HOW TO:

INSTALL JULIA & THE OWCF

(ON MAC OS X)

BY HENRIK JÄRLEBLAD

HOW TO:

INSTALL JULIA & THE OWCF (ON MAC OS X)

BY HENRIK JÄRLEBLAD

Alright, so this is a guide on how to successfully install the Julia programming language and the Orbit Weight Computational Framework (OWCF), on MacOS X (any new MacOS operating system).

This guide will most likely get updated continuously, as the programming languages and packages evolve. Also, as of this version, **the estimated amount of space you need on your hard drive (Julia language + packages) is approx 1GB**. **Please make sure you have that available (preferably more)**.

Please check the date below, to find out if you need to acquire a more recent version of this guide.

This version was written on 14th of October 2022.

Henrik Järleblad
+45 3 11 55 767
henrikj@dtu.dk
Fysikvej, Building 309
2800 Kgs. Lyngby
DTU
www.henrikjarleblad.se

A CODE EDITING PROGRAM

So, there are many programs which you can use to edit your code. And many integrated development environments (IDE) also offer the possibility of executing the code directly after writing it in the editor. Such as Eclipse, Atom (for Julia, Juno), IntelliJ and many, many more.

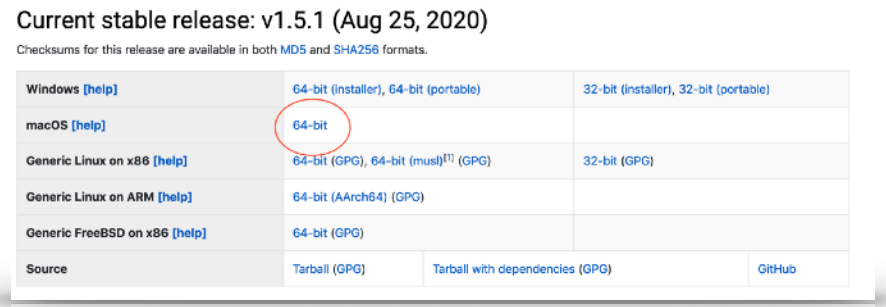
I suggest that you start by doing some googling, to find out if the IDE you are currently using is compatible with the Julia programming language. If not, then I suggest you download visual studio code (VSCode), the most basic but still sophisticated code-editing program in the world (in my humble opinion). Just google “visual studio code download”, and you should find your way. What you will then do, when working with the Julia orbit tomography packages, is writing the code in VSCode, and executing it using the MacOS command line. I will show you how to do this. However, you could also execute code and code blocks directly in VSCode using the Julia VSCode extension. You can read about that here https://www.julia-vscode.org/docs/stable/gettingstarted/ and here https://www.julia-vscode.org/docs/stable/userguide/runningcode/ .

You can of course use your own IDE. Just make sure that it is compatible with Julia.

1.
DOWNLOADING JULIA

To download the Julia programming language, please go to https://julialang.org/downloads/ and choose the latest stable release for MacOS, and then the 64-bit version. Go ahead and download that file. PLEASE NOTE! It is ok if it's a newer version than v1.5.1. In fact, the newer the better.

If you want to be really safe that you have got the correct file, distributed by the officials of the Julia programming language (and that no-one has hacked their website, and replaced the downloadable files with a virus or something similar), you should verify that you get the same SHA-256 hash as they claim. The risk that the official website has been hacked is of course very low. But still, it is not zero. Please refer to the last section of this document, to find a short guide on how to verify the downloaded installation file using SHA-256.

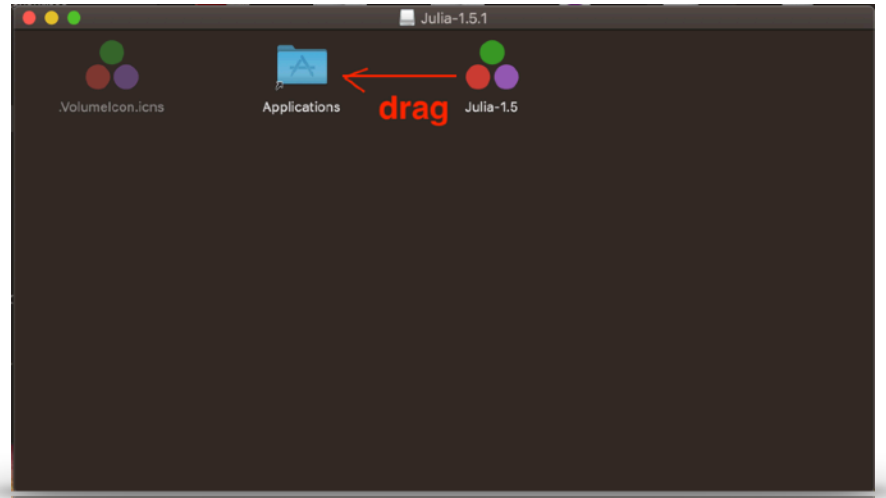


Current stable release: v1.5.1 (Aug 25, 2020)

Checksums for this release are available in both MD5 and SHA256 formats.

Windows [help]	64-bit (installer), 64-bit (portable)		32-bit (installer), 32-bit (portable)
macOS [help]	64-bit		
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl)[1] (GPG)		32-bit (GPG)
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)		
Generic FreeBSD on x86 [help]	64-bit (GPG)		
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

Alright, you should now have the correct Julia installation file in your downloads folder. Go ahead and install it, follow the instructions (if any). Usually when installing something on Mac, the last step is to visually drag the program (represented by an icon) to the Applications folder. Do this, and your Julia installation should be complete (shown below for the 1.5.1-version of the Julia programming language).



The next step is to add the Julia programming language to the MacOS PATH variable. That way, when your IDE or your MacOS command terminal tries to execute “julia”, they will find the correct file to execute.

2.
ADDING JULIA TO PATH

So what you should do now, if you haven't done so already, is enable your MacOS Finder (file browser) to show hidden files. This will simplify things. If you have already enabled the option to display hidden files on your Mac, you can skip to the next big letter.

First of all, WikiHow has a great guide on how to easily enable the option to display hidden files on your Mac. https://www.wikihow.com/Show-Hidden-Files-and-Folders-on-a-Mac If this link still works, use it, and follow Part 1 but not Part 2.

Otherwise, basically open your Finder (file browser) on your Mac. Look in the top-left corner. You should see "Go" next to "View". Click it, and click "Computer". You should now see an icon representing your hard drive (Macintosh HD or something similar). Double-click it. Now, here is the trick. Press cmd+shift (the up-arrow) + dot (.). It's important to do it in that order or shift+cmd+. . If you did it correctly, all the hidden files on your Mac should now appear.

Go to your home folder. This is usually the one you named yourself. If you are unsure, open your Finder (file browser), click "Go" in the top-left corner, click "Home" and you should be there. Now, locate your ".bash_profile" file. This will only be visible if you have enabled the option to display hidden files. Open it with TextEdit (or a text-editor of your choice. VSCode works of course). You should see a bunch of lines. If not, please use the text written in the box below the first one below. Create a

new line and write (**changing the "Julia-1.5.app" part to the version you are using. So if you are using 1.6, you write "Julia-1.6.app", and if you are using 1.7, you write "Julia-1.7.app" and so on**)

```
export PATH=$PATH:/Applications/Julia-1.5.app/Contents/Resources/julia/bin
```

or, if you have no lines in your ".bash_profile" file, write

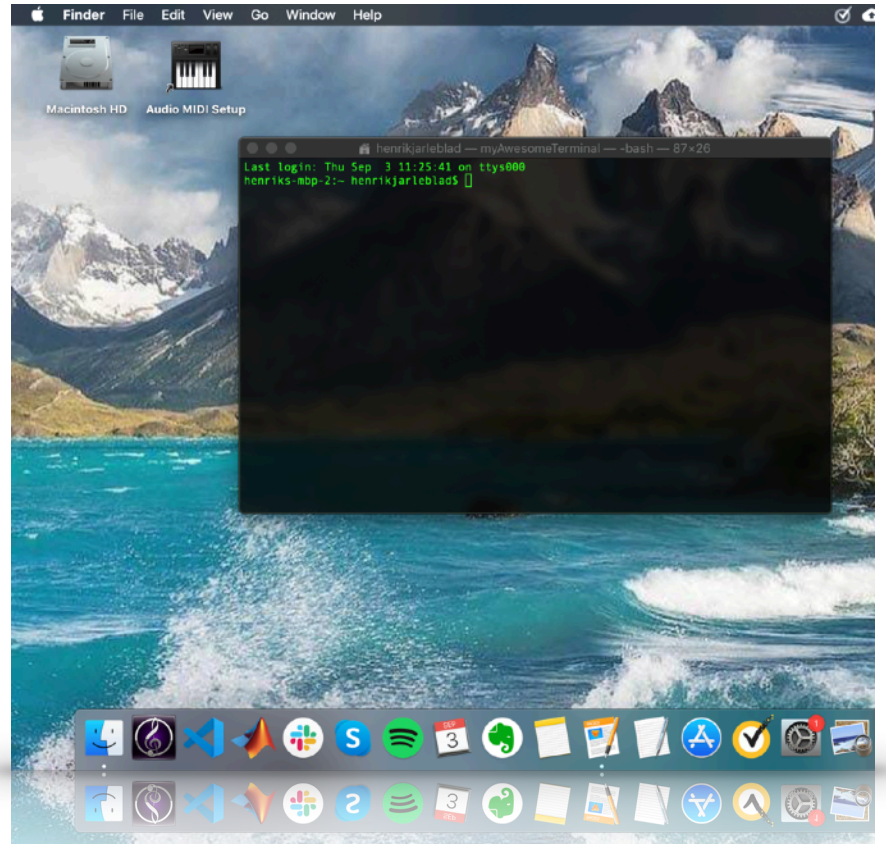
```
export PATH=""
export PATH=$PATH:/Applications/Julia-1.5.app/Contents/Resources/julia/bin
```

Save the file, and move on to the next section.

3.
INSTALLING PACKAGES

Here is where we start using the MacOS command terminal. Even if you are planning on using Julia and the OWCF in an IDE, you can still benefit from reading this section, because all the packages should be installed and ready-to-use when you start up your IDE later. However, Julia has an environment functionality. So, for every project you work on, Julia has the possibility of loading different packages. What I want to say is, that even if you follow the steps in this section and use your IDE afterwards, you might still run into some problems. Unfortunately, I do not know which IDE you are using, so I have no way of helping you there. Please turn to the almighty google (or duck-duck-go) if that should be the case.

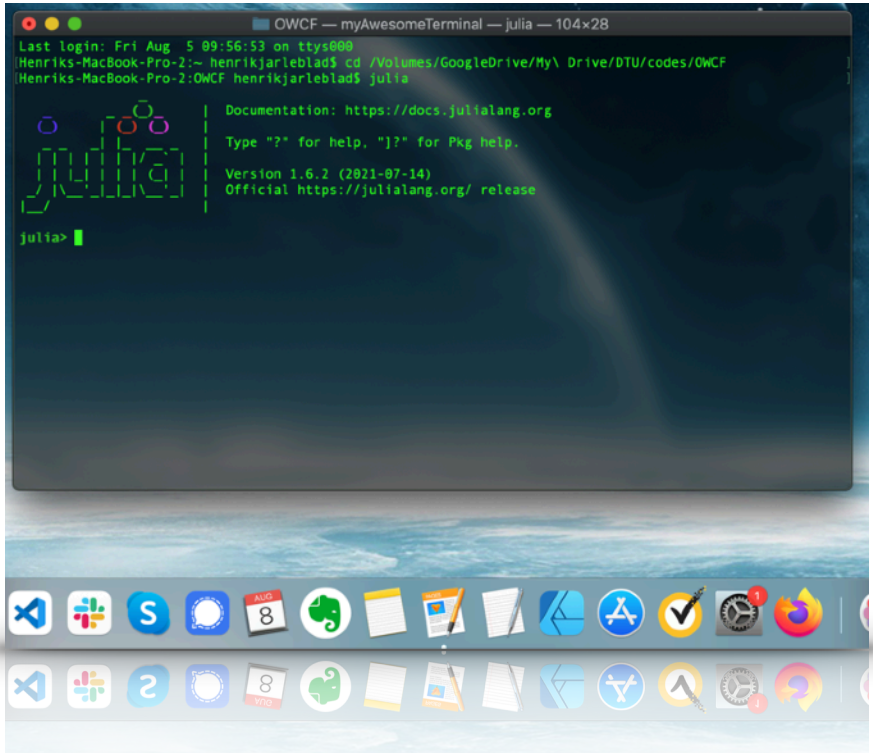
Alright, let's start. Go to your Mac desktop (or start a new one, if you're a multi-tasker like me) and press cmd+space. Type "terminal". Press enter. You should now have opened a terminal in your Mac home folder.



Don't worry if your terminal doesn't look exactly like mine (colors, text, opacity etc). I customised mine. You can too, if you want to.

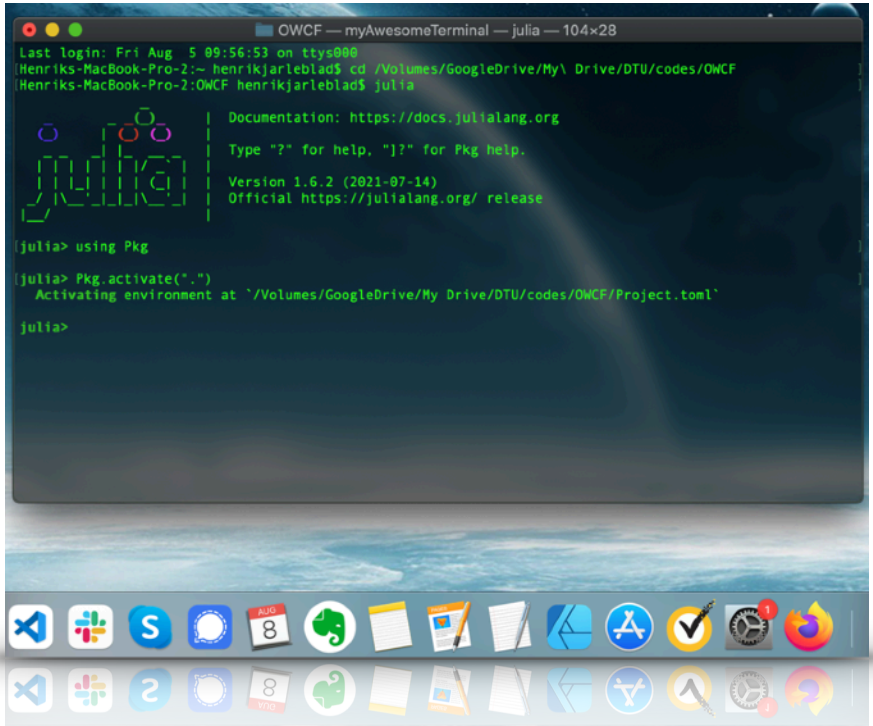
Anyway, make sure the little square is blinking (it means the terminal is waiting for your input, if you are completely new to this. If it is not blinking, just click on the terminal window). Make sure that you have unzipped your download of the OWCF.zip folder. It's a good idea to save it in a useful folder

on your computer, because you will be using it a lot. Use the 'cd' command to navigate to the OWCF folder, and start Julia by typing 'julia' and hit enter, as the screenshot below shows. **PLEASE NOTE!** Your OWCF folder will **not** be "/Volumes/GoogleDrive/My\ Drive/DTU/codes/OWCF/". It is only an example. Please find out where you have saved and unzipped your OWCF-folder, and use the 'cd' command to navigate there!



Now, type 'using Pkg', hit enter, 'Pkg.activate(".")' and hit enter. Press ']' to confirm that you have indeed activated the OWCF environment. As the screenshot below shows. Press backspace to go back to normal Julia mode. The ']' key in Julia activates package mode. Read more about that by Googling/Duck-duck-going 'Julia Pkg.jl'.

Here we pause for a bit, because I need to tell you something **before you proceed further and start installing any**



packages (more specifically, before you install the PyCall.jl package). If you already have Python installed on your Windows, you should let Julia know of the path to the Python executable prior to installing the PyCall.jl package. You do this by typing

ENV[“PYTHON”] = "... path of the python executable ..."

into the Julia prompt. You can find out the path of your Python executable by starting Python, type ‘import sys’, hit enter, type ‘print(sys.exec_prefix)’, hit enter, and note the path. You then add ‘/python.exe’ to the end of the path when specifying the ENV[“PYTHON”] Julia information. In my case (most likely **not in your case**), this means

ENV[“PYTHON”] = “/usr/local/Cellar/python/3.7.1/Frameworks/Python.framework/Versions/3.7/python.exe”

Also, remember, if you want to write Python code directly in Julia (yep, that can be done with Julia) you have to make sure that the Python packages are installed in your Python (so for example, if you want to write Python code in Julia that uses the *h5py* Python package, you have to have it installed in Python first). The OWCF actually uses the following Python packages

- numpy
- scipy
- netCDF4
- h5py

so please make sure that you have those installed in Python before trying to use the OWCF. To connect Julia with your Mac's pre-installed Python executable, after you have done the 'ENV["PYTHON"]="..."' step, do the following

- Pkg.add("PyCall")
- ENV["PYTHON"] = "... path of the python executable ..."
- Pkg.build("PyCall")

The second step is often redundant (it's exactly the same as the earlier step). But it's good to perform anyway, just in case. As Figure A below shows.

Now, back to installing Julia packages and the OWCF. For the next step, simply press ']', type 'instantiate' and hit enter. As Figure B below shows.

This will be a good time to go grab a coffee, or even go for lunch. Julia is installing and pre-compiling all the packages needed for the OWCF (yes, **pre**-compiling. It is unique to Julia. It optimises the efficiency and speed of the Julia language). This long process might instead have happened during the previous Julia-Python connection steps.

Anyhow, when you come back, you should now have all the packages you need and be ready to start using the OWCF. Congratulations!

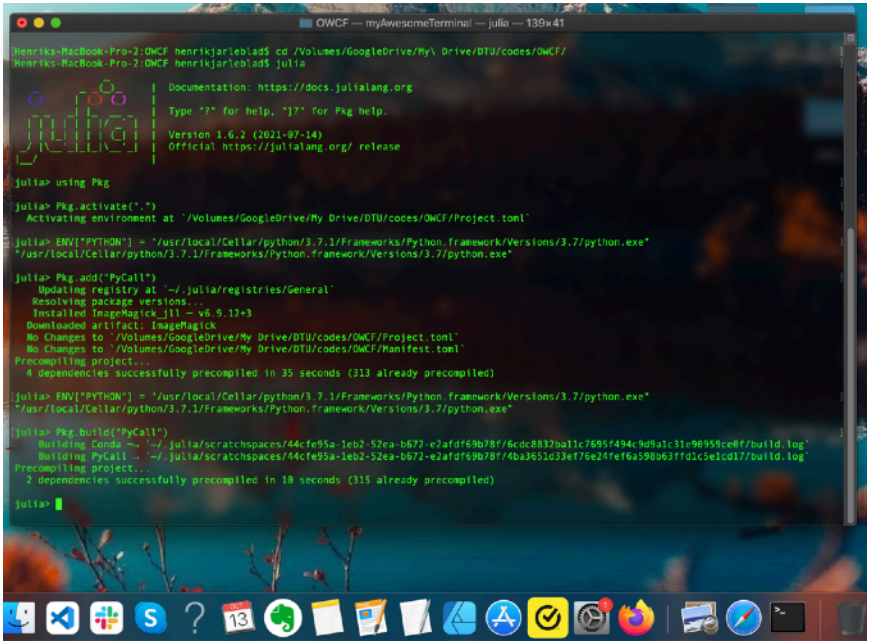


Figure A: How to connect Julia with your Mac's pre-installed Python executable. Your log prints are likely going to differ significantly from those in the screenshot. The important thing is that there are no error messages.

Figure B: How to instantiate and finalise the OWCF installation. Your log print are likely going to differ from those in the screenshot. The important thing is that there are no error messages.

If you have not used Julia before. I would advice you to at least read up on the basics on the www.julialang.org website. For example, how do you close Julia down now that you have installed everything?

4.

WRITING AND EXECUTING YOUR VERY FIRST JULIA FILE

You can write code in so many different ways and using so many different code editing programs. Heck, if you want to use the basic Mac TextEdit program, go ahead. However, I will show you how to use the VSCode code editing program to write the code and the MacOS terminal to execute it. VSCode now also supports execution of Julia code and plotting directly in the IDE, so I will show you how to do that as well. Let's get started.

First, open an empty document and save it as 'test.jl' (note the '.jl' file extension). Next, write these lines of code to change directory to the OWCF/ folder and activate the environment.

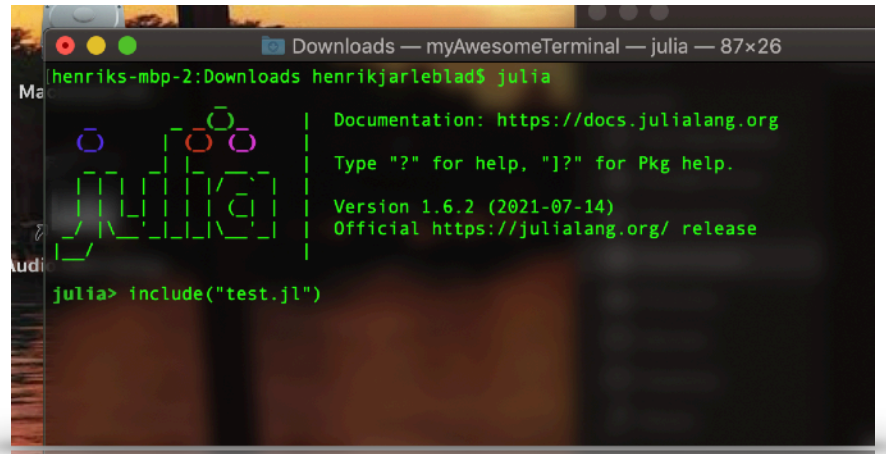
```
1   folderpath_OWCF = "/the/path/to/folder/of/your/OWCF/"
2   cd(folderpath_OWCF)
3   using Pkg
4   Pkg.activate(".")
```

Next, write the following lines to load the *Plots.jl* package, generate a 1D array of integers (x) between 1 and 50, a 1D array of 50 random floats (y) and plot y as a function of x (please see the figure at the top of the next page).

```
5    using Plots
6    x = collect(1:50)
7    y = rand(50)
8    Plots.plot(x,y)
```

In Julia, you can generate a range without having to allocate memory for all the elements. This is done using the ':' and via the syntax '*start:increment:stop*'. If you don't provide an increment, Julia will simply assume evenly spaced elements. Then, if you want Julia to actually allocate memory for the range as an array, you use the *collect()* function. This will convert the type of the variable from a *Range* to an *Array*. Use the function *typeof()* anytime you want to find out what type a variable is. Now, let's execute your cute little first script!

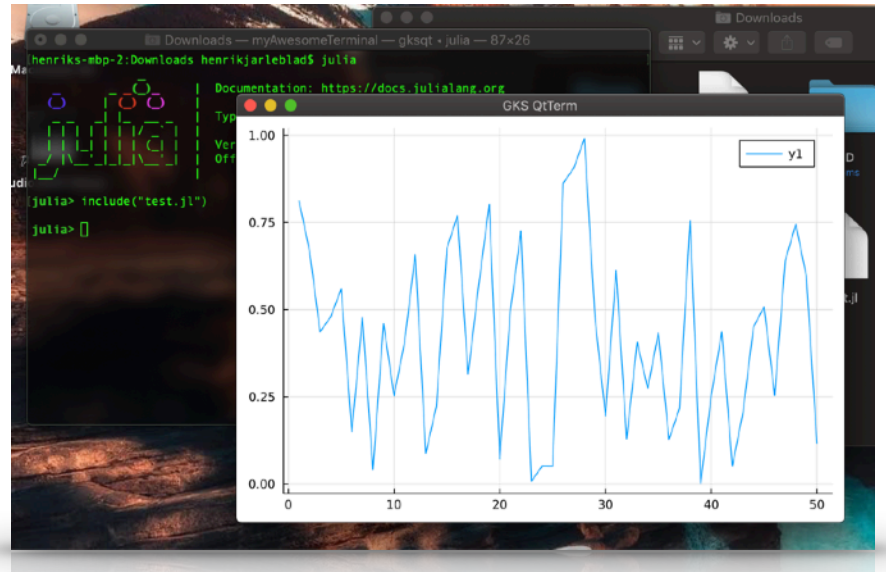
Open a MacOS terminal and navigate to the folder where you have saved your 'test.jl' Julia script (you navigate using the 'cd [TARGET FOLDER]' command). Then, you can either start a Julia session with 'julia' and execute the script in Julia with the command 'include("test.jl")' or simply typing 'julia



How to start a Julia session and execute a .jl script

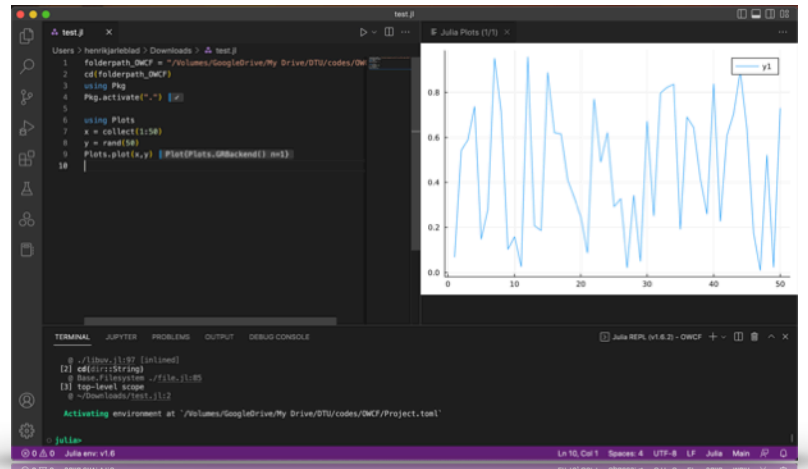
test.jl' directly in the MacOS terminal. The former way of doing it might be more suitable for plotting. So go ahead and try it!

If you did it correctly, after a couple of seconds a pop up window should appear (powered by the GKS QtTerm program) showing you the plotted data. The reason that it takes a couple of seconds the first time is that Julia loads the whole *Plots.jl* package with the 'using Plots' code line. If you have started an active Julia session, these needs to be done only once in the beginning. So go ahead and try executing the script with 'include("test.jl")' and you should see the plot window immediately appearing. When you're done, don't forget to type 'exit()' into the Julia session REPL (prompt) to close Julia and ⌘Q while highlighting the GKS QtTerm to close the plotting functionality.



Now, there is also a way of plotting directly in VSCode, using the Julia extension and integrated plotting functionality. Let's take a look at how to do that.

Open your 'test.jl' script using VSCode. If you look to the left, you should see a symbol that resembles three connected boxes and a fourth disconnected box. These are your VSCode extensions. Click on the symbol. Search for 'julia' in the search bar and download the Julia VSCode extension (Julia Language support). After it has installed, you might need to restart VSCode. Then, as explained here https://www.julia-vscode.org/docs/stable/userguide/runningcode/, put the curser at the end of your first line (so right after the second 's' in the 'using Plots' line, before the line break) and hit ctrl + enter. A terminal will open up in VSCode and execute the code line. Then, as explained on the website linked above, you can use different delimiters ('##') and different commands to execute your Julia code in VSCode. For now, just hit ⇧ + enter and watch how the plot window appears to the right, integrated right into VSCode.

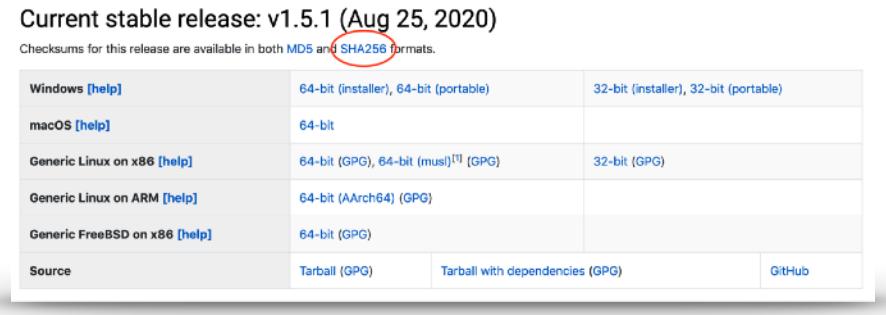


Plotting with VSCode

That is it! Good luck with your Julia programming!

EXTRA: USE SHA-256 TO VERIFY THE DOWNLOADED JULIA FILE

To verify the installation file with SHA-256, start by downloading the SHA-256 file, just above the Julia programming language installation files (https://julialang.org/downloads/)



Current stable release: v1.5.1 (Aug 25, 2020)

Checksums for this release are available in both MD5 and SHA256 formats.

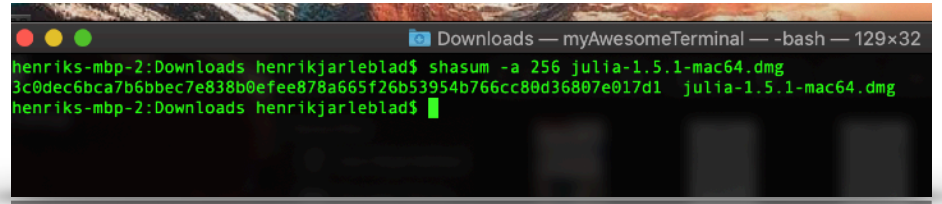
Windows [help]	64-bit (installer), 64-bit (portable)		32-bit (installer), 32-bit (portable)
macOS [help]	64-bit		
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl)[1] (GPG)		32-bit (GPG)
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)		
Generic FreeBSD on x86 [help]	64-bit (GPG)		
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

Make sure you have both the Julia installation file and the .sha256 file in the same folder (presumably your Downloads folder?). Open a MacOS command terminal (cmd+space, type “terminal”, hit enter). You should be in your home folder. Type “cd Downloads” (or navigate to the folder where you put the files. If you put them on your desktop, type “cd Desktop”). Hit enter. Now, you need to perform SHA-256 on the installation file to generate a bunch of letters and numbers. In your command terminal, type

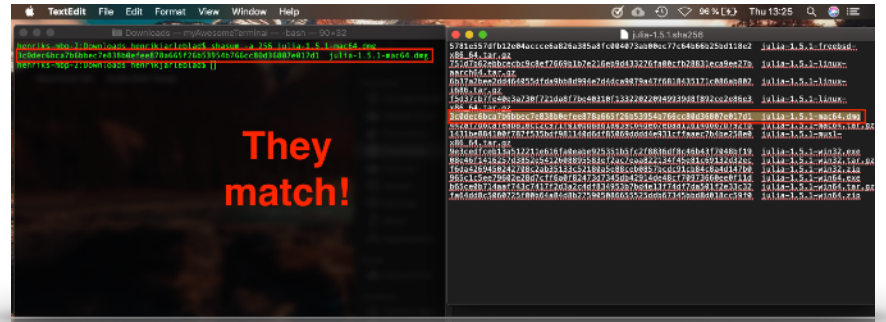
```
shasum -a 256 julia-1.5.1-mac64.dmg
```

where you change the “julia-1.5.1-mac64.dmg” filename to whatever version you’re currently attempting to install. (So if you want to install Julia 1.6.1, you write “julia-1.6.1-mac64.dmg”

instead and so on). Hit enter. This should generate a bunch of letters and numbers, as shown below (could be different).



Your job is now to make sure that these letters and numbers match those that the Julia language officials have written in the .sha256 file. So, open the .sha256 using TextEdit (or a text-editor of your choice). Compare the two sets of letters and numbers. Make sure that they are identical. If they are not, you might have just downloaded a virus. Don't open it.



A FEW EXTRA THOUGHTS

When you use a package for the first time, it might need to do something unique (sort of) to Julia. It might need to pre-compile. This takes a loooong time (especially for large packages) but the result is that many things that would need to be compiled in other languages is already compiled to machine code. This is one of the things that makes Julia so fast (in some cases, as fast as C). So don't worry if pre-compiling some packages takes a long time. It's normal. Just wait and take a coffee or similar. Julia will be super-fast afterwards. As always. Except for plotting. Julia is super slow sometimes when it comes to plotting. At least for now (Julia v1.6-v1.7).