Department of Computing

Software Development Tools (2024)

Summary

This tutorial has been prepared for students registered in COM221 - Advanced Programming and INF221 - Web Design and Development. Therefore, any concepts discussed and applied herein will be used in all subsequent modules requiring students to collaborate on software development projects. In this regard all the programs that you develop from now on must be uploaded to Github.

Objective (s)

The main objective of this tutorial is for students to understand GIT as a tool that can be used to facilitate collaboration on software development.

What is Git?

Version control systems, VCS', come in many types (1.1 Getting Started - About Version Control). Some are based on the client-server architecture, thus, are centralised. While others are distributed in nature. Git is an example of a version control system (VCS). Git is a simple text-based key-value data storage mechanism built on the client-server architecture. It stores and addresses any content for later retrieval. Any content may be stored in its filesystem and a specific key to this content returned for later retrieval. Compared to other types of VCS', Git is faster, smarter, more flexible and safer to use.

Advantages

The following are some of its advantages in software development:

- Tracking projects history
- Helps to manage multiple versions of a single project through the use of branches.
- Helps software developers to share code and collaborate on software projects. Coordinates teamwork - centre of the project for people and tools.
- Its services like github, bitbucket, gitlab and others, help developers and product managers to effectively manage software projects: managing software components integrations, system building and deployment.

Disadvantages

Git like any VSC tool also has its own shortcomings:

- Not ideal for binary files
- Not user friendly. To a beginner, Git may initially seem hard to learn but becomes easy to use when mastered. Nonetheless, its concepts are better understood through practice and use during software development.

Using Git & Its Services - Ecosystem

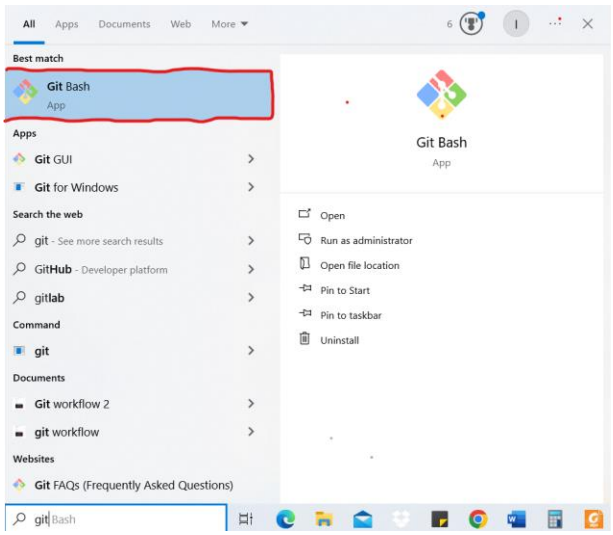
Almost all modern software development tools support Git.

- Developer Tools - any modern software development environment - IDE ie. Visual Studio, Visual Studio Code, Intellij.
- Configuration and Team Tools - Jenkins, Travis CI, Jira.
- Cloud providers - Azure, AWS, Heroku.
- Git based services - Github, Bitbucket, Gitlab.

Task 1 - Installation

For Git to be used during software development, one has to download and install it on a personal computer. Several operating system (OS) specific build executable and/or installable files can be downloaded from the official website or documentation site. ***This lab will demonstrate Git installation for Windows operating systems***, but similar steps can be followed to install the software for any other OS (1.5 Getting Started - Installing Git). Download the latest Git from the official website and follow subsequent steps to install and use Git on your computer.

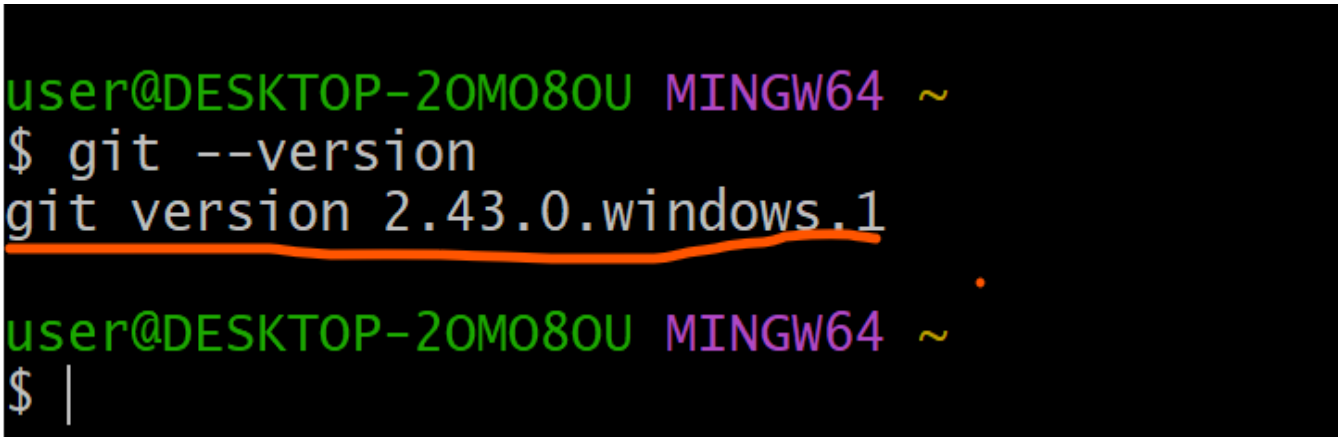
After successful git installation, proceed to open Git as follows:



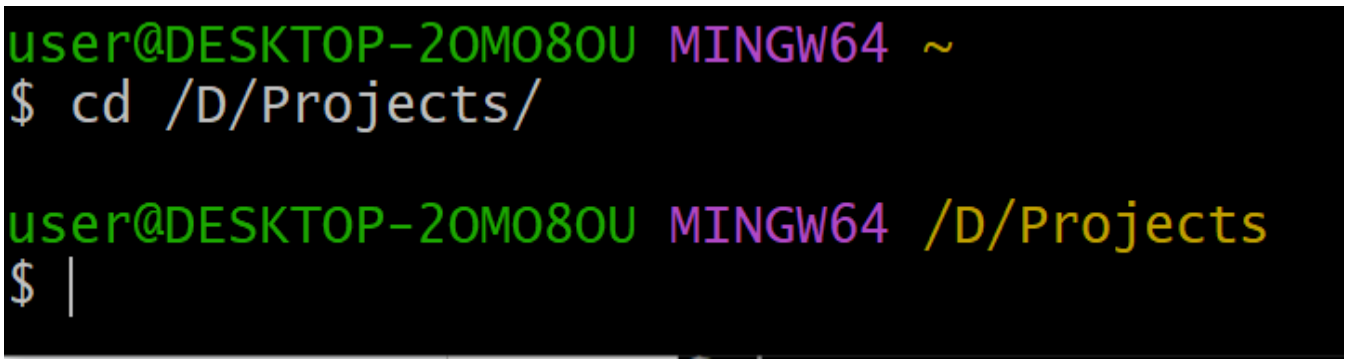
Proceed to run the following command in the computer's command prompt or the terminal window opened as above.

git --version

The output should be similar to the following information:



Change directory into any folder of choice within your computer. For example, the following command changes directory from the current location into the **Projects** folder.



While in the **Projects** folder, run the following command to list all contents of the current folder.

The above commands demonstrate how one uses the terminal window to navigate the local computer's file system. Refer to 1st year lab materials for further information.

Git Repository

The source code and any related files are kept in a unique file system structure separate from the local computer's file system. This unique file structure is called a Git repository. In order to create a Git repository, create a folder, call it **MyFirstProject**, in your computer. This folder will house all the project's source code. Then run the following command inside this folder.

git init

The command above initialises a directory or a folder as a Git repository. After running the above command, check that a **.git** folder and **.gitignore** file have been created. The **.git** folder has its own file structure and will be used for tracking any file you will add to the folder and any changes you will make to any file inside this folder. The .gitignore file contains lines of file or folder paths indicating which files or folders inside the **MyFirstProject** folder to be ignored ie. not tracked by git. These ignored files or folders will only exist inside your local computer.

Task 2 - Identity

In order to effectively use git to track your source code, there is a need for setting up-identity management mechanisms - to track ownership to which project's files or directories. To configure user identity on your local computer, successively run the following commands inside your terminal.

git config --global user.name "ShukranIsaac"
git config --global user.email "imwakabira@unima.ac.mw"

Verify that configurations have been set successfully, by running the following. Make sure that the set username and email values are part of the list of configurations.

git config --list

Thus far, all local configurations have been successfully set. However, there is a need to be able to upload the local source code, tracked by git on the local computer, to a remote computer on the internet - web service.

Git has several web services which can be used to track and store all project versions i.e. GitHub, GitLab, and Bitbucket. Thus far what has been installed on your computer needs to be connected to one or some of these web services. For this lab, we will use Github - click to follow the link. Therefore, proceed to create an account on github using the same email address and username as in the identity configuration section above, in this case, I have used **ShukranIsaac** and **imwakabira@unima.ac.mw**, because this is what is configured on my local computer.

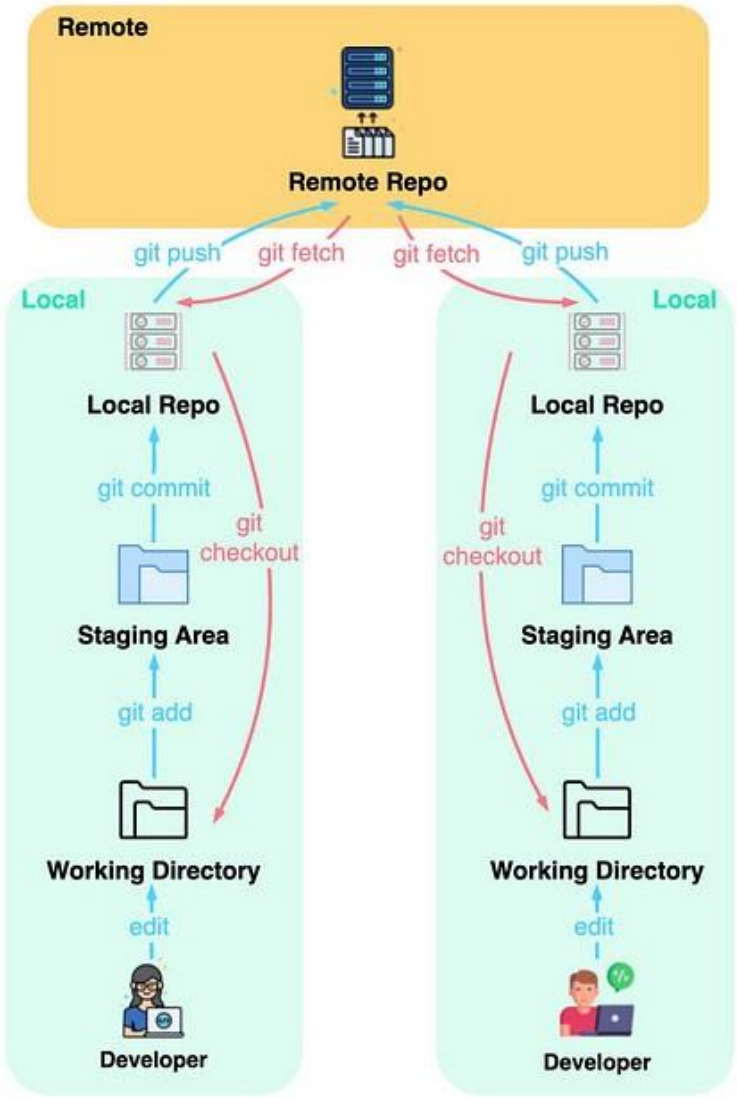
Git Model

The Git repository contains about 4 objects. An object can be a **file**, a **directory**, a **commit** or a **tag**. Files and directories are represented by **BLOBs** and **TREEs**, respectively. **TAGs** act as named references to other objects. Tags may contain additional descriptive information about other objects - *metadata*. Every object can be uniquely identified by a SHA-1, 40 HEX digit number. A **COMMIT** object contains the following:

- a tree id
- zero or more parents, which are commit ids
- an author (name, email, date)
- a committer (name, email, date)
- a log message

How Does Git Work?

As previously stated, git isolates its contents from the local computer's file system.
Its distributed nature allows multiple people to work on the same project simultaneously and independently. The following figure shows how Git works.



How Git Works (***Source***: blog.bytebytego.com)

The following concepts, as dipicted in the figure above, briefly describe how Git works. **Repository** - is a collection of files and their history. It resides on your local machine (local repository) and/or on a remote server (remote repository), such as GitHub or GitLab. **Working Directory** - when you create a new repository or clone an existing one, Git creates a working directory on your local machine. This directory contains the files of the project and allows you to modify them. **Staging Area** (*Index*) - before committing (**`git commit -am "your-message-here"`**) changes to the repository, you need to add (**`git add .`**) them to the staging area. This is like preparing a package for shipment. You can stage specific files or all the changes in the working directory. **Committing Changes** - once changes are staged, you commit them to the repository.

A commit is a snapshot of the project at a particular point in time. Each commit has a unique identifier (hash) and contains information about the changes made, such as who made them and when (discussed earlier). **Branches** - Git allows you to create branches to work on new features or experiments without affecting the main codebase. Branches are lightweight and easy to create, and you can switch between them easily. **Merging** - when you're done with a feature or bug fix on a branch, you merge it back into the main branch (usually `master` or `main`). Git automatically merges the changes, but if there are conflicts (i.e., changes in both branches that cannot be automatically merged), you'll need to resolve them manually. **Remote Repositories** - in addition to your local repository, you can have one or more remote repositories hosted on servers like GitHub, GitLab, or Bitbucket. You can push your changes to these repositories to collaborate with others and share your work. **Pulling Changes** - if others have made changes to the remote repository, you can fetch those changes and merge them into your local repository using the git pull command.

In summary, Git tracks changes to files in your project, allows you to work on different features simultaneously using branches, and facilitates collaboration with others through remote repositories. Its decentralised nature and powerful branching model make it an essential tool for modern collaborative software development.

References

- What Is a GIT Repository? - GeeksforGeeks
- Git - Reference
- Understanding Git
- 2.3 Git Basics - Viewing the Commit History