Unit 1-6

**Introduction**

Completed100 XP

* 3 minutes

Imagine you're a new software developer at a firm that writes avionics software for commercial airliners. Quality control is critical, and developers work in small teams using Git for version control. You've heard of version control, but you've never used Git, so you're eager to catch up!

You decide to build a website that lets you and your friends share pictures of your cats, so you can learn Git in a fun environment before bringing that knowledge to work. You set out to build the site by using Git to keep track of changes and keep all the source code files backed up in case the server goes down. But before diving head-first into Git, you must cover the basics

In this module, you'll get an introduction to version control, and Git. Git can seem a little cryptic at first, and it can even be frustrating at times. But if you learn it step by step, you'll find that there's a reason Git is quickly becoming the world's most popular version control system—not just for software developers, but also for teams that write documentation and collaborate on other work.

**Watch a video**

For an overview of the exercises in this module, see the video [Introduction to Git Recap](https://www.youtube.com/watch?v=9uGS1ak_FGg%3Fazure-portal%3Dtrue).

**Learning objectives**

In this module, you'll:

* Learn what version control is
* Understand distributed version control systems, like Git
* Recognize the differences between Git and GitHub and the roles they play in the software development lifecycle

**Prerequisites**

None.

Unit 2-6

**What is version control?**

Completed100 XP

* 5 minutes

A version control system (VCS) is a program or set of programs that tracks changes to a collection of files. One goal of a VCS is to easily recall earlier versions of individual files or of the entire project. Another goal is to allow several team members to work on a project, even on the same files, at the same time without affecting each other's work.

Another name for a VCS is a software configuration management (SCM) system. The two terms often are used interchangeably—in fact, Git's official documentation is located at [git-scm.com](https://git-scm.com/). Technically, version control is just one of the practices involved in SCM. A VCS can be used for projects other than software, including books and online tutorials.

With a VCS, you can:

* See all the changes made to your project, when the changes were made, and who made them.
* Include a message with each change to explain the reasoning behind it.
* Retrieve past versions of the entire project or of individual files.
* Create *branches*, where changes can be made experimentally. This feature allows several different sets of changes (for example, features or bug fixes) to be worked on at the same time, possibly by different people, without affecting the main branch. Later, you can merge the changes you want to keep back into the main branch.
* Attach a tag to a version—for example, to mark a new release.

Git is a fast, versatile, highly scalable, free, open-source VCS. Its primary author is Linus Torvalds, the creator of Linux.

**Distributed version control**

Earlier instances of VCSes, including CVS, Subversion (SVN), and Perforce, used a centralized server to store a project's history. This centralization meant that the one server was also potentially a single point of failure.

Git is *distributed*, which means that a project's complete history is stored both on the client *and* on the server. You can edit files without a network connection, check them in locally, and sync with the server when a connection becomes available. If a server goes down, you still have a local copy of the project. Technically, you don't even have to have a server. Changes could be passed around in e-mail or shared by using removable media, but no one uses Git this way in practice.

**Git terminology**

To understand Git, you have to understand the terminology. Here's a short list of terms that Git users frequently use. Don't be concerned about the details for now; all these terms will become familiar as you work your way through the exercises in this module.

* **Working tree**: The set of nested directories and files that contain the project that's being worked on.
* **Repository (repo)**: The directory, located at the top level of a working tree, where Git keeps all the history and metadata for a project. Repositories are almost always referred to as *repos*. A *bare repository* is one that isn't part of a working tree; it's used for sharing or backup. A bare repo is usually a directory with a name that ends in *.git*—for example, *project.git*.
* **Hash**: A number produced by a hash function that represents the contents of a file or another object as a fixed number of digits. Git uses hashes that are 160 bits long. One advantage to using hashes is that Git can tell whether a file has changed by hashing its contents and comparing the result to the previous hash. If the file time-and-date stamp is changed, but the file hash isn’t changed, Git knows the file contents aren’t changed.
* **Object**: A Git repo contains **four** types of *objects,* each uniquely identified by an SHA-1 hash.
  + - A *blob* object contains an ordinary file.
    - A *tree* object represents a directory; it contains names, hashes, and permissions.
    - A *commit* object represents a specific version of the working tree.
    - A *tag* is a name attached to a commit.
* **Commit**: When used as a verb, *commit* means to make a commit object. This action takes its name from commits to a database. It means you are committing the changes you have made so that others can eventually see them, too.
* **Branch**: A branch is a named series of linked commits. The most recent commit on a branch is called the *head*. The default branch, which is created when you initialize a repository, is called main, often named master in Git. The head of the current branch is named HEAD. Branches are an incredibly useful feature of Git because they allow developers to work independently (or together) in branches and later merge their changes into the default branch.
* **Remote**: A remote is a named reference to another Git repository. When you create a repo, Git creates a remote named origin that is the default remote for push and pull operations.
* **Commands**, **subcommands**, and **options**: Git operations are performed by using commands like git push and git pull. git is the command, and push or pull is the subcommand. The subcommand specifies the operation you want Git to perform. Commands frequently are accompanied by options, which use hyphens (-) or double hyphens (--). For example, git reset --hard.

These terms and others, like push and pull, will make more sense shortly. But you have to start somewhere, and you might find it helpful to come back and review this glossary of terms after you finish the module.

**The Git command line**

Several different GUIs are available for Git, including GitHub Desktop. Many programming editors, like Microsoft [Visual Studio Code](https://code.visualstudio.com/), also have an interface to Git. They all work differently and they have different limitations. None of them implement *all* of Git's functionality.

The exercises in this module use the Git command line—specifically, Git commands executed in Azure Cloud Shell. However, Git's command-line interface works the same, no matter what operating system you're using. Plus, the command line lets you tap into *all* of Git's functionality. Developers who see Git only through a GUI sometimes find themselves confronted with error messages they can't resolve, and they have to resort to the command line to get going again.

**Git and GitHub**

As you work with Git, you might wonder about differences between the features it offers, and the features offered on [GitHub](https://github.com/).

As mentioned earlier, Git is a distributed version control system (DVCS) that multiple developers and other contributors can use to work on a project. It provides a way to work with one or more local branches and then push them to a remote repository.

GitHub is a cloud platform that uses Git as its core technology. GitHub simplifies the process of collaborating on projects and provides a website, more command-line tools, and overall flow that developers and users can use to work together. GitHub act s as the remote repository mentioned earlier.

Key features provided by GitHub include:

* Issues
* Discussions
* Pull requests
* Notifications
* Labels
* Actions
* Forks
* Projects

To learn more about GitHub, see the [Introduction to GitHub](https://learn.microsoft.com/en-us/training/modules/introduction-to-github) Microsoft Learn module or the [Getting started with GitHub](https://docs.github.com/free-pro-team@latest/github/getting-started-with-github) help documentation.

The next step is to try out Git for yourself!

Unit 3-6

Before you can create your first repo, you must make sure that Git is installed and configured. Git comes preinstalled with Azure Cloud Shell, so we can use Git in Cloud Shell to the right.

**Configure Git**

1. In Cloud Shell, to double-check that Git is installed, type git --version:

BashCopy

git --version

**Tip**

You can use the **Copy** button to copy commands to the clipboard. To paste, right-click on a new line in the Cloud Shell terminal and select **Paste**, or use the Shift+Insert keyboard shortcut (⌘+V on macOS).

1. You should see output that looks something like this example:

OutputCopy

git version 2.7.4

1. To configure Git, you must define some global variables: user.name and user.email. Both are required for you to make commits.
2. Set your name in Cloud Shell with the following command. Replace <USER\_NAME> with the user name you want to use.

BashCopy

git config --global user.name "<USER\_NAME>"

1. Now, use this command to create a user.email configuration variable, replacing <USER\_EMAIL> with your e-mail address:

BashCopy

git config --global user.email "<USER\_EMAIL>"

1. Run the following command to check that your changes worked:

BashCopy

git config --list

1. Confirm that the output includes two lines that are similar to the following example. Your name and e-mail address will be different from what's shown in the example.

OutputCopy

user.name=User Name

user.email=user-name@contoso.com

**Set up your Git repository**

Git works by checking for changes to files within a certain folder. We'll create a folder to serve as our *working tree* (project directory) and let Git know about it, so it can start tracking changes. We tell Git to start tracking changes by initializing a Git repository into that folder.

Start by creating an empty folder for your project, and then initialize a Git repository inside it.

1. Create a folder named *Cats*. This folder will be the project directory, also called the working tree. The project directory is where all files related to your project are stored. In this exercise, it's where your website and the files that create the website and its contents are stored.

BashCopy

mkdir Cats

1. Change to the project directory by using the cd command:

BashCopy

cd Cats

1. Now, initialize your new repository and set the name of the default branch to main:

If you're running Git version 2.28.0 or later, use the following command:

BashCopy

git init --initial-branch=main

Or use the following command:

BashCopy

git init -b main

For earlier versions of Git, use these commands:

BashCopy

git init

git checkout -b main

After you run the initialize command, you should see output that's similar to this example:

OutputCopy

Initialized empty Git repository in /home/<user>/Cats/.git/

Switched to a new branch 'main'

1. Now, use a git status command to show the status of the working tree:

BashCopy

git status

Git responds with this output, which indicates that main is the current branch. (It's also the only branch.) So far, so good.

OutputCopy

On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)

1. Use an ls command to show the contents of the working tree:

BashCopy

ls -a

Confirm that the directory contains a subdirectory named *.git*. (Using the -a option with ls is important because Linux normally hides file and directory names that start with a period.) This folder is the Git *repository*—the directory in which Git stores metadata and history for the working tree.

You typically don't do anything with the *.git* directory directly. Git updates the metadata there as the status of the working tree changes to keep track of what's changed in your files. This directory is hands-off for you, but it's incredibly important to Git.

**Get help from Git**

Git, like most command-line tools, has a built-in help function that you can use to look up commands and keywords.

1. Type the following command to get help with what you can do with Git:

BashCopy

git --help

1. The command displays the following output:

OutputCopy

usage: git [--version] [--help] [-C <path>] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

reset Reset current HEAD to the specified state

rm Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

checkout Switch branches or restore working tree files

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc

merge Join two or more development histories together

rebase Forward-port local commits to the updated upstream head

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

Read through the different options available with Git and note that each command comes with its own help page, for when you start digging deeper. Not all these commands will make sense yet, but some might look familiar if you have experience using a VCS.

In the next lesson, you learn more about the commands you just tried and the basics of Git.

Unit 4-6

**Basic Git commands**

Completed100 XP

* 10 minutes

Git works by remembering the changes to your files as if it's taking snapshots of your file system.

We'll cover a few basic commands to start tracking files in your repo. Then, you'll save your first "snapshot" for Git to compare against.

**git status**

The first and most commonly used Git command is git status. You've already used it once, in the preceding exercise, to see that you had initialized your Git repo properly.

git status displays the state of the working tree (and of the staging area—we'll talk more about the staging area soon). It lets you see which changes are currently being tracked by Git, so you can decide whether you want to ask Git to take another snapshot.

**git add**

git add is the command you use to tell Git to start keeping track of changes in certain files.

The technical term is *staging* these changes. You'll use git add to stage changes to prepare for a commit. All changes in files that have been added but not yet committed are stored in the *staging area*.

**git commit**

After you've staged some changes for commit, you can save your work to a snapshot by invoking the git commit command.

*Commit* is both a verb and a noun. It has essentially the same meaning as when you commit to a plan or commit a change to a database. As a verb, committing changes means you put a copy (of the file, directory, or other "stuff") in the repository as a new version. As a noun, a commit is the small chunk of data that gives the changes you committed a unique identity. The data that's saved in a commit includes the author's name and e-mail address, the date, comments about what you did (and why), an optional digital signature, and the unique identifier of the preceding commit.

**git log**

The git log command allows you to see information about previous commits. Each commit has a message attached to it (a commit message), and the git log command prints information about the most recent commits, like their time stamp, the author, and a commit message. This command helps you keep track of what you've been doing and what changes have been saved.

**git help**

You've already tried out the git help command, but it's worth reminding you about. Use this command to easily get information about all the commands you've learned so far, and more.

Remember, each command comes with its *own* help page, too. You can find these help pages by typing git <command> --help. For example, git commit --help brings up a page that tells you more about the git commit command and how to use it.

Unit 5-6

**Knowledge check**

Completed200 XP

* 5 minutes

Top of Form

**1.**

**Which of the following scenarios is a common use case for a version control system?**

Deleting earlier versions of a project or file, so you know you are working only with the most current file or data.

Making experimental changes to your project in an isolated branch.

**Correct! Using branches to create different sets of changes to a project is a key use case for version control.**

Gathering feature requirements for a large project and communicating them to stakeholders.

**2.**

**What is another name for a version control system?**

Version management software (VMS)

Software control management (SCM) system

Software configuration management (SCM) system

**Correct!**

**3.**

**What’s the difference between Git and GitHub?**

Git lets you work with one or more local branches and push changes to a remote repository. GitHub acts as the remote repository, which is accessed through a website or command-line tools.

**Correct! Git is the tool you can use to work with a local branch and push changes to a remote repository. GitHub acts as the remote repository.**

Git is a distributed version control system (DVCS) that runs in the cloud. GitHub is an interface layer that provides access to Git technology.

Git is used by an individual contributor. GitHub is used by multiple contributors to simplify group development work.

**4.**

**What Git command gives information about how to use Git?**

git init

git status

git help

**Correct! Use git help to view information about how to use Git.**

Bottom of Form

Unit 6-6

**Summary**

Completed100 XP

* 3 minutes

Congratulations! In this module, you learned the basics of using Git.

You learned:

* An overview of Version Control Systems (VCS)
* Important Git terminology.
* The differences between Git and GitHub.
* How to configure Git.
* Some basic Git commands.

At this point, you know enough about Git to use version control by yourself on a basic project. Collaboration with other developers is where version control shines. Check out the other modules in this learning path for more about using Git with others!

**Resources**

If you'd like to dig deeper, here are more resources:

* Run the git help tutorial and git help tutorial-2 commands.
* Visit the [Everyday Git](https://git-scm.com/docs/everyday) site or use the git help everyday command.
* Review [Git and GitHub learning resources](https://help.github.com/en/articles/git-and-github-learning-resources).
* Watch the [Introduction to Git Recap](https://www.youtube.com/watch?v=9uGS1ak_FGg%3Fazure-portal%3Dtrue) video.
* Check out the [documentation section](https://git-scm.com/doc) of [Git's official website](https://git-scm.com/).