**Prerequisite before commit**

* First git expects to give some basic information about the commit person
* This can be done with help of below commands

git config --global user.name "pavani" git config --global user.email "mail.id"

* Then create your project folder
* CD to create folder
* Run command git init
* Create some empty files

**commit 1**

| **Task** | **Command** | **Notes** |
| --- | --- | --- |
| Check untracked files | git status | To check untracked files in repo |
| Stage files | git add basic.sh | use git status again to check if file is staged |
| Commit files | git commit –m "First commit" | commit message can be user defined |

**commit 2**

* Let's try to modify the file again and do second commit

| **Task** | **Command** | **Notes** |
| --- | --- | --- |
| Modify the source |  |  |
| Check untracked files | git status | To check untracked files in repo |
| Stage files | git add basic.sh | use git status again to check if file is staged |
| Commit files | git commit –m "Second commit" | commit message can be user defined |

**Key commands**

| **Task** | **Command** | **Notes** |
| --- | --- | --- |
| Check commit logs | git log | view history of commits |
| Check commit logs | git log --oneline | view commit logs in oneline |
| Unstage the file | git rm --cached basic.sh | can unstage the file if incase required |

# Git head

* You can see the "pointer" head from command "git log" command
* This actually helps us to understand which commit we are using currently
* By default head points to the last commit
* If in-case we go back to any of the previous state of project then head will point to the relevant commit

### You can learn more details about of head using command

git show HEAD

* This command provides more visibility about commit
* This command also provides more visibility on commit difference (which we can see at later point)

### Note : You can also view this output by providing head ID or array number

git show 0d196fa

git show HEAD~1

### Refer git-checkout file to understand how to navigate between commits.

**Git branch**

* As you know default branch in every repository is Master (Main) branch
* Your master branch may contains the original code of project
* Sometime you may need to add new feature to your project at the same time you don’t want to touch or break the current state of your project
* In such cases the best solution would be creating new branch and commit all your modified project contents over there
* Once you tested your new feature and find it working fine, you can go-ahead and merge it with your Master branch
* You can create many number of branches as per requirement
* This approach would be helpful when several developers are working in same project and each of them can work on different branch without breaking the master code

**commands**

* create branch in local
* git branch <branch-name>
* check what branch we are working on
* git branch
* check head point
* git log (To check Head is pointing to which branch)
* delete branch
* git branch –D <branch name>
* delete branch in local machine and in github
* git push -d origin <branch name>

**Git Checkout**

* checkout helps us in 2 ways
  + checkout between commits
  + checkout to different branches
* using checkout we can go back in time & check historical state of our files
* checkout is very safe command to use to switch between commits because its read only
* This can proved by going back to previous state file and modify something and make a commit. However this commit is not going to result anywhere

**Roll back to previous commit using head id**

| **Task** | **Command** | **Notes** |
| --- | --- | --- |
| checkout | git checkout <head-id>" | move head to specified commit ID |
| checkout | git checkout master | move back to main branch again |

**Simple scenario to prove checkout is readonly :**

* do checkout to any of your previous commit git checkout <head-id>
* modify any of existing file
* stage the file & make test commit
* git status & git log command is going to show you the commit
* go back to current head using git checkout master your new test commit will be no longer exist.

**checkout command to switch between branches**

git checkout <branch-name>

**Git ignore**

* You might be in an situation to maintain some files not tracked by Git
* To solve this kind of situation git has an solution called .gitignore
* Create file called .gitignore under your project directory
* Create some sample files which need to be ignored & update those file details in .gitignore

**sample .gitignore file structure**

* ignore files
* aws\_credentials
* azure\_credentials
* ignore folder and files inside
* apache/index.html
* apache/config.html
* ignore complete folder
* apache/\*

**Note**

* same procedure might not work if incase you are trying with existing files which is already managed by git, because those files are exist before we create .gitignore file
* if incase you want to add existing file
  + first add the file in .gitignore
  + then just try to do some modification of the file the make a new commit of it.

**Git Reset**

* Git reset is going to help us to delete the commits which we made
* Git reset has 3 different flags
  + soft
  + mixed
  + hard
* we can test reset by creating simple file and do stage & commit the file

**Reset commands**

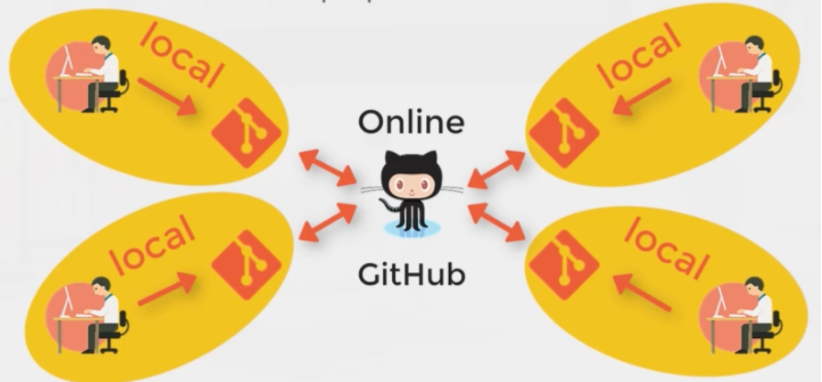
| **Tasks** | **Command** | **Notes** |
| --- | --- | --- |
| soft reset | git reset --soft <head-id> | will delete the commit & keep the files in staging |
| hard reset | git reset --hard <head-id> | bit dangerous - will delete the commit & files from workspace |
| mixed reset | git reset --mixed <head-id> | will delete the commit & un-stage the files |

**Note :**

* if we use reset command without any flag, by default it may use --mixed flag

**Github**

* GitHub is an online service where you can share your code or files in order to collaborate with different people
* To put in simple words, GitHub would have installed in one centralized machine where ever developers would commit their code from their local machine

[](https://github.com/vijayprabhu04/wic-devops-mar-22/blob/main/2-git/images/github.png)

**Difference between git and github**

* Often people get confuse with Git & GitHub

| **Git** | **Github** |
| --- | --- |
| version control tool to manage project locally | hosting platform to host project files in remote server |
|  | user friendly public platform with nice visual interface |
|  | millions of users sharing their projects to entire world |
| push & pull files from local to central server |  |

**Create first repo in github**

* Navigate to github page and login with your account
* Click the tab Repositories
* Click New
* Provide the Repository name
* Choose if it should be Public or Private
* You can also add ReadMe files which can help others to understand your project
* Create the Repository once all done

**git push**

* Once after repo is created you can push the files from local to github repo using push command
* To push files to remote server we may need to first setup the remote in local server
* git remote add origin <git-link> (origin is an optional name here, you can create remote with any name)
* Once remote is set our local repo is linked with remote GitHub
* Then you can start pushing the code from local to remote
* git push origin <branch-name>

**git pull**

* To keep your local repo updated you can use pull command
* git pull origin <branch-name>

**pull request**

* Pull request is something which you can create from github gui once after you make a commit
* This allows other people to validate your commit before merge with main branch

**merge**

* once after your pull request is approved, you can merge your changes with main branch
* This can be directly done from github gui
* This can also done from command line using merge command
* git merge <branch-name>

**difference**

* diff command helps to find the difference between commits
* It will just display the difference between the recent modification with last commit git diff main <branch\_name> git difftool main <branch\_name>

**fork**

* Fork allows you to create the copy of original repo in your github account

**clone**

* Using clone option you can download the complete repo to your local machine from github
* git clone <link>

**What is Git merge conflits?**

* Conflicts generally arise
  + when two people have changed the same line in file
  + one person delete the file while other person trying to modify the same file
* While this situation happens Git cannot automatically determine what is correct isn't
* That's where it becomes our responsibility to fix the conflict and merge the changes

**Let's try an example here**

* We shall create a file with certain lines and commit the changes to Git repo (main branch)

sample README.md file which is already in main branch

# This is the readme file created to understand the product code

- This code has 3 folders

- Payment - Created to enable payment authentication

- Otp - Created to enable Otp authentication

- Login - Created to enable Login authentication

* Let's assume 2 people are working with this code repository
* Person A & Person B :
  + Both think that there should be some changes needed with the file what we created
  + Note: Files is already exist in main branch

**Person A : (machine 1)**

* clone the repo to local machine
* creating new branch
* updating the file with new content

sample update

# This is the readme file created to understand the product code

- This code has 3 folders

- Payment - Created to enable payment gateway's

- Otp - Created to enable Otp access

- Login - Created to enable Login access

* making a commit
* but holding the merge with main branch

**Person B : (machine 2)**

* clone the repo to local machine
* creating new branch
* updating the file with new content

sample update

# This is the readme file created to understand the product code

- This code has 3 folders

- Payment - Folder to enable payment gateway

- Otp - Folder to enable Otp access

- Login - Folder to enable Login access

* making a commit
* merge with main branch as well

**Merge conflict**

* Now if person A try to merge the code now after creating the PR there will be an conflict
* person A & B should talk to resolve the conflicts

**How to resolve? (Resolve to be done in Person A machine in this case)**

* git pull
* vi README.md
* Delete the decided lines that can be removed
* git add modified file
* git commit
* git push orgin head