UNIVERSITY OF ALBERTA

Git

Version Control System

What *is* version control?

While making changes to files in a project, version control allows for us to take “snapshots” of these files when we’d like to.

You can restore the project to a previous snapshot at a later time if you need to, or if something goes wrong.

About Git.

Git was invented by Linus Torvalds and his colleagues to help he and his team keep track of the Linux kernel development as it continued to grow and evolve.

It is now the most popular version control software available, which is why we're covering it. Note that alternatives like BitKeeper and Mercurial do exist.

What is Git?

Older version control systems were often “centralized,” wherein there is only a single copy of the project that everyone may commit changes to.

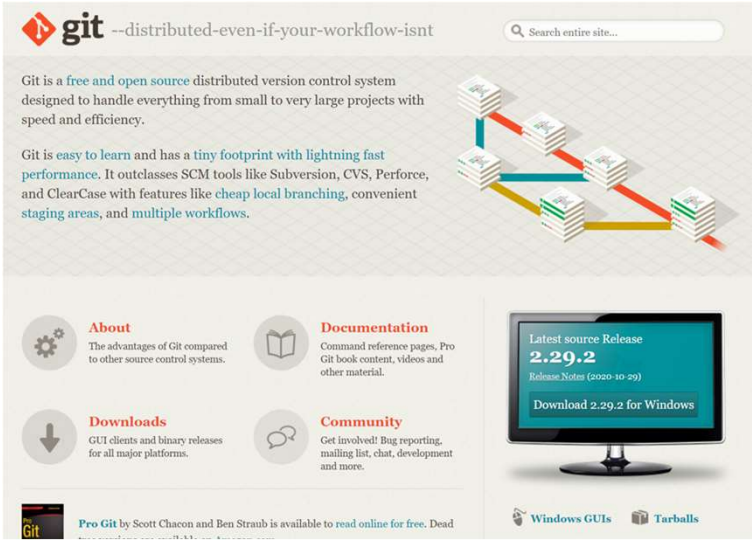
Git, in contrast, is a “distributed” version control system. With Git, team members create a copy (or “clone”) of the project (or “repository) and work on that in their own environment.

There is still often a master copy of the project that everyone can copy from, or commit to, which lends itself especially well to team environments.

Download and install Git.

Navigate to the official website:
https://git-scm.com/

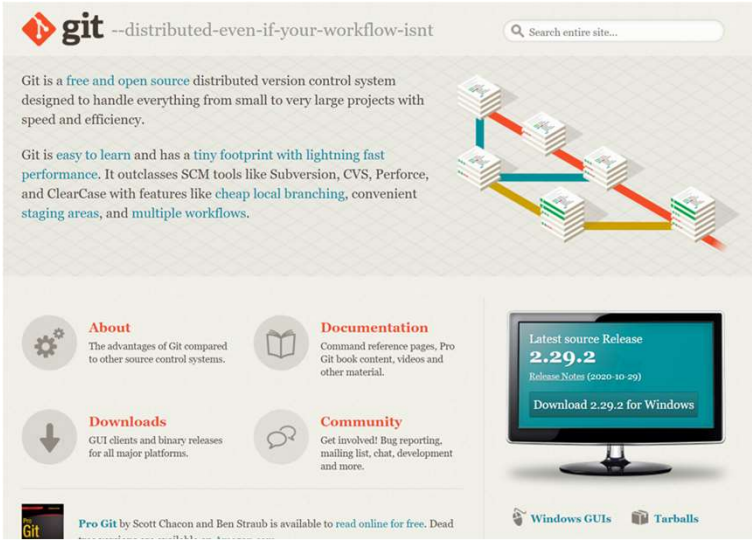
Download the installer and proceed through the steps in the wizard.



Download and install Git.

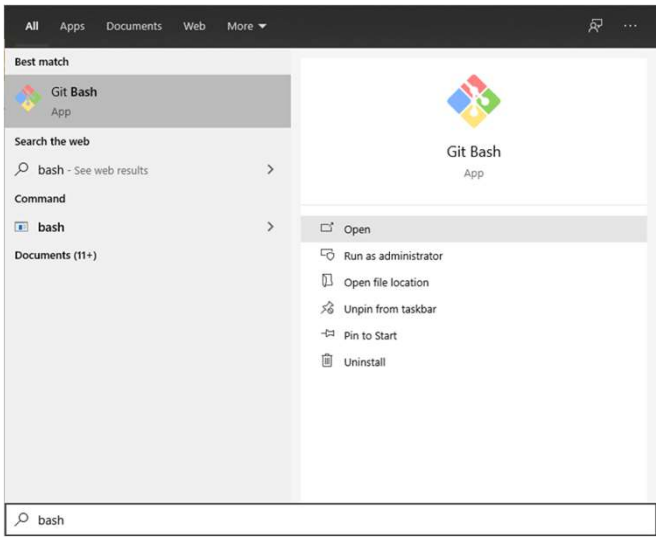
Navigate to the official website:
https://git-scm.com/

Download the installer and proceed through the steps in the wizard.



Using Git.

There are different programs you can use to interface with your installed Git software, but the way we'll use in this course is via the included "Git Bash" terminal. This is to say, we're going to make use of Git's command-line interface, or "CLI."



Getting started.

Try typing the following command…

git --version

If you have Git installed and configured correctly, you'll see your terminal display the version of Git you have installed.

```
$ git --version
git version 2.25.1.windows.1
```

Setting up Git.

There are a number of configurations you can make to help customize your experience. A couple to get you started include your name and e-mail—these will be stamped on the snapshots (or “commits”) you’ll make. This is incredibly helpful in team environments.

Enter your name, and then your e-mail, using the following:

```
git config --global user.name “My Name”
git config --global user.email your@email.com
```

Replace “My Name” and “your@email.com” with your info.

Initializing a repository.

When you're starting a new project, you will navigate into its folder via your terminal and run the following command:

git init

This tells Git that we would like it to keep track of changes to files in this folder, so that we can make snapshots (or "commits") as we work files here.

```
$ cd ~/
$ mkdir projects
$ cd projects
$ mkdir my-first-git-project
$ cd my-first-git-project
$ git init
Initialized empty Git repository
```

Behind the scenes.

The Git repository is initialized! So how and where *does* Git keep track of everything?

Try typing the following into your command-line:

ls -A

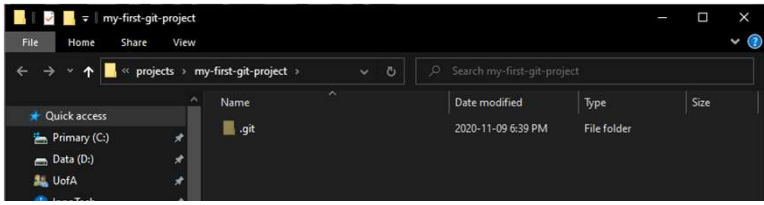
What do you see?

The hidden “.git” folder.

Folders featuring a name that starts with a period (“.”) are considered hidden. Often, they are hidden because they are not intended to be opened or manipulated by the average user.

Note that each folder you initialize a repository in will have a “.git” folder created in it. It is very important that you only run commands like “git init” in the folder(s) you intend, it is easy to accidentally run this in a folder you don’t want Git to keep track of!

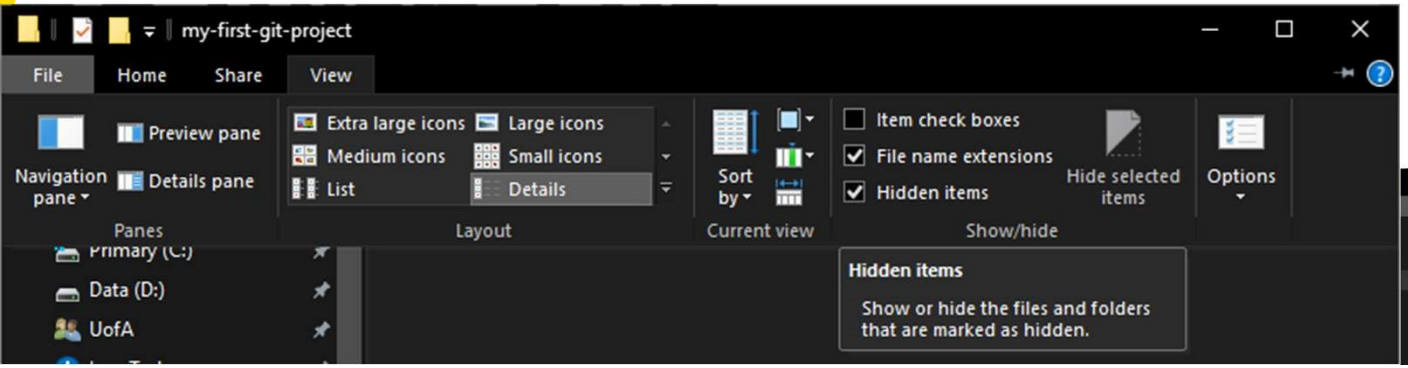
```
$ ls -A
.git/
```

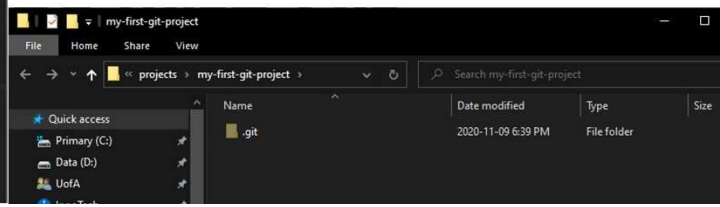


Not seeing hidden files and folders in your File Explorer?

If you don’t see hidden files and folders in your File Explorer, click the “View” tab at the top of the window.

This will open the “View” section of the ribbon interface. Ensure the “Hidden Items” checkbox is checked.





Checking the status of your repository.

It will matter which folder you're in, as well, when running the other commands and options that Git offers.

One of the most common and useful commands is…

git status

This will output some information regarding your repository and whether or not there have been changes since your last snapshot (or "commit.")

```
$ git status
On branch master

No commits yet

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

Your first change.

For us to be able to make a snapshot (or “commit”) of our project, we’ll need to add a new file, or change an existing one.

There are no files in the folder yet, so let’s add one!

touch my-text-file.txt

Now let’s see if the status changed...

git status

```
$ touch my-text-file.txt
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        my-text-file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Untracked changes.

You'll see, now that there is a new file, Git noticed!

It has yet to track these changes, though. If we want to snapshot the repository in its current state, we'll need to "stage" the changes. This means, telling Git which files we should include in our snapshot.

Git affords us the flexibility of making decisions in this process.

```
$ touch my-text-file.txt
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        my-text-file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Staging changes.

To tell Git which files to include in our snapshot, we use…

git add my-text-file.txt

Note that after the “add” command we can tell Git which file we are trying to add to the stage.

Use “git status” to check that it was added successfully!

```
$ git add my-text-file.txt
```

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   my-text-file.txt
```

Commiting staged changes.

To save your snapshot—or, in Git terms, commit your staged changes—we need to use the “git commit” command.

Whenever you save a commit, you are expected to include a little message with details about what changed. If you do need to restore to a previous commit, it is this message that will give you context as to what you’re restoring to.

git commit -m “Added my text file.”

```
$ git commit -m "Added my text file."
[master (root-commit) 01a21a6] Added my text file.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 my-text-file.txt
```

Viewing the log.

Alright, so the commit has been made!

Note we can keep adding changes to our stage, and committing them.

We can view a log of all of our previous commits for this repository with the following command…

git log

```
$ git log
commit 01a21a6f367195627573a2f25652c37fcdad8127 (HEAD -> master)
Author: Warren Uhrich <hello@warren.codes>
Date:   Mon Nov 9 19:46:19 2020 -0700

    Added my text file.
```

UNIVERSITY OF ALBERTA

Frequently check status to tell where things are at.

You can see when the commit was made, the message you wrote with it, and the author (again, this starts making a bigger difference when you're working with a team!)

Use "git status" frequently if you're ever unsure of the state that your repository is in.

```
$ git status
On branch master
nothing to commit, working tree clean
```

The standard steps during work on a project.

Time to get used to the workflow.

1. git add --all
2. git commit -m “Message about my change.”

In order to add a change to the stage, and follow that with a commit, we do need to firstly have a change in the repository’s files!

Making and checking on your commit.

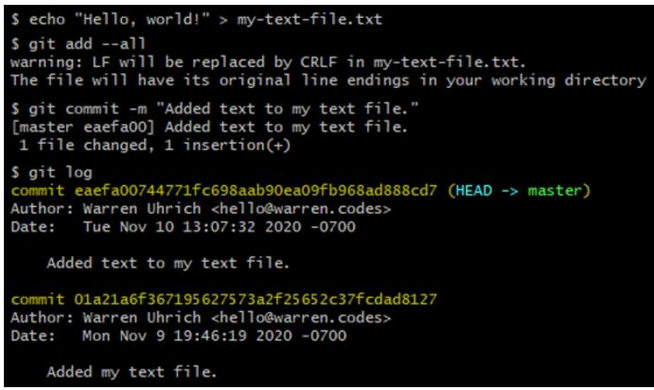
Add some text, and go through our steps...

echo “Hello, World!” > my-text-file.txt

git add --all

git commit -m “Added text to my text file.”

git log



Checking the differences in files since the last commit.

Add a bit more text, and then run “git diff”...

echo “ One more change.” >> my-text-file.txt

git diff

This command will show you which (if any) files were updated, and which lines were added, altered, or removed.

```
$ echo " One more change." >> my-text-file.txt
$ git diff
warning: LF will be replaced by CRLF in my-text-file.txt.
The file will have its original line endings in your working directory
diff --git a/my-text-file.txt b/my-text-file.txt
index af5626b..6ebb45b 100644
--- a/my-text-file.txt
+++ b/my-text-file.txt
@@ -1 +1,2 @@
 Hello, world!
+ One more change.
```

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   my-text-file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Let's commit the change we saw.

git add --all

git commit -m "Added a bit more text to the text file."

```
$ git add --all
warning: LF will be replaced by CRLF in my-text-file.txt.
The file will have its original line endings in your working directory
$ git commit -m "Added a bit more text to the text file."
[master 64bc4ac] Added a bit more text to the text file.
 1 file changed, 1 insertion(+)
```

Reverting to a previous commit.

Let's say we aren't happy with our update, and we need to roll back to our first text. When we "git log," we can see the commit IDs and use this to decide which commit we want to reset to.

git log

git reset --hard YOUR-COMMIT-ID

UNIVERSITY OF ALBERTA

Git reset.

Give it a try!

```
$ tail my-text-file.txt
Hello, world!
 One more change.

$ git log
commit 64bc4ac53edb1af9598fca659db2bb4869e3978f (HEAD -> master)
Author: Warren Uhrich <hello@warren.codes>
Date:   Tue Nov 10 13:30:42 2020 -0700

    Added a bit more text to the text file.

commit eaefa00744771fc698aab90ea09fb968ad888cd7
Author: Warren Uhrich <hello@warren.codes>
Date:   Tue Nov 10 13:07:32 2020 -0700

    Added text to my text file.

commit 01a21a6f367195627573a2f25652c37fcdad8127
Author: Warren Uhrich <hello@warren.codes>
Date:   Mon Nov 9 19:46:19 2020 -0700

    Added my text file.

$ git reset --hard eaefa00744771fc698aab90ea09fb968ad888cd7
HEAD is now at eaefa00 Added text to my text file.

$ tail my-text-file.txt
Hello, world!
```

Let’s get your files back up-to-speed.

Woah, it’s the way they were a commit ago! Let’s get them back to our latest commit again.

git log

git reset --hard YOUR-COMMIT-ID

There; back to normal!

```
$ tail my-text-file.txt
Hello, world!
$ git log
commit eaefa00744771fc698aab90ea09fb968ad888cd7 (HEAD -> master)
Author: Warren Uhrich <hello@warren.codes>
Date:   Tue Nov 10 13:07:32 2020 -0700

    Added text to my text file.

commit 01a21a6f367195627573a2f25652c37fcdad8127
Author: Warren Uhrich <hello@warren.codes>
Date:   Mon Nov 9 19:46:19 2020 -0700

    Added my text file.

$ git reset --hard 64bc4ac53edb1af9598fca659db2bb4869e3978f
HEAD is now at 64bc4ac Added a bit more text to the text file.

$ tail my-text-file.txt
Hello, world!
 One more change.
```

Git commands we covered.

git...

- status
  Current repo status.
- diff
  Changes in any repo files.
- add
  Stage and ready changes for commit.
- commit
  Save a “snapshot” of the repo based on the stage.
- log
  Display list of commits.
- reset
  Revert to a previous commit.

Cheat sheet(s).

When learning a tool like Git, cheat sheets can be a huge help. With repetition, you'll likely end up eventually memorizing the commands you use most often.

Recommended Reading

If you're interested in fleshing out your understanding of Git, try out the following:

- Loeliger, J., McCullough, M. (August 2012). *Version Control with Git, 2nd Edition*. O'Reilly Media, Inc.
- Tsitoara, M. (November 2019). *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer.* Apress.


UNIVERSITY OF
ALBERTA