University of Central Missouri

Upsilon Pi Epsilon Git Workshop Script

0. Prep

You should have Git on your machine installed via the instructions we sent out. You also need to have an account on Github.com.

On Windows, open either the Windows Terminal application (if you specified installing the Git Bash modifications to Terminal when you installed Git) and open a Git Bash tab, or you can open the standalone Git Bash application.

On Mac, open Terminal application.

On Linux, open a Terminal window.

1. Help

Git has a built in help mechanism, as well as guides for specific topics.

```
$ git help                      # General help
$ git help --guide              # Common Git guides
```

2. Configuration

Get help on configuration. It will open in a browser window.

```
$ git help config               # See FILES section
```

Show the current configuration for all projects, which is done by using the --global option. Without the global option, you will be changing options for the current git repo only.

```
$ git config --global --list    # ~/.gitconfig
```

Check specific config values for your user name and email.

```
$ git config user.name
$ git config user.email
```

Set your user name that Git will use when you do a commit.

```
$ git config --global user.name "Joe Cool"
```

Set your user email that Git will use when you do a commit.

```
$ git config --global user.email joe.cool@example.com
```

Set default Git branch to `main`. The default branch after you install git is `master`, but almost everyone uses main instead now.

```
$ git config --global init.defaultBranch main
```

Open the global config in an editor, note default editor is vi!

```
$ git config --global --edit      # Open config in editor
```

Set editor to Visual Studio Code: (assumes command line for VSCode enabled)

```
$ git config --global core.editor "code --wait"
```

3. Git 101

Change to file system location of your choice
...so we can make directory for today
Note, Mac Terminal will open to your home directory which is located in the directory /Users/username. Note, Windows Git Bash will open to your home directory C:\Users\username.

```
$ mkdir git-hands-on
$ cd git-hands-on
```

Let's create a new repository

```
$ mkdir git101
$ cd git101
$ git init
```

Can also create directory and init with one command "$ git init git101"

Let's see what we have - We can list a directory's contents in Unix with 'ls' command. This is going to show an empty directory as there are no non-hidden files

```
$ ls
```

However, the local git repository data is in the hidden directory .git, so let's take a look at what is in it:

```
$ ls .git
```

Of particular note:

```
- .git/objects/              (empty)
- .git/refs/heads/           (empty)
- .git/config                `git config -e`
- .git/HEAD                  ref: refs/heads/master
```

What does Git think it has? We can find out using the status command:

```
$ git status
```

Let's commit something! We are going to create an empty README file

```
$ touch README.md                 # Create README.md
$ git status                      # Untracked files!
```

Stage the README file we have created

```
$ git add README.md               # Track the README.md file
$ git status                      # Changes!
```

Commit the README file

```
$ git commit -m "Add empty README"
```

Let's look at our Git history now that we have done a committ.

```
$ git log
```

4. Fork a Github Repo

We are going to use a UCM Organization Github repo for this workshop.
However, since all of you will not have write permissions on this repo,
we are going to have you Fork the repo into your own Github account.

Make sure you are logged in to your Github account.

In your web browser go to:
https://github.com/ucmo-cs/git-hands-on.git

In the upper right hand area, click on the Fork Button
On the page that loads, check that your user account is correct then,
Click on the Create Fork button.

You should now have a forked repo, and you can go to your version of the repo:
https://github.com/{your_github_username}/git-hands-on.git

4. Clone a Remote Repository

We will now get a local copy of the repo we forked to your account.

```
$ cd ..   # Should be in git-hands-on directory
$ git clone https://github.com/{your_github_username}/git-hands-on.git
```

Let's see what we have by listing the directory:

```
$ cd git-hands-on
$ ls
```

What does Git think it has?

```
$ git status
```

What local branches do we have?

```
$ git branch
```

We can list the remote repos known to your local repo with the 'git remote -v' command (The -v means verbose). We can list the remote branches with the 'git branch -r' command (The -r means remote).

```
$ git remote -v
$ git branch -r
```

What is 'origin/main'? Origin means it comes from the remote repo name origin, and main is the name of the branch in that repo.

Now, lets look at the Git history of this repo, displayed in two different formats.

```
$ git log
$ git log --oneline --graph
```

5. Push to a Remote Repository

Change the test.txt file to be:

```
This is the first line.
My second line.
```

Then lets stage our changes in Git:

```
$ git add test.txt
$ git status
```

What if we change our minds before we commit? We can unstage it....

```
$ git restore --staged test.txt
$ git status
```

And then if we want, we can remove the changes also....

```
$ git restore test.txt
$ cat test.txt
```

Change the test.txt file to again be:

```
This is the first line.
My second line.
```

Stage and commit the changes this time

```
$ git add
$ git commit -m "Add second line to test.txt"
```

Look at what Git thinks it has now

```
$ git status                      # Branch is ahead
```

Look at the history

```
$ git log
$ git log --oneline --graph
```

Let's share our changes to test.txt with the remote repository.

```
$ git push
```

Check on web interface that changes are there.

We can go back to any point in our history with checkout

```
$ git log
$ git checkout {hash number of commit one before latest}
```

Look at the file now, it should have contents before our last change:

```
$ cat test.txt
```

Note we can achieve the above of backing up one commit with the following, which means one commit before the head of the current branch. Note, now we don't even have a test.txt file.

```
$ git checkout HEAD~1
$ ls
```

Go back to the latest commit on the current branch main.

```
$ git checkout main
$ cat test.txt
```

6. Working Locally with Branches

Start work on feature 1, we are going to create a new branch for it with the name feature1, and checkout that branch.

```
$ git branch feature1             # Create branch from here
$ git switch feature1           # Switch to different branch
$ git branch
```

Check we are on correct branch, then create an empty file, stage and commit it.

```
$ git status
$ touch feature1.md
$ git add feture1.md
$ git commit –m "Add feature 1"
```

Check that the file exists in this new branch:

```
$ ls
```

Switch back to main and look at files in repo

```
$ git switch main
$ ls
```

Start work on feature 2

```
$ git branch feature2             # Create branch from here
$ git switch feature2           # Switch to different branch
$ git branch
```

Edit test.txt for the feature to be like:

```
This is the first line in feature2.
My second line.
The third line.
```

Use git diff to look at the changes that haven't been committed

```
$ git diff test.txt
```

Commit the changes for feature2

```
$ git add test.txt
$ git status
$ git commit –m "Edit test.txt for feature2"
```

Check on the diffs between two branches

```
$ git diff feature1..feature2
```

7. Working Remotely with Branches

First get latest commits from GitHub, other people may have made changes

```
$ git fetch --all
```

Now let's share some code, our feature1.

```
$ git push origin feature1
```

If we look on the Github web interface now, we should see the two new branches.

Now let's pretend someone else is working on your fork on GitHub:

1. Use **Branch** dropdown to switch to feature1
2. Click on `test.txt`, then the pencil to edit
3. Change something and commit changes directly to `feature1`

Nothing changes locally until we explicitly fetch:

```
$ git branch -r
$ git fetch --all
$ cat test.txt
```

We should also bring in changes from feature1 branch.

```
$ git switch feature1
$ git status                # behind!
$ git pull origin feature1  # get latest & merge (no conflicts only 1 side changed)
$ cat test.txt
```

8. Merging

Assume we are done with implementing feature2
We want to merge our code back into the main branch so that we can release it.

First though, while we were working on feature2, someone else changed the main branch.

```
$ git switch main
```

Let's edit test.txt in the main branch to simulate that:

```
This is the first line in main.
My second line.
Good things come to those who main.
```

And commit those changes. Note, to do the add and commit all in one step, we can use the -a flag to git commit to tell it to add all changed files, then do the commit:

```
$ git commit -a -m "Local changes to main"
```

Now let's try to merge the branch locally.

```
$ git switch main      # We always switch to the branch we want to merge INTO
$ git merge --no-ff feature2   # And we specify the branch we want to merge FROM
```

That failed and left us with the merge in process.
Git tried to combine our versions, but both branches modified the same files,
and more specifically the same lines, so it couldn't do it automagically, it needs our help.

Edit test.txt to resolve the merge conflict.
When done, add the file and complete the merge

```
$ git add test.txt
$ git commit -m "Merged feature2 into main"
$ git log
```

9. Next Steps

This workshop has given you a basic intro to Git and it's usage. The best way to learn more Git is to use it on your own projects and homework. The official Git book, which you can get in ebook form here: https://git-scm.com/book/en/v2, is a great resource for answering the questions that you may have as you explore Git!!!