

Version Control

Now that we've got a handle on R, RStudio, and projects, there are a few more things we want to set you up with before moving on to the other courses - understanding version control, installing Git, and linking Git with RStudio. In this lesson, we'll give you a basic understanding of version control.

What is version control?

First things first: What is version control? Version control is a system that records changes that are made to a file or a set of files over time. As you make edits, the version control system takes snapshots of your files and the changes, and then saves those snapshots so you can refer or revert back to previous versions later if need be! If you've ever used the "Track changes" feature in Microsoft Word, you have seen a rudimentary type of version control, in which the changes to a file are tracked, and you can either choose to keep those edits or revert to the original format. Version control systems, like [Git](#), are like a more sophisticated "Track changes" - in that they are far more powerful and are capable of meticulously tracking successive changes on many files, with potentially many people working simultaneously on the same groups of files.

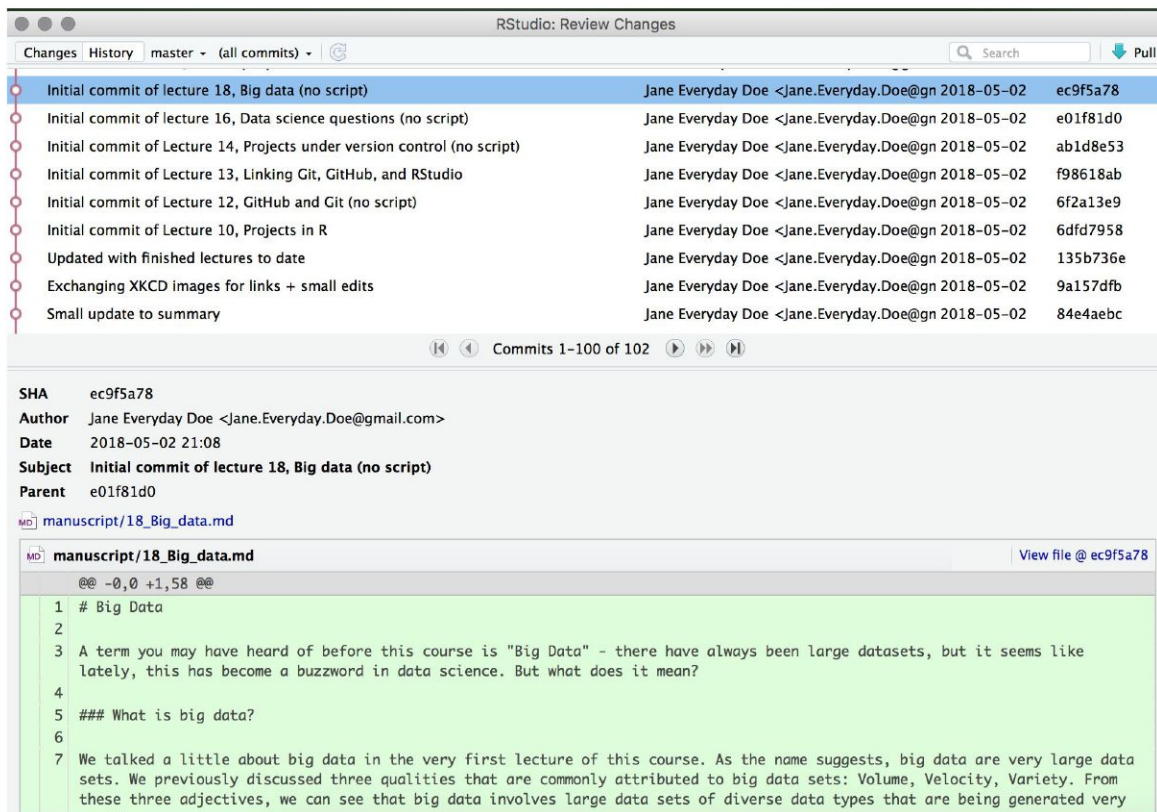
If you've ever worked collaboratively on a document before, [this comic](#) from PHD Comics might resonate with you.

Hopefully, once you've mastered version control software, `Paper_Final_FINAL2_actually_FINAL.docx` will be a thing of the past for you!

What are the benefits of using version control?

As we've seen in the example, without version control, you might be keeping multiple, very similar copies of a file. And this could be dangerous - you might start editing the wrong version, not recognizing that the document labelled "FINAL" has been further edited to "FINAL2" - and now all your new changes have been applied to the wrong file! Version control systems help to solve this problem by keeping a single, updated version of each file, with a record of *all* previous versions AND a record of exactly what changed between the versions.

Which brings us to the next major benefit of version control: It keeps a record of all changes made to the files. This can be of great help when you are collaborating with many people on the same files - the version control software keeps track of who, when, and why those specific changes were made. It's like "Track changes" to the extreme!



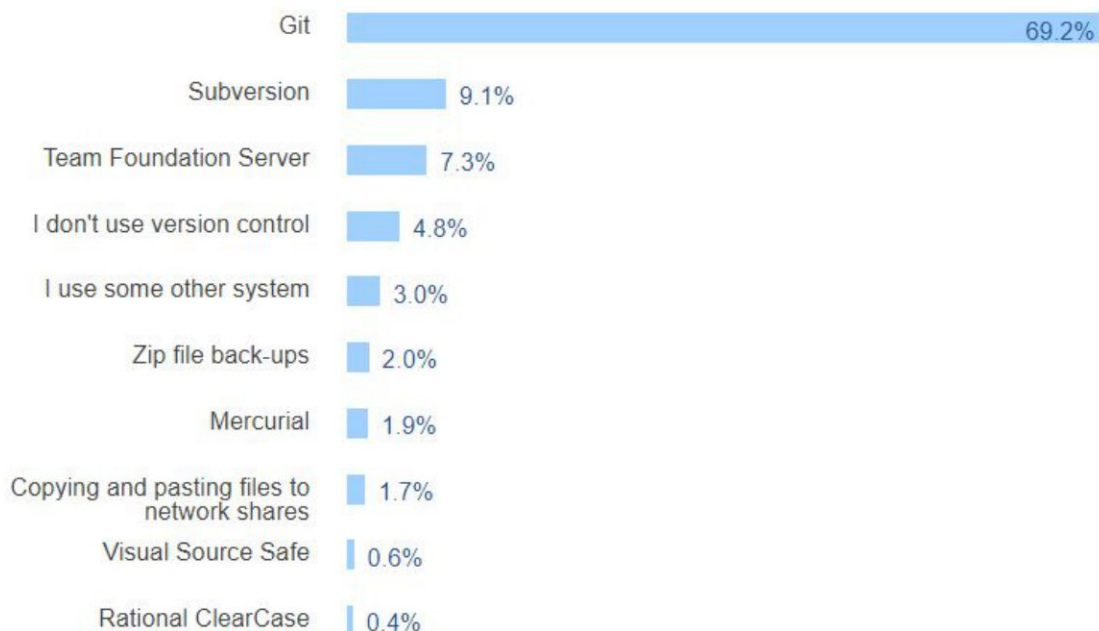
An example of the version control history for the development of this course!

This record is also helpful when developing code, if you realize after some time that you made a mistake and introduced an error. You can find the last time you edited that particular bit of code, see the changes you made, and revert back to that original, unbroken code, leaving everything else you've done in the meanwhile untouched!

Finally, when working with a group of people on the same set of files, version control is helpful for ensuring that you aren't making changes to files that conflict with other changes. If you've ever shared a document with another person for editing, you know the frustration of integrating their edits with a document that has changed since you sent the original file - now you have two versions of that same original document. Version control allows multiple people to work on the same file and then helps merge all of the versions of the file and all of their edits into one cohesive file.

What is Git? Why should you use it?

Git is a free and open source version control system. It was developed in 2005 and has since become *the* most commonly used version control system around! StackOverflow, which should sound familiar from our Getting Help lesson, surveyed over 60,000 respondents on which version control system they use, and as you can tell from the chart below, [Git is by far the winner](#).



<https://insights.stackoverflow.com/survey/2017#work-version-control>

Results of a StackOverflow survey asking which version control software their respondents use

And as you become more familiar with Git and how it works and interfaces with your projects, you'll begin to see why it has risen to the height of popularity. One of the main benefits of Git is that it keeps a local copy of your work and revisions, which you can then edit offline, and then once you return to internet service, you can sync your copy of the work, with all of your new edits and tracked changes to the main repository online. Additionally, since all collaborators on a project have their own local copy of the code, everybody can simultaneously work on their own parts of the code, without disturbing that common repository.

Another big benefit that we'll definitely be taking advantage of is the ease with which RStudio and Git interface with each other. In the next lesson, we'll work on getting Git installed and linked with RStudio and making a GitHub account.

What is GitHub?

GitHub is an online interface for Git. Git is software used locally on your computer to record changes. GitHub is a host for your files and the records of the changes made. You can sort of think of it as being similar to DropBox - the files are on your computer, but they are also hosted online and are accessible from any computer. GitHub has the added benefit of interfacing with Git to keep track of all of your file versions and changes.

Version control vocabulary

There is a lot of vocabulary involved in working with Git, and often the understanding of one word relies on your understanding of a different Git concept. Take some time to familiarize yourself with the words below and read over it a few times to see how the concepts relate.

Repository: Equivalent to the project's folder/directory - all of your version controlled files (and the recorded changes) are located in a repository. This is often shortened to **repo**. Repositories are what are hosted on GitHub and through this interface you can either keep your repositories private and share them with select collaborators, or you can make them public - anybody can see your files and their history.

Commit: To commit is to save your edits and the changes made. A commit is like a snapshot of your files: Git compares the previous version of all of your files in the repo to the current version and identifies those that have changed since then. Those that have not changed, it maintains that previously stored file, untouched. Those that have changed, it compares the files, logs the changes and uploads the new version of your file. We'll touch on this in the next section, but when you commit a file, typically you accompany that file change with a little note about what you changed and why.

When we talk about version control systems, commits are at the heart of them. If you find a mistake, you revert your files to a previous *commit*. If you want to see what has changed in a file over time, you compare the *commits* and look at the messages to see why and who.

Push: Updating the repository with your edits. Since Git involves making changes locally, you need to be able to share your changes with the common, online repository. Pushing is sending those committed changes to that repository, so now everybody has access to your edits.

Pull: Updating your local version of the repository to the current version, since others may have edited in the meanwhile. Because the shared repository is hosted online and any of your collaborators (or even yourself on a different computer!) could have made changes to the files and then pushed them to the shared repository, you are behind the times! The files you have locally on *your* computer may be outdated, so you pull to check if you are up to date with the main repository.

Analogies to these concepts

Staging: The act of preparing a file for a commit. For example, if since your last commit you have edited three files for completely different reasons, you don't want to commit all of the changes in one go; your message on why you are making the commit and what has changed will be complicated since three files have been changed for different reasons. So instead, you can stage just one of the files and prepare it for committing. Once you've committed that file, you can stage the second file and commit it. And so on. Staging allows you to separate out file changes into separate commits. Very helpful!

To summarize these commonly used terms so far and to test whether you've got the hang of this, files are hosted in a **repository** that is shared online with collaborators. You **pull** the repository's contents so that you have a local copy of the files that you can edit. Once you are happy with your changes to a file, you **stage** the file and then **commit** it. You **push** this commit to the shared repository. This uploads your new file and all of the changes and is accompanied by a message explaining what changed, why and by whom.

Branch: When the same file has two simultaneous copies. When you are working locally and editing a file, you have created a branch where your edits are not shared with the main repository (yet) - so there are two versions of the file: the version that everybody has access to on the repository and your local edited version of the file. Until you push your changes and merge them back into the main repository, you are working on a branch. Following a branch point, the version history splits into two and tracks the independent changes made to both the original file in the repository that others may be editing, and tracking your changes on your branch, and then merges the files together.

Merge: Independent edits of the same file are incorporated into a single, unified file. Independent edits are identified by Git and are brought together into a single file, with both sets of edits incorporated. But, you can see a potential problem here - if both people made an edit to the same sentence that precludes one of the edits from being possible, we have a problem! Git recognizes this disparity (**conflict**) and asks for user assistance in picking which edit to keep.

Conflict: When multiple people make changes to the same file and Git is unable to merge the edits. You are presented with the option to manually try and merge the edits or to keep one edit over the other.

****A visual representation of these concepts, from <https://www.atlassian.com/git/tutorials/using-branches/git-merge>****

Clone: Making a copy of an existing Git repository. If you have just been brought on to a project that has been tracked with version control, you would *clone* the repository to get access to and create a local version of all of the repository's files *and all of the tracked changes*.

Fork: A personal copy of a repository that you have taken from another person. If somebody is working on a cool project and you want to play around with it, you can fork their repository and then when you make changes, the edits are logged on *your* repository, not theirs.

Best practices

It can take some time to get used to working with version control software like Git, but there are a few things to keep in mind to help establish good habits that will help you out in the future.

One of those things is to make purposeful commits. Each commit should only address a single issue. This way if you need to identify when you changed a certain line of code, there is only one place to look to identify the change and you can easily see how to revert the code.

Similarly, making sure you write informative messages on each commit is a helpful habit to get into. If each message is precise in what was being changed, anybody can examine the committed file and identify the purpose for your change. Additionally, if you are looking for a specific edit you made in the past, you can easily scan through all of your commits to identify those changes related to the desired edit.

You don't want to get in the same habit that [XKCD](#) has!

Finally, be cognizant of the version of files you are working on. Frequently check that you are up to date with the current repo by frequently pulling. Additionally, don't hoard your edited files - once you have committed your files (and written that helpful message!), you should push those changes to the common repository. If you are done editing a section of code and are planning on moving on to an unrelated problem, you need to share that edit with your collaborators!

A summary of the main best practices to keep in mind as you work with version control

Summary

Now that we've covered what version control is and some of the benefits, you should be able to understand why we have three whole lessons dedicated to version control and installing it. We looked at what Git and GitHub are, and then covered much of the commonly used (and sometimes confusing!) vocabulary inherent to version control work. We then quickly went over some best practices to using Git – but the best way to get a hang of this all is to use it! Hopefully you feel like you have a better handle on how Git works than the people in [this XKCD comic](#)! So let's move on to the next lesson and get it installed!

GitHub and Git

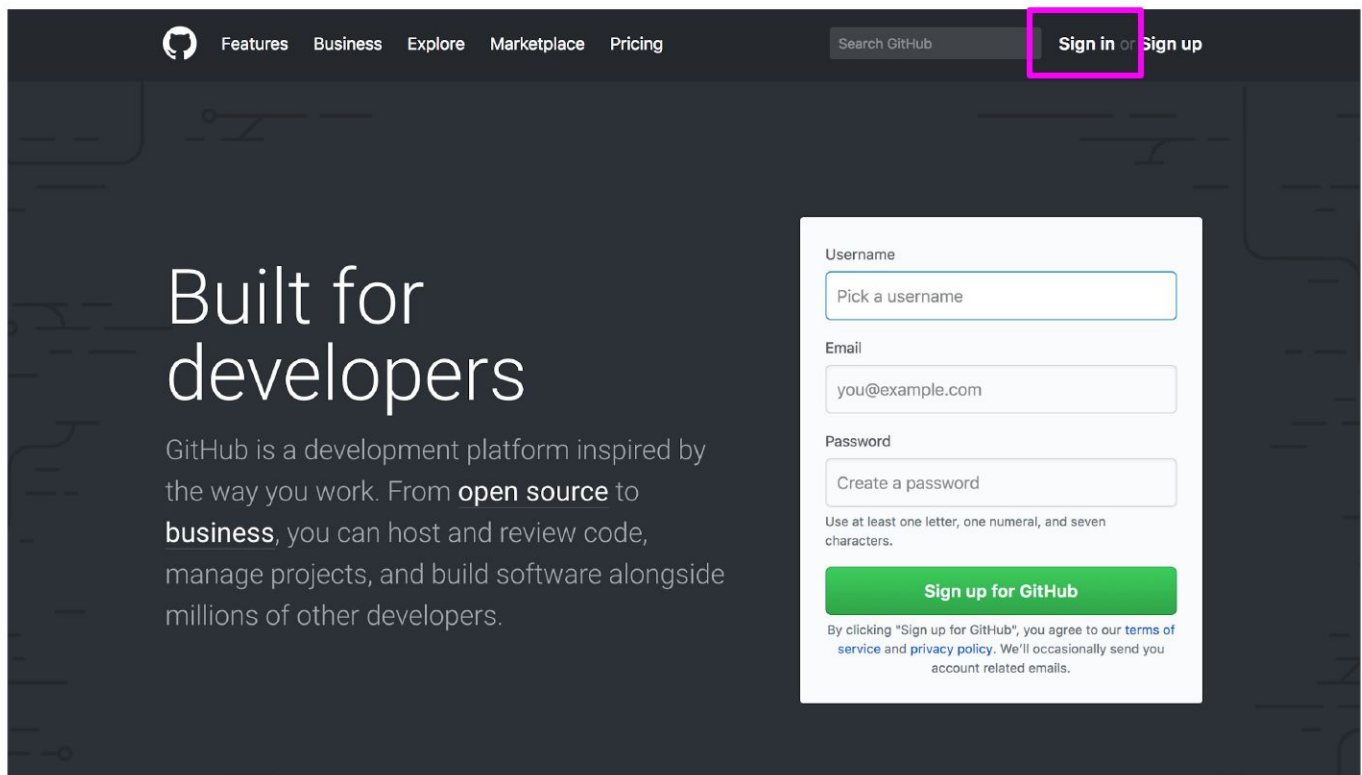
Now that we've got a handle on what version control is, in this lesson, you will sign-up for a GitHub account, navigate around the GitHub website to become familiar with some of its features, and install and configure Git; all in preparation for linking both with your RStudio!

What is GitHub?

As we previously learned, [GitHub](#) is a cloud-based management system for your version controlled files. Like DropBox, your files are both locally on your computer *and* hosted online and easily accessible. Its interface allows you to manage version control and provides users with a web-based interface for creating projects, sharing them, updating code, etc.

Signing up for GitHub

To get a GitHub account, first go to <https://github.com/>. You will be brought to their homepage, where you should fill in your information - make a username, put in your email, choose a secure password, and click "Sign up for GitHub."


The image is a screenshot of the GitHub homepage. The top navigation bar is dark with the GitHub logo on the left and links for Features, Business, Explore, Marketplace, and Pricing. A search bar is in the center, and on the right, the 'Sign in' and 'Sign up' links are highlighted with a pink rectangular box. The main content area has a dark background with the text 'Built for developers' in large white font. Below this, it says 'GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside millions of other developers.' On the right side, there is a white sign-up form. The form has three input fields: 'Username' with the placeholder 'Pick a username', 'Email' with the placeholder 'you@example.com', and 'Password' with the placeholder 'Create a password'. Below the password field, there is a note: 'Use at least one letter, one numeral, and seven characters.' At the bottom of the form is a green button that says 'Sign up for GitHub'. Below the button, there is a small disclaimer: 'By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.'

Signing up for GitHub

Logging in to GitHub

You should now be logged in to GitHub! In the future, to log on to GitHub, go to <https://github.com/>, where you will be presented with the homepage. If you aren't already logged in, click on the "Sign in" link at the top.

Once you've done that, you will see the log in page where you will enter in your username and password that you created earlier.



Sign in to GitHub

Username or email address

Password [Forgot password?](#)

Sign in

New to GitHub? [Create an account.](#)

github.com/login

GitHub's log in page

Once logged in, you will be back at <https://github.com/>, but this time the screen should look like this:

****GitHub's homepage at <https://github.com/>****

The homepage

We're going to take a quick tour of the GitHub website, and we'll particularly focus on these sections of the interface:

1. User settings
2. Notifications
3. Help files
4. The GitHub guide

Following this tour, we'll make your very first repository using the GitHub guide!

Some major features of GitHub

User settings

Now that you've logged on to GitHub, we should fill out some of your profile information and get acquainted with the account settings. In the upper right corner, there is an icon with an arrow beside it, click this and go to "Your profile"

Where to find user settings

This is where you control your account from and can view your contribution histories and repositories.

Your profile

Since you are just starting out, you aren't going to have any repositories or contributions yet - but hopefully we'll change that soon enough! What we can do right now is edit your profile.

Go to "Edit profile" along the lefthand edge of the page. Here, take some time and fill out your name and a little description of yourself in the "Bio" box, and if you like, upload a picture of yourself! When you are done, click "Update profile"

Editing your profile page

Along the lefthand side of this page, there are many options for you to explore. Click through each of these menus to get familiar with the options available to you. To get you started, go to the account page.

Your account page

Here, you can edit your password or if you are unhappy with your username, change it. Be careful though, there can be [unintended consequences](#) when you change your username - if you are just starting out and don't have any content yet, you'll probably be safe though.

Continue looking through the personal setting options on your own. When you are done, go back to your profile.

Once you've had a bit more experience with GitHub, you'll eventually end up with some repositories to your name. To find those, click on the "Repositories" link on your profile. For now, it will probably look like this:


Your repositories page


By the end of the lecture though, check back to this page to find your newly created repository!

Notifications

Next, we'll check out the [notifications menu](#). Along the menu bar across the top of your window, there is a bell icon, representing your notifications. Click on the bell.

Location of the bell icon


 Notifications

 Watching

Unread0

Participating0

All notifications



No new notifications.

Depending on [your notification settings](#), you'll see updates here for your conversations in watched repositories.

Your notifications

Once you become more active on GitHub and are collaborating with others, here is where you can find messages and notifications for all the repositories, teams, and conversations you are a part of.

Help files

Along the bottom of *every. single. page.* there is the [“Help” button](#). GitHub has a great help system in place - if you ever have a question about GitHub, this should be your first point to search! Take some time now and look through the various help files, and see if any catch your eye.

At the bottom of every page, you can find the Help page

Sometimes you just need a little help.



Bootcamp

- › Set Up Git
- › Create A Repo
- › Fork A Repo
- › Be Social

Setup

- › Signing up for a new GitHub account
- › Verifying your email address
- › About commit email addresses
- › Setting your commit email address on GitHub
- › Setting your commit email address in Git
- › Blocking command line pushes that expose your personal email address
- › Setting your username in Git
- › Dealing with line endings
- › Supported browsers

<https://help.github.com/>

GitHub's help files

The GitHub guide

GitHub recognizes that this can be an overwhelming process for new users, and as such have developed a mini tutorial to get you started with GitHub. Go through [this guide](#) now and create your first repository! When you are done, you should have a repository that looks something like this:

JaneEverydayDoe / hello-world

Watch 0Star 0Fork 0

<> CodeIssues 0Pull requests 0Projects 0WikiInsightsSettings

My first repository!

Edit

Add topics

3 commits1 branch0 releases1 contributor

Branch: masterNew pull requestCreate new fileUpload filesFind fileClone or download

JaneEverydayDoe Merge pull request #1 from JaneEverydayDoe/readme-editsLatest commit c9a6e3e a minute ago

README.mdUpdating README with location of instructions3 minutes ago

README.md

hello-world

My first repository!

Here is my first repository on this account. I created it entirely within the GitHub interface using the instructions located at: <https://guides.github.com/activities/hello-world>

Your first repository

Take some time to explore around the repository - Check out your commit history so far. Here you can find all of the changes that have been made to the repository, and you can see **who** made the change, **when** they made the change, and provided you wrote an appropriate commit message, you can see **why** they made the change! It should look like similar to this:

JaneEverydayDoe / hello-world

Watch0Star0Fork0

<> CodeIssues0Pull requests0Projects0WikiInsightsSettings

Branch: master

Commits on Mar 9, 2018

Merge pull request #1 from JaneEverydayDoe/readme-edits

Verifiedc9a6e3e<>

JaneEverydayDoe committed 2 minutes ago

Updating README with location of instructions

Verifieda26ec0e<>

JaneEverydayDoe committed 4 minutes ago

Initial commit

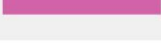

Verified4e050e4<>

JaneEverydayDoe committed 7 minutes ago

NewerOlder

Your first repository's commit history

Once you've explored all of the options in the repository, go back to your user profile. It should look a little different from before:



ProTip! Updating your profile with your name, location, and a profile picture helps other GitHub users get to know you.

[Edit profile](#)

Overview

Repositories 1

Stars 0

Followers 0

Following 0

Popular repositories


[hello-world](#)

My first repository!

Customize your pinned repositories

Joined 4 hours ago

Organizations




10 contributions in the last year

Contribution settings

	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Mon												
Wed												
Fri												

[Learn how we count contributions.](#)

Less  More

Your profile now shows your first repository

Now when you are on your profile you can see your latest repository created and for a complete listing of your repositories, click on the “Repositories” tab. Here you can see all of your repositories, a brief description, the time of the last edit, and along the right hand side, there is an activity graph, showing when and how many edits have been made on the repository.

Your shiny new repository page!

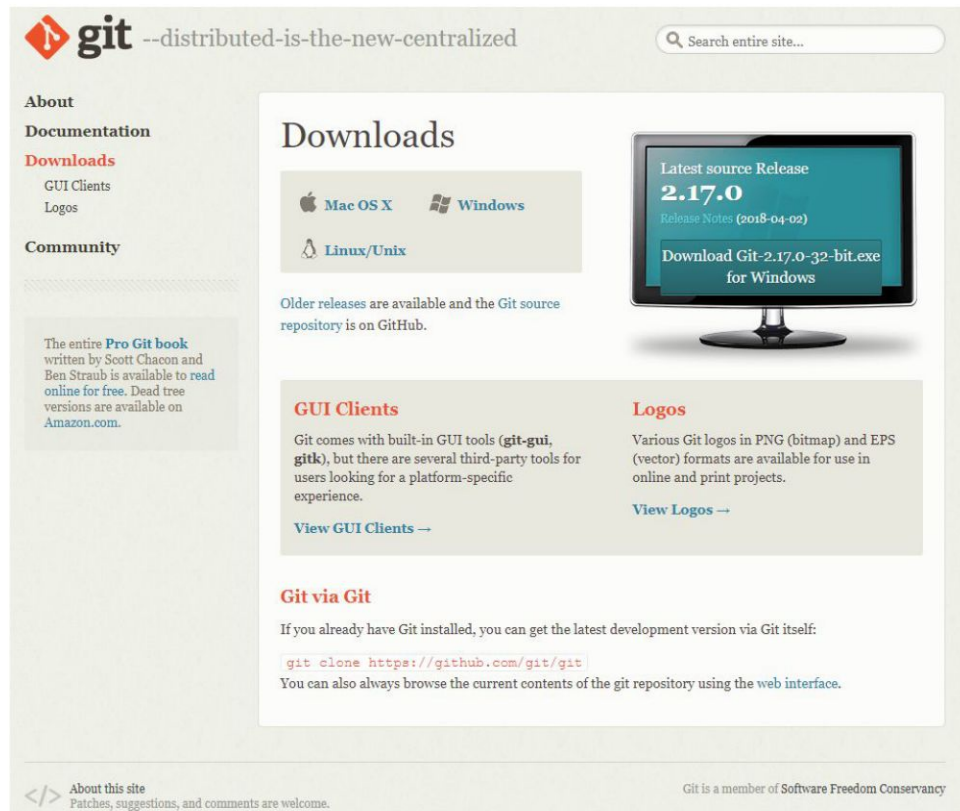
Git

As you may remember from our last lecture, Git is the free and open source version control system which GitHub is built on.

One of the main benefits of using the Git system is its compatibility with RStudio; however, in order to link the two software together, we first need to download and install Git on your computer.

Downloading and installing Git

To download Git, go to <https://git-scm.com/download>. You should arrive at a webpage like this:



Downloading Git from git-scm.com/download

Click on the appropriate download link for your operating system. This should initiate the download process.

For Windows

Once the download is finished, open the .exe file to initiate the installation wizard. If you receive a security warning, click “Run” and/or “Allow.” Following this, click through the installation wizard, generally accepting the default options unless you have a compelling reason not to.

Installation wizard for Git on Windows

Click “Install” and allow the wizard to complete the installation process. Following this, check the “Launch Git Bash” option, and unless you are curious, deselect the “View Release Notes” box, as you are probably not interested in this right now.

Finishing the install process

Doing so, a command line environment will open. Provided you accepted the default options during the installation process, there will now be a Start menu shortcut to launch Git Bash in the future. You have now installed Git.

Git Bash is the command line interface you will use to configure Git

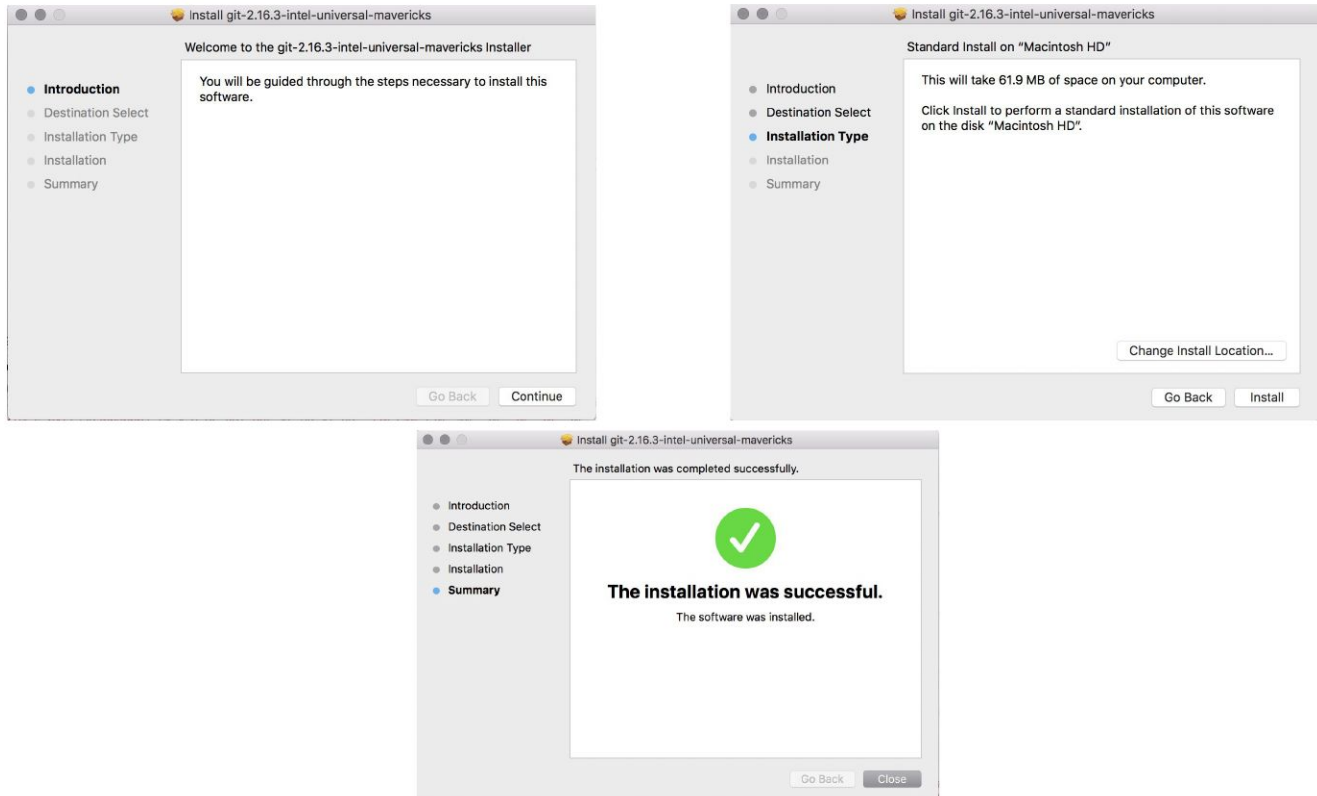
For Mac

We will walk you through the most common installation process however, there are multiple ways to get Git onto your Mac. You can follow the tutorials at <https://www.atlassian.com/git/tutorials/install-git> for alternative installation routes.

After downloading the appropriate Git version for Macs, you should have downloaded a DMG file for installation on your Mac. Open this file. This will install Git on your computer. A new window will open.

Installation wizard for Git on Mac

Double click on the .pkg file and an installation wizard will open. Click through the options, accepting the defaults. Click Install. When prompted, close the installation wizard. You have successfully installed Git!



Steps to a successful installation of Git!

Configuring Git

Now that Git is installed, we need to configure it for use with GitHub, in preparation for linking it with RStudio.

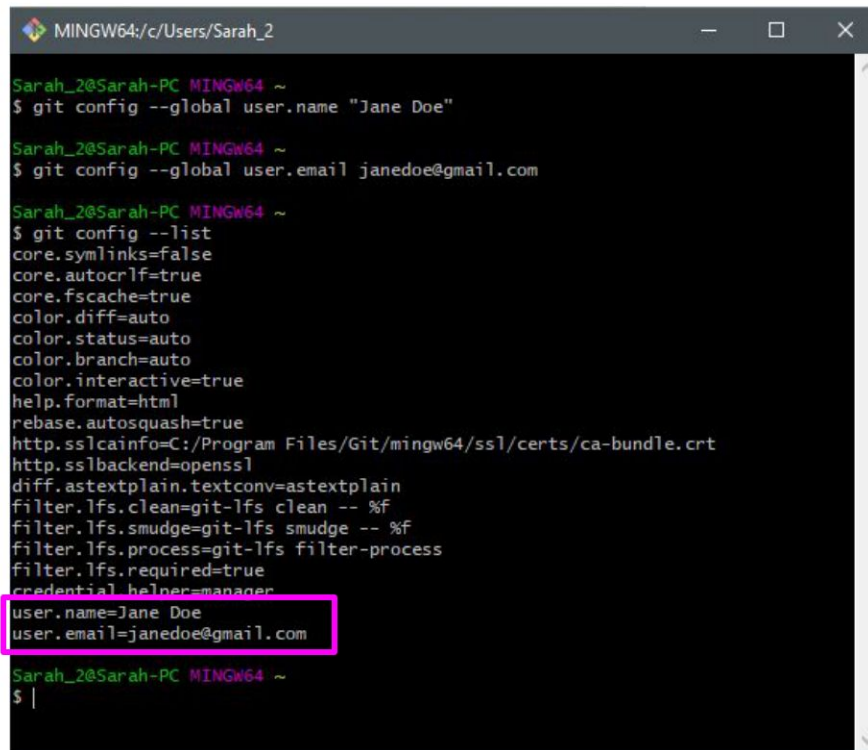
We need to tell Git what your username and email are, so that it knows how to name each commit as coming from you. To do so, in the command prompt (either Git Bash for Windows or Terminal for Mac), type: `git config --global user.name "Jane Doe"` with your desired username in place of "Jane Doe." This is the name each commit will be tagged with.

Following this, in the command prompt, type: `git config --global user.email janedoe@gmail.com` **MAKING SURE TO USE THE SAME EMAIL ADDRESS YOU SIGNED UP FOR GITHUB WITH!**

Configuring Git to tag each commit with your name and interface with GitHub

Confirming your configuration

At this point, you should be set for the next step, but just to check, confirm your changes by typing: `git config --list`



```
MINGW64: c:/Users/Sarah_2

Sarah_2@Sarah-PC MINGW64 ~
$ git config --global user.name "Jane Doe"

Sarah_2@Sarah-PC MINGW64 ~
$ git config --global user.email janedoe@gmail.com

Sarah_2@Sarah-PC MINGW64 ~
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
user.name=Jane Doe
user.email=janedoe@gmail.com

Sarah_2@Sarah-PC MINGW64 ~
$ |
```

Confirming your user name and user email

Doing so, you should see the username and email you selected above. If you notice any problems or want to change these values, just retype the original config commands from earlier with your desired changes.

Once you are satisfied that your username and email is correct, exit the command line by typing `exit` and hit Enter. At this point, you are all set up for the next lecture!

Exiting the command prompt

Summary

In this lesson, we signed up for a GitHub account and toured the GitHub website. We made your first repository and filled in some basic profile information on GitHub. Following this, we installed Git on your computer and configured it for compatibility with GitHub and RStudio.

Linking Git/GitHub and RStudio

Now that we have both RStudio and Git set-up on your computer and a GitHub account, it's time to link them together so that you can maximize the benefits of using RStudio in your version control pipelines.

Linking RStudio and Git

In RStudio, go to Tools > Global Options > Git/SVN

Use the Global Options menu to tell RStudio you are using Git as your version control system

Sometimes the default path to the Git executable is not correct. Confirm that git.exe resides in the directory that RStudio has specified; if not, change the directory to the correct path. Otherwise, click OK or Apply.

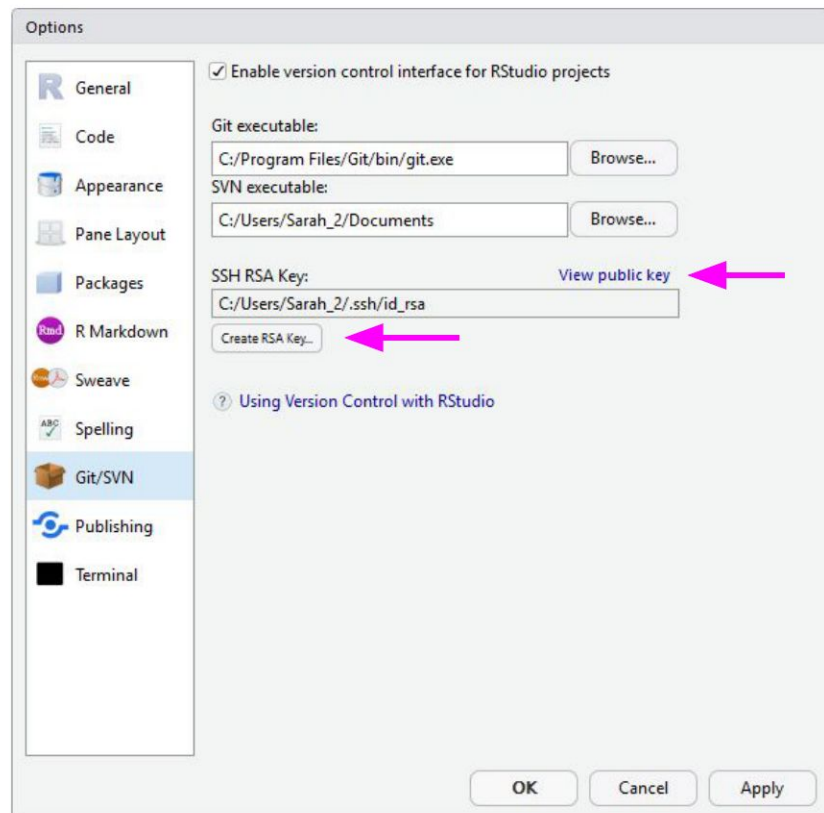
Confirm that the directory RStudio points to for the Git executable is correct

RStudio and Git are now linked.

Linking RStudio and GitHub

In that same RStudio option window, click “Create RSA Key” and when this completes, click “Close.”

Following this, in that same window again, click “View public key” and copy the string of numbers and letters. Close this window.



Generate an RSA key and copy the public key to your clipboard

You have now created a key that is specific to you which we will provide to GitHub, so that it knows who you are when you commit a change from within RStudio.

To do so, go to github.com/, log-in if you are not already, and go to your account settings. There, go to “SSH and GPG keys” and click “New SSH key”. Paste in the public key you have copied from RStudio into the Key box and give it a Title related to RStudio. Confirm the addition of the key with your GitHub password.

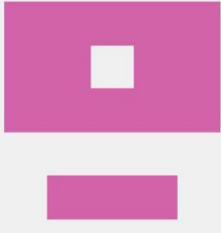
Location of “SSH and GPG keys” on your profile settings

Telling GitHub the public SSH key generated in RStudio

GitHub and RStudio are now linked. From here, we can create a repository on GitHub and link to RStudio.

Create a new repository and edit it in RStudio

On GitHub, create a new repository (github.com > Your Profile > Repositories > New). Name your new test repository and give it a short description. Click Create repository. Copy the URL for your new repository.



JaneEverydayDoe

[Add a bio](#)

[Edit profile](#)

🕒 Joined 2 hours ago

ProTip! Updating your profile with your name, location, and a profile picture helps other GitHub users get to know you. [Edit profile](#) ✕




Overview **Repositories 0** Stars 0 Followers 0 Following 0


Popular repositories

You don't have any public repositories yet.

1 contribution in the last year Contribution settings ▾

	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb
Mon												
Wed												
Fri												■

[Learn how we count contributions.](#) Less    More

This is your **contribution graph**. Your first  is for joining GitHub and you'll earn more as you make [additional contributions](#). More contributions means a darker green square for that day. Over time, your chart might start looking [something like this](#).

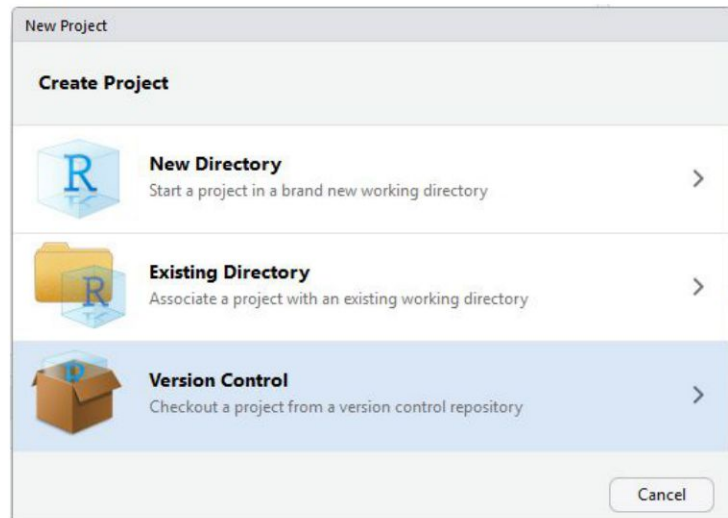
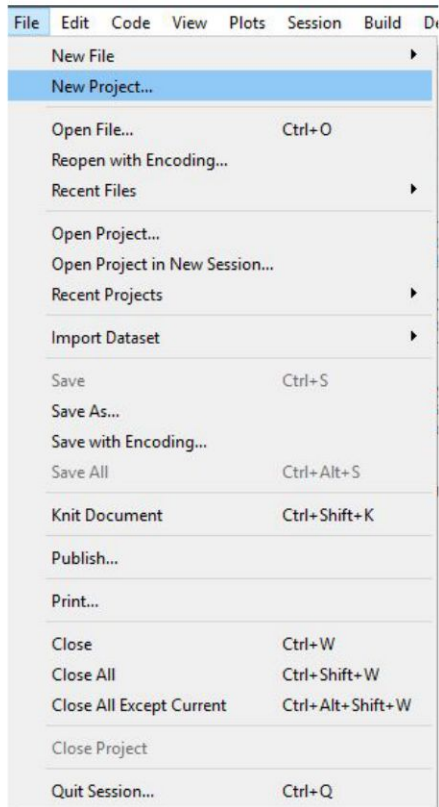
We have a quick guide that will show you how to create your first repository and earn more green squares!

[Read the Hello World guide](#)

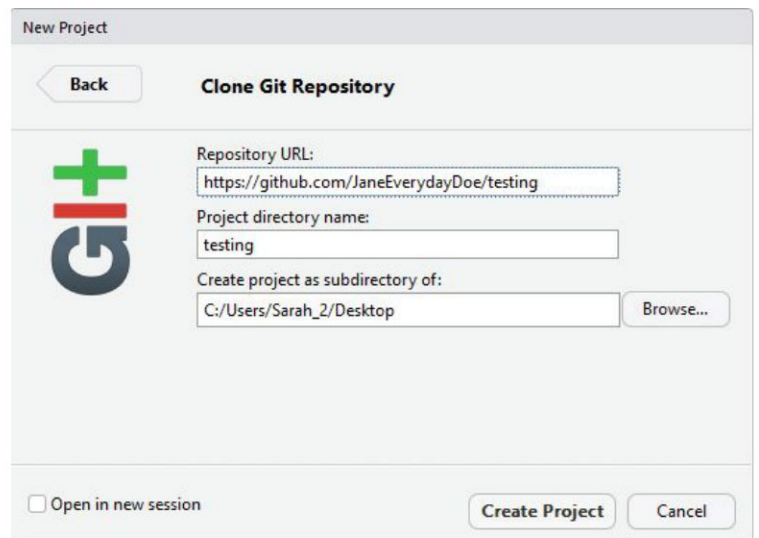
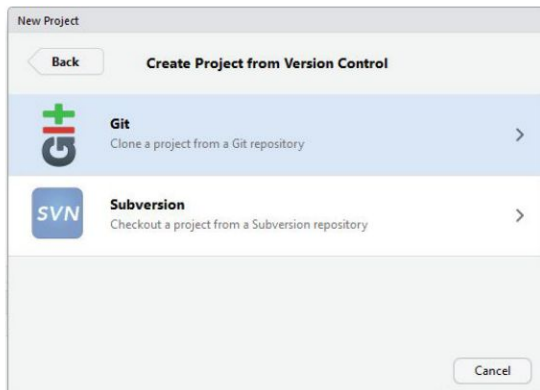
Location of the “Repositories” link on your profile

Creating a new repository on GitHub

In RStudio, go to File > New Project. Select Version Control. Select Git as your version control software. Paste in the repository URL from before, select the location where you would like the project stored. When done, click on “Create Project”. Doing so will initialize a new project, linked to the GitHub repository, and open a new session of RStudio.



Creating a version controlled project on RStudio



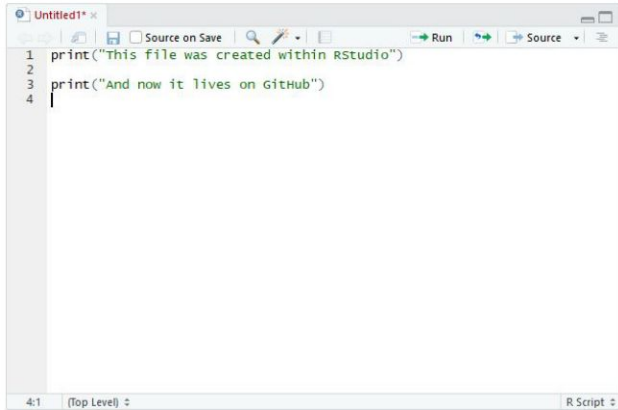
Cloning your Git repository to RStudio

Create a new R script (File > New File > R Script) and copy and paste the following code:

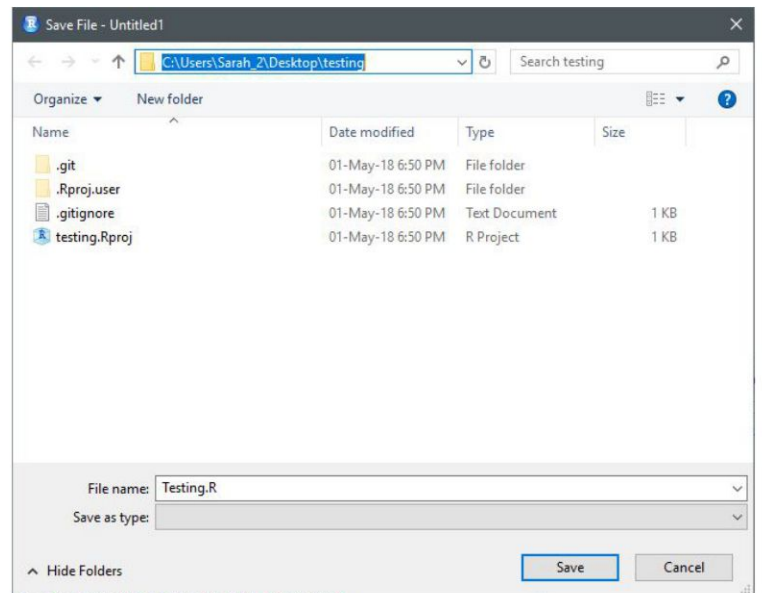
```
print("This file was created within RStudio")
```

```
print("And now it lives on GitHub")
```

Save the file. Note that when you do so, the default location for the file is within the new Project directory you created earlier.



```
1 print("This file was created within RStudio")
2
3 print("And now it lives on GitHub")
4
```

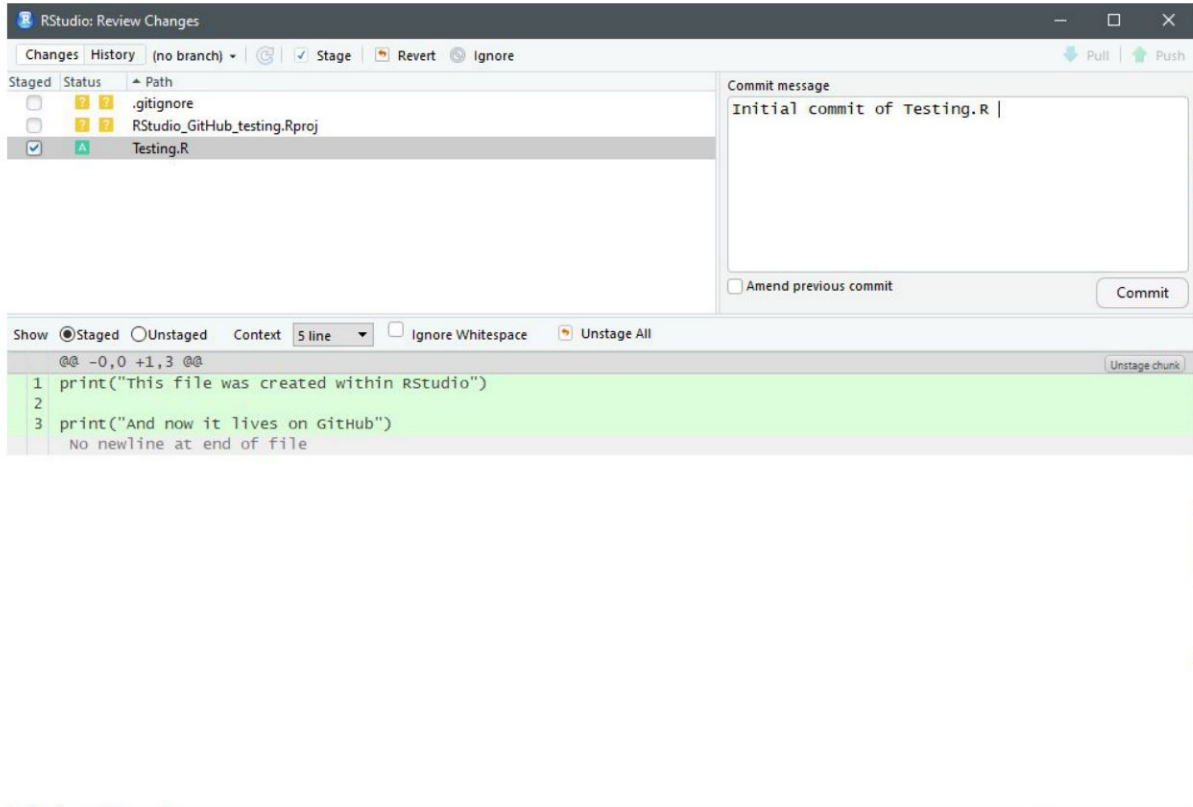


Saving your first script for this project

Once that is done, looking back at RStudio, in the Git tab of the environment quadrant, you should see your file you just created! Click the checkbox under “Staged” to stage your file.

All files that have been modified since your last pull appear in the Git tab

Click “Commit”. A new window should open, that lists all of the changed files from earlier, and below that shows the differences in the staged files from previous versions. In the upper quadrant, in the “Commit message” box, write yourself a commit message. Click Commit. Close the window.



Committing your R Script to the repository!

So far, you have created a file, saved it, staged it, and committed it. If you remember your version control lecture, the next step is to push your changes to your online repository. Push your changes to the GitHub repository.

How to push your commit to the GitHub repository

Go to your GitHub repository and see that the commit has been recorded.

You've just successfully pushed your first commit from within RStudio to GitHub!

Summary

In this lesson, we linked Git and RStudio, so that RStudio recognizes you are using Git as your version control software. Following that, we linked RStudio to GitHub, so that you can push and pull repositories from within RStudio. To test this, we created a repository on GitHub, linked it with a new project within RStudio, created a new file, and then staged, committed, and pushed the file to your GitHub repository!

Projects under version control

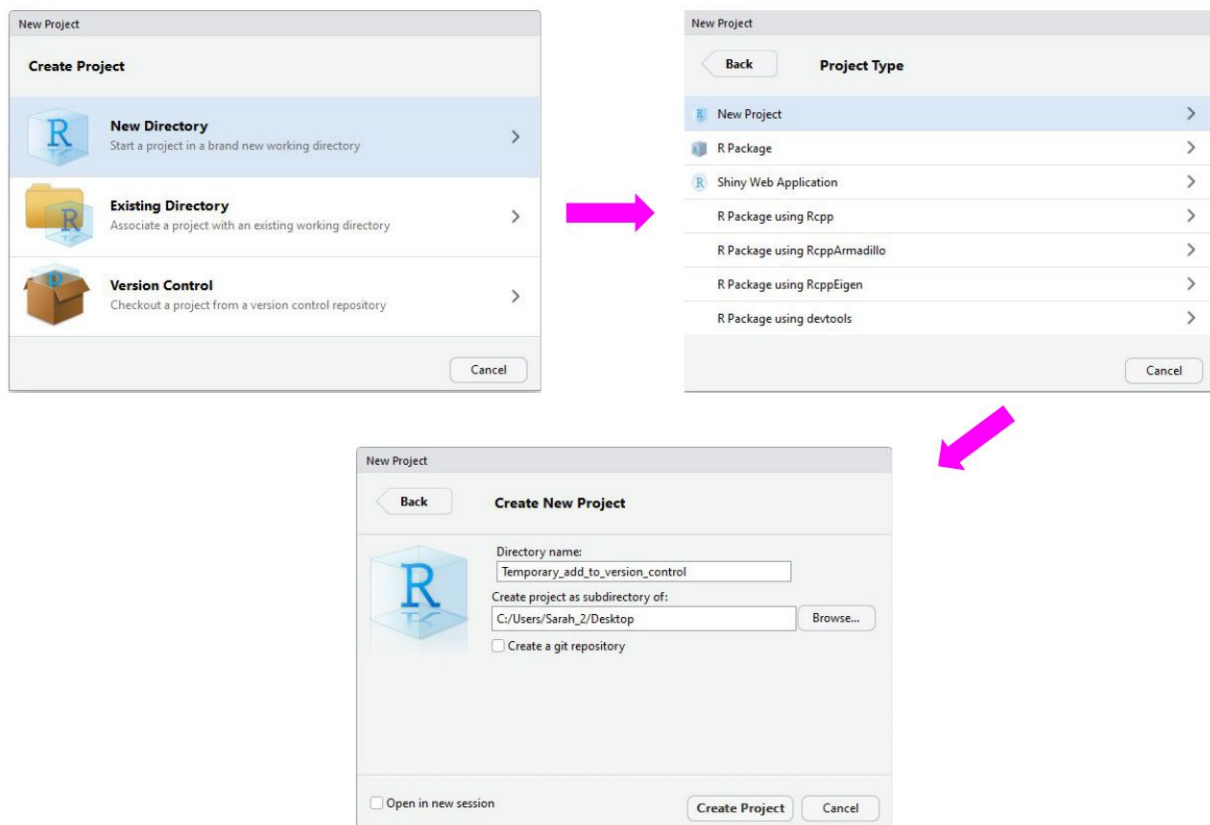
In the previous lesson, we linked RStudio with Git and GitHub. In doing this, we created a repository on GitHub and linked it to RStudio. Sometimes, however, you may already have an R Project that isn't yet under version control or linked with GitHub. Let's fix that!

Linking an existing Project with Git

So what if you already have an R Project that you've been working on, but don't have it linked up to any version control software (tut tut!)?

Thankfully, RStudio and GitHub recognize this can happen and have steps in place to help you (admittedly, this is slightly more troublesome to do than just creating a repository on GitHub and linking it with RStudio before starting the project...).

So first, let's set up a situation where we have a local project that isn't under version control. Go to File > New Project > New Directory > New Project and name your project. Since we are trying to emulate a time where you have a project not currently under version control, do **NOT** click "Create a git repository". Click Create Project.



Creating a project that is not under version control

We've now created an R Project that is not currently under version control. Let's fix that. First, let's set it up to interact with Git. Open Git Bash or Terminal and navigate to the directory containing your project files. Move around directories by typing `cd ~/dir/name/of/path/to/file`

When the command prompt in the line before the dollar sign says the correct directory location of your project, you are in the correct location. Once here, type `git init` followed by `git add .` - this initializes (*init*) this directory as a git repository and *adds* all of the files in the directory (.) to your local repository. Commit these changes to the git repository using `git commit -m "Initial commit"`

Linking the project folder with Git so it is now under version control

At this point, we have created an R Project and have now linked it to Git version control. The next step is to link this with GitHub.

Linking this project with GitHub

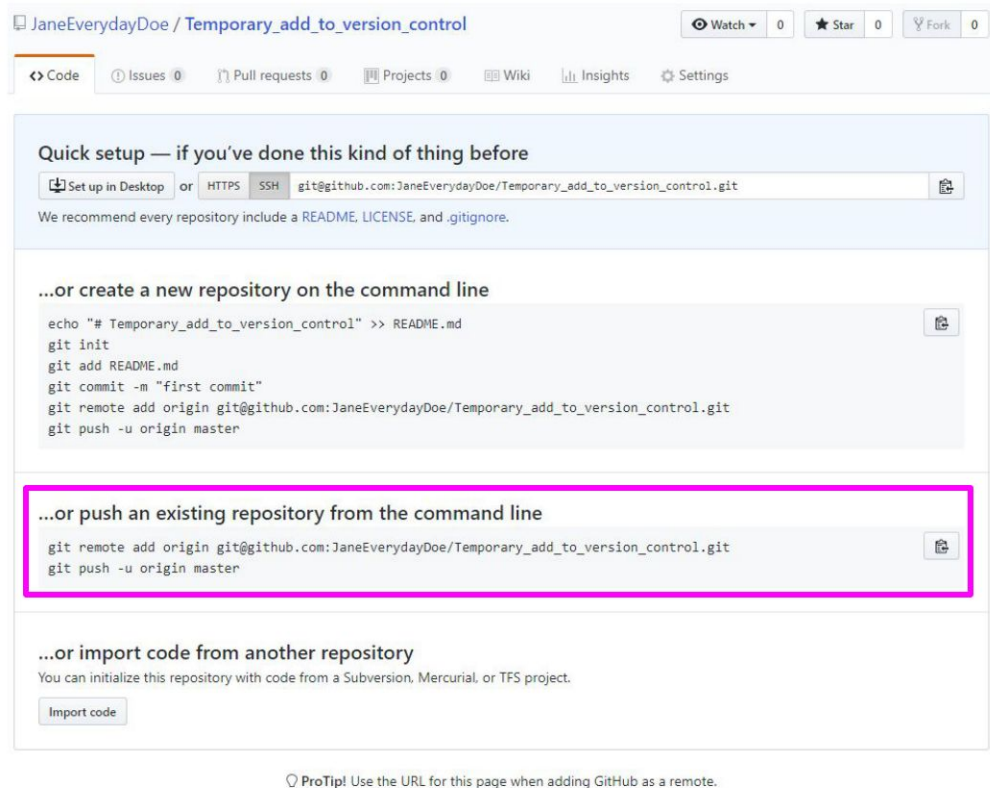
To do this, go to GitHub.com, and again, create a new repository:

- 1) Make sure the name is the **exact same** as your R project;

2) Do **NOT** initialize a README file, .gitignore, or license.

Creating a repository on GitHub that is named the same as your R project

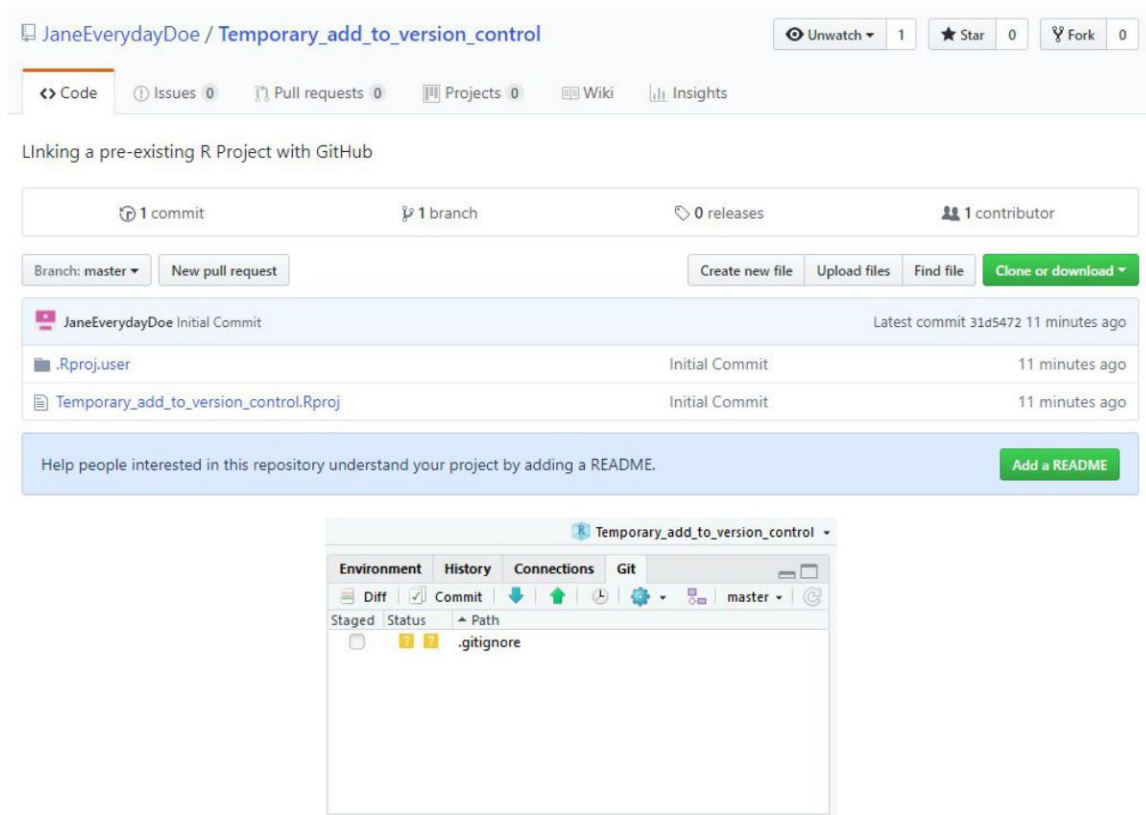
Upon creating the repository, you should see a page like this:



Your new repository on GitHub containing code to push from the command line

You should see that there is an option to “Push an existing repository from the command line” with instructions below containing code on how to do so. In Git Bash or Terminal, copy and paste these lines of code to link your repository with GitHub. After doing so, refresh your GitHub page and it should now look something like the image below.

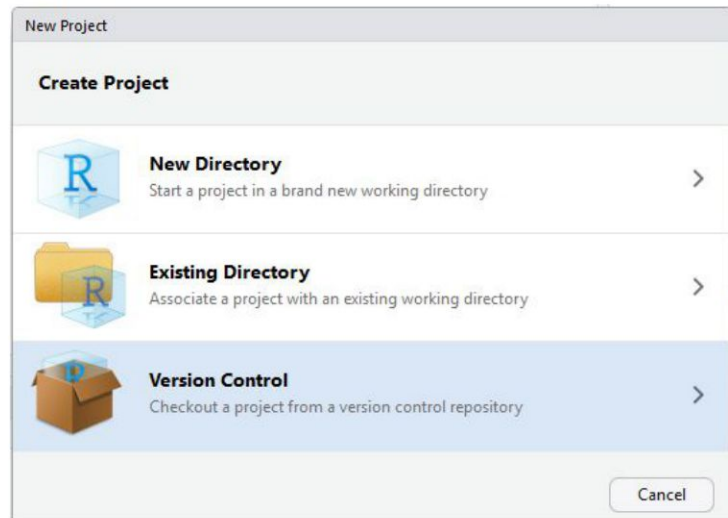
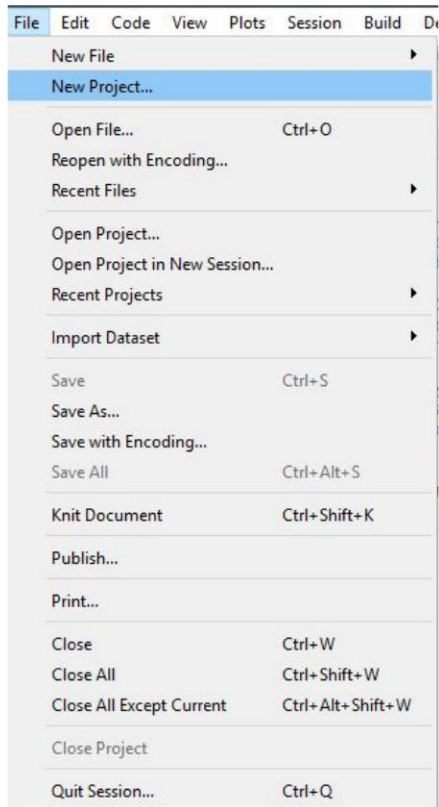
When you re-open your project in RStudio, you should now have access to the Git tab in the upper right quadrant and can push to GitHub from within RStudio any future changes.



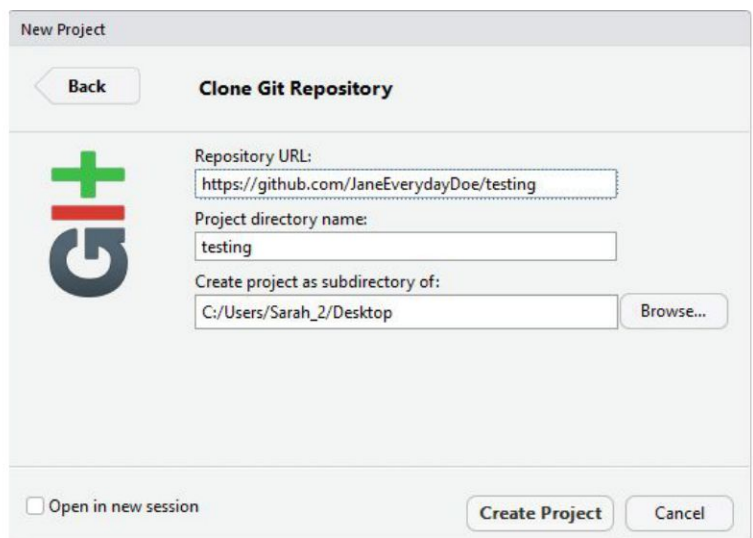
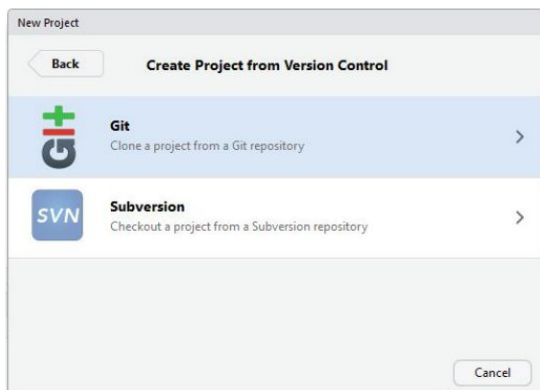
You've now pushed your R project repository to your GitHub repository of the same name

Working on an existing GitHub repository

If there is an existing project that others are working on that you are asked to contribute to, you can link the existing project with your RStudio. It follows the exact same premise as that from the last lesson where you created a GitHub repository and then cloned it to your local computer using RStudio. In brief, in RStudio, go to File > New Project > Version Control. Select Git as your version control system, and like in the last lesson, provide the URL to the repository that you are attempting to clone and select a location on your computer to store the files locally. Create the project.



Follow the same steps as previously done to clone your own repository to a new project in RStudio



Clone an existing project from GitHub from within RStudio

All the existing files in the repository should now be stored locally on your computer and you have the ability to push edits from your RStudio interface. The only difference from the last lesson is that you did not create the original repository, instead you cloned somebody else's.

Summary

In this lesson, we went over how to convert an existing project to be under Git version control using the command line. Following this, we linked your newly version controlled project to GitHub using a mix of GitHub commands and the command line. We then briefly recapped how to clone an existing GitHub repository to your local machine using RStudio.