**Git & GitHub script & required URLs**

Version Control & its types

1. How the files are saved before version control
2. Why do we need version control?
3. What is version control?
4. Different types of version control – local (remote server), CVS, DVCS
5. Different tools for both CVS and DVCS ---
6. What is DVCS
7. Architecture of Git with diagram
8. Starting with sample…
9. Git init
10. ls –a --- will list the hidden files .git

Add a new file

1. Create a new file index.html
2. Git add
3. Git status
4. Git commit
5. Git status
6. Git log
7. Git log –oneline
8. Modify index.html
9. Git status
10. Git commit –a –m
11. Create new file contact.hml

Modify the existing file and show git diff is to show unstaged changes

1. Edit index.html
2. git status
3. git diff index.html
4. it Explain minus and plus in the file list
5. Git add index.html

Git diff –staged is to show staged changed so before commit view the staged changes

1. Git diff –staged index.html
2. Now before commit edit the file again to show staged and unstaged changes
3. Vim index.html
4. Git status – will show both staged and unstaged file – explain difference between both
5. Git commit –a –m

To delete from both WD and local repo

1. Git rm index.html
2. Will delete file from both WD and local repo
3. Git status – now explain the status.. whether the file is added or deleted both will be displayed

Add contact.html and commit

To delete a file from local repo

1. Git rm –cached contact.html

Now the file will be in WD and not in localrepo

1. Git ls & git ls-files
2. Git commit –m “deleted the file”
3. Git status

Now the file will be listed as untraced file because it is considered as new file

To inform git not to track some files. ,…. To tell git not to tract my contact.html file…

1. Vim .gitingnore (include contact.html in the file) / \*.html….any directory can also be given here
2. Git status – now the message about contact.html will not be displayed.
3. Git add .gitignore
4. Git commit –m “added ignorefile”

To revert the commit ---similar to undo

Now to undo add .gitignore i.e to delete the .gitignore file

1. Git log –oneline -> display the log with commit id
2. Git revert 61fa8d2 (commit id)
   1. –will ask for message, enter msg and save
3. Git log –oneline

* Will list the new commit id

1. Git ls – will display the contact.html
2. Git ls-files -> will not display gitignore file

To revert a revert

1. Git log –oneline
2. Git revert (commit id of removing ignore file)
3. Git ls-files -> display .gitignore file

To reset the HEADER -> which will delete the commits and set the header at the specified point

1. Git reset –hard (commit id)

Here explain revert and reset

Undo changes

1. Undo working directory changes
   1. If by mistake is we delete some lines of code in working dir and closed the file.

we want to get the code back. To do this

git checkout – index.html

Branching – what is branching, why do we need to branch and advantages of branching

1. Creating a branch b1 from master

git checkout -b b1 master

1. Add a file in branch b1 – vim about.html, add , commit
2. Git status
3. Git log –oneline
4. Check whether this file is in master
5. Git checkout master
6. Git log –oneline

Now merge the branch into master – without conflicts or smooth merge

1. Git merge b1 master (or) git merge b1 (if the command is given from master branch)
2. Git log –oneline (for both the branches)
3. Add new file in b1 say, vim site.html, add & commit
4. Add new file in master say, vim site.html, add & commit
5. Now merge b1 with master
6. Git merge b1 master

* Shows merge conflict in site.html

1. How to solve merge conflict?
2. Now edit the site.html (resolve the conflict)
3. Git merge b1 -> Already upto date.

Stash and unstash – Don’t want to commit the modified changes… stash will store the files in tmp area

1. Edit contact.html
2. Git status
3. Git stash
4. Git stash list -> will give stash number
5. Git show (stash#) -> will display changes in new file..and what is stashed.
6. Git status -> nothing to commit…

To bring back the files to working dir and delete the stash

1. Git stash pop (stash#)

To bring back the files to working dir and not to delete the stash

1. Git stash apply (stash#)

Partial stash – only some of the files would be stashed

Modify any two files and do partial stash

1. Git stash –p
2. Git stash pop
3. Creating a repository and add the project.
4. From already created project clone it.

Creating a repository and add the project.

1. Create an account in github
2. Create a new repository
3. Then for already existing project in local we add remote repository address
4. After creating remote repository get the address

>git add remote origin <https://github.com/anandhi-k/samplewebsite.git>

We are adding the project in local repository to github.

Origin is an alias name for the given url.

For further access instead of using the url everytime alias name can be used.

1. To add the code to created repository
   1. >git push origin master

Will ask for git user id and password.

Now the code will be available in remote repository.

1. To add a branch
   1. >git push origin b3
2. To delete the branch in local repo
   1. Git branch –d b3

If merge is pending will not allow to delete.

So give confirmed delete

>git branch –D b3

This will not delete the branch in remote repository

8. To delete the branch in remote which already deleted from local.

>git push origin –delete b3

For every change in local it has to be pushed to remote to get reflect.

From already created project in remote:

1. Create a remote repository with Readme.txt file (with some content).
2. Create a folder in local
3. Mkdir gitcs
4. Cd gitcs
5. >git clone <https://github.com/anandhi-k/gitcs.git>

Will create a project inside the folder with the name ‘gitcs’.

2. For further changes in remote, to update local repo
   1. >git pull origin
3. Now create a new file in remote (using create )
4. To check the status in remote repo use
   1. >git fetch origin

Fetch will not merge content from remote instead will show the changes just what happened in remote. Only the verbose i.e what changes done in remote repo.