# Lecture # 1.3
# Git & GitHub for Data Scientists

Dr. Muhammad Arif Butt, Ph.D.

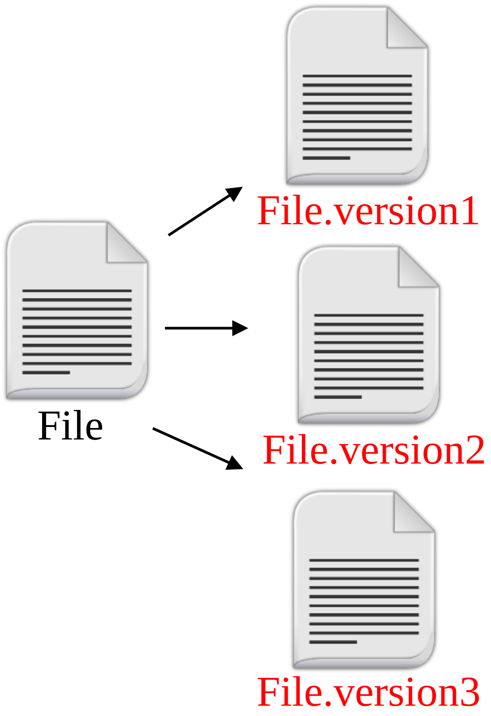
# Today’s Agenda

- Overview and Types of Version Control Systems
  - Local Data Model
  - Centralized Data Model
  - Distributed Data Model
- Overview & Working of git
- Branching & Merging
  - Overview of git branches
  - Merge branches
  - Handling merge conflicts
- Web Portals & Cloud Hosting Services for git
  - Creating remote repository, uploading files and inviting collaborators
  - Cloning a remote repo from gitHub
  - Pushing a local repo to GitHub
  - Fetch vs Pull
  - Forking a repo from GitHub

# Overview and Types: Version Control System

# Overview of Revision/Version Control System

- A Version Control System is a software tool that records changes to a file or a set of files over time, so that you can recall specific versions later.



VCS allows to maintain history of different versions of a file

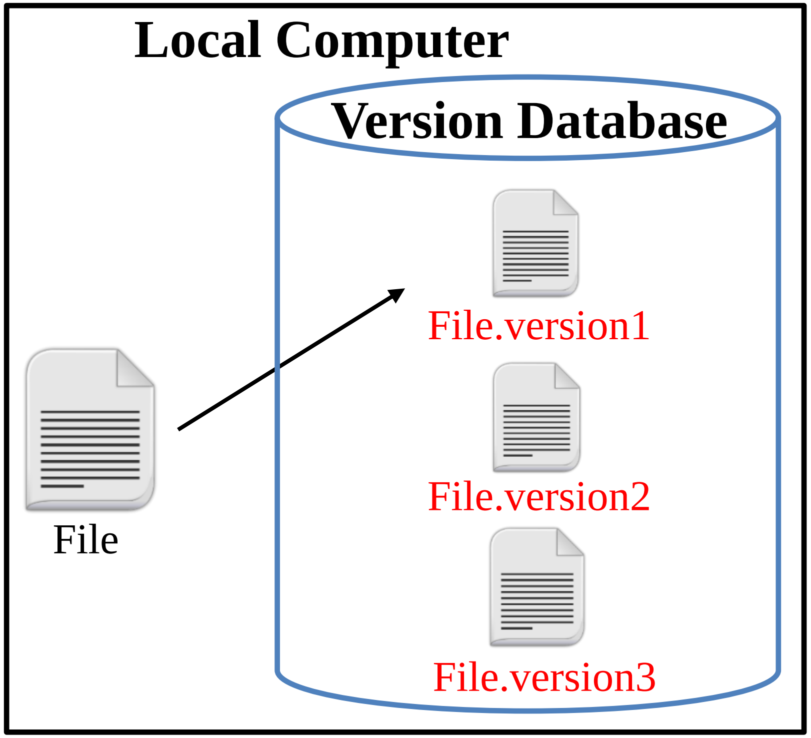
To move back and forth between these versions

Compare different versions

Merge multiple versions of same file

Lock other users when one user is altering a file

# Local Data Model



A local VCSs maintains a version database that keep track of all the changes made to file(s)

By applying the change sets you can move from one file version to the other

**Source Code Control System (SCCS-1972)**

- It was written in C, developed by AT&T and was for UNIX only
- It just save the snapshot of the changes, If you want ver.3 of a file, you take ver.1 of the file and apply two set of changes to it to get to ver.3

**Revision Control System (RCS-1982)**

- It was written in C, developed at Purdue University, and other than UNIX works on PCs as well
- RCS keeps the most recent version of a file in its whole form and if you want a previous version, you make changes to the latest version to re-create the older version

**Limitations of Local VCSs:**

- You can track changes in a single file
- Only one user can work with a file at a single time, team members cannot collaborate and work on the same project

# Centralized Data Model

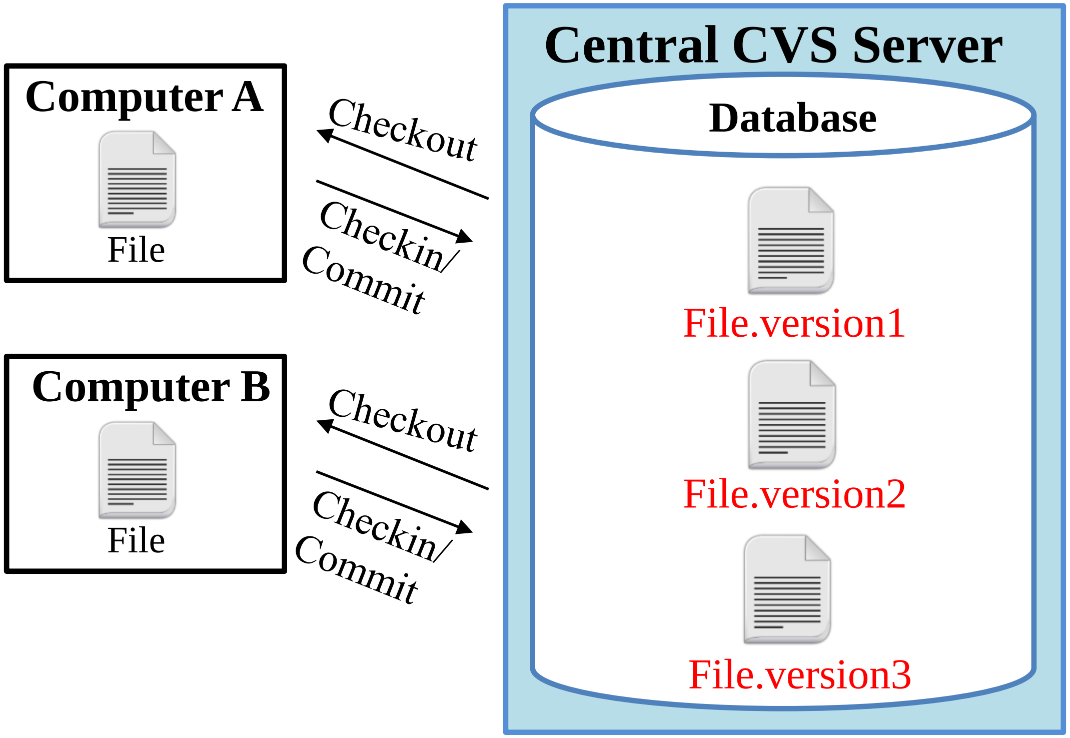
In central VCSs, there is a server machine that contains the version database (repository) which keeps track of number of clients working on those file(s)

**Concurrent Version System (CVS-1990)**

- Written in C, is open source, and available for UNIX and MS OSs
- Introduced the idea of branching
- CVS lack atomic operations
- File renaming not possible as CVS cannot track directories

**Apache Subversion System (SVN-2000)**

- Written in C, is open source, is cross platform and is faster than CVS
- Supports atomic commits
- Can track directories, so you can rename files within directories
- It can also track non-text files like images

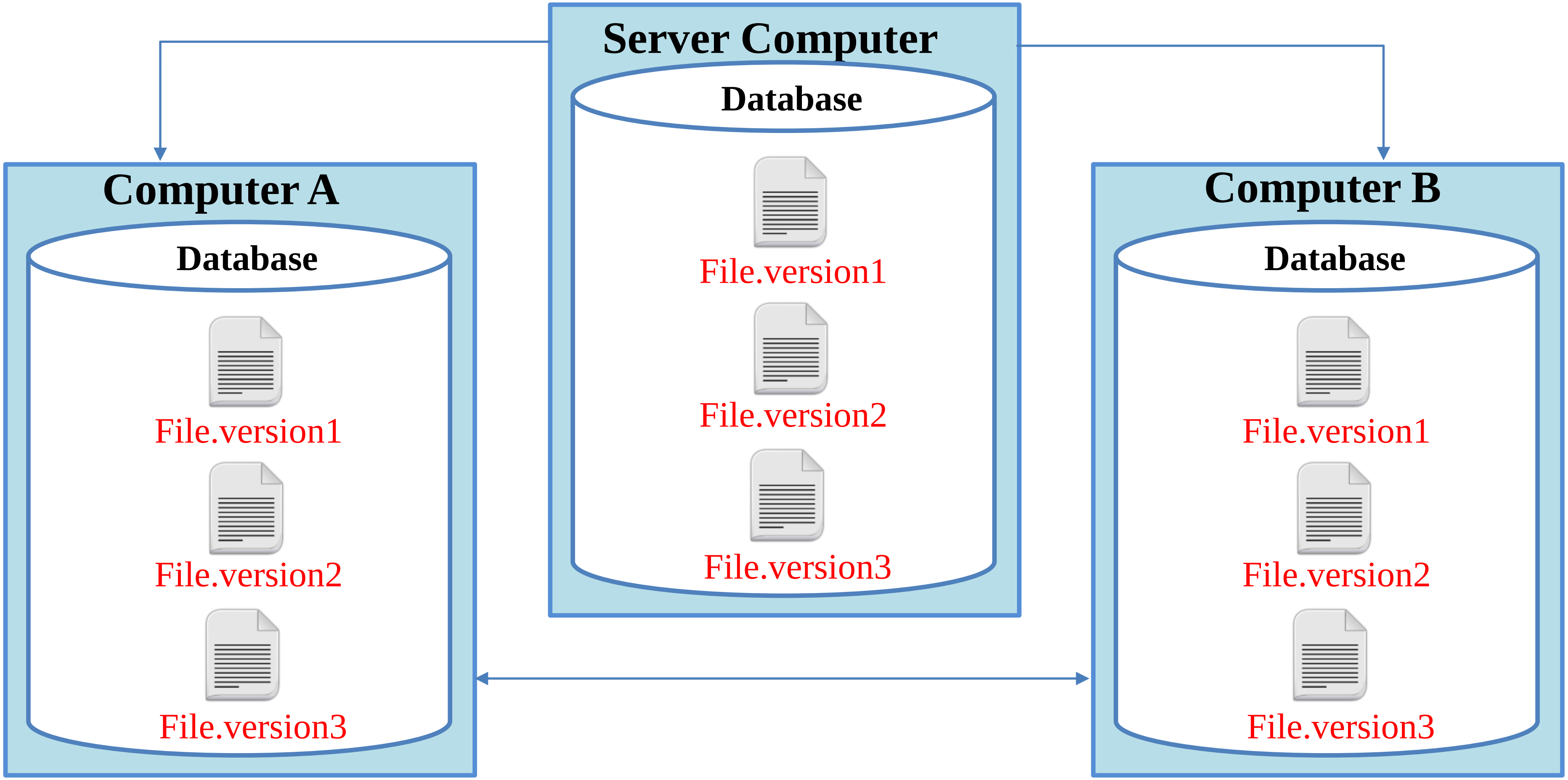


**Limitations of Centralized VCSs:**

- Single point of failure as the centralized server containing the version database may crash
- No collaboration if server is down

# Distributed Data Model

- In a DVCS, clients don't just check out the latest snapshot of the files; they fully mirror the entire repository (version database).
- Each developer works with his own local repository and changes are finally pushed or committed on the remote repository as a separate step.



# 3- Distributed Data Model (cont..)

**Bitkeeper -2000**

It was written in C, and is proprietary and closed source

Bitkeeper with limited functionalities was free and used to manage Linux Kernel

In 2005, the "community version of bitkeeper" stopped being free and it was then git was born

**git -2005**

Developed by Linus Torvald in 2005, is free and open source

It is compatible with all UNIX-like systems & MS Windows, written in C, TCL, Perl & python

**Pros:**

- Faster speed
- No risk of loosing history, as every user has complete mirror of repository

**Cons:**

- More space occupied on local disk of user
- More load on network while checking out project in local repository and committing project in remote repository

# Overview & Working of Git

# Downloading & Installation

## ➢ On Linux

```
https://git-scm.com
sudo apt-get install git
which git
git version
git help <git/tutorial/everyday>
```

You can Download git from official website

Or Download & install git using this command

Confirm the installation

To get help about any command or any concept

## ➢ On Windows

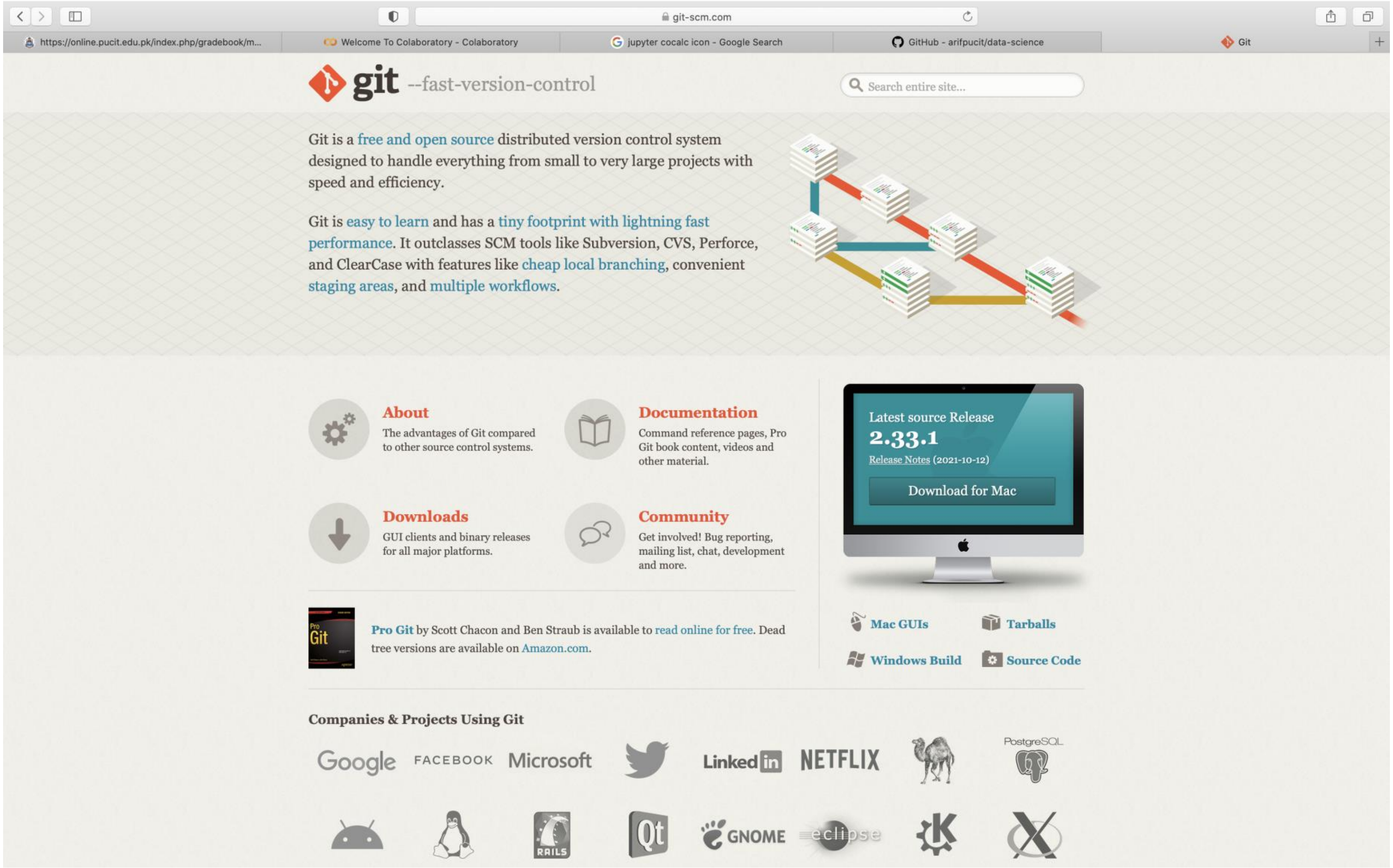
```
Git for Windows installer
git version
git help
```

You can Download git GUI, CMD & bash interfaces

Confirm the installation

To get help about any command or any concept

# Downloading & Installation


git-scm.com
https://online.pucit.edu.pk/index.php/gradebook/m...
Welcome To Colaboratory - Colaboratory
jupyter cocalc icon - Google Search
GitHub - arifpucit/data-science
Git
git --fast-version-control
Search entire site...

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

**About**
The advantages of Git compared to other source control systems.

**Documentation**
Command reference pages, Pro Git book content, videos and other material.

**Downloads**
GUI clients and binary releases for all major platforms.

**Community**
Get involved! Bug reporting, mailing list, chat, development and more.

Pro Git by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Latest source Release
2.33.1
Release Notes (2021-10-12)

**Companies & Projects Using Git**

# GIT: GUI-Clients

**SourceTree**
Platforms: Mac, Windows
Price: Free
License: Proprietary

**GitHub Desktop**
Platforms: Mac, Windows
Price: Free
License: MIT

**GitKraken**
Platforms: Mac, Windows, Linux
Price: Free/Paid
License: Proprietary

**TortoiseGit**
Platforms: Windows
Price: Free
License: GNU GPL

**Git-Cola**
Platforms: Mac, Windows, Linux
Price: Free
License: GNU GPL

# Git Configuration

- ➢ User Configuration

~/.gitconfig

```
https://git-scm.com
$ git config --global user.name “Arif Butt”
$ git config --global user.email “arif@pucit.edu.pk”
$ git config --global core.editor “vim”
$ git config --global --list
$ cat ~/.gitconfig
```

User Configuration Attributes

You can check values of these configurations using these commands

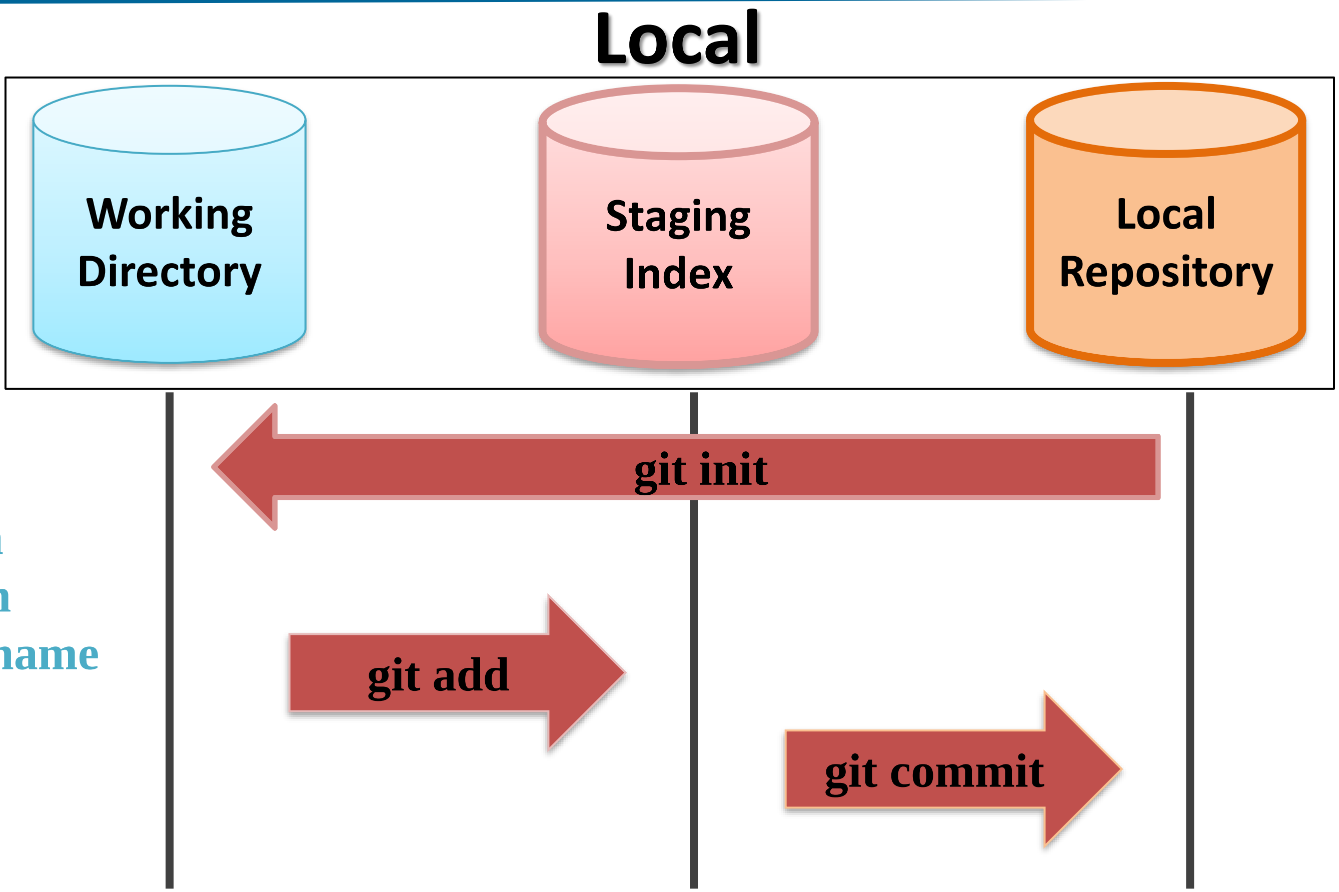
- ➢ System Configuration

/etc/gitconfig

- ➢ Project Configuration

<proj>/.git/config

# Basic Workflow of git



- File creation
- Modification
- Deletion/Rename
- Ignore files

**Working directory** is any directory on your file system that has a a subdirectory named .git inside it

**Staging Index** is an intermediate area, changes doesn't commit directly from the working tree to repository. Instead changes are first made in the staging index

**Repository** or object store holds the changes in your source code over time as you perform commit ops

# Initialization & Life Cycle of file in git

```
Initializing git
$ git init
```

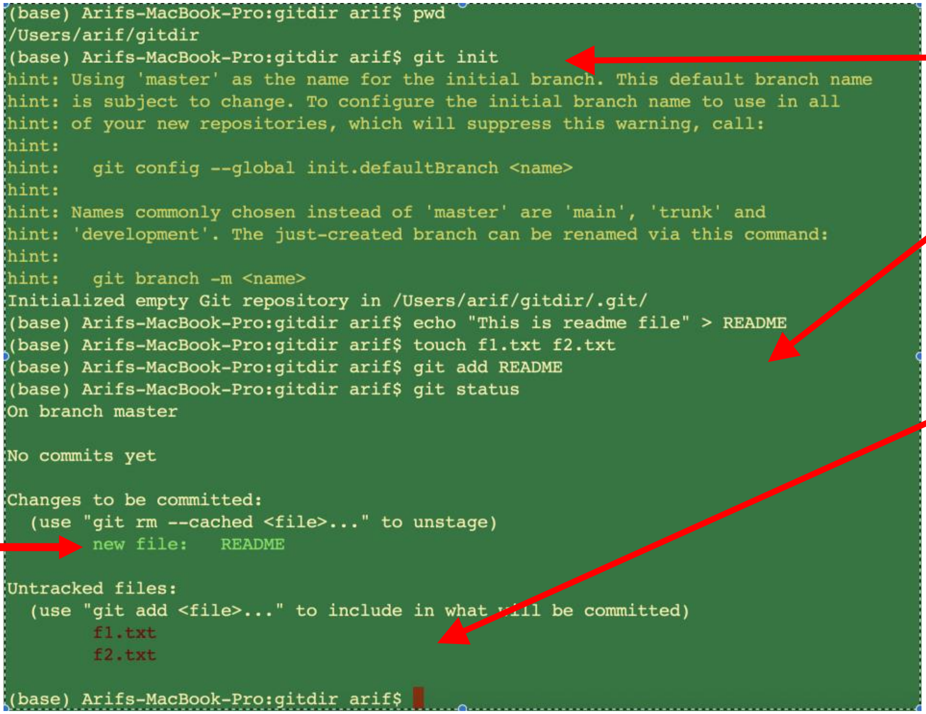
After configuration, next step is to initialize repository. It will make a hidden folder named .git in this directory. This is your local versioning database that track all the files/ inside the root directory of your project folder

```
$ git status
$ git add <filename>
```

This will tell which files are tracked and which are un-tracked

**Tracked files:** All the files which have been added at least once, or the files that were there in the last snapshot

- Unmodified
- Modified
- Staged



```
(base) Arifs-MacBook-Pro:gitdir arif$ pwd
/Users/arif/gitdir
(base) Arifs-MacBook-Pro:gitdir arif$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/arif/gitdir/.git/
(base) Arifs-MacBook-Pro:gitdir arif$ echo "This is readme file" > README
(base) Arifs-MacBook-Pro:gitdir arif$ touch f1.txt f2.txt
(base) Arifs-MacBook-Pro:gitdir arif$ git add README
(base) Arifs-MacBook-Pro:gitdir arif$ git status
On branch master

No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f1.txt
        f2.txt

(base) Arifs-MacBook-Pro:gitdir arif$ 
```

**Initialize Repository**

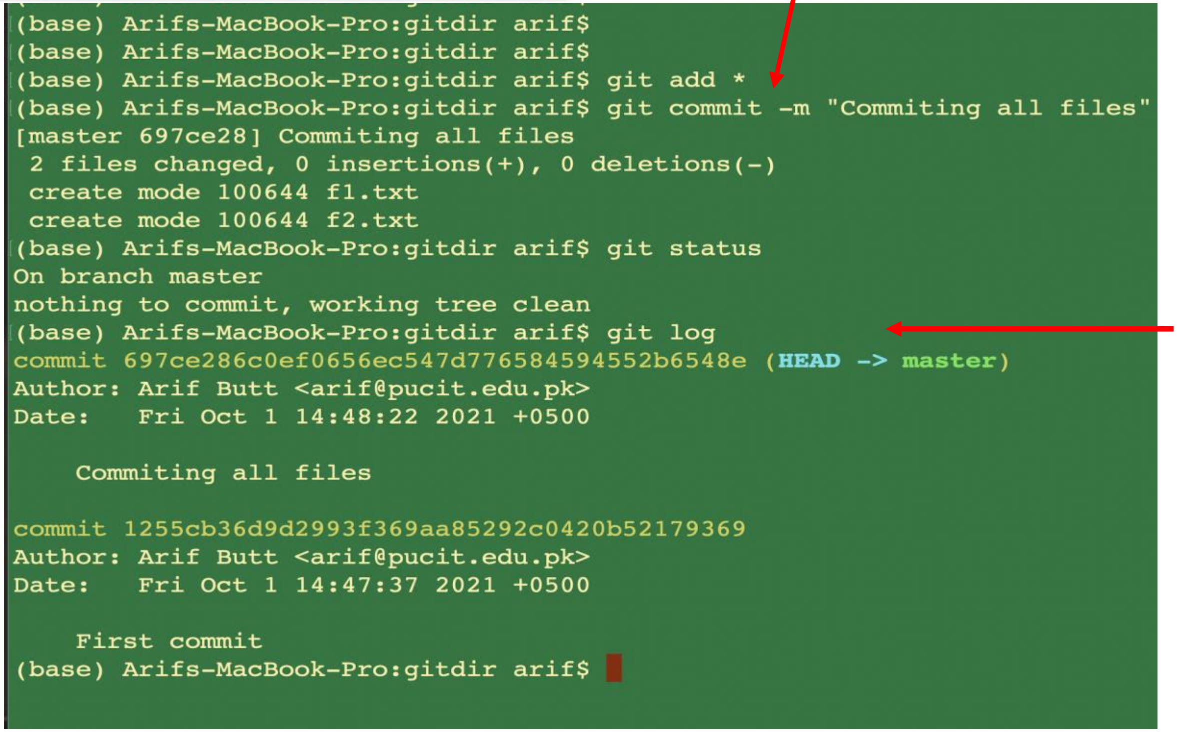
**Create some files and add one to Staging Index**

**Untracked files**: All the files in the working directory that have never been part of repository and are not even in the staging area

# Commit file & view commit log

```
$ git commit -m “message”
```

After adding all files to staging area now they are ready to commit



```
(base) Arifs-MacBook-Pro:gitdir arif$
(base) Arifs-MacBook-Pro:gitdir arif$
(base) Arifs-MacBook-Pro:gitdir arif$ git add *
(base) Arifs-MacBook-Pro:gitdir arif$ git commit -m "Commiting all files"
[master 697ce28] Commiting all files
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f1.txt
 create mode 100644 f2.txt
(base) Arifs-MacBook-Pro:gitdir arif$ git status
On branch master
nothing to commit, working tree clean
(base) Arifs-MacBook-Pro:gitdir arif$ git log
commit 697ce286c0ef0656ec547d776584594552b6548e (HEAD -> master)
Author: Arif Butt <arif@pucit.edu.pk>
Date:   Fri Oct 1 14:48:22 2021 +0500

    Commiting all files

commit 1255cb36d9d2993f369aa85292c0420b52179369
Author: Arif Butt <arif@pucit.edu.pk>
Date:   Fri Oct 1 14:47:37 2021 +0500

    First commit
(base) Arifs-MacBook-Pro:gitdir arif$
```

Check log

```
$ git log [--oneline][--author=“name”]
commit <sha of commit o/p as 40 hex digits>
Author: username <email>
Date: <date and time>
<commit message>
```

You can check log of commits and by whom it is committed

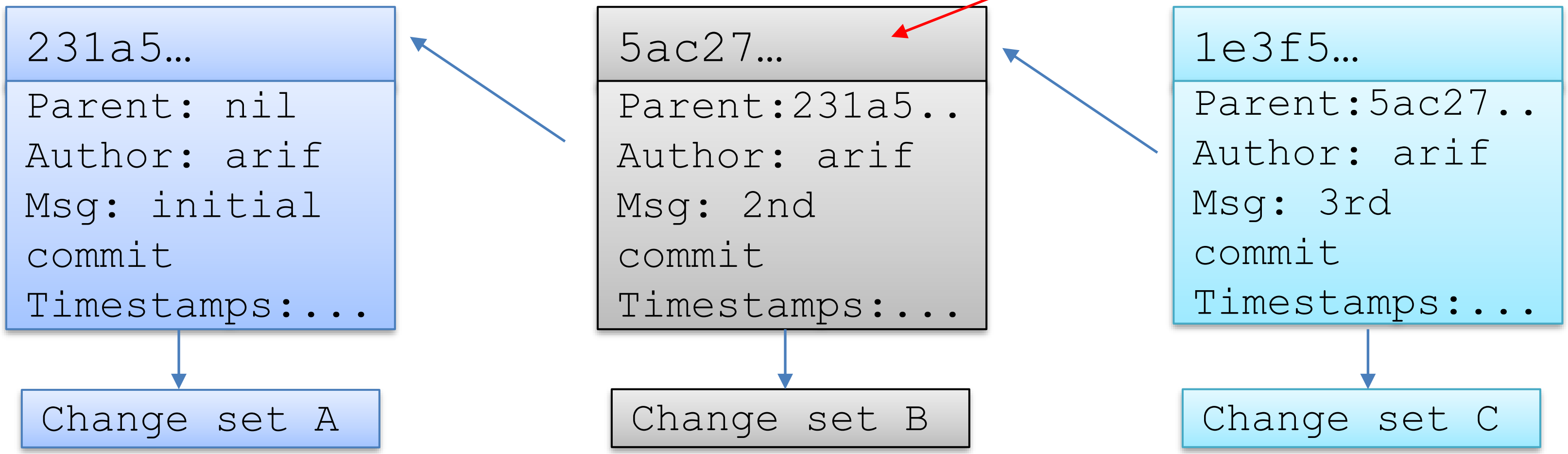
It will show you list of all commits in the following format:

# Basic Workflow of git

# Commit objects and Head pointer in git

Suppose we have made three commits in our project, that means there are three change sets. Each commit object refers to a change set.

Checksum generated through Secure Hash Algorithm



231a5...
Parent: nil
Author: arif
Msg: initial commit
Timestamps:...

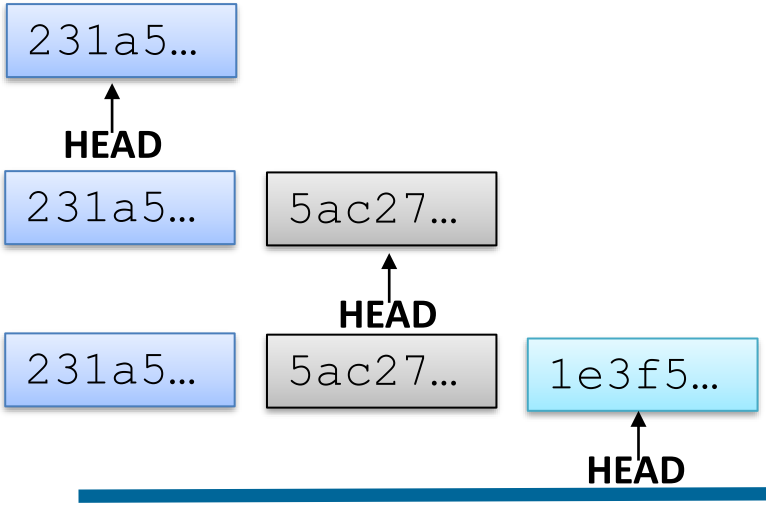
5ac27...
Parent:231a5..
Author: arif
Msg: 2nd commit
Timestamps:...

1e3f5...
Parent:5ac27..
Author: arif
Msg: 3rd commit
Timestamps:...

Change set A

Change set B

Change set C



231a5...
HEAD

231a5... 5ac27...
HEAD

231a5... 5ac27... 1e3f5...
HEAD

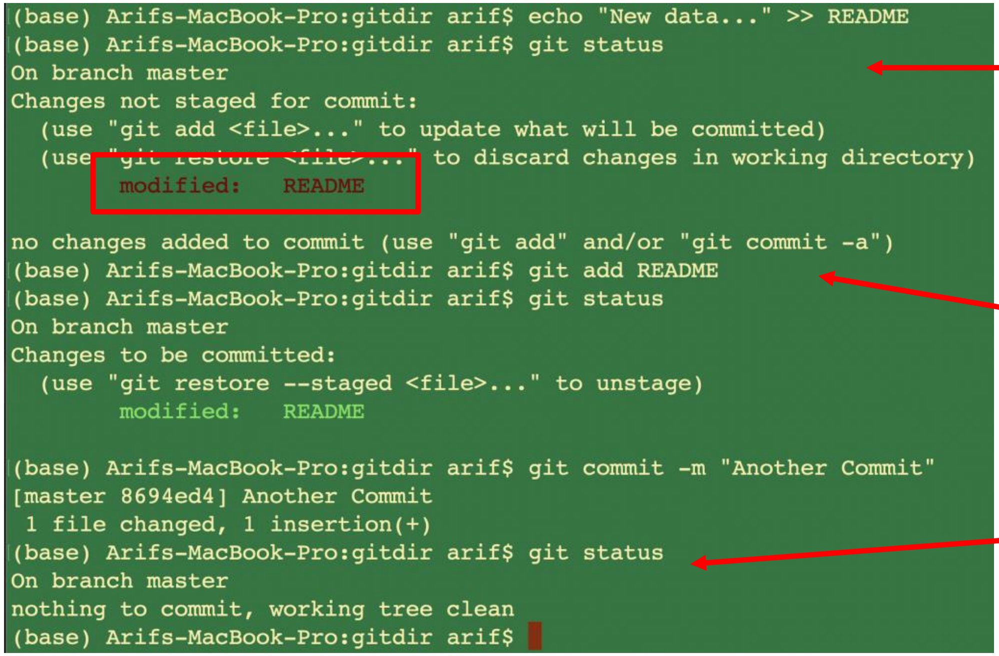
git maintains a reference variable called HEAD, which points to a specific commit in repo

As we make a new commit the HEAD moves to point the next commit

```
$ cat .git/HEAD
refs/heads/master
$ cat .git/refs/heads/master
5ac27..
```

# Edit, Delete a File in git Repo

## ➢ Edit File



```
(base) Arifs-MacBook-Pro:gitdir arif$ echo "New data..." >> README
(base) Arifs-MacBook-Pro:gitdir arif$ git status
On branch master
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:   README

no changes added to commit (use "git add" and/or "git commit -a")
(base) Arifs-MacBook-Pro:gitdir arif$ git add README
(base) Arifs-MacBook-Pro:gitdir arif$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README

(base) Arifs-MacBook-Pro:gitdir arif$ git commit -m "Another Commit"
[master 8694ed4] Another Commit
 1 file changed, 1 insertion(+)
(base) Arifs-MacBook-Pro:gitdir arif$ git status
On branch master
nothing to commit, working tree clean
(base) Arifs-MacBook-Pro:gitdir arif$
```

We have already created a file README, added in staging index and then committed it to the repo. Make changes in the file and check status.

You again need to add and commit the file

Check status

## ➢ Delete File

```
$ rm f1.txt
$ git add f1.txt
$ git commit -m "deleted"
```

Option 1: Move the file out from the working dir into trash and then tell git about it

```
$ git rm f1.txt
$ git commit -m "deleted"
```

Option 2: Tell git to remove the file and add it to staging index in a single command

# Rename a File in git Repo

- Rename file

```
$ mv f1.txt newf1.txt
$ git add newf1.txt
$ git rm f1.txt
$ git commit -m “rename”
```

Option 1: Move or rename files using the GUI file browser or file system commands.

Then come back and tell git about those changes

```
$ git mv f1.txt newf1.tx
$ git commit -m “renamed”
```

Option 2: Move/rename file from git command line.

# Ignoring Files in git

Write files/directories names to be ignored in a text file.

Git normally checks `gitignore` patterns from multiple sources, with the following order of precedence:

- The patterns read from a file named `.gitignore` in the same directory or in any parent directory upto the top level of the working tree.
- The patterns read from `.git/info/exclude` file in the project directory.
- The patterns read from file specified by the configuration variable `core.excludesFile`

```
$ git config –global core.excludesfile ~/.abc
```

```
*.o
*.tar.gz
*.log
*.[oa]
*.exe
myexe
logs/**
dir1/
```

# Moving to a Previous Commit

## Soft Reset

```
$ git reset --soft <Commit ID>
```

- Head is moved to specific commit ID
- No changes are made in the staging index and working directory

## Mixed Reset

```
$ git reset --mixed <Commit ID>
```

- Head is moved to specific commit ID
- Staging index is also changed to match the local repository
- No changes are made in the working directory

## Hard Reset

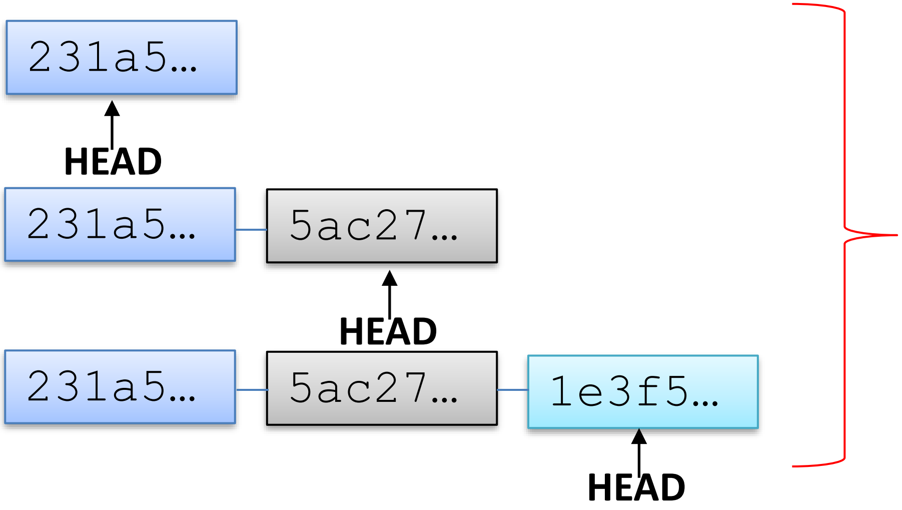
```
$ git reset --hard <Commit ID>
```

- Head is moved to specific commit ID
- Staging index and working directory both match the local repository

# Edit, Delete, Rename and Ignore Files in git

# Branching & Merging

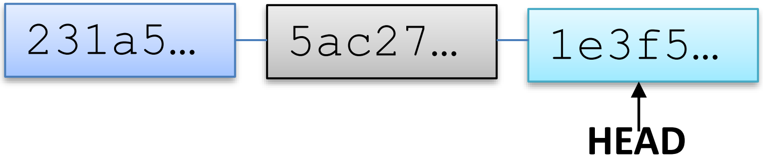
# Overview of git Branches



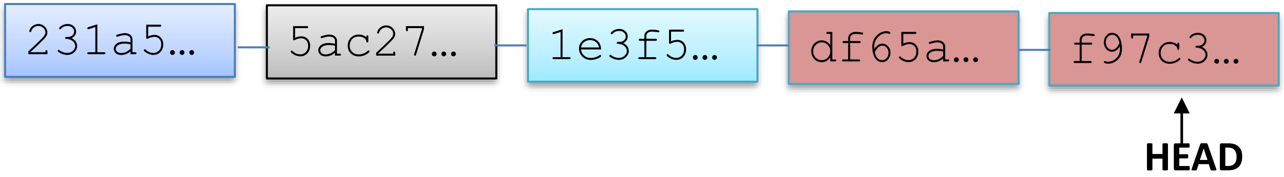
A git branch represents an independent line of development

Every git repository has at least one branch called the master branch

- Suppose you are working on a project and have done some commits on the master branch. You think of adding a new feature to your project but you are not sure whether it will work or not.

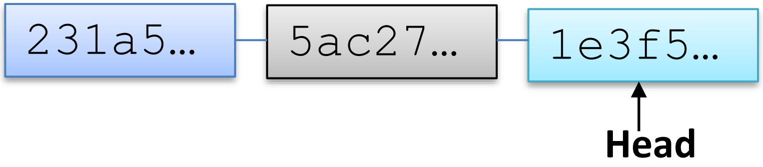


- **OPTION 1:** You continue working on the same branch

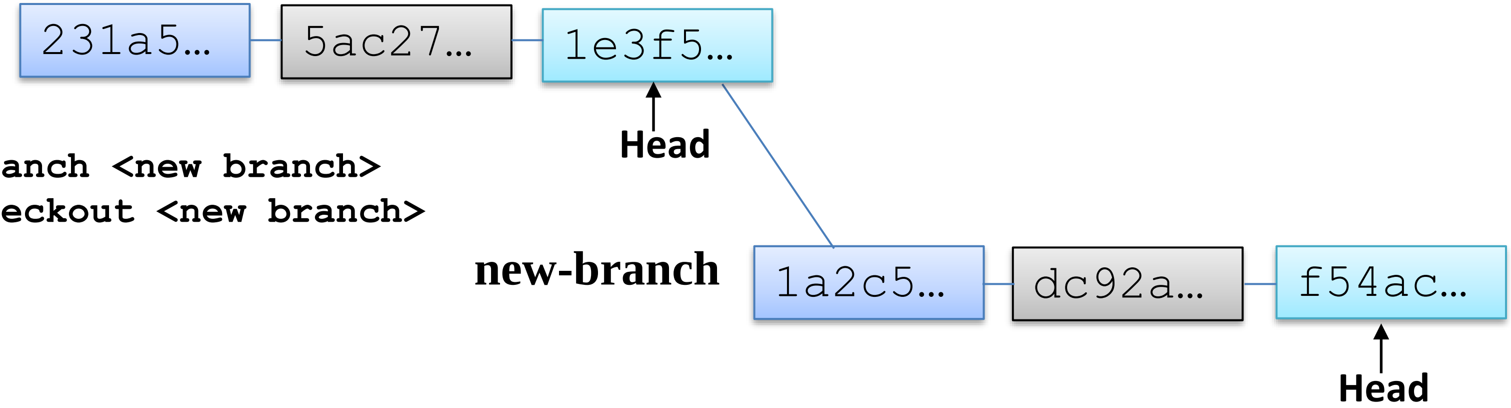


- If it is a success GR8. If it is a failure, you roll back to commit with SHA `1e3f5…`

# Overview of git Branches (cont..)



- **OPTION 2:** Create a new branch and try your new ideas there and if those ideas do not work you just throw away that branch and your master branch continues moving ahead without any issues

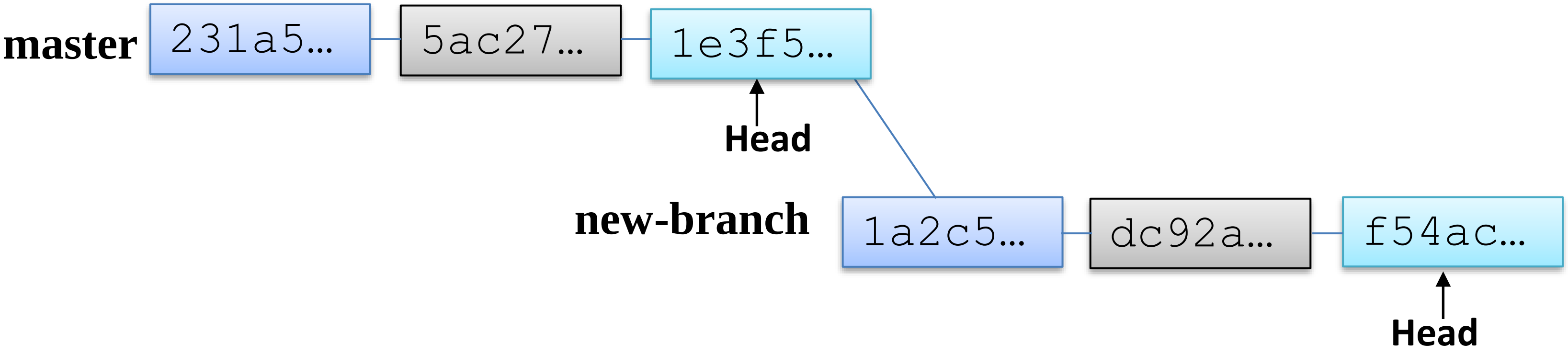


```
$ git branch <new branch>
$ git checkout <new branch>
```

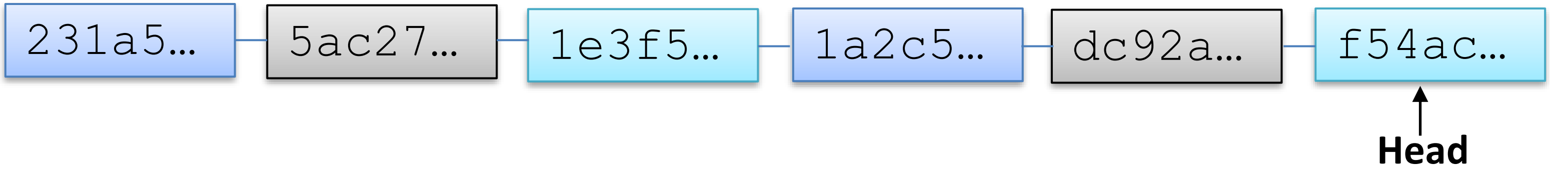
- If the new branch is a success, then you need to merge your new-branch with the master branch

# Merging Branches: Fast Forward Merge

- Suppose you made a new branch and no further commits have been done on the master branch after the creation of new-branch as shown:



- In this case, git will by default do a **Fast Forward Merge**



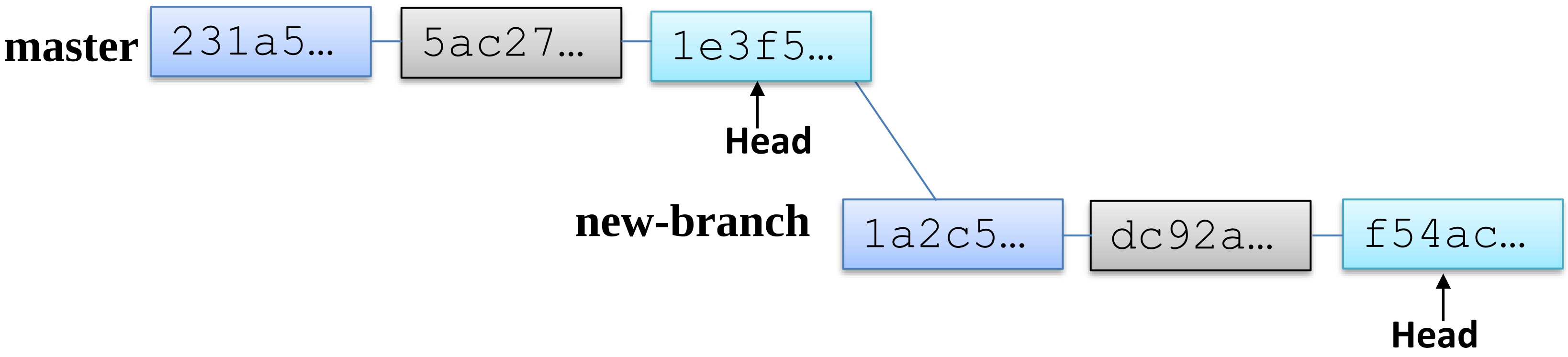
```
$ git checkout master

$ git merge new-branch
```

Before you give merge command, your current branch should be the receiving branch

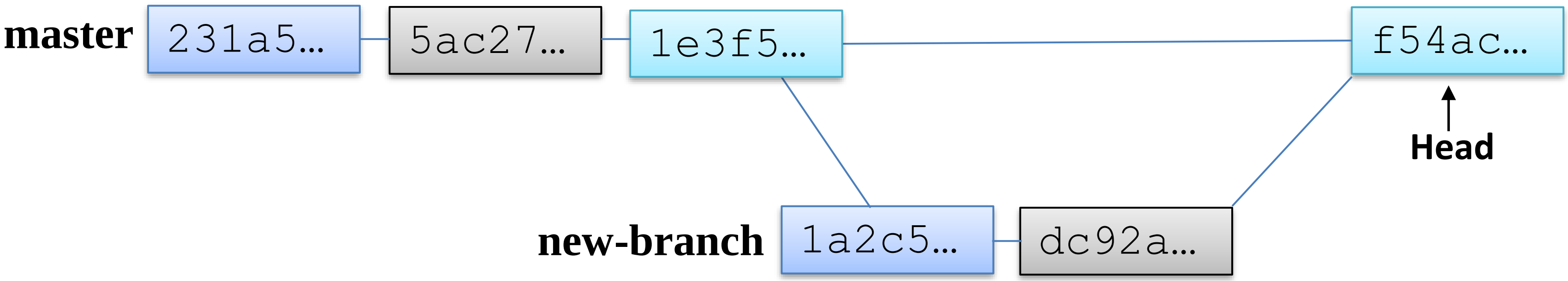
Merge Master branch with new branch

# Merging Branches: Real Merge

- Suppose you made a new branch and no further commits have been done on the master branch after the creation of new-branch as shown:



- You can always force git NOT to do a fast forward merge, rather do an additional commit merge. This can be forced by giving the `--no-ff` option to `git merge` command



```
$ git checkout master

$ git merge –no–ff new-branch
```

Before you give merge command, your current branch should be the receiving branch

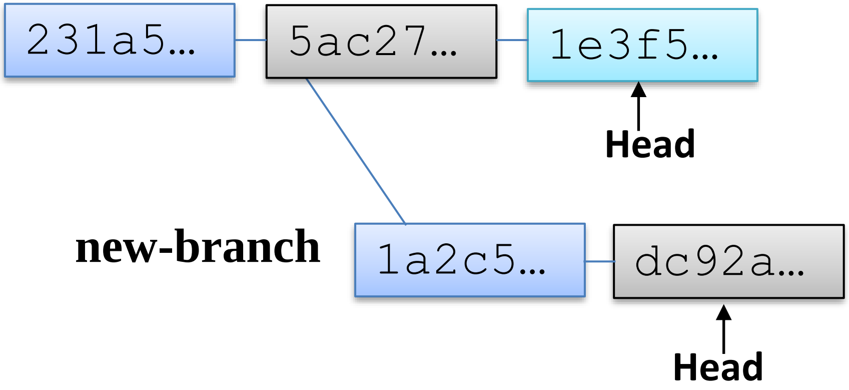
Merge Master branch with new branch

Instructor: Muhammad Arif Butt, Ph.D.

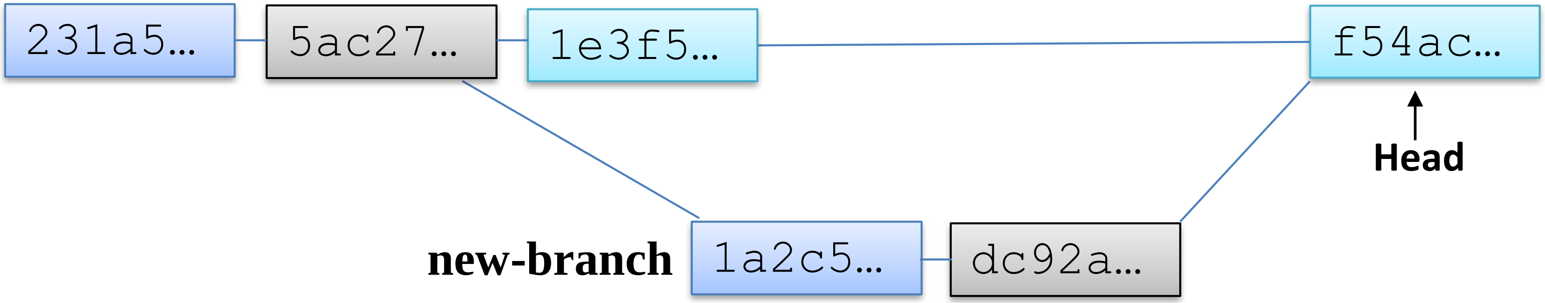
# Merging Branches: Real Merge

In the following scenario a fast forward merge is not possible. So once you do a merge, git will perform a real merge.

- Before Merging

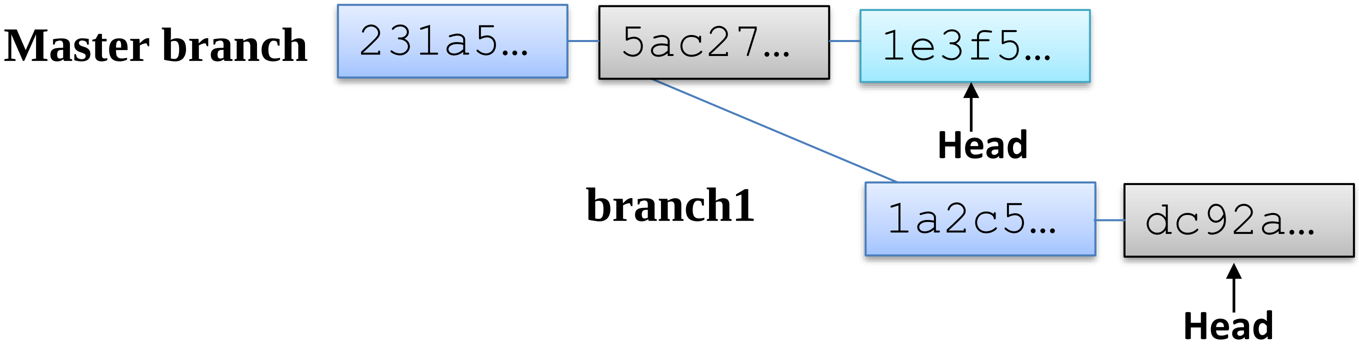


- After Merging



# Handling Merge Conflicts

Suppose there are two branches master and branch1, both have a file f1.txt, which is of-course similar in both. A developer on master branch edit line#25 of file1.txt and do a commit. Another developer on branch1 edit line#50 of file1.txt and do a commit



Now if you merge, it will be a success, because both have made changes to same file, but to different lines. However, if both the developers have made changes to same line or set of lines a conflict will occur, which git cannot handle and it will give a message that auto-merging failed. In case of a merge conflict we have three choices to resolve the conflict

- **Abort merge:** `$ git merge –abort`
- **Make changes Manually:** Perform changes manually in some editor, add, commit, and finally perform merge
- **Use merge tools**: You can use for this purpose like araxis, diffuse, kdiff3, xxdiff, diffmerge: `$ git mergetool --tool=diffuse`

# Overview of git Branches (cont..)

- To Create a New Branch

```
$ git branch [<new-branch>]
```

- To Rename a Branch

```
$ git branch -m <old> <new>
```

- To Delete a Merged Branch

```
$ git branch -d <branch-name>
```

- To Delete an Un-merged Branch

```
$ git branch -D <branch-name>
```

- To Compare two Branches Branch

```
$ git diff <branch1> <branch2>
```

- To Switch to another Branch

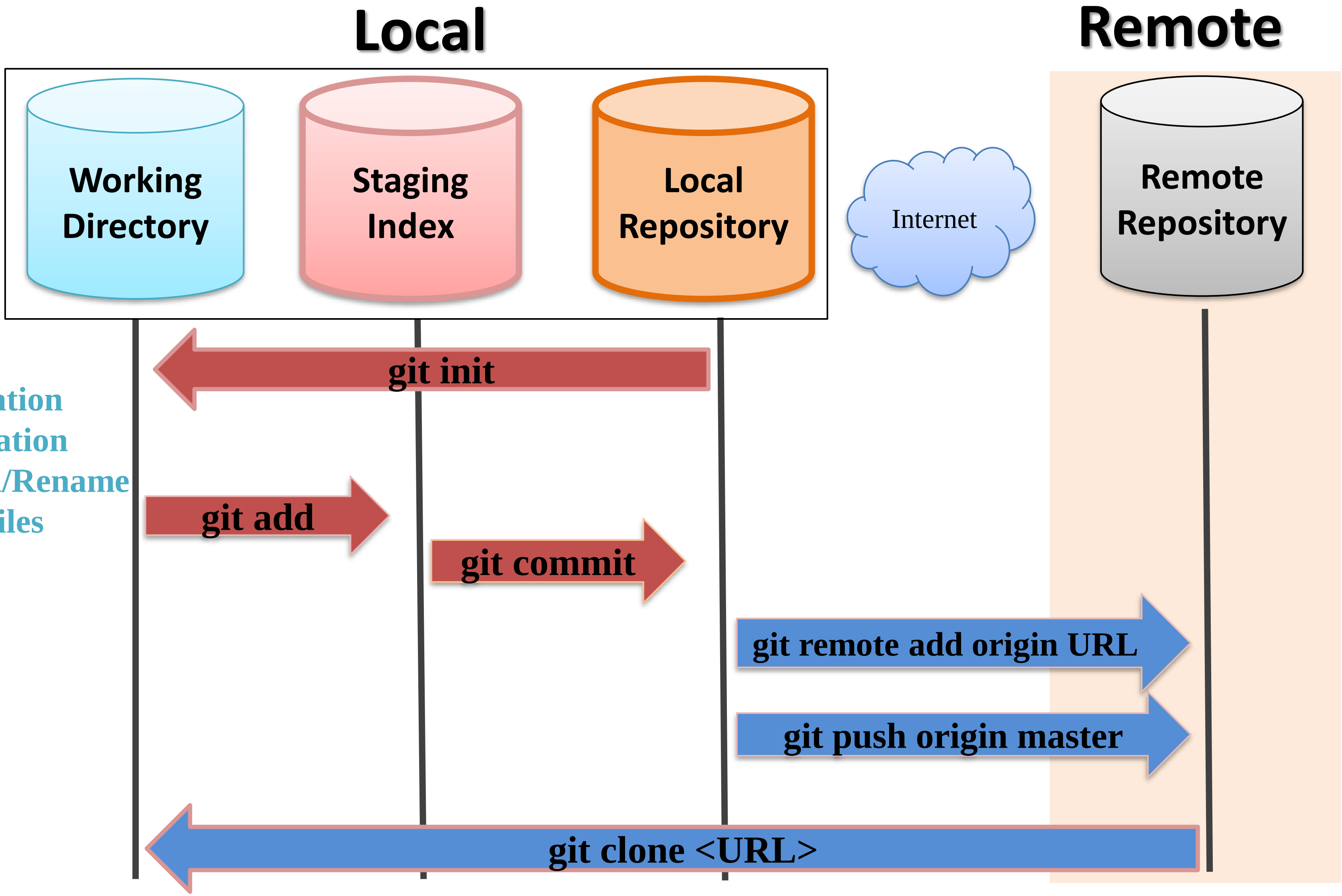
```
$ git checkout new-branch
```

# Branches in git

# Web Portals & Cloud Hosting Services for git

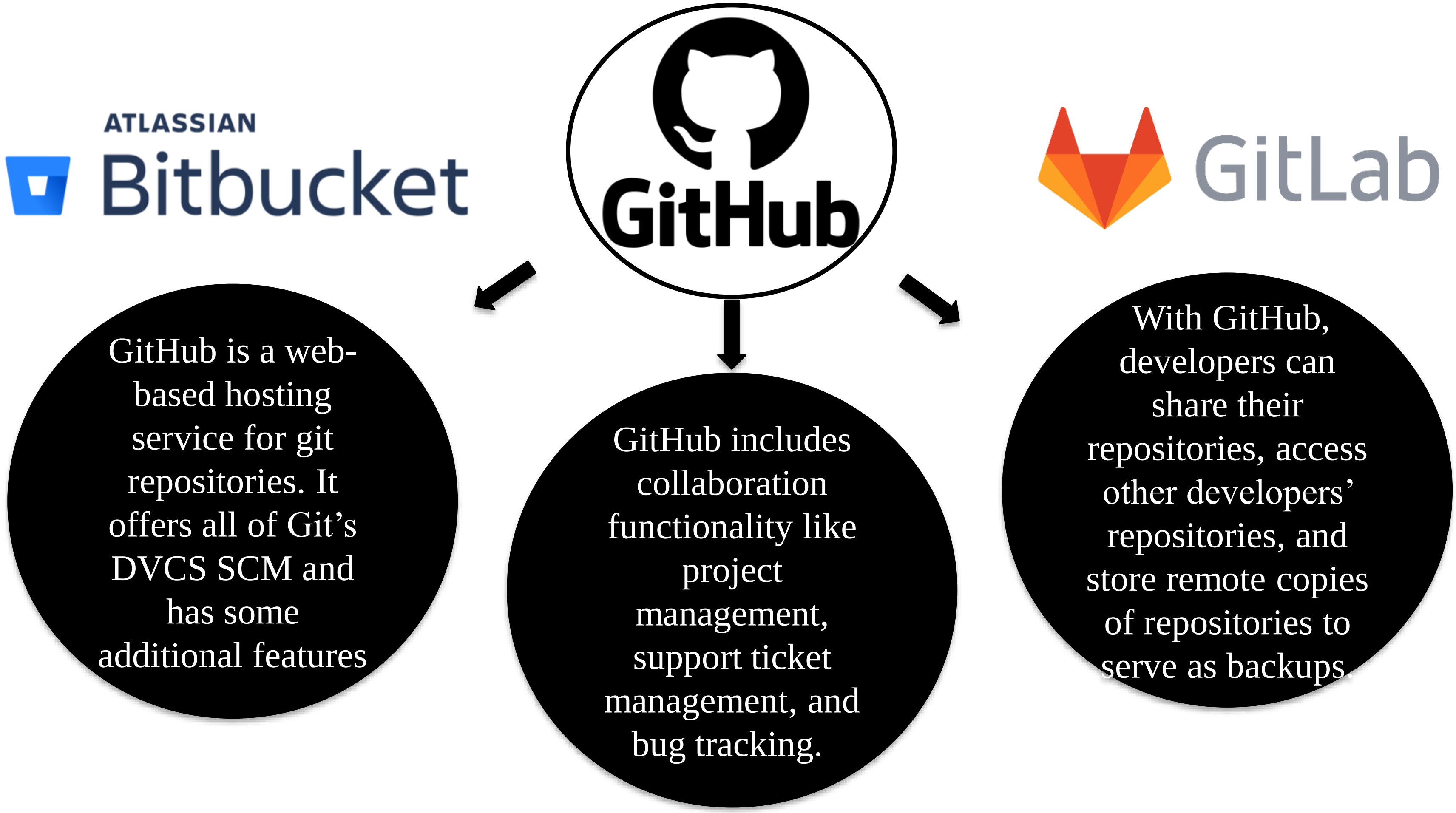
# Concept of Remote Repository

- File creation
- Modification
- Deletion/Rename
- Ignore files



# Hosting Services for git Repositories

The way there are different web hosting services available on the Internet cloud, similarly there are hosting services available for repositories of distributed versioning systems as well



GitHub is a web-based hosting service for git repositories. It offers all of Git's DVCS SCM and has some additional features

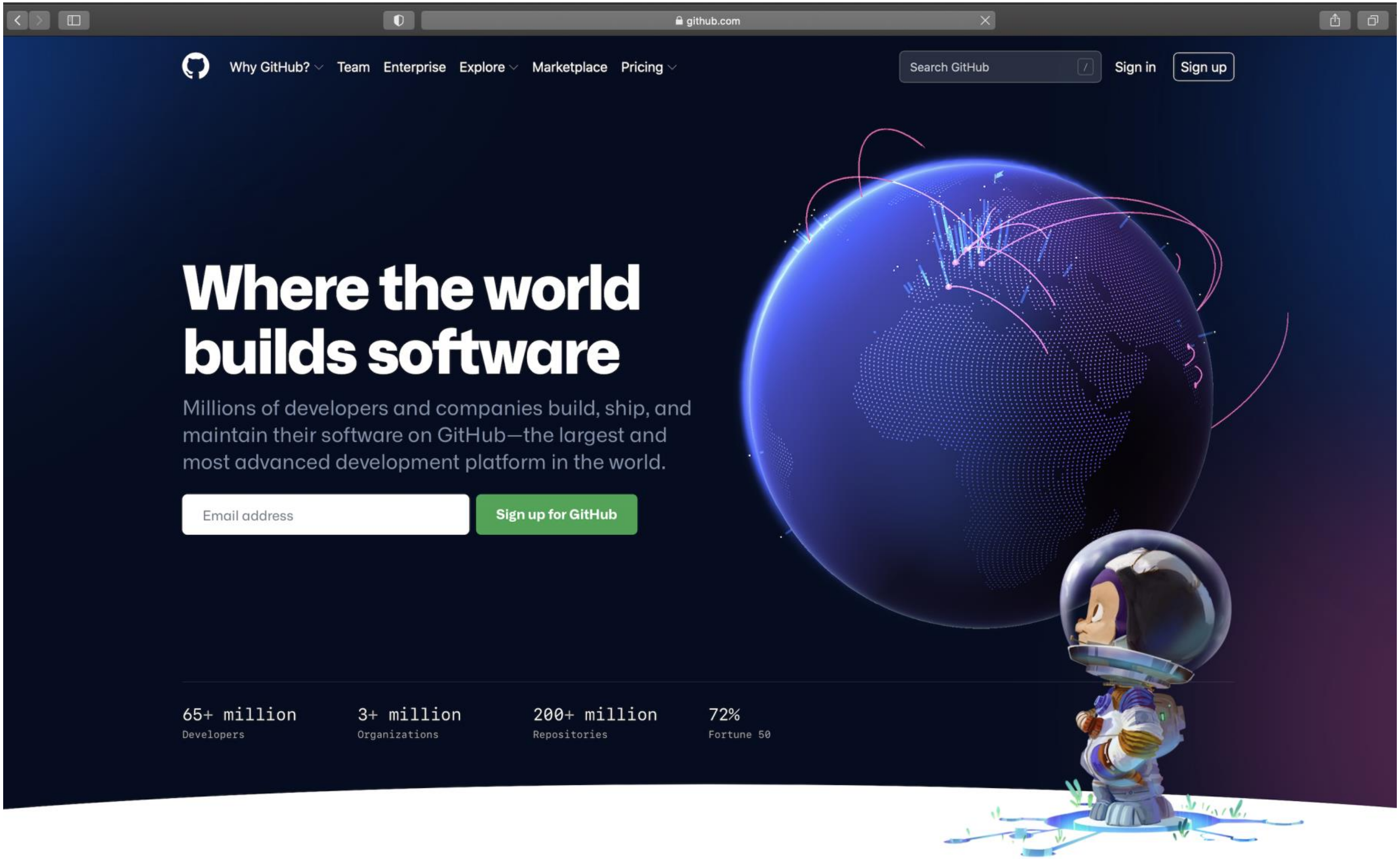
GitHub includes collaboration functionality like project management, support ticket management, and bug tracking.

With GitHub, developers can share their repositories, access other developers' repositories, and store remote copies of repositories to serve as backups.

# Creating a Remote Repository on GitHub

# Creating a Personal Account on GitHub

To create your repositories on GitHub or contribute to other open source projects, you will need to create a personal account GitHub



**Where the world builds software**

Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.

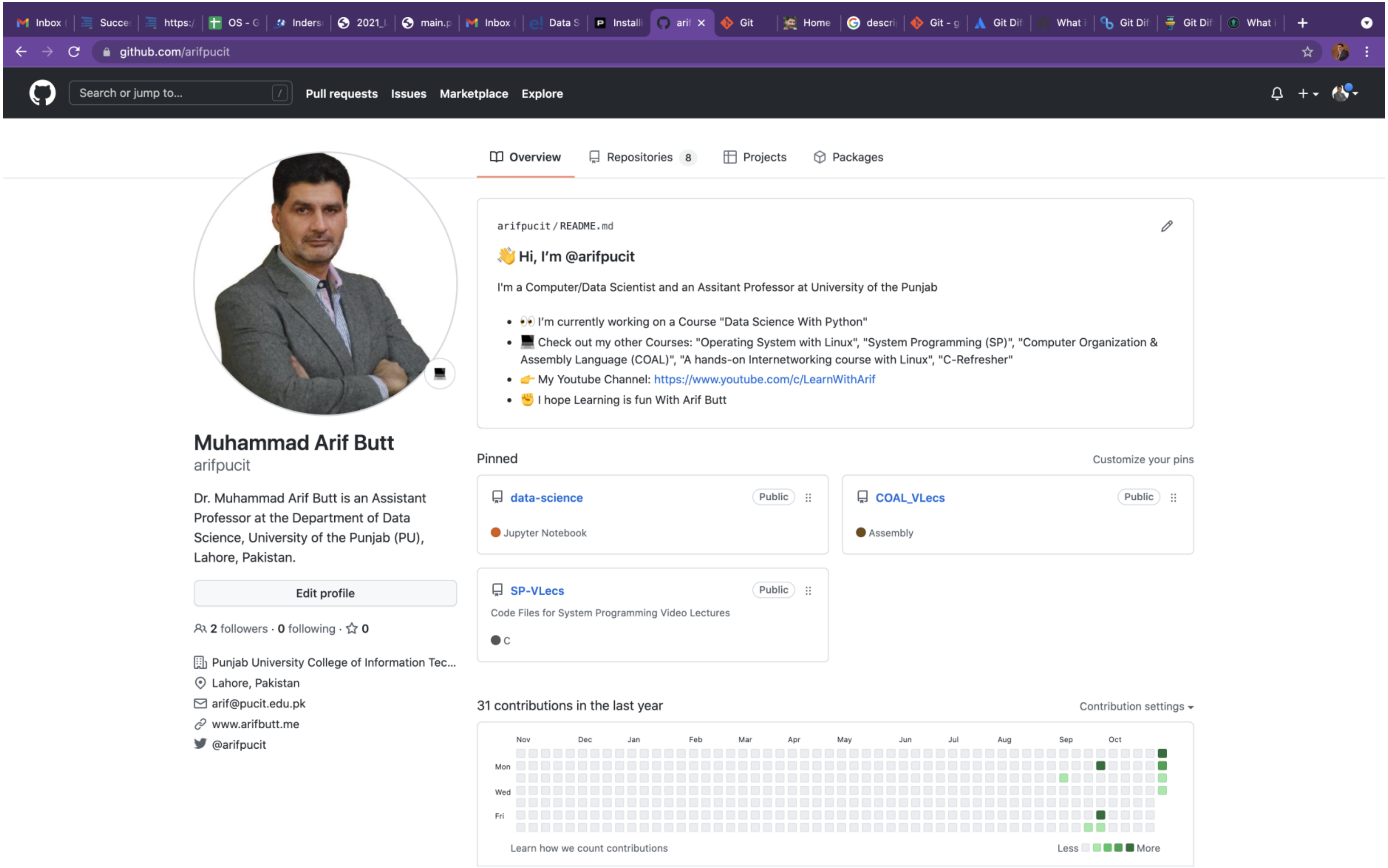
65+ million Developers

3+ million Organizations

200+ million Repositories

72% Fortune 50

# Login into your GitHub Account



arifpucit / README.md

👋 Hi, I'm @arifpucit

I'm a Computer/Data Scientist and an Assitant Professor at University of the Punjab

- 👀 I'm currently working on a Course "Data Science With Python"
- 💻 Check out my other Courses: "Operating System with Linux", "System Programming (SP)", "Computer Organization & Assembly Language (COAL)", "A hands-on Internetworking course with Linux", "C-Refresher"
- 👉 My Youtube Channel: https://www.youtube.com/c/LearnWithArif
- ✊ I hope Learning is fun With Arif Butt

Muhammad Arif Butt
arifpucit

Dr. Muhammad Arif Butt is an Assistant Professor at the Department of Data Science, University of the Punjab (PU), Lahore, Pakistan.

Edit profile

2 followers · 0 following · ☆ 0

Punjab University College of Information Tec...
Lahore, Pakistan
arif@pucit.edu.pk
www.arifbutt.me
@arifpucit

Pinned | Customize your pins

data-science | Public
● Jupyter Notebook

COAL_VLecs | Public
● Assembly

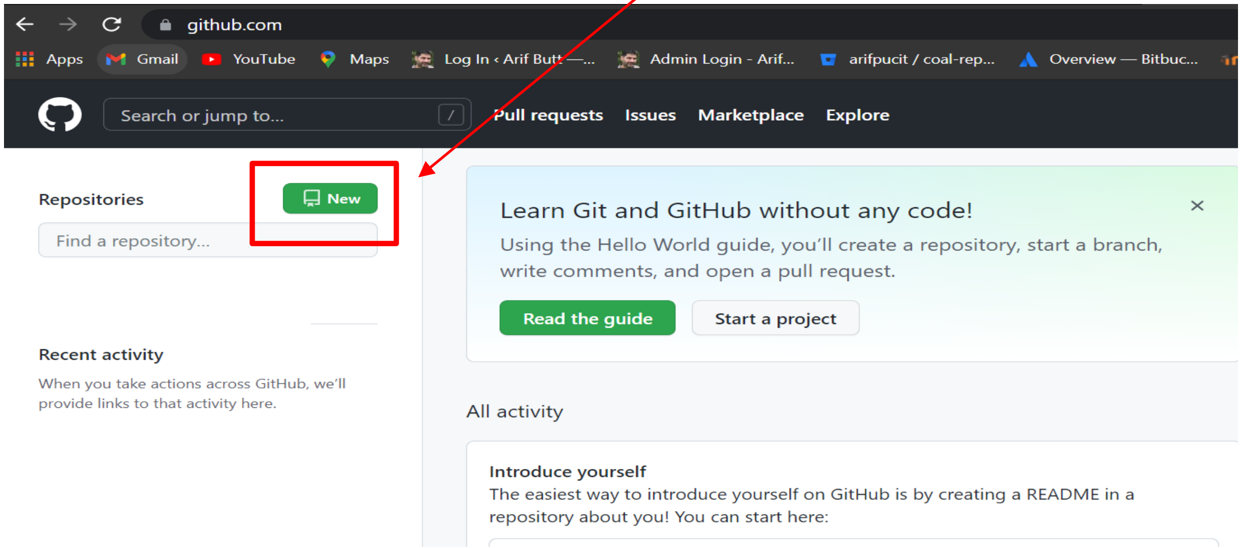
SP-VLecs | Public
Code Files for System Programming Video Lectures
● C

31 contributions in the last year | Contribution settings ▾

Instructor: Muhammad Arif Butt, Ph.D.

# Creating a Remote Repository on GitHub

Once you are logged in and are on the homepage, you will notice a button, that will let you to create your own Repository



Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Recent activity

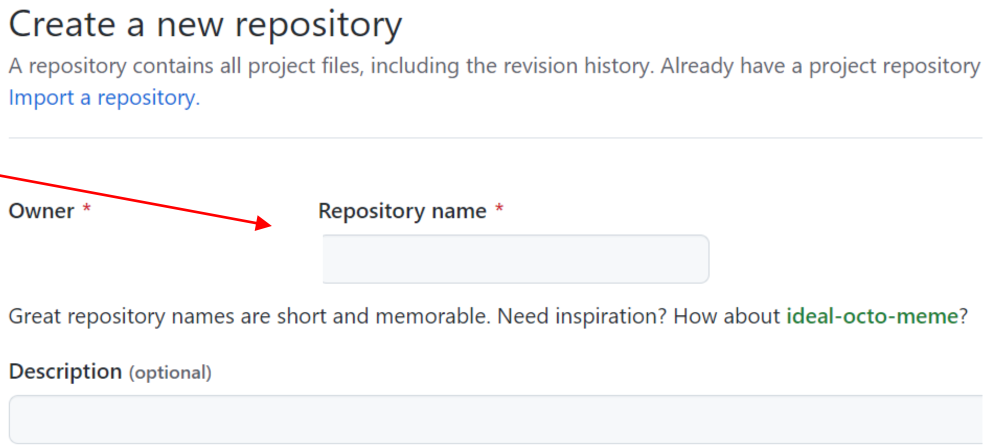
When you take actions across GitHub, we'll provide links to that activity here.

All activity

Introduce yourself

The easiest way to introduce yourself on GitHub is by creating a README in a repository about you! You can start here:

Once you click on the 'New' button, GitHub will redirect you to a different page where you will have to provide a name for the repository. Additionally, you can add a description of your repository.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository

Import a repository.

Owner *

Repository name *

Great repository names are short and memorable. Need inspiration? How about ideal-octo-meme?

Description (optional)

# Public & Private Repositories

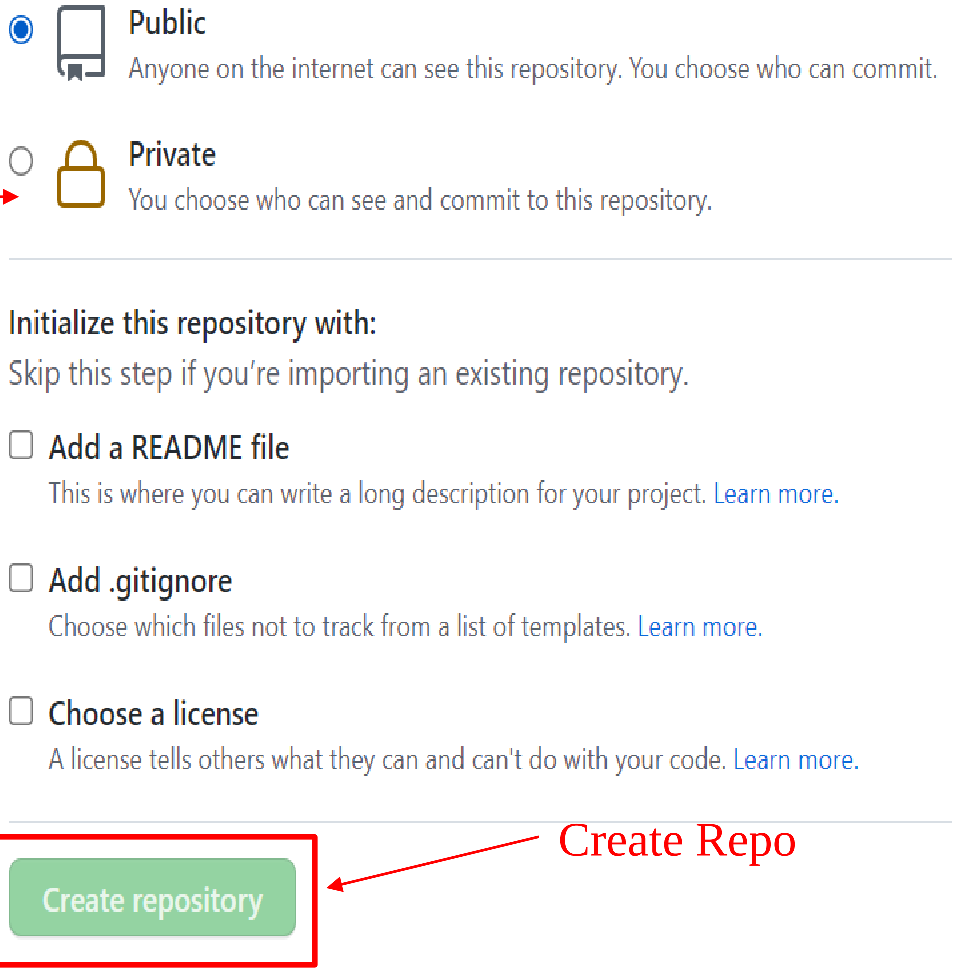
Besides providing a name and description, you need to choose whether you want your repository to be public or private.

*Public repository* is accessible to anyone. Anyone is able to see the codebase and clone this repository to their local machine for use.

*Private repository*, on the other hand, is only visible to people who you have chosen. No other person is able to view it.

Another decision you will have to make while creating a new repository is whether or not you'll create a *README* file.

Finally, you will be able to choose whether or not you want a *.gitignore* file. The purpose of the .gitignore file is to filter out files and subdirectories in your repository that you do not want Git to keep track of.



● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

□ Add a README file
This is where you can write a long description for your project. Learn more.

□ Add .gitignore
Choose which files not to track from a list of templates. Learn more.

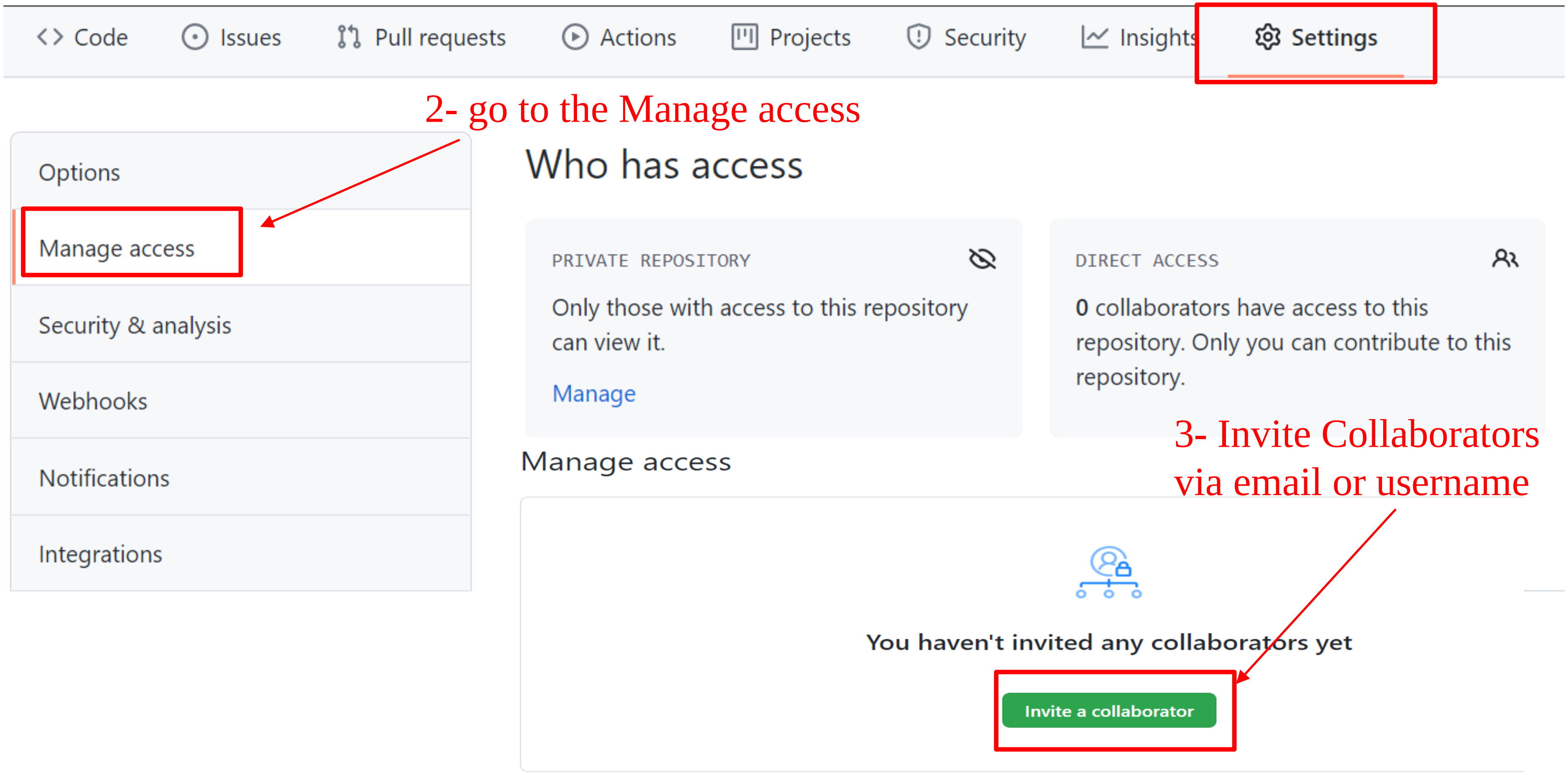
□ Choose a license
A license tells others what they can and can't do with your code. Learn more.

Create Repo

# Invite Collaborators

You can decide and manage, who can access your private repository and make collaboration.

1- After creating a private repo, click the settings tab



2- go to the Manage access

Who has access

PRIVATE REPOSITORY

Only those with access to this repository can view it.

Manage

DIRECT ACCESS

0 collaborators have access to this repository. Only you can contribute to this repository.

Manage access

You haven't invited any collaborators yet

3- Invite Collaborators via email or username

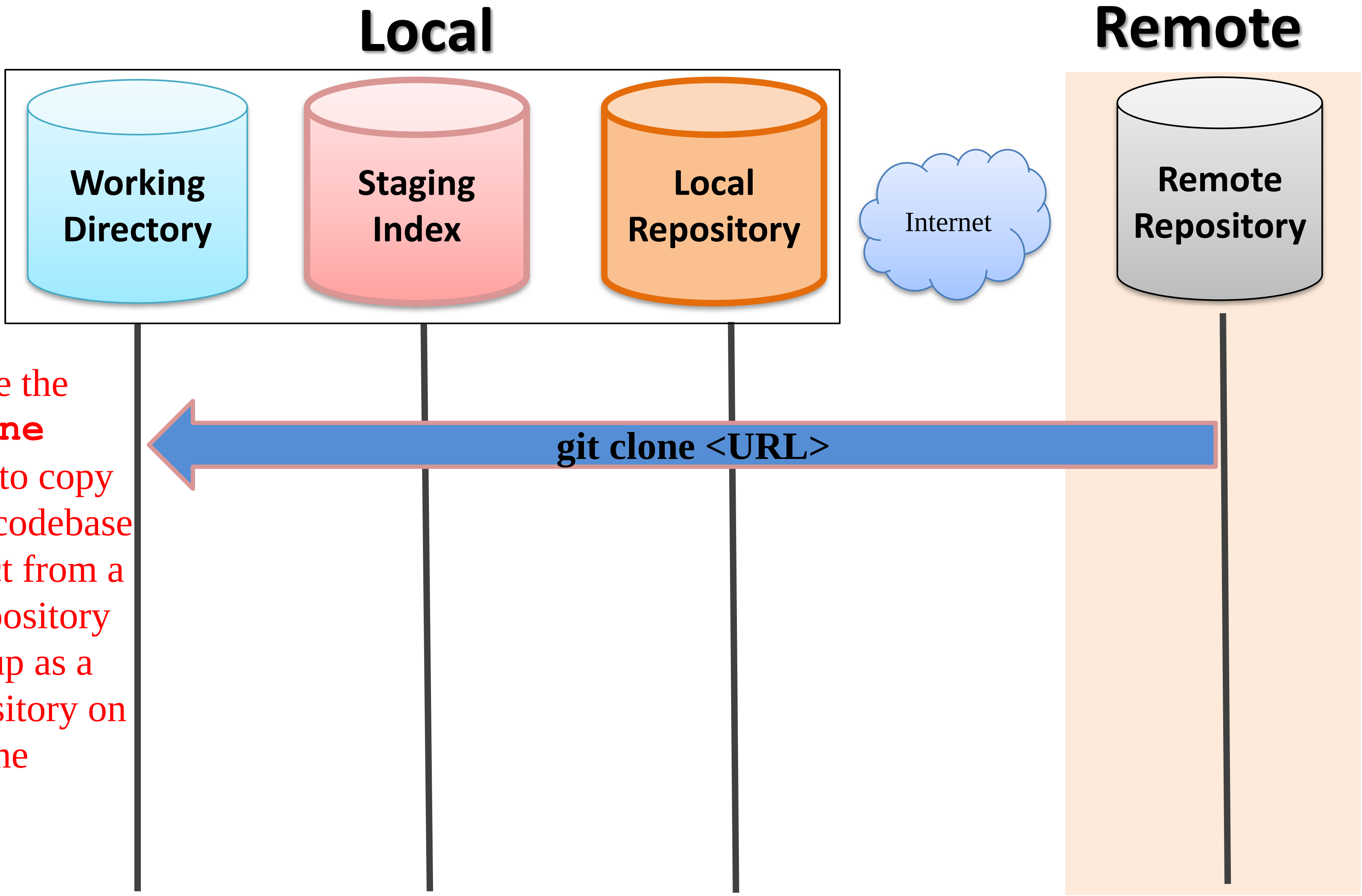
# Working with GitHub Repositories

**Demonstration**

[https://github.com/arifpucit](https://github.com/arifpucit)

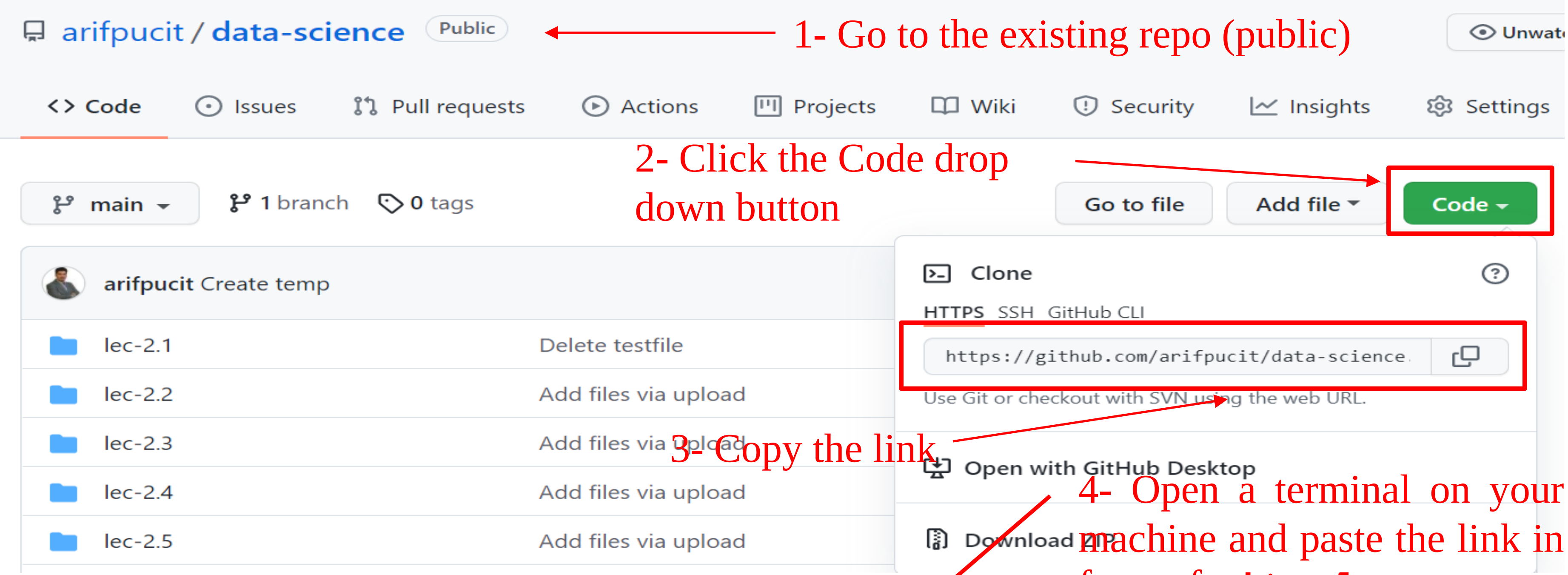
# Clone a Remote Repository

# Cloning Remote Repo to Local Repo



We can use the **`git clone`** command to copy the entire codebase of a project from a remote repository and set it up as a local repository on our machine

# Clone Remote Repo in Local Repo



arifpucit Create temp

| | |
|---|---|
| lec-2.1 | Delete testfile |
| lec-2.2 | Add files via upload |
| lec-2.3 | Add files via upload |
| lec-2.4 | Add files via upload |
| lec-2.5 | Add files via upload |

https://github.com/arifpucit/data-science.

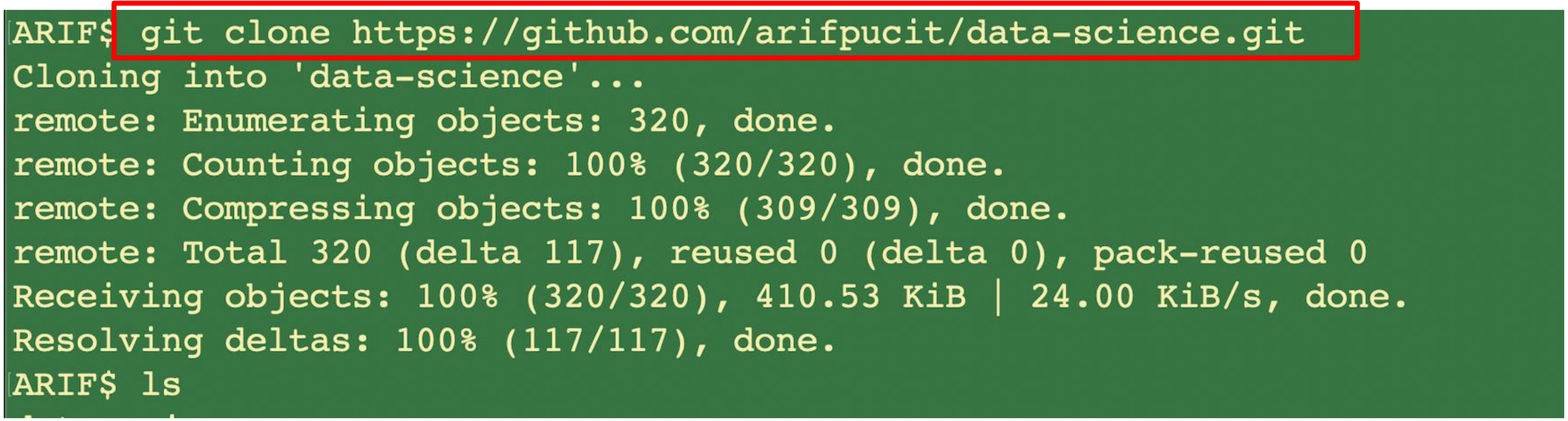
Use Git or checkout with SVN using the web URL.

1- Go to the existing repo (public)

2- Click the Code drop down button

3- Copy the link

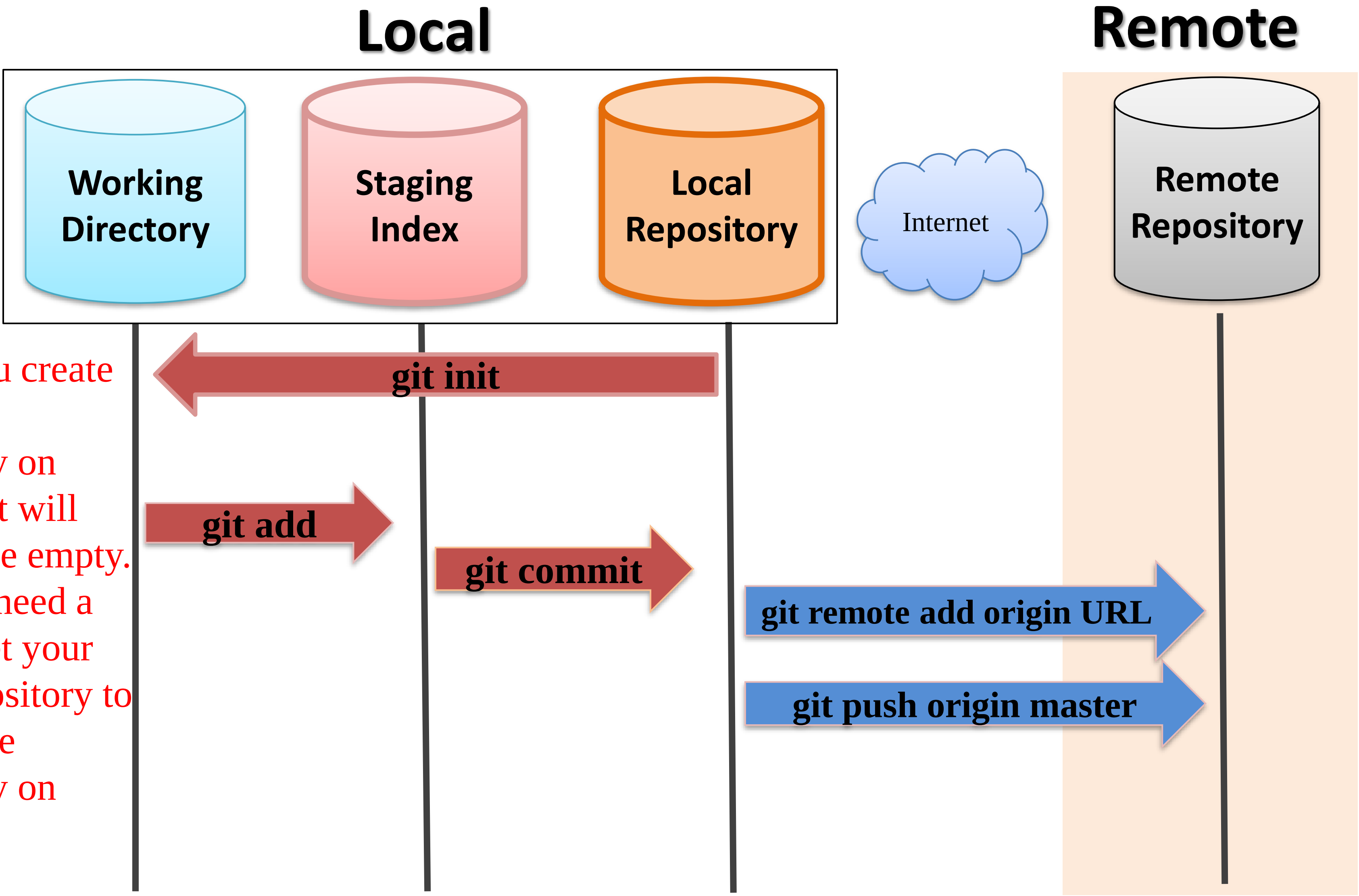
4- Open a terminal on your machine and paste the link in front of **`git clone`**



```
ARIF$ git clone https://github.com/arifpucit/data-science.git
Cloning into 'data-science'...
remote: Enumerating objects: 320, done.
remote: Counting objects: 100% (320/320), done.
remote: Compressing objects: 100% (309/309), done.
remote: Total 320 (delta 117), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (320/320), 410.53 KiB | 24.00 KiB/s, done.
Resolving deltas: 100% (117/117), done.
ARIF$ ls
```

# Push local Repo to Remote Repo

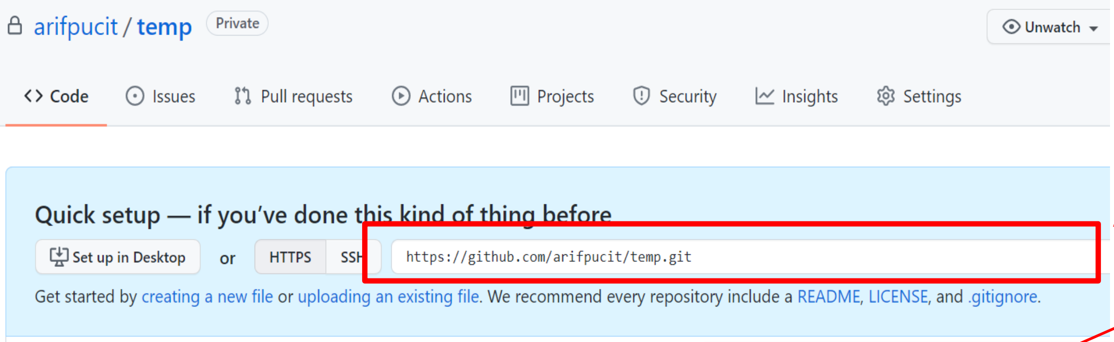
# Pushing a Local Repo to Remote Repo



When you create a remote repository on GitHub, it will initially be empty. You will need a way to get your local repository to the remote repository on GitHub

Instructor: Muhammad Arif Butt, Ph.D.

# Pushing a Local Repo to Remote Repo



arifpucit / temp Private

Unwatch

<> Code · Issues · Pull requests · Actions · Projects · Security · Insights · Settings

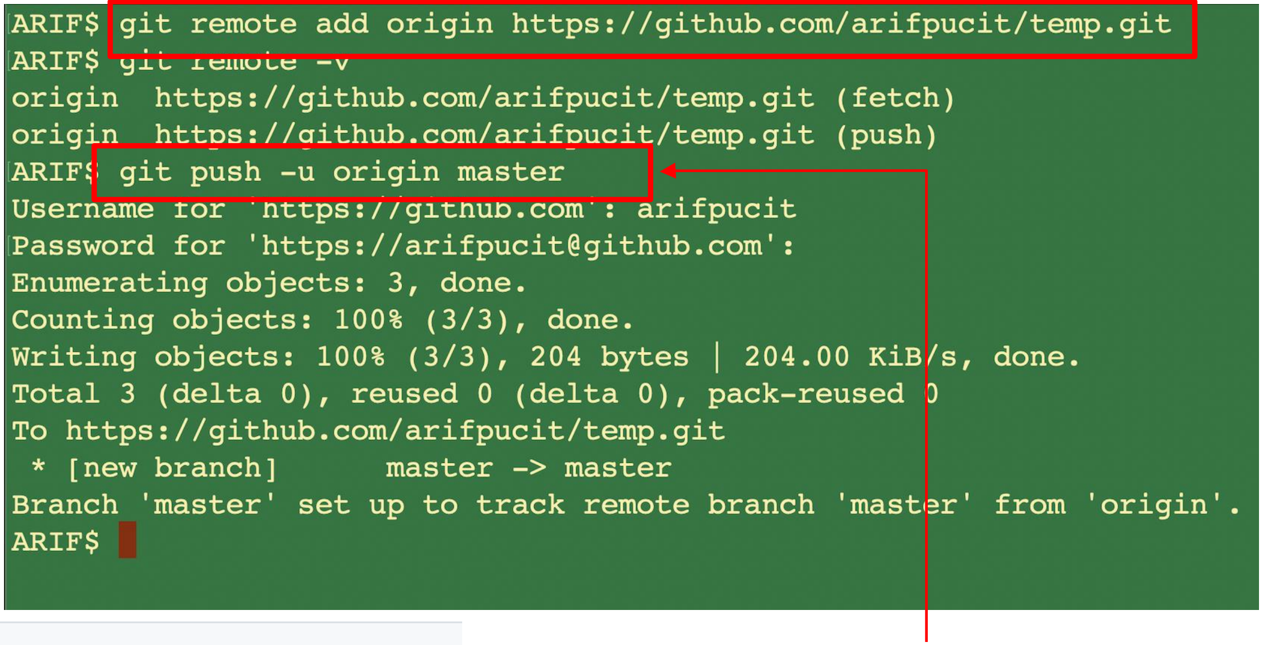
Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/arifpucit/temp.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

1- Copy URL of remote repo from gitHub

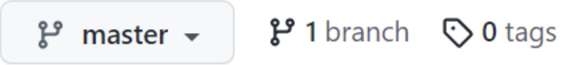
2- Connect local repository with remote repository



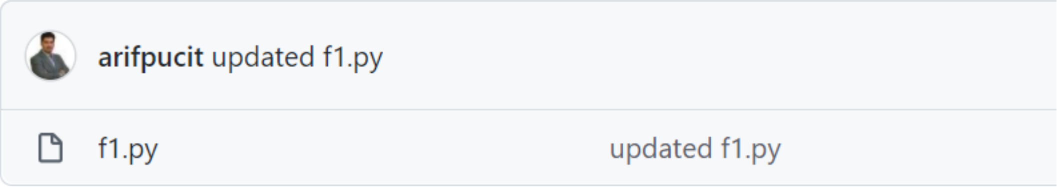
```
ARIF$ git remote add origin https://github.com/arifpucit/temp.git
ARIF$ git remote -v
origin  https://github.com/arifpucit/temp.git (fetch)
origin  https://github.com/arifpucit/temp.git (push)
ARIF$ git push -u origin master
Username for 'https://github.com': arifpucit
Password for 'https://arifpucit@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 204 bytes | 204.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/arifpucit/temp.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
ARIF$
```

3 – Upload local code and its revision history to the remote repo

4 – Verify that local repo has been pushed on Remote Repo



master · 1 branch · 0 tags



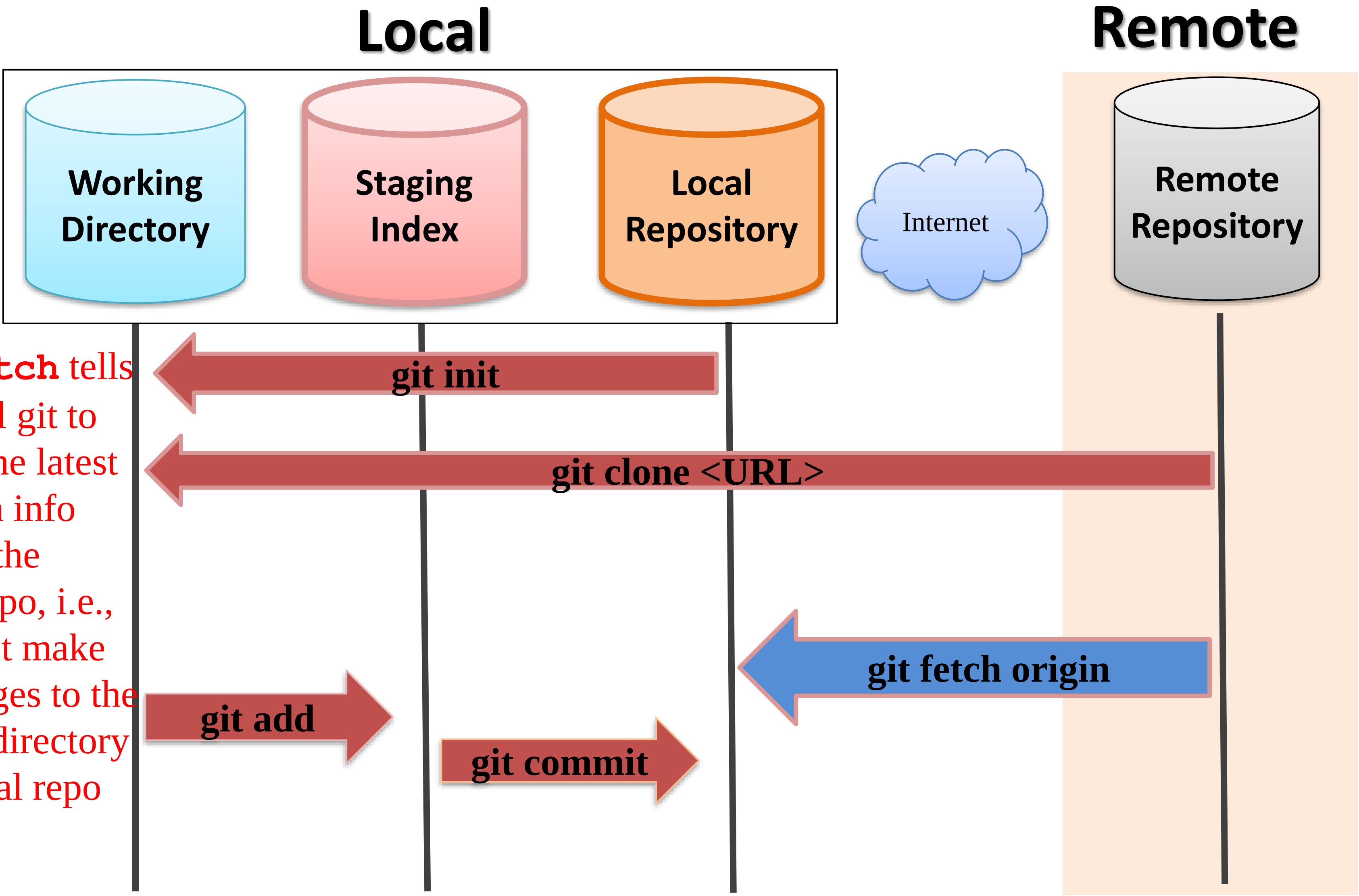
arifpucit updated f1.py

f1.py    updated f1.py

Instructor: Muhammad Arif Butt, Ph.D.

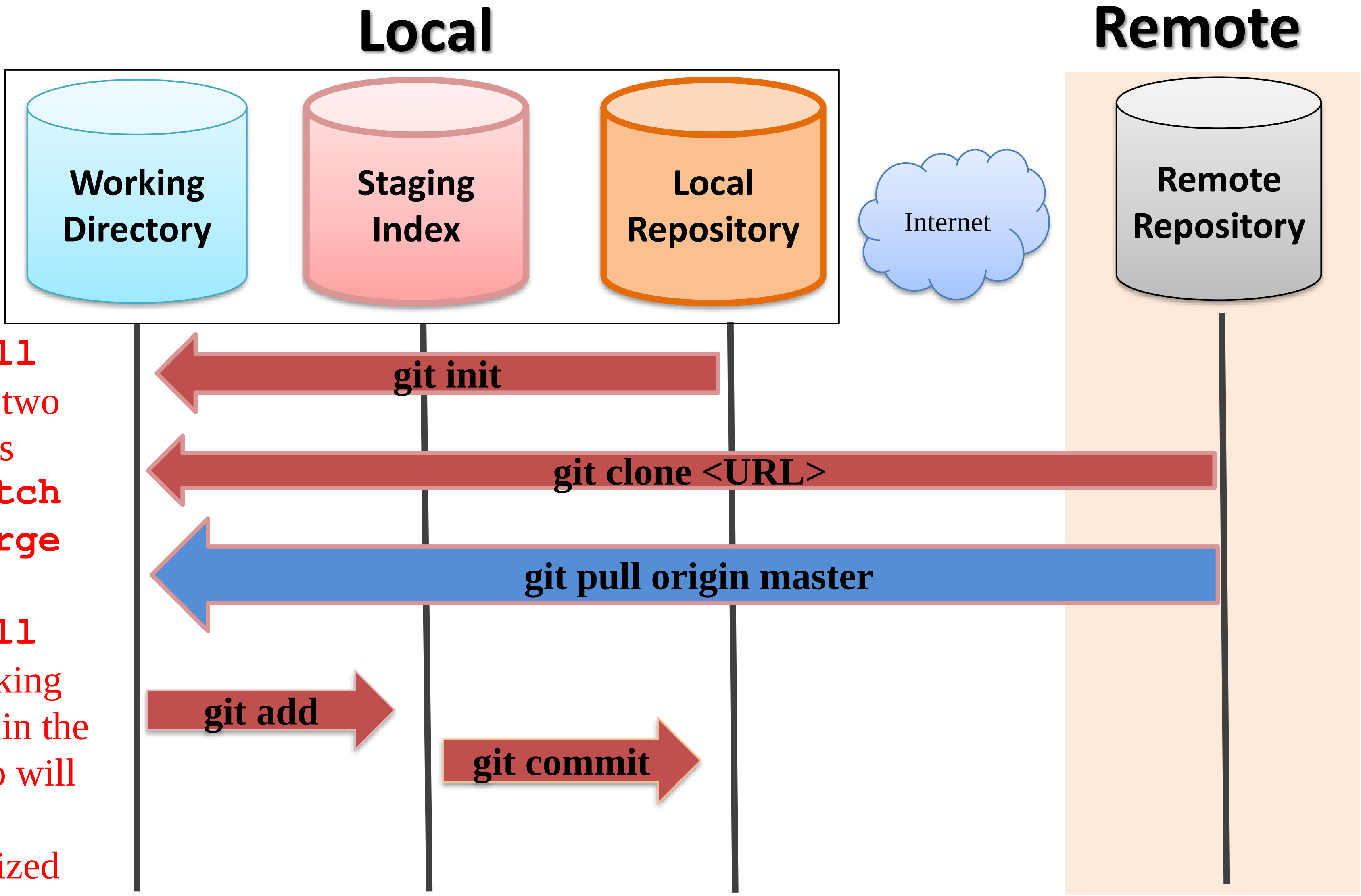
# Fetch vs Pull

# Git Fetch



**`git fetch`** tells your local git to retrieve the latest meta-data info from the the remote repo, i.e., it does not make any changes to the working directory in the local repo

# Git Pull

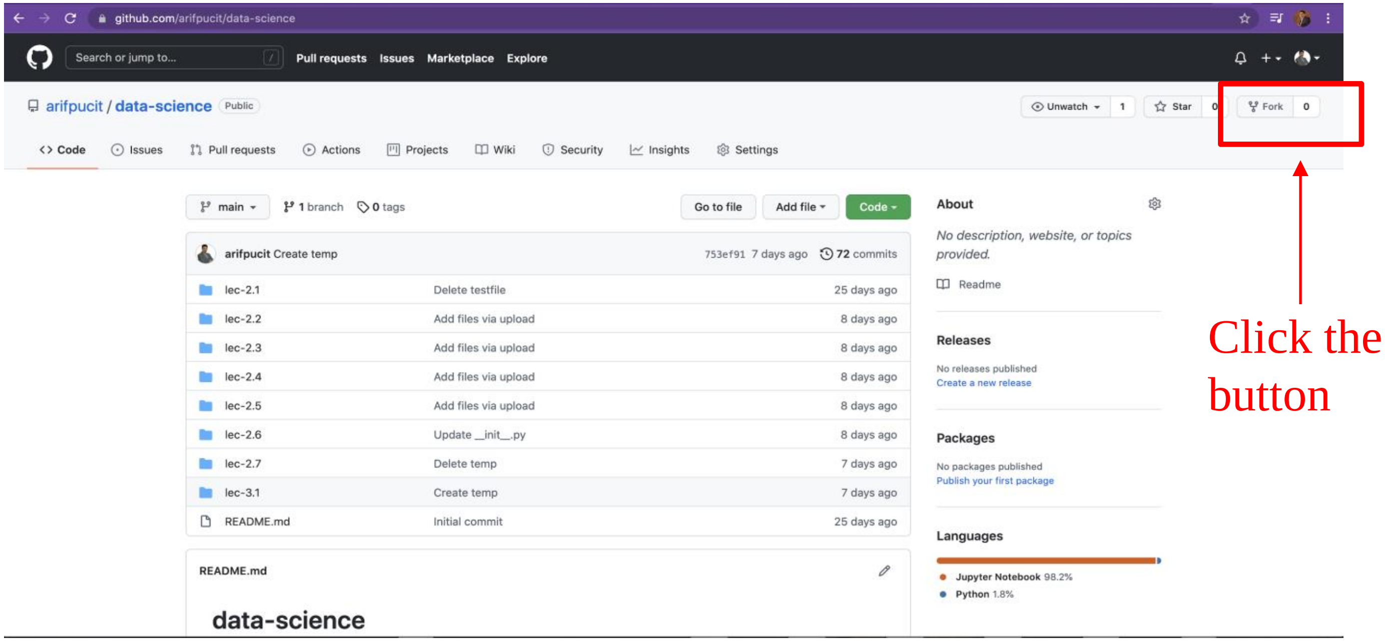


**git pull** performs two operations **git fetch** **git merge** So after a **git pull** your working directory in the local repo will also be synchronized with the remote repo

# Clone vs Fork

# Fork a Repository from GitHub

- Forking means creating a copy of complete repo from some one else’s GitHub account on your GitHub account. You can do this to collaborate on a open source project, or use the existing state of the project as a starting point for your own project
  - ✓ On GitHub navigate to someone’s repository that you want to fork, and click the Fork button, then check the repository availability on your GitHub account.
  - ✓ Clone this repo on your local machine, make a new branch, fix a bug, add/enhance a functionality, and then push it back to your own remote repo
  - ✓ Finally click pull request to open a new pull request to the actual project owner



arifpucit Create temp 753ef91 7 days ago 72 commits

| | | |
|---|---|---|
| lec-2.1 | Delete testfile | 25 days ago |
| lec-2.2 | Add files via upload | 8 days ago |
| lec-2.3 | Add files via upload | 8 days ago |
| lec-2.4 | Add files via upload | 8 days ago |
| lec-2.5 | Add files via upload | 8 days ago |
| lec-2.6 | Update __init__.py | 8 days ago |
| lec-2.7 | Delete temp | 7 days ago |
| lec-3.1 | Create temp | 7 days ago |
| README.md | Initial commit | 25 days ago |

*No description, website, or topics provided.*

No releases published
Create a new release

No packages published
Publish your first package

- Jupyter Notebook 98.2%
- Python 1.8%

Click the fork button

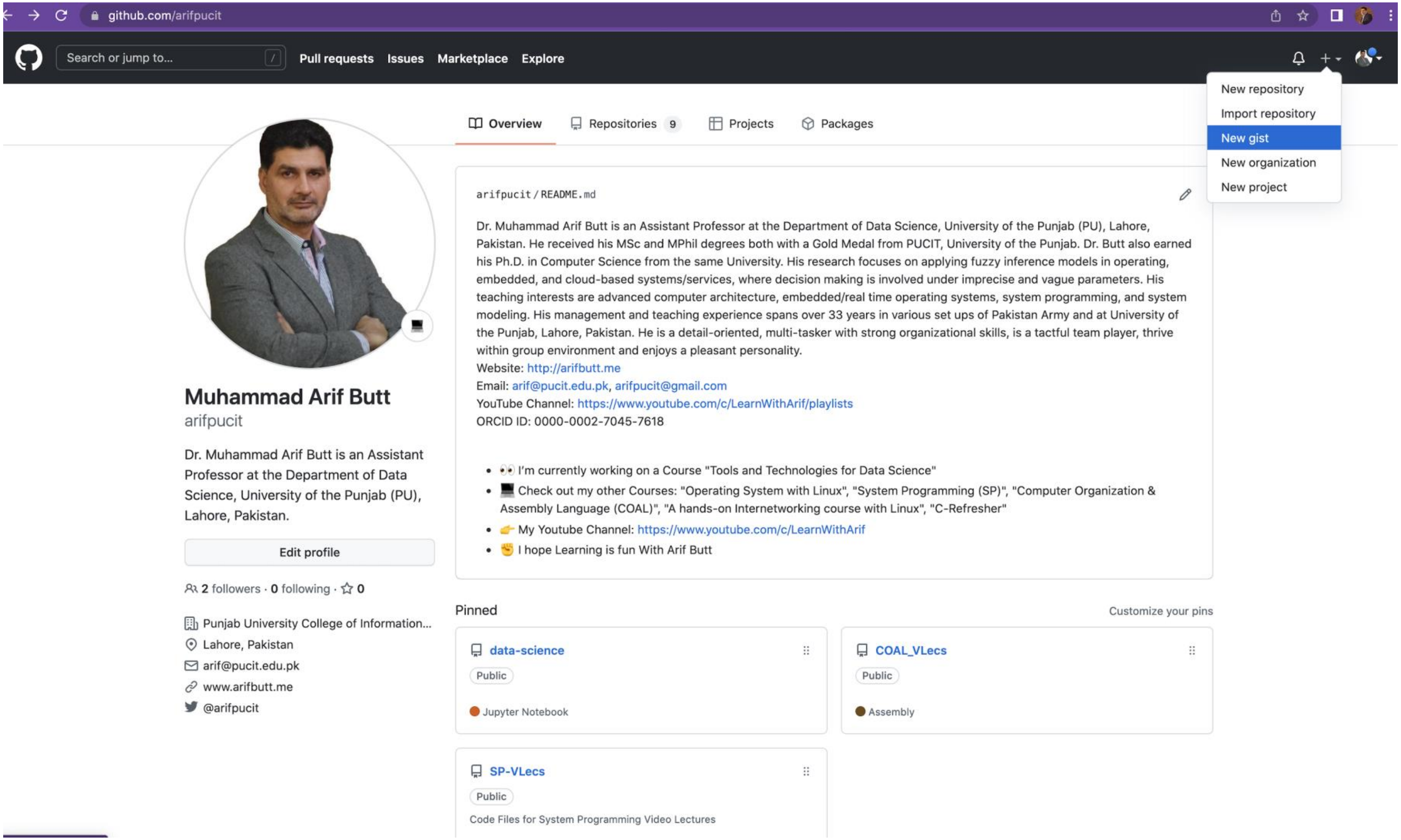
# Collaborating with Open Source Projects

**Demonstration**

***https://github.com/arifpucit/data-science.git***

# GitHub Gists

# Overview of Revision/Version Control System



arifpucit / README.md

Dr. Muhammad Arif Butt is an Assistant Professor at the Department of Data Science, University of the Punjab (PU), Lahore, Pakistan. He received his MSc and MPhil degrees both with a Gold Medal from PUCIT, University of the Punjab. Dr. Butt also earned his Ph.D. in Computer Science from the same University. His research focuses on applying fuzzy inference models in operating, embedded, and cloud-based systems/services, where decision making is involved under imprecise and vague parameters. His teaching interests are advanced computer architecture, embedded/real time operating systems, system programming, and system modeling. His management and teaching experience spans over 33 years in various set ups of Pakistan Army and at University of the Punjab, Lahore, Pakistan. He is a detail-oriented, multi-tasker with strong organizational skills, is a tactful team player, thrive within group environment and enjoys a pleasant personality.
Website: http://arifbutt.me
Email: arif@pucit.edu.pk, arifpucit@gmail.com
YouTube Channel: https://www.youtube.com/c/LearnWithArif/playlists
ORCID ID: 0000-0002-7045-7618

- 👀 I'm currently working on a Course "Tools and Technologies for Data Science"
- 💻 Check out my other Courses: "Operating System with Linux", "System Programming (SP)", "Computer Organization & Assembly Language (COAL)", "A hands-on Internetworking course with Linux", "C-Refresher"
- 👉 My Youtube Channel: https://www.youtube.com/c/LearnWithArif
- ✊ I hope Learning is fun With Arif Butt

**Muhammad Arif Butt**
arifpucit

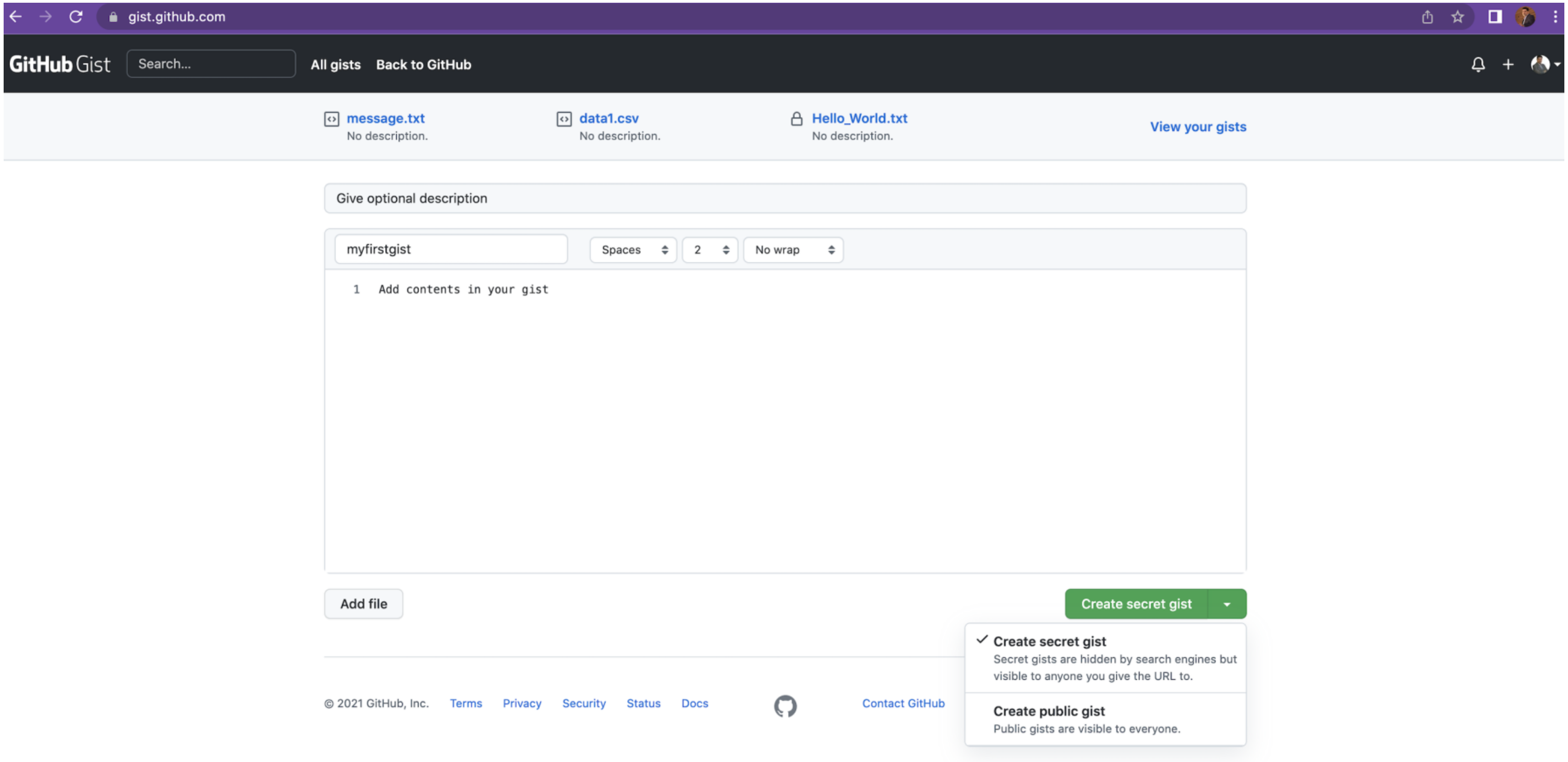
Dr. Muhammad Arif Butt is an Assistant Professor at the Department of Data Science, University of the Punjab (PU), Lahore, Pakistan.

2 followers · 0 following · ☆ 0

Punjab University College of Information...
Lahore, Pakistan
arif@pucit.edu.pk
www.arifbutt.me
@arifpucit

| | |
|---|---|
| data-science<br>Public<br>● Jupyter Notebook | COAL_VLecs<br>Public<br>● Assembly |
| SP-VLecs<br>Public<br>Code Files for System Programming Video Lectures | |

# Overview of Revision/Version Control System



gist.github.com

GitHub Gist | Search... | All gists | Back to GitHub

message.txt
No description.

data1.csv
No description.

Hello_World.txt
No description.

View your gists

Give optional description

myfirstgist | Spaces | 2 | No wrap

```
Add contents in your gist
```

Add file

Create secret gist

✓ **Create secret gist**
Secret gists are hidden by search engines but visible to anyone you give the URL to.

**Create public gist**
Public gists are visible to everyone.

© 2021 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub

# Things To Do

- Install git on your machine and practice working on a local repository by performing lots of commits, create branches and merge them
- Create your GitHub account using your RollNo and official email ID
- Create a private repository and share it with myself and your friends

- Create a local repository on your machine and do lot of commits on it. Create an empty remote repository on your GitHub account. Finally push your local repository on GitHub repository.
- Clone https://github.com/arifpucit/data-science repository, make improvements in it and see if you can push/submit those changes to this public repository of mine
- Fork https://github.com/arifpucit/data-science repository, clone it, fix any bugs or improve documentation and submit a pull request to the repository owner

**Coming to office hours does NOT mean you are academically weak!**