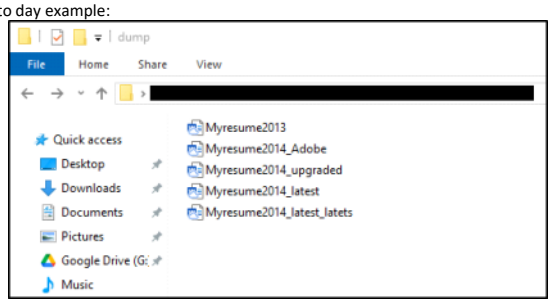
GIT series - PART A

Tuesday, July 16, 2024 5:42 AM

GIT YT: Tech Study Buddy: https://youtube.com/playlist?list=PLuY3WvFumiBCV8cMjocXSz3Aa3dTrYmZd&si=_NN4TVu9jGy4QkjV

SOCIAL MEDIA ETIQUETTE:
I hope this is useful for your personal study to help you save time in your study and have quality content.
You are free to use it for your personal study and building your notes but please do tag me in case you reuse these notes and images commercially or for any other purposes.

GIT - BASIC TO INTERMEDIATE
Session 2:

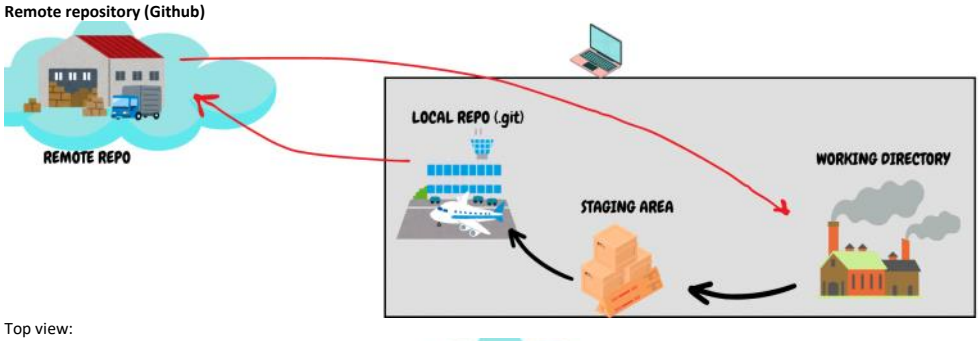
- DON'T TRY TO REMEMBER ALL GIT COMMANDS. Use reference guide
- **Intro to GIT**: What is Version control, GIT, GITHUB? ,
  - ⇒ **Version control software? What is it?** https://www.atlassian.com/git/tutorials/what-is-version-control
    - Software for tracking and managing changes in digital assets overtime.
    - Day to day example:
      - 

    - Challenge is when we have 100s of different files and multiple people working on them.
  - ⇒ **GIT**
    - It is a free and open source version control system primarily used by developers to maintain code.
    - Its a distributed version control System. Distributed - everyone has a copy on their local machine
    - it's a local software on the computer
  - ⇒ **GITHUB**- place / website/ service where you store your code online
    - i. web-based Git repository hosting service, which offers all of the revision control and source code management (SCM) functionality of Git
  - ⇒ **Diff:**

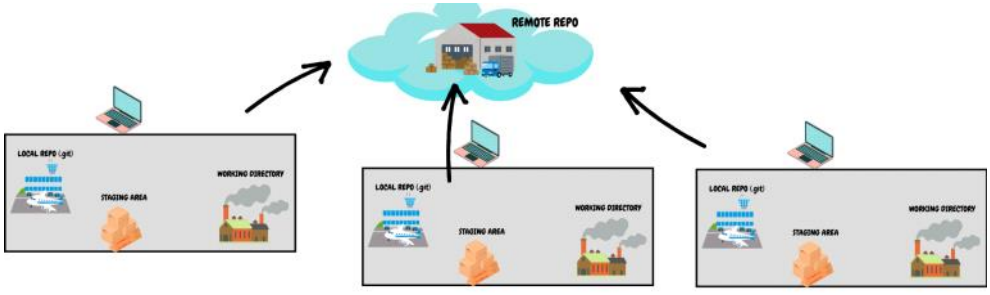
Git	GitHub
Git is a software.	GitHub is a service.
Git is a command-line tool, installed locally on the system	GitHub is a graphical user interface, hosted on the web
Git is maintained by linux.	GitHub is maintained by Microsoft.
Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
Git provides a Desktop interface named Git Gui.	GitHub provides a Desktop interface named GitHub Desktop.
Git competes with CVS, Azure DevOps Server, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, etc.

Credit: <https://www.geeksforgeeks.org/difference-between-git-and-github/>

- **Architecture of Version control systems ?**
  a. **Working directory**
  b. **Staging area** - holding or collection area till the next commit
  c. **Repository (.git folder)** : Sits inside the working directory . It manages the git commit history
  d. **Remote repository (Github)**



- Top view:



- **Common terms:**

a. Directory -> Folder
b. Working copy /working dictionary/ workspace
c. Terminal / Command line - > software on your computer where you can run text commands and move between files and folders .
   i. Eg: if you are double clicking to check what is inside a folder, in terminal we would be just writing a command cd and enter
d. Code editor - > where you write code
e. Repository -> folder in Github where you place all the files of your project.

Session 3 a b c d :

Software /interfaces needed and installation:

a. **Hosting service** ----> Setting up GITHUB
   i. Create a GITHUB account https://github.com/signup
   a. Introduce GITHUB:
      - Create repo
b. **Code editor**:
   a. During the installation, Git prompts you to select a text editor.
   b. There are many code editors out there. My choice is VSC. VSC is a Microsoft product and you could install it for free fromhttps://code.visualstudio.com/download
   c. Opening for the first time
   d. Ways to open:
      1) UI
      2) Cmd: code
   e. Open a folder in VSC
   f. Installing an extension
c. **Git** installation
   1. NOTE: **Git comes installed by default on most Mac and Linux machines!**
   2. Recommended to install with gitforwindows:
      - Install gitforwindows https://www.atlassian.com/git/tutorials/install-git
        i. Install (use gitbash instead of cmd) https://gitforwindows.org/
        ii. Introduce yourself to git
            - Official docs
              - https://git-scm.com/docs/git-config/ - can be overwhelming
              - https://docs.github.com/en/get-started/getting-started-with-git/setting-your-username-in-git
      - Check version -
        git version
   3. Git Credential Manager (GCM): https://microsoft.github.io/Git-Credential-Manager-for-Windows/Docs/CredentialManager.html
      - provides secure Git credential storage for Windows.
      - Check version:
        git credential-manager --version
   4. Link the Git to a GitHub Account.
      - Even if you don't provide the information , git will try to automatically figure it out, but some say this is unreliable.
      - Code:
        git config --global user.name "TechStudyBuddy4u"
        git config --global user.email "techstudybuddy4u@gmail.com"
      - Confirming :
        - git config --global --list
   5.
d. **Command line** or terminal : https://gfranzini.gitbooks.io/tracer/content/support/command-line-mac-vs.-windows.html
   a. **GUI or command line?** - Once you understand how it works on cmd line, its very easy to use the GUI. Analogy of geared vehicles
   b. Opening the cmd
   c. Some common commands to work with files:
      - cd <filename> ---->change directory , entering the folder or go to the folder
      - dir (ls) ----> list files in this folder
      - dir /a----> list files in this folder including the hidden files
      - cd.. ---> go back to outer folder
      - mkdir <foldername> ----> create a folder

Session 4:

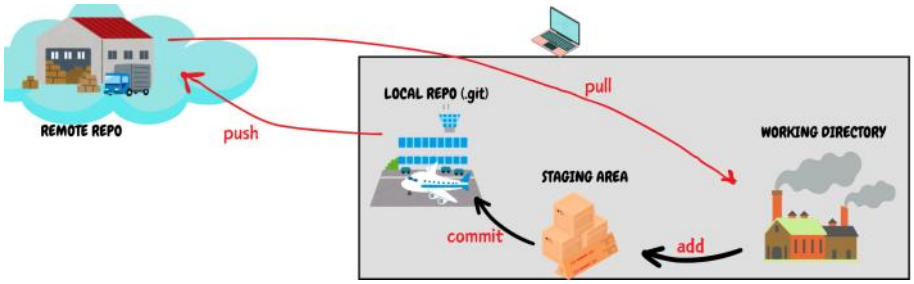
Creating project folder and Initializing GIT:

- **Creating project folder**
  a. UI: VSC> file>Open Folder > select or create your folder 'UIProjects'
  b. Cmd:
     a) Win/mac:
        - Windows - use GitBash
        - Mac OS X - use iTerm2

        The commands will be same
        # use 'mkdir' cmd to create a directory called 'GITProjects'
        - Mkdir Projects
        #navigate into 'Projects' directory
        - cd Projects
        # create some subproject folders
        - mkdir ProjectA ProjectB ProjectC ProjectD
- **GIT initializing: Now, GIT recognizes you but What should GIT track?**
  - # getting into ProjectA
  - Cd ProjectA
    #checking git status
  - git status ---> it does not recognize
  1) Introducing the folders to git
     a) New repo from cmd [THIS IS ONE TIME STEP]
        #initialize our local repository here
        git init
        # checking the status now:
        git status

     b) What happened?
        a) .git file was created. Lets check using 'dir' command
        b) This file has many subfolders as well
        c) WE DO NOT MESS WITH THIS FOLDER.

Intro to commands:



- add -> moving the new changes into the staging area
- commit -> moving files into the local repo.
- Push -> Upload to remote repo i.e. Github
- Pull-> download changes from remote repo to out local machine.

First GIT commit :

a. Create 3 files (1 with cmd , 2 with GUI)

```
#creating files
echo "testing my first commit" >> myfile1.txt
# create 2 more files from UI -myfile2.txt , myfile3.txt

#Commit myfile1
Git commit -m "adding file myfile1"

# checking the status now:  ---> acknowledges the changes to be committed. We could unstage or commit at this point
Git status
```

b. Commands to remember:

- git add <filename.txt> # adding into staging area
- git add . # adding all the files
- git status # to check the status
- git commit -m "Commit message" # adding to history and hereon can be traced and retrieved
- git commit # same as before but comment to be written in VSC
- Git log

c. Error you could face:

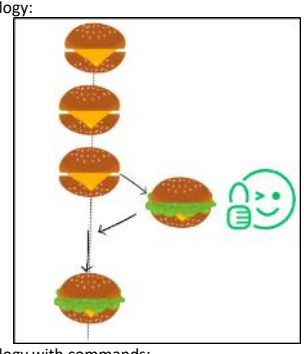
- Git commit without message https://codewithhugo.com/git-gh-cli-editor-vim-vscode/
  - Will open up a editor for you so you could write the message. Use Esc+:+q to quit.
  - To change it to the VSC,
    - git config --global core.editor "code --wait"
  - We have to ensure `code` CLI tool is installed using the VSCode command palette. i.e. The computer needs to know what does "code" mean
    - In VSCode command palette.(Ctrl + shift +P) select " Install 'code' command in PATH"

d. Where is this git config file? https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config
Based on configuration levels i.e local , global , system

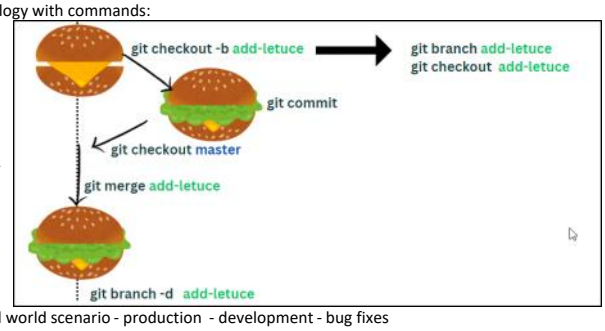
Session 5:

Branching: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

- What is it? independent line of development away from the main branch.
- Why do you use branching?
  - Analogy:



  - Analogy with commands:

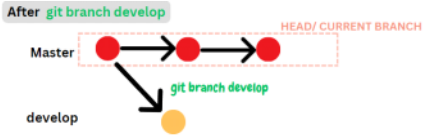


  - Real world scenario - production - development - bug fixes
- Working with Branches

```
# which is the current branch
Git branch  ---------> master      #many VCS use main currently. This can be renamed to anything basically.
```

```
$ git branch
* master
```

```
Git branch -vva                    # displays the local as well as remote branch from our last pull
```

# create branch - usually 'develop' branch 	
Git branch develop
Git branch
`$ git branch`
`  develop`
`* master`
	After git branch develop
	 

Move and create file	After git checkout develop

move into the branch
Git checkout develop
OR
Git switch develop

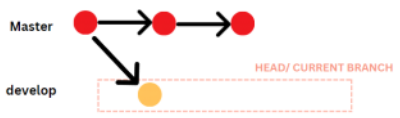
```
$ git checkout develop
Switched to branch 'develop'

User@DESKTOP-BS6B37D MINGW64 /g
p)
$ git branch
* develop
  master
```

Create a file fileindevelop.txt in develop branch and add + commit
echo "develop file" >> fileindevelop.txt
git add .
git commit –m "Some commit message"

```
$ echo "develop file" >> fileindevelop.txt

User@DESKTOP-BS6B37D MINGW64 /g/My Drive/YTProjec
p)
$ git add .
warning: in the working copy of 'fileindevelop.tx
 the next time Git touches it

User@DESKTOP-BS6B37D MINGW64 /g/My Drive/YTProjec
p)
$ git commit -m " add fileindevelop.txt"
[develop a7b688f]  add fileindevelop.txt
 1 file changed, 1 insertion(+)
 create mode 100644 fileindevelop.txt
```

Switch to master and Create a file fileinmaster.txt and add+commit
echo "develop file" >> fileinmaster.txt
git add .
git commit –m "Some commit message"

```
$ echo "master file" >> fileinmaster.txt

User@DESKTOP-BS6B37D MINGW64 /g/My Drive/
)
$ git add .
warning: in the working copy of 'fileinmas
the next time Git touches it

User@DESKTOP-BS6B37D MINGW64 /g/My Drive/
)
$ git commit -m " add fileinmaster.txt"
[master 5f01276]  add fileinmaster.txt
 1 file changed, 1 insertion(+)
 create mode 100644 fileinmaster.txt
```

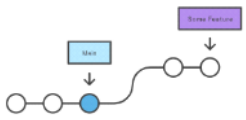
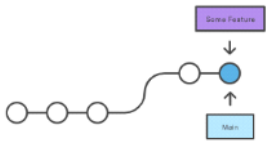


- Summarizing Commands :
  - Git branch #list branch
  - Git branch --list # list all
  - Git branch <branch> # create a branch
  - Git checkout <branch> # switch between the branches without making a commit
  - or
  - git switch develop
  - git branch -a #Remote branch list
  - git branch -m <old branch ><new branch > #Rename branch

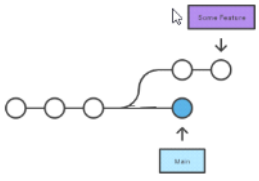
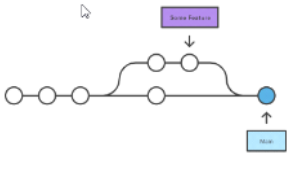
  Shortcut:
  - git checkout -b <name> # create branch <name> and switch to it
- https://learngitbranching.js.org/

➢ Merging and Deleting branch:

a. My preference: always move to the MASTER and merge the branch
b. 2 types: https://www.atlassian.com/git/tutorials/using-branches/git-merge
   a) Fast forward merge - when change is only in the branch and nothing has changed in master

Before	After fast forward merge
	

   b) 3 way merge :

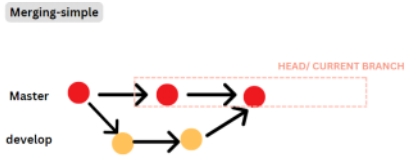
Before	After 3 way merge
	

c. Practicals:

Merging:
#Switch to master
Git switch master

#quick merge
Git merge develop

```
$ git switch master
Switched to branch 'master'

User@DESKTOP-BS6B37D MINGW64 /g/My Dri
)
$ git merge develop
Merge made by the 'ort' strategy.
 fileindevelop.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 fileindevelop.txt
```



Lets create conflict: work on same file in master and in branch

#check if in master

a)

Git branch

#create a file iamgoingtoconflict.txt
echo "develop file" >> iamgoingtoconflict.txt

#create branch
Git checkout feature1
#update iamgoingtoconflict.txt with some text
Add text
git add .
git commit –m "Some commit message"

#move to master
#update iamgoingtoconflict.txt with some text
git add .
git commit –m "Some commit message"

```
$ code iamgoingtoconflict.txt

user@DESKTOP-BS6837D MINGW64 /g/My Drive/YTPro
)
$ git add .

user@DESKTOP-BS6837D MINGW64 /g/My Drive/YTPro
)
$ git commit -m "commit from master branch"
[master bee0776] commit from master branch
 1 file changed, 1 insertion(+)
```

#switch to master branch
Git switch master

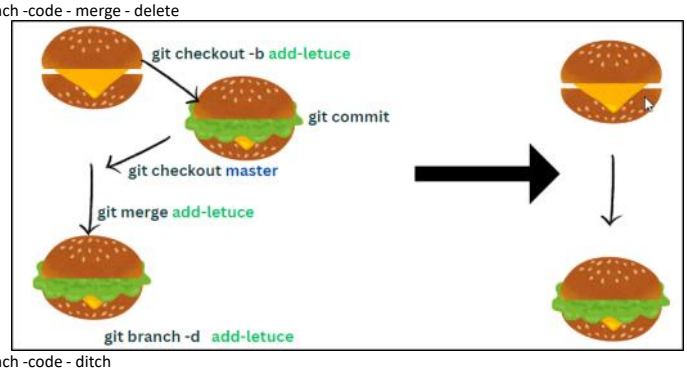
Now, merge [WICKED SMILE]
Git merge feature1

```
$ git merge feature1
Auto-merging iamgoingtoconflict.txt
CONFLICT (content): Merge conflict in iamgoingtoconflict.txt
Automatic merge failed; fix conflicts and then commit the result.
```

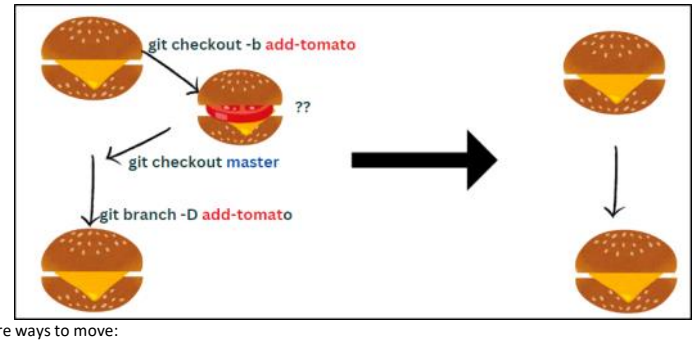
merging with conflicts

d. Merge with conflicting files [NOT SCARY AS IT SEEMS] https://github.com/jennybc/happy-git-with-r/issues/139
   a) STEPS
      (1) Open up the file with conflicts
      (2) You can choose to keep either of the changes or both. Git helps you with it.
      (3) If you choose to manually change, you will need to also cleanup the extras "== and " <<<"
      (4) Once changes are done, ADD and COMMIT it
   b) Undo merge:
      $ git merge --abort (also available $ git rebase --abort)
   c) Apps that help: kaleidoscope
e. Deleting the branch - [practice in some organizations]
   $ git branch -d <name> # delete branch <name>
   $ git branch -D <name> # delete unmerged branch

f. Typical workflows:
   a) Branch -code - merge - delete



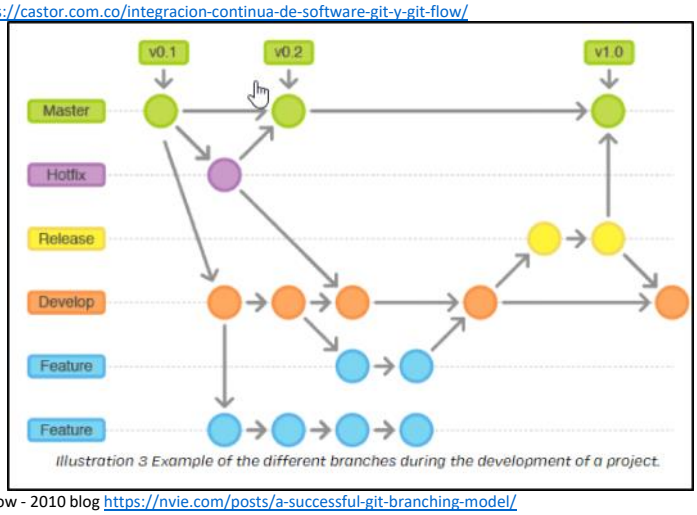
   b) Branch -code - ditch



- Some more ways to move:

Git checkout<commit id hash>	Move to some commit in the past		
Git switch main	In case , you moved to some previous commit and need to come back to where you were		
Git checkout head ~2	Moves 2 commits behind	Git checkout head ~2 	

Session 6:

- **GitFlow , GitHub Flow, plugins**
  - Git workflows:
    - https://castor.com.co/integracion-continua-de-software-git-y-git-flow/
    - 

    - Gitflow - 2010 blog https://nvie.com/posts/a-successful-git-branching-model/
  - few plugins in our Vscode - GitGraph ,Gitlens , GitHub Pull Requests
- **Best practices with Commit , Branching and Merging :**
  - While commit,
    - About the messages:
      - Clear, concise commit messages will help you and other get the gist of what's happening.
      - Have to be in imperative mood i.e. meaning like giving order, request or Instruction.
        - Eg: 'Create a checkbox' instead of 'Created a checkbox' or 'Creating a checkbox'
        - "Fix bug" and not "Fixed bug" or "Fixes bug."
      - Recommended format: https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project
        - Start with a Capitalized Imperative Verb for the Subject Line. no more than about 50 characters and that describes the changeset concisely
        - followed by a blank line. Makes it more readable.
        - followed by a more detailed explanation.
          - This should include : whys, whats
          - Reference the Requirement ID, Ticket number that will help in tracing
          - Wrap it to about 72 characters . Why 72 characters: https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html
            - Readable when you Git log since it does not wrap text.
            - On an 80 column terminal, if we subtract 4 columns for the indent on the left and 4 more for symmetry on the right, we're left with 72 columns.
          - Use bullet points for lists if the commits have several changes
    - Use GIT Status especially before commits - will save you lots of future possible human assumption errors
    - Use git logs abundantly
  - Branching:
    - commit your code before switching the branch . What if your feature is half done? Stash it
    - know your branch and where your head is pointing:
      - Use git branch - the *
      - Use tools like gitgraph
      - Verify in .git folder > HEAD folder
  - Merging
    - Don't panic :-)
    - Discuss with your colleague who's code is conflicting with yours and come to a conclusion https://www.linkedin.com/posts/abhisoni-dev_git-github-funny-activity-6928949921995714560-rkva/

Session 7:

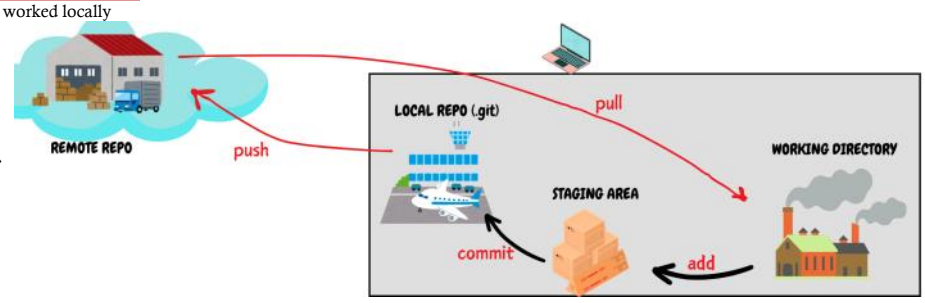
- **GIT logs:** https://git-scm.com/docs/git-log
  a. displays all the commits being made in that repository i
  b. Commands:

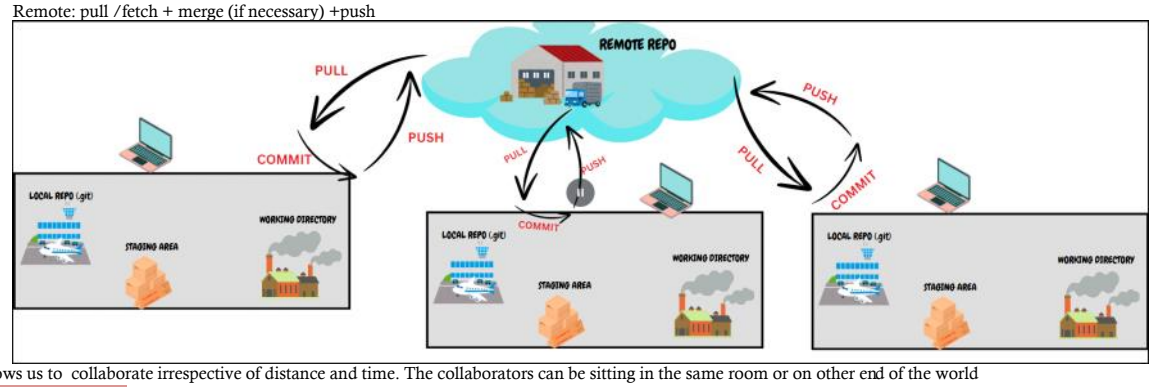
#git logs git log		
# condenses each commit to a single line git log --oneline	• displays only the commit ID and the first line of the commit message • Used when getting a high-level overview	
Prettier log: git log --oneline --decorate git log --decorate	• Displays branch or tag each commit is associated with	
#with filters #limiting the no of commits git log -3 #by date git log --after="2024-6-25" git log --before="yesterday" git log --after="2024-6-25" --before="2024-7-2" #By user: git log --author="Ren" #By file git log -- file2.py file3.py #By Commit message: $ git log --grep=" Commit message."		
#Viewing a single git commit's history git show <commit-hash>		
#git view commit	see the files in logs that are being committed	

git log --stat	being committed	
#where was it patched $ git log -p		
git log <branch-name>	commit history for the specified branch and commits shared by it's parent branches	

c. Navigating git logs:
   a) Between lines: Use keyboard ↓ and ↑ or 'j' and 'k' keys
   b) Between pages: spacebar for down and 'b' for up
   c) Exit 'q'

Session 8 :

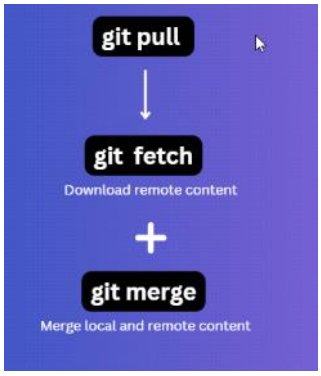
- **Into the cloud (Github - first step):**
  - ⇒ What we did till now:
    - We worked locally
      - 

    - We covered the basics: initialing GIT , creating , modifying , deleting files, getting info about commits , branching, merging and deleting branches
  - ⇒ Pending few intermediate and advance concepts - will be covered later
  - ⇒ Source code management (SCM) is used to track modifications to a source code repository. More of it in Session 2.
  - ⇒ Remote repositories (git hosting services): SCM or VCS on cloud
    - Lets look at the flow :
      Local: add+ commit
      Remote: pull /fetch + merge (if necessary) +push
      - 

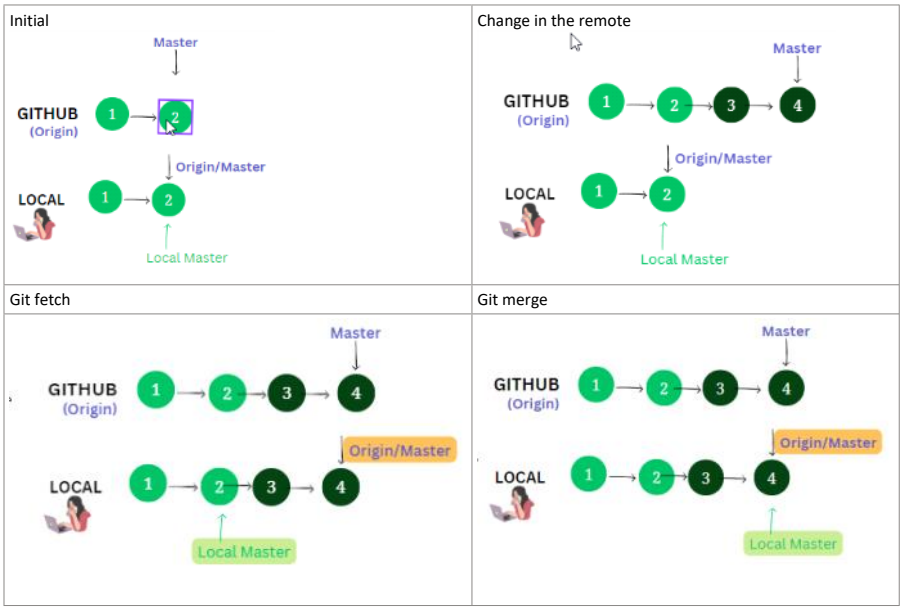
    - Allows us to collaborate irrespective of distance and time. The collaborators can be sitting in the same room or on other end of the world
  - ⇒ Remote repositories:
    - GitHub http://github.com/
    - Gitlab https://gitlab.com/
    - Bitbucket https://bitbucket.org/product
    - Source Forge https://sourceforge.net/
    - AWS Code Commit https://aws.amazon.com/codecommit/
  - ⇒ GitHub :
    - cloud-based Git repository hosting service. We have already created an account in session 3a
    - Allows real time collaboration.
    - We can branch, merge , push , commit, pull, fork, clone
    - Other than being a SCM on cloud, this is now a "social networking" site for people in software industry since we could follow people, contribute to open projects, communicate and now is being used as a portfolio by many.
  - ⇒ Pushing/Cloning with GitHub:

Session 9

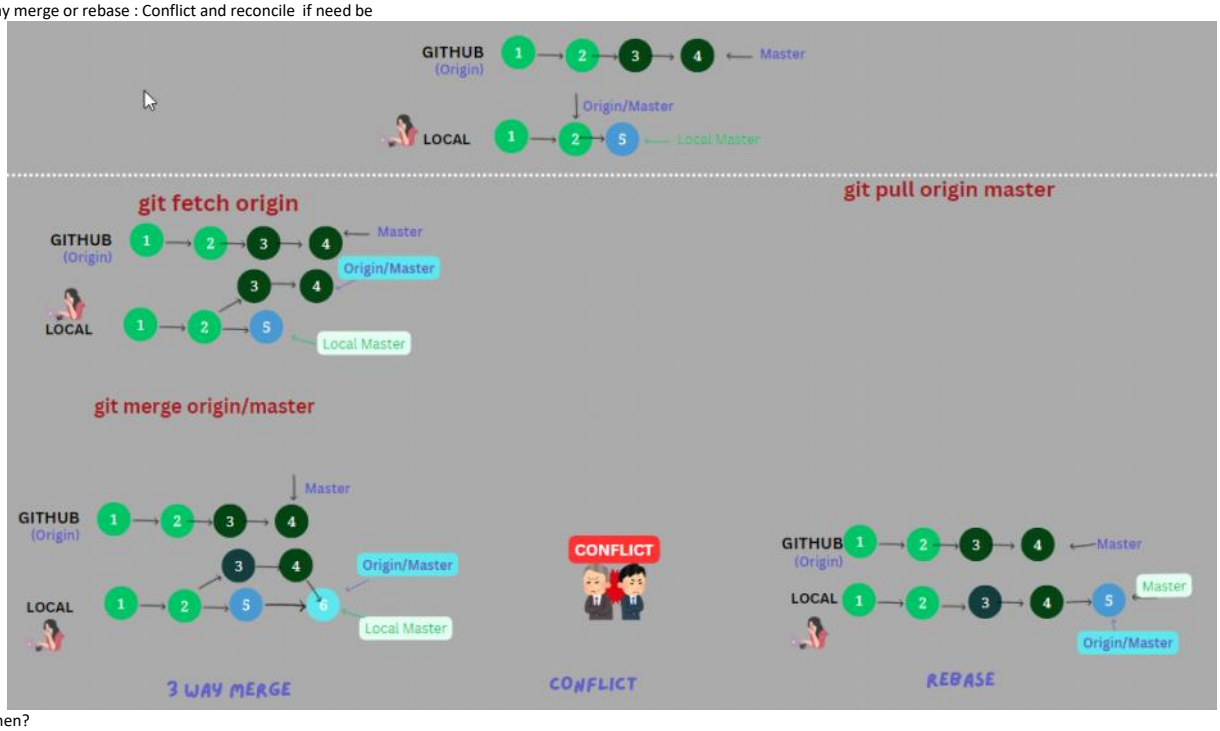
- Pull/ fetch :https://git-scm.com/docs/git-pull
  - What is fetch and pull?
    - What is git fetch?
      - Checking if there are any changes in the remote repository. No merging
      - Does it download?
        - Yes, downloads commits, files, and refs from a remote repository into your local repo.
        - Official link https://git-scm.com/docs/git-fetch
          - § NAME
            git-fetch - Download objects and refs from another repository
      - Does it affect your work @ local?
        - Git isolates fetched content from existing local content. Doesn't affect the local code and HEAD position does not change.
    - What is git pull
      - Git pull = git fetch +git merge

- 

- Use cases:
  - Basic case:

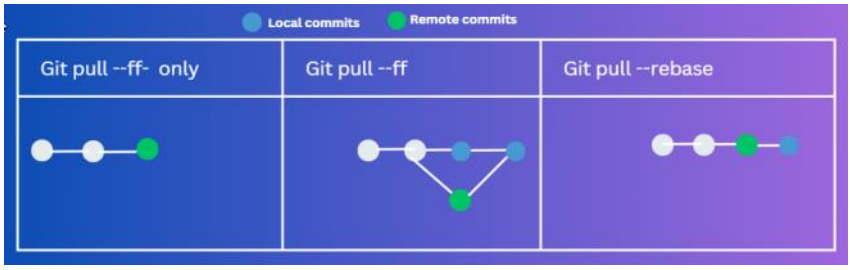


  - Conflict case:
    - Progress in remote branch and local branch
    - When fetch - local and remote branch is diverged
    - 3 way merge or rebase : Conflict and reconcile if need be



- Why use fetch then?
  - Just want to look into the progress in other branches where others are working
  - When you want to control next steps
  - If main had an extra commit and your local had a extra commit , during pull you will need to specify if you wanna merge or rebase. So many prefer 2 step process or fetch and merge
- Git pull flavors:

Git pull --ff- only	Git pull --ff	Git pull --rebase
# fast forward only [My preference, convinient] # it will stop even it there is a 3 way merge and for conflicts	# if its not ff (branch is diverged), it will merge + will ask for message. Note, possibility of conflicts	# rebases such that local commit happened after the remote commit.



LET's LOOK AT THIS IN PRACTICE

- When new changes only in master = fetch + merge = pull

git remote show origin	If there is a change in remote , it will show "local out of date"
Git fetch	Why? To see changes in remote What does it do? Copies the commits from remote repo to remote branches, not local branches
Checking the logs in remote repo: Git log origin/master	Remote head is pointing at latest remote commit and local head to latest local commit
Git status	Clearly tells us our local is behind
Merge changes from remote to local Git merge origin/master **Check the heads now:** Git log	. Displays the details of the changes Heads all at the latest commit which means we have updated our branch to the latest changes

- Working with remote branches

If you want to move to that branch : Git checkout --track <Branch name>	# git copies the contents of the remote branch to local branch. Track establishes the tracking connection with remote branch
When you want to just check remote branch and not update: Git remote update	Based on the changes we can then merge or rebase

- When changes at both in local and remote: (possibility of conflict)
  - Creating the scenario
  - Steps:

# so we need to pull the changes from the remote Git pull (or fetch +merge) # verify the conflicts, fix it, add and commit it	Gives conflict as changes are in both places

- Create a branch in local and pushing to remote branch

  # create a branch
  Git checkout -b testingbranch
  # make changes and commit
  Git commit -a -m "messgatestingbranche"
  #push it
  Git push -u origin testingbranch #since it's the first time , you have to use -u to set upsteam
- Rebasing - covered later

- Important commands:

git clone URL	Cloning remote repo to local
Git push	Push commits form local repo to remote repo
Git pull	Pull in any changes in the remote repo into the local repo
git remote	Manage set of tracked remote repositories https://git-scm.com/docs/git-remote
git remote -v	V =verbose , same as git remote +show remote url after name
git remote show <name>	Gives some information about the remote <name>. https://git-scm.com/docs/git-remote#Documentation/git-remote.txt-emshowem
git remote update	Fetch updates for remotes or remote groups in the repository but without automatically merging
git fetch	etch branches from one or more other repositories https://git-scm.com/docs/git-fetch
git branch -r	-r --remotes List or delete (if used with -d) the remote-tracking branches.

- Best Practices:

Session 10:

- **PR (Pull Request) Process**
- Engineering analytical tools: https://devdynamics.ai/

Session 11:

- **Rebase:**
- Rebasing with remote : Changing the base commit that is used for our branch

GIT advanced:

Session 12:

- **Behind the scenes with .git folder**

Corner cases:

- **Comparing two commits - Git Diff** : https://www.freecodecamp.org/news/git-diff-command/
- **Working with conflicts:**
- **Git merge Vs Git rebase**
- **GIT ignore** https://git-scm.com/docs/gitignore
- **Stashing - the temporary shelf**
- **Fork**

- **GIT blame**
- **Cherry pick:**
- **GIT Tag and Releases**
- **GIT Aliases**
- **Undoing :**
- **SSH:**
- **Rewriting git history -other than rebase:**
- **GitHub Pull Requests extension**
- **Note on the parent branch :**

GIT L:

https://www.linkedin.com/feed/update/urn:li:activity:7173265489199058944/