# Exercise - Try out Git

15 minutes

Before you can create your first repo, you must make sure that Git is installed and configured.

## Configure Git

In bash Shell, to double-check that Git is installed, type git --version:

# git –version

You should see output that looks something like this example:

git version 2.7.4

To configure Git, you must define some global variables: user.name and user.email. Both are required for you to make commits.

Set your name in Shell with the following command. Replace <USER\_NAME> with the user name you want to use.

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

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

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

Run the following command to check that your changes worked:

*git config --list*

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.

*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.

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.

*mkdir Cats*

Change to the project directory by using the cd command:

*cd Cats*

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:

*git init --initial-branch=main*

Or use the following command:

*git init -b main*

For earlier versions of Git, use these commands:

*git init*

*git checkout -b main*

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

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

Switched to a new branch 'main'

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

*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.

On branch main

No commits yet

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

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

*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.

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

*git --help*

The command displays the following output:

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.