CONFIGURING NAME AND EMAIL

--------------------------

//To config username and email

git config --global user.name "nandeesh"

git config --global user.email "nandeesh.apjkalam@gmail.com"

//To check username and email

git config user.name

git config user.email

UNIX COMMANDS

-------------

//to list contents of the current folder

ls

ls -a [to see hidden files]

//To list content on specific folder

ls foldername

ls folderpath

//To open the current folder in GUI

start .

//To clear content

clear

//To see current directory

pwd

//To change directory [parent to child]

cd foldername

cd folderpath

//To move backward one level [click one back arrow]

cd ..

//To create new file in current directory

touch filename.extension

touch filename1.extension filename2.extension

touch folder/folder/filename.extension

//To create new folder in current directory

mkdir foldername

mkdir "foldername with spaces"

mkdir foldername1 foldername2

Note:no spaces in foldername

//To delete or remove files permanently from current directory

rm filename1

rm filename1 filename2

//To delete folder

rm -rf foldername

ADD AND COMMIT

-------------------------

//To create new .git repo

git init

--runs one time per project

--local repository

//Gives current status of git repo and content

git status

--gives status of tracked and untracked files

--tracked: green Added to statging area

untracked:red Not added to satging area

States:

Working directory---git add---->Staging area--git commit---->Local repo ---git push---->Remote repo

[add/modify/delete]

//Adding files to staging area

git add file1.extension file2.extension file3.extension

git add .

//Committing files to local repository

git commit -m "message"

//To get details of commit

git log

--Author/Date/Message/Commit ID/Branch

git log --oneline OR git log --pretty

//Git document

https://git-scm.com/doc

Note: Keep commit atomic

Keep each commit focus on a single thing

//To redo or update previous commit

to add some new files into previous commit itself

git commit -m "new" //now u missed some file to add in this commit itself then

git add . //now this changes to be added to previous commit

git commit --amend

--now editor will be opened, if needed u can add new commit message

or u can just save and close

//Ignoring files from commit

To not move this to local repository

Ex: api key or cred not moving to repo

Steps:

1. create .gitignore file in the repo

touch .gitignore

2. add files/folder to be ignored

Ex: .ds\_store git will ignore comiting this file

filename.txt Ex: secrets.txt config.properties git will ignore comiting these file

foldername/ git will ignore comiting this folder

\*.log git will ignore comiting .log extension files

Website to see which files to be ignored:

https://www.toptal.com/developers/gitignore

Ex: in python , type python anc click enter u get all files to be ignored in python

GIT BRANCHING

-------------

Head: Pointer pointing to curent branch and latest commit on that branch

//Viewing all the branches

git branch

\* and green color indicates current branch

//Creating and switching branch

git branch <branchName>

Note: there must be atleast one commit to create any new branch from other branch

//To switch branch

git switch <branchName>

git checkout <branchName>

Switch to new branch and do commit to see proper changes

//To add and commit

git commit -a -m "message"

//To create and switch at one go

git switch -c <branchName>

Note: if we do any changes in one branch and tries to switch to

other branch without commiting previous changes then it will throw error

so we need to commit previous changes and switch to other branch

//Deleting branch

git branch -D <branchName>

Note: user should not be in that branch

//To rename the branch name

Switch to the branch

git branch -m <newBranchName>

MERGING BRANCHES

-----------------

Notes:

We merge branches not commits [all the new commits will be merged by default]

We always merge to the current HEAD branch

//Fast forward merge

Ex: Need to merge feature branch to master branch

1. Switch to master branch

git switch master

2. Use git merge command to merger feature branch to master branch

git merge feature

//Merge conflict :

change in same file by 2 users and try to merge

Need to merge feature to master but one additional commit done in master in between

then you get merge conflict

Ex:

Switch to feature branch---Edit a.txt--Commit

Switch to master---edit a.txt --Commit

Now try to merge feature to master --You get conflict in a.txt file which both had modified

Step to solve:

1. Open the file having conflict and keep either one change or both change

2. Remove the markers added

3. Save, add and commit again with new message

BEFORE:

adada

adad

<<<<<<< HEAD

by master 1

by master 2

=======

by feature 1

by feature 2

>>>>>>> feature

AFTER: kept both user changes

adada

adad

by master 1

by master 2

by feature 1

by feature 2

GIT HUB

-------

Host/Share/Collobotare Repo

//Clonning git repo

To get local copy of existing repo hosted in Gut Hub

git clone <url>

Local copy of a repo will be created in our machine and also .git local repo will be created

Note:

U should not be in a repository

U can clone any public repo

Need permission to push the repo

//SSH config

//Creating repo in github

1. We have already existing repo locally and need to push to new remote repo

Create new repo in github

Connect ur local repo [add a remote]

Pushup ur changes to github

2. We dont have local repo and to create new repo in github

Create new repo in Github

Clone it down to ur machine

Do some work locally

Push up ur changes to github

Method 1: We have already existing repo locally and need to push to new remote repo

1. Create new repo in git hub

2. Connect local repo with new created repo

//To view is any remote repo existed

git remote -v

//Adding new remote if not added already

git remote add origin <URL>

origin: default name to remote repo

//To push code from local repo to remote repo

git push <remote> <branch name>

<branch name> : Branch we are pushing

local branch that has to be pushed to remote repo

<remote>: where we are pushing

Ex:git push origin master

master to master

To push local master branch commits to remote master branch

All the files in master branch local repo will be pushed to remote repo

Ex:git push origin feature

feature to feature

To push local feature branch commits to remote feature branch

All the files in feature branch local repo will be pushed to remote repo

Ex: To push local feature branch to remote master branch

feature to master

git push origin feature:master

OR

git push -u origin master //first time:match local master branch to remote master branch

git push //later

git push -u origin feature//first time:match local feature branch to remote feature branch

git push //later

git push -u origin feature:master //first time:match local feature branch to remote master branch

git push //later

Method 2: We dont have local repo and to create new repo in github

1. Create new repo in github

2. Clone it down to your machine

git clone <URL>

3. Do some work locally

Add and Commit

4. Push up changes to Github

git push origin main //push local main branch to remote main branch

//To rename master to main

git branch -M main

//The branch that points on last commit to remote

Branch where last we communicated to github

origin/main

we can move to that branch

git checkout origin/main

Note: when we clone we get only the default branch to our local

git clone

git branch //master or main

we wont see all the branches present in git hub

we work on main branch and will be connected with remote origin/main branch

//to know about all the remote branches

git branch -r

op:origin/master

origin/puppies

Note: Local main branch will be connected to remote origin/main branch by default

//Suppose we need to work on any specific branch locally not default main branch and that has to be

connected to same in remotes

Ex: we need to work on puppies branch in local and to be connected to puppies in remote

Steps:

1. CLone repo into local

git clone <URL>

2. Get to know all the branches in remote

git branch -r

3. Switch to that branch

git switch <puppies>

now user will be in puppies branch and local puppies branch will be connected to remote origin/puppies branch

4. Do all the changes in that branch, add and commit

5. Push to remote

git origin puppies

local puppies branch changes will be pushed to remote puppies branch

FETCHING AND PULLING

-------------------

To get all updated commits/changes of remote repo to my local we use fetch and pull

State Diagram:

WORKSPACE git add STAGING git commit LOCAL REPO git push REMOTE REPO

REMOTE REPO git fetch LOCAL REPO

REMOTE REPO git pull WORKSPACE

//To fetch all recent commits/changes in repo to local

git fetch <remote>

git status

Ex: git fetch origin

It fetches all the history and branches[changes of all the branches] from remote repo to local repo

//To fetch specific branch from remote to local

git fetch <remote> <branch name>

git status

Ex: git fetch origin master

fetch all the changes of master branch of remote repo

GIT PULL:

---------

//To retrieve all the changes from remote repo to working directory

git pull =git fetch + git merge

git switch <branch>

git pull <remote> <branch>

Ex 1: user is in master branch and execute

git pull origin master

fetch all the latest changes from the remote master branch and merge those changes to current master branch

Ex 2:If we want latest changes of remote repo of feature branch to our local feature branch then

git switch feature

git pull origin feature

NOTE:

git log

(HEAD -> feature, origin/feature) This indicates that current local branch HEAD is pointing is to feature and remote branch is origin/feature

and both are same [changes are same]

NOTE:

There can be merge conflicts when we pull the changes from remote to local repo, if same file changed by 2 colloborators

Ex: a.txt changed by user A, user B also changed a.txt and pushed to repo

now if user A tries to pull those remote changes to local repo then he may face conflicts

Note: always get all the latest changes [pull] before pushing changes to remote repo

//If you face conflicts when pulling changes

Steps to solve:

1. Open the file leading the conflicts

2. Do all changes [kepp one or both changes or add new]

3. Add and commit

4. Push those changes to remote repo [same branch to same branch]

5. Now try to pull

Ex:

User A is on feature branch

User B changed c.txt file content and pushed to remote repo

User A changes content of c.txt in local, add and commit

User A try to pull chnages of remote repo before pushing to remote repo his changes as good practise

//git pull origin feature

Now you get merge conflicts in c.txt

Solution:

Open c.txt

Keep either or both changes

Removes decorators

Add and commit the changes

Push the changes to same branch to remote repo

//git push origin feature

Now you can do pull

//git pull origin feature

Now all the changes from remote feature branch will be moved to local feature branch

MORE ON GITHUB

-------------

//Public vs Private repo

Public:anyone can see and clone that repo

To push we need coloborator permission

Private: only visible to owners and people having grant access

We can change public to private by moving to setting-option-danger zone

//Adding git hub collaborators

Allow the users to push changes to repo which they have clonned

STeps:

Go to setting

Go to Manage access

CLick on Invite collaborators

Enter their Git buh username

Select that name

Now notification will be sent to that new user

That new user should open his mail and accept invitattion

Now that user can push changes to that repo

We can also delete him if needed from settings

Note:Setting tab will be available in main owner account only

//To rename master to main

git branch -M main

First time flow to be followed

--------------------------------

Lead:

Create new remote repo and add coloborator[testers]

Craete new framework in local and push to remote repo

git init

git add README.md

git commit -m "message"

git branch -M main

git remote add origin <url of git repo>

git push -u origin main

Now colaborator can clone that repo to local ,work on it and can push changes

to remote repo

Note: Pull before push

//README File

Tells about

What the project does

How to run the project

WHy its noteworthy

Who maintains the project

Metadata of the project

Add this while creating the remote repo or new file and create new file

README.txt

Readme files have .md as extension

README.md

//Markdown

README.md

There is a format to prepare readme files

Refer that video or any tutorial

//Collaborating workflow

Case 1:

Remote has 1 commit

User A clone and worked in local and commited with 2 commits

User B has done few changes and pushed his changes to remote repo

Now if User A tries to push his changes to remote repo then it will throw error, he needs to pull and then push back her changes to remote repo

FEATURE BRANCH WORKFLOW

-----------------------

Note: Branch will be created based on feature or bug fixes

NOte:

//pull changes from remote repo to local

git fetch origin main

//Then check branches

git branch -r

//Then switch to any specific changes

git switch <branch name>

Now u can see all the changes in working directory

Centralized Workflow:

LEAD:

1. Create repo in Github

2. Clone it to local by LEAD

3. Create a Framework in local by LEAD

4. Add/Commit and Push to Remote

5. Create new branch for Sprint1 and Push to Remote by LEAD

git switch -c Sprint1

git push origin Sprint1

QA:

6. QA CLone the repo

git clone <URL>

7. Check Branches in Remote and Switch to Sprint1 branch

git branch -r

git switch Sprint1

8. Do all needed changhes in local repo, Add, Commit and Push to remote Sprint1

Note: before push to Sprint1 branch take Pull from SPrint1 branch to get all the latest changes

git pull origin Sprint1

Note: If you got merge conflicts in any file when you pull changes from remote to local

Open the file, keep either changes

Add, Commit and Push the changes

git push origin Sprint1

LEAD:

9. Move to Sprint1 branch if you are not in Sprint1 and Take pull from remote Sprint1 branch

git switch Sprint1

git pull origin Sprint1

10. Move to main branch

git switch main

11. Merge Sprint1 branch to main branch

git merge Sprint1

12. Push changes of local main branch to remote main branch

git push origin main

Note:

Main branch will be controlled by Lead

Only finalized code will be pushed to main branch at last by Lead

QA will work on Feature branch created by Lead and they will push all there changes to Feature branch, which is later Merged to main branch by Lead

After all your work is done in Sprint1 branch then we can delete it

//PULL REQUEST

Workflow:

1. Do some changes locally on Feature branch [Add+Commit]

2. Push feature branch to git hub

3. Open pull request using feature branch[Sprint1] to merge Sprint1 to main branch using Git Hub

Note: even we can compare Sprint1 with main branch if needed

Now Lead will receive Pull request notification in mail

4. Wait for PR to be approved and merged

5. Lead will check, conversation will go on through Commit[comment] section, he will review and finally merge Sprint1 branch to Main branch

6. At end he will delete Sprint1 branch

Now even Qa in local can delete that Sprint branch

git branch -D Sprint1

Now QA can pull main to local repo

//Conflicts when merging feature branch to main branch in guthub by Lead

Solution 1:

1. Click on resolve conflict

2. Do all changes in the colflict files

3. Mark as Resolve

4. Commit message

5. Click on Merge pull request

6. Confirm

Solution 2:

1. Lead pull changes from remote Sprint1 to local Sprint1

git switch Sprint1

git push origin Sprint1

2. Switch to main branch

git switch main

3. Merge Sprint1 to main branch

git merge Sprint1

4. Get pull of remote main to local main

git pull origin main

You get conflict , solve it

Open the conflict file, Keep any one change and save

5. Add/ Commit and push to main repo

git add.

git commit -m "message"

git push origin main

BRANCH PROTECTION

------------------

1. Wihtout pull request u cant merge to main

Go to setting

Click om Branches

Add a rule

Branch pattern name:main

Select first option :Require a pull request before merging

Now before merging anything to main pull request is mandatory

FORK:

-----

Creating personal copy of other user repo in our git hub account

Steps:

Go to anyone repo and click on Fork button

Local copy of his repo will be created in our account

Now clone to our local and do changes and push to this remote repo

Note:

We can fork other repo to our account

Do changes and push to remote repo in our account

we can also send pull request to real owner and can merge these changes to that main user account remote repo if needed

GIT STASHING:

-------------

If we want to move from one branch to other branch without commiting new changes then we need to do stashing and move to other branch

later we can come back to original branch and do pop to revert those changes back

Case 1:Work on any Main branch, commit and switch to feature branch --there will be no problem

Case 2: Work on Main branch do not commit and try to switch to feature branch

2 option:

Changes will come from main branch to feature branch [if we not do satshing]

Git will not allow to switch to feature branch without commit [we need to do satsh]

Note: while moving from one branch to other its good to commit and switch to other branch OR do stashing

Solution:

//do stashing and switch to feature branch [Move data from working directory or staging area to temporary buffer]

git stash

Come back to main branch and do [Bring back content from temporary buffer to working directory or to staging area]

git stash pop

to revert all the satshed changes and to working directory or to staging area