Part 4: Keep track of changes with version control

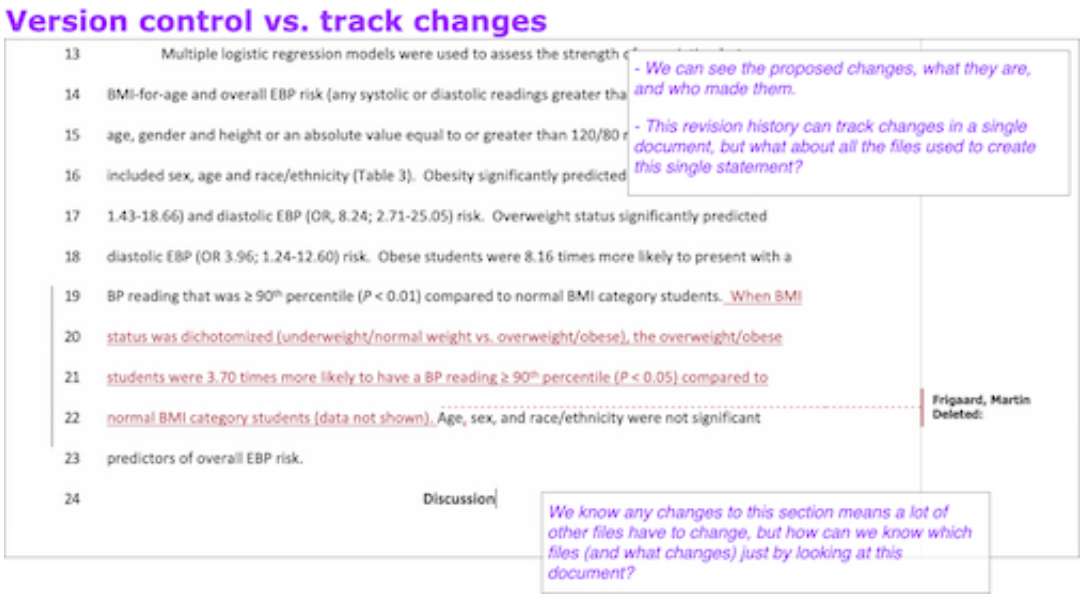
In the previous sections, we've covered using **Terminal** in the RStudio. If you're unfamiliar with these topics, please start there. This section will include tracking changes with version control, specifically Git, Github, and RStudio.

Tracing your steps

'Sharing your' can take a few forms. You can finish a project, then share your work for people to see and use in what they're doing.
Another option is to share what you're currently working on in a way that allows other people can collaborate with you along the way.

To accomplish the second option, we need a means showing how our work has changed over time. For example, maybe you've used the 'Review' tools in Microsoft Word, or you've collaborated in a Google sheet document. Both are types of version control because they're a formal system of managing changes to information over time.

Consider the image below of a .docx file:



The file is an earlier version of a manuscript. A coauthor has suggested changes to the results section. Sound version control systems let us see four aspects of changes:

1) what the differences are,
2) who recommended them,
3) the time/date of the proposed changes, and
4) any comments about the change

Unfortunately, tracked changes in Word only applies to a single document at a time. When you're working collaboratively (which is quite often), you know asking someone to change a single sentence can result in changes to dozens of files. That's why we need a way to track changes across a project's multiple files.

Git

Git is a version control system (VCS), which is somewhat like the **Tracked Changes** in Microsoft Word or the **Version History** in Google Docs, but extended to every file in a project. Git will help you keep track of your documents, datasets, code, images, and anything else you tell it to keep an eye on.

Why use Git?

You will eventually ask yourself, *why am I subjecting myself to this–is there another way?*

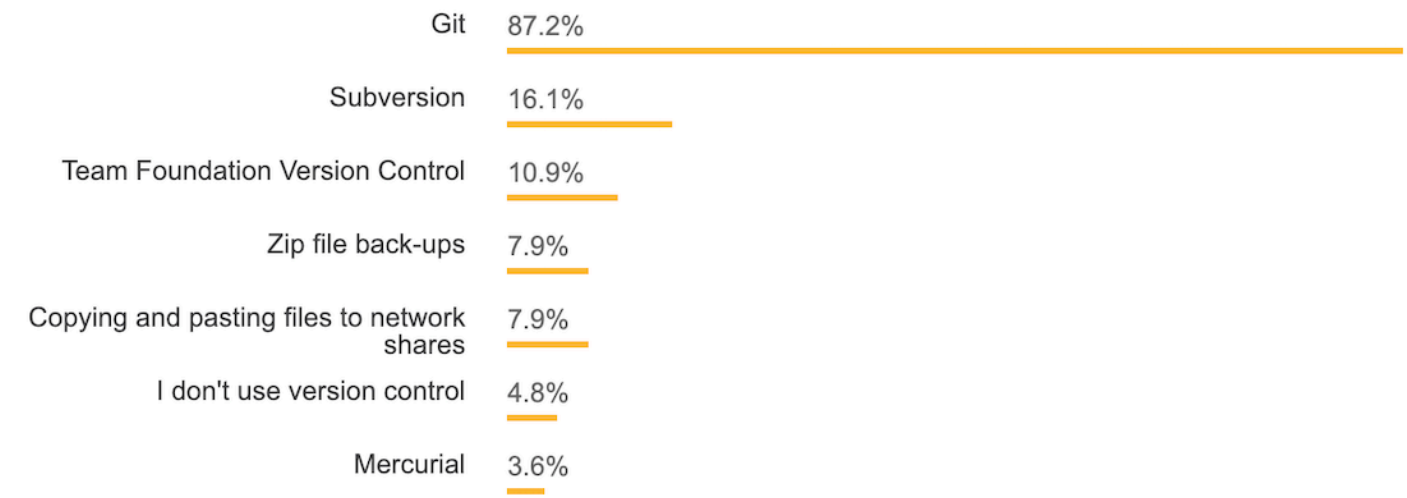
We've included these sections to remind you that you're making a sound choice.

Plain text + Git

Software developers keep track of their code with Git. We learned in the last chapter that most code files get kept in plain text, so Git plays well with plain text.

Everyone else is doing it

Git has become the most common version control system used by programmers.



Git is the dominant choice for version control for developers today, with almost 90% of developers checking in their code via Git.

source: StackOverflow Developer Survey Results

Git is a useful way to think about making changes

Git is also a helpful way of thinking about the changes to your project. The terminology of Git is strange at first, but if you use Git long enough, you'll be thinking about your code in terms of 'adds', 'commits,' 'pushes,' 'pulls,' and 'repos.'

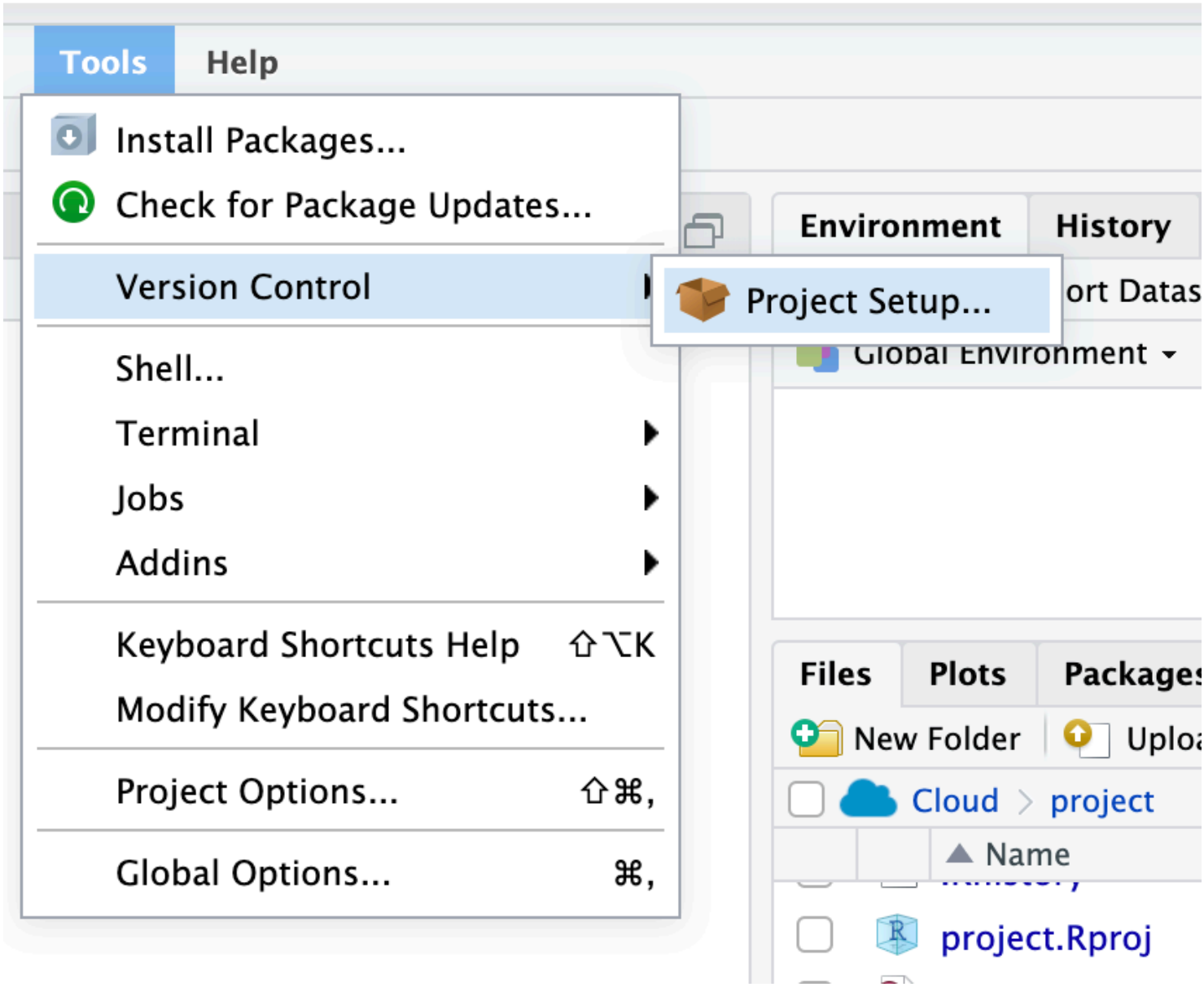
As someone who analyzes data regularly, these concepts are also countable, which means you can quantify change and work in more exciting ways.

Setting up Git

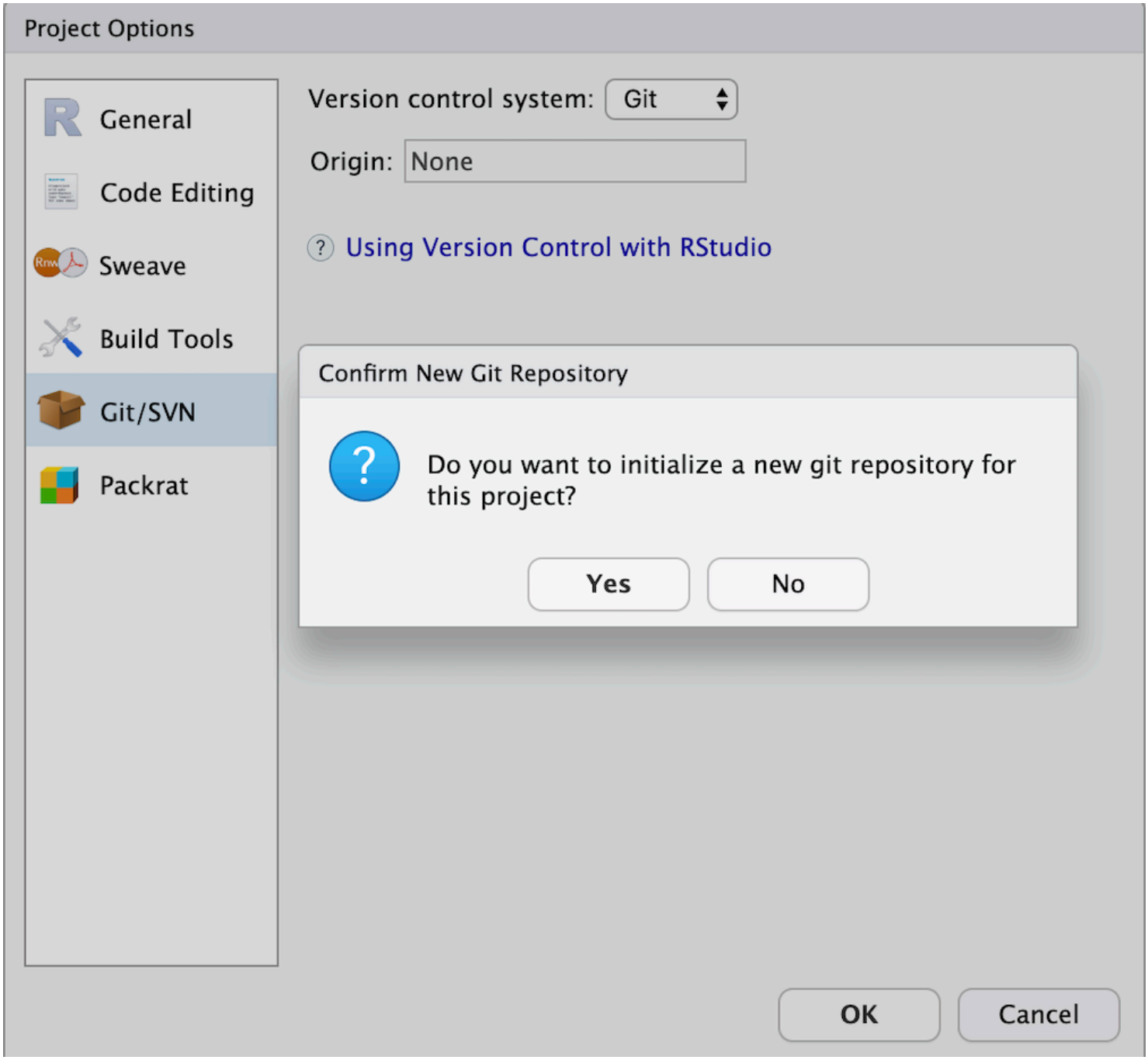
If you'd like to install Git on your local machine, you can do so following these two links:

1. Download and install Git.
2. Create a Github account.

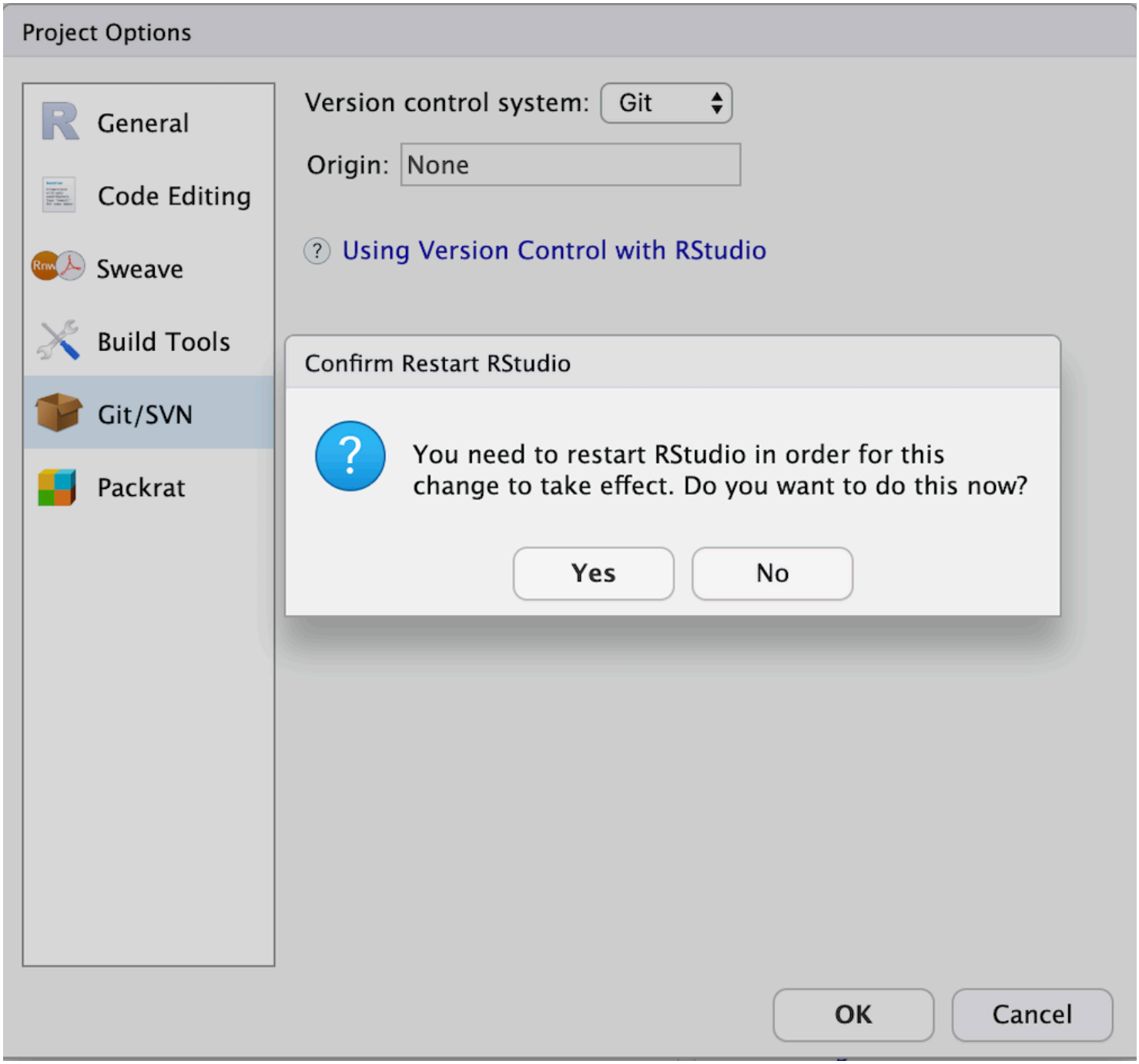
In RStudio.Cloud, we want to add version control to this project from *Tools* > *Version Control* > *Project Setup*



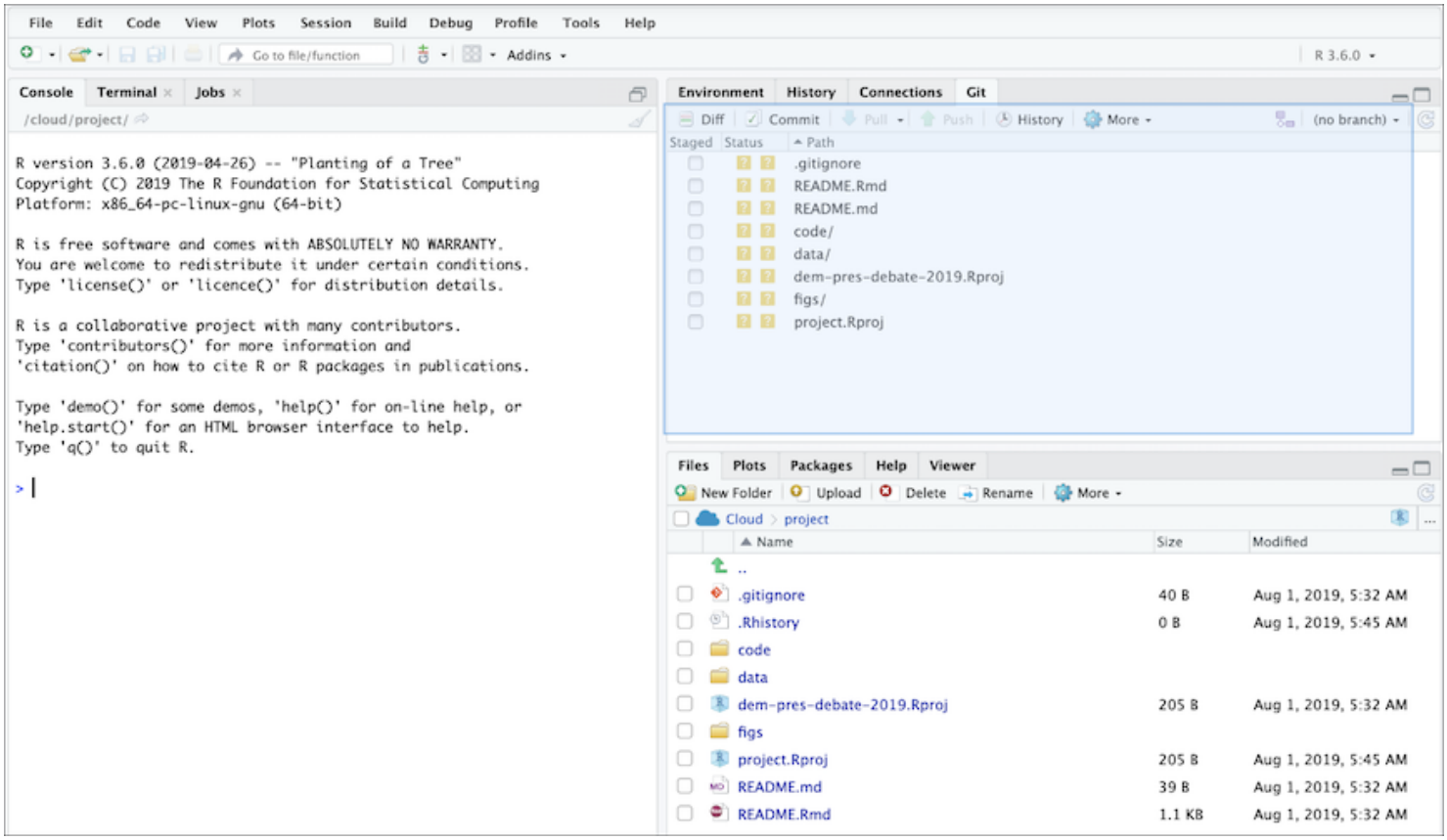
From here, you will see the *Git/SVN* option on the sidebar, where you will select *Git* from the dropdown list next to *Version control system*. After this, RStudio.Cloud will ask if you want to *initialize a new git repo*, which you do.



Then you will be asked if we are ok to restart RStudio.Cloud (and we do).



After restarting the RStudio.Cloud IDE, we should see the **Git** tab in one of the panes.



Configuring Git

Git needs a little configuration before we can start using it and linking it to Github. There are three levels of configuration within Git, `system`, `user`, and `project`.

1) For **system** level configuration use:
`git config --system`

2) For **user** level configuration, use:
`git config --global`

3) For **Project** level configuration use:
`git config`

We'll set our Git `user.name` and `user.email` with `git config --global` so these are configured for all projects.

```
$ git config --global user.name "Martin Frigaard"
$ git config --global user.email "mjfrigaard@gmail.com"
```

We can check what we've configured with `git config --list`.

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

At the bottom of the output, we can see the changes.

```
user.name=Martin Frigaard
user.email=mjfrigaard@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
```

On our local machine, the `user.name` and `user.email` are in the `.gitconfig` file.

We can view this using:

```
$ cat .gitconfig
[user]
    name = Martin Frigaard
    email = mjfrigaard@gmail.com
```

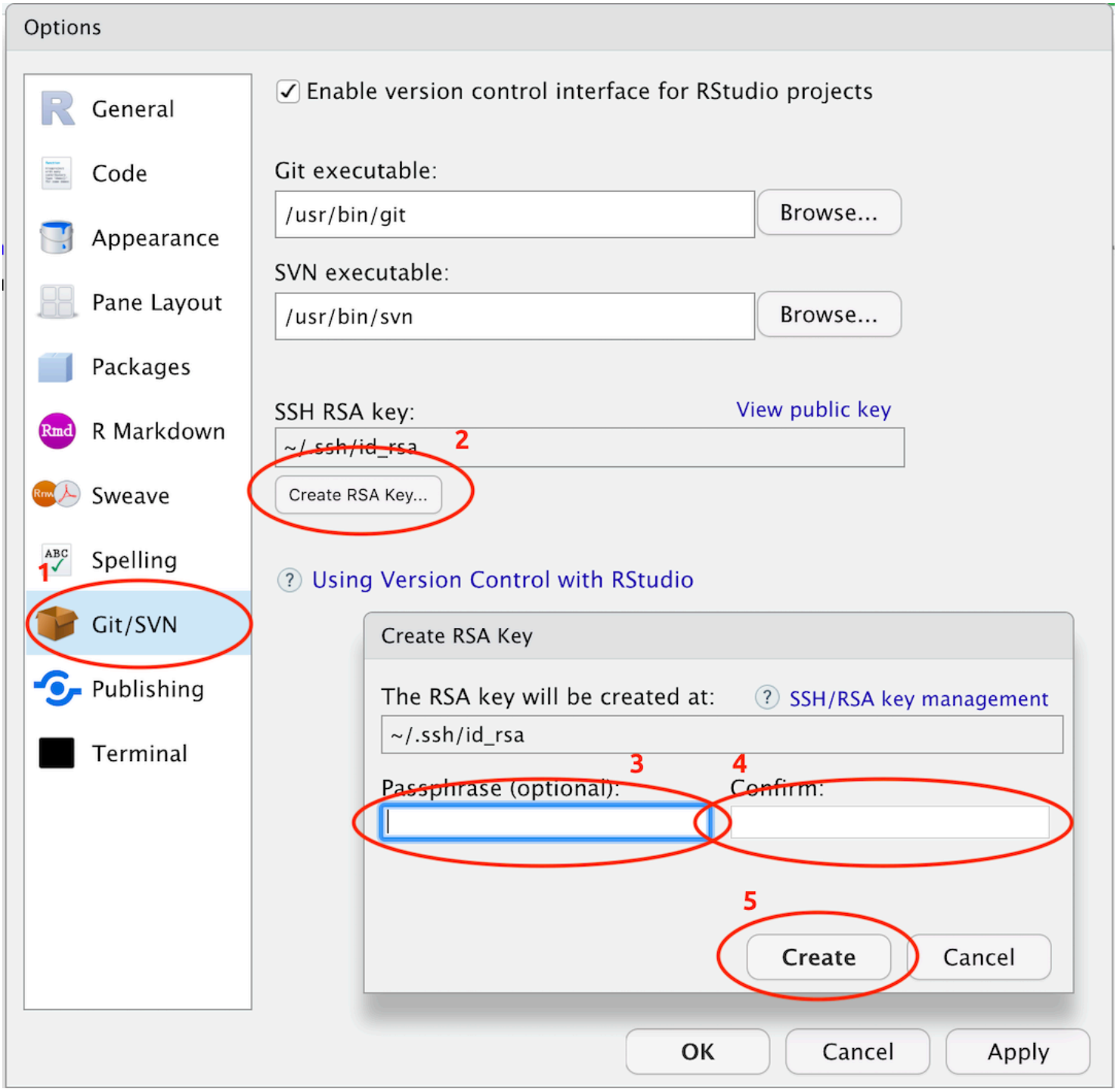
Synchronizing RStudio and Git/Github

Jenny Bryan has created the online resource Happy Git and GitHub for the useR has all in the information you will need for connecting RStudio and Git/Github. We echo a lot of this information below (with copious screenshots).

The first step is setting up the RSA Key and passphrase.

Go to *Tools* > *Global Options* > …

- 1. Click on *Git/SVN*
- 1. Then *Create RSA Key…*
- 3, 4, and 5. In the dialog box, enter a passphrase (and store it in a safe place), then click *Create*.



The result should look something like this:

```
Generating public/private rsa key pair.
Your identification has been saved in /home/rstudio-user/.ssh/id_rsa.
Your public key has been saved in /home/rstudio-user/.ssh/id_rsa.pub.
The key fingerprint is:
[redacted]
The key's randomart image is:
+---[RSA 2048]----+
|      .=0=..o0Oo  |
|     ..=. o ++oo  |
|      o .o oo.ooo.|
|       .  Xoo.Eo..|
|     [redacted]   |
|                  |
|                  |
+----[SHA256]-----+
```

Or like this on your local machine.

```
whoeveryouare ~ $ ssh-keygen -t rsa -b 2891 -C "USEFUL-COMMENT"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/username/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/username/.ssh/id_rsa.
Your public key has been saved in /Users/username/.ssh/id_rsa.pub.
The key fingerprint is:
SHA483:g!bB3r!sHg!bB3r!sHg!bB3r!sH USEFUL-COMMENT
The key's randomart image is:
+---[RSA 2891]----+
|      o+   . .   |
|     .=.o . +    |
|     ..= + +     |
|      .+* E      |
|     .= So =     |
|    .  +. = +    |
|   o.. = ..* .   |
|  o ++=.o =o.    |
| ..o.++o.=+.     |
+----[SHA483]-----+
```

Great! We need to go back to Terminal and store this `SSH` from RStudio.

Adding a key SSH in Terminal

In the Terminal, we enter the following commands.

```
$ eval "$(ssh-agent -s)"
```

The response tells us we're an `Agent`.

```
Agent pid 007
```

Now we want to add the *SSH RSA* to the keychain. There are three elements in this command: the `ssh-add`, the `-K`, and `~/.ssh/id_rsa`.

- The `ssh-add` is the command to add the *SSH RSA*
- The `-K` stores the passphrase we generated, and
- `~/.ssh/id_rsa` is the location of the *SSH RSA*.

```
$ ssh-add -k /home/rstudio-user/.ssh/id_rsa
```

Enter the passphrase, and then git should tell us the identity has been added.

```
Enter passphrase for /home/rstudio-user/.ssh/id_rsa:
Identity added: /home/rstudio-user/.ssh/id_rsa (/home/rstudio-user/.ssh/id_rsa)
```

Create the `.ssh/config` file

Most operating systems require a `config` file. We can do this using the Terminal commands above.

First, I move into the `.ssh/` directory.

```
$ cd /home/rstudio-user/.ssh
```

Then we create this `config` file with `touch`

```
$ touch config
# verify
$ ls
config        id_rsa        id_rsa.pub
```

We use `echo` to add the following text to the `config` file.

```
Host *
 AddKeysToAgent yes
 UseKeychain yes
```

Recall the >> will send the text to the `config` file.

```
# add text
$ echo "Host *
>  AddKeysToAgent yes
>  UseKeychain yes" >> config
```

Finally, we can check `config` with `cat`

```
# verify
```

```
$ cat config
Host *
 AddKeysToAgent yes
 UseKeychain yes
```

Great! Now I am all set up to use Git with RStudio. In the next section, we'll extend our Github skills by moving the contents of a local folder to Github.

More on Git and Github and data organization

Fortunately, many articles have come out in the last few years with excellent, practical advice on organizing data analysis projects. I recommend reading these before getting started (you'd be surprised at the cacophony of files a single project can produce). We've listed a few 'must-reads' below:

- the importance of using version control
- sharing data with collaborators
- how to name your files