OSC

Get To Git

Session 1

Amna Ahmed

Agenda

01

Introduction To Open Source

What is Open Source?

What is Open Source?

There are many definitions, but we will pick the **FOSS** definition!

The 4 freedoms of FOSS

01

Freedom to run

the program as you wish, for any purpose.

02

Freedom to study

how the program works, and modify it.

03

Freedom to distribute

copies so you can help others.

04

Freedom to distribute modified

copies of the program.

The 4 freedoms of FOSS

01

Freedom to run

the program as you wish, for any purpose.

02

Freedom to study

how the program works, and modify it.

03

Freedom to distribute

copies so you can help others.

04

Freedom to distribute modified

copies of the program.

Access to the source code is a precondition for these.

Why is FOSS great?

01	**Customizability:** You are free to tailor the software to your own needs, and redistribute it if you wish.
02	**Security and privacy:** Involvement of many people exposes bugs and security threats more quickly.
03	**Quality and efficiency:** Collaboration and sharing source code leads to higher quality, robust software.
04	**Efficiency:** Open-source software minimizes wasted effort "reinventing the wheel".

02

Contributing To Open Source

Why Should You Contribute?

It teaches you real skills...

- Navigating a code base
- Problem solving
- Working with others
- Documenting your work

It improves the tools you use

- Add a feature
- FIx a bug
- Improve documentation

Why Should You Contribute?

It builds your reputation...

- Showcases your work
- Networking opportunity
- Builds credibility and trust

It's fun!

- A chance to leave your mark on technology
- Getting a contribution accepted is very rewarding

I'm in!

But **how** can I contribute?

You need 3 main things to contribute

Relevant Skills

Depending on the project and contribution, you might need technical skills in the language/ framework.

Opportunity

Projects to contribute to. This could be any open source project, or through an organized event.

Git Skills

This will allow you to make changes and upload them to be reviewed in a systematic way.

You need 3 main things to contribute

Relevant Skills

Depending on the project and contribution, you might need technical skills in the language/ framework.

Opportunity

Projects to contribute to. This could be any open source project, or through an organized event.

Git Skills

This will allow you to make changes and upload them to be reviewed in a systematic way.

Contribution Opportunities

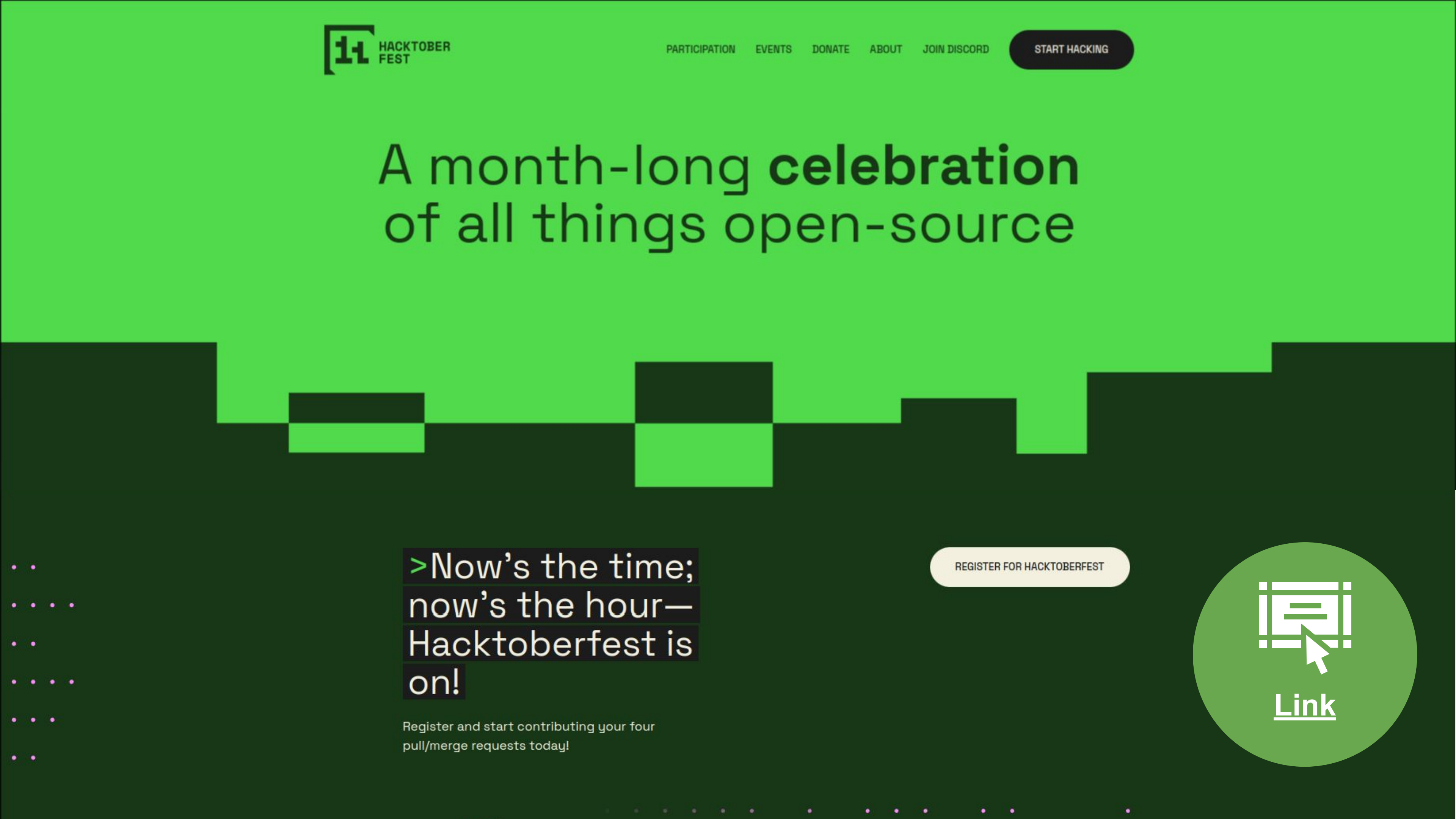
HACKTOBER FEST

Google Summer of Code

Hacktoberfest

Hacktoberfest is an annual event that encourages people to contribute to open source throughout October.

After all, open source is driven by contributors, just like you!



HACKTOBER FEST

PARTICIPATION EVENTS DONATE ABOUT JOIN DISCORD START HACKING

A month-long **celebration** of all things open-source

>Now's the time; now's the hour—Hacktoberfest is on!

REGISTER FOR HACKTOBERFEST

Register and start contributing your four pull/merge requests today!

Link

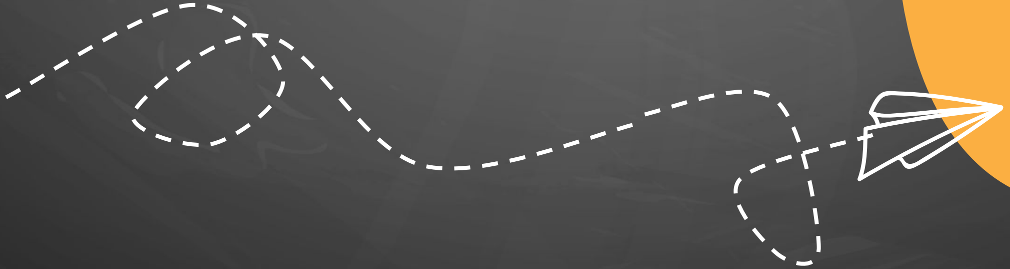
Participating in Hacktoberfest

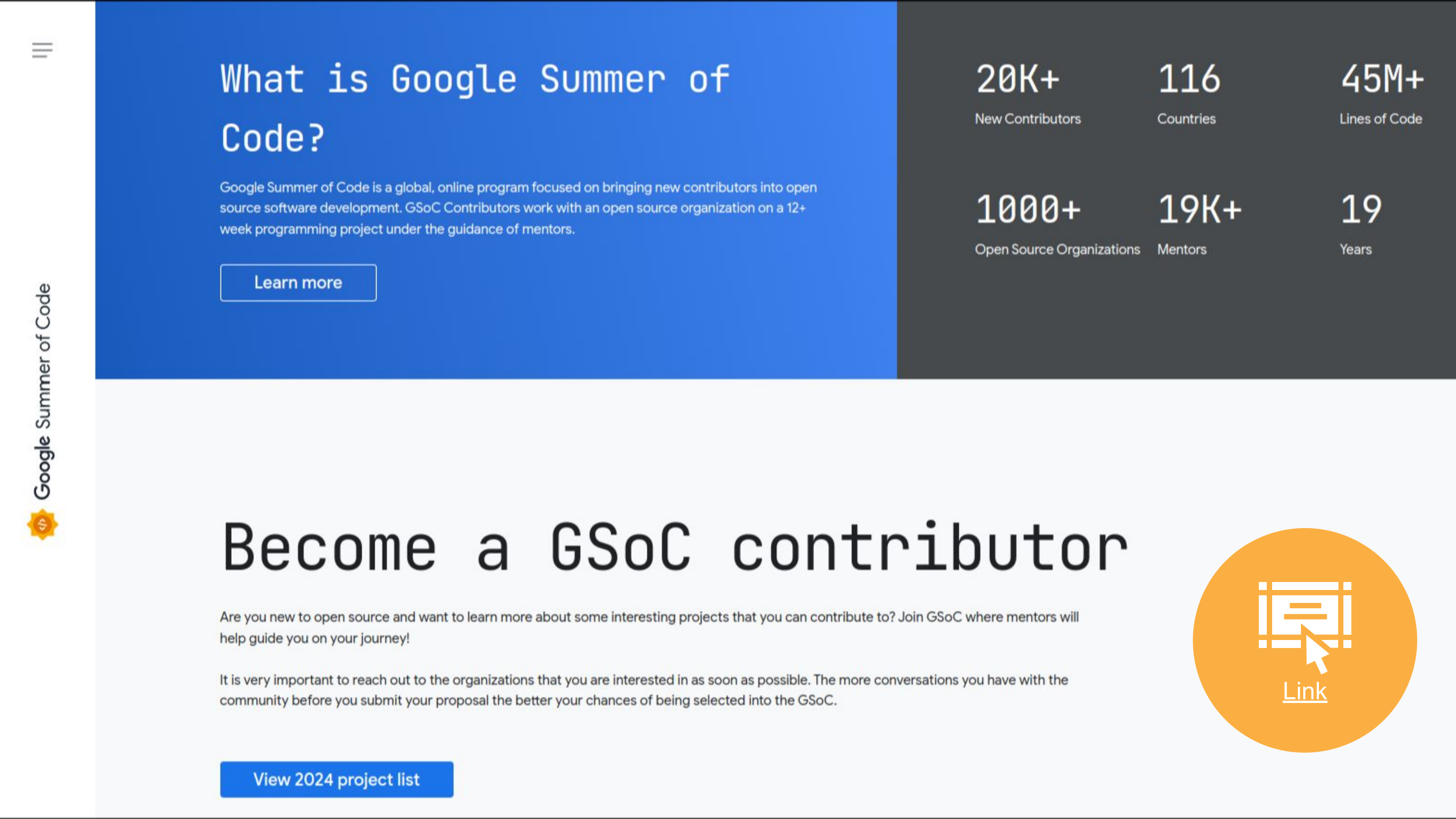
- Register here. This will grant you a digital badge.
- Explore organizations that are marked with "hacktoberfest." You can search here :
  Finder, GitHub, contribhub
- Aim to submit four high-quality pull requests, with project maintainers accepting your pull/merge requests for them to count toward your total.

Google Summer of Code (GSoC)

GSoC is a global, online program focused on bringing new contributors into open source software development.

GSoC is a great chance for students to get involved in open source under the guidance of a large organization.





What is Google Summer of Code?

Google Summer of Code is a global, online program focused on bringing new contributors into open source software development. GSoC Contributors work with an open source organization on a 12+ week programming project under the guidance of mentors.

Learn more

20K+ New Contributors	116 Countries	45M+ Lines of Code
1000+ Open Source Organizations	19K+ Mentors	19 Years

Become a GSoC contributor

Are you new to open source and want to learn more about some interesting projects that you can contribute to? Join GSoC where mentors will help guide you on your journey!

It is very important to reach out to the organizations that you are interested in as soon as possible. The more conversations you have with the community before you submit your proposal the better your chances of being selected into the GSoC.

View 2024 project list

Link

Get involved in Open Source

Events are not the only way to contribute to open source!

You can find good first issues here and here.

You can also checkout the issues section in any open source project.

03

Bash

What is Bash?

Bash is a shell program, which is a type of program that allows you to interact with your operating system via a command-line interface.

We will mainly learn Git through the command line, so it will help to have some basic Bash knowledge.

Basic Bash Commands

pwd
Prints the current directory.

ls
List the contents of the current directory.

cd
Go to the given directory.

mkdir
Create a directory with the given name.

touch
Create a file with the given name.

cat
Print the contents of the given file.

rm [-r]
Remove the given file or directory (-r).

cp [-r]
Copy the given file or directory (-r).

echo
Print the given text.

Hands-On #1

Hands-On #1

- Create a directory named "OSC" and move into it.
- Create two files, “slides.txt” and “notes.txt”, then add text to them.
- Get out of “OSC” directory and and create another directory called "Backup".
- Copy the entire "OSC" **directory** into the "Backup" directory.

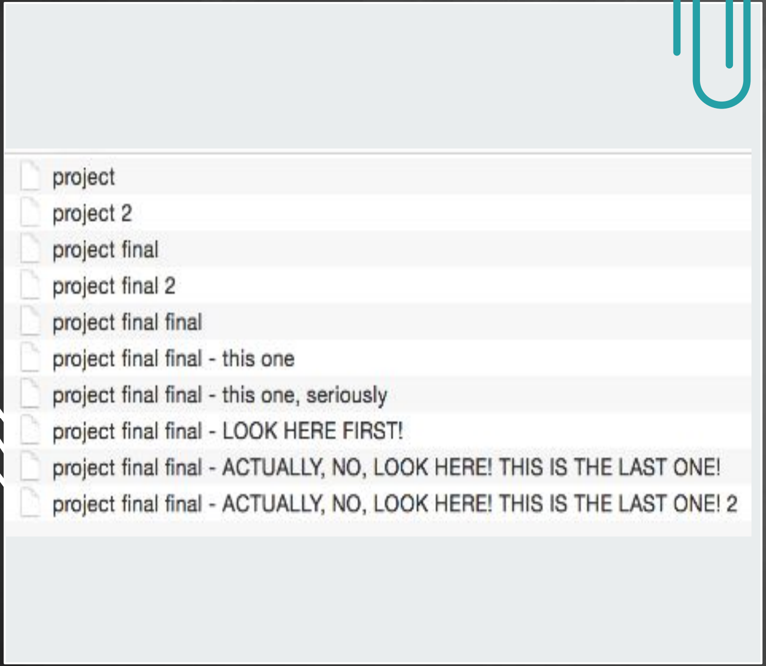
Hands-On #1

- Navigate to the "Backup/OSC" directory, list its contents, and verify the copied files.
- Move back to the main directory and delete the original "**OSC**" folder. Make sure to leave the one inside "Backup".

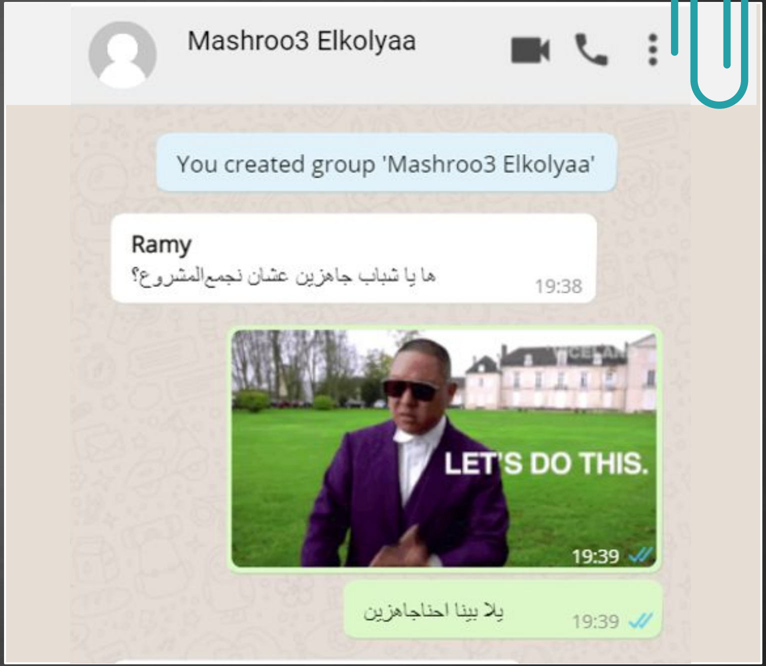
04

Version Control Systems

Life Before Version Control



project
project 2
project final
project final 2
project final final
project final final - this one
project final final - this one, seriously
project final final - LOOK HERE FIRST!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE!
project final final - ACTUALLY, NO, LOOK HERE! THIS IS THE LAST ONE! 2



Mashroo3 Elkolyaa

You created group 'Mashroo3 Elkolyaa'

Ramy
ها يا شباب جاهزين عشان نجمعالمشروع؟ 19:38

19:39

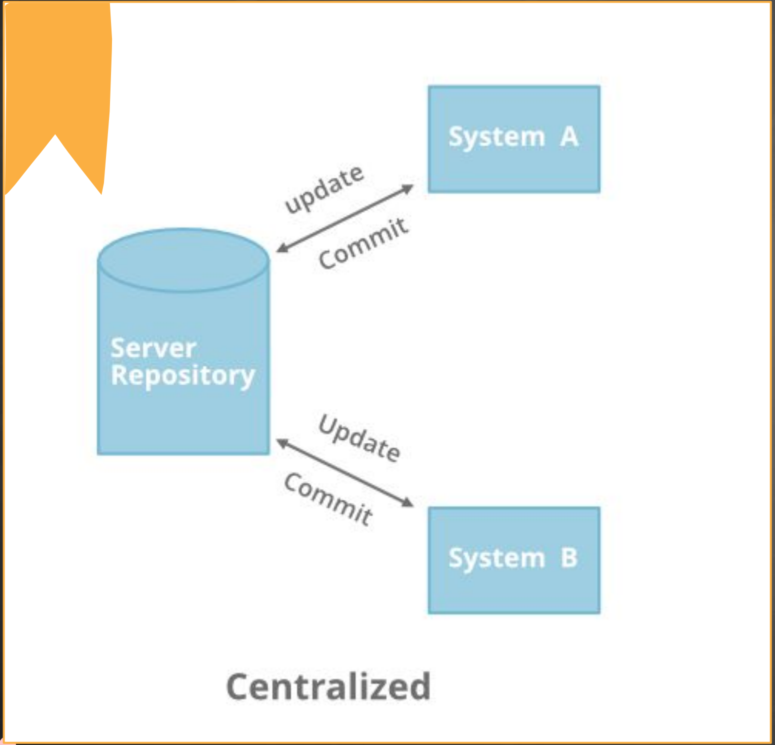
يلا بينا احناجاهزين 19:39

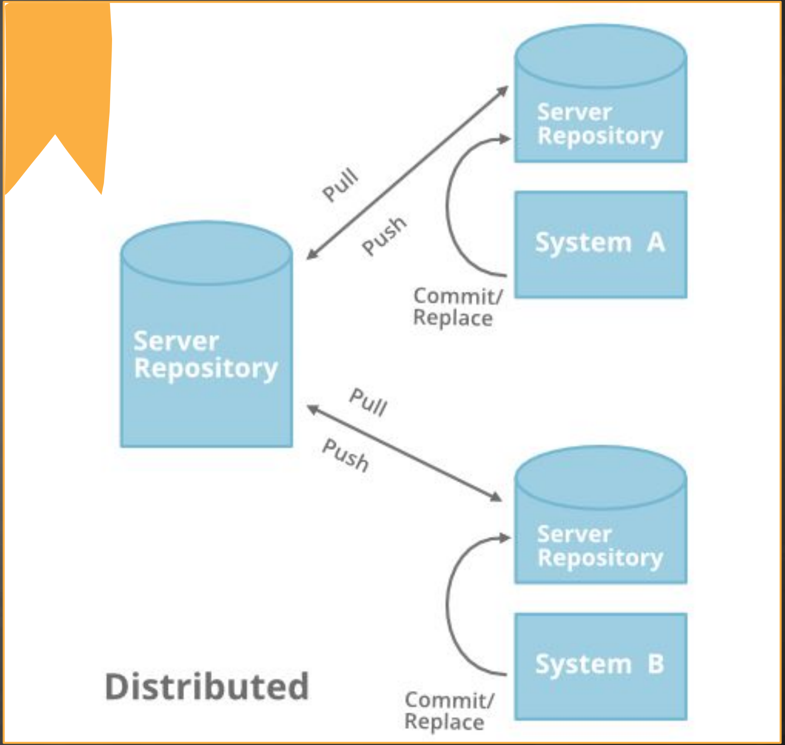
What is a Version Control System (VCS)?

Version control is the software engineering practice of managing and tracking different versions of computer files; primarily, source code text files.

A **version control system** is a software tool that automates version control.

Types of VCSs





Types of VCSs

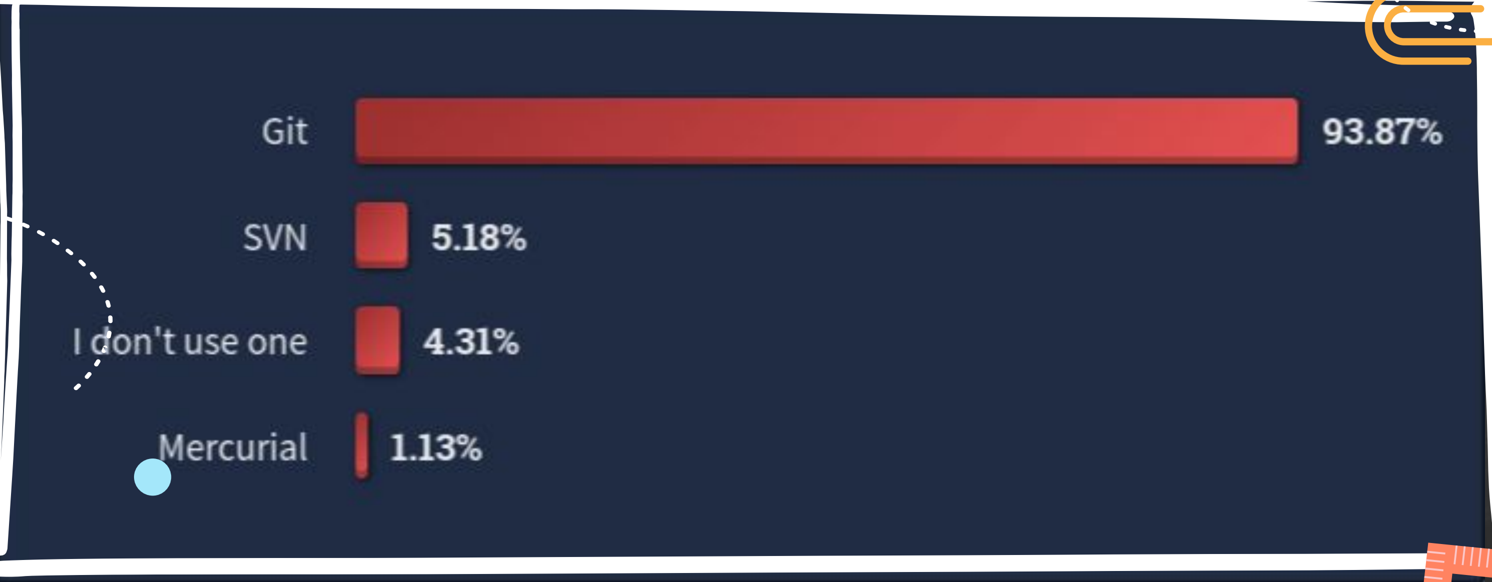
Centralized

- Client-server model
- Single repository
- Users must be online to make any change
- Common operations are slower
- Example: Subversion

Distributed

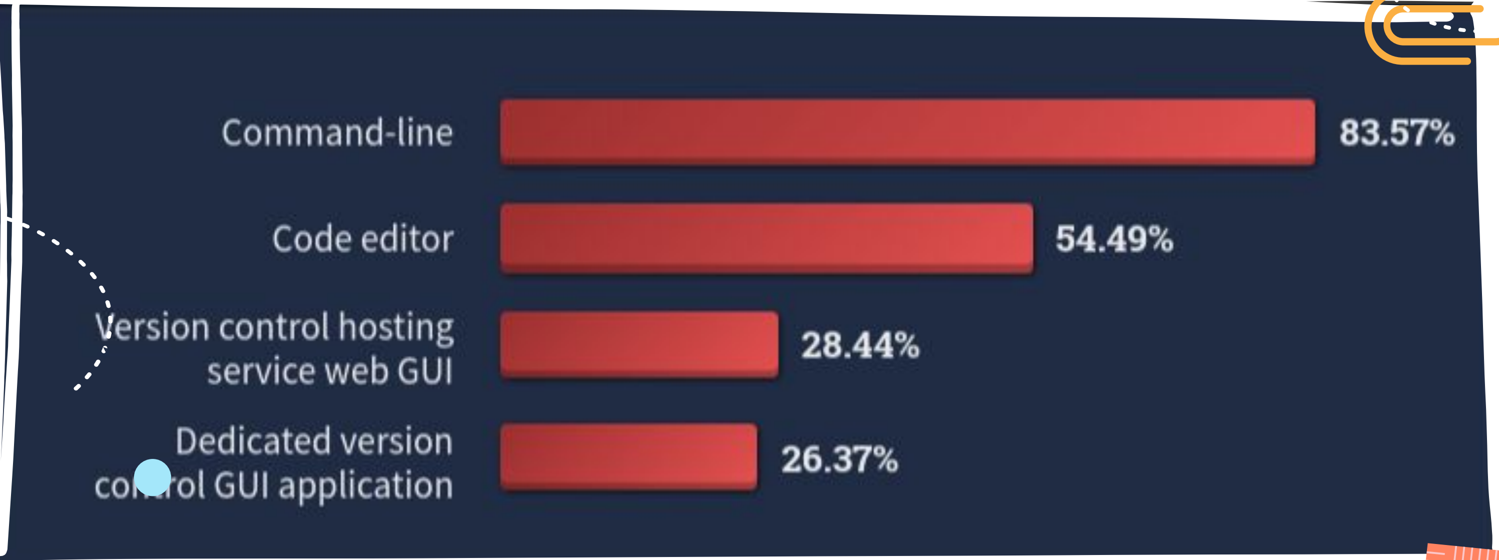
- Peer-to-peer model
- Multiple repositories
- Users can work offline until they need to publish their changes to others
- Common operations are faster
- Examples: Git, Mercurial

The Primary VCSs Used By Developers*



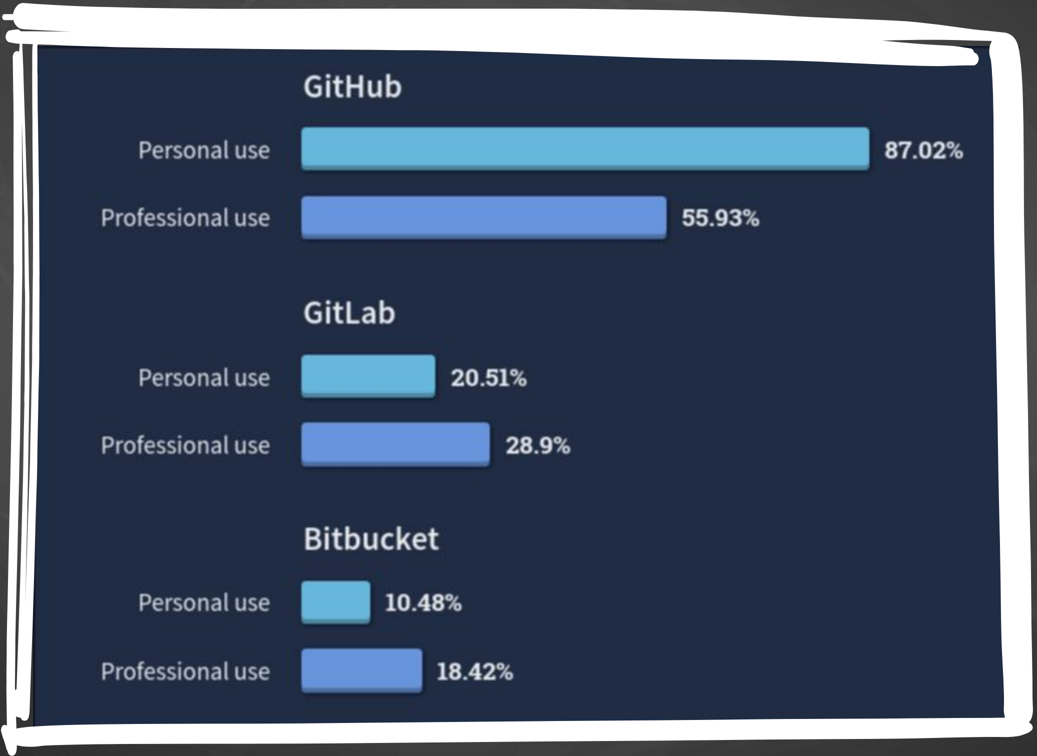
* Stack Overflow 2022 Developer Survey

How Developers Interact With VCSs*



* Stack Overflow 2022 Developer Survey

Most Used Version Control Platforms*



* Stack Overflow 2022 Developer Survey

05

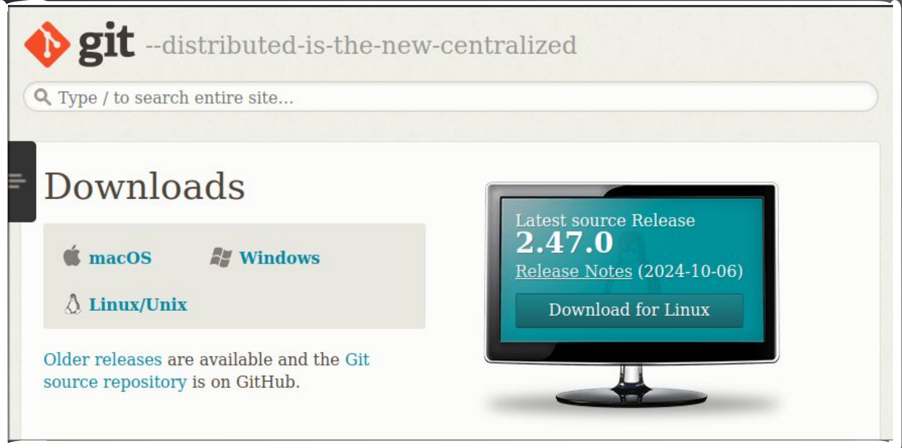
Getting Started With Git

What is Git?

Git is a **FOSS**, **distributed** version control system designed to handle everything from small to very large projects with speed and efficiency. More about it

Git was created by Linus Torvalds in 2005 for the development of the Linux kernel, with other kernel developers contributing to its initial development.

Installing and Configuring Git


git --distributed-is-the-new-centralized

Type / to search entire site...

Downloads

Older releases are available and the Git source repository is on GitHub.

Installation

Follow this link and choose your platform, then download the latest version of Git.

Configuration

After installing Git, run the following commands.

```
git config --global user.name <your name>

git config --global user.email <your email>
```

Git Components

Git Components

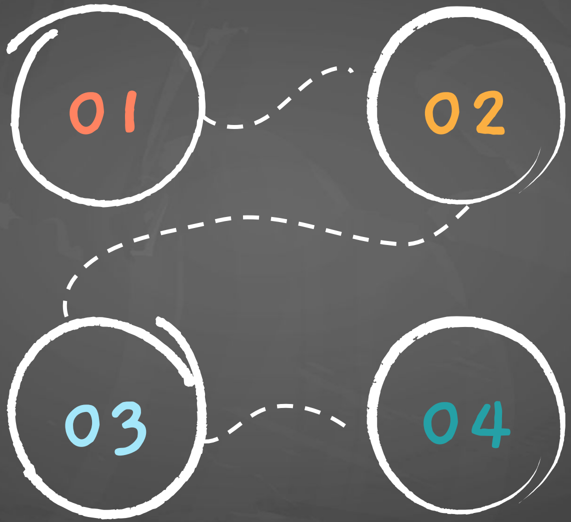
Working Directory

Staging Area (Index)

Commit History

Branches

Next session!



Working Directory

It is the place where you actually edit files, compile code, and develop your project.

You can treat the working directory as a normal folder, except that you now have access to Git commands that can manage the content of that folder.

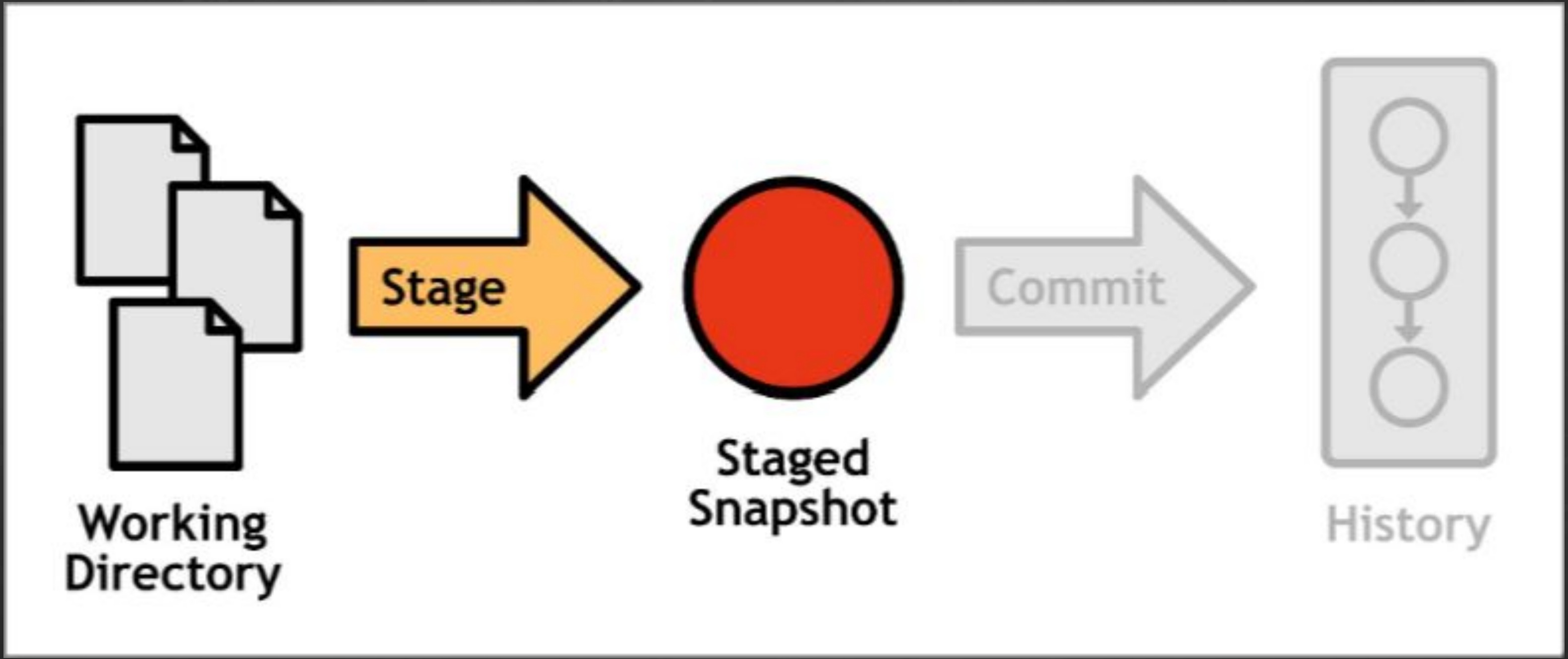
Staging Area (Index)

It is an intermediary point between the working directory and the project history.

Instead of forcing you to commit all changes at once, Git lets you group them into related sets of changes.

Note: staged changes are not yet part of your history.

Staging Area (Index)

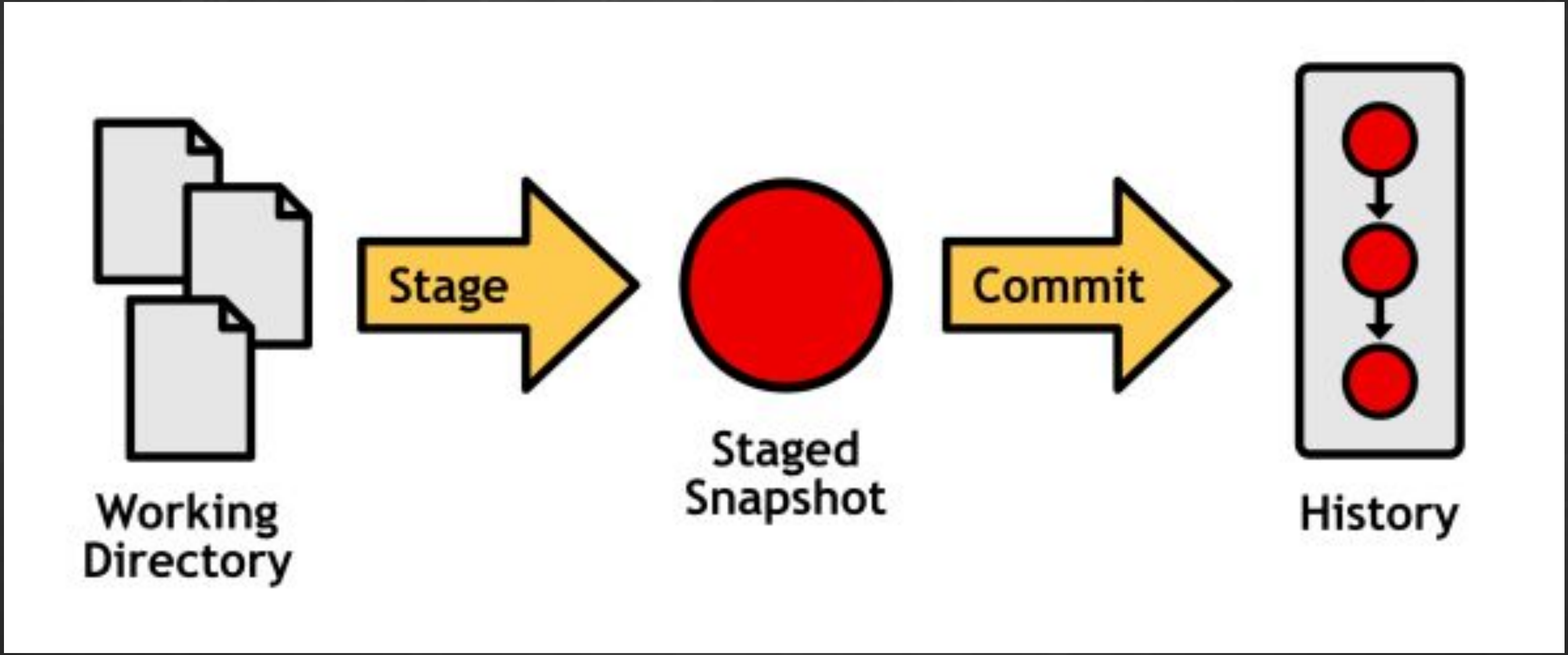


Commit History

This contains changes that have been recognized by Git as checkpoints in the project history.

These changes were first added to the staging area and then **committed** .

Commit History



Working With Git

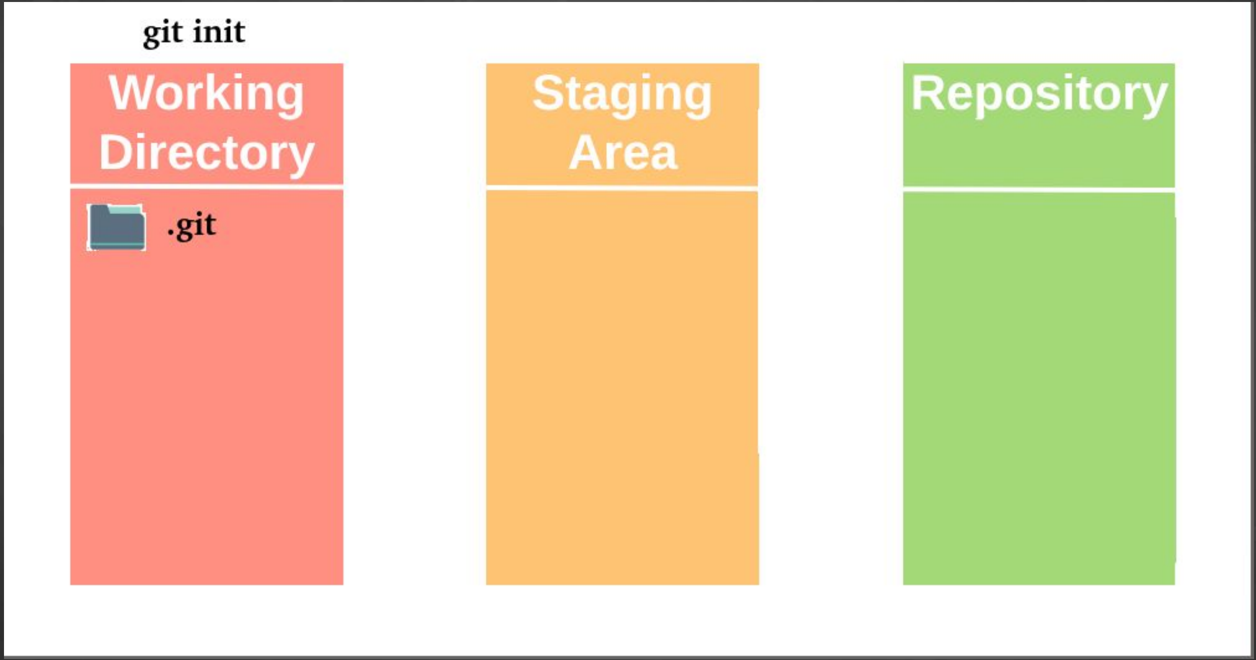
Initialization

First, we need to ask Git to keep an eye on our directory. From within the directory, run:

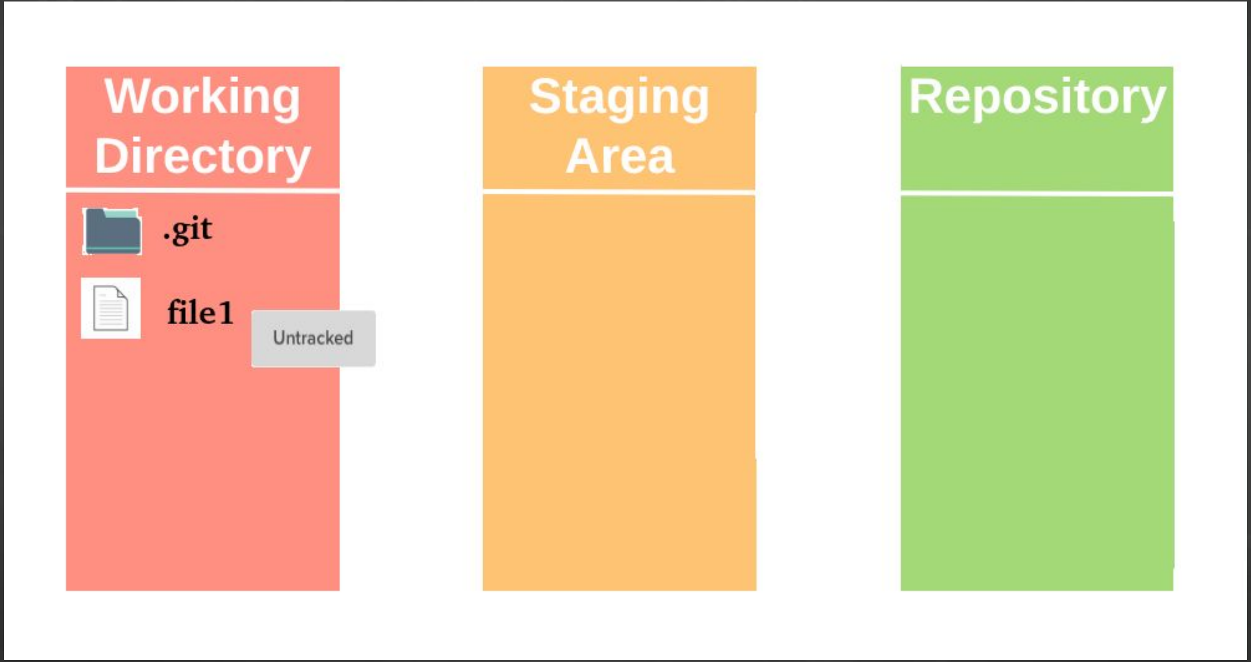
```
git init
```

This creates a subdirectory name .git that will hold all the files needed by Git.

Initialization



Making Changes



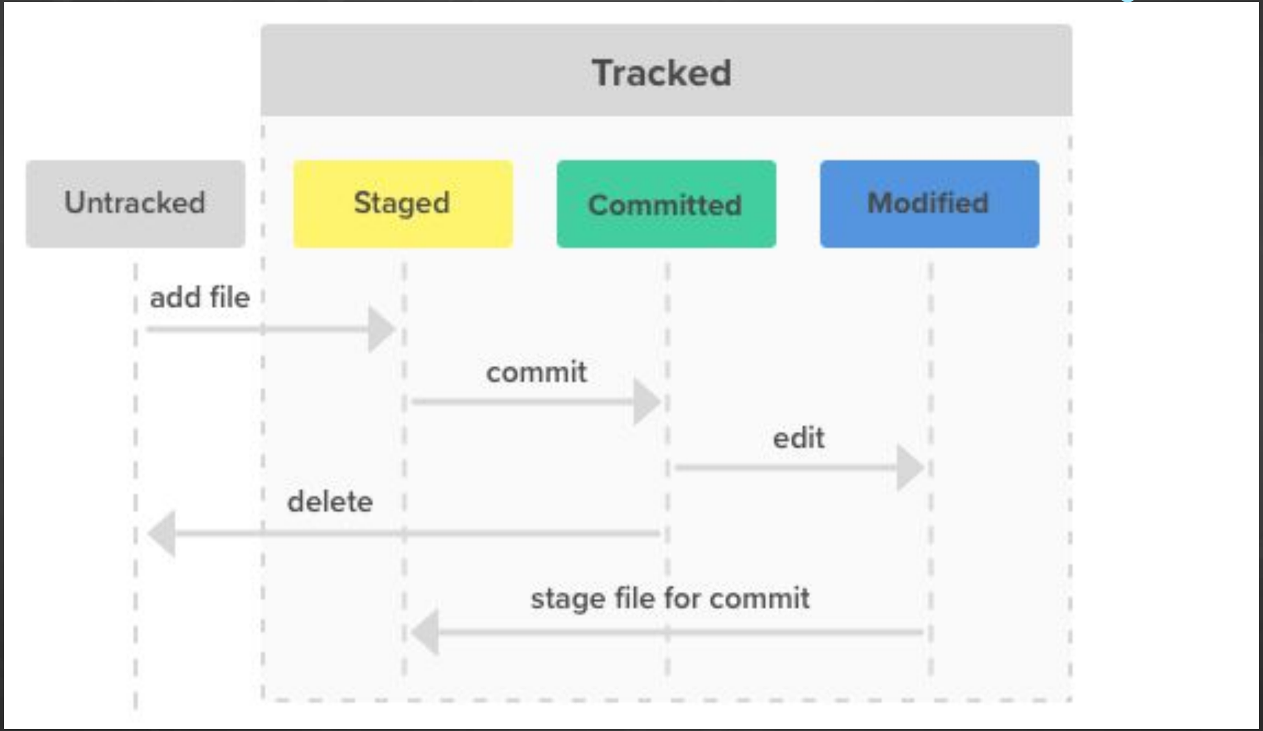
Viewing Changes

```
git status
```

This displays the status of every file in the working directory.

Each file can be in one of four states:
Untracked, Staged, Committed, Modified

File States in Git

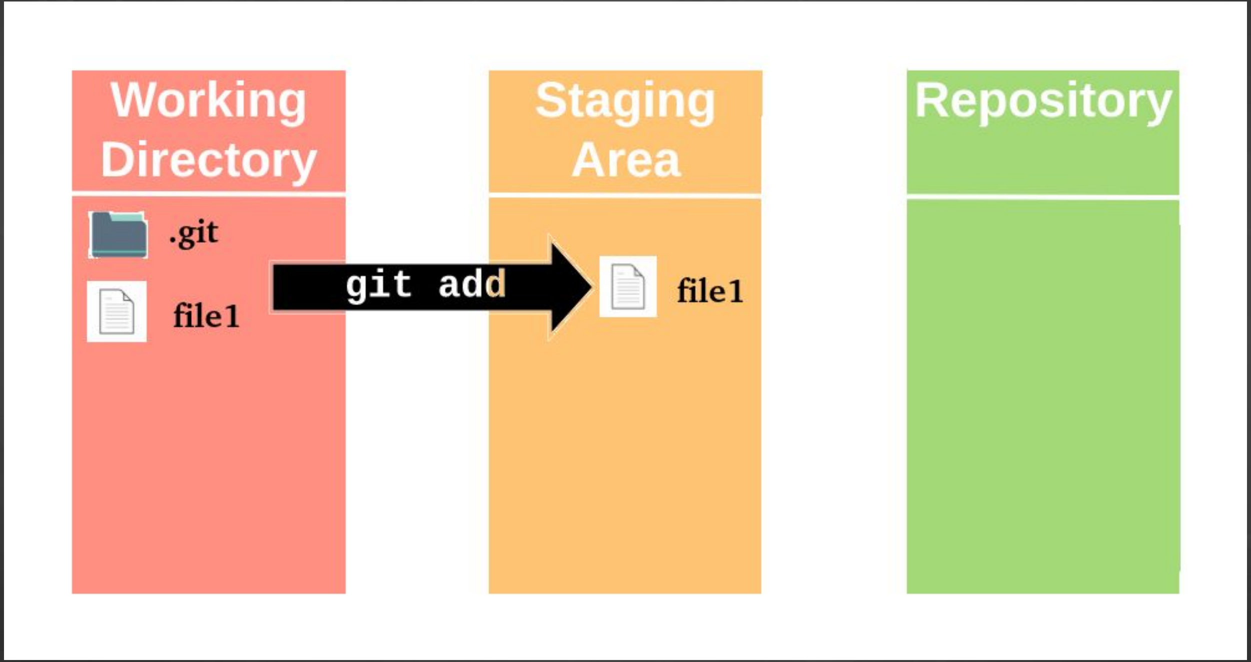


Staging Changes

```
git add <what to add>
```

- File name: `file.txt`
- Path to directory: `.` (current directory), `mydir/`
- Pattern: `*.cpp`, `*/`
- All files in the directory: `-A` or `--all`

Staging Changes



Ignoring Files

What if there are files you don't need to track?

You can write them in a file named `.gitignore`, and git will simply ignore them.

Ignoring Files

If you want to ignore a file that has already been staged, run:

```
git rm [-r] --cached [what to remove]
```

This makes Git no longer track the file without affecting its changes.

- File name: `file.txt`
- Path to directory: `.` (current directory), `mydir/`
  - Note: This requires the option `-r`
- Pattern: `*.cpp`, `*/`

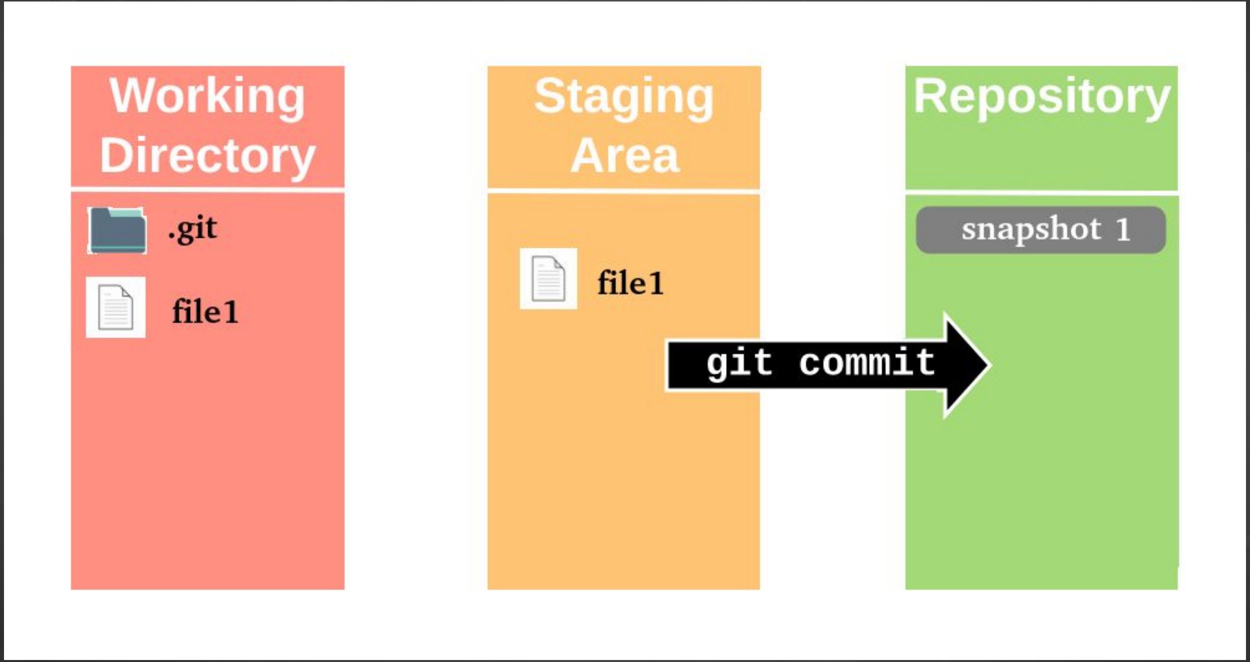
Committing Changes

When you’re satisfied with your staged changes, run:

```
git commit -m [commit message]
```

This will save the state of your staging area as a checkpoint in history.

Committing Changes



Unstaging Changes

If you no longer want a staged change to be committed, run:

```
git restore --staged [what to remove]
```

This will remove all changes from the staging area, but will keep them in your working directory.

- File name: `file.txt`
- Path to directory: `.` (current directory), `mydir/`
- Pattern: `*.cpp`, `*/`

Discarding Unstaged Changes

If you want to discard all changes that you haven’t staged, run:

```
git restore [what to restore]
```

This will remove all changes from the staging area as well as your working directory.

- File name: `file.txt`
- Path to directory: `.` (current directory), `mydir/`
- Pattern: `*.cpp`, `*/`

Viewing History

```
git log [--oneline]
```

This shows the commit history, including commit ID, author, timestamp, and message.

With the option `--oneline`, only ID and message are shown per commit, and each commit is displayed on a single line..

You realized that you messed up on that last commit, and you want to start over...

Good thing you're using Git!

Uh-oh!

Undoing Commits

```
git reset [--soft] [commit]
```

This resets out timeline to the specified commit.

The option --soft is used by default, and means that the changes after the given commit will be removed from the commit history, but will remain in the staging area and working directory.

Undoing Commits

```
git reset --mixed [commit]
```

This resets out timeline to the specified commit.

Using the option --mixed means that the changes after the given commit will be removed from the commit history and staging area, but will remain in the working directory.

Undoing Commits

```
git reset --hard [commit]
```

This resets out timeline to the specified commit.

Using the option --hard means that the changes after the given commit will be removed from the commit history, staging area and working directory. **Use this with care!**

Undoing Commits

```
git reset --hard [commit]
```

This resets out timeline to the specified commit.

However, using the option --hard means that the changes after the given commit will be deleted from the log, staging area and working directory.

Hands-On #2

Hands-On #2

1. Create a folder named "project", move into it, and initialize it as a Git repository.

2. Create a file called “README.txt” and write something in it.

3. Check the Git status and stage the file for commit.

4. Commit the staged file with a message describing the changes.

Hands-On #2

5. Create another file called “todo.txt”, then stage it and commit it.

6. Edit “README.txt” and check the Git status,

7. Restore “README.txt”to the last committed version.

8. You regret adding“todo.txt”. Go back to the first commit.

06

What’s Next?

Next Session Agenda

01 Branches

02 Conflict Resolution

03 GitHub

04 Managing Repos

05 Forking

06 Pull Requests

It is easy to shoot your foot off with Git, but also easy to revert to a previous foot and merge it with your current leg."

— Jack William Bell

Thank You!