###### *STUDY GUIDE*

##### *Magic xpa and Git*





**Last Revision: Tuesday, March 28, 2023**

**Developed by: Roberto Ramírez Cervantes**

marionette\_66@fastmail.com

Table of Contents

[About this Study Guide 3](#_Toc131110204)

[Typographical Conventions 4](#_Toc131110205)

[Manual Notes 4](#_Toc131110206)

[Where can I Find the Main Topics of this Manual? 5](#_Toc131110207)

[Feedback Form 5](#_Toc131110208)

[Part 1 – Introduction to Git 6](#_Toc131110209)

[Lesson 1 – The Git Concept 6](#_Toc131110210)

[Lesson 2 – Installation, Setup and Git Service 9](#_Toc131110211)

[Lesson 3 – Setup Environment and the Git Workflow 22](#_Toc131110212)

[FEEDBACK FORM 40](#_Toc131110213)

# About this Study Guide

The goal of this Study Guide is to provide a Magic developer (that is you!) with the basic knowledge and guidelines to start working with Git as the version control system for Magic Projects either as a solo programmer or in a corporate setting.

We will assume that you have no idea that is Git or how to use it, and as such, we will cover “Level 0” topics like:

* The installation and configuration of required software,
* The creation of an account on the Git service that will host our code,
* The Git Workflow and the commands to move files within it,
* The procedure to connect a Folder in your computer with a Git service,
* The procedure to connect a Project in your computer with a Git service,
* The basic day-to-day operations to update your local with code, validate it, modify it, and update the remote repositories to make it available to other programmers,
* Best practices to manage code changes,
* As well as advance features like Branching, Forking, Cloning, and resolving code conflicts between programmers.

This is something that MSE has proven to be uncappable and\or unwilling to do.

If you want to know more about Git itself, there are plenty of websites and books that deal with this topic solely.

## Typographical Conventions

The following table will show the different type fonts used on this manual and their significance:

| Type Style | Means |
| --- | --- |
| *Italic* | Menu names or the name of options. For example: *File | New Project* means select the *New Project* entry from the *File* menu  Italic type also signals a new term, and it is also used for references in the text to names of publications, such as the *Mastering Magic xpa* document |
| Courier | Anything that you must type or select exactly as it appears |
| **Bold** | Placeholders for information you are expected to provide to Magic. For example: “... enter in the **Last Name** field the name for the person you are looking for” |
| ALL CAPITALS | Full paths and file names from your computer |
| Initial Capitals | The name of tables, programs, models, screens |
| <Enter> | The name of a key on your keyboard |
| <Shift+F3> | The combination of two keys on your keyboard that you must press by holding down the first key while you press the second key |

## Manual Notes

Throughout this manual you will find sections that will contain important notes, step-by-step instructions, or exercises, just to mention some. The following table describes each one of these sections:

| Icon | Description |
| --- | --- |
|  | Perform the following operations as described here |
|  | A cautionary note regarding the current section |
|  | A helpful hint regarding the current topic |
|  | A piece of information that should not be forgotten |
|  | A task to perform for the current section |
|  | An Exercise to perform at the end of each Lesson. **The actual Exercise can be found towards the end of this document** |
|  | A possible error that might occur at this moment and its solution |

All Icons made by Freepik from [www.flaticon.com](http://www.flaticon.com)

## Where can I Find the Main Topics of this Manual?

If you are brand new to Git, Lesson 1 is a must for you. It will explain what Git does and the functionality that offers to programmers.

Lesson 2 covers the installation, configuration of software required to use Git with Magic and the creation of an account on a specific Git service host.

Lesson 3 will explain in more detail the four elements of Git and the commands that will allow us to move files within the Git Workflow.

## Feedback Form

Each time study guide is used, or this manual is read, we want to make it better.

Please take one or two moments and fill up the Feedback form at the end of this document and let us learn from you on ways to improve it for the next person; tell us what to change, what to add, what do delete, and what to leave alone.

Thank you,

Roberto Ramírez Cervantes

marionette\_66@fastmail.com

# Part 1 – Introduction to Git

## Lesson 1 – The Git Concept

#### PURPOSE OF THIS LESSON

To introduce the concept behind Git

#### KEY TERMS

* Source Code
* Version Control
* Git Services

#### OBJECTIVES

* To understand the basic functionality of Git
* Understand that Git is just a set of rules\protocols that drive the management of code\file changes
* Be aware of Git several services and their offerings

#### CONTENT OUTLINE

* + 1. Introduction – What is Git?
    2. The Git Concept

### 1.1.1 Introduction – What is Git?

You asked that fine question right there. In a nutshell, Git is a distributed version control system that stores code\files and keeps track of every change performed to that code\file.

If you have worked with a version control system like Visual SourceSafe or Team Foundation Server (TFS), Git provides the same basic code\file storage, update and restore features, but it offers far more options and controls surpassing these previous examples.

Git was born out from the Linux community after they no longer wanted to use a proprietary source control program called BitKeeper. The outcome of this effort has become a widely use, open source code, platform independent set of rules\protocols that keeps a watchful eye over the code\files of one or multiple programmers and controls how this information is stored, managed, updated, and restored as well as to how to resolve conflicts when two or more people update the same piece of code\file.

As we just mentioned, Git is just a set of rules\protocols that helps you keep track of code\files and the changes made to them; now take in consideration the following points:

* There is a set of rules\protocols controlling how an email message is written, stored, distributed, and displayed,
* We have a multitude of email services (Gmail, Yahoo Mail, Outlook, etc.) available to us,
* Each email service offers different features and add-ons to help you manage your emails,
* All these services must comply with the basic rules\protocols on how to work with an email message.

For Git, it follows the same path as described above:

* There is a set of rules\protocols controlling how the change to code\file is written, stored, distributed, and displayed,
* We have a multitude of Git services (Bitbucket, GitHub, GitLab, Codebase, etc.) available to us,
* Each Git service offers different features and add-ons to help you manage your code\files,
* All these services must comply with the basic rules\protocols on how to work with code\files changes.

Based on this, do not be confused when you hear the names of these services tossed around like salad, all of them will comply with the basic set of rules\protocols guiding the core functionality of Git and it should not matter if you need to access code\files from multiple services.

### 1.1.2 The Git Concept

While the idea of Git might be simple (tell me how to store code\files and keep track of their changes), there are several rules and procedures that every programmer must be aware of and follow to successfully use Git “the correct way”. Just like in Magic, every programmer must be aware of and follow the rules and procedures governing the Magic Engine to create a program “the correct way”; same difference.

While working with Git, you will find that everyone of your files will be in one of these basic locations: a **Working Area**, a **Staging Area**, a **Local Repository**, and a **Remote Repository**.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

Figure 1. The four basic locations for Git.

While the **Working**, **Staging** and **Local Repository** locations are in your local computer, the **Remote Repository** will rely on an off-site Git service (Bitbucket, GitHub, GitLab, Codebase, etc.) to store your code\files.

The **Working Area** is basically your Project’s folder (i.e. {Magic Folder}\Projects\GitTest01). This folder should contain not only the source code for you Project but any supporting files (Color File, Font File, etc.) as well as any 3rd party software that your Project requires to run.

The **Staging Area** is the middle section between the code that is constantly being updated while you work, and the code that has been tested\verified and ready to be made public (sent to the Remote Repository).

The **Local Repository** will not only contain code that has been tested\verified, but the actual process of “grouping” the code from the Staging Area to the Local Repository is very important, and you will hear the word “commit” plenty of times at this stage.

In your day-to-day work, you will need to pay attention on how to group and organize the changes of your code\files between these different locations, this is crucial to help you consolidate, test, review, approve\undo changes and of course, resolve conflicts. This “best practices” will be explained at a later Lesson.

The **Remote Repository** is the end of the line for your code\files. The information in this location can be seen and used by any another programmer that has access to this Repository; this is where other programmers will get the code\files to update their Working Area and do their own jobs.

There are different Git commands that will “move” your files between these basic locations, and either you can execute them manually using the Git commands or using the Magic xpa interface. And while both concepts will be explained in this document, first we will learn how to use these commands manually and then we will use the Magic interface to duplicate them.

## Lesson 2 – Installation, Setup and Git Service

#### PURPOSE OF THIS LESSON

Learn how to install, configure required software, and register into a Git service.

#### KEY TERMS

* Git for Windows
* TortoiseGit
* GitHub Service

#### OBJECTIVES

* Installing and Configure Software
* Creating an Account for a Git Service

#### CONTENT OUTLINE

* + 1. Introduction
    2. System Requirements
    3. Create a Git Service Account
    4. Install and Configure Git Software
* Git for Windows
* TortoiseGit

### 1.2.1 Introduction

This lesson will show you how to install and configure of the two different pieces of software required by Magic xpa to interact with Git, and how to register with a Git service to host your Remote Repository.

### 1.2.2 System Requirements

MSE introduced support for Git in Magic xpa version 3.3, this is your minimum xpa version to use Git.

This Study Guide will use version 4.9 for all its examples and exercises.

All the screen captures in this Study Guide will be taken using a Window 10 computer as the working environment.

Download the following software required by Magic to use Git:

* Git for Windows (<https://gitforwindows.org/>)
* TortoiseGit (<https://tortoisegit.org/>)

As of the writing of this document, the latest version of Git for Windows was 2.40.0, and 2.14.0 for TortoiseGit.

You will also need the to install the “Compare and Merge” software from MSE, available on the download area (<https://downloads.magicsoftware.com/>).

Due to the nature of the source code generated by Magic, the only “easy” way to see, compare and resolve code conflicts between Magic programmers is by using this program.

Lesson <(todo> insert lesson number here) will guide you on how to use this software to review code changes and resolve conflicts. You should ask your sales person for a temporary license while learning to use it, but in the real world, each programmer must have a license for it.

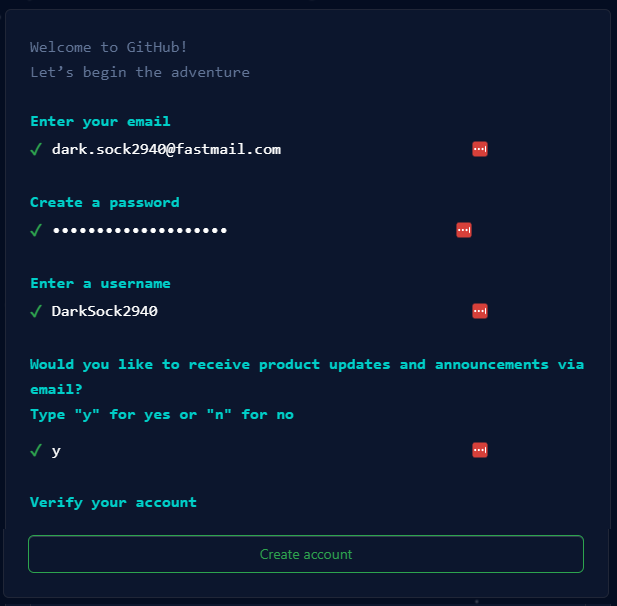
### 1.2.3 Create a Git Service Account

As we have mentioned before, the Remote Repository within the Git Workflow relies on an offsite, cloud-base, storage service that can understand the Git rules and protocols.

For the purposes of this document, we will use GitHub as our Remote Repository but you are free to use any other available Git service. We will also use the email address dark.sock2940@fastmail.com throughout this document but you should use your own email.

Follow these steps to create a new GitHub account:

1. Open your browser and go to: github.com
2. Select the option to Sign Up, enter the email address that you want to use, a password, and a username (the username must be unique within GitHub, so it is possible that your first choice will not be available).



1. As a verification step, GitHub might have sent a one-time code you the email address above and then ask you to enter it into a form.
2. GitHub might ask information about you and the way you want to use their site; you can skip this step.
3. Once this is done you will be presented with your GitHub page. In the upper-right corner you will find a menu to access your Profile, Settings, Repositories, etc. you can update your Profile at your discretion.

GitHub offers plenty of options, internal processes and connectivity to other systems to help you streamline your coding cycle, we are not going to dive into those features here.

### 1.2.4 Install and Configure Git Software

### Magic by itself does not know to talk to Git, instead, it relies on to 3rd party software to do all the things that Git does: keep track of code\files, their changes, move code\files to the different areas\repositories, push code to the Remote Repository, pull down code from the Remote Repository, how to deal with conflicts, etc.

Git for Windows

### Follow these steps to install and configure Git for Windows:

1. Execute the installation program for Git for Windows:

Graphical user interface, text, application

Description automatically generated

1. Select the Location where Git for Windows will be installed, we will use the default location:

Graphical user interface, text, application, email

Description automatically generated

1. Select the Components to install, we will use the default settings:

Graphical user interface, text

Description automatically generated

1. Select the name for the Start Menu folder, we will use the default name:

Graphical user interface, text, application, email

Description automatically generated

1. Select the default editor for Git, we will use the default editor (this setting can be changed later):

Graphical user interface, text, application, email

Description automatically generated

1. Select the name for the default branch, we will use main as the default branch name for new repositories (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Adjust your PATH Environment, we will use the default option:

Graphical user interface, text, application, email

Description automatically generated

1. Choose the SSH executable, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Choose the HTTPS transport backend, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Configure the line ending conversion, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Configure the terminal emulator, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Configure the behavior of “git pull”, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Choose a credential helper, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Configure extra options, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. Configure experimental options, we will use the default setting (in a corporate environment, please consult with the Git administrator):

Graphical user interface, text, application, email

Description automatically generated

1. The installation process will run for a couple of minutes:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. When completed, you can just press the *Finish* button:

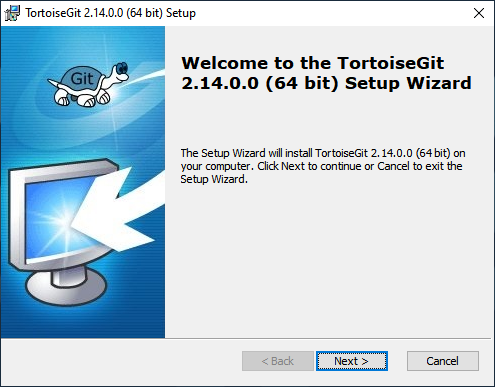
Graphical user interface, text, application

Description automatically generated

TortoiseGit

### Follow these steps to install and configure TortoiseGit for Windows:

1. Execute the installation program for TortoiseGit:



1. Read the information presented on this screen and press the *Next* button:

Graphical user interface, text, application, email

Description automatically generated

1. For the Custom Setup, we will use the default settings:

Graphical user interface, text, application

Description automatically generated

1. Press the *Install* button:

Graphical user interface, text, application, email

Description automatically generated

1. Once the installation is completed, select the option to *Run first start wizard* and press the *Finish* button:

Graphical user interface, text, application

Description automatically generated

1. Select the Language that you would like to use:

Graphical user interface, text, application, email

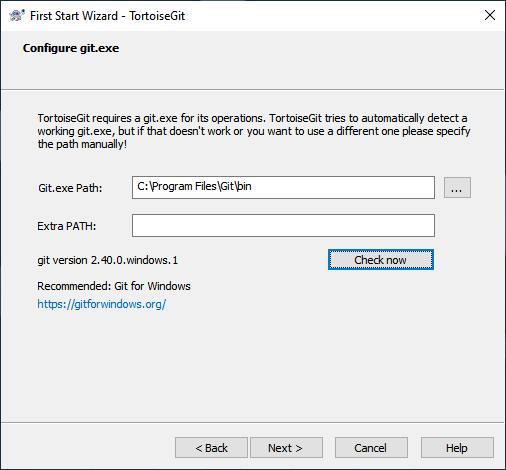
Description automatically generated

1. Read the information presented on this screen and press the *Next* button:

Graphical user interface, text, application

Description automatically generated

1. To Configure git.exe, we will use the default settings, press the *Check now* button to ensure that TortoiseGit can talk to Git for Windows:



1. Configure the user information by entering your name and email address, this email address must match the one used for the creation of your Git service account:

Graphical user interface, text, application, email

Description automatically generated

1. For the Authentication and credential store, we will use the default settings:

Graphical user interface, text, application, email

Description automatically generated

Once you press the *Finish* button, the screen will disappear and the setup will be completed.

That’s it, we just installed and configure the required software that Magic will use to keep track of our Project’s code.

## Lesson 3 – Setup Environment and the Git Workflow

#### PURPOSE OF THIS LESSON

Learn how to setup Git to keep track of file in a Folder, initialize the Git environment and move a file throughout the Git Workflow.

#### KEY TERMS

* Init Git Environment
* Git Workflow
* Working Area
* Staging Area
* Local Repository
* Remote Repository

#### OBJECTIVES

* Configure Git on a Folder
* Move a file thru the Git Workflow
* Utilize basic Git commands

#### CONTENT OUTLINE

1.3.1 Introduction

1.3.2 Setup Git

* Setup a Repository
* Create a Folder
* Initialize the Git Environment

1.3.3 Understand Git Workflow

* Working Area
* Staging Area
* Local Repository
* Remote Repository

### 1.3.1 Introduction

This lesson will show you how to configure Git on a brand new Folder and keep track of a file as it moves throughout the different states of the Git Workflow.

We will not use Magic at all in this Lesson, the main objective is to show you the basic Git commands to:

* Stablish a connection between a Folder in your computer and a Repository in GitHub,
* Keep track of a file inside this folder and “move” it to the different areas and repositories within the Git workflow,
* Find the status of this file at every location,
* Detect changes made on this file, and
* Restore an older version of the file.

### 1.3.1 Setup Git

This is a two step process, we need to create a Remote Repository first, then setup the Git environment on a specific Folder in our local computer.

Setup a Remote Repository

The first step in this process is the creation of a Remote Repository in Git Service of your choosing. Go to your browser, log into your GitHub account, select the option *Repositories* to see all your Repositories in your account and create a new one.

Enter a name for this Repository (it must be unique within your account), type a Description, set it as Public and press the *Create repository* button:

Graphical user interface, text, application, email

Description automatically generated

Once the Repository is created you will be presented with a Quick setup screen:

Graphical user interface, text, application, email

Description automatically generated

This screen conveniently contains all the commands that we will need to stablish a connection between a Folder in our computer and the Remote Repository we just created, as well as several commands to move a file between the different Areas and Repositories.

We are going to execute all these commands but not one after another, we are going to take our time to understanding each Area\Repository and the commands that affect them.

Create a Folder

Using File Explorer or CMD create a brand new Folder anyplace in your computer. Later, you can later delete this Repository and the contents of the Folder if you do not need this example anymore.

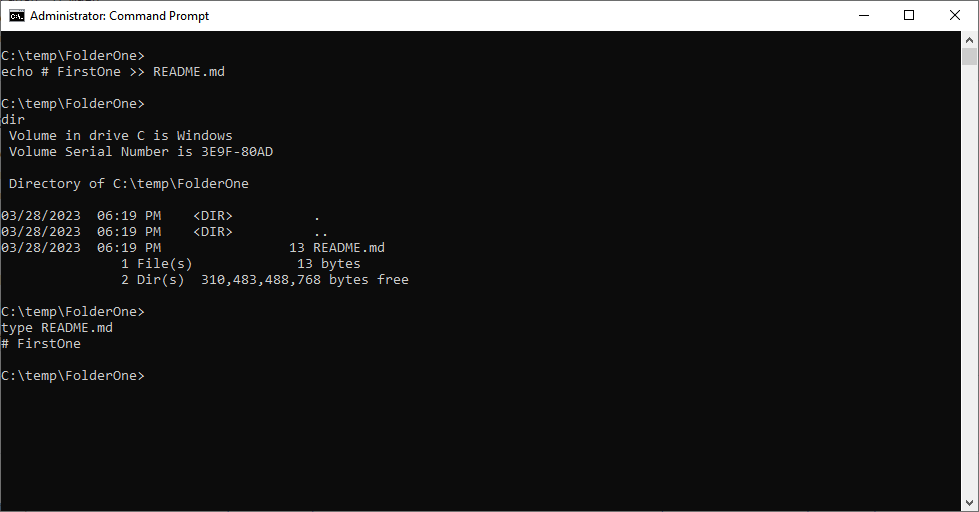
|  |  |
| --- | --- |
| Icon  Description automatically generated | Make sure you create this Folder as the Administrator of your computer; otherwise, Git might return an error while trying to determine the rightful owner of this Folder. |

Open a CMD window as the Administrator of your computer and go to the place where you created this folder, for this documentation, we will use C:\temp\FolderOne.

A picture containing shape

Description automatically generated

From the GitHub page, within the *create a new repository and the command line* section, copy the first line of this section, and execute it inside the CMD window, ***without*** the double quotes:



A quick use of the dir and typecommands will confirm that we just created a file called README.md containing a single line of text.

The README.md file is the first file that someone else will look at when they arrive at your GitHub Repository, it is good standards for it to contain a description of the program\project it belongs to and any basic documentation or information about it.

Initialize the Git Environment

Next, we need to create all the supporting Folders and Files that will allow Git to keep track of our file, to do that we will use the git init command:

Text

Description automatically generated

You might not be able to see it in the CMD window, but we just created a new folder called .git containing lost other Folders and Files:

Graphical user interface, application

Description automatically generated

That is all, we have configured Git in this Folder.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Executing the git init command more than one time on the same Folder will not overwrite any existing configuration. |

|  |  |
| --- | --- |
| Icon  Description automatically generated | At any point, you can delete the contents of the .git folder, and your git environment will start from scratch. |

### 1.2.6 Understand Git Workflow

If we bring back the four basic locations of Git and sprinkle a couple of commands between them we will get a Git Workflow.

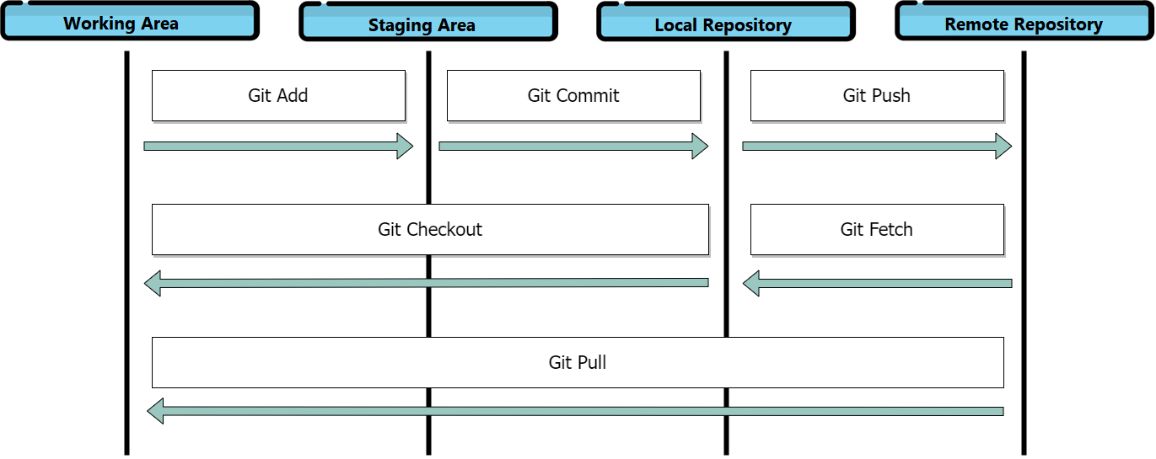


Figure 2. The Git Workflow

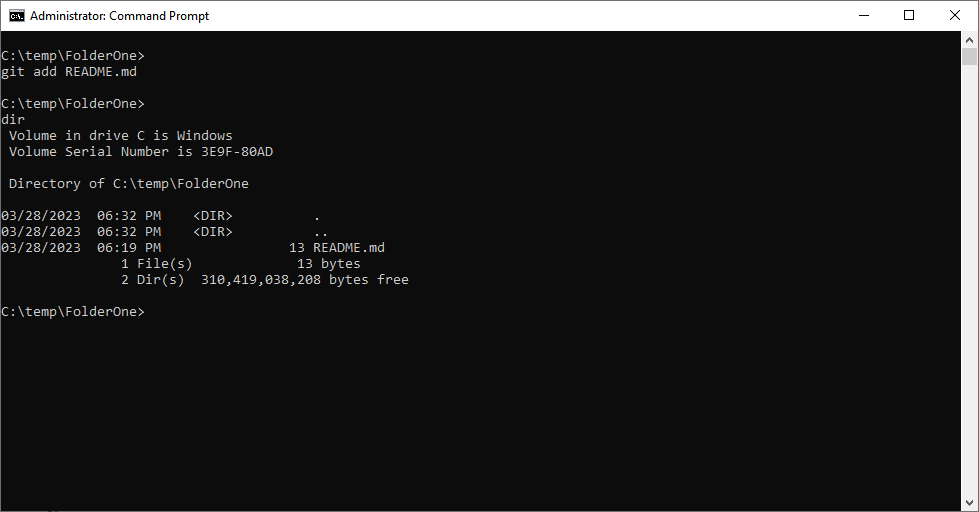
By making another reference to the Magic Engine, I hope you can see how working *within* the predetermined areas and commands of Git, you can get your code\files to the desired location with ease.

Working Area

The README.md file that we created in the previous section is inside our Working Area. You can see it, you can update it, you can delete it and even when we just initialized the Git environment here, Git is completely oblivious to this file.

Staging Area

The Staging Area is how we first make Git aware of a file we need to track. To “move” a file to this area we will use the git add <filename> command:



After we executed this command nothing seems to have happened, but in reality, new information was created inside the .git folder related to this file, and if we execute the git status command we can see that Git is now aware of this file and it is within the list of files to be committed (we will explain more about committing files when we cover the Local Repository):

Text

Description automatically generated

Now is a good time to clarify that the term “moving” a file across the Git Workflow sometimes it just means that Git created reference files within the .git folder that points back to the File, and sometimes and actual copy of the File was made into a **Local** or **Remote Repository**.

From this point forward, the word *move\moving* will be presented without double-quotes.

There are several ways you can use the git add command:

* git add <file> will move this file to the **Staging Area**.
* git add <folder> will move the contents of this folder to the **Staging Area**.
* git add . will move all the current files and folders to the **Staging Area**.

Since Git is now keeping an eye out for this file, let’s mess with it. Using any text editor change the contents of the README.md file, then execute the git status command:

Text

Description automatically generated

You can see that Git is telling two things here: One, the original README.md file is still ready to be committed, and two, the latest changes to the README.md file are not staged for commit. In other words, if we perform a commit operation right now, only original README.md file will be moved to the **Local Repository**.

If we manually (using the same text editor) revert all the changes made to this file and run the git status command again:

Text

Description automatically generated

Git is now happy, like nothing happened.

Let’s do this again.

Modify the file and save it, run the git status command to confirm that Git is aware of the change, execute the brand new git reset command and then the git status command:

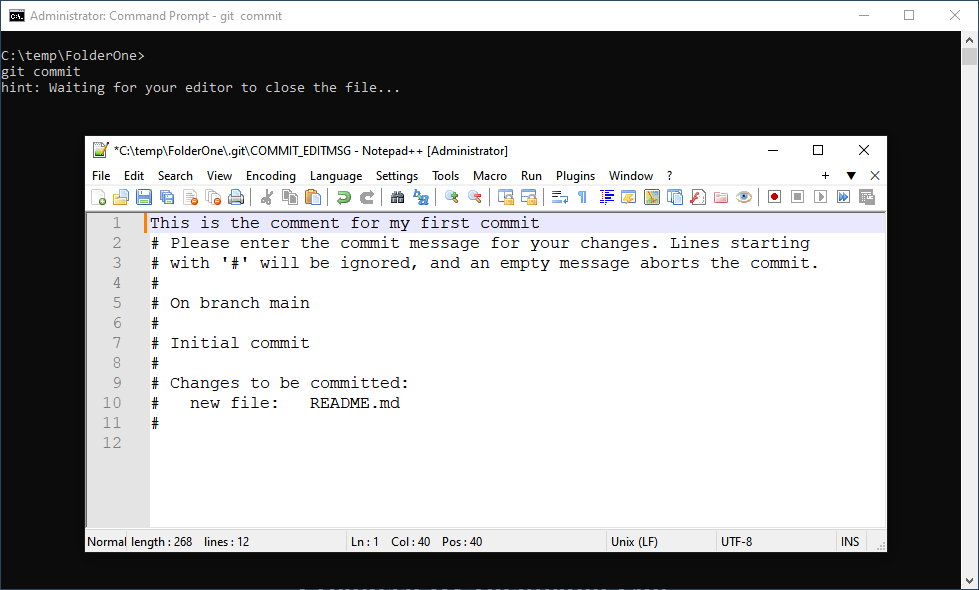
Text

Description automatically generated

The git reset command basically deletes a file from the Staging Area (notice that there are no changes to be committed); at the same time Git is now aware that the README.md file was at some point tracked. That is the reason is showing this file as “untracked”, it exists in the Working Area, but it is not in the Staging Area.

Local Repository

The next section in the Git Workflow is the Local Repository. Add the README.md file to the Staging Area one more time and then execute the git commit command:



Every commit action requires a comment describing the changes included in this process, here you can see that Git opened the default text editor program asking to enter this information.

Once we have entered a comment and close the text editor, the actual commit process will execute:

Text

Description automatically generated

The commit command creates an encoded copy of this file and places it in inside the .git folder in a way that is not to be easy to access it and\or change its contents.

If you include the comment within the commit command itself like this: git commit -m “This is the comment for my first commit”, no text editor will appear.

Executing the git status command now will return no information because no files are stored in the Staging Area any more, but there are other commands that we can issue to check the contents of the Local Repository.

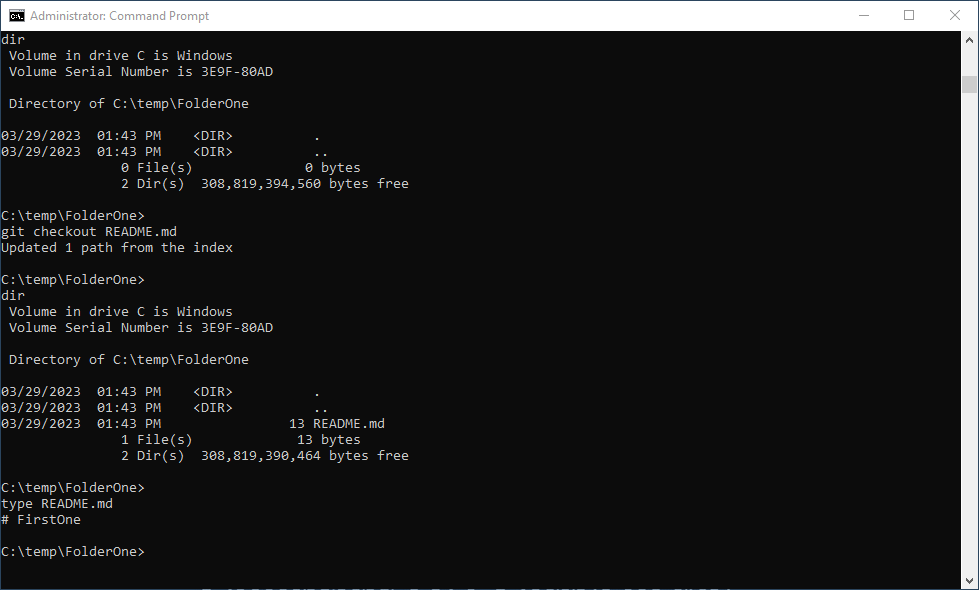
You can use the commands git log, git log --oneline and git log --stat to get information about the file(s) inside the Local Repository:

Text

Description automatically generated

We are not going to explain every detail of these output here, our focus is getting to know the different stages of the Git Workflow.

Now let’s see if we can have access to that copy of the file we just committed. Delete the README.md file from the Working Area, then execute the git checkout README.md command (the name of the file is case-sensitive).



If everything is ok, we will have a restored version of this file from the copy made by the commit command.

In comparison with the Staging Area, the Local Repository has more restrictions and nuances due to the nature of Git (keeping track of files, never losing a file change, etc.); be warned, we are going to start going into the weeds of Git.

We are going to continue work in this Local Repository by looking at the way Git keeps track of changes.

Open the README.md file and add a second line of text to it:

Graphical user interface, text, application

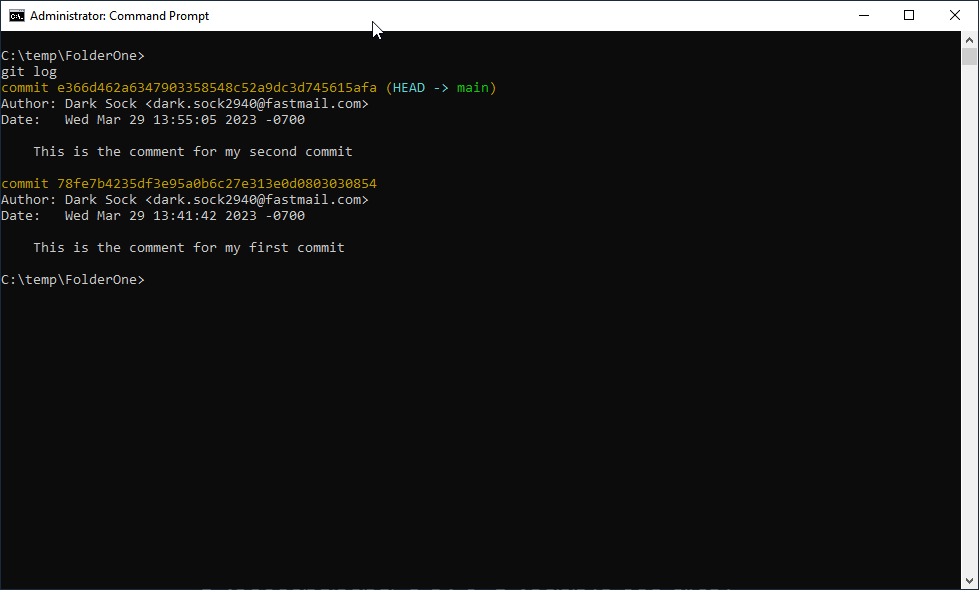
Description automatically generated

Confirm that Git is aware of this change, and move the file to the Staging Area and then to the Local Repository:

Text

Description automatically generated

Issuing the git log command will reveal two entries (changes) in the Local Repository:



The order in which this information is displayed goes from most recent change at the top and oldest change at the bottom. The long alphanumeric string next to the commit token is a unique ID assigned by Git to every commit procedure, using this number and the description associated to it, we can work with each one of these entries.

For example, if we want to bring to the Working Area the first version of this file (the one with a single line of text in it), we can use the git checkout command but instead of the name of the file, we will use the first 7 characters of the unique ID for that commit:

Text

Description automatically generated

A couple of things happened here: Yes, we got the first version of the file into our Working Area, and a message telling us the HEAD is now at 78fe7b4.

Do not be scared now, but we are on a state called “Detached HEAD” (no joke).

If you go back a couple of pages to the place where we made our first commit operation, you see the text (HEAD -> main) next to that operation; and more recently, that same text appeared next to our second commit operation. Furthermore, if we issue the git log command, we can see that we not only have a single entry in the log (our first commit entry) but the text (HEAD) is pointing to nowhere:

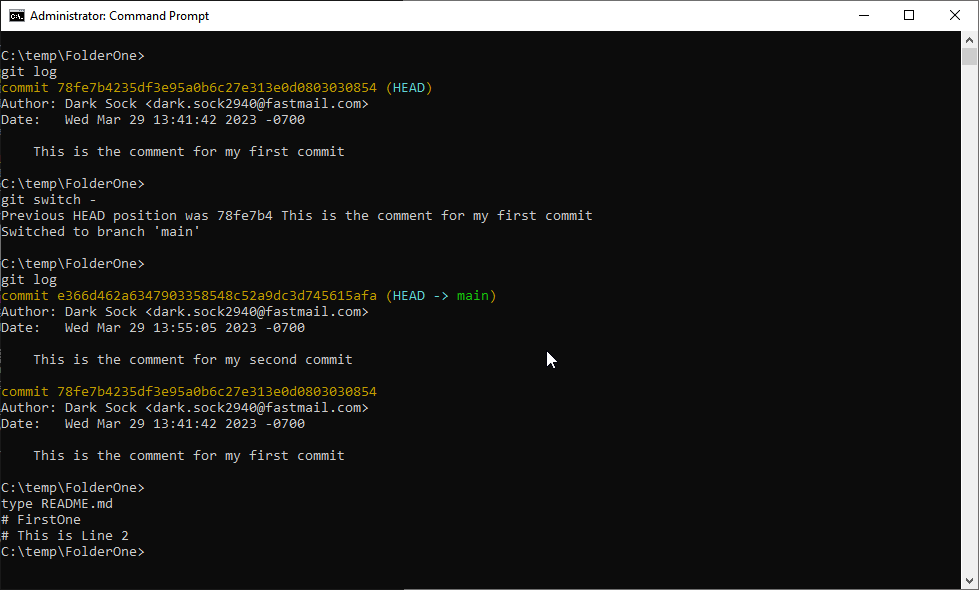
Text

Description automatically generated

Ok, so why this happened?

* In Git, the (HEAD) is supposed to always point to the latest commit entry, since it tells Git where to insert the information for the next commit operation.
* By issuing the git checkout command and requesting an older commit entry, we told Git to move the (HEAD) pointer to that commit and get that version of the file for us.

So now we can do two things: Save this old version of the file into a new Branch (since we will cover Branches at a later Lesson we will pass on this option for now), or undo this whole operation with the git switch – command and go back to sanity:

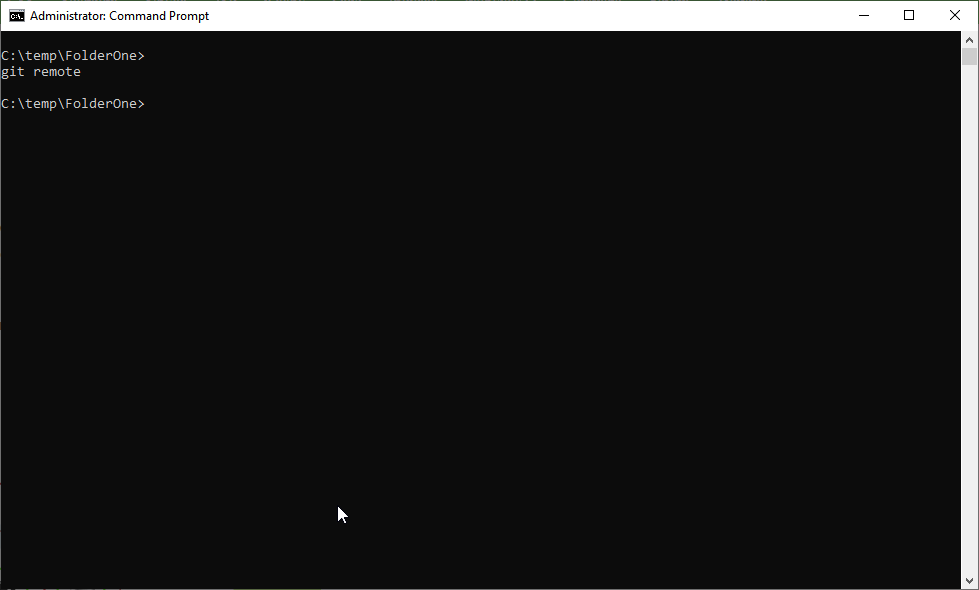


Remote Repository

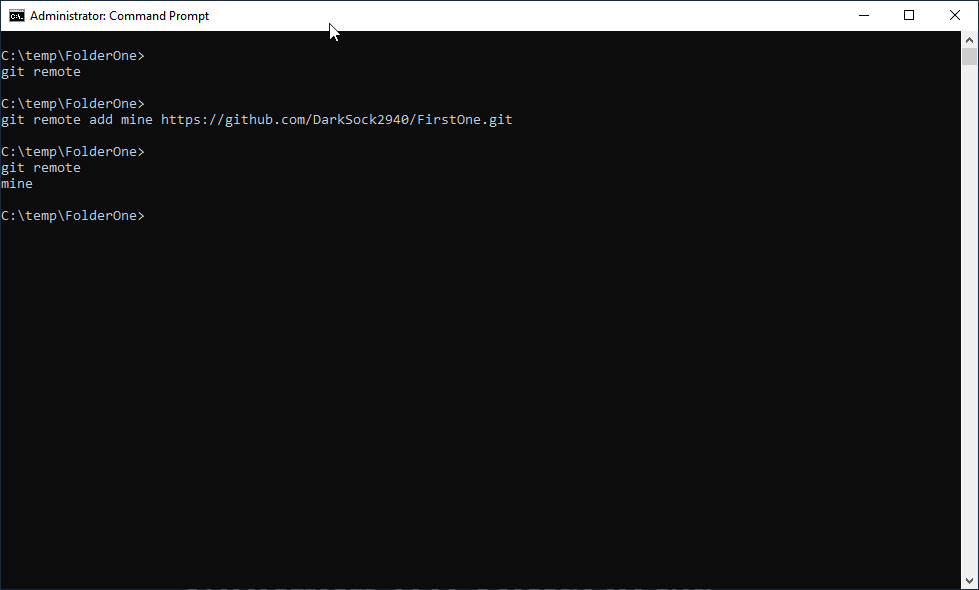
The last section inside the Git Workflow is the Remote Repository.

This is the part where we upload our code\files to the cloud (the Git service) so other programmers can see and use our changes, but first we need to create and configure a connection to our Remote Repository, and for that we will use the git remote command.

In Git, remote refers to a Remote Repository, and since we have stablished the abundance of Git services, we need to tell our local configuration of Git what service we are going to connect to. If we issue the git remote command, we should get no response:



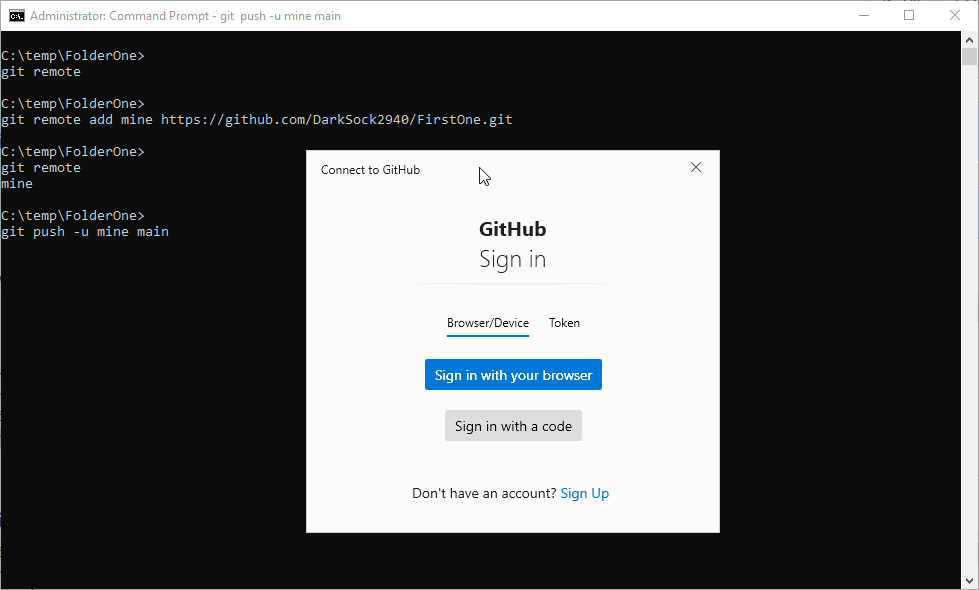
To setup a connection to our GitHub account we are going to use the provided command line from the GitHub screen (git remote add origin https://github.com/DarkSock2940/FirstOne.git), but we are going to change the word origin with mine:



If in the future, you need to connect to another Git Service, you can setup this entry with a name that makes sense to you.

Now that we have this Remote Repository configured, is time to upload our code\file to it.

From the GitHub site we will use the last command from the page (git push -u origin main) without forgetting to make the same change as before. It is possible that a screen will popup asking us to provide the required credentials to prove that we are allowed into this Repository.



After verifying that we have access to this Repository, we should see some text lines informing us of the status of this process:

Text

Description automatically generated

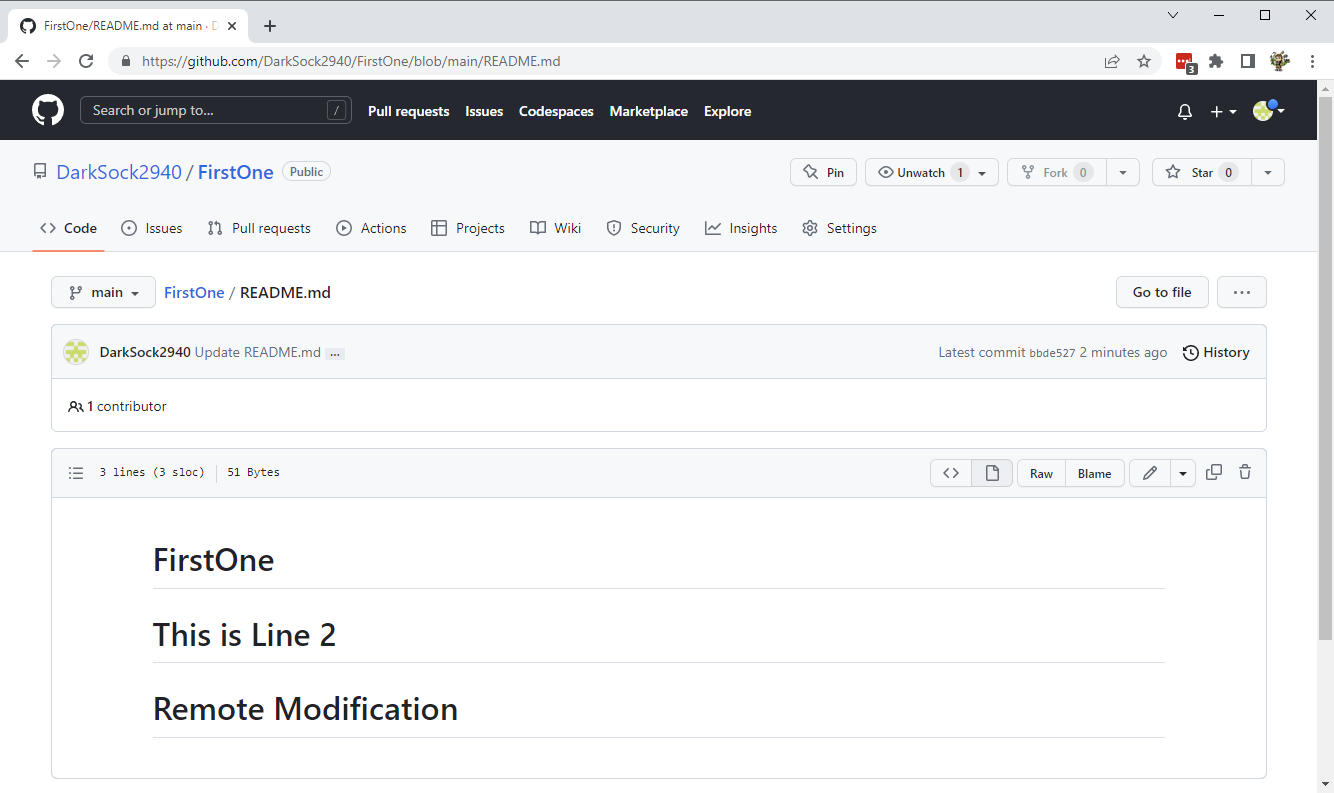
If we go to our browser and click in our Repository, we should see the latest version of the README.md file displayed on the screen:

Graphical user interface, text, application, email

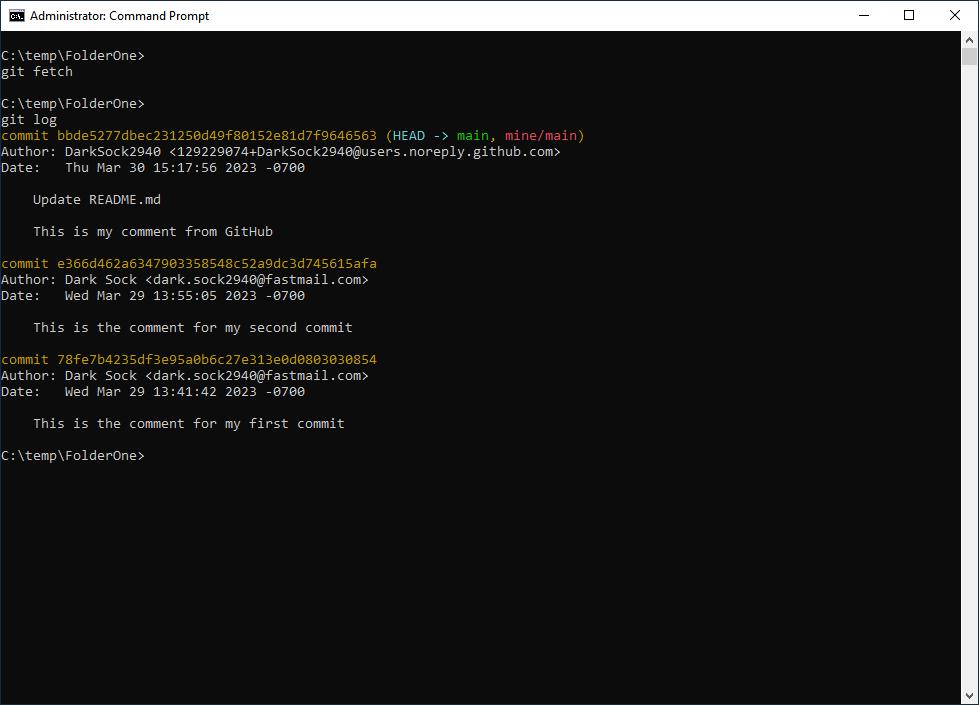
Description automatically generated

We are almost done with this section; the last step is to make changes to this file using GitHub, and then get a copy of this version of the file back to our computer.

To edit this file in GitHub, press the pencil icon to the right of the README.md file, create a 3rd line of text on this file, enter a comment, and commit the change:

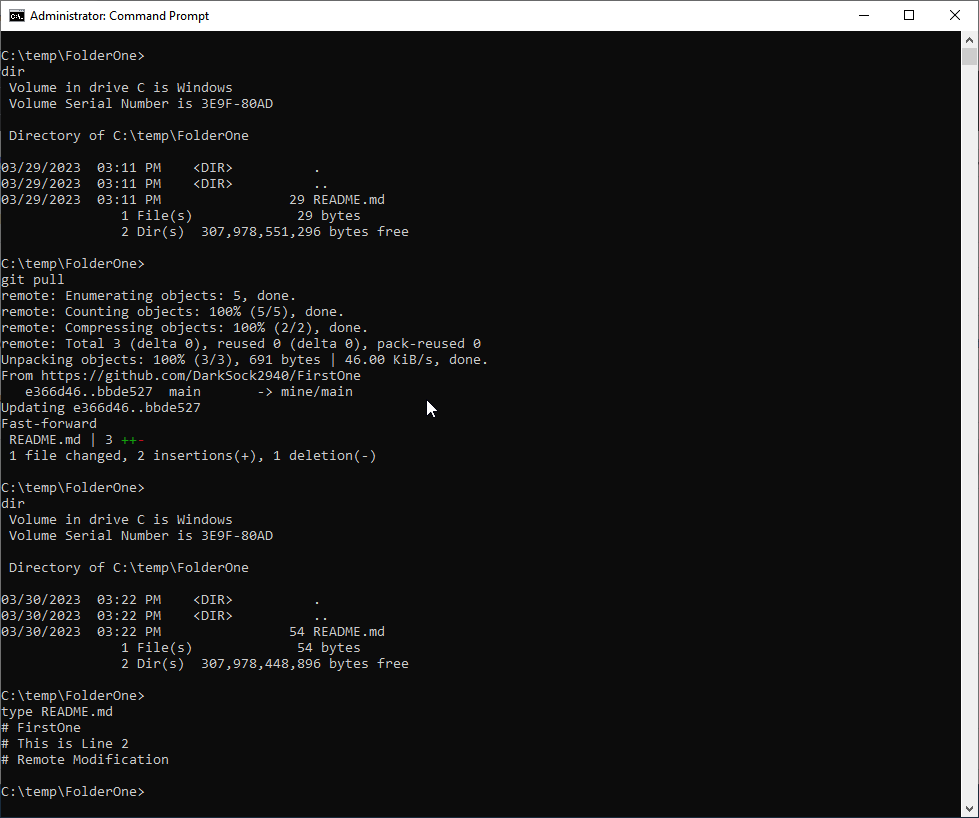


Let’s go back to the CMD window and use the git fetch command to get a copy of our Remote Repository files into our Local Repository:



And there you go; we got a copy of that file into our local computer and a third change has been recorded by Git into the log.

Lastly, use the git pull command to get a copy of our Remote Repository files all the way into Working Directory:



That is all Folks!

We have gone over every location within the Git Workflow, we used several Git commands to move files from location to location and seen the consequences of modifying existing files and checking out older versions.

As mentioned at the beginning of this document, this is just the basics of Git with just enough information to get you started with it. Next Lesson will bring Magic into the mix and see the process of moving code between the different Areas and Repositories for the Git Workflow.

## FEEDBACK FORM

Course Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Instructor's Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date:\_\_\_\_\_\_\_\_\_

Company Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Was the class\course relevant to your needs?

o Very relevant o Relevant o No relevant at all

2. What was your level of experience using this tool before coming into the class?

o Fairly proficient o Intermediate o Basic o Little or no knowledge

3. How was the quantity of information presented on this class\course?

o Too much o Just right o Not enough, because: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. How was the instructor\course’s pace during the course?

o Too fast o Just right o Too slow

5. How useful was this documentation?

o Very useful o Useful o Not useful, because: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. How would you rate the overall learning experience?

o Learned a lot o Learned something o Learned little, because:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. I would like to add the following topics to this course:

8. I wish the course did not spend so much time in:

9. I wish the course could have spent more time in:

10. If I were to take another course of Magic, I will like it to covert the following topics:

11. Additional Comments: