Get the best out of Live Sessions

HOW?

e!

Check your Internet Connection

Log in 10 mins before, and check your internet connection to avoid any network issues during the LIVE session.

Speak with the Instructor

By default, you will be on mute to avoid any background noise. However, if required you will be **unmuted by instructor**.

Clear Your Doubts

Feel free to clear your doubts. Use the "**Questions**" tab on your webinar tool to interact with the instructor at any point during the class.

Let us know if you liked our content

Please share feedback after each class. It will help us to enhance your learning experience.

edureka!

edureka!

DevOps Certification Training

COURSE OUTLINE
MODULE 02

1. Introduction to DevOps
2. Version Control with Git
3. Git and Jenkins
4. Continuous Integration with Jenkins
5. Configuration Management using Ansible
6. Containerization using Docker Part - I
7. Containerization using Docker Part - II
8. Container Orchestration Using Kubernetes Part-I
9. Container Orchestration Using Kubernetes Part-II
10. Monitoring Using Prometheus and Grafana
11. Provisioning Infrastructure using Terraform Part - I
12. Provisioning Infrastructure using Terraform Part - II

edureka!

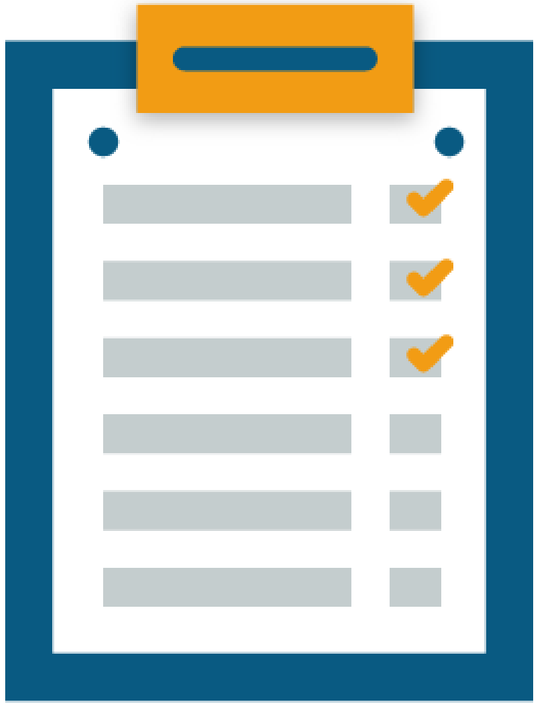
Module 2 – Version Control with Git

edureka!

Topics

Following are the topics covered in this module:

- Version Control
- Git Introduction
- Git Installation
- Commonly used commands in Git
- Working with Remote repository



Objectives

After completing this module, you should be able to:

- Understand Version Control
- Install Git
- Perform management of files for small as well as large projects
- Perform various Git commands such as git add, git fetch, git commit, git init, etc.
- Working with remote repositories
- Implement the part of Edureka's Project associated with Git

Why Need Version Control?

edureka!

Why Need Version Control?

Let's consider a multinational company that has its offices and employees all around the globe

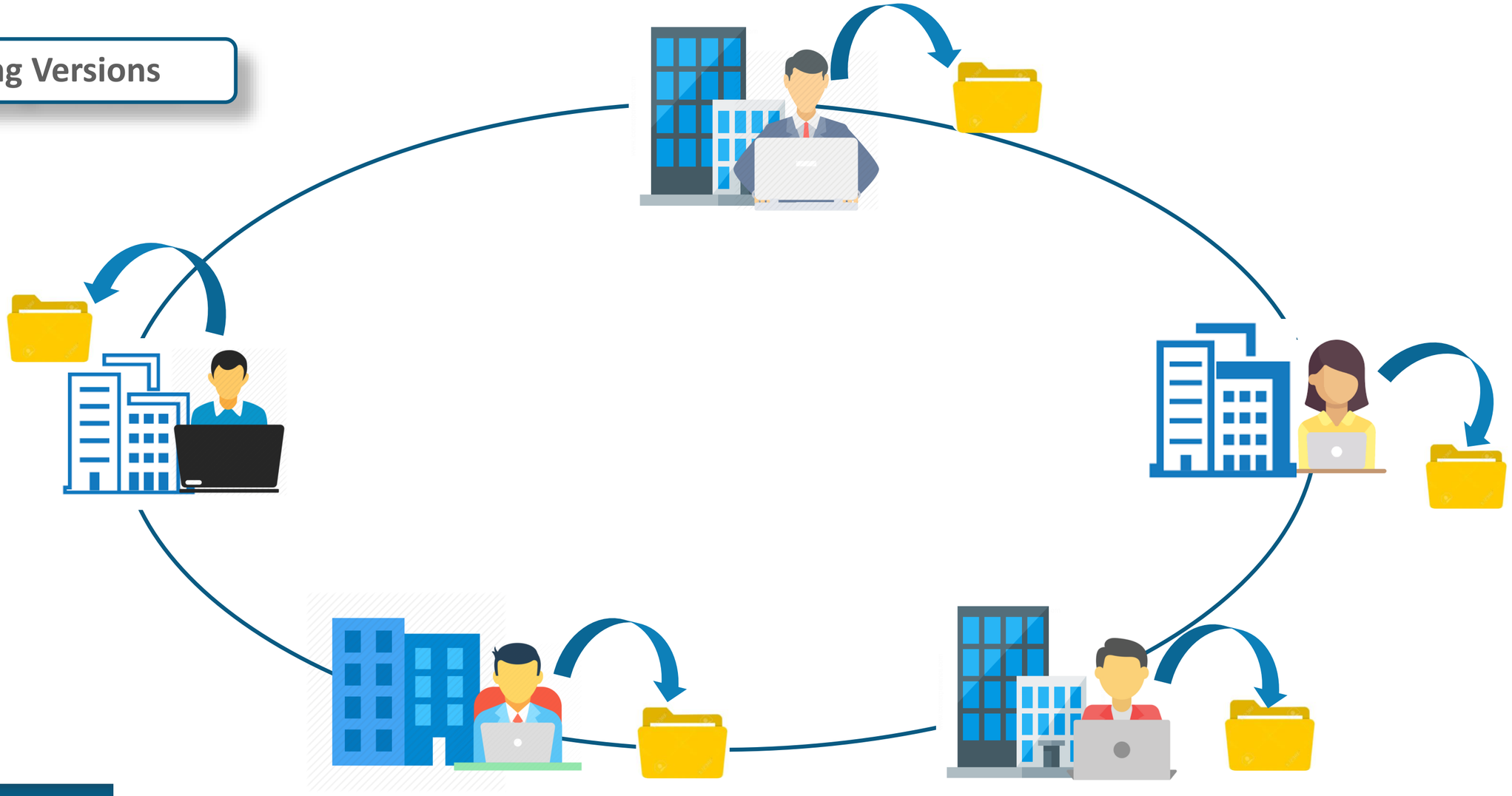
Why Need Version Control?

Problems that may arrise

Collaboration

Why Need Version Control?

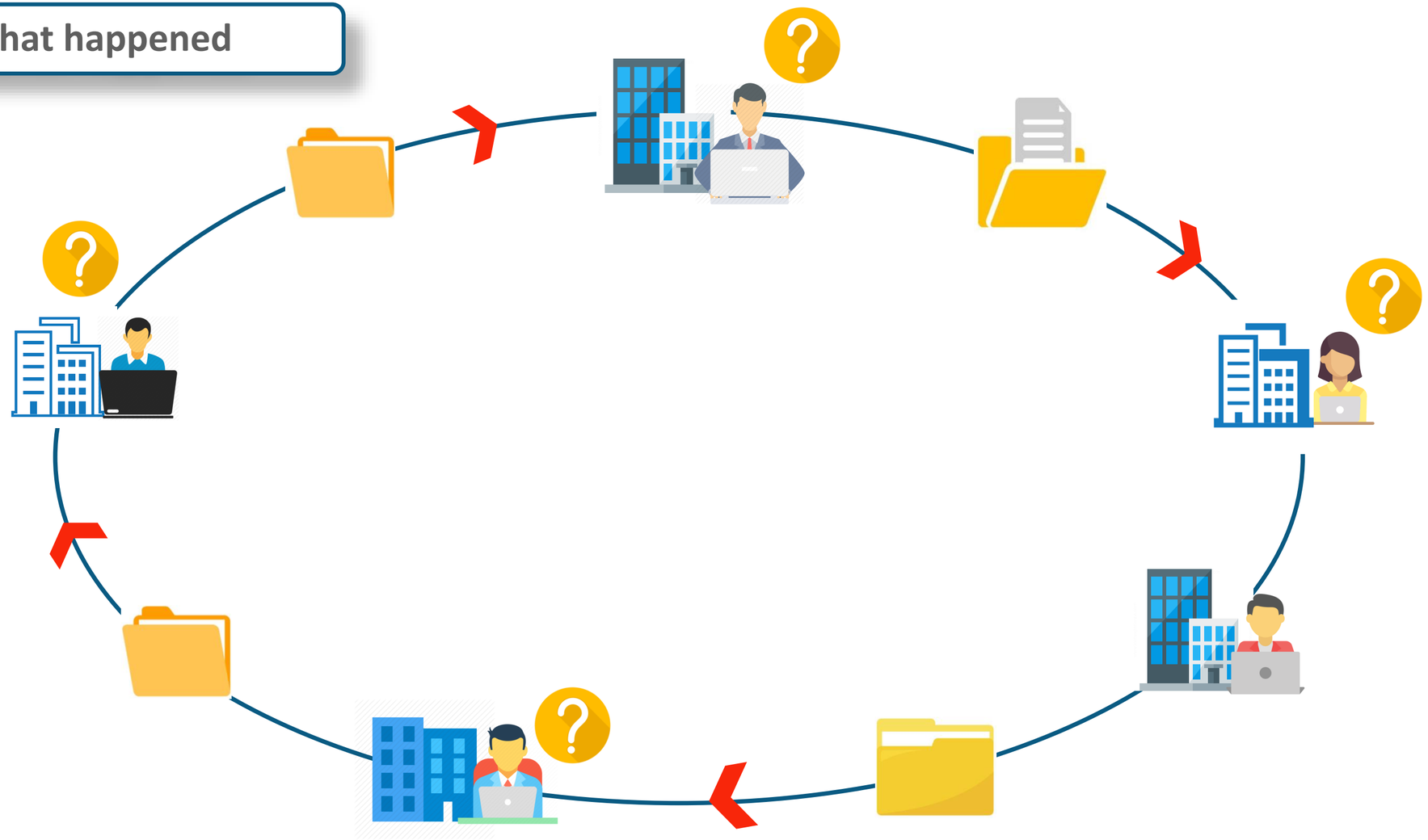
Storing Versions



edureka!

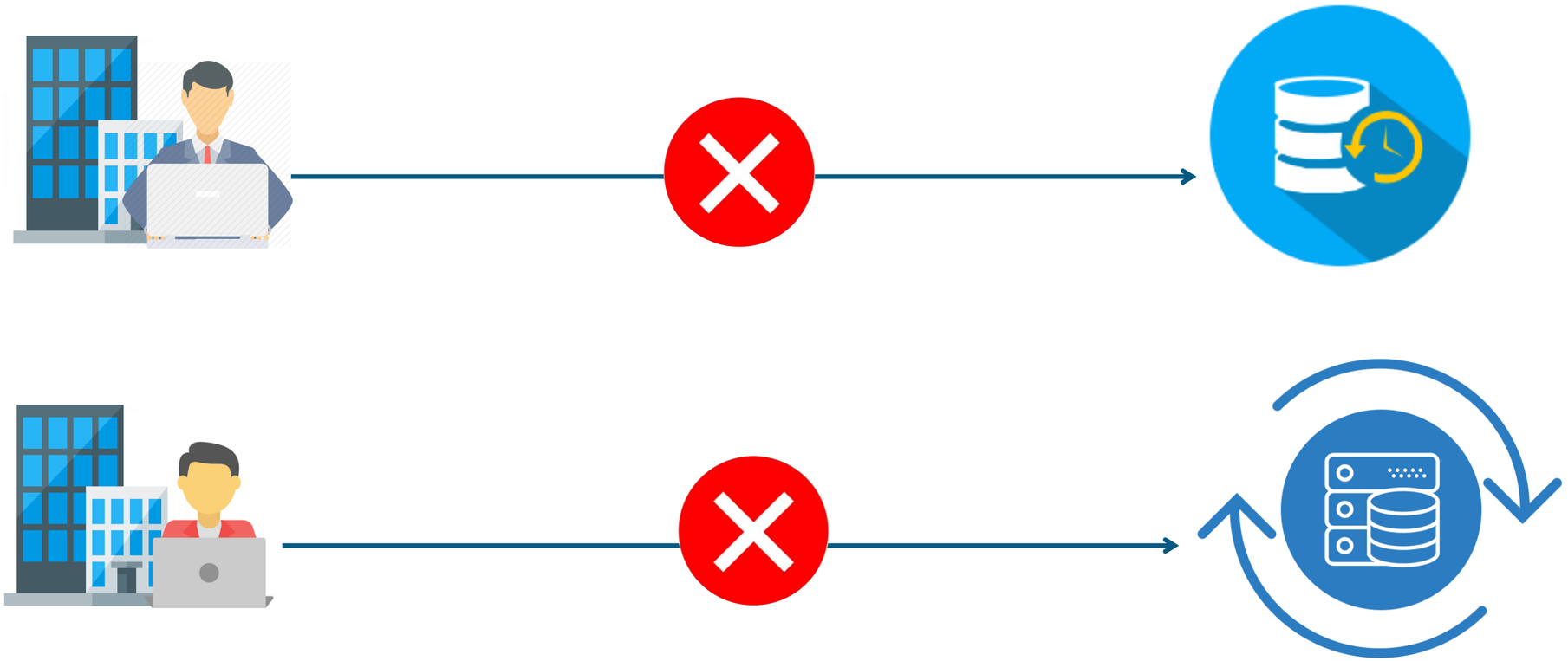
Why Need Version Control?

Figuring out what happened



Why Need Version Control?

Backup



Why Need Version Control?

Solution for all these problems

All these problems can be solved with the help of a "**Version Control System**"

Why Need Version Control?

- Version control helps the teams to solve these kinds of problems, by tracking every individual change by each contributor and helps prevent concurrent work from conflicting.
- A version control software supports a developer's preferred workflow without imposing one way of working.

Issues Without Version Control

- Once saved, all the changes made in the files are permanent and

  cannot be reverted back

- No record of what was done and by whom

- Downtime that can occur because of a faulty update could cost

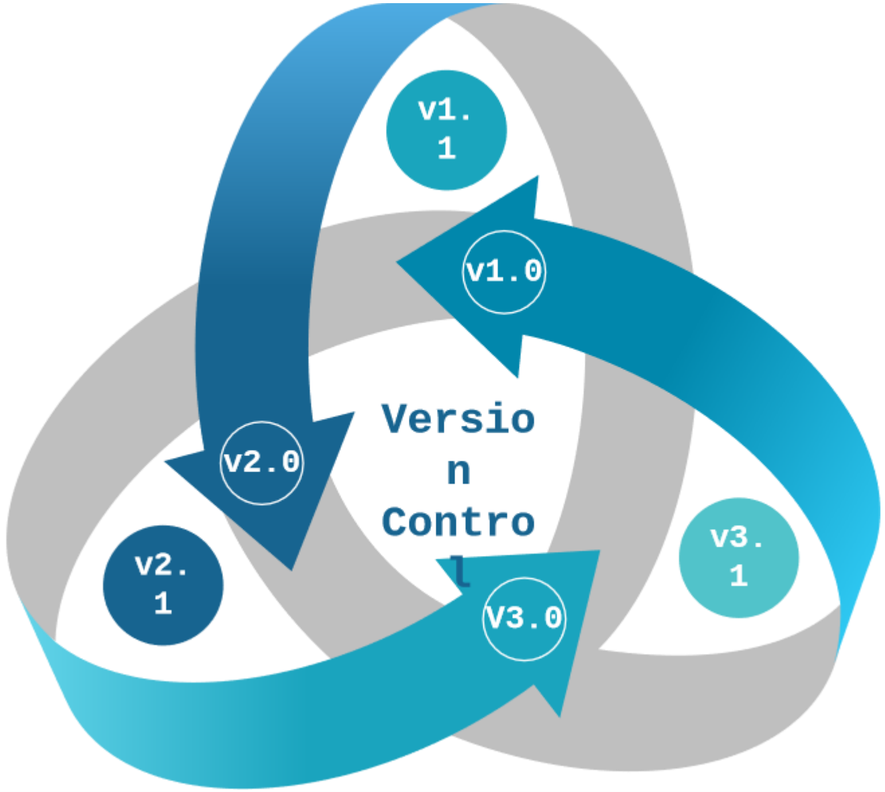
  Millions in losses

Version Control

edureka!

What Is Version Control?

Version Control is a system that documents changes made to a file or a set of files. It allows multiple users to manage multiple revisions of the same unit of information. It is a snapshot of your project over time.



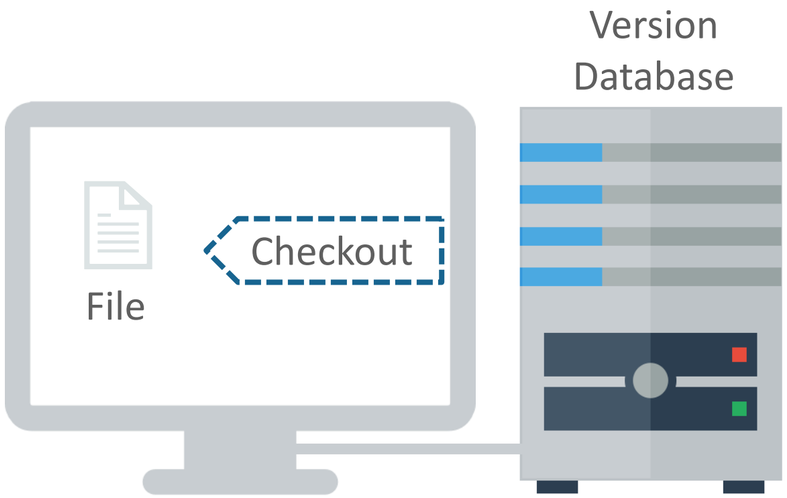
edureka!

Version Control Types

Local Version Control (LVC)

- The practice of having the Version Database in the local computer

- Local database keeps a record of the changes made to files in version database

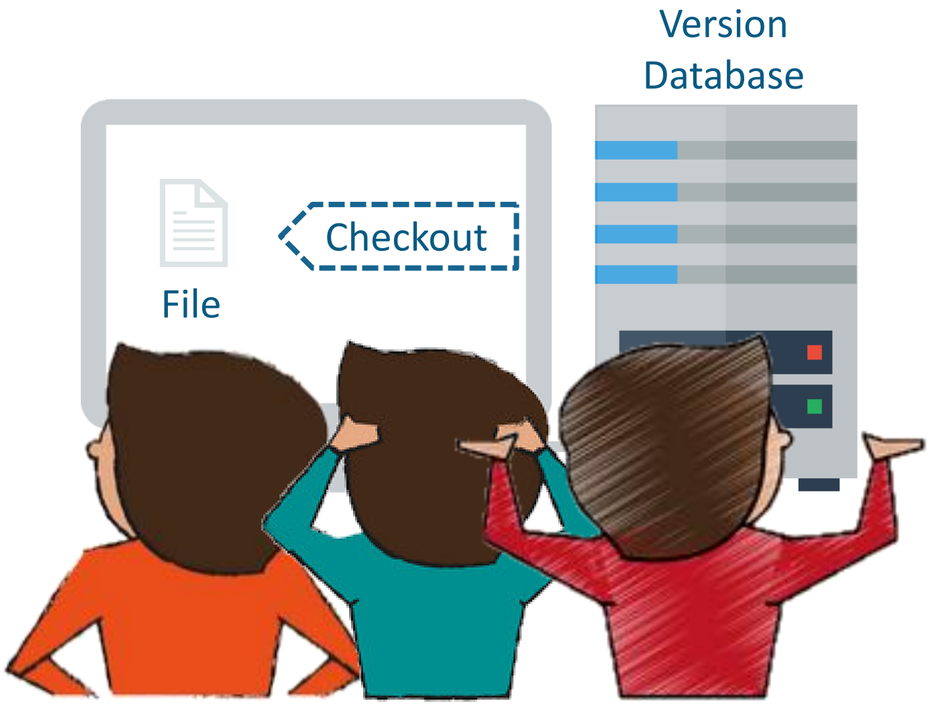


Version Control Types

Local Version Control: Issue

- **Issue:** Multiple people parallelly working on the same project

- **Solution:** Centralized Version Control



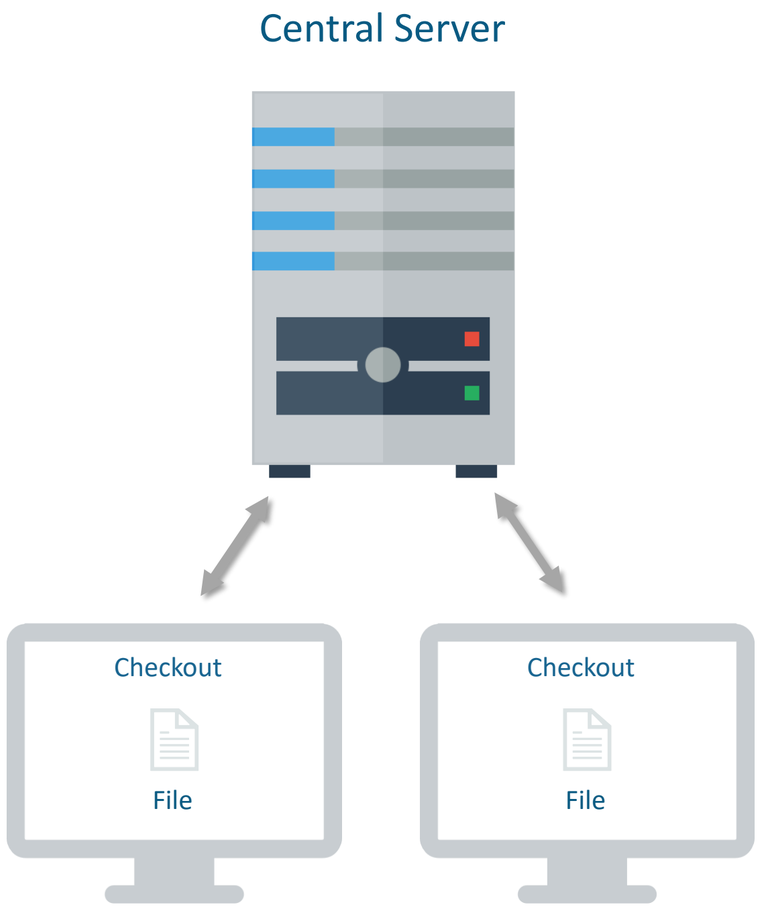
Version Control Types

Centralized Version Control (CVC)

- Local Version Control's issues are resolved by Centralized Version Control

- In CVC, a central repository is maintained where all the versioned files are kept

- Now users can checkout, and check-in files from their different computers at any time

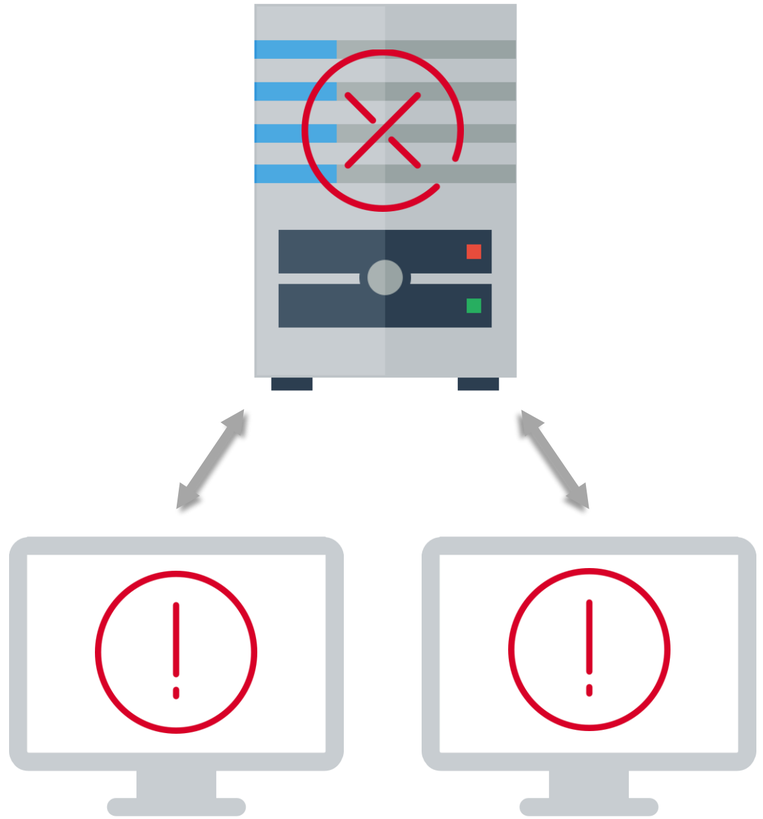


Version Control Types

Centralized Version Control: Issue

- **Issue:** In case of central server failure whole system goes down

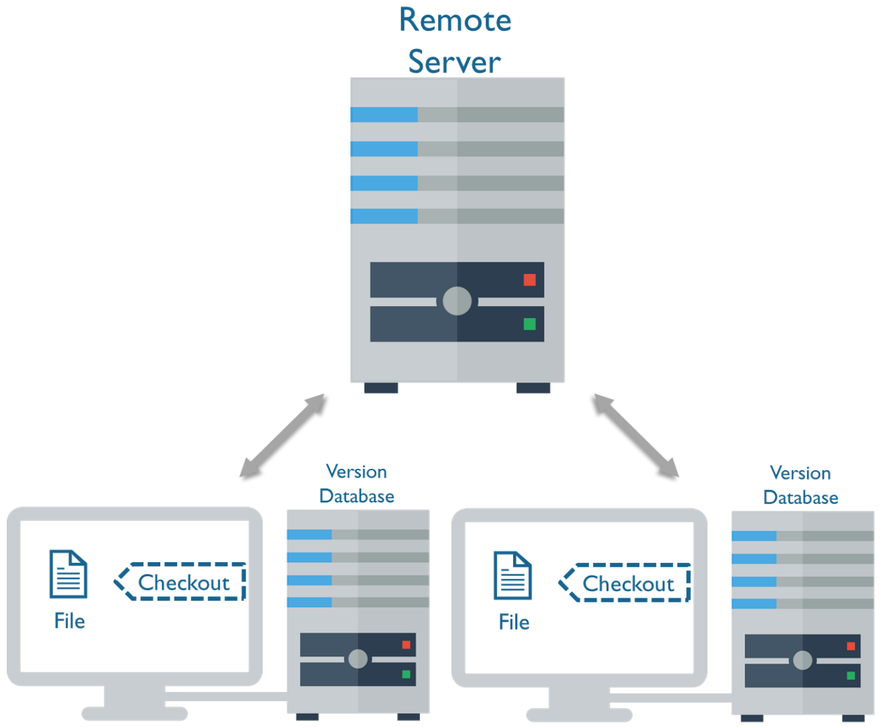
- **Solution:** Distributed Version Control



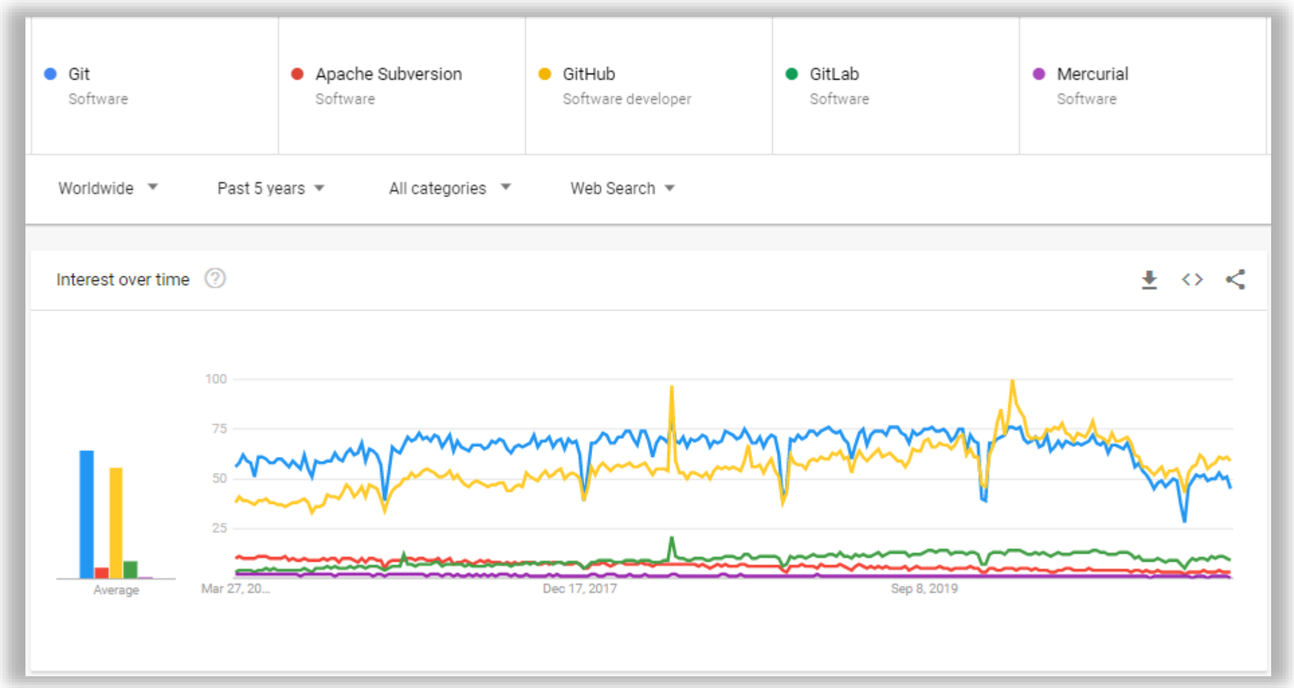
Version Control Types

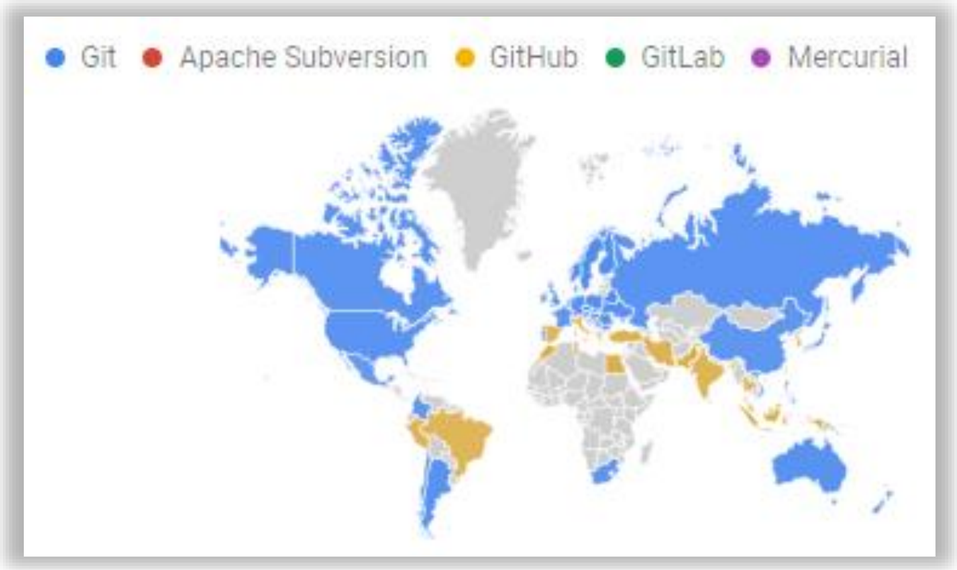
Distributed Version Control

- Version Database is stored at every users' local system and at the remote server
- Users manipulate the local files and then upload the changes to the remote server
- If any of the servers die, a client server can be used to restore



Why Git?





Note: The above graph shows the usability of the popular VCS tools in the past 5 years

edureka!

Why Git is a Clear Winner?

Snapshots

Git records changes made to a file rather than file itself. That means if a file isn't changed it isn't stored again.

Fast Operations

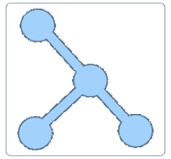
Almost every operation on git is local, hence the speed offered by Git is lightning fast compared to other VCS's.

Distributed

Every user has his own copy of the repository data stored locally allowing full functionality even on disconnection.

Branch Handling

Every collaborator's working directory is in itself a branch. Different branches can be merged with ease.



Integrity

Check-sum before storing ensures that you can't make any changes to anything without Git recording that change.

Robust

Nearly every task in Git is undoable and it is really hard to lose any change or data in Git.

edureka!

Introduction to Git

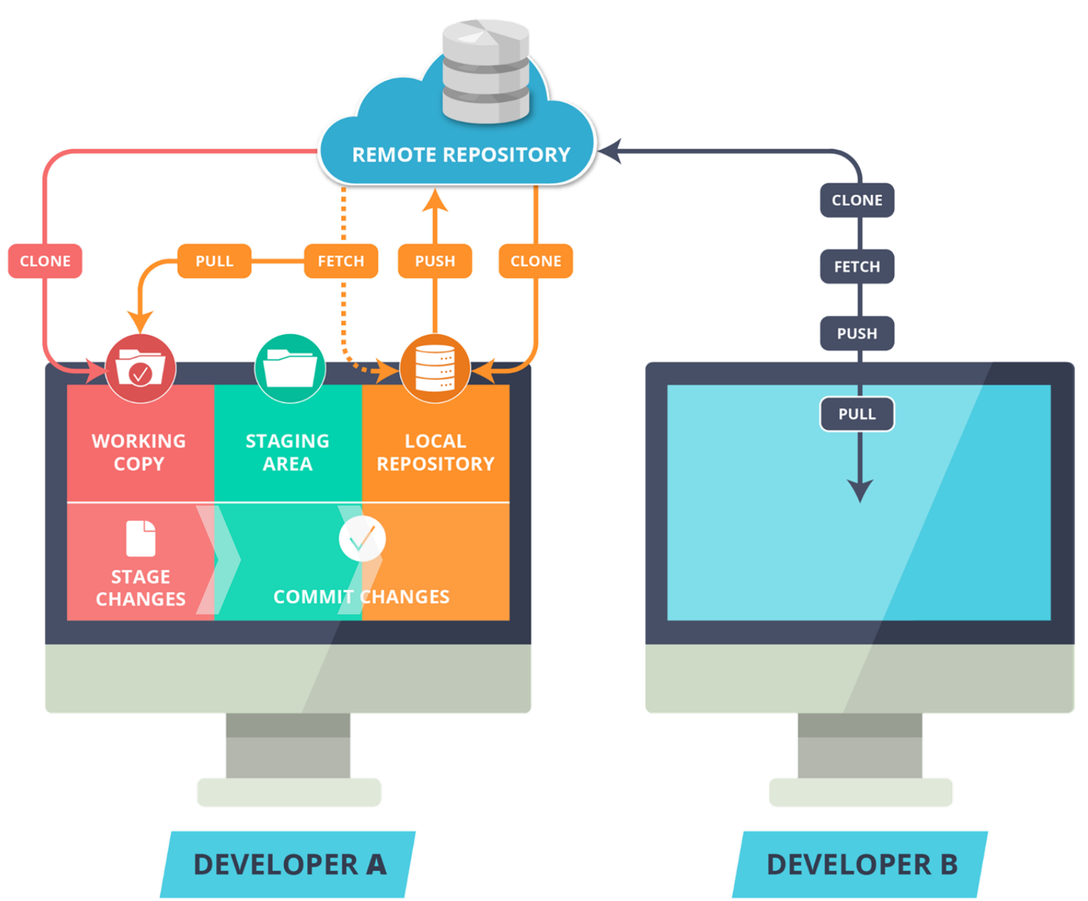
edureka!

What is Git?

Git is an open-source Distributed Version Control System(DVCS) which records changes made to the files laying emphasis on **speed**, **data integrity** and **distributed**, **non-linear workflows**

The Git File Workflow

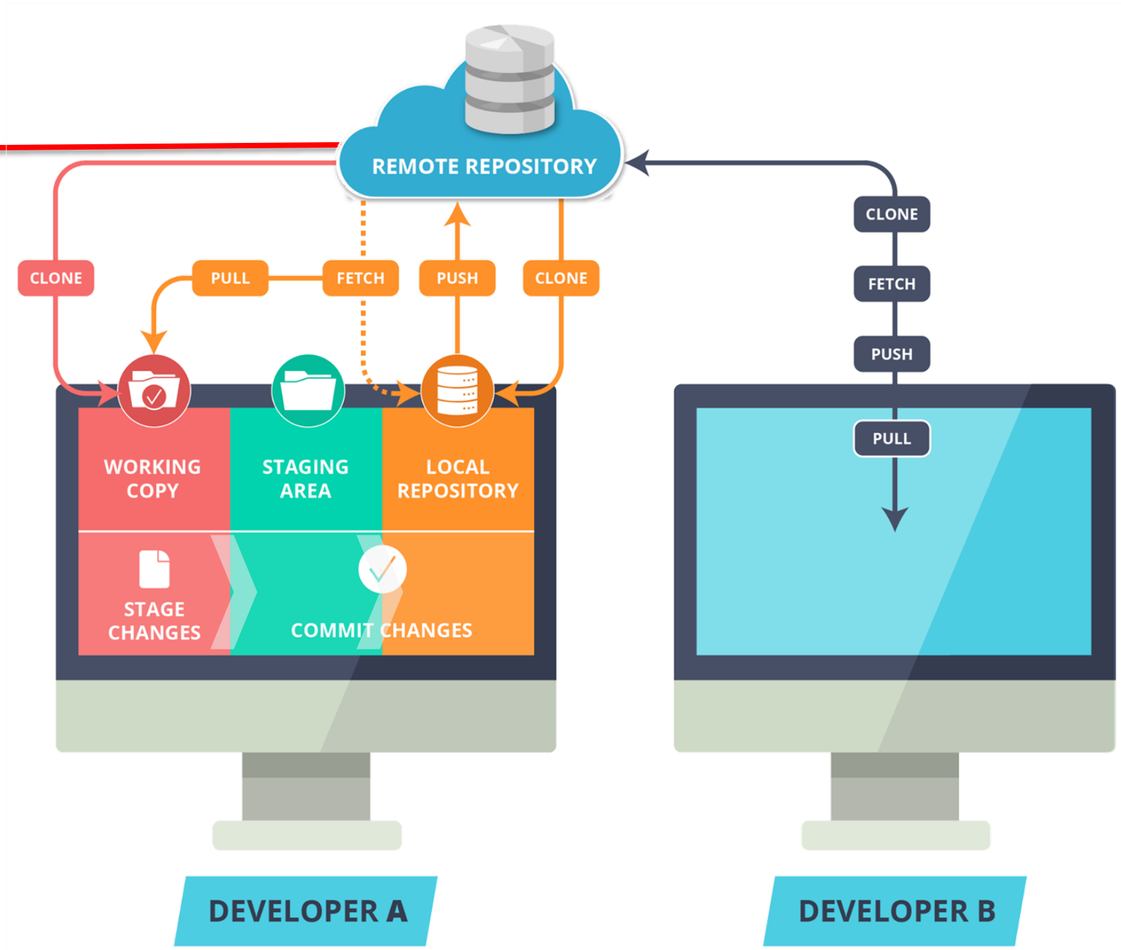
- Use Git workflow to manage your project effectively
- Working with set of guidelines increases Git's consistency and productivity



edureka!

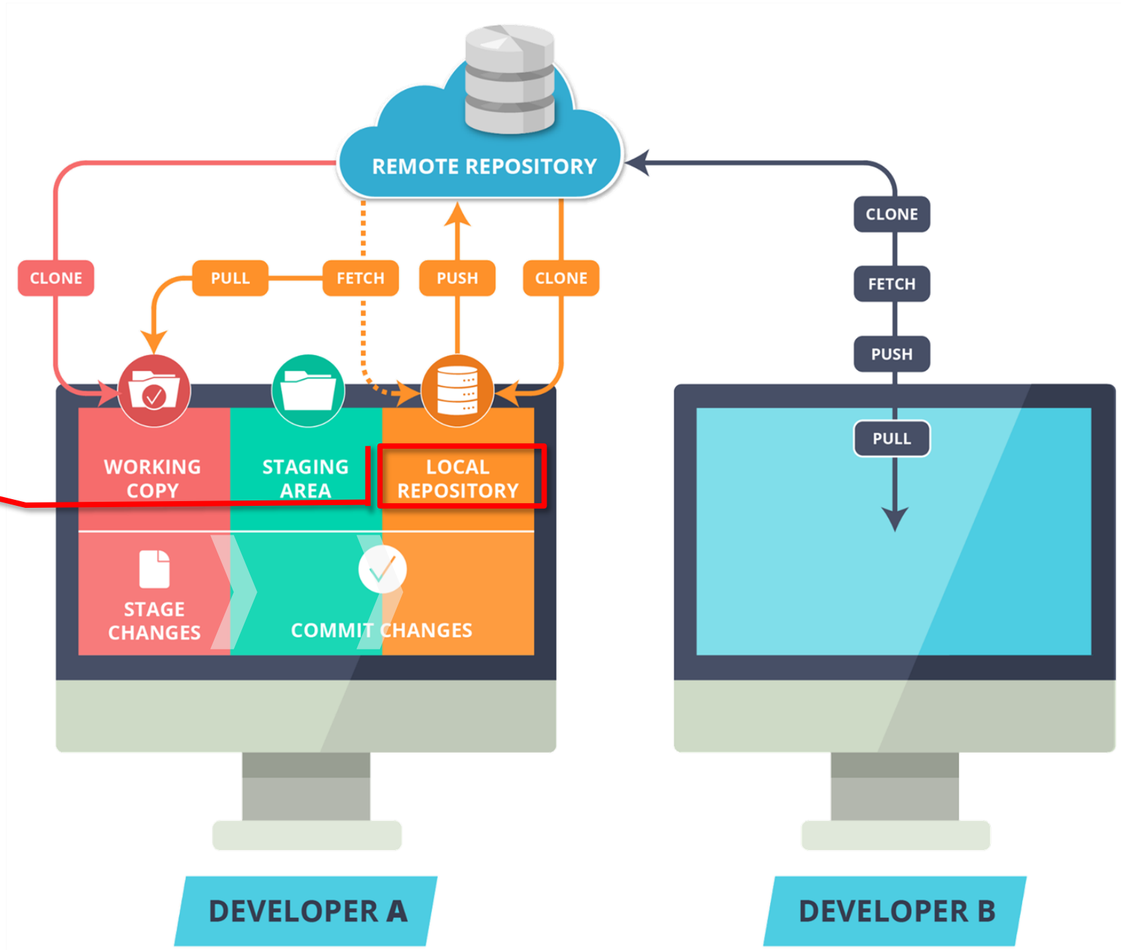
The Git File Workflow

The Remote Repository is the server where all the collaborators upload changes made to the files



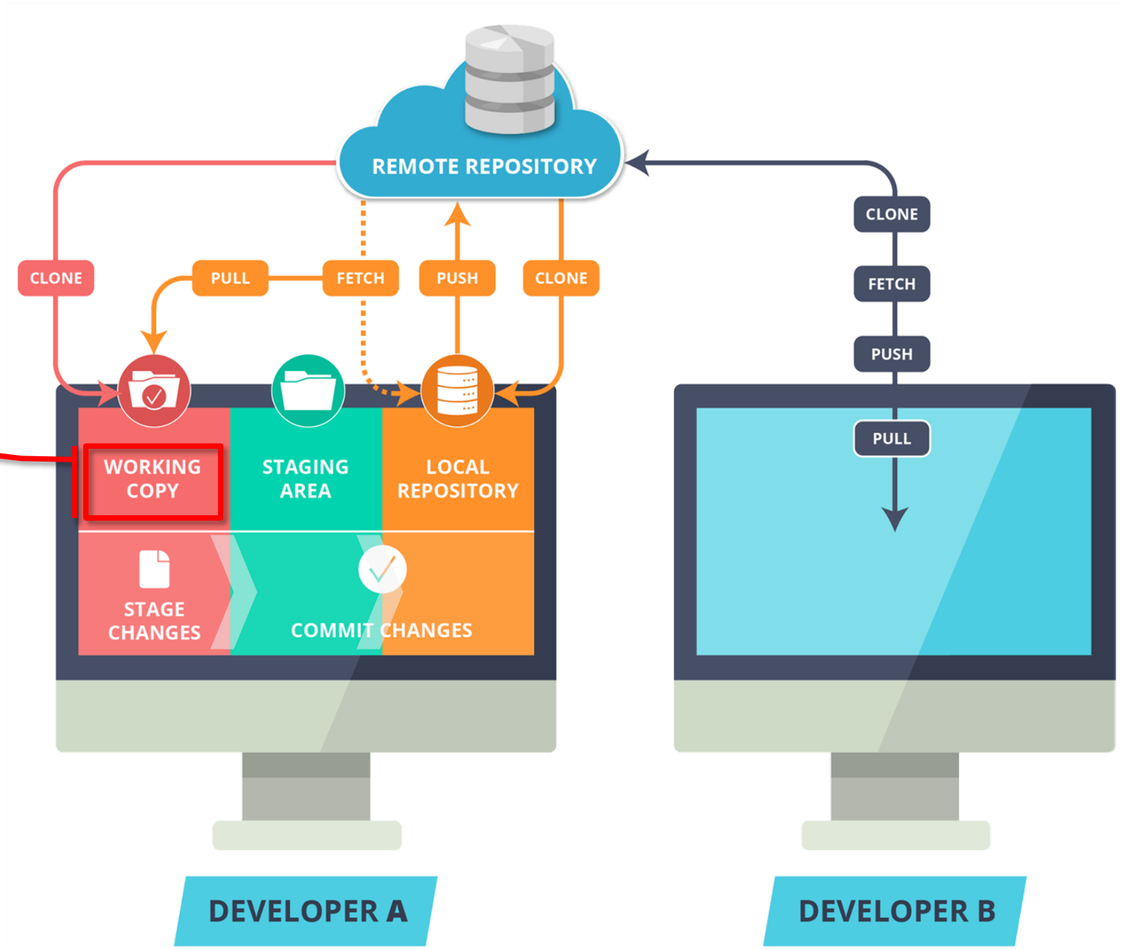
The Git File Workflow

- "**Local Repository**" is user's copy of the Version Database
- The user accesses all the files through local repository and then push the change made to the "**Remote Repository**"



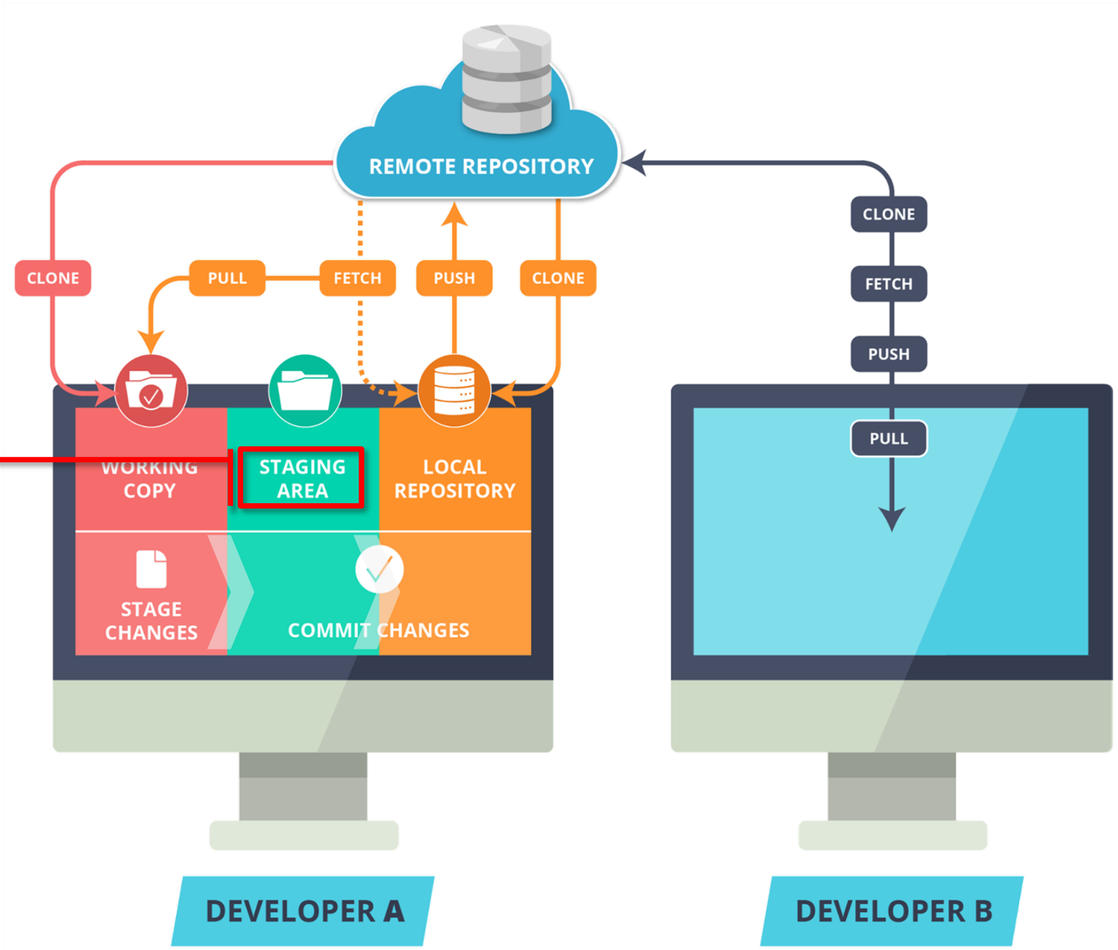
The Git File Workflow

- **“Workspace”** is user’s active directory
- The user modifies existing files and creates new files in this space. Git tracks these changes compared to your Local Repository



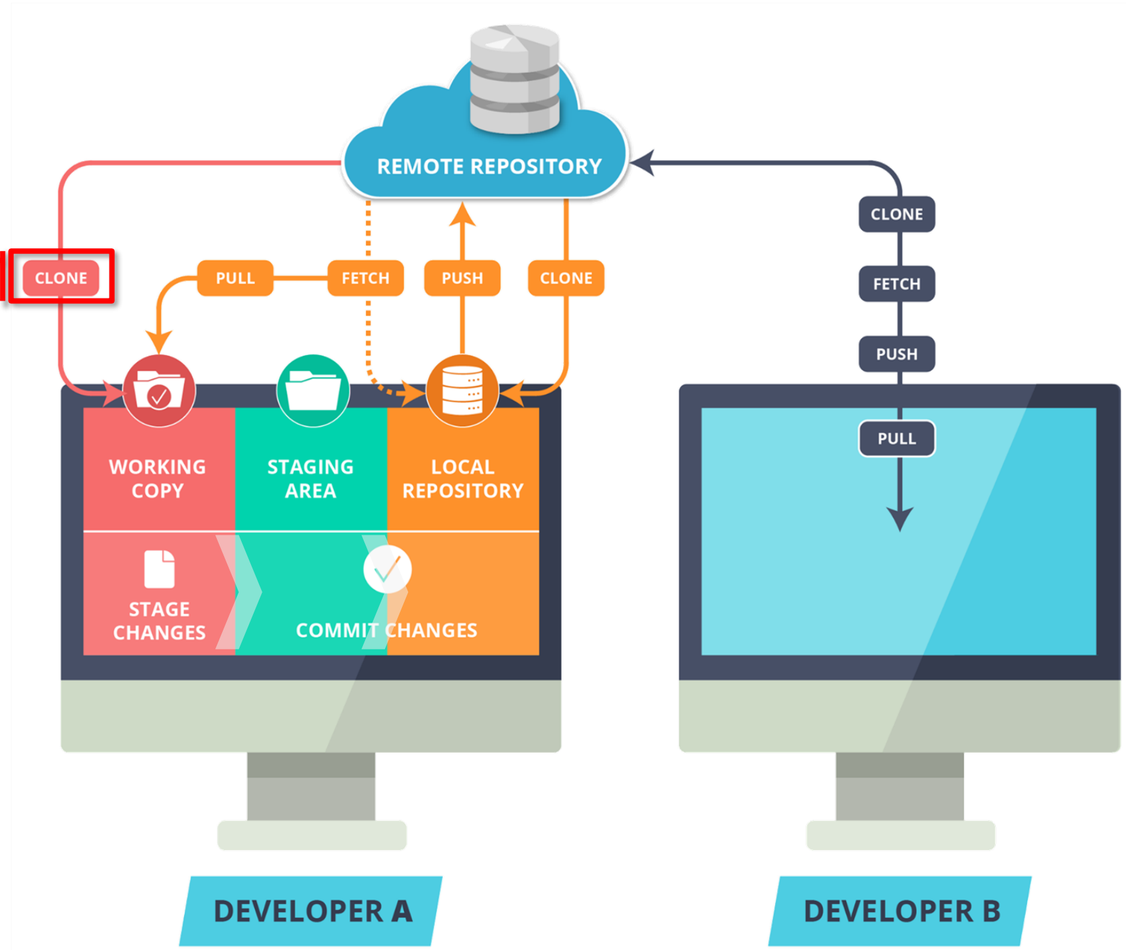
The Git File Workflow

Stage is a place where all the modified files marked to be committed are placed.



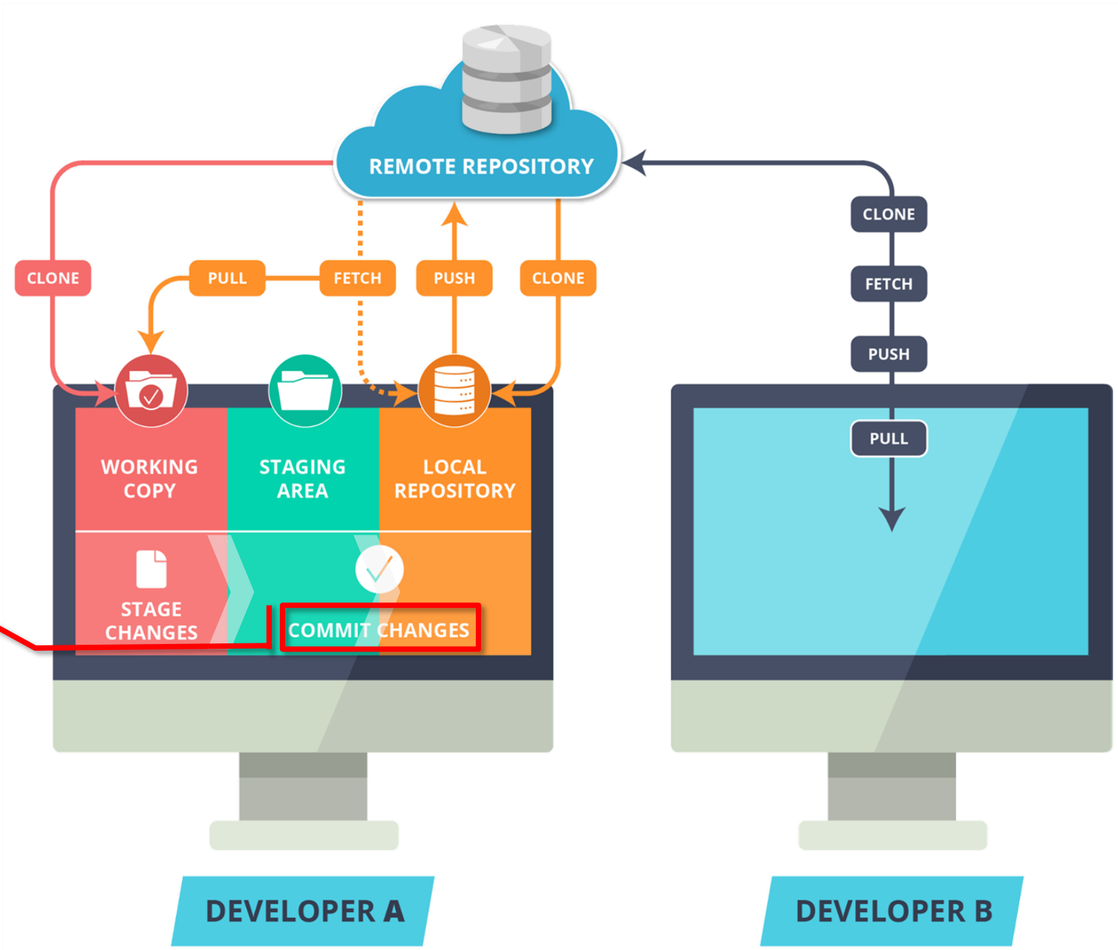
The Git File Workflow

Clone command creates a copy of an existing Remote Repository inside the Local Repository.



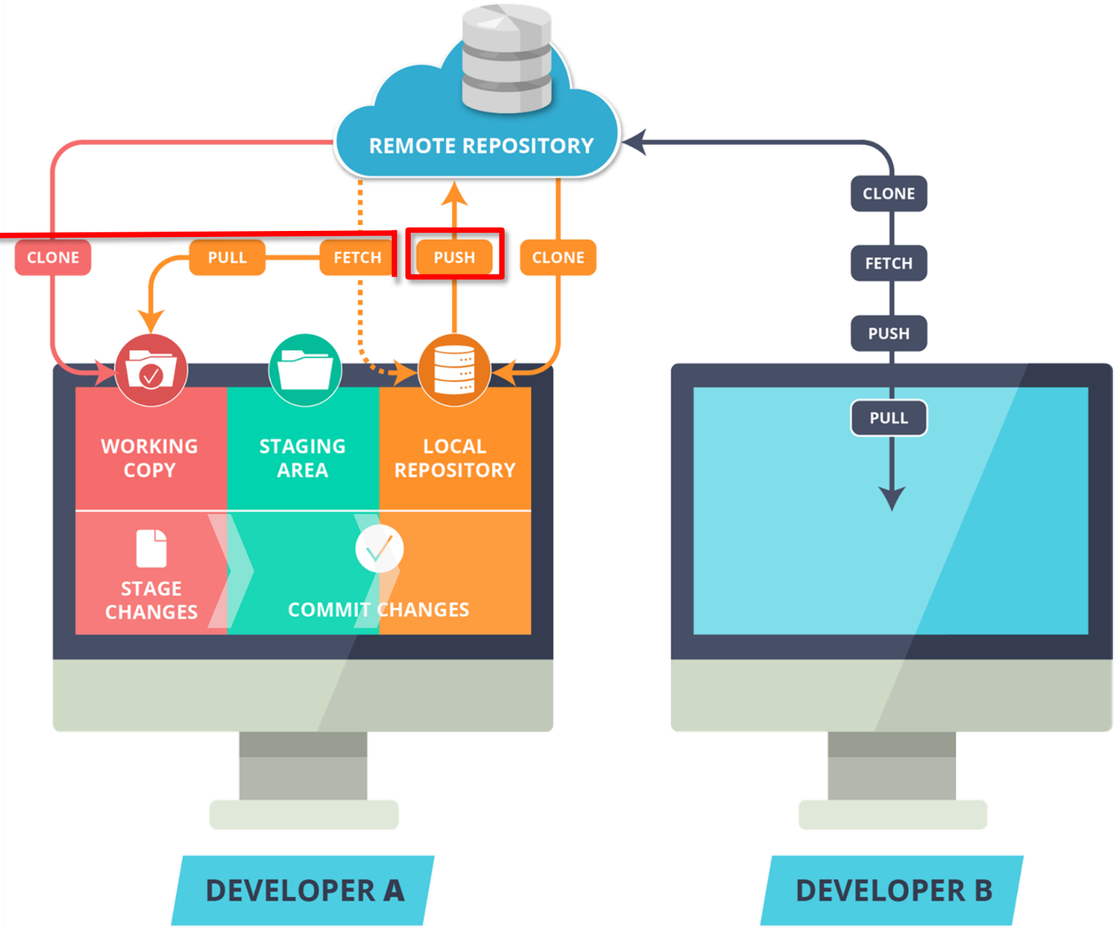
The Git File Workflow

Commit command commits all the files in the staging area to the local repository.



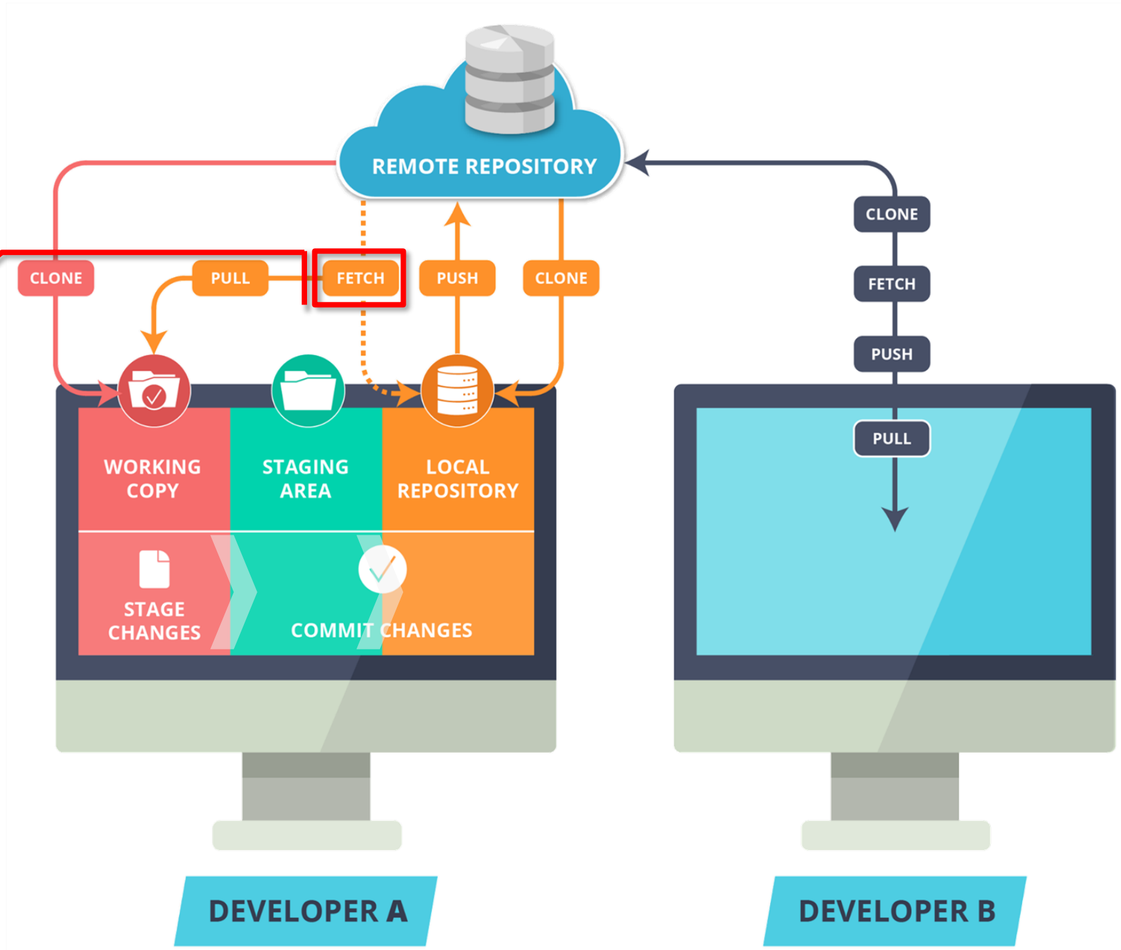
The Git File Workflow

Push command pushes all the changes made in the Local Repository to the Remote Repository



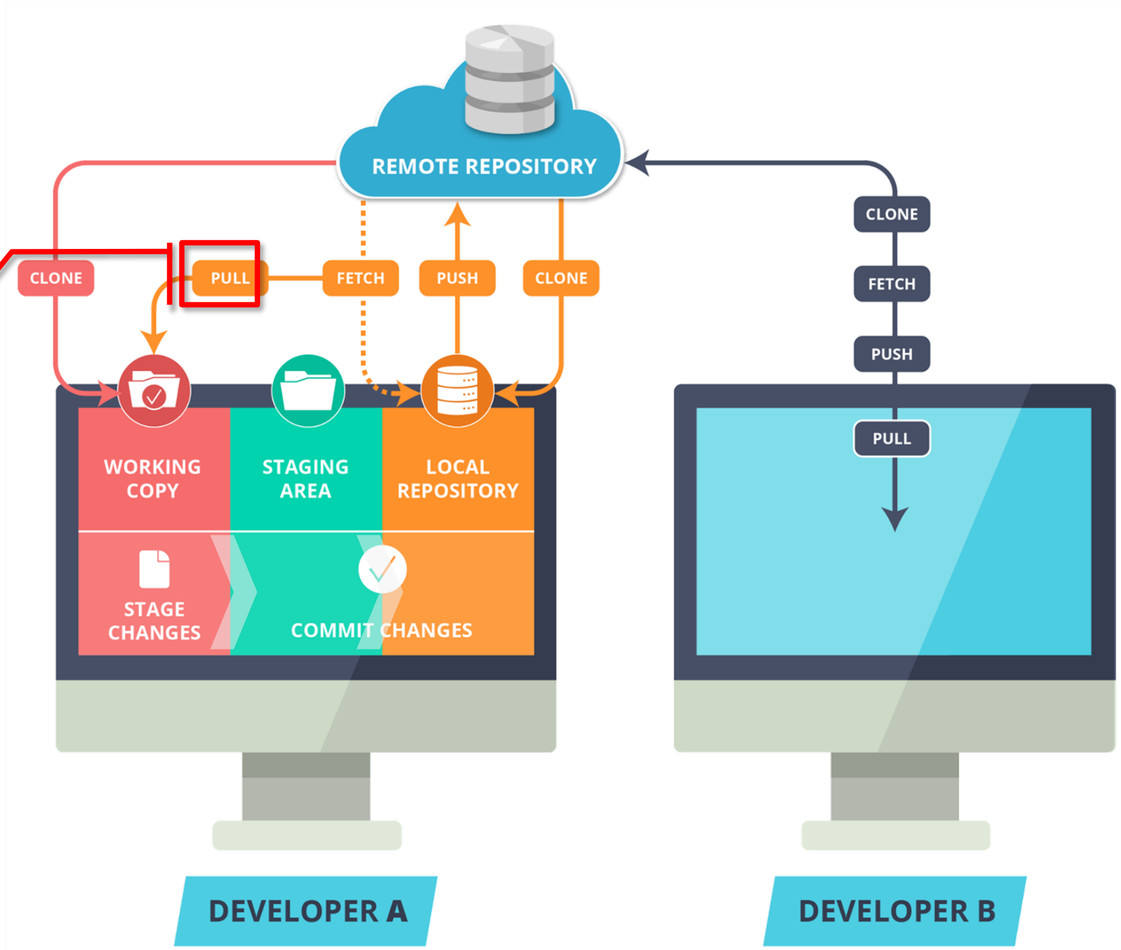
The Git File Workflow

Fetch command collects the changes made in the Remote repository and copies them to the Local Repository. This command doesn't affect our Workspace.



The Git File Workflow

- Pull like Fetch, gets all the changes from the remote repository and copies them to the Local Repository
- Pull merges those changes to the current working directory



Git Common Commands

edureka!

Adding Files And Checking Status

- To **add** a file to the **staging area**

```
Syntax: git add <filename>
```

- To check the working tree **status**

```
Syntax: git status
```

```
[labuser@master Demo]$ git add Demo.java
[labuser@master Demo]$ git status
On branch <Edureka>

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Demo.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Demo.class

[labuser@master Demo]$
```

Committing Changes

To **commit** the **staged** files to your **local repository:**

```
Syntax: git commit
```

```
[labuser@master ~]$ git commit
```

```
  GNU nano 2.9.3   /home/ubuntu/Demo/.git/COMMIT_EDITMSG    Modified

Added the First file to the repo
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   Demo.java
#
# Untracked files:
```

Commit message and description can be added here

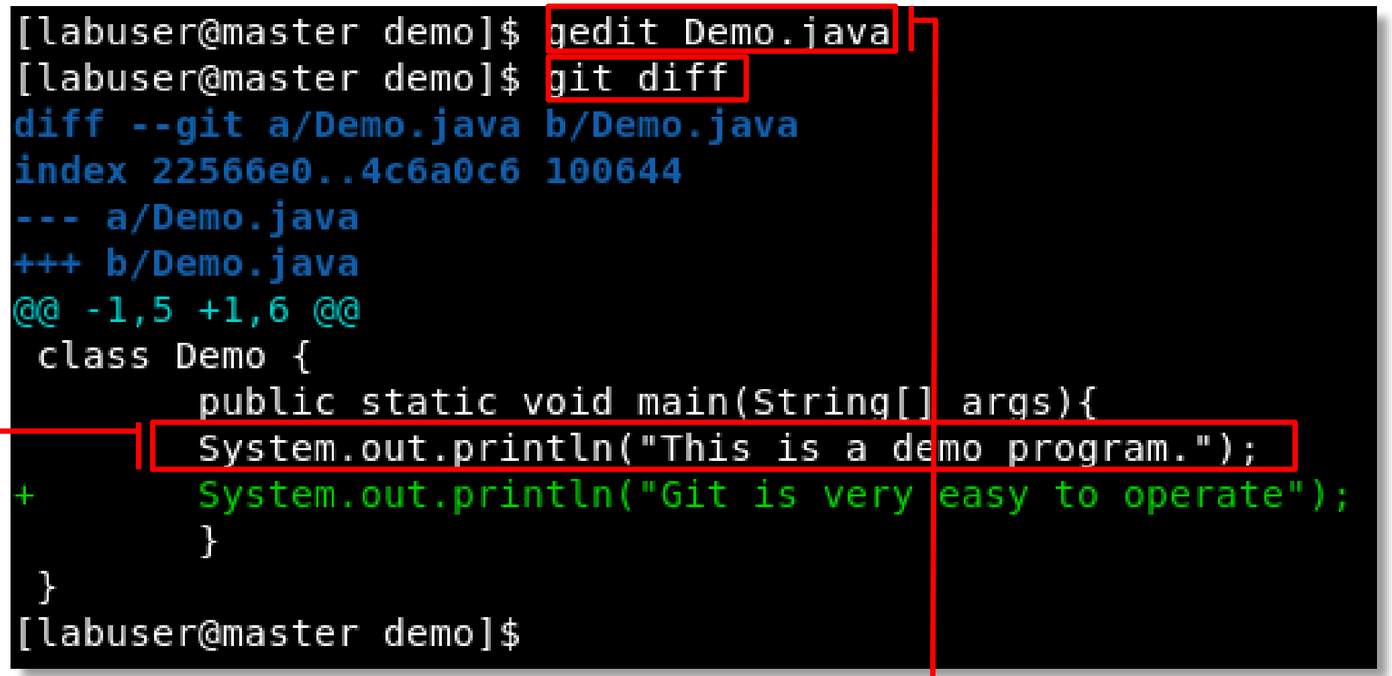
```
[labuser@master demo]$ git commit
[<Edureka> (root-commit) d9e4edf] Added the first file to the repo
 1 file changed, 5 insertions(+)
 create mode 100644 Demo.java
[labuser@master demo]$
```

edureka!

Tracking Changes

The git **diff** command displays all the changes made to the **tracked** files

Syntax: git diff



The changes made are displayed on the console

Changes made to the **Demo.java** file

Staging And Committing Multiple Files

- To **stage and commit** multiple files at once we use -a flag with the commit command

- Commit with -a flag automatically stages all the modified files and commits changes to the local repository

```
Syntax: git commit -a -m ‘message’
```

```
[labuser@master demo]$ git status
On branch <Edureka>
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Demo.java
        new file:   Demo1.class
        new file:   Demo1.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Demo.class

[labuser@master demo]$
```

Staged and unstaged files before **commit -a**

Staging And Committing Multiple Files

```
[labuser@master demo]$ git commit -a -m 'New files added and Demo.java modified'
[<Edureka> e922000] New files added and Demo.java modified
 3 files changed, 8 insertions(+)
 create mode 100644 Demo1.class
 create mode 100644 Demo1.java
[labuser@master demo]$
```

```
[labuser@master demo]$ git status
On branch <Edureka>
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Demo.class

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Demo.java
```

Removing Files From Repository

- The **git rm** command deletes the file from git repository as well as users system

```
Syntax: git rm <filename>
```

```
[labuser@master demo]$ git rm Demo.class
```

- To remove the file from git repository but not from the system, we use **--cached** option

```
Syntax: git rm --cached <filename>
```

```
[labuser@master demo]$ git rm --cached Demo.class
rm 'Demo.class'
[labuser@master demo]$
```

- An error shows up if you try to delete a staged file

- You can force remove a staged file by using **–f** flag

```
Syntax: git rm -f <filename>
```

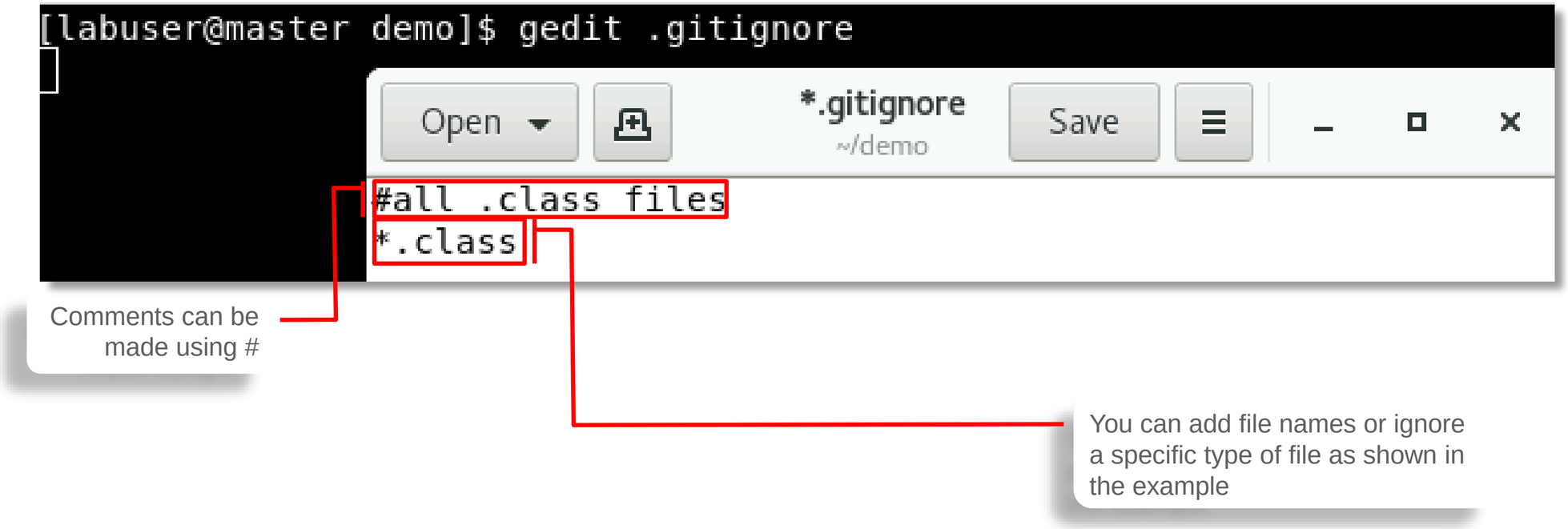
Ignoring Files

What can you do if you don’t want Git to track select files in the repository?

- You can create a .gitignore file in your repository and mention all the files you want Git to ignore.
- Alternatively you can also download the .gitignore file from github.

Creating a Gitignore File

You can create a **.gitignore** file and add all the untracked files you want Git to ignore



Adding a Gitignore File in the Repository

```
[labuser@master demo]$ git status
On branch <Edureka>
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Demo.class

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Demo.java

[labuser@master demo]$
```

Before adding **.gitignore** file

```
[labuser@master demo]$ git status
On branch <Edureka>
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Demo.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
[labuser@master demo]$
```

After adding **.gitignore** file

edureka!

Commit: Git Log

The **git log** command shows all the commits so far on the current branch

```
Syntax: git log
```

```
[labuser@master demo]$ git log
commit d17da4398c8d4065a0bda537c0487428eda35a86 (HEAD -> <Edureka>)
Author: “edureka” <“content@edureka.co”>
Date:   Wed Apr 28 03:06:54 2021 +0000

    Removed class extension files

commit e922000b56b3077e006899f97fd84220c3d25308
Author: “edureka” <“content@edureka.co”>
Date:   Wed Apr 28 02:37:52 2021 +0000

    New files added and Demo.java modified

commit d9e4edf701b8ee9917596755e2baa967dae0738b
Author: “edureka” <“content@edureka.co”>
Date:   Wed Apr 28 02:29:57 2021 +0000

    Added the first file to the repo
[labuser@master demo]$
```

edureka!

Commit: Amend

You can change the last commit message using --amend option

```
[labuser@master demo]$ git commit --amend
[<Edureka> 9941162] Removed class extension files
 Date: Wed Apr 28 03:06:54 2021 +0000
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 Demo1.class
[labuser@master demo]$
```

Commit Tags

What if I want to access a particular commit?

- You don’t have to remember the entire hexcode(commitID) of that commit
- You can use commit tags as aliases and keep the track of different commits

Commit: Git Tag

- Commit tags provide an alias for commitID

```
Syntax: git tag --a <annotation> --m <message>
```

- You can also view all the tags you have created

```
Syntax: git tag
```

```
[labuser@master demo]$ git tag -a v1.3 -m 'Version 1.3'
[labuser@master demo]$ git tag
v1.3
[labuser@master demo]$
```

Commit Tags

- Adding a tag to one of the previous commits

```
Syntax: git tag --a <annotation> <commit id> --m <message>
```

- Commit id can be obtained from git logs

- All these tags can be viewed in git

```
Syntax: git show <tag-name>
```

```
[labuser@master demo]$ git tag -a v0.3 99411624fcc4647169bdc6b22b52dabdf9f2c84c -m 'Added ta
g to previous commit'
[labuser@master demo]$ git show v0.3
tag v0.3
Tagger: “edureka” <“content@edureka.co”>
Date:   Wed Apr 28 03:17:24 2021 +0000

Added tag to previous commit

commit 99411624fcc4647169bdc6b22b52dabdf9f2c84c (HEAD -> <Edureka>, tag: v1.3, tag: v0.3)
Author: “edureka” <“content@edureka.co”>
Date:   Wed Apr 28 03:06:54 2021 +0000

    Removed class extension files

diff --git a/Demo1.class b/Demo1.class
deleted file mode 100644
index dbd7e9c..0000000
Binary files a/Demo1.class and /dev/null differ
[labuser@master demo]$
```

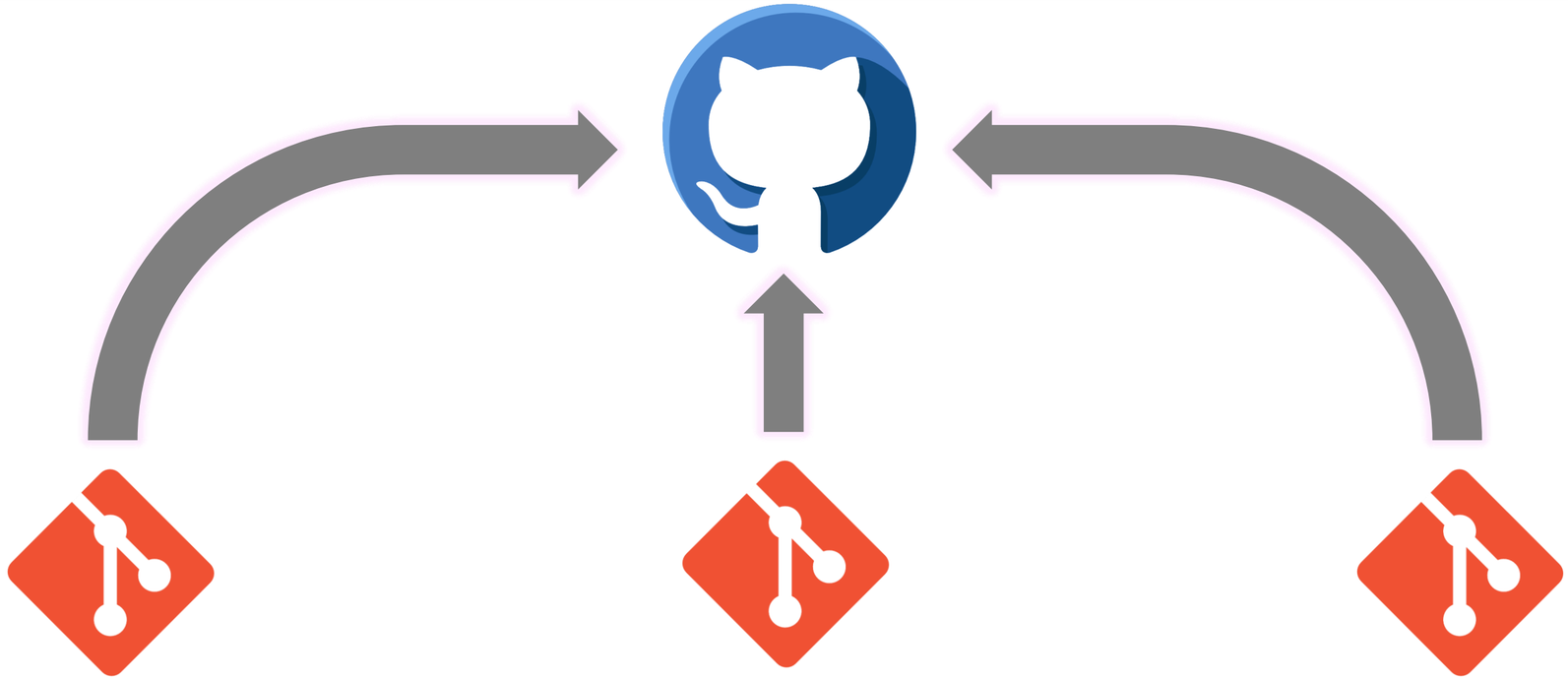
edureka!

Working With Remote Repositories

edureka!

What Is A Remote Repository?

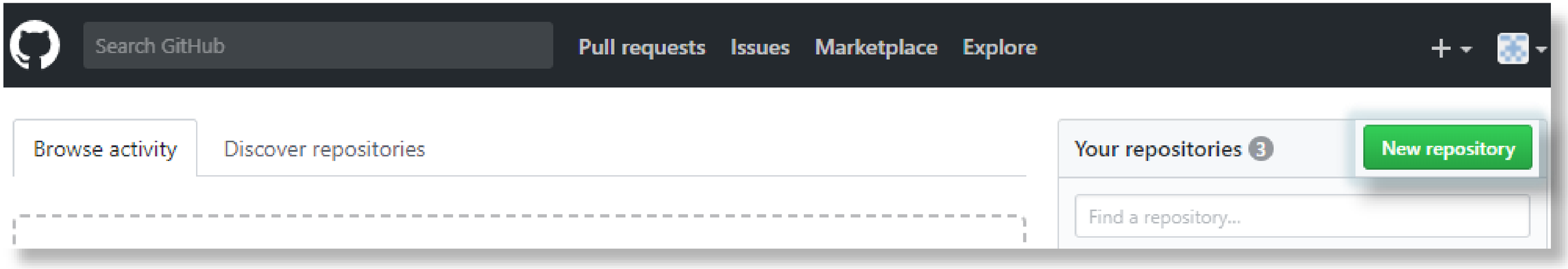
Mostly the users work on a local repository. But in order to collaborate with other people, we use a remote repository. A remote repository is place where the users upload and share their commits with other collaborators.



Creating A Remote Repository: New Repository

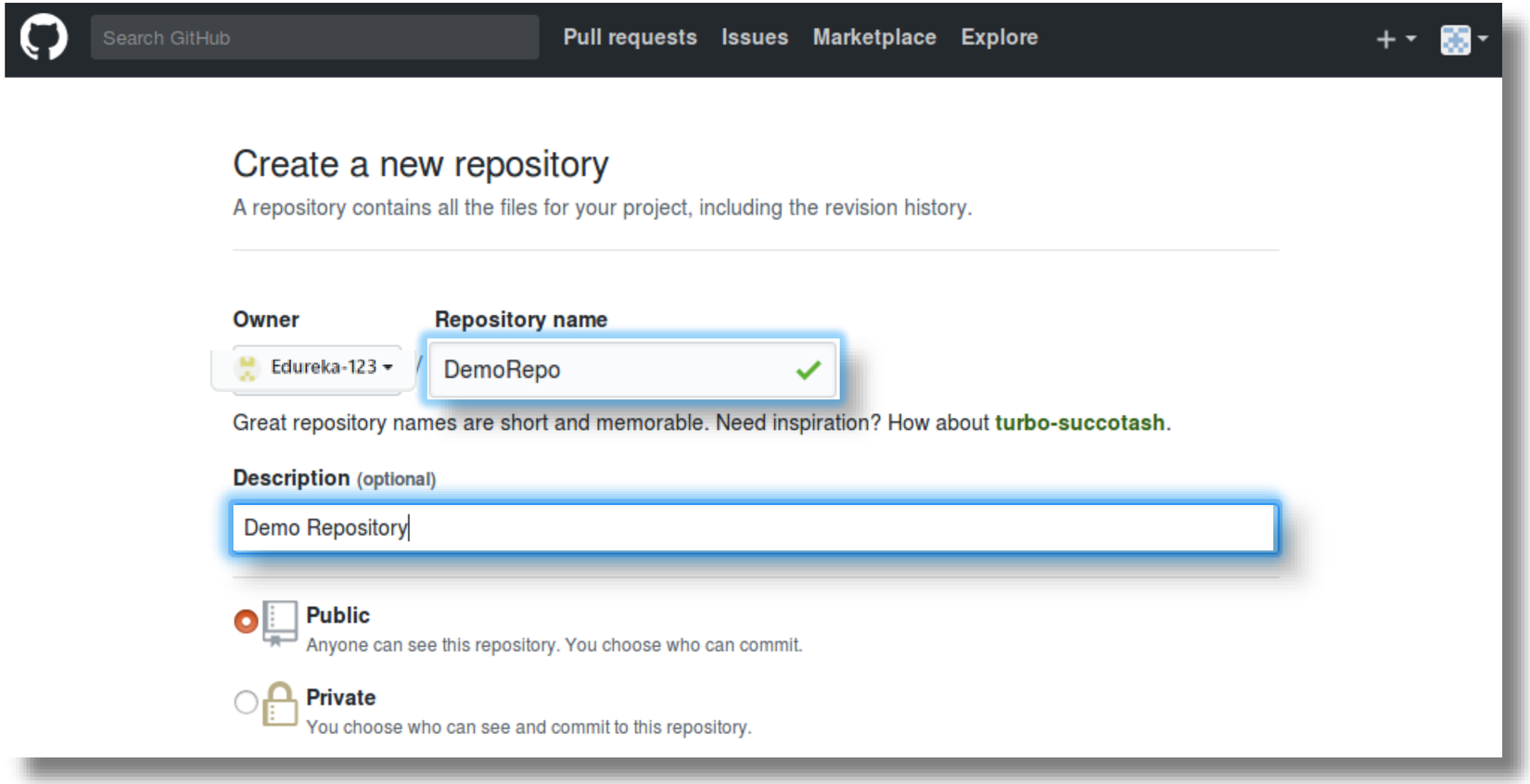
- Sign-up at **github.com**

- Click on New repository to create a new repository



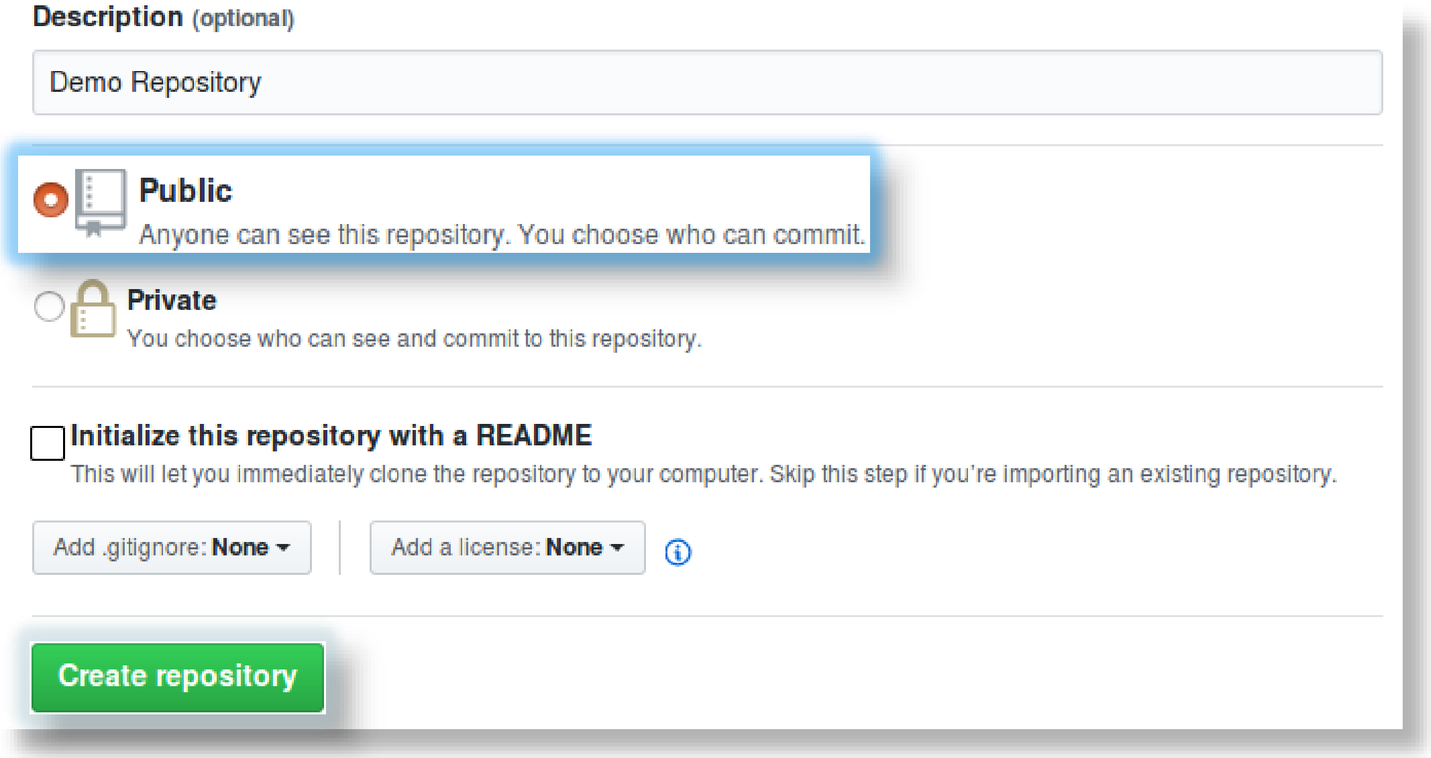
Creating A Remote Repository: Adding Description

- Under **Repository name**, give a name to your repository
- Give some Description about your repository under **Description** section.



Creating A Remote Repository: Create Repository

- For a free repository choose public
- For a private repository, a monthly premium needs to be paid
- Finally click on Create Repository

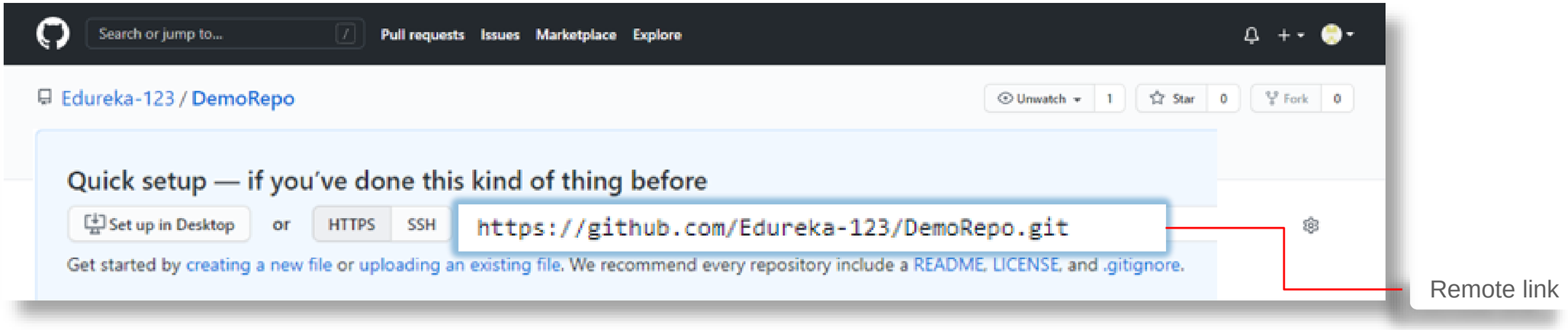


Adding Remote To Local

To add Remote repository to local use **git add remote** followed by remote link

```
Syntax: git add remote origin <remote link>
```

```
[labuser@master demo]$ git remote add origin https://github.com/Edureka-123/DemoRepo.git
[labuser@master demo]$
```



Push Local Repository To Remote

To push **Local repository** to remote use **push** command

```
Syntax: git push origin master
```

Origin is used as an alias for your remote

```
[labuser@master demo]$ git push origin master
Username for 'https://github.com': Edureka-123
Password for 'https://Edureka-123@github.com':
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 2 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (10/10), 1.20 KiB | 1.20 MiB/s, done.
Total 10 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Edureka-123/DemoRepo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Refers to master branch in local repo

Pushing Tags

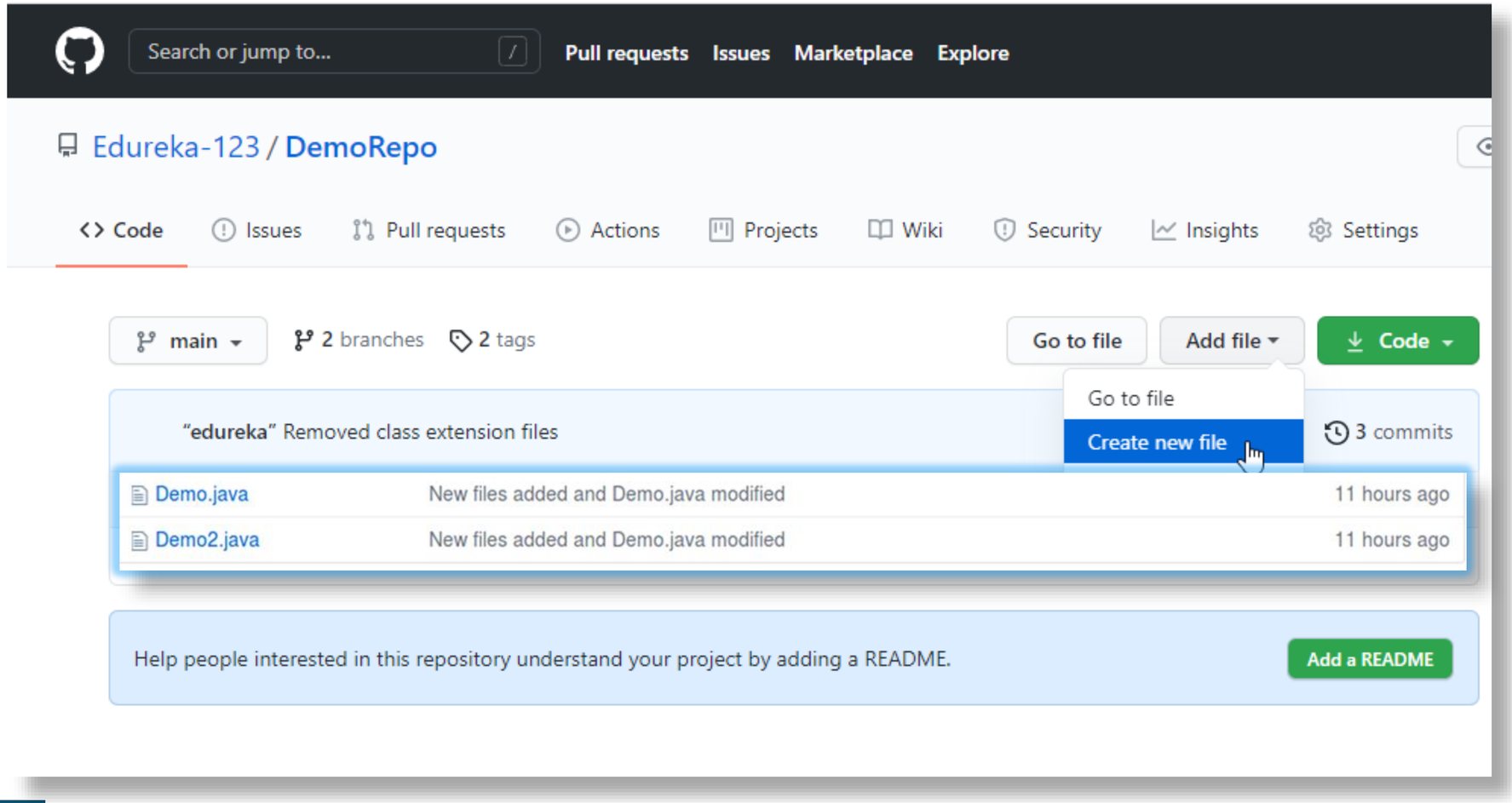
Tags can be pushed, viewed and shared on Remote

```
Syntax: git push origin --tags
```

```
[labuser@master demo]$ git push origin --tags
Username for 'https://github.com': Edureka-123
Password for 'https://Edureka-123@github.com':
Enumerating objects: 2, done.
Counting objects: 100% (2/2), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 234 bytes | 234.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/Edureka-123/DemoRepo.git
 * [new tag]         v0.3 -> v0.3
 * [new tag]         v1.3 -> v1.3
[labuser@master demo]$
```

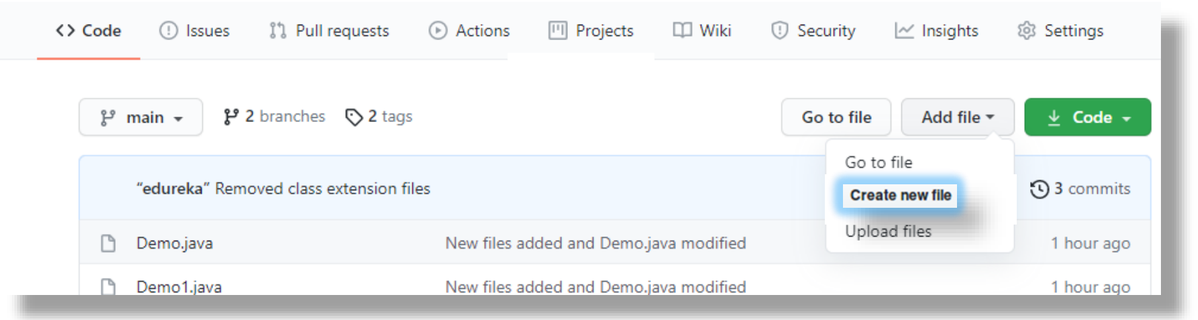
Push Local Repository To Remote

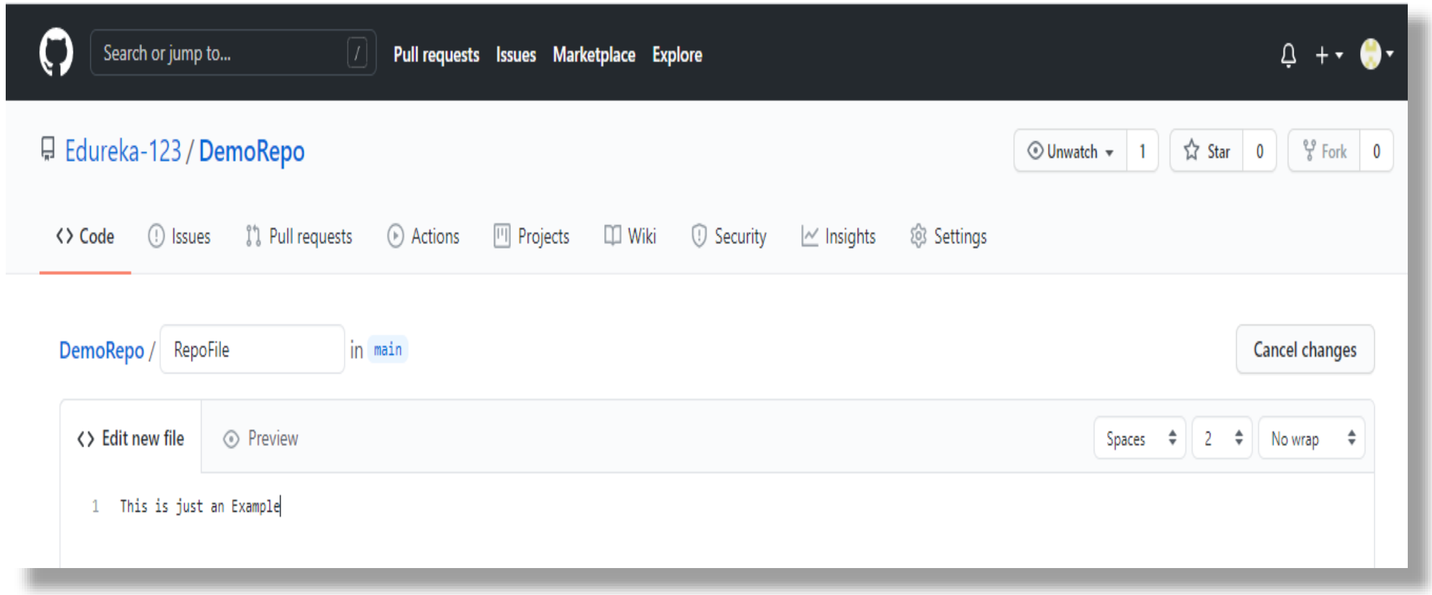
The changes can be seen in Remote repository



Working On Remote

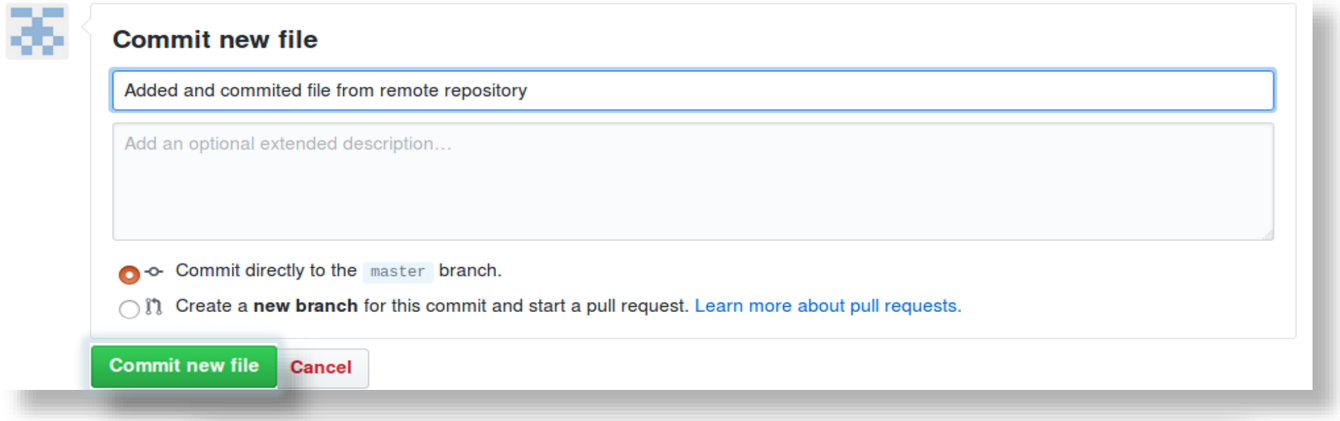
Files can be created and edited on remote

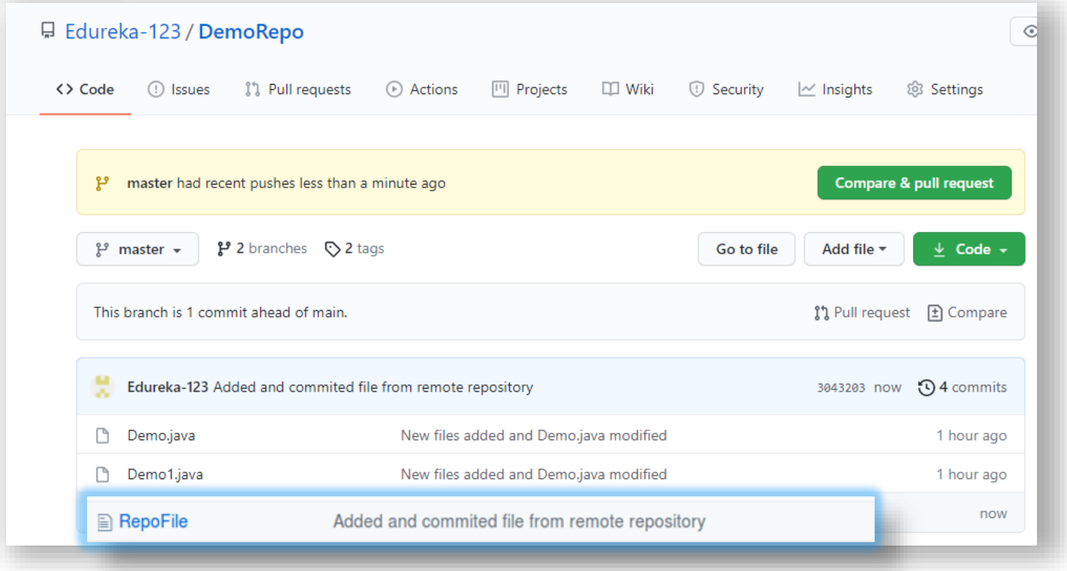




Working On Remote

These files can then be committed on the remote





edureka!

Remote List

To list all the remotes attached to your Local repository

```
Syntax: git remote -v
```

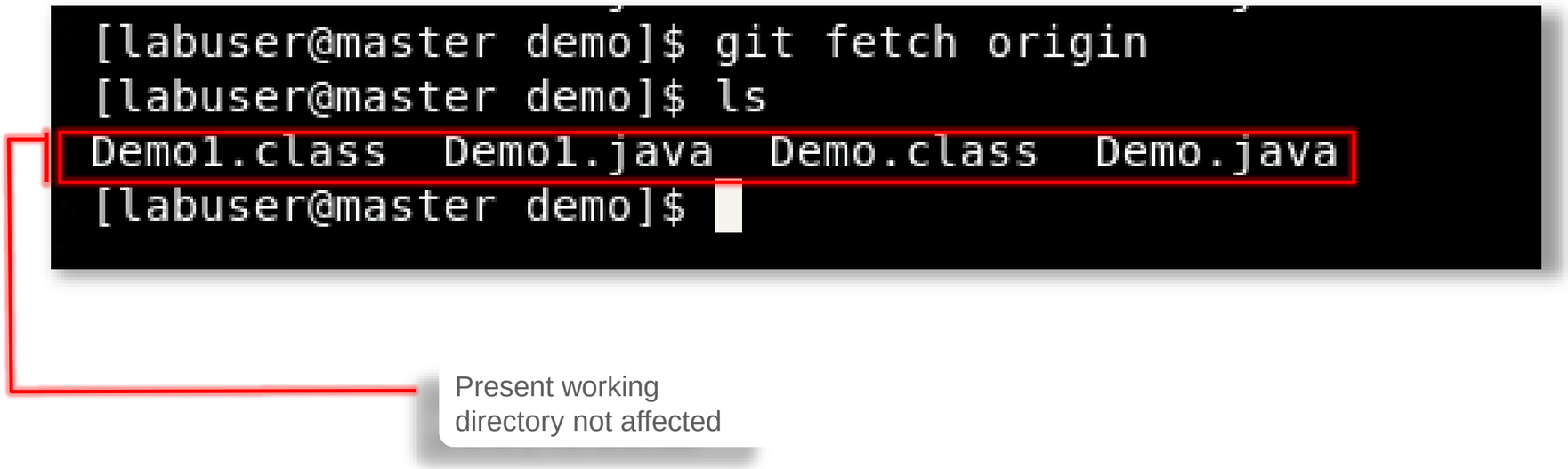
```
[labuser@master demo]$ git remote -v
origin  https://github.com/Edureka-123/DemoRepo.git (fetch)
origin  https://github.com/Edureka-123/DemoRepo.git (push)
[labuser@master demo]$
```

Fetch

- **Fetch** command copies the changes from remote to local repository

```
Syntax: git fetch origin
```

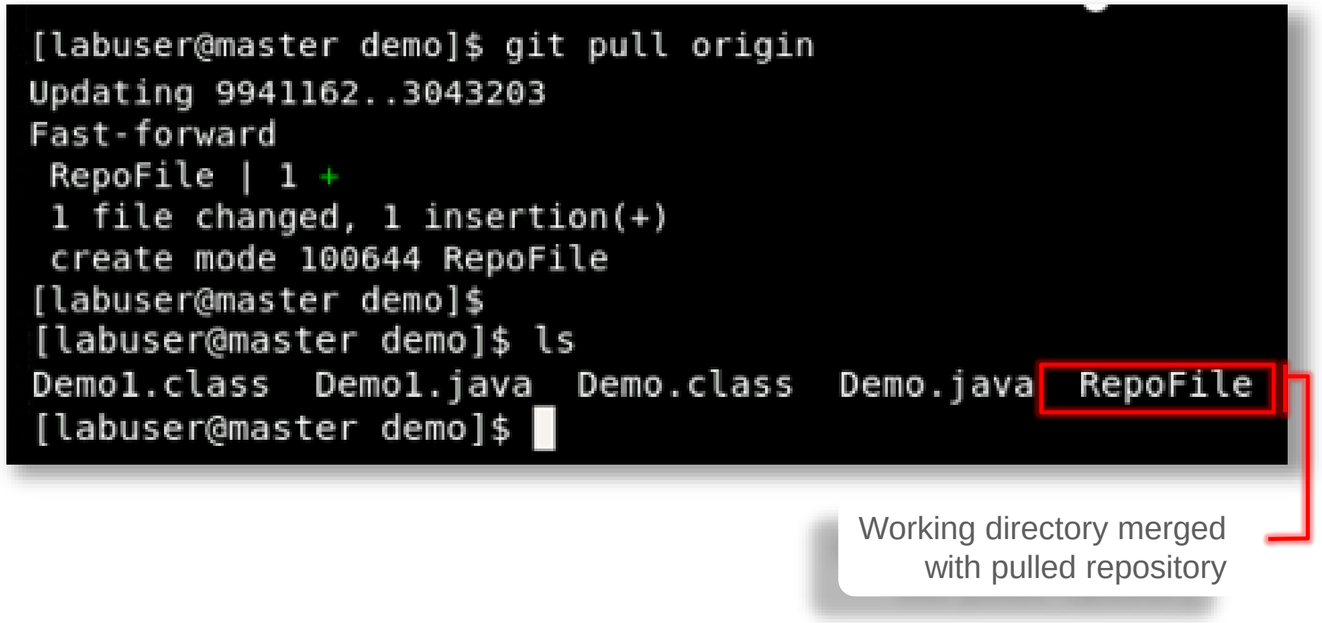
- Fetch **does not** affect the present working directory



Pull

- Pull copies all the changes from remote to local repository
- It then merges the changes with the present working directory

```
Syntax: git pull origin
```



edureka!

Summary

What Is Version Control?

Version Control is a system that documents changes made to a file or a set of files. It allows multiple users to manage multiple revisions of the same unit of information. It is a snapshot of your project over time.

edureka!

Why Git is a Clear Winner?

Snapshots
Git records changes made to a file rather than file itself. That means if a file isn't changed it isn't stored again.

Fast Operations
Almost every operation on git is local, hence the speed offered by Git is lightening fast compared to other VCS's.

Distributed
Every user has his own copy of the repository data stored locally allowing full functionality even on disconnection.

Branch Handling
Every collaborator's working directory is in itself a branch. Different branches can be merged with ease.

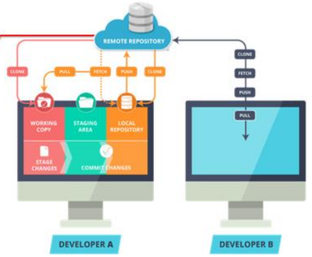
Integrity
Check-sum before storing ensures that you can't make any changes to anything without Git recording that change.

Robust
Nearly every task in Git is undo-able and it is really hard to loose any change or data in Git.

edureka!

The Git File Workflow

The Remote Repository is the server where all the collaborators upload changes made to the files



edureka!

Removing Files From Repository

- The **git rm** command deletes the file from git repository as well as users system

```
Syntax: git rm <filename>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demoS git rm Demo.class
```

- To remove the file from git repository but not from the system **--cached** option

```
Syntax: git rm --cached <filename>
```

```
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demoS git rm --cached Demo.class
rm 'Demo.class'
amrinderjeet@amrinderjeet-VirtualBox:~/Documents/demoS
```

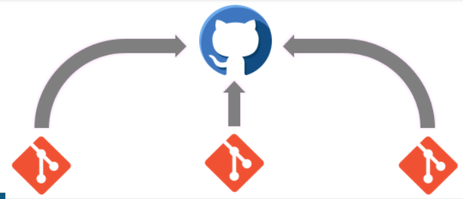
- An error shows up if you try to delete a staged file
- You can force remove a staged file by using **–f** flag

```
Syntax: git rm -f <filename>
```

edureka!

What Is A Remote Repository?

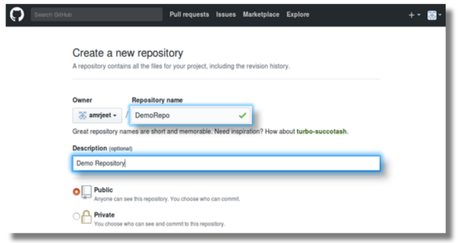
Mostly the users work on a local repository. But in order to collaborate with other people, we use a remote repository. A remote repository is place where the users upload and share their commits with other collaborators.



edureka!

Creating A Remote Repository: Adding Description

- Under **Repository name**, give a name to your repository
- Give some Description about your repository under **Description** section.



edureka!

edureka!

edureka!

Thank You

For more information please visit our website
www.edureka.co