

Git Version Control and Change Tracking

Common Commands and Syntax

Common Commands and Syntax.....	1
General Repository Administration.....	2
Adding/Modifying Files in Repository.....	2
Deletion of Files from Repository.....	4
Collaborative Git Administration.....	5
Creating/Entering Branches.....	5
The Use Case of Branches.....	5
Merging Branches.....	7
Remote Repositories using GitHub.....	9
Commit Tracking.....	10
Cloning.....	11
Track All Changes Across All Branches.....	13

General Repository Administration

Adding/Modifying Files in Repository

```
MINGW64/d/GitVersionControl

Carson@DESKTOP-RNCU7UO MINGW64 ~
$ cd /d/GitVersionControl

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl
$ git init
Initialized empty Git repository in D:/GitVersionControl/.git/

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ echo "# Welcome to the Git Version Control Lab repository! See branches for the Standard Installation Procedures of specific ESXi software versions" > README.md

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ touch ESXi-SIP

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git add .

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   ESXi-SIP
        new file:   README.md

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git commit -m "Initial Commit"
[master (root-commit) 61388e9] Initial Commit
 2 files changed, 1 insertion(+)
 create mode 100644 ESXi-SIP
 create mode 100644 README.md

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ |
```

1) \$ cd "d/GitVersionControl"

puts the Git Bash terminal into the directory that is desired for the repository to be created in

2) \$ git init

creates the local repository

3) \$ echo "# Welcome to the Git Version Control Lab repository!" > [README.md](#)
creates the [README.md](#) file and inserts the message in quotations into the file

4) `$ touch ESXi-SIP`

creates the ESXi-SIP file in the repository

5) `$ git add .`

puts the files in the directory into Git's temporary staging area to be committed

6) `$ git status`

outputs what files have been added to the staging area to be committed

7) `$ git commit -m "Initial Commit"`

takes the file and modification of the file created and moves it from the staging area to the local repository (the -m allows you to leave a commit description for change control clarity). If you are adding a file to the repository, you could make "added example.txt" the description. If you are committing a change to a file, you could make "modified example.txt" the description

Deletion of Files from Repository

```
MINGW64:/d/GitVersionControl

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git rm --cached README.md
rm 'README.md'

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:      README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git commit -m "Deleted README File"
[master eed1097] Deleted README File
1 file changed, 1 deletion(-)
delete mode 100644 README.md

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

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

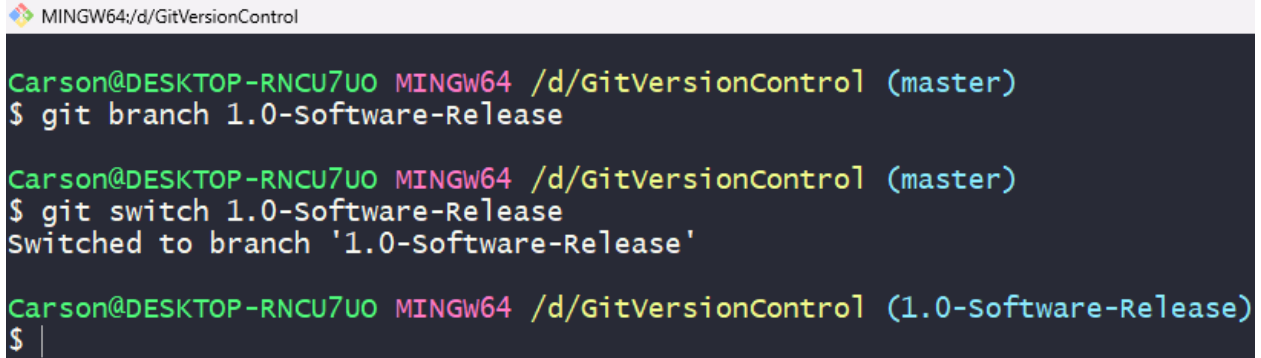
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$
```

1) `$ git rm --cached README.md`
deletes the file from the repository

2) `$ git commit -m "Deleted README.md"`
commits the deletion of the README file to the repository (This does NOT delete the file from your desktop however)

Collaborative Git Administration

Creating/Entering Branches

A screenshot of a Windows command prompt window titled 'MINGW64; d:/GitVersionControl'. The prompt shows a user named 'Carson' at a desktop named 'DESKTOP-RNCU7U0' in a 'MINGW64' environment. The user is in the directory '/d/GitVersionControl' and is currently on the 'master' branch. They enter the command '\$ git branch 1.0-Software-Release'. The prompt then shows the user entering '\$ git switch 1.0-Software-Release', and the output 'Switched to branch '1.0-Software-Release'' is displayed. Finally, the prompt shows the user is now on the '1.0-Software-Release' branch, with the prompt text updated to '(1.0-Software-Release)'.

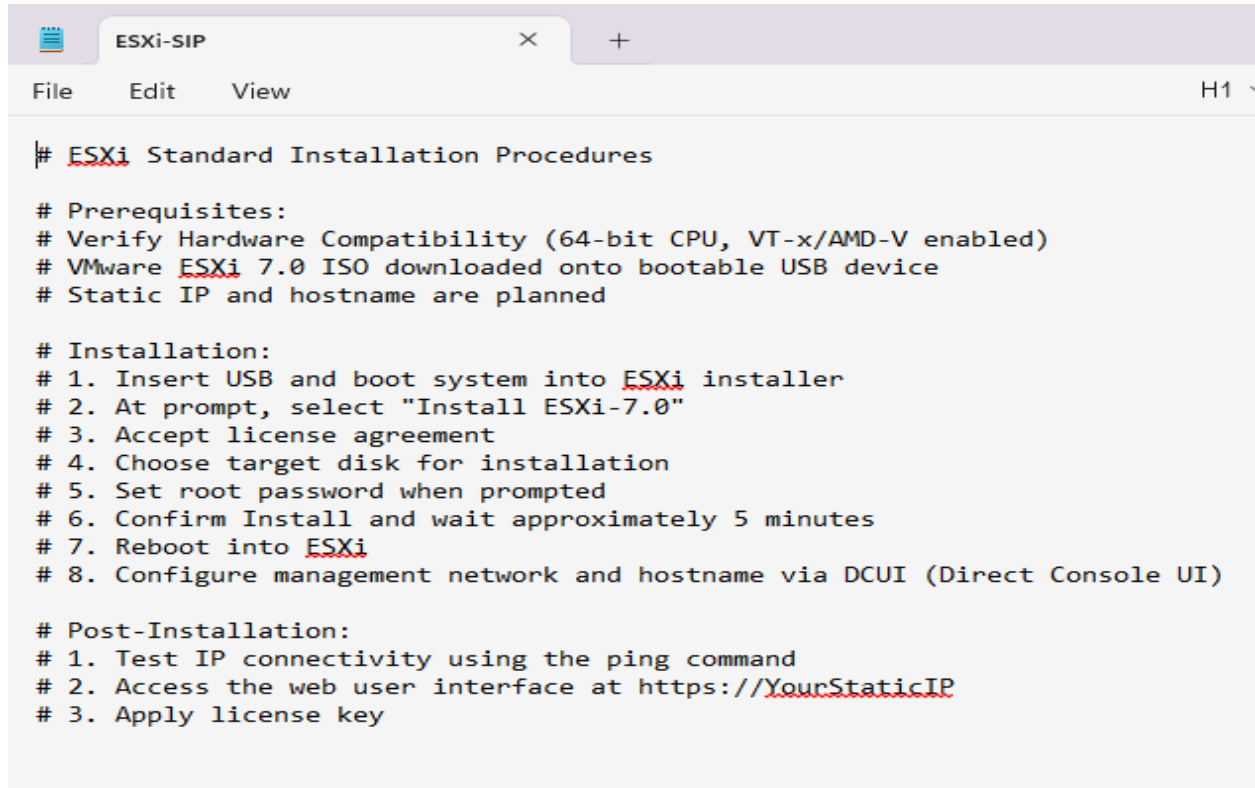
1) \$ git branch 1.0-Software-Release
creates a branch of the repository titled “1.0-Software-Release” (Notice the output of the next line shows we are still in the master branch)

2) \$ git switch 1.0-Software-Release
switches to the “1.0-Software-Release” branch

The Use Case of Branches

After switching from the master branch to the 1.0-Software_Release branch (important step), I then went into the [README.md](#) file on my local desktop and added content to the .txt file in notepad, to specifically reflect a version of ESXi installation (version 7.0).

File content while working in the 1.0-Software-Release branch:



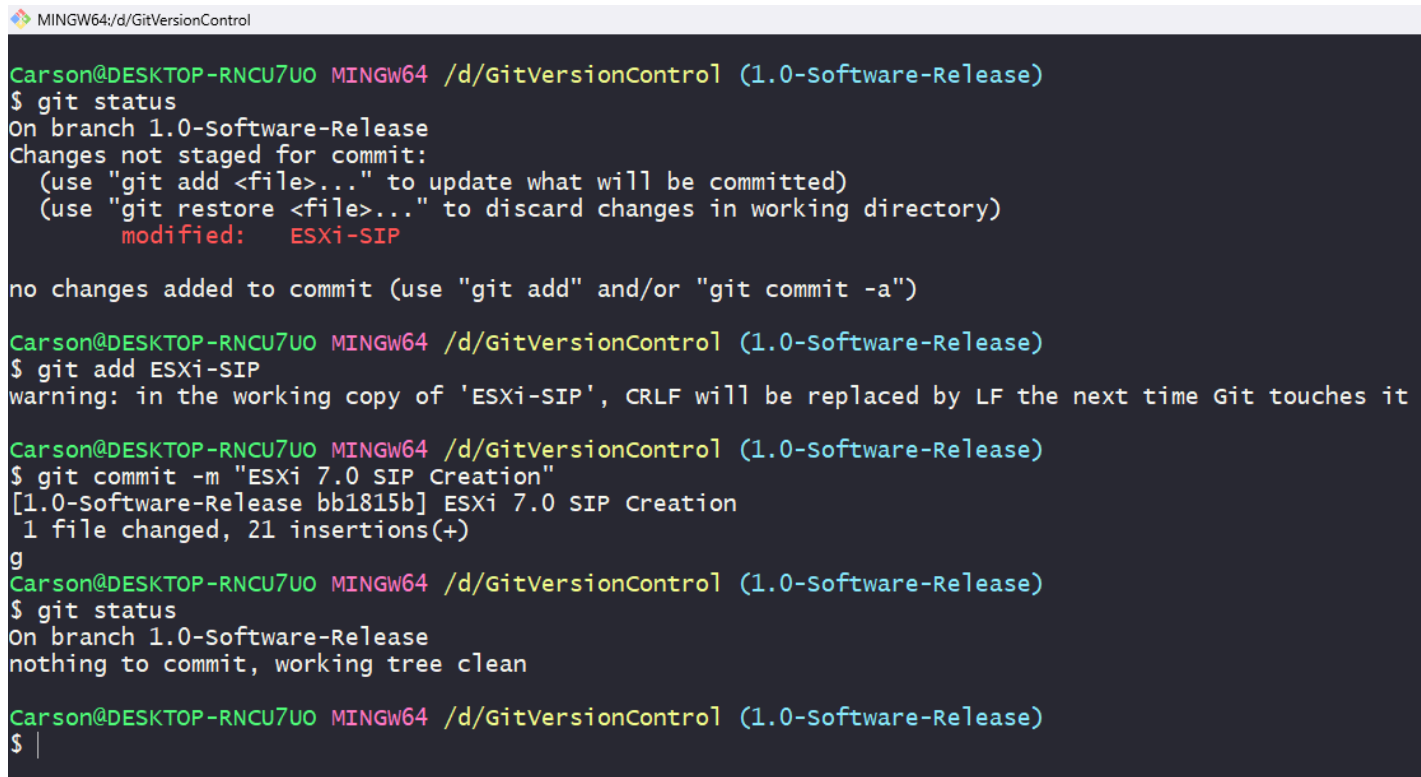
```
# ESXi Standard Installation Procedures

# Prerequisites:
# Verify Hardware Compatibility (64-bit CPU, VT-x/AMD-V enabled)
# VMware ESXi 7.0 ISO downloaded onto bootable USB device
# Static IP and hostname are planned

# Installation:
# 1. Insert USB and boot system into ESXi installer
# 2. At prompt, select "Install ESXi-7.0"
# 3. Accept license agreement
# 4. Choose target disk for installation
# 5. Set root password when prompted
# 6. Confirm Install and wait approximately 5 minutes
# 7. Reboot into ESXi
# 8. Configure management network and hostname via DCUI (Direct Console UI)

# Post-Installation:
# 1. Test IP connectivity using the ping command
# 2. Access the web user interface at https://YourStaticIP
# 3. Apply license key
```

The next steps to commit this modification:



```
MINGW64:/d/GitVersionControl

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (1.0-Software-Release)
$ git status
On branch 1.0-Software-Release
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:   ESXi-SIP

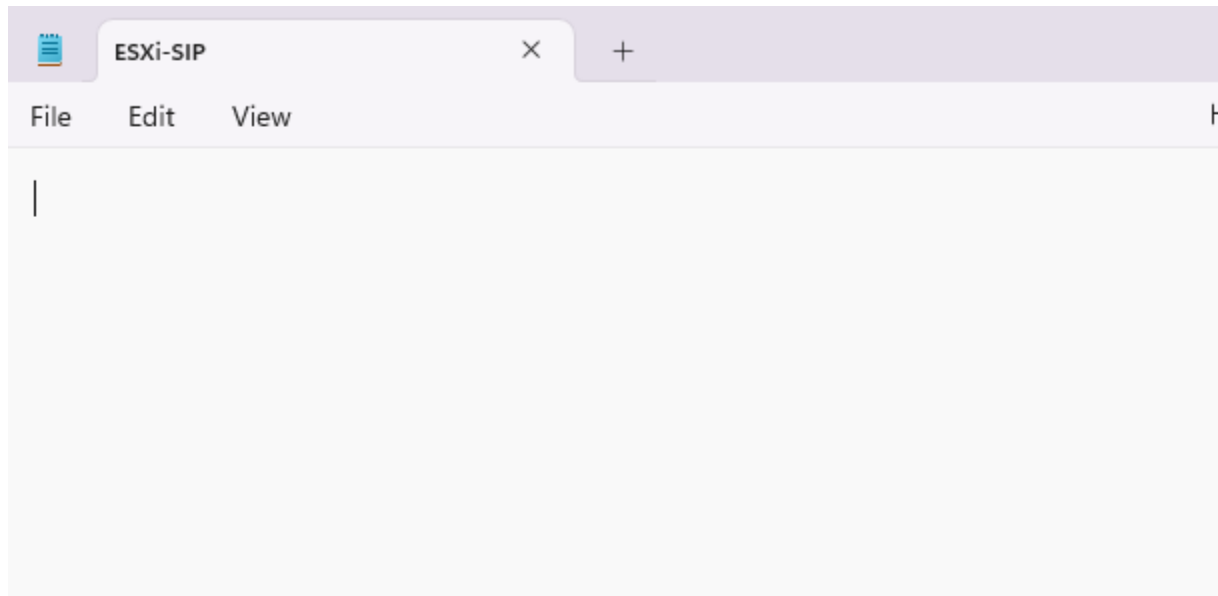
no changes added to commit (use "git add" and/or "git commit -a")

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (1.0-Software-Release)
$ git add ESXi-SIP
warning: in the working copy of 'ESXi-SIP', CRLF will be replaced by LF the next time Git touches it

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (1.0-Software-Release)
$ git commit -m "ESXi 7.0 SIP Creation"
[1.0-Software-Release bb1815b] ESXi 7.0 SIP Creation
1 file changed, 21 insertions(+)
g
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (1.0-Software-Release)
$ git status
On branch 1.0-Software-Release
nothing to commit, working tree clean

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (1.0-Software-Release)
$ |
```

After this I then switched back to the master branch, using “git switch master”, to see the ESXi-SIP file’s content when being in that branch on my terminal. After switching branches, The output of the file immediately changed to reflect the content of the file that is within the master branch.



The file has no content because I did not put any content into the ESXi-SIP file that lives in the master branch.

You could make endless amounts of branches in your repository that each have their own version of a file in order to account for differing software versions, and those versions differences/needs. This is the idea of version-based control. This is a major aspect of Git’s value in the context of software development.

Merging Branches

Say a company decides to cut out all other ESXi version implementations, besides ESXi software version 7.0. You could then make the master branch include everything that the 1.0-Software-Release includes, using the merge command.

Note: When doing “git merge”, if there is any content on the branch that is different from the branch you are merging it to, the user will be prompted to resolve merge conflicts and is asked which version of the file(s) they wish to keep.

The steps to do this are found in the screenshot below:

```
MINGW64:/d/GitVersionControl

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (1.0-Software-Release)
$ git switch master
Switched to branch 'master'

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git merge 1.0-Software-Release
Updating 27a33d3..29a437e
Fast-forward
 README.md | 20 +++++
 1 file changed, 20 insertions(+)

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$
```

Now see the file content that is outputted while being in the repository's master branch below:

```
ESXi-SIP
File Edit View H1

# ESXi Standard Installation Procedures

# Prerequisites:
# Verify Hardware Compatibility (64-bit CPU, VT-x/AMD-V enabled)
# VMware ESXi 7.0 ISO downloaded onto bootable USB device
# Static IP and hostname are planned

# Installation:
# 1. Insert USB and boot system into ESXi installer
# 2. At prompt, select "Install ESXi-7.0"
# 3. Accept license agreement
# 4. Choose target disk for installation
# 5. Set root password when prompted
# 6. Confirm Install and wait approximately 5 minutes
# 7. Reboot into ESXi
# 8. Configure management network and hostname via DCUI (Direct Console UI)

# Post-Installation:
# 1. Test IP connectivity using the ping command
# 2. Access the web user interface at https://YourStaticIP
# 3. Apply license key
```


Remote Repositories using GitHub

- 1) Go to github.com/new in your browser to create a new remote repository.
- 2) Give the repository a name, a description, and choose if you want to a README file to be created along with the repository. Click “Create a new repository”.

Upon creation, a “Quick setup” screen within the ‘Code’ tab appears showing you various commands on how to either create a new repository on the command line, or push an existing repository from the command line.

- 3) Use the “\$ git remote add <repo name> <repo url>” command in Git Bash to interact with the remote repository of choice using your local machine

The URL of your repository is listed on the “Quick setup” screen

- 4) Next use the “\$ git push -u <repo name> <branch you want to push>” for each branch that you would like to push to the remote repository

Note: Ensure all of your code works and will be compatible with the remote repository’s contents before pushing it because a failure to do so could cause overwrites and conflicts.

MINGW64:/d/GitVersionControl

```
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git remote add GitTest https://github.com/CarsonBex/GitTest.git
```

```
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git push -u GitTest master
info: please complete authentication in your browser...
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 1.05 KiB | 1.05 MiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/CarsonBex/GitTest.git
 * [new branch]      master -> master
branch 'master' set up to track 'GitTest/master'.
```

Commit Tracking

```
MINGW64:/d/GitVersionControl

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git config --global user.name Carson-Bex

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ git config --global user.email Carson.Bex@gmail.com

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$
```

- 1) `$ git config --global user.name <your username>`

After entering this command, the remote repository will track/label all of your commits to it within the metadata and the commit history, by your username. This provides non-repudation and accountability for collaborators.

- 2) `$ git config --global user.email <your email>`

After entering this command, the remote repository will track/label all of your commits to it within the metadata and the commit history, by your email address. This provides non-repudation and accountability for collaborators.

Cloning

```
MINGW64:/d/GitVersionControl/ClonedRepo

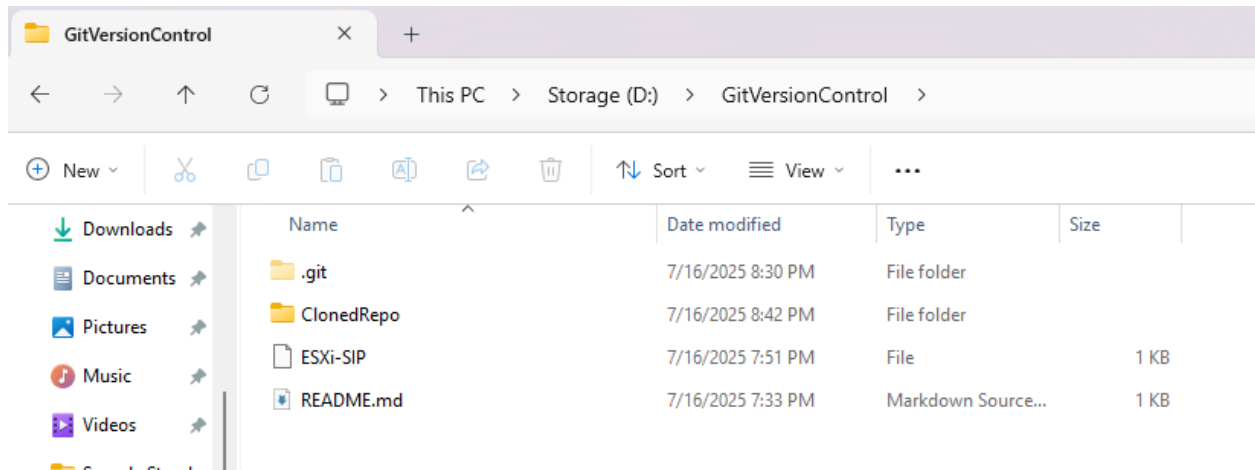
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ mkdir ClonedRepo

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ cd ClonedRepo

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl/ClonedRepo (master)
$ git clone https://github.com/CarsonBex/GitTest.git
Cloning into 'GitTest'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), done.

Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl/ClonedRepo (master)
$
```

- 1) The “\$ mkdir” command is used to create a folder with a name of your choice, as seen below



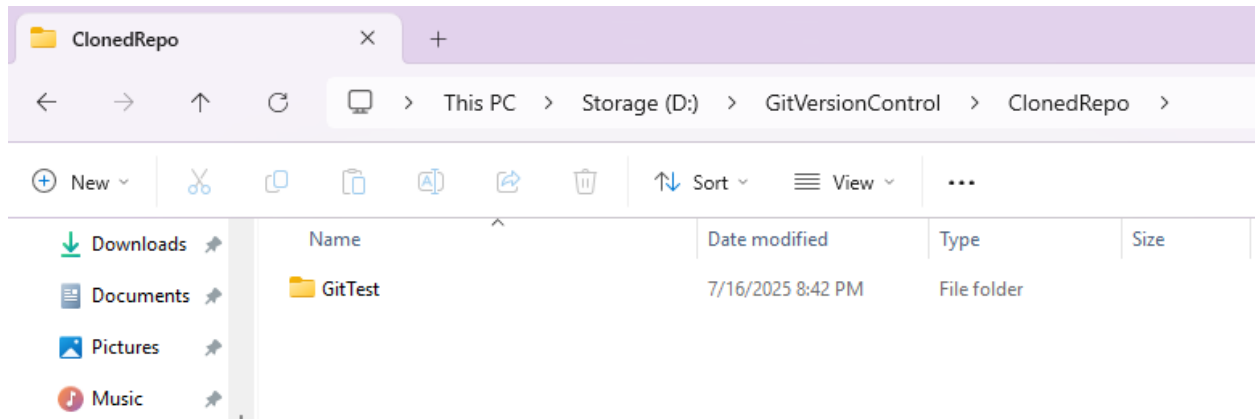
- 2) After making the folder to clone the remote repository into, use “\$ cd </folder>” to move to the newly created folder

```
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl (master)
$ cd ClonedRepo
```

- 3) Use “\$ git clone <Remote Repo URL>” to place it into your working folder

```
Carson@DESKTOP-RNCU7UO MINGW64 /d/GitVersionControl/ClonedRepo (master)
$ git clone https://github.com/CarsonBex/GitTest.git
Cloning into 'GitTest'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), done.
```

As you can see the Remote repository is now copied into the folder I chose:



- 4) After cloning, you will then want to use “\$ git pull <Remote Repo Name> <branch>” to fetch any changes from the remote repository. This should be done repeatedly to keep your local repository up to date.

```
Carson@DESKTOP-RNCU7U0 MINGW64 /d/GitVersionControl/ClonedRepo (master)
$ git pull GitTest master
From https://github.com/CarsonBex/GitTest
* branch          master      -> FETCH_HEAD
Already up to date.
```

Track All Changes Across All Branches

For a full audit trail across every branch (local and remote), listing usernames and emails, I created this batch file:

```
git-modification-tracking.bat X
git-modification-tracking.bat
1  @echo off
2
3  REM Get the path to the user's Downloads folder
4  set "downloads=%USERPROFILE%\Downloads"
5  set "logfile=%downloads%\git_version_output.txt"
6
7  REM STEP 1: Fetch remote updates
8  echo STEP 1: Fetching remote updates .. > "%logfile%"
9  git fetch --all >> "%logfile%"
10 echo. >> "%logfile%"
11 echo. >> "%logfile%"
12
13 REM STEP 2: Show full commit history across all branches
14 echo STEP 2: Showing full commit history across all branches, listing usernames and emails.. >> "%logfile%"
15 echo. >> "%logfile%"
16 git log --all --pretty=format:"%h %an <%ae> %s" >> "%logfile%"
17
18 REM Automatically open the file in Notepad
19 notepad "%logfile%"
```

When running this batch file in Windows, a command prompt screen appears on the user's screen indicating the batch file is running, then the output of the file is included on a notepad file that pops up soon after. The file is automatically saved to the executor's \Downloads folder.

The script consists of a few Windows Command Prompt commands, as well as 2 Git commands. The two Git commands are:

git fetch --all

This downloads all latest data from the remote repository, without altering or merging your current working directory, ensuring your local repository has all recent changes pushed by contributors.

git log --all --pretty=format:"%h %an <%ae> %s"

This gives you a visual timeline of every commit, who made it, and what branch it was executed in.