Research Fellows - Session Three

Version Control – Git, Github, Conda Environments

Summer 2022

Northwestern | Kellogg

Version Control without Git

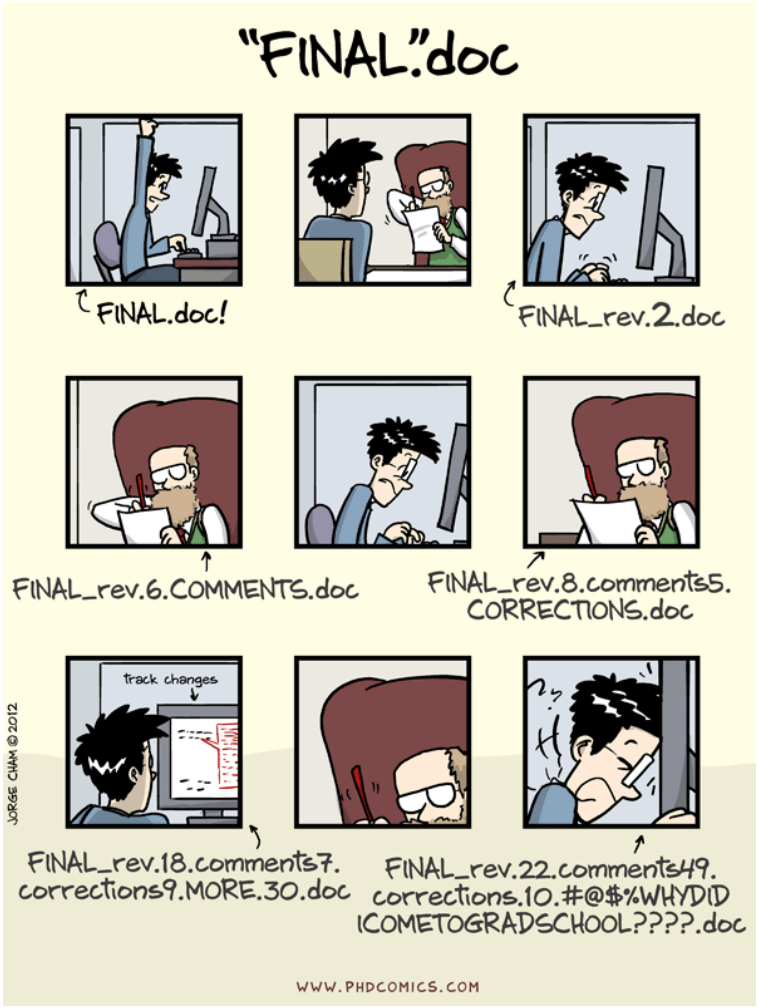


Image taken from: http://phdcomics.com/comics/archive_print.php?comicid=1531

Today we will Cover

- The difference between Git and Github
- Using Git locally as a version control tool on KLC
- Syncing your local Git directory to a Github repository
- Discussion: Using Git and Github Smartly
- Creating conda environments on KLC

Git vs. Github

Git vs. GitHub

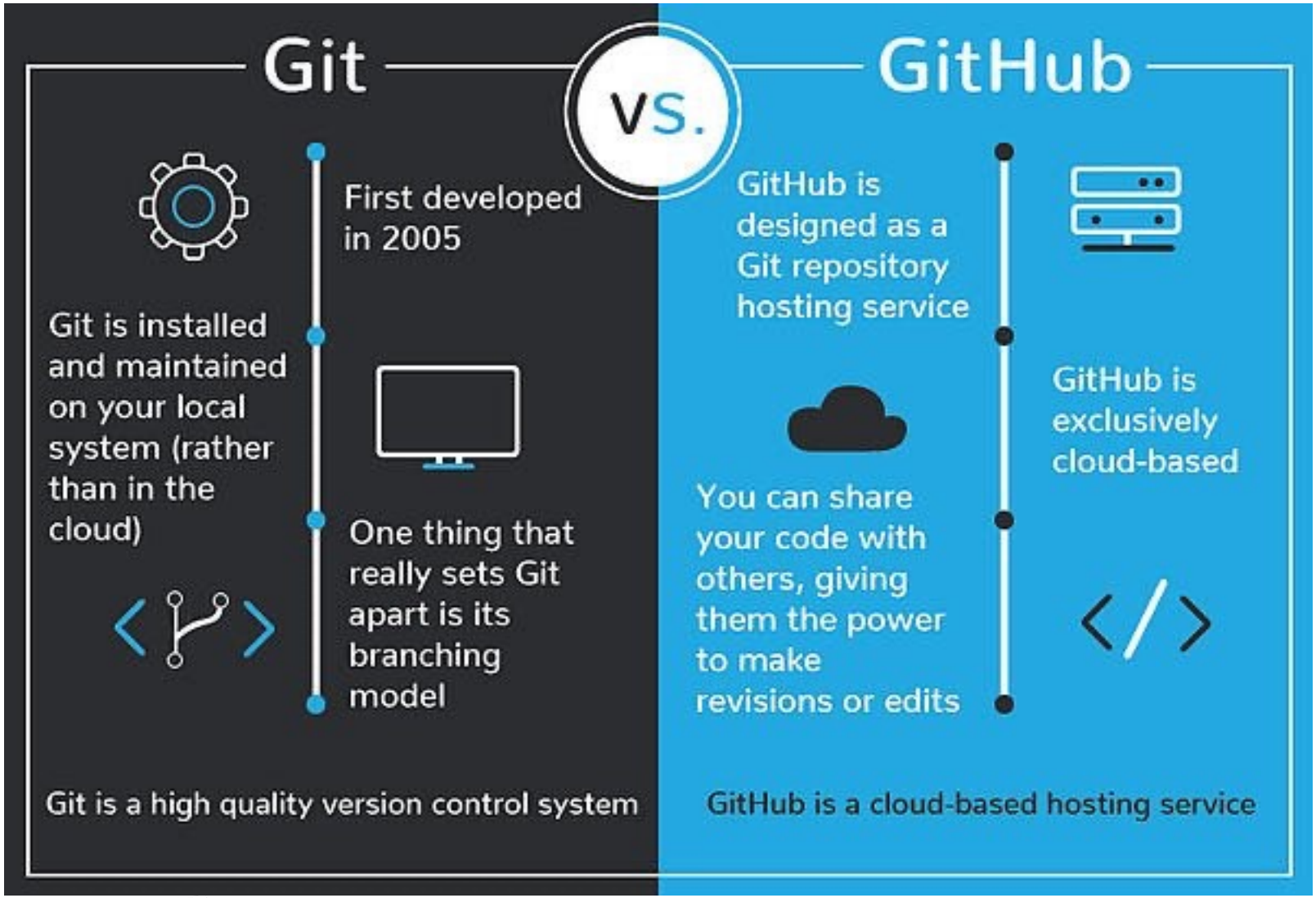


Image taken from: https://blog.devmountain.com/git-vs-github-whats-the-difference/

Git and Github Overview

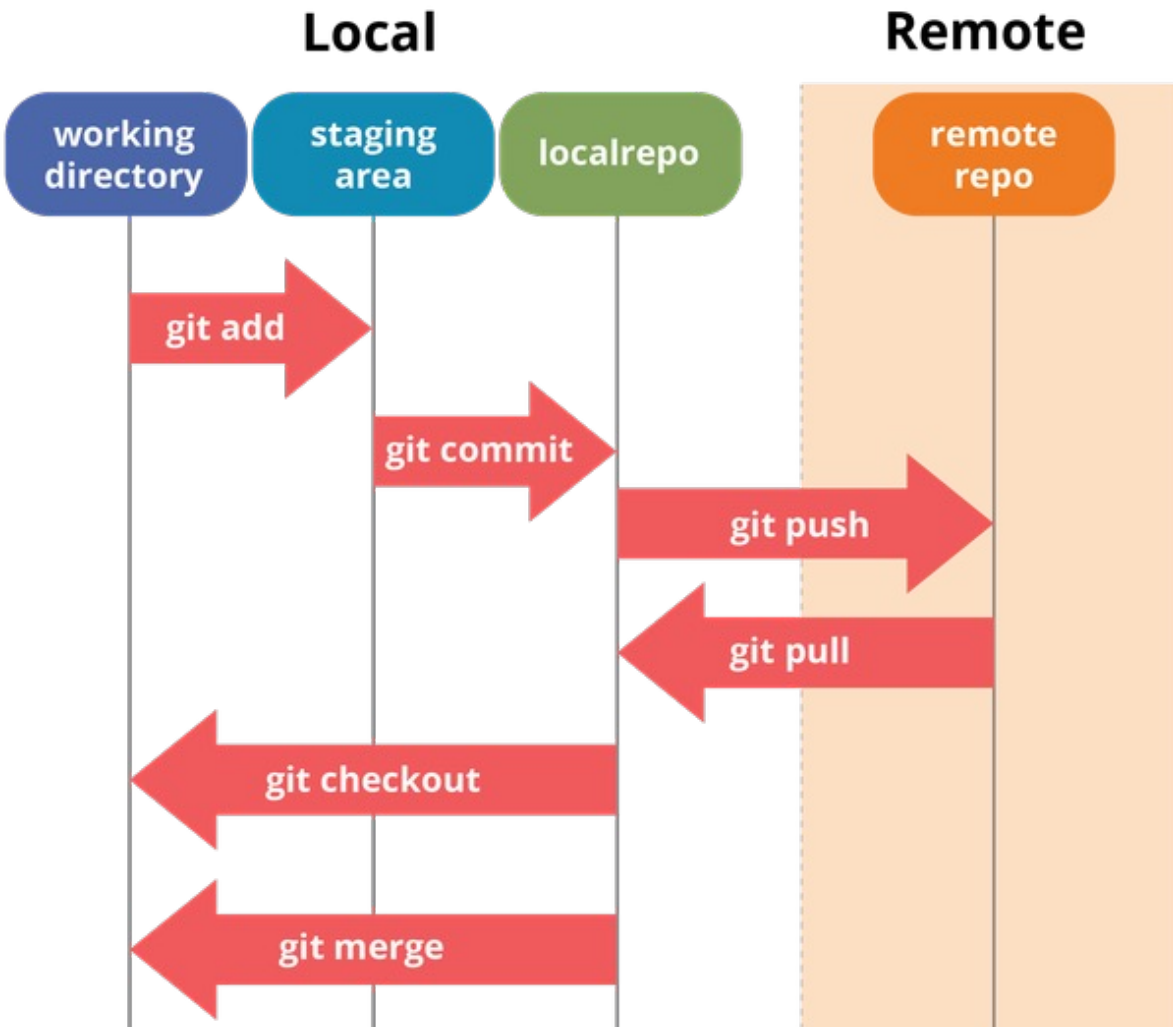


Image taken from: https://dev.to/mollynem/git-github--workflow-fundamentals-5496

Using Git on KLC

Loading git on KLC

Recall that no modules are preloaded in a new KLC session. You will need to load everything you use.

To see what version of a software package are available type:

```
module load git
```

The first time you load git on KLC, you need to set the username and email it will use when it records your commits. Note that this will have no bearing on your Github login later on.

```
git config --global user.name "John Doe"
git config --global user.email johndoe@northwestern.edu
git config --list
```

Resources:
NU IT page: https://kb.northwestern.edu/page.php?id=78598
NU IT github: https://github.com/nuitrcs/intermediate-git-workshop

Cloning a repo and initializing a local repo

A Git **repository** (”repo”) is essentially a **container** for your Git code.
You can create a Git repo on KLC from scratch or you can download an existing public Github repo and work on the code locally on KLC.

Let’s first clone a public Github repo to your home directory:

```
git clone https://github.com/rs-kellogg/fellows_workshop_2022
```

It should download a directory called: `fellows_workshop_2022.`

Create a subfolder called `practice` within the Week 3 directory

```
cd fellows_workshop_2022/Session3_Version_Control/
mkdir practice
cd practice
```

The first step to start version control with Git is to initialize your (empty) directory:

```
git init
```

Make Changes to your repo

Note that when you initialize a repo a new hidden folder will be saved here:

```
ls -a
```

If you type the following, you notice there is no activity in your Git repo:

```
git status
```

Let's make some basic changes to the Git repo we initialized. For instance, let's start by moving in all the FEC files we cloned from Github.

```
mv ~/fellows_workshop_2022/Session3_Version_Control/fec*
~/fellows_workshop_2022/Session3_Version_Control/practice/
```

Now running `git status` will show untracked changes.

Saving Changes to a local git repo

Since a Git repo consists of two spaces: a “staging area” and the actual “repository”, saving changes requires two steps:
1.) an “add” and 2.) a “commit”.

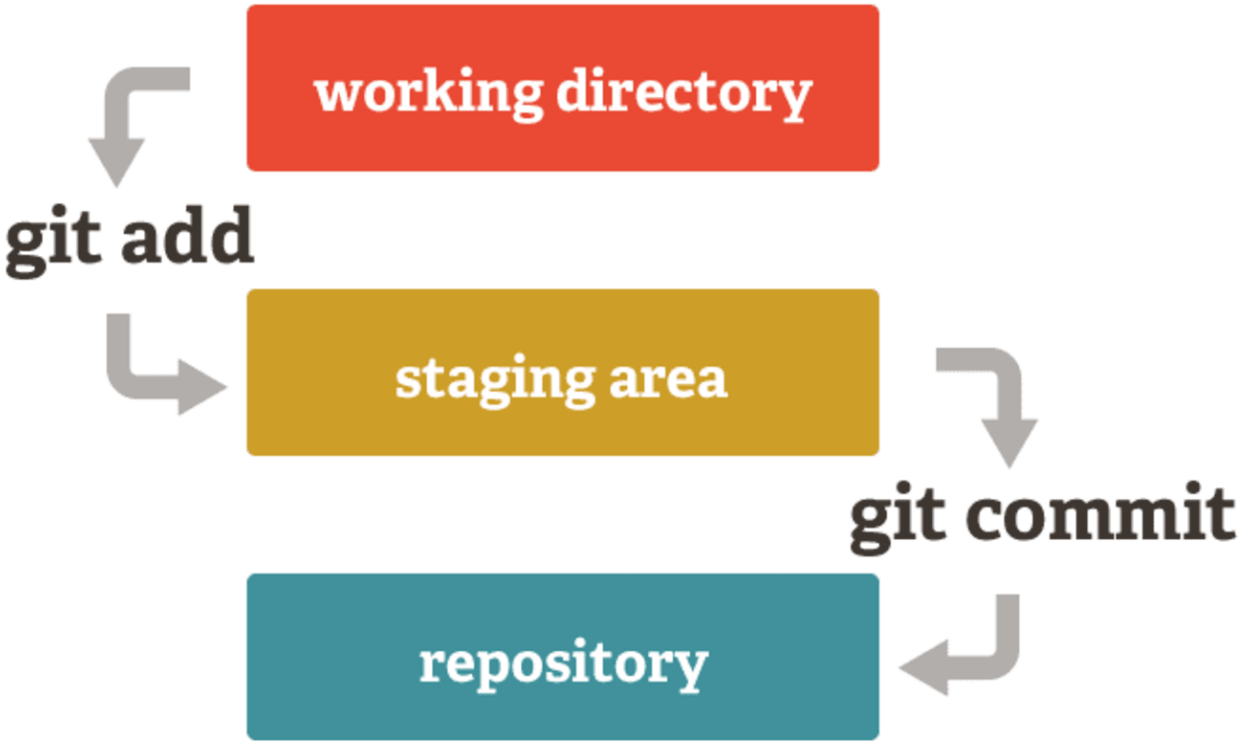


Image taken from:
https://medium.com/tech-and-the-city/changing-a-super-old-git-commit-history-20346f709ca9

Saving Changes to a local git repo

We can add the files we moved to the staging area with the following:

```
git add fec_all.sh fec_cron.sh fec_extract.py fec_process.R
```

OR

```
git add .
```

We can move the files from staging area to the local Git `practice` repo with:

```
git commit -m "Add all FEC files to repo”
```

After making these changes, we can retrieve a log:

```
git log
```

Retrieving a Previous Version

If we go back to our `git log`, there is an alpha-numeric sequence associated with each commit. These are commit-hashes.

To retrieve a previous version of a document or set of documents, use the commit-hash:

```
git checkout <commit hash>
```

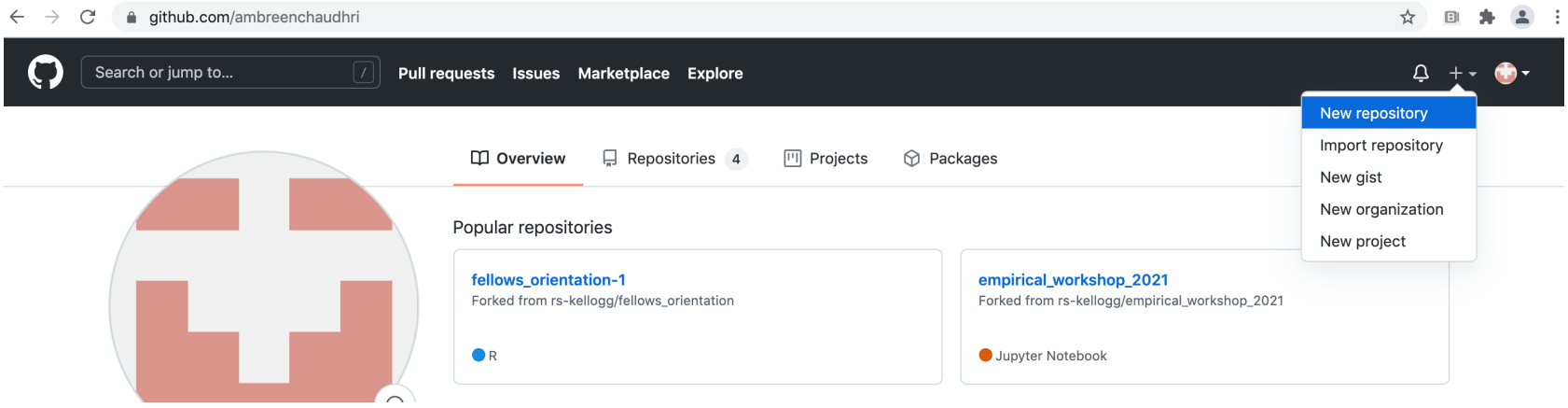
Files in a git repo are saved in **branches**. Branches are essentially timelines on Git. For now, we have been working in the main branch called “master”. You can return to the main timeline with:

```
git checkout master
```

Syncing local Git directory to Github repo

Creating and accessing a Github Repo

We can go to a Github account to create a new private repo to save our local changes.



Follow the steps to create a repo and you will receive a page with instructions to access it.

Accessing a github repo on KLC

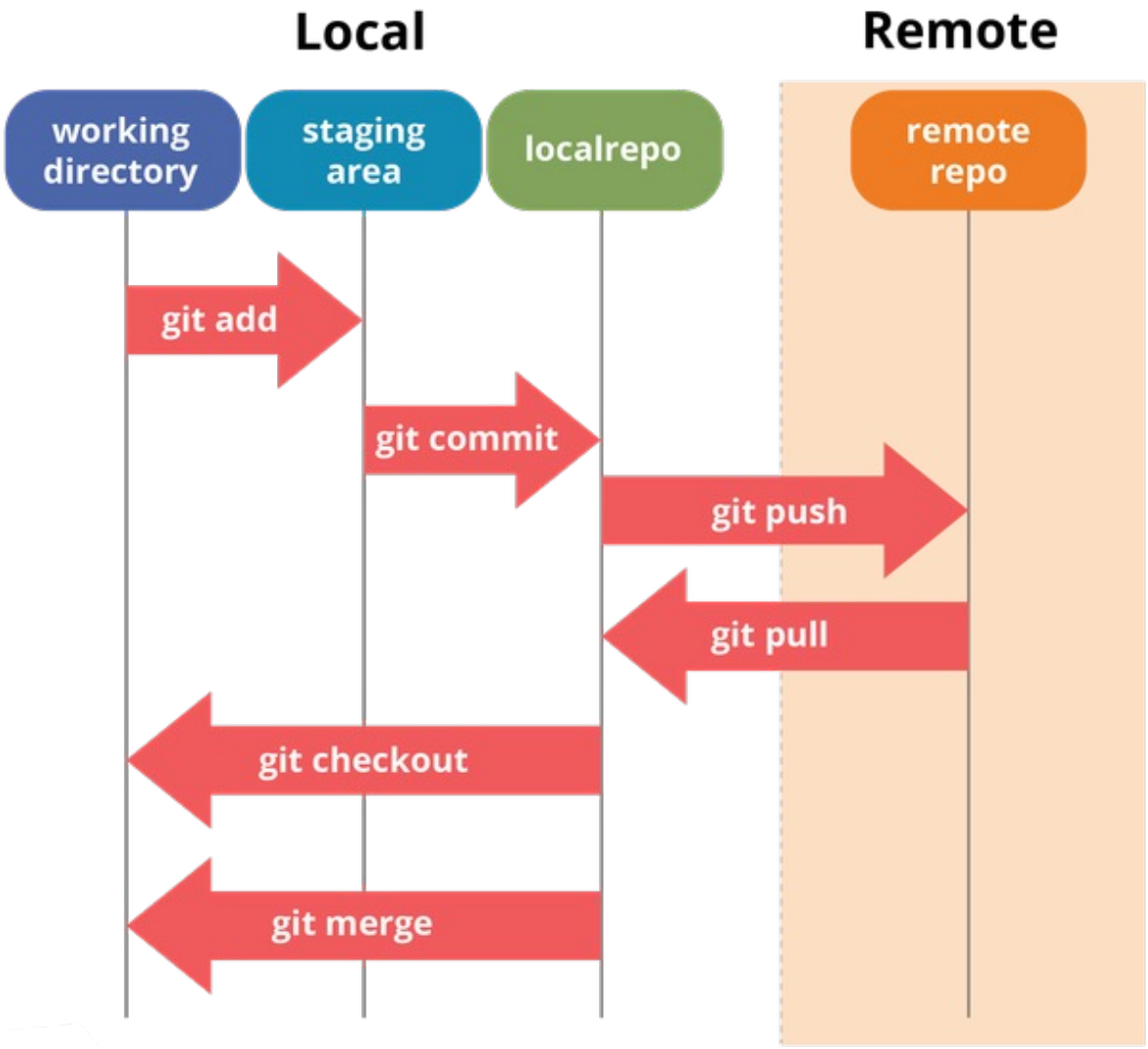
The instructions should list how to add a ”remote” to your Github repo, which is equivalent to a label for your Github url.

```
git remote add origin https://github.com/<user>/git_practice.git
```

Note that “origin” is just a generic name. You can use any name you like. The following command will show you the remote that is set.

```
git remote -v
```

Saving Changes to a Github repo



Saving local Git repo to Github

To push the changes in your local repo to Github, do the following:

```
git branch –M main
git push –u origin main
```

The push request will require you to sign into your Github account on KLC. In order to do so you will need a **personal access token**. You can acquire one by going to:

Settings → Developer Settings → Personal Access Token → Generate New Token

The steps are described here: https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token

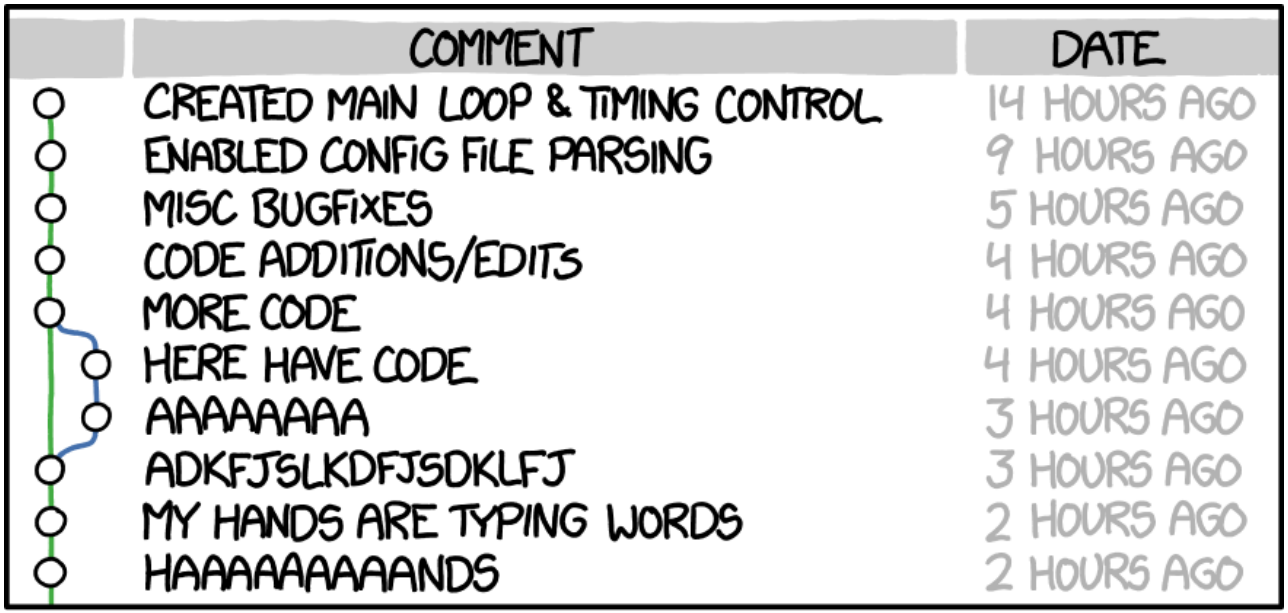
Saving Github changes to local repo

If you manually add files or make other changes on Github, you can sync those changes to your local git repo on KLC by doing the following:

```
git pull origin main
```

Discussion: Using Git Smartly

Discussion: Using Git and Github Smartly



Discussion: Using Git and Github Smartly

Git tools are not a substitute for good documentation. You are still responsible for making decisions about whether the code is tested, what version is saved, how descriptive the comments are. Placing bad code in Git/Github is still bad code!

Creating Conda Environments on KLC

Creating a Conda Environment

Instead of loading each module you would like to use in your shell script separately, you can also create conda environment. Follow the steps below only once to create the environment:

```
module load python-miniconda3/4.12.0
conda create -n automate_env r-essentials r-base
source activate automate_env
conda install r-fs
conda install pandas
```

To leave the environment:

```
source deactivate automate_env
```

In the future, you will only need to activate the environment to load all modules and libraries.

```
source activate automate_env
```

Conda Environments Using a .yml file

The basic commands for creating a conda environment only allow you to install one package at a time. A .yml file allows you to install multiple packages while accounting for dependencies between libraries. Using a text editor, create the following file on KLC called `automate_env.yml`.

```
# Create automate_env.yml file
# begin automate_env.yml
name: automate_env
channels:
  - defaults
dependencies:
  - pandas=1.4.3
  - r-base=3.6.1
  - r-essentials=3.6.0
# end automate_env.yml
```

To run this file from the command line, type:

```
conda env create -f automate_env.yml
```

Appendix

Appendix – Branching on Git

A branch is an independent line of development. You could think of a branch as a timeline of a project that could be created in parallel with the main branch. To see a list of branches a project contains, type:

```
git branch
```

If you'd like to make an experimental branch of a project and start working on it, type the following:

```
git branch experimental
git checkout experimental
```

The changes you make in a new branch will remain independent of the master branch you are working on. Once you are comfortable with these changes, you can merge them back to the master branch by doing the following:

```
git checkout master
git merge experimental
```

Appendix – Forking a Github Repo

Much like cloning, forking enables a user to copy all the contents of a Github repository. However, forking a repo also enables a user to suggest and possibly implement changes to the original repository with the creator's approval. Forking is a useful tool to collaborate with others to improve open-source code.

There is no command line tool in the Git module on KLC to fork a repository. Instead, navigate to the main page of a repository. On the right side, select the "Fork" button. The selected repository will be copied to your list of Github repos.