**Training**

| **Material** | **Duration** |
| --- | --- |
| [Linux Terminal Tutorial Episode 1: Back to Basics](https://www.youtube.com/watch?v=2FiQSLdnBqA) | 11:32 |
| [Code Academy Command Line](https://www.codecademy.com/learn/learn-the-command-line) | - |
| [Git & GitHub Crash Course For Beginners](https://www.youtube.com/watch?v=SWYqp7iY_Tc) | 32:41 |
| [Forks and Pull Requests](https://www.youtube.com/watch?v=_NrSWLQsDL4) | 12:22 |

**Optional**

*If you've got time and/or want to dig deeper, consider the following:*

| **Material** | **Duration** |
| --- | --- |
| [Git Tutorial for Beginners: Command-Line Fundamentals](https://www.youtube.com/watch?v=HVsySz-h9r4) | 30:32 |

* Proceed to read the next 2 chapters of git <https://git-scm.com/book/en/v2>
* Try <https://try.github.io/>
* GIT, the simple guide: <http://rogerdudler.github.io/git-guide/>
* Bookmark: <http://ohshitgit.com/>
* Windows users might enjoy [Total Commander](https://www.ghisler.com/)

## Git

Git is free **open source software** (the [source code is public](https://github.com/git/git)) written by Linus Torvalds who also wrote the Linux operating system.

Git is a program for keeping track of changes over time, known in programming as**version control**. If you've used a track changes feature in a text editing software then you're already familiar with the concept!

### Install Git

We recommend installing Git on your computer by downloading the [GitHub Desktop](https://desktop.github.com) app. We'll not use the desktop app in Git-it (we're learning terminal!) but it includes Git and is the easiest way to install Git on all platforms in the same way.

* **Windows**: Use the **Git Shell** for your terminal.
* **Mac**: Open GitHub Desktop and from Preferences, select the command line tools install. Use the **terminal** app as your terminal.

Already have Git or not sure? Type git --version in your terminal and if it returns a version number higher than 1.7.10, you're set! For more information, visit the[Git website](http://git-scm.com).

## Git Software

The GitHub Desktop app can do a lot of things with Git but not all, which is why learning the terminal is important. But once you've got that down, you'll be glad to have the desktop app because it organizes your project's information more visually, like the GitHub website.

Git on its own isn't like other programs on your computer. You'll likely not see an icon on your desktop, but it will always be available to you and you'll be able to access it at anytime from your terminal or Git desktop applications.

### Configure Git

Once GitHub Desktop (and therefore Git) is installed, open your **terminal**. You can verify that Git is really there by typing:

git --version

This will return the version of Git on your computer and look something like this:

git version 1.9.1

Next, configure Git so it knows to associate your work to you:

Set your name:

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

Now set your email:

git config --global user.email "youremail@example.com"

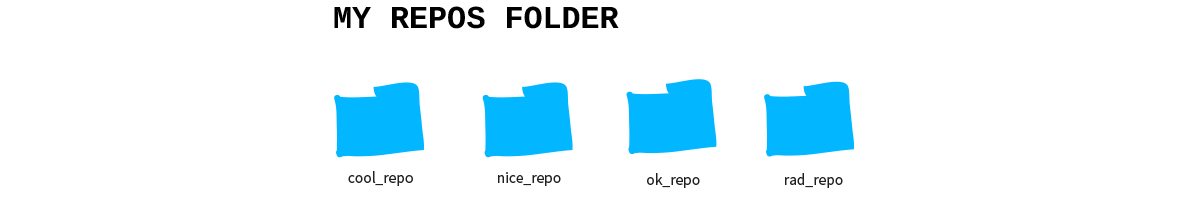
You're done with your first challenge! Click the 'Verify' button to check the challenge.

GIT

Create a new repository on your computer.

## Repositories

A **repository** is a collection of related items. In our case, when writing software, it is a collection of files related to a software project. You can imagine it as a project folder with all the relevant files inside of it. In fact, that's what it will look like on your computer anyways. Sometimes they're called "repos" for short.



You tell Git what your project is and Git will start tracking all of the changes to that folder. This makes it a Git repository: a folder of items being tracked by Git. Git tracks when files are added, subtracted or even a single letter in a single file is changed. All of this plus who did it and when is tracked by Git. In software, tracking changes like this is called **version control**.

## Using Terminal

**Terminal** (or Bash) is a way to instruct your computer to do things by typing commands rather than clicking applications with your mouse. You can rename files, open files, create new folders, move between directories (folders) and more all by running typed commands. You can even use a text editor for code (like [Vim](http://en.wikipedia.org/wiki/Vim_(text_editor))) in your terminal and never have to leave!

Besides navigating your computer, you can also use programs in Terminal that have a **command-line interface (CLI)**, meaning they can be run with commands in terminal. Git is one of these. The first part of the command lets your computer know you're talking to Git. The parts following that are the commands and the different options you want Git to act on.

In Git-it you'll learn a few basic command-line actions (in addition to Git commands) for navigating your computer; they're described within the steps.

### Create a Repository

You're going to create a new project folder and then initialize it as a Git repository. We'll name the folder 'hello-world' in this challenge, but you can choose a different name if you already have a folder named this (it's a common first project name).

Time to open your terminal!

In your terminal window, type these commands, one at a time, pressing enter/return after each.

First, make a new folder:  
Tip: mkdir stands for make directory

mkdir hello-world

Then go into that folder:  
Tip: cd stands for change directory

cd hello-world

Finally, tell Git to initialize (start tracking) the folder you are now in:

git init

The last command should return something starting with "Initialized empty Git repository". The others commands do not return anything.

You did it! If you want to be double-sure that it's a Git repository, type git statusand if it doesn't return 'fatal: Not a git repository...', you're golden!

## Commits

Commits are core to using Git. They are the moments in which you save and describe the work you've done. They are the ticks in the timeline of your project's history.

### Create a New File

Now that you've got a repository started let's add a file to it.

Open your text editor and create a new empty file. Now write a little bit of text, perhaps type "Hello!", and save the file as 'readme.txt' in the 'hello-world' folder you created in the last challenge.

### Check Status + Add and Commit Changes

Next check the **status** of your repository to find out if there have been changes. You know you have changed something, but does Git?

Make sure you're still within your 'hello-world' directory when you're running these commands. Use Git to see what changed in your repository:

First, check the status:

git status

Git should tell you that a file has been added.

Then **add** the file you just created so that it becomes a part of the changes you will**commit** (aka save) with Git:

git add readme.txt

Finally, **commit** those changes to the repository's history with a short (m) message describing the updates.

git commit -m "Created readme"

### Step: Make More Changes

Now add another line to 'readme.txt' and save the file again.

In terminal, you can view the **diff**erence between the file now and how it was at your last commit.

Tell Git to show you the **diff**:

git diff

Now with what you just learned above, commit this latest change.

## Working Better, Together

The repository you've created so far is just on your computer, which is handy, but makes it pretty hard to share and work on with others. No worries, that's what GitHub is for!

GitHub is a website that allows people everywhere to upload what they're working on with Git so that everyone can easily work together on the same project. GitHub which acts as the central repository for you and everyone else to share. You push your changes to it and pull down changes from others.

### Create a GitHub Account

Visit [github.com](http://github.com) and sign up for a free account. **High five, welcome!**

### Add GitHub username to Git

Add your GitHub username to your Git configuration. We'll do this just for the sake of Git-it; it makes it easier to verify the upcoming challenges. Save it exactly as you created it on GitHub and **capitalize where capitalized**. Note, you don't need to enter the "<" and ">".

Add your GitHub username to your Git configuration:

git config --global user.username <USerNamE>

You can double check what you have set in your Git config by typing:

git config --global user.username

VERIFY

Make sure you type "**user.username**" above and not "user.name", which would override the name you set in the first challenge and leave you with no username property! If you found you did that, it's ok, just repeat the step in the first challenge to add your name and then complete the step above.

#### GitHub & Git config usernames do not match

A common error is not having your GitHub username match the case of the one you set with git config. For instance, 'JLord' isn't the same as 'jlord'

To change your username set with Git, just do the same command you did earlier, but with the correct capitalization:

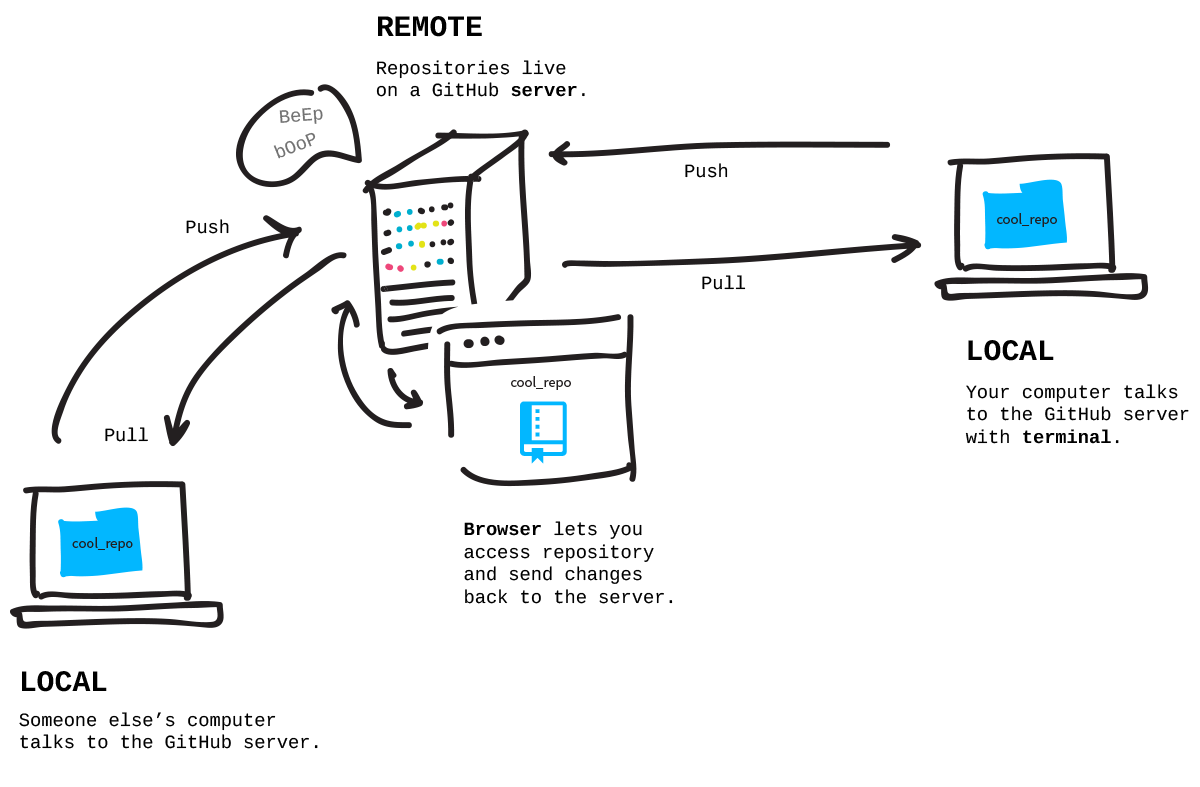
git config --global user.username <USerNamE>

When you've made your updates, verify again!

## Remotes

When you put something on GitHub that copy lives on one of GitHub's servers. This makes it a **remote** repository because it is not on your computer, but on a server, "remote" and somewhere else. By **pushing** your **local** (on your computer) changes to it, you keep it up to date.

Others can always then get the latest from your project by **pulling** your changes down from the remote (and onto their computer). This is how everyone can work on a project together without needing access to your computer where your local copy is stored.



### Create a Remote Repository

You want to sync your **local** version with one stored on GitHub.com. First create a new **remote** repository on GitHub.com.

* Go to github.com, log in, and click the '+' in the top right and then click 'New repository'.
* Give it a name that matches your local repository's name, 'hello-world', and a short description.
* Make it public. This means it'll be listed on your public profile.
* Don't initialize with a README because we already have a file, locally, named 'readme.txt'. This is a helper option from GitHub if you hadn't already made it.
* Leave '.gitignore' and 'license' set to 'none'. We won't use them this tutorial.
* Click create repository!

## Readmes, .gitignores and Licenses

These are common files in open source projects so to make starting a new project easier, GitHub provides an option for creating them for you.

A **readme** explains what the project is, how to use it, and often times, how to contribute (though sometimes there is an extra file, 'CONTRIBUTING.md', for those details).

A **.gitignore** is a list of files that Git should not track, for instance, files with passwords!

A **license** file is the type of license you put on your project. This lets others know how they can use it. Information on the types is here: [choosealicense.com](http://www.choosealicense.com).

### Connect your Local to your Remote

Now you've got an empty repository started on GitHub.com. At the top you'll see 'Quick Setup', make sure the 'HTTPS' button is selected and copy the address—this is the address of your repository on GitHub's servers.

Back in your terminal, and inside of the 'hello-world' folder that you initialized as a Git repository in the earlier challenge, you want to tell Git to remember the address of the remote version on GitHub's servers. You can have multiple remotes so each requires a name. The primary remote is typically named origin.

To add a remote named 'origin' to your repository:

git remote add origin <URLFROMGITHUB>

Your **local** repository now knows where your **remote** repository, named 'origin', lives on GitHub's servers. Think of it as adding a name and address on speed dial—now when you need to send something there, you can.

**Windows Users:**

If you have **GitHub Desktop** on your computer, a remote named 'origin' is automatically created in your local repository. In that case, you'll just need to tell it what URL to associate with 'origin'. Use this command instead of the 'add' one above:

git remote set-url origin <URLFROMGITHUB>

### Push Work to your Remote

Next you want to **push** (send) everything you've done locally to your remote repository on GitHub. This is something you'll do often so that your remote version is up to date and matching the state of your local version.

Git has a branching system so that you can work on different parts of a project at different times. We'll learn more about that later, but by default the first branch is named 'master'. When you push (and later pull) from a project, you tell Git the**branch name** you want and the name of the **remote** that it lives on.

In this case, we'll send our branch named 'master' to our remote on GitHub named 'origin'.

git push origin master

Now go to your remote repository's page on GitHub.com and refresh the page.**Wow!** Everything is now the same locally as it is remotely. Congrats on your first public repository!

SELECT DIRECTORY VERIFY

* **Add remote connections**

git remote add <REMOTENAME> <URL>

* **Set a URL to a remote**

git remote set-url <REMOTENAME> <URL>

* **Pull in changes**

git pull <REMOTENAME> <BRANCHNAME>

* **View remote addresses**

git remote -v

* **Push changes**

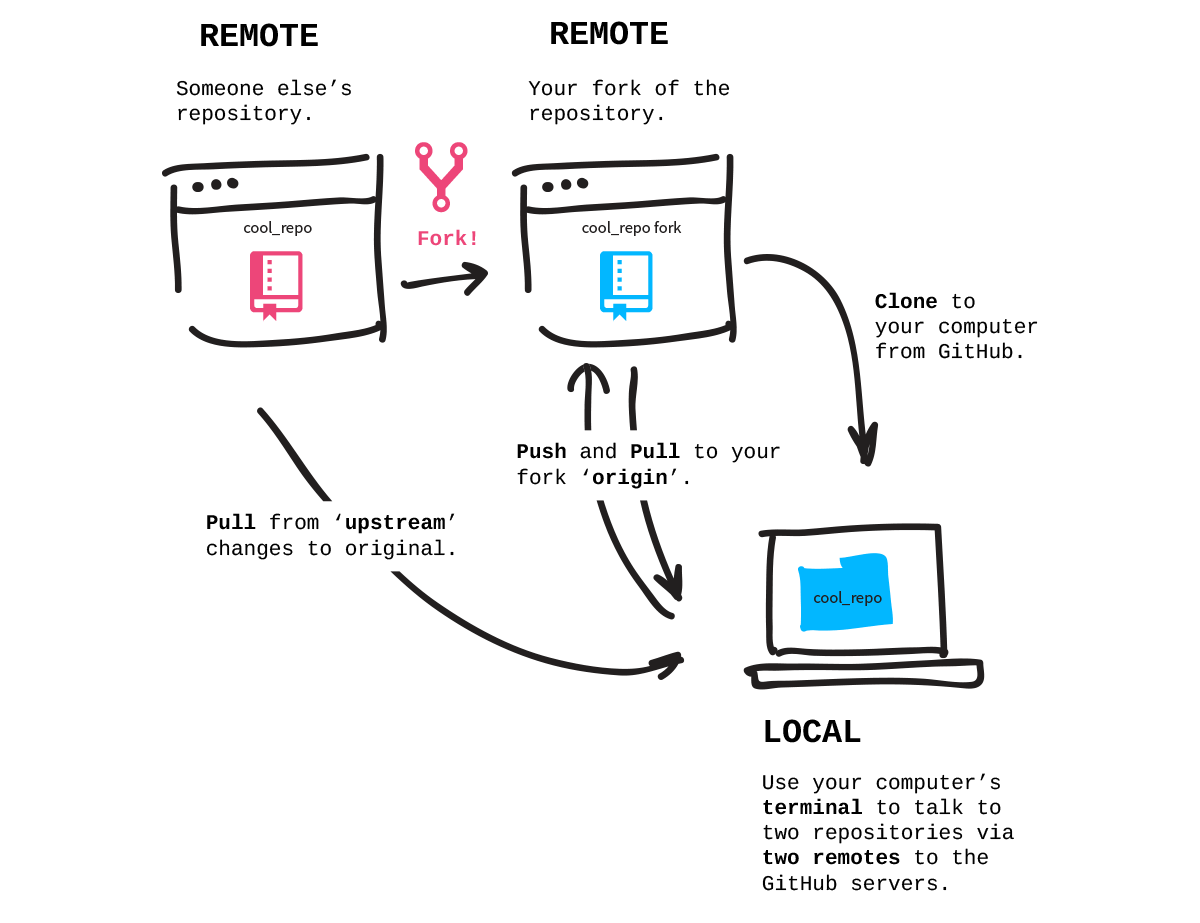
git push <REMOTENAME> <BRANCH>

## Forks

So far you have made a project locally and pushed it to GitHub, but that's only half the fun! The other half is working with other people and other projects.

When you **fork** a repository, you're creating a copy of it on your GitHub account. Your forked copy begins its life as a **remote** repository—it exists just on your GitHub account, not on your computer. Forks are used for creating your own version of a project (this diversion from the original is like taking a fork in the road) or contributing back your changes (such as bug fixes or new features) to the original project.

To get a forked repository from your GitHub account onto your computer you**clone** it. This cloning action copies the remote repository onto your computer so that you can work on it locally.



### Fork the Patchwork Repository

The project we'll work with is at [github.com/jlord/patchwork](http://github.com/jlord/patchwork). Go to that page and click the 'Fork' button at the top right. Once the forking animation is complete, you have a copy on your account. Copy your fork's HTTP URL from the address bar in your browser, this is the address of your fork on GitHub's servers.

### Clone Your Fork Locally

Now, in terminal, you'll clone the repository onto your computer. It will automatically create a new folder for the repository so there is no need to create one yourself. But make sure you aren't cloning it inside of another Git repository folder!

So be sure you're not inside of the 'hello-world' directory from the previous challenges. To back out and leave this folder so that you can clone, follow these steps:

Back out of your 'hello-world' folder:  
Tip: the two dots mean step out of a directory one level

cd ..

Now that you're no longer in another Git repository, clone your fork:

git clone <URLFROMGITHUB>

Go into the folder it created for your local copy of the fork (in this case, named 'patchwork').

cd patchwork

Now you've got a copy of the repository on your computer and it is automatically connected to the remote repository (your forked copy) on your GitHub account. Type git remote -v to see that the address to the fork is already set up.

### Connect to the Original Repository

What if the original repository you forked happens to change? You'll want to be able to **pull** in those changes too. So let's add another remote connection, this time to the original, [github.com/jlord/patchwork](http://github.com/jlord/patchwork), repository with its URL.

You can name this remote connection anything you want, but typically people use the name 'upstream'; let's use that for this.

git remote add upstream https://github.com/jlord/patchwork.git

To be sure you have the correct remotes set up, type git remote -v to list out the addresses you have stored. **You should have an 'origin' remote with your fork's address and then an 'upstream' remote with the address to the original, the URL noted above in this step.**

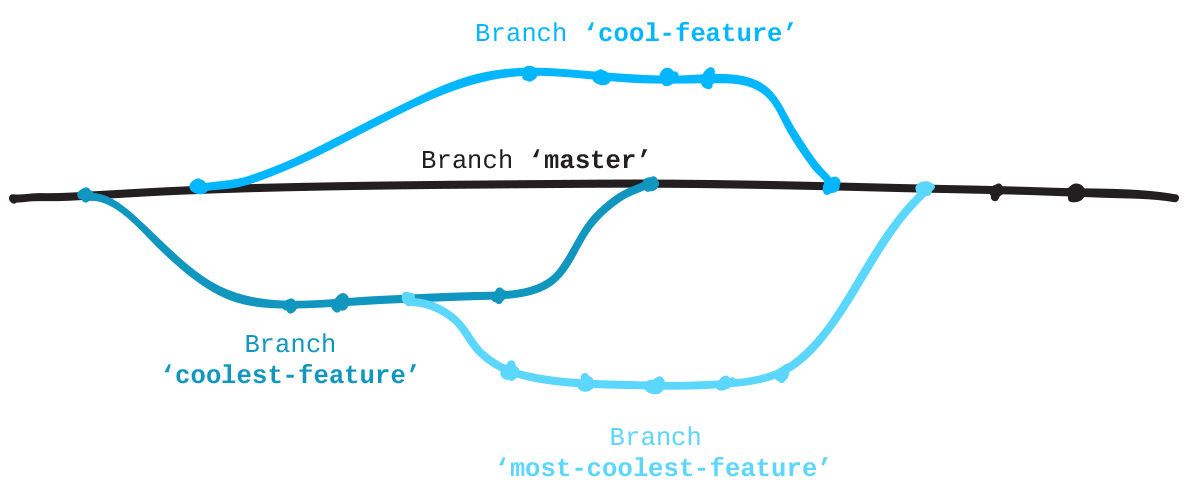
SELECT DIRECTORY VERIFY

* **Add a remote**
* git remote add <REMOTENAME> <URL>
* **Change a remote URL**
* git remote set-url <REMOTENAME> <URL>
* **View remote connections**
* git remote -v

## Branches

Git repositories use branches to isolate work when needed. It's common practice when working on a project or with others on a project to create a **branch** to keep your working changes in. This way you can do your work while the main, commonly named 'master', branch stays stable. When the work on your branch is finished you merge it back into the 'master' master branch.

The diagram below shows how you can branch off of your 'master' branch, do work and the merge those changes back into 'master'. You can even branch off of a branch if you need to, the 'master' branch doesn't have to be the base.



For a great visualization on how branches work in a project, see this GitHub Guide:[guides.github.com/overviews/flow](http://guides.github.com/overviews/flow/)

## GitHub Pages

GitHub will automatically serve and host static website files in branches named 'gh-pages'. This free service is called [GitHub Pages](http://pages.github.com). Since the project you forked creates a website, its main branch is named 'gh-pages' instead of 'master'. All repositories that have a 'gh-pages' branch with website files can be found, live online, using this pattern for the URL:

http://githubusername.github.io/repositoryname

### Create a branch

When you create a branch, Git copies everything from the current branch you're on and places it in the branch you've requested be made.

While still inside of your local 'patchwork' repository, type git status to see what branch you're currently on. Git should reply that you're on the 'gh-pages' branch.

Now create a new branch and name it "add-<username>", where 'username' is your username. For instance, "add-jlord". **Branches are case-sensitive so name your branch exactly the way your GitHub name appears**.

git branch <BRANCHNAME>

Now you have a branch with a new name that is identical to 'gh-pages'.

To go into that branch and work on it you **checkout** a branch. Go on your new branch:

git checkout <BRANCHNAME>

### Step: Create a new file

Back in your text editor:

* Create a new file named "add-<USERNAME>.txt", where 'username' is your username. For instance, "add-jlord.txt".
* Then, just write your GitHub username in it, that's it and that's all. For instance, I'd type 'jlord'.
* Save this file in the 'contributors' folder in Patchwork:**Patchwork/contributors/add-yourusername.txt**
* Next, check in your changes (see below).

### Check-in

Go through the steps for checking in a project:

git status

git add <contributors/FILENAME>

git commit -m "commit message"

Now push your update to **your fork**, 'origin', on GitHub:

git push origin <BRANCHNAME>

SELECT DIRECTORY VERIFY

#### Permission denied...error: 403

You are pushing changes to a repository you don't have write access to. In this case, you're likely pushing ot the original 'jlord/patchwork'. Make sure that you're pushing to 'origin' and that it points to your fork's address on GitHub. To check and see what your remotes are and where they point run git remote -v. You should have 'upstream' pointing to 'jlord/patchwork' and 'origin' pointing to 'yourusername/patchwork'.

To fix a remote that is pointing to the wrong place you can re-set its url:git remote set-url origin .

#### Authentication failed...error: 401

Your identity could not be verified. You may have 2FA (Two Factor Authentication) turned on in which case you must use a personal access token as your password. You can generate one by [following these instructions](https://help.github.com/articles/creating-an-access-token-for-command-line-use). You'll need to keep this and use it in place of your password when prompted. You can also [save this access token](https://help.github.com/articles/caching-your-github-password-in-git/) on your computer so that you don't have to re-enter it.

#### File NOT in contributors folder

The file you create should be placed inside the existing 'contributors' folder in the Patchwork repository. If you put it somewhere else, simply use Finder or Windows Explorer to move your file into the folder. You can check git status again and you'll find it sees your changes. Stage and then commit "all" (-A) of these changes (additions and deletions) with the commands below.

git add -A

git commit -m "move file into contributors folder"

#### Branch name expected: \_\_\_\_\_

The branch name should match your user name exactly. To change your branch name:

git branch -m <NEWBRANCHNAME>

When you've made your updates, verify again!

* **You can create and switch to a branch in one line**
* git checkout -b <BRANCHNAME>
* **Create a new branch**
* git branch <BRANCHNAME>
* **Move onto a branch**
* git checkout <BRANCHNAME>
* **List the branches**
* git branch
* **Rename a branch you're currently on**
* git branch -m <NEWBRANCHNAME>
* **Verify what branch you're working on**
* git status