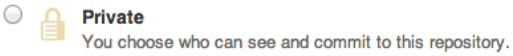
Create your own repository on github

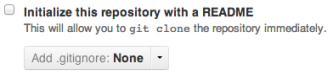
In your github browser window, click the "Create Repository" icon in the top-right part of the screen.

Make the repository name "`ASTR2600_assignments`". Add a description - e.g. 'Adam Ginsburg's homework for ASTR 2600'. Make sure you select the *Private* button.



If it asks you to pay, that means you haven't yet been hooked up with an educational account. Please re-do the first git tutorial. You can then complete the rest of this tutorial, but replace `ASTR2600_materials` with `ASTR2600_public` and use the public option here. You MUST complete this tutorial with a private account by next Wednesday, though!

Click the "Initialize this repository with a README" button.



Click "Create repository" [if it asked you to pay, this won't work]

If that goes well, you'll be shown your repository. It will contain a file `README.md` which contains the name of your repository and the description you entered before.

Before we go further, you need to deal with some security precautions.

Making an SSH key

SSH stands for "secure shell". It is a cryptographic network protocol, which means it's a way to communicate between computers securely so that no one can see what you send or receive.

When you're sending data to and receiving data from github, git uses the "ssh protocol" to protect your data.

For this process to work properly and easily, you need a way for the remote repository (the github website) to know that you, and not someone else pretending to be you, are asking for the data.

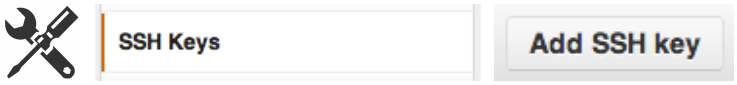
So, we're going to make a "public key" to send to github. Encryption is a very complicated topic on its own, so we're not going to discuss it in detail - this will just be a simple "how-to".

Run the command `ssh-keygen`. When it prompts you to "Enter file in which to save the key", press `enter` to accept the default. Press `enter` twice more to accept "no passphrase".

To see the contents of the public key, use the `cat` command:

```
$ cat ~/.ssh/id_rsa.pub
```

Now load up the github web page again. Go to "settings", then select the "SSH Keys" tab on the left side. Click "Add SSH Key" in the top right.



Copy and paste the text shown from running the `cat` command into the "Key" text box. You can use any title you'd like, but I recommend calling it "cosmos" since that's the name of the computer it represents.

Now you should be able to send/receive data from cosmos very easily.

git and cloning

We're going to make a copy of the Class Materials directory, which contains stuff that I have made and am handing out to you.

Back at your terminal, type the following command (note that you can copy & paste the URL from the web page that is currently open):

```
$ git clone git@github.com:ASTR2600s13/ASTR2600_class_materials.git ASTR2600_materials
```

- `git` is the command for the version control program. All commands using `git` will start with this.
- `clone` tells git that you want to 'clone', i.e. create an exact copy of, the repository
- `git@github.com:ASTR2600s13/ASTR2600_class_materials.git` is the full URL of the repository, with special weird syntax. Don't worry about it; I always copy & paste this stuff myself (but if `https` is in this URL, something's wrong).
- `ASTR2600_materials` is the name of the directory we have copied everything into

You may see a message like: `The authenticity of host 'github.com (207.97.227.239)' can't be established.` Answer "yes". Technically, you should make sure that it's right before saying yes, but unless someone is explicitly trying to attack our computer (cosmos) right now, it should be fine.

Now, `cd` into the `ASTR2600_materials/examples/` directory:
`$ cd ASTR2600_materials/examples/`

Do an `ls` to show the contents. You should see these files:
`README.md  coyote  snake.pro  spectrum.pro tvcircle.pro`

For kicks, open up IDL and do the following (I'll tell you exactly what it does later):

```
IDL> .r snake.pro
```

If all goes well, you should see an (extremely poor) animation of something vaguely resembling a snake. Next up are badgers and mushrooms (ignore this sentence).

So far, you have created an account for yourself and acquired some code that I wrote. This is a great way to share code, but it is only the tip of the iceberg.

Quit out of IDL before going to the next section.

Create your own local repository

This is to create a version control system for your own work. It will include "incremental history" and backups. That means that, if you use the version control system correctly, you will have a history of your work and all the changes you've made, AND you'll have a nice set of backups.

You will clone the remote repository you created above. The following command will create a directory with all of your materials in it. You must replace the text `YOURUSERNAME` with your username!

```
$ git clone git@github.com:YOURUSERNAME/ASTR2600_assignments.git
```

Now `cd` into this directory and `ls` to see the contents. It should look like this:

```
$ cd ASTR2600_assignments
$ ls
README.md
```

The `README.md` file is what shows up on the git web page when you go there. It's written in a language called markdown, which we will not cover.

The next step is to add some files to your repository. Make an `assignment0` directory:
`$ mkdir assignment0`

(`mkdir` is short for make directory)

Find the journal `.pro` file you made in the last class, and move it to the `ASTR2600_assignments/assignment0` directory. In the following command, you have to replace `/path/to` with the path to your file. That means, if your file is in your home directory (`~`), you could use the tilde:
`$ mv /path/to/YourName_01_14_2013.pro assignment0/`

We need to tell `git` to *track* the file now. That means that the software will keep track of any changes made to the file, so that you can undo them later. `$ git add assignment0/YourName_01_14_2013.pro`

We'll now "commit changes" to the repository. Up until now, you haven't created a backup of your files, but once you complete this command, they will all be stored & backed up in the local repository.
`$ git status` will report that you have one uncommitted change
`$ git commit`

This will pop up an editor called `nano` with all of the text you saw in `git status`, plus room at the top for you to enter a message. Do that! Your message should include a brief note about what you're adding (e.g., assignments, assignment numbers) and a statement that this is the first commit. In general, you want these notes to be long and verbose!

When you're done entering your message, save and quit out of `nano` or whichever editor you're using. To quit out of `nano`, press `control-x`. You'll get a long verbose message, but short story is your code has been added.

Finally, we want to back up our data remotely (just in case our local machine crashes, but also for sharing code). We *push* changes from our local repository to the remote one.
`$ git push`

Open up your repository on `github.com` and make sure your file is there.

Proof that you backed up your file

Is your file on github? If not, DO NOT CONTINUE!

We're going to demonstrate that, if you accidentally delete a file, all is not lost! But we'll do it by intentionally deleting a file.

Delete the journal file!

```
$ rm assignment0/YourName_01_14_2013.pro
$ ls assignment0
```

It's no longer there! (or at least, it shouldn't be)
How do we get it back? With git, you can "revert to the last commit state":

```
$ git checkout -f HEAD
$ ls assignment0
```

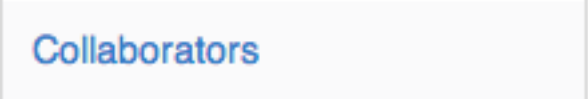
Now it's back! Hooray! `checkout -f` means "force a checkout of the state, even if it means throwing away local changes". That might not clarify anything, but the point is that your lost data becomes recoverable with git. So, no "but my computer crashed and now the file's gone!" excuses.

Add Cameron and Adam as collaborators

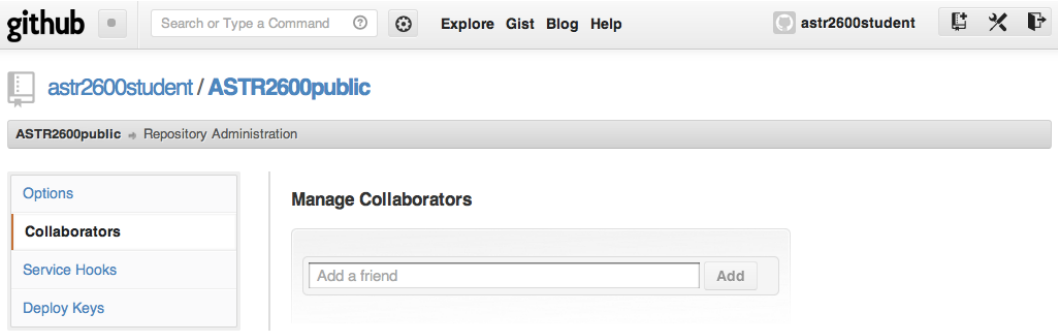
This part is necessary for us to see the work you turn in.

In your repository, click the settings button

Once in admin, click the Collaborators button



Then, add us as collaborators. The usernames are:
keflavich
cwedge90



What next?

From now on, any time you make changes to any of your files, you should use `git commit -a` to back up your files[1]. Then use `git push` to send your data to the remote repository.

You can - and should - browse around github to look at the source code as it is stored on the server. See if you can figure out how to look at `diff`s (we'll do it together at some point).

Any time you create a new file or folder, you will need to do a `git add` to add the new files, then do a `git commit` to store them.

[1] The `-a` is required to tell git to include your changes; it doesn't do that by default