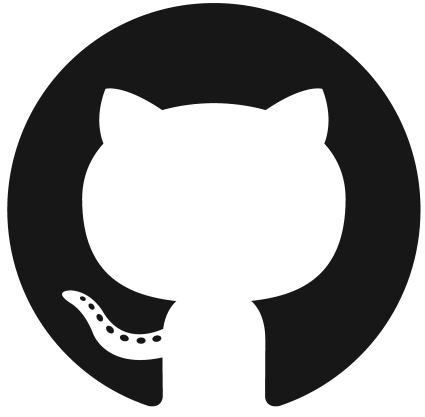
Version Control with Git

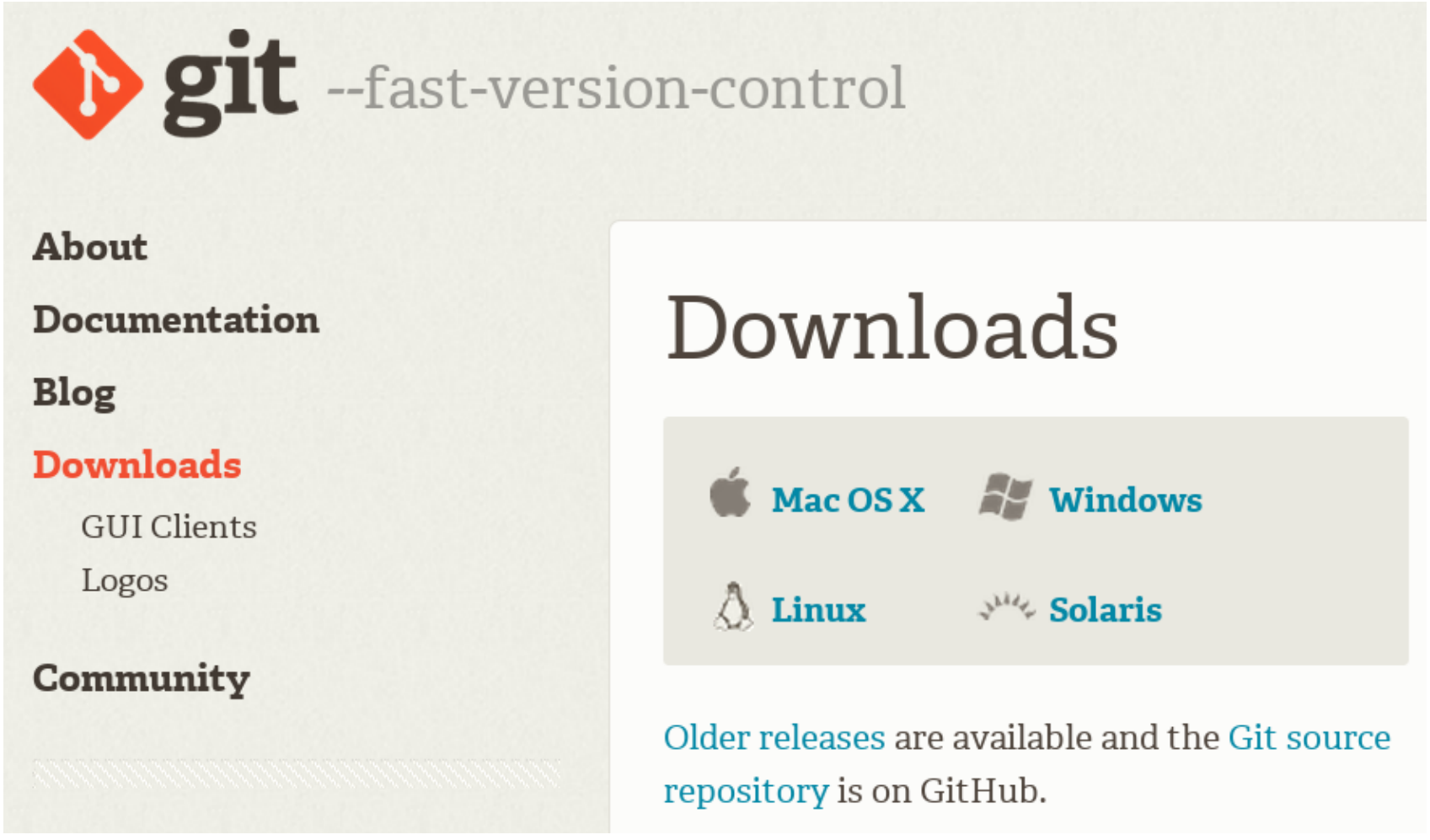


Ben R. Fitzpatrick
PhD Candidate, Statistical Science, Mathematical Sciences School, Queensland University of Technology

0000-0003-1916-0939 github.com/brfitzpatrick/ @benrfitzpatrick

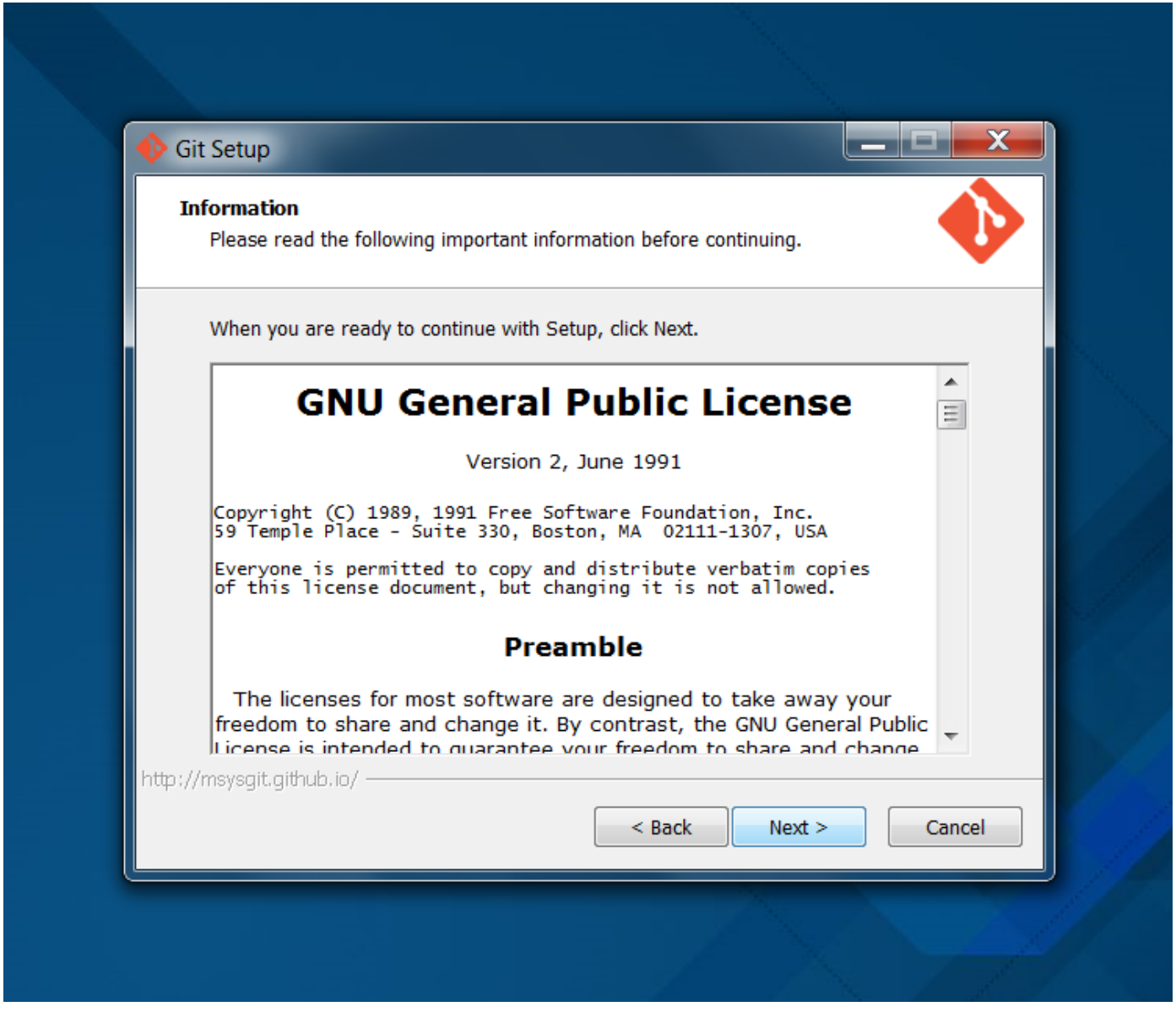
Download Git for your Operating System

`http://git-scm.com/downloads`

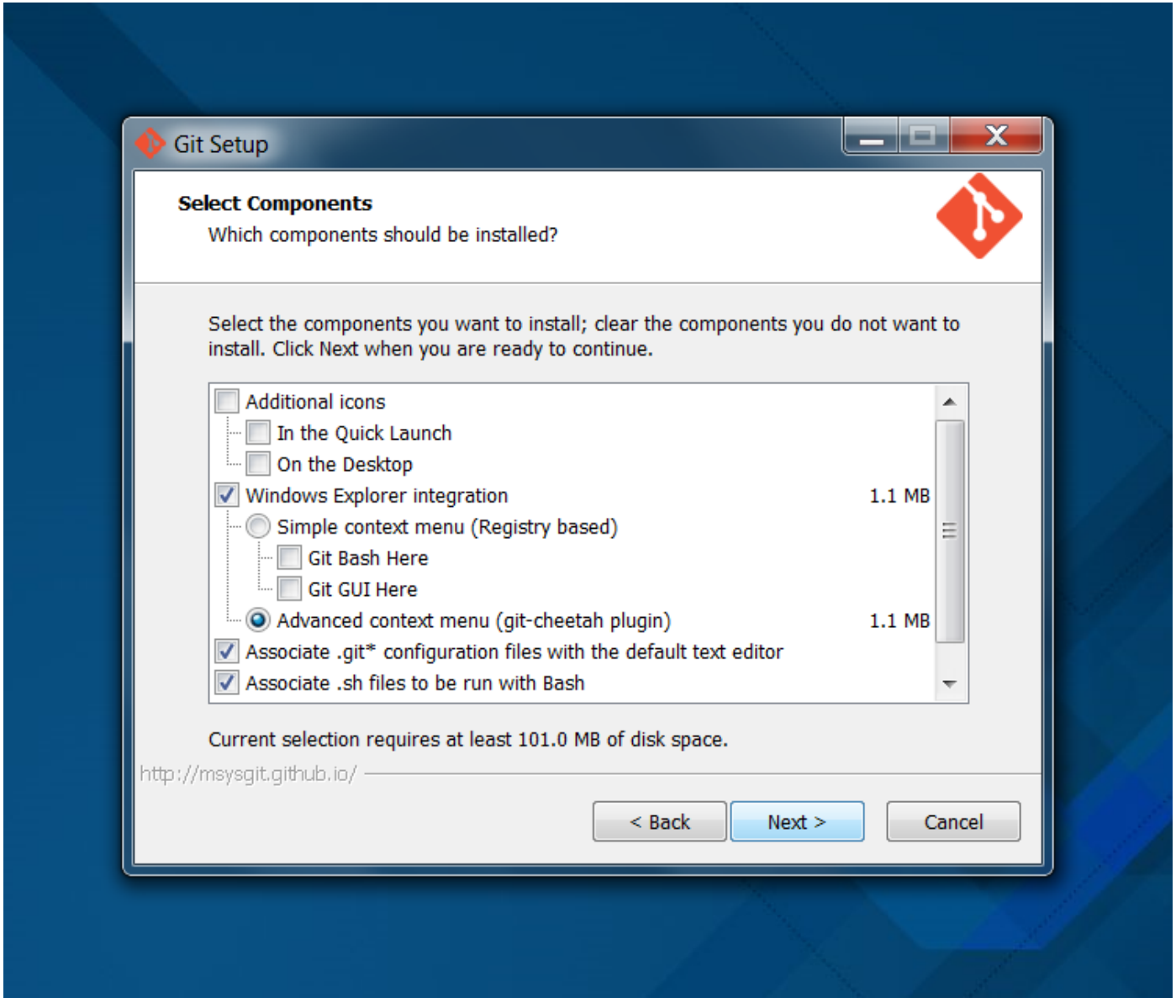


Installing Git on MS Windows: Step 1

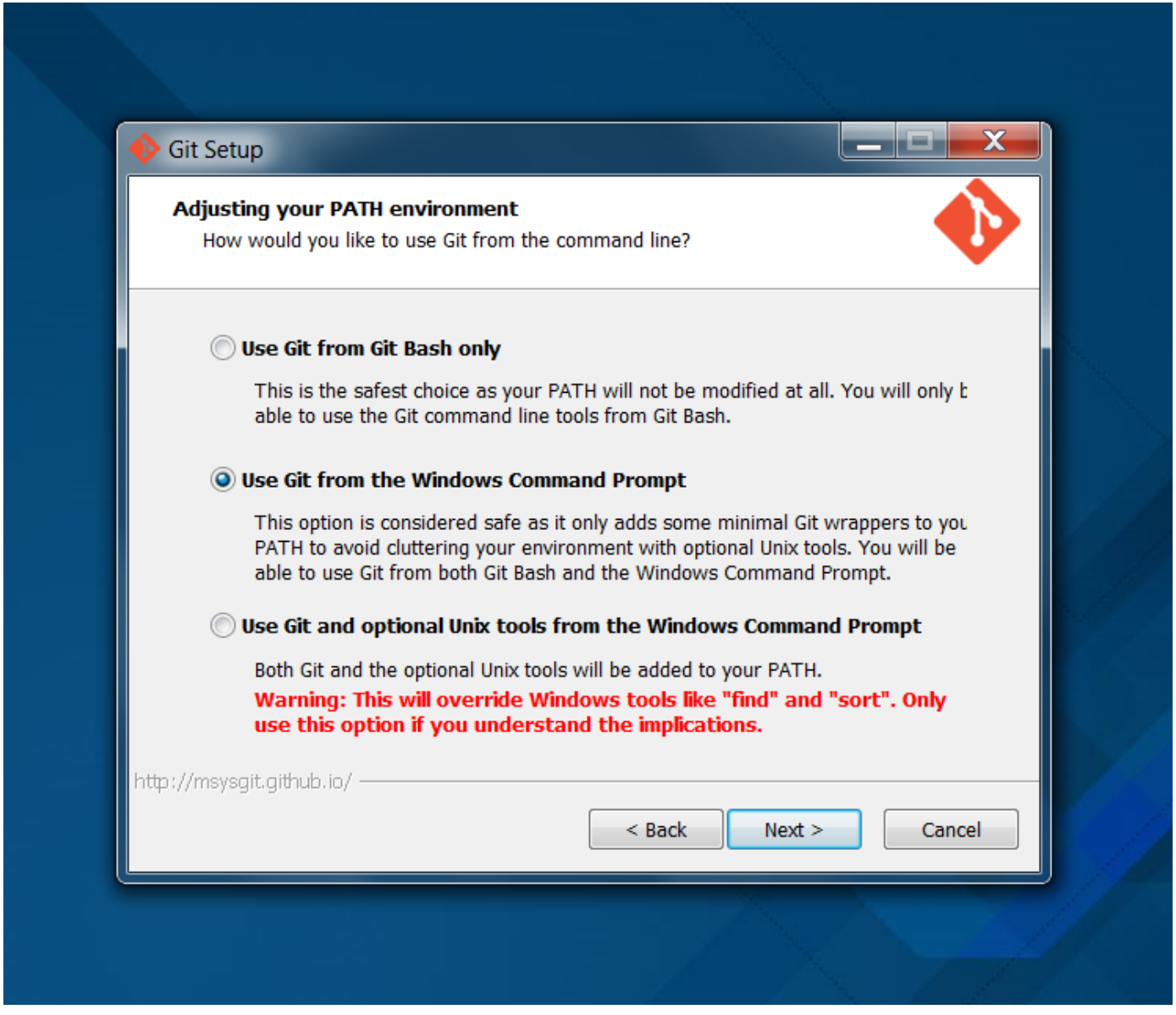
Installing Git on MS Windows: Step 2



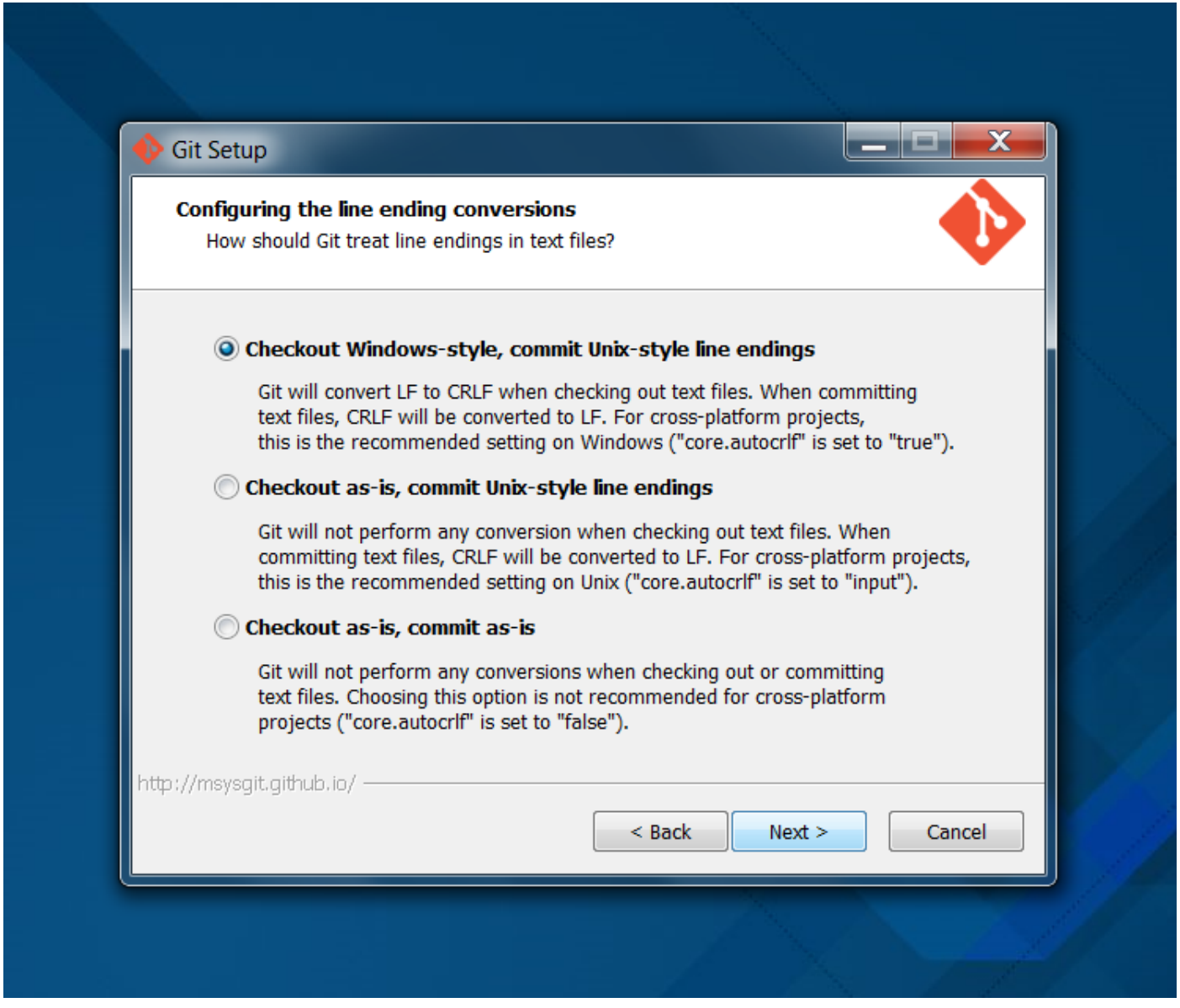
Installing Git on MS Windows: Step 3



Installing Git on MS Windows: Step 4

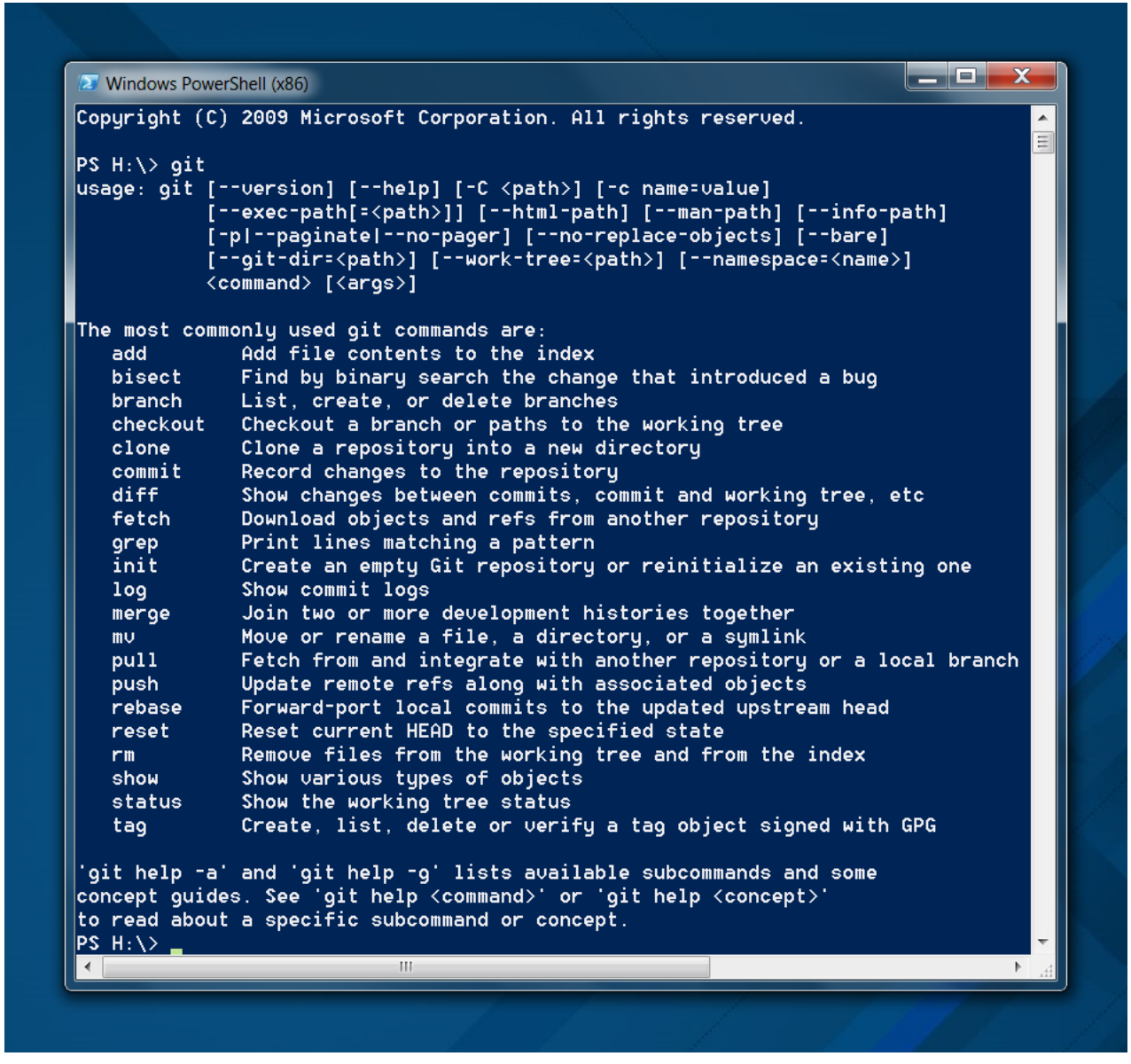


Installing Git on MS Windows: Step 5

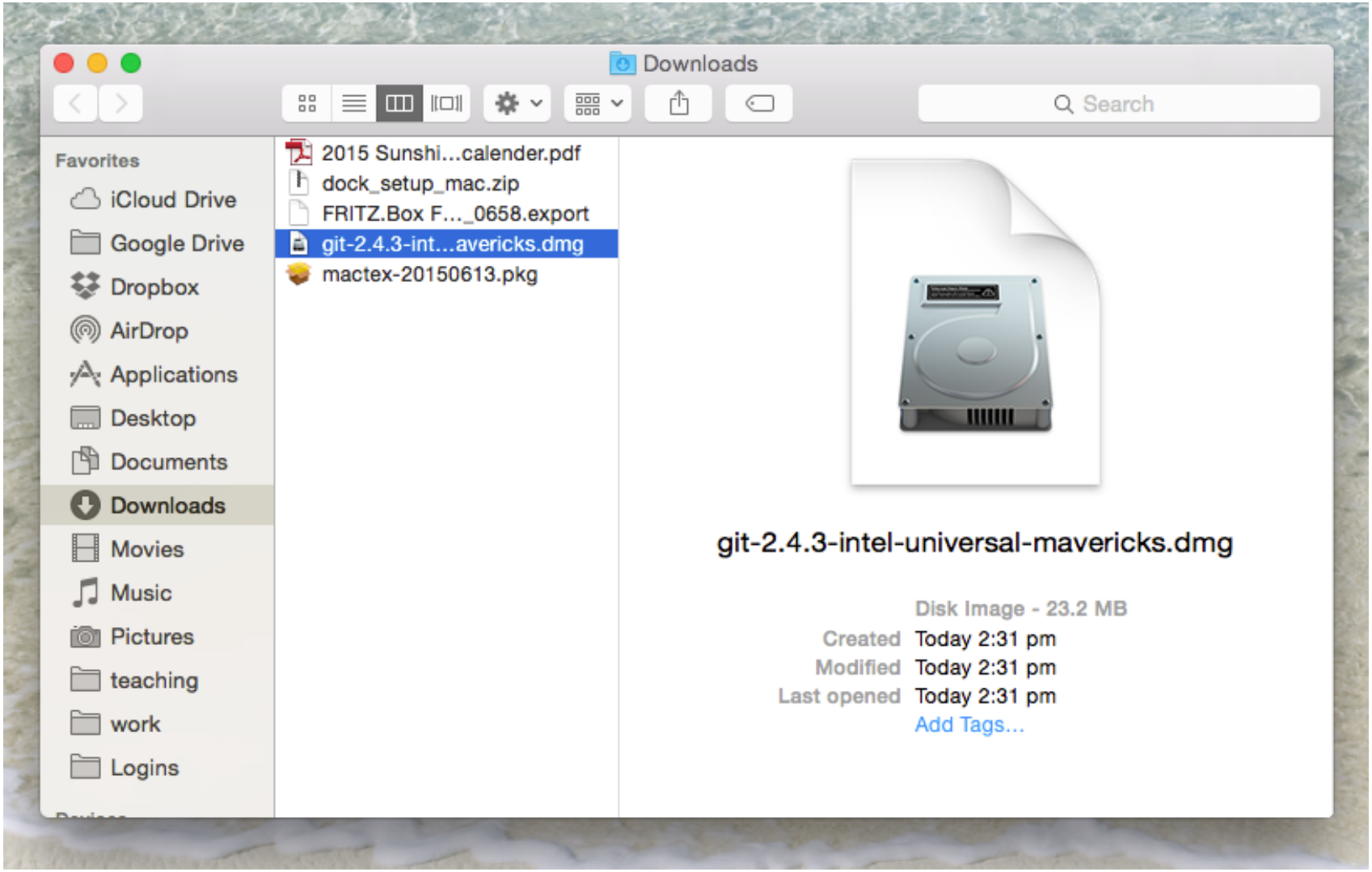


Check Git is installed on MS Windows

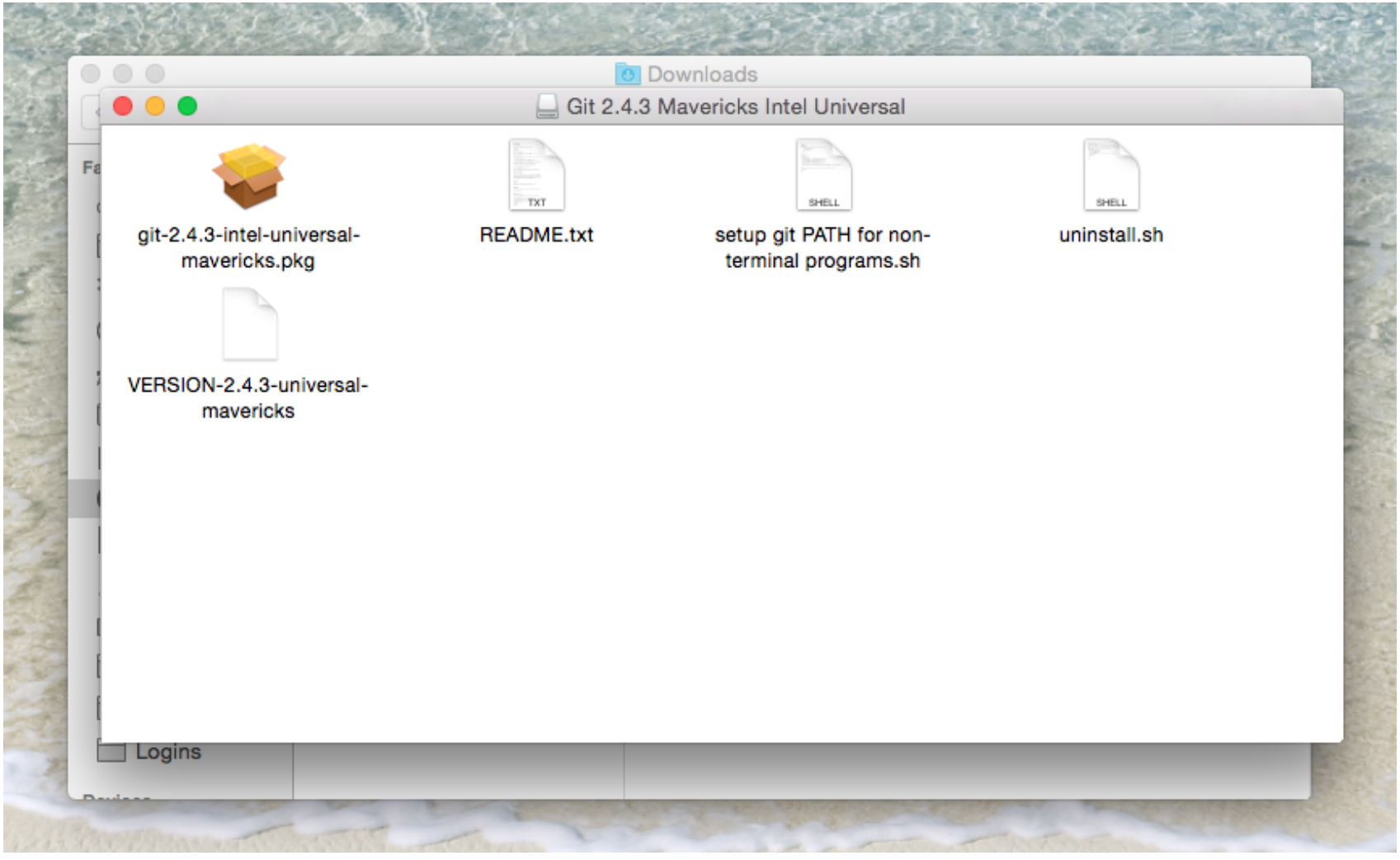
Open a PowerShell and type *git*



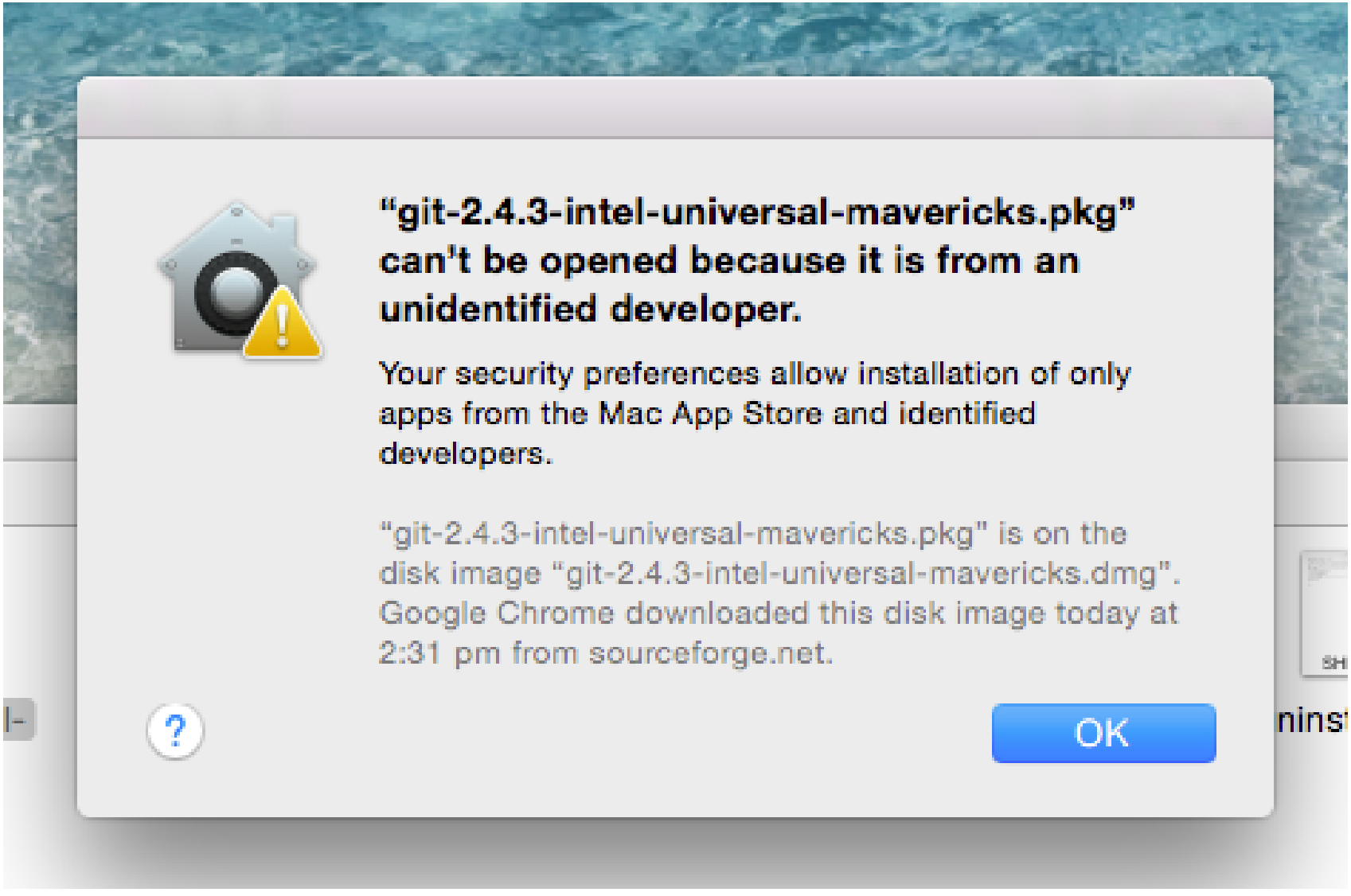
Installing Git on MacsOS: Step 1



Installing Git on MacsOS: Step 2



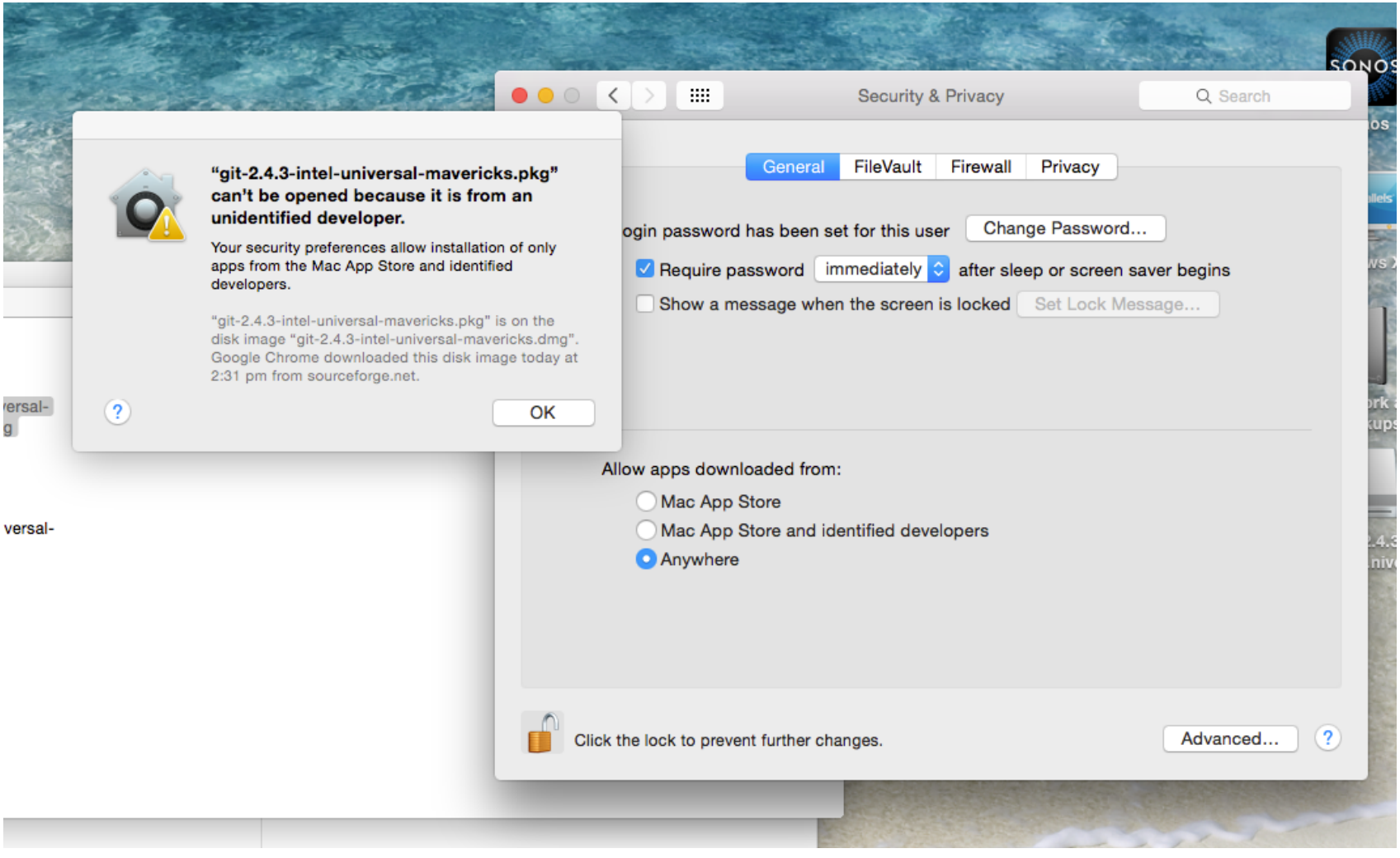
Installing Git on MacsOS: Step 3



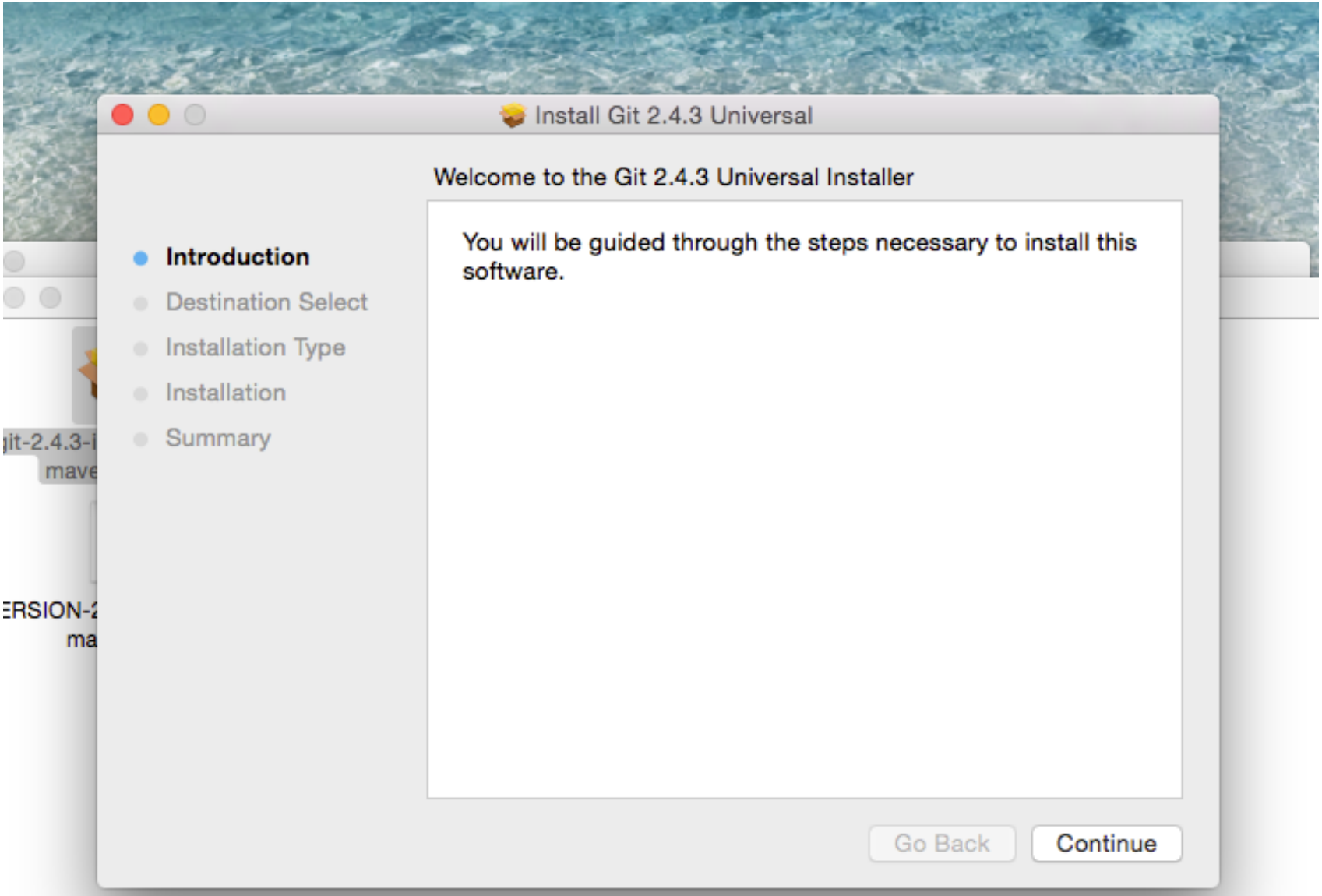
Installing Git on MacsOS: Step 4

You may need to relax your security setting briefly

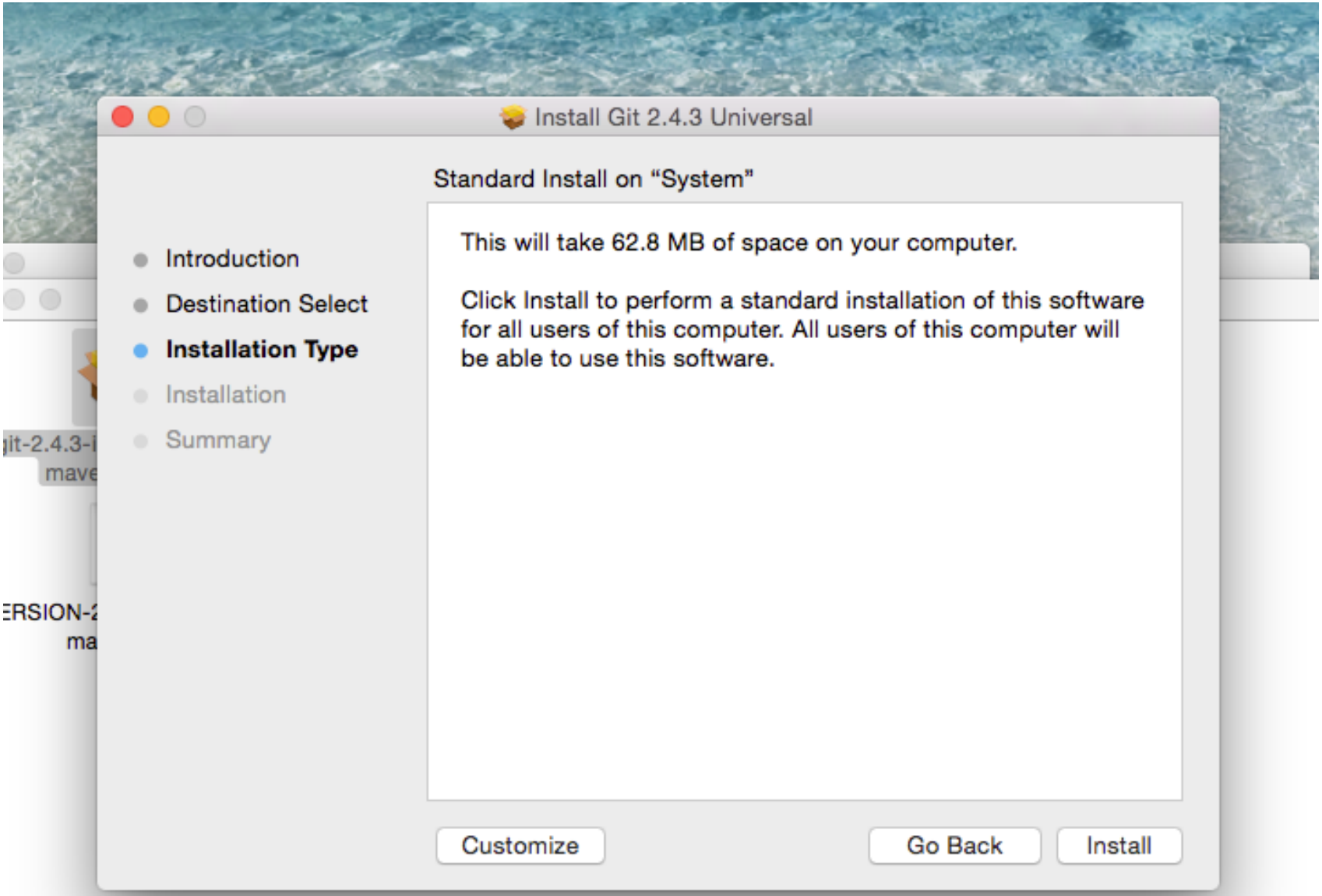
If this makes you uncomfortable you're welcome to watch the session rather than participating



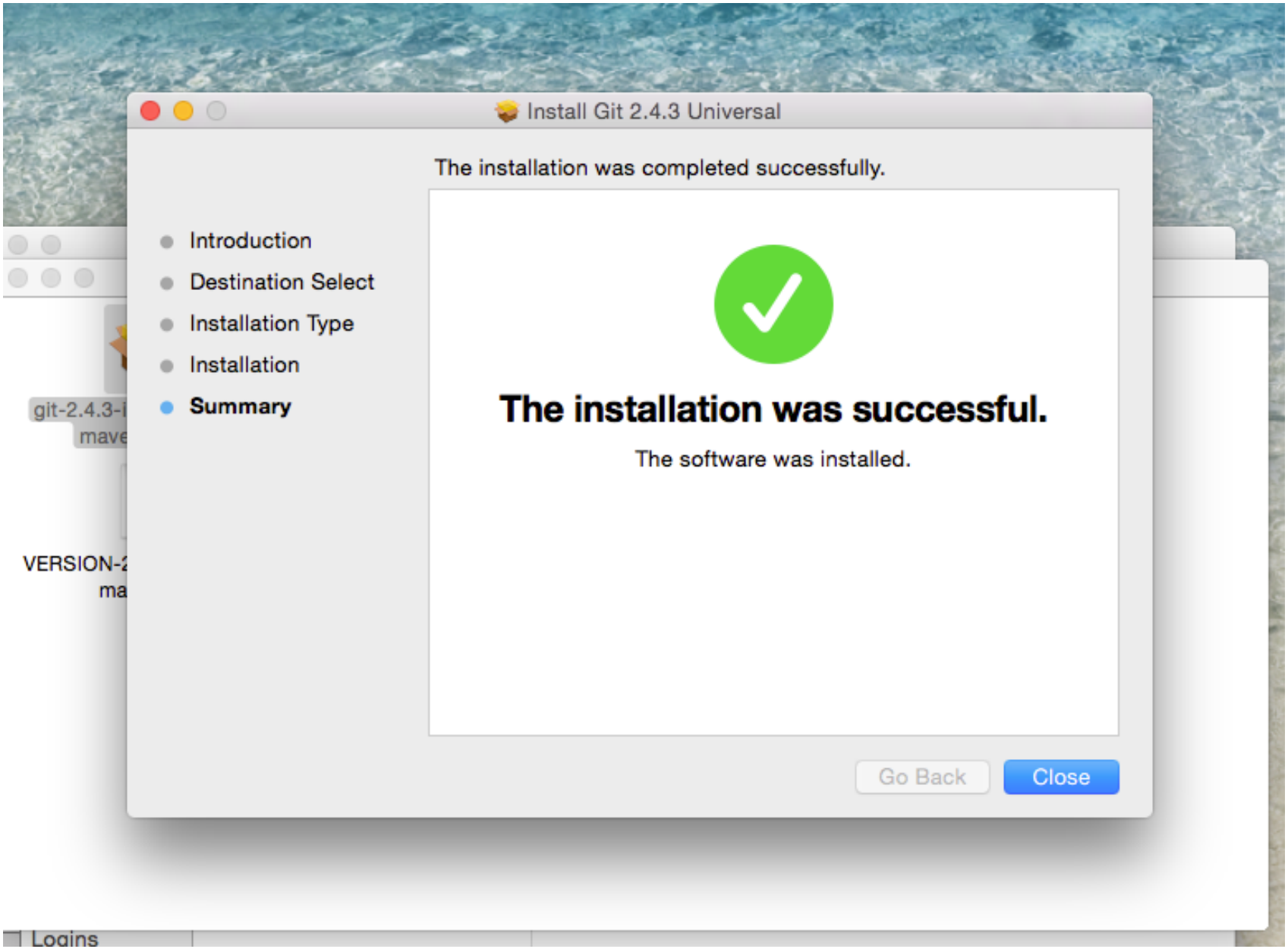
Installing Git on MacsOS: Step 5



Installing Git on MacsOS: Step 6

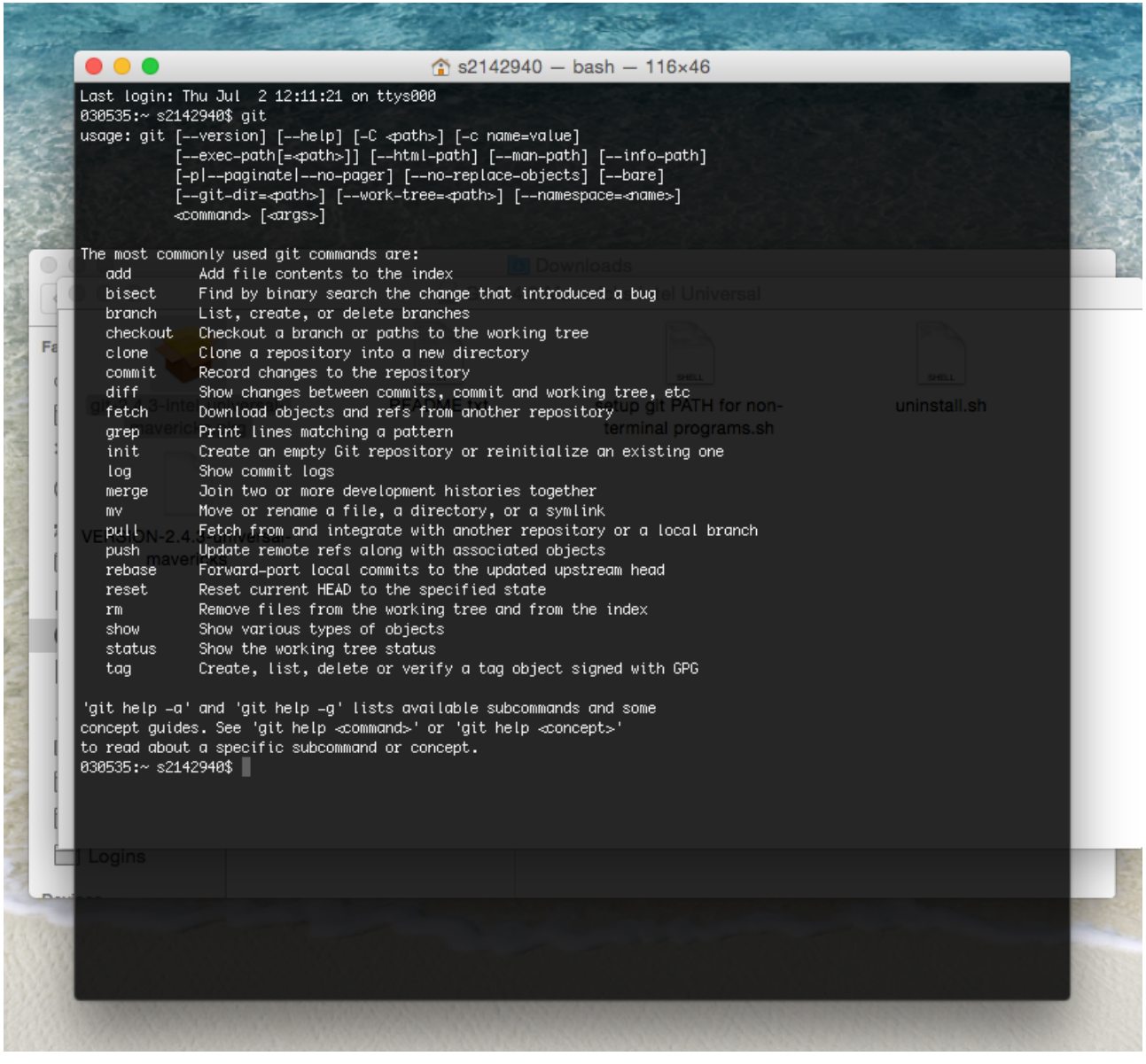


Installing Git on MacsOS: Step 7



Installing Git on MacsOS: Step 8

Check Git is installed:



Version Control Software

Have you ever put numbers or dates at the end of a file name to keep different versions of it?

- If yes then you have already used version control!

Have you ever used track changes and comments to take turns editing a MS Word document with one or more others?

- If yes then you have collaboratively edited a file!

Version Control Software:

- formalizes these concepts
- can do a lot of the organisational work for you
- makes reverting to previous versions without losing work easy

Version Control when Working Alone

Version Control for Solo Authoring File

- You essentially have a time stamped undo button which will allow you to go review or revert to previous versions of the file
- you can even make multiple child versions fo the same file say for different audiences or to trial different orders and styles in which to write a report or program

Version Control when Collaborating

Greatly simplifies your life Control over which changes different contributers make get added to the definitive version of a file support for file to split into two child copies which can then be edited towards very different end points but still having the option to systematically recombine them at any stage

Why Git?

Git is free and open source

Git is available for most operating systems

Git is Distributed and Fast

Two major code hosting services (GitHub & BitBucket) support version control with Git and both services have free account options

Git is Distributed Version Control

Every contributor has a copy of the entire database (all revisions of all files) two contributors can both work on the files in their copy of the database independently then their combined modifications can be merged systematically

Why Git on the Command Line?

Makes discrete components of the workflow obvious this in turn will enable you to migrate easily to one of the many GUIs

The workflows we will practise today will work exactly the same way on MS Windows, MacOS and GNU+Linux

We have now been using a command line interface for 1.5 days to interact with the R program it's time to take it to the next level

Plan for this Module - Part 1

Solo Version Control

- Set up Git on your laptops to communicate with the GitHub servers
- Create a local 'clone' of a remote repository
- Make some changes to your local copy of the files
- commit these changes to the git version control system
- push these changes to your repository
- make some more changes
- revert to a previous version of the file

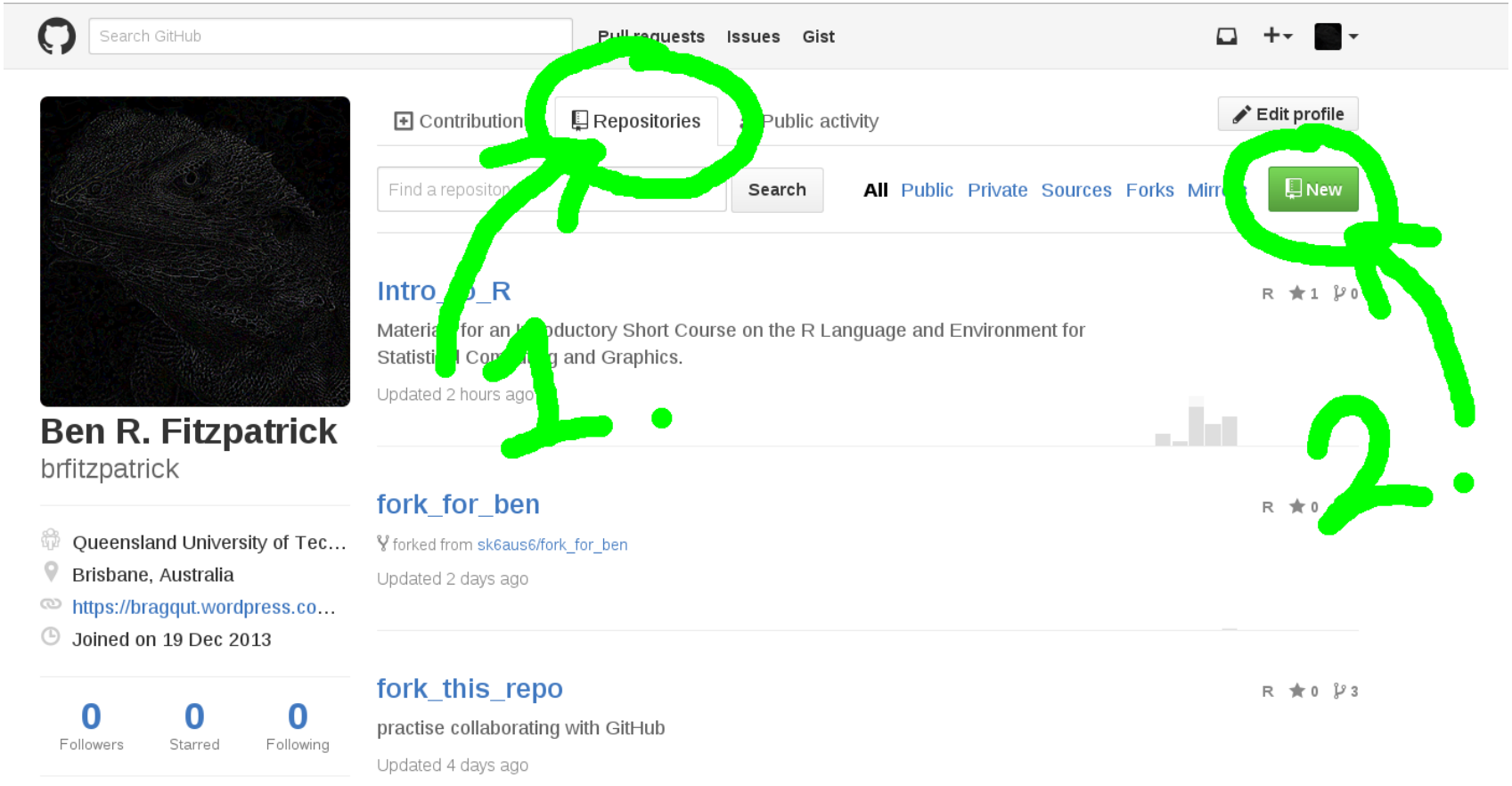
Plan for this Module - Part 2

Collaborating with Git & GitHub

- fork
- branch
- edit
- commit
- pull request
- merge

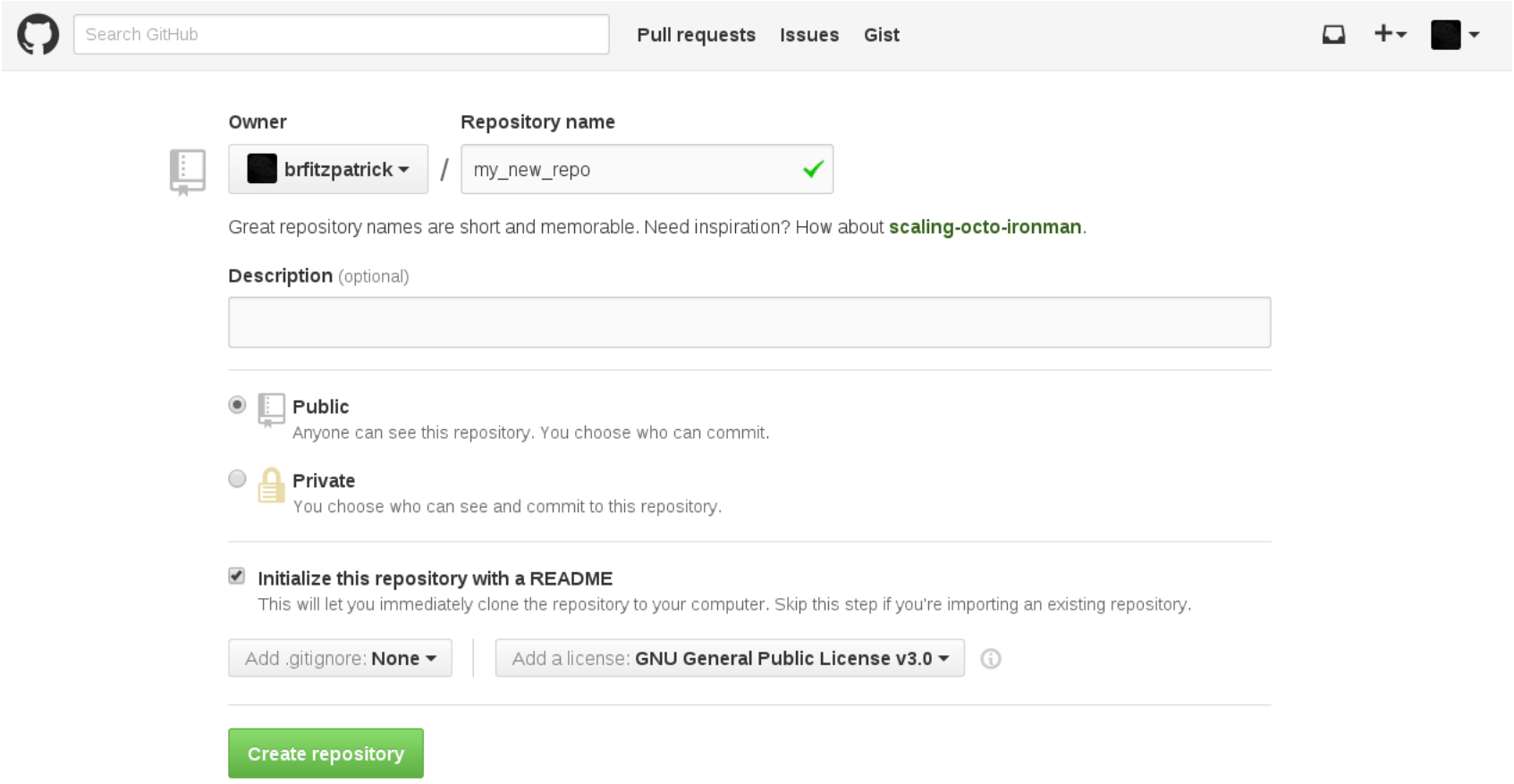
Creating a Repository on the GitHub Servers

Log into the GitHub Web Service



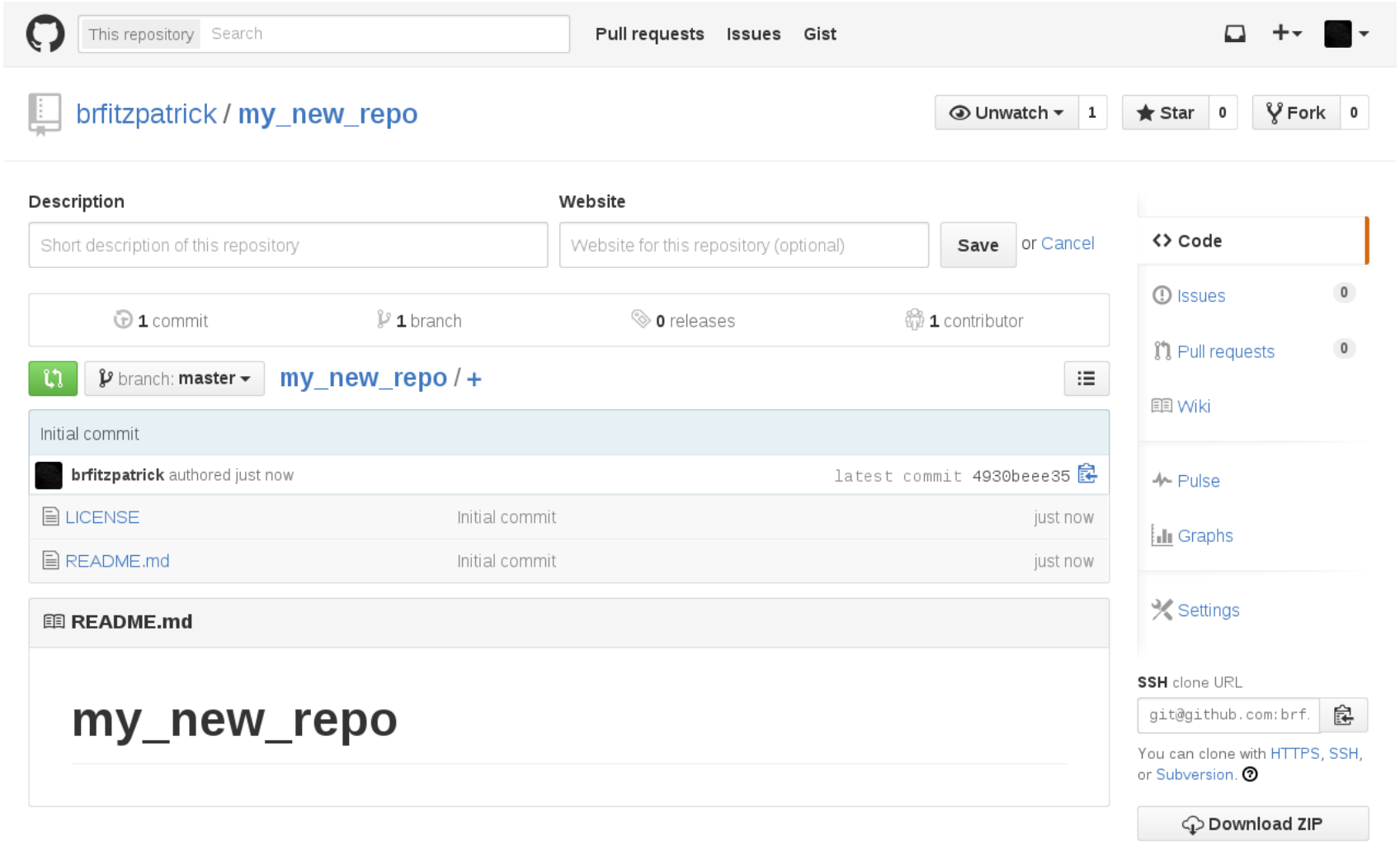
Creating a Repository on the GitHub Servers

Complete the Details



Creating a Repository on the GitHub Servers

View your New Repository



Cloning your new Repository to your Hard Drive

First, create a folder on your Hard Drive in which to store your Git Repositories.

We are now going to use the Git command line application to 'clone' your new repository from the GitHub server to your hard drive.

Git is distributed version control so a Git 'clone' is a complete copy of the entirity of the repository i.e.

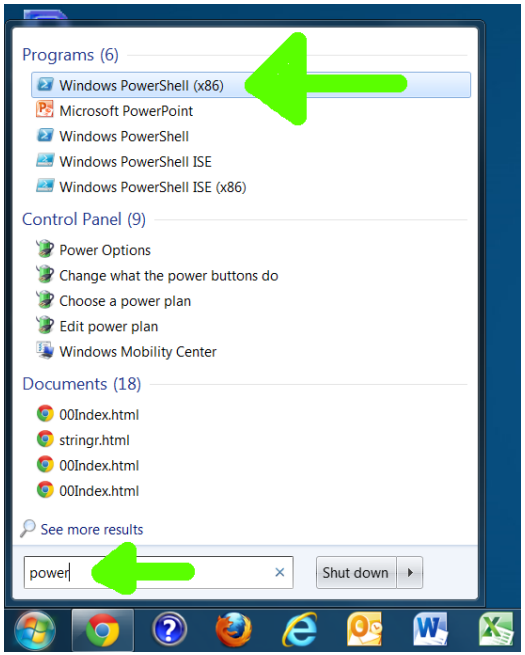
- all the files
- the entire history of snapshots of files states created each time you committed to the respository

Being a distributed version control system Git doesn't require you to be connected to the GitHub servers to work on your files and commit them to your branch of the repository.

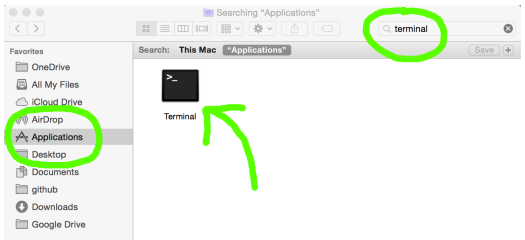
Git on the Command Line

Accessing a command line interface in your OS of choice

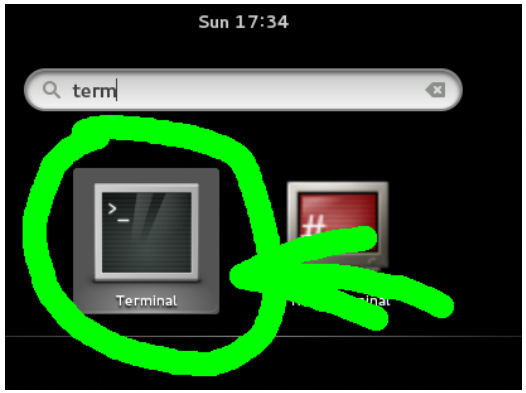
MS Windows
Open a ‘PowerShell’



Mac OS X & later
Open a ‘Terminal’

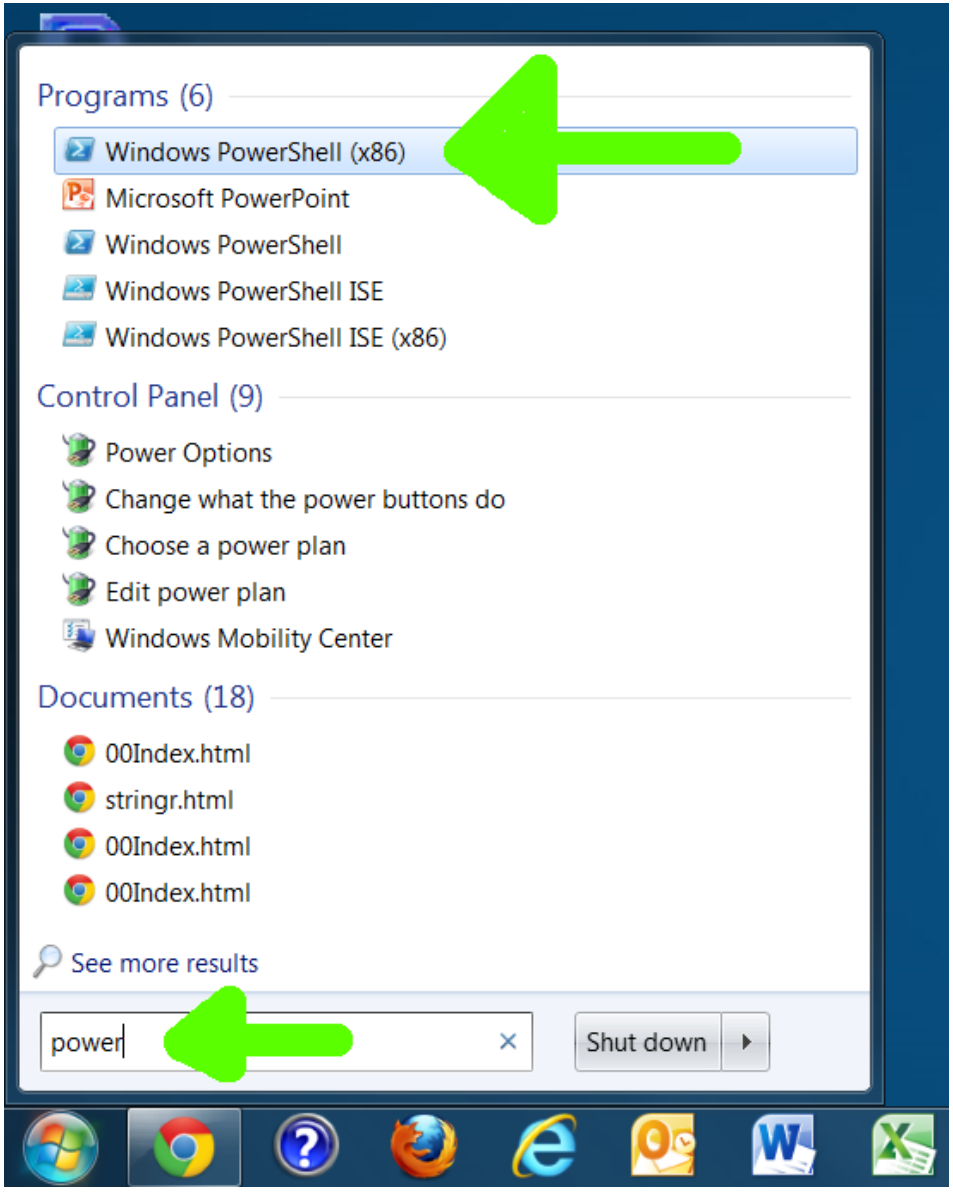


GNU+Linux
Open a ‘Terminal’



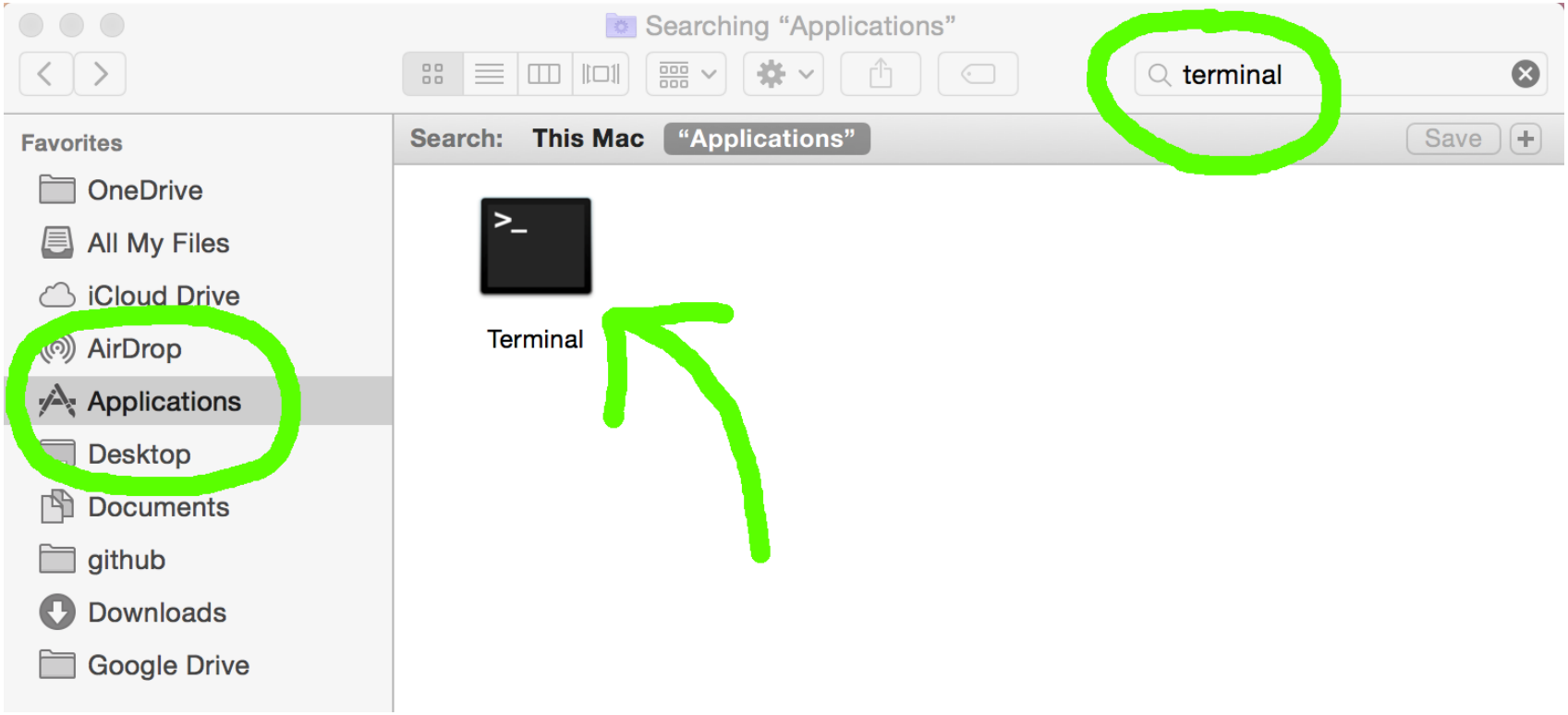
Accessing a command line interface from MS Windows

Open a 'PowerShell'Powershell



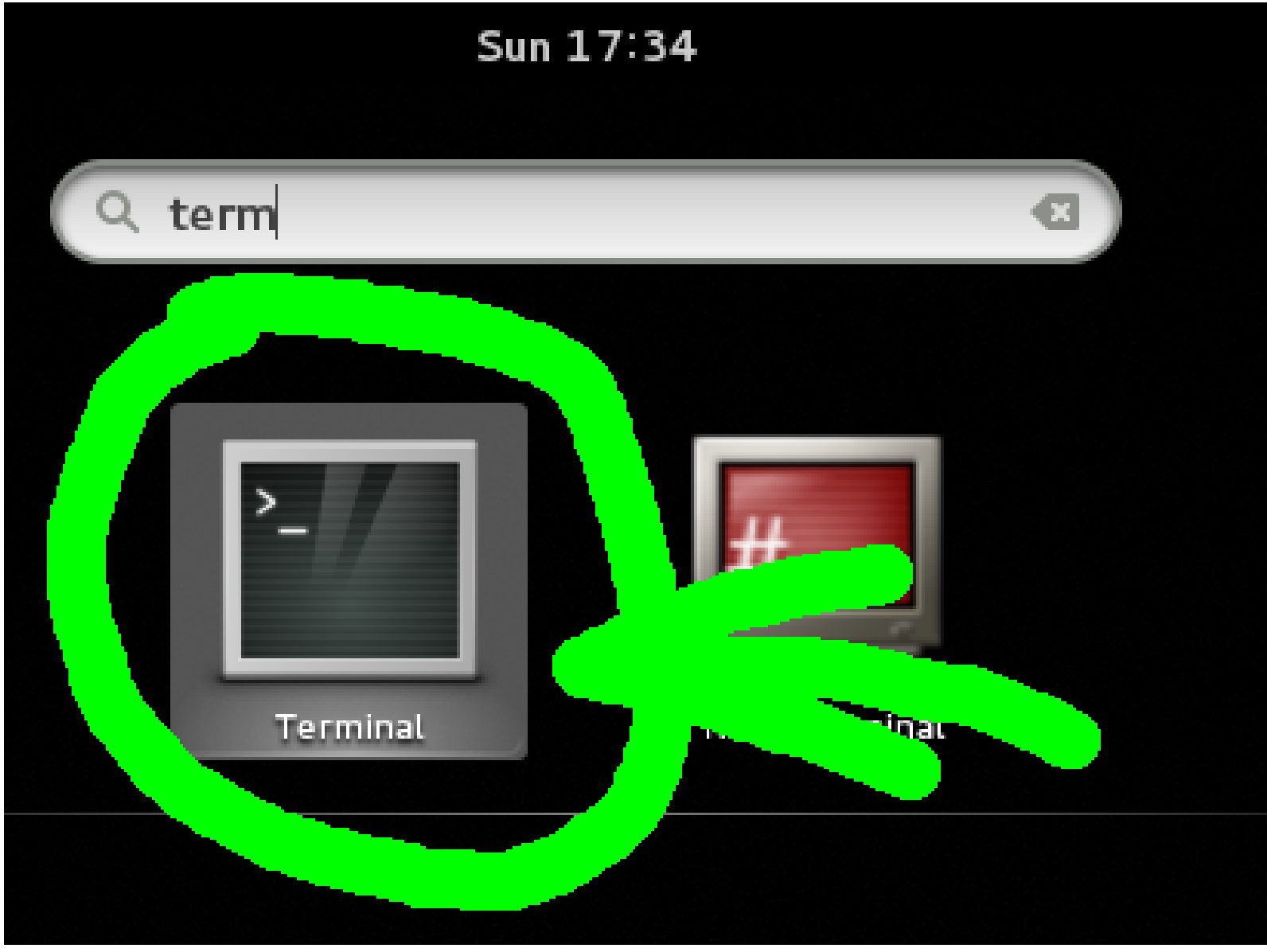
Accessing a command line interface from MacOS

Open a 'Terminal'



Accessing a command line interface from GNU+Linux

Open a ‘Terminal’



Configuring Git for the first time

All Users

```
> git config --global user.name "Your Name"
> git config --global user.email you@mail.com
```

The global editor is used to write your commit messages.

Default is Vim but Vim is a little heavy on keyboard short cuts for some...

Other Options:

- Notepad (available on all Windows PCs)
- Atom: `https://atom.io/docs/v1.0.0/getting-started-installing-atom`
- Sublime: `http://www.sublimetext.com/2`
- TextMate: `http://macromates.com/download`
- Emacs:

```
sudo apt-get install emacs
```

Configuring Git for the first time

https://help.github.com/articles/associating-text-editors-with-git/

Set a Text Editor of your choice (you'll need it installed

Use one of the below

```
# Notepad:
> git config --global core.editor notepad.exe
# Atom:
> git config --global core.editor "atom --wait"
# Sublime:
> git config --global core.editor "subl -n -w"
# TextMate:
> git config --global core.editor "mate -w"
# Emacs:
> git config --global core.editor "emacs"
```

Cloning Your Repository from the GitHub Servers

Set the working directory to the location on your hard drive to which you would like the repository cloned:

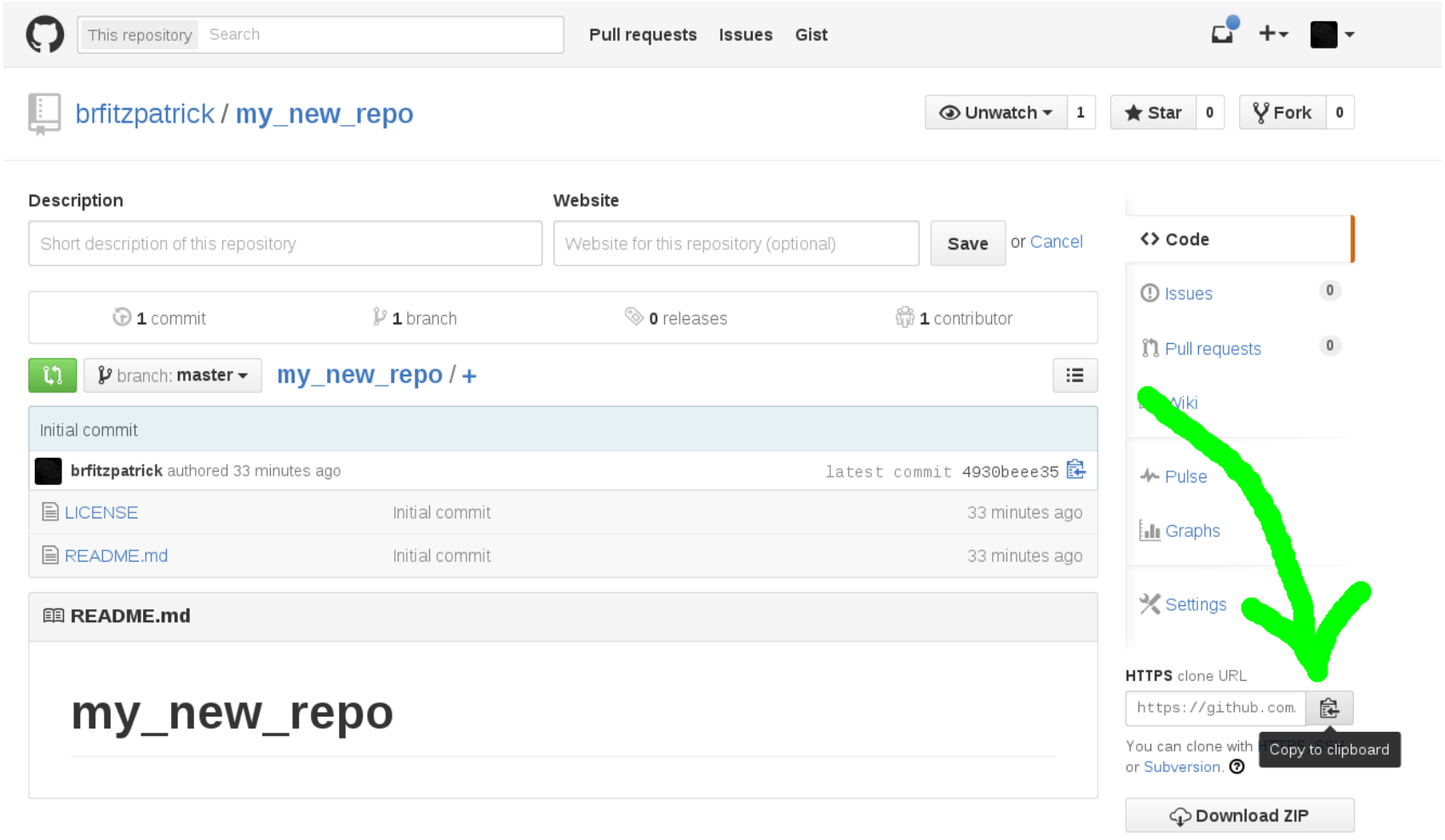
Powershell Users

```
> C:
> cd C:\Users\username\Documents\GitHub_Repos\
> dir
```

Terminal Users

```
> cd /home/ben/Documents/GitHub_Repos
> ls
```

Copy HTTPS URL to use when cloning repository



Clone your Repository: GitHub Servers → your Hard Drive

Use the HTTPS URL you copied earlier:

All Users

```
> git clone https://github.com/.../my_new_repo
  .git
> cd ./my_new_repo/
> ls
> git status
```

Files in a directory which you are version controlling with Git can exist in one of three states:

- **committed** data stored in the database for the associted repository
- **modified** local copy of a file (in your working directory) is different to the most recent version of that file stored in the database for the associted repository i.e. you have changed something but not comitted the changes (yet)
- **staged** the local copy of a file which you have modified is marked to be added to the database in the next commit snapshot you send to the database

- you add files or modify existing files in your local copy of the repository
- you stage these modified files ready to be committed to the database for the repository
- you perform a commit which takes the files as they were when staged and stores a snapshot of them in the database for the repository

1. Open your favourite program for authoring R code (e.g. RStudio)
2. Create an new R script
3. Save your R Script in the ‘clone’ of your repository

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what
will be committed)
```

We have to tell Git that we want it to track the changes we make to this new file

We have to tell Git that we want it to track the changes we make to this new file

```
> git add filename.R
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

new file:   filename.R
```

We can still change our mind about adding the file to the repository at this stage quite easily.

Once you're ready to 'commit' this change to the permanent record of changes in this repository we make a 'commit'

```
> git commit
[master 311acfb] Initial Commit of some R code.
1 files changed, 0 insertions(+), 0 deletions
(-) create mode 100644 filename.R
```

You will be prompted to write a short description of the changes you are committing - these become invaluable as your project grows and if you are collaborating. Once you've written a commit message, save it (and close the text editor if you used one).
You should see confirmation of the commit immediately below the git commit line.

Git

Now that you've commited some changes to your local clone of the repository this local version will be ahead of the version on the GitHub Server

```
> git status
On branch master
Your branch is ahead of 'origin/master' by 1
commit.
  (use "git push" to publish your local commits)
```

Seeing as we are all currently connected to the internet, it's a good time to 'push' these changes to GitHub server ('origin/master' in the default setup) to bring the remote copy of this repository up to date.

pushing to the remote servers

```
> git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100\% (2/2), done.
Writing objects: 100\% (3/3), 353 bytes | 0 bytes/s
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:brfitzpatrick/my_new_repo.git
   4930bee..99d61f1  master -> master
```

If you return to the page for your repository on the GitHub website and refresh the page you should see your most recent commit listed on the repository page.

the whole process

Furthermore, now that we have ‘commited’ and ‘pushed’ our most recent changes our local copy of the repository and the remote copy of the repository on the GitHub servers will be identical

```
> git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Undoing Things Safely

Checking out Previous Commits to a New Branch

To revert to a previous version of a file that you have ‘snapshot’ of from a previous ‘commit’
Click on the file name on the GitHub website and click the ‘History’ button:

Provided we have commited all changes to the file then it is safe to use the checkout command which will overwrite the file in the current local working directory with the file from the previous ‘commit’.

You can think of commits a bit like save points, if we have two we can move between them but if we load and old save without first making a current save we will loose the unsaveed changes

choose a commit you'd like to go back to...copy the SHA

```
# List existing branches of current directory
git branch

# checkout a previous commit (SHA = 58...a44)

> git checkout -b testing 589172506c13572be6cc05270
  Switched to a new branch 'testing'

> git branch
    master
  * testing
```

Now if you open your .R file you'll see that you have a previous version of the file without the new changes

```
> git branch
  # make some changes on testing branch
  # push those changes upsteam to server
> git push --set-upstream origin testing
  # check website - your repo should now have two b
  # Now to merge your changes from 'testing' back i
  # to merge into a branch we must be on that branc
> git checkout master
  # merge testing1 into current branch (master)
> git merge testing1
  Auto-merging Car_Bar_Charts.R
  CONFLICT (content): Merge conflict in Car_Bar_Cha
  Automatic merge failed; fix conflicts and then co
```

Resolving a Merge Conflict with a Merge Tool

```
git mergetool

This message is displayed because 'merge.tool' is n
See 'git mergetool --tool-help' or 'git help config
'git mergetool' will now attempt to use one of the
meld opendiff kdiff3 tkdiff xxdiff tortoisemerge gv
Merging:
Car_Bar_Charts.R

Normal merge conflict for 'Car_Bar_Charts.R':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (meld):
```

Resolving a Merge Conflict with a Merge Tool

Insert Screen shot of **Meld** in action

Now open you're R file to see the results of your merge

and open 'filename.R' and our changes are gone

checkout the most recent commit to get them back

if you want to go back to a previous version of a file and try something new without overwriting your most recent versions you can 'checkout' and old 'commit' to a new 'branch' of the repository

One way to safely experiment with past versions of a file is to create a new 'branch' in the repository in which to do so

branches are separate lines of development of the same files

let's make a new 'testing3' branch checking out a previous version of the file filename.R

```
> git checkout 99d61f19d85ae7aec691feb81ca8aef59f6a0
```

```
> git push --set-upstream origin testing3
```

we can then move between branches with the checkout command

```
> git checkout master
```

```
> git checkout testing3
```

Merging Branches

To merge the ‘master’ and ‘testing3’ branches

```
> git checkout master
> git merge testing3
```

if you get a merge conflict use a mergetool to resolve the conflict (you should have one installed by default from when you installed Git)

```
> git mergetool
```

Forking

Fork a repository Use the Sync button in Windows/MacOS client to get new changes from server (and submit any changes you have made)

Create a Repository, Add Some Files

- Log into your Github account
- Click the Repositories Tab
- Click the 'New' button (it's green)
- name your repository
- make it public (we can do a private one later)
- intialize repository with a README
- add a license GPL or MIT are easy FOSS licenses
- clone the repository to your hard drive
- add a file to your local clone of the repository
- commit it to the repository
- push your changes to the remote repository

Fork a Friends Repository, Add Some Files

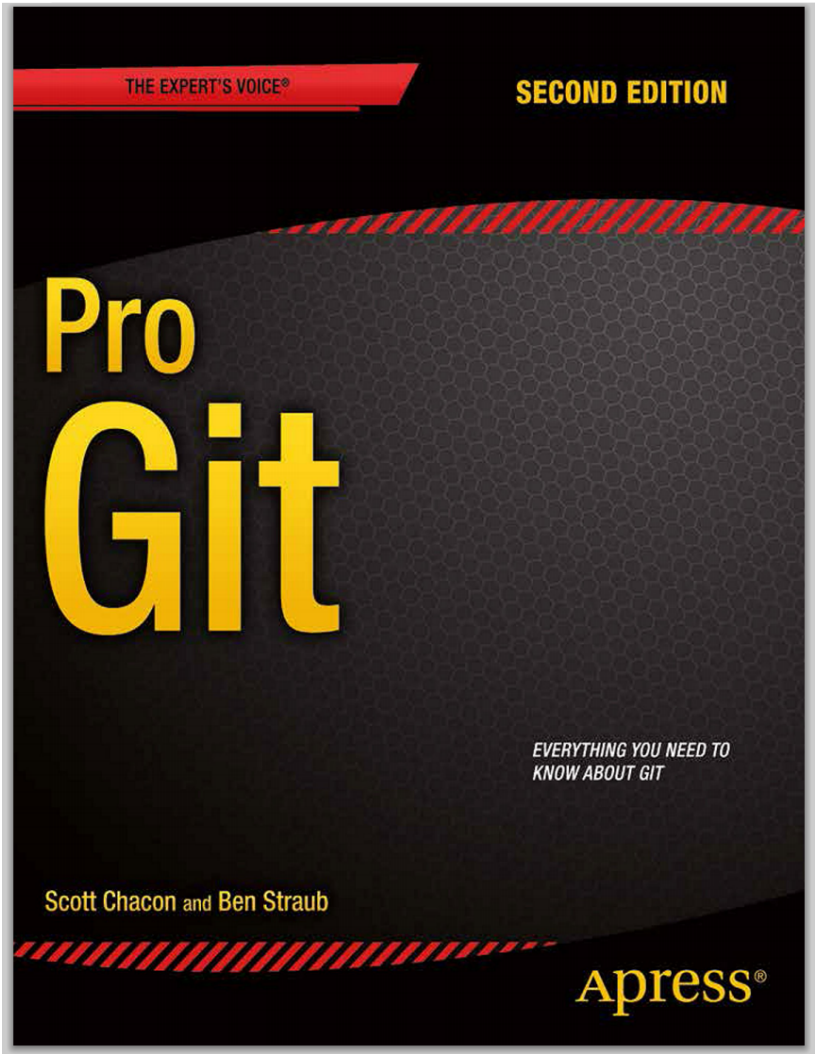
- search for a friend's github repository
- on their repository page click fork
- clone your fork of their repository to your hard drive
- edit one of their files
- push your chages to their repo

Accepting and rejecting Pull Requests

Merging your friends' branches

Recommended Further Reading

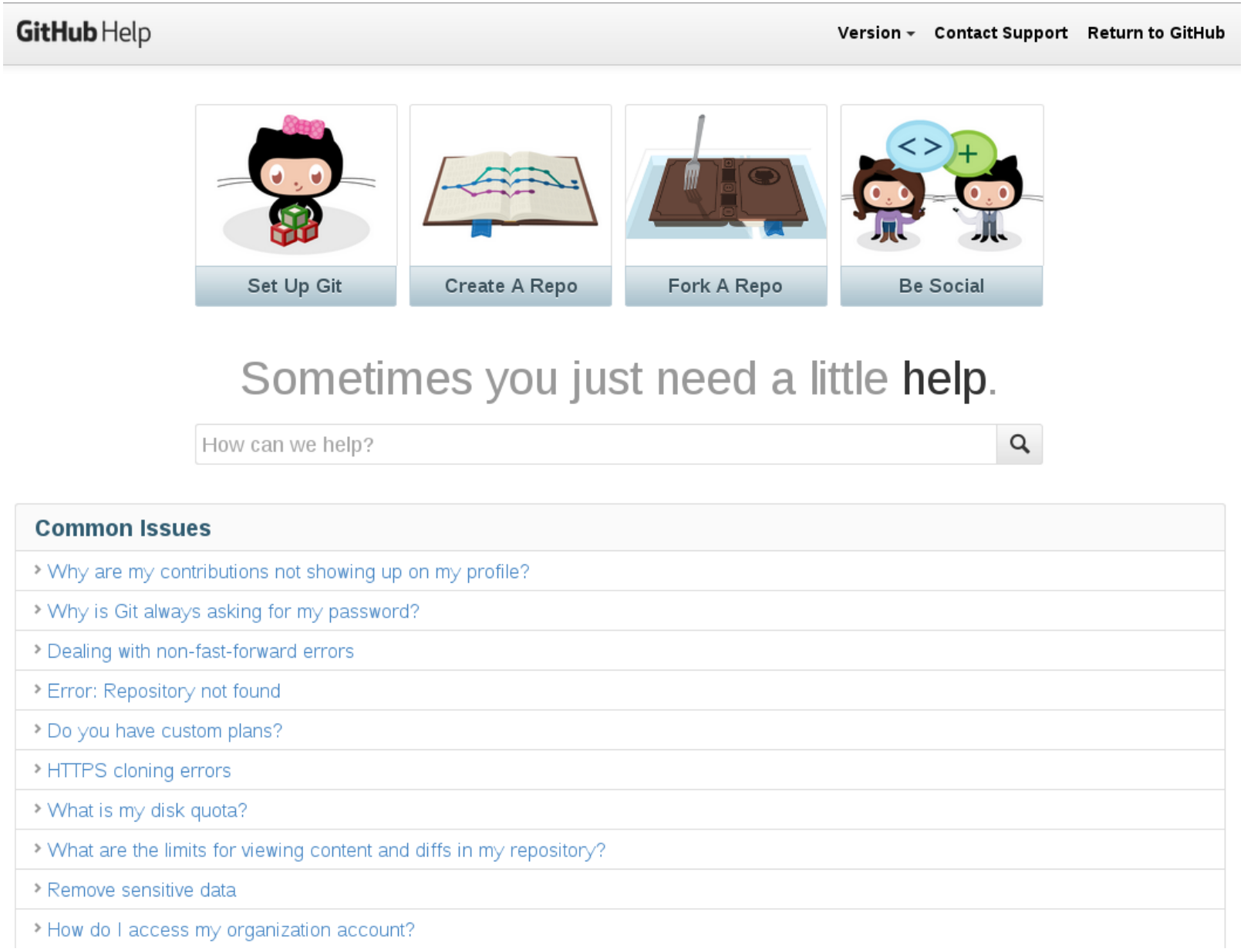
`http://git-scm.com/book/en/v2`



[?]

Recommended Further Reading:

https://help.github.com/



References

Image Credits

- Git Logo by Jason Long
  `http://git-scm.com/downloads/logos`
- GitHub Logo
- R Logo