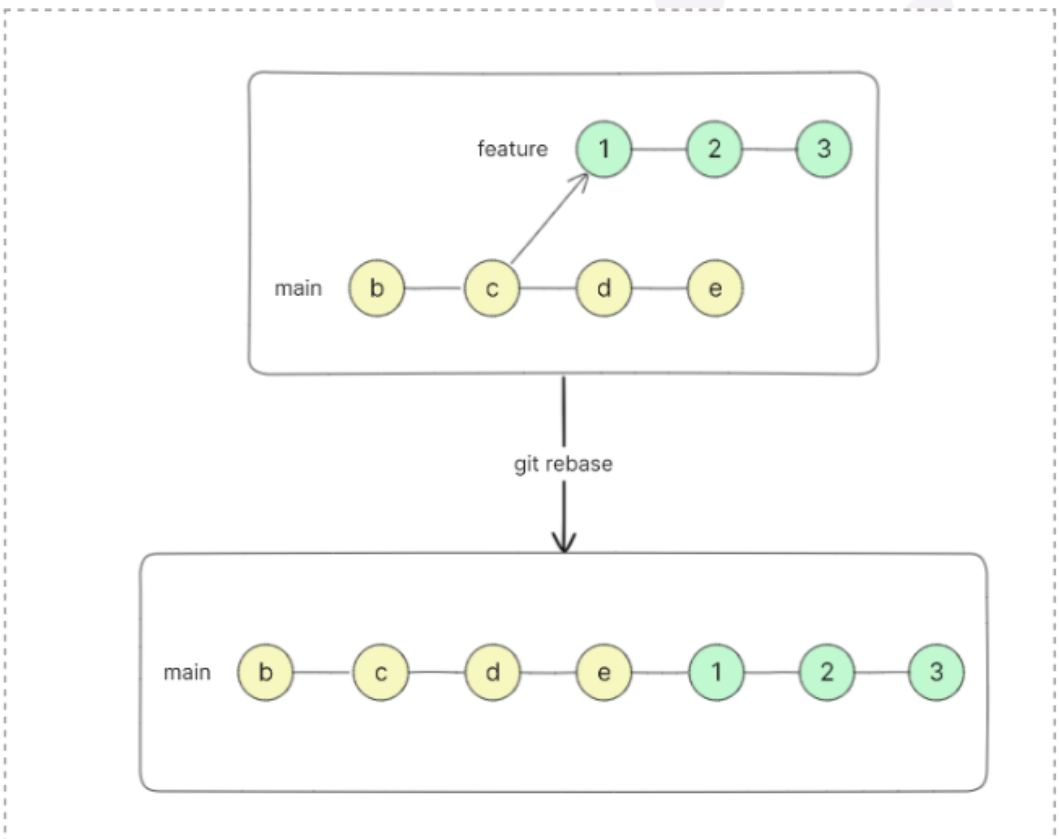

# git rebase

The git rebase is a git command used to integrate changes from one branch into another by applying the changes from one branch on top of another. It's a way to combine the changes of one branch with another branch, usually to keep a cleaner and linear history.

example

```
Unset
# Suppose you have a branch “feature” and want to rebase on top of the main
branch -


#1. Make sure you are on the main branch
git checkout main
git pull origin main


#2. Switch to the feature branch
git checkout feature

#3. Rebase the feature branch on the main branch
git rebase main_branch
```

Illustrations diagram of Git rebase



# git tag

The git tag is a git command used to create, list, delete, and manage tags. Tags are used to mark specific points in Git history, usually to denote important events like releases, milestones, or significant commits.
Example -

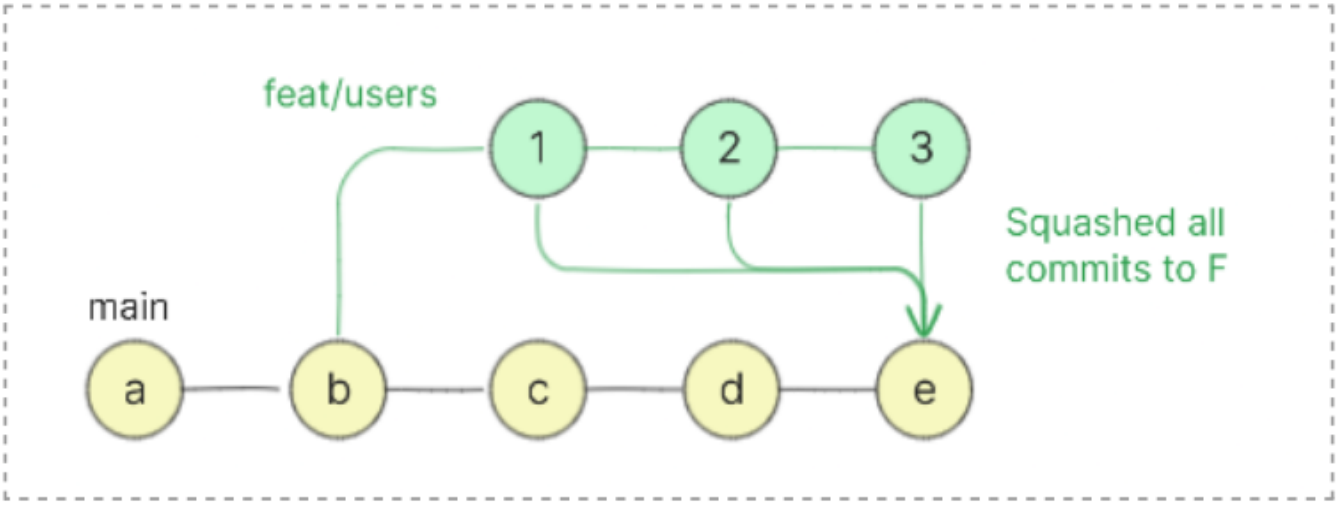
```
Unset
# creating a tag
git tag <tagname>

# Annotated tags
git tag -a <annotation_tag_name>

# Annotation tags with message
git tag -a <tag-name> "tag_message"

# Lightweight Tags - created with the absence of the -a, -s or -m
git tag <tag_name>

# listing tag
git tag

# deleting tag
git tag -d <tag_name>
```

# git squash

The git squash is not a standalone Git command, but it refers to a technique used to combine multiple Git commits into a single, more meaningful commit. This is typically done to maintain a cleaner and more organized Git history.

```
Unset
# syntax
git merge --squash <branch_name>
```

Illustration diagram of Git squash




# git prune

The git prune is a Git command used to remove objects from the Git object database that are no longer reachable and are no longer needed. It helps clean up the local Git repository by removing unreferenced and dangling objects (such as commits, trees, and blobs) that are no longer part of the Git history.
Example -

```
Unset
git prune
```

# git reflog

The git reflog is a Git command that stands for "reference logs." It's a powerful tool that tracks changes to the Git references in your repository, such as branches and tags. The git reflog command allows you to view a history of all reference updates, helping you to recover lost commits, branches, or changes.

Example uses of git reflog

```
Unset
# display the reference log
git reflog


# show reference log for a specific branch
git reflog <branch_name>


# display the reference log for HEAD of the current branch
git reflog HEAD
```

# git clean

The git clean is a Git command used to remove untracked files from the working directory. These are files that are not under version control (i.e., not added to the Git repository) and are not included in the .gitignore file.
Example -

```
Unset
# Dry run to see what would be removed
git clean -n

# Remove untracked files
git clean -f

# Remove untracked files and directories
git clean -fd
```

# git help

The git help is a Git command that is used to show you all the documentation shipped with Git about any command.

Example

```
Unset
# syntax
git help <any_git-commit>

# command will open the man page of the git commit command providing detailed
information about how to use it, its option, and examples

git help commit

# List all available Git commands for which you access help
git help -a
```

# git blame

The git blame is a Git command that annotates the lines of any file with which the commit was the last one to introduce a change to each line of the file and what person authored that commit. This is helpful in order to find the person to ask for more information about a specific section of your code.

Example

```
Unset
git blame <file>
```

# git revert

The git revert command is used to create a new commit that undoes the changes made by a previous commit or a range of commits.  It effectively applies the inverse of the specified commit(s) to the current branch, allowing you to 'revert the changes without altering the commit history.

Example-

```
Unset
# Single commit revert
git revert <commit_hash>

# Range of commit revert
git revert <start_commit_hash>..<en_commit_hash>
```

# git remote

The git remote is a git command used to manage connections to remote repositories. Git allows us to work with multiple remote repositories, and the git remote command helps us to manage and interact with them.

Example of git remote

```
Unset
# list remote repositories
git remote

# list remote repositories with details
git remote -v

# Add new remote repository
git remote add <name> <url>
# Remove a remote repository
git remote remove <name>

# Rename a remote repository
git remote rename <old_name> <new_name>

# show information about a remote repository
git remote show <name>
```