MACAULAY2 - RTG SEMINAR

OCTOBER 16TH, 2023

We'll learn about git today.

`git` is a piece of software for tracking changes in code (including LaTeX) or files more generally. The way git works is that you make a copy of all the code (by `clone`ing a repository). After you make changes you `commit` them to your local repository (this only changes your code, but it does let you easily roll back changes you made locally). Once you want to share your changes, you `push` them. To get other people's changes you `pull` them.

Make sure you have git installed on your local machine. Open a terminal and run `git`. If you get something like "command not found", go download and install git. Git is probably installed by default on linux, and mac. If you are running windows, you might only want it on the linux side anyways.

Step 1 (Make an account on github).

If you don't already have an account, go to

`https://github.com/`

and sign up for an account.

Let me know your username and I can add you to our temporary working repository that we'll be playing with today.

After that's done, see if you can view:

`https://github.com/kschwede/GitExamples`

You should see something if you are logged in correctly and you have been added to the project.

Step 2 (Clone a repository)

Depending on how you are accessing git, there are different ways to do this. Lots of people interact via the command line, but there are lots of guis too, and your favorite editors may have git support (ie, `emacs`, `vscode`, `vim` etc). I will be showing how `vscode` interacts with git a bit later. For now though, we will use the command line.

From a terminal, run:

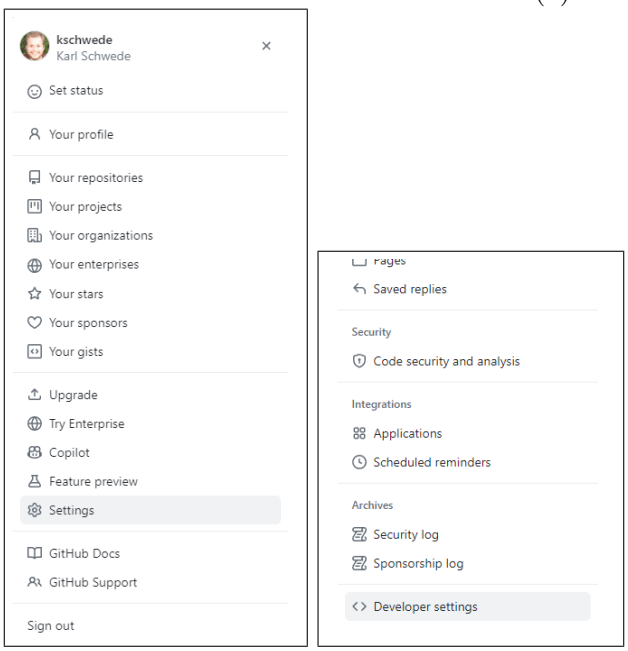
```
git clone https://github.com/kschwede/GitExamples.git
```

You can actually grab the url by clicking the green "¡¿ Code" button on the github page for todays project (above).

At this point, you will probably be asked for a username and password (depending on how things are configured). Your username is just the email associated to your github account. However, *github has stopped using passwords.* Instead, you may need to generate a key (depending on your setup, this isn't the best way to do this necessarily, but it should work on all platforms).

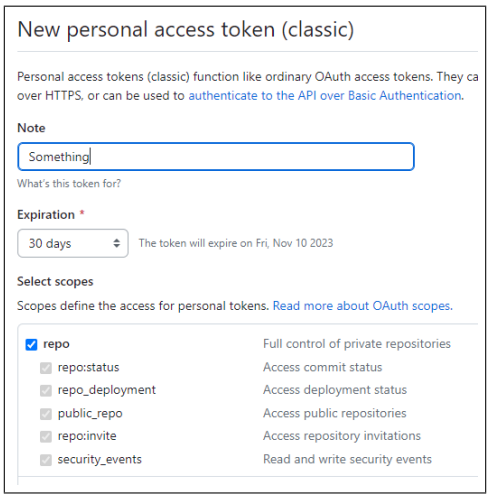
(1) Click on your account in github, and navigate to "Settings".
(2) Click on "Developer Options".
(3) Click on "Personal access tokens" (and then "Tokens (classic)").

FIGURE 1. Where to click for (1) and (2)



(4) Generate a new token (classic).
(5) Call it something, and select "repo" scope.
(6) Scroll to the bottom and generate the token.
(7) Copy the text and paste it as your password.

FIGURE 2. Where to click for (5)



If all goes according to plan, it should automatically create a folder called `GitExamples`. There should be several files in it.

Make sure to *SAVE* your personal access token (feel free to delete it later). Note in many terminals, simply right-clicking will paste.

Step 3 (Making changes)

Navigate to your new folder.

```
cd GitExamples
```

Before doing anything else, let's change some settings for this repository. Run the following:

```
git config user.name "Your name"
git config user.email "email.address@whatever.com"
git config pull.rebase false
git config core.editor "emacs"
```

To make these changes global (for all repositories), add `--global` after `config` in each command. You might not want to set your editor to be emacs. Some other choices might be: `code, vim, nano, gedit` etc. If you want some help deciding, ask me.

Create a new file (use any editor your want, or create a blank file with `touch YourFileName`). Choose a file name no one else will pick.

Run the command

```
git add YourFileName
```

This adds your file to your local repository. Next you want to commit your changes.

```
git commit -a
```

This should open up the editor you selected above. In that editor write a *short* message about what you did. Then save the file and close your editor. You should have committed your first change. But this is only to your *local* repository. To share the change with everyone else run:

```
git pull
git push
```

Assuming you didn't choose the same file name as anyone else, that should just work.

Go see if you can view your file on github.com. You can probably see other people's files too. Your local repository probably also got copies of them when you ran `git pull`.

Now, go edit your file again, add at least 3 or 4 lines with text, `commit -a` your changes (with a short message), and then `pull` and then `push`.

Tokens

It can be somewhat annoying to keep that token around and enter it whenever you pull/push. You can see some instructions of how to avoid this via the https protocol here:

`https://docs.github.com/en/get-started/getting-started-with-git/caching-your-github-credentials-in-git`

You can also control the timeout before it asks you for your token again with the command

```
git config credential.helper 'cache --timeout=900'
```

(that command sets it to 15 minutes instead of the default 10). Alternately you can use ssh repositories instead of https. (This is slightly more complicated to setup, but you can find how to do it via googling).

GUI's

There are various ways to interact with git without using the command line. Try running `gitk` from the command line. This can let you view the changes. If you'd rather `add` files, `commit` changes, and `pull` and `push` from a gui, you can try running `git gui`.

Many text editors also have git support (or have packages that add git support). `code, emacs, vim` etc. For example, try running

```
code .
```

from the terminal (in GitExamples folder). Then hit `ctrl-shift-g`.

Step 4 (Conflict resolution)

Ok, now you and a neighbor both choose a file to manipulate (one of the files one of you created). Make sure you are both working on the same version of the file by running `git pull`.

Merging without conflict

One of you make a change to one line of the file. The other make a change to another line of the file. Both `commit` your changes.

Then one person should pull/push. When the other runs pull and push, they will need to merge. Then pull/push again. Most likely, it will let you do this.

Now repeat the process, and let the other person `pull` second and do the merge.

Conflict

Sometimes though, you both modify the same line of the file. You can fix this. But let's explore how it looks. Ok, both of you should try modifying the same line of the file, you should both `commit`. Now have one person pull/push.

When the other pulls, you should get an error that looks like.

```
Auto-merging Test.txt
CONFLICT (content): Merge conflict in Test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

If you open the file again, you should see something like:

```
<<<<<<< HEAD
Your line.
=======
The other person's line.
>>>>>>> 95edbf691d891411cf4aa316f175a8e77901098a
```

If you open it in certain editors (vscode), it might automatically ask you which to pick. Otherwise, you can simply choose the version you like, delete the other, delete the `<<<... HEAD` and `>>>...`, commit the change.

Modify the same line again, switch roles and let the other person repair the conflict.

Reverting a commit

Sometimes you make a commit, it causes a conflict, and you just want to undo what you just did. Or you just created a huge bug and you just want to go back!

You can see your recent local commits by running

```
git log origin/main..HEAD
```

(or you can run `gitk`, or open in various other editors such as vscode or other GUIs). Regardless, generally speaking you'll see something like this.

```
commit 229ff9ee5198269d79eae3ed99ec167d9a4dce70 (HEAD -> main)
Author: Karl Schwede <kschwede@gmail.com>
```

You can then undo that commit by running (switching the code below for the appropriate one for your case)

```
git revert 229ff9ee5198269d79eae3ed99ec167d9a4dce70
```

If you are in the middle of a (failing) merge, you need to run

```
git merge --abort
```

first.

Alternately, if you have *ONLY* made mistakes locally (and not pushed them) you can use `git reset`. For example, you can run the following to revert the last change.

```
git reset --soft HEAD~1
```

Generally speaking, `git reset` can mess lots of people up. So be careful (maybe avoid this entirely).

Finally, if you made some changes you don't like, but haven't yet committed them, you can just run `git stash` to stash them (for instance, forget about them).

Step 5 (Make your own repository)

Make a repository. Play around with it.

Better yet, work with you group and make a repository for your group. Add all your group members to it.

There's lots of other things you can do with git. A big topic I didn't touch is branches. For larger projects with lots of collaborators, probably branches are the way to go. For us though, it's probably not necessary.