

09-Module05 Git Concepts and Basics Lab

Purpose:

The purpose of this lab is to give the student some experience with Git.

Pre-Lab Setup:

- The Student must have Git Bash (for Windows) or Git activated (for Mac).

Installing Git on your laptop:

If you have not already done so, it's time to install Git on your laptop now.

For Mac users, go to this URL:

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

About half-way down the page are the instructions for installing on the Mac.

For PC users, go to this URL:

<http://git-scm.com>

Follow the instructions to download git. Note: watch for references to "Git Bash" and be sure to include it in your download.

Mac Installations:

All Mac computers from the last few years or so will come with Git installed. So before you install Git on a Mac computer, it's a good idea to check to see if you already have it installed. To do so, simply open a terminal window, type in "**git**", and press **Enter**.

If it responds with a help message, then Git is already installed.

If it responds with some other message, then Git isn't installed and the Mac will ask if you want it to install git for you- let it do so.

Windows Installations:

Download the latest Git for Windows installer:

<https://git-for-windows.github.io/>

When you've successfully started the installer, you should see the Git Setup wizard screen. You can follow the “Next” and “Finish” prompts to complete the setup, but there is one part of the process we will need to look at closely- the options for configuring your installation. The following slides will show which ones we’ll use.

Configure Your Git Installation:

To **set your default identity** for all repositories on your workstation, enter the following commands:

```
git config --global user.name <your name> #sets your name
git config --global user.email <your email address> #sets your email address
```

Optional - For the sake of this course, we’ll be using the default editor, vi:

```
git config --global core.editor /path/to/editor #sets your editor of choice
```

Otherwise can be configured for Notepad++, Sublime and others.

To **set an identity for a specific repository**, enter the following commands (note the subtle difference: no “-global” option):

```
cd /path/to/repositorydir
git config user.name <your name> #sets your name
git config user.email <your email address> #sets your email address
```

These configurations allow git to identify you, the repositories you own, and the changes you make.

GitHub Lab

The purpose of this Lab is to give you firsthand experience with a tool like GitHub. You can set up an account for free.

Pre-requisite: You must have already installed Git on your laptop.

Instructions:

- Open command line on your laptop.
- Create a new folder (directory) to hold files related to the Class, with the recommended name of “devopsclass”
- Open this link: <https://github.com/eespinoza13/DevOpsTraining/>
- Find the file CreateGitHubAccount in that DevOpsTraining folder and download it. After it downloads, move it to the directory you just made.
 - If the CreateGitHubAccount.docx file has lost its extension and looks like an RTF file named “CreateGitHubAccount”, then rename it to be of type “docx”.

- Open the file named “CreateGitHubAccount.docx” and follow the instructions from there.

This lab should take about 10 minutes.

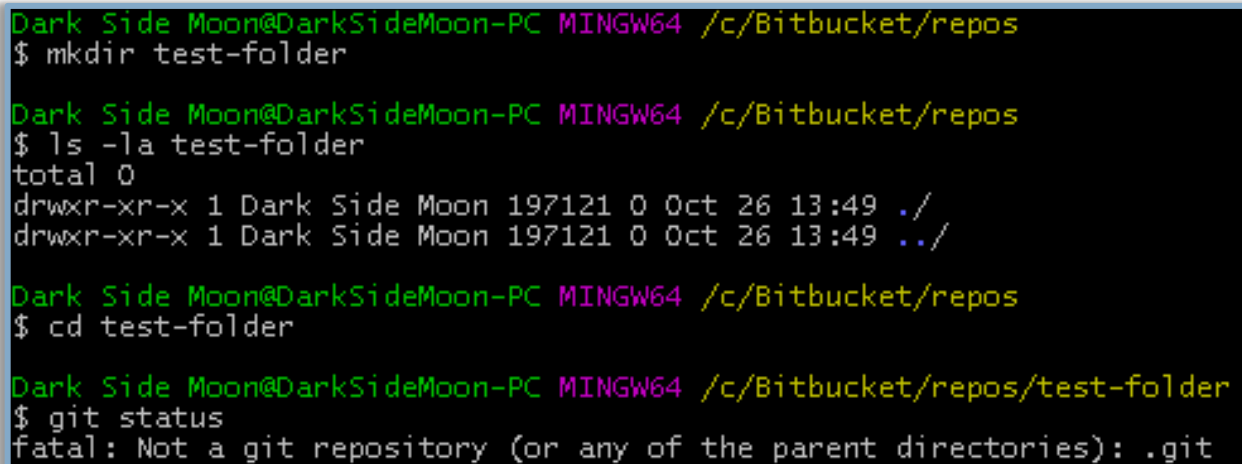
Wait for your instructor to tell you to begin → Lab 1: Create an Empty Repository

How to create an empty repository using “git init”:

Open up a terminal and cd to a path where you want to store your Git repositories. Enter the each of the following commands separately:

```
$ mkdir test-folder - Makes a new directory
$ ls -la test-folder - Shows the contents of the folder
$ cd test-folder - Change directory
$ git status - Displays state of working directory
$ git init
$ git status
```

As you enter in some of these commands, the terminal will print output in response. You should see something like this:



```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos
$ mkdir test-folder

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos
$ ls -la test-folder
total 0
drwxr-xr-x 1 Dark Side Moon 197121 0 Oct 26 13:49 ./
drwxr-xr-x 1 Dark Side Moon 197121 0 Oct 26 13:49 ../

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos
$ cd test-folder

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder
$ git status
fatal: Not a git repository (or any of the parent directories): .git
```

Note that the git repository has not yet been created, so the git status command doesn't work.

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder
$ git init
Initialized empty Git repository in C:/Bitbucket/repos/test-folder/.git/

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master

Initial commit

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

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$

```

That's all there is to making a new empty repository!

Wait for your instructor to tell you to begin → Lab 2: Add and Commit Files

Entering “ls -l” confirms that we haven’t added any files to the repository. The output after entering “git status” is a *little* more interesting, nevertheless all it’s really telling us is that we have ‘nothing to commit’. Let’s change that before we continue.

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos
$ cd test-folder

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls -l
total 0

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master

Initial commit

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

```

We need to create a file to continue this example.

Within test-folder, create a new file by using the terminal’s vi editor. Enter:

```
$ vi newFile
```

In the vi editor:

1. Press ‘i’ to enter insert mode, then type in a few lines of text.
2. When you’re done, press ‘Esc’ or ‘Ctrl+C’ to exit the insert mode.
3. Type ‘:wq’ to write and quit.

It’ll look something like this:

```
Type a few lines of text
The first of many
~
~
~
~
:wq
```

You can enter “ls” to verify that your file has been created, followed by “git status” to see how the repository reacts to ‘newFile’:

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls
newFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master

Initial commit

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

        newFile

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

This shows that your Git repository is aware of ‘newFile’, but it isn’t tracking any changes to it yet- and it won’t until you tell it to do so.

To tell Git you intend to begin tracking ‘newFile’, enter “git add newFile”.

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git add newFile
warning: LF will be replaced by CRLF in newFile.
The file will have its original line endings in your working directory.

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master

Initial commit

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

        new file:   newFile
```

There’s just one last step we need to finalize the addition of ‘newFile’ and have Git begin tracking it, and that’s executing “git commit -m “Initial commit””:

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git commit -m "Initial commit"
[master (root-commit) 3519b48] Initial commit
1 file changed, 2 insertions(+)
create mode 100644 newFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Wait for your instructor to tell you to begin → Lab 3: Renaming, Removing, Reverting Files and undoing A File Remove

While in the ‘test-folder’ repository, let’s rename ‘newFile’ to ‘oldFile’ using the “git mv” command. We’ll simply enter the following:

```
$ ls
$ git status
$ git mv newFile oldFile
```

Your results should look something like this:

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls
newFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git mv newFile oldFile
```

Entering “ls” after this shows the file has been renamed. Running “git status” shows us that Git recognizes the change and is waiting for it to be committed:

```
$ ls
$ git status
```

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls
oldFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    newFile -> oldFile

```

Make the change permanent by executing “git commit -m “Renamed newFile to oldFile””. Enter “git status” for verification:

```

$ git commit -m "Renamed newFile to oldFile"
$ git status

```

The results should look something like this.

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git commit -m "Renamed newFile to oldFile"
[master d8f7031] Renamed newFile to oldFile
1 file changed, 0 insertions(+), 0 deletions(-)
rename newFile => oldFile (100%)

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean

```

After changing the name of ‘newFile’ to ‘oldFile’, let’s see how it can be removed from the repository:

```

$ ls
$ git status
$ git rm oldFile

```

The results should look something like this.

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls
oldFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git rm oldFile
rm 'oldFile'

```

Using “ls” shows that ‘oldFile’ is no longer listed in the directory. “git status” shows that Git is aware of the removal and is waiting for the change to be committed:

```
$ ls
$ git status
```

The results should look something like this.

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    oldFile
```

Enter “git commit -m “Removed oldFile”” and the change becomes official. “git status” shows that our working tree is clean- there’s nothing more to commit.

```
$ git commit -m "Removed oldFile"
$ git status
```

The results should look something like this.

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git commit -m "Removed oldFile"
[master 5f087aa] Removed oldFile
1 file changed, 2 deletions(-)
delete mode 100644 oldFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Now let’s think about what we would have needed to do if we changed our mind about deleting ‘oldFile’ right before running “git commit”.

To get started:

Use “git log” to find the <commit ID> of the point in time we want to trace back to.

Enter “git revert <commit ID>” to undo that commit.

Save the default commit message for the revert.

Run “git rm oldFile” again.

Study the following commands to see how this is done.

Enter “git log” to find the last commit. Then copy the commit ID and use it in the “git revert” command.

```
$ git log
```

The results should look something like this.

```
$ git log
commit 0917550d89a1da3d6d6533a659cae2d1dcce0476
Author: atbitgit <someone@example.com>
Date: Sat Oct 29 16:35:40 2016 -0500

    Removed oldFile
```

Open	
Copy	Ctrl+Ins
Paste	Shift+Ins

Copy the commit ID. Paste it into the revert command:

```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git revert 0917550d89a1da3d6d6533a659cae2d1dcce0476
```

Git will open a ‘vi’ session for you to enter a comment, so save the default message for the revert:

```
Revert "Removed oldFile"
This reverts commit <ID>
~
# Please enter the commit..
# On branch master
# Changes to be committed:
#   new file:   oldFile
:wq
```

Enter the following:

Enter Press ‘Esc’ or ‘Ctrl+C’

Enter “:wq”

Entering “ls” shows that ‘oldFile’ has returned. “git status” shows that there are no new changes to be committed. We’re right back to where we were before the delete was ever made. Let’s go ahead and remove ‘oldFile’ one more time.

```
$ ls
$ git status
$ git rm oldFile
```

The results should look something like this.

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls
oldFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git rm oldFile
rm 'oldFile'

```

Run “ls” and “git status” to see how the working directory and staging area have changed:

```

$ ls
$ git status

```

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    oldFile

```

It’s at this point where we can unstage ‘oldFile’ if we’ve changed our minds about removing it from the repository. Run the following commands:

```

$ git reset HEAD -- oldFile
$ git checkout -- oldFile

```

The results should look something like this.

```

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git reset HEAD -- oldFile
Unstaged changes after reset:
D      oldFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git checkout -- oldFile

```

To verify ‘oldFile’ has returned to where it was before removing it, run the following commands:

```

$ ls
$ git status

```

The results should look something like this.

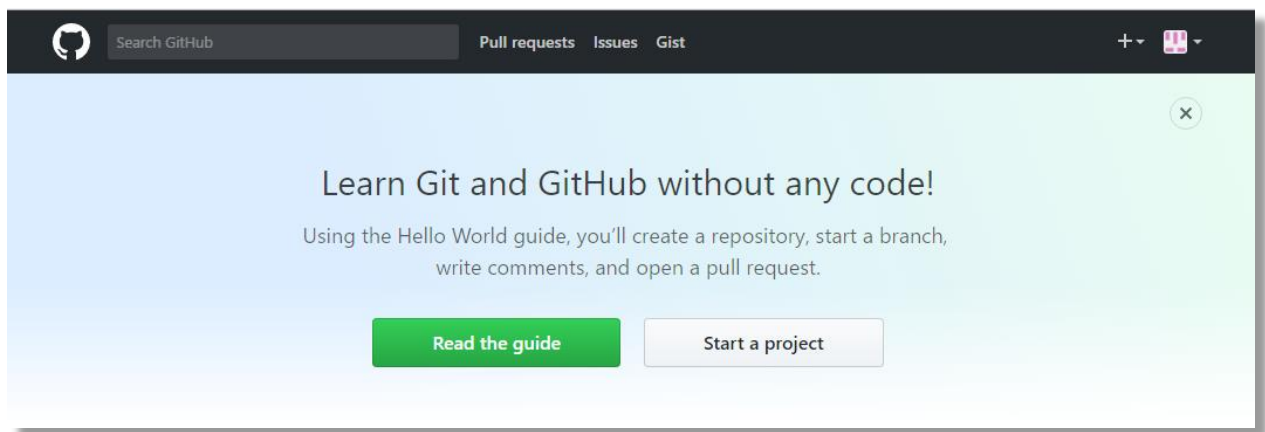
```
Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ ls
oldFile

Dark Side Moon@DarkSideMoon-PC MINGW64 /c/Bitbucket/repos/test-folder (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Wait for your instructor to tell you to begin ➔ Lab 4: Fork, Clone, Push, Fetch & Merge

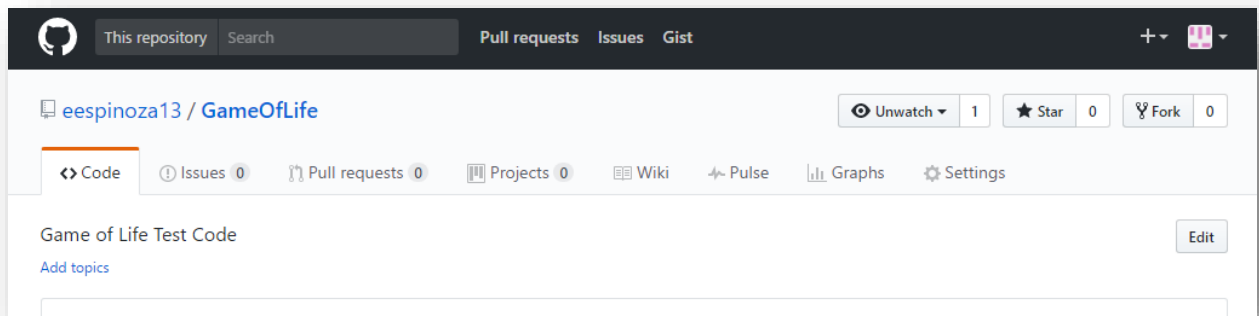
First, let's walk through forking a repository on GitHub:

Example: <https://github.com/eespinoza13/GameOfLife>



1. Log into (or create) a Github account.
2. Locate the repository you want to fork.
3. You can use the search area in the GitHub menu bar to do so.

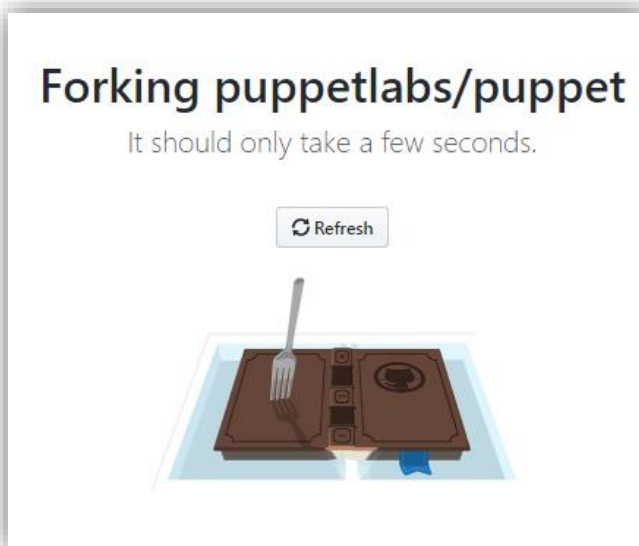
Click the Fork button below:

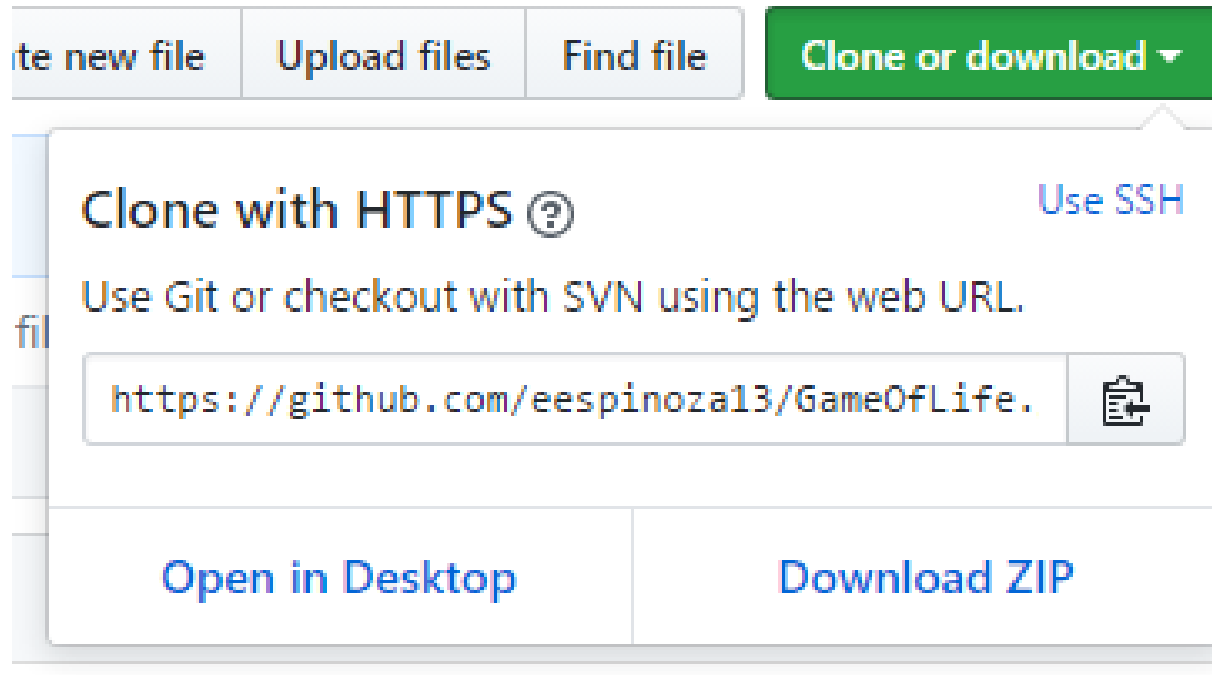


3. Navigate to the repository – in this case, GameOfLife

4. Click on the Fork button.

You will see a loading page.





Example: <https://github.com/eespinoza13/GameOfLife>

Although you can work on the repository from here, it's best to go one step further and create a local copy of it on your computer so you may work on it at any time, from any place. This is what cloning is for.

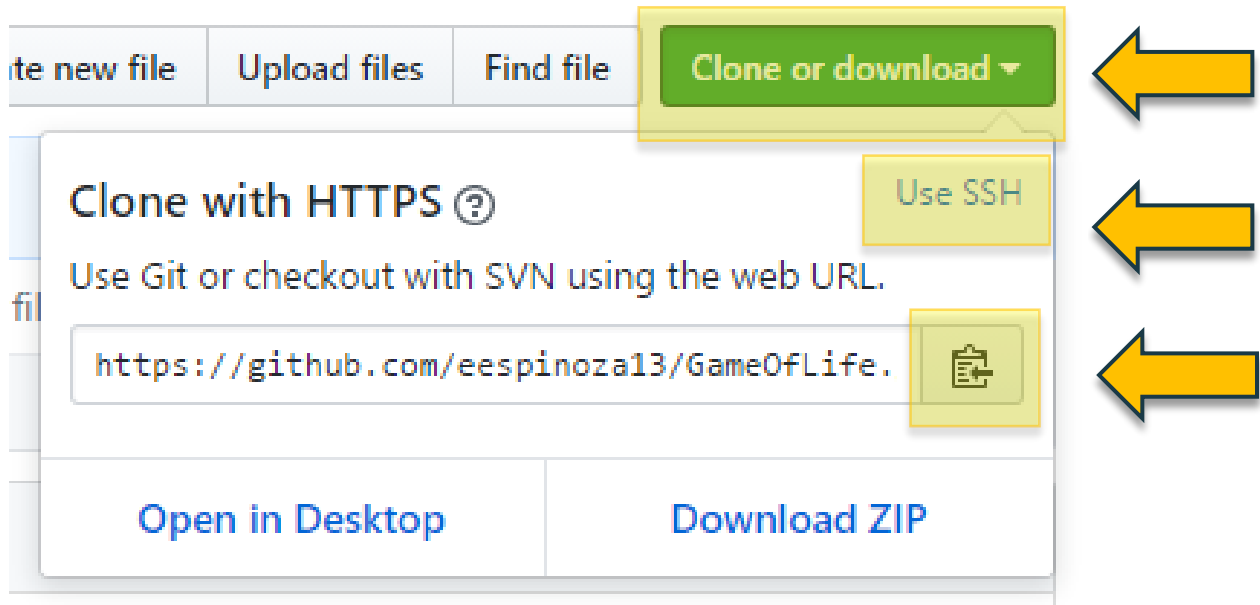
To clone a repository using the terminal:

1. Locate the repository you want to clone.
2. Copy the path of the repository.
 - GitHub provides a “**Clone**” button on the web interface of each repository it hosts. This allows you to easily copy the command you need.
3. Open up the terminal and “cd” into a directory you use for storing repositories.
4. If you're typing in the following command, substitute <repo> for the path of the repository:

```
git clone <repo>
```

5. Press enter.

To clone a repository using the GitHub website:



For a GitHub repo, click the green **“Clone or Download”** button on the right side of the page. A pop-up will appear. Click **“Use SSH”** – this will show the repo path for you to enter into your `“git clone <repo>”` command. Copy this by clicking the clipboard.

Open up your terminal and cd into the directory you want your cloned repo to be. Enter in the “git clone <repo>” command with the copied URL from the previous step. In this example, the following was entered:

```
git clone https://github.com/eespinoza13/GameOfLife.git
```

Afterward, watch as Git goes right to work cloning your repository.

```

Elisa E@Elisa MINGW64 ~/Repositories (master)
$ git clone https://github.com/eespinoza13/GameOfLife.git
Cloning into 'GameOfLife'...
remote: Counting objects: 1782, done.
remote: Compressing objects: 100% (1702/1702), done.
remote: Total 1782 (delta 1121), reused 0 (delta 0), pack-reused 3
Receiving objects: 100% (1782/1782), 15.10 MiB | 9.08 MiB/s, done.
Resolving deltas: 100% (1121/1121), done.
Checking out files: 100% (1521/1521), done.

Elisa E@Elisa MINGW64 ~/Repositories (master)
$ ls -la
total 44
drwxr-xr-x 1 Elisa E 197121    0 Apr 27 13:05 ./
drwxr-xr-x 1 Elisa E 197121    0 Apr 27 12:45 ../
drwxr-xr-x 1 Elisa E 197121    0 Apr 27 12:45 beta/
-rw-r--r-- 1 Elisa E 197121 1791 Apr 26 15:03 branchmerge.sh
-rw-r--r-- 1 Elisa E 197121 3517 Apr 26 15:08 error
drwxr-xr-x 1 Elisa E 197121    0 Mar  7 19:25 Game-of-Life/
drwxr-xr-x 1 Elisa E 197121    0 Apr 27 13:05 GameOfLife/

```

Next, enter “cd <path to your local repository>”, then “ls -la” to see that everything that existed in your Bitbucket repository is now copied onto your local machine:

```

Elisa E@Elisa MINGW64 ~ (master)
$ cd "C:/Users/Elisa E/Repositories/Game-of-Life-Github"

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ ls -la
total 46
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 12:03 ./
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 11:28 ../
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 11:30 gameoflife-acceptance-tests/
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 11:30 gameoflife-build/
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 11:30 gameoflife-core/
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 11:30 gameoflife-deploy/
drwxr-xr-x 1 Elisa E 197121    0 Apr 24 11:28 gameoflife-web/
-rw-r--r-- 1 Elisa E 197121   154 Feb 21 11:48 ignore.gitignore
-rw-r--r-- 1 Elisa E 197121    52 Feb 21 11:49 infinitest.filters
-rw-r--r-- 1 Elisa E 197121 22787 Feb 21 11:49 pom.xml
-rw-r--r-- 1 Elisa E 197121 3090 Feb 21 11:48 README.markdown

```

Now let’s see how “git push” can be used to share our local changes with a remote repository. Start by opening the vi editor to create a new file called ‘TeamMembers.txt’. Type in your username or alias and then save the file.

```
Team Member #1
```

```
~  
~  
:wq
```

```
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)  
$ vi TeamMembers.txt  
  
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)  
$ git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    ../../.bash_history  
    ../../.eclipse/  
    ../../.gitconfig  
    ../../.oracle_jre_usage/  
    ../../.p2/
```

Enter “git add TeamMembers.txt” to stage the file, then
Enter ‘git commit -m “Adding a team member list”’ to commit it.

```
$ git add TeamMembers.txt  
$ git status  
$ git commit -m "message"
```

```
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)  
$ git add TeamMembers.txt  
warning: LF will be replaced by CRLF in Repositories/Game-of-Life-Github/TeamMembers.txt.  
The file will have its original line endings in your working directory.  
  
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)  
$ git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
  
    new file:   TeamMembers.txt  
  
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)  
$ git commit -m "Adding a team member list"  
[master 2a63776] Adding a team member list  
1 file changed, 2 insertions(+)  
create mode 100644 Repositories/Game-of-Life-Github/TeamMembers.txt
```


Pushing Local Changes

Check the status of your local repository with “git status”. Then, to share your newest addition of ‘TeamMembers.txt’ with your remote repository, enter in “git push origin master”. You may be asked to enter credentials for your repository in order to continue from here.

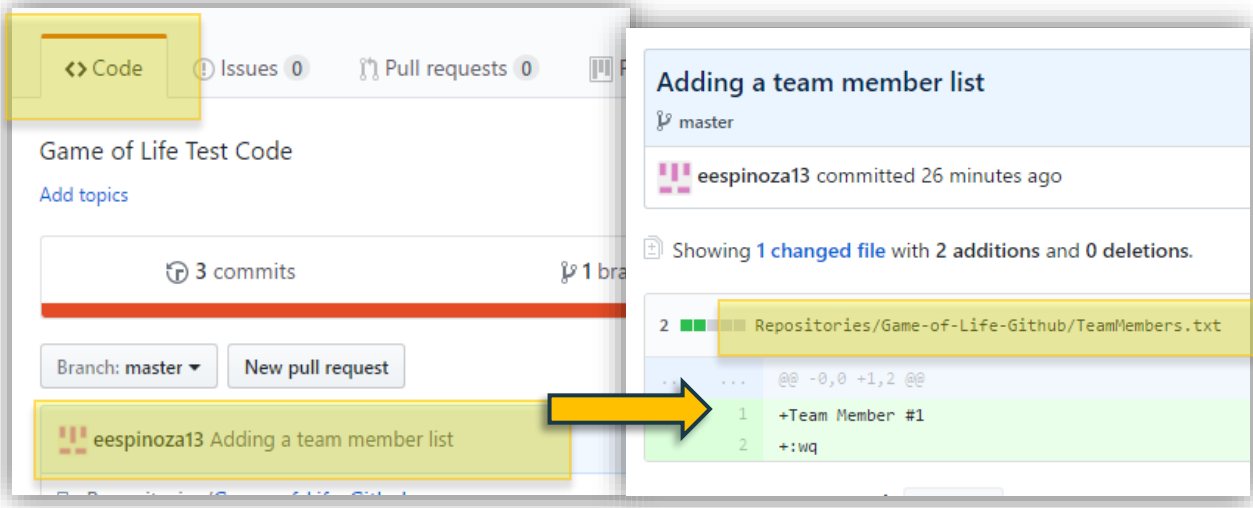
```
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Untracked files:
  (use "git add <file>..." to include in what will be committed)

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git push origin master
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 444 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/eespinoza13/GameOfLife.git
   b86b6fd..2a63776  master -> master
```

“git status” will show that your branch is up-to-date with your remote repository:

```
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
  (use "git push" to publish your local commits)
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

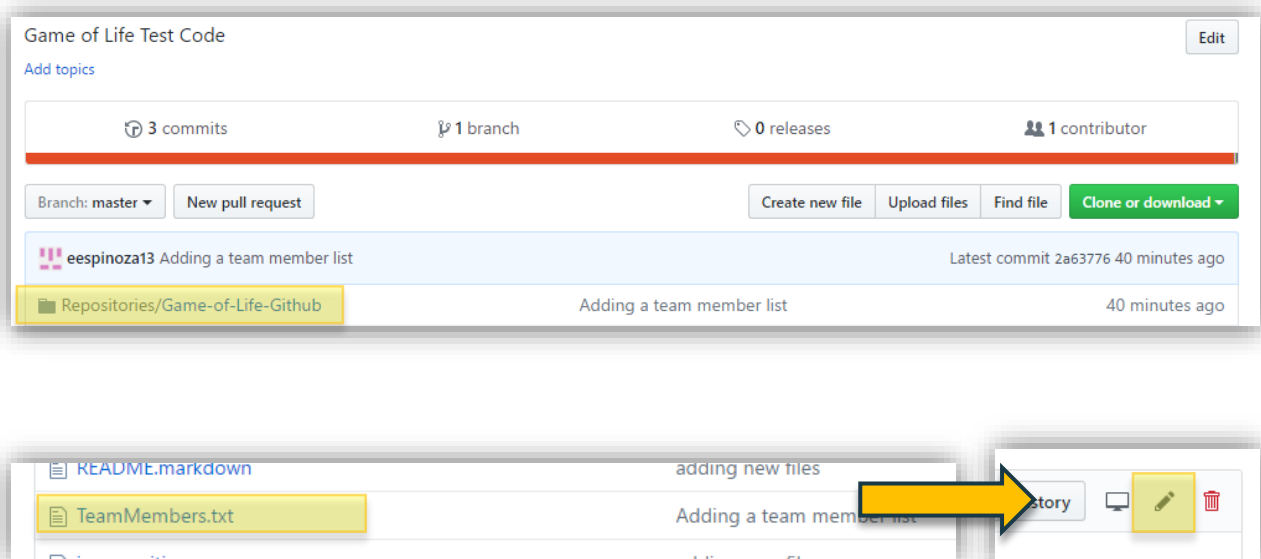
In a web browser, navigate to the ‘Code’ tab of your GitHub repository. Scroll down and you should see your commit listed at the top of the Code list. Click on the commit & it should take you to the file “TeamMembers.txt”.

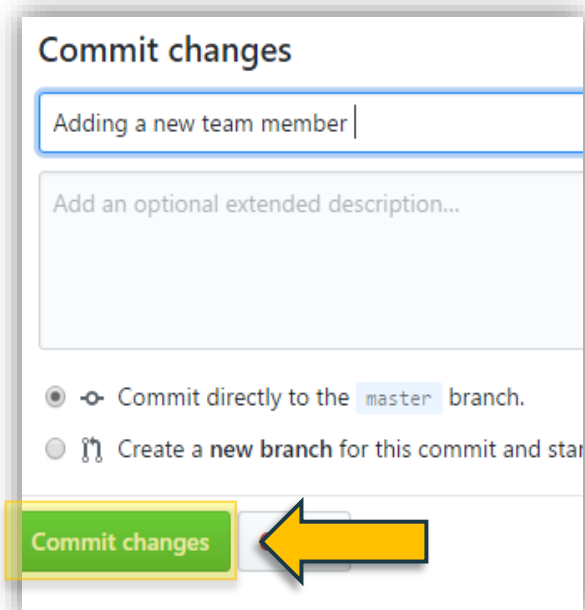
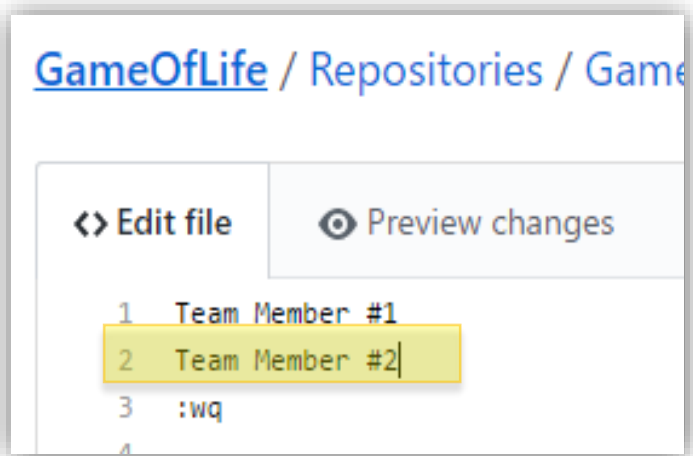


Fetch and Merge Changes

How would we incorporate this change into our local repository?

1. Make your changes in the text editor.
2. Enter a commit message.
3. Click 'Commit Changes'.
4. Go back to the terminal after the commit.





In the directory of the local repository, enter the following commands, one-by-one:

```
$ git status
$ git fetch origin
$ git status
$ git merge origin/master
$ git status
$ cat TeamMembers.txt
```

Here's what you should see:

```

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git fetch origin
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
From https://github.com/eespinoza13/GameOfLife
 2a63776..bba323f  master    -> origin/master

```

```

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
.. (use "git pull" to update your local branch)

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git merge origin/master
Updating 2a63776..bba323f
Fast-forward
 Repositories/Game-of-Life-Github/TeamMembers.txt | 1 +
 1 file changed, 1 insertion(+)

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ cat TeamMembers.txt
Team Member #1
Team Member #2

```

We got through a lot in this lab. To summarize, here's what we've learned:

1. How to fork a remote repository using Bitbucket.
2. How to use "git clone <repo>" to copy the forked repository onto our local machine.
3. How to use "git push <remote> <branch>" to add local changes to a remote repository.
4. How to use "git fetch <remote>" to discover remote changes.
5. How to incorporate those changes to our remote repository using "git merge <remote>".

Wait for your instructor to tell you to begin → Lab 5: Branch and Merge

- To set up this lab, find 'branchmerge.sh' located in <https://github.com/eespinoza13/DevOpsTraining>
 - Navigate to the directory where you keep your repositories and add this file there.
- You will supply the name of a repository to create (we use "beta" in the lab) and where to create it.
- The purpose of this lab is to show you how to manage a merge conflict of a text file. When you execute this script, it will create a git repo with a master branch. It will then populate the branch with some files. It will create two more branches (feature and develop) and change the same line in the file "even" differently in each branch. It will stage and commit the file in each branch.
- Now, when you manually merge the develop and feature branches back into the master branch (in the lab), you will get a merge conflict.

To set up this lab, your instructor should have given you a file named 'branchmerge.sh'. Navigate to the directory where you keep your repositories and add this file there. Then enter 'ls -l' to verify. For example:

```
Elisa E@Elisa MINGW64 ~/Repositories/Game-of-Life-Github (master)
$ cd ..

Elisa E@Elisa MINGW64 ~/Repositories (master)
$ ls -l
total 16
-rw-r--r-- 1 Elisa E 197121 1791 Apr 26 15:03 branchmerge.sh
drwxr-xr-x 1 Elisa E 197121  0 Mar  7 19:25 Game-of-Life/
drwxr-xr-x 1 Elisa E 197121  0 Apr 26 14:51 Game-of-Life-Github/
drwxr-xr-x 1 Elisa E 197121  0 Mar  8 12:37 HangMan/
```

To get started, enter the following using 'beta' in place of <repo name>:

```
$ sh branchmerge.sh <base directory> <repo name> 2>> error
```

For example:

```
Elisa E@Elisa MINGW64 ~/Repositories (master)
$ sh branchmerge.sh "/c/Users/Elisa E/Repositories" beta 2 >> error
```

Once the script is complete, “cd” into ‘beta’ to see its contents and what state the repository is in:

```
$ cd beta
$ ls -l
$ git branch
$ git status
```

```
Elisa E@Elisa MINGW64 ~/Repositories (master)
$ cd beta

Elisa E@Elisa MINGW64 ~/Repositories/beta (feature)
$ ls -l
total 3
-rw-r--r-- 1 Elisa E 197121 197 Apr 26 15:08 even
-rw-r--r-- 1 Elisa E 197121 261 Apr 26 15:08 stuff
-rw-r--r-- 1 Elisa E 197121 23 Apr 26 15:08 stuffaswell

Elisa E@Elisa MINGW64 ~/Repositories/beta (feature)
$ git branch
  develop
* feature
  master

Elisa E@Elisa MINGW64 ~/Repositories/beta (feature)
$ git status
On branch feature
nothing to commit, working tree clean
```

Enter “git checkout master”, then examine the contents and state of the branch:

```
$ git checkout master
$ ls -l
$ cat even
$ git status
```

Merge the develop branch into the master branch using “git merge <branch>”. Examine the file and repository status.

```
$ git merge develop
$ cat even
$ git status
```

```
Elisa E@Elisa MINGW64 ~/Repositories/beta (master)
$ git merge develop
Updating 96434df..3555e20
Fast-forward
 even | 1 +
 1 file changed, 1 insertion(+)

Elisa E@Elisa MINGW64 ~/Repositories/beta (master)
$ cat even
total 0
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 even
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuff
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuffaswell
"hello good-looking"

Elisa E@Elisa MINGW64 ~/Repositories/beta (master)
$ git status
On branch master
nothing to commit, working tree clean
```

The Merge
Succeeded! No
Conflict!

Now try merging feature into master. Auto-merging fails and Git reports that there is a conflict in the file ‘even’. This needs to be resolved before we can continue.

```
$ git merge feature
$ git status
```

```
Elisa E@Elisa MINGW64 ~/Repositories/beta (master)
$ git merge feature
Auto-merging even
CONFLICT (content): Merge conflict in even
Automatic merge failed; fix conflicts and then commit the result.

Elisa E@Elisa MINGW64 ~/Repositories/beta (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

        both modified:   even

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

The Merge Failed due to Conflict!

Let's take a closer look at the conflicting content in file 'even':

Note the following:

<<<< Marks the beginning of the conflict.

HEAD refers to the current branch (master)

HEAD's conflicting content is displayed.

==== Equal signs separate the conflicting data.

Feature's conflicting content is displayed.

>>>> Marks the end of the conflict.

The name of the target branch is displayed (feature)


```

Elisa E@Elisa MINGW64 ~/Repository
$ cat even
total 0
-rw-r--r-- 1 Elisa E 197121 0 Apr
-rw-r--r-- 1 Elisa E 197121 0 Apr
-rw-r--r-- 1 Elisa E 197121 0 Apr
<<<<<<< HEAD
"hello good-looking"
=====
"I've seen all Good People!"
>>>>>>> feature

```

Using the vi editor, decide how you want to resolve the conflict by either incorporating both versions, deleting one and not the other, etc.

```

$ cat even
$ vi even

```

```

Elisa E@Elisa MINGW64 ~/Repositories/beta (master|MERGING)
$ cat even
total 0
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 even
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuff
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuffaswell
<<<<<<< HEAD
"hello good-looking"
=====
"I've seen all Good People!"
>>>>>>> feature

Elisa E@Elisa MINGW64 ~/Repositories/beta (master|MERGING)
$ vi even

```

```
total 0
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 even
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuff
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuffaswell
<<<<<< HEAD
"hello good-looking"
"I've seen all Good People!"
>>>>>> feature|
~
```

Change it to this:

```
total 0
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 even
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuff
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuffaswell
<<<<<< HEAD
"hello good-looking" "I've seen all Good People!"
>>>>>> feature|
~
```

Enter “:wq” when finished.

Once you are satisfied with how you have resolved the conflict, use “git add” to stage ‘even’ and then “git commit” to complete the merge.

```
$ git add even
$ git status
$ git commit -m "Solved conflict"
```

```

Elisa E@Elisa MINGW64 ~/Repositories/beta (master|MERGING)
$ git add even

Elisa E@Elisa MINGW64 ~/Repositories/beta (master|MERGING)
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
      modified:   even

Elisa E@Elisa MINGW64 ~/Repositories/beta (master|MERGING)
$ git commit -m "Solved Conflict"
[master b3ac7aa] Solved Conflict

```

Finished! You’ve manually resolved a merge conflict between two branches. Use “cat even” and “git status” to confirm that your changes were indeed committed to the master branch.

```

$ cat even
$ git status

```

```

Elisa E@Elisa MINGW64 ~/Repositories/beta (master)
$ cat even
total 0
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 even
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuff
-rw-r--r-- 1 Elisa E 197121 0 Apr 26 15:08 stuffaswell
<<<<<<< HEAD
"hello good-looking" "I've seen all Good People!"
>>>>>>> feature

Elisa E@Elisa MINGW64 ~/Repositories/beta (master)
$ git status
On branch master
nothing to commit, working tree clean

```