upGrad

#LifeKoKaroLift

Version Control System - Git

upGrad

Course: Object-Oriented Analysis, Design & Programming

Lecture On: Version Control System - Git

Instructor: Sayan Nayak

Learning Objective

upGrad

The purpose of this course:

- This course will help you design and develop any application using the Object-Oriented Programming.

upGrad

Course Map

1. How to Maintain different versions of files using Git.

2. How to collaborate with others using GitHub.

1. How to read requirement doc for the new system.

2. How to draw class diagram using the requirement doc.

1. What are fundamental principles of OOP.

2. How to develop the application based on the class diagram.

upGrad

Today's Agenda

- **Maintaining Different Versions to Data Files using Git**
  - Introduction to VCS and Git.
  - Introduction to Vim editor and Terminal commands
  - Starting with Git - git init and git config commands
  - Making changes in Local Git Repo - git status, git add, git commit, git log
  - Branching in Git - git branch, git checkout, git merge
  - Syncing with Central Repo - git clone, git push, git pull, git fetch
  - .gitignore file

upGrad

Introduction to **Version Control System** and **Git**

Question: How will you maintain different versions of a file?

Suppose you are writing a big essay in a .txt file. But you know from your past experiences that you should maintain several versions of this file as some point of time in the future you may realize that what your writing currently makes no sense and you want to restart with that version of essay which was there 3 days ago.

Question: How will you maintain different versions of a file?

Answer: Create a new copy each time you want to save the current state as a separate version.

Essay.txt | Essay1.txt | Essay1_update.txt | Essay_13Oct.txt | Essay_13Oct_v2.txt

Issues with manual version control:

- Difficult to identify the changes between two files
- Not efficient with projects having thousands of files
- Difficult to collaborate with the team and other programmers
- Difficult to store metadata such as who made what changes, date and reason

As you can see, it is quite difficult to maintain file versions manually. This is where the ***Version Control System*** helps you. You may have experienced Version Control System if you have worked on Google Drive or Google Office Suite.

Introduction to VCS

upGrad

What is VCS?

The Version Control System (VCS) helps you manage the changes made to a specific directory or a project over the course of time.

VCS is the most important tool to learn for a software developer. No matter how big the codebase is, For example codebase of Google or Facebook, they all uses some sort of version control system.

Advantages of VCS

upGrad

Following are the advantages of VCS over manually controlled versions:

- Makes it easy to find out the changes between different versions of a file
- Highly efficient for large projects with thousands of files
- Makes it easy to collaborate with other programmers
- Makes it easy to store metadata
- Makes it easy to create backups and revert changes
- Makes it easy to share code on multiple systems/users

How Does the VCS Work?

upGrad

- **Control versions:**
  - You can have any directory under version control using a VCS tool. Such a directory is called local repository.
  - You can make changes to the local repository and save the new state (previous version + new changes) as a separate version.
  - In this way, you can maintain several versions of the files present in the local repository using VCS tools.

How Does the VCS Work?

upGrad

- **Collaboration:**
  - You can upload your local repository to a server, which includes all the versions of files or code. Such a repository is called a remote repository or central repository.
  - Collaborators can download this remote repository as their local repository.
  - They can make changes to their local repository and add new versions.
  - They can also update the remote repo by pushing these newly added versions to the remote repo.
  - Other collaborators can download and update their local repositories with the new changes.

Poll 1 (15 sec)

Which of the following is an important usage of the Version Control System? Multiple answers may be correct.

1. Testing the code
2. Checking the code functionality
3. Collaborating with the team
4. Maintaining versions

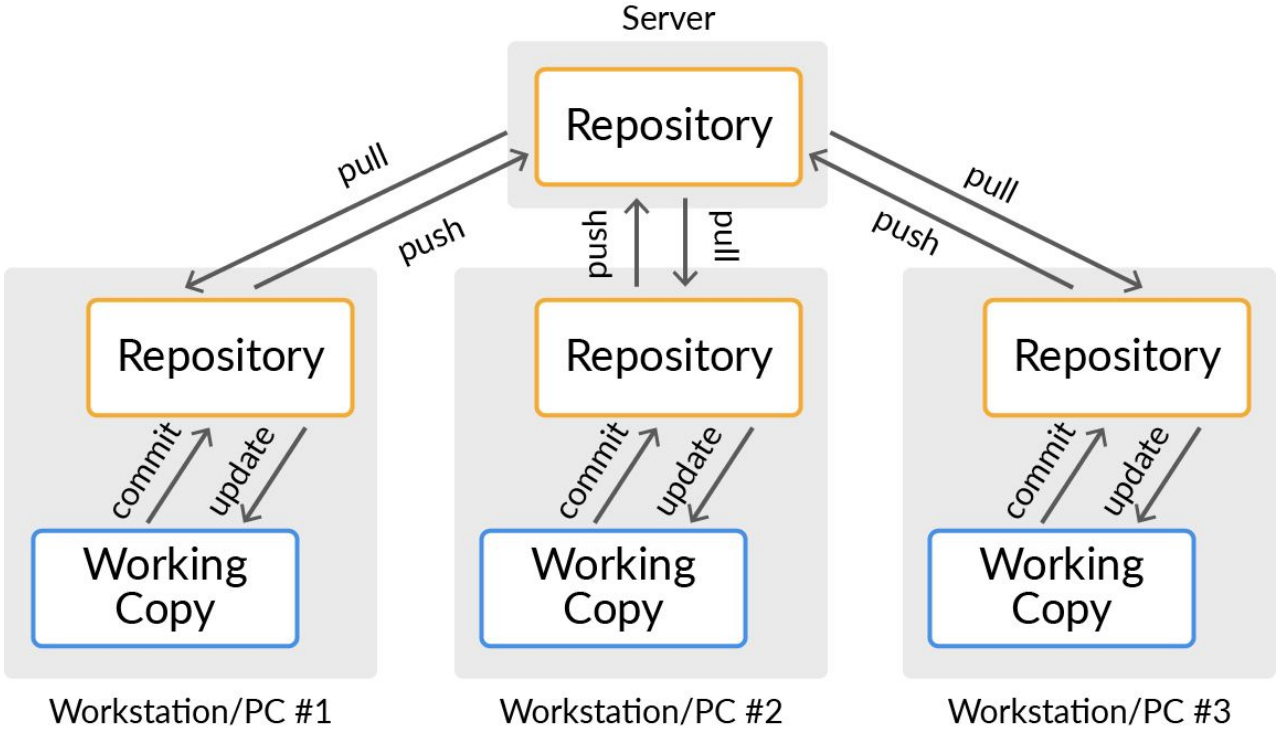
Poll 1 (15 sec)

Which of the following is an important usage of the Version Control System? Multiple answers may be correct.

1. Testing the code
2. Checking the code functionality
3. **Collaborating with the team**
4. **Maintaining versions**

How Does the VCS Work?

upGrad



How Does the VCS Work?

upGrad

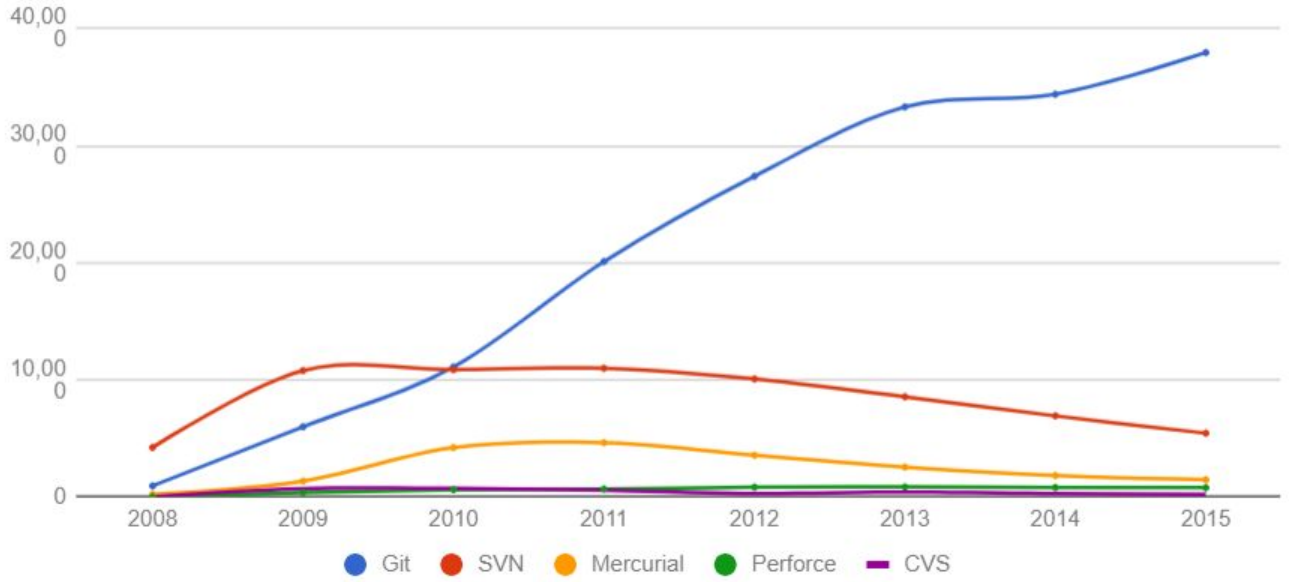
Some keywords used while discussing VCS:

- **Remote/Central/Master Repository** - Present on server/cloud service like GitHub, Bitbucket etc., where we can store the main codebase
- **Local Git Repository** - Present on local system where we store all the changes before pushing to the remote repository
- **Pushing** - Updating remote repository by pushing the newly added versions of the local repository to the remote repository
- **Pulling** - Updating the local repository by downloading the newer versions from the remote repository which were added by other collaborators

What is Git?

upGrad

Git is one of the most popular VCSs

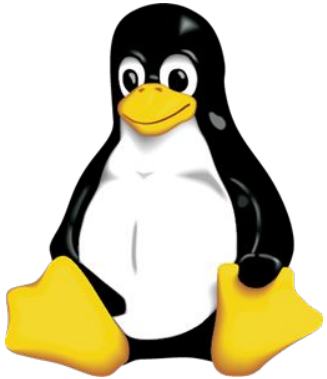


Ref - https://rhodecode.com/insights/version-control-systems-2016

Git Trivia

upGrad

Git was created by **Linus Torvalds**, who was also the creator of Linux.



Benefits of Git

upGrad

- Faster as most of the operations are performed locally
- Enables flexible version control
- Easy to revert changes and move to the previous state
- Useful for distributed teams
- Maintains integrity while using hash values

Git vs GitHub

upGrad

Are Git and GitHub the same?
Answer: No

Git: It is an open-source version control tool that is used by programmers for controlling versions of their codebase and collaborate with others.

GitHub: It is one of the most popular online Git service where we can create remote repositories based on Git.

Alternatives to GitHub

upGrad

1. BitBucket

2. GitLab

3. SourceForge

4. Most Cloud Providers

Poll 2 (15 sec)

GitHub is _____.

1. Another name of Git.
2. Another VCS tool similar to Git.
3. Online server to create Git based remote repositories.
4. The company which developed Git.

Poll 2 (15 sec)

GitHub is _____.

1. Another name of Git.
2. Another VCS tool similar to Git.
3. **Online server to create Git based remote repositories.**
4. The company which developed Git.

Starting with Git

upGrad

In this session, you will be working with the data files for an imaginary bank, known as U-Bank.

There will be two data files:

1. *Accounts.txt*: contains information about account number and name

Account Number	Name
1001	ishwar
1002	srishti

2. *Transactions.txt*: contains information about transactions between two account holders

From	To	Amount
1001	1002	500

Question: Why you need to maintain different versions of these data files, which are Accounts.txt and Transactions.txt?

Question: Can you open a folder, create new files, delete existing files or move files from one folder to another folder without using mouse? Or, even without opening the file explorer?

Question: Can you open a folder, create new files, delete existing files or move files from one folder to another folder without using mouse? Or, even without opening the file explorer?

Ans: You can perform all such tasks using Terminal (Mac, Linux) or Command Prompt (Windows) without the need of UI. Terminal or Command Prompt is a piece of software that allows you to accomplish all the computer related tasks by typing commands. For example, you can create a new folder using the following command.

mkdir <folder name>

You can open command prompt or Terminal by searching **cmd** in Windows and **terminal** in Mac or Linux.

Basic Commands for Terminal or Command Prompt

upGrad

Description	Windows	Mac & Linux
Move to different folder relative to current one	cd	cd
Clear the terminal screen	cls	clear or ctrl + L
Delete a file	del	rm
List down content of the current directory	dir	ls
Create a new folder	mkdir	mkdir
Move file from one folder to another folder	move	mv
Delete a folder	rmdir	rmdir

Introduction to Vim Editor

upGrad

In this session, we will be using *vim editor* to create and modify files.

- To create a new file and open or open an already existing file to modify -
  - *vim <FileName>*
- To start inserting or editing a file, you need to press ***“i”.***
- To save and exit a file, you need to press ***“Esc”*** and then use ***“:wq”***.

Press ‘i’ to go to INSERT mode.

Press ‘esc’ and then ‘:wq’ to save and exit

Quick Walkthrough of Git Installation Steps

upGrad

Windows

Mac

Linux

Initialize an Existing Directory to Git Repo

First, Open Git Bash or Terminal and go to the project directory using the **cd** command.

For example: *cd d:/U-Bank*

Second, use the command - ***git init***

This command will create a **.git** folder inside the current directory to convert it to a Git repository.

Most of the Git commands do not work outside of a Git repository. So, this is usually the first Git command you will run in a new project.

Command - git init

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d
$ cd d:/U-Bank

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank
$ git init
Initialized empty Git repository in D:/U-Bank/.git/

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Initializing local Git repo.

upGrad

Set username and email to Git

You can set username and email to your Git by executing the following commands:

git config --global user.name "<your name>"
git config --global user.email <your email>

You can check your username and email by executing the following commands:

git config user.name
git config user.email

Command - git config

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git config --global user.name "Ishwar Soni"

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git config --global user.email ishwar.soni@upgrad.com

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git config user.name
Ishwar Soni

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git config user.email
ishwar.soni@upgrad.com

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Adding username and email id to Git.

Poll 3 (15 sec)

Which of the following commands helps you to initialize an existing directory to Git repo?

1. git initialize
2. git init
3. initialize git
4. init git

Poll 3 (15 sec)

Which of the following commands helps you to initialize an existing directory to Git repo?

1. git initialize
2. **git init**
3. initialize git
4. init git

Coding Questions

upGrad

1. Create a new directory, named ‘U-Bank’.
2. Convert the newly created directory into the local Git repository.
3. Setup username and email to Git bash.

Ref - Code Reference

Hands-On Experience

upGrad

1. Create a new directory, named ‘My-Bank’.
2. Convert the newly created directory into the local Git repository.
3. Setup username and email to Git bash.

Making Changes in the Local Git Repo

Making Changes in the Local Git Repo

upGrad

To understand how Git works, you need to understand the three states and areas of a Git project:

3 States

1. Modified, Untracked
2. Staged
3. Committed

3 Areas

1. The working tree
2. The staging area
3. The Git directory

upGrad

The Git Workflow

1. When you add new files to the local Git repo, those files are called *untracked files* and reside in the *Working Tree*.

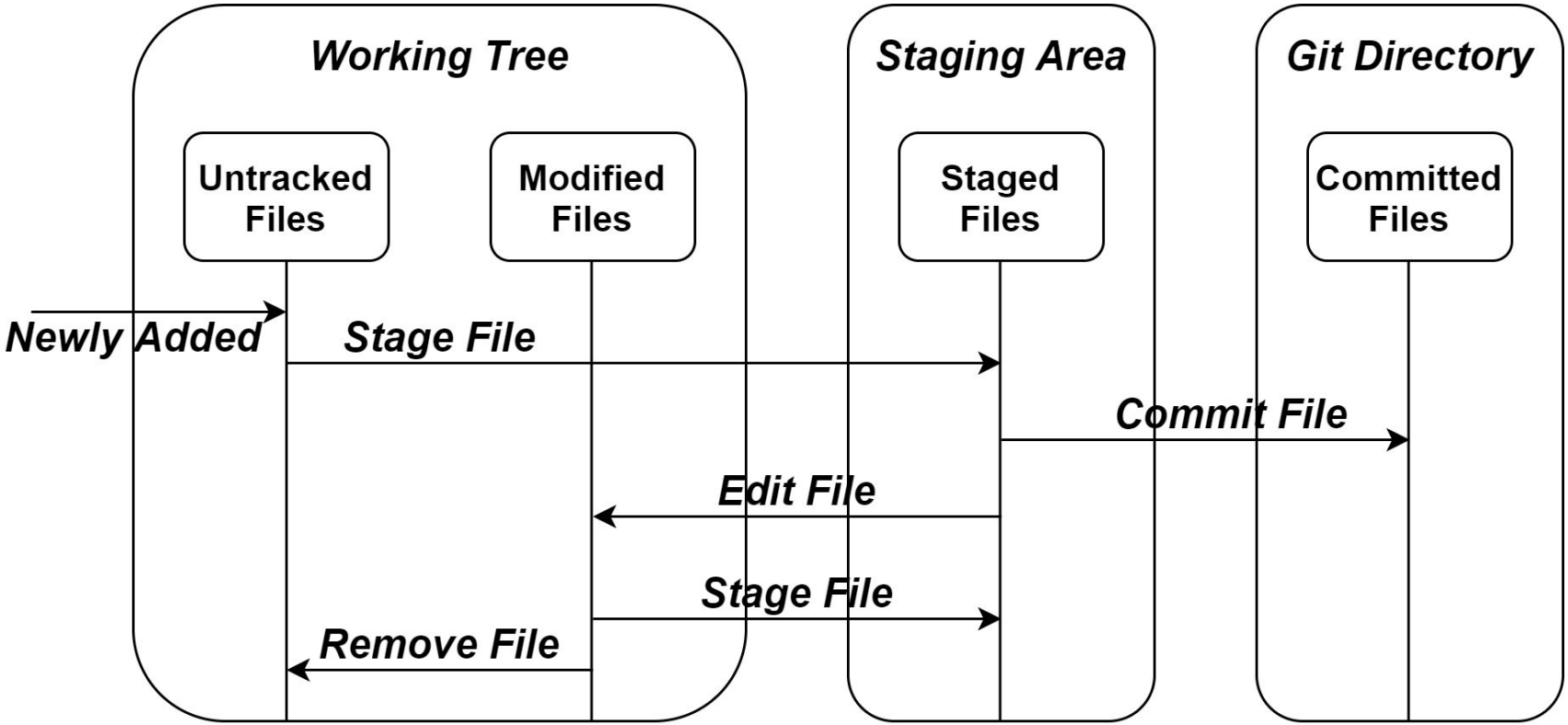
2. The files you want to save in the next version need to be moved from *Working Tree* to *Staging Area*. Such files are called *staged files*.

3. You can save the staged files in the next version by moving them to the Git directory. These files will be called *committed files*.

4. If you make changes to the staged files before moving them to the Git directory, they are moved back to the *working tree* and are called *modified files*.

Making Changes in the Local Git Repo

upGrad



States and Areas of a local Git repo

Command - git status

upGrad

- Used to check the status of the working tree and the staging area
- Shows the status of all the untracked and tracked files (whether staged or unstaged)

  Syntax: ***git status*** or ***git status -s***

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Transactions.txt
```

Command - git status

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
?? Accounts.txt
?? Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Additional Note: Symbols in Short Status

upGrad

In the short status, each file is appended with a symbol in the form of XY, where X shows the status of staging area and Y shows the status of working tree.

Different symbols

1. ‘ ’ - Unmodified
2. M - Modified
3. A - Added
4. ?? - Untracked

Additional Note: Symbols in Short Status

upGrad

Different combinations

1. A - Added to staging area and not modified (tracked + staged)
2. AM - Added to staging area but again modified (tracked + unstaged)

You can see the complete list of symbols and combinations here.

Poll 4 (15 sec)

The ***git status*** command indicates the status of:

1. Untracked files
2. Staged files
3. Modified files
4. All of the above

Poll 4 (15 sec)

The ***git status*** command indicates the status of:

1. Untracked files
2. Staged files
3. Modified files
4. **All of the above**

Command - git add

upGrad

- This command is used to move the unstaged files (both tracked and untracked) files from the *working tree* to the *staging area*.
- The files present in the *staging area* will be present in the next commit.

Syntax:

- ***git add <fileName1> <fileName2>*** - Used to stage the two files named *fileName1* and *fileName2*
- ***git add --all*** or
  git add .
  Used to move all the files to the staging area

Command - git add

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
?? Accounts.txt
?? Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git add Accounts.txt
warning: LF will be replaced by CRLF in Accounts.txt.
The file will have its original line endings in your working directory

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
A  Accounts.txt
?? Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Accounts.txt
```

Adding files to staging area using the ***git add <filename>*** command.

Command - git add

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
AM Accounts.txt
?? Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git add .
warning: LF will be replaced by CRLF in Accounts.txt.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in Transactions.txt.
The file will have its original line endings in your working directory

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
A  Accounts.txt
A  Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Adding files to staging area using the ***git add .*** command.

Poll 5 (15 sec)

The ***git add*** . command commits only the modified tracked files to the local repository. It will not commit new untracked files.

1. True
2. False

Poll 5 (15 sec)

The ***git add*** . command commits only the modified tracked files to the local repository. It will not commit new untracked files.

1. True
2. **False**

Command - git commit

upGrad

- It is used to commit/save all the changes to the local Git repo.
- It is used to move staged files from *staging area* to *git directory*.

***Syntax*:**

- ***git commit -m <message>*:** Commits all the staged files with the given commit message
- ***git commit -a -m <message>*:** Used to stage all the **tracked** files and commit them with the given message
  This command lets you skip the step of executing *git add* command.

Command - git commit

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
A  Accounts.txt
A  Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git commit -m 'added Accounts.txt file and Transactions.txt file'
[master (root-commit) 79fdf38] added Accounts.txt file and Transactions.txt file
 2 files changed, 3 insertions(+)
 create mode 100644 Accounts.txt
 create mode 100644 Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Transactions.txt
```

Committing the changes using the ***git commit -m 'message'*** command.

Command - git commit

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
 M Accounts.txt
 M Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git commit -a -m 'modified Accounts.txt and Transactions.txt files'
warning: LF will be replaced by CRLF in Accounts.txt.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in Transactions.txt.
The file will have its original line endings in your working directory
[master 7775f66] modified Accounts.txt and Transactions.txt files
 2 files changed, 2 insertions(+)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Committing the changes using the ***git commit -a -m 'message'*** command.

Command - git log

upGrad

- Helps to view information about previous commits.
- We can filter commits based on multiple properties such as number of days, author, recent number of commits done.

***Syntax*:**

- ***git log --all*** OR
  git log
  shows details of all the commits
- ***git log -n*:** shows details of last ‘n’ number of commits
- ***git log --committer=“<name>”*:** shows details of the commits made by “name”
- ***git log --oneline*:** shows details about each commit in one line

Command - git log

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log
commit 7775f6601d7b00fb61210ba6d2585374de95e803 (HEAD -> master)
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Mon Jul 6 19:17:36 2020 +0530

    modified Accounts.txt and Transactions.txt files

commit 79fdf38d7082742ed327e2084169f8820742e67b
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Mon Jul 6 18:47:52 2020 +0530

    added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log -1
commit 7775f6601d7b00fb61210ba6d2585374de95e803 (HEAD -> master)
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Mon Jul 6 19:17:36 2020 +0530

    modified Accounts.txt and Transactions.txt files
```

Checking logs or commit history using ***git log*** and ***git log -n*** command.

Command - git log

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log --committer="Ishwar Soni"
commit 7775f6601d7b00fb61210ba6d2585374de95e803 (HEAD -> master)
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Mon Jul 6 19:17:36 2020 +0530

    modified Accounts.txt and Transactions.txt files

commit 79fdf38d7082742ed327e2084169f8820742e67b
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Mon Jul 6 18:47:52 2020 +0530

    added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log --oneline
7775f66 (HEAD -> master) modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Checking logs or commit history using ***git log --committer=<name>*** and the ***git log --oneline*** command.

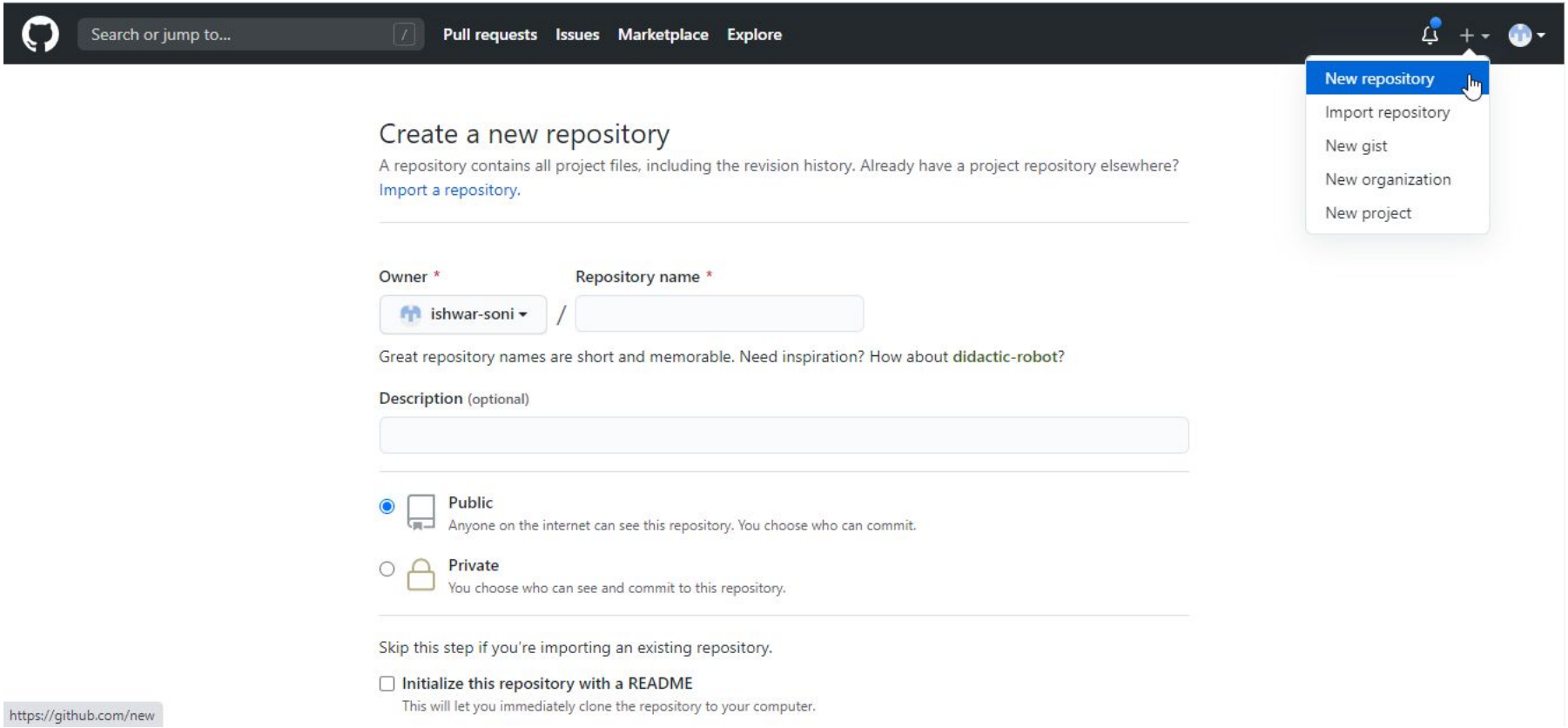
Command - git push

upGrad

- Before we can push the changes from the local Git repo to the remote Git repo, first we have to create a remote Git repository and establish connection between local Git repo and remote Git repo.

Create Remote Repository on GitHub

upGrad



Command - git remote add origin

upGrad

Link remote Git repo to the local Git repository

While creating a Git repository, you need to add a remote/cloud-based repository for the following purposes:

- To keep a copy of the source code available at any time or location
- To collaborate with team members and other programmers across multiple locations

You can add a remote to your Git repository using the following command:
git remote add origin <SSH or HTTPS URL>
git remote add origin https://github.com/ishwar-soni/U-Bank.git

This command adds a new remote location with the name ‘origin’ and connects to the given link of the remote repository. We can add several remote repositories to our local Git repository.

Command - git remote add origin

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d
$ cd d:/U-Bank

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank
$ git init
Initialized empty Git repository in D:/U-Bank/.git/

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git remote add origin https://github.com/ishwar-soni/U-Bank.git

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Adding remote to local Git repo.

Command - git push

upGrad

- Used to update all the changes from local Git repository to the remote server/repository, which can later be fetched by other collaborators
- Used to push/save changes (additions, modifications, deletions) in a remote repository

***Syntax*:**

- ***git push origin master*:** Used to push the code to the remote repo 'origin' and the branch 'master'
- ***git push -u origin master*:** Same as previous command, except add upstream (-u) so all subsequent push can be done using just the ***git push*** command
- ***git push*:** Used to push the code to the remote repo, the details of which are being set using the previous command (second bullet point)

Command - git push

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git push origin master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 699 bytes | 87.00 KiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ishwar-soni/U-Bank.git
 * [new branch]      master -> master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
 M Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git commit -a -m 'added one more transaction between 1001 and 1002'
warning: LF will be replaced by CRLF in Transactions.txt.
The file will have its original line endings in your working directory
[master d4c1fc2] added one more transaction between 1001 and 1002
 1 file changed, 1 insertion(+)
```

Pushing changes using ***git push origin master.***

Command - git push

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 341 bytes | 341.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ishwar-soni/U-Bank.git
   7775f66..d4c1fc2  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status -s
 M Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git commit -a -m '1002 paid back all the money'
warning: LF will be replaced by CRLF in Transactions.txt.
The file will have its original line endings in your working directory
[master 1afa471] 1002 paid back all the money
 1 file changed, 1 insertion(+)
```

Pushing changes using ***git push -u origin master.***

Command - git push

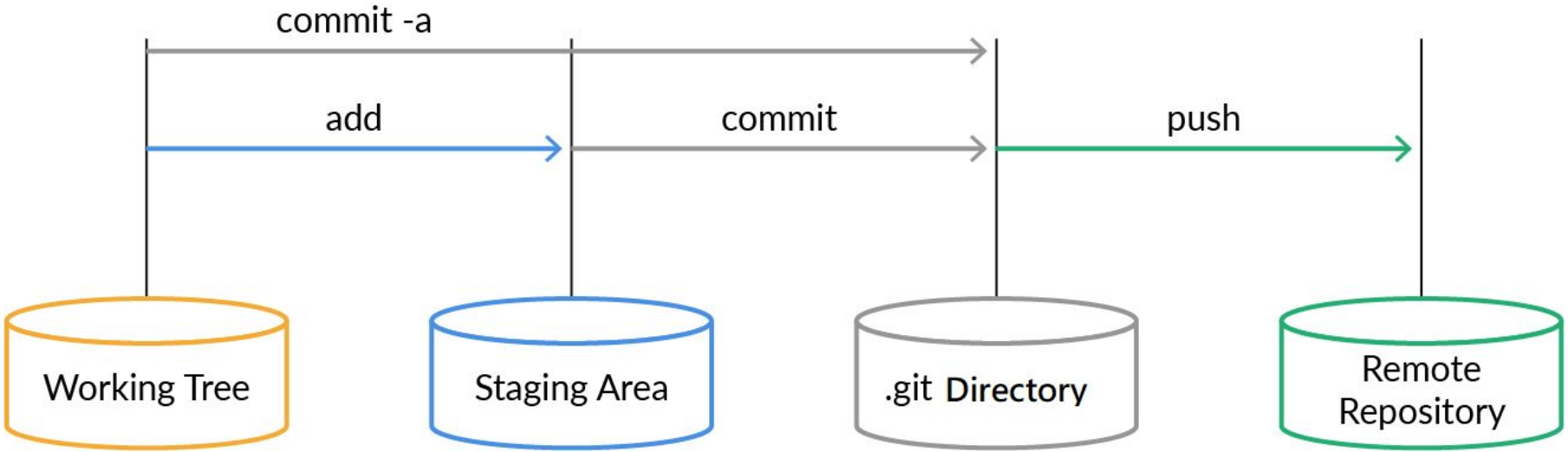
upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes | 168.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ishwar-soni/U-Bank.git
   d4c1fc2..1afa471  master -> master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Pushing changes using ***git push.***

Flow

upGrad


commit -a
add
commit
push
Working Tree
Staging Area
.git Directory
Remote Repository

Poll 6 (15 sec)

Which of the following commands moves all the latest commits to the remote repository?

1. git add .
2. git push
3. git commit
4. None of the above

Poll 6 (15 sec)

Which of the following commands moves all the latest commits to the remote repository?

1. git add .
2. **git push**
3. git commit
4. None of the above

Coding Questions

upGrad

1. Add and modify the files in the working tree.
2. Use the *git add* command to stage the files.
3. Use the *git commit* command to commit the changes to the local Git repository.
4. Create a new remote GitHub repo named ‘U-Bank’.
5. Add the remote GitHub repo to the local repo.
6. Use the *git push* command to push the changes to the remote repo.
7. Use the *git log* command to check the commit history.
8. Use the *git status* command to check the current status of local Git repo.

Ref - Code Reference

Hands-On Exercise

upGrad

1. Add new files (Branches.txt, Employees.txt) and add some data into these files.
2. Use the *git add* command to stage all the files.
3. Make changes to Branches.txt file and add just this file to staging area.
4. Use the *git commit* command to commit the changes to the local repo.
5. Create a new remote GitHub repo named ‘U-Bank’.
6. Add the remote GitHub repo to the local repo.
7. Use the *git push* command to push the changes to the remote repo.
8. Use the *git log* command to check the commit history.

Note: Use *git status* command after every change in repo, to check and verify with the changes and current status.

upGrad

Branching in Git

Why do We need Branching in Git?

upGrad

Scenario: You have a running live website, and you want to add some new features to it.

Approach 1: Stop running the website and then make changes to the code in the master codebase.

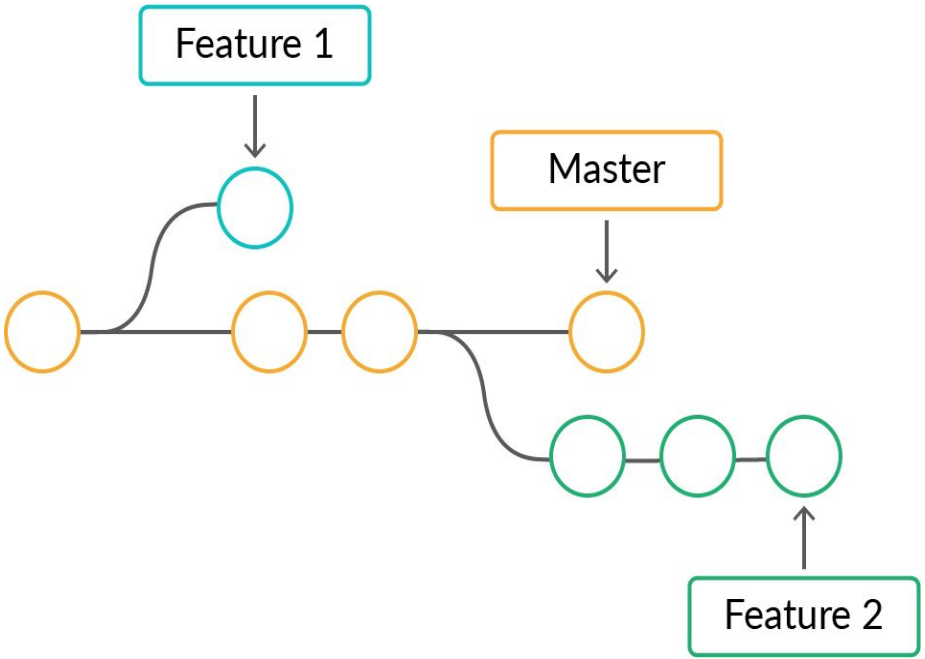
Approach 2: Make changes to the code in the master codebase of the running website.

Is there a way in which the master codebase of the website remains untouched until you complete the updation in the code of the new feature?

Branching - Introduction and Need

upGrad

- Branching is a Git feature that helps you create new branches.
- You need to create new branches to add or fix changes in the source code, without affecting the main codebase.
- When you finish making the changes, you can merge those branches with the master branch and update the main codebase.



Command - git branch

upGrad

- Used to create new branches
- Used to delete or list down the existing branches

***Syntax*:**

- ***git branch*:** Used to list down all the existing branches and highlight the current branch
- ***git branch <branchName>*:** Used to create a new branch with the given branch name
- ***git branch -d <branchName>*:** Used to delete the branch with the given branch name

Need for the git branch Command

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch
* master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch dev

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch
  dev
* master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch -d dev
Deleted branch dev (was 1afa471).

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch
* master
```

Creating and deleting branches using the ***git branch*** command.

Command - git checkout

upGrad

- Used to navigate between branches
- Used to create and simultaneously checkout the new branch
- Used to move to a previous version in the same branch

Syntax:

- ***git checkout <branchName>*:** Used to checkout the branch with the given branch name
- ***git checkout -b <branchName>*:** Used to create a new branch with the given branch name and simultaneously checkout to the newly created branch
- ***git checkout <commit id>*:** Used to checkout the code to the commit ID provided

Command - git checkout

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch
* master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git branch feature1

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git checkout feature1
Switched to branch 'feature1'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git branch
* feature1
  master
```

Checking out branches using ***git checkout <branchName>.***

Command - git checkout

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git checkout -b feature2
Switched to a new branch 'feature2'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git branch
  feature1
* feature2
  master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git log --oneline
1afa471 (HEAD -> feature2, origin/master, master, feature1) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file
```

Creating and checking out branches using ***git checkout -b <branchName>.***

Command - git checkout

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git checkout d4c1fc2
Note: switching to 'd4c1fc2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at d4c1fc2 added one more transaction between 1001 and 1002

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank ((d4c1fc2...))
$ git checkout feature2
Previous HEAD position was d4c1fc2 added one more transaction between 1001 and 1002
Switched to branch 'feature2'
```

Checking out to a previous commit using ***git checkout <commit hash>.***

Command - git merge

upGrad

- Used to merge any two branches

Syntax:

- ***git merge <branchName>:*** Merges the *branchName* with the current branch

Command - git merge

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank ((d4c1fc2...))
$ git checkout feature2
Previous HEAD position was d4c1fc2 added one more transaction between 1001 and 1002
Switched to branch 'feature2'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git log --oneline
1afa471 (HEAD -> feature2, origin/master, master, feature1) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ vim Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git status -s
 M Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git commit -a -m 'added one more account'
warning: LF will be replaced by CRLF in Accounts.txt.
The file will have its original line endings in your working directory
[feature2 aa30e39] added one more account
 1 file changed, 1 insertion(+)
```

Making changes in the ***feature2*** branch.

Command - git merge

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git log --oneline
aa30e39 (HEAD -> feature2) added one more account
lafa471 (origin/master, master, feature1) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log --oneline
lafa471 (HEAD -> master, origin/master, feature1) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file
```

Checking out master branch.

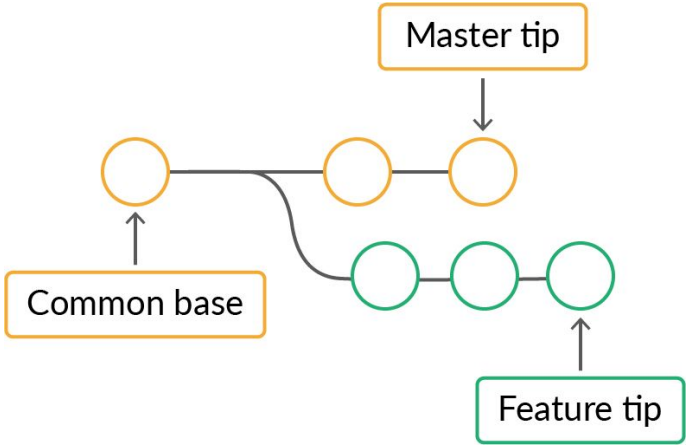
Command - git merge

upGrad

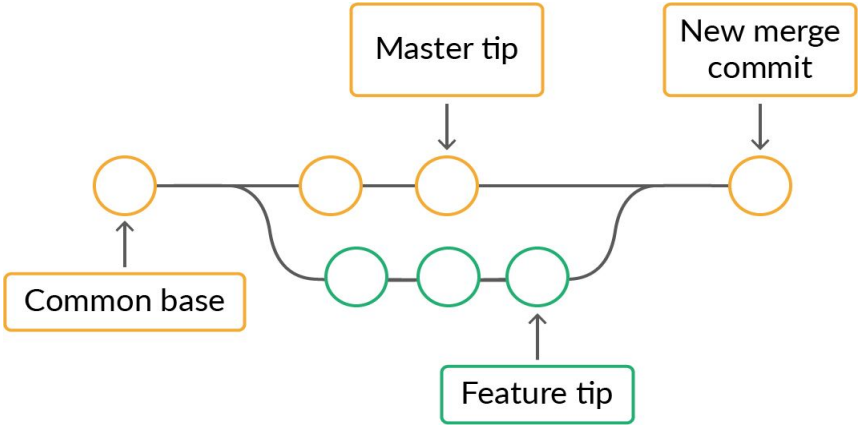
```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git merge feature2
Updating 1afa471..aa30e39
Fast-forward
 Accounts.txt | 1 +
 1 file changed, 1 insertion(+)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log --oneline
aa30e39 (HEAD -> master, feature2) added one more account
1afa471 (origin/master, feature1) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Merging ***feature2*** branch with the ***master*** branch using ***git merge <branchName>.***

Command - git merge



Before merging feature branch to master branch



After merging feature branch to master branch

What are merge conflicts in Git?

- A merge conflict occurs when you are merging two branches and Git is unable to automatically resolve the differences in code/files between the two branches.

When does a merge conflict occur?

- A merge conflict occurs when two different changes are made to the same line in the same file.
- For example, if one user deletes a line and another user modifies the same line on a different branch, then a merge conflict occurs.
- And later, when two branches having different changes on the same line are to be merged, merge conflict occurs.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git checkout feature1
Switched to branch 'feature1'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git log --oneline
1afa471 (HEAD -> feature1, origin/master) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ vim Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git commit -a -m 'added one more account with details <1003, akash>'
[feature1 0de208e] added one more account with details <1003, akash>
 1 file changed, 1 insertion(+)
```

Making changes in the ***Accounts.txt*** file in the ***feature1*** branch causes a merge conflict.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git log --oneline
0de208e (HEAD -> feature1) added one more account with details <1003, akash>
1afa471 (origin/master) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git merge feature1
Auto-merging Accounts.txt
CONFLICT (content): Merge conflict in Accounts.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Merging the ***feature1*** branch into the ***master*** branch causes a merge conflict.

How to check if there is a merge conflict?

- Check if two or more people have edited the same line in a file.
- Use the ***git status*** command to check if you have ‘unmerged paths’. It will help to find the merge conflict files while merging.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   Accounts.txt

no changes added to commit (use "git add" and/or "git commit -a")

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git status -s
UU Accounts.txt
```

Checking status of unmerged files or conflicted files using ***git status*** and ***git status -s***
Note: UU = Unmerged, modified by both the branches.

How to resolve merge conflicts?

- Git displays merge conflict areas in the file between the conflict markers as shown below.

  <<<<<<< HEAD

  <Changes in the current branch>

  =======

  <Changes in the incoming branch>

  >>>>>>> *branchName*

- We can resolve the conflict manually by editing the piece between the markers.
- We should also remove the markers added by Git.
- We can then commit the changes and, thus, fix the merge conflict.

Merge Conflicts

upGrad

```
Account Number          Name
1001                    ishwar
1002                    srishti
<<<<<<< HEAD
1003                    sreejit
=======
1003                    akash
>>>>>>> feature1
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
Accounts.txt [dos] (22:17 06/07/2020)                                   4,12 All
"Accounts.txt" [dos] 8L, 124C
```

Accounts.txt file before resolving the merge conflict.

Merge Conflicts

upGrad

```
Account Number          Name
1001                    ishwar
1002                    srishti
1003                    sreejit, akash
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
Accounts.txt[+] [dos] (22:17 06/07/2020)                          4,17-34 All
-- INSERT --
```

Accounts.txt file after resolving the merge conflict.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ vim Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git status -s
UU Accounts.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git add .

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git commit -m 'resolved merge conflict in Accounts.txt file'
[master 596a15e] resolved merge conflict in Accounts.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log --oneline
596a15e (HEAD -> master) resolved merge conflict in Accounts.txt file
0de208e (feature1) added one more account with details <1003, akash>
aa30e39 (feature2) added one more account
1afa471 (origin/master) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file
```

Committing the ***Accounts.txt*** file after resolving the merge conflict.

Git commands are used to resolve merge conflicts.

You can resolve merge conflicts by editing the file and removing conflicts, but some Git commands can help you in a better way. These include the following:

- ***git status*:** Gives an idea of merge conflicts that have occurred.
- ***git log --merge*:** Produces a log with all the commits that have caused conflicts during the merging of branches.
- ***git diff*:** Finds the differences between the states of a repository/files This can help you find and prevent merge conflicts.

Use the following Git commands to fix merge conflicts:

***git checkout*:** Used to undo changes made to **unstaged** files (Recall that this is also used for switching to branches or commits.)

***git reset*:** Used to undo changes made to the working directory and the staging area to the last stable commit. This basically works for the set of files in the aforementioned areas unlike *git checkout*, which can even work on specified files.

***git merge --abort*:** Used to exit from the merge process and return the branch to the state prior to the start of the merging process

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git checkout feature1
Switched to branch 'feature1'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ vim Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git commit -a -m '1001 trasnferred 2000rs to 1003'
[feature1 648d463] 1001 trasnferred 2000rs to 1003
 1 file changed, 1 insertion(+)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git checkout feature2
Switched to branch 'feature2'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ vim Transactions.txt

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git commit -a -m '1001 trasnferred 3000rs to 1003'
[feature2 837ad33] 1001 trasnferred 3000rs to 1003
 1 file changed, 1 insertion(+)
```

Making changes in the same lines in the ***Transactions.txt*** file through the ***feature1*** and ***feature2*** branches.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git merge feature1
Merge made by the 'recursive' strategy.
 Transactions.txt | 1 +
 1 file changed, 1 insertion(+)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git log --oneline
d6f61d5 (HEAD -> master) Merge branch 'feature1'
648d463 (feature1) 1001 trasnferred 2000rs to 1003
596a15e resolved merge conflict in Accounts.txt file
0de208e added one more account with details <1003, akash>
aa30e39 added one more account
1afa471 (origin/master) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file
```

Merging the ***feature1*** branch with the ***master*** branch.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git merge feature2
Auto-merging Transactions.txt
CONFLICT (content): Merge conflict in Transactions.txt
Automatic merge failed; fix conflicts and then commit the result.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   Transactions.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Merging the ***feature2*** branch with the ***master*** branch, which will cause merge conflict.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git log --merge
commit 837ad332ba76417f7c65f34325d769349a6c4382 (feature2)
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Tue Jul 7 08:12:14 2020 +0530

    1001 trasnferred 3000rs to 1003

commit 648d4635c379888945af59ff94e4577cb12c9579 (feature1)
Author: Ishwar Soni <ishwar.soni@upgrad.com>
Date:   Tue Jul 7 08:10:55 2020 +0530

    1001 trasnferred 2000rs to 1003
```

Checking history of commits which have caused the merge conflict using ***git log --merge***.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git diff
diff --cc Transactions.txt
index fc400a9,fc28019..0000000
--- a/Transactions.txt
+++ b/Transactions.txt
@@@ -2,4 -2,4 +2,8 @@@ From     To      Amoun
  1001  1002    500
  1001  1002    1000
  1002  1001    1500
++<<<<<<< HEAD
 +1001  1003    2000
++=======
+ 1001  1003    3000
++>>>>>>> feature2
```

Checking changes made to conflicting files by the git merge command using the ***git diff*** command.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git diff master feature2
diff --git a/Accounts.txt b/Accounts.txt
index c1309ce..7a68169 100644
--- a/Accounts.txt
+++ b/Accounts.txt
@@ -1,4 +1,4 @@
 Account Number         Name
 1001                   ishwar
 1002                   srishti
-1003                   sreejit, akash
+1003                   sreejit
diff --git a/Transactions.txt b/Transactions.txt
index fc400a9..fc28019 100644
--- a/Transactions.txt
+++ b/Transactions.txt
@@ -2,4 +2,4 @@ From    To      Amount
 1001   1002    500
 1001   1002    1000
 1002   1001    1500
-1001   1003    2000
+1001   1003    3000
```

Checking the difference between files present in the different branches using the ***git diff <branch1> <branch2>*** command.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master|MERGING)
$ git merge --abort

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git checkout feature2
Switched to branch 'feature2'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git reset --hard HEAD~1
HEAD is now at aa30e39 added one more account

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git log --oneline
aa30e39 (HEAD -> feature2) added one more account
1afa471 (origin/master) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file
```

Aborting the merging process using the ***git merge --abort*** command and removing the commits from the ***feature2*** branch using the ***git reset --hard HEAD~1*** command.

Merge Conflicts

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git reset --hard HEAD~1
HEAD is now at 1afa471 1002 paid back all the money

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git log --oneline
1afa471 (HEAD -> feature2, origin/master) 1002 paid back all the money
d4c1fc2 added one more transaction between 1001 and 1002
7775f66 modified Accounts.txt and Transactions.txt files
79fdf38 added Accounts.txt file and Transactions.txt file

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git merge feature2
Already up to date.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Merging the ***feature2*** branch to the ***master*** branch after removing the conflicting commits from the ***feature2*** branch.

Poll 7 (15 sec)

Which of the following commands can help you check for solving merge conflicts?

1. git push
2. git commit
3. git status
4. None of the above

Poll 7 (15 sec)

Which of the following commands can help you check for solving merge conflicts?

1. git push
2. git commit
3. **git status**
4. None of the above

Poll 8 (15 sec)

Which of the following commands can reset files to a stable state when merge conflicts happen?

1. git merge --abort
2. git checkout
3. git reset
4. All of the above

Poll 8 (15 sec)

Which of the following commands can reset files to a stable state when merge conflicts happen?

1. git merge --abort
2. git checkout
3. git reset
4. **All of the above**

Coding Question

upGrad

1. We will create new branches for the given repository.

2. Then we will merge the two branches and check the final outcome.
   Ref - Code Reference

3. We will try to create merge conflicts.

4. After this, we will fix the conflicts and merge the branches properly.
   Ref - Code Reference

Merge two branches:

1. Create two branches: *feature1* and *feature2*.
2. Checkout the *feature1* branch and make changes to the Branches.txt file.
3. Commit the changes in the *feature1* branch and checkout the *master* branch.
4. Merge the *feature1* branch to the *master* branch.

Create Merge Conflict and resolve it:

1. Checkout the *feature2* branch and make changes to the Branches.txt file.
   a. Note: Make changes in the same lines as made for *feature1*
2. Commit the changes in the *feature2* branch and checkout the *master* branch.
3. Merge the *feature2* branch to the *master* branch; you should get merge conflict.
4. Use *git status* command and *vim* tool to resolve the conflict.
5. Commit the changes.

Hands-on Exercise

upGrad

Create Merge Conflict and resolve it using git commands:

1. Checkout the *feature1* branch, make changes to the Employees.txt file and commit the changes.
2. Checkout the *feature2* branch, make changes to the Employees.txt file and commit the changes.
3. Checkout the *master* branch and merge the *feature1* branch to the *master* branch.
4. Now try merging the *feature2* branch to the *master* branch; you should get merge conflict.
5. Use *git status* to check the conflicting files.
6. Use *git log --merge* to check the conflicting commits.

upGrad

Create Merge Conflict and resolve it using git commands:

7. Use *git diff* and *git diff master feature2* to check the conflicting changes.
8. Use *git merge --abort* to abort the merging process
9. Checkout the *feature2* branch.
10. Use *git reset --hard HEAD~1* command to remove the commits that are causing conflicts.
11. Checkout the *master* branch.
12. Now try merging the *feature2* branch to the *master* branch; you should not get merge conflict.

Syncing with the Central Repository

Command - git clone

upGrad

Used to copy a remote repository into the local system along with the complete version history.

Syntax:

- ***git clone <url>:*** Used to clone a repo from the provided remote repo, it will create a directory with the name of the remote repo.
- ***git clone <url> <directory_Name>:*** Used to clone a repo from the provided remote repo and save it in a new directory with the name ***directory_Name***

Command - git clone

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/projects
$ git clone https://github.com/upgrad-edu/phone-directory.git
Cloning into 'phone-directory'...
remote: Enumerating objects: 99, done.
remote: Total 99 (delta 0), reused 0 (delta 0), pack-reused 99
Receiving objects: 100% (99/99), 141.65 KiB | 656.00 KiB/s, done.
Resolving deltas: 100% (54/54), done.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/projects
$ git clone https://github.com/upgrad-edu/phone-directory.git my_phone_dir
Cloning into 'my_phone_dir'...
remote: Enumerating objects: 99, done.
remote: Total 99 (delta 0), reused 0 (delta 0), pack-reused 99
Receiving objects: 100% (99/99), 141.65 KiB | 306.00 KiB/s, done.
Resolving deltas: 100% (54/54), done.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/projects
$ dir
my_phone_dir  phone-directory

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/projects
$
```

Cloning a remote Git repo as local Git repo using the ***git clone*** command.

Command - git pull

upGrad

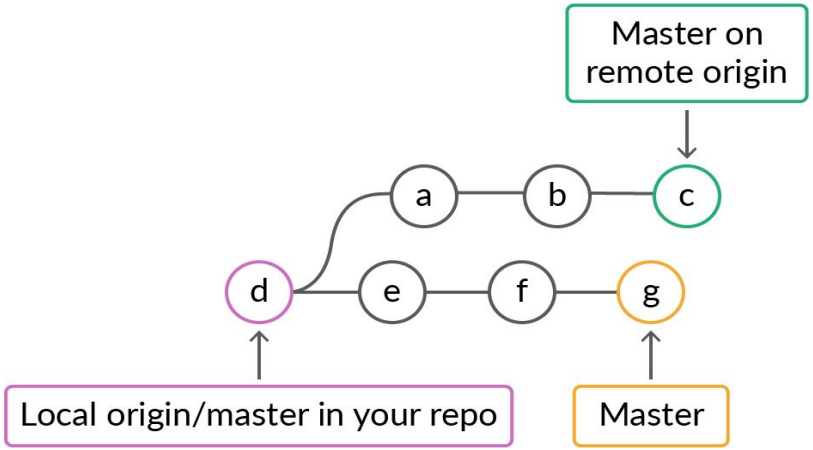
Used to **update** all the changes from a remote repository to the local repository

***Syntax*:**

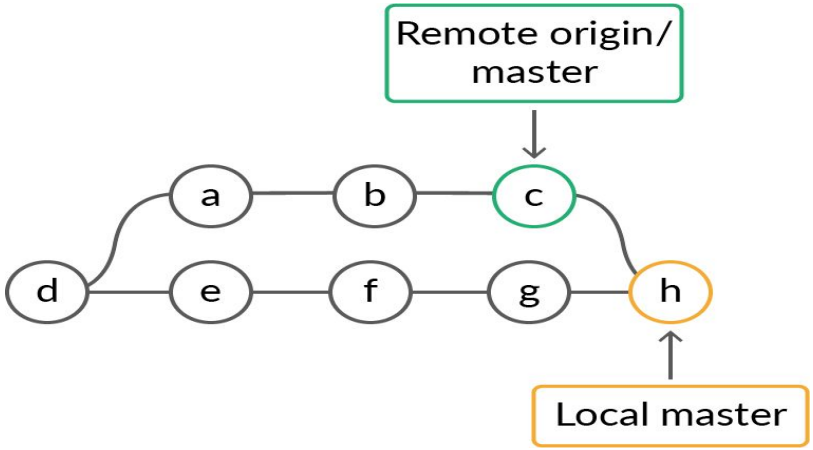
- ***git pull <remote> <branch>:*** Pulls updates from the given remote repo and branch and merges them with the current branch.
- ***git pull <remote>:*** Pulls updates from the given remote repo and master branch and merges them with the current branch.
- ***git pull:*** Pulls updates from the given origin and master branch and merges them with the current branch.

Command - git pull

upGrad



Before git pull



After git pull

Command - git pull

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 1.46 KiB | 498.00 KiB/s, done.
Total 14 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/ishwar-soni/U-Bank.git
   1afa471..d6f61d5  master -> master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git checkout feature1
Switched to branch 'feature1'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git push -u origin feature1
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature1' on GitHub by visiting:
remote:      https://github.com/ishwar-soni/U-Bank/pull/new/feature1
remote:
To https://github.com/ishwar-soni/U-Bank.git
 * [new branch]      feature1 -> feature1
Branch 'feature1' set up to track remote branch 'feature1' from 'origin'.
```

Pushing the changes from the ***master*** branch and ***feature1*** branch.

Command - git pull

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git checkout feature2
Switched to branch 'feature2'

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git push -u origin feature2
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature2' on GitHub by visiting:
remote:      https://github.com/ishwar-soni/U-Bank/pull/new/feature2
remote:
To https://github.com/ishwar-soni/U-Bank.git
 * [new branch]      feature2 -> feature2
Branch 'feature2' set up to track remote branch 'feature2' from 'origin'.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git pull origin feature1
From https://github.com/ishwar-soni/U-Bank
 * branch            feature1   -> FETCH_HEAD
Updating lafa471..648d463
Fast-forward
 Accounts.txt     | 1 +
 Transactions.txt | 1 +
 2 files changed, 2 insertions(+)
```

Pulling changes from the ***feature1*** branch into the ***feature2*** branch.

Command - git pull

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git pull origin feature2
From https://github.com/ishwar-soni/U-Bank
 * branch            feature2    -> FETCH_HEAD
Already up to date.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git pull
Already up to date.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature2)
$ git checkout feature1
Switched to branch 'feature1'
Your branch is up to date with 'origin/feature1'.
```

Pulling changes from the ***feature1*** branch and ***master*** branch into the ***feature2*** branch.

Command - git pull

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git pull origin feature1
From https://github.com/ishwar-soni/U-Bank
 * branch            feature1   -> FETCH_HEAD
Already up to date.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git pull origin feature1
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 709 bytes | 3.00 KiB/s, done.
From https://github.com/ishwar-soni/U-Bank
 * branch            feature1   -> FETCH_HEAD
   648d463..b77147a  feature1   -> origin/feature1
Updating 648d463..b77147a
Fast-forward
 Transactions.txt | 1 +
 1 file changed, 1 insertion(+)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$
```

Pulling changes from the ***feature1*** branch before and after making changes from the GitHub repo.

Command - git fetch

upGrad

- Used to **download** all the changes from a remote repository to the local repository
- Also, used to check all the updates in the remote repository

Syntax:

- ***git fetch <remote>:*** Fetches all the data from the given remote repo link
- ***git fetch <remote> <branch>:*** Fetches all the data from the given remote repo link and the specified branch
- ***git fetch --all:*** Fetches all the data from all the remote locations added in the local repository

Command - git fetch

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git fetch origin feature1
From https://github.com/ishwar-soni/U-Bank
 * branch            feature1   -> FETCH_HEAD

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git fetch origin feature2
From https://github.com/ishwar-soni/U-Bank
 * branch            feature2   -> FETCH_HEAD

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git fetch origin

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git fetch --all
Fetching origin
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git fetch --all
Fetching origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 710 bytes | 1024 bytes/s, done.
From https://github.com/ishwar-soni/U-Bank
   b77147a..f87fb8a  feature1   -> origin/feature1
```

Fetching changes from the ***feature1, feature2*** and ***master*** branches into the ***feature1*** branch.

Command - git fetch

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git fetch --all
Fetching origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 710 bytes | 1024 bytes/s, done.
From https://github.com/ishwar-soni/U-Bank
   b77147a..f87fb8a  feature1   -> origin/feature1

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git merge
Updating b77147a..f87fb8a
Fast-forward
 Transactions.txt | 1 +
 1 file changed, 1 insertion(+)
```

Merging the changes fetched using the ***git fetch*** command. This is equivalent to the ***git pull*** command.

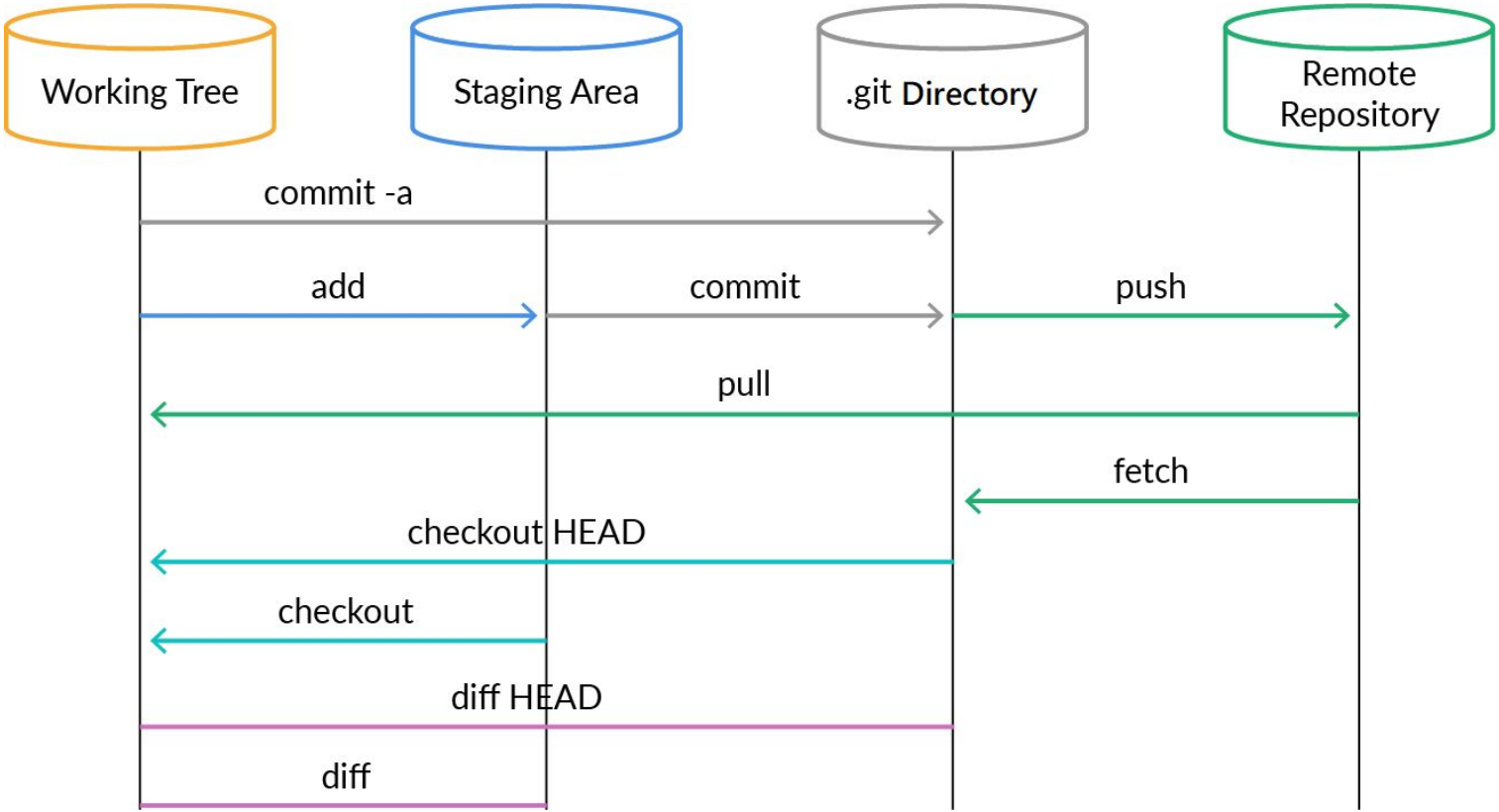
git fetch VS. git pull

upGrad

- The ***git pull*** command is a combination of two other commands, ***git fetch*** followed by ***git merge***.
- For example, suppose you are currently in the *feature1* branch and execute ***git pull origin feature2*** command.
- In the first stage of the operation, git will execute ***git fetch origin feature2*** command and download all the changes from the remote repository to the local repository corresponding to the *feature2* branch.
- Once the content is downloaded, git will execute ***git merge feature2*** command and merge the *feature2* branch with the current branch, which is *feature1*, if there is no merge conflict.

Final Flow

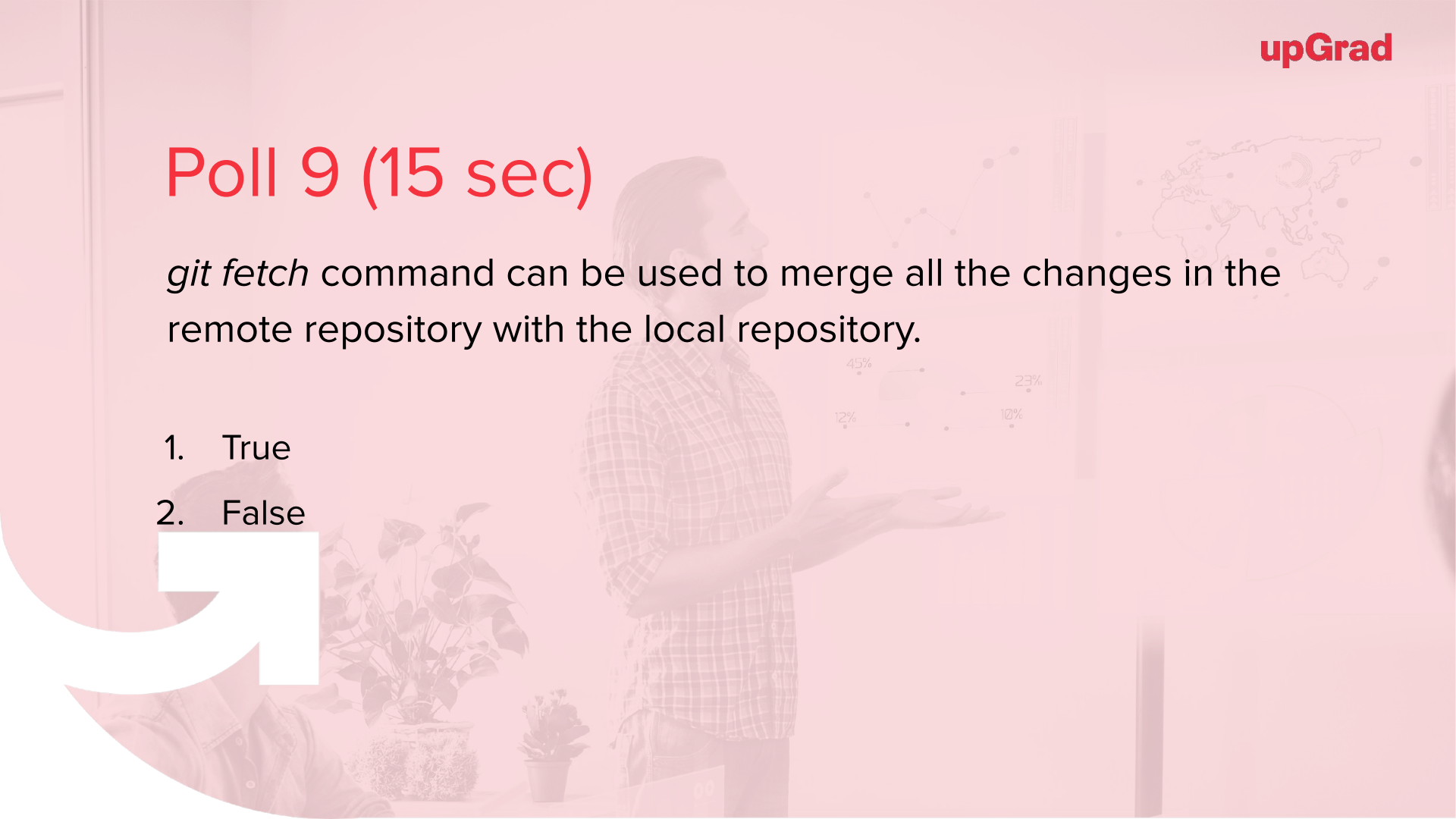
upGrad



Poll 9 (15 sec)

git fetch command can be used to merge all the changes in the remote repository with the local repository.

1. True
2. False



Poll 9 (15 sec)

git fetch command can be used to merge all the changes in the remote repository with the local repository.

1. True
2. **False**

Coding Question

upGrad

1. Use *git clone* command to clone remote repo as local repo.

Ref - Code Reference

2. Checkout the *feature2* branch and **pull** changes from the *feature1* branch, *feature2* branch and *master* branch.
3. Checkout the *feature1* branch and **fetch** changes from the *feature1* branch, *feature2* branch and *master* branch.
4. Merge the fetched changes to the *feature1* branch.

Ref - Code Reference

Hands-On Exercise

upGrad

1. Use *git clone* command to clone remote repo as local repo.
   a. https://github.com/upgrad-edu/phone-directory.git
2. Checkout the *feature2* branch and **pull** changes from the *feature1* branch, *feature2* branch and *master* branch.
   a. While pulling, make changes to Employees.txt file using the GitHub repo.
3. Checkout the *feature1* branch and **fetch** changes from the *feature1* branch, *feature2* branch and *master* branch.
   a. While fetching, make changes to Employees.txt file using the GitHub repo.
4. Merge the fetched changes to the *feature1* branch.

.gitignore file

.gitignore file

upGrad

- Used to ignore **untracked** files.
- For example, suppose you do not want to include the log files with ***.log*** extension.
- You can achieve this using the .gitignore file.
- In the .gitignote file, you have to enter a line with the following pattern.
  - *.log
- Now, all the files with the .log extension will be ignored by the Git.
- This file should be present at the root level (the same folder in which .git folder is present).
- You can see the complete list of rules for writing patterns in the .gitignore file here.

.gitignore file

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (feature1)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim tran_logs.log

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

nothing added to commit but untracked files present (use "git add" to track)
```

Files with .log extension are getting tracked by the Git.

.gitignore file

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim .gitignore

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

nothing added to commit but untracked files present (use "git add" to track)

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git add .gitignore
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git commit -m 'added .gitignore file'
[master d484499] added .gitignore file
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

Adding .gitignore file.

.gitignore file

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 330 bytes | 110.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ishwar-soni/U-Bank.git
   d6f61d5..d484499  master -> master

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Committing .gitignore file.

.gitignore file

upGrad

```
Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ vim account_logs.log

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Ishwar.Soni@UE-B-LAP-0121 MINGW64 /d/U-Bank (master)
$
```

Files with .log extension are not getting tracked by the Git.

Ref: Code Reference

All the codes used in today's session can be found at the link provided below:

https://github.com/upgrad-edu/tm-java/tree/session1/session1/examples

Important Concepts and Questions

Important Questions

upGrad

1. What is a ‘conflict’ in Git?
2. How does a local repository differ from a remote repository?
3. What is the difference between *git clone* and *git checkout* commands?
4. What is the difference between *git pull* and *git fetch* commands?

upGrad

Doubt Clearance Window

upGrad

Today, we learnt about the following:

1. VCS and its need
2. How Git works and how it differs from GitHub
3. How you can start with VCS using Git
4. Basic Git commands: git init, git remote origin
5. Synchronisation of the remote repo and the local repo using the following commands:

   git push, git pull, get fetch, etc.
6. Branching in Git: git branch, git checkout, git merge, etc.
7. Merge conflicts and how they are resolved

Home Work

upGrad

Please follow the steps given below to complete the Homework:

- Create a new folder using ***mkdir*** command and name it 'upgrad'.
- Change to the 'upgrad' folder using ***cd*** command.
- Initialize this directory under version control using the ***git init*** command.
- Create a new file 'courses.txt' using the ***vim*** command (vim <filename>).
- Type 'DSA' in the file and save and exit (Esc + :wq).
- Check status of the repo using ***git status*** command.
- Add the changes to the staging are using the ***git add .*** command.
- Set up the git using the ***git config --global user.email "you@example.com"*** and ***git config --global user.name "Your Name"*** commands.
- Commit the changes using the ***git commit*** command with message 'added DSA course'

Home Work

upGrad

Please follow the steps given below to complete the Homework:

- Create a new branch 'dev' using ***git branch*** command and check-it-out using ***git checkout*** command (Or you can do both using ***git checkout -b*** command).
- Update the 'courses.txt' file using the ***vim*** command and add 'Java' in the new line. So now 'courses.txt' file would contain three lines, first line would have 'DSA', second line would have 'Java' and third line would be empty.
- Check the status, stage the changes and commit the new changes with message 'added Java course'.
- So 'master' branch should have just one commit corresponding to DSA and 'dev' branch should have two commits corresponding to DSA and Java.

upGrad

Tasks to Complete After Today's Session

MCQs
Coding Questions
Homework

upGrad

#RahoAmbitious

Thank You!