TEKsystems
Own change

Git, GitHub, and Version Control

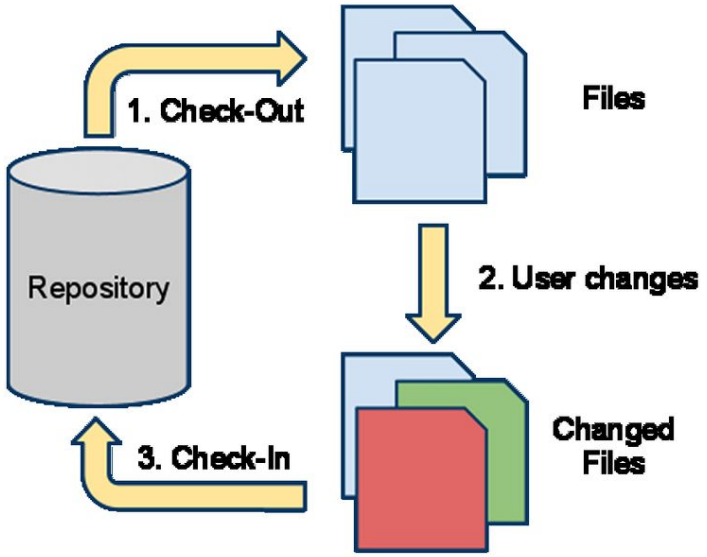
Table of content

- What is Version Control?
- What is Git?
- Git Basics
- Git's Role In DevOps
- Installing Git Software
- Git Configuration
- Fundamental Elements in the Git Workflow
- Git Local Workflow
- Working Directory and Staging Area (Index)
- Git - Local Repository
- Create Local Repositories
- Binary Large Object (BLOB)
- What is Commits?
- What is Head?
- What is Tags?
- How to Change/Undo in Git
- What is Pull Command
- What is Push Command?
- What is Revision Command?
- What is URL Git?
- Overview to Distributed Development
- What is GitHub?
- Git vs. GitHub?
- What is "Clone" command?
- What is Git Remote?
- Forking?

Lesson 1

Version Control System

What is Version Control System?



- Version control systems are a category of software tools that help a software team manage changes to source code over time.
- Version control enables rapid collaboration and iteration on new features, helping teams deliver business value and meet customer demand.
- Software development teams are continually writing new source code and changing existing source code. One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code. Version control helps teams track every individual change, by each contributor, and helps prevent concurrent work from conflicting.

Use Case

Let's say that there is a multinational company that has offices and employees around the globe. They may face the below challenges and problems:

- **Collaboration between Developer, Tester, Designers, Consultant, and other employees.**
- **Storing versions of source code.**
- **Restoring the previous version** - Sometimes, it is important to go back to find the root cause of the bug and error.
- **Backup** - If an employee system or disk suffers damage or breakdown, then all efforts will be lost.

Using a version control system will help an organization overcome these and other potential challenges.

Types of Version Control Systems

The two most popular types of version control systems are **centralized** and **distributed**. Centralized version control systems store all the files in a central repository, while distributed version control systems store files across multiple repositories. Other less common types of version control systems include **lock-based** and **optimistic**.

1: Distributed version control system

A distributed version control system (DVCS) is a type of version control system that allows users to access a repository from multiple locations. DVCSs are often used by developers who need to work on projects from multiple computers or who need to collaborate with other developers remotely.

2: Centralized version control system

A centralized version control system (CVCS) is a type of version control system (VCS) where all users are working with the same central repository. This central repository can be located on a server or on a developer's local machine. CVCSs are typically used in software development projects where a team of developers needs to share code and track changes.

Types of Version Control Systems (continued)

The two most popular types of version control systems are **centralized** and **distributed**. Centralized version control systems store all the files in a central repository, while distributed version control systems store files across multiple repositories. Other less common types of version control systems include **lock-based** and **optimistic**.

3: Lock-based version control system

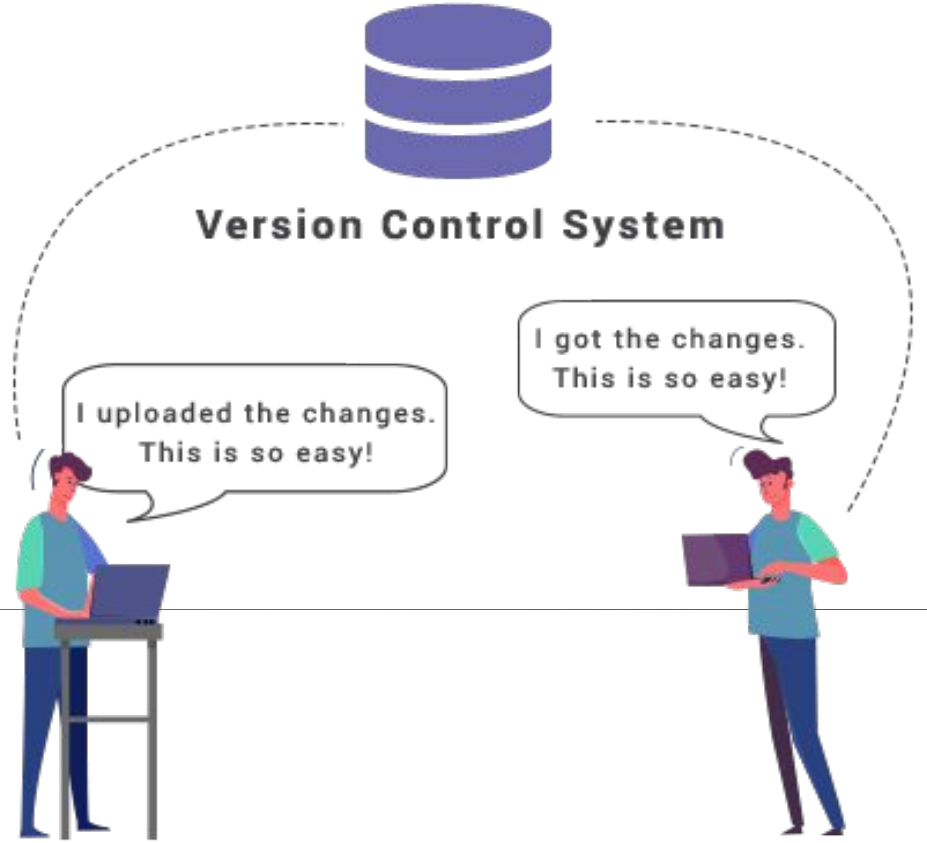
A lock-based version control system is a type of version control system that uses locks to manage concurrent access to files and resources. Locking prevents two or more users from making conflicting changes to the same file or resource.

4:Optimistic version control system

In an optimistic version control system, every user has their own private workspace. When they want to share their changes with the rest of the team, they submit a request to the server. The server then looks at all the changes and determines which ones can be safely merged together.

Version Control Systems/Tools

- Git.
- Microsoft Team Foundation Server.
- Helix Core.
- Subversion.
- AWS CodeCommit.
- Rational ClearCase.
- Mercurial.
- Micro Focus AccuRev.
- CVS.



Lesson 2

Git and GitHub

What is Git?

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 the most commonly used version control system today, and is quickly becoming the standard for version control.

Source; githubcom

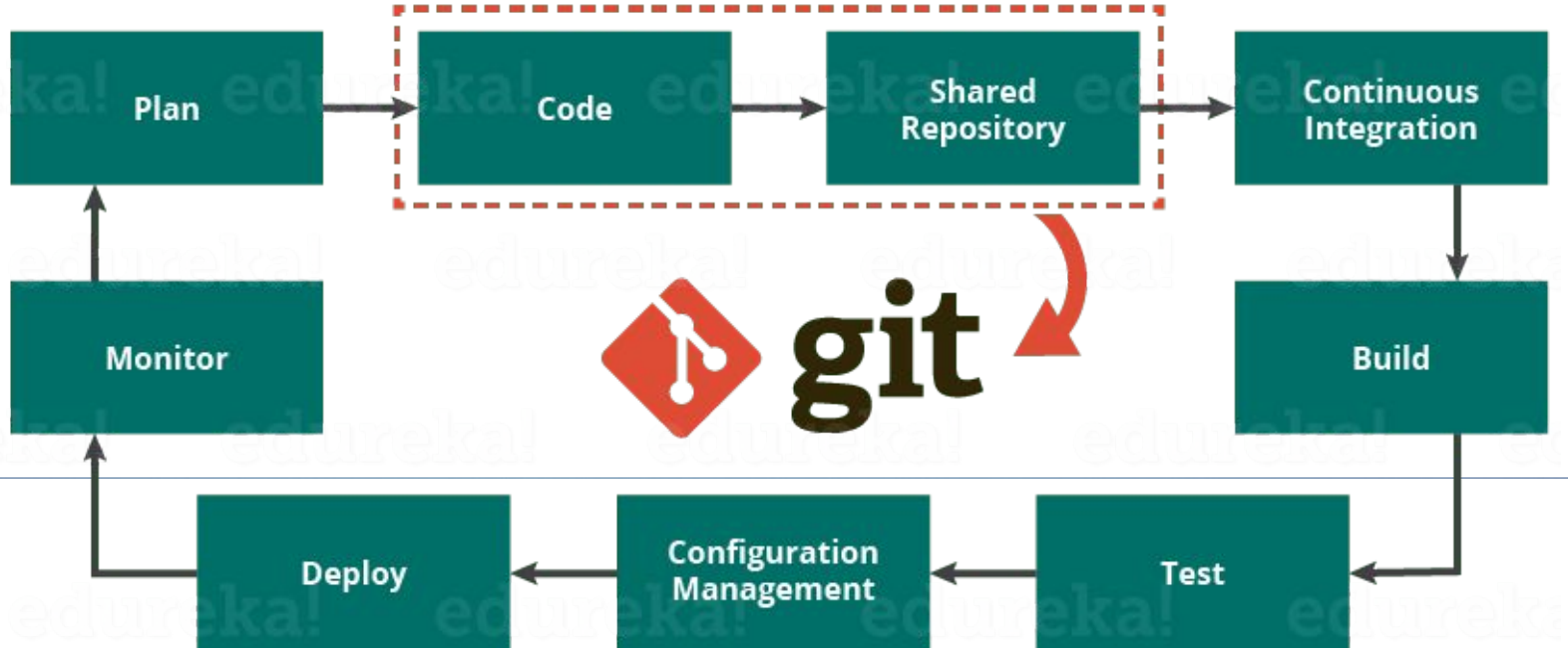
Git Basics

Each time work is saved, Git creates a commit.

A commit is a snapshot of all files at a point in time. If a file has not changed from one commit to the next, then Git uses the previously stored file. This design differs from other systems, which store the initial version of a file, and then keeps a record of details over time.

Git's Role In DevOps

The diagram below depicts the DevOps LifeCycle and displays how Git fits into DevOps.

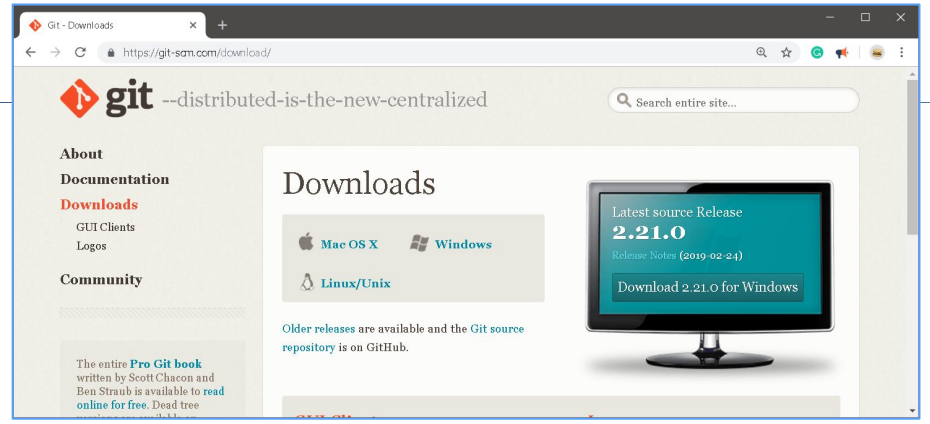


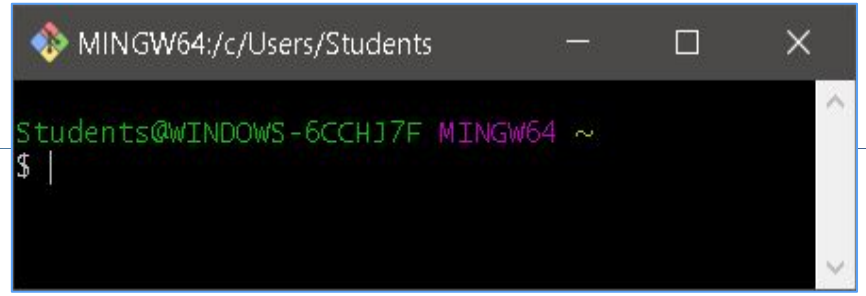
source: edureka

Installing Git Software

Complete the **Lab - 302.1.1 - Git installation on Windows**, you can find this lab on canvas under Guided Lab section

For more information about Git, and to download the appropriate version of Git for your operating system, visit: https://git-scm.com/download/





After installation, you should have access to the git shell / git bash.

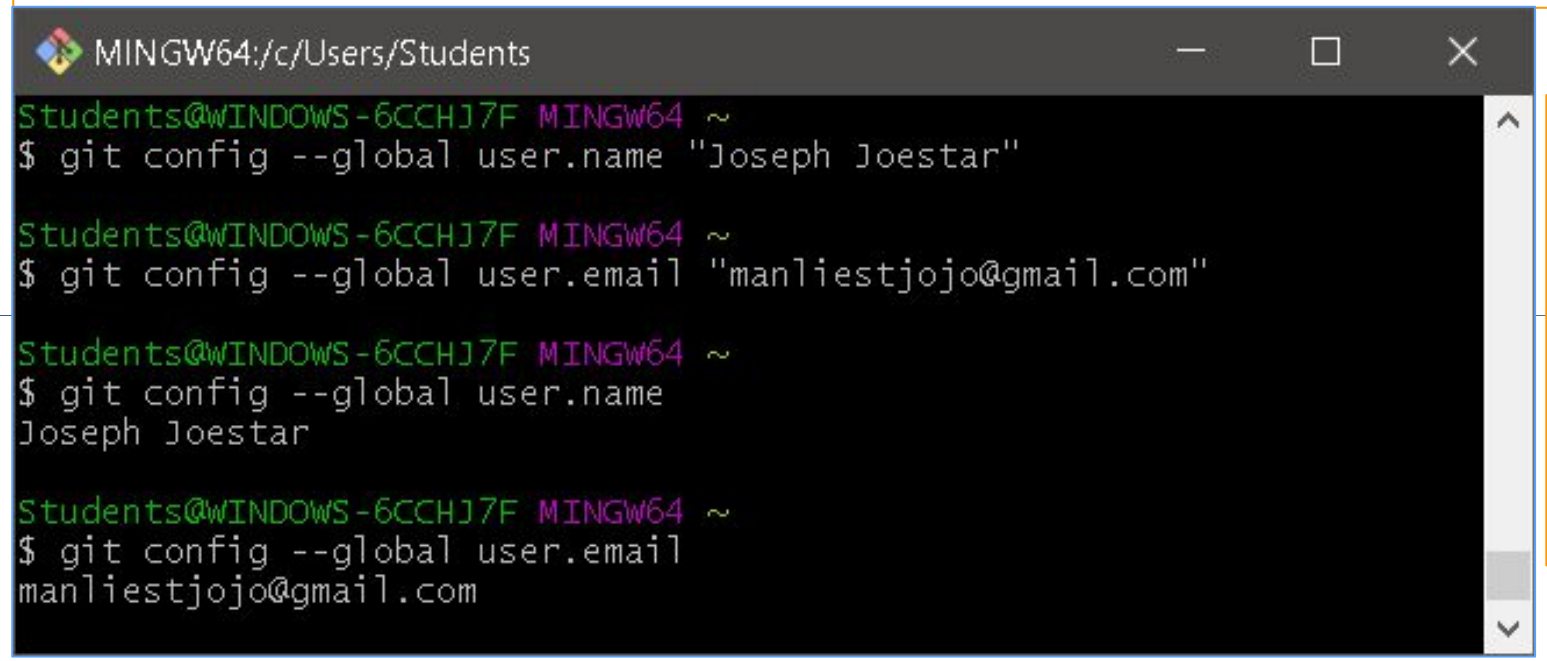
TEKsystems
Own change

Git Configuration

It is important to configure your Git username and email address. Every Git commit will use this information to identify you as the author.

On your shell, type the following commands to add your username and email address:

- git config --global user.name "YOUR_USERNAME“
- git config --global user.email "your_email_address@example.com"



Verify the correct username or email address by using the following commands:

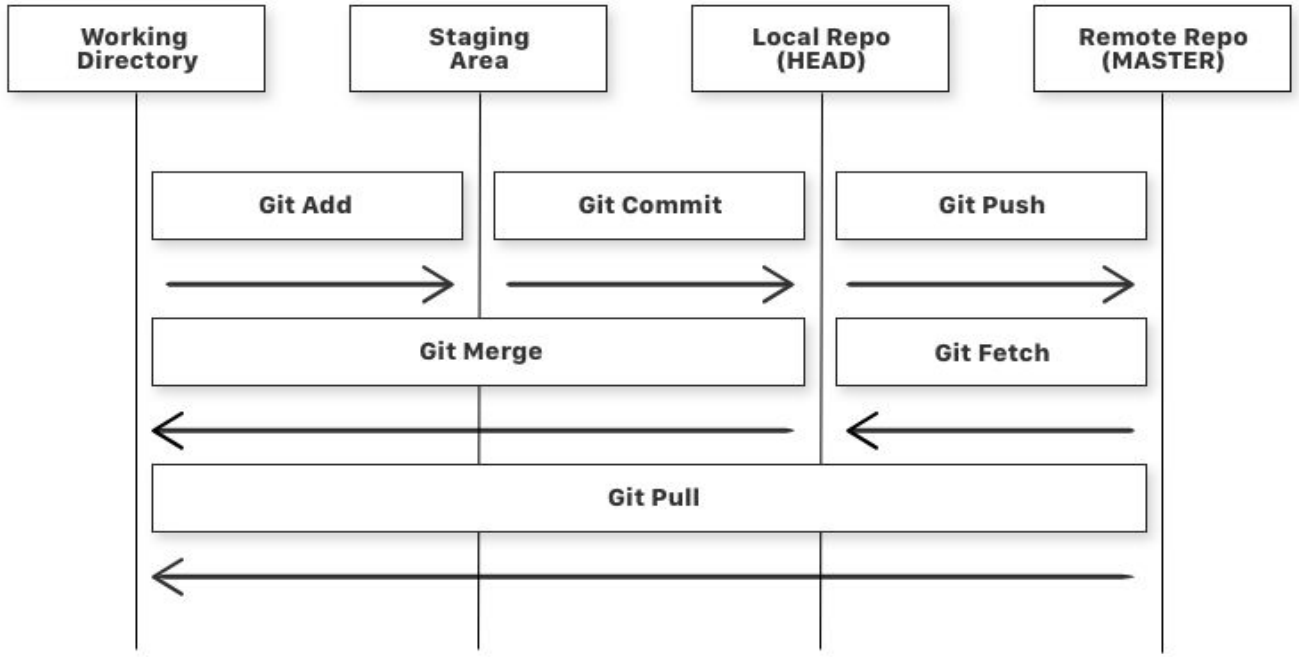
- git config --global user.name
- git config --global user.email
  OR
- git config --global --list

TEKsystems
Own change

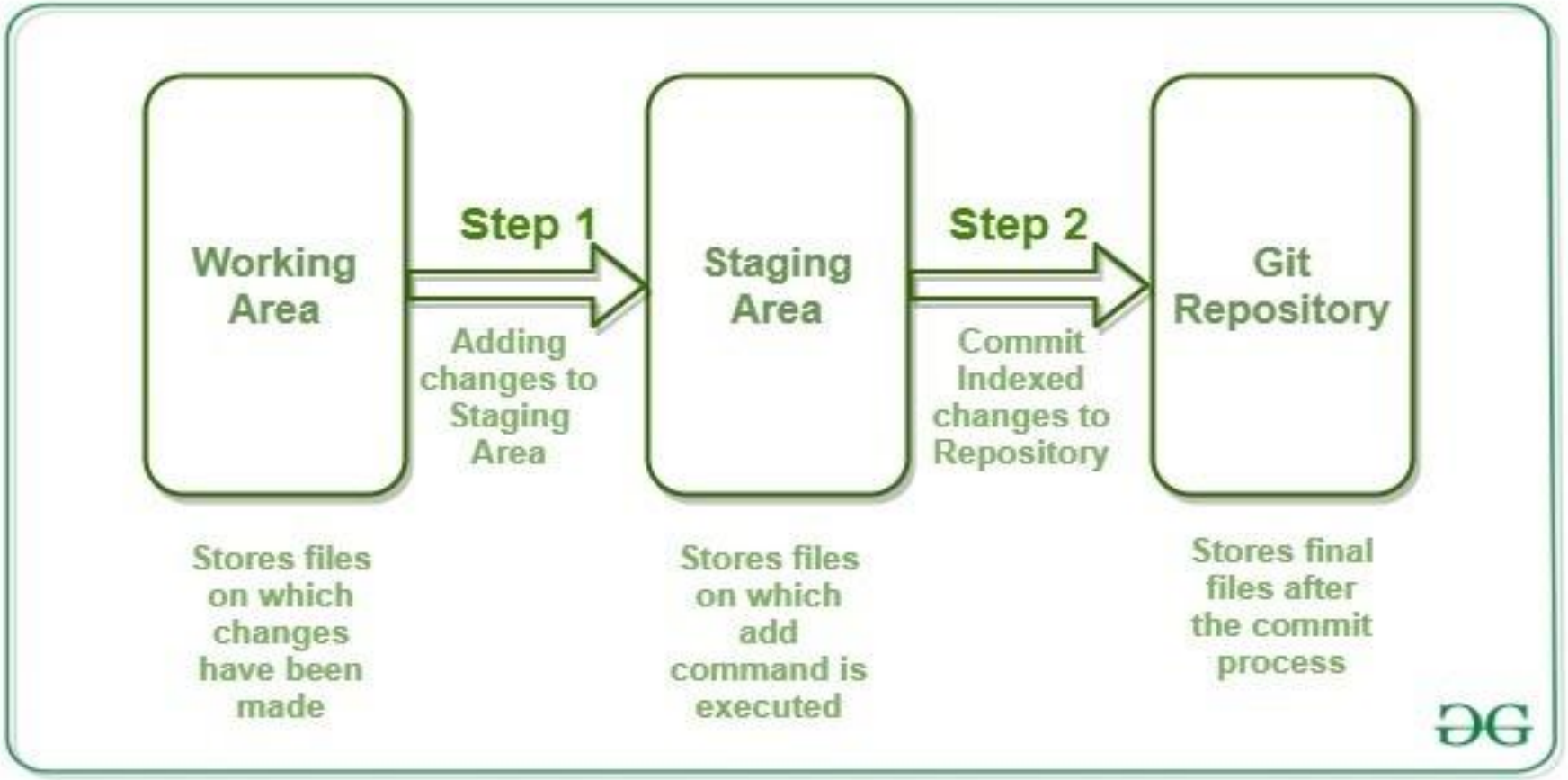
Fundamental Elements in the Git Workflow

There are four fundamental **elements** in the Git Workflow:

1) Working Directory, 2) Staging Area, 3) Local Repository, and 4) Remote Repository.



Git Local Workflow

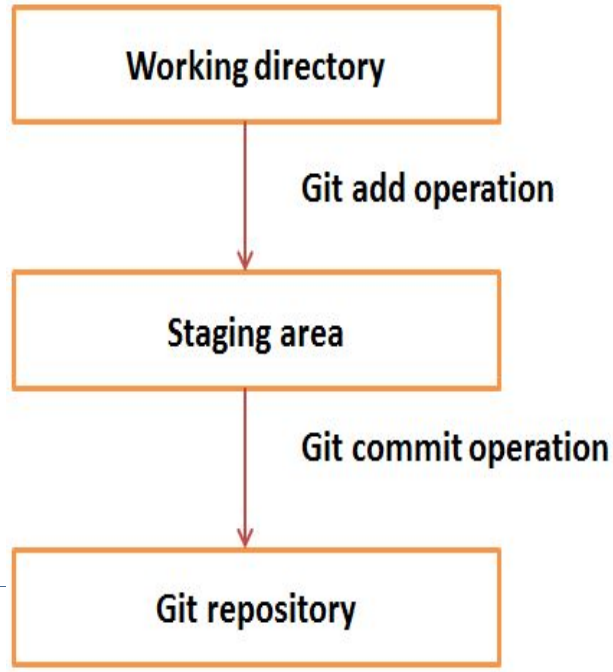


Working Directory and Staging Area (Index)

The working directory is the place where files are checked out. When a **commit operation** is performed, Git looks for the files present in the staging area. Only files present in the **staging area** are considered for commit — not all the modified files.

The workflow includes:

1. Modifying a file from the working directory.
2. Adding the files to the staging area.
3. Performing a commit operation that moves the files from the staging area, and after a *push* operation, stores the changes permanently to the Git repository.



Git - Local Repository

- A local repo (repository) provides a private workspace as a *working copy*. Developers make changes in their private workplace, and after a commit, these changes become a part of the repo. Git provides a private copy of the entire repo. Users can perform many operations with this repo, such as add files, remove files, rename files, move files, commit changes, and much more.
- Repositories can contain folders, files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a README file, or a file with information about your project.

Create Local Repositories

- When starting a new local repository, you only need to type in the **git init** command.
- If you have a project directory that is currently not under version control, and you want to start controlling it with Git, you first need to go to that project's directory.
  - **For Linux:**

    ```
    $ cd /home/user/my_project
    ```

  - **For macOS:**

    ```
    $ cd /Users/user/my_project
    ```

  - **For Windows:**

    ```
    $ cd C:/Users/user/my_project
    ```

    and type:

    ```
    $ git init
    ```

    → This creates a new subdirectory named `.git` that contains all of your necessary repository file.

Binary Large Object (BLOB)

- A Git Binary Large Object (BLOB) is the object type used to store the contents of each file in a repository.

- It contains your original data files and all the log messages, author information, dates, and other information required to rebuild any revision or branch of the project. It is a binary file, and in the Git database, it is named **SHA1 hash** of that file. In Git, files are not addressed by names; everything is content-addressed.

For more information visit: https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

SHA-1 Hash

All of the information needed to represent the history of a project is stored in files referenced by a 40-digit "object name" that looks something like this:

8hf67f4664981e4397625791c8eabbb5f2279a31

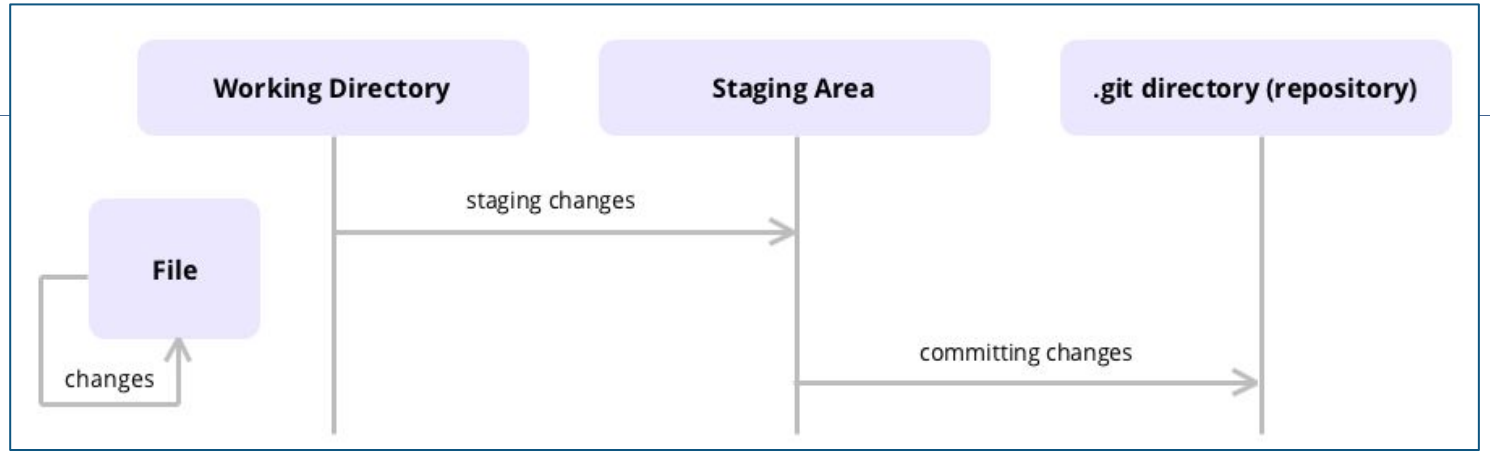
SHA-1 (Secure Hash Algorithm 1) is a one-way cryptographic hash function, which takes an input and produces a 160-bit hash value, known as a *message digest* – typically rendered as a hexadecimal number, which is 40 digits long.

SHA-1 Hash Example:

c3499c2729730a7f807efb8676a92dcb6f8a3f8f

What is Commits?

- In Git, commit is the term used for **saving changes**. Git does not add changes to a commit automatically, you need to indicate which file and which changes need to be saved before running the Git commit command.
- The commit command does not save changes in remote servers; only in the local repository of Git.
- Commits hold the current state of the repository. A commit is also named by **SHA-1 Hash**.

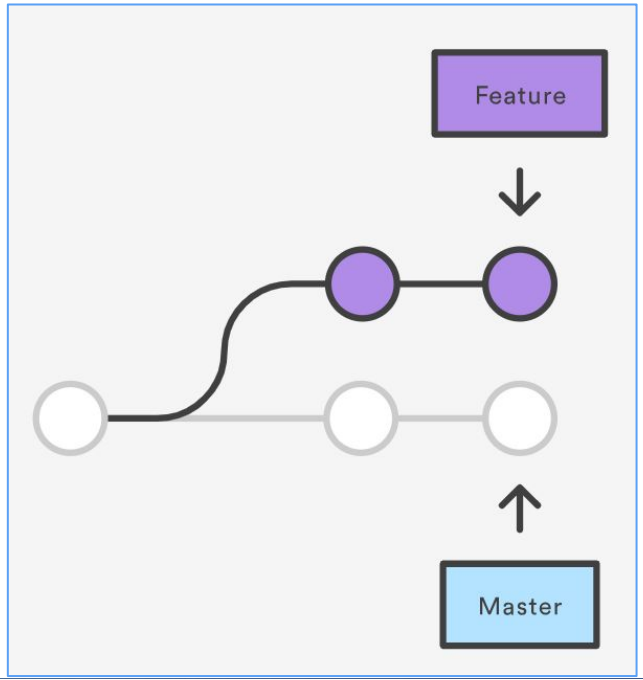


What is Branch?

Branches provide an isolated environment for every change to the codebase. When a developer wants to start working on new feature, they can create a new branch. This ensures that the master/main branch always contains a production-quality code.

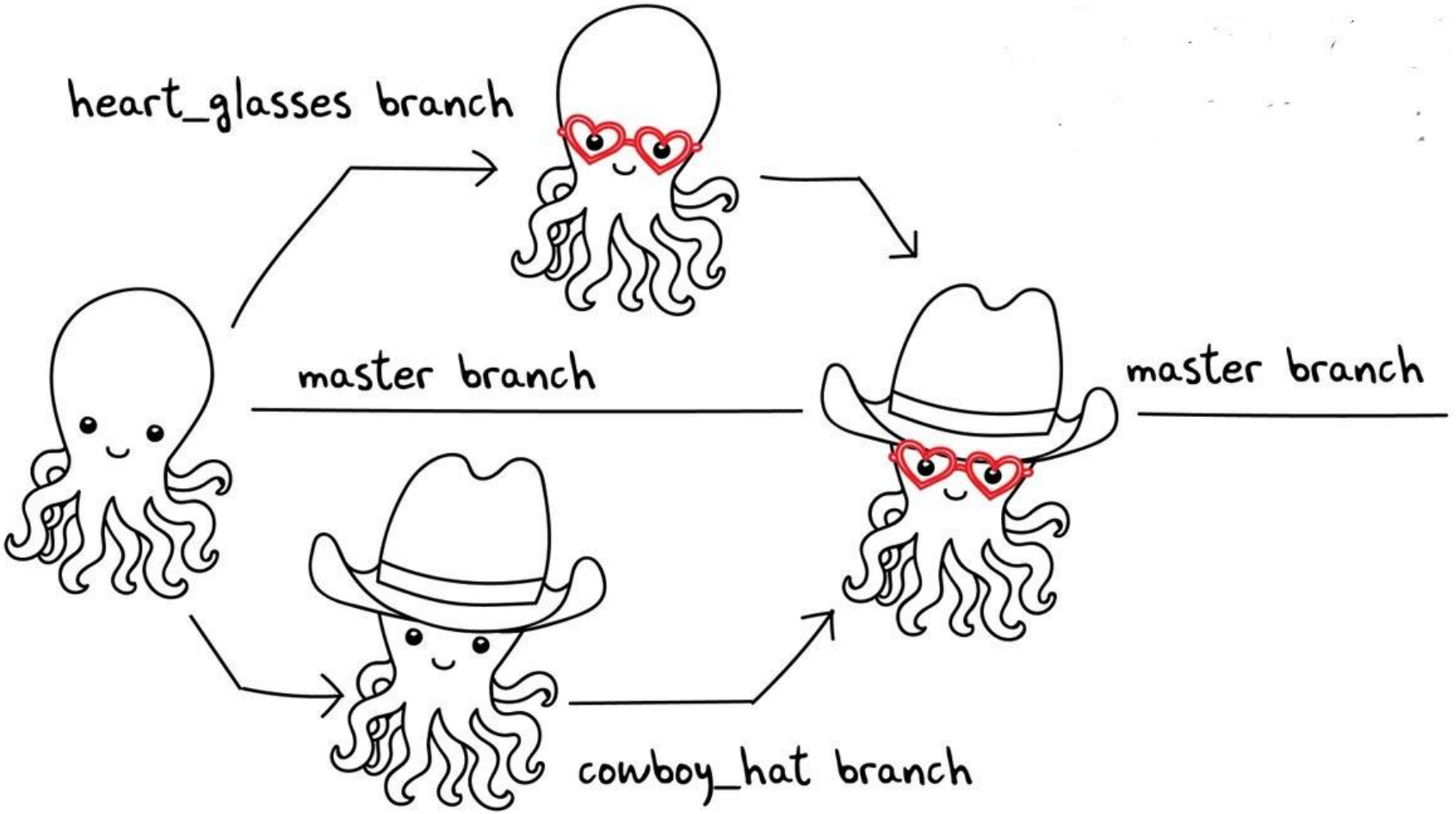
Each developer saves changes to their own local code repository. As a result, you can have many different changes based off of the same commit. Git provides tools for isolating changes, and later for merging them back together.

The new branches can be used to test changes to code without affecting the main project code. If the changes work, the branch can be merged back with the main branch.




```
git branch my-new-branch      // create a new branch
git branch                    // show all list of branches
git checkout my-new-branch    // switch master branch to new branch
git branch                    // show all list of branches
```

Branches (continued)



What is Head?

The term HEAD refers to the current commit you are viewing.

HEAD is a reference to the last commit (latest commit) in the branch. It is similar to a pointer to any reference.

The Head can be understood as the "current branch." When you switch branches with 'checkout,' the Head is transferred to the new branch.

What are Tags?

- Git has the ability to tag specific points in a repository's history as *important*.
- Tags assign a meaningful name with a specific version in the repository.
- Tags are very similar to branches; however, tags are immutable. This means that a tag is a branch that is not intended for modification. Once a tag is created for a particular commit, if a new commit is created, the tag will not be updated. Usually, developers create tags for product release.

Create Git Tag for Commit

```
git tag <tag_name> <commit_sha>
```

```
For more information: https://devconnected.com/how-to-create-git-tags/
```

How to Change/Undo in Git

`git revert`

git revert is the safest way to change history with Git. Instead of deleting existing commits, git revert looks at the changes introduced in a specific commit, and then applies the inverse of those changes in a new commit. It functions as an "undo commit" command without sacrificing the integrity of your repository's history. **git revert** is always the recommended way to change history when it's possible.

`git revert commitid``(rollback and Revert some existing commits)`

How to Change/Undo in Git (continued)

`git reset`

Sometimes, a commit includes sensitive information and needs to be deleted. **git reset** is a very powerful command that may cause you to lose work. By resetting, you move the HEAD pointer and the branch pointer to another point in time - maybe making it seem like the commits in between never happened!

`git reset --hard (commitid)` `(Revert and rollback some existing commits but cannot forward)`

Before using **git reset**:

- Be sure to talk with your team about any shared commits.
- Research the three types of reset to see which is right for you (--soft, --mixed, --hard).
- Commit any work that you do not want to be lost intentionally - work that is committed can be retrieved; uncommitted work cannot.

What is Revision Command?

Revision represents the version of the source code. Revisions in Git are represented by commits, which are identified by SHA-1 secure hashes.

What is URL Git?

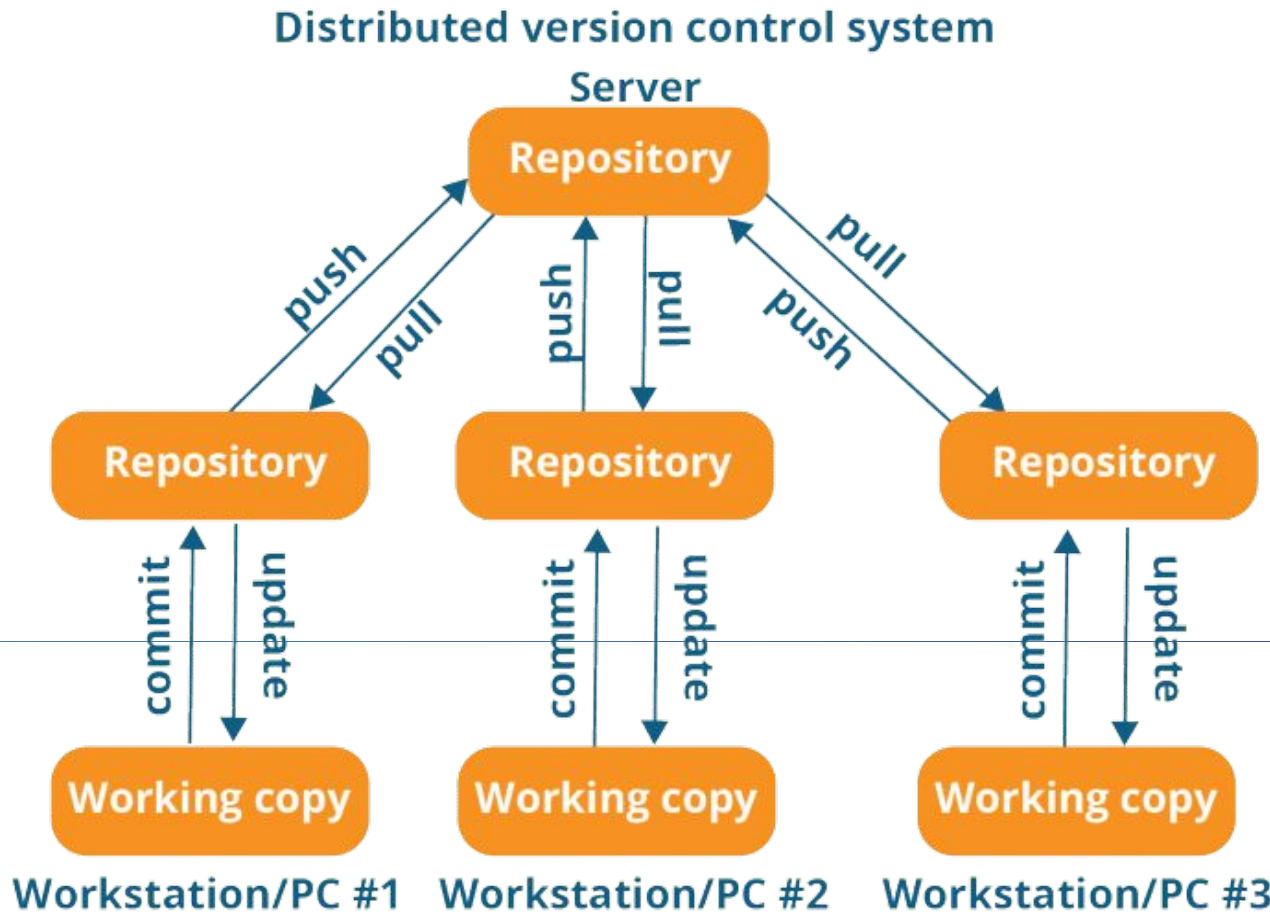
URL represents the location of the Git repository. Git URL is stored in a configuration file.

Git remote set-url origin: git@github.com:gitusername/repository.git

Overview to Distributed Development

Git is a distributed version control system, which means that the local copy of code is a complete version control repository. These fully functional local repositories make it is easy to work offline or remotely.

Distributed development creates a more reliable environment. Even if a developer obliterates their own repository, they can simply clone someone else's and rebuild from there.



What is Pull Command?

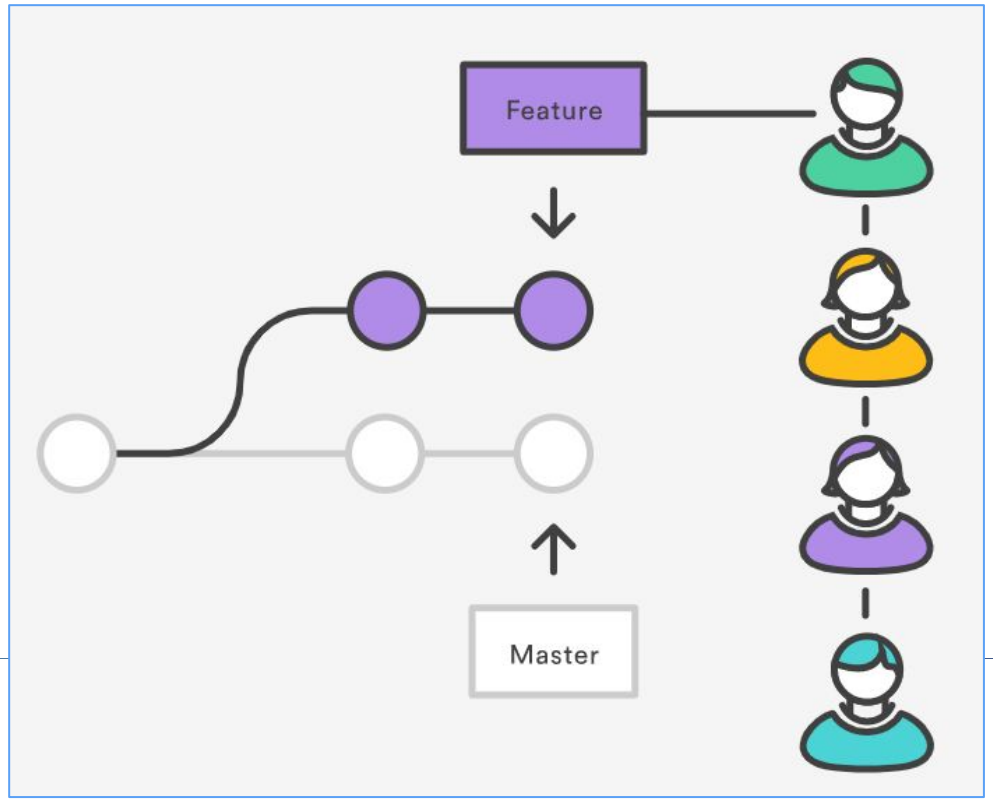
Pull operations copy the changes from a remote repository instance to a local instance. The pull operation is used for synchronization between two repository instances.

What is Pull Command?
(continue)

A Pull request is a way to ask another developer to merge one of your branches into their repository.

This allows developers to initiate discussions around work before integrating it with the rest of the codebase.

When a developer encounters a difficulty, they can open a pull request to ask for help from the rest of the team.

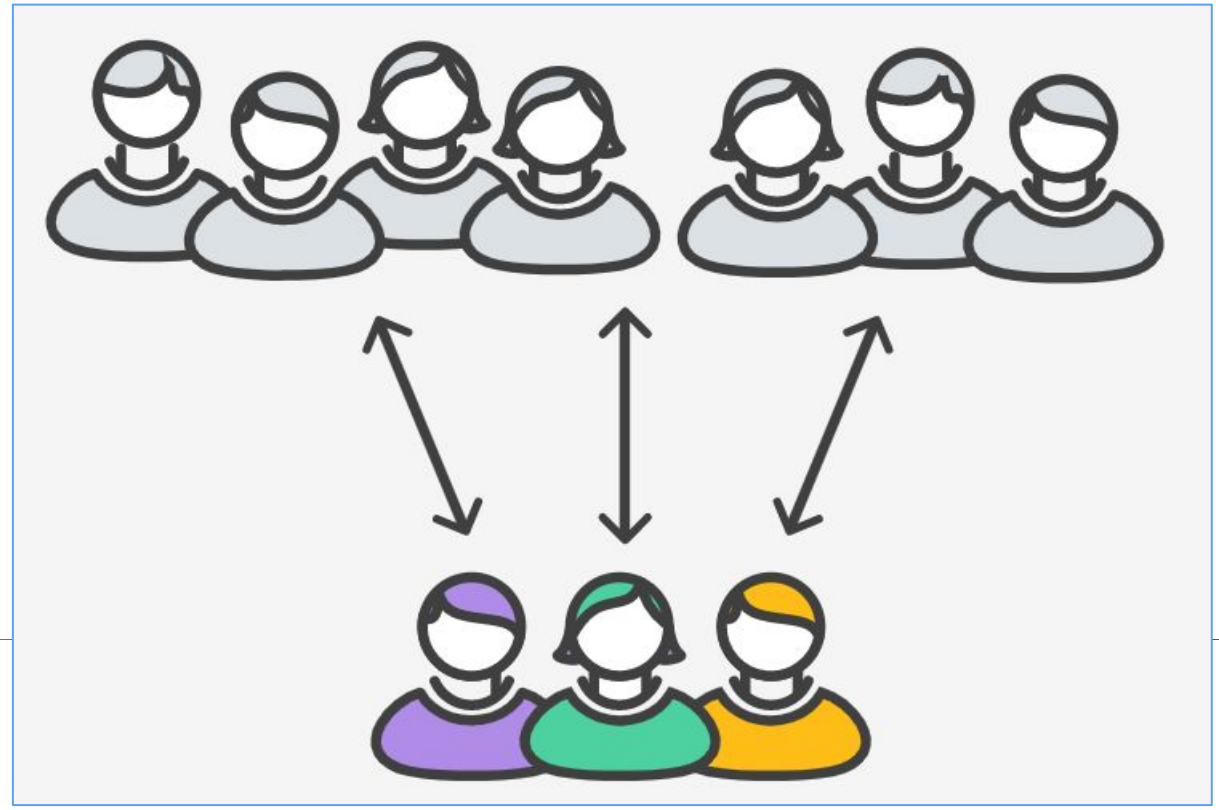


What is Push Command?

Push operations copy changes from a local repository instance to a remote repository instance. This is used to store changes permanently into the Git repository.

Git Community

Git is very popular among open-source projects. This means that it is easy to leverage third-party libraries and encourage others to fork your own open source code.



What is GitHub?

GitHub is a web-based Git repository hosting service that creates a centralized storage space where users can store and access their web development projects.

GitHub is also a social network for developers. In many cases, your GitHub account is more important to employers than your LinkedIn account or even your resume.

For more information, visit:

https://lab.github.com/githubtraining/introduction-to-github

https://github.com/

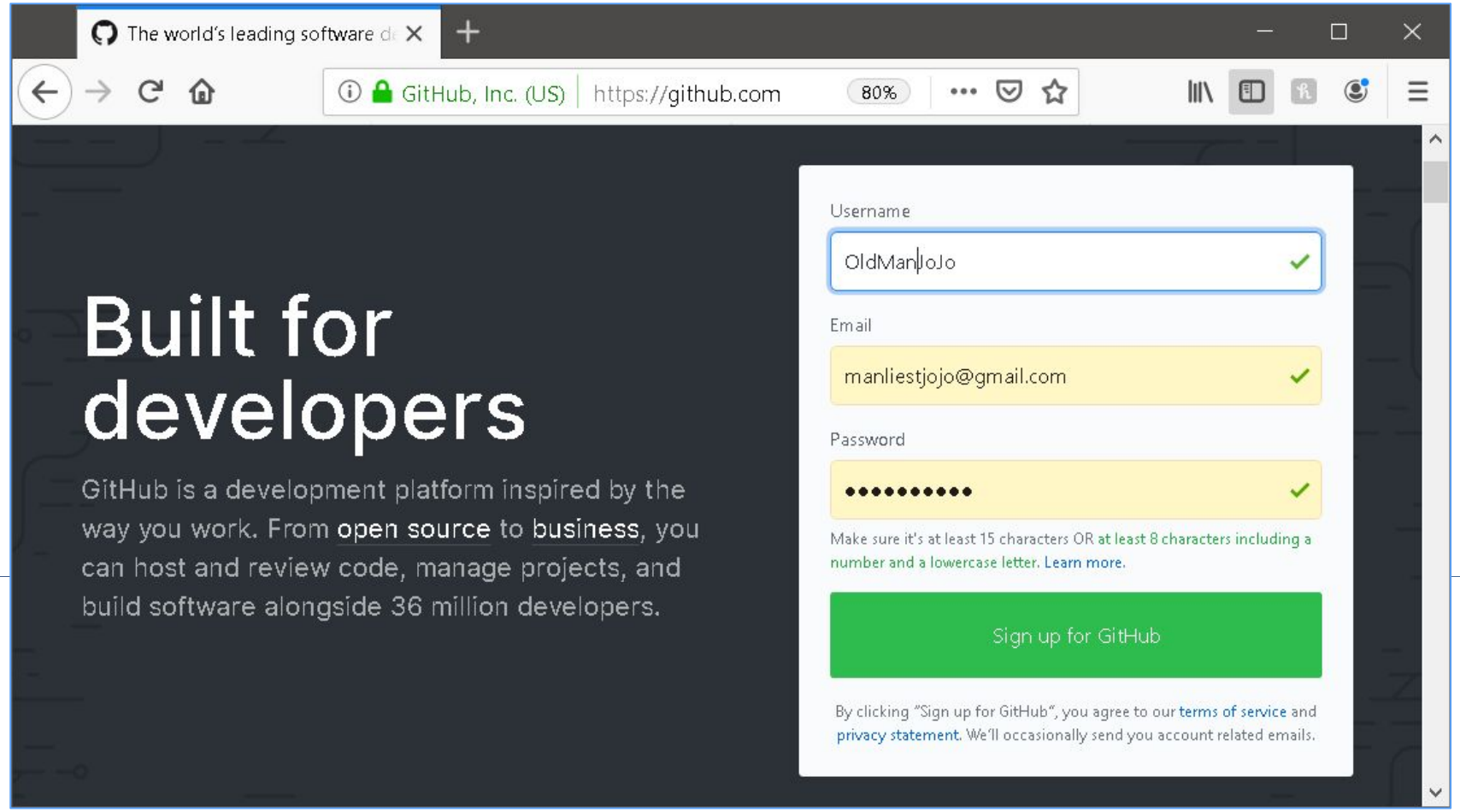
Git vs. GitHub?

Git	GitHub
Git is a distributed version control system which tracks changes to source code over time.	GitHub is a web-based hosting service for Git repository to bring teams together.
Git is a command-line tool that requires an interface to interact with the world.	GitHub is a graphical interface and a development platform created for millions of developers.
It creates a local repository to track changes locally rather than store them on a centralized server.	It is open-source which means code is stored in a centralized server and is accessible to everybody.
It stores and catalogs changes in code in a repository.	It provides a platform as a collaborative effort to bring teams together.
Git can work without GitHub as other web-based Git repositories are also available.	GitHub is the most popular Git server but there are other alternatives available such as GitLab and BitBucket.

Git is a version control system that manages and tracks changes to your source code, whereas GitHub is a cloud-based hosting platform that manages all of your Git repositories.

GitHub is cloud-based hosting platform that you can connect Git to, using either one of your command line interfaces or GitHub's GUI interface. You can push and pull data to and from "remote" repository stores on GitHub's server. To do this, you must already have a local repository on your computer.

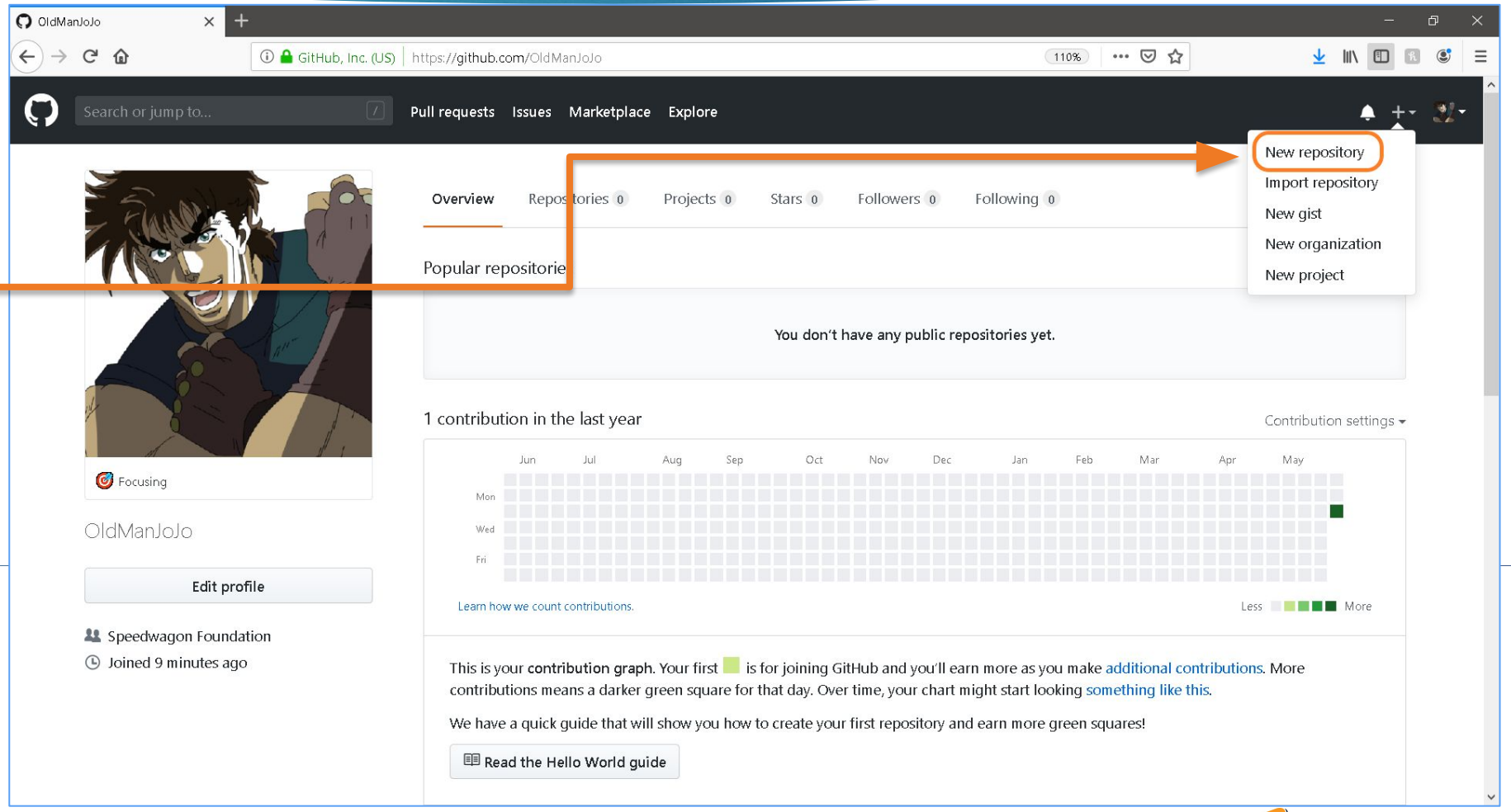
Create Remote Repository



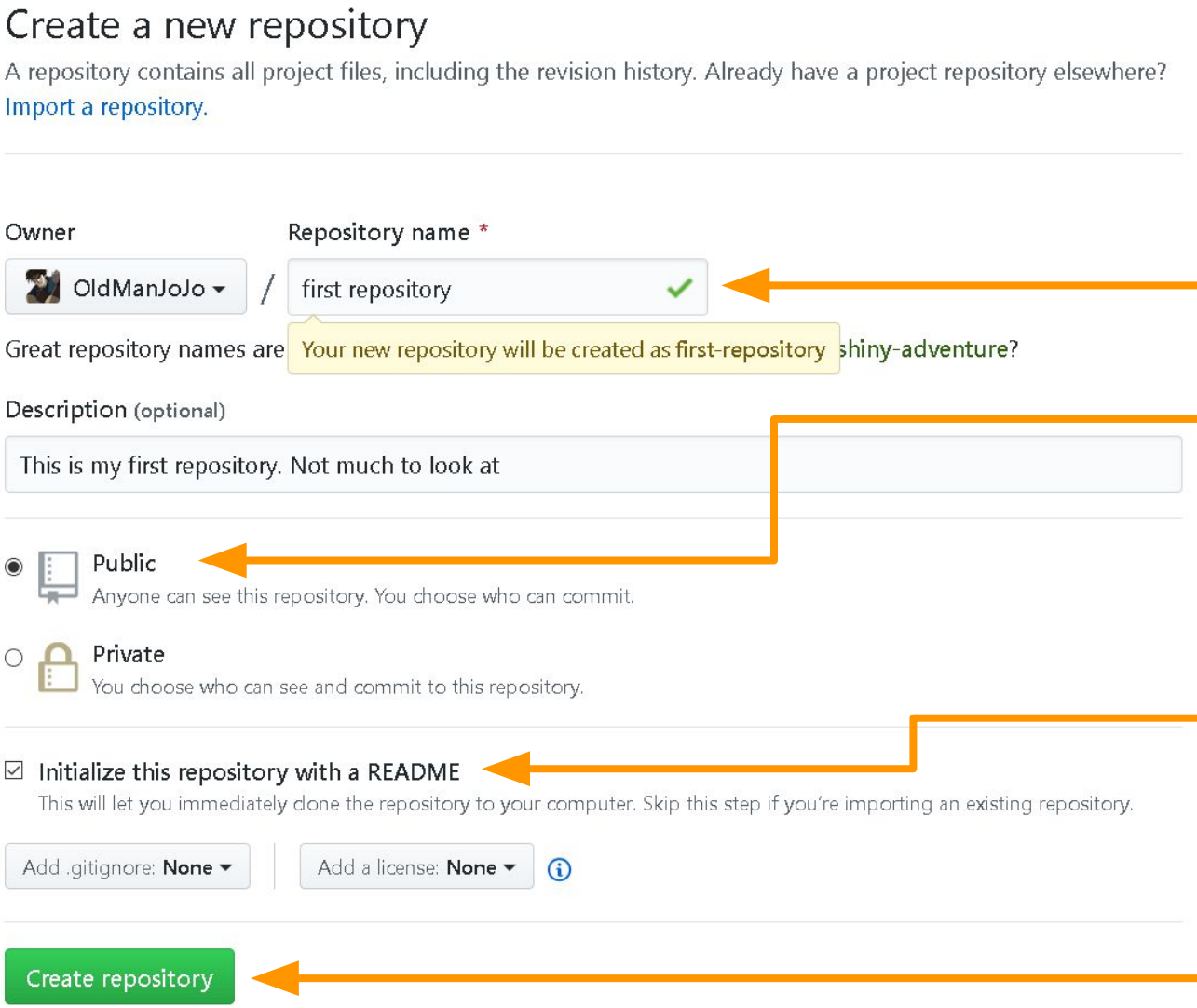
Visit https://github.com/ to sign in to GitHub or to sign up for GitHub.

Create Remote Repository (continued)

Once signed in, look in the upper-right corner next to your avatar, and select (click) *New repository*.



Create Remote Repository (continued)



Name and write a short description of your repository.

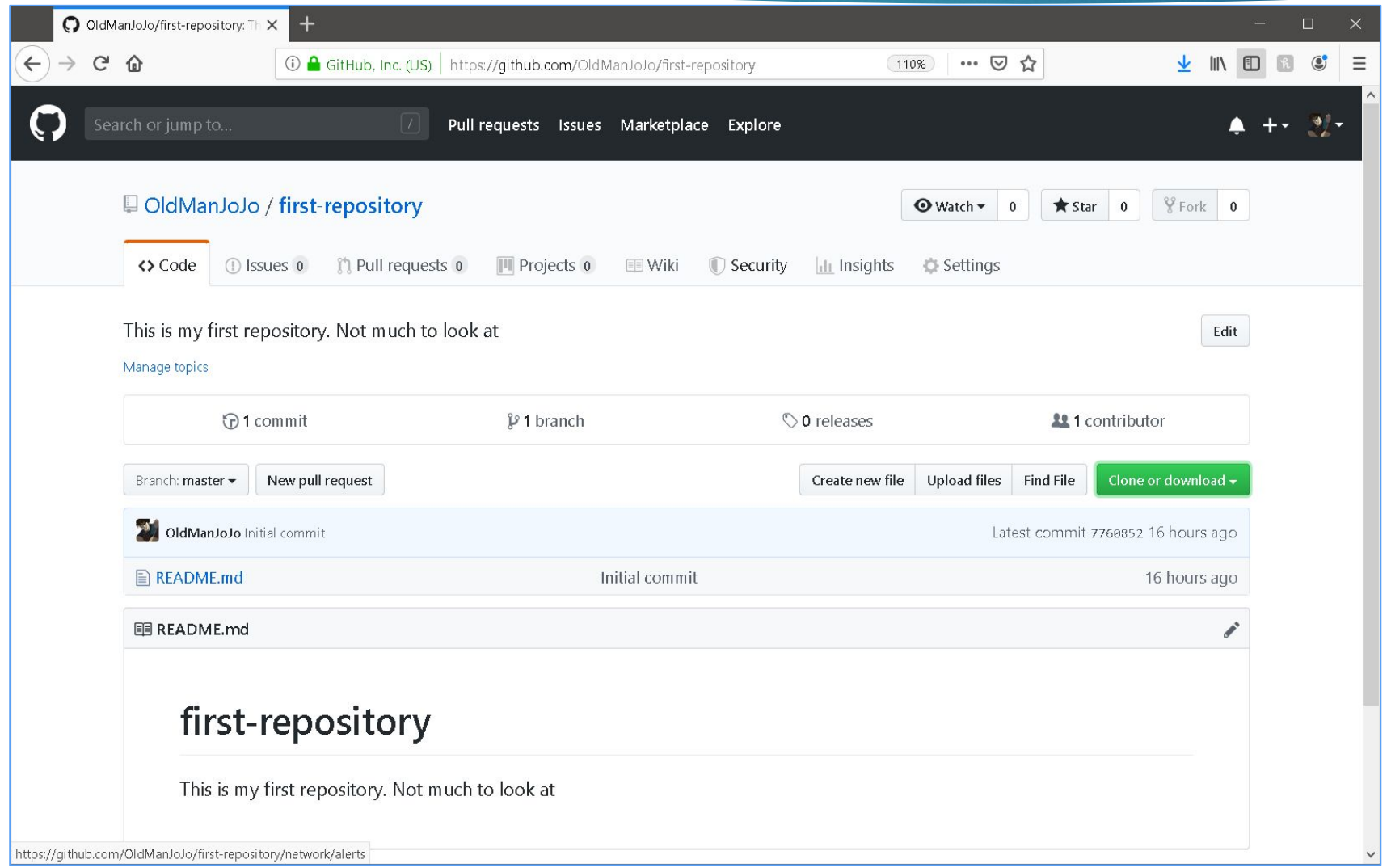
Select "public" so that everyone can see your new repository.

Select "Initialize this repository with a README." This will let you immediately clone the repository to your computer. Skip this step if you are importing an existing repository.

Finally, click "Create repository."

TEKsystems
Own change

Create Remote Repository (continued)



Congratulations! You just created your first remote Git repository.

A remote in Git is a common repository that all team members can use to store and exchange changes to the code. In most cases, such a remote repository is stored on a code hosting service such as GitHub, or on an internal server.

TEKsystems
Own change

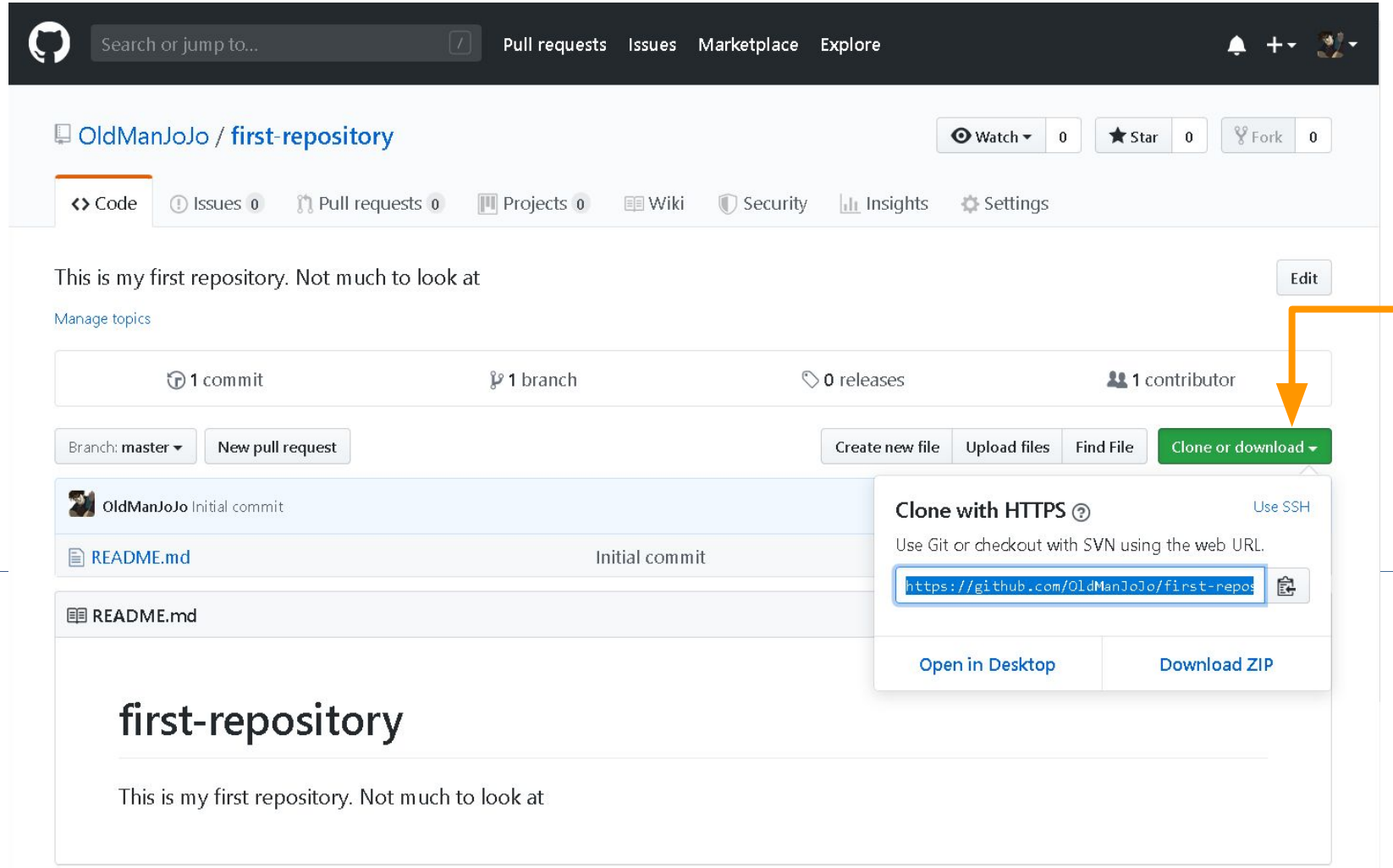
Link Your Repository with Local

Two easy methods to link your local (Git) and remote/online versions of your project are "git clone" and "git remote add:

- Clone: If you have not created the project on your local server (desktop computer), **git clone** is used to copy or clone a repository.
- Remote add: **git remote add** is used to refer to a remote repository, or your central repository if you already started with your project locally. But you should have a new or more current version of the repository.

git clone != git remote

Link Your Repository with Local (continued)



Click on "clone or download," and copy the link in the Clone with HTTPs section. We will use this link to clone/pull this repository using the git bash.

Note: Alternatively, you can download your repository as a zip file or open it using GitHub Desktop (a graphical user interface to use Git) if it is already installed in your system.

What is “Clone” Command?

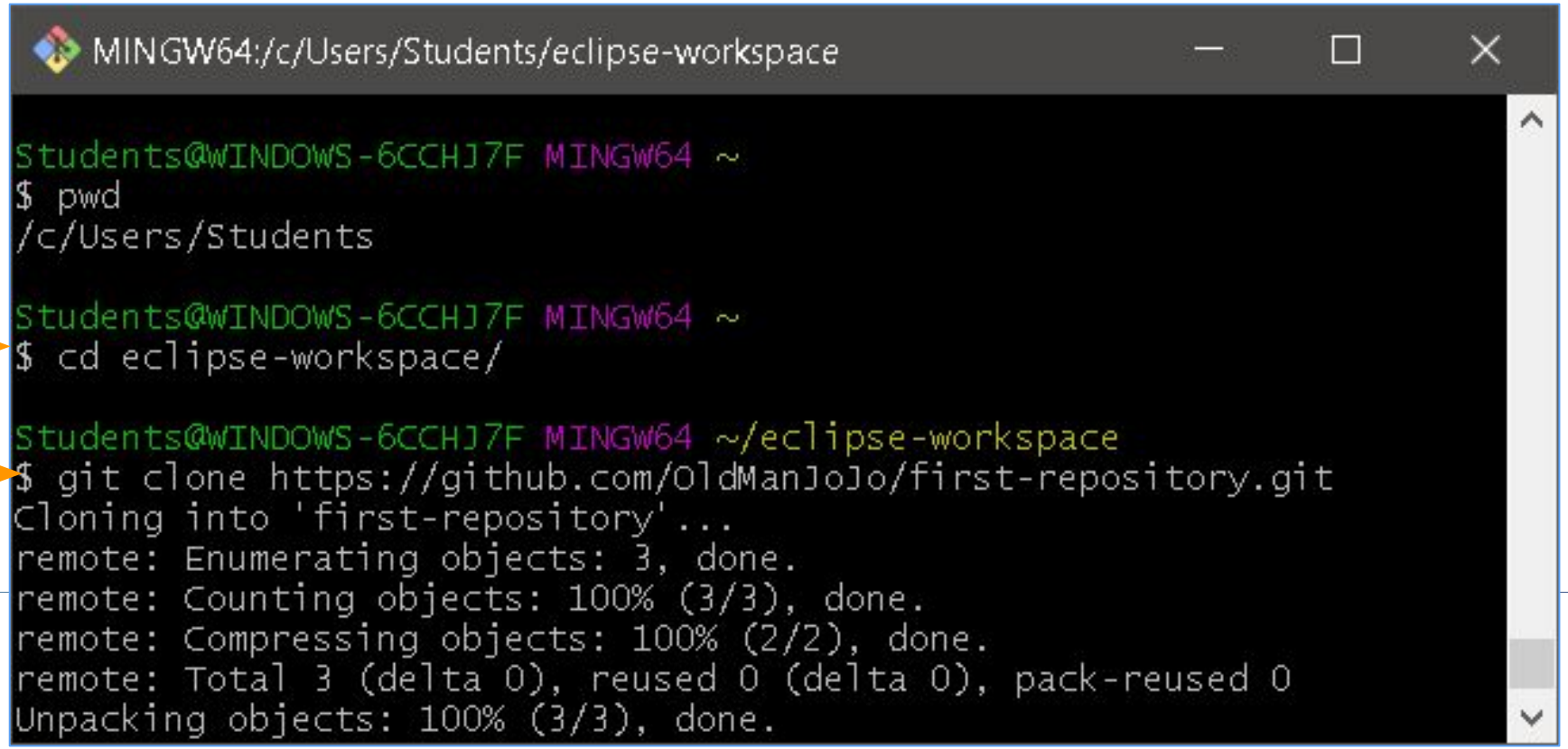
Clone operations create the instance of the repository from another instance. Clone operations not only check out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository. The only time networking gets involved is when the repository instances are being synchronized.

Clone a local version of a repository, including all commits and branches.

Link Your Repositories Using Git Clone

Open Git Bash and make sure that the current working directory is the directory where you want the cloned directory downloaded. In this case, we want to be in the "eclipse-workspace" directory.

Type **git clone**, and then enter/paste your repository URL.

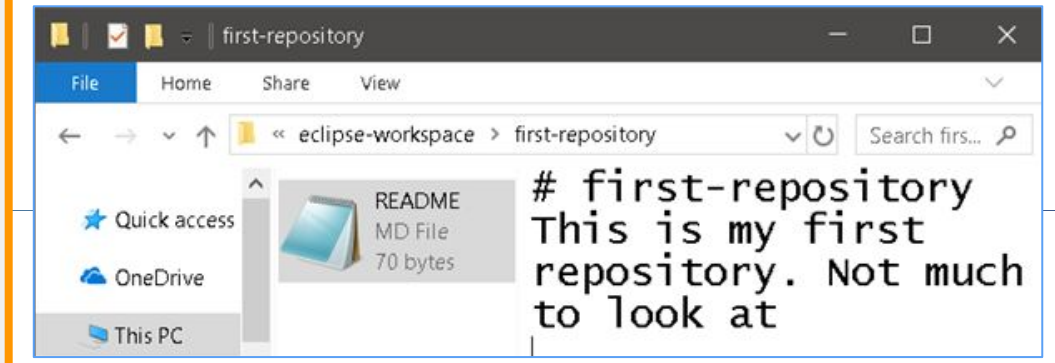


Finally, press **Enter**. Your local clone will be created.

Git Clone


```
MINGW64:/c/Users/Students/eclipse-workspace/first-repository

Students@WINDOWS-6CCHJ7F MINGW64 ~
$ pwd
/c/Users/Students

Students@WINDOWS-6CCHJ7F MINGW64 ~
$ cd eclipse-workspace/

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace
$ git clone https://github.com/OldManJoJo/first-repository.git
Cloning into 'first-repository'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace
$ cd first-repository/

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository
(master)
$ ls
README.md

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository
(master)
$ cat README.md
# first-repository
This is my first repository. Not much to look at

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$
```

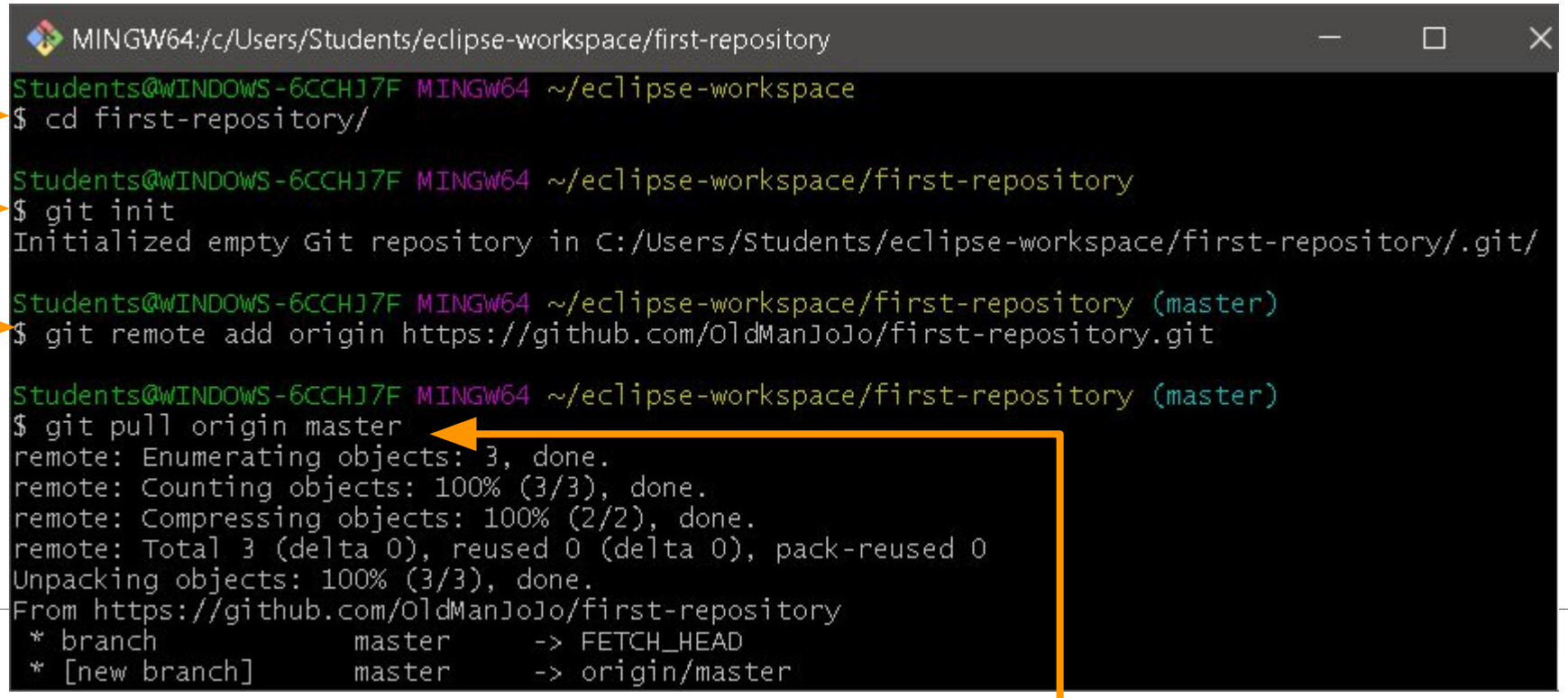
After executing git clone, a directory with the content of your cloned/remote repository is created/downloaded in your local workspace.



TEKsystems
Own change

Link Repositories Using Remote Add

Open Git Bash and make sure that the current working directory is the correct directory for this project. In this case, we want to be in the "first-repository" directory.



Use **git init** to initialize git on that folder.

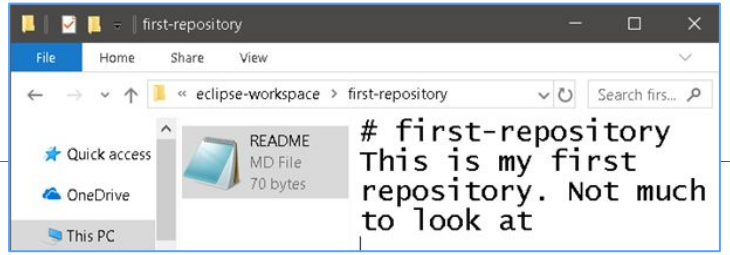
Use git remote add origin <repository Url> to link with your remote repository.

Finally, use git pull origin <branch_name>

Link Repositories Using Remote Add (continued)


```
MINGW64:/c/Users/Students/eclipse-workspace/first-repository
Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace
$ cd first-repository/

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository
$ git init
Initialized empty Git repository in C:/Users/Students/eclipse-workspace/first-repository/.git/

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$ git remote add origin https://github.com/OldManJoJo/first-repository.git

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$ git pull origin master
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/OldManJoJo/first-repository
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$ ls
README.md

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$ cat README.md
# first-repository
This is my first repository. Not much to look at
```

After executing git pull, the content of your linked repository is copied in your local workspace.

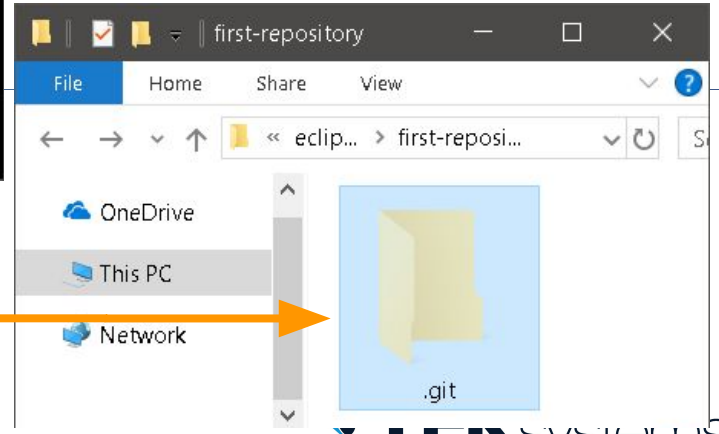


TEKsystems
Own change

What is “git init” Command


```
MINGW64:/c/Users/Students/eclipse-workspace/first-repository

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository
$ ls -A

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository
$ git remote add origin https://github.com/OldManJoJo/first-repository.git
fatal: not a git repository (or any of the parent directories): .git

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository
$ git init
Initialized empty Git repository in C:/Users/Students/eclipse-workspace/first-repository/.git/

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$ ls -A
.git/

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$ git remote add origin https://github.com/OldManJoJo/first-repository.git

Students@WINDOWS-6CCHJ7F MINGW64 ~/eclipse-workspace/first-repository (master)
$
```

Many other Git commands are not available outside of an initialized repository, so this is usually the first command you will run in a new project.



The **git init** command creates a new Git repository. It can be used to convert an existing, un-versioned project to a Git repository, or initialize a new, empty repository.

What is Git Remote?

The Git remote command lets you create, view, and delete connections to other repositories. Remote connections are more like bookmarks than direct links into other repositories. Instead of providing real-time access to another repository, they serve as convenient names that can be used to reference a not-so-convenient URL.

Remote Add


```
$ git remote add origin
https://github.com/bsteveb/repo-1.git
```

The git remote add command takes two arguments:

- A remote name (e.g., "origin").
- A remote URL, which you can find on the "clone or download" button on your Git repository.

To **add** a new **remote**, use the **git remote add** command on the terminal in the directory in which your repository is stored.

Git Commands

- **git status** - Gives the overview of files.
- **git add** - Adds file for commit.
- **git commit** - Saves for upload/push.
- **git push** - Uploads to remote server.
- **git pull** - Brings the change from remote to local.
- **git branch** - Sees a list of branches.
- **git checkout** - Changes branch.

GitHub Log

You can tag a version of your repository in any state. *Checkout* allows you to view/use another version.

```
$ git log --pretty=oneline
26c863dab1269d36a04c1e309dab134e1b633b42 (HEAD -> main, tag: v1.4, origin/main,
origin/HEAD) Merge pull request #1 from bsteveb/test2222
bee4c60a65839aacd6fc8be5b5354123329b088c (origin/test2222) Create Lalala
dc3d7e321b8b8d6070a71dded3e33bbe7b29d86b Create passwd
1d17275fdc90ebef408110157765 08ad79aeeb51 Create HelloWorld.yml
5cd503244adcb0b2fef9fe0b36843d4cc3e9ca82 Initial commit
$ git tag v1111 dc3d7e321b8b8d6070a71dded3e33bbe7b29d86b
$ git tag
v1.4
v1111
$ git checkout v1111
Note: switching to 'v1111'.
```

GitHub Branch

You can create branches on a GitHub project just like in a local repository. They can be pulled or tested independently before merging them back into your main branch.

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/bsteveb/repo-1
  Push  URL: https://github.com/bsteveb/repo-1
  HEAD branch: test2222
  Remote branches:
    main     tracked
    test2222 tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)

steve@HPLaptop MINGW64 ~/code/repo-1 (main)
```

Branches and Checkout

Checkout allows you to view/use another version.

```
$ git branch
* main
  test2222
$ git checkout v1111
Note: switching to 'v1111'.
```

Forking

In the open-source world, it is very common for a codebase, once reaching a certain point, to split into two distinct projects — each with a different goal (though still a shared ancestry).

The most canonical example is the Linux codebase. Today, Linux has many forks (e.g., RedHat), all stemming from a shared historical baseline.

What is important to note is that these flavors are not just temporary development paths — they are codebases with distinct identities, with no intention of re-integration with one another.

Branch or Fork?

Forking tends to empower a divergent evolution of the codebase, which makes it ideal for an environment that is looking to empower broad experimentation on a theme.

Branches are always intended to be 'convergent' development paths. Branches are ephemeral by their very nature. A branch is really nothing more than a pointer at the head of a commit lineage. Both this pointer and the branch are eventually destroyed and purged from the Git history after the branch is merged into its origin/master.

Saving Changes

What goes here other than info about pushing and/or committing?

Git Diff

The “git diff” command is very handy when you want to see the difference between your local copy of a repository and the remote repository.

Git Bash vs. Git Shell

Git Bash and Git Shell are two different command line programs, which allow you to interface with the underlying git program. Bash is a Linux-based command line (which has been ported over to Windows), while Shell is a native Windows command line. You can use either of them. They will have different auxiliary commands. For example, Bash has "is" instead of "dir."

Hands-On LAB - Git and GitHub

Complete the **LAB - 302-2.1 - Hands-on Activity Git** and GitHub, you can find this lab on Canvas under Guided lab section.

Practice Lab Resources

1. Interactive Git Branching
2. Interactive Git Visualization
3. Visualizing Git Concepts with D3

References

- https://git-scm.com/
- https://www.atlassian.com/git/tutorials/what-is-version-control
- https://docs.microsoft.com/en-us/azure/devops/learn/git/what-is-git
- http://www.differencebetween.net/technology/difference-between-git-and-github/
- https://www.atlassian.com/git/tutorials/comparing-workflows
- https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html
- https://guides.github.com/activities/hello-world/
- https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes
- https://www.atlassian.com/git/tutorials/syncing
- https://www.edureka.co/blog/git-tutorial/
- https://www.vogella.com/tutorials/Git/article.html

Questions?

End of Section