Index

1. What is Version control and why?
2. Version Control Tools
3. GitHub & Git
4. Case Study: Dominian Enterprises
5. Git Features
6. Git Operations and Commands
7. **What is Version control and why?**

Every time you make change to your project, VCS makes a snapshot of these changes. These snapshots are changes. *See below 3 versions*

|  |
| --- |
|  |

* 1. **Why VCS?**

Suppose multiple developers are working on the project at the same time so it enables collaboration.

Storing versions/creating versions.

Knowing what has changes from version 1 to version 2. *See below.*

\*\*There is only 1 version that you’re working on is saved to your disk while the rest if the versions are neatly packed in VCS.

How VCS are helpful?

1. Storing Versions

|  |
| --- |
|  |

1. Backup

|  |
| --- |
|  |

1. Analyze

|  |
| --- |
|  |

3) Git & GitHub

Github is the central repository while Git creates local repo

|  |
| --- |
|  |

How is local repo connected to cloud repo?

1. Add origin
2. Via SSH
3. Central Server Repo
4. **Case Study: Dominian Enterprises**

Dominian Enterprise is a leading marketing company that works with hundereds of enterprises. They have distributed technical teams. Each Website they work has separate Tech Team, has own goals, own projects.

Objective of the case study:

They want to show their work, codes to other agencies and have more transparency. To achieve this, they have GitHub Enterprise as a platform.

1. What is Git

Is a distributed VS tools that supports distributed non-linear workflow.

|  |
| --- |
|  |

Questions:

Can we commit to another person’s repo?

No we can only commit to our local repo as well as Remote Repository.

**Features of Git**

|  |
| --- |
|  |

**What is a Repository?**

|  |
| --- |
|  |

**Local vs Central**

|  |
| --- |
|  |

**Git Operations and commands**

**Overall Picture**

|  |
| --- |
|  |

**Create Repository**

|  |
| --- |
|  |

After creating a Remote Repository on GitHub, let’s create a local repository on local machine using git init & git clone

Right click in the folder – git bash and below colourful emulator will open up.

|  |
| --- |
|  |

**Syncing Repo**

|  |
| --- |
|  |

*Command to clear screen – clear*

Code to add Origin

|  |
| --- |
|  |

Now that local repo is linked we will pull all the contents from Central Repo to local Repo.

**Git pull origin master**

|  |
| --- |
|  |

**Making Changes**

|  |
| --- |
|  |

*Git has an intermediate layer that resides between your workspace and your local repository. When you want to make/commit changes in your local repository, you have to add those files into index first. Command to add to index is* **git add.**

*Once**you have added files to index, you can check which files are added to the index and which are not using* **git stats***.*

*After you have added to index, you can use the command* **git commit** *to make the changes to the local repo*

|  |
| --- |
|  |

Create a new file in local repo – init folder as edu1.txt

Run **git status**

|  |
| --- |
|  |

*Above it says untracked file which is not added in the index.*

If we want to commit changes edu1.txt we will have to add to index

|  |
| --- |
|  |

Now let’s check the status again using below command

|  |
| --- |
|  |

As you can see it says changes are ready to be committed and edu1.txt is added to the index.

Now we will commit:

|  |
| --- |
|  |

-m is the message so everyone can see what changes were made

**Commit multiple files**

Create two files edu2.txt and edu3.txt and write something in them and check status below

|  |
| --- |
|  |

To add multiple files, first add **git add – A**

Then Check **git status**

|  |
| --- |
|  |

Now we will commit all files that have been added to index like above.

|  |
| --- |
|  |

Check how git stores all these commits.

*Always commit with a message, -m flag*

|  |
| --- |
|  |

How to do parallel/Non-linear development

|  |
| --- |
|  |

Create and switch to new Branch:

|  |
| --- |
|  |

*Now we can start doing all the work in the first branch*

Create a new file and check git status on the new branch. Then git add and commit

|  |
| --- |
|  |

These changes are done on in the *first branch* and not on the master.

Check the contents of the first branch. *Files from master are also there as they originated from the master branch*

|  |
| --- |
|  |

Files from first branch is not there in master as those changes were made in the first branch.

|  |
| --- |
|  |

Merging:

Merging means combining the work of different branches altogether.

|  |
| --- |
|  |

Whenever you’re merging you have to always checkout to the destination branch.

|  |
| --- |
|  |

Check the contents of master branch, mil3.tx has been added.

|  |
| --- |
|  |

Commit all (-a) the file. Mil3.txt was already added to the index so only commit will work and there is no need to add the file to index.

|  |
| --- |
|  |

Check git cat

|  |
| --- |
|  |

\*remember, I have not merged it yet and master file also contains the copy of mil3.txt

Difference between git pull and git fetch:

When you do a git pull, it actually pulls all the files from a central repo and connects them to a master branch.

Fetch also does the same thing but when you fetch, you need to merge the files to master branch.

Therefore, git pull = git fetch + git merge

Rebasing:

|  |
| --- |
|  |

Rebasing also does the same tasks as git merge but they do it in a different way. Rebasing ensures cleaner project history and linear workflow like above.

Go to branch

|  |
| --- |
|  |

Add more files, like mil4.txt and mil5.txt\

Git add -A

Then Commit:

Git commit -a -m “adding files for commit”

*See new files below*

|  |
| --- |
|  |

Now go to master:

Git checkout master

|  |
| --- |
|  |

Now will not use git merge but use rebase

|  |
| --- |
|  |

Whatever was in the master branch is now there in firstbranch

And whatever was in first branch is now in master

|  |
| --- |
|  |

Push changes:

Contribute changes to the central repo meaning github.

When you push code to the central repo, it becomes publicly available. You don’t want people to change or push the wrong code so you need certain rights.

So we are going to connect to our central repo via SSH to push the changes to central repo.

In order to connect via SSH, you need to generate a public SSH key and add that key into your github account. After that you can start pushing changes.

Generate SSH Key

|  |
| --- |
|  |

|  |
| --- |
|  |

Add SSH Key:

Go to settings > SSH and GPG Keys > New SSH Key

|  |
| --- |
|  |

Copy SSH using below

|  |
| --- |
|  |

Add SSH

|  |
| --- |
|  |

Use the below command to authenticate the ssh

|  |
| --- |
|  |

Now let’s say whatever you have built is needed to be stored at a branch at central repository but don’t to affect the master repo so other can identify that it’s your branch.

Now create a branch using gitbash on central repo from a branch (firstbranch) in local repo.

|  |
| --- |
|  |

A New branch has been created:

|  |
| --- |
|  |

Now if you select firstbranch from the drop down you will see below:

|  |
| --- |
|  |

Likewise, you can push master to central repo

|  |
| --- |
|  |

Git Flow:

|  |
| --- |
|  |