



INTRODUCTION TO GIT AND GITHUB

UPDATED: 10.10.2019

CONTENTS

Click on the "TU" icon in the corner of work pages to navigate back here

SECTION 01

GIT OVERVIEW	5
AGENDA	6
SOURCE CONTROL REVISIT	6
GIT STRUCTURE AND FILE STORAGE	8
FILE INTEGRITY INSIDE GIT	9
WORKFLOW	10
BASIC BASH COMMANDS	14
CONFIGURATION	14
LAB ACTIVITIES	16

SECTION 02

GIT YOUR CODE

IN THE REPO	19
GIT STATUS	20
GIT ADD	21
GIT COMMIT	22
VIEWING THE COMMIT HISTORY	24
VIEWING THE DIFFERENCES	24
LAB ACTIVITIES	26

SECTION 03

BRANCHING & MERGING	29
CREATE A BRANCH	31
LIST BRANCHES	31
GIT THE DIFFERENCE	32
LAB ACTIVITIES	34

SECTION 04

CONFLICT RESOLUTION	39
IDENTIFY THE CONFLICT	41
RESOLVE THE CONFLICT	42

SECTION 05

GITHUB OVERVIEW	45
WHY USE GITHUB	46
ABOUT GITHUB	46
LAB ACTIVITIES	46

SECTION 06

UNDERSTANDING REMOTES	51
LUKE CREATES AND MANAGES HIS REMOTE	53
LAB ACTIVITIES	56

SECTION 07

GIT HAPPENS	61
LAB ACTIVITIES	63

SECTION 08

COLLABORATING

WITH REMOTES	67
FORK	68
UPSTREAM	69
PULL REQUEST	69
ORIGIN	70
LAB ACTIVITIES	71
REVIEW THE CHANGES	76
MERGE PULL REQUEST	78
UPDATE LOCAL REPOSITORY	78

SECTION 09

GIT FLOW AND YOU	81
-------------------------	----



"Do you want to know who you are?
Don't ask. Act!
Action will delineate and define you."
Thomas Jefferson



SECTION 01

GIT OVERVIEW

AGENDA

Today we will complete the following:

- Git overview
- Introduction to Bash
- GitHub Overview
- Introduction to Remotes
- Collaborating using GitHub
- Basic Git Workflow
- All the Labs

SOURCE CONTROL REVISIT

What are some benefits of source control?

In source control, where do we store our code?

Why would you want to create a branch in source control?

[illegible]

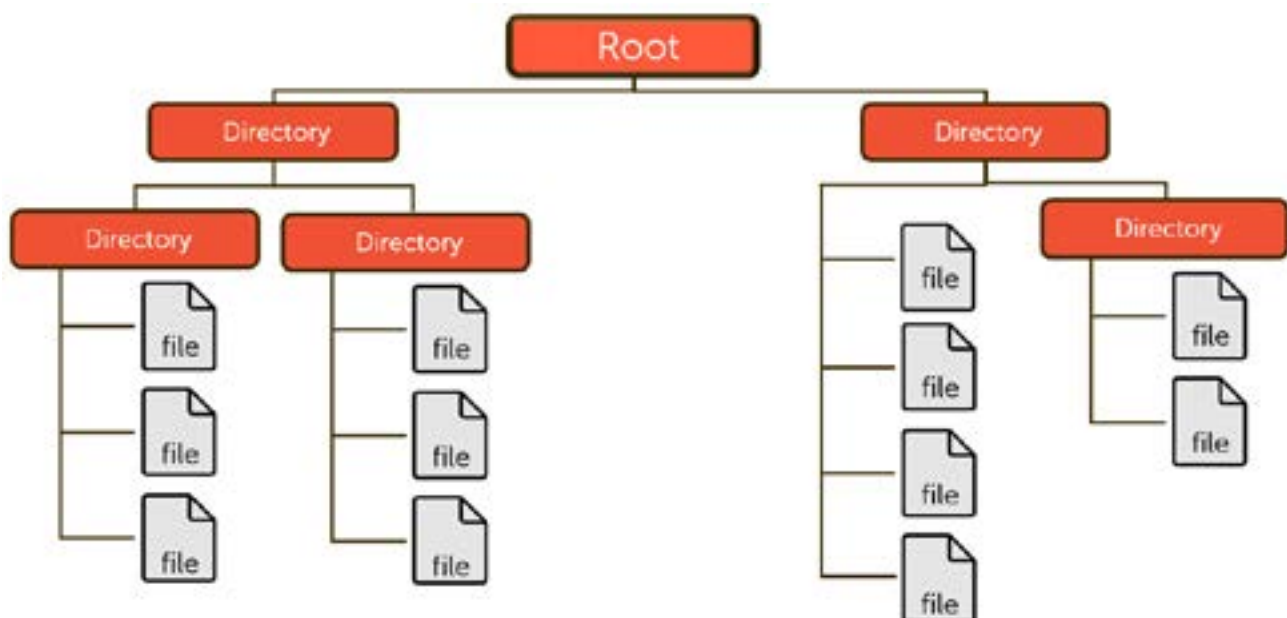
It is important to know how Git works and is structured to use it effectively.

- **Disturbed Source Control:** Git is an example of distributed source control.
- **Operations are Local:** Because Git uses distributed source control, almost every operation in Git is done locally. Everything committed goes to your local repository (or repo) and stays there until they are pushed to a master repository. Developers can branch and merge locally. This allows developers the ability to work offline.
- **Directories:** Directories are the same as folders (Think of the windows folder structure). These directories hold files and sometimes sub-directories.

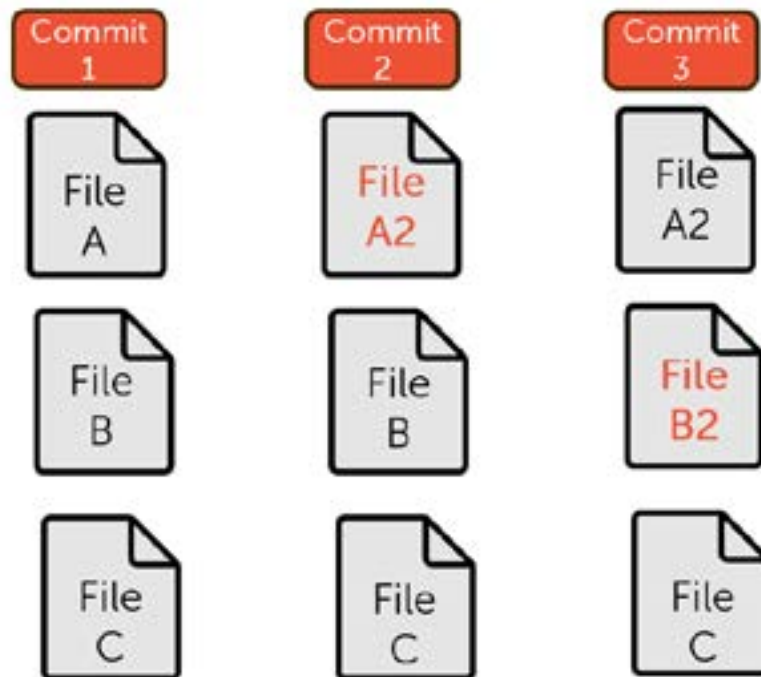
REPOSITORIES ARE OFTEN CALLED "REPOS".

GIT STRUCTURE AND FILE STORAGE

In Git, a directory is the equivalent of a folder. Within directories, you can have subdirectories and files. A File System is a collection of files and directories into a tree-like pattern.



Every time a file is committed, Git takes a snapshot of everything that has changed in the local repository. If there are files in the repo that have not changed, Git doesn't store the file again.



FILE INTEGRITY INSIDE GIT

When we transfer files, it is important to maintain file integrity. To do this, Git uses a “checksum” action to verify that the files are the same before the file is stored. This helps to identify if a file was lost in transit or corrupted.

Git uses a revision identifier known as a Secure Hash Algorithm 1 or SHA-1 hash to identify revisions to a file. The SHA-1 hash algorithm produces a 40-character string made up of hexadecimal characters, providing a unique identifier for each revision.

GIT STORES EVERYTHING IN ITS DATABASE BY THE HASH VALUE OF THE FILE, NOT BY THE FILE NAME.

Example SHA-1 Hash

24b9da6552252987aa493b52f869cd6d3b00373

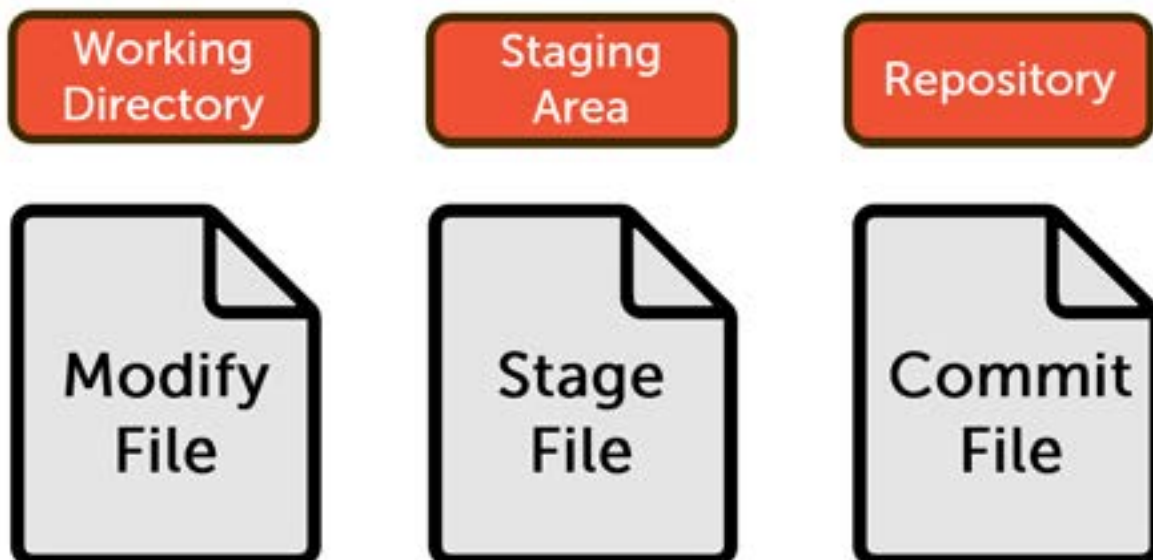
Hexadecimal Characters

0-9 and a-f

WORKFLOW

Files in Git have three possible states: committed, modified, and staged. The workflow process is:

1. **Modify:** You modify or change the files in your working directory.
2. **Stage:** When you are done modifying the file, you stage the files by placing them in the staging area. The staging area stores information about what will go into your next commit and it's sometimes referred to as the index.
3. **Commit:** After the files are staged, they can be committed, which stores the files permanently in your **local repo**.



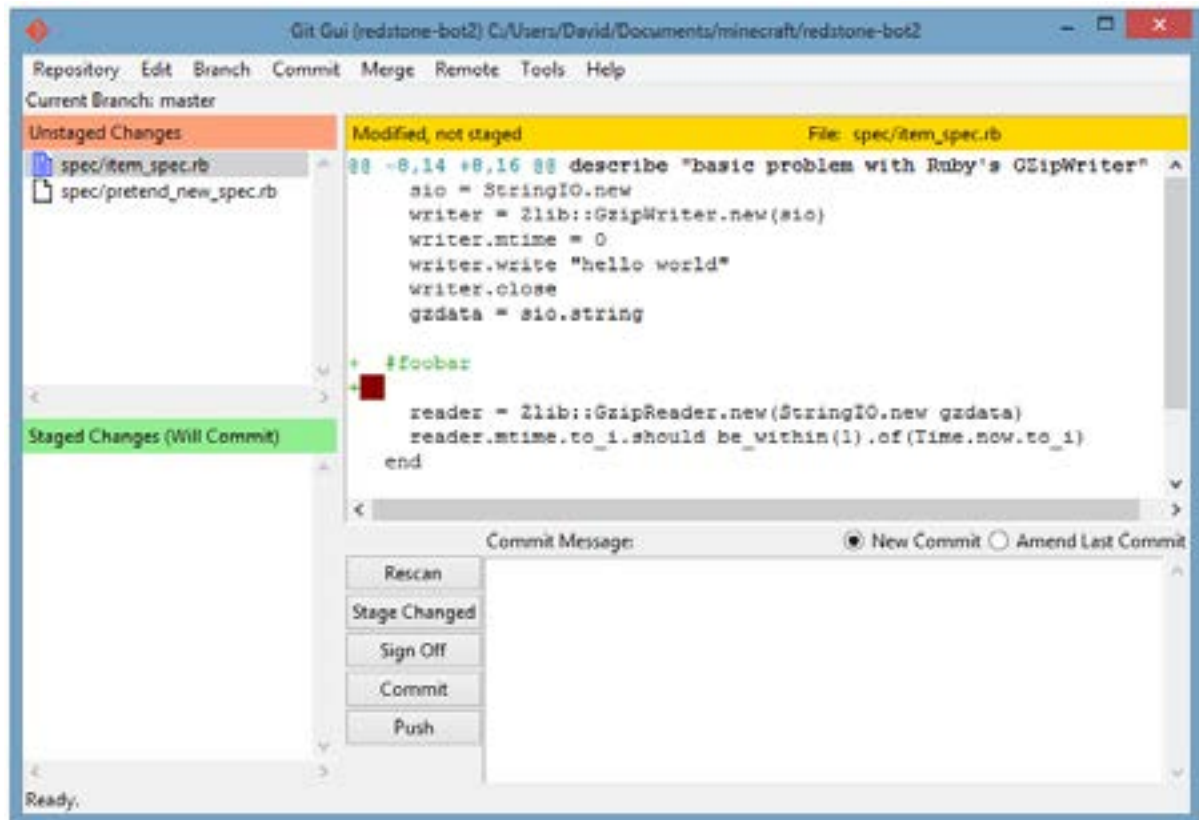
So far, everything has been completed locally. At some point the changes will need to be “pushed” to a master repository. At Quicken Loans we use GitHub Enterprise as our repository hosting site.

NOTES

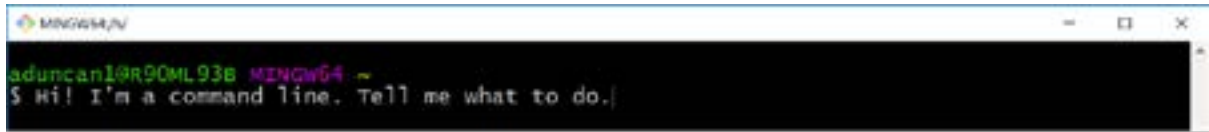
[illegible]

There are two ways we can tell Git what to do, through a graphical user interface (GUI) or through a command line.

A GUI allows you to interact with a program by pointing and clicking.



The command line is a text window (or shell) where you type commands for the program to execute.



NOTES

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

BASH is the shell, or the interpreter for command line language. The name is an acronym which stands for Bourne-Again SHell. Although the name behind BASH doesn't make much sense without a Google search, BASH commands are easy and straight forward to use.

BASIC BASH COMMANDS

- | | |
|-------------------------------|---|
| • pwd | Identifies the directory you are currently in |
| • ls | Lists the directory contents |
| • cd <i>name of directory</i> | Changes to the directory entered |
| cd.. | Navigates up one directory |
| cd | By itself navigates back to your home directory |
| • clear | Clears all text on the screen |



REFER TO THE COMMAND LINE REFERENCE HANDOUT OR APPENDIX A FOR MORE BASH COMMANDS AND INFORMATION.

CONFIGURATION

To create a local repository, you need to do the following:

- Configure account
- Create a directory
- Create text file
- Navigate to newly created directory
- Turn directory into a Git repository

Configuring your account allows Git to know who is making changes and how to contact them.

```
git config --global user.name "Your Name"
```

FYI, when you do this in lab please ensure you are entering your actual name and not the words "Your Name". The global inside this command means that this changes the configuration for all local repositories created by this user. So, if I had 5 local repos on my computer, this command will affect all 5 repos.

```
git config --global user.email "YourName@something.com"
```

Both of these configuration settings are very important if you are working as part of a development team. They will allow other team members to know who made changes and how to contact them.

- `git config --list`

This command allows you to check your settings. If there is a mistake, rerun the command and it will overwrite the old entry.



REMEMBER, DIRECTORY = FOLDER

What command do we use to navigate from one directory to another?

What command do we use to initialize a Git repository from a directory?

LAB ACTIVITIES

CONFIGURE ACCOUNT

1. Ensure you are in BASH
2. Type: **git config --global user.name "Your Name"**
3. Press: **Enter**
4. Type: **git config --global user.email "yourname@something.com"**
5. Press: **Enter**

Result: Your name and email will now be linked to all operations inside your repository

CREATE DIRECTORY

1. Open File Explorer
2. Navigate to your C Drive, then click on temp
3. Inside your temp folder, right click, scroll to new, and then click on folder
4. Name the folder ***YourInitials_git_lab***
5. Press: **Enter**

CREATE TEXT FILE

1. Double-click on ***YourInitials_git_lab***
2. Right-click on open space inside folder
3. Point cursor to New
4. Click Text Document
5. Name text document **file_1**
6. Press: **Enter**
7. Double-click on **file_1.txt**
8. Insert 3 lines of text into blank file (Feel free to be creative but keep it short and work-safe)
9. Save and close file

NAVIGATE TO DIRECTORY

1. Ensure you are in BASH
2. Find out what directory you are currently in
3. Type `cd`
4. Press: Enter
5. Type `cd ..`
6. Press: Enter
7. Type `cd /c/temp`
8. Press: Enter
9. Type `cd YourInitials_git_lab`
10. Press: Enter

TURN DIRECTORY INTO GIT REPOSITORY

1. Ensure you are in ***YourInitials_git_lab*** directory
2. Type: **`git init`**
3. Press: **Enter**

Result: You have initialized (or created) an empty repository

"What is now proved was
once only imagined."
William Blake



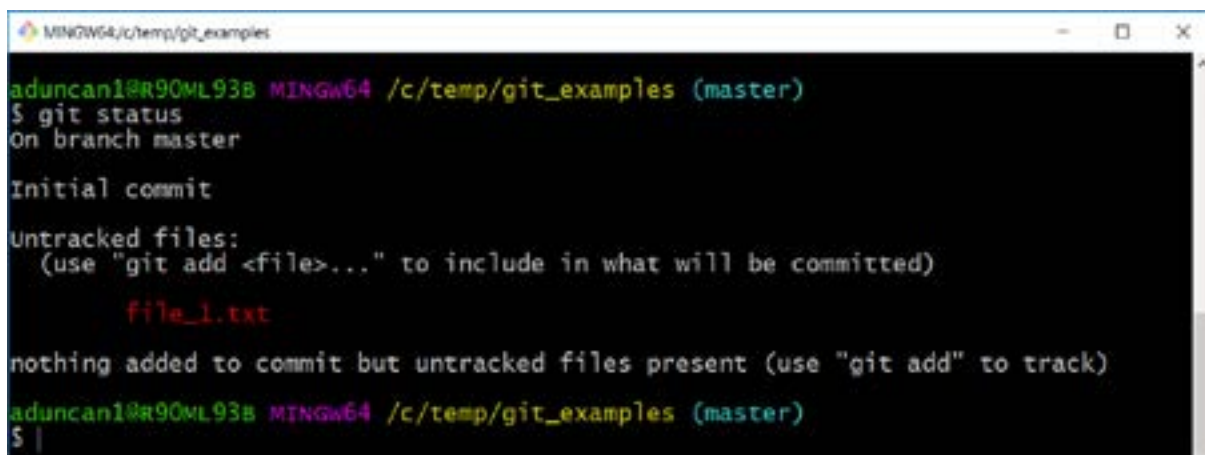


SECTION 02

GIT YOUR CODE IN THE REPO

After you turn your directory into a repository, you must stage and then commit any files that were inside. Git's 2-step process of staging and then committing files allows you to have more control over what you commit and when, so you can split up large changes into multiple commits, review changes, and handle merge conflicts more easily.

GIT STATUS



```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git status
on branch master

Initial commit

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

    file_1.txt

nothing added to commit but untracked files present (use "git add" to track)
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ |
```

Depending on the situation, git status can tell us one of the following:

- **Untracked:** If you have any untracked files, which are files that haven't been staged or committed, Git will not track any changes that are made to those files. Untracked files are color-coded in red.
- **Modified:** If you have files that have been modified but haven't been staged.
- **Staged:** If you have files that are staged and ready to be committed. The file name(s) will now be in green.

Additionally, git status shows us:

- **On Branch:** The branch you are on displays. If there are multiple branches it is important to know which branch you are working on.

GIT ADD

Git add moves files from the working directory and puts them in the staging area.



NOTICE THAT THE NAME OF THE FILE BEING MOVED IS PLACED DIRECTLY AFTER THE COMMAND.

```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git add file_1.txt
```



THE GIT ADD COMMAND DOES NOT OUTPUT A MESSAGE WHEN RUN. TO VIEW THE RESULTS, RUN THE GIT STATUS COMMAND.

```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file_1.txt

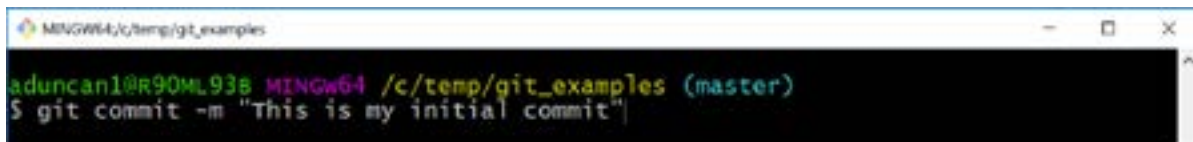
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ |
```

In the example above, Git is telling us that we have a file to be committed. In this case, the file we just added.

GIT COMMIT

As the name implies, git commit takes our staged files and commits them to our repository.

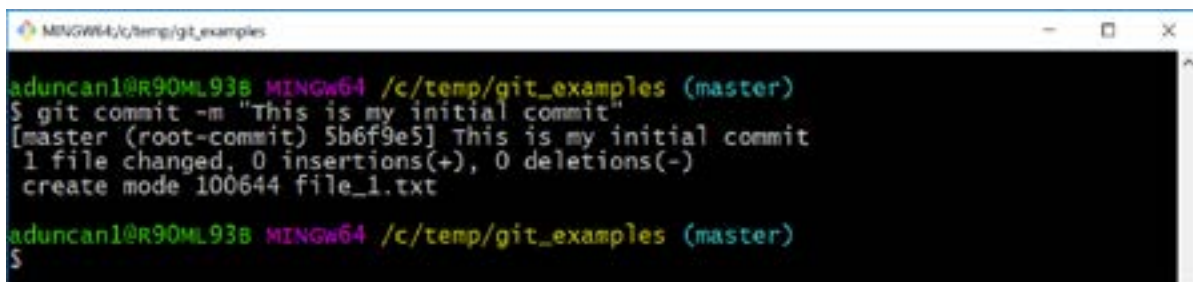
Notice the `-m` after git commit. This is called an option and most Git commands have options you can add onto the command which will give it additional functionality. In this case, the `-m` allows you to add a message (aka comment). Messages should be descriptive and give insight into what you are doing.



```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git commit -m "This is my initial commit"
```

After we run the command, Git provides a summary of what was committed. In the example below, Git is saying that this commit happened on the master branch followed by a series of characters. ([master (root-commit) 734c7a8]) This is the first seven characters of the SHA-1 hash that was generated.

Under that we see the number of files changed and inserted and the name of the file or files committed.



```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git commit -m "This is my initial commit"
[master (root-commit) 5b6f9e5] This is my initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file_1.txt
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$
```

[illegible]

VIEWING THE COMMIT HISTORY

"The commit history will provide you a list of all commits, with the most recent commit listed first. The history provides you with the commit id, the author of the file, the date it was committed, and the commit message."

"To view commit history, use the git log command."



IF YOU DON'T USE AN OPTION, GIT WILL LIST ALL OF THE COMMITS MADE TO THAT REPO.

```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git log
commit 41192eef02fbc93fd33c7e9646a158732e38c2
Author: Ali Duncan <aliduncan@quickenloans.com>
Date: Thu Nov 29 16:24:25 2018 -0500

    Adding another line of text to file_1 for git log example

commit 5b6f9e587be1f138904e2fd70715c4ee467d9437
Author: Ali Duncan <aliduncan@quickenloans.com>
Date: Thu Nov 29 16:15:12 2018 -0500

    This is my initial commit

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ |
```

VIEWING THE DIFFERENCES

Git can show the differences between the file version in the working directory and the version that is in the repository. It can also show the difference between two versions that were previously committed to the repo.

To view the differences between the files, use the git diff <filename> command.

```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git diff diff_ex1.txt
diff --git a/diff_ex1.txt b/diff_ex1.txt
index 1147856..163f2f1 100644
--- a/diff_ex1.txt
+++ b/diff_ex1.txt
@@ -1,3 +1,4 @@
+Humpty Dumpty
  Sat on a wall
  Humpty Dumpty
  Had a great fall.
\ No newline at end of file

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ |
```

Another thing we can use git diff for is viewing the differences between versions of the same file. It is helpful to view the diff of two versions so you can see bug fixes, code reviews, double-check that what you are committing is what you want to commit, and when you want to know how the file has changed between two points in time.

```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git diff 1cd034a..44d026f
diff --git a/diff_ex3.txt b/diff_ex3.txt
index bc56367..ced2909 100644
--- a/diff_ex3.txt
+++ b/diff_ex3.txt
@@ -1,4 +1,4 @@
-Jack and Jill drove up the mountain
+Jack and Jill went up the hill
  To fetch a pail of water;
  Jack fell down and broke his crown
  And Jill came tumbling after.
\ No newline at end of file

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$
```

LAB ACTIVITIES

CHECKING THE STATUS OF OUR FILE

- 1) Type: **git status**
- 2) Press: **Enter**

What is your file's status?

STAGING YOUR FILE

- 1) Type: **git add file_1.txt**
- 2) Press: **Enter**

RESULT: Your file is placed in the staging area (or index)

COMMITTING YOUR FILE

- 1) Type: **git commit -m "This is my first commit"**
- 2) Press: **Enter**

RESULT: Your file has been permanently stored in your repository

- 3) Type: **git status**
- 4) Press: **Enter**

What is the status of your working directory?

VIEWING THE DIFFERENCE BETWEEN YOUR WORKING DIRECTORY AND YOUR REPO

- 1) Open **file_1.txt**
- 2) At the top of the document, add a line of text
- 3) Save and close your file
- 4) Inside BASH, view your file's status



DO NOT STAGE YOUR FILE

- 5) Type: **git diff file_1.txt**
- 6) Press: **Enter**

RESULT: The differences between your working directory and your repo are displayed.

- 7) Stage the file by typing: **git add file_1.txt**
- 8) Press: **Enter**
- 9) Commit the file by typing: **git commit -m "Your message here"**
- 10) Press: **Enter**

VIEWING THE DIFFERENCE BETWEEN TWO VERSIONS

- 1) View your commit history using

What are the first 7 characters of your first commit ID? _____

What are the first 7 characters of your last commit ID? _____

- 2) Type: **git diff ID of first commit..ID of last commit** (such as: a24c1eb...b035d1a)
- 3) Press: **Enter**

RESULT: The difference between your first and last commit are displayed.

"Knowledge is a treasure,
but practice is the key to it."
Lao Tzu

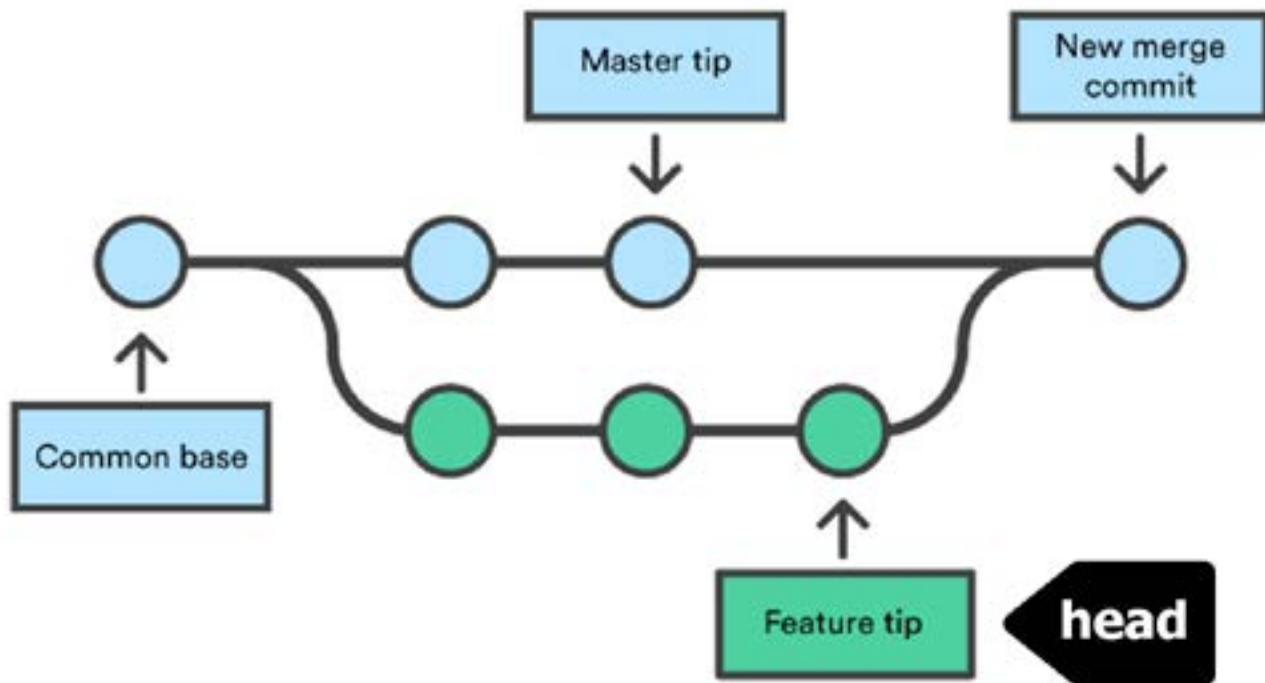




SECTION 03

BRANCHING & MERGING

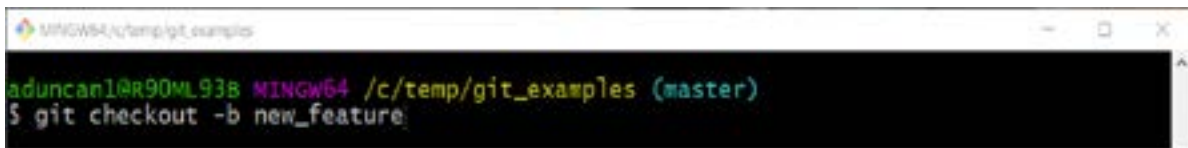
Branching allows you to isolate the code base, so you can create new features, test code, or fix bugs. Branching strategies will vary from team to team. Below is an example of branching.



Git uses the head pointer which points to the last commit of the branch that you are currently on.

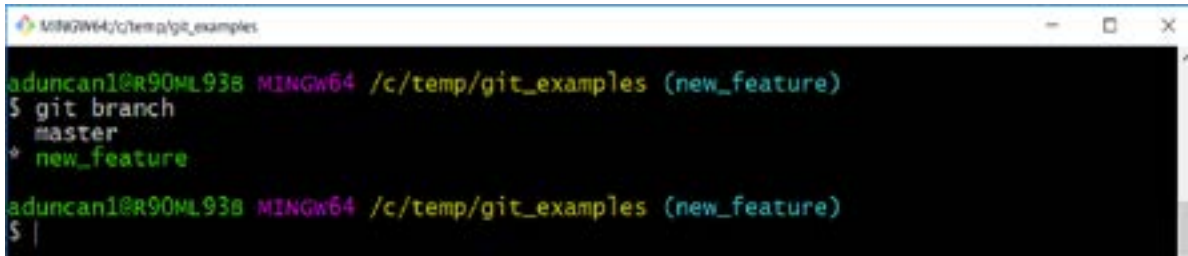
CREATE A BRANCH

To create a branch, use the command `git checkout -b <new branch name>`. In the example below, the branch being created is called `new_feature`.

A terminal window titled 'MINGW64/c/temp/git_examples' showing a user 'aduncan1@R90ML938' in a 'MINGW64' environment at the path '/c/temp/git_examples' on the 'master' branch. The user enters the command 'git checkout -b new_feature'.

LIST BRANCHES

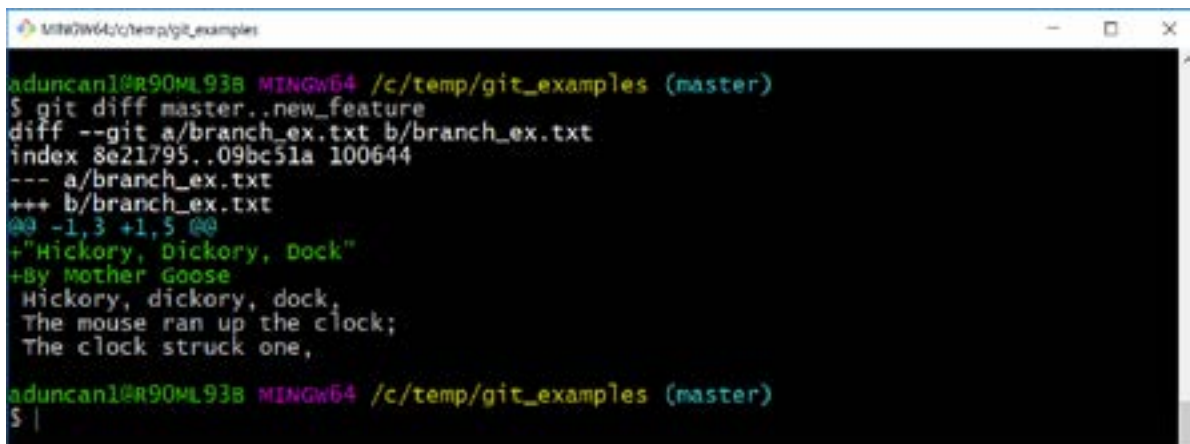
If the `git branch` command is run, it shows that there are now two branches, `master` and `new_branch`. In the example below notice that `new_branch` is green and has an asterisk. This indicates that you are currently on `new_branch`.

A terminal window titled 'MINGW64/c/temp/git_examples' showing the same user and environment as the previous screenshot, but now on the 'new_feature' branch. The user enters the command 'git branch'. The output shows two branches: 'master' and 'new_feature'. The 'new_feature' branch is listed with an asterisk and is highlighted in green, indicating it is the current branch. The prompt then shows the user is still in the 'new_feature' branch.

GIT THE DIFFERENCE

After staging and committing changes on the selected branch, there are now two different versions, each on a different branch. To view the differences, remember to run the `git diff` command. In this example, we are viewing the difference between `master` and `new_branch`.

Git displays the differences indicating that the version on `new_branch` has two additional lines that are not in the version on `master`. The new lines are in green and use a plus sign to indicate they are additional lines of code.



```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git diff master..new_feature
diff --git a/branch_ex.txt b/branch_ex.txt
index 8e21795..09bc51a 100644
--- a/branch_ex.txt
+++ b/branch_ex.txt
@@ -1,3 +1,5 @@
+"Hickory, Dickory, Dock"
+By Mother Goose
 Hickory, dickory, dock,
 The mouse ran up the clock;
 The clock struck one,

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ |
```

NOTES

[illegible]

After you are satisfied with the changes made to code on a branch, it is time to merge the different versions together, ensuring that the latest or correct version is merged into the other version. Merging recombines codebases that were separated during branching.



IT IS IMPORTANT TO BE ON THE BRANCH THAT YOU ARE MERGING INTO WHEN YOU RUN THE MERGE COMMAND.

```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (new_feature)
$ git checkout master
Switched to branch 'master'

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ |
```

After ensuring that you are on the branch you want to merge into, use the git merge command along with the name of the branch that is being merged from. In the example below, we are merging from new_branch into master.

```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git merge new_feature
Updating f831387..7315f71
Fast-forward
 branch_ex.txt | 2 ++
 1 file changed, 2 insertions(+)

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$
```

The results of the merge display. Git indicates it was a simple merge with the term Fast-Forward. It also indicates the number of files merged and the number of lines added.



IF YOU WANT TO DELETE THE BRANCH NOW THAT THE INFORMATION IS MERGED INTO THE MASTER USE THE COMMAND, GIT BRANCH -D BRANCHNAME. FOR EXAMPLE, GIT BRANCH -D NEW_BRANCH.

LAB ACTIVITIES

CREATE A NEW BRANCH

- 1) Type: **git checkout -b new_feature**
- 2) Press: **Enter**

A branch has been created with the name new_feature

- 3) Type: **git branch**
- 4) Press: **Enter**

What branches are listed?

-
- 5) Create a new text file called **file_2**
 - 6) Open **file_2.txt** and insert 2 lines of text
 - 7) Save and close your file
 - 8) **Stage** your file
 - 9) View your file's status
 - 10) **Commit** your file and add a comment

SWITCH BRANCHES

- 1) Type: **git checkout master**
- 2) Press: **Enter**

RESULT: You are now on the branch, master

- 3) Open file explorer and navigate to your repo

What do you see in your repo?



THE FILE YOU JUST CREATED, FILE_2.TXT, SHOULD NOT BE VISIBLE.

VIEW DIFFERENCES BETWEEN BRANCHES

- 1) Type: **git diff master..new_feature**
- 2) Press: **Enter**

RESULT: Differences between master and new_feature are shown

PERFORM MERGE

- 1) Ensure you are on the master branch
 - If not, switch to master
- 2) Type: **git merge new_feature**
- 3) Press: **Enter**

RESULT: Changes from new_feature have been merged to the master branch

What commit IDs were listed? Updating

- 4) View your commit history

What commit ID do you think HEAD is pointing to?

[illegible]

"The only person who is educated
is the one who has learned
how to learn and change."
Carl R. Rogers

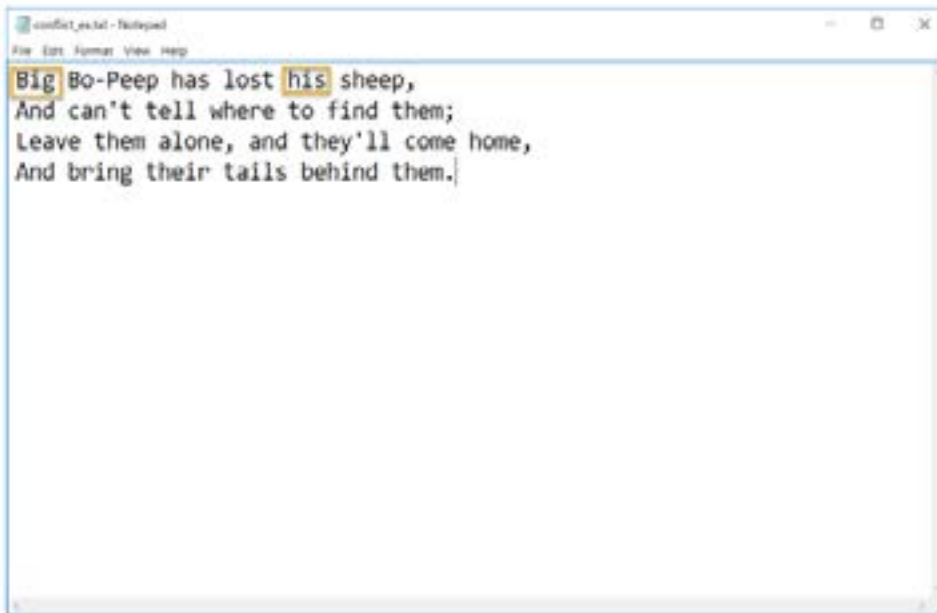


SECTION 04

CONFLICT RESOLUTION

Merge conflicts can happen when the same part of a file was changed differently on two branches that you are merging together.

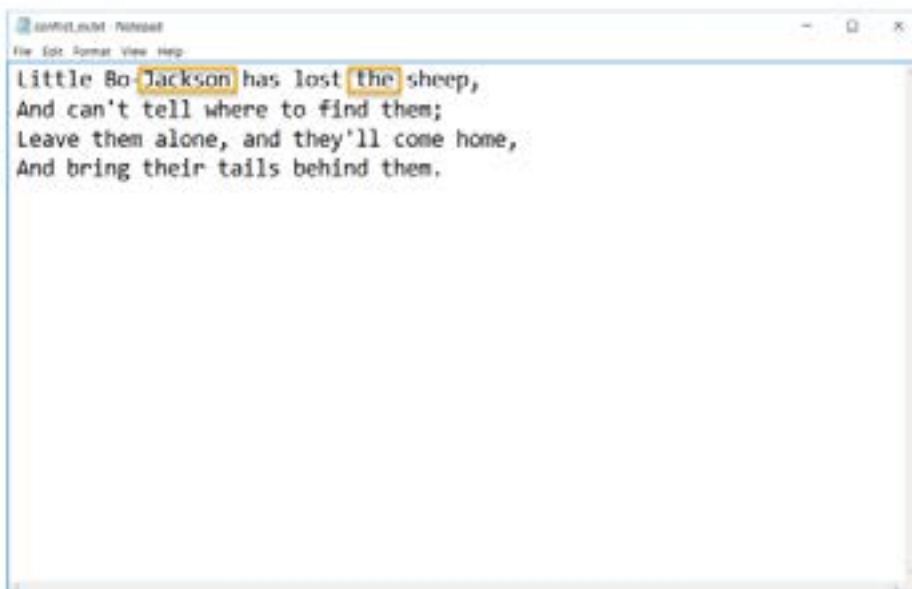
Master Branch



A screenshot of a Notepad window titled "conflict_example - Notepad". The text inside the window is:

```
Big Bo-Peep has lost his sheep,  
And can't tell where to find them;  
Leave them alone, and they'll come home,  
And bring their tails behind them.
```

Resolution_Example Branch

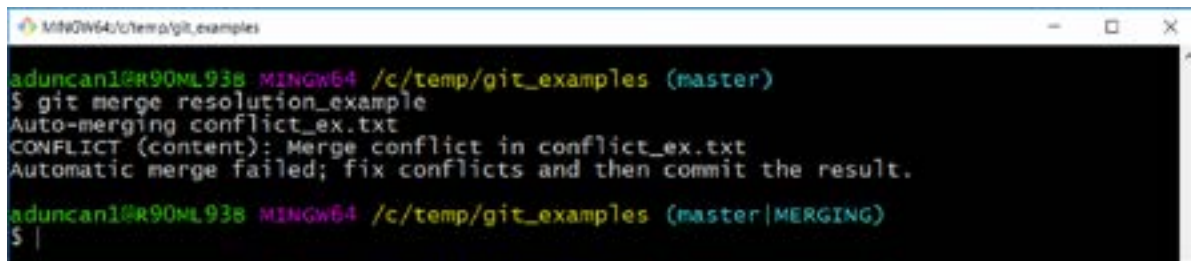


A screenshot of a Notepad window titled "conflict_example - Notepad". The text inside the window is:

```
Little Bo-Jackson has lost the sheep,  
And can't tell where to find them;  
Leave them alone, and they'll come home,  
And bring their tails behind them.
```

IDENTIFY THE CONFLICT

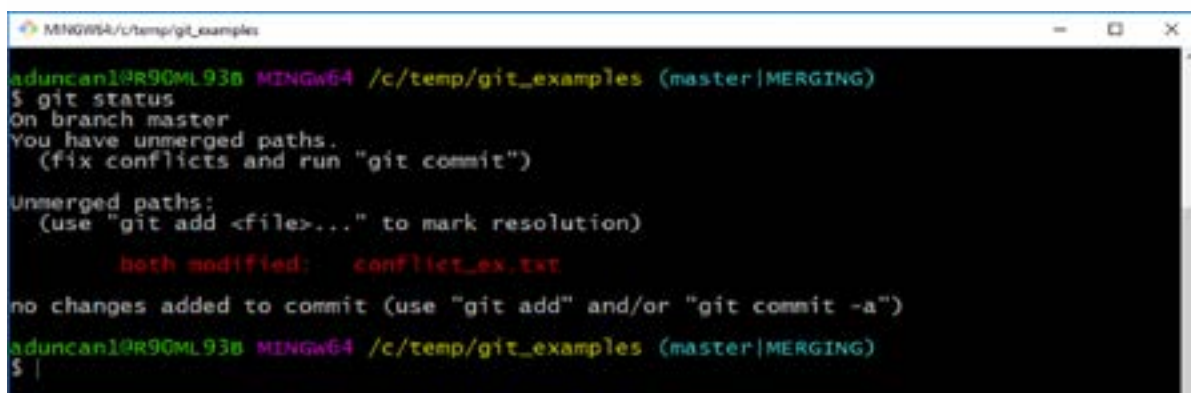
If there is a merge conflict Git displays a message when trying to merge and gives suggestions for what to do to fix the conflict.



```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$ git merge resolution_example
Auto-merging conflict_ex.txt
CONFLICT (content): Merge conflict in conflict_ex.txt
Automatic merge failed; fix conflicts and then commit the result.

aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master|MERGING)
$ |
```

Running the git status command tells you that you have unmerged paths.



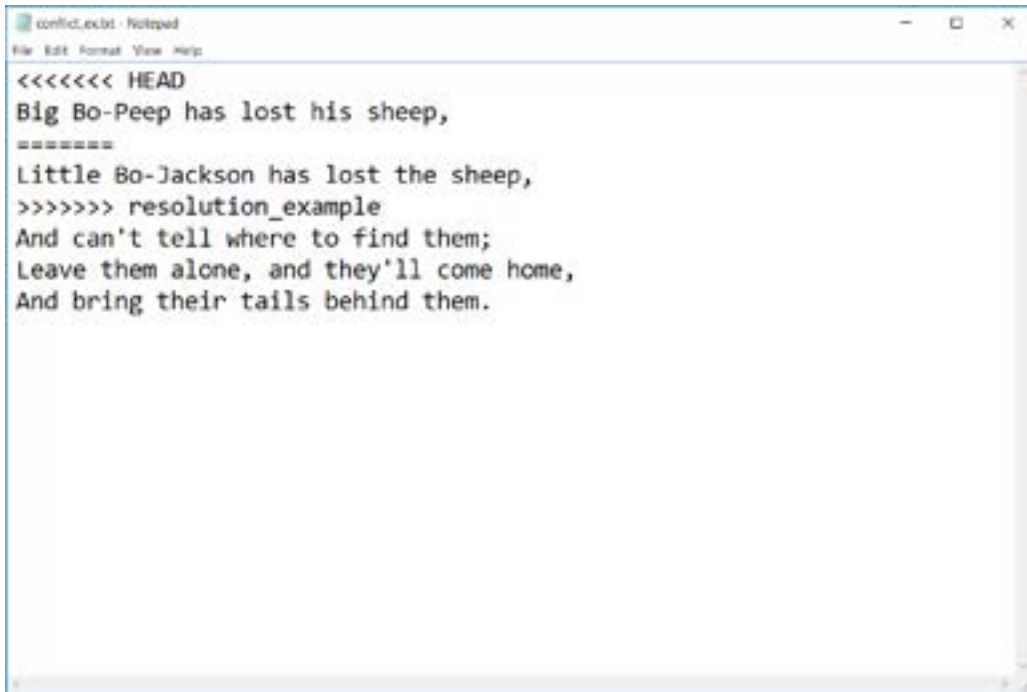
```
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   conflict_ex.txt

no changes added to commit (use "git add" and/or "git commit -a")
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master|MERGING)
$ |
```

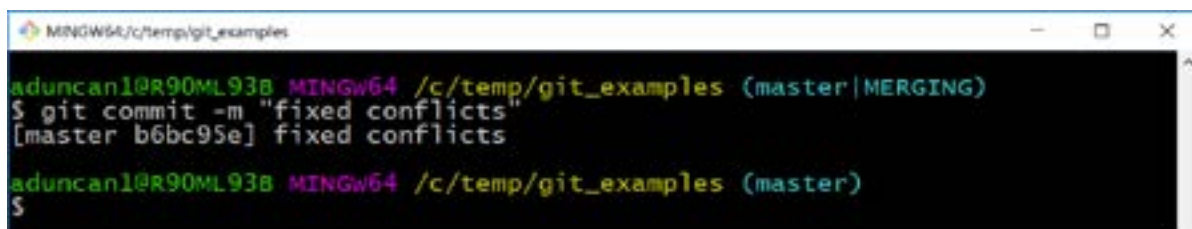
If you go back to the code, Git places markers inside your file to help you identify what caused the conflict.



```
conflict.txt - Notepad
File Edit Format View Help
<<<<<< HEAD
Big Bo-Peep has lost his sheep,
=====
Little Bo-Jackson has lost the sheep,
>>>>>> resolution_example
And can't tell where to find them;
Leave them alone, and they'll come home,
And bring their tails behind them.
```

RESOLVE THE CONFLICT

After identifying the conflict, fix the code or file and run a git add to add the updated information. Then run the git commit command to commit the changes.



```
MINGW64/c/temp/git_examples
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master|MERGING)
$ git commit -m "fixed conflicts"
[master b6bc95e] fixed conflicts
aduncan1@R90ML93B MINGW64 /c/temp/git_examples (master)
$
```

Notice that after the commit, the merge completes, and a git status command would show that there is nothing to commit and the branch is clean. It also no longer displays Merging next to the branch name (master) indicating that the merge is complete.

REVIEW

How does Git ensure file integrity?

What command stages our work?

What is the other name for the staging area?

What happens to our work when we commit a file?

Why is it important to have descriptive commit messages?

What purpose does the head pointer serve during a merge conflict?

"Share your knowledge. It is a way to
achieve immortality."
Dalai Lama XIV





SECTION 05

GITHUB OVERVIEW

WHY USE GITHUB

Han and Luke are our example software developers for this part of the course. Han is familiar with GitHub and Luke is not.

Because Luke does all his work locally he is:

- Running the risk of losing all his work if his computer malfunctions
- Unable to collaborate with other software developers

ABOUT GITHUB

GitHub is a repository hosting site that stores Git repositories on the web. Repo hosting sites allow us to restore our local code in case our computer is damaged. They also encourage collaboration, which helps create better code.

Quicken Loans uses a version of GitHub called GitHub Enterprise. The way the site looks and behaves is essentially the same. The main difference is that we are storing our code on our own servers, not the web.

LAB ACTIVITIES

CREATE A GITHUB ACCOUNT

- 1) Open Google Chrome
- 2) Navigate to <http://github.com>



- 3) Create a username

- 4) Enter email address
- 5) Create a password
- 6) REMEMBER YOUR USERNAME AND PASSWORD. YOU WILL NEED IT FOR LABS.
- 7) Click **Sign up for GitHub**
- 8) Click **Continue**



YOU ARE SIGNING UP FOR A PUBLIC ACCOUNT THAT EVERYONE HAS ACCESS TO. DO NOT PLACE ANY PROPRIETARY INFORMATION IN YOUR PUBLIC REPOSITORIES.

- 9) Click **Skip this step**
- 10) YOU MIGHT NEED TO SCROLL DOWN

Completed Set up a personal account

Step 2: Choose your plan

Step 3: Tailor your experience

How would you describe your level of programming experience?

☐ Totally new to programming ☐ Somewhat experienced ☐ Very experienced

What do you plan to use GitHub for? (check all that apply)

☐ Development ☐ Design ☐ School projects
☐ Research ☐ Project Management ☐ Other (please specify)

Which is closest to how you would describe yourself?

☐ I'm a student ☐ I'm a professional ☐ I'm a hobbyist
☐ Other (please specify)

What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

[skip this step](#)

- 11) Navigate to Your Outlook email
- 12) Open the email from GitHub
- 13) Click **Verify email address**



START A PROJECT

- 1) In GitHub, click Start a project



YOU WILL BE BROUGHT TO THE PAGE SHOWN BELOW. STOP HERE.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner / Repository name

Great repository names are short and memorable. Need inspiration? How about `verbose-journey`.

Description (optional)

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: Add a license:



STOP AT THIS POINT

"You don't have to be great to start,
but you have to start to be great. "
Zig Ziglar



SECTION 06

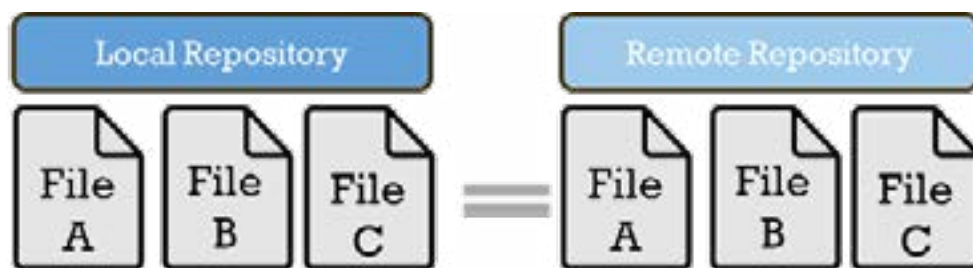
UNDERSTANDING REMOTES

Before you can collaborate, you need get your local code on GitHub. To do this, you use remote repositories (aka Remotes). A remote is a version or copy of your local repository(s) hosted on GitHub or in our case GitHub Enterprise.

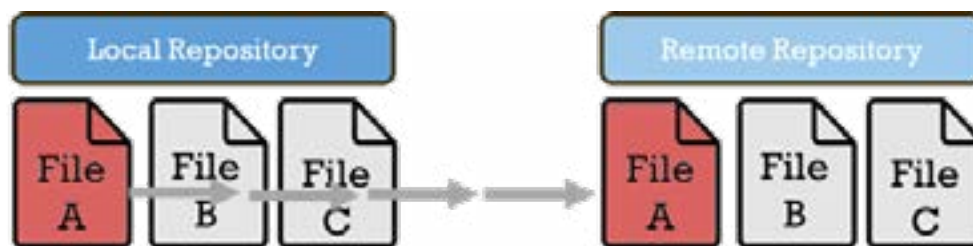
There are a few key terms for understanding remotes:

CLONE: Copies an existing remote and places that copy on your local machine. This operation does the following actions.

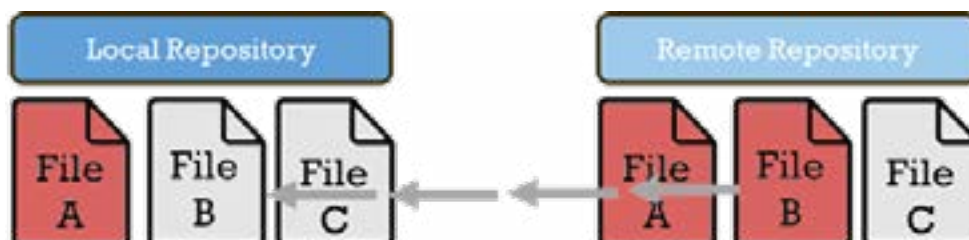
- Creates a new directory on your computer
- Initializes that directory as a Git repository
- Adds a remote named origin, along with the remote's URL.
- Downloads all files to local repository



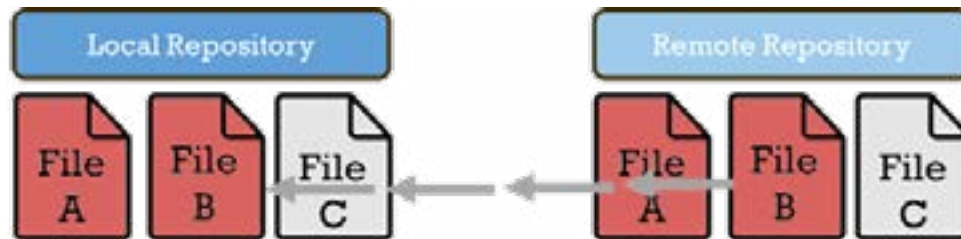
PUSH: Takes local changes and places them in your remote repository.



FETCH: Retrieves updates from a remote but does not merge them into your local branch.

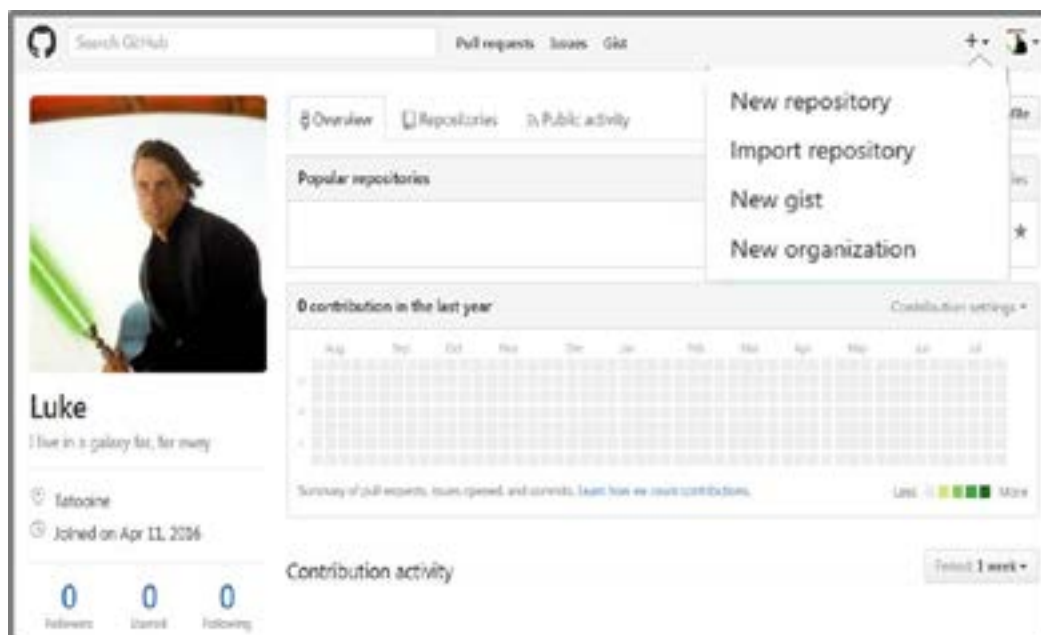


PULL: Fetches updates from a remote and automatically merges them into your local branch.

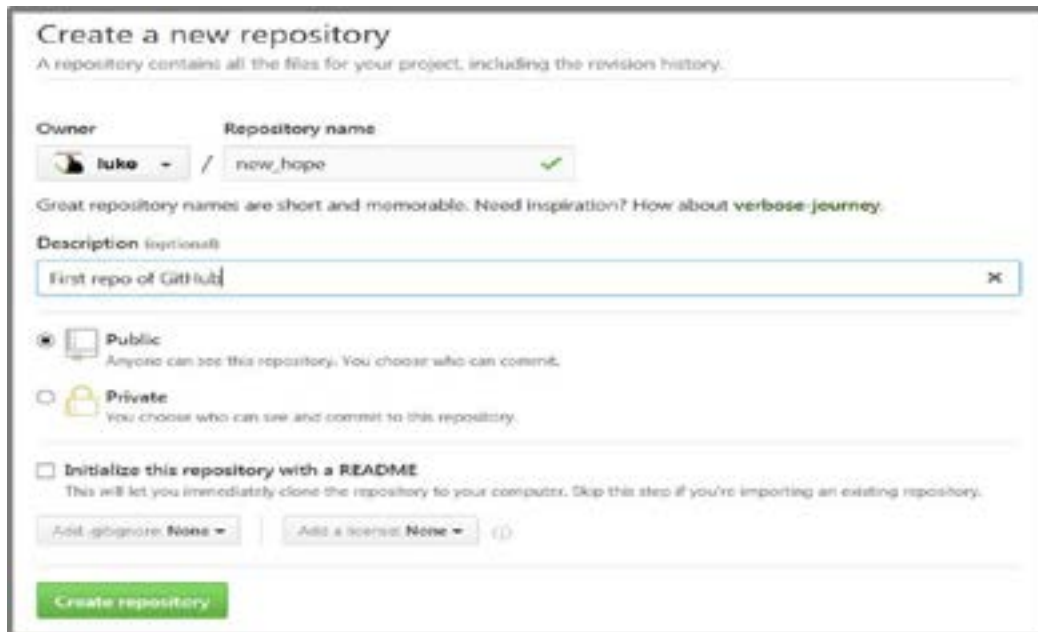


LUKE CREATES AND MANAGES HIS REMOTE

After creating a GitHub account, select the plus sign and then select New repository.

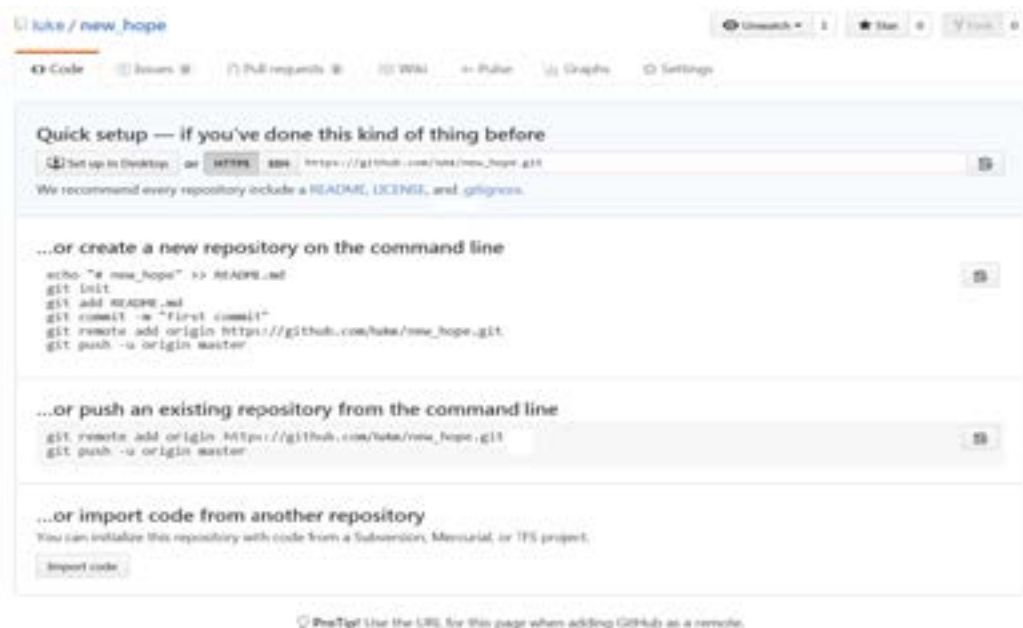


Name the repository, enter a description, and then select Create Repository.



The screenshot shows the 'Create a new repository' form on GitHub. At the top, it says 'Create a new repository' and 'A repository contains all the files for your project, including the revision history.' Below this, there are two main sections: 'Owner' and 'Repository name'. The 'Owner' is set to 'luke' and the 'Repository name' is 'new_hope', which has a green checkmark next to it. Below these, there is a 'Description (optional)' text area containing 'First repo of GitHub'. Under the description, there are two radio buttons: 'Public' (selected) and 'Private'. Below these, there is a checkbox for 'Initialize this repository with a README'. At the bottom, there are two buttons: 'Add .gitignore: None' and 'Add a license: None'. A green 'Create repository' button is at the bottom right.

The setup page appears.



The screenshot shows the repository setup page on GitHub for the repository 'luke / new_hope'. The page has a navigation bar with links for 'Code', 'Issues', 'Pull requests', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below the navigation bar, there are three main sections: 'Quick setup — if you've done this kind of thing before', '...or create a new repository on the command line', and '...or push an existing repository from the command line'. The 'Quick setup' section includes a 'Set up in Desktop' button and a text area with the URL 'https://github.com/luke/new_hope.git'. The '...or create a new repository on the command line' section includes a code block with the following commands:

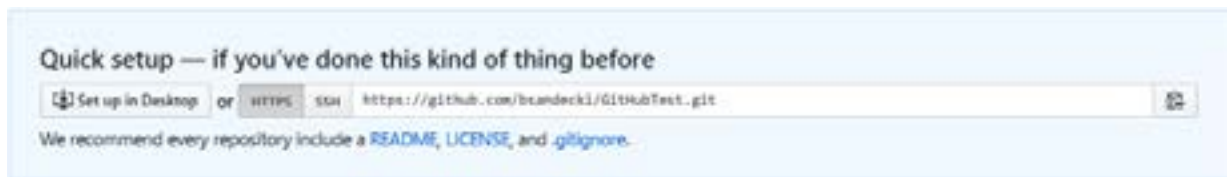
```
echo "# new_hope" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/luke/new_hope.git
git push -u origin master
```

 The '...or push an existing repository from the command line' section includes a code block with the following commands:

```
git remote add origin https://github.com/luke/new_hope.git
git push -u origin master
```

 At the bottom, there is a section for '...or import code from another repository' with an 'Import code' button. A footer note says 'ProTip! Use the URL for this page when adding GitHub as a remote.'

SETUP PAGE BREAKDOWN



The Quick setup option allows you to setup the repo using GitHub Desktop, which is a GUI client that you must install, or you can copy the URL of the repo if you are using the command line version of Git.



This option provides you the commands to create a README file, create a new repo, stage the README file, commit the file, add your remote, and push the README to your remote.



If you already have an existing local repository, this option provides you with the commands to add a remote and push the repo to your remote.



The last option allows you to import code from another type of source control software.

LAB ACTIVITIES

CREATE A REMOTE

- 1) In the Repository name field, type *first initial last initial_project_lab* (example kb_project_lab).

The screenshot shows the 'Create a new repository' form on GitHub. It includes fields for 'Owner' (a dropdown menu), 'Repository name' (with a red arrow and '1)' pointing to it), and 'Description (optional)' (with a red arrow and '2)' pointing to it). There are radio buttons for 'Public' (selected) and 'Private'. A checkbox for 'Initialize this repository with a README' is unchecked. At the bottom, there are dropdowns for 'Add gitignore: None' and 'Add a license: None', and a green 'Create repository' button (with a red arrow and '3)' pointing to it).

- 2) In the **Description** field, type important about the repository (example KB First GitHub repository)



DO NOT SELECT INITIALIZE THIS REPOSITORY WITH A README

- 3) Click **Create repository**
- 4) Under **Quick setup**, click the **clipboard** icon to copy the remote repository URL

The screenshot shows the 'Quick setup' section with the heading 'Quick setup — if you've done this kind of thing before'. It has buttons for 'Set up in Desktop', 'HTTPS', and 'SSH'. The 'HTTPS' button is selected, showing the URL 'https://github.com/luke/new_hope.git'. A red arrow points to a clipboard icon at the end of the URL. Below this, it says 'We recommend every repository include a README, LICENSE, and .gitignore.'

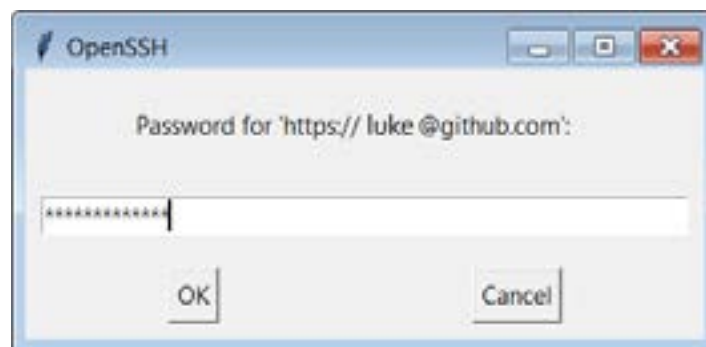
- 5) In Git BASH type: `git remote add origin` and paste the remote repository URL
- 6) Press: **Enter**
- 7) Type: `git remote -v`
- 8) Press: **Enter**



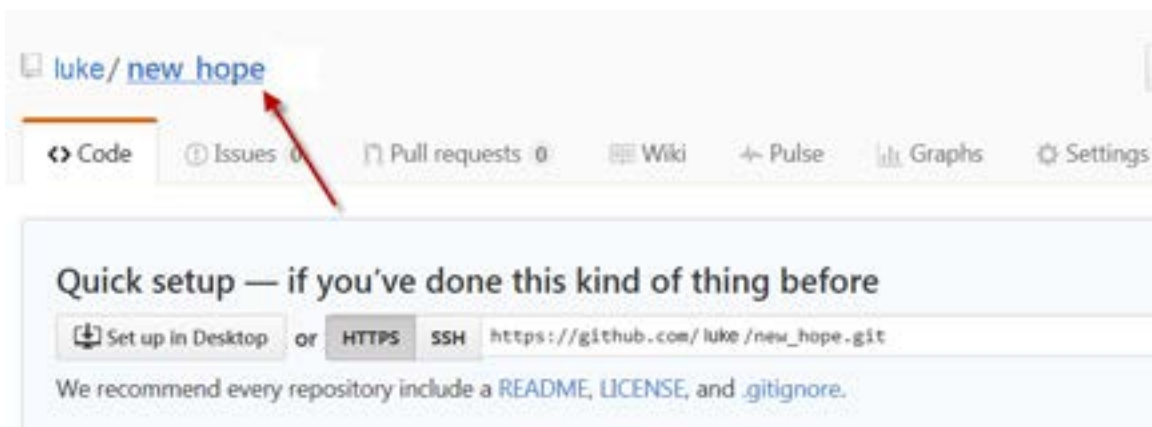
THIS COMMAND ALLOWS YOU TO VERIFY THE REMOTE REPOSITORY URL

MANAGE REMOTE

- 1) In Git BASH type, `git push origin master`
- 2) Press: **Enter**
- 3) Type your GitHub username and password (if prompted)



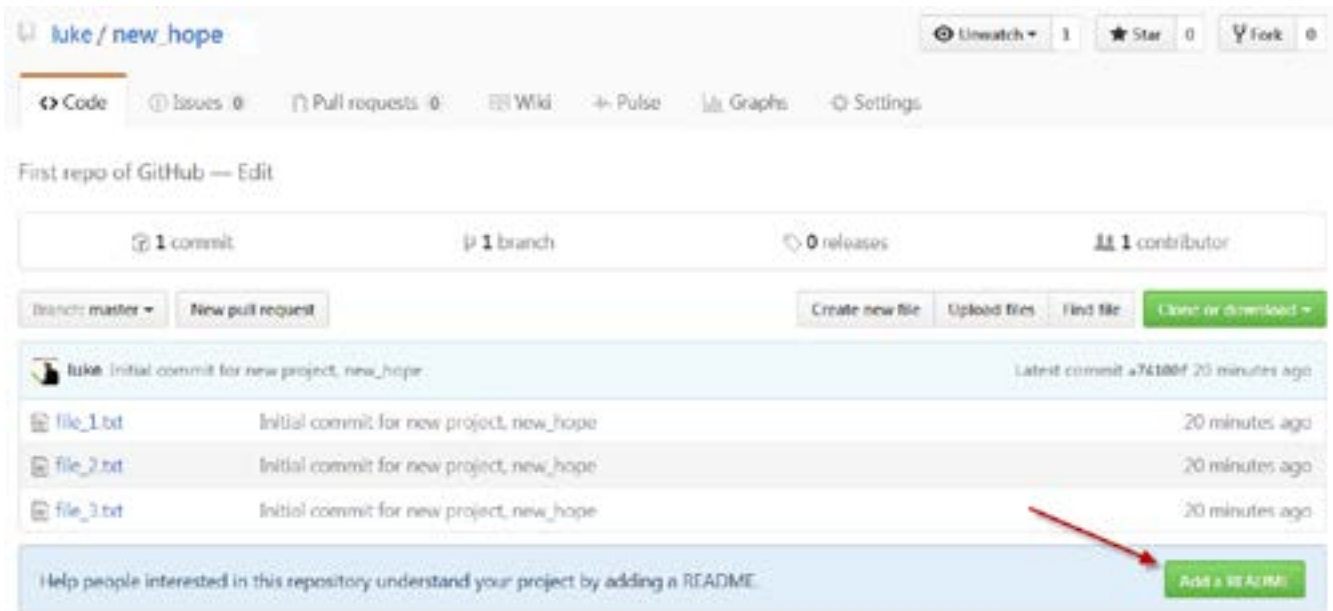
- 4) On GitHub, click on your remote repository name.



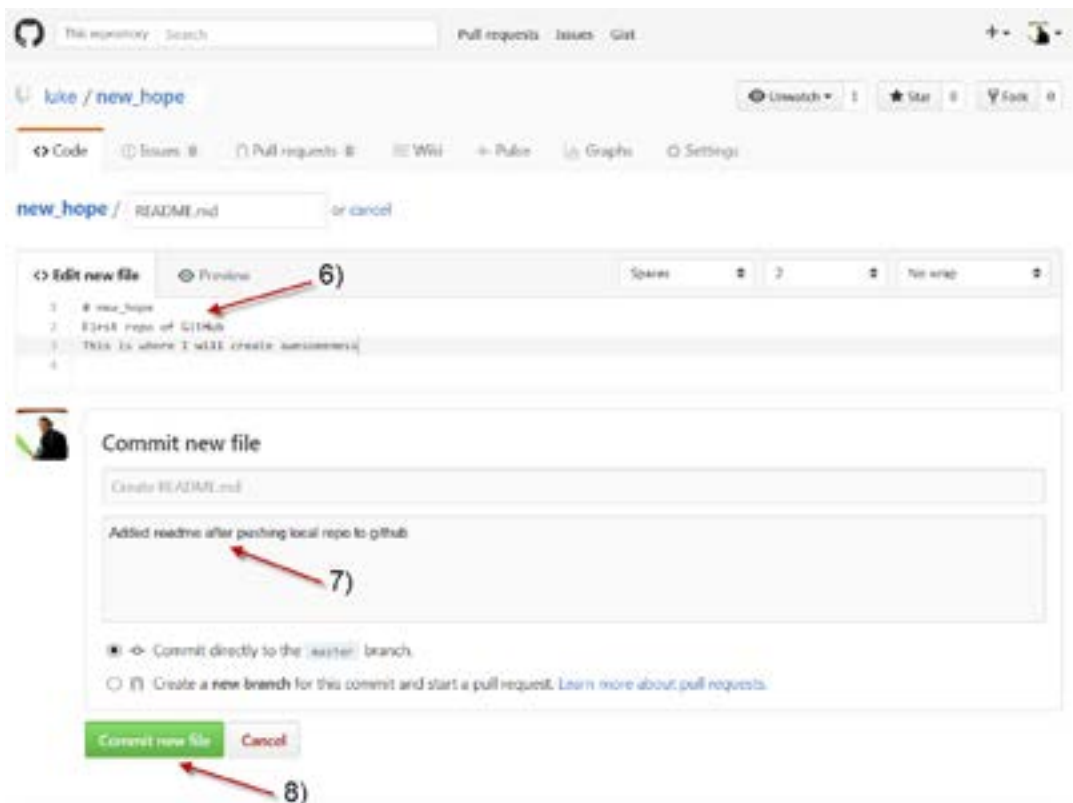
SECTION 06 UNDERSTANDING REMOTES

58 of 84

5) Click **Add a README**



6) Enter a description for your repo.



- 7) Under **Commit new file**, add a commit message.
- 8) Click **Commit new file**
- 9) In Git BASH type: git fetch origin
- 10) Press: **Enter**
- 11) Type: git diff master origin/master
- 12) Press: **Enter**
- 13) Type: git merge origin/master
- 14) Press: **Enter**



STOP AT THIS POINT

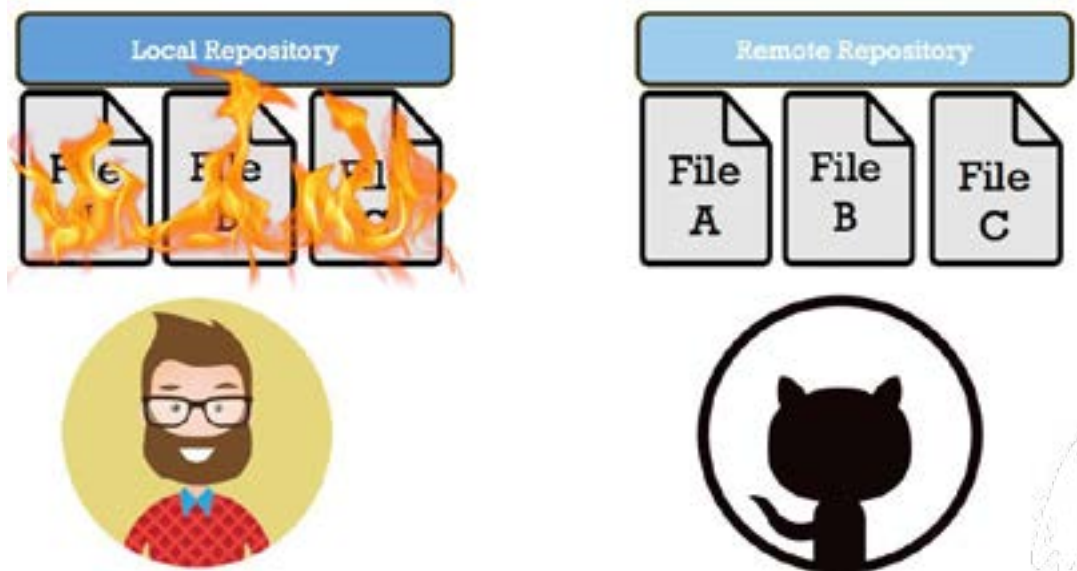
"Learning and innovation go hand in hand. The arrogance of success is to think that what you did yesterday will be sufficient for tomorrow."
William Pollard



SECTION 07

GIT HAPPENS

Using a repo hosting site allows us to restore our files when our computer is damaged or destroyed.



What operation does Luke need to perform to get his code from his remote onto his local machine?

What are the specific actions that this command does?

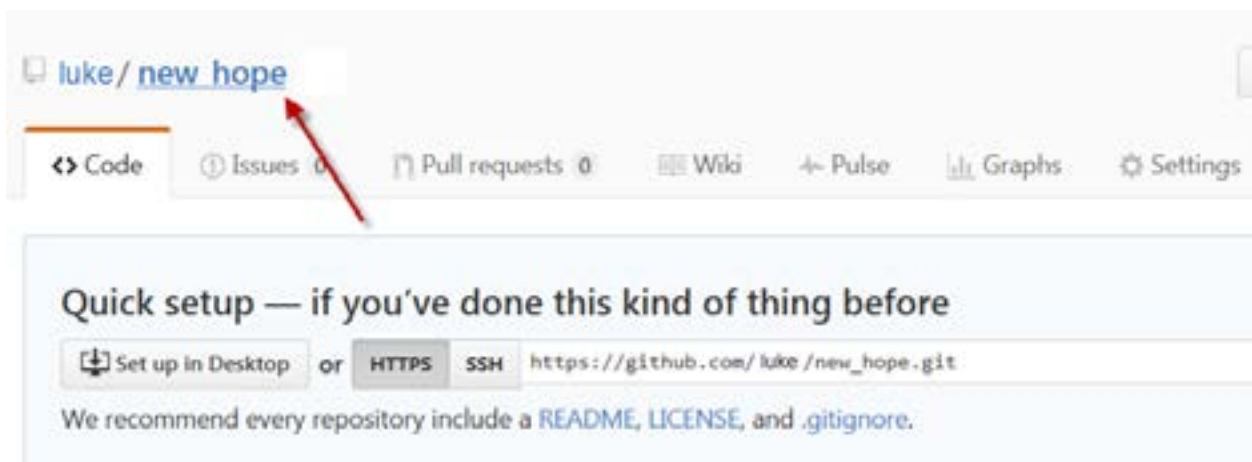
What command can Luke run to see the files in his repo?

What operation does he have to do to get his remote up-to-date with his local repo?

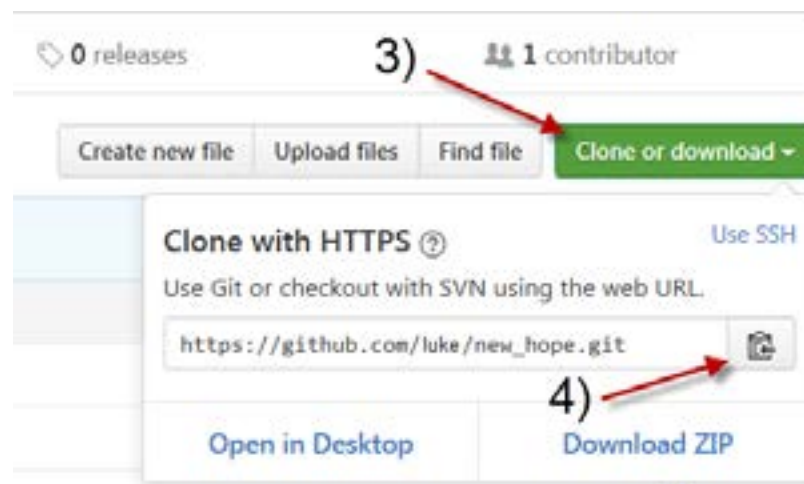
LAB ACTIVITIES

RETRIEVE FILES

- 1) In File Explorer, delete ***yourinitials_git_lab*** folder.
- 2) On GitHub, click on your remote repository.



- 3) Click the **Clone or download** drop-down.
- 4) Click on the **clipboard** icon to copy the remote's URL



- 5) In Git BASH type: `cd`
- 6) Press: **Enter**
- 7) Type `cd /c/temp`
- 8) Press: **Enter**
- 9) Type `git clone` and paste the remote repository URL
- 10) Press: **Enter**

COMMIT LOCALLY

- 1) In File Explorer, open ***your initials_project_lab*** (example kb_project_lab).



YOUR FOLDER NAME IS DIFFERENT BECAUSE THE CLONE OPERATION WILL CREATE A DIRECTORY AND NAME IT THE SAME NAME AS THE REMOTE THAT IT WAS CLONED FROM

- 2) Open file_2.txt
- 3) Inside file_2.txt, insert 1 line of text.
- 4) Save and close file.
- 5) In Git BASH type: `cd your initials_project_lab`
- 6) Press: **Enter**
- 7) Type: `git add file_2.txt`
- 8) Press: **Enter**
- 9) Type: `git status`
- 10) Press: **Enter**
- 11) Type: `git commit -m "type a comment between the quotation marks"`
- 12) Press: **Enter**
- 13) Type: `git status`
- 14) Press: **Enter**

What was the output of this command?

PUSH CHANGES

- 1) Type: `git push origin master`
- 2) Input your GitHub username and password (if prompted)
- 3) Press: **Enter**
- 4) Type: `git status`
- 5) Press: **Enter**

What was the output of this command?



STOP AT THIS POINT

NOTES

[illegible]

"Listen with curiosity. Speak with
honesty. Act with integrity."
Roy Bennett





SECTION 08

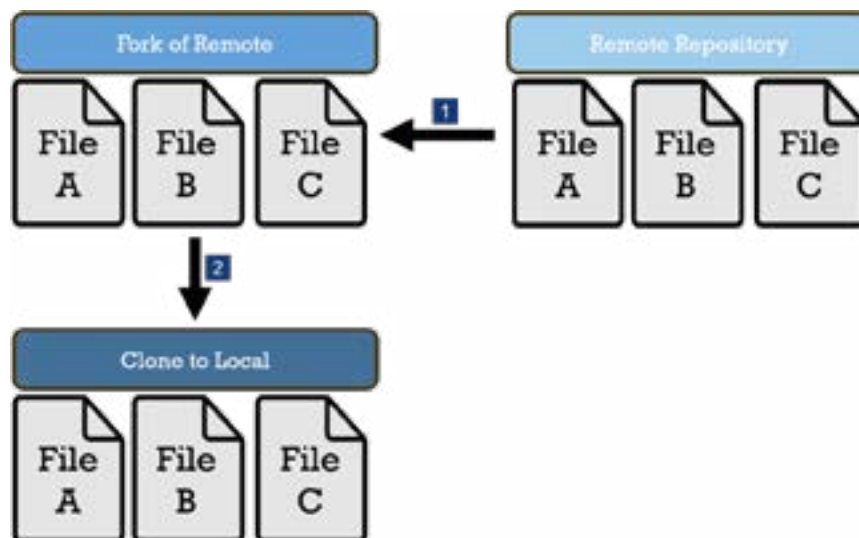
COLLABORATING WITH REMOTES

There are a few things to know before collaborating with others on GitHub.

- **FORK:** Your own remote copy of someone else's remote repository.
- **UPSTREAM:** Upstream refers to the remote that holds the official copy of code that everyone will fork from.
- **PULL REQUEST:** You submit a pull request to the project owner when you want to propose changes.
- **ORIGIN:** Do not name every remote "origin". The remote that you forked from upstream is your origin. If you add other remotes, name them something other than origin. Developer's names are an option.

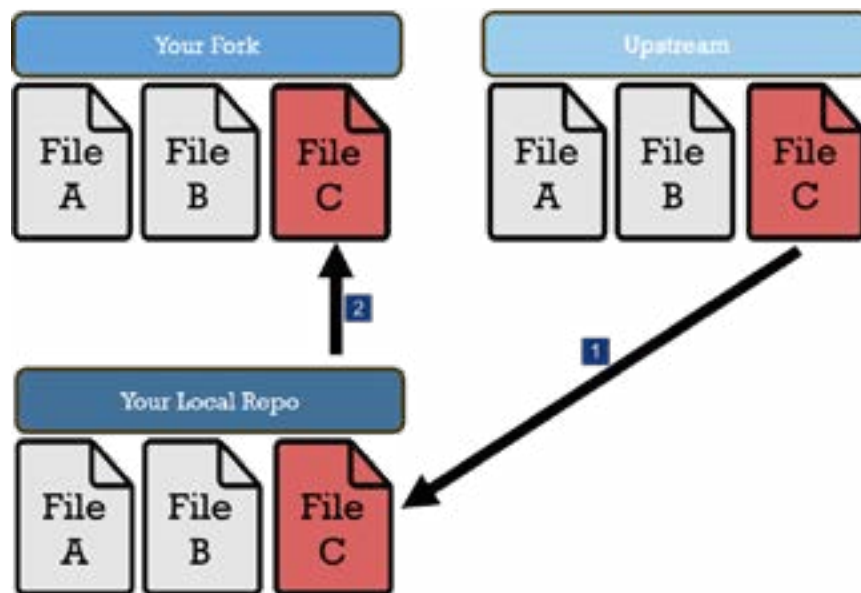
FORK

If you see a project on GitHub that you want to contribute to, you can fork that remote repository. Forking allows you to freely experiment with changes to code without affecting the original project. Your fork exists on GitHub and you would have to clone it in order to have a local copy on your computer.



UPSTREAM

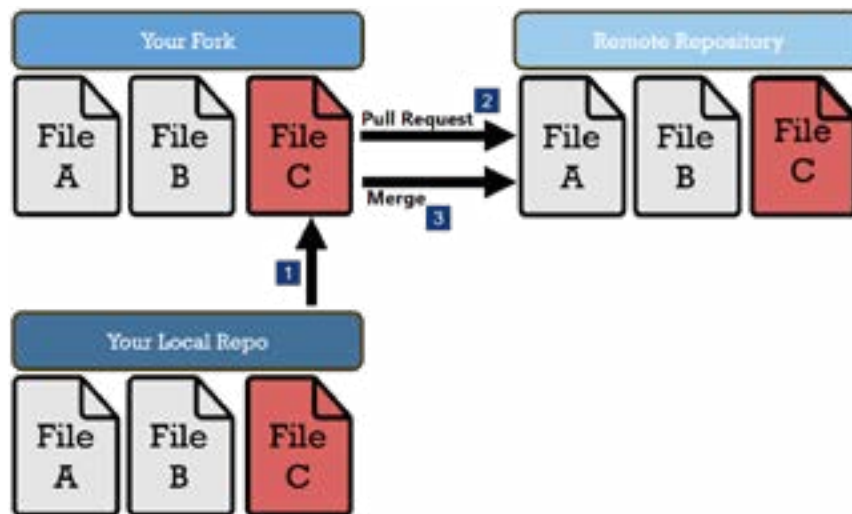
It is important to ensure that you are staying up to date with your upstream remote. If someone else on your team has made a change to the official copy, you will need to fetch or pull from the upstream remote to keep your local repo up-to-date. You would then need to push the change up to your fork, to keep that up-to-date.



PULL REQUEST

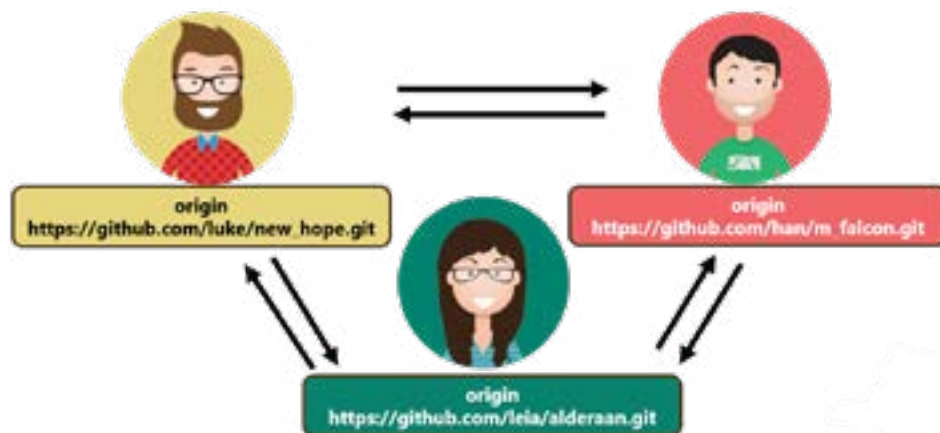
When collaborating on projects, there will come a time when you want to share your changes with the team. This is done through pull requests. A pull request notifies the project owner that changes have been proposed.

After you complete your change and have tested it, you push your code to your fork and then you open a pull request. When creating a pull request, it is important to include comments on what has changed and why. If the project owner accepts the changes, they will merge the changes into the upstream remote.



ORIGIN

The “origin” is an alias for your remote’s URL. It’s important to name the remote something that is easy to identify. Suggestions are to use the developers name. Remember to use something unique and to NOT actually use the word “origin”.



```
luke@vm_WIN7 MINGW64 ~/Desktop/new_hope (master)
$ git remote -v
origin https://github.com/luke/new_hope.git (fetch)
origin https://github.com/luke/new_hope.git (push)
han https://github.com/han/m_falcon.git (fetch)
han https://github.com/han/m_falcon.git (push)
leia https://github.com/leia/alderaan.git (fetch)
leia https://github.com/leia/alderaan.git(push)

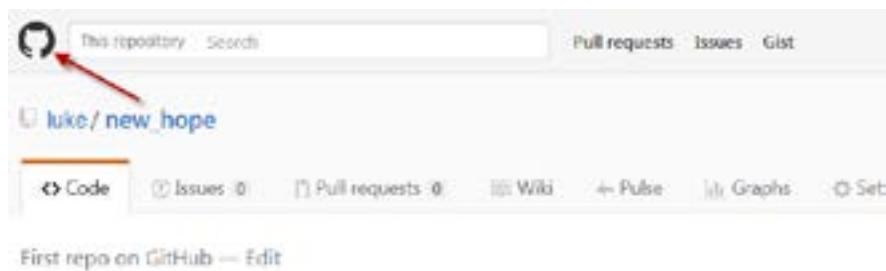
luke@vm_WIN7 MINGW64 ~/Desktop/new_hope (master)
$
```


LAB ACTIVITIES

Follow the steps that Han had to perform to fork Luke's remote, clone it, make changes and push to your remote repo, and open a pull request.

FORK

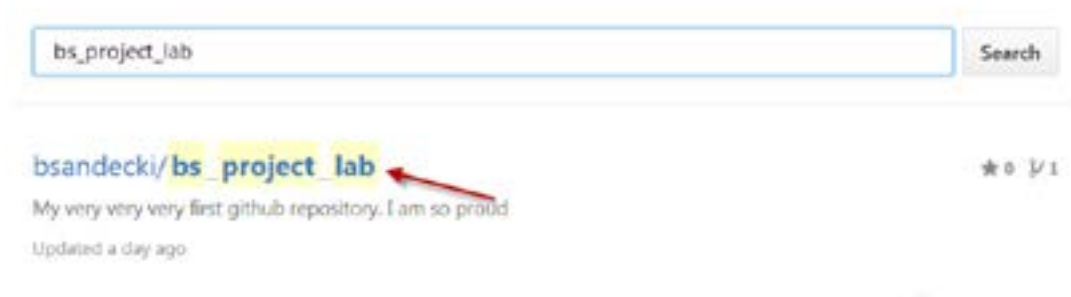
- 1) From your repo page, click **Octocat**.



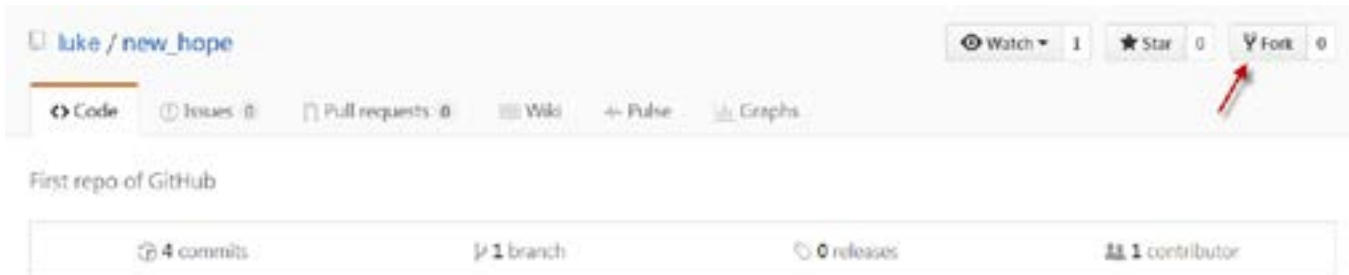
- 2) In the **Search GitHub**, field, enter the name of your buddy's repo.



- 3) Click your buddy's repo.

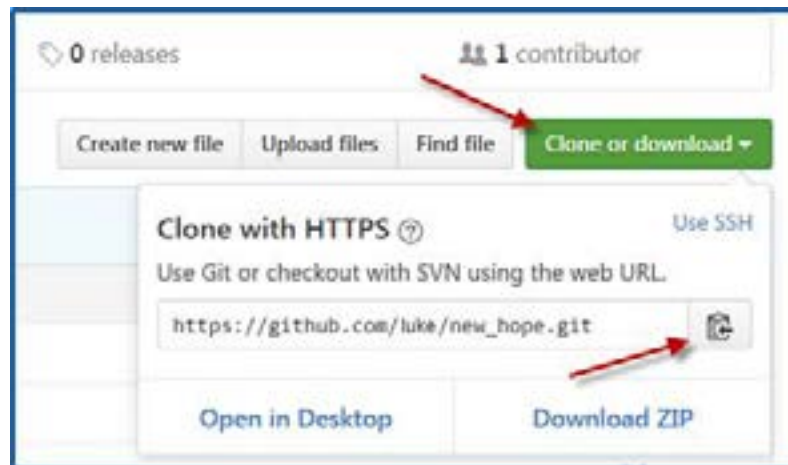


- 4) Click the **Fork** icon.



CLONE

- 1) Click the **Clone or download** drop-down.



IF YOU ARE UNABLE TO CLICK ON THE DROP-DOWN, REFRESH THE PAGE BY PRESSING F5.

- 2) Click the **clipboard** icon to copy the fork's URL
- 3) In GitBASH type: `cd`
- 4) Press: **Enter**
- 5) Type: `cd /c/temp`
- 6) Press: **Enter**
- 7) Type: `git clone` and then paste the fork URL
- 8) Press: **Enter**
- 9) Type: `cd buddies_initials_project_lab` (example `bs_project_lab`)



ENSURE YOU ARE IN THE CORRECT FOLDER. THIS IS YOUR FORK'S FOLDER.

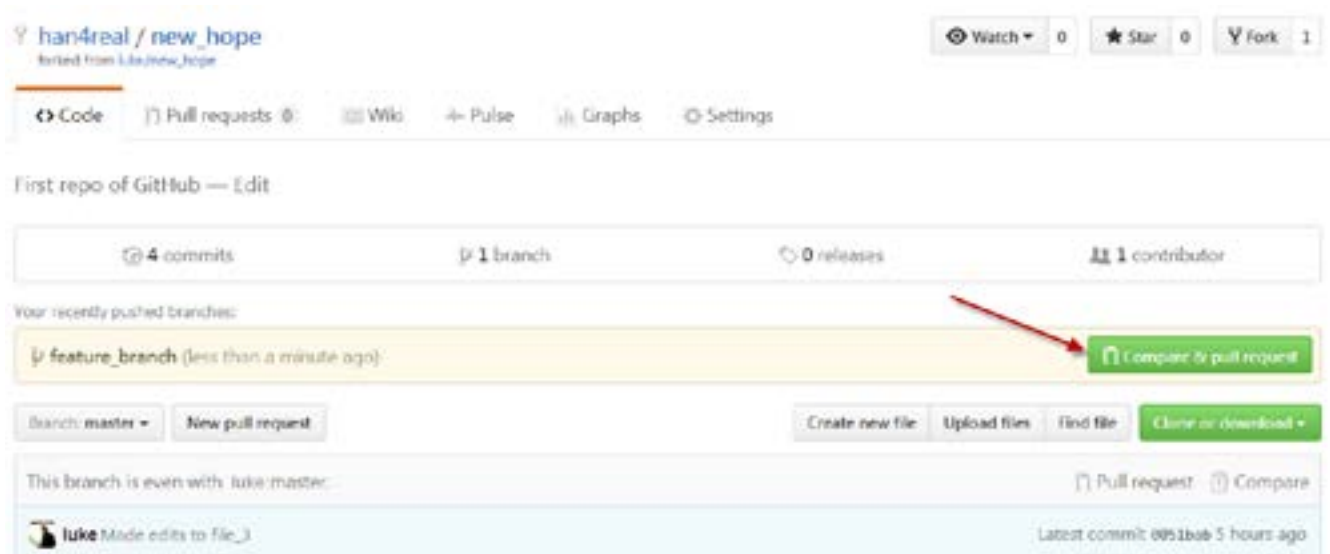
- 10) Press: **Enter**

MAKE CHANGES

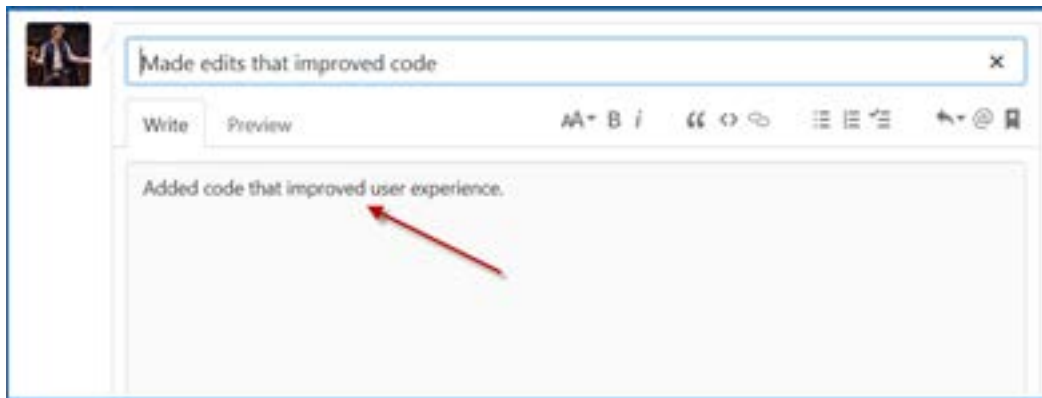
- 1) Type: `git checkout -b feature_branch`
- 2) Press: **Enter**
- 3) Inside **buddy's initials_project_lab**, open **file_1.txt**
- 4) Inside **file_1.txt**, insert 3 lines of text.
- 5) Save and close file
- 6) Type: `git add file_1.txt`
- 7) Press: **Enter**
- 8) Type: `git commit -m "Your comment inside the quotes"`
- 9) Press: **Enter**
- 10) Type: `git status`
- 11) Press: **Enter**
- 12) Type: `git push origin feature_branch`
- 13) Press: **Enter**
- 14) Input your GitHub username and password (if prompted)

CREATE PULL REQUEST

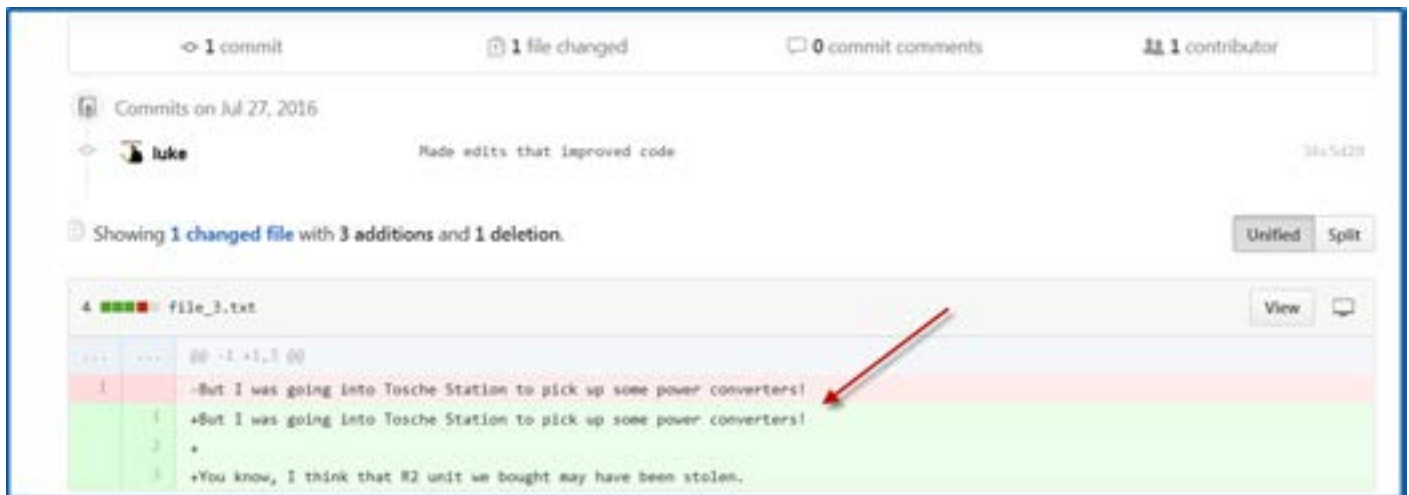
- 1) In GitHub, click **Compare & pull request**



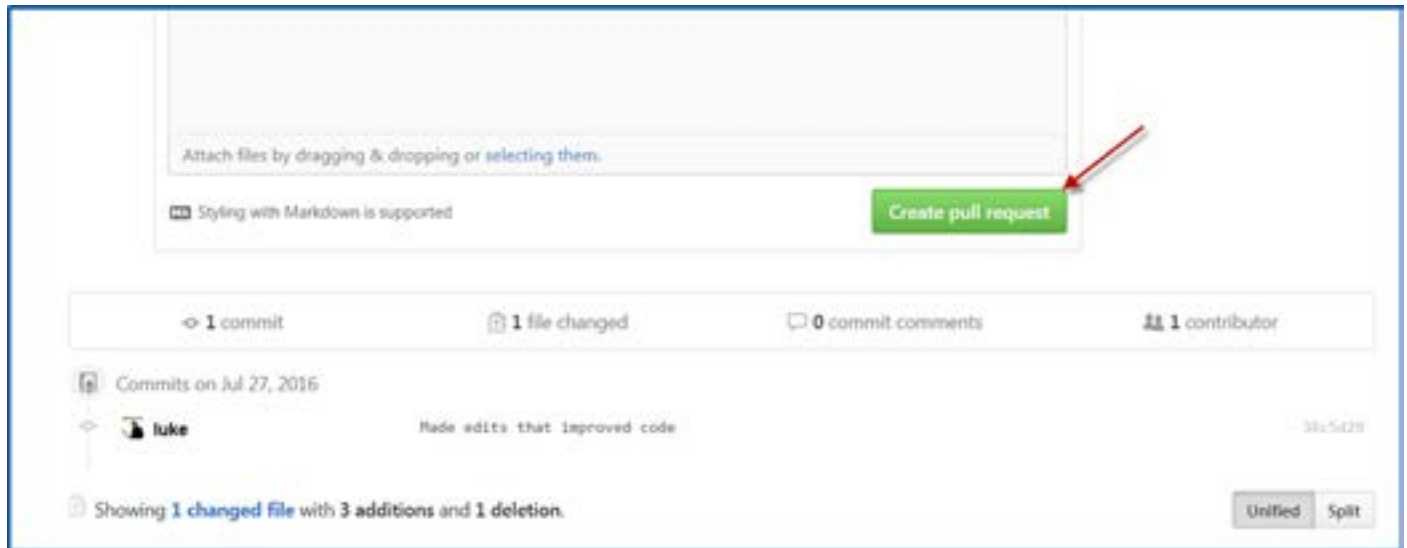
- 2) Add a detailed comment.



- 3) Compare changes.



- 4) Click **Create pull request**

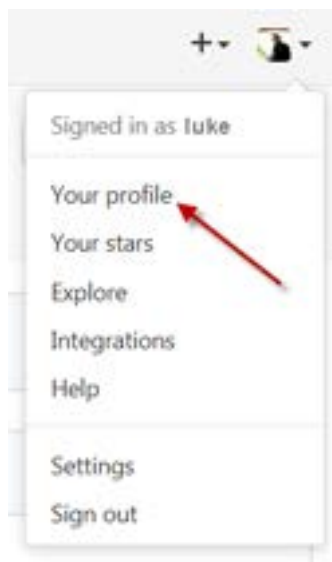


STOP AT THIS POINT

The previous lab was based on the steps that Han had to do. You will now follow the steps that Luke had to do to review the changes, merge the changes, and update his local repository.

REVIEW THE CHANGES

- 1) In GitHub, click **Your profile**.



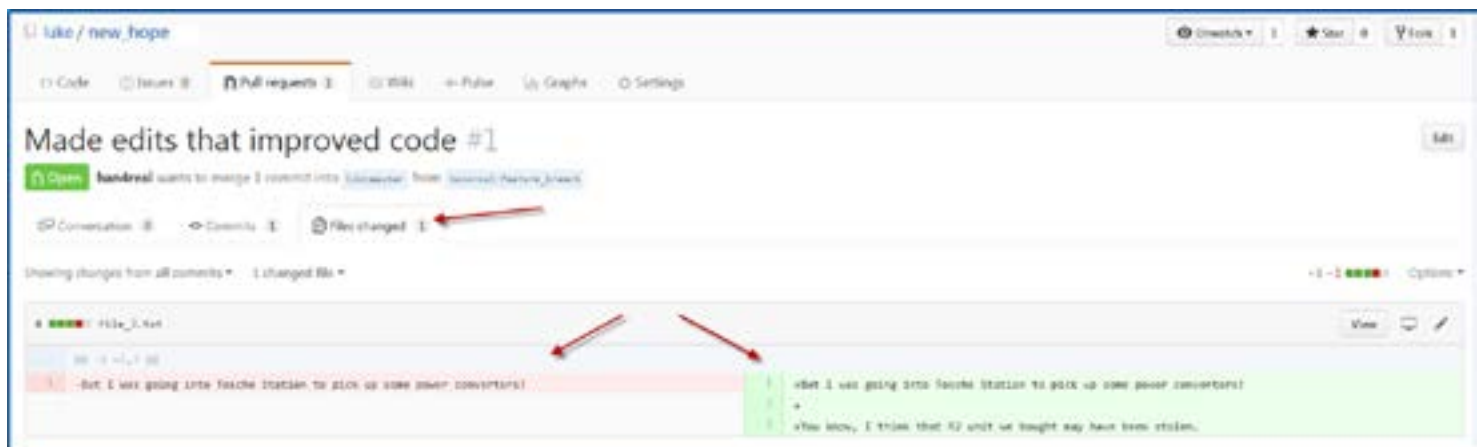
- 2) Click your repository



- 3) Click the **Pull Requests** tab



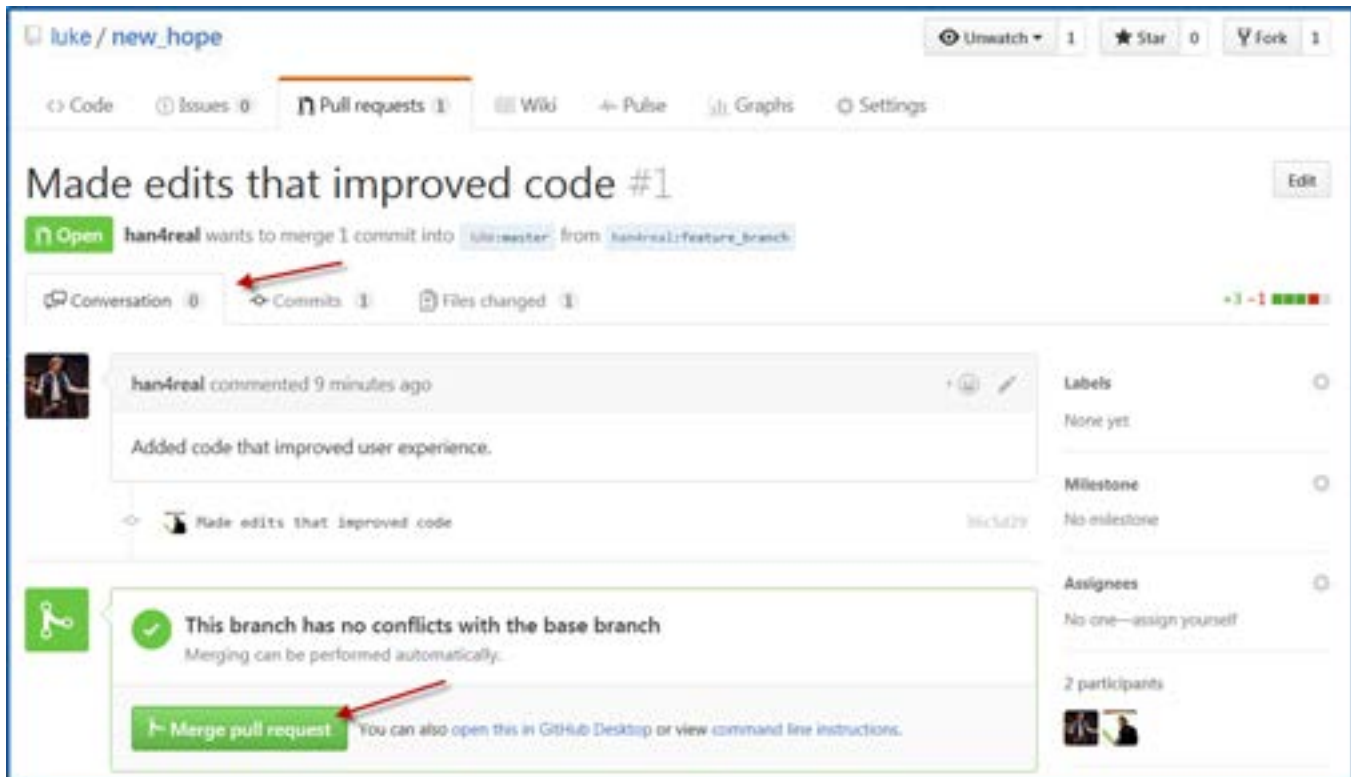
- 4) Click the **Files Changed** tab.



- 6) Compare changes.

MERGE PULL REQUEST

- 1) Click the **Conversation** tab.



- 2) Click **Merge pull request**.
- 3) Click **Confirm merge**.

UPDATE LOCAL REPOSITORY

- 1) In GitBASH type: `cd . .`
- 2) Press: **Enter**
- 3) Type `cd your_initials_project_lab`
- 4) Press: **Enter**



YOU NEED TO BE IN YOUR PROJECT LAB FOLDER IN ORDER TO PULL THE UPDATES FROM THE REMOTE.

- 5) Type `git pull origin master`

6)



STOP AT THIS POINT

NOTES

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

"We Generate Fears While We Sit.
We Overcome Them By Action."
Dr. Henry Link



SECTION 09

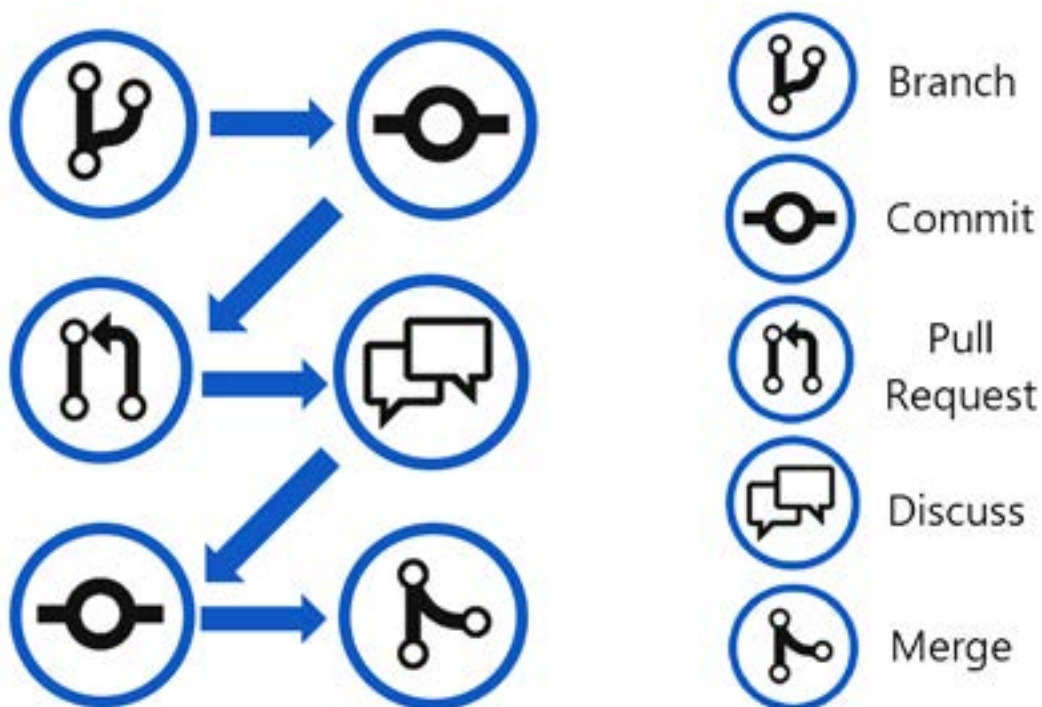
GIT FLOW AND YOU

There are many different workflows that developers use when using Git and that's also very true here at Quicken Loans. Workflows can vary from team to team and developer to developer, but a great starting point is the Git flow. It's a very simple workflow that is suitable for many different situations.

The Git Flow is based on one rule. **Anything in the master branch is deployable.** This keeps the master branch stable and makes it safe for anyone to fork off of at any time. This also reduces issues in the long run.

GIT FLOW STEPS

- 1) Create a feature branch
- 2) Commit locally and push regularly
- 3) Open a pull request
- 4) Discuss with your team
- 5) Make any necessary changes and commit
- 6) Project owner merges pull request



REVIEW

What command will show you if you have untracked, modified, or staged files?

What happens when you clone a repo?

What is the difference between a fetch and a pull?

What does the term upstream refer to?

When could you open a pull request?
