DSApps @ TAU 2020: Installations

2020 Sem. B, Mon. 8-11

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Please note: Installations are not always seamless. Do not despair.
Remember Google and Stack Overflow.

Contents

1. Terminal

1.1. Mac Users

You already have a terminal. It's in Applications → Utilities → Terminal. This seems like a reasonable link.

1.2. Windows 10 Users

You already have a terminal, it's called PowerShell. It's a tad annoying in the sense that it's not Linux-based, so you might want to use Git Bash instead (see section 2.2). But if you can't wait, go to main menu (Windows key) and start typing "powershell", choose it:
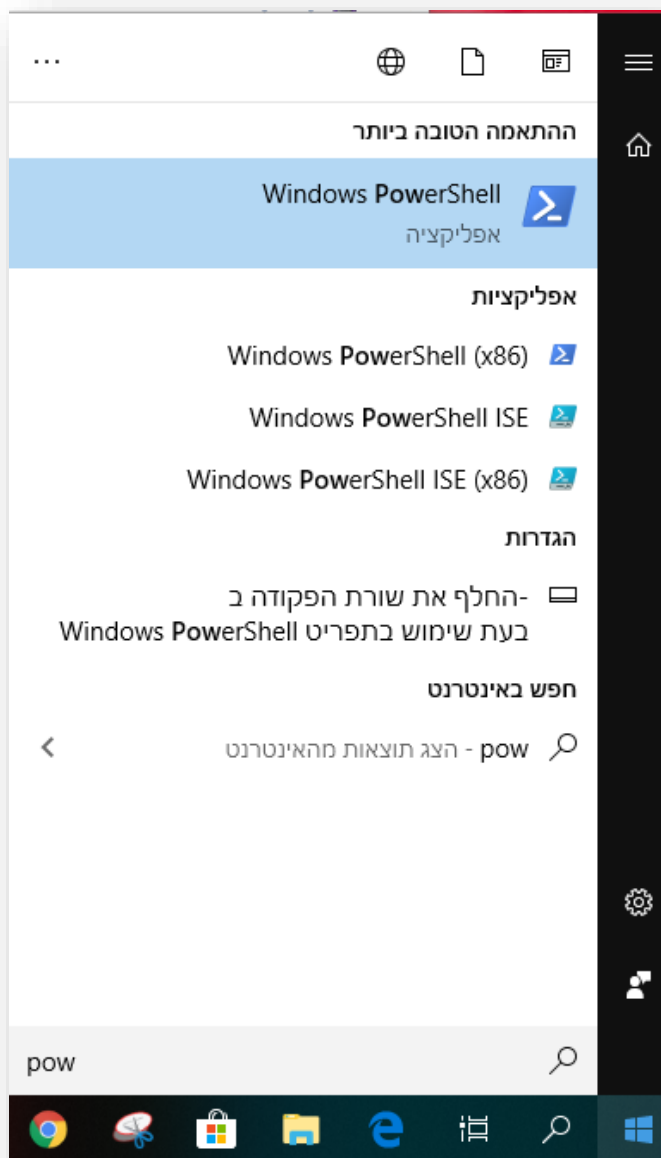
1.3. Windows 7 (or less) Users

(a) ☹

(b) Work with Git Bash (see section 2.2) or the CLI which comes from installing Docker Toolbox (see section 3.1).

1.4. Use Terminal

I don't expect you to be/come a Terminal Ninja. For now, I just want you to be able to:

1.4.1. Know your current directory: pwd
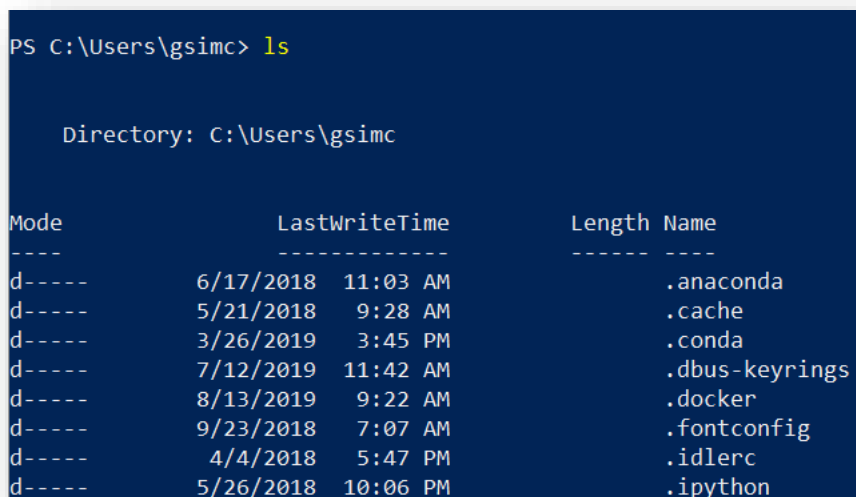
1.4.2. List files in current directory: ls

1.4.3. Create a new directory: mkdir

```
PS C:\Users\gsimc> mkdir new_dir


    Directory: C:\Users\gsimc


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       11/12/2019   10:58 AM                new_dir


PS C:\Users\gsimc>
```

1.4.4. Change directory: cd

```
PS C:\Users\gsimc> cd new_dir
PS C:\Users\gsimc\new_dir> ls
PS C:\Users\gsimc\new_dir> cd ..
PS C:\Users\gsimc>
```

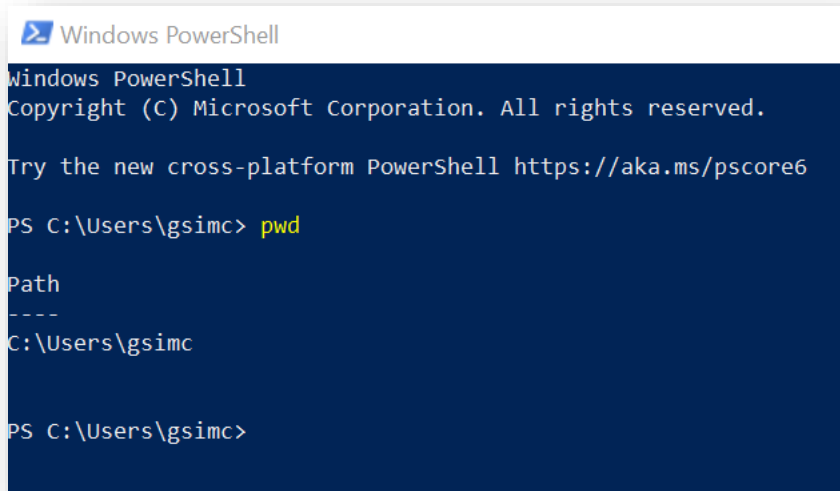### 1.4.5. Create a new text file with some text: echo

```
PS C:\Users\gsimc>
PS C:\Users\gsimc> cd new_dir
PS C:\Users\gsimc\new_dir> echo "some text" >> filename.txt
PS C:\Users\gsimc\new_dir> ls


    Directory: C:\Users\gsimc\new_dir


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----       11/12/2019   11:10 AM             68 filename.txt


PS C:\Users\gsimc\new_dir>
```

Mac users (and Windows Git Bash/Docker users working on a Linux-based environment) might prefer "touch" and edit a file using the Vim editor.

1.4.6. Print the contents of a text file: cat

```
PS C:\Users\gsimc\new_dir>
PS C:\Users\gsimc\new_dir> cat filename.txt
some text
PS C:\Users\gsimc\new_dir>
```

1.4.7. Remove a file/directory: rm

```
PS C:\Users\gsimc\new_dir> ls


    Directory: C:\Users\gsimc\new_dir


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        11/12/2019   11:22 AM             24 filename.txt


PS C:\Users\gsimc\new_dir> rm filename.txt
PS C:\Users\gsimc\new_dir> ls
PS C:\Users\gsimc\new_dir>
PS C:\Users\gsimc\new_dir> cd ..
PS C:\Users\gsimc> rm -r new_dir
PS C:\Users\gsimc>
```

1.4.8. Later

Once you're working in a Linux environment (Linux users, Mac users, Windows 10 users who use WSL, Cygwin or dare use Docker), you'll probably need more out of your terminal, like editing files, changing permissions, scheduling cron jobs and more. This looks like a good place to start: https://medium.com/@grace.m.nolan/terminal-for-beginners-e492ba10902a

2. Git and Github

2.1. What is Git?

In a nutshell:

> By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).

2.2. Do you have Git already?

Try making a new directory, navigating to it and start a git repository using `git init`.

If you get something like:

```
PS C:\Users\gsimc> mkdir new_dir


    Directory: C:\Users\gsimc


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/12/2019   11:37 AM                new_dir


PS C:\Users\gsimc> cd new_dir
PS C:\Users\gsimc\new_dir> git init
Initialized empty Git repository in C:/Users/gsimc/new_dir/.git/
PS C:\Users\gsimc\new_dir>
```

Congrats! You already have Git on your system. Go straight to 2.3 and start using Git.

If you get an error:

```
PS C:\Users\schud> mkdir new_dir


    Directory: C:\Users\schud


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        11/12/2019  11:34 AM                new_dir


PS C:\Users\schud> cd new_dir
PS C:\Users\schud\new_dir> git init
git : The term 'git' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the
spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ git init
+ ~~~
    + CategoryInfo          : ObjectNotFound: (git:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\schud\new_dir>
```
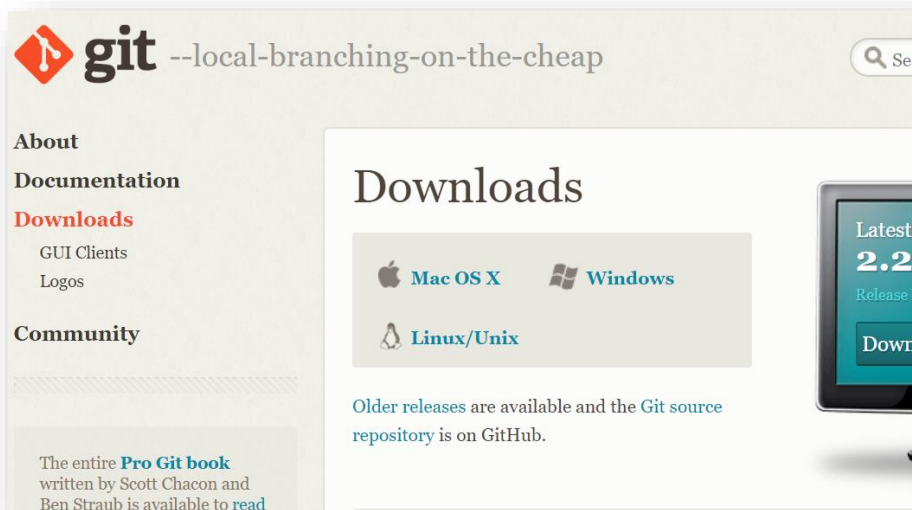
Then you must install Git.

2.3. Install Git

Download the relevant file (Mac/Windows/Linux) from the Git downloads page: https://git-scm.com/download/

Once the download is finished, press the ".exe" file and start the installation wizard:

ed, **click here to download manually.**

ownloads

mbdrive edition")

ble.

ble.

2.24.0. If you want the newer version, you can build it

⌃ Git-2.24.0.2-64-bit.exe

Just go with the flow, accepting all defaults, when done press Finish:

Git 2.24.0.2 Setup — □ ✕

Information
Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change

https://gitforwindows.org/

Next > | Cancel

Congrats! You have Git on your system.

Windows users: Now, with your Git installation, if you accepted all defaults, comes a nice terminal you could use, called Git Bash (just press the Windows key and start typing "git", it should be there):

Alternatively, you can now use Git in your PowerShell (though you'd have to restart it), see next section.

2.4. Use Git

First, it's best practice to tell Git who you are:

```
git config --global user.name "YOUR FULL NAME"
git config --global user.email "YOUR EMAIL ADDRESS"
```

Please use the email address which you will use to open a Github account in the next section.

You can confirm this actually happened with `git config --global --list`

Now, let's create an empty git repository called "test_repo" using `git init`:

```
PS C:\Users\gsimc> mkdir test_repo


    Directory: C:\Users\gsimc


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/12/2019  12:40 PM                test_repo


PS C:\Users\gsimc> cd test_repo
PS C:\Users\gsimc\test_repo> git init
Initialized empty Git repository in C:/Users/gsimc/test_repo/.git/
PS C:\Users\gsimc\test_repo>
```

A good command to execute before anything you might do is `git status`:

```
PS C:\Users\gsimc\test_repo> git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
PS C:\Users\gsimc\test_repo>
```

Git thinks of our repository as a tree with branches, the main branch being called "master". It tells us there are no "commits" yet, and that there's nothing to commit – makes sense, our repo is empty.

So, let's create a simple text file and do `git status` again:

```
PS C:\Users\gsimc\test_repo> echo "some text" >> file1.txt
PS C:\Users\gsimc\test_repo> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file1.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\gsimc\test_repo>
```

There are still no "commits" but Git now sees that we have an untracked file, file1.txt. We'll track it using `git add`:

```
PS C:\Users\gsimc\test_repo> git add file1.txt
PS C:\Users\gsimc\test_repo> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   file1.txt

PS C:\Users\gsimc\test_repo>
```

There are still no commits, but Git now tracks file1.txt and suggests committing it, so let's do that using `git commit`:

```
PS C:\Users\gsimc\test_repo> git commit -m "Adds file1.txt"
[master (root-commit) 8879936] Adds file1.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
PS C:\Users\gsimc\test_repo>
```

Notice we added the -m parameter followed by a short commit message. But what is "commit"? Think of committing as saving changes to a file, telling Git you are *committing* to the current version of the file. If we change file1.txt by adding some more text:

```
PS C:\Users\gsimc\test_repo> echo "some more text" >> file1.txt
PS C:\Users\gsimc\test_repo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gsimc\test_repo>
```

You can see Git is "aware" of our changes and considers file1.txt as *modified*. If we're happy with file1.txt we can *add* it and *commit* it again:

```
PS C:\Users\gsimc\test_repo> git add file1.txt
PS C:\Users\gsimc\test_repo> git commit -m "Adds some more text to file1.txt"
[master 28ad21d] Adds some more text to file1.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\gsimc\test_repo> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\gsimc\test_repo>
```

You might have noticed this weird info at the beginning of the line after you commit: `[master 28ad21d]`. This means we're at branch master and the identifier of our current commit, the current state of the code, is 28ad21d (when you try it, obviously you'll get a different identifier). To see past commits, their author, their time, their identifiers and commit messages, use `git log`:

```
PS C:\Users\gsimc\test_repo> git log
commit 28ad21d07652f7c475cc7dee01567d86704f5218 (HEAD -> master)
Author: gsimc <giora@vfunction.com>
Date:   Tue Nov 12 12:59:42 2019 +0200

    Adds some more text to file1.txt

commit 8879936d65a82c9325d01bb136943ab500a17db7
Author: gsimc <giora@vfunction.com>
Date:   Tue Nov 12 12:52:27 2019 +0200

    Adds file1.txt
PS C:\Users\gsimc\test_repo>
```

(Use `git log -2` to see the last 2 commits if you have many)

One great thing about Git as a version control system is that it allows you to track changes between commits, thus figuring out what has changed (say, if a bug has occurred and you have no idea why and when). Use `git diff` and two commits identifiers to compare between them:

```
PS C:\Users\gsimc\test_repo> git diff 8879936 28ad21
diff --git a/file1.txt b/file1.txt
index 959be8c..54c8b42 100644
Binary files a/file1.txt and b/file1.txt differ
PS C:\Users\gsimc\test_repo>
```

Here Git is not very informative because of the way we created file1.txt 🙁, but it will be much more informative once you use a proper file editor and actual code.

Another great thing about using Git is that you can easily return to a previous version of your code (hence, "version control" …). Use `git checkout` and the identifier of the commit you wish to checkout:

```
PS C:\Users\gsimc\test_repo> cat file1.txt
some text
some more text
PS C:\Users\gsimc\test_repo> git checkout 8879936
Note: checking out '8879936'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 8879936 Adds file1.txt
PS C:\Users\gsimc\test_repo> cat file1.txt
some text
PS C:\Users\gsimc\test_repo>
```

Notice that after warning us that we're in a "detached HEAD" state, Git indeed changed file1.txt back to the previous commit state. But nothing is lost! If you want to return to the most current commit (a.k.a HEAD) do `git checkout master`.

If you want Git to really delete all changes from this previous commit onwards, and have your master branch operate from this commit from now on:

```
PS C:\Users\gsimc\test_repo> git checkout -b xy
Switched to a new branch 'xy'
PS C:\Users\gsimc\test_repo> git branch -f master
PS C:\Users\gsimc\test_repo> git checkout master
Switched to branch 'master'
PS C:\Users\gsimc\test_repo> git status
On branch master
nothing to commit, working tree clean
```

> **WARNING: Be sure you're OK with deleting your changes after that previous commit. See what branches are for.**
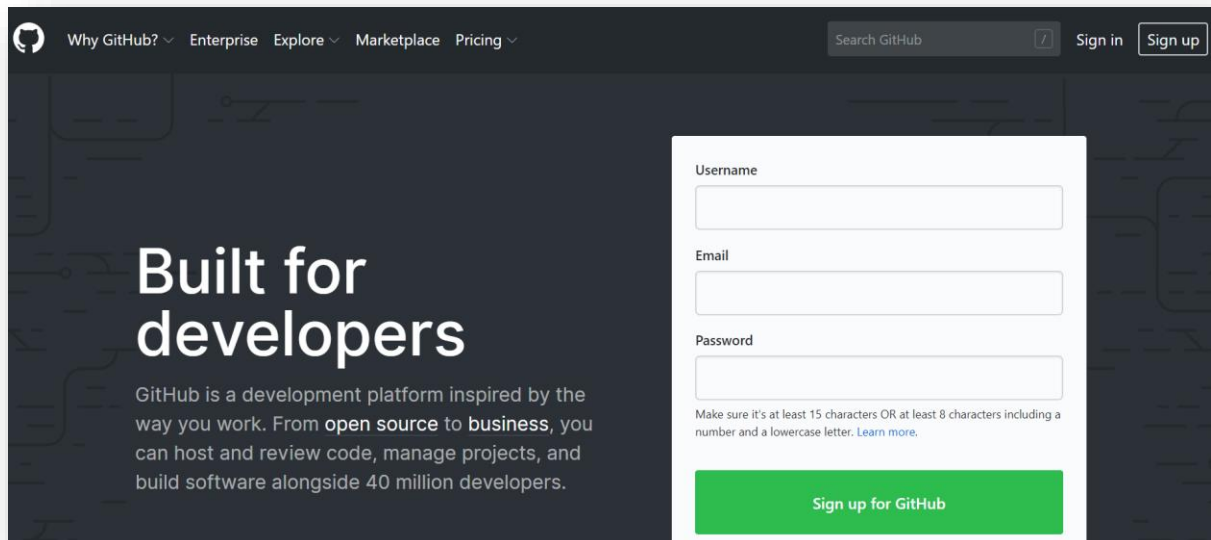
There is much more to Git, specifically working with branches, but enough for now.

2.5. Get a Github account

Notice that up until now we have been working with Git *without* Github. This is important to note, Git is installed on your local system, it is a system for managing versions of code. Github is sort of a Dashboard for Git, where you can store repositories (like interfacing with a remote directory in Google Drive, Dropbox or any other cloud provider). And as such, there are other options to use, like BitBucket and Gitlab. Github comes especially handy when you collaborate with other coders on the

same repo, when you need to use other people's Open Source code and when you (yes, you!) would like to contribute to an existing Open Source project.

Got to https://github.com/ and sign up, using the same email from section 2.3 where you configured git:

You might be asked to verify your account using some weird strategy, and then choose the Free plan (unless you're rich):

You might be asked to answer a short survey, you may choose to skip it, and then you're asked to verify your mail address:
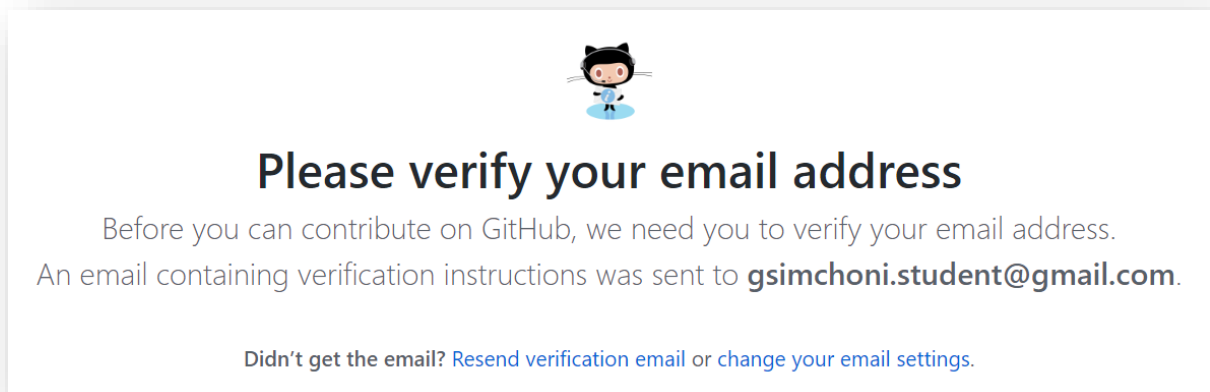
Once you've verified your mail address (pressing "Verify email address" in the email which was sent to you), you're done:
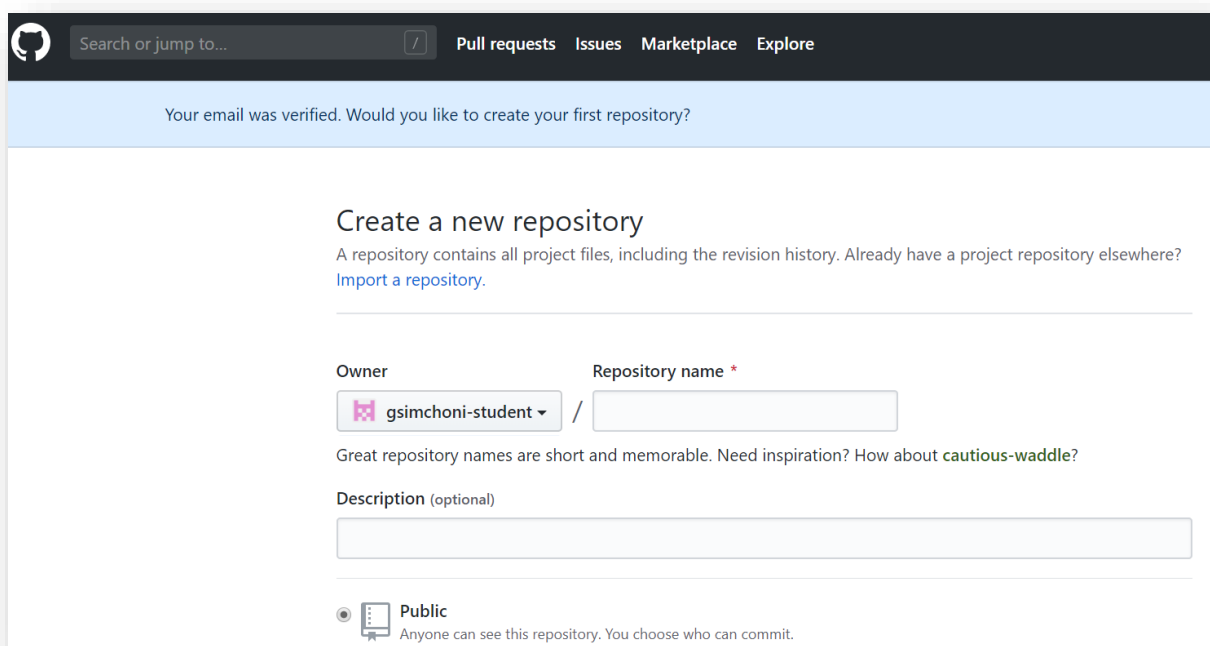
You can start a new *remote* repository, Github's instructions are quite useful, or you can move on to the next section for our basic instructions.

2.6. Use Git with Github

There are a few main workflows of starting a new repository:

2.6.1.   You started a local Git repo and you want to connect it to a new, non-existing remote Github repo

Let's connect our local test_repo folder to a remote test_repo Github repo.

Create a new remote repo in Github using the "+" sign, call it by the same name, "test_repo":

You should see:

Keep all defaults (no need to add a README or ".gitignore" at this stage, keep it public), press Create repository.
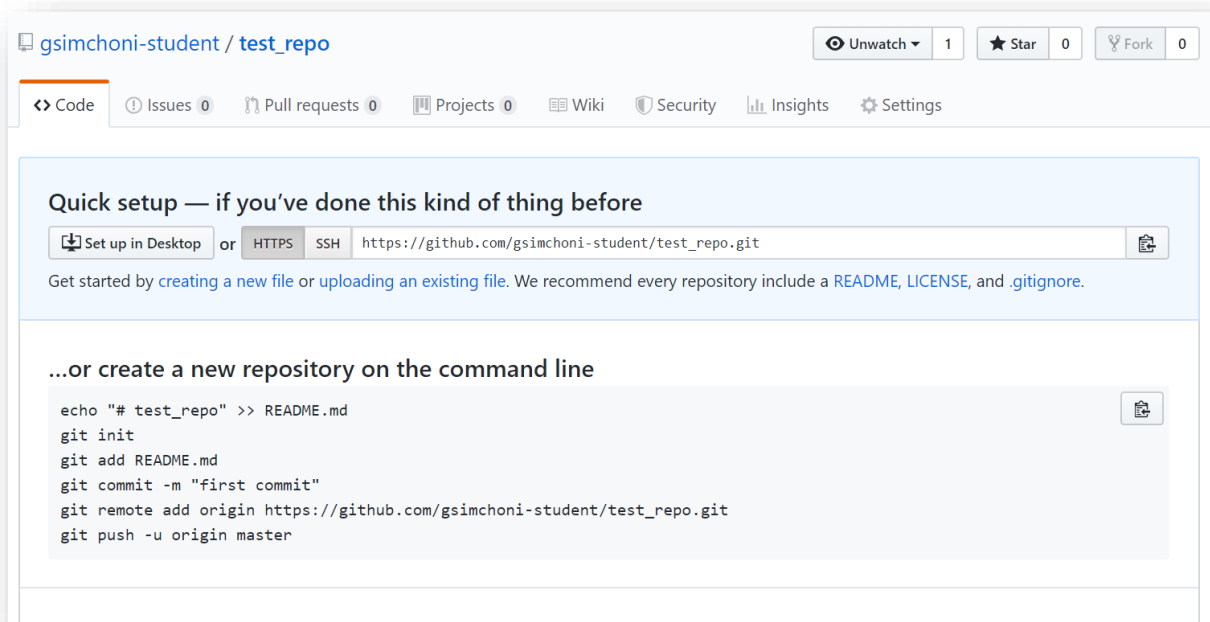
You now have an empty remote Github repo, and Github already gives you some tips on what to do next.

In the terminal, inside your local repo, add this remote Github repo with these two commands:

```
git remote add origin https://github.com/your_user/test_repo.git
git push -u origin master
```

In my case:

The first time you do this you will be asked to provide your Github username and password. If you run into this error:

This might mean that Git is already configured on your machine to work with a different username, like your work username, in which case this link might be useful.

After successfully "pushing" your local repo to the remote, refresh Github and you'll see your test_repo out in public:
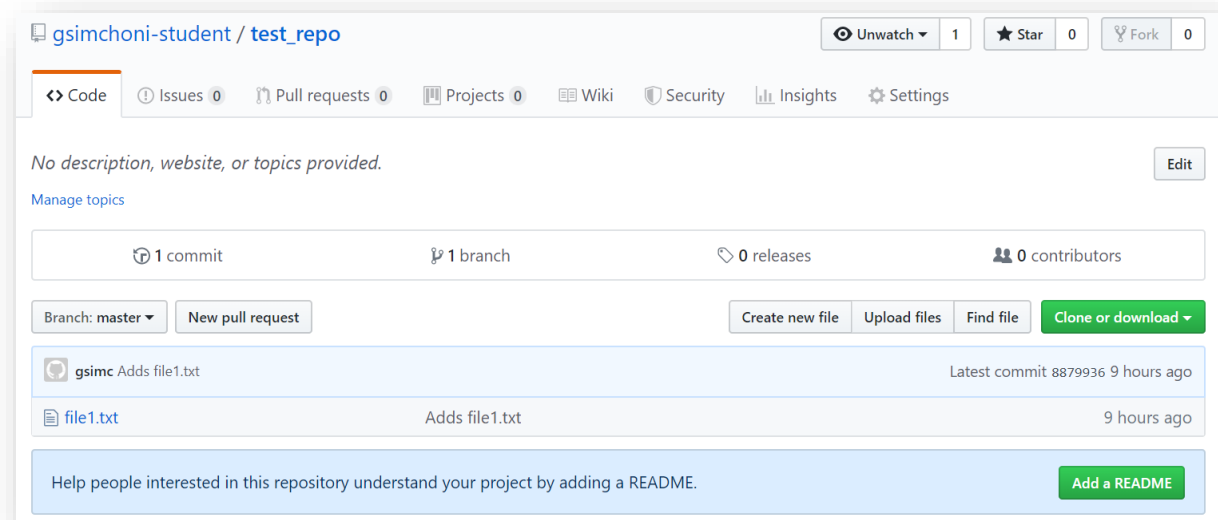
2.6.2. You started a remote Github repo using Github GUI in the browser, you want to connect it to a new local repo

Let me start a new repository in the Github GUI in the browser, call it test_repo2:
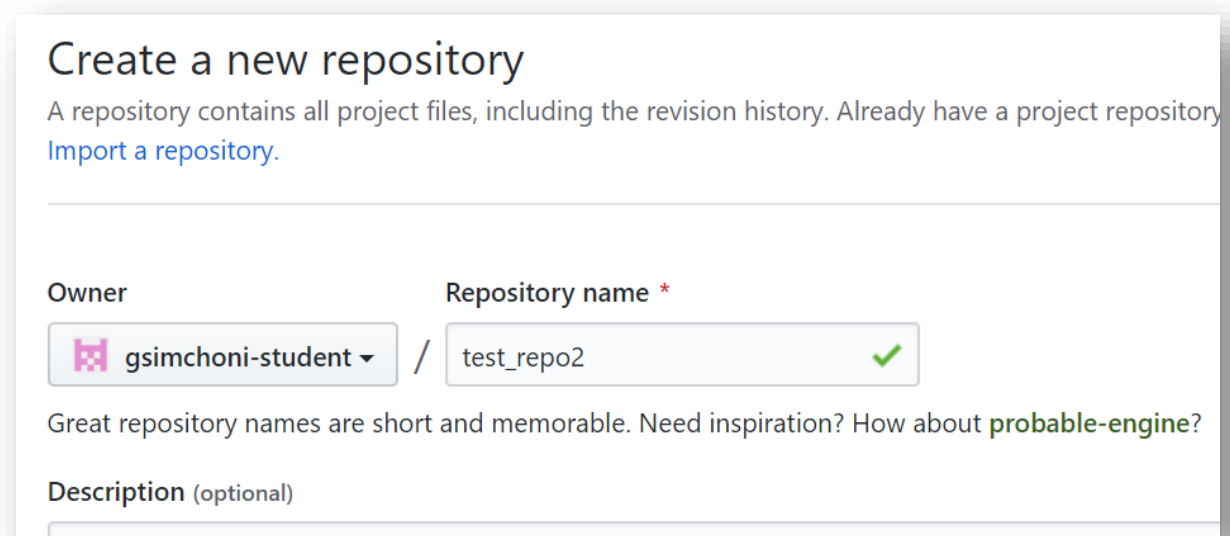
Once it is created, Github tells us exactly what to do to connect it to a new local clone.

Create a new directory, preferably with the same name test_repo2, initialize it as a Git repo and add the remote Github test_repo2:

```
PS C:\Users\gsimc\test_repo>
PS C:\Users\gsimc\test_repo> cd ..
PS C:\Users\gsimc> mkdir test_repo2


    Directory: C:\Users\gsimc


Mode                LastWriteTime        Length Name
----                -------------        ------ ----
d-----        11/12/2019   9:52 PM               test_repo2


PS C:\Users\gsimc> cd test_repo2
PS C:\Users\gsimc\test_repo2> git init
Initialized empty Git repository in C:/Users/gsimc/test_repo2/.git/
PS C:\Users\gsimc\test_repo2> git remote add origin https://github.com/gsimchoni-student/test_repo2.git
PS C:\Users\gsimc\test_repo2>
PS C:\Users\gsimc\test_repo2>
```

You can create a new file, stage it (`git add`), commit it (`git commit`) and push to remote (`git push`):

```
PS C:\Users\gsimc\test_repo2> echo "yet some more text" >> file2.txt
PS C:\Users\gsimc\test_repo2> git add file2.txt
PS C:\Users\gsimc\test_repo2> git commit -m "Adds file2.txt"
[master (root-commit) b72035a] Adds file2.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2.txt
PS C:\Users\gsimc\test_repo2> git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 253 bytes | 126.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/gsimchoni-student/test_repo2.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
PS C:\Users\gsimc\test_repo2>
```

Refresh your remote Github repo test_repo2 to see file2.txt on the remote.

2.6.3. You cloned someone else's Github repo to local, you want to work with it locally

Cloning a remote Github repo simply means downloading it to your local machine. Try cloning the course Assignment 0 Repo with:

```
git clone https://github.com/DSApps-2020/HW0.git
```

```
PS C:\Users\gsimc> git clone https://github.com/DSApps-2020/HW0.git
Cloning into 'HW0'...
remote: Enumerating objects: 134, done.
remote: Counting objects: 100% (134/134), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 134 (delta 60), reused 100 (delta 26), pack-reused 0
Receiving objects: 100% (134/134), 1021.67 KiB | 8.00 KiB/s, done.
Resolving deltas: 100% (60/60), done.
PS C:\Users\gsimc>
```

This created a local copy, a local folder called HW0, which you can navigate to, run scripts from, it is physically on your machine:

```
PS C:\Users\gsimc> cd HW0
PS C:\Users\gsimc\HW0> ls


    Directory: C:\Users\gsimc\HW0


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        2/26/2020     5:45 PM                binder
d-----        2/26/2020     5:45 PM                docker
d-----        2/26/2020     5:45 PM                images
-a----        2/26/2020     5:45 PM            174 .gitignore
-a----        2/26/2020     5:45 PM            218 HW0.Rproj
-a----        2/26/2020     5:45 PM          13058 hw0_python.ipynb
-a----        2/26/2020     5:45 PM          14313 hw0_r.Rmd
-a----        2/26/2020     5:45 PM           1619 README.md


PS C:\Users\gsimc\HW0>
```

Is this local repo connected to remote?

```
PS C:\Users\gsimc\HW0> git remote -v
origin  https://github.com/DSApps-2020/HW0.git (fetch)
origin  https://github.com/DSApps-2020/HW0.git (push)
PS C:\Users\gsimc\HW0>
```

Yes, it is. Can you create a new file, stage it, commit it and push to remote?

```
PS C:\Users\gsimc\HW0> echo "more more text" >> file1.txt
PS C:\Users\gsimc\HW0> git add file1.txt
PS C:\Users\gsimc\HW0> git commit -m "Adds file1.txt"
[master 18bd80b] Adds file1.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
PS C:\Users\gsimc\HW0> git push -u origin master
remote: Permission to DSApps-2020/HW0.git denied to gsimchoni-student.
fatal: unable to access 'https://github.com/DSApps-2020/HW0.git/': The requested URL returned error: 403
PS C:\Users\gsimc\HW0>
```

You can add a new file, stage it, commit it, but you can't push to remote because it is NOT YOUR REMOTE!

You can change the remote to be a new remote Github repo in your account, and work with it. But that's not what cloning is for, cloning is for working with a remote repo locally. If you want to contribute to someone else's remote repo (say if you're working in collaboration with others on a single project), that's what forking is for.

2.6.4. You forked someone else's remote Github repo to your remote Github account, you want to contribute to that repo

Forking is easy in the Github GUI in your browser. Go to the course public Assignment 0 repo, and press the "fork" button:
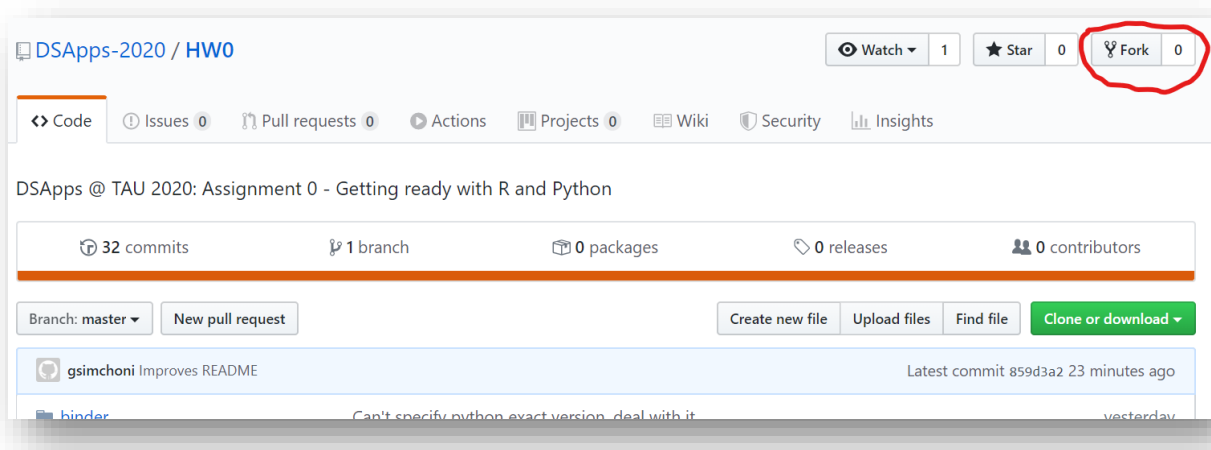
You now have your own copy of remote Github repo of Assignment 0:

gsimchoni-student / HW0

forked from DSApps-2020/HW0

Watch 0 · Star 0 · Fork 1

<> Code   Pull requests 0   Actions   Projects 0   Wiki   Security   Insights   Settings

DSApps @ TAU 2020: Assignment 0 - Getting ready with R and Python   Edit

Manage topics

32 commits   1 branch   0 packages   0 releases   0 contributors

Branch: master ▾   New pull request   Create new file   Upload files   Find file   Clone or download ▾

This branch is even with DSApps-2020:master.   Pull request   Compare

gsimchoni Improves README   Latest commit 859d3a2 26 minutes ago

binder   Can't specify python exact version, deal with it   yesterday

You can clone it locally, add or change files, and push to remote which is now YOUR remote (here I am first FORCE deleting the previous copy HW0 in PowerShell):


```
PS C:\Users\gsimc> rm -r HW0 -fo
PS C:\Users\gsimc> git clone https://github.com/gsimchoni-student/HW0.git
Cloning into 'HW0'...
remote: Enumerating objects: 134, done.
remote: Counting objects: 100% (134/134), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 134 (delta 60), reused 100 (delta 26), pack-reused 0 eceiving o
B/s
Receiving objects: 100% (134/134), 1021.67 KiB | 28.00 KiB/s, done.
Resolving deltas: 100% (60/60), done.
PS C:\Users\gsimc> cd HW0
PS C:\Users\gsimc\HW0> echo "more more text" >> file1.txt
PS C:\Users\gsimc\HW0> git add file1.txt
PS C:\Users\gsimc\HW0> git commit -m "Adds file1.txt"
[master a06cf7c] Adds file1.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
PS C:\Users\gsimc\HW0> git push -u origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 147.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/gsimchoni-student/HW0.git
   859d3a2..a06cf7c  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
PS C:\Users\gsimc\HW0>
```

Verify file1.txt was added to YOUR remote fork of the course repo:
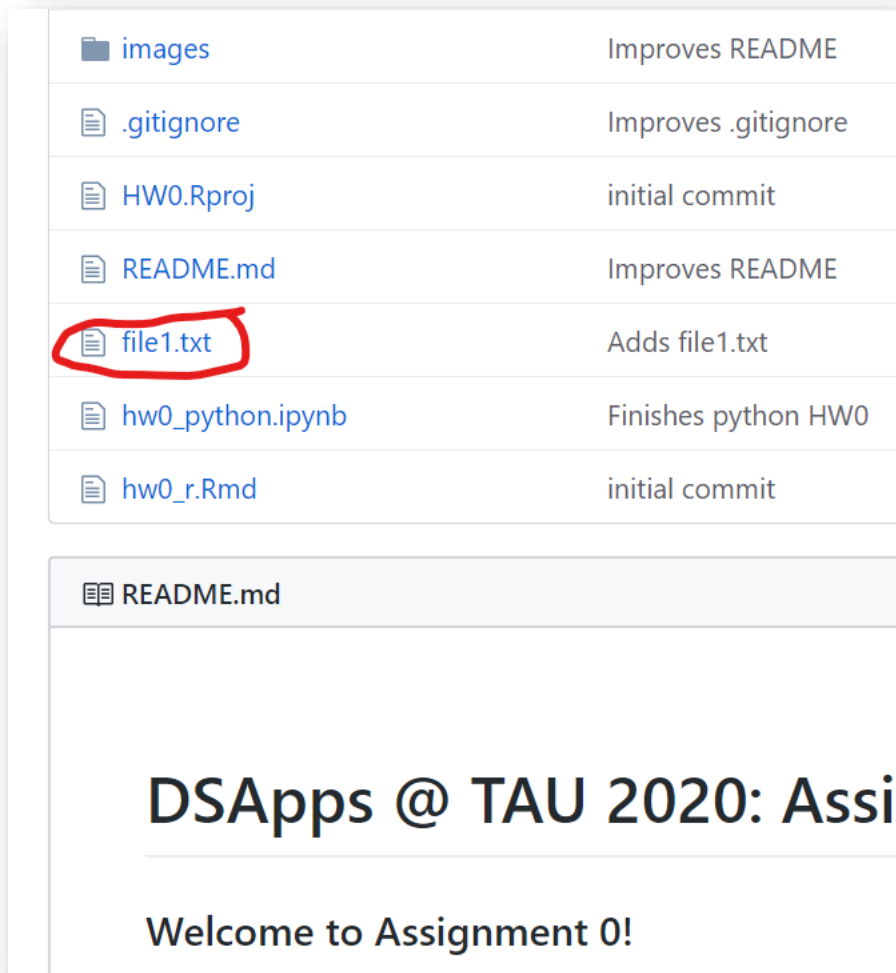
What if you want to contribute to the course repo? Or the main repo you're working on with your friends? That's what a "Pull Request" is for (you're requesting the main repo to "pull" your changes) but we don't need that for now.

2.6.5. Pulling changes from a remote repo

If the remote Github repo contains changes not yet existing in your local repo, you can `git pull` to pull them locally.

> **WARNING: Pulling will not work if your local repo contains changes not pushed to the remote Github repo! The remote Github repo must be synced to your local repo AND then have some changes, not the other way around.**

What can cause this? When working on a real public project this mainly occurs once a collaborator has changed the public Github repo, and you want these changes to be pulled to your local version. Here, we will simply add a file the remote Github repo in the browser, and pull this file to local.

First, verify that your local repo test_repo is "up to date" with the remote Github repo:

```
PS C:\Users\gsimc> cd test_repo
PS C:\Users\gsimc\test_repo> git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
PS C:\Users\gsimc\test_repo>
```

Second, in the your Github remote repo test_repo, create a new file, e.g. by pressing the "Add a README" button:
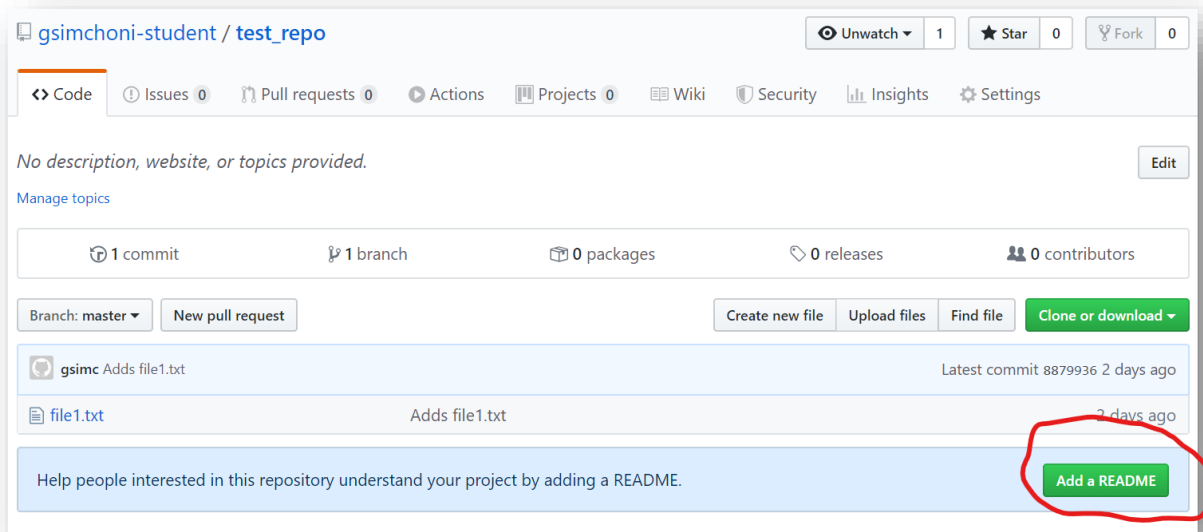
A README is usually a Markdown file containing important info about your project. For now you can just add a title and press the "Commit new file" button below:
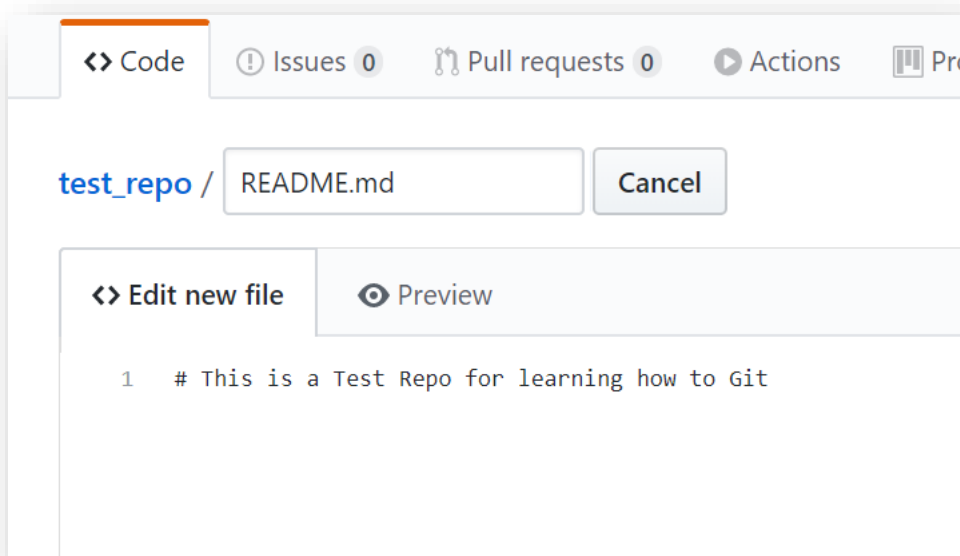
You can see that the repo now presents whatever you put in that README nicely on the main page:
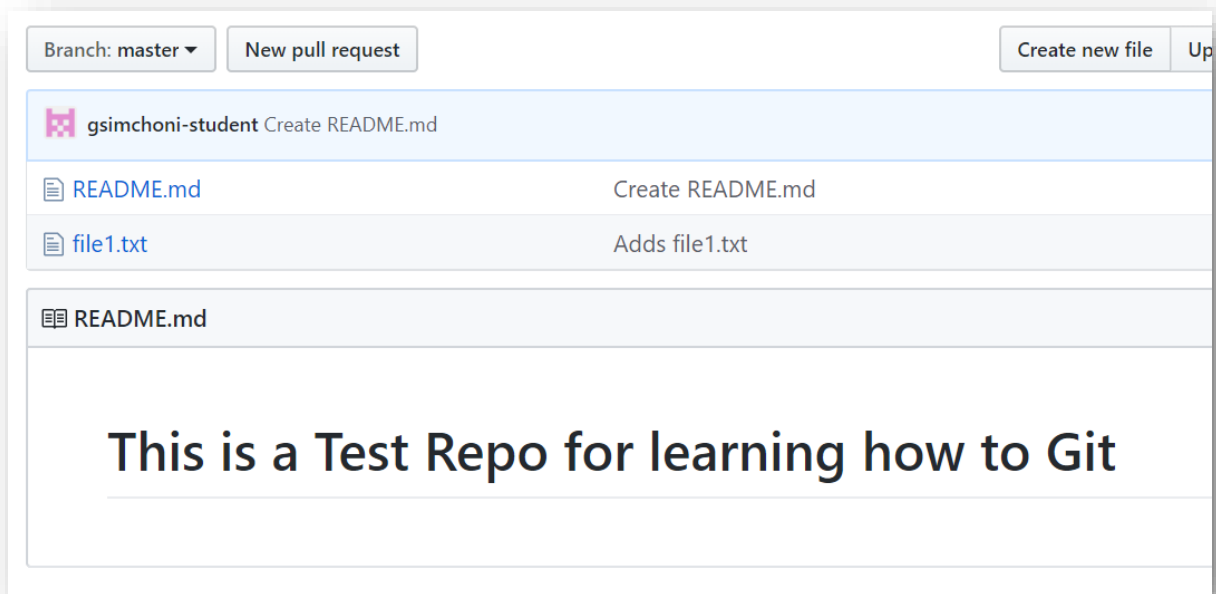
You can now `git pull` in your local repo to pull README locally:

3. Docker

3.1. What is Docker?

In a nutshell:

> Docker is a tool designed to make it easier to create, deploy, and run applications by using *containers*. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

You DON'T HAVE TO have Docker to work successfully in this course. You can happily skip it and install R and RStudio, Python and Jupyter notebooks, in the next sections.

BUT there are advantaged to using Docker:

- Saving Installation Time and Headache: In today's Open Source culture libraries change, and fast! Things I'm showing you work in Python 3.6 but did not work in Python 3.5. And I have no idea what's lurking around in your personal computer. Docker will make sure we all work in the exact same environment (the only difference being our initial Hardware specs and the amount of memory each of us allocates Docker). No more "Well it worked on my machine" and "Oh, you need Pandas 0.23 for that to work, sorry".
- Reproducibility: If you containerize your project it is fully reproducible, and that's a good thing. Anyone can repeat exactly what you did and reach your results. For example, you're a scientist running an experiment. You are no longer required to release only the data and code you used for analysis, but also a simple file (see section 3.5) which lets anyone reproduce it. You can even add in a simple interface app (e.g. with shiny) for the user to query your results – and we will 😊
- Testing: you could test if your code is running in R 3.3, R 3.4, R 3.5 (etc.) without installing them, but using different containers!
- Parallelizing: with enough memory you could open 5 containers of the same Docker "image" at once doing 5 jobs finishing the parent job in x5 less time.
- Docker is becoming more and more in use in the industry, so you might want to start using it now.

3.2. Install Docker

3.2.1. Create a Docker Account
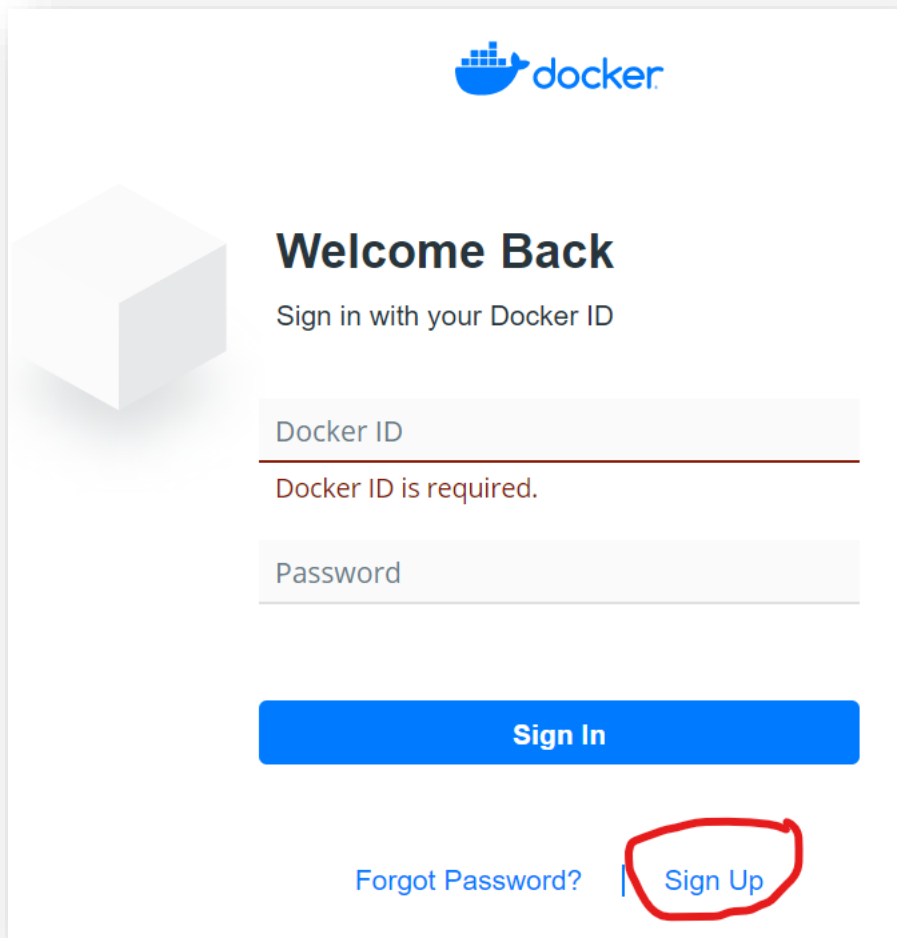
First create a DockerHub account in order to install: https://www.docker.com/

For some reason I found creating this account less than trivial, so I'll document it here.

Press "Sign In" (yes, I know):
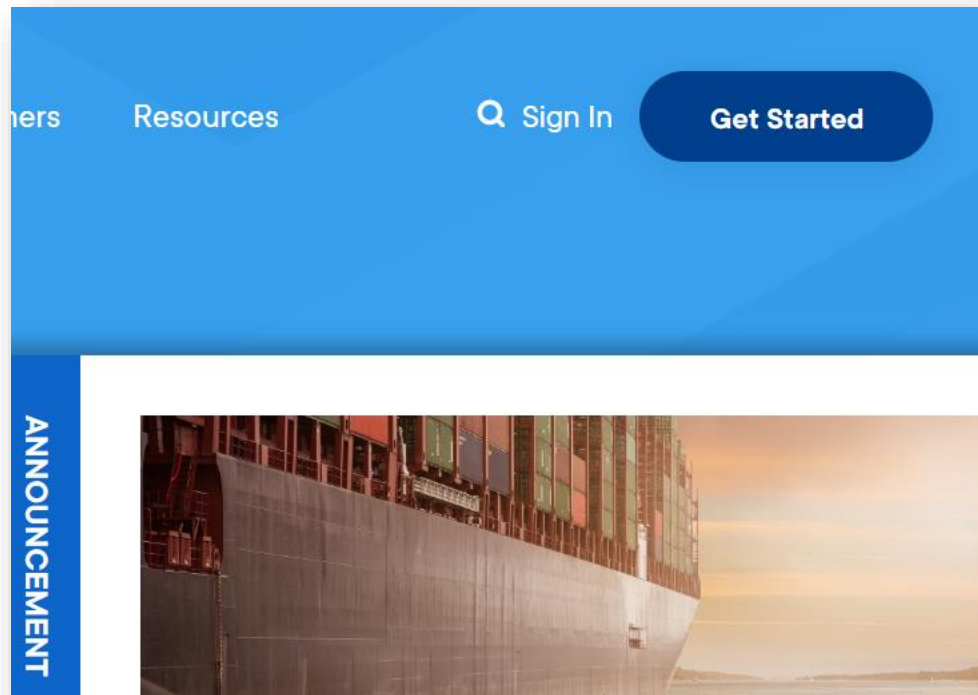
Press "Sign Up":

Fill in a username, email and password:

Fill in profile details (doesn't really matter) and verify the verification email you'll get. You should be redirected to the DockerHub signing in page and sign in with your credentials there. If not, go to https://hub.docker.com/ and sign in.

You now have a DockerHub account. Hint: it is very similar to having a Github account.

At this stage you might choose to follow Docker's instructions (e.g. by pressing "Get started with Docker Desktop") or you can follow the next links.

3.2.2. Install Docker locally

Mac: https://docs.docker.com/docker-for-mac/install/

Linux: https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/

Windows 10 Pro: https://docs.docker.com/docker-for-windows/install/

Windows 10 Home, Windows 7, install Docker Toolbox (will also install Oracle VM VirtualBox): https://docs.docker.com/toolbox/toolbox_install_windows/

At this stage I'm leaving you to your own, not because I want to, but because Docker's instructions in those links are nicer than anything I would concoct, and because Docker isn't mandatory.

Tips:
- (a) Installation may require restarting, might want to save your work before.
- (b) You don't have to go through the entire manual, make sure `docker run hello-world` works.
- (c) If `docker version` worked but `docker run hello-world` didn't: try `docker login` first to enter your DockerHub account details.
- (d) Windows users: I believe Docker initially has permissions to connect to any folder under your C:/Users/ folder. You can have it connect to any folder under drive C by checking it in the shared services tab, entering the user and password for your computer's account:
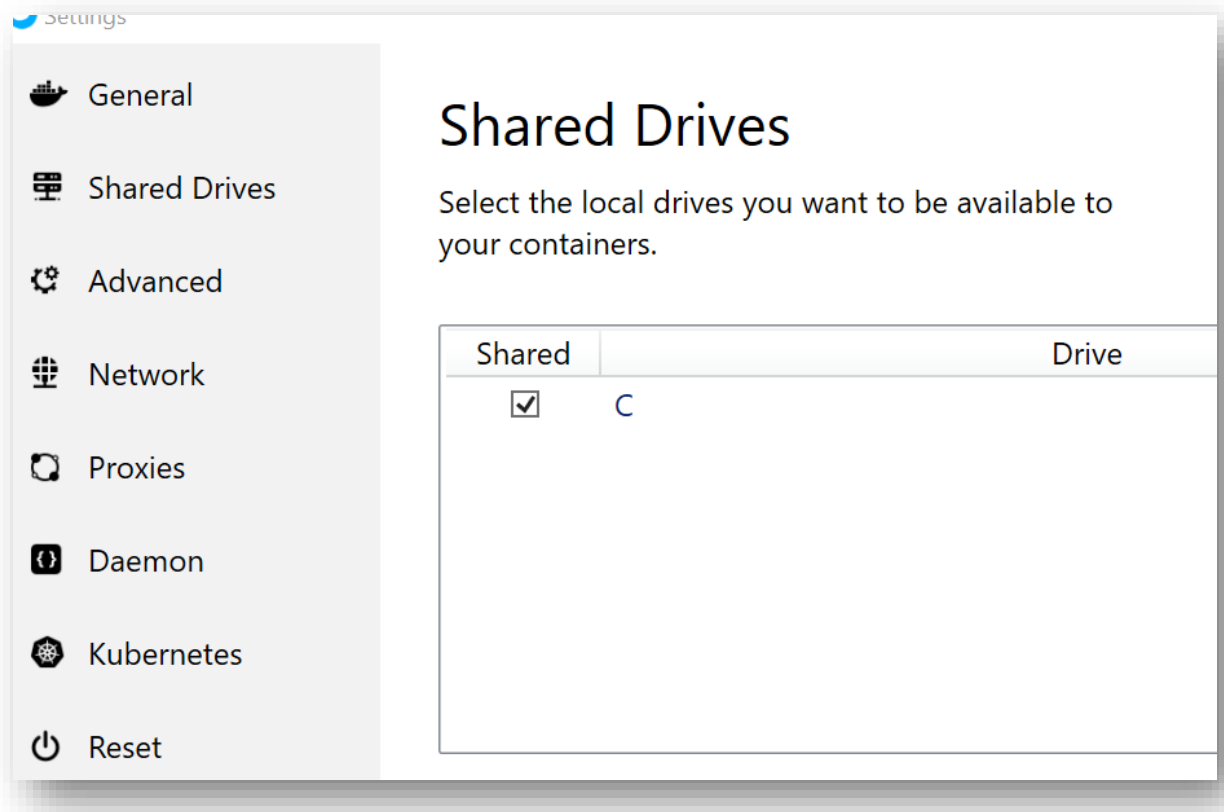
Docker Toolbox users should consult the [manual](), you need to define in in VirtualBox.
- (e) The more RAM you have on your machine, consider increasing Docker's defaults:
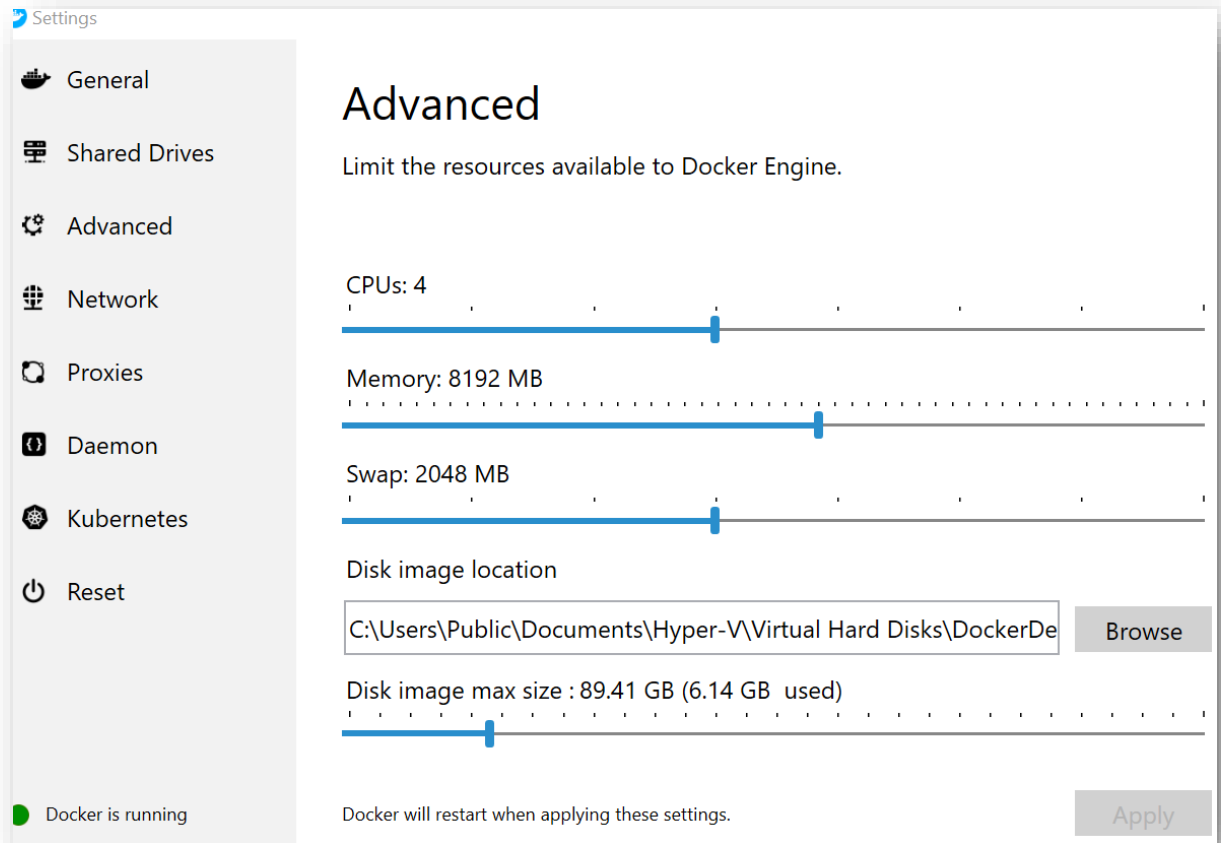
3.3. Use Docker

So, assuming you managed to run the `docker run hello-world` example, let's see some more Docker magic.

In the terminal do `docker run -ti ubuntu`. Docker will search for an Ubuntu (a Linux environment) on your machine, if it can't find one it will download the latest Ubuntu image from DockerHub, and…

```
PS C:\Users\gsimc> docker run -ti ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
7ddbc47eeb70: Pull complete
c1bbdc448b72: Pull complete
8c3b70e39044: Pull complete
45d437916d57: Pull complete
Digest: sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d
Status: Downloaded newer image for ubuntu:latest
root@00682d695ee8:/#
```

Voila! You are working in Linux!

To exit the container, just enter `exit`. To see a list of all the containers, do `docker container ls -a`:

```
PS C:\Users\gsimc> docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                    PORTS
       NAMES
5040579e6cd1        ubuntu              "/bin/bash"         12 seconds ago      Exited (0) 7 seconds ago
       ecstatic_lovelace
```

You can see our container isn't running, its status is "Exited". It is based on the "ubuntu" image, it has an identifier ("Container ID") we can use and a random weird name "ecstatic_lovelace".

You can remove such a dangling container which isn't running with `docker container rm` and the ID or name of the container, and see that the list of containers is empty:

```
PS C:\Users\gsimc> docker container rm ecstatic_lovelace
ecstatic_lovelace
PS C:\Users\gsimc> docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                    PORTS
NAMES
PS C:\Users\gsimc>
```

See the list of available images (I'm working on my machine so naturally I have a few you won't yet) with `docker image ls`:

```
PS C:\Users\gsimc> docker image ls
REPOSITORY                               TAG            IMAGE ID         CREATED         SIZE
rocker/tidyverse                         latest         518b50a44ce4     5 days ago      2.1GB
vfunction-docker-dev-local.jfrog.io/mysql  latest       509ddd6b08dc     10 days ago     456MB
ubuntu                                   latest         775349758637     13 days ago     64.2MB
vf_shiny                                 latest         2f48e0b17635     3 months ago    1.9GB
rocker/shiny-verse                       3.6.0          e46ffcae560b     3 months ago    1.86GB
PS C:\Users\gsimc>
```

Remove an image you're not using with `docker image rm` and its ID. Here I'm removing the Ubuntu image we've just created:

```
PS C:\Users\gsimc> docker image rm 775349758637
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d
Deleted: sha256:775349758637aff77bf85e2ff0597e86e3e859183ef0baba8b3e8fc8d3cba51c
Deleted: sha256:4fc26b0b0c6903db3b4fe96856034a1bd9411ed963a96c1bc8f03f18ee92ac2a
Deleted: sha256:b53837dafdd21f67e607ae642ce49d326b0c30b39734b6710c682a50a9f932bf
Deleted: sha256:565879c6effe6a013e0b2e492f182b40049f1c083fc582ef61e49a98dca23f7e
Deleted: sha256:cc967c529ced563b7746b663d98248bc571afdb3c012019d7f54d6c092793b8b
PS C:\Users\gsimc> docker image ls
REPOSITORY                               TAG            IMAGE ID         CREATED         SIZE
rocker/tidyverse                         latest         518b50a44ce4     5 days ago      2.1GB
vfunction-docker-dev-local.jfrog.io/mysql  latest       509ddd6b08dc     10 days ago     456MB
vf_shiny                                 latest         2f48e0b17635     3 months ago    1.9GB
rocker/shiny-verse                       3.6.0          e46ffcae560b     3 months ago    1.86GB
PS C:\Users\gsimc>
```

For many other use cases Google your way into to the Docker documentation, it is lovely.

3.4. "Mount a Volume"

Suppose you are working in Python 3.6 and wish to work in a Python 3.8 environment, which at the time this document is written is the most current Python version released to the general public.

Without any upgrade you can do `docker run -it python:3.8` and in a matter of 1 minute you are in a Python 3.8 console!

```
PS C:\Users\gsimc> docker run -ti python:3.8
Unable to find image 'python:3.8' locally
3.8: Pulling from library/python
c7b7d16361e0: Pull complete
b7a128769df1: Pull complete
1128949d0793: Pull complete
667692510b70: Pull complete
bed4ecf88e6a: Pull complete
8a8c75f3996a: Pull complete
bfbf6161579f: Pull complete
6e3c2947832c: Pull complete
5bab73b08276: Pull complete
Digest: sha256:514a95a32b86cafafefcecc28673bb647d44c5aadf06203d39c43b9c3f61ed52
Status: Downloaded newer image for python:3.8
Python 3.8.0 (default, Oct 17 2019, 05:36:36)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

Now suppose you would like to start your session at the container terminal, add `bash` to this command. You can call Python, exit it:

```
PS C:\Users\gsimc> docker run -ti python:3.8 bash
root@69c6a9f7f892:/# python
Python 3.8.0 (default, Oct 17 2019, 05:36:36)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> exit()
root@69c6a9f7f892:/#
```

But while in the terminal – where are you exactly? Can you find your local files?

```
root@69c6a9f7f892:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@69c6a9f7f892:/# ls home
root@69c6a9f7f892:/# ls usr
bin  games  include  lib  local  sbin  share  src
root@69c6a9f7f892:/# ls mnt
root@69c6a9f7f892:/#
```

Problem: your local files are no where to be seen. You need to connect Docker into a folder you'd like to read from and write to, a.k.a "mount a volume". Go the test_repo and use the `-v` flag like so to mount your volume and see your local files:

```
PS C:\Users\gsimc> cd test_repo
PS C:\Users\gsimc\test_repo> ls


    Directory: C:\Users\gsimc\test_repo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         11/12/2019   1:43 PM             24 file1.txt
-a----         11/14/2019  10:22 AM             47 README.md


PS C:\Users\gsimc\test_repo> docker run -it -v ${PWD}:/home/ python:3.8 bash
root@b89f813d334a:/# ls home
README.md  file1.txt
root@b89f813d334a:/#
```

And there you have your files, ready to be processed by Python 3.8. The `-v ${PWD}:/home` part
we've added means "mount my working directory to the container's home directory".

3.5. Course Docker and Dockerfile

Now when inside the container, suppose you wish to install the [Pympler](#) library for easy memory-
size profiling. Using `pip install` should work and we should be able to use it in Python (e.g.
measuring the size of a simple dictionary in bytes):

```
root@b89f813d334a:/# pip install pympler
Collecting pympler
  Downloading https://files.pythonhosted.org/packages/26/75/d38ea74acc62acbd4
       |                                        | 184kB 413kB/s
Building wheels for collected packages: pympler
  Building wheel for pympler (setup.py) ... done
  Created wheel for pympler: filename=Pympler-0.8-cp38-none-any.whl size=1647
  Stored in directory: /root/.cache/pip/wheels/6c/61/cc/4bdf1e8c8b1c04d810432
Successfully built pympler
Installing collected packages: pympler
Successfully installed pympler-0.8
root@b89f813d334a:/# python
Python 3.8.0 (default, Oct 17 2019, 05:36:36)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pympler import asizeof
>>> my_dict = {'a': 1, 'b': 2}
>>> asizeof.asizeof(my_dict)
408
>>>
```

But if we exit the container, and run a new one…

```
PS C:\Users\gsimc\test_repo> docker run -it -v ${PWD}:/home/ python:3.8 bash
root@af89dea3eb80:/# python
Python 3.8.0 (default, Oct 17 2019, 05:36:36)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pympler import asizeof
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'pympler'
>>>
```

How sad… it is a different container and the library installation isn't persistent, and so we get an error when trying to use it. It is not there. There is a way of exiting the container, keep it running, return to it – Google your way.

But how do we always get the container we want, a Python 3.8 with Pympler installed?

That's what Dockerfiles are for.

Open a new file in your favorite text editor, call it "Dockerfile" (no extension), put it in the test_repo folder.

Input these lines and save:

```
FROM python:3.8

RUN pip install pympler
```

From within the test_repo folder do `docker build -t python38_with_pympler .` This will build a new image custom made by you called "python38_with_pympler".

```
PS C:\Users\gsimc\test_repo> ls


    Directory: C:\Users\gsimc\test_repo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        11/14/2019   6:41 PM             42 Dockerfile
-a----        11/12/2019   1:43 PM             24 file1.txt
-a----        11/14/2019  10:22 AM             47 README.md


PS C:\Users\gsimc\test_repo> docker build -t python38_with_pympler .
Sending build context to Docker daemon  73.73kB
Step 1/2 : FROM python:3.8
 ---> d6a7b0694364
Step 2/2 : RUN pip install pympler
 ---> Running in 4aa00b824cf3
Collecting pympler
  Downloading https://files.pythonhosted.org/packages/26/75/d38ea74acc62acbd4609f3f02bb9
Building wheels for collected packages: pympler
  Building wheel for pympler (setup.py): started
  Building wheel for pympler (setup.py): finished with status 'done'
  Created wheel for pympler: filename=Pympler-0.8-cp38-none-any.whl size=164713 sha256=0
  Stored in directory: /root/.cache/pip/wheels/6c/61/cc/4bdf1e8c8b1c04d8104322eb2508b6a2
Successfully built pympler
Installing collected packages: pympler
Successfully installed pympler-0.8
Removing intermediate container 4aa00b824cf3
 ---> 930c4d040527
Successfully built 930c4d040527
Successfully tagged python38_with_pympler:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Doc
recommended to double check and reset permissions for sensitive files and directories.
```

You can see that it is a legit image:

```
PS C:\Users\gsimc\test_repo> docker image ls
REPOSITORY                                    TAG        IMAGE ID        CREATED          SIZE
python38_with_pympler                         latest     930c4d040527    2 minutes ago    940MB
rocker/tidyverse                              latest     518b50a44ce4    5 days ago       2.1GB
vfunction-docker-dev-local.jfrog.io/mysql     latest     509ddd6b08dc    10 days ago      456MB
python                                        3.8        d6a7b0694364    3 weeks ago      932MB
vf_shiny                                      latest     2f48e0b17635    3 months ago     1.9GB
rocker/shiny-verse                            3.6.0      e46ffcae560b    3 months ago     1.86GB
PS C:\Users\gsimc\test_repo>
```

And when you run a container for this image…

```
PS C:\Users\gsimc\test_repo> docker run -it -v ${PWD}:/home/ python38_with_pympler bash
root@fc9f5154b690:/# python
Python 3.8.0 (default, Oct 17 2019, 05:36:36)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pympler import asizeof
>>> s = set([1,2,3])
>>> asizeof.asizeof(s)
312
>>>
```

You can work with Pympler without installing! Hurray.

There are many other elements a Dockerfile may contain, e.g. specifying a port in which your app will be accessed in the browser.

The Assignment 0 repo contains a few Dockerfiles. You can clone the repo, navigate to it and build one of these Dockerfiles, exactly as we did, specifying which Dockerfile you want with the `-f` flag.

For example, to work with the latest R, the latest Tidyverse packages and latest RStudio, use the Dockerfile-R and do `docker build -t dsapps-r-hw0 -f docker/Dockerfile-R .`

If you want to try it without cloning the repo just write the following Dockerfile into some folder (e.g. test_repo) and build it specifically (that's the beauty of Dockerfile, it is a recipe for an environment!) create your own Dockerfile-R file in that directory:

```
FROM rocker/tidyverse

RUN apt-get update -qq && apt-get -y --no-install-recommends
install \
  libpython3-dev \
  python3-pip \
  && install2.r --error \
    --deps TRUE \
    reticulate
```

And build the course R environment:

```
PS C:\Users\gsimc\test_repo> docker build -t dsapps_r -f Dockerfile-R .
Sending build context to Docker daemon   74.75kB
Step 1/2 : FROM rocker/tidyverse
 ---> 518b50a44ce4
Step 2/2 : RUN apt-get update -qq && apt-get -y --no-install-recommends install   li
 ---> Running in 9384106c0c86
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libpython3.5 libpython3.5-dev python-pip-whl
Recommended packages:
  build-essential python3-dev python3-setuptools python3-wheel
The following NEW packages will be installed:
```

Run a container, mounting your working directory to `/home/rstudio`, with:

```
docker run --rm -it -e PASSWORD=password -p 8787:8787 -v
${PWD}:/home/rstudio dsapps_r bash
```

We've added here:

- the `--rm` flag for telling Docker to delete a container once exited
- the the `-e` flag for adding environmental variable such as the password for RStudio (required)
- the `-p` flag for mapping your local port 8787 to the port where the container will run RStudio in (see soon)

```
PS C:\Users\gsimc\test_repo> docker run --rm -it -e PASSWORD=password -p 8787:8787 -v ${PWD}:/home/rstudio dsapps_r bash
root@865079fb1ff9:/# ls home/rstudio
Dockerfile  Dockerfile-R  file1.txt  kitematic  README.md
root@865079fb1ff9:/#
```
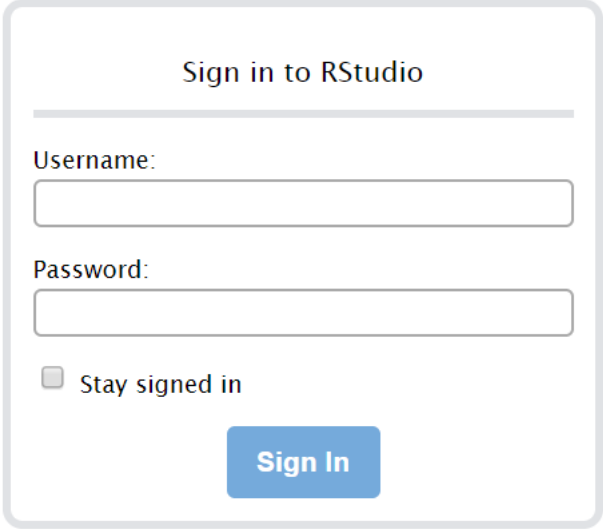
We're now in the terminal, we can see our test_repo files are in `/home/rstudio`. The files shown here aren't very interesting, but you get the potential.

To start RStudio simply enter `/init` (alternatively do not enter `bash` at the end of the previous command and you'll get straight to starting RStudio):

```
root@865079fb1ff9:/# /init
[s6-init] making user provided files available at /var/run/s6/etc...exited 0.
[s6-init] ensuring user provided files have correct perms...exited 0.
[fix-attrs.d] applying ownership & permissions fixes...
[fix-attrs.d] done.
[cont-init.d] executing container initialization scripts...
[cont-init.d] add: executing...
Nothing additional to add
[cont-init.d] add: exited 0.
[cont-init.d] userconf: executing...
[cont-init.d] userconf: exited 0.
[cont-init.d] done.
[services.d] starting services
[services.d] done.
```

The container seems to hang but this is normal behavior. In your browser (Google Chrome preferably) go to http://localhost:8787/

RStudio is asking you for username and password, input "rstudio" and "password" (if that's indeed the password you used when running the container):
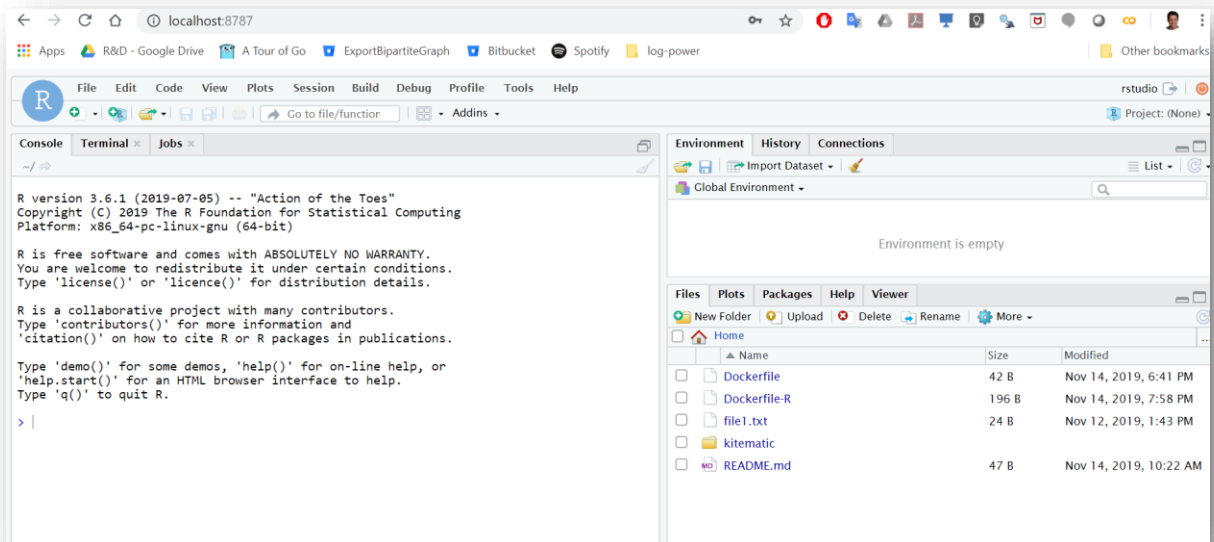
You are now operating RStudio without installing it locally, with all the packages installed for you and your local files at your side. You can run course notebooks, do assignments – Isn't that exciting?

Let's create a silly data frame, sink it to local and see it's there after exiting the container:

```
> library(tidyverse)
── Attaching packages ──────────────────────────────── tidyverse 1.2.1 ──
✓ ggplot2 3.2.1     ✓ purrr   0.3.3
✓ tibble  2.1.3     ✓ dplyr   0.8.3
✓ tidyr   1.0.0     ✓ stringr 1.4.0
✓ readr   1.3.1     ✓ forcats 0.4.0
── Conflicts ───────────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
> silly_df <- tibble(col1 = letters[1:10], col2=runif(10))
> silly_df
# A tibble: 10 x 2
   col1    col2
   <chr> <dbl>
 1 a      0.787
 2 b      0.818
 3 c      0.389
 4 d      0.779
 5 e      0.319
 6 f      0.471
 7 g      0.970
 8 h      0.596
 9 i      0.452
10 j      0.465
> getwd()
[1] "/home/rstudio"
> write_csv(silly_df, "silly_df.csv")
>
```

Now simply close the browser tab and in the terminal terminate the container by pressing Ctrl-C. See that the file is indeed in your working directory:

```
Hangup
root@865079fb1ff9:/# exit
exit
PS C:\Users\gsimc\test_repo> ls


    Directory: C:\Users\gsimc\test_repo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/14/2019     8:31 PM                .rstudio
d-----        11/14/2019     8:06 PM                kitematic
-a----        11/14/2019     8:31 PM             67 .Rhistory
-a----        11/14/2019     6:41 PM             42 Dockerfile
-a----        11/14/2019     7:58 PM            196 Dockerfile-R
-a----        11/12/2019     1:43 PM             24 file1.txt
-a----        11/14/2019    10:22 AM             47 README.md
-a----        11/14/2019     8:33 PM            223 silly_df.csv


PS C:\Users\gsimc\test_repo>
```
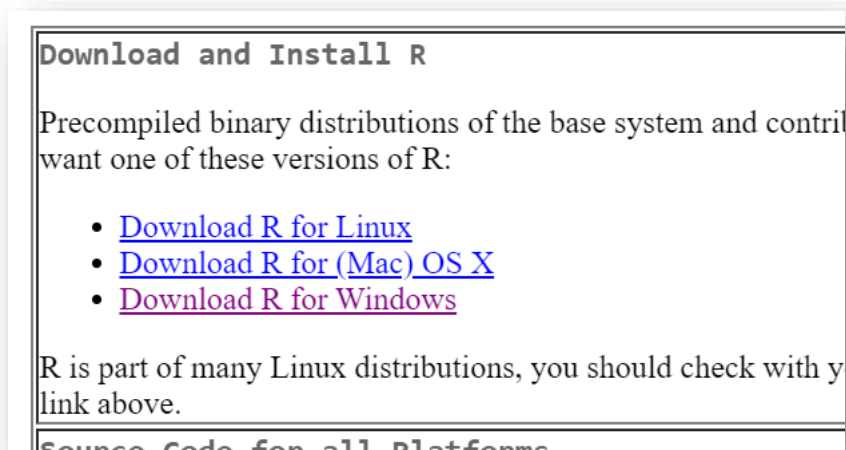
4. R

4.1. Install R

(This should also work if you already have R installed and you want to update it – you would simply have another version of R on your machine – but in this case you might want to look at the installr package)
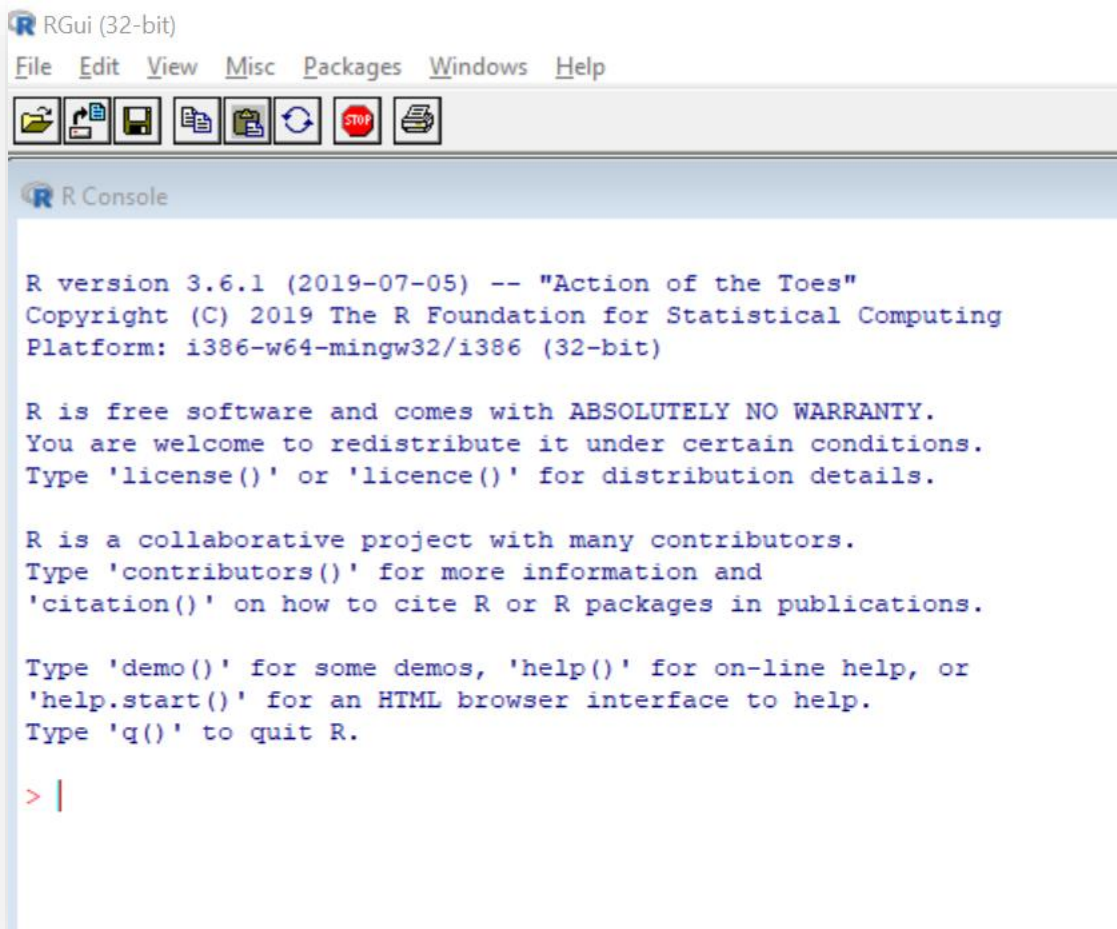
Go to https://cloud.r-project.org/.

Windows users: choose "base", then "Download R 3.6.2 for Windows".

Mac users: pretty sure you need R-3.6.2.pkg (unless by the time you read this it is 3.6.3…), probably best to read some of those instructions first.

Follow the installation wizard's steps accepting all defaults. I *should* recommend, even if your computer's main language isn't English, to select it as R's language:
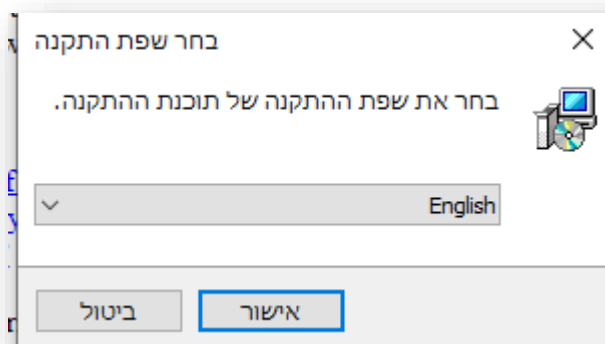
You now have R and R's default GUI ("RGUI"), open it and enjoy (this screenshot shows R 3.6.1 though):

```
R RGui (32-bit)
File  Edit  View  Misc  Packages  Windows  Help

R R Console

R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

For the heavier developers, you might want to start R from the terminal. In order to do that you need to add R to you PATH.

Mac users – this looks like a reasonable link, but the truth is I have no idea 😊

In Windows – there are a few ways of adding a software to your PATH. One way is getting to where you edit "Environment Variables" and add it there.

Windows 7 – Computer icon → Advanced System Settings → Environment Variables → and see similar instructions for Windows 10

Windows 10 – Go to Settings, start typing "envi"…
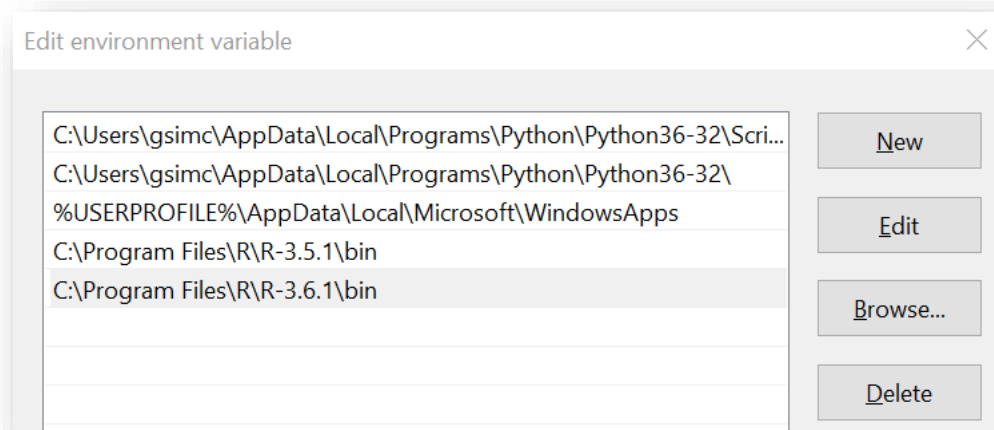
You'll get the "Edit environment variables for your account" option, press it, select PATH:
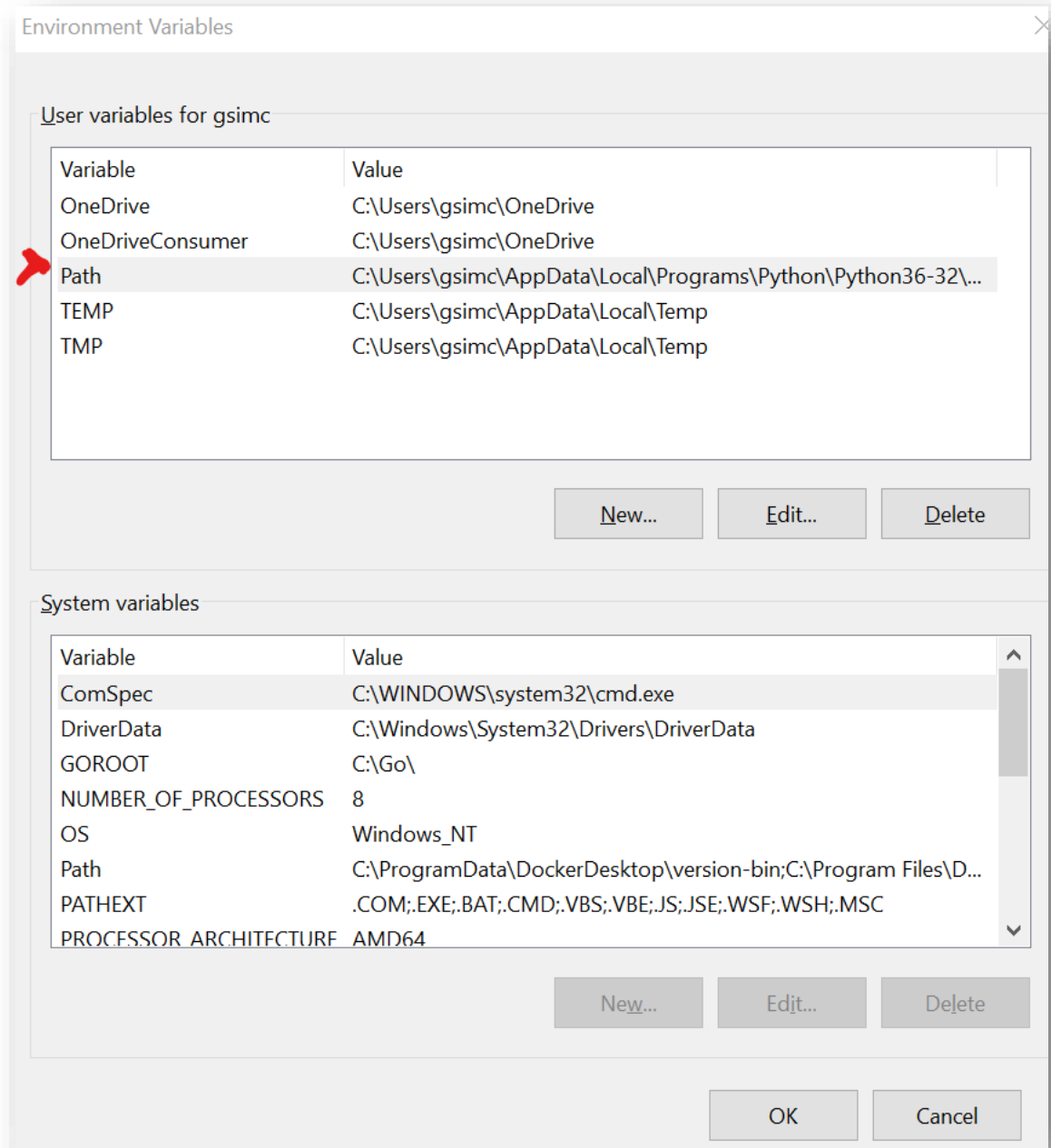
Click Edit and, then New:
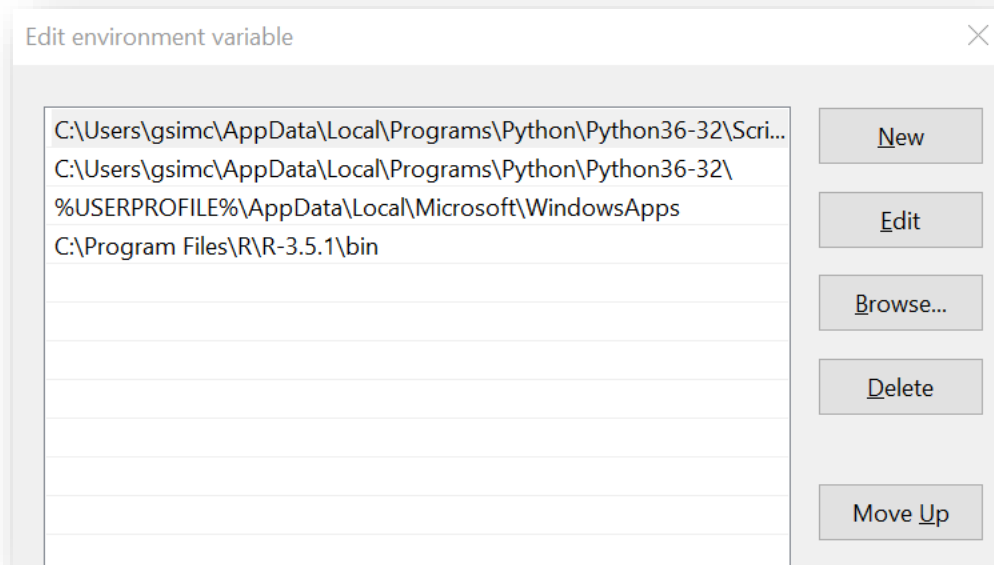
In the text box type the path to your R.exe application, press Enter. In my laptop, this is the default "C:\Program Files\R\R-3.6.2\bin" (this screenshot shows R 3.6.1 though), it will probably be similar on your machine:

Press OK and you're supposed to be able to enter R in the terminal:

```
PS C:\Users\gsimc> R
R : Cannot locate most recent history.
At line:1 char:1
+ R
+ ~
    + CategoryInfo          : InvalidOperation: (:) [Invoke-Histo
    + FullyQualifiedErrorId : InvokeHistoryNoLastHistoryEntryFoun

PS C:\Users\gsimc> R.exe

R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

I think in Linux/Mac just "R" should work, here I had to do "R.exe"…

5. RStudio

5.1. Install RStudio

RStudio is the one of the best IDEs for R out there, and I'm being politically correct. I'm not sure what would happen if you try to install RStudio *before* installing R itself, so let's avoid that, eh?

Go to https://rstudio.com/products/rstudio/download/.

Choose the "Free" program.

RStudio should identify which OS you have and recommend the proper installer, if you scroll down you'll be able to see all latest installers, choose the one for you.

All Installers

Linux users may need to import RStudio's public code-signing key prior to insta

RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you

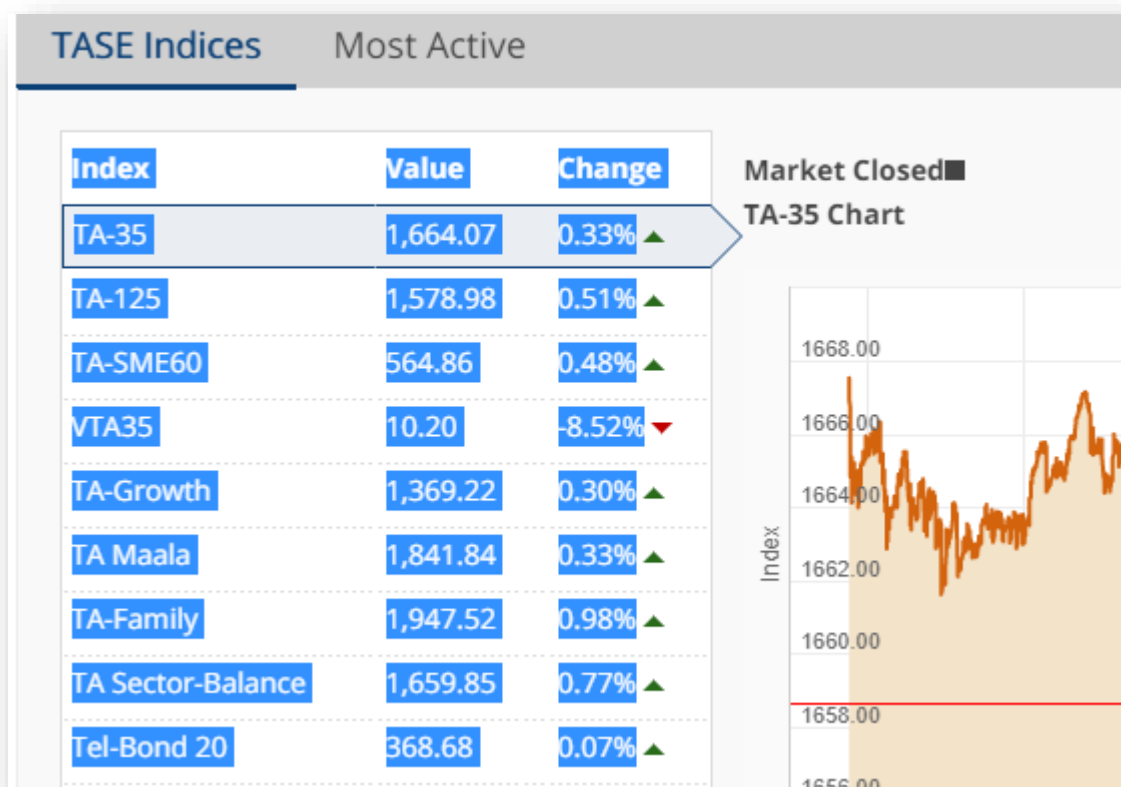| OS	Download
Windows 10/8/7	⬇ RStudio-1.2.5019.exe
macOS 10.12+	⬇ RStudio-1.2.5019.dmg
Ubuntu 14/Debian 8	⬇ rstudio-1.2.5019-amd64.deb
Ubuntu 16	⬇ rstudio-1.2.5019-amd64.deb
Ubuntu 18/Debian 10	⬇ rstudio-1.2.5019-amd64.deb

Go with the installation wizard's flow and open RStudio:
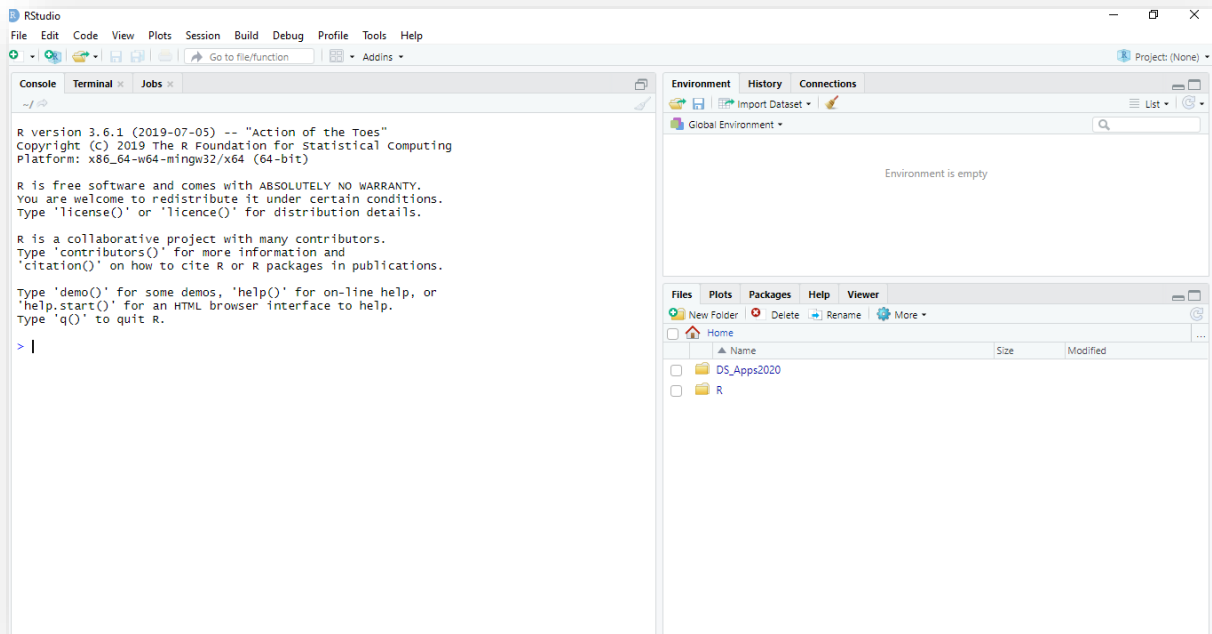
If you've never worked in RStudio, this seems like a good intro.

5.2. Some Necessary Packages

I'm not going to talk about these packages right now, just list a few useful ones. You can install all of them by using the following command (it might take some time; R downloads the package's source and sometime a package would have other packages it depends on…):

```
install.packages(c("tidyverse", "dtplyr", "devtools", "usethis",
"datapasta", "pryr", "testthat", "infer", "shiny"))
```

If you successfully installed the datapasta package for example, let's see a neat trick. Go to the Israeli Stock Exchange English website, mark the first table you see and copy it to clipboard:

Go to RStudio, open a new script, go to "addins", the "DATAPASTA" main, press "paste as tribble":

And there you have it, in your script:

The weird HTML table you've just copied, at your fingers. Is it perfect? No. That's where the Tidyverse comes in handy. Later.

5.3. Use R Notebooks

For a comprehensive look on what is R Markdown in general and what are R Notebooks, see [R Markdown: The Definitive Guide](#).
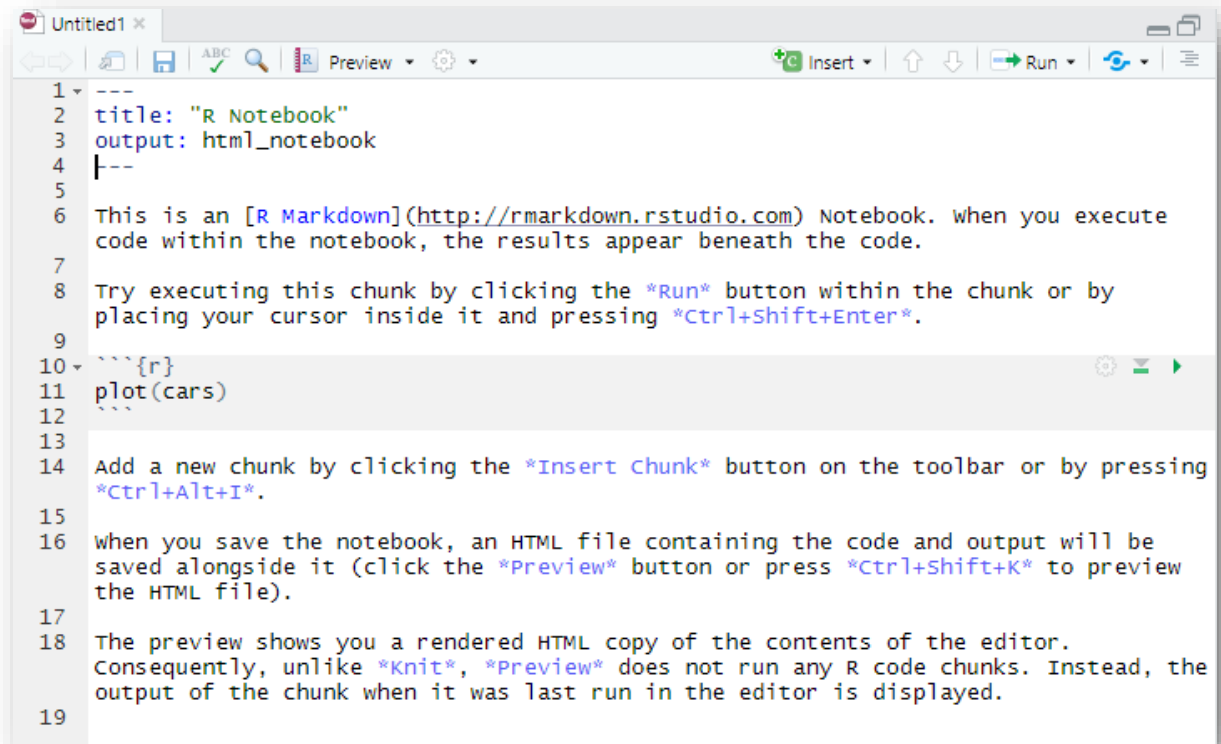
At this stage I need to you to know the basics.

- Notebooks are fully *reproducible* reports, including (formatted) text explaining your analysis in detail, code and visuals
- Text is usually written in markdown syntax (as we saw in the README, not just a R thing)
- Code goes into chunks, which can be run interactively, separated by three backticks
- Both in R and Python, can be easily converted into a presentation, pdf, html file, even a word document

Open one of the course notebooks or a new one:
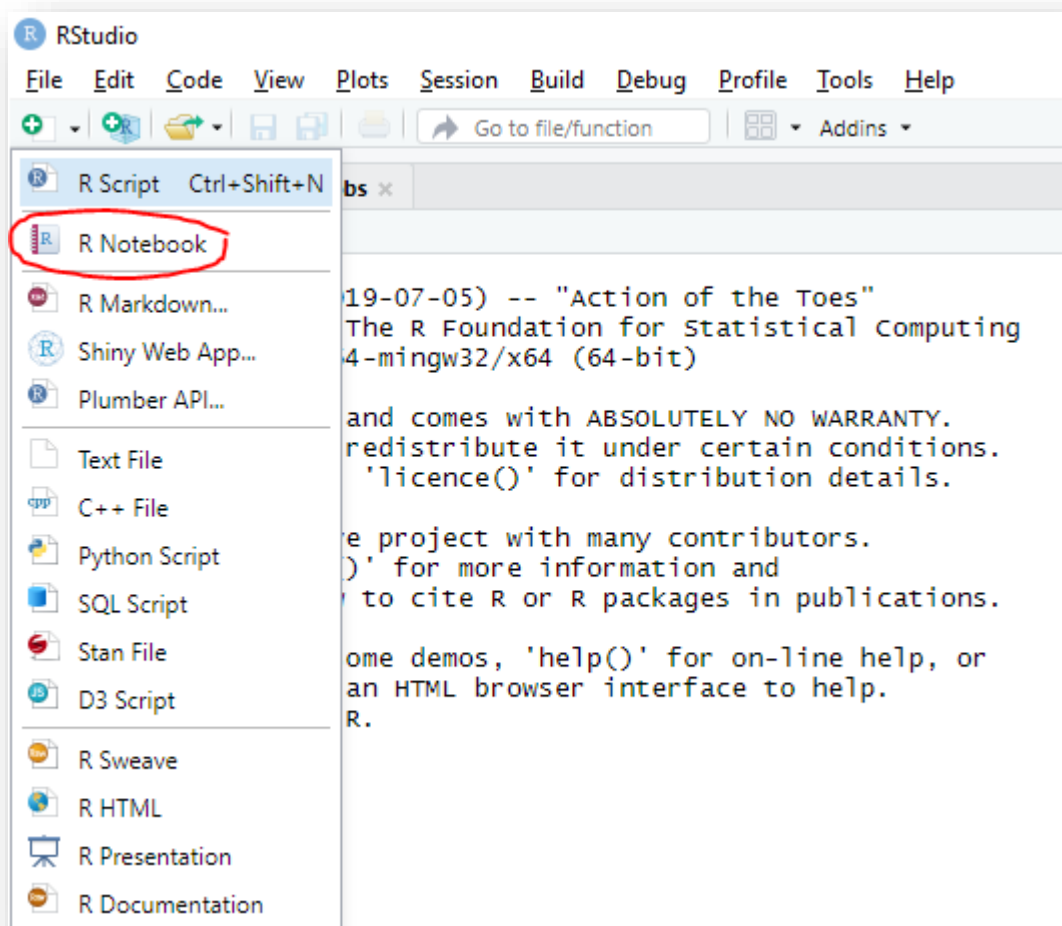
RStudio opens a simple template for a new notebook:

```
1  ---
2  title: "R Notebook"
3  output: html_notebook
4  ---
5
6  This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute
   code within the notebook, the results appear beneath the code.
7
8  Try executing this chunk by clicking the *Run* button within the chunk or by
   placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
9
10 ```{r}
11 plot(cars)
12 ```
13
14 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing
   *Ctrl+Alt+I*.
15
16 When you save the notebook, an HTML file containing the code and output will be
   saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview
   the HTML file).
17
18 The preview shows you a rendered HTML copy of the contents of the editor.
   Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the
   output of the chunk when it was last run in the editor is displayed.
19
```

Read it carefully and don't be shy to try out what it suggests, mainly:

- Running a chunk of code (Ctrl+Shift+Enter)
- Inserting a new chunk of code (Ctrl+Shift+I), putting some code and running it
- Saving the notebook somewhere under test.Rmd (the file extension for R Markdown files)
- Previewing the notebook as a HTML report (Ctrl+Shift+K)
- Showing/Hiding chunks of code
- Adding links, bold text, lists, executable R code (not available in Python)
- Add yourself as an author in the yaml

Much more can be achieved quickly with this R Markdown cheat sheet.

6. Python

NOTE: If Docker is running, you can skip this section, no need to install anything else locally!
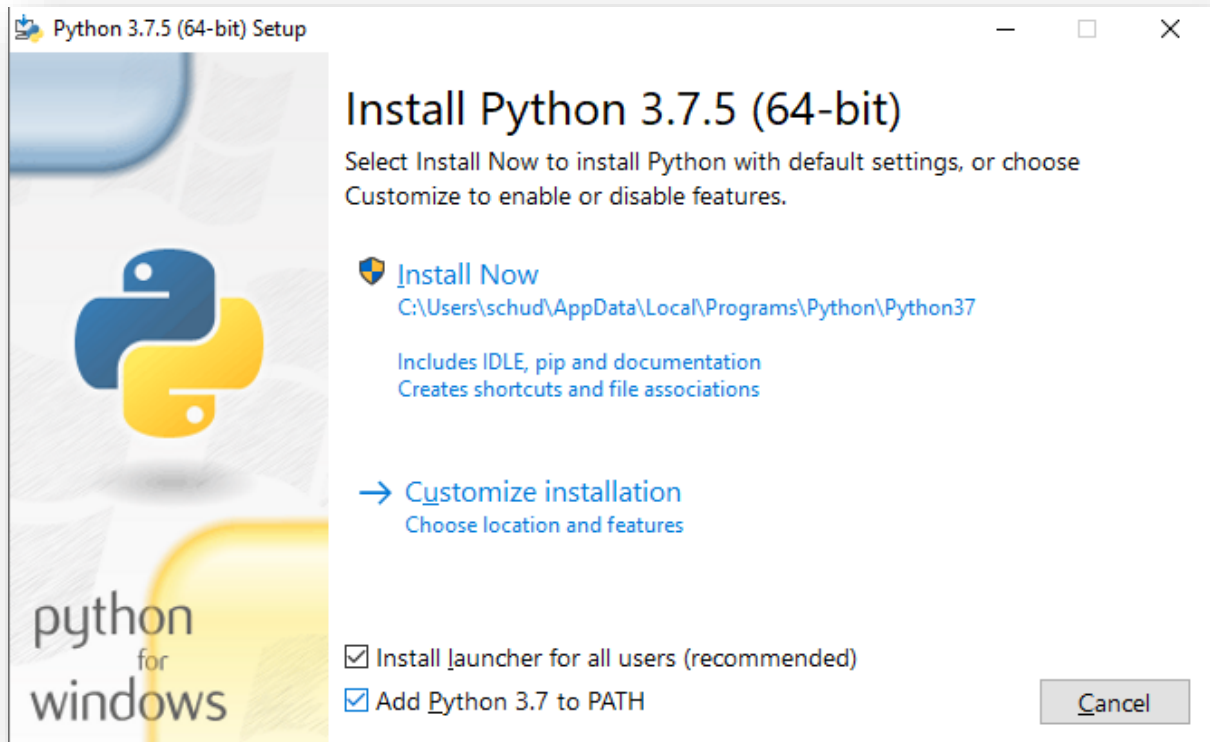
6.1. Install Python

We'll go with Python 3.7.5 (at the time of writing this Python 3.8 won't work when we get to some libraries like Keras over tensorflow). If you have Python 3.7.4 or 3.7.6 all should work as well.

Go to https://www.python.org/downloads/release/python-375/

You have a few installers available:

| Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
macOS 64-bit/32-bit installer	Mac OS X	{Deprecated} for Mac OS X 10.6 and later
macOS 64-bit installer	Mac OS X	for macOS 10.9 and later
Windows help file	Windows	
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64
Windows x86 embeddable zip file	Windows	
Windows x86 executable installer	Windows	
Windows x86 web-based installer	Windows	

See if your system is 32 or 64 bit (e.g. like this) and choose the appropriate "executable installer".

When asked, add Python to your PATH (otherwise you'd have to do it manually like in R):

Once installation is done you should have Python's default GUI called Idle:
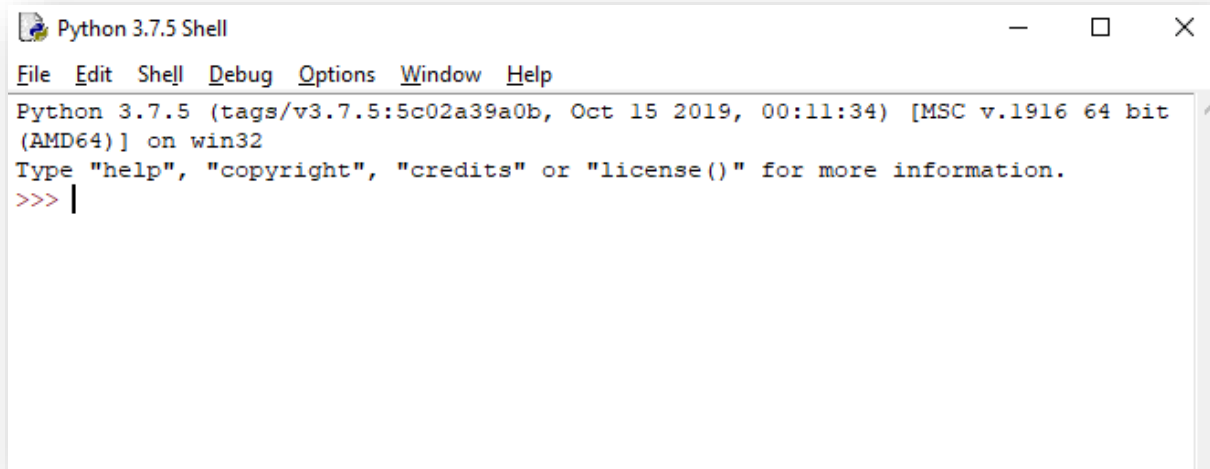
Make sure you can call Python from the terminal, e.g. in Powershell:
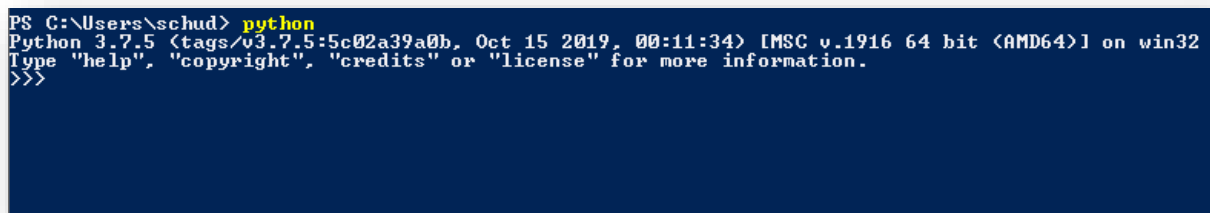
(In case you also have Python2 installed, you would probably need to open Python3 with "python3", not just "python", and later use "pip3 install", not just "pip install")

Some possible IDEs for Python (though we will be working with Jupyter notebooks only so you don't have to use them):
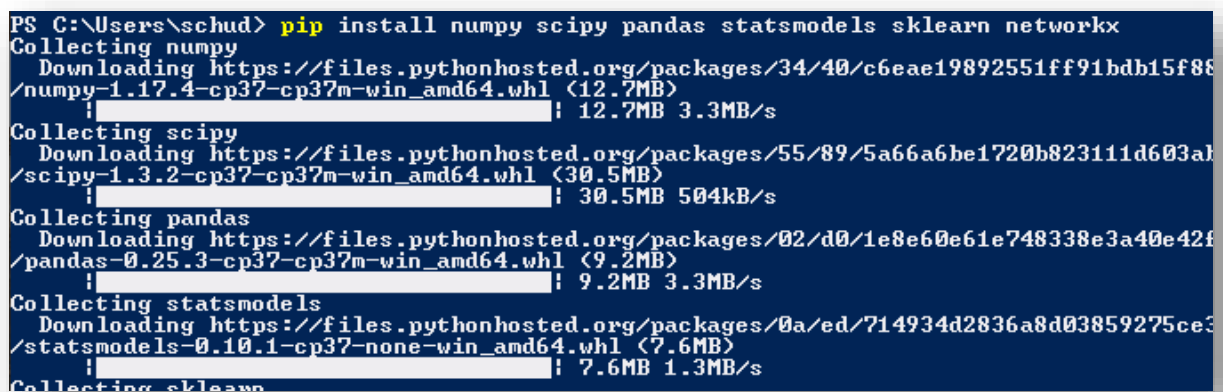
- [Pycharm](#) – what I use for work
- [Spyder](#) – an obvious copycat of RStudio…
- [Rodeo](#) – another obvious copycat of RStudio…
- [RStudio](#) (!) – ever since RStudio v1.2 you can switch between R and Python, maybe we'll get to that

6.2. Some Necessary Packages

You should be able to install Python libraries using `pip install package_name`. In some cases, Python will complain it also needs the `--user` flag.

Like in R, I won't talk about the packages just now, only list them. Let's start with these packages:

```
pip install numpy scipy pandas statsmodels sklearn networkx keras
matplotlib seaborn
```


If all goes well you should get a message like "Successfully installed decorator-4.4.1 joblib-0.14.0 networkx-2.4 numpy-1.17.4 pandas-0.25.3 patsy-0.5.1 python-dateutil-2.8.1 pytz-2019.3 scikit-learn-0.21.3 scipy-1.3.2 six-1.13.0 sklearn-0.0 statsmodels-0.10.1…".

7. Jupyter

7.1. Install Jupyter

Simply do `pip install jupyter`
You will get many other dependencies, don't be alarmed.

7.2. Use Jupyter Notebooks

Open one of the course notebooks or a new one. Go to the course repo or a folder you wish to start a new notebook in and enter `jupyter notebook`:

```
PS C:\Users\gsimc> cd test_repo
PS C:\Users\gsimc\test_repo> jupyter notebook
[I 07:15:44.700 NotebookApp] Serving notebooks from local directory: C:\Users\gsimc\test_repo
[I 07:15:44.700 NotebookApp] 0 active kernels
[I 07:15:44.700 NotebookApp] The Jupyter Notebook is running at:
[I 07:15:44.700 NotebookApp] http://localhost:8888/?token=ebb2eade0fc020f08a48ffe4a8dc4af163d2b0fb
[I 07:15:44.700 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to
[C 07:15:44.702 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=ebb2eade0fc020f08a48ffe4a8dc4af163d2b0fb4ceeb7b0&token=ebb2ea
4af163d2b0fb4ceeb7b0
[I 07:15:45.243 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

You should get redirected to a new tab in the browser showing you the GUI to the Jupyter server in your folder. If not, do exactly what it says, copy-paste the http://localhost:88/?token=ebb... address to your browser.
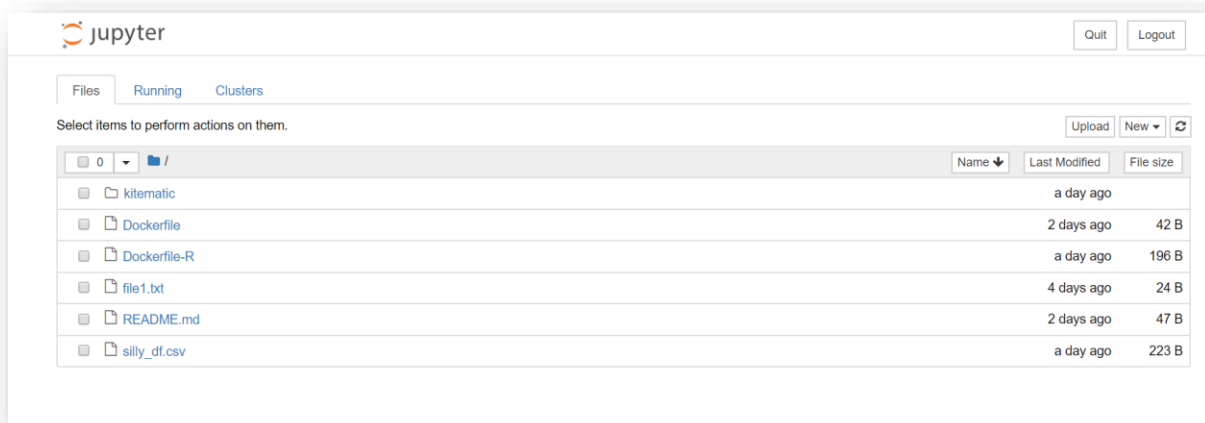
Once there you either open an existing notebook or start a new one pressing the "New" button. Jupyter should ask you whether you'd like a "Python 3" notebook (or Python 2 if you have both on

your machine), a folder, a terminal. Choose "Python 3" and Jupyter automatically opens a new notebook called "Untitled" in a new tab, using the Python version you're working in:
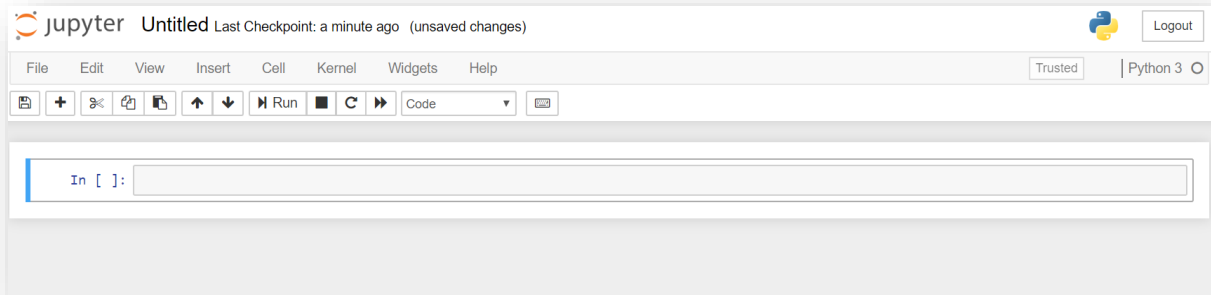
Again, not going to go through *everything* you can do in a notebook, but here are some highlights:
- Change the name of the notebook just by pressing "Untitled", say to "test"
- In the code chunk write some Python expression and do Ctrl+Enter to perform the expression
- Press "B" (as in "Below") to add another cell (when you'd like "Above" guess what you should press)
- Press "M" to make that cell "Markdown".
- Press Enter to enter the cell, type "$$c = \sqrt{a^2 + b^2}$$" to enter a LaTeX expression, Ctrl+Enter
- Another "B" for a new cell, then "M"
- Press Enter to enter the cell, type "# A Heading1 Title" to enter a title, Ctrl+Enter
- Another "B"
- Press Enter to enter the cell, type `%ls` to see a list of files in the current folder
- In the menu go to Kernel → Restart and run all

You should get something like:
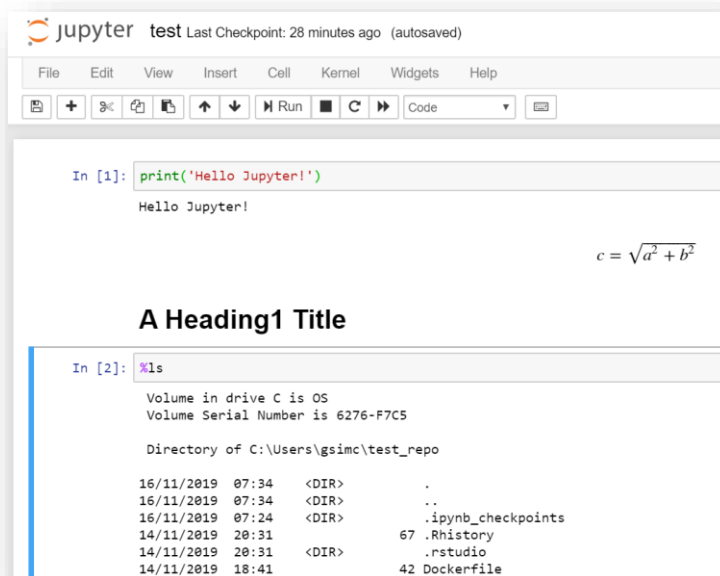
For a much more comprehensive tour of Jupyter notebooks, this looks nice, this cheat sheet is OK.