**RStudio and GitHub**

# **Overview of Git and GitHub**

**Git** and **GitHub**, which are tools used for **version control** in software development and data science. Here’s a simplified explanation and summary:

Simple Explanation:

* **Version Control**: Think of it like keeping track of changes to a document. If you make a mistake, you can go back to an earlier version easily. It also helps when multiple people are working on the same document, so everyone can see what changes were made.
* **Git**: This is a free tool that allows you to manage versions of your projects. It lets you keep a copy of your project on your computer and share it with others.
* **GitHub**: This is a website where you can store your Git projects online. It makes it easy to collaborate with others.

Key Terms:

* **Repository**: A folder that contains your project files.
* **Fork**: A copy of someone else's project that you can modify.
* **Pull Request**: A way to ask someone to review your changes before they are added to the main project.
* **Working Directory**: The files on your computer that are part of a Git project.

Basic Git Commands:

* **git init**: Create a new repository.
* **git add**: Stage changes to be saved.
* **git commit**: Save the staged changes.
* **git status**: Check the current state of your project.
* **git log**: View the history of changes.

Summary:

The video explains how Git and GitHub help manage changes in projects, making collaboration easier. It introduces basic concepts and commands that are essential for using Git effectively. For hands-on practice, it suggests visiting GitHub's resources.

# **Introduction to GitHub**

**GitHub** and its role as a source repository for developers. Here’s a simplified explanation and summary:

Explanation:

* **GitHub** is a platform where developers can store and manage their code.
* It was created to replace an earlier system called **BitKeeper** when it became paid.
* **Key Features of Git:**
  + **Distributed Development:** Each developer has their own copy of the entire project history, allowing them to work independently.
  + **Non-linear Development:** Developers can work on different features at the same time without interfering with each other.
  + **Version Control:** Git tracks changes to the code, so developers can see what has changed over time.
  + **Collaboration:** Developers can work together, review each other's code, and merge changes easily.

Summary:

* GitHub is an online service for hosting Git repositories, which store code and track changes.
* It allows developers to collaborate effectively and manage their projects with features like version control and branching.
* Git is designed for tracking source code and supports agile development, making it easier for teams to work together.

# **GitHub Repositories**

**sign up for a GitHub account** and **create a repository**. Here’s a simplified explanation and summary:

Simple Explanation:

1. **Signing Up for GitHub**:
   * Go to the GitHub website.
   * Choose a username, enter your email, and create a password.
   * Complete a quick test to prove you're human.
   * Select a free personal account.
   * You may skip some questions about your work and interests.
   * Confirm your email to activate your account.
2. **Creating a Repository**:
   * A repository is like a folder where you store your project files, including code.
   * You can make it **public** (anyone can see it) or **private** (only you and invited people can see it).
   * Inside the repository, you can manage files, track issues, and collaborate with others.

Summary:

* The video teaches you how to create a GitHub account and set up a repository.
* A repository is essential for storing and managing your project files and collaborating with others.
* GitHub provides various tools to help you work effectively on your projects.

# **GitHub- getting started**

use GitHub to create and manage a repository. Here’s a simple breakdown:

1. **Create a GitHub Account**: First, you need to register and log in to GitHub.
2. **New Repository**: Click on the "+" icon and select "New Repository." You will need to:
   * Name your repository.
   * Optionally add a description.
   * Choose if it will be public or private.
   * Select the option to initialize it with a README file.
3. **Edit README**: After creating the repository, you can edit the README file by clicking the pencil icon. Make your changes and save them by committing the changes with a message.
4. **Create a New File**: You can add a new file by clicking "Add File" and then "Create New File." For example, you can create a Python file named firstpython.py.
5. **Upload Files**: You can also upload files from your local system by selecting "Upload files" and choosing the files you want to add.
6. **Commit Changes**: After making changes or uploading files, you need to commit those changes to save them in the repository.

Summary:

* The video teaches you how to create a GitHub repository, edit files, and commit changes using the web interface. You learn to create new files, edit existing ones, and upload files from your local system, all while managing your repository effectively.

# **GitHub- Working with Branches**

Simple Explanation:

* **Branches in GitHub**: Think of a branch as a copy of your project where you can make changes without affecting the main version (called the master branch).
* **Creating a Child Branch**: If you want to try out new ideas or make changes, you create a child branch from the master. This way, you can experiment without risking the main project.
* **Making Changes**: In the child branch, you can add files and make edits. For example, you might create a new file called test\_child.py and write some code in it.
* **Committing Changes**: After making changes, you save them with a commit message that describes what you did. This helps everyone understand the changes later.
* **Merging Changes**: Once you're happy with your changes, you can merge the child branch back into the master branch. This is done through a Pull Request (PR), which allows others to review your changes before they become part of the main project.

Summary:

* A **branch** is a snapshot of your project where you can make changes safely.
* You create a **child branch** from the master to test and edit without affecting the main project.
* After making changes, you **commit** them with a message.
* To finalize your changes, you create a **Pull Request** to merge the child branch back into the master, allowing for team review and collaboration.

# **Getting Started with GitHub Branches and Git Commands**

**[Optional] Getting Started with Branches using Git Commands**

You would typically use Git commands from your own desktop/laptop. However, so you can get started using the commands quickly without having to download or install anything, we are providing an IDE with a Terminal on the Cloud. Simply click the Open Tool button below to launch the Skills Network Cloud IDE and in the new browser tab that launches, follow the instructions to practice the Git commands. After completing this lab you will be able to use git commands to start working with creating and managing your code branches, including:

1. create a new local repository using **git init**
2. create and add a file to the repo using **git add**
3. commit changes using **git commit**
4. create a branch using **git branch**
5. switch to a branch using **git checkout**
6. check the status of files changed using **git status**
7. review recent commits using **git log**
8. revert changes using **git revert**
9. get a list of branches and active branch using **git branch**
10. merge changes in your active branch into another branch using **git merge**

# **Branching and Merging**

**Lab: Branching and Merging (Web UI)**

**Estimated time**: 15 minutes

**Objectives**

After completing this lab, you will be able to:

1. Create a branch
2. Commit changes to a child branch
3. Open a pull request
4. Merge a pull request into the main branch

**Prerequisites**

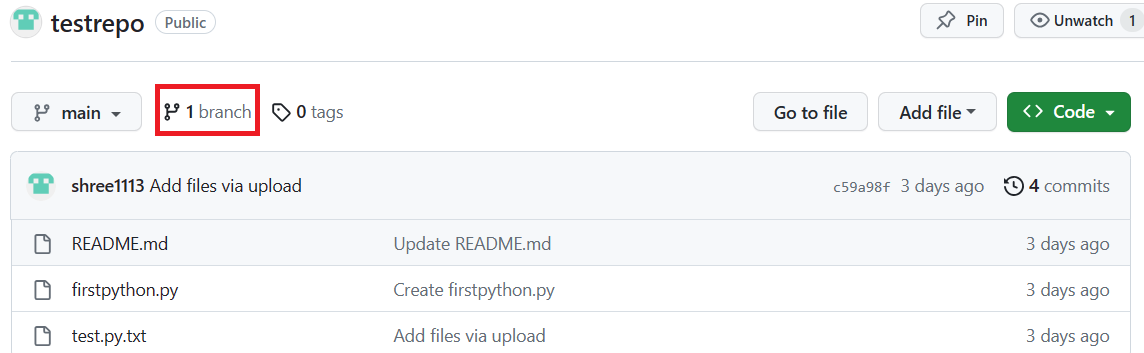
This hands-on lab requires you to have created a GitHub account with a repository in it, as covered in [Getting started with GitHub](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-CD0101EN-SkillsNetwork/labs/GitHubLabs/GitHub1_Getting_Started.md.html) lab.

*NOTE: In the past the default branch in your GitHub repo used the name master. Effective Oct 1, 2020, all new GitHub repositories will use the more inclusive term main as the name of the default branch instead of master.*

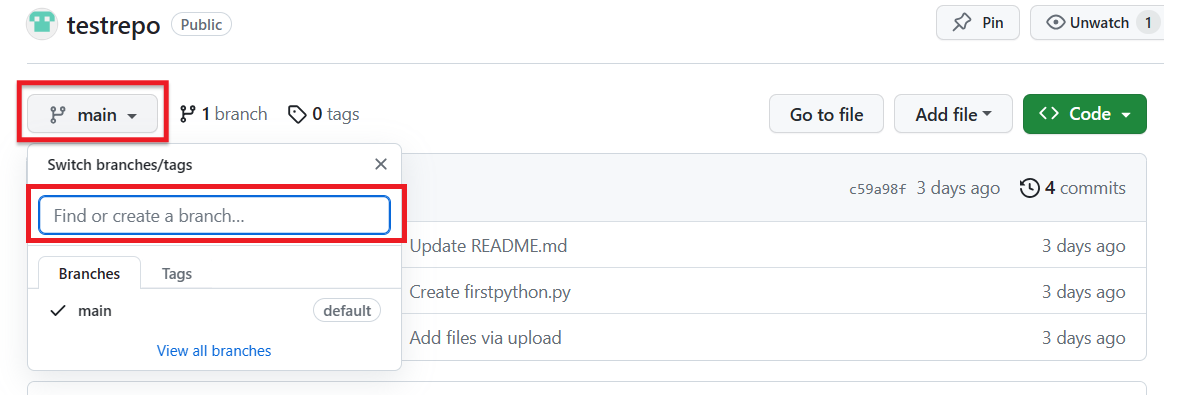
**1. Create a branch**

You can create or delete branches using your repository's GitHub web page. To add a branch to your repository, complete the following steps:

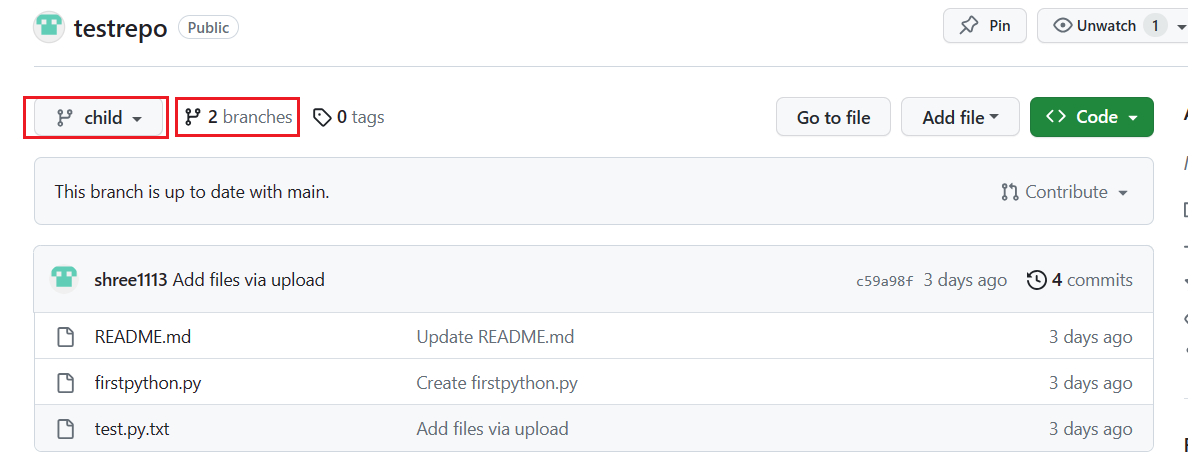
1. Go to your repository's main page. Note that when you created your repository, the one branch named **main** was created for you.



1. At the top of the file list, locate the **Branch** drop-down menu. (By default, the menu displays **Branch: main**.) Click the drop-down menu, type the name of the branch you want to create, and press Enter or return on your keyboard.



Your repository now has two branches: **Main** and **Child\_Branch**. You can click the drop-down menu to see your branches.

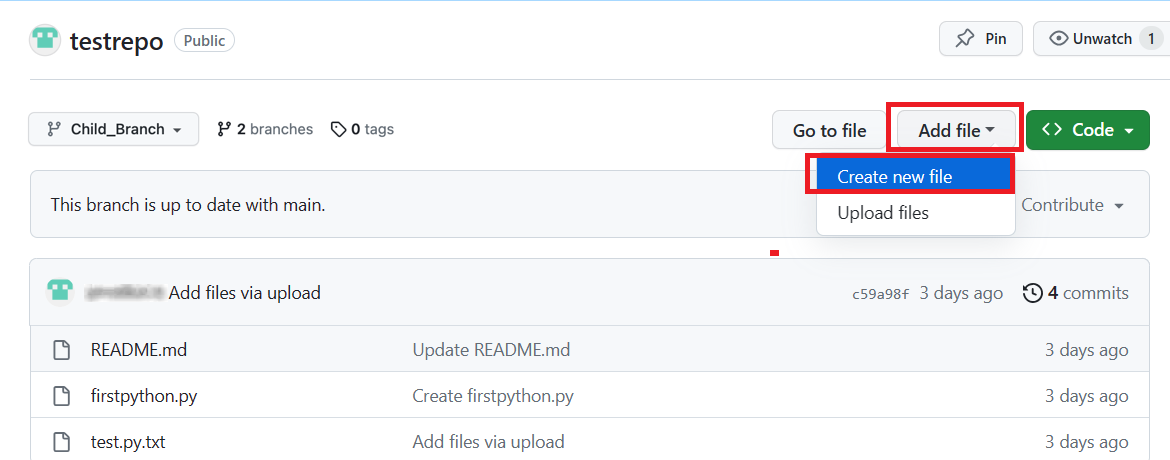


Any files that were in the **main** branch are reflected in the **Child\_Branch**. Note that when you add or edit a file in Child\_Branch, that change will not automatically reflect in the main branch

**2. Add a file to a branch**

To add a file to your new branch, ensure that the name you gave the new branch (which in the case of the example showcased is **Child\_Branch**) is displayed in the **Branch** drop-down menu and complete the following steps:

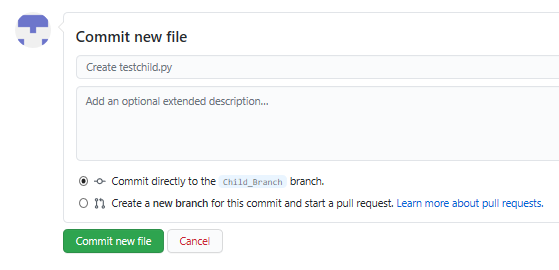
1. Click **Add file > Create new file** to create a file in the repository.



1. Type a name and extension for the file, for example, testchild.py, and add the following lines to the body of the new file:



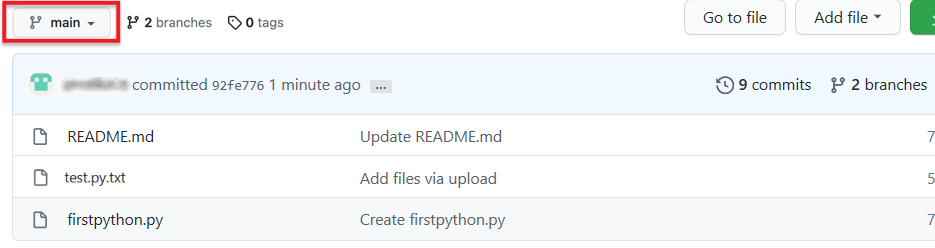
1. Scroll to the bottom of the page, add a description of the file you are about to add (note that the description is optional), and click **Commit**.



The file is added to your child branch.

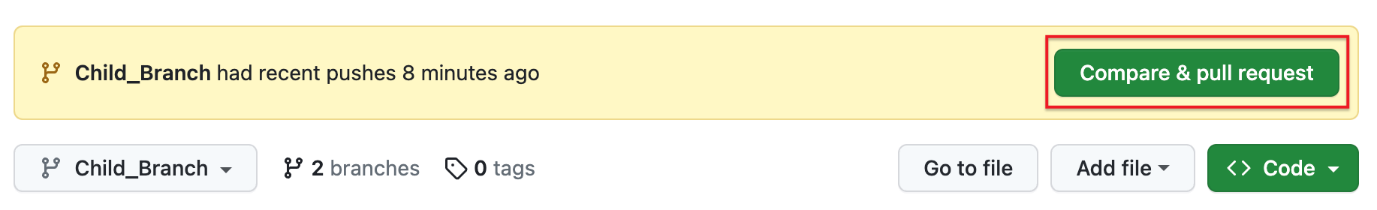
**3. Open a pull request**

The file that you added to your child branch is not automatically added to the **main** branch. (You can check this by using the **Branch** drop-down menu to go to the **main** branch; note that there is no testchild.py file in the file list).

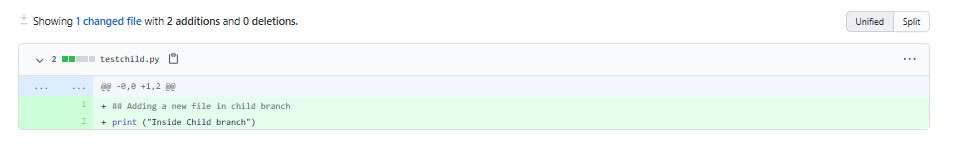


You can also compare the two branches and open a *pull request*, which will enable you to merge the changes that you've made in the child branch – in this case, adding a new file – to the **main** branch.

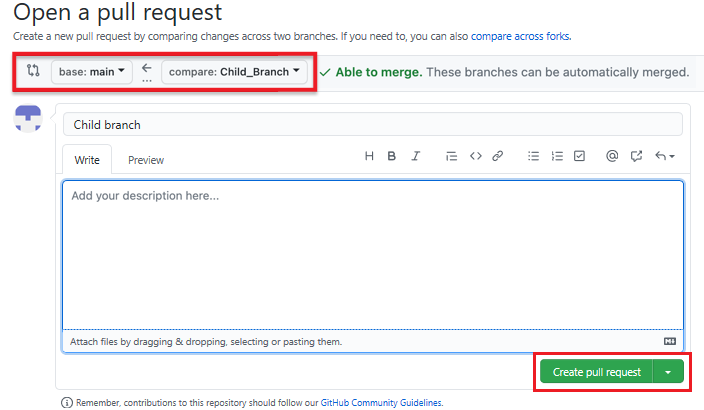
1. In **Child\_Branch**, click the **Compare & pull request** button.



1. Scroll to the bottom of the page and note that there is **1 changed file** listed and the changes are highlighted in green.



1. Scroll up and note that GitHub is comparing the **main** and **Child\_Branch** branches and that there are no conflicts between the two. Optionally, you can add a comment to the pull request. Click **Create pull request**.

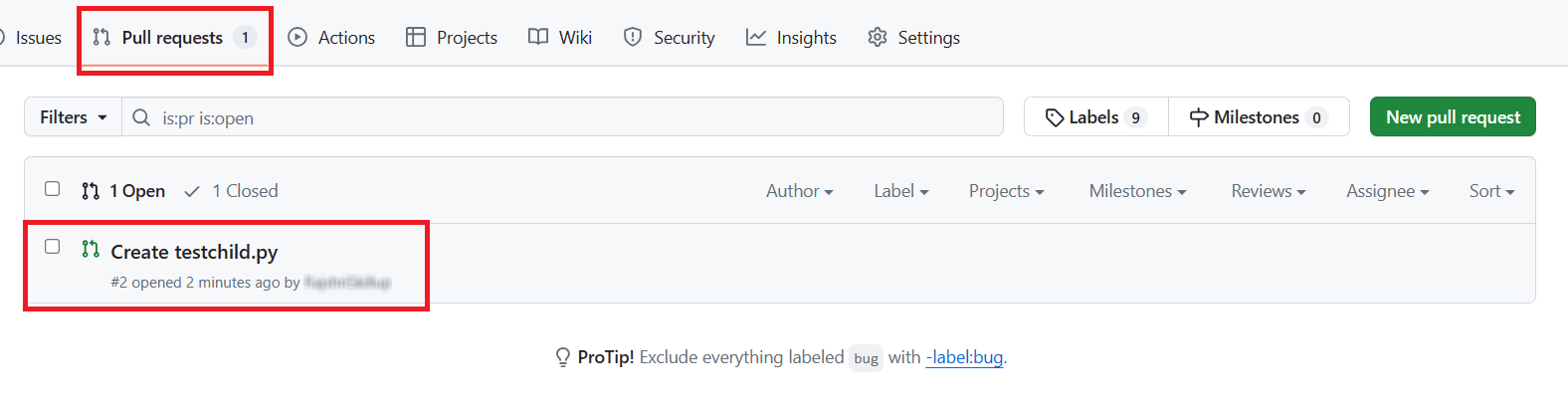


The pull request has been successfully created and is now ready to be merged by a repository administrator. For all the repositories that you create, you automatically have administrative rights.

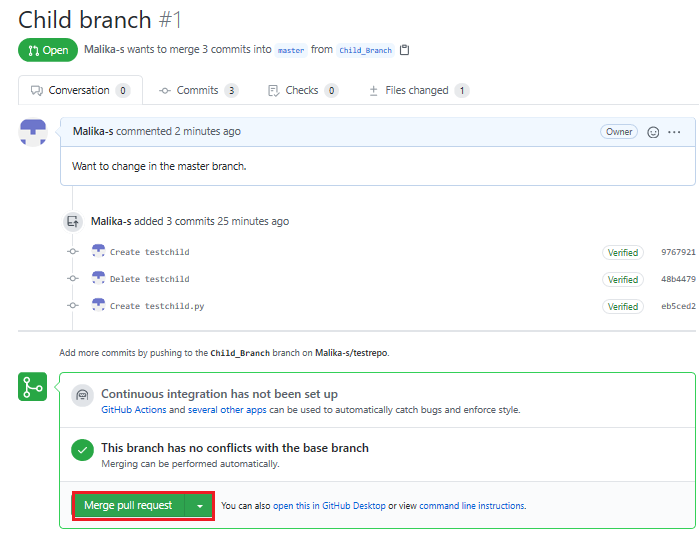
**4. Merge a pull request**

To merge a pull request into a project, complete the following steps:

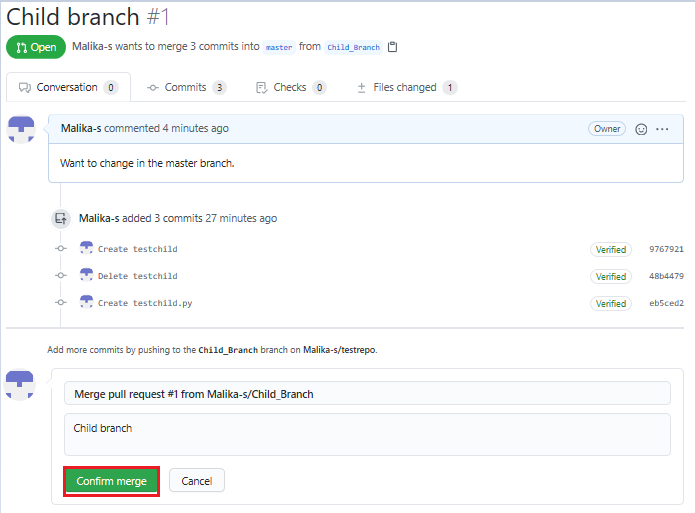
1. Click the **Pull requests** tab. A list of pending pull requests is displayed.



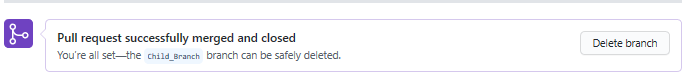
1. Click the pull request that you want to merge into the main project. Review the changes, click **Merge pull request** to accept the pull request and merge the updates. (You can optionally add a comment.)



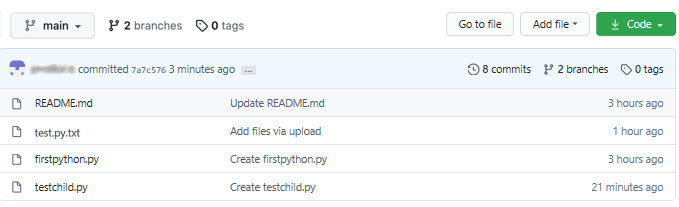
1. When you click **Merge pull request**, a **Confirm merge** button is displayed. Click that button to complete the merge.



The pull request has now been merged successfully. Note that you can delete the child branch because your changes have been incorporated into the **main** branch.



Check the list of files in the **main** branch to confirm that it now includes the file that you added in the pull request.



**Summary**

Congratulations! You've now learned how to create a branch, edit and commit changes in that branch, open a pull request, and merge the pull request into your main project. We encourage you to continue to experiment with branches and pull requests to become more familiar with the concepts and processes.

# **Module 5 Summary**

* The capabilities of R and its uses in Data Science.
* The RStudio interface for running R codes.
* Popular R packages for Data Science.
* Popular data visualization packages in R.
* Plotting with the inbuilt R plot function.
* Plotting with ggplot.
* Adding titles and changing the axis names using the ggtitle and lab’s function.
* A Distributed Version Control System (DVCS) keeps track of changes to code, regardless of where it is stored.
* Version control allows multiple users to work on the same codebase or repository, mirroring the codebase on their own computers if needed, while the distributed version control software helps manage synchronization amongst the various codebase mirrors.
* Repositories are storage structures that:
  + Store the code
  + Track issues and changes
  + Enable you to collaborate with others
* Git is one of the most popular distributed version control systems.
* GitHub, GitLab and Bitbucket are examples of hosted version control systems.
* Branches are used to isolate changes to code. When the changes are complete, they can be merged back into the main branch.
* Repositories can be cloned to make it possible to work locally, then sync changes back to the original.