Git Basics

Site:	Edge Academy	Printed by:	Riddhi More
Course:	C386 Foundations of SRE	Date:	Wednesday, 27 March 2024, !
Book:	Git Basics		

Table of contents

Overview

Throughout this course, you will use Git and GitHub to manage the files you create. Git is a version control software tha
manage updates to files, while GitHub is an online repository that allows developers to back up their files to a server and shar
with others.

Learning Objectives

By the end of this module, you will be able to:

- Explain what version control software does and give examples of how developers use it.
- Set up Git to manage files in a folder on your computer.

What is Git?

Git is a distributed **version control system** (VCS) or **source control system**. A VCS is an application that keeps track and all the changes that are made to it. It also allows us to share our code with others and collaborate with one another withou overwriting each other's changes. In this course, we are going to give you a brief overview of Git and teach you how to save y code to a repository.

Learning Objectives

By the end of this lesson, learners will be able to:

1. Describe the benefits of source control in software development processes.
2. Explain the purpose of Git.

Why Source Control?

Source control is an integral part of the software development process. The proper use of source control can be the difference between having a copy of files when they appear to be lost and really losing the files. Hard drive failures, files being accidenta overwritten, and other catastrophes are hard to avoid; they just happen sometimes.

Source control management systems are designed to keep history and revision information about the files and projects storec a repository. By storing this information, we have a complete history of everything that happened to the projects within the rep In the event of a catastrophe, we can recover a previously saved version. Also, by saving our changes, we can easily share w other developers and collaborate on projects regardless of location.

It is advised that you always use some kind of source control for all projects. Git is a great, easy way to get started with source control, and you can continue its use in enterprise and large-scale projects. It grows with your needs.

Git's Origins

Git was created by Linus Torvalds, who is known as the father of Linux. Torvalds became frustrated with the processes used b Linux development community and wanted to create a better way to share code with other open-source developers who were working on the platform. Git came into being in 2005 with five primary goals: speed, simple design, support for parallel branch fully distributed workflows, and the ability to handle large projects.

Since its inception, Git has evolved and matured into a world-class source repository and has been adopted by both the open community and enterprise development teams.

Why Choose Git?

There are many reasons for choosing Git.

- Git is cross-platform: it works on Windows, Mac, and Linux.
- Git is an open-source project and does not have any licensing costs. You may need licensing if you choose to use ar established online provider or if you choose to purchase third-party tools, but the system itself does not have licensir
- Git is also easy to learn and can be very simple, but it offers complex features that you can take advantage of if you them or as you grow and require more of their use.

With its flexibility and availability, Git makes for a very compelling choice over other version control software options.

Set Up Git

Git is free and open-source software. It is one type of **version control software** (VCS) or **source control manager** (S
that developers use to manage code at different stages of development. Git can be installed on Windows, Linux, or macOS.

Once you have installed Git, you can set your e-mail address and username so that this information is used with each of the repositories you create locally. This will be saved in the Git configuration at the machine level and used for all repositories on t
machine.

Learning Objectives

By the end of this lesson, learners will be able to:

- Install Git on a personal computer.
- Set up Git identification on your computer.

Getting Started

To set up Git (and GitHub in a later step), you will be asked to provide an existing email address for the account. In most case
should use the **same** email address for both Git and GitHub. We also recommend that you use the email address associated
Engage account you are using for this course.

If you already have a GitHub account, make sure that you can log into it and know what the email address is for that account.
don't yet have a GitHub account, we provide instructions to set one up in a separate lesson.

If you use multiple computers on a regular basis, Git with GitHub is an excellent way to ensure that your work is synchronized
your computers. While you must install Git on at least one computer for this course, you can use these instructions to install G
additional computers if you wish.

Install Git

Git is installed as a service on your computer. Once it is installed, it will start automatically each time you log into your c
then it will run automatically in the background as you work. This means that once it is set up, you will not see a separate winc
Git nor will you have to remember to open it when you need to use it.

Each operating system has its own process for installation, so use the "Getting Started - Installing Git" article in Git's documer
to select and follow the steps for your operating system.

For both Windows and macOS, we recommend using the appropriate installer file linked on that page to ensure that Git is inst
with all recommended settings.

When you install Git on Windows using Git For Windows, that package will also include Git Bash, a terminal application that y
use to run Git commands. MacOS users will use the built-in Terminal application for this purpose.

After you have installed Git, return to this page to finish setting it up.

Configuring Your Username and Email

Because Git is intended to store code on a server where multiple developers can work on it at the same time, it is impc
know who you are so that when you make changes to existing code, it can track the changes to you.

You should use the same name and email address that you use for your GitHub account.

To set up your Git identity on your computer:

1. Open Git Bash (a Windows application installed with Git For Windows) or the Terminal application (macOS or Linux)
2. Run the command to set your username:
   `git config --global user.name "username"`
3. Run the command to set your e-mail address:
   `git config --global user.email "email@domain.com"`

An example of this is:

```
git config --global user.name "Caitlin Stuart"
git config --global user.email "cstuart@email.com"
```

Pay close attention to the spacing and extra characters like periods. Git is a developer tool, and developer tools require precis
you add an extra space omit a period or quotation mark, the commands will not work.

Git provides extensive documentation, which is available through Git-SCM.com.

Basic Git Commands

Overview

Once you have your repository set up both on your computer and in GitHub, you will use your preferred Git tool (Git Bash or T
for example) to manage files stored in your GitHub repository.

Learning Objectives

By the end of this lesson, you will be able to:

- Add files in a repository to Git.
- Commit files to Git to prepare for upload.
- Use Git to pull files from a remote repository.
- Use Git to push files to a remote repository.
- Create branches within a repository.

Add, Commit, Pull, Push

The purpose of a repository is to add files and commit them, creating a save point in the history of your code. These save poir
how we ensure that files do not get lost, changes are saved, and developers can continue to collaborate on a single project.

In this section, we are going to focus on the four basic commands you will use. These commands represent the basic functior
Git. There are far more commands available, but you should not need them until you begin working with other developers in la
teams or complex situations.

Add

To add a file to the repository, we can use the **add command**. Open your Git command tool (Git Bash or Terminal for e
use the *cd* command to navigate to your local repository.

```
$ cd '[path to directory]'
```

Once you are there, you can run the following command:

```
$ git add --all
```

The use of the **--all** parameter ensures that all files in all subdirectories will be staged, including new files, modified files, and a
removed from the repository. Alternatively, you can name specific files, which can streamline things if the repo is large and you
only changed a few files.

It is important to note that even though a file may physically exist in the directory, it will not be tracked by Git and included in y
repository until it is "added" to the repository using the add command, so you should always run **git add** before running **git co**

Commit

The **git commit** command will create a new save point in the history of the repository. These are the points we review
when it comes to the repository. They provide the save points in the repository that allow us to return to the exact state of the
they existed at that moment we did the commit.

```
$ git commit -m "[MESSAGE]"
```

Replace `[MESSAGE]` in the command above with your description for this save point in the repository. This should be a relativ
short message that explains why you are committing this code or what state the code is in at that point in time. For example, i
have just added a Contact page to a website, an appropriate commit command might look like this:

```
$ git commit -m "new contact page in website"
```

Whether you are describing a new feature you added or a bug that you fixed, this information will help you and your collaborat
understand the purpose of the changes.

The Commit Message is Required

Each time you commit changes, you must include a message that describes changes made since the last commit. If you forget to include the `-m` option with the message, the command line window will open a text editor for that message. When this editor opens, it displays an empty window with the cursor at the top of the window. This is a Linux-based editor that can be difficult to navigate. By default, the content is read-only, so to enter the message, you may have to hit the "i" key to begin inserting characters. When you have finished making changes, hit the Esc key and then Shift + : (colon) to move the insertion point to the command line at the bottom of the editor window. Enter the command `wq` to write the file and quit. This can be an unpleasant experience if you are not familiar with these tools, so it is best practice to include the `-m` modifier and a message when performing a commit from the command line.

Pull

We pull from the remote when we have or someone else has checked code into it from another computer and we need changes on our local machine. Pulling changes from a remote repository is quite easy. We run the **git pull** command.

```
$ git pull [repository] [branch]
```

The **git pull** command allows us to specify the remote repository and the branch within that repository, which is useful if we ar working in multiple repositories. If you are working in the original repository, then the easiest way to reference it is with the par **origin**.

Branches are a little more complicated. By default, the initial branch of any repository is called **main**, and as a starting point, y use the command:

```
$ git pull origin main
```

You can leave off the repository and branch information, but it's a good habit to always include them to be sure you are getting files you expect to get.

Push

The **git add** and **git commit** commands only make changes within your local Git installation, and they will not change a
in the online repository. To send the committed files to the repository, you must use **git push**, whose syntax is similar to **git pu**

```
$ git push [repository] [branch]
```

To push files to the main branch of your original repository, the command is:

```
$ git push origin main
```

After pushing changes, you should see the changes in your GitHub repository and those changes will be visible to your instru

Branching

As you work on a project (or especially if you are working on a project in a team of developers), though, you may want new **branch** to work on so that your changes don't affect the main branch that other people may be working in. Essentially, a is a copy of a repository with its own name but that Git still associates with the initial main repository. After you have made cha the code in your own branch, you can easily merge those changes to the main branch and make them available to everyone e the team.

Using a new branch involves two steps: create a branch and check it out in Git. The commands look like this:

```
$ git branch [branch-name]
```

```
$ git checkout [branch-name]
```

As a rule of thumb, while it is possible to use **git push** without specifying a repository and branch, you should always name th be sure you know where those files will show up in GitHub.

⚠ **Important** ⚠

The **git checkout** command changes your local Git installation to use that branch by default for all future pushes. If you use branches, it is important to always name the branch you want to use in any **git push** command.

If you want to check to see what branch is currently active in Git, use the `git status` command.

Summary

It is a good idea to get in the habit of always pulling your repository after you have done a commit but before you do a p
pattern of **add, commit, pull, push** will allow you to do the following:

1. **Add**: Start tracking new files and changes when you finish the code you are working on.
2. **Commit**: Create a save point for your changes in the repository.
3. **Pull**: Retrieve other code changes from the remote repository AND merge those changes with your own. This is impo
because if there are any conflicts between the code you have written and committed and the code that you have pull
the remote repository, you will need to fix it before you send it to the remote server.
4. **Push**: Send the committed code to the remote server.

Git has far more commands than the ones we covered here. It is a complex system with many options for tracking our code cl
over time and allowing collaborative work on application source code. However, using a simple workflow like the one we desc
here works well for most small teams and projects.