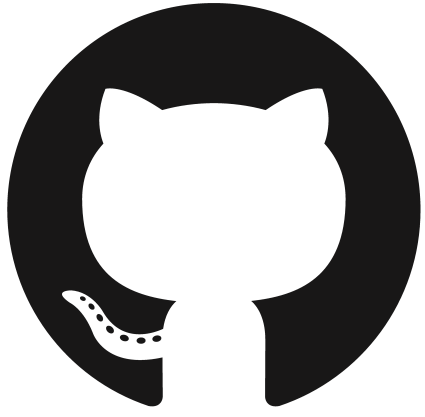
Version Control with Git

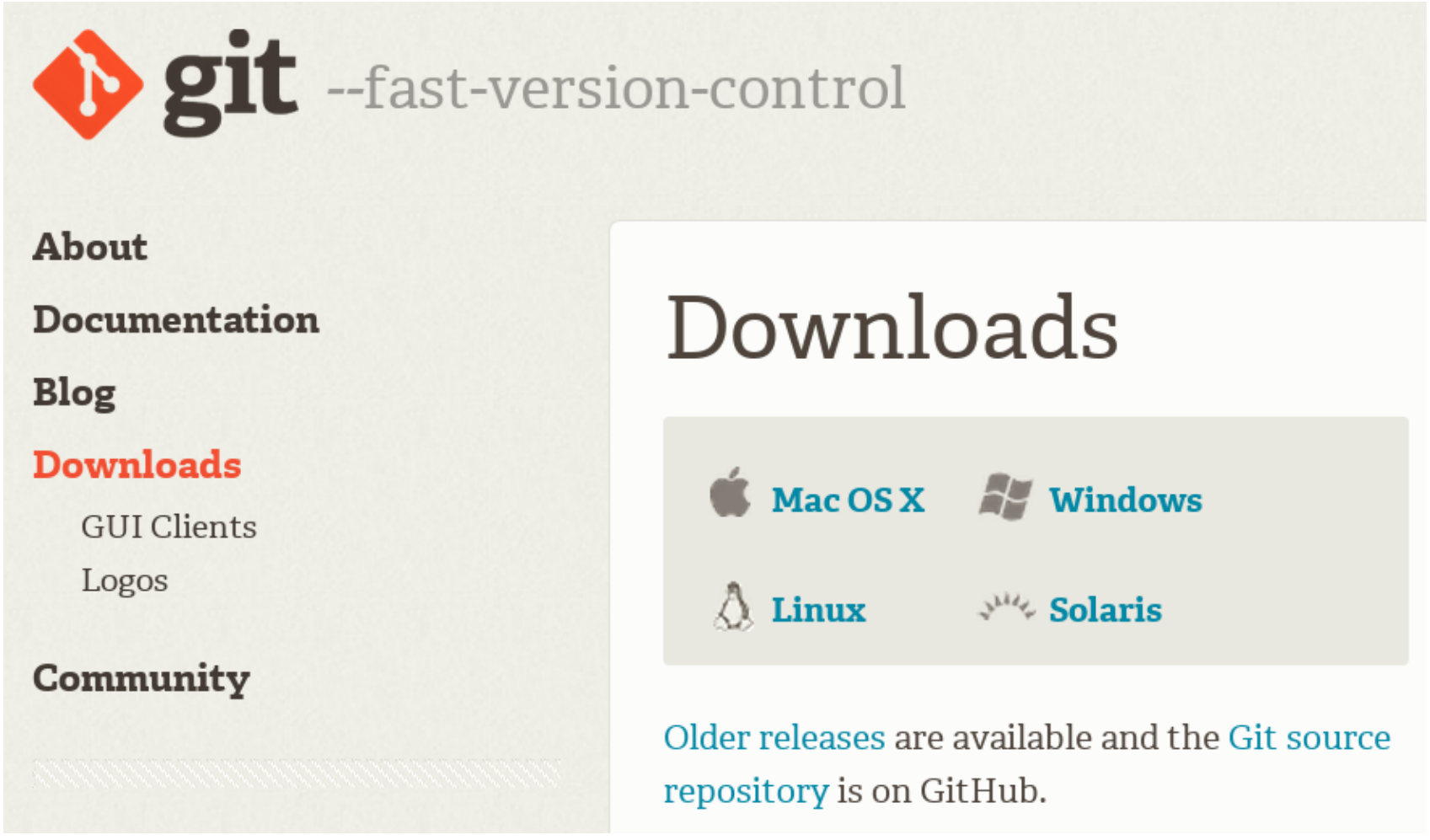


Ben R. Fitzpatrick
PhD Candidate, Statistical Science, Mathematical Sciences School, Queensland University of Technology

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

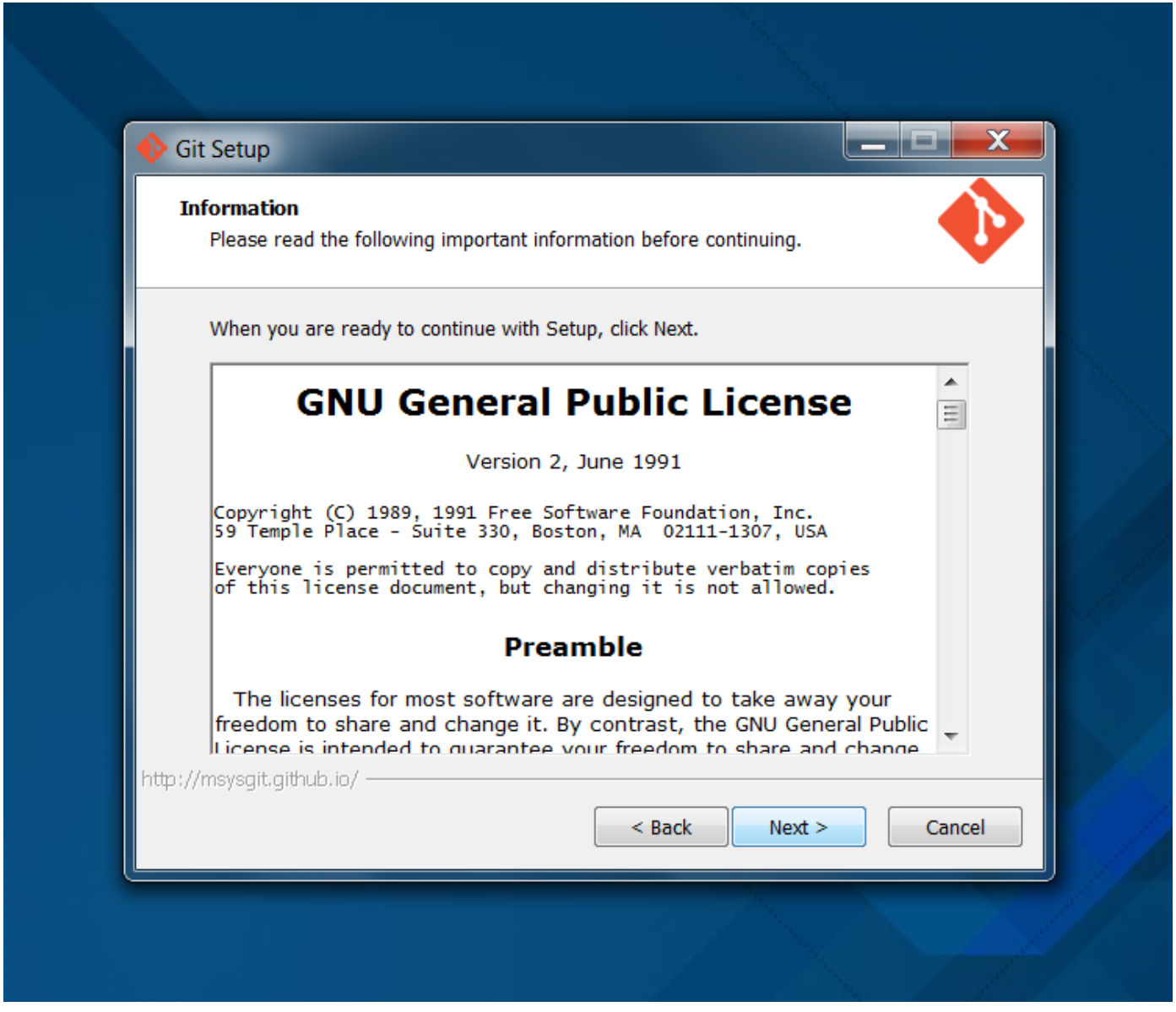
Download Git for your Operating System

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

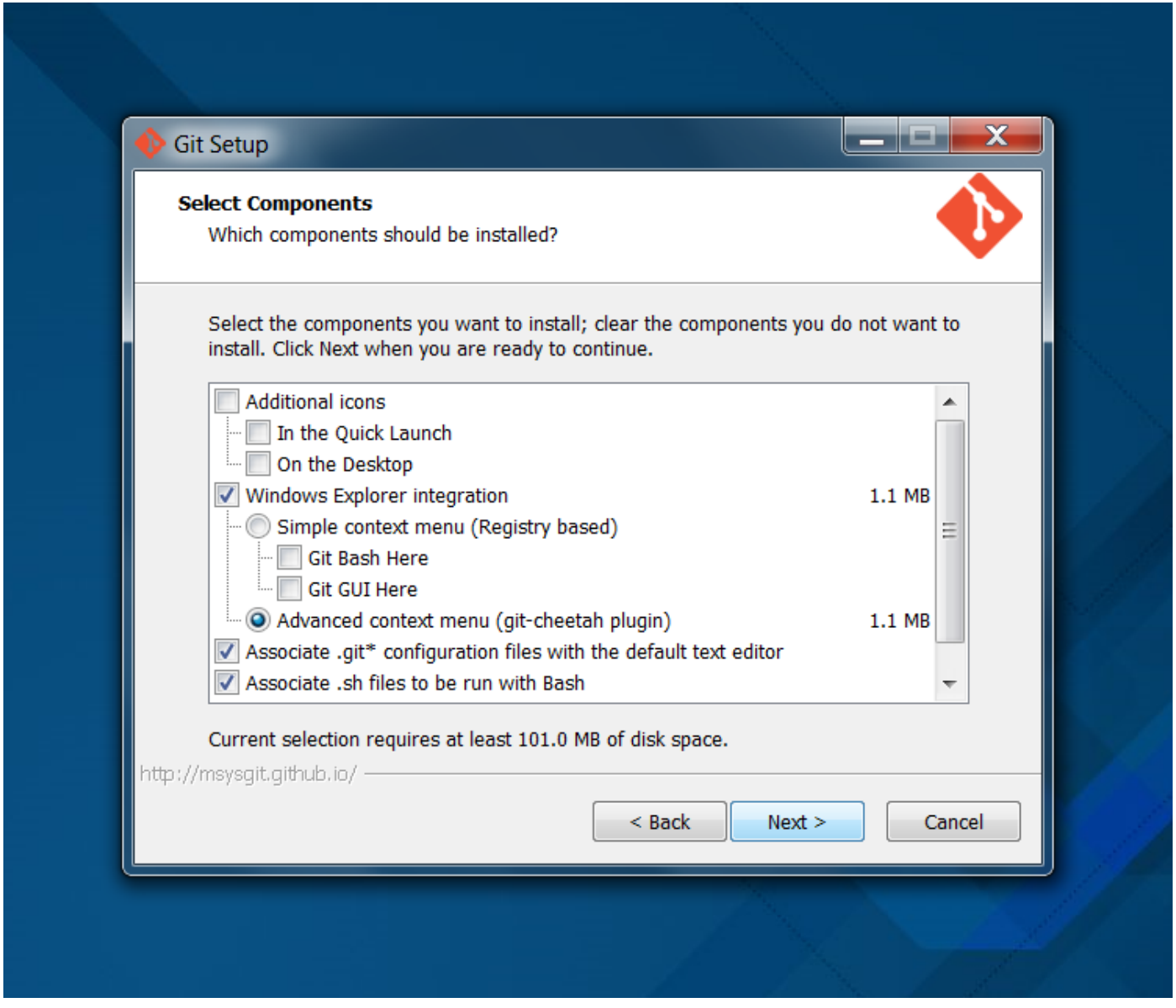


Installing Git on MS Windows: Step 1

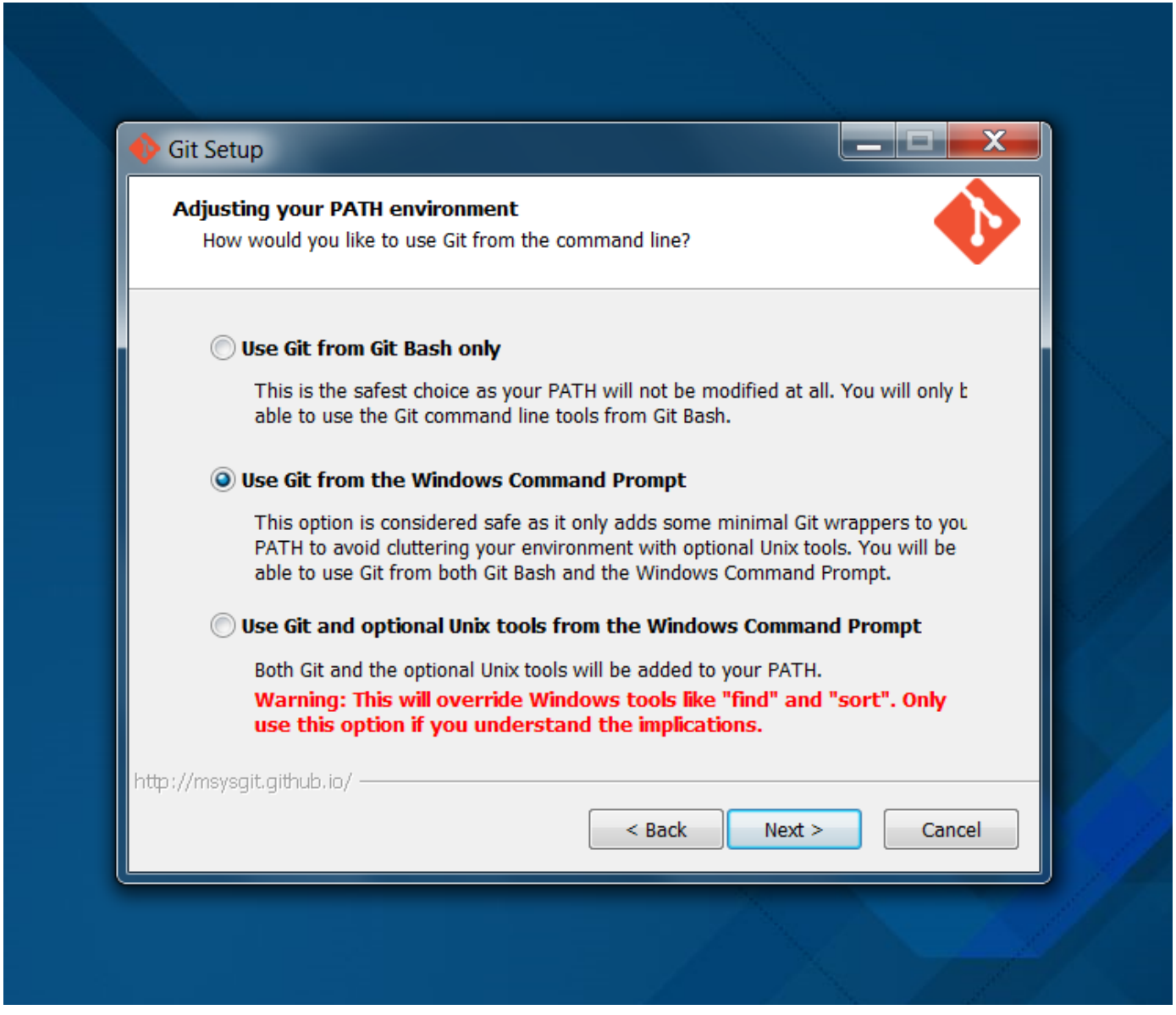
Installing Git on MS Windows: Step 2



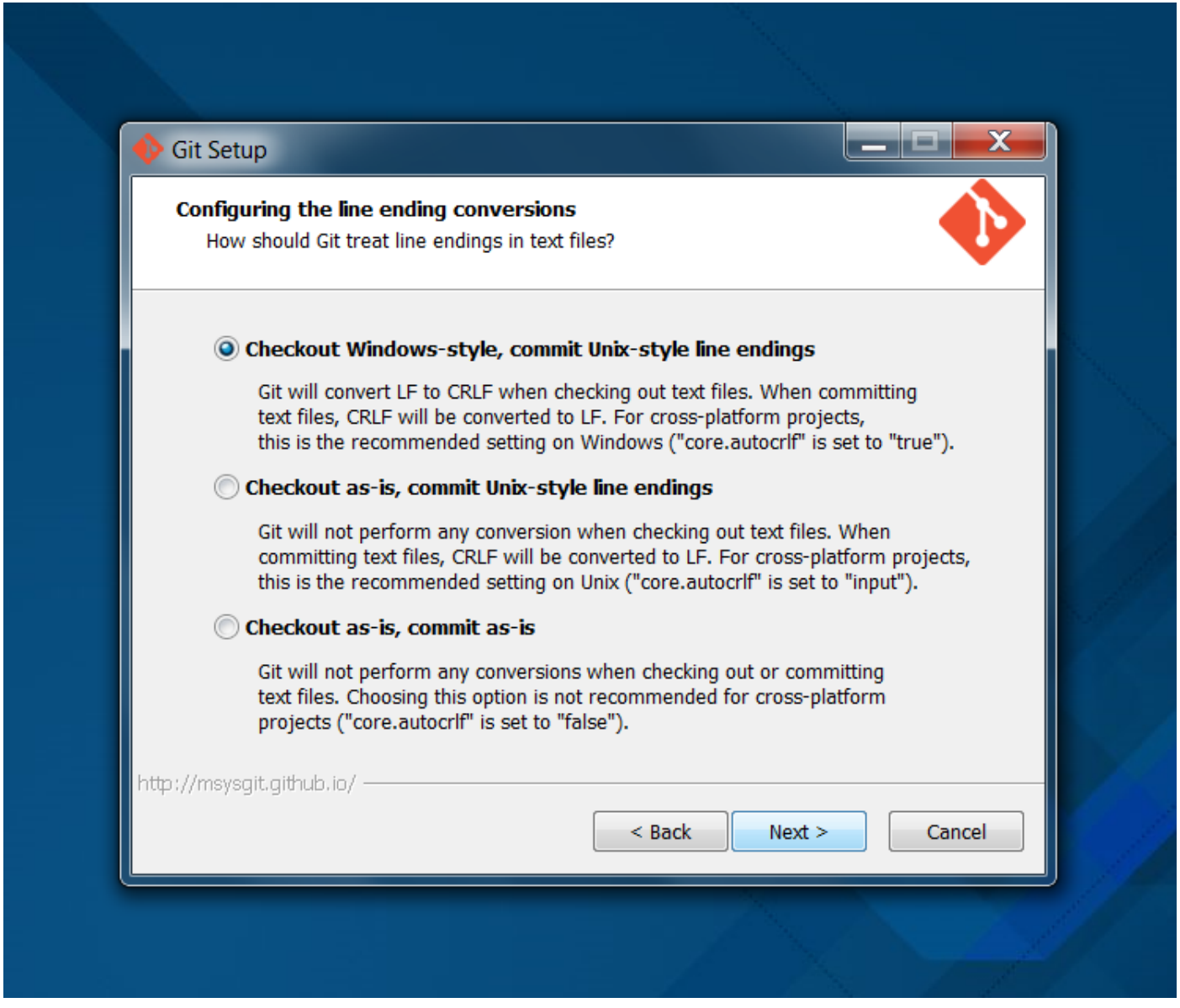
Installing Git on MS Windows: Step 3



Installing Git on MS Windows: Step 4

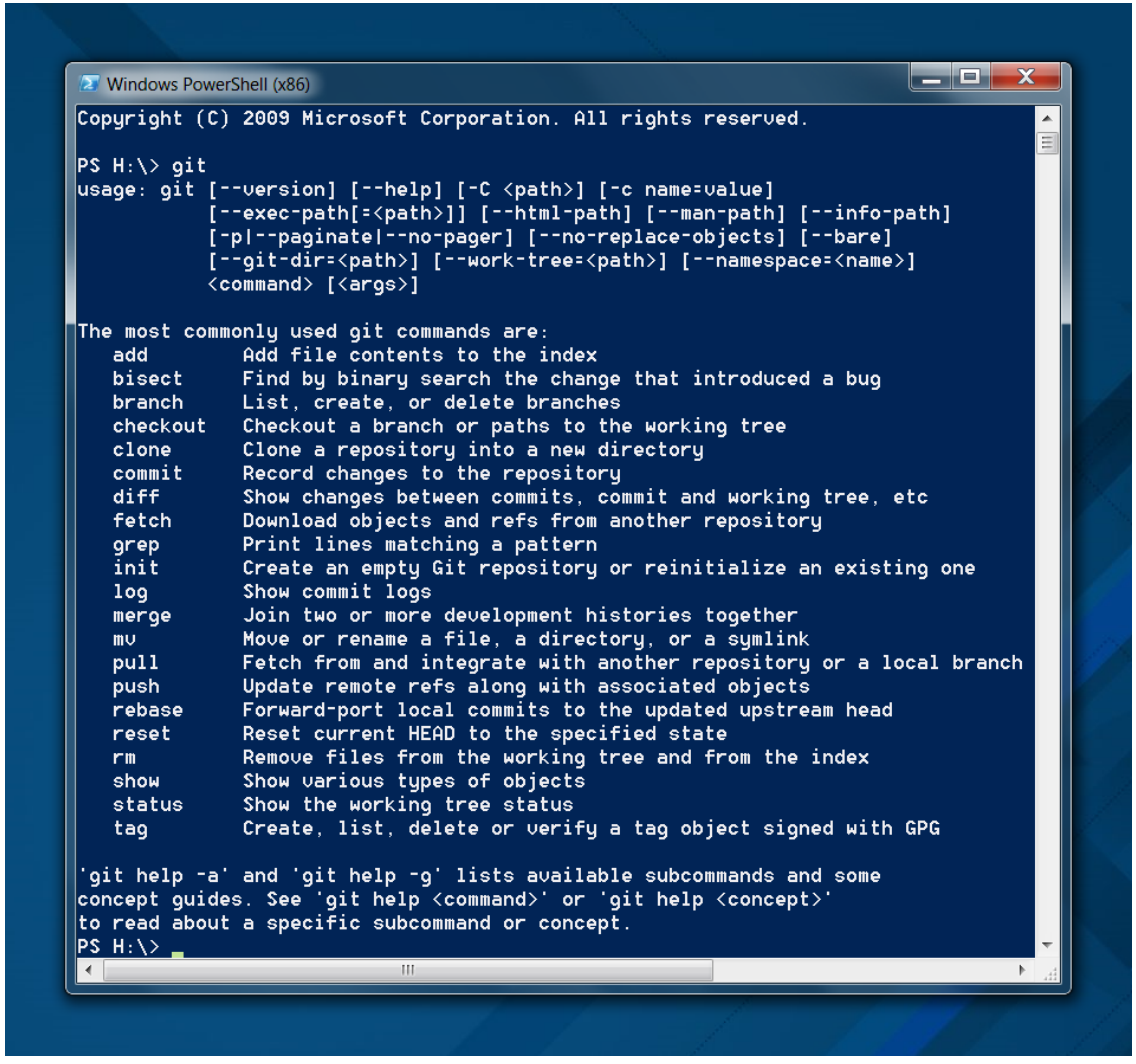


Installing Git on MS Windows: Step 5

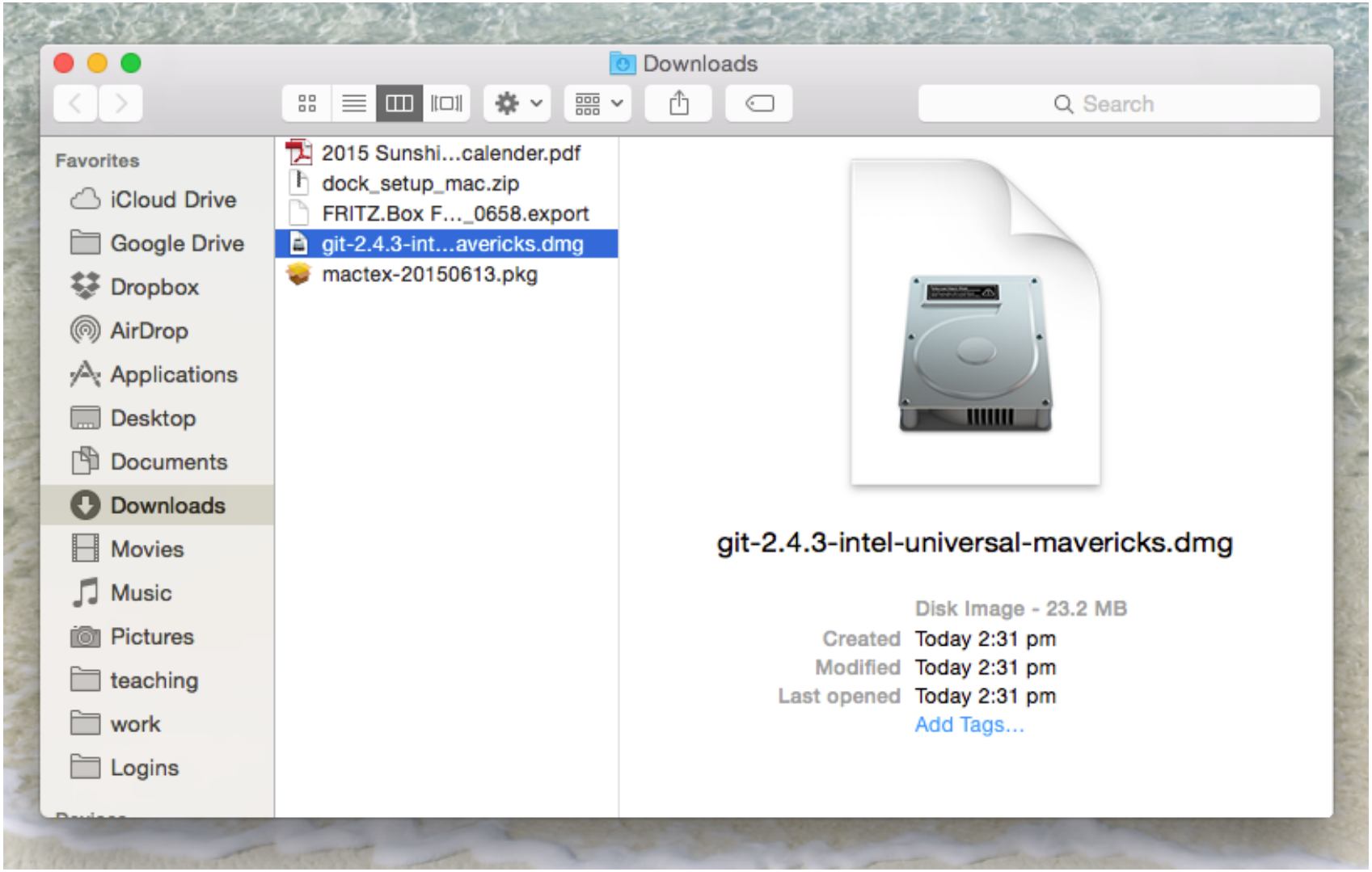


Check Git is installed on MS Windows

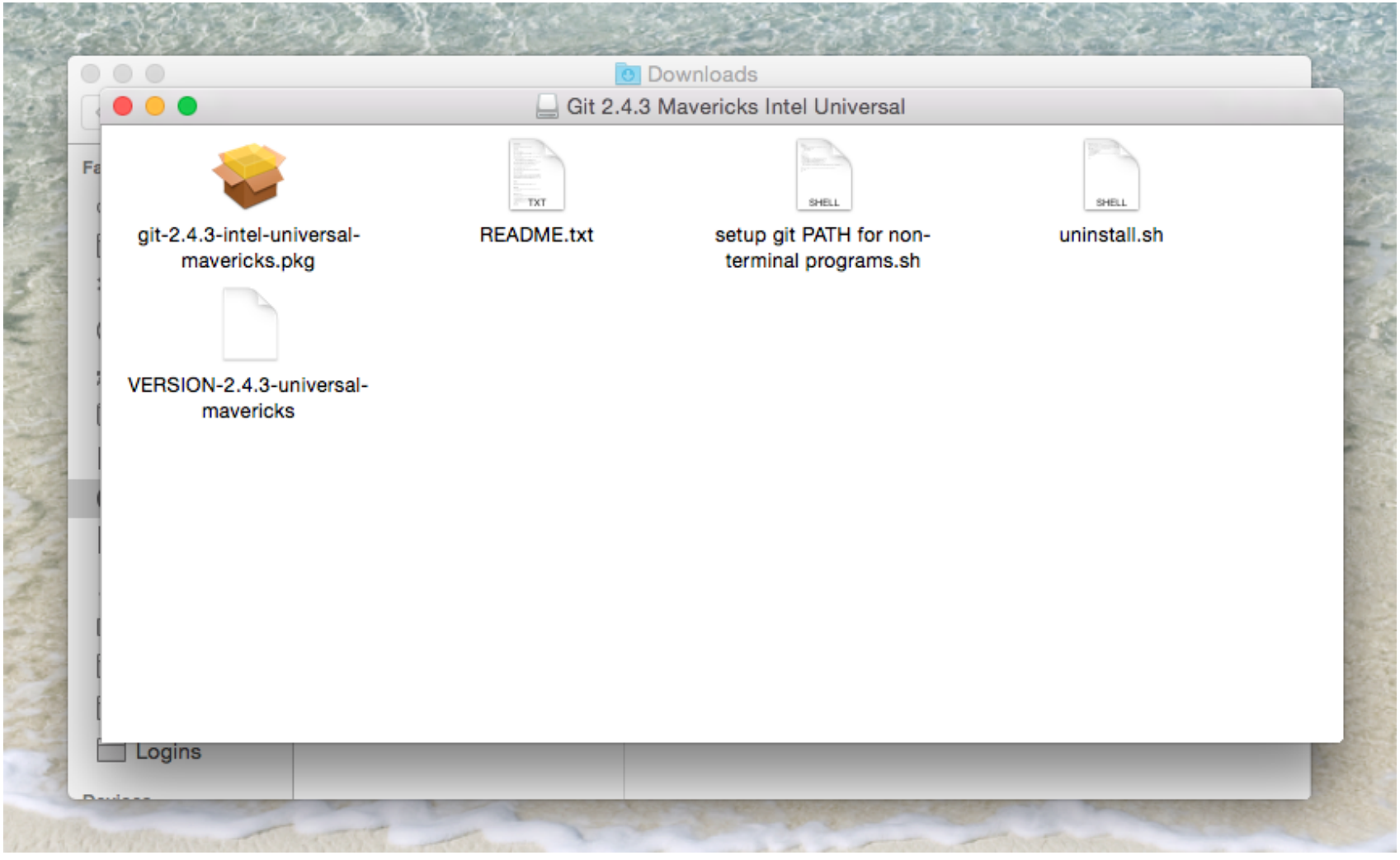
Open a PowerShell and type *git* then press 'Enter' and you should see output like that below:



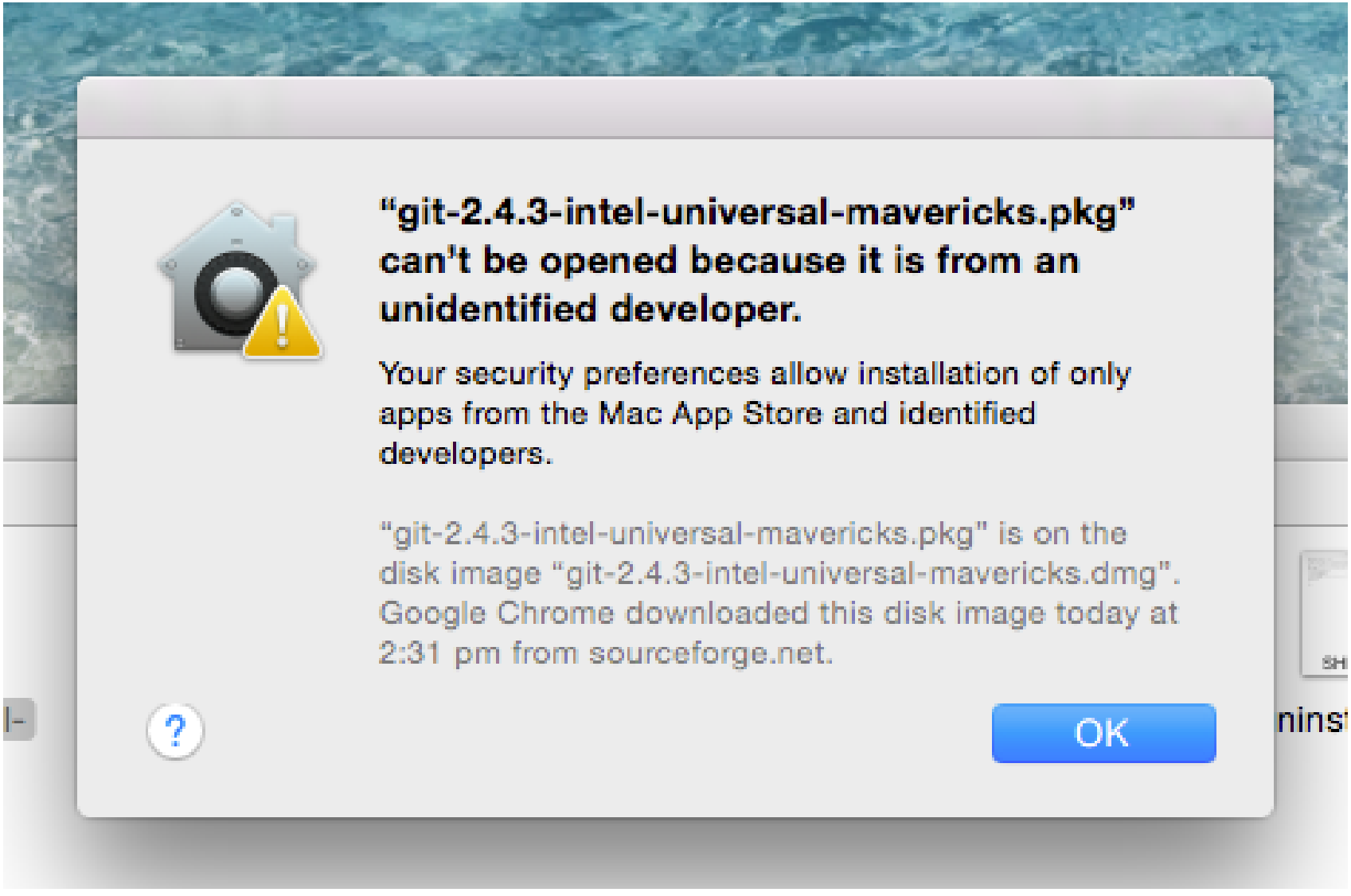
Installing Git on MacsOS: Step 1



Installing Git on MacsOS: Step 2



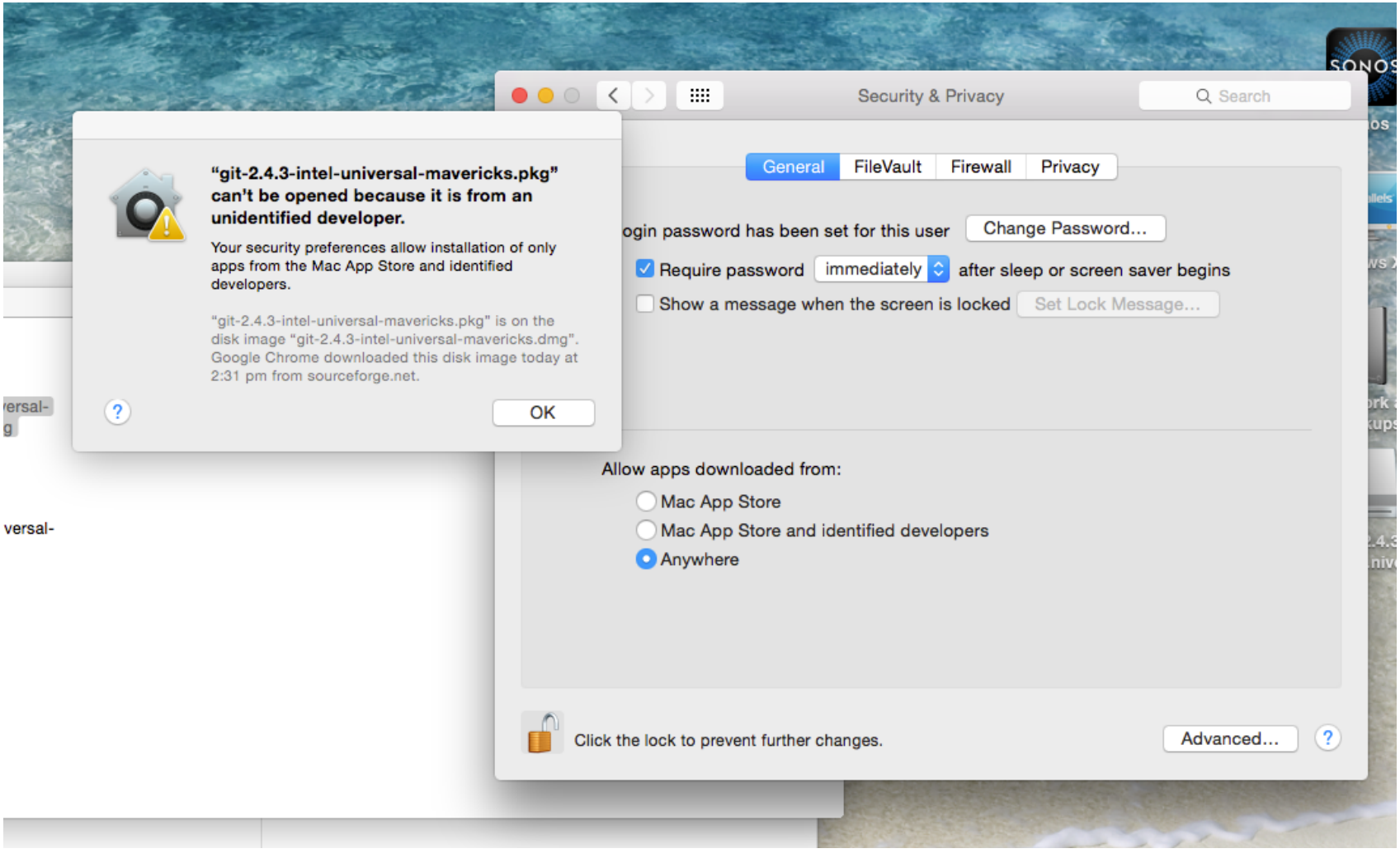
Installing Git on MacsOS: Step 3



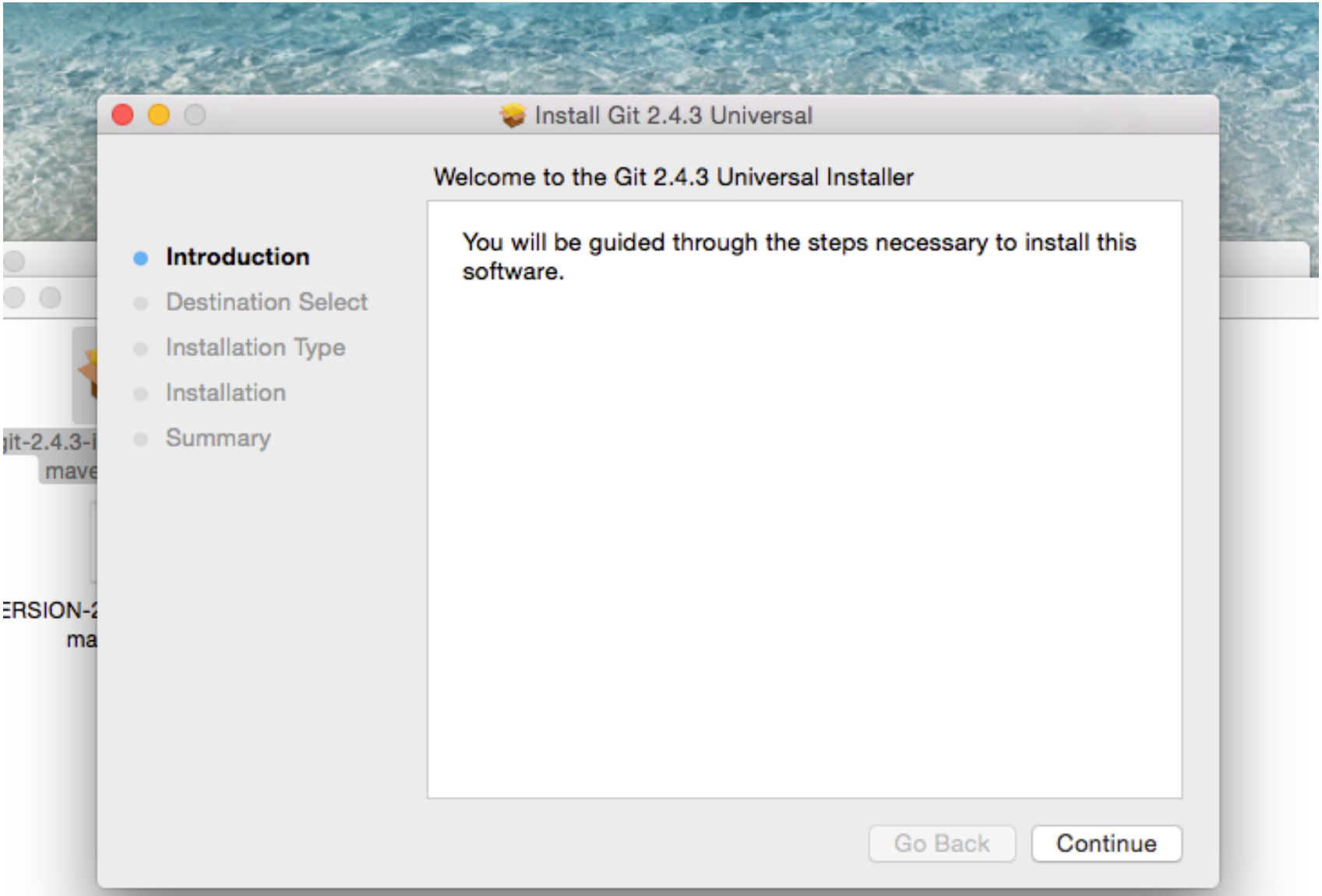
Installing Git on MacsOS: Step 4

You may need to relax your security setting briefly

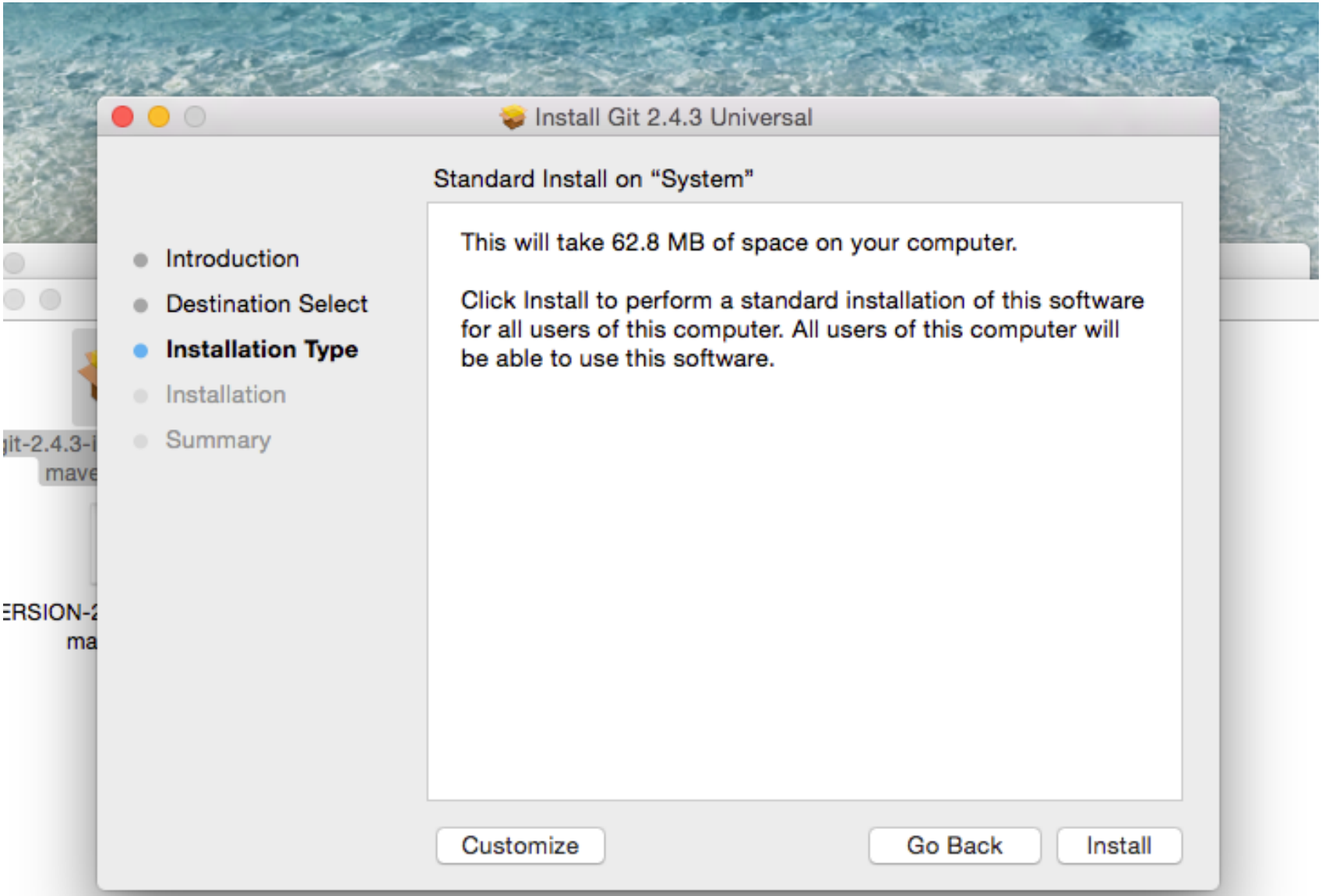
If this makes you uncomfortable you're welcome to watch the session rather than participating



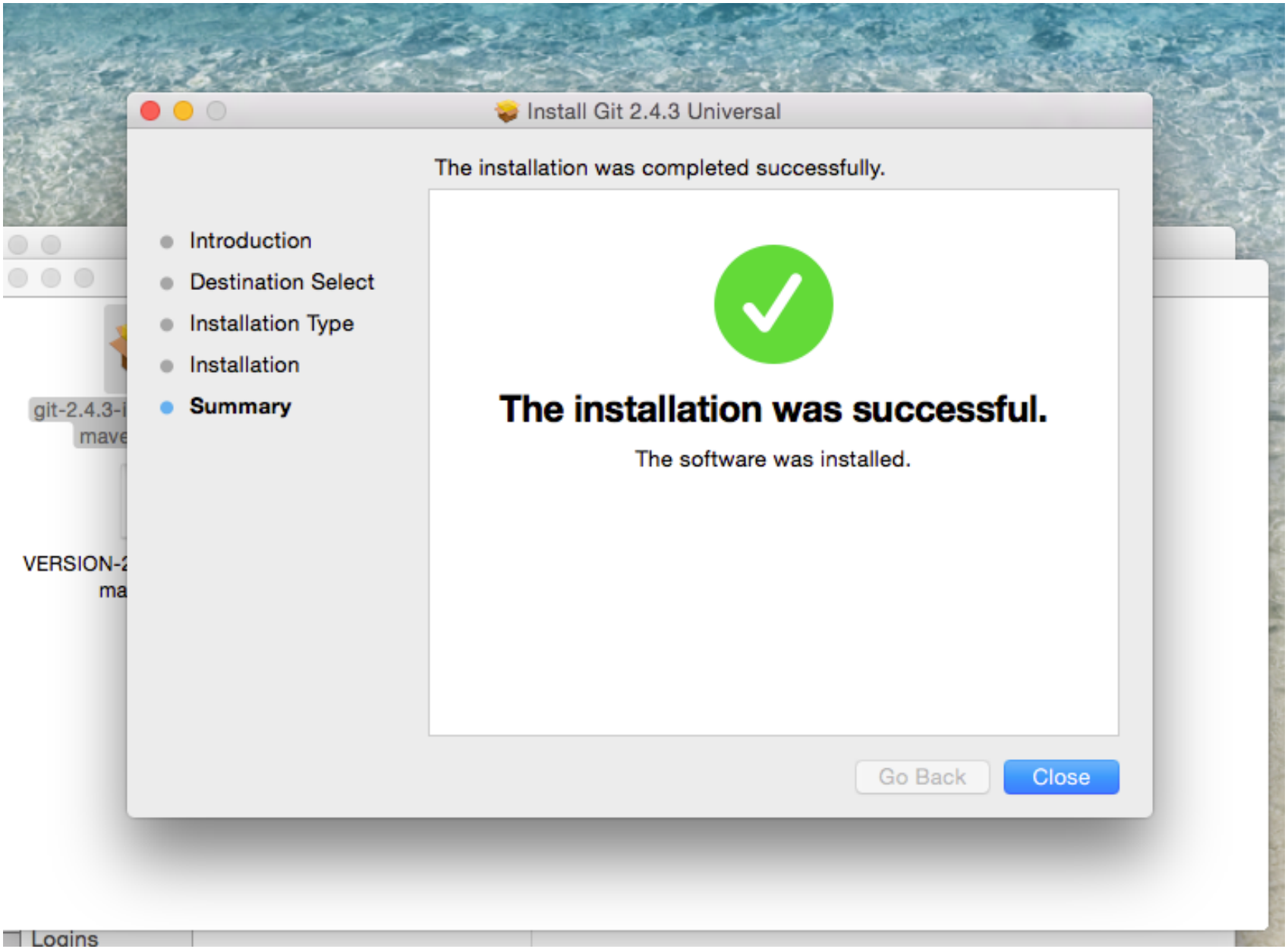
Installing Git on MacsOS: Step 5



Installing Git on MacsOS: Step 6



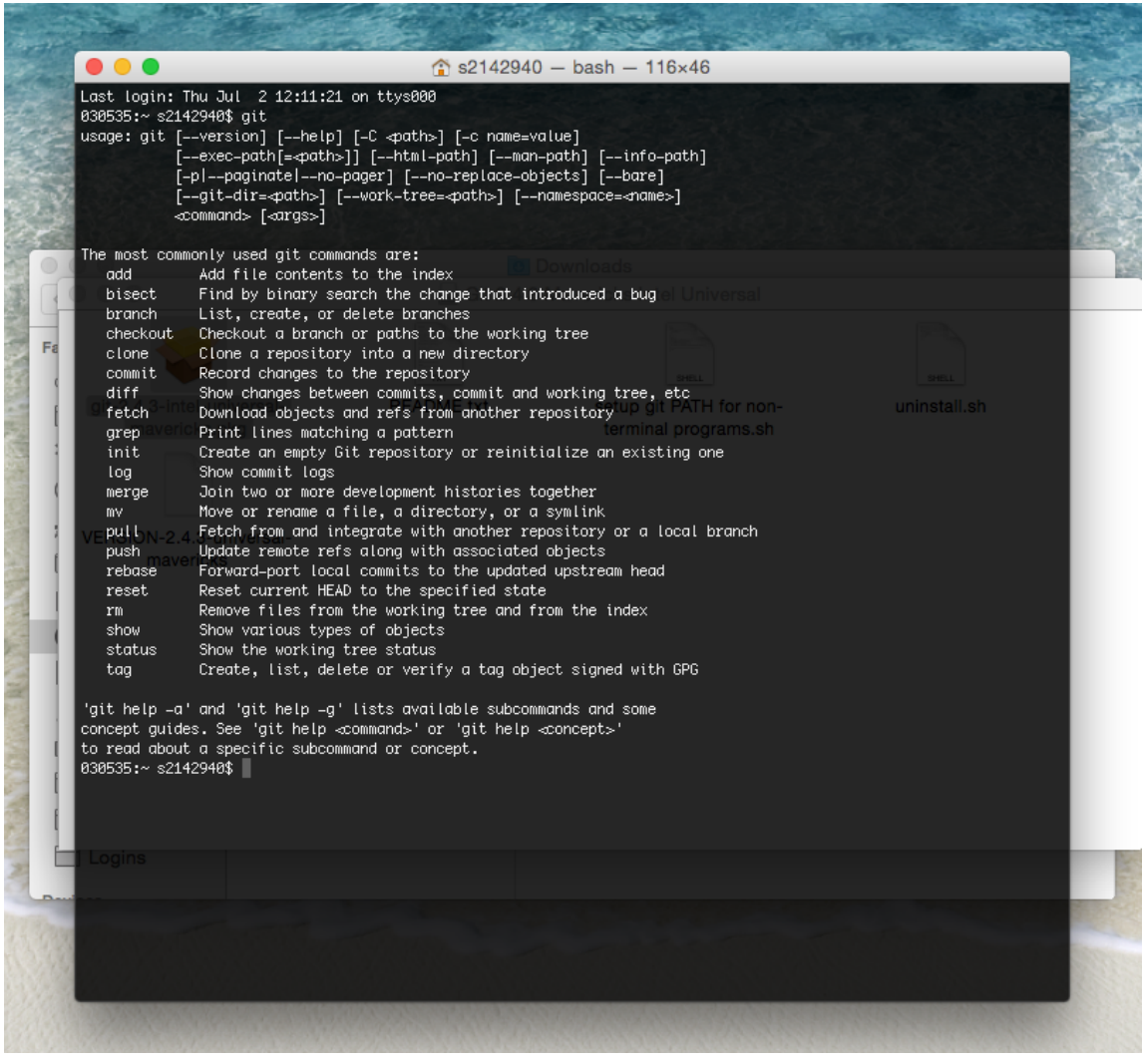
Installing Git on MacsOS: Step 7



Installing Git on MacsOS: Step 8

Check Git is installed:

Open a terminal, type *git* and press 'Return' and you should see output like that below:



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

Benefits of Using Version Control Software

Version Control for Working Solo

- Easy to revert to previous versions of a file without loosing any of the subsequent revisions
- Simple to create multiple versions of a file and edit them in parallel with the option combine them again at a later date

Version Control for Collaborating

- Greatly simplifies your life
- Different contributors can edit different versions of the same files concurrently
- Tools exist to systematically combine versions of the same file from different contributors
- time stamped record of all changes from all contributors - if someone breaks something you can revert to when it last worked and work forward to discover the bug

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 - both with free plans) support version control with Git

Git is Distributed Version Control

Distributed Version Control means that:

- Every contributor has a copy of the entire database up to when they copied it (all revisions of all files)
- two or more contributors can work on the files in their copy of the database independently then combine their modifications systematically at a later date

Results is a fast and flexible version control system:

- contributors can work independently offline connecting to the server sporadically to submit their changes
- most operations are performed locally (on individual contributors' computers) so less waiting for the server

Why Git on the Command Line?

- Makes discrete components of the workflows obvious.
- This in turn will enable you to migrate easily to one of the many GUIs at a later date should you so desire.
- The workflows we will practise today function identically at MS Windows, MacOS and GNU+Linux command line interfaces.

Plan for this Module

Part 1

Version Control for Working Alone

1. Set up Git on your laptop to communicate with the GitHub servers
2. Create a local **clone** of a **remote repository**
3. Make some changes to the files in your local copy of the repository
4. **commit** these changes to your copy of the history of revisions of these files
5. **push** these changes to your local copy of the repository to the remote copy on the GitHub servers
6. make some more changes
7. revert to a previous version of a file in this repository safely by **checking out** an previous **commit** to a new **branch** of the revision history of that file

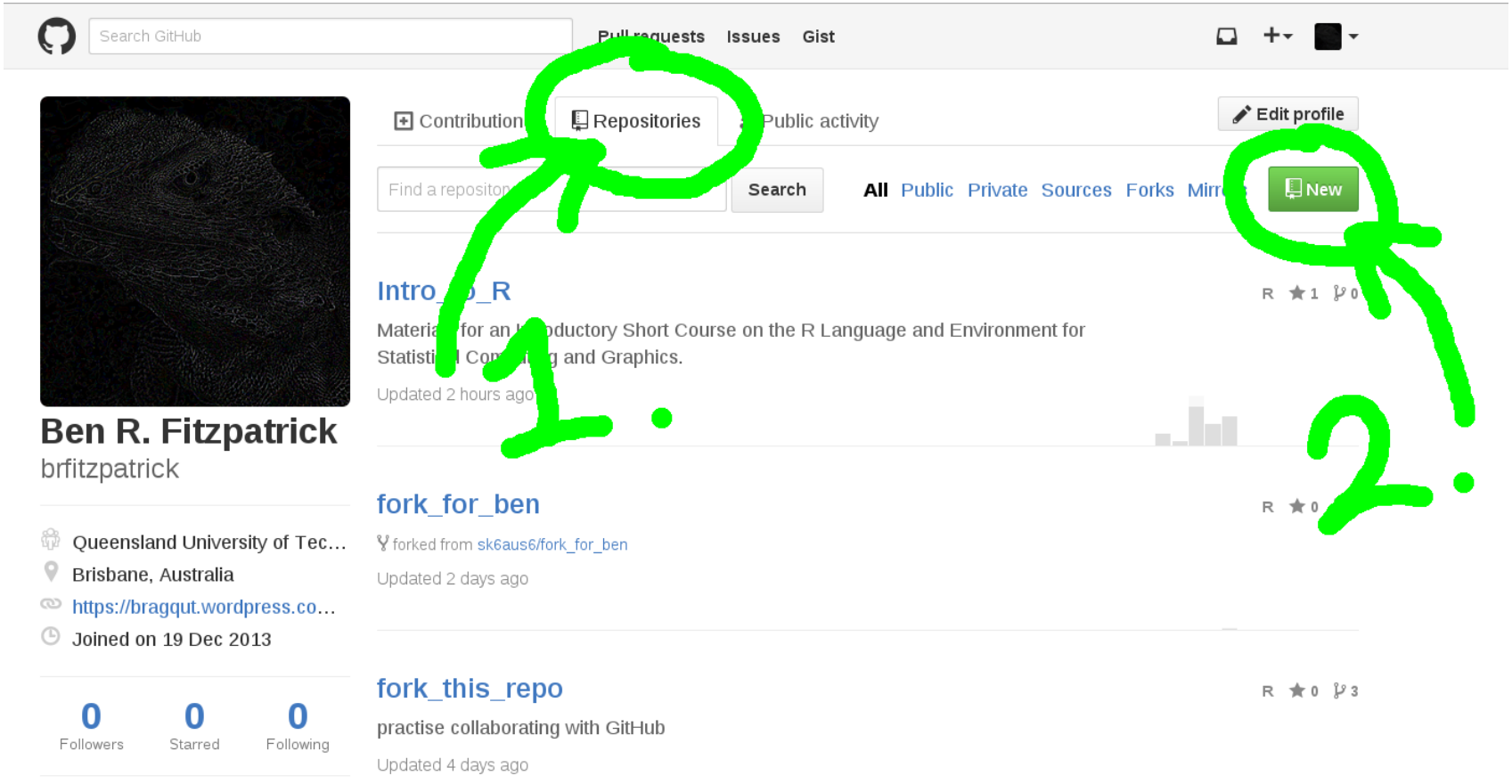
Plan for this Module
Part 2

Collaborating with Git & GitHub

1. fork the repository of a fellow course attendee
2. clone your fork of their repository creating a local copy on your hard drive
3. edit one of their files in your local copy of their repository
4. commit your changes to your local copy of their repository
5. summit a pull request - requesting the repository owner merge your changes with their branch of the repository
6. watch the merge process on your fellow course attendee's computer
7. switch roles and repeat

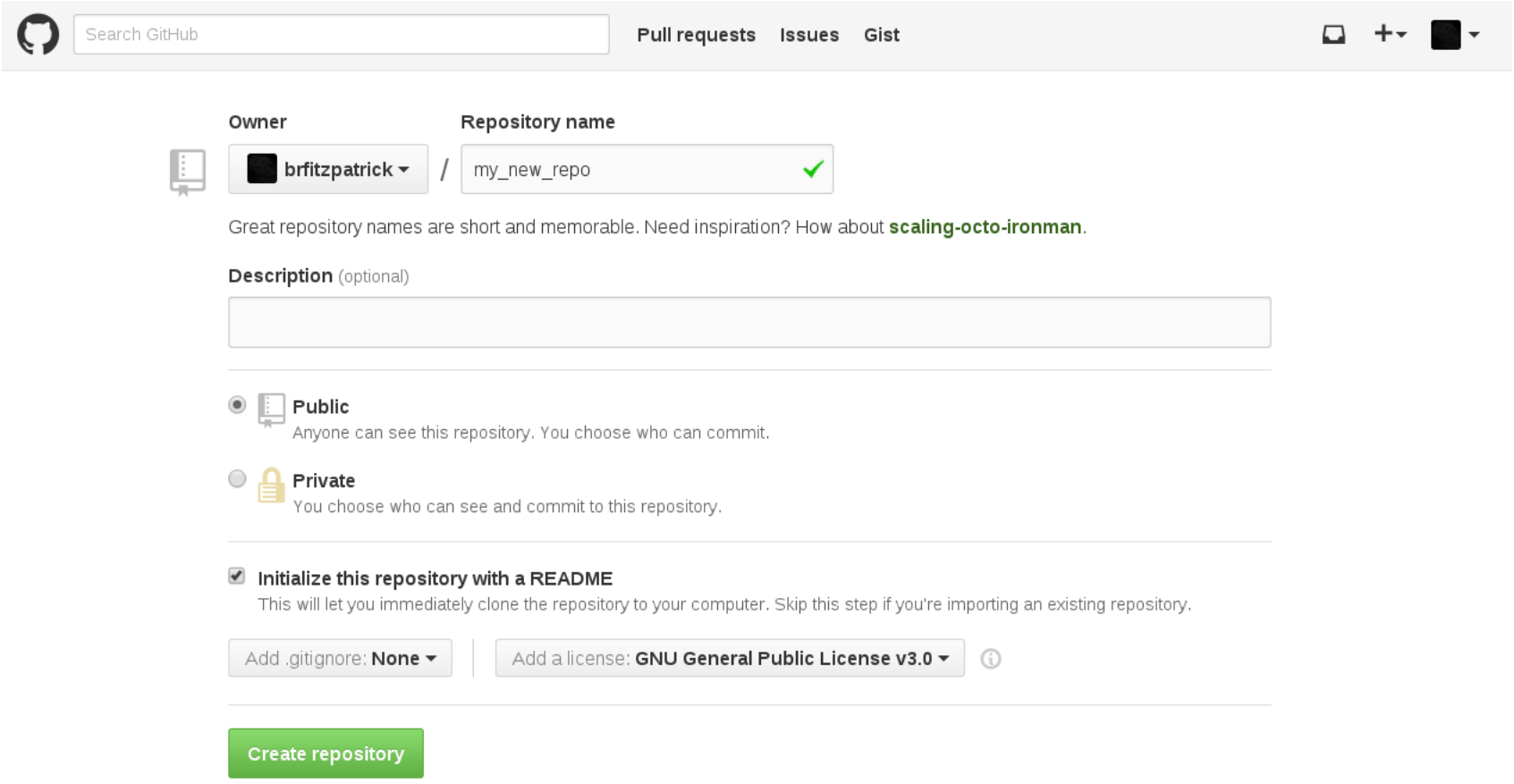
Creating a Repository on the GitHub Servers

Log into the GitHub Web Service & Create a New Repository



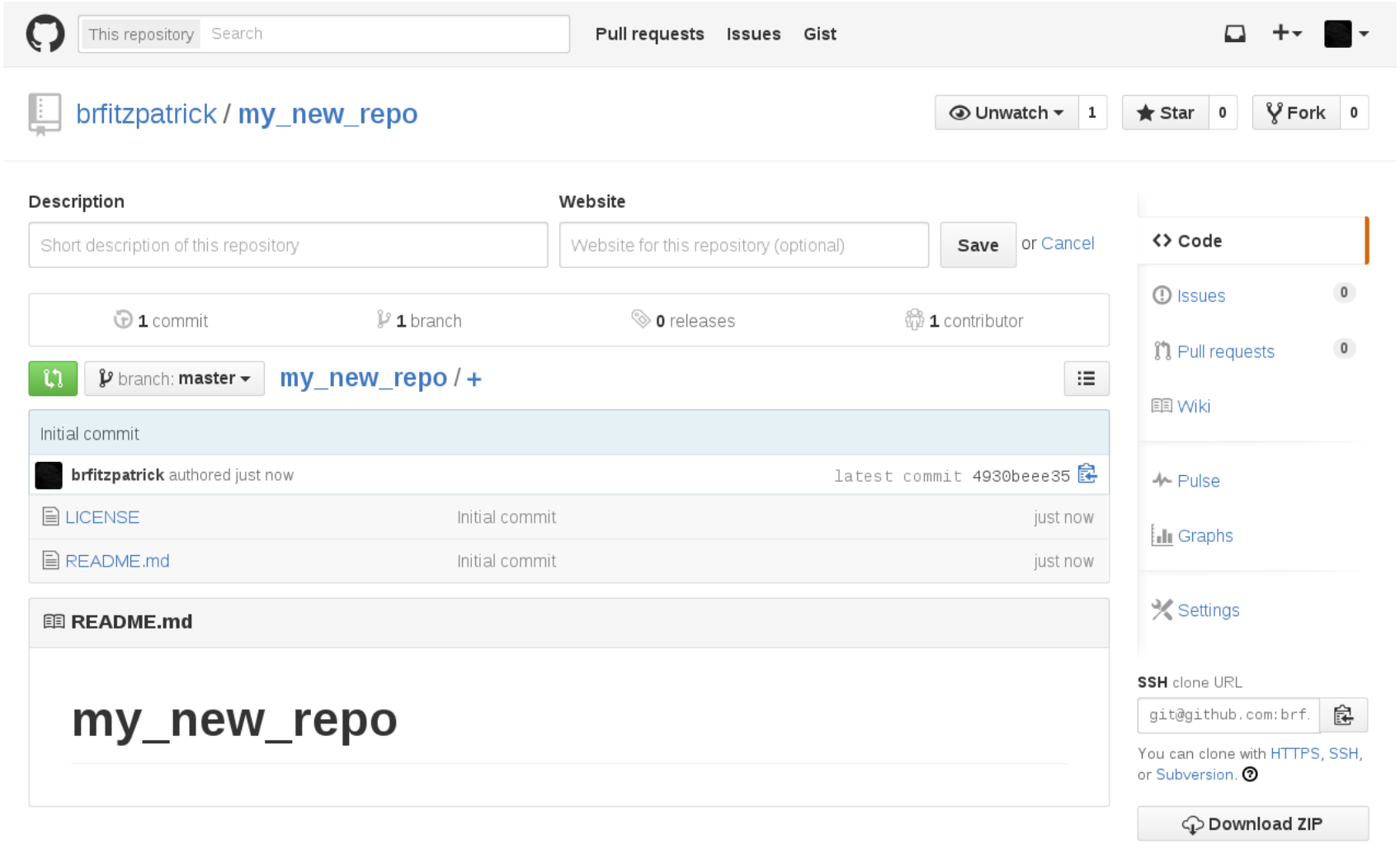
Creating a Repository on the GitHub Servers

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 servers to your hard drive.

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

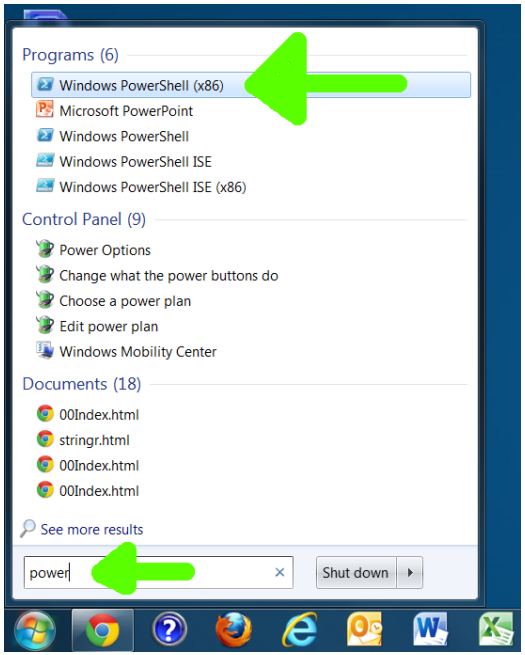
- all the files
- the entire history of snapshots of files states created each time you commit modifications to the respository

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 local clone of the repository.

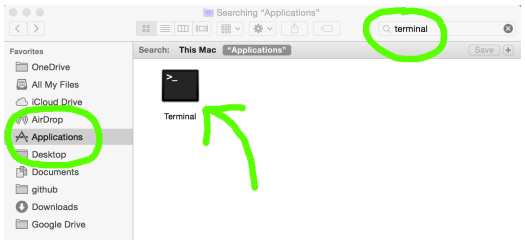
Git on the Command Line

Accessing a command line interface in your OS of choice

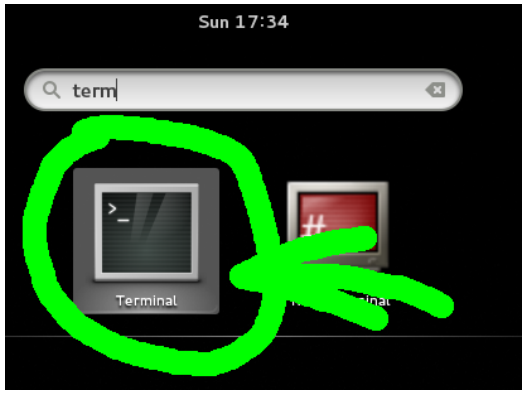
MS Windows
Open a ‘PowerShell’



Mac OS X & later
Open a ‘Terminal’

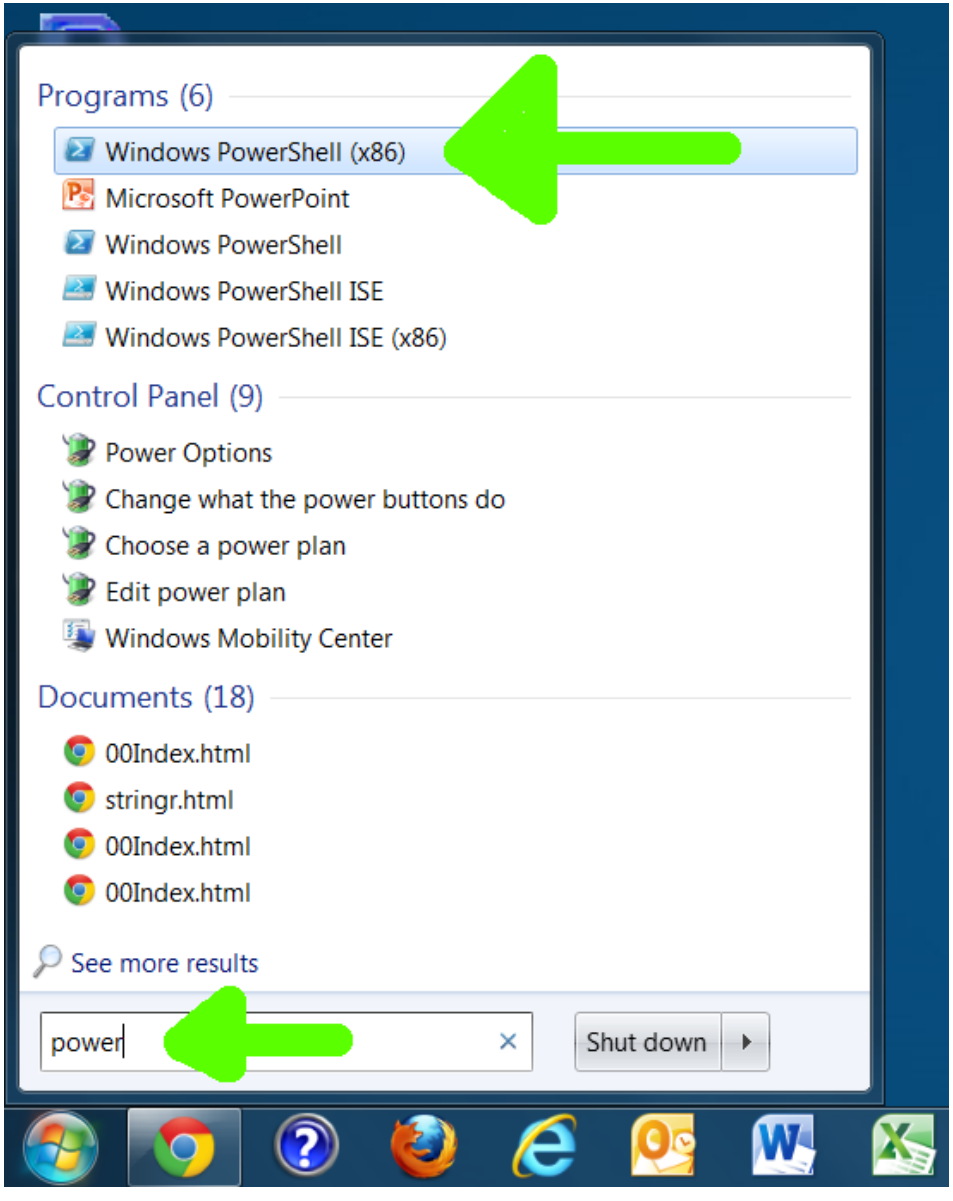


GNU+Linux
Open a ‘Terminal’



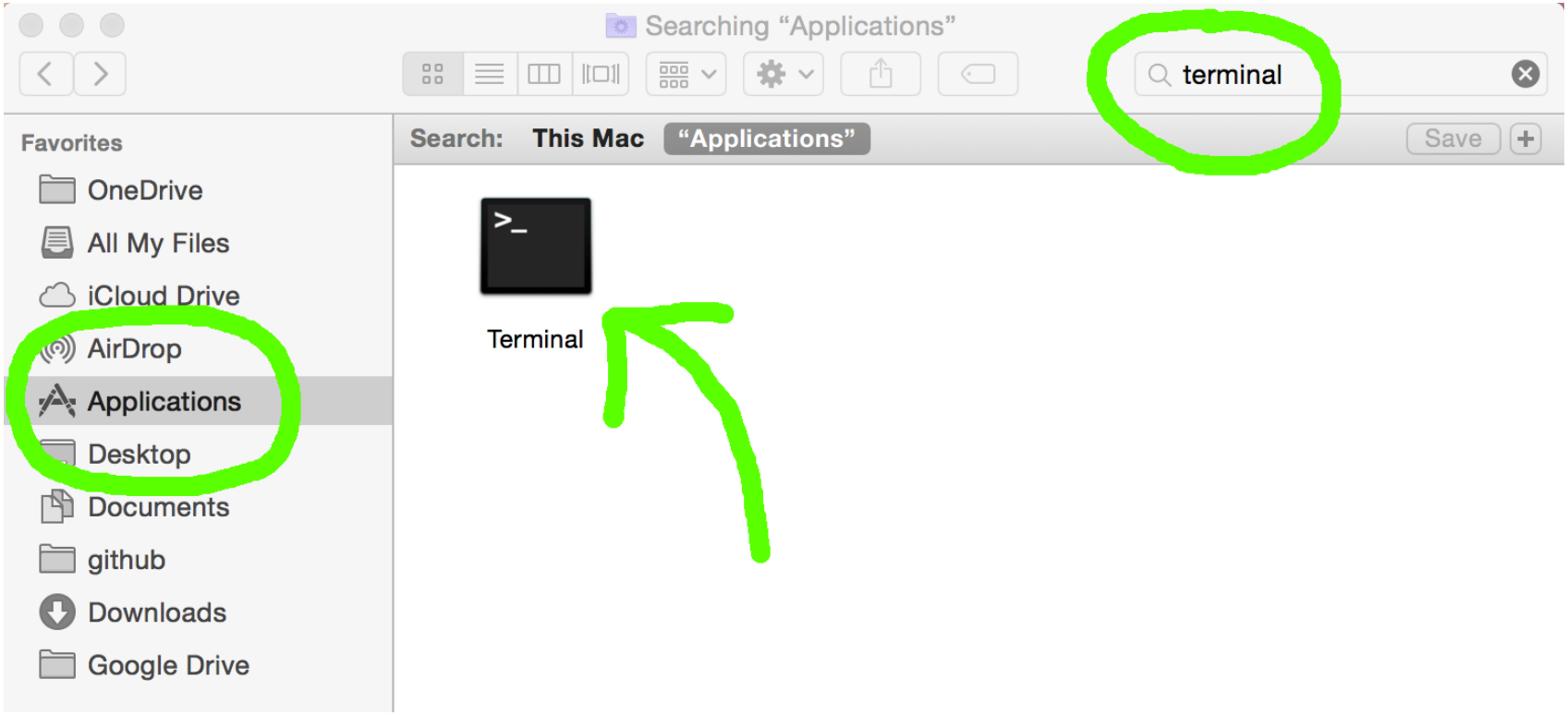
Accessing a command line interface in MS Windows

Open a 'PowerShell'



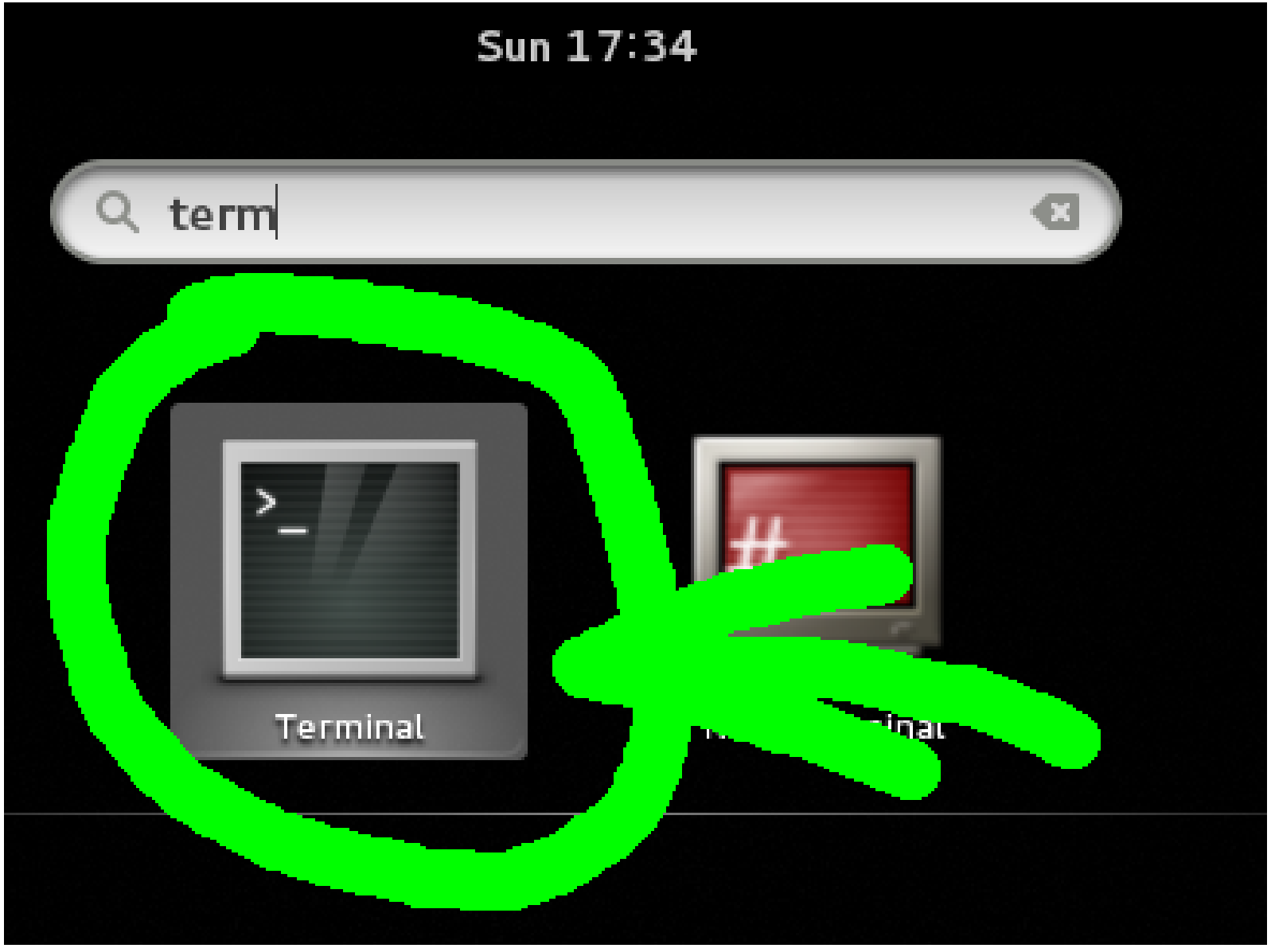
Accessing a command line interface in MacOS

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 as the global editor

You'll need relevant the text editor installed for the below to work:

```
# 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 then list the files and folders present at that location:

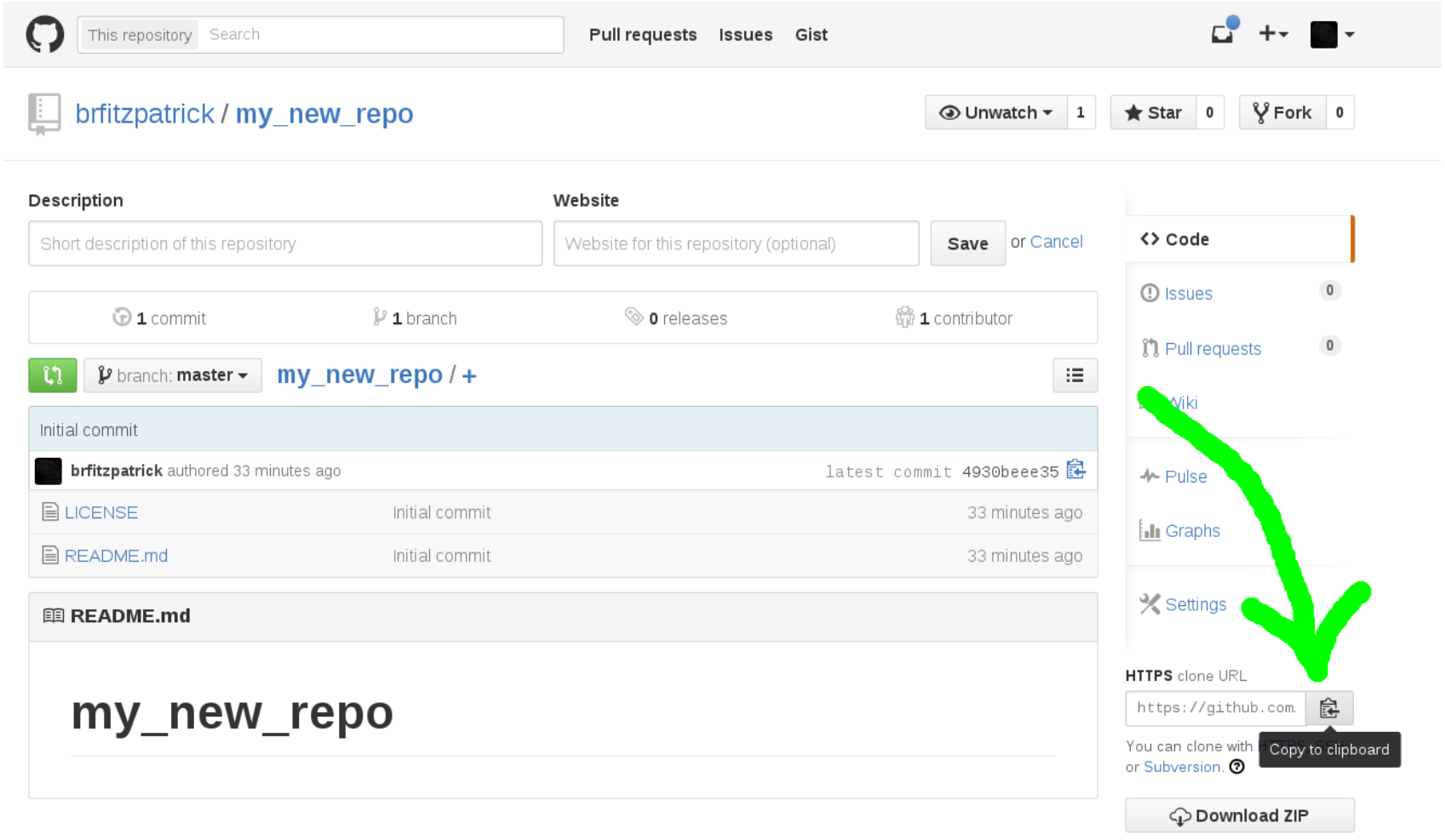
Powershell Users

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

Terminal Users

```
> cd /home/username/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 four 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
- **untracked** Git is not recording revisions to this file

1. you add files to or modify existing files in your local copy of the repository
2. you stage these modified files to be committed to the database for the repository
3. 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
4. you push the changes recorded in this commit to your remote copy of this repository on the GitHub servers

1. Open your favourite program for authoring R code (e.g. RStudio)
2. Create an new R script (it can be very short)
3. Save your R Script in the local version 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 you used to write it.
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.

```
> 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.

Furthermore, now that we have ‘pushed’ our most recent changes our local copy of the repository to the remote copy of the repository on the GitHub servers these two versions of the repository 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 a ‘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 an old save without first making a current save we will loose the unsaveed changes

List existing branches of current repository:

```
git branch
```

then, choose a commit you'd like to go back to, copy the corresponding SHA from the history tab on the website and checkout this previous commit to a new branch called 'testing', here abbreviated to ($SHA = 58...a44$):

```
> git checkout -b testing 58...a44
  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.
Next up we'll make some changes to the version of this file on the branch 'testing' so that it will diverge from the version on the branch 'master'.
Once again check you're on the branch called 'testing'

```
> git branch
```

then make some changes on testing branch by editing the version of the file currently in your working directory then save those changes, add them, commit them and push them 'upsteam' to server:

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

- Now if you check the website your repository should now have two branches.

- We
  also know that each branch contains a slightly different file now.

- Next we'll 'merge' your changes from 'testing' back into the 'master' branch.

To merge into a branch we must be on that branch:

```
> git checkout master
```

Now we will attempt to merge 'testing1' into current branch (master)

```
> git merge testing1
  Auto-merging filename.R
  CONFLICT (content): Merge conflict in
  filename.R
  Automatic merge failed;
  fix conflicts and then commit the result.
```

Resolving a Merge Conflict with a Merge Tool

```
git mergetool

This message is displayed because 'merge.tool'
is not configured.
See 'git mergetool --tool-help' or 'git help
 config' for more details.
'git mergetool' will now attempt to use one
of the following tools:
meld opendiff kdiff3 tkdiff xxdiff
...
emerge vimdiff
Merging:
filename.R
```

Resolving a Merge Conflict with a Merge Tool

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

Resolving a Merge Conflict with a Merge Tool

Demonstrate **Meld**

Collaborating

Forking a repository is creating a branch from a repository you don't own.

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

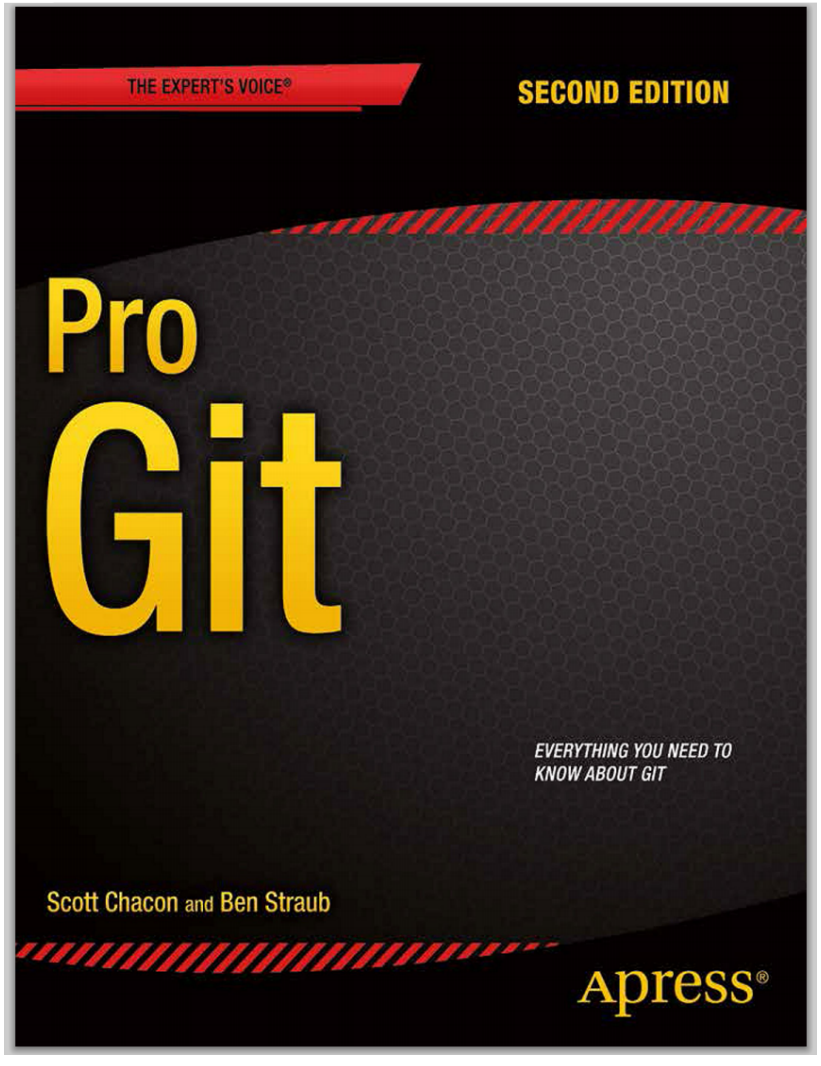
Accepting and rejecting Pull Requests

Merging your friends' branches

- Create a new branch to merge your friend's changes into
- switch to that branch
- merge their changes with **Meld**

Recommended Further Reading

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



Recommended Further Reading:

https://help.github.com/

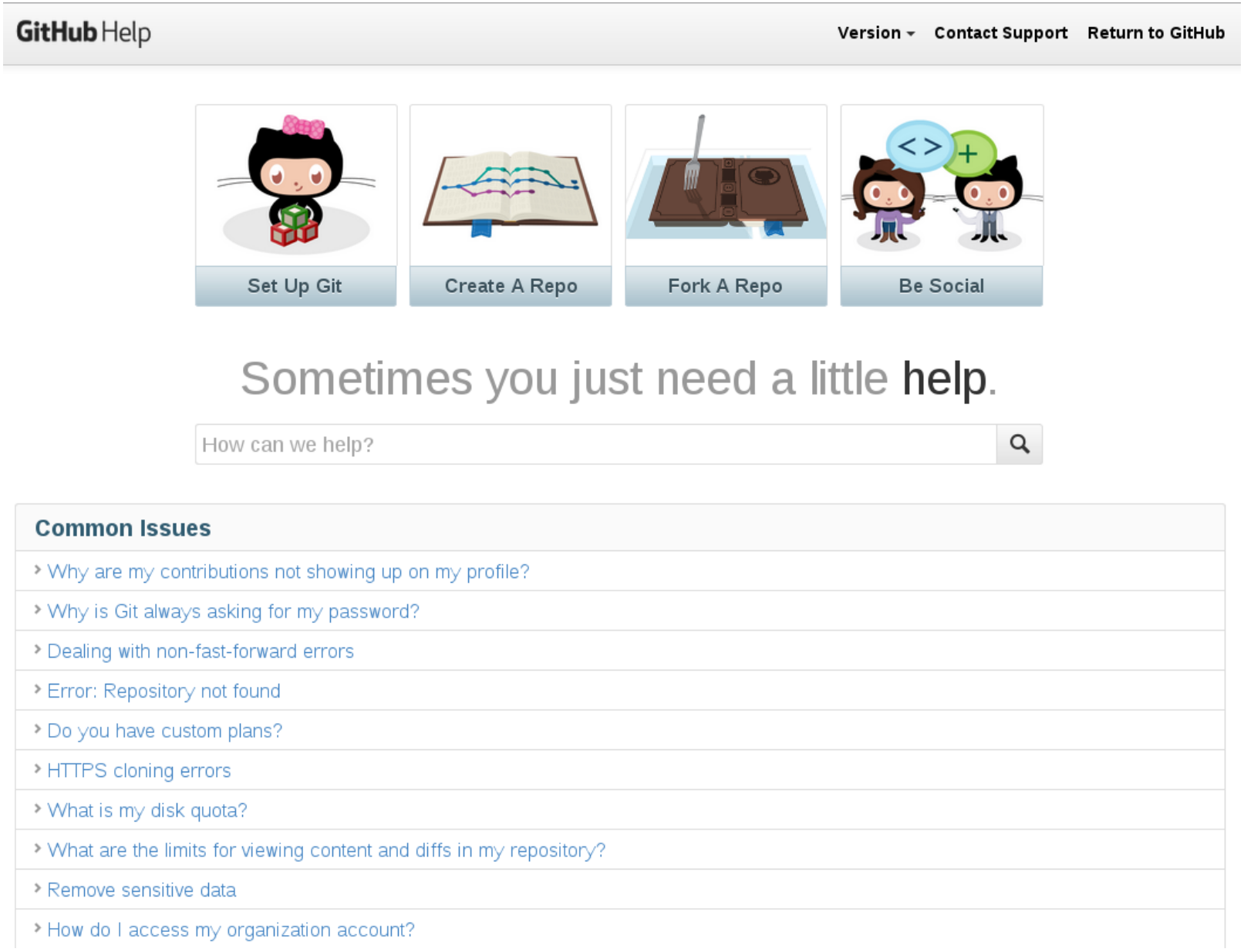


Image Credits

- Git Logo by Jason Long
  `http://git-scm.com/downloads/logos`
- GitHub Logo, `www.github.com`
- R Logo, `www.r-project.org`