Advanced data science for Geographers

Professor Itai Kloog

The Department of Geography and Environmental Development

Ben Gurion University

(Updated: 2023-03-24)

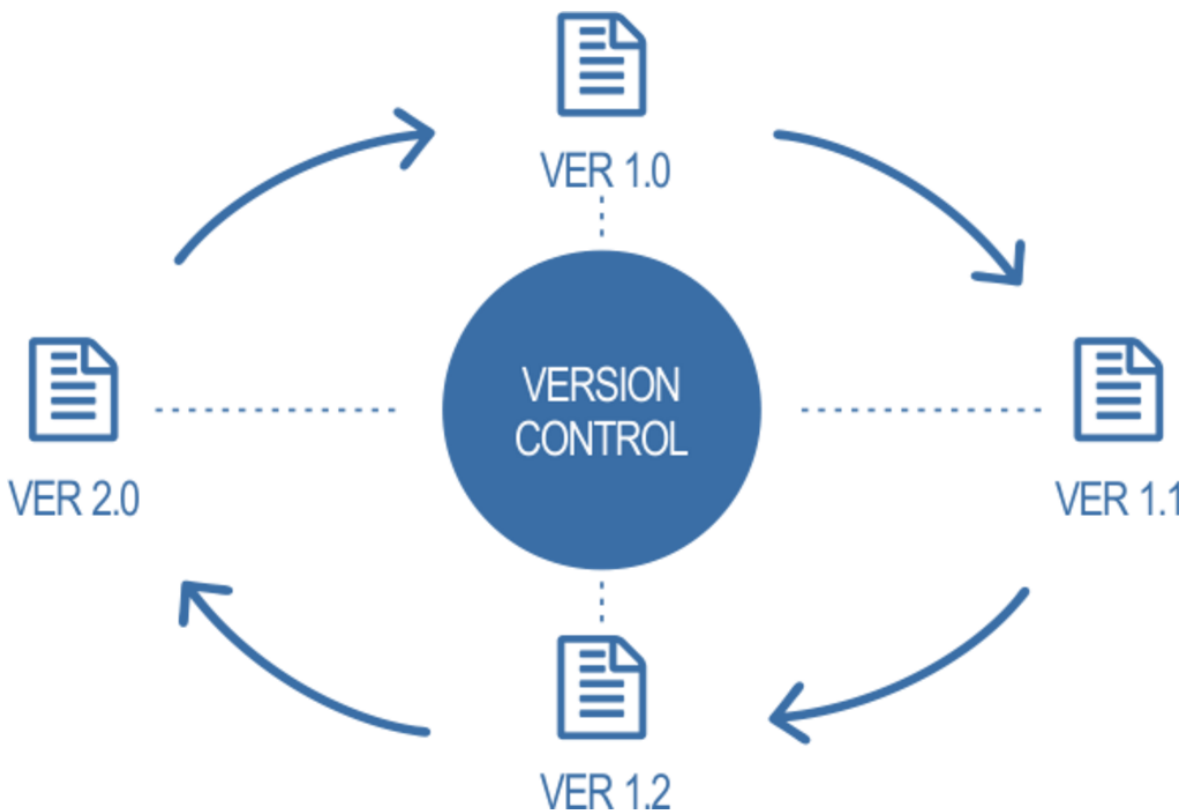
Week 3: Introduction to Git

Version Control

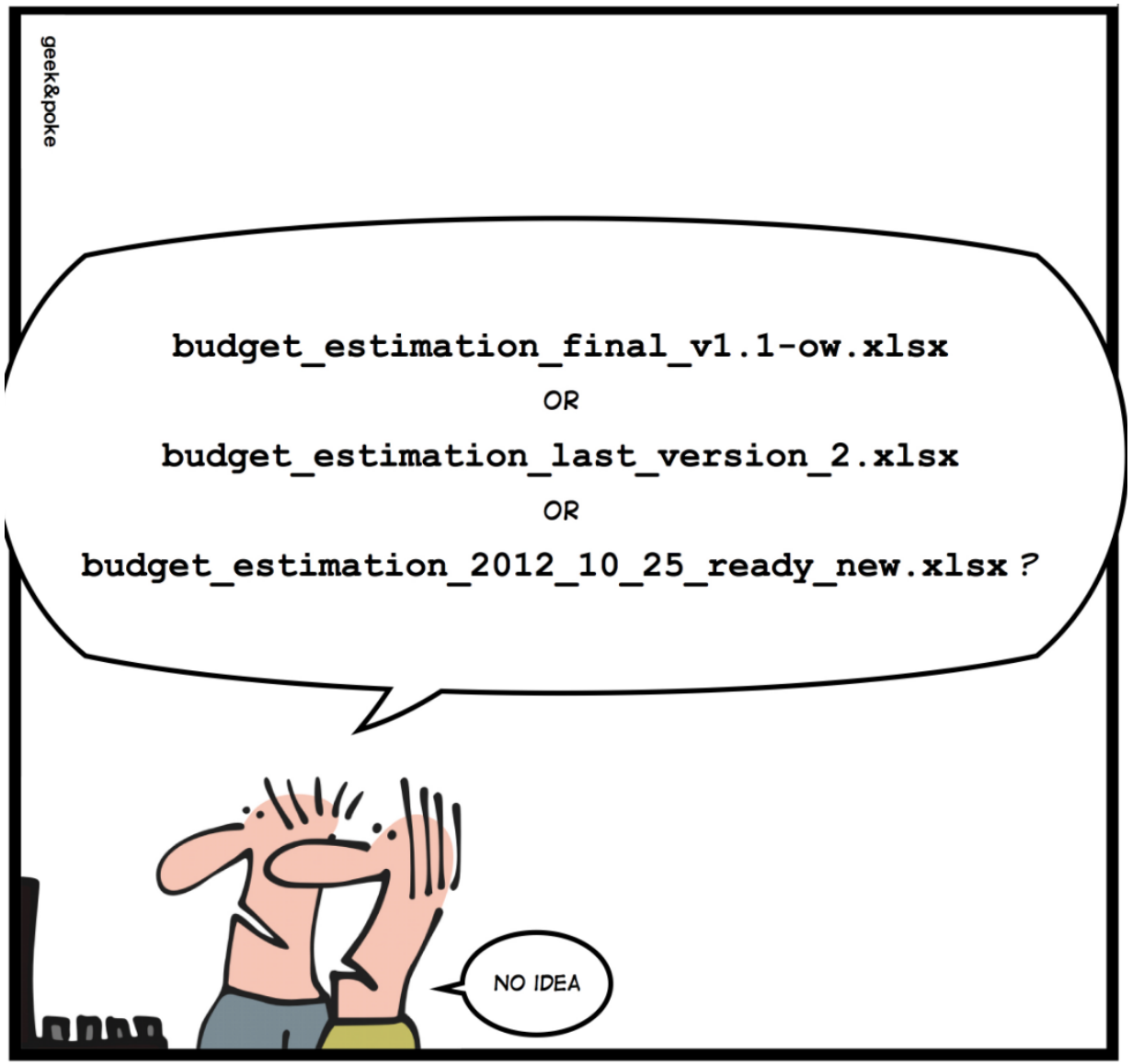
What is Version Control?

“Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.”

http://git-scm.com/book/en/Getting-Started-About-Version-Control



What is Version Control?



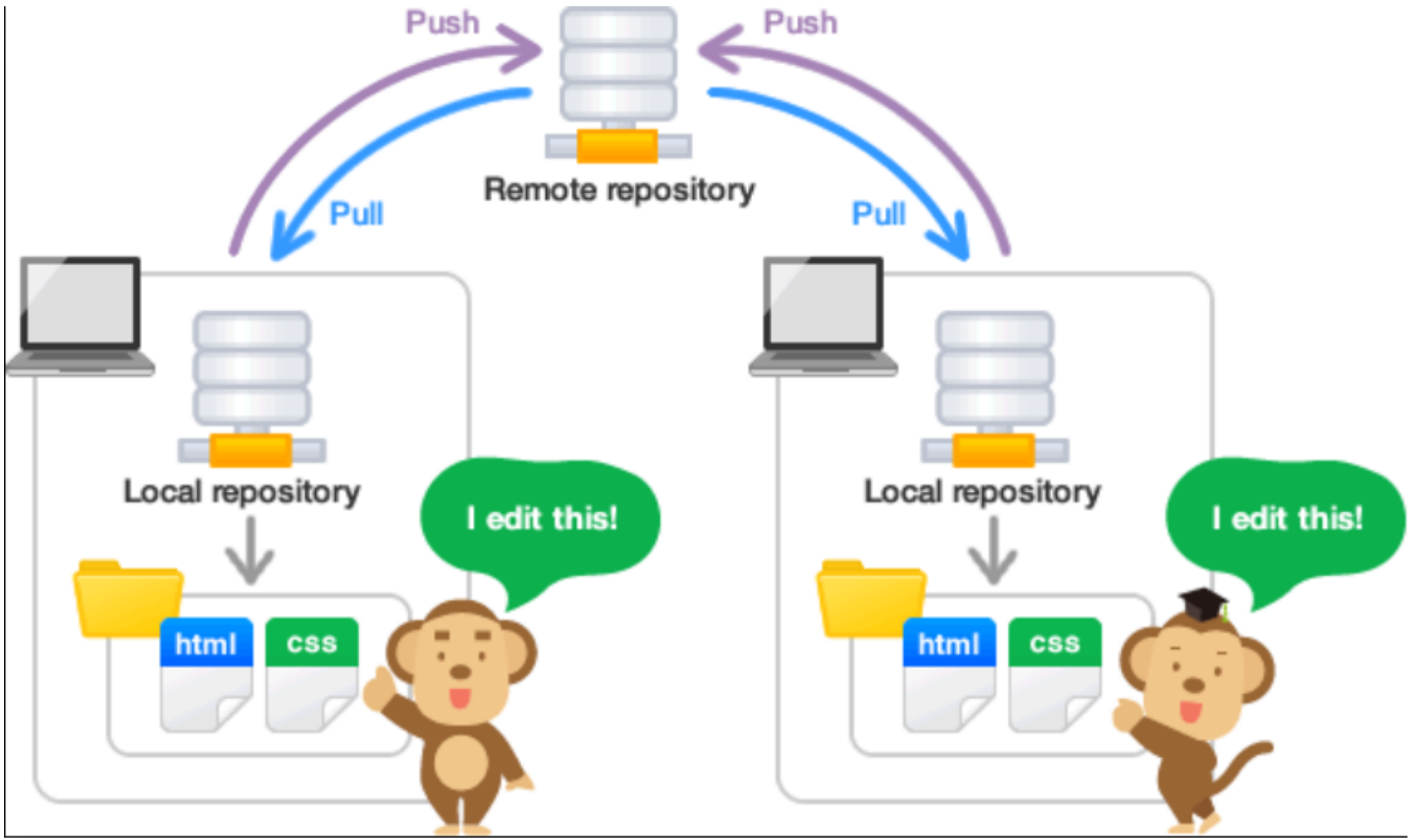
What is Version Control?

- Many of us constantly create something, save it, change it, then save it again
- Version (or revision) control is a means of managing this process in a reliable and efficient way
- Especially important when collaborating with others

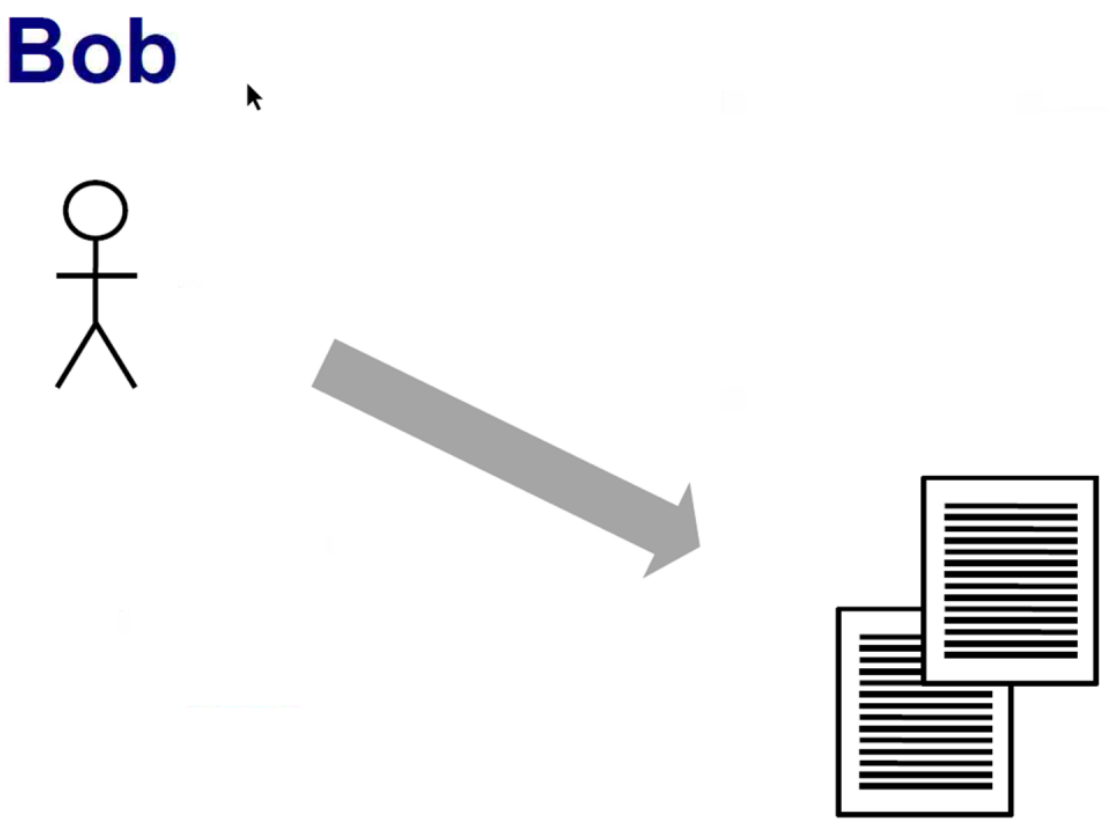
http://en.wikipedia.org/wiki/Revision_control

What is Version Control?

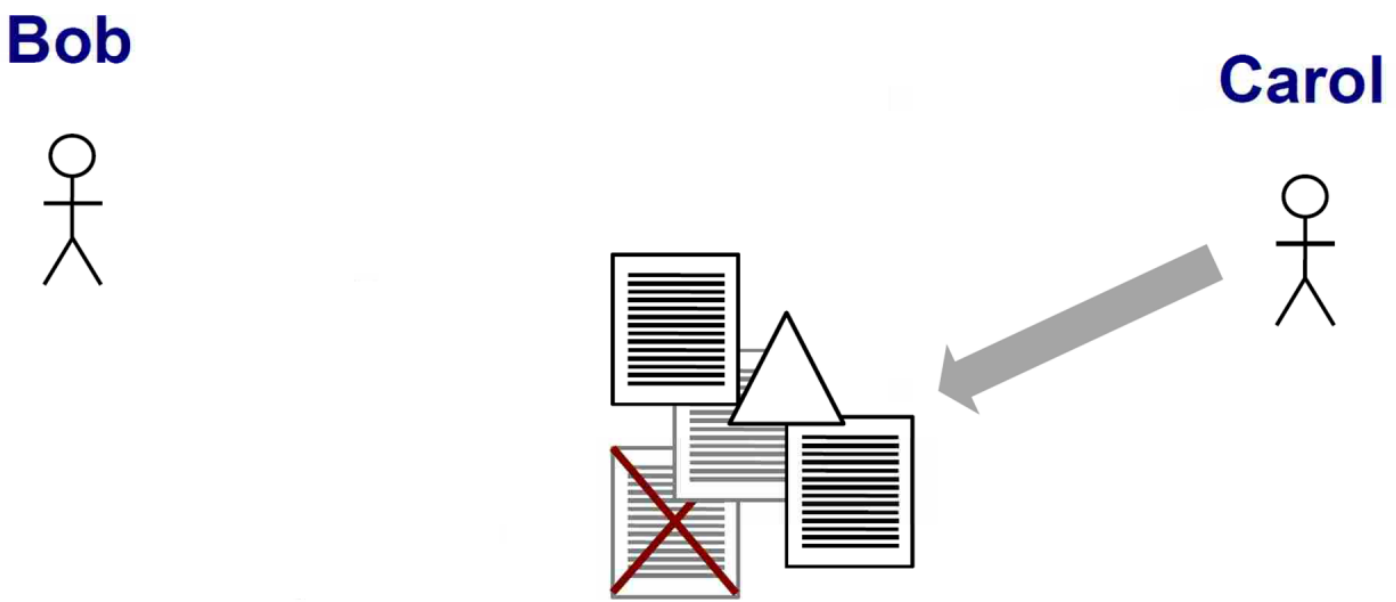
The collection of files and their complete history are stored in a repository



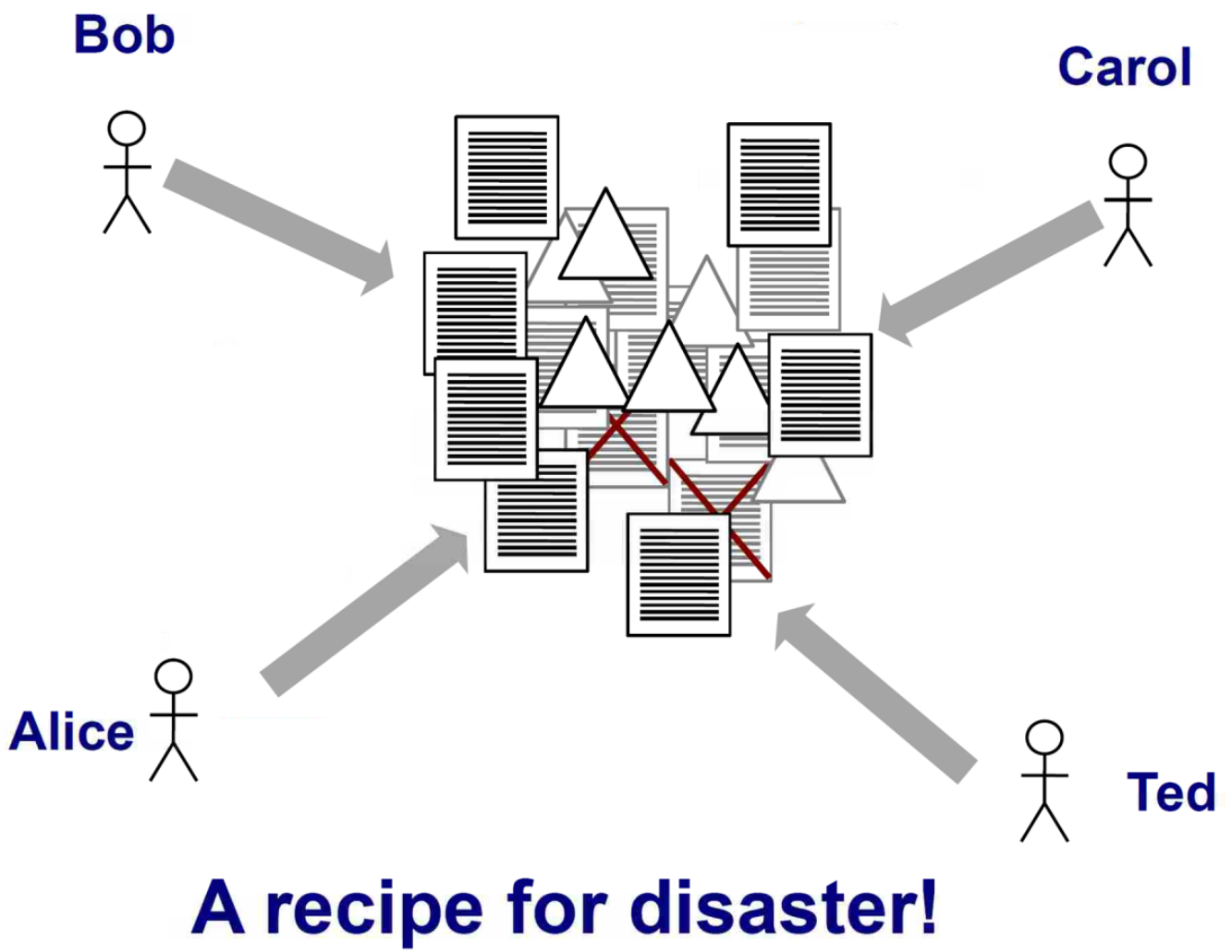
What is Version Control?



What is Version Control?



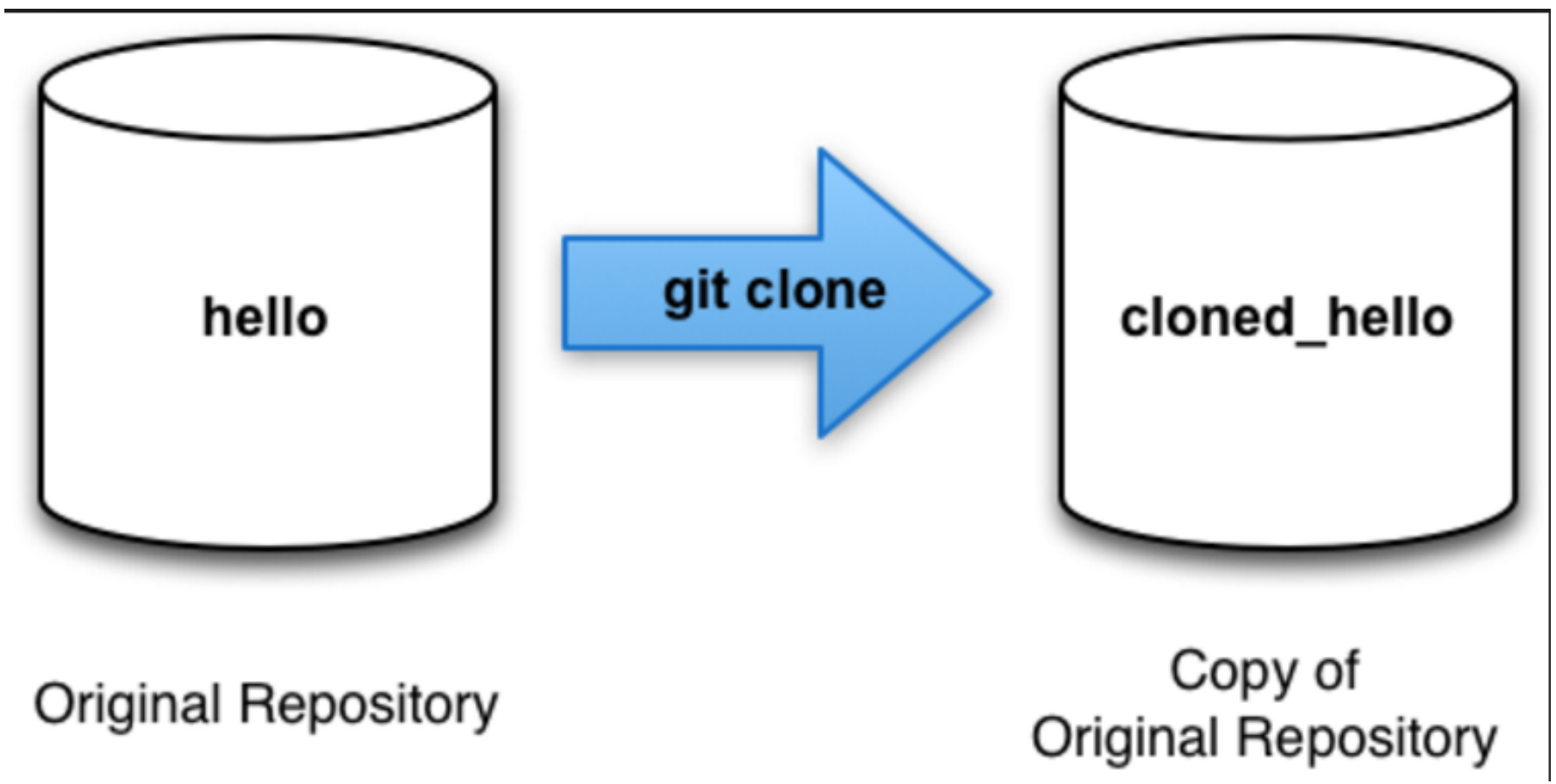
What is Version Control?



What is Version Control?

A distributed version control system does not have a central server which stores the data.

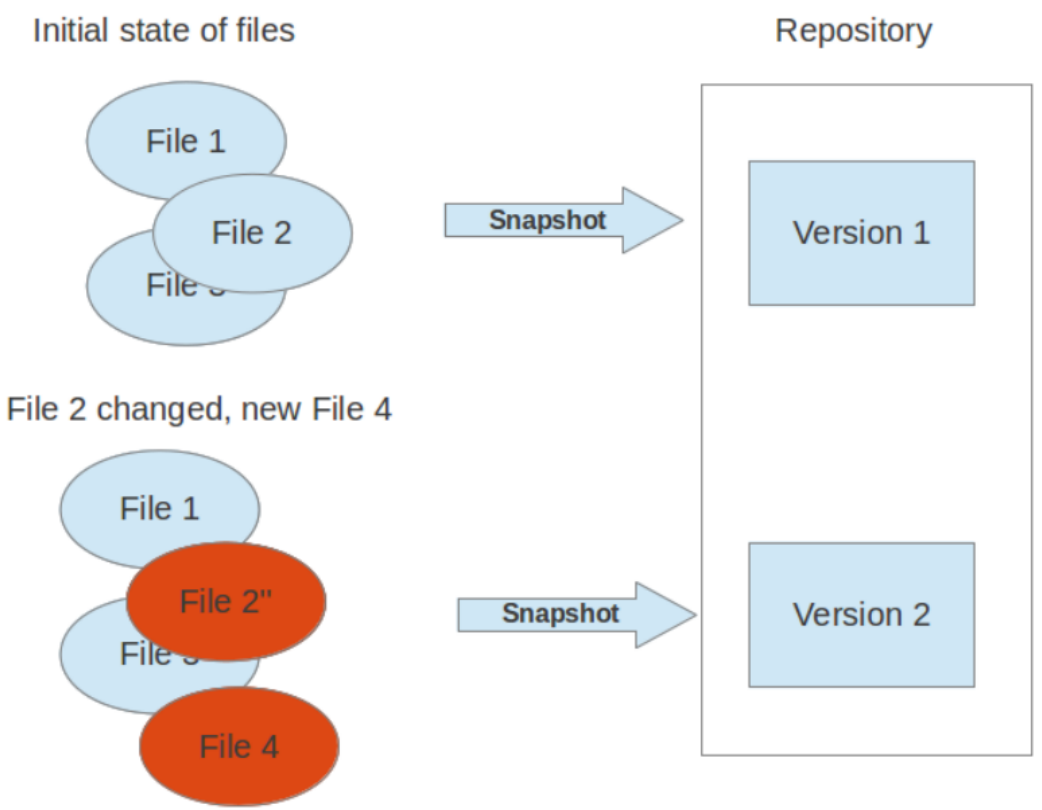
The user can copy an existing repository. This copying process is typically called cloning in a distributed version control system.



What is Version Control?

These snapshots can be used to change your collection of files. You may, for example, revert the collection of files to a state from 2 days ago.

Or you may switch between versions for experimental features.



Current leading Version control systems

The defacto standart

Resource	Options
Distributed VCS	Git (https://git-scm.com)
	Mercurial (https://mercurial.selenic.com)
	Bazaar (http://bazaar.canonical.com)
Online hosting site	GitHub (https://github.com)
	Bitbucket (https://bitbucket.org)
	GitLab (https://about.gitlab.com)
	Source Forge (http://sourceforge.net)
Git installation	https://git-scm.com/downloads
Git tutorials	Software Carpentry (https://swcarpentry.github.io/git-novice)
	Pro Git (https://git-scm.com/book)
	A Visual Git Reference (https://marklodato.github.io/visual-git-guide)
	tryGit (https://try.github.io)
Graphical User Interface for Git	https://git-scm.com/downloads/guis

doi:10.1371/journal.pcbi.1004668.t001

The defacto standart

What is Git?

“Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.”

http://git-scm.com/

Git the defacto standart

- The most popular implementation of version control today
- Everything is stored in local repositories on your computer
- Operated from the command line

http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git

Git the defacto standart

- developed by Junio Hamano and Linus Torvalds in 2005.
- Focus on speed and efficiency
- Quite a unique design and therefore sometimes a bit scary and difficult to understand

Github

GitHub is a web-based hosting service for Git repositories which allows you to create a remote copy of your local version-controlled project.

This can be used as a backup or archive of your project or make it accessible to you and to your colleagues so you can work collaboratively.

Who uses Git…

Some of the companies that use git:

Git in Academia

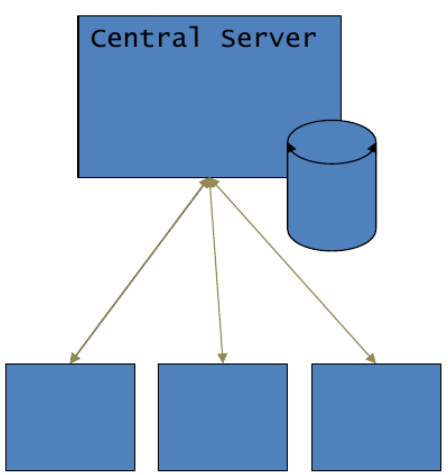
Scientists and academic researchers heavily use Git as well

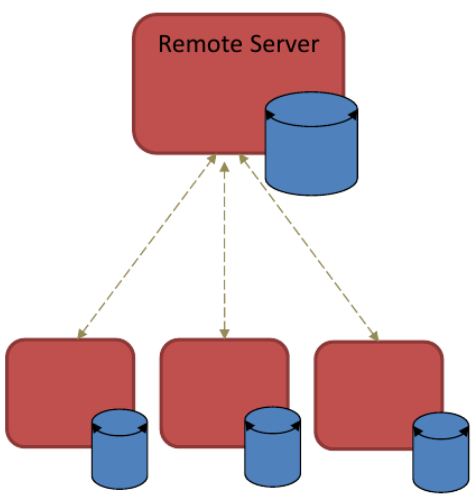
- Benefits of VC and collaboration tools aside, Git(Hub) helps to operationalise the ideals of open science and reproducibility.
- Journals have increasingly strict requirements regarding reproducibility and data access. GH makes this easy (DOI integration, off-the-shelf licenses, etc.)
- Hosting labs code,data and papers on github

Distributed:

- Everyone has the complete history
- Everything is done offline
- No central authority
- Changes can be shared without a server

Centralized VC vs. Distributed VC





Can I only use this for code?

Version control works for any file format (code, pdfs, docs, xls, jpg, etc.)

…But works best for text-based files (txt, Python, R, Matlab, any other code)

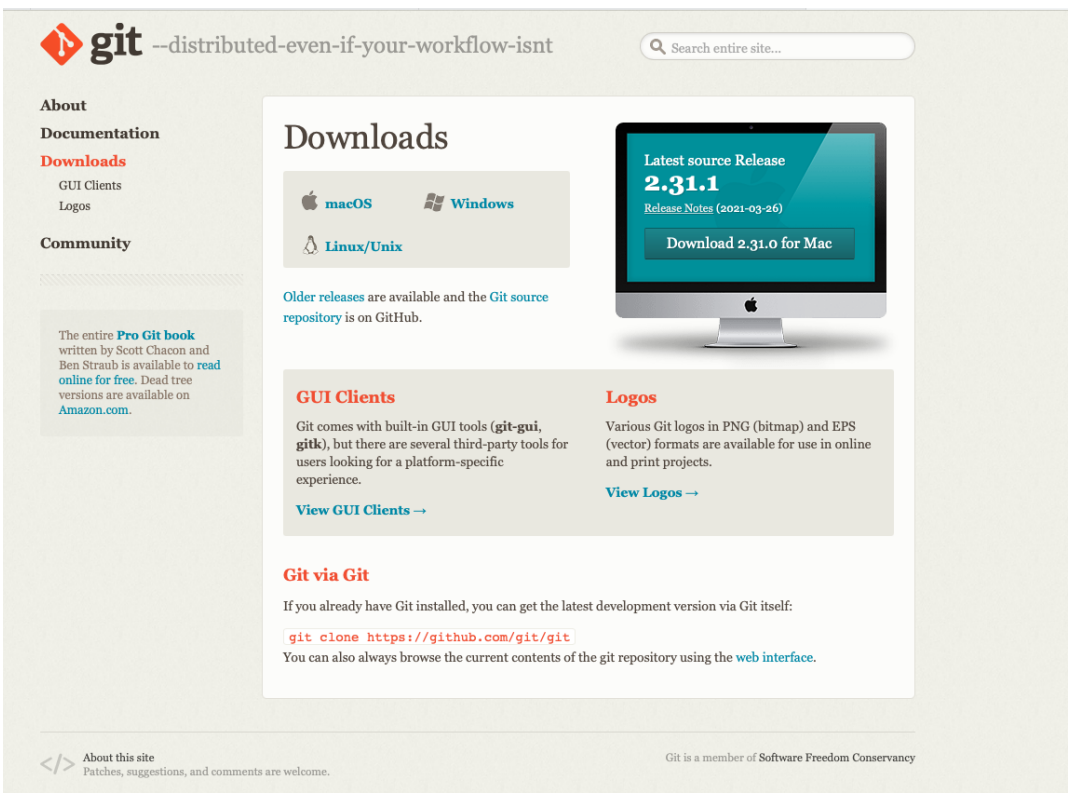
Setting up git

Windows: Download and install git

See previous lesson on installing git-bash

- Go to the following website and click on the Windows download link:

http://git-scm.com/downloads

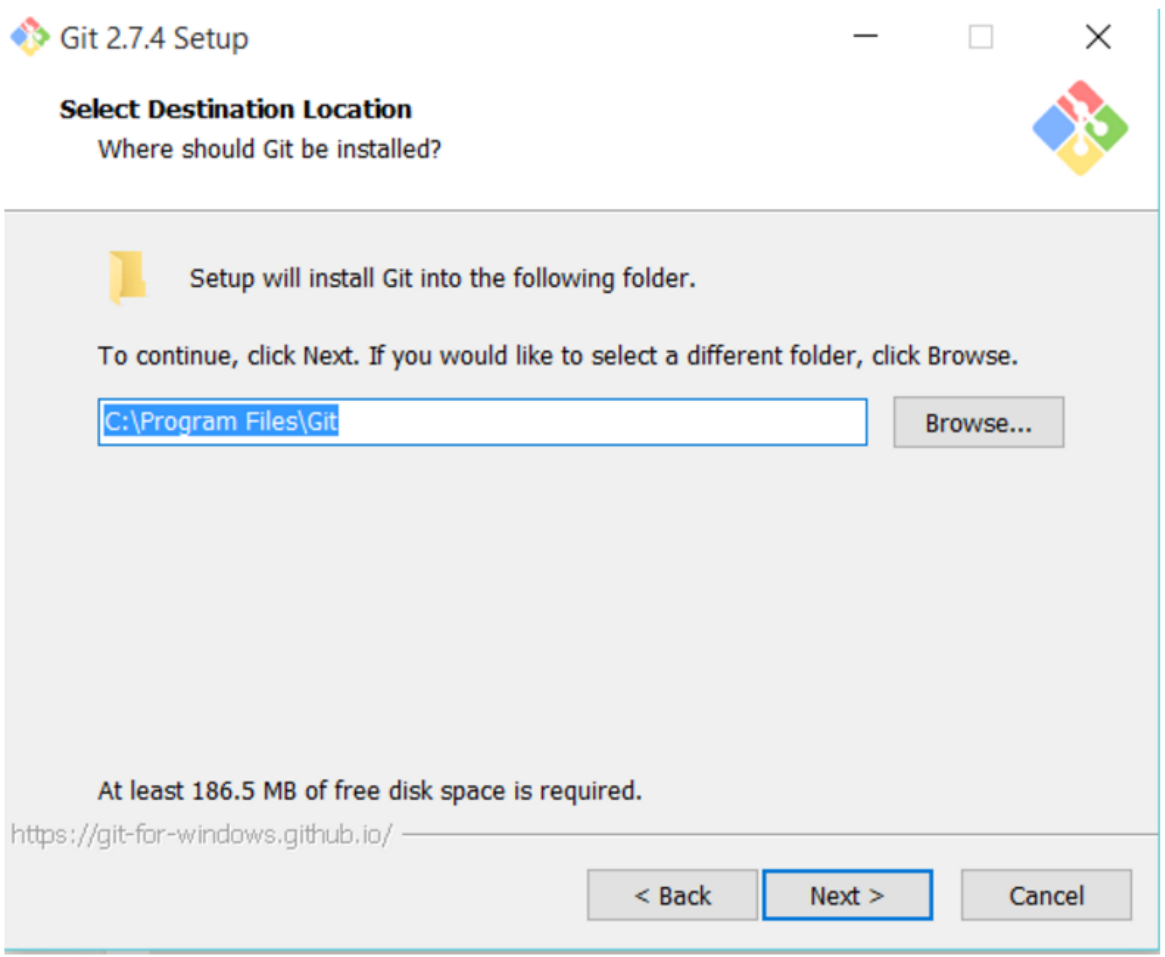


Download and install git

- Once the file is done downloading, open it up to begin the Git installation

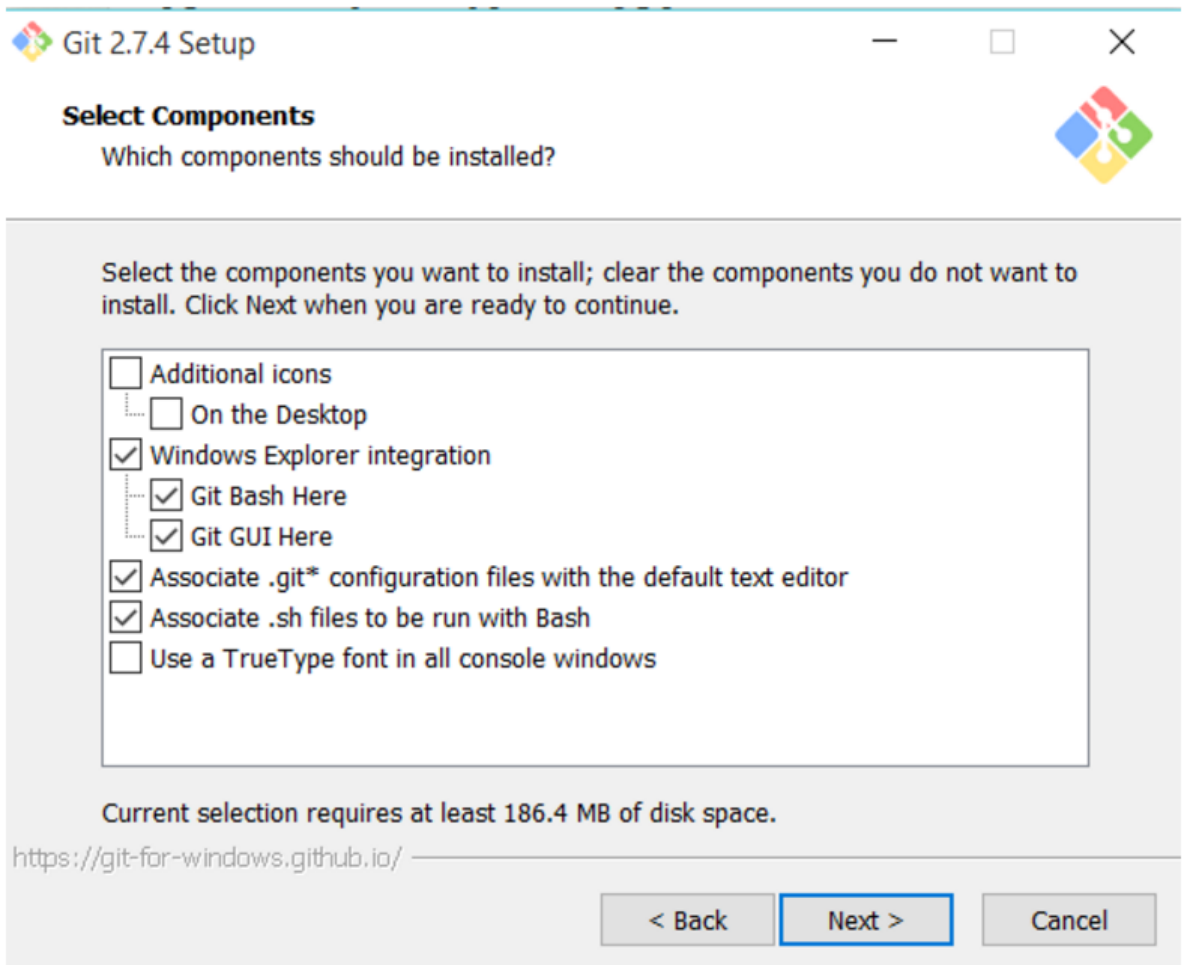
Download and install git

Unless you really know what you are doing, just go with the default options at each step of the installation



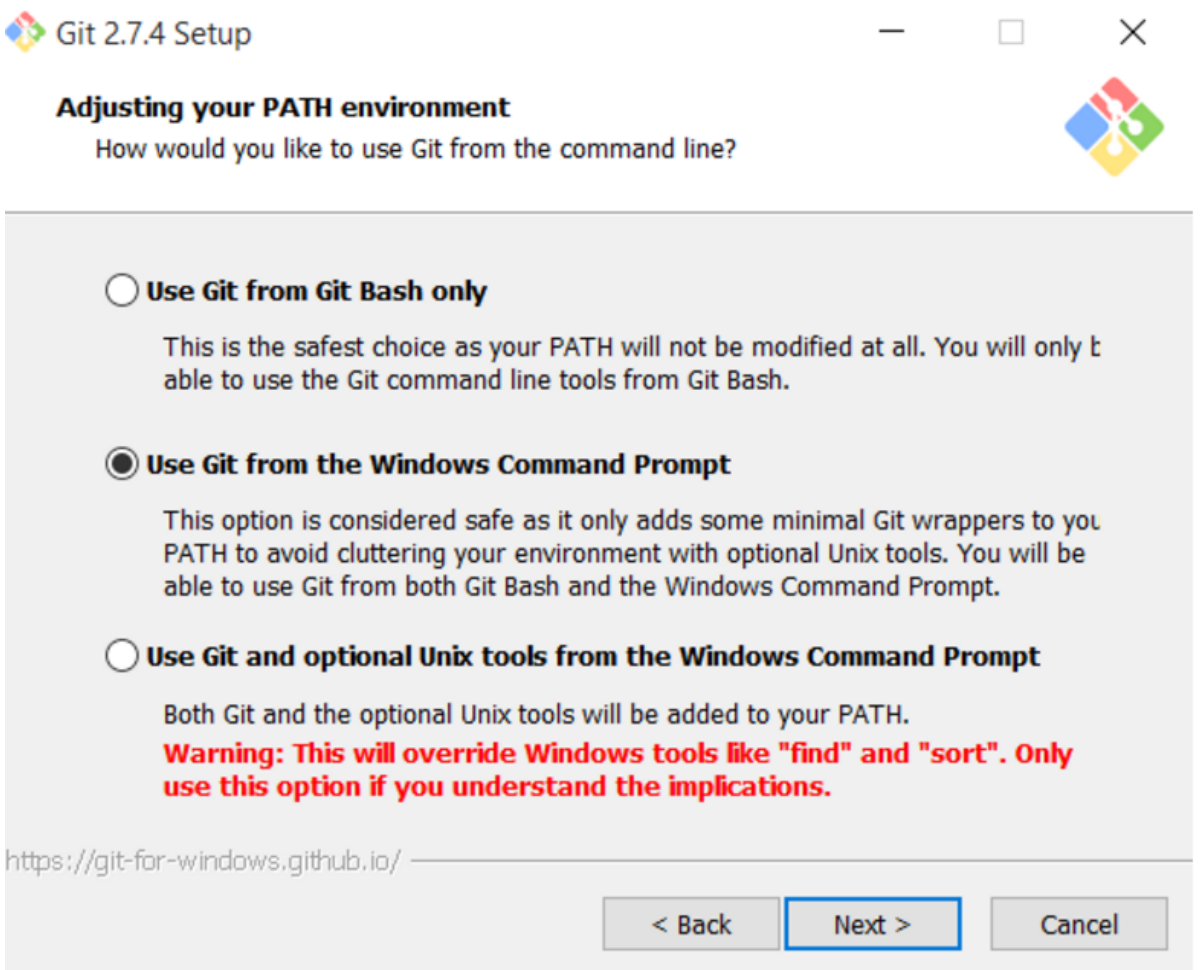
Download and install git

Select the components you want to install. Ensure that at least the boxes shown in the screenshot below have been checked. Click Next.



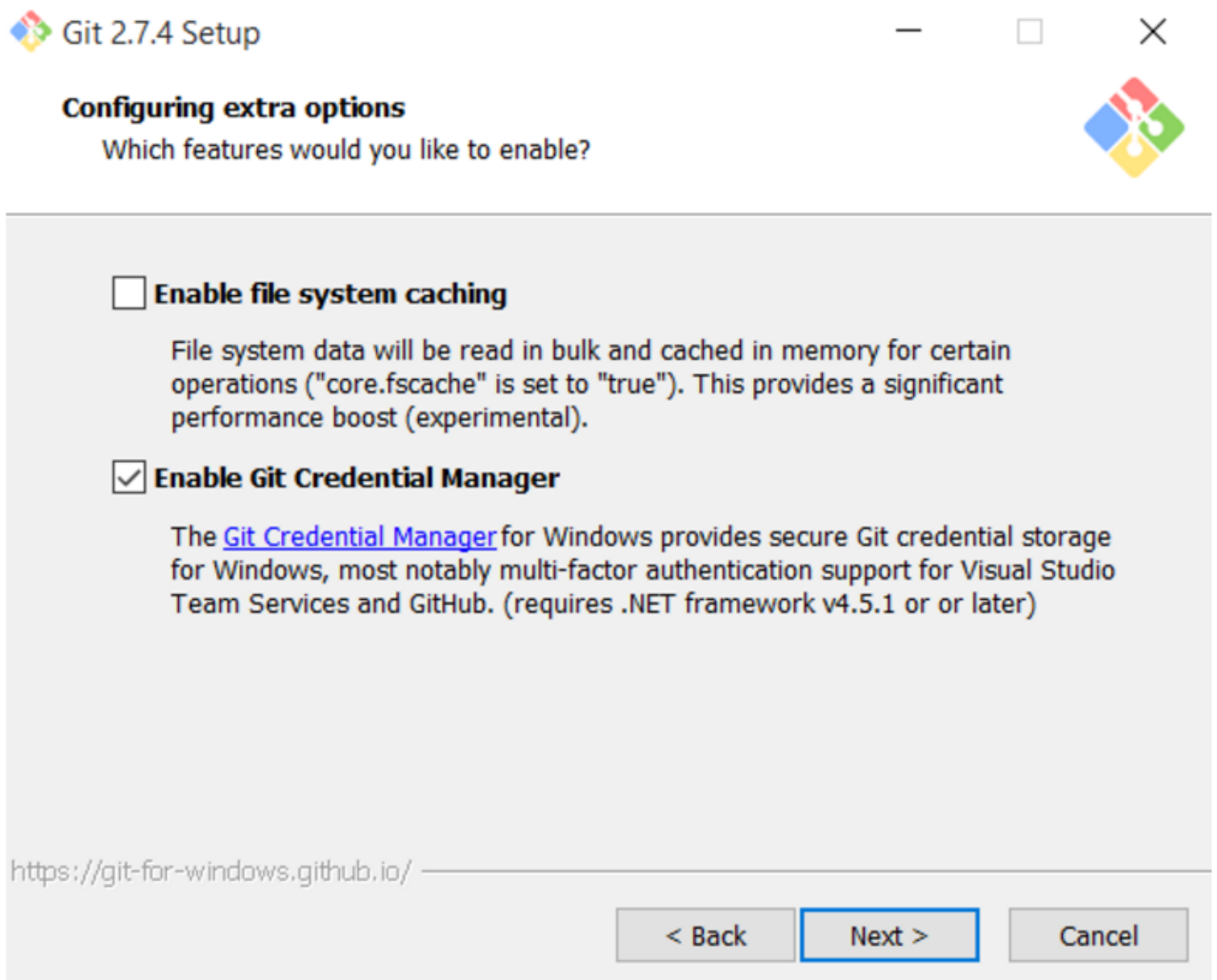
Download and install git

Git can be used from the command line also. Selecting the second option allows you this flexibility for when you become familiar with Git



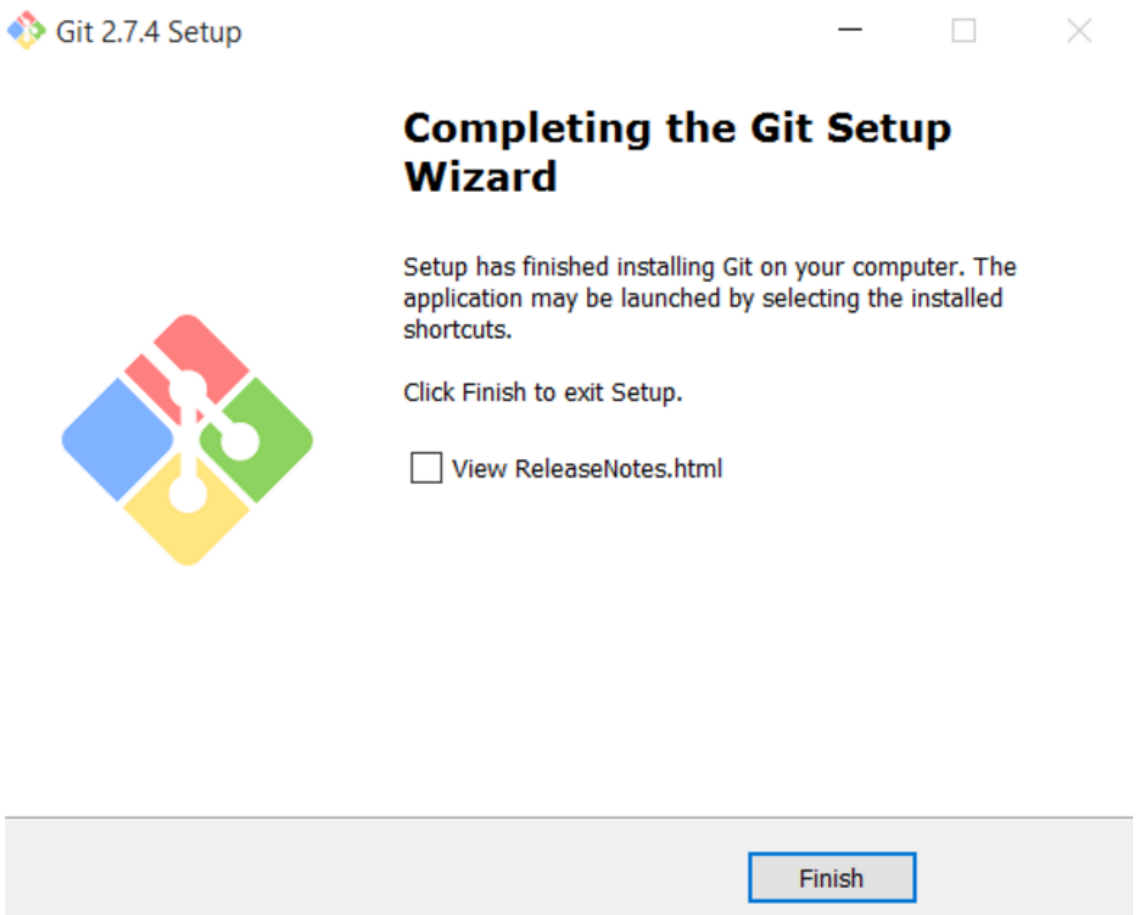
Download and install git

Ensure that at least the Enable Git Credential Manager box is checked



Download and install git

- Once the install is complete, hit the “Finish” button (you may want to uncheck the box next to “Review ReleaseNotes.rtf”)



Mac

If using a Mac computer download Git from Git - Downloads

and proceed to install in the usual way (double click on the installer package once downloaded).

> If you've previously installed Xcode on your Mac and want to use a more up to date version of Git then you will need to follow a few more steps documented here. If you've never heard of Xcode then don't worry about it!

Linux

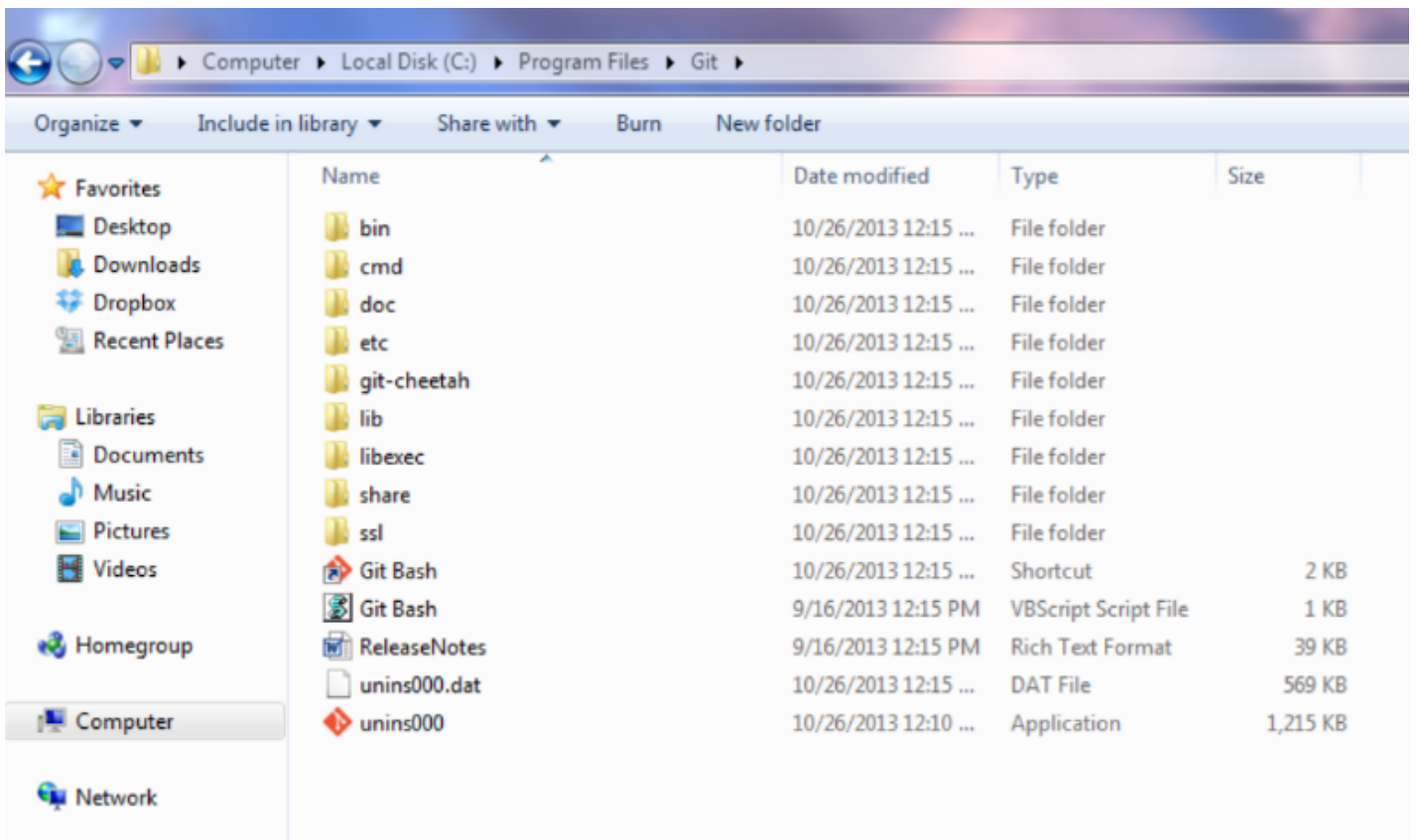
For those of you lucky enough to be working on a Linux machine you can simply use your OS package manager to install Git from the official repository.

For Ubuntu Linux (or variants of) open your Terminal and type

```
1 sudo apt update
2 sudo apt install git
```

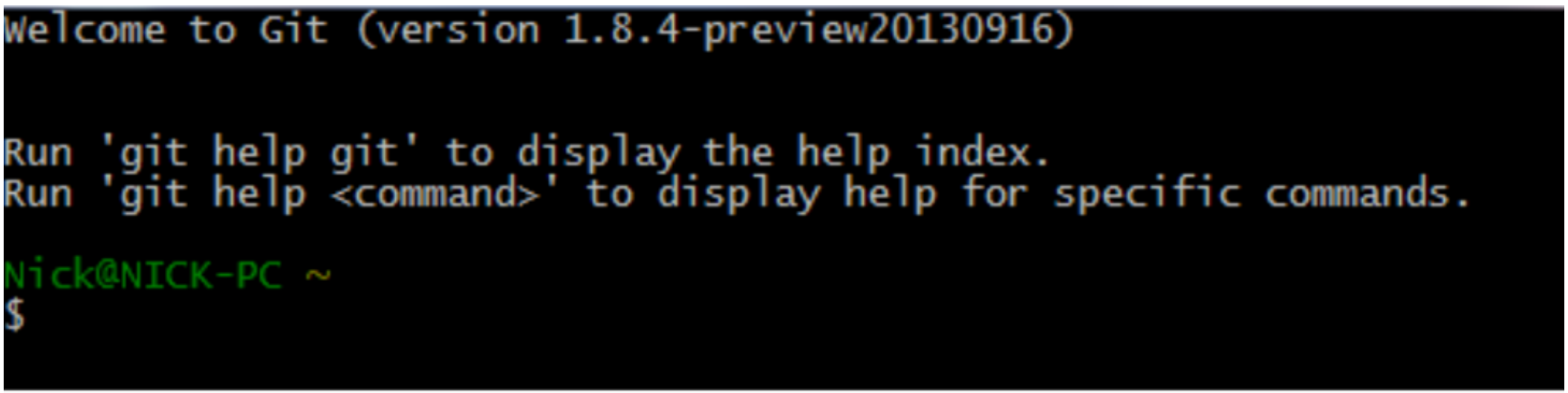
Open Git Bash

- Find a program called Git Bash, which is the command line environment for interacting with Git
- It should be located in the Git directory within your Start Menu (or in the directory into which Git was installed)



Open Git Bash

- Once Git Bash opens, you'll see a short welcome message followed by the name of your computer and a dollar sign on the next line
- The dollar sign means that it's your turn to type a command



Open Git Bash

- Each commit to a Git repository will be “tagged” with the username of the person who made the commit
- Enter the following commands in Git Bash, one at a time, to set your username and email:

```
$ git config --global user.name "Your Name Here"
$ git config --global user.email "your_email@example.com"
```

- Make sure there are 2 dashes side-by-side before the word “global”
- You’ll only have to do this once, but you can always change these down the road using the same commands

Open Git Bash

- Now type the following to confirm your changes (they may be listed toward the bottom):

```
$ git config --list
```

✅ **TIP** : Make sure there are 2 separate dashes (side-by-side) before the word “global”

Open Git Bash

- Go ahead and close Git Bash with the following command:

```
1       $ exit
```

- Now that Git is set up on your computer, we're ready to move on to GitHub, which is a web-based platform that lets you do some pretty cool stuff

Configure RStudio

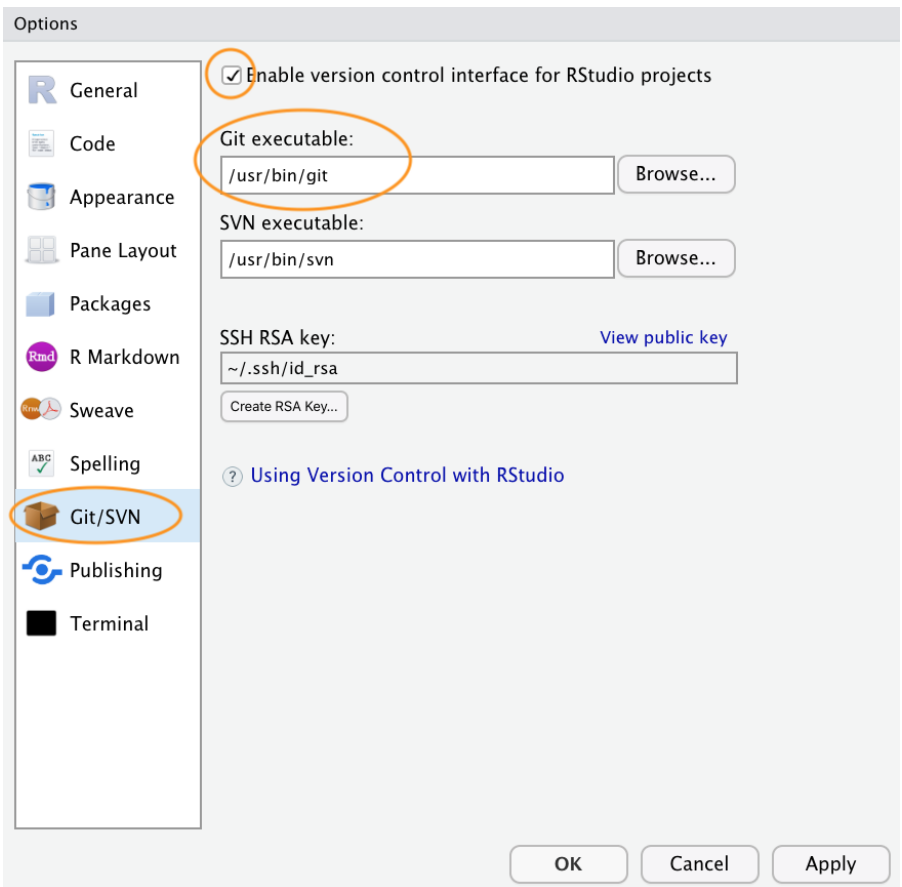
If you want to use RStudio's Git integration you need to check that the path to the Git executable is specified correctly. I

n RStudio, go to the menu `Tools -> Global Options -> Git/SVN`and make sure that 'Enable version control interface for RStudio projects' is ticked and that the 'Git executable:' path is correct for your installation.

Configure RStudio

If it's not correct hit the Browse… button and navigate to where you installed git and click on the executable file.

You will need to restart RStudio after doing this.



Rstudio and Git

One of the (many) great features of RStudio is how well it integrates version control into your everyday workflow.

Even though Git is a completely separate program to R, they feel like part of the same “thing” in RStudio.

Starting with Git

create a new git repo (git init)

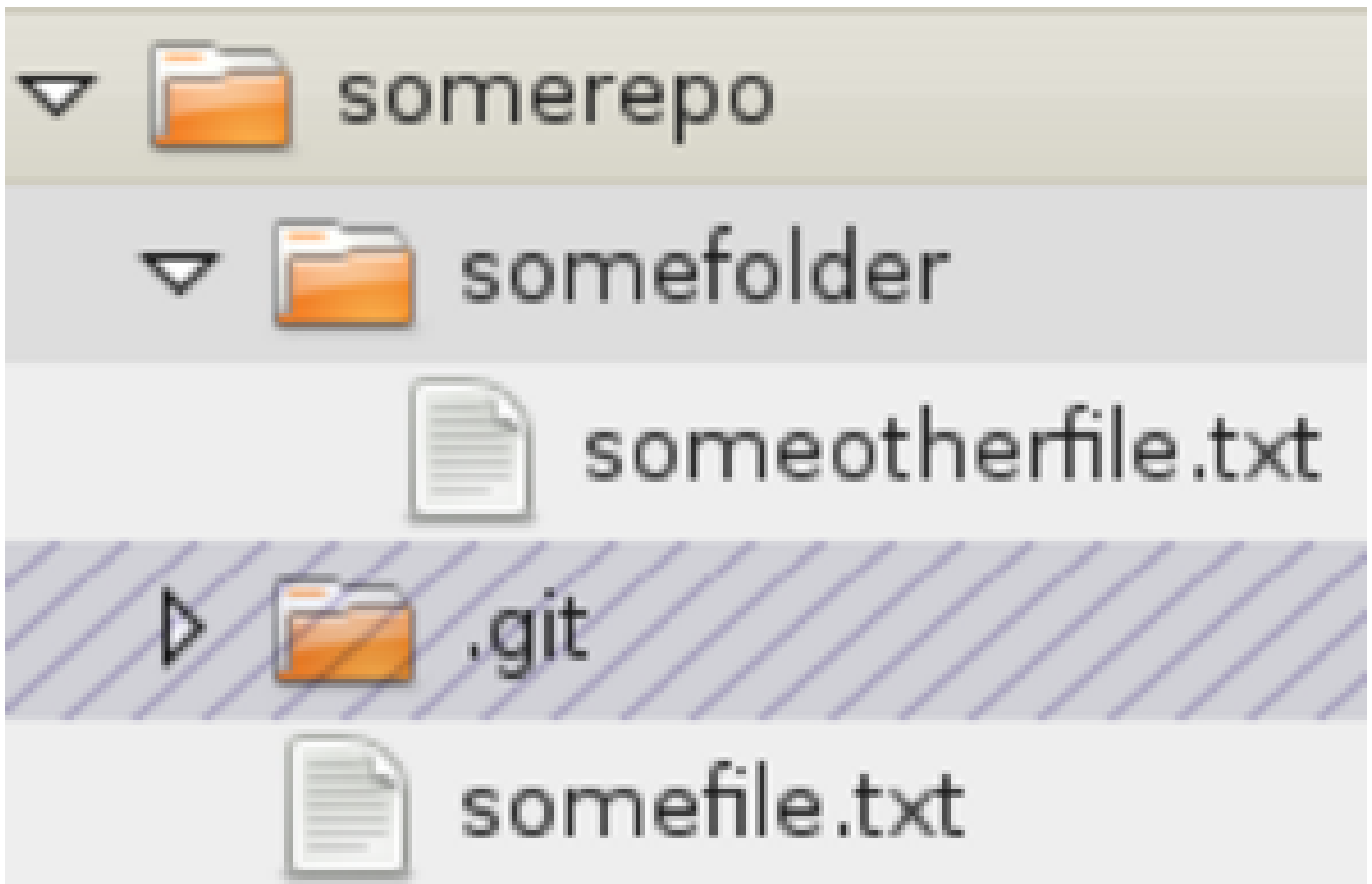
Open a Rstudio terminal (or use your UNIX terminal of choice)

then to create a new git repository, create a directory and enter it and then issue:

```
mkdir ~/test
cd ~/test
git init
```

create a new git repo (git init)

The whole repository content is stored in a folder named .git in the root of the project folder:



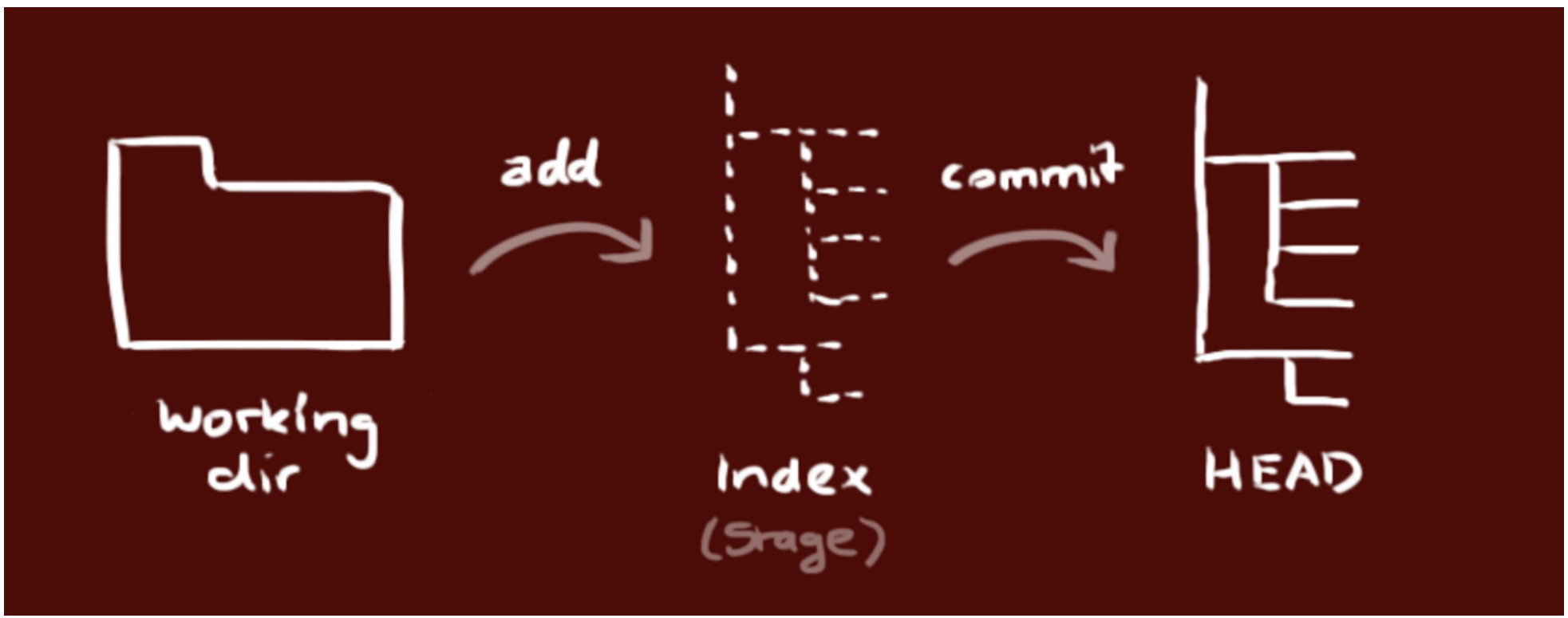
workflow

your local repository consists of three “trees” maintained by git.

the first one is your Working Directory which holds the actual files.

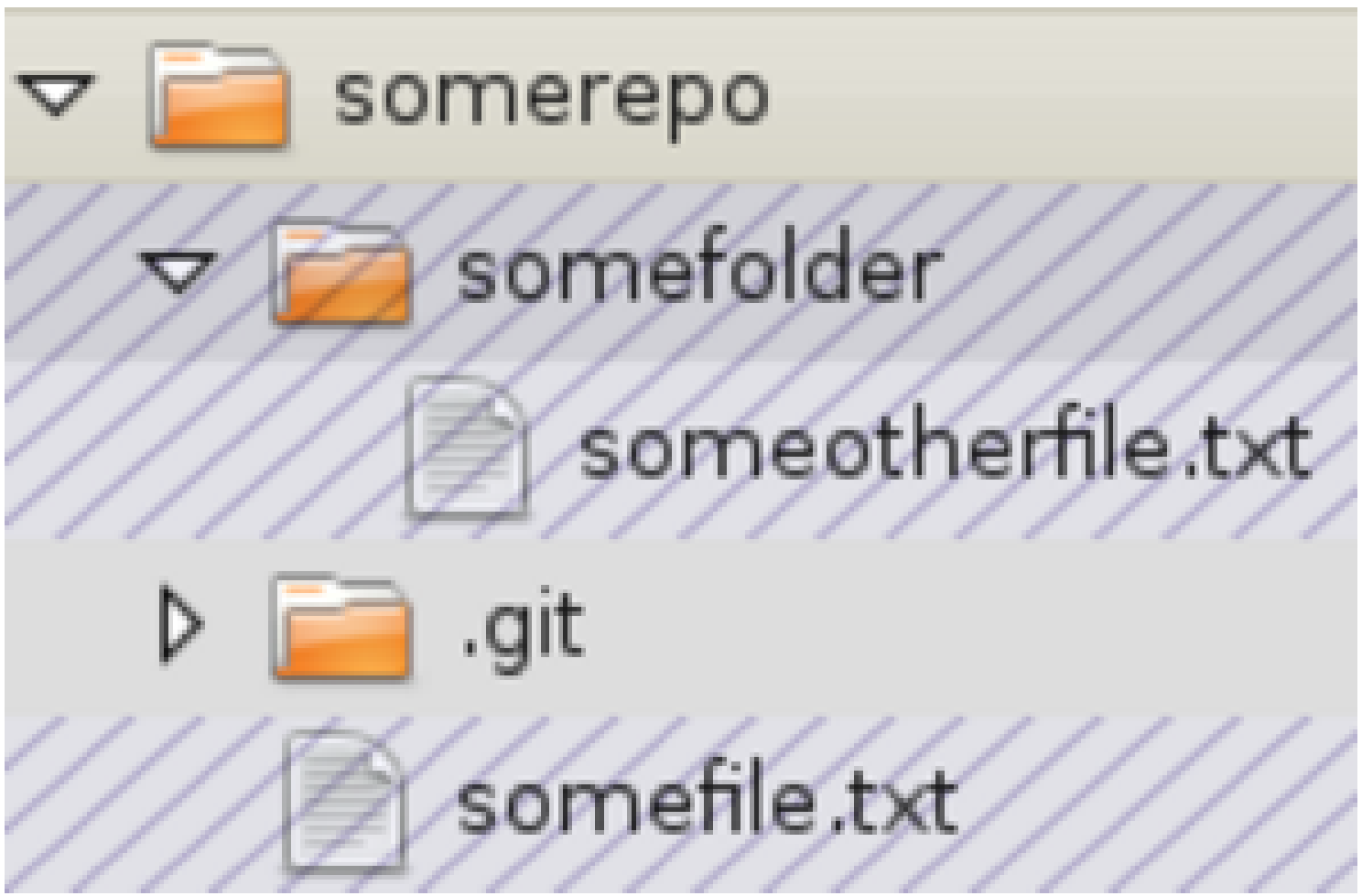
the second one is the Index which acts as a staging area

finally the HEAD which points to the last commit you’ve made.



working tree

Your own files in the repository folder are called working tree:



The staging index

Git internally holds a thing called the index, which is a snapshot of your project files.

After you have just created an empty repository, the index will be empty.

You must manually stage the files from your working tree to the index using git add:

```
1    git add somefile.txt
```

git add works recursively, so you can also add whole folders:

```
1    git add somefolder
```

Adding

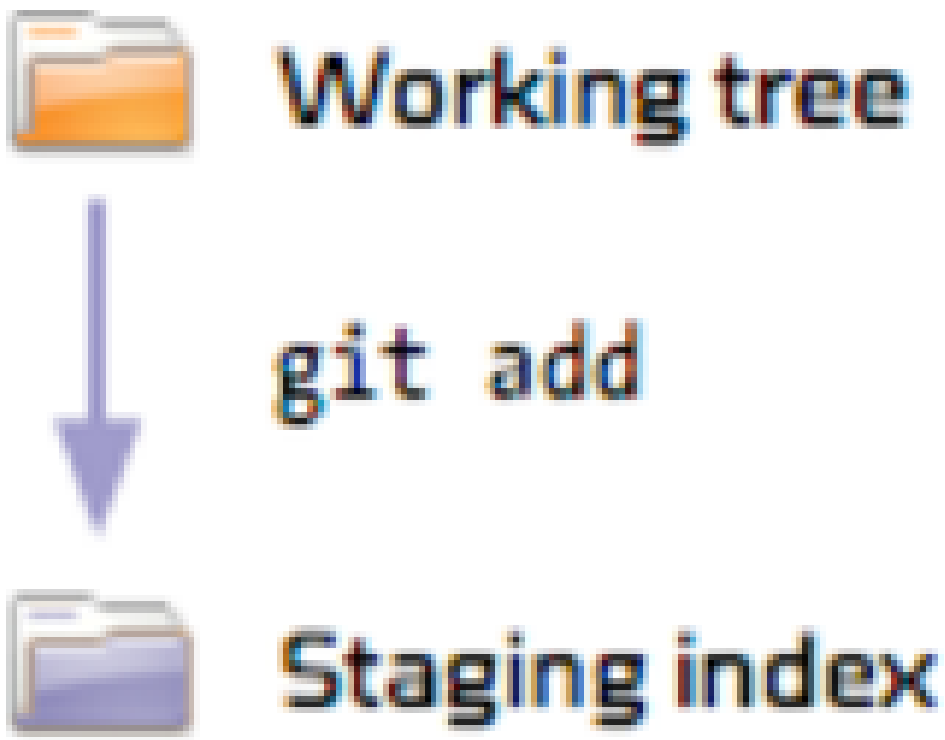
- You need to let Git know that you want it to pay attention to these files (i.e. “track” these files)
- From the directory where the repo is located on your computer (in Git Bash or Terminal, depending on whether you’re on Windows or Mac, respectively):
- `git add .` adds all new files (note the period after `add`, which represents “all files”)
- `git add -u` updates tracking for files that changed names or were deleted
- `git add -A` or `git add --all` does both of the previous

Adding

The same applies if you change a file in your working tree - you have to add this change to the index with git add:

```
edit somefile.txt
git add somefile.txt
```

Adding



It’s important to realize that the index is a full snapshot of your project files - it is not just a list of the changed files.

Committing


COMMENT
DATE
CREATED MAIN LOOP & TIMING CONTROL
14 HOURS AGO
ENABLED CONFIG FILE PARSING
9 HOURS AGO
MISC BUGFIXES
5 HOURS AGO
CODE ADDITIONS/EDITS
4 HOURS AGO
MORE CODE
4 HOURS AGO
HERE HAVE CODE
4 HOURS AGO
AAAAAAAA
3 HOURS AGO
ADKFJSLKDFJSDKLFJ
3 HOURS AGO
MY HANDS ARE TYPING WORDS
2 HOURS AGO
HAAAAAAAAANDS
2 HOURS AGO
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Committing

- You want to organize and save “snapshots” of the files you’ve staged for commit
- You type the command
- `git commit -m "your message goes here"`, substituting a useful description (between the double quotes) of what changes you made since the last committed changes
- This only updates your local repo, not the remote repo on GitHub

Committing

git commit takes the contents of the index and creates a new commit:

```
1       git commit -m "the 1st commit"
```

Committing

Committing is a completely local operation, not related to sending something to a remote server.

It just takes the contents of the index and keeps a snapshot of your project files as they were in the index:

Committing

Similar to the index a commit is a full snapshot of your project files.

Different from traditional version control systems, commits are not numbered.

Instead, a commit gets assigned a SHA-1 hash of the snapshot contents:

```
1  ## commit
2  42e2e5af9d49de268cd1fda3587788da4ace418a
```

This may look awkward the first time you see it.

But it brings a huge advantage with it: every commit, which is a full snapshot of your project files, is identified by a cryptographically tamper-proof signature of your file contents.

Committing

If somehow one byte of the contents or history of your files changes, you would end up with an entirely different hash.

So you’re guaranteed to get out what you put into a git repository.

Also, you don’t need to write the full commit hash when you want to refer to some specific commit - you can always abbreviate them by their first characters.

The first seven characters are usually enough to identify one commit uniquely.

Commit history

The workflow for editing files in a git repository looks like this:

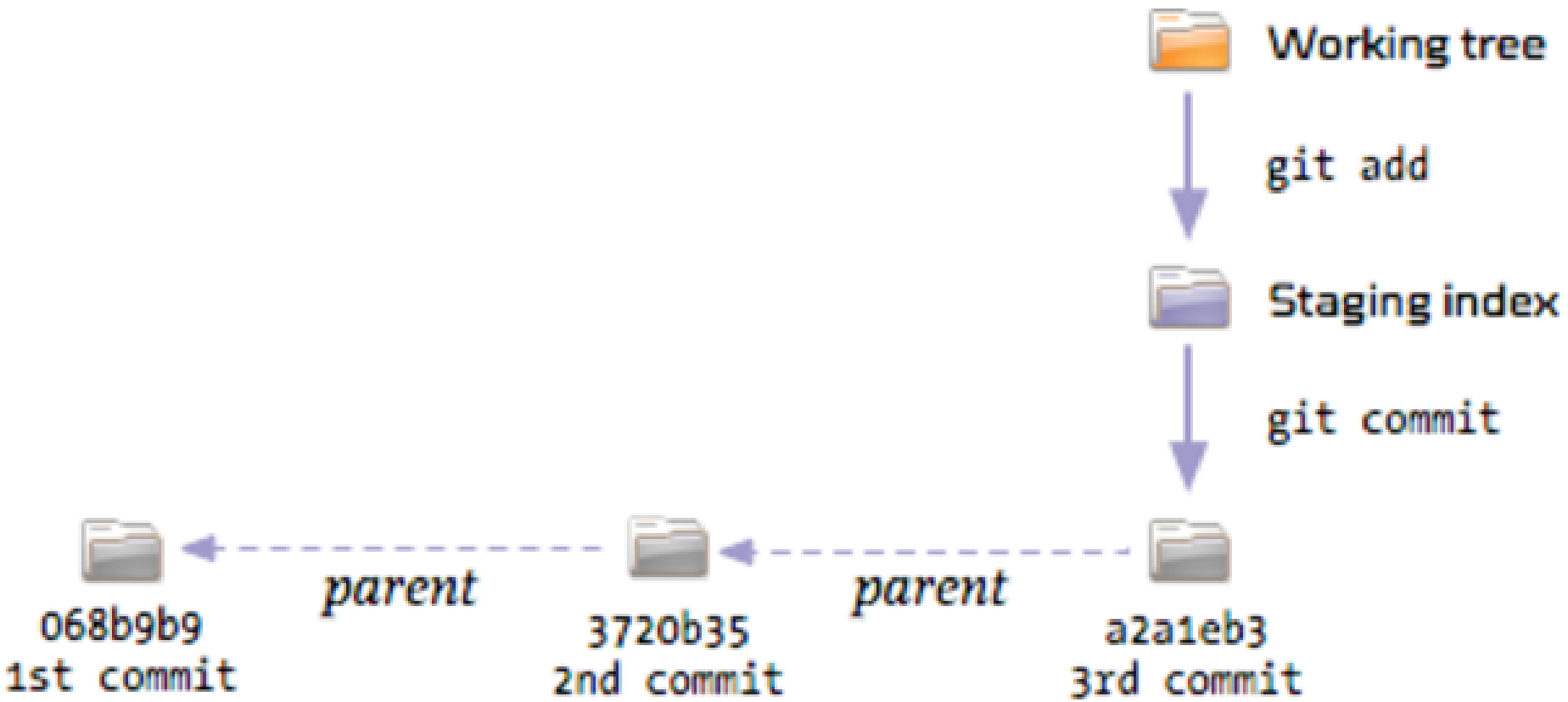
You make changes to the working tree files.

You add these changes to the index using git add.

You create a new commit from the index using git commit.

Commit history

As you do this repeatedly, you will create a new commit each time, pointing back to the previous commit:



Commit history

This is how git keeps track of the project history.

It stores snapshots of the project files as commits.

These commits point back to the commit they were created from.

Of course, all these snapshots are saved in a very efficient manner occupying only a fraction of disk space compared to a full copy of all your files.

Log

You can see the history using git log:

```
    git log

    commit 068b9b9...
    Author: Bob <bob@example.com>
    Date:   Wed Jun 17 17:21:16 2009 +0200

        the 3rd commit

    commit 3720b35...
    Author: Bob <bob@example.com>
    Date:   Wed Jun 17 17:21:10 2009 +0200

        the 2nd commit

    commit a2a1eb3...
    Author: Bob <bob@example.com>
    Date:   Wed Jun 17 17:21:10 2009 +0200

        the 1st commit
```

Log

in its simplest form, you can study repository history using.. git log

To see a log of the commits you've made locally, type

```
1    git log
```

- Spacebar advances page by page
- Return advances line by line
- Typing the letter "Q" exits the log

Log

You can add a lot of parameters to make the log look like what you want. To see only the commits of a certain author:

```
1    git log --author=bob
```

Log

To see a very compressed log where each commit is one line:

```
1    git log --pretty=oneline
```

Log

Or maybe you want to see an ASCII art tree of all the branches, decorated with the names of tags and branches:

```
1     git log --graph --oneline --decorate --all
```

Log

See only which files have changed:

```
1    git log --name-status
```

Log

These are just a few of the possible parameters you can use. For more, see

```
1    git log --help
```

Log

example for a short log

```
    git log --pretty=oneline --abbrev-commit

    068b9b9 the 3rd commit
    3720b35 the 2nd commit
    a2a1eb3 the 1st commit
```

git status

git status shows you how the working tree is different from the index and how the index is different from the last commit:

```
1       git status
```

git status

At first, you will see the changes that were already added to the index. This list represents what will be in your next commit:

```
1       ## Changes to be committed:
2       #
3       #   modified:   changed_file_added.txt
```

git status

Also, git will show the changes that have been made to the working tree, but were not yet added to the index:

```
    ## Changed but not updated:
    #
    #   modified:   changed_file.txt
    #
    ## Untracked files:
    #
    #   newfile.txt
```

Shortcut: How to add changed files when committing

When committing, you can leave the task of adding the changed files to git using the -a command line option:

```
> git commit -a -m "commit message"
```

This will add all changed (but not new) files to the index before committing.

Seeing the diff between commits

To see the difference from one commit compared to its parent, use git show :

```
1    git show 3720b35
```

Seeing the diff between commits

To compare two specific commits, use git diff <commit$_{from}$>..<commit$_{to}$>:

```
1    git diff a2a1eb3..068b9b9
```

Seeing the diff between commits

To see the diffs for the complete history, use git log -p

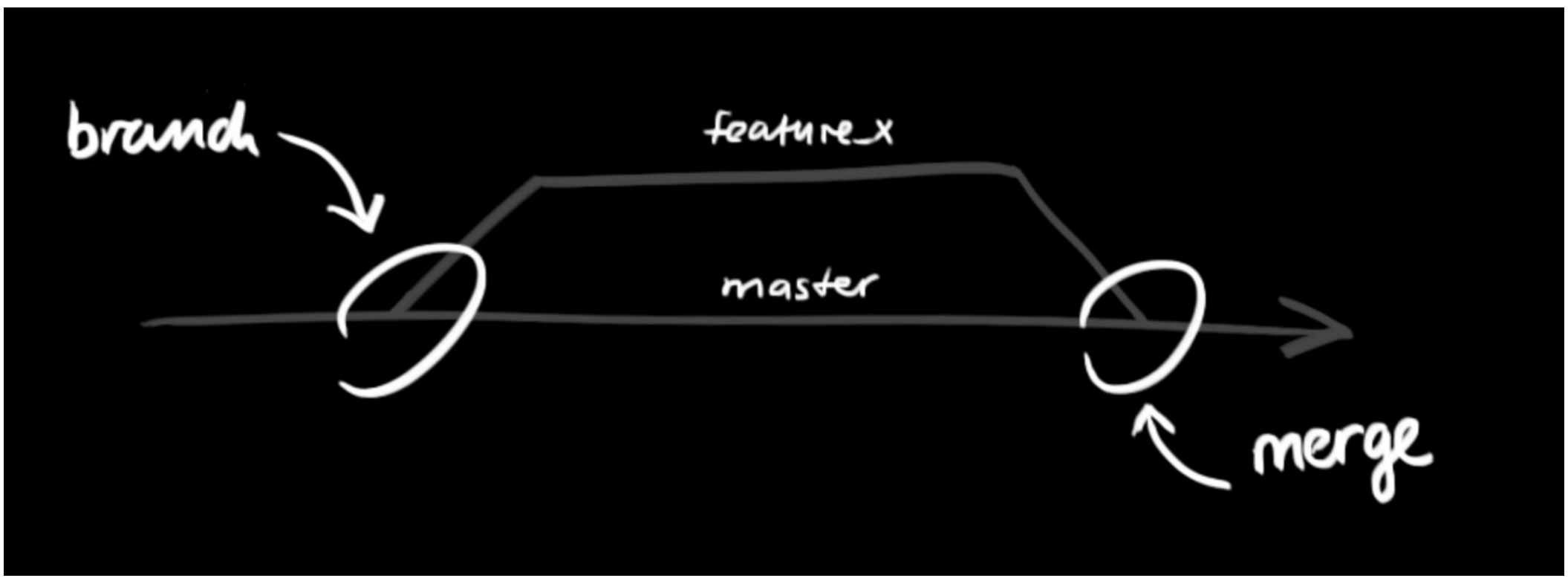
```
1    git log -p
```

Branching

Branches are used to develop features isolated from each other.

The master branch is the “default” branch when you create a repository.

Only once you (and your collaborators) are 100% satisfied, would you merge it back into the master branch



Branching

This is how most new features in modern software and apps are developed.

It is also how bugs are caught and fixed.

But researchers can easily use it to try out new ideas and analysis

If you aren’t happy, then you can just delete the experimental branch and continue as if nothing happened.

Branching

Example branch commands

```
# Create a new branch on your local machine and switch to it:
$ git checkout -b NAME-OF-YOUR-NEW-BRANCH
# Push the new branch to GitHub:
$ git push origin NAME-OF-YOUR-NEW-BRANCH
# List all branches on your local machine:
$ git branch
#Switch back to (e.g.) the master branch:
$ git checkout master
#Delete a branch
$ git branch -d NAME-OF-YOUR-FAILED-BRANCH
$ git push origin :NAME-OF-YOUR-FAILED-BRANCH
```

Merging branches

Commit your final changes to the new branch (say we call it “new-idea”) then

```
# Switch back to the master branch
$ git checkout master
#Merge in the new-idea branch changes:
$ git merge new-idea
#Delete the new-idea branch (optional):
$ git branch -d new-idea
```

tagging

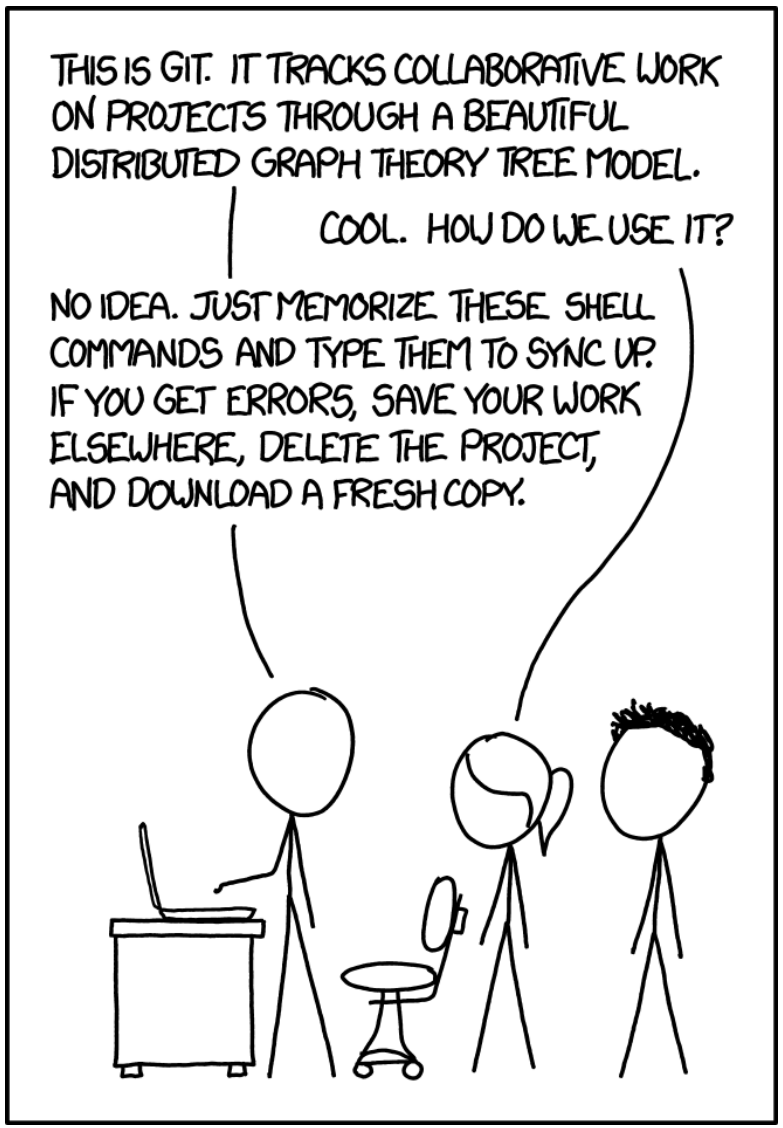
it’s recommended to create tags for software releases.

you can create a new tag named 1.0.0 by executing

```
1       git tag 1.0.0 1b2e1d63ff
```

the `1b2e1d63ff` stands for the first 10 characters of the commit id you want to reference with your tag. You can get the commit id by looking at the logs

Git base summary



Gitignore

Gitignore

From time to time, there are files you don’t want to follow via Git or check in to GitHub.

There are a few ways to tell Git which files to ignore

If you create a file in your repository named .gitignore, Git uses it to determine which files and directories to ignore, before you make a commit.

A .gitignore file should be committed into your repository, in order to share the ignore rules with any other users that clone the repository.

✅ **TIP** : Next lesson we will discuss gitignore in Rstudio

example content of Git Ignore File

```
## History files
.Rhistory
.Rapp.history
## Session Data files
.RData
## Example code in package build process
*-Ex.R
## Output files from R CMD build
/*.tar.gz
## RStudio files
.Rproj.user/
## produced vignettes
vignettes/*.html
vignettes/*.pdf
## Temporary files created by R markdown
*.utf8.md
*.knit.md
```

Create The File

In File manager/Terminal, navigate to the location of your Git repository.

create an empty text file (via a text editor or using touch CLI) and save it as .gitignore in your git repo root

If you already have a file checked in, and you want to ignore it, Git will not ignore the file if you add a rule later.

In those cases, you must untrack the file first, by running the following command in your terminal:

```
1       git rm --cached FILENAME
```

Create A Global .Gitignore

You can also create a global .gitignore file, which is a list of rules for ignoring files in every Git repository on your computer.

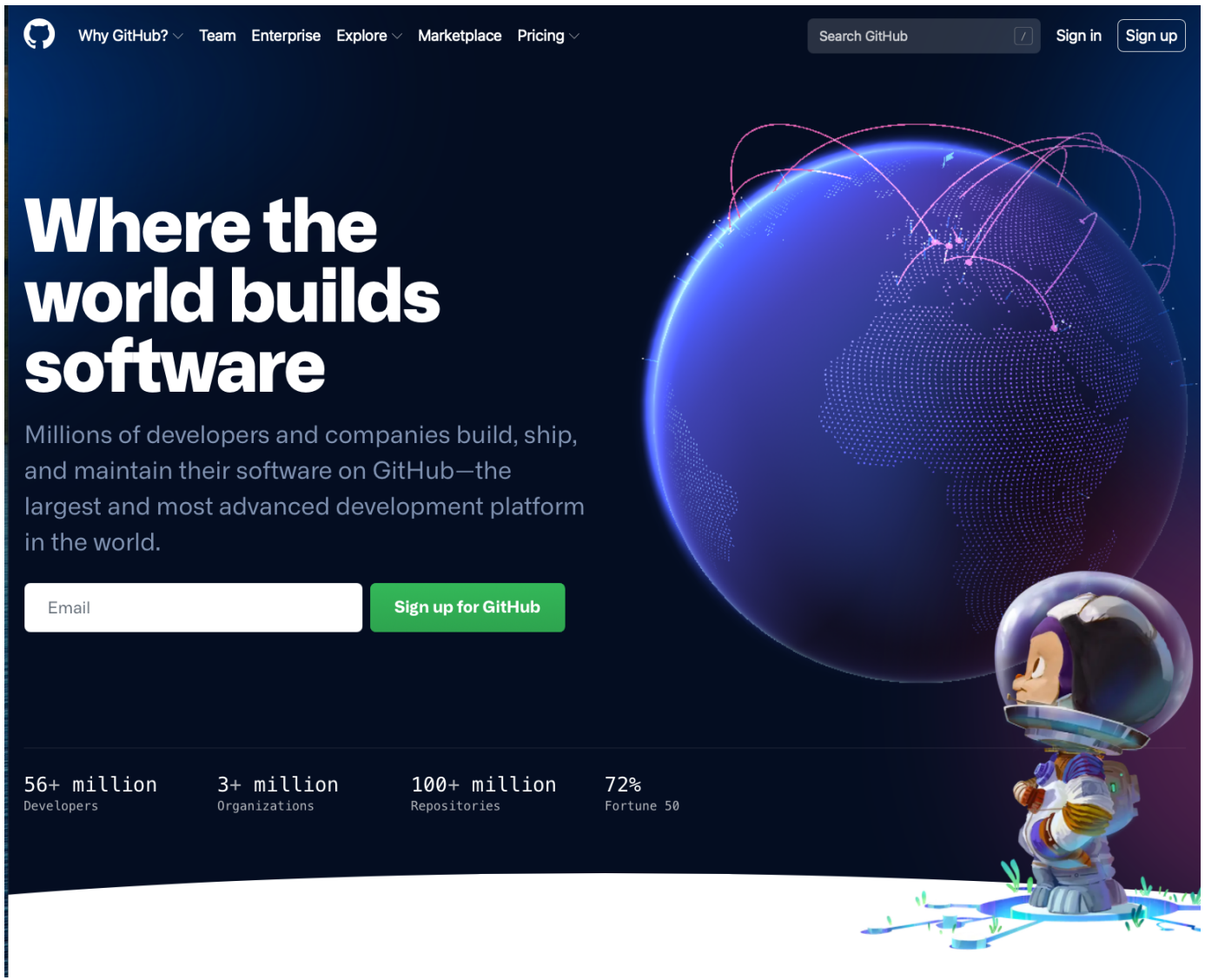
For example, you might create the file at ~/.gitignore$_{global}$ and add some rules to it.

Open Terminal. Run the following command in your terminal:

```
1    git config --global core.excludesfile ~/.gitignore_global
```

Github

Github


Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email
Sign up for GitHub
56+ million
Developers
3+ million
Organizations
100+ million
Repositories
72%
Fortune 50

What is GitHub?

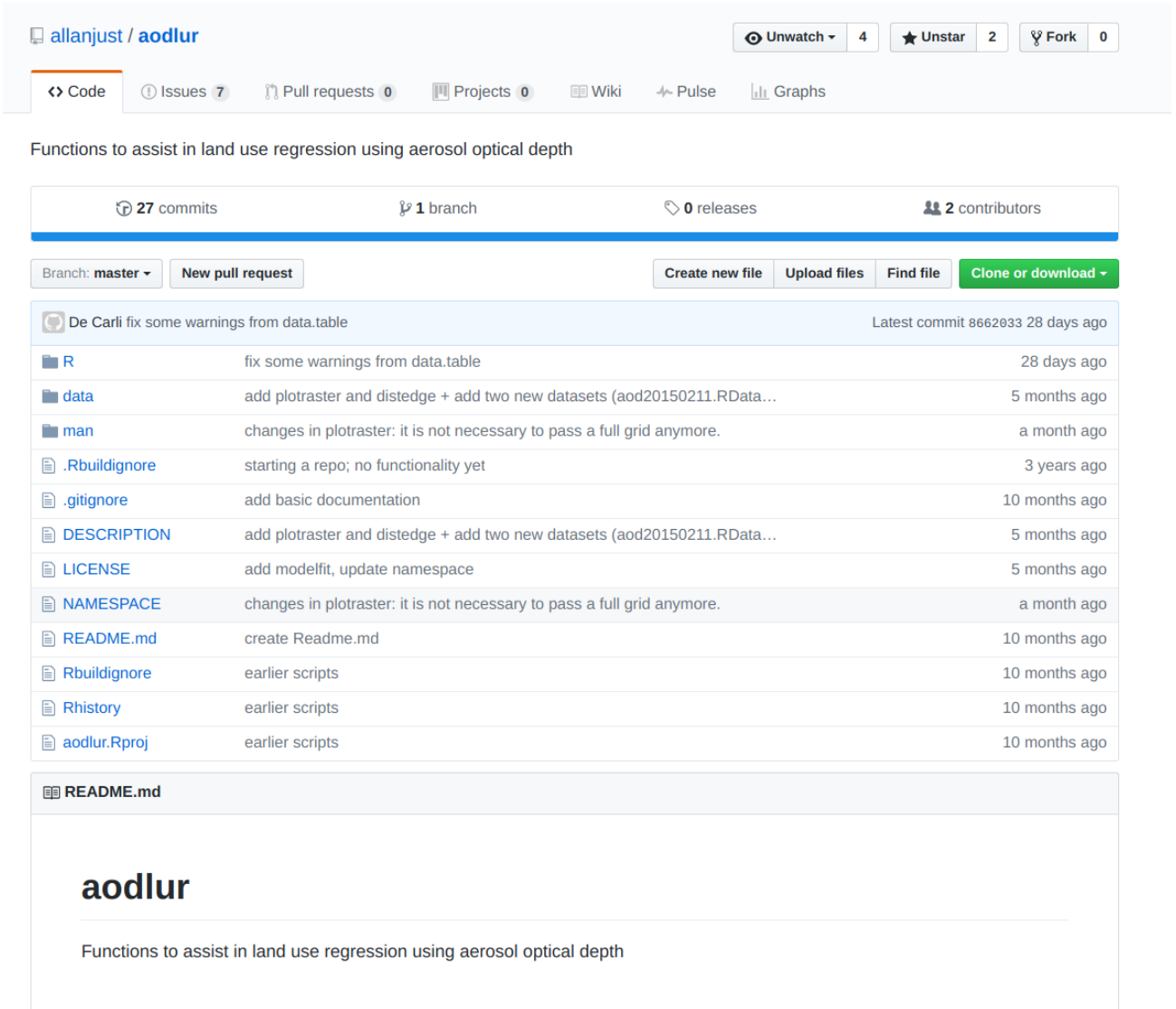
GitHub is a web-based hosting service for software development projects that use the Git revision control system.

http://en.wikipedia.org/wiki/GitHub

What is GitHub?

- Allows users to “push” and “pull” their local repositories to and from remote repositories on the web
- Provides users with a homepage that displays their public repositories
- Users’ repositories are backed up on the GitHub server in case something happens to the local copies
- Social aspect allows users to follow one another and share projects

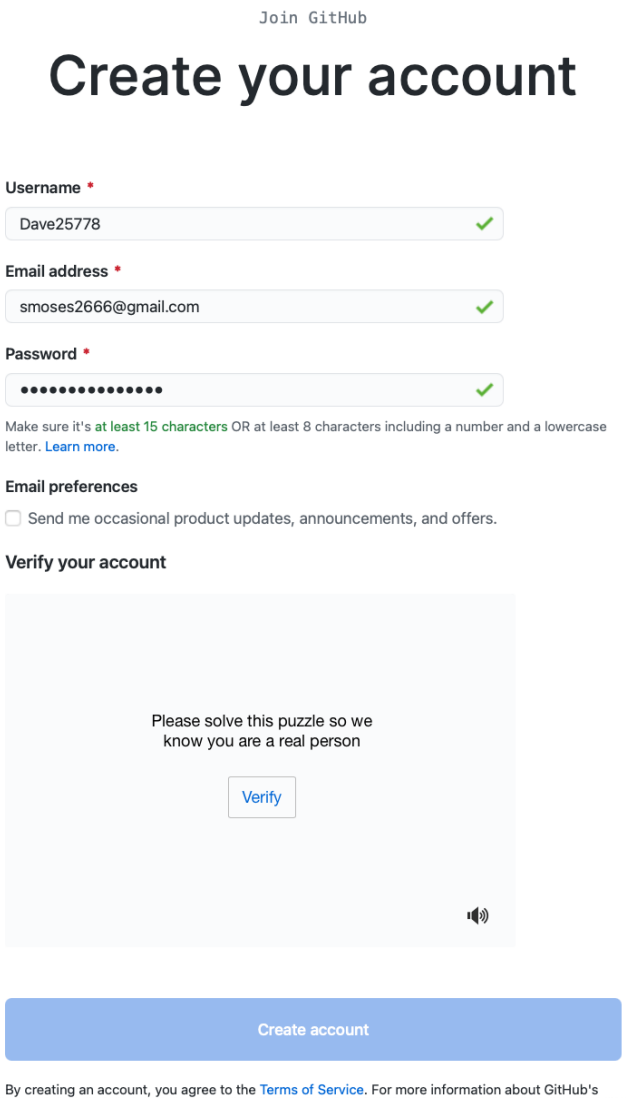
What is GitHub?



Set Up a GitHub Account

- Go to the GitHub homepage at https://github.com/
- Enter a username, email, and password and click “Sign up for GitHub”
- 🔥 **NOTE** 🔥: Use the same email address that you used when setting up Git in the git setup!!!

Set Up a GitHub Account



Set Up a GitHub Account

open an account and go through the process of sign up

Navigating GitHub

Your GitHub Profile

Your GitHub Profile

- Finally, if you click on “Edit Your Profile” in the top righthand portion of the screen you can add some basic information about yourself to your profile
- This is totally optional, but if you do good work, you ought to take some credit for it!
- In the next lecture, we’ll get you started by walking you through two ways of creating a repository
- In the meantime, feel free to explore the GitHub site for interesting projects that others are working on

Recap: Git vs. GitHub

- You don’t need GitHub to use Git
- Git = Local (on your computer); GitHub = Remote (on the web)
- GitHub allows you to:
  1. Share your repositories with others
  2. Access other users’ repositories
  3. Store remote copies of your repositories (on GitHub’s server) in case something happens to your local copies (on your computer)

Working with Git in Rstudio

Setting up a project in RStudio

Now that you're all set up, let's create your first version controlled RStudio project.

There are a couple of different approaches you can use to do this.

You can either setup a remote GitHub repository first then connect an RStudio project to this repository (we'll call this Option 1).

Another option is to setup a local repository first and then link a remote GitHub repository to this repository (Option 2).

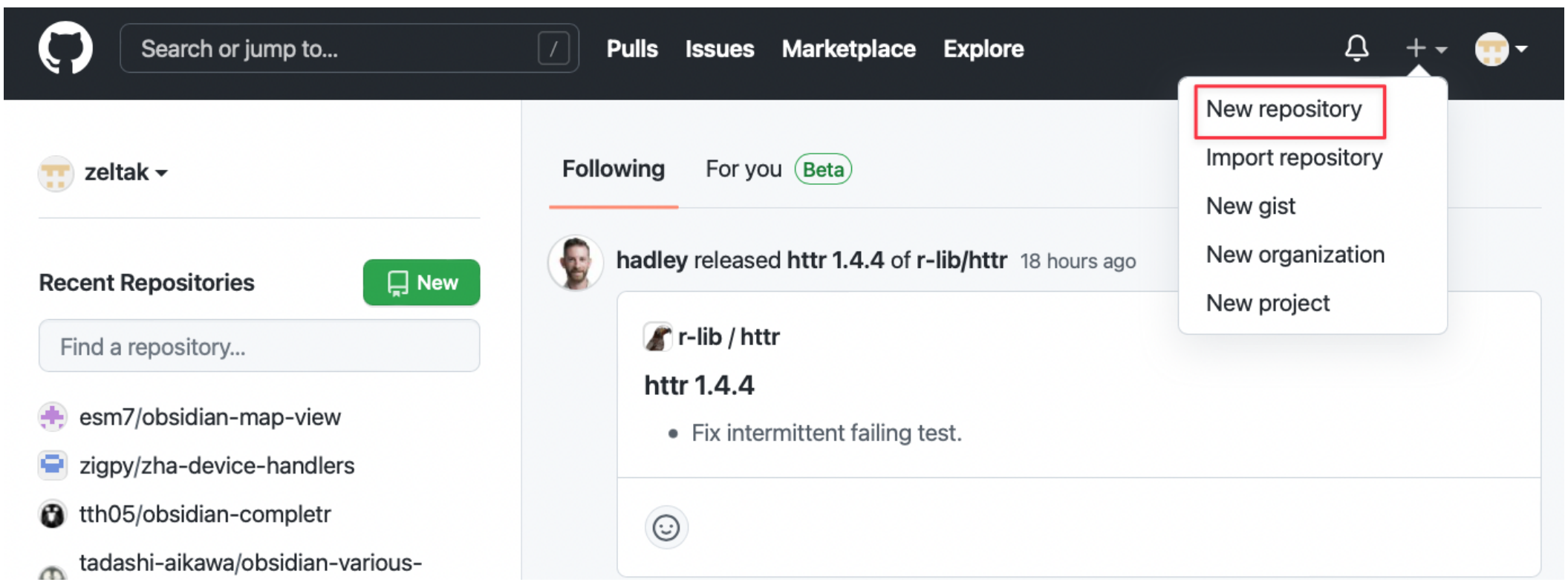
We will only cover option 1 in this course

Setting up git with Rstudio

To use the GitHub first approach you will first need to create a **repository (repo)** on GitHub.

Go to your GitHub page and sign in if necessary.

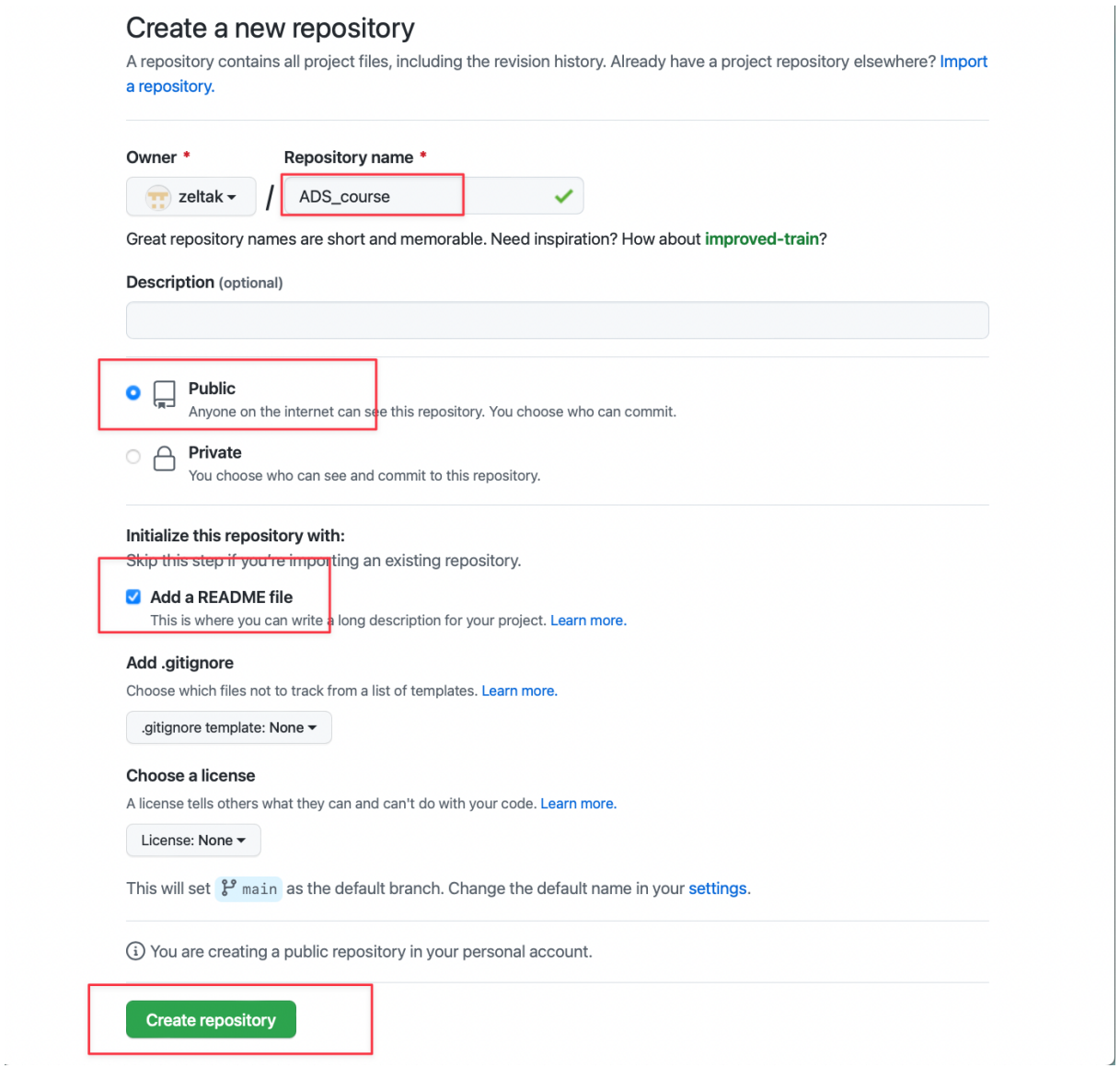
Click on the 'plus' sign at the top and then `new repository``



Setting up git with Rstudio

Give your new repo a name (let's call it `ADS_course`), select 'Public', tick on the 'Initialize this repository with a README' (this is important) and then click on 'Create repository' (ignore the other options for now).

Setting up git with Rstudio

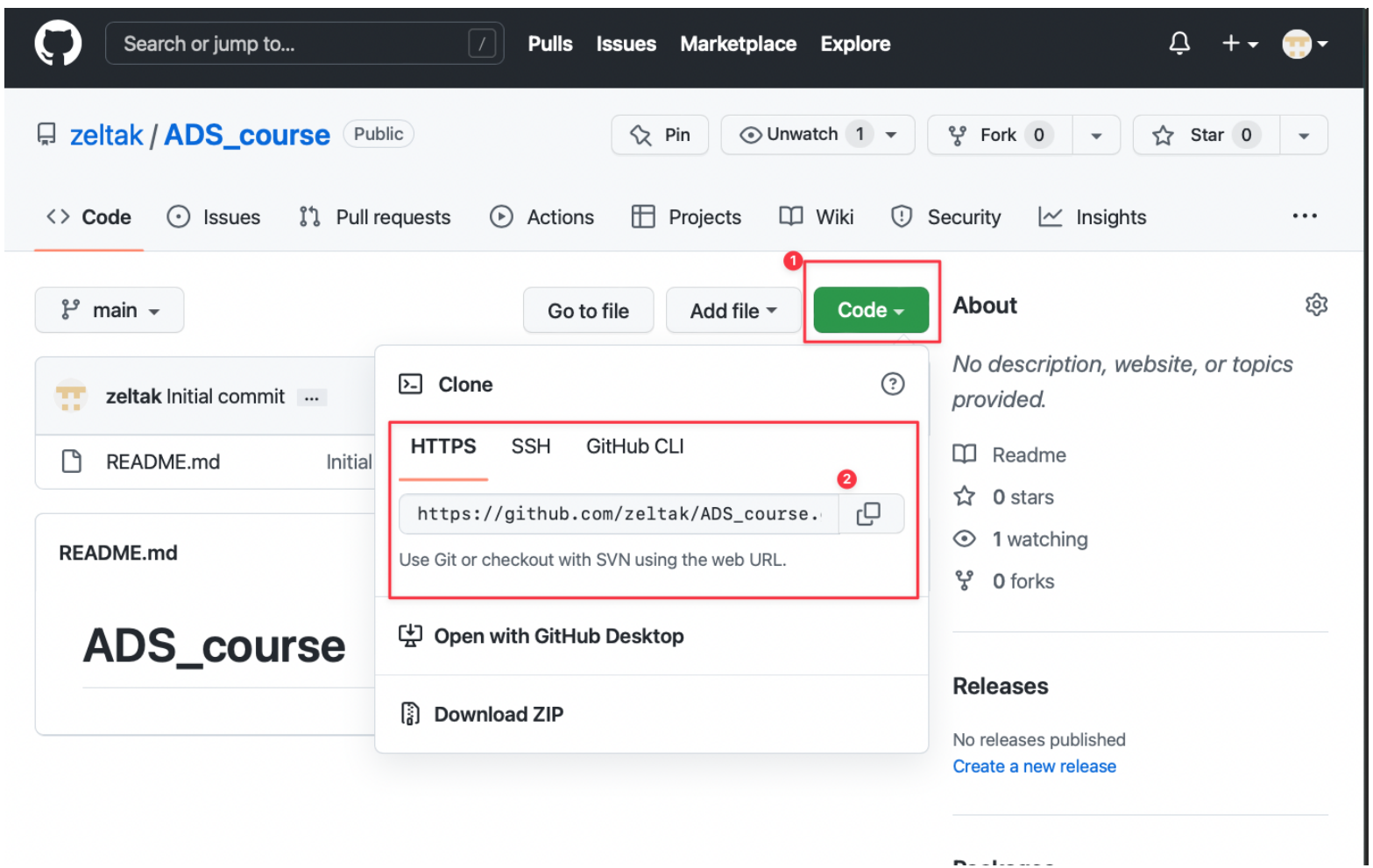


Setting up git with Rstudio

Your new GitHub repository will now be created. and the readme file as a markdown (.md) format.

Setting up git with Rstudio

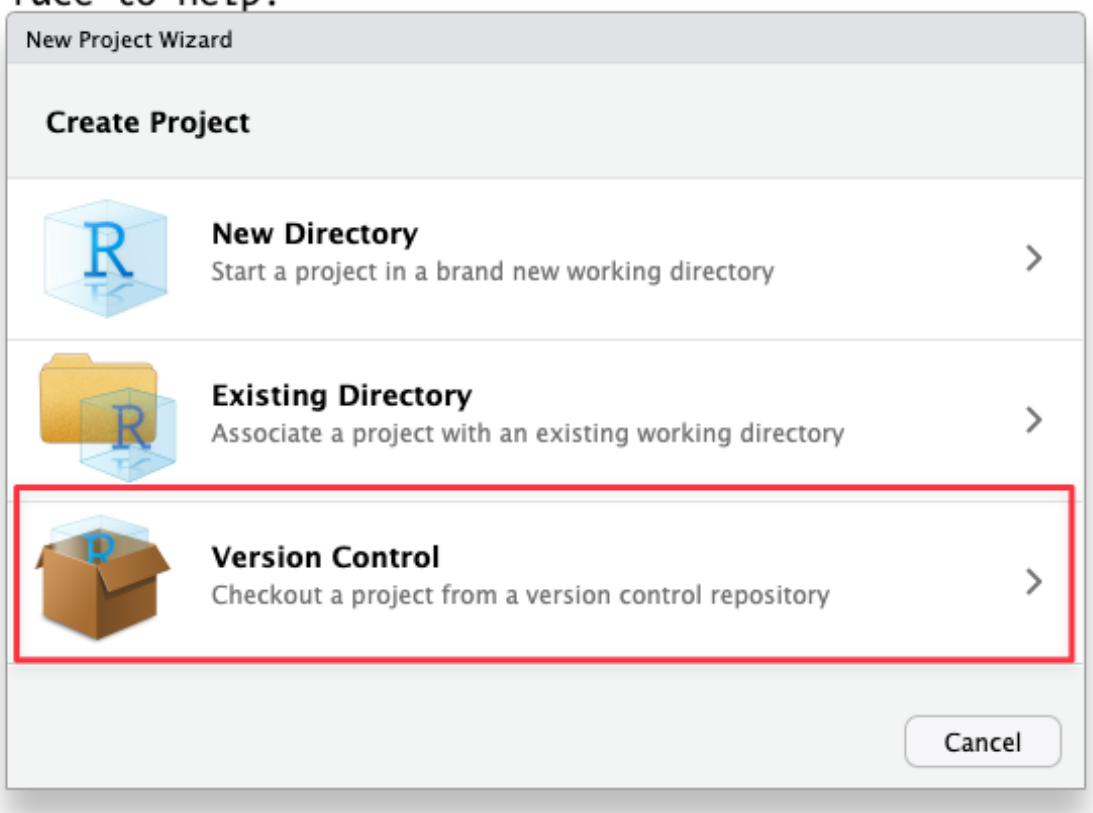
Next click on the green ‘Clone or Download’ button and copy the `https//...` URL that pops up for later (either highlight it all and copy or click on the copy to clipboard icon to the right).



Setting up git with Rstudio

Ok, we now switch our attention to RStudio. In RStudio click on the `File` -> `New Project` menu. In the pop up window select `Version Control`.

```
lp()' for on-line help, or
r interface to help.
```



Setting up git with Rstudio

Now paste the the URL you previously copied from GitHub into the `Repository URL:` box.

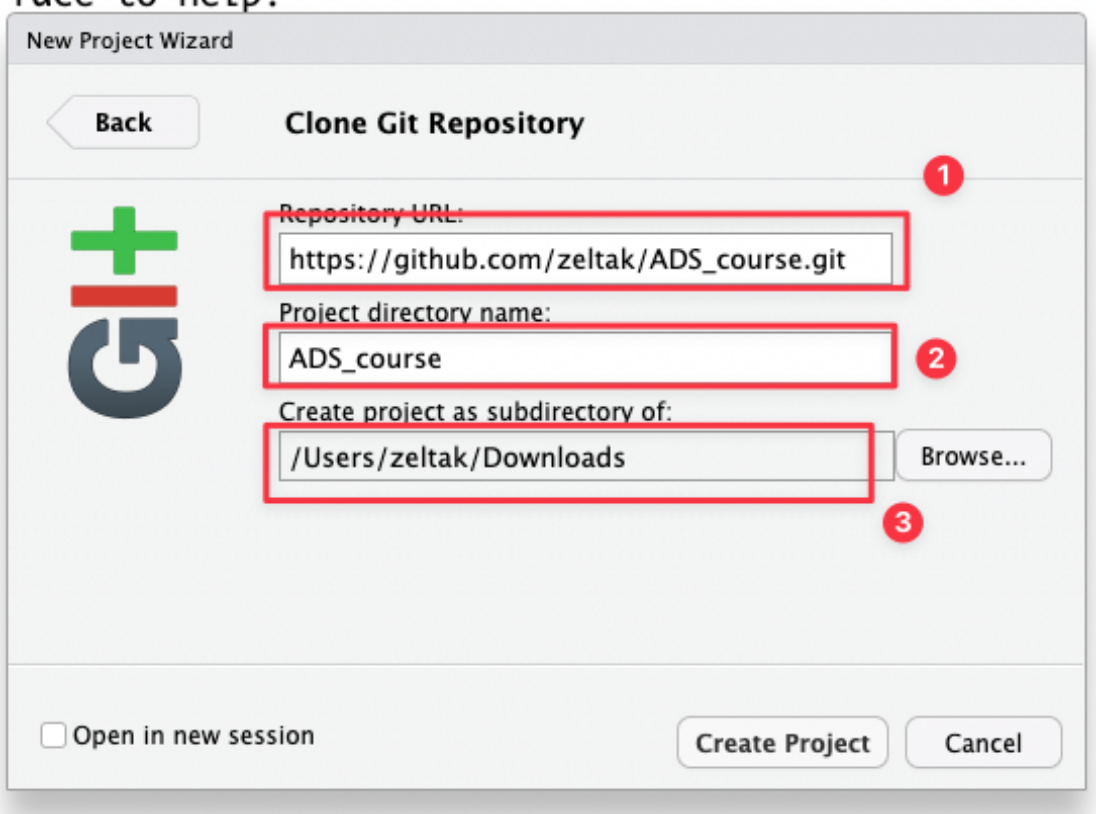
This should then automatically fill out the `Project Directory Name:` section with the correct repository name (it's important that the name of this directory has the same name as the repository you created in GitHub).

You can then select where you want to create this directory by clicking on the `Browse` button opposite the `Create project as a subdirectory of:` option.

Setting up git with Rstudio

Navigate to where you want the directory created and click OK. We also tick the `Open in new session` option.

```
()' for on-line help, or
 interface to help.
```



Setting up git with Rstudio

RStudio will now create a new directory with the same name as your repository on your local computer and will then **clone** your remote repository to this directory.

The directory will contain three new files; `first_repo.Rproj` (or whatever you called your repository), `README.md` and `.gitignore`.

You can check this out in the `Files` tab usually in the bottom right pane in RStudio. You will also have a `Git` tab in the top right pane with two files listed (we will come to this later on in the Chapter).

Setting up git with Rstudio

You now have a remote GitHub repository set up and this is linked to your local repository managed by RStudio. Any changes you make to files in this directory will be version controlled by Git.