WEEK 1

diff - command

Take two files or two directory and show difference between them.

Eg. "diff rearrange1.py rearrange2.py"

diff command line only returns/ only shows difference between this two files.

rearrange1.py	rearrange2.py
def fun(x): return x+1	def fun(x): return x ** 2

→ output of command will be :

```
< return x+1
> return x ** 2
```

< first file lines

> second file lines

→ diff -u validate1.py validate2.py

↓

unified format

+ , - etc.

+ -

→ wdiff highlights words that are diffrent instead of ~~lines~~ lines.

meld, KDiff3, Vimdiff are such similar commands.

→ diff -u old.py new.py > change.diff

Optional

diff of two files is stored in change.diff file.

this is known as diff file | patch file

It contain changes of two files in addition to content needed to under the code.

→ We have diff file and want to apply changes to file. for this purpose, PATCH command is used.

patch cpu-usage.py < cpu-usage.diff

→ patch command will only apply patch & can't be used to send patch file.

→ git diff filename.

→ Version Control System (VCS) keeps track of changes we make to our files.

→ Source Control Managment (SCM)

→ Git is a free & open-source VCS or SCM.

→ Commit is a collection of edits which has been submitted to the version control system for safe keeping.

→ git --version

→ git config --global user.email "d@gmail.com"
git config --global user.name "My name"

→ there are two ways to start with git repository:
git init (start from scratch)
git clone (make copy of already made directory)

→ git init
ls -la (or)
ls -l

This will initialize an empty git repo in current directory.

git add --all add all new & changed files. All at once.
git add.

→ git add minion.py (Added file to staging area)

→ "git status" command can be used to get some info. about current working tree & depending changes.

→ git commit (when we want to save our changes)

press 'i' to enter inline insert mode
esc to exit this mode.
:x! to exit git commit window.

→ Any git project consist of three sections:
- Git directory, • Working Tree,
- staging area

→ contain history of all file changes

→ contain current stage of the project.

→ Area outside git directory is called working tree. Working tree is current version of your project.

Tracked file can be in anly of these 3 stage:

i) Modifying → File is modifying
ii) Staging → After modying, we need to stage the life, so that changes will be included in next commit.
iii) Commit

→ After modifying file, run "git add ..." Command ~~to~~ & then git commit -m "(changes)".

→ mkdir scripts
cd scripts
git init
git config -l

Create file all-checks.py main()

→ git add all-checks.py
move new file from untracked to stage area.

Add check_reboot() in all-checks.py file.

Anatomy of a Commit Message

→ cat all-checks.py
display file content.

→ "git log" command can be used to check history of commit messages.

5c5, 3c3, etc.	c = changed	
11,12c11		line 11,12 in first file is replaced by single line 11 in second file.
16a15	a = added	
14d12	d = deleted	

WEEK 2

skipping the staging area :-

→ git Commit -a → flag -m
A shortcut to stage any changes to tracked files & commit them in one step.

→ Git uses HEAD alias to represent the Currently checked out snapshot of your project.

→ git log -p { show what is deleted/added to code. +, -
exit of window by pressing 'q'.

→ git show (code) commit ID { to see specific commit
without commit ID, by default shows last commit.

→ ~~git~~ --stat flag
git log --stat { statistical display

→ git diff { shows unstaged changes
change file content & don't save it.

→ git diff --staged { show staged but not commited files

Deleting & Renaming files :-

→ git rm { remove file from repository

we need to write a commit message of why we are deleting it.

→ "git mv" to rename files in repository.
git mv check_usage.py disk_usage.py

→ We need to ignore .DS files generated by computer. for this ".gitignore" is used.

echo .DS_Store > .gitignore
ls -la

git add .gitignore

Undoing changes before Committing

→ use "git checkout" to discard changes in working directory.

→ ./all_checks.py

→ ./all_checks.py > output.txt
git add *
↳ ~~stage~~ all all unstaged ~~work~~ changes in our working tree.

→ git reset HEAD output.txt
↓
will unstage file

"reset" can be understood as a counterpart of "add".

add => add changes to staging area.
reset => remove changes to staging area.

Amending Commits :-

create two new files using touch command.

→ touch auto-update.py
touch gather-information.sh

only 1 file is staged { git add auto-update.py
git commit -m "Add two files"

→ git add gather-information.sh
git commit --amend
→ this opens commit window to edit previous commit.

→ Avoid amending commits that have already been made public.

Rollback Commit (Abort) :-

→ git revert HEAD
git revert commit ID
git log -p -2

Identifying a Commit.

→ Commit ID is 40 character long jumble world called hash. which is calculated using algo SHA1.

→ Everytime after ~~commi~~ amending, commit ID changes.

→ git show 30e07 (First 5 characters)

What is a branch?

→ A pointer to a particular Commit.

→ Default branch that git creates for you when a new repository is initialized is called master.

Creating new branches :-

→ "git branch" command is used to list, create, delete & manipulate branches.

→ git branch branch-name => create branch
→ git branch => list branches

→ git checkout new-feature { Switch to branch

→ git checkout -b even-better-feature
{ Shortcut to create & switch to branch

free_memory.py

create a new file in branch.

→ git add free_memory.py
git commit -m 'Add an empty free_memory.py'

free_memory is in even-better-feature branch
check_free_space is in master branch.

delete a branch.

→ git branch -d new-feature

Merging

→ The term that Cit uses for combining branched data & history together.

→ git branch
even-better-feature
* master

git merge even-better-feature

(even-better-feature branch merges to master branch).

Cit uses two different algorithm to perform a merge :
1) FAST FORWORD MERGE
2) THREE WAY MERGE

→ git log --graph --oneline
shows a summarized view of commit history for a repo.

→ git merge --abort
If there are merge conflict, --abort can be used to abort the merge action.

Week 3 — Introduction to Github

→ Git is distributed VCS.
Each developer has a copy of the whole repository on their local machine.

→ Github is an web based Git repository System.

→ BitBucket, GitLab

→ For real Configuration & development work, you should use a secure & private Git server, & limit the people authorized to work on it.

→ Github provides free access to Git server for both public & private purpose.

→ Create local copy of the repository.

git clone URL — we made local repo to interact with remote repo.

if this is your first repo, Bash will ask username & password.

git push --set-upstream Project master
(repo name, branch name)

(.md = Markdown)

README.md

→ We can send our changes to remote repository by using the "git push" command.
git remote add <name> <url>
git push <name>

→ Everytime you push / pull request, Bash ask for password.

→ To avoid it, we will create "SSH key - pair".
Alternatively "Credential helper" can also be used.

git Config --global credential.helper cache

→ We will be cached for 15 mins.

→ To check this git pull command can be used.

→ We use "git pull" Command to retrieve ~~changes~~ new changes from the repository.
Git pull Command is used to fetch & download content from a remote repository & immediately Update the local repo.

Working with remotes

→ git remote -v { Give info about our remote

→ git remote show origin

→ git branch -r { branch list / info

If we want to make a change to remote branch,
pull the remote branch,
merge it with local branch,
then push it back to its origin.

→ "git fetch" Copies the commit done in the remote repository to the remote branches, so we can see what other people have committed.

→ git merge origin/master

→ Git uses remote branches to keep copies of the data that's stored in the remote repository.

fetching new changes

when working with remote → Workflow for Making changes

→ modify → staging area → commit (our local changes)
↓ fetch any new changes if any
merge if necessary → only then push our changes to remote repo

→ git branch → list all local branches
git branch -r → list all remote branches
git branch -a → both

→ git checkout branchname { switch to another branch

git switch - { switch back to previous branch

→ The default remote repository is called origin.

master ⇒ local repo master branch
origin/master ⇒ remote repo master branch

Updating local repository.

→ git pull command does both fetching & merging.

→ Running git pull will fetch the remote copy of the current branch & automatically try to merge it into the current local branch.

→ If we want to get the contents of remote branches without automatically merging any content into local branches, we can call "git remote update".

→ when using git to work on a new feature or a big refactor of some kind, it's recommended best practice to create seperate branches.

→ First time we push our ~~repo~~ branch to our remote repo, we have to add some more parameters.

```
git push -u origin refactor
```

↓ another way of referring to remote repositories.

→ "git merge" merges the specified branch into currently active branch.

For eg.

```
* master
  new-branch
```

(Currently active branch is master).

```
git merge new-branch
```

This will merge new-branch to main (master) branch.

→ `git branch -d refactor`

→ `git rebase`

`git rebase -i`

Week 4

Pull Request

→ Forking : A way of creating a copy of the given repository so it belongs to our user. Make a copy of someone else's Github repo in your own Github account.

→ Pull Request : A commit or series of commit that you send to the owner of the repository so that they incorporate it into their tree.

Code Review :-

→ code review means going through someone else's code, documentation or configuration and checking that it makes sense & follow the expected pattern.

→ Codes ~~reviews~~ v tools allows us to make ~~commit~~ comment on ~~other projects~~ someone else's code.

Managing Collaboration :-

→ Issue tracker tells us that the task that need to be done, the state they're in & who's working on them.

→ There's a popular bug tracker called Bugzilla, which is used by quite a ~~often~~ few open source projects.

→ In Github, each issue or pull request in a project has a unique number associated with it.

→ so if we have pull request with ID five, there won't be an issue with ID five.

→ Github automatically reference issues, & pull request, comments when we mention them using the hash tag number format. closes:#4.

→ Self assign

CI

→ Continuous Integration System is used to ~~test~~ run tests automatically.

→ Once we have our code automatically built & tested, the next automation step is continuous deployment, which is sometimes called Continuous delivery (CD).

→ Pipelines specify that need to run to get the result you want.

git pull <remote> <branch>

git clone <URL> <Location>
Remote repo url.