# Module 2: Intro to Git and GitHub

This module will introduce you to Git and GitHub. Because many developers who use our Linux offerings use Git and GitHub, it's important that you have a high-level understanding of these technologies.

The concepts you learn in this module can be applied to other version control systems as well.

## Prerequisites

The following prerequisites are necessary before participating in this Boot Camp.

* Install Git on your local machine.
* A free GitHub account.

## Goal

After completing this Boot Camp, you will have:

* An understanding of basic Git and GitHub concepts.
* A working knowledge of using Git and GitHub.
* An understanding of Git and GitHub terminology.

# What is Version Control?

All modern software is developed by a team of developers. (At Microsoft, most of our developers are Software Engineers.) At any specific time, different developers are working on many different areas of the source code. Some may be developing new features. Others may be fixing bugs against the current version of code. To handle this kind of workflow (and to make it easy to roll back to a previous version if new code breaks something), development teams use version control systems.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | You'll often see the terms *version control system* and *source control system* used interchangeably. They mean the same thing. |

# What is Git?

Git is a version control system developed in 2005 by Linus Torvalds, the creator of the Linux kernel. There are many version control systems available to developers, such as Mercurial, Subversion, Team Foundation Server, and more. However, Git is certainly the most popular.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | It's important that users always keep Git up-to-date! When security vulnerabilities are found, they are fixed quickly and users should always install those updates quickly. |

There are many reasons why Git is so popular, but some of them are:

* Everything in a Git repository is secured with an SHA1 hash, and every change can be traced to its source.
* Git is fast and efficient.
* Git is distributed, meaning that all developers have the entire source tree history on their development machine.

This last bullet point cannot be emphasized enough. This is one of the key benefits of Git. As much as we feel like we're always connected to the Internet, the truth is that we are often disconnected. A developer working on source code maintained in a Git *repository* (a repository is database of files in a project, along with all of the version control information) can continue working even when he or she is disconnected because the entire history of the source tree is contained on their computer. Changes that a developer makes are *committed* to the source tree. When the developer is connected again, he or she can then *push* all changes to the remote repository.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | The above paragraph is a simplistic description of a Git flow. You'll learn more later. You'll also get more information soon about some of the terminology involved in Git. |

# What is GitHub?

You already have a general sense of what Git is, and you know that developers eventually push their changes to a source control tree to a remote repository. GitHub is a website that provides a place to store your remote repository in the cloud. GitHub was recently acquired by Microsoft, but we have pledged to keep it as an open repository.

GitHub supports only the Git version control system, but because Git is the most popular version control system, GitHub is the most popular place to host your source code repository.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | Bitbucket is another popular service that can host your source code repository. In addition to Git, Bitbucket also supports Mercurial, another popular version control system. |

Developers can create a repository in GitHub and then use Git to interact with that repository. There are also applications that are native for both Windows and MacOS that developers can use to interact with GitHub.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | GitHub isn't just for source code! You can use it to manage any kind of file. In fact, all our public-facing documentation is managed in GitHub. You'll find out why a little later. |

# Repositories

Before we get into the flow of how you use Git, you need to understand the concept of Git repositories. A repository is just a folder on a computer that contains the files that are under version control. Git uses two different types of repositories; *bare* repositories and *no-bare* repositories.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML30c12ab.PNG | You will often hear people use *repo* to refer to a repository. |

A bare repository is where developers share the files under version control and the changes that have been made to them. A bare repository in Git is often referred to as the *central repository* or a *remote repository*.

Developers don't work on the files in the central repository. Instead, they create their own version of the repository on their own machine. (Remember, Git is a distributed version control system.) The local version of the repository is a no-bare repository, and it's where developers will make changes to files. Once they are ready to share their changes, they *push* them to the central repository.

# Typical Workflow

Now that you know about repositories, let's look at a simple workflow using Git.

## 1: Create the Central Repository

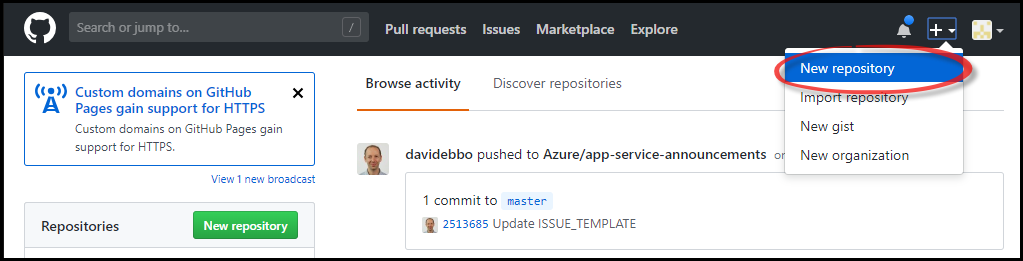
The central repository is located on a server. This can be a server within your network, but it can also be in GitHub or Bitbucket.

If you want to create a central repository on your own server, the first thing you do is change into the directory on the server where you want the central repository to be. You then run the following command to create the repository.

|  |
| --- |
| git init --bare *<repository\_name>*.git |

You don't have to use the .git extension, but standard naming conventions in Git recommend that you do. When you run this command, it will create a new directory called *<repository\_name>.git* in the folder on the server.

To create a central repository in GitHub, you simply browse to GitHub.com, login with your credentials, and then click the **+** button and click **New Repository** from the menu as shown below.



## 2: Create the Local Repository

To create a local repository so that you can work on files in the central repository, you *clone* the central repository.

To clone the repository, change into the local directory where you want the repository to be created and run the following command.

|  |
| --- |
| git clone https://github.com/*<username>*/*<repository\_name>* |

If the central repository is located on a server instead of GitHub, the command would use SSH to point to the central repository. Here's an example.

|  |
| --- |
| git clone ssh://*username*@*server\_name*/*path*/*repository*.git |

Once you clone the central repository, you'll have a local copy of all the files. You can then make changes on those local copies.

## 3: Commit Changes to the Local Repository

After you make changes to files in the local repository, you will need to *commit* those changes into your local repository. To do that, run the following commands.

|  |
| --- |
| git add *<filename>* #file to add to the repo  git rm *<filename>* #file to remove from the repo  git commit |

When you run *git add*, you are adding that single file to your local repository. This is called *staging a file*. It allows you to easily control exactly what gets committed to the repository. However, you don't have to stage files. You can also run the following command which will add any files that have been modified and remove any files that have been deleted.

|  |
| --- |
| git add -a |

-- or --

|  |
| --- |
| git add --all |

When you run *git commit*, you are committing the files that you added or removed to the local repository. This step is taken in preparation for pushing those changes back to the central repository.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | You can tell Git to ignore files you don't want to track by using a .gitignore file. See https://git-scm.com/docs/gitignore for more information. |

## 4: Pushing Changes to the Central Repository

Once you've committed your changes to your local repository, you'll likely want to push them to the central repository so that others working on your project will have your changes. To do that, you run the following command.

|  |
| --- |
| git push origin master |

This command pushes your changes to the *master branch*. The master branch is the main branch of a Git repository. There may be other branches as well. For example, while you're working on a new version of a release, you might create a branch called something like "development," and you might have developers push their change into that branch using the following command.

|  |
| --- |
| git push origin development |

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | To create a new branch, you *fork* an existing branch into a new branch in the same repository. |

## Handling Conflicts

You may be working in a scenario where multiple developers are working on the same files within a branch. In that scenario, it's possible that another developer has pushed changes to the same files you've edited, and in that case, the files in your local repository may not be up-to-date. When that happens, Git will notify you when you attempt to push your changes with an error message telling you that you need to merge the remote changes before you can push.

To merge remote changes to your local repository, you would perform a *pull* from the branch you're trying to commit to. For example:

|  |
| --- |
| git pull origin master |

In truth, you should also use the "--rebase" option when you're merging remote changes.

|  |
| --- |
| git pull --rebase origin master |

When you rebase, each of your local commits are sent to the central repository one-by-one. Using this method, you'll have a much easier time identifying exactly what changes were made.

If there's a conflict during the rebase, Git will return an error message showing you the conflict. To find details on the conflict, you run the following command.

|  |
| --- |
| git status |

The output of this command will show you the paths that were not merged in the rebase due to a conflict. You can then open those files, make any necessary edits, and then run the following commands to continue the rebase.

|  |
| --- |
| git add *<edited\_filename>*  git rebase --continue |

# Another Workflow - Pull Requests

What you've learned so far is only one way that you can use Git. There are other workflows that work better for different situations. For example, as I said earlier, all of our public-facing documentation exists within a Git repository called MicrosoftDocs/azure-docs. You can easily contribute to our documentation using another kind of workflow in Git known as *pull requests*.

Naturally, our documentation team isn't going to allow just anyone to commit changes to the documentation. Therefore, in order to contribute, you use the following workflow.

1. Fork the MicrosoftDocs/azure-docs repository into your own copy. (This creates your own copy of the repository that you can edit.)
2. Make any changes to the file.
3. Commit those changes to your forked repository.
4. Create a *pull request*.

The pull request notifies the owner of the file you changed that you would like to merge that change into the MicrosoftDocs/azure-docs repository. They can then accept and merge your change into the repository or they can reject your change.

GitHub makes this extremely easy. You can find all of the details on how to contribute to our docs in the README.md file in the repository located at https://github.com/MicrosoftDocs/azure-docs.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | GitHub uses *Markdown* syntax for writing files. That's why you'll see files with a .md file extension. You can find out all about Markdown at https://guides.github.com/features/mastering-markdown/. |