PAGE NO.:
DATE: 20/Jan/21

DAY:1 GIT & GIT-HUB WORKSHOP

- Workstation: System where the code is developed.
- Workspace: Dir where the code is stored. (WIP: Work In Progress)
- Commit Area: Area where the backup/Snapshot is stored.
- GIT is a SCM tool (Source code Management)
- Rollback: Restoration of code from backup (previous versions)/Snapshot.
- VCS (Version Control System): In backup timeline we create a version of our source code.

```
# mkdir /Desktop/ws
# cd /ws
# notepad mainmode.py
```

- Staging Area: Where all the changes are stored (More like a database).

URL: git bash download (git-scm.com/a..)

Git Bash

```
$ git --version
$ cd Documents/
$ mkdir gitws-workshop
$ cd git-----.
$ mkdir ws
$ cd ws
$notepad main.py
```

PAGE NO. :
DATE : / /

first line
^same

$ git commit main.py
↳ Failed!! (not a git repository)

• Git Repository = Staging Area + Commit Area
$ git init
↳ creates a new repo.
$ ls -a
$ git commit main.py
↳ Error !!
$ git status
$ git add main.py
$ git reflog
$ git commit main.py
↳ open a text editor
↳ write and save the comments

• Edit main.py
$ git status
• Add we have to do once and commit everytime we want to create a snapshot.

$ git commit -m "sec comment" main.py
$ git log
↳ Now there are 2 versions
$ cat >> main.py
#third line
fourth line
third

$ git commit -m "third" main.py
$ git log
↳ 3 versions available.
$ git reset <commit_id> main.py
↳ changes to the previous versions.
$ git status
↳ it is modified
$ git checkout -- main.py
↳ Now the changes are visible.
$ git reset 0662 main.py
↳ latest v.
↳ First 4 digits are sufficient as they are unique
$ git checkout -- main.py
↳ latest version is visible.
$ git log --oneline
↳ shows only one line per version with comment.
$ git status
↳ On branch master
$ vi main.py
=
fifth line

- We can also send the data through feature branch to production. But it is advisable to first merge the data to master and then send the data to production.
- master branch has the logs of all the branches.

$ git commit -m "4th comment" main.py
$ git branch
↳ lists all the branches.

- checkout: If we want to go to another branch

$ git checkout -b dev1
↳ It will first create dev1 & switch to it.
$ git log
↳ Entire history of master is copied.

$ vi main.py (dev1)
Sixth line
$ git status (dev1)

$ git branch --show-current
↳ dev1
$ git branch
↳ lists all branches.
$ git commit -m "six by dev1" main.py
$ git log
↳ six by dev (Head → dev1)
& master is still on the 4th comment
$ git checkout master
$ git log
↳ only 4 comment is there

- BRC in master timeline latest version is "4th comment". Won't be able to see sixth line. But as soon as we change the branch to dev1, sixth line is visible.
  (It is also updated in GUI at real time)

- Dev1 → Feature branch / upstream

$ git commit -m "7 by dev1" main.py
$ git log --oneline
  ↳ dev1 is 2 version ahead of the master branch.
$ git checkout master
$ git merge dev1
  ↳ entire history of dev1 will be appended to the master branch.
$ git log --oneline
  ↳ all 7 versions are appended in master. Now both are heading the 7th V.

- Merge strategy
  ↳ Fast Forward (FF)

$ git checkout dev1
$ main.py
  8th line
$ git commit -m "8th" main.py
$ git log
  ↳ dev1 is 1V ahead of master.

$ git branch -h (master)
$ git branch --set-upstream-to=dev1
↳ It'll set a tracker for dev1. master will be notified of any commit of dev1.
$ git status
↳ "your branch is behind 'dev1' by 1 commit."

$ git ~~merge~~ pull
↳ Only work if the current branch know the upstream
$ git log --oneline
↳ Master is now updated.

- GitHub → VCS
↳ create an account
↳ create a new repo

$ cd ..
$ mkdir www
$ cd www
$ cat > index.html
# my home page.
$ git init

GH → create a new repo
↳ www-git-ws (name)
↳ Public
↳ Initialize this repo with:
(NO OPTION SELECTED) >> CREATE REPO
$ git add .
↳ adds all the files in the active dir.

$ git commit -m "first" .
$ git status

$ git remote -v
↳ it doesn't know abt. any centralized system
$ git remote add mygithub <url of g.h. repo>
$ git push mygithub master
↳ Copies master timeline to github.

- remote == origin

$ git log
↳ Head → master, mygithub
$ cat >> index.html
# sec line from local

$ git commit -m "sec".
$ git log
$ git push mygithub master

- We can also edit from gh.
↳ commit changes
↳ third commit

$ git pull mygithub master
↳ fetches the data from gh to local.

$ git log
↳ file updated.

- fetch is better than pull

PAGE NO. :
DATE : / /

$ git fetch
↳ Failed

$ git branch -u=mygithub/master
master

$ git fetch
↳ worked!!
↳ Tell that master is behind but won't merge it.

$ git pull
↳ Now it will merge the data.

- Fetch will only get the history. It won't merge. To merge the data we need to use pull cmd.

$ git log --oneline
↳ Data is updated.

$ git show c6be
↳ Tell in this version; this data was there and what were the operations.

$ git diff fb63 c6be
↳ compares two versions.

PAGE NO. :
DATE : / /

- p4merge → tool for diff as well as to resolve the merge conflicts.

$ git config -l

$ git config --global -e

$ git clone <URL>
↳ downloads the git repo

PAGE NO.:
DATE: 31/Jan/21

DAY2 GIT AND GITHUB WORKSHOP

- Delta: It is the part that is stored in the newer version. OR the difference bet. two versions.

```
$ cd workspace
$ mkdir ws1
$ cd ws1
$ git init
$ touch a.txt
$ git add a.txt
$ git commit -m "a" a.txt
$ git cat-file
$ git cat-file -p 9b34
        ↳ give metadata of this commit id
$ touch b.txt
$ git add b.txt
$ git commit -m "b" b.txt
        ↳ It will create a new version
          but it also depends on v1/parent data
$ git cat-file -p "370s (b.txt)
    ↳ tree ——
      parent ——↖ (a.txt)
$ touch c.txt
$ git checkout
```

```
$ cd ws2
$ git init
$ touch a.txt
$ git add a.txt
    Add data & commit (twice)
```

$ git checkout a.thor.
$ git log --oneline

- CVCS: centralised version control system
  eg: GitHub

- ~~Git scripts are Routines or Triggers.~~

GH
- Create a repo: git-workshop
  - Public
  - Add a README file
    >> Create Repo.

git bash
$ git clone <URL of repo>
$ ls
  - git-workshop
$ git remote -v
  - It is already set up.
$ cat > index.html
  # my name from local computer
$ git add .
$ git commit -m "1 from local".
$ git log --oneline
$ git status
  - upstream is already set.
$ git push
  - uploads data to github

GH
setting > Webhooks (left panel)
>add webhook.

PAGE NO. :
DATE : / /

↳ Payload URL : http://<Jenkins URL>/github-webhook/
(15.207.14.184)
↳ send me everything (trigger)
↳ Activate

↳ git-ws
↳ Freestyle
↳ SCM
↳ Git
↳ <Repo link>

{{ Jenkins config }}

° webpage from git shows.

$ cat >> index.html
second commit
$ git push
↳ Refresh the webpage ; new content shows up.
$ cd .git
$ ls
$ cd hooks/
$ ls.
$ notepad pre-commit.sample

URL: git hooks
$ ~~touch~~ post-commit
$ touch pre-commit

PAGE NO. :
DATE : / /

$ vim post-commit
#!/bin/sh
git push

$ vim index.html
<body color='aqua'>
welcome!!
</body>
$ git commit -m "good code" index.html
↳ Now it will automatically be pushed.

- Fork : copies the public repo. Now we can edit & create branches. Now we can commit the changes.

$ git clone <URL of git-workshop>

- If we change the code. It will only reflect in our account. Now if we want to contribute. Then we need to send a pull request

Web: gitkraken
↳ GUI tool

PAGE NO.:
DATE: / /

```
$ cd /mydir ws3
$ git init
$ cat > a.txt
    aaaaa
$ git add .
$ git commit -m "a" .
```

gitk
  open
    ↳ open Repo
      ↳ ... ws3

$ edit & commit.

gitk
  It will show all in GUI.
  in real time.

```
$ git switch master
$ git merge dev1
```

- If we want to merge only the latest version not the history.
  In fast-forward we use squash.

```
$ git switch master
$ git merge --squash dev1
$ git log
    ↳ Not visible
$ git status
    ↳ a.txt is modified
$ git commit -m "merge u. sq" a.txt
    ↳ Now master has the entire branch. But does not copy the history.
```

$ git log --online
↳ latest data is visible

$ git checkout -b dev2
$ cat >> a.txt
10 from dev2
$ git commit -m '10 fr dev2" .
$ git log
$ cat >> a.txt
11 fr. dev 2
$ git commit -m "11 fr dev2" .

$ git switch master
change & commit

$ git switch dev2
↳ it won't able to see master's data
$ git rebase master
$ git switch master
$ git checkout -b dev3
$ touch d3
$ g. add.
$ g commit -m " dev3 " d3
$ g switch master
$ touch mfile
$ g add .
$ g commit -m "master" .
$ g switch dev3
$ g rebase master
↳ in feature branch instead of merge we use rebase.

PAGE NO. :
DATE : / /

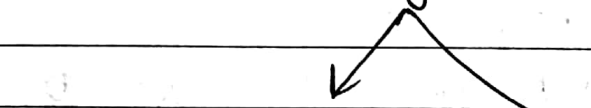
```
$ git log --oneline

$ git switch dev3
$ touch d1
$ g add .
$ g comm -m "d3 d1file" .

master
$ touch m1
$ g add m1
$ g commit -m "m file" m1
```

- Now master is 1 step ahead of dev3.

Merging Strategy



fast forward Recursive

```
$ git merge dev3.
```

↳ This time they will use recursive strategy.

```
$ git cat-file -p 2ce1
```

↳ master branch. 'dev3'

↳ TWO parents

↳ master & dev3

- If dev is ahead of master.

PAGE NO.:
DATE: / /

- cherry-pick : Picks only one version

$ git cherry-pick <version>

- If a conflict arises in merging. Then git copies both the code. Then we manually resolve the conflict

$ git commit
↳ After resolving the merging conflicts.

- If we change, let's say 3rd line in master as well as dev4. while merging, merge conflict arises.

*. Download P4merge from drive

$ git mergetool

- After merging we always have to commit.

config P4merge

$ git config --global -e
[merge]
tool = p4merge
[mergetool "p4merge"]
path = "——/p4merge.exe"

[diff]
tool = pymerge
[difftool]
path = " ___ "

*. Whole config in drive

• Stash : It memorises the code for some time and creates a stash in WIP.

--mixed
$ git reset HEAD~2
↳ It will rm 2 versions and point to the 3rd v.

$ git reset --soft HEAD~1
↳ File is still intact.

• If we want to reuse this file we have to again commit it.

$ git reset --hard HEAD~1
↳ DANGEROUS!!
It deletes the file, we loose our data.

$ git revert 93b4
↳ File is removed but the structure is same.

PAGE NO. :
DATE : / /

linux

creat workspace (wsgit)
git init
cat > ww.txt
ww
git add .
git commit -m "ww os" .
git log --oneline

Git Bash

$ git clone root@192.168.0.129:/wsgit
↑ Linux ip