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](https://git-scm.com/), 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.

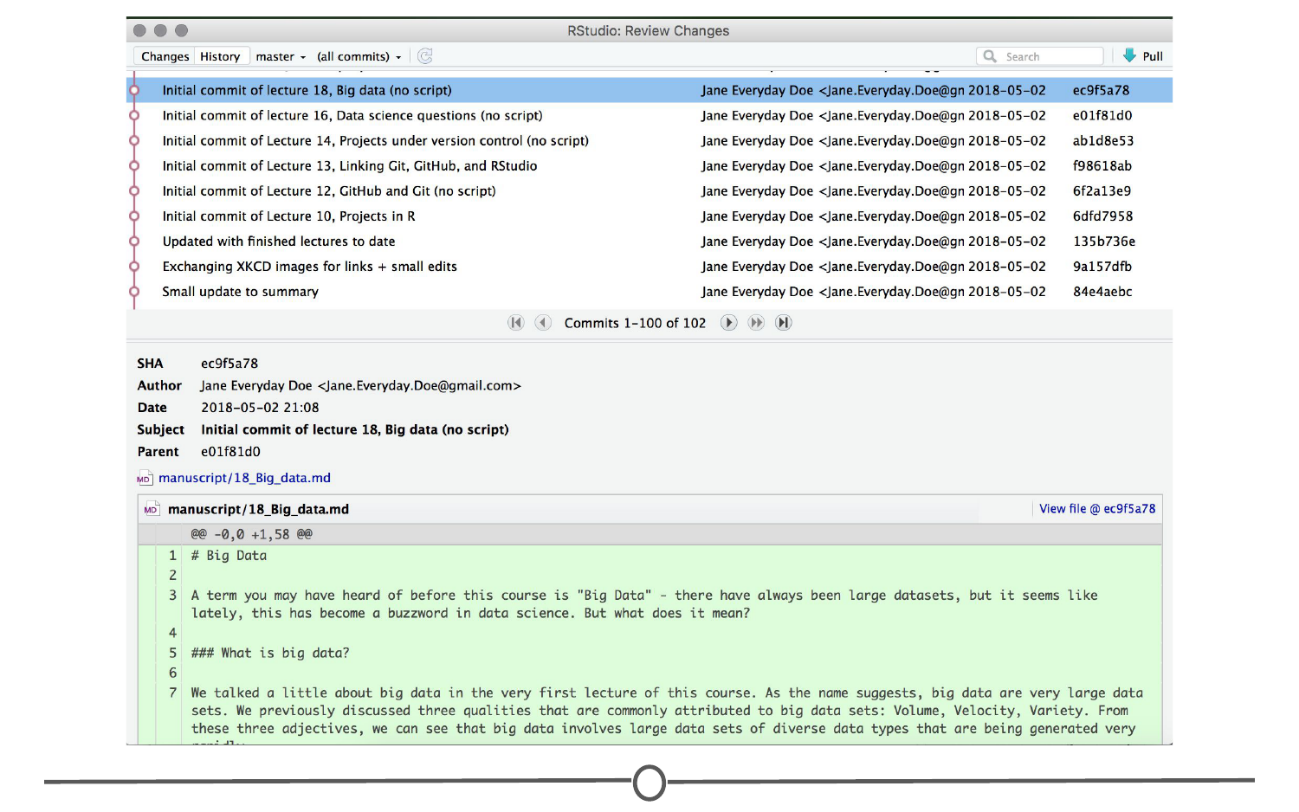
If you’ve ever worked collaboratively on a document before, [this comic](http://phdcomics.com/comics/archive.php?comicid=1531) from PHD Comics might resonate with you.

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!



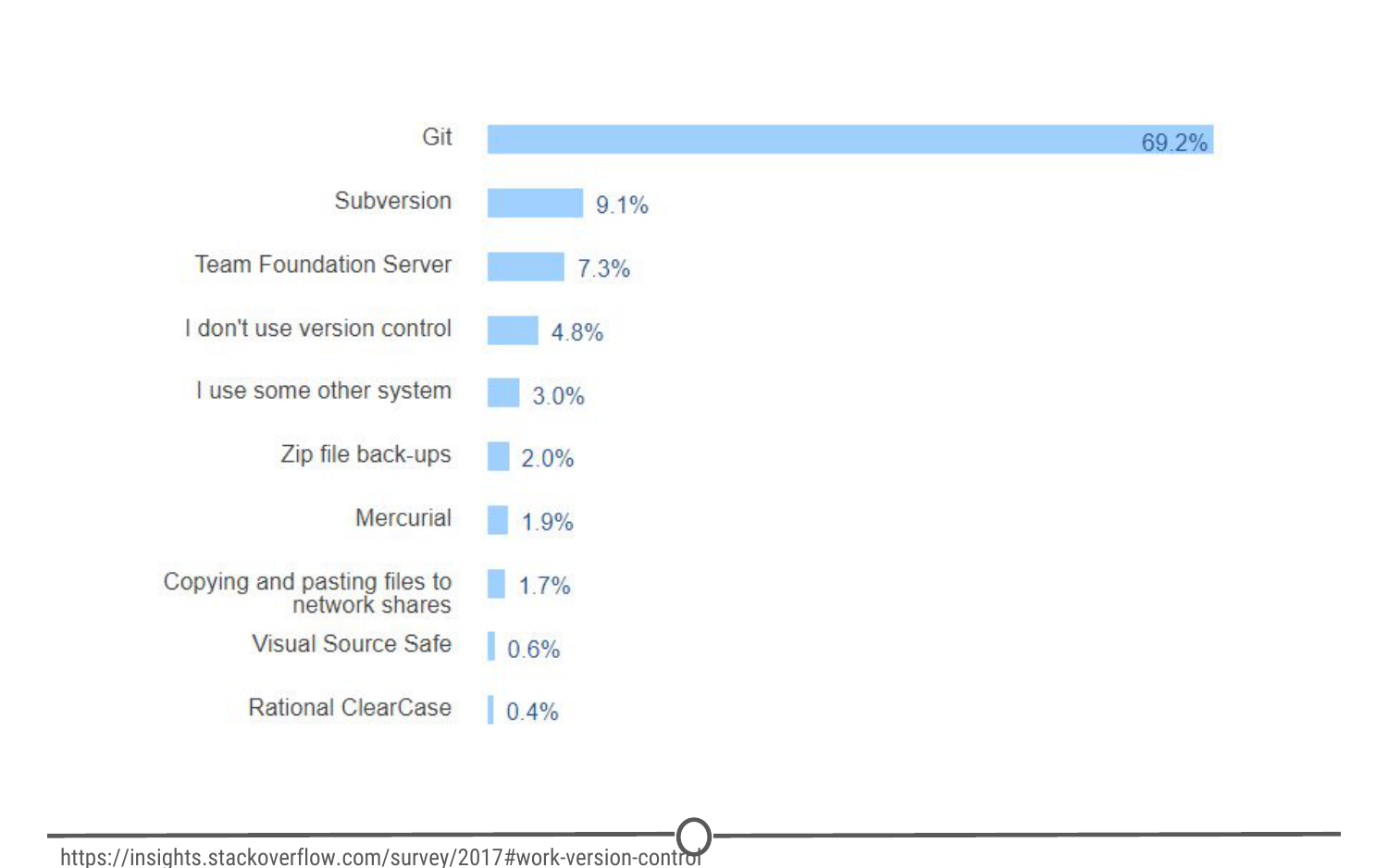
**An example of the version control history for the development of this course!**

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 is 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.

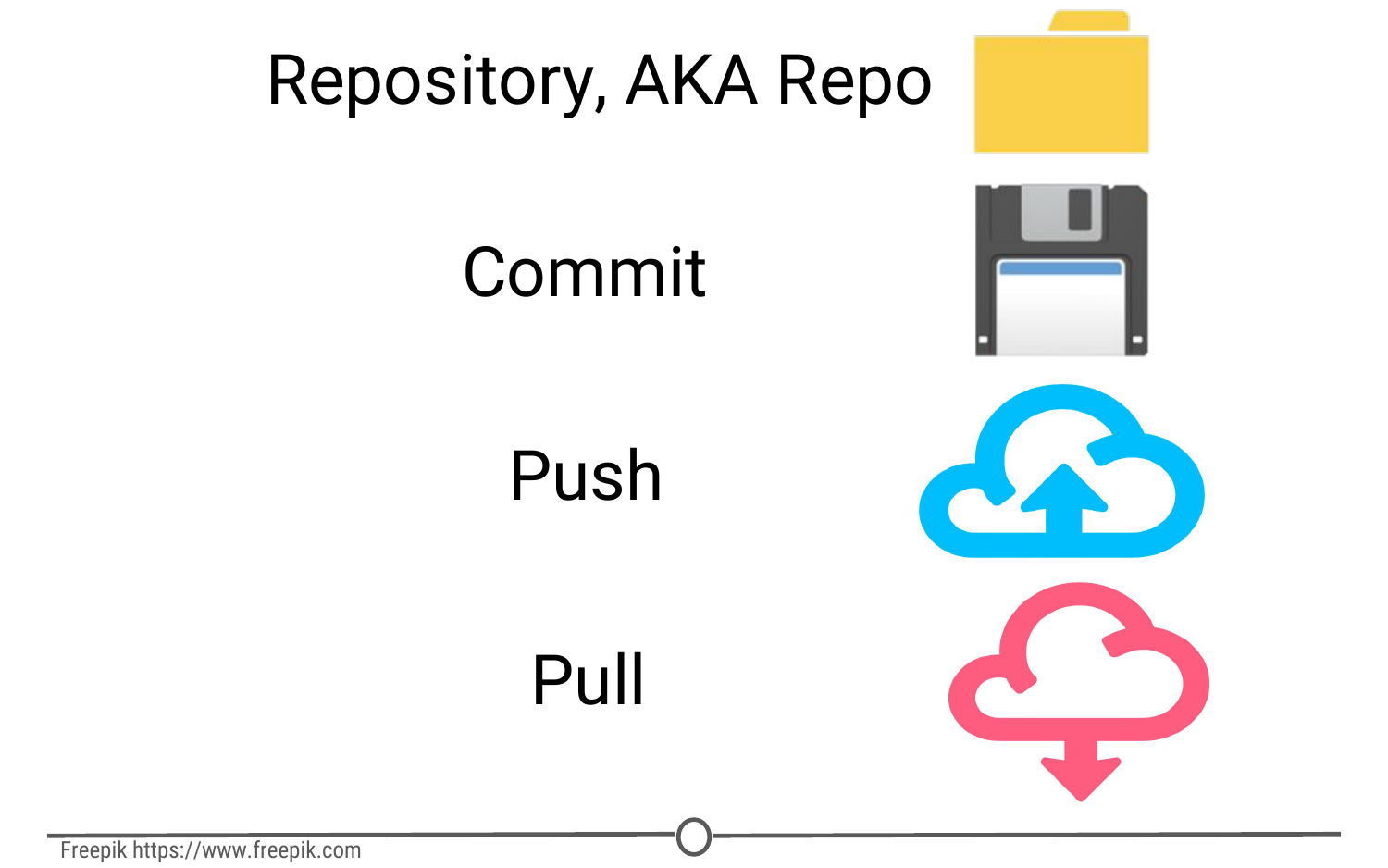
**Repository:** Equivalent to the project’s folder/directory - all of your version controlled files (and the recorded changes) are located in a repository. This is often shortened to **repo**. Repositories are what are hosted on GitHub and through this interface you can either keep your repositories 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**

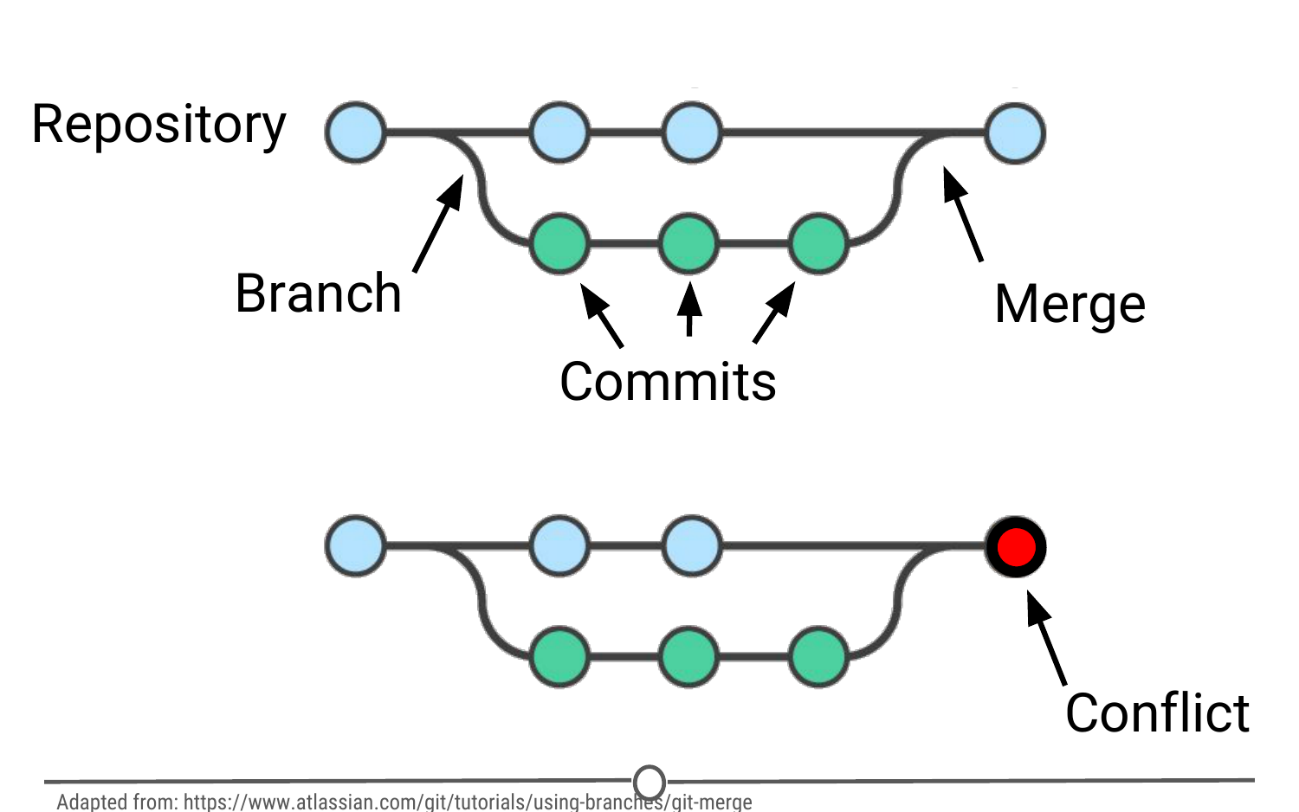
**Staging:** The act of preparing a file for a commit. For example, if since your last commit you have edited three files for completely different reasons, you don’t want to commit all of the changes in one go; your message on why you are making the commit and what has changed will be complicated since 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\*\*](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

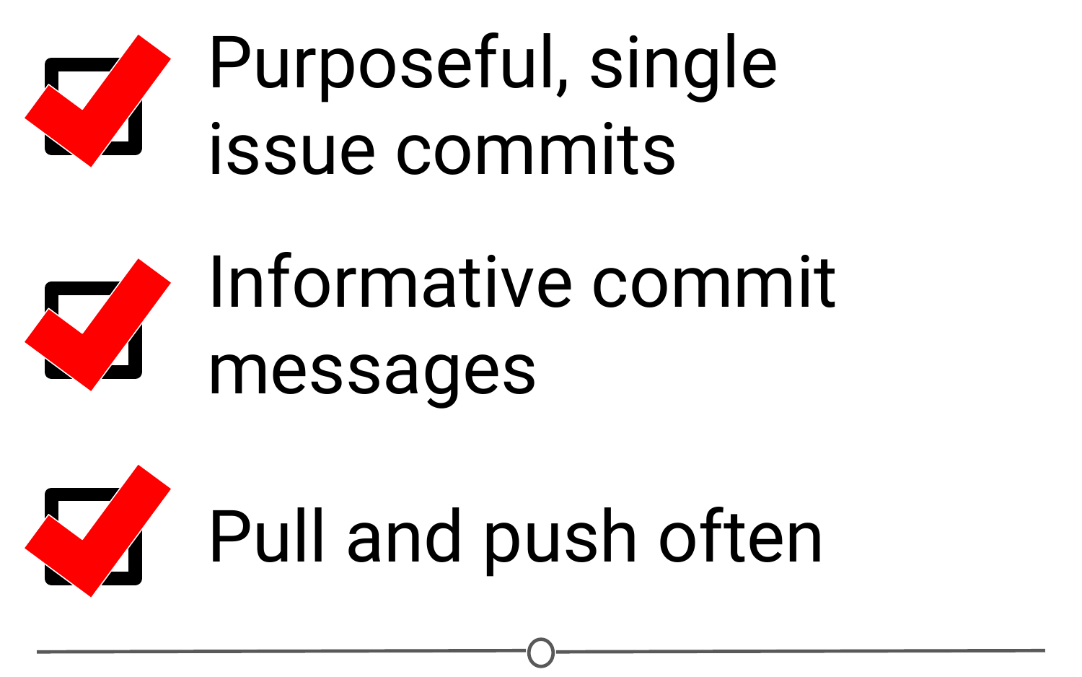
It can take some time to get used to working with version control software like Git, but there are a few things to keep in mind to help establish good habits that will help you out in the future.

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](https://xkcd.com/1296/) 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 horde 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](https://xkcd.com/1597/)! So let’s move on to the next lesson and get it installed!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Control de versiones

Ahora que tenemos un control sobre R, RStudio y los proyectos, hay algunas cosas más con las que queremos configurarlo antes de pasar a los otros cursos: comprender el control de versiones, instalar Git y vincular Git con RStudio. En esta lección, le brindaremos una comprensión básica del control de versiones.

¿Qué es el control de versiones?

Lo primero es lo primero: ¿Qué es el control de versiones? El control de versiones es un sistema que registra los cambios que se realizan en un archivo o un conjunto de archivos a lo largo del tiempo. A medida que realiza ediciones, el sistema de control de versiones toma instantáneas de sus archivos y los cambios, y luego guarda esas instantáneas para que pueda consultar o volver a versiones anteriores más adelante si es necesario. Si alguna vez ha utilizado la función "Seguimiento de cambios" en Microsoft Word, ha visto un tipo rudimentario de control de versiones, en el que se realiza un seguimiento de los cambios en un archivo, y puede optar por mantener esas ediciones o volver al original. formato. Los sistemas de control de versiones, como Git, son como un "Seguimiento de cambios" más sofisticado, en el sentido de que son mucho más potentes y son capaces de realizar un seguimiento meticuloso de los cambios sucesivos en muchos archivos, con muchas personas trabajando simultáneamente en los mismos grupos de archivos.

Si alguna vez ha trabajado en colaboración en un documento, este cómic de PHD Comics podría resonar en usted.

Con suerte, una vez que domine el software de control de versiones, Paper\_Final\_FINAL2\_actually\_FINAL.docx será una cosa del pasado para usted.

¿Cuáles son los beneficios de usar el control de versiones?

Como hemos visto en el ejemplo, sin el control de versiones, es posible que conserve varias copias muy similares de un archivo. Y esto podría ser peligroso: puede comenzar a editar la versión incorrecta, sin reconocer que el documento etiquetado como "FINAL" se ha editado más a "FINAL2", ¡y ahora todos los cambios nuevos se han aplicado al archivo incorrecto! Los sistemas de control de versiones ayudan a resolver este problema al mantener una única versión actualizada de cada archivo, con un registro de todas las versiones anteriores Y un registro de exactamente lo que cambió entre las versiones.

Lo que nos lleva al siguiente gran beneficio del control de versiones: mantiene un registro de todos los cambios realizados en los archivos. Esto puede ser de gran ayuda cuando está colaborando con muchas personas en los mismos archivos: el software de control de versiones realiza un seguimiento de quién, cuándo y por qué se realizaron esos cambios específicos. ¡Es como "Realizar un seguimiento de los cambios" hasta el extremo!

¡Un ejemplo del historial de control de versiones para el desarrollo de este curso!

¡Un ejemplo del historial de control de versiones para el desarrollo de este curso!

Este registro también es útil al desarrollar código, si después de un tiempo se da cuenta de que cometió un error y lo introdujo. Puede encontrar la última vez que editó ese fragmento de código en particular, ver los cambios que realizó y volver a ese código original e ininterrumpido, ¡dejando intacto todo lo demás que haya hecho mientras tanto!

Por último, cuando se trabaja con un grupo de personas en el mismo conjunto de archivos, el control de versiones es útil para garantizar que no realice cambios en los archivos que entren en conflicto con otros cambios. Si alguna vez ha compartido un documento con otra persona para editarlo, conoce la frustración de integrar sus ediciones con un documento que ha cambiado desde que envió el archivo original; ahora tiene dos versiones del mismo documento original. El control de versiones permite que varias personas trabajen en el mismo archivo y luego ayuda a fusionar todas las versiones del archivo y todas sus ediciones en un archivo cohesivo.

¿Qué es Git? ¿Por qué debería usarlo?

Git es un sistema de control de versiones de código abierto y gratuito. Fue desarrollado en 2005 y desde entonces se ha convertido en el sistema de control de versiones más utilizado. StackOverflow, que debería sonar familiar por nuestra lección Obtención de ayuda, encuestó a más de 60,000 encuestados sobre qué sistema de control de versiones usan y, como puede ver en la tabla a continuación, Git es, con mucho, el ganador.

Resultados de una encuesta de StackOverflow en la que se pregunta qué software de control de versiones utilizan sus encuestados

Resultados de una encuesta de StackOverflow en la que se pregunta qué software de control de versiones utilizan sus encuestados

Y a medida que se familiarice con Git y cómo funciona e interactúa con sus proyectos, comenzará a ver por qué ha alcanzado la cima de la popularidad. Uno de los principales beneficios de Git es que mantiene una copia local de su trabajo y revisiones, que luego puede editar sin conexión, y luego, una vez que regresa al servicio de Internet, puede sincronizar su copia del trabajo con todos sus nuevos ediciones y seguimiento de cambios en el repositorio principal en línea. Además, dado que todos los colaboradores de un proyecto tienen su propia copia local del código, todos pueden trabajar simultáneamente en sus propias partes del código, sin perturbar ese repositorio común.

Otro gran beneficio que definitivamente aprovecharemos es la facilidad con la que RStudio y Git interactúan entre sí. En la siguiente lección, trabajaremos para instalar Git y vincularlo con RStudio y crear una cuenta de GitHub.

¿Qué es GitHub?

GitHub es un interf en línea as para Git. Git es un software que se usa localmente en su computadora para registrar cambios. GitHub es un host para sus archivos y los registros de los cambios realizados. Puede pensar que es similar a DropBox: los archivos están en su computadora, pero también están alojados en línea y se puede acceder a ellos desde cualquier computadora. GitHub tiene el beneficio adicional de interactuar con Git para realizar un seguimiento de todas las versiones y cambios de sus archivos.

Vocabulario de control de versiones

Hay mucho vocabulario involucrado en el trabajo con Git y, a menudo, la comprensión de una palabra se basa en su comprensión de un concepto de Git diferente. Tómese un tiempo para familiarizarse con las palabras siguientes y léalas varias veces para ver cómo se relacionan los conceptos.

Repositorio: equivalente a la carpeta / directorio del proyecto; todos los archivos controlados por la versión (y los cambios registrados) se encuentran en un repositorio. Esto a menudo se abrevia como repositorio. Los repositorios son los que se alojan en GitHub y, a través de esta interfaz, puede mantener sus repositorios privados y compartirlos con colaboradores seleccionados, o puede hacerlos públicos: cualquiera puede ver sus archivos y su historial.

Confirmar: Confirmar es guardar sus ediciones y los cambios realizados. Una confirmación es como una instantánea de sus archivos: Git compara la versión anterior de todos sus archivos en el repositorio con la versión actual e identifica los que han cambiado desde entonces. Aquellos que no han cambiado, mantiene intacto ese archivo previamente almacenado. Aquellos que han cambiado, compara los archivos, registra los cambios y carga la nueva versión de su archivo. Hablaremos de esto en la siguiente sección, pero cuando envías un archivo, normalmente acompañas ese cambio de archivo con una pequeña nota sobre lo que cambiaste y por qué.

Cuando hablamos de sistemas de control de versiones, las confirmaciones están en el corazón de ellos. Si encuentra un error, revierte sus archivos a una confirmación anterior. Si desea ver qué ha cambiado en un archivo a lo largo del tiempo, compare las confirmaciones y observe los mensajes para ver por qué y quién.

Push: Actualización del repositorio con sus ediciones. Dado que Git implica realizar cambios localmente, debe poder compartir sus cambios con el repositorio común en línea. Empujar es enviar esos cambios confirmados a ese repositorio, por lo que ahora todos tienen acceso a sus ediciones.

Pull: Actualizando su versión local del repositorio a la versión actual, ya que otros pueden haber editado mientras tanto. Debido a que el repositorio compartido está alojado en línea y cualquiera de sus colaboradores (¡o incluso usted mismo en una computadora diferente!) Podría haber realizado cambios en los archivos y luego enviarlos al repositorio compartido, ¡está atrasado! Los archivos que tiene localmente en su computadora pueden estar desactualizados, por lo que debe verificar si está actualizado con el repositorio principal.

Analogías a estos conceptos

Analogías a estos conceptos

Puesta en escena: El acto de preparar un archivo para una confirmación. Por ejemplo, si desde su última confirmación ha editado tres archivos por razones completamente diferentes, no desea confirmar todos los cambios de una sola vez; su mensaje sobre por qué está realizando la confirmación y qué ha cambiado será complicado ya que se han cambiado tres archivos por diferentes razones. Entonces, en su lugar, puede organizar solo uno de los archivos y prepararlo para su confirmación. Una vez que haya confirmado ese archivo, puede preparar el segundo archivo y confirmarlo. Etcétera. La puesta en escena le permite separar los cambios de archivos en confirmaciones separadas. ¡Muy útil!

Para resumir estos términos de uso común hasta ahora y para probar si lo dominas, los archivos se alojan en un repositorio que se comparte en línea con los colaboradores. Extrae el contenido del repositorio para tener una copia local de los archivos que puede editar. Una vez que esté satisfecho con los cambios realizados en un archivo, organice el archivo y luego lo confirme. Envíe esta confirmación al repositorio compartido. Esto carga su nuevo archivo y todos los cambios y va acompañado de un mensaje que explica qué cambió, por qué y por quién.

Rama: Cuando un mismo archivo tiene dos copias simultáneas. Cuando está trabajando localmente y editando un archivo, ha creado una rama donde sus ediciones no se comparten con el repositorio principal (todavía), por lo que hay dos versiones del archivo: la versión a la que todos tienen acceso en el repositorio y su versión local editada del archivo. Hasta que envíe sus cambios y los vuelva a fusionar en el repositorio principal, estará trabajando en una rama. Después de un punto de bifurcación, el historial de versiones se divide en dos y realiza un seguimiento de los cambios independientes realizados tanto en el archivo original en el repositorio que otros pueden estar editando, como en el seguimiento de sus cambios en su bifurcación, y luego fusiona los archivos.

Fusionar: las ediciones independientes del mismo archivo se incorporan en un solo archivo unificado. Git identifica las ediciones independientes y las reúne en un solo archivo, con ambos conjuntos de ediciones incorporados. Pero, puede ver un problema potencial aquí: si ambas personas hicieron una edición en la misma oración que impide que una de las ediciones sea posible, tenemos un problema! Git reconoce esta disparidad (conflicto) y solicita ayuda al usuario para elegir qué edición conservar.

Conflicto: cuando varias personas realizan cambios en el mismo archivo y Git no puede fusionar las ediciones. Se le presenta la opción de intentar fusionar manualmente las ediciones o mantener una edición sobre la otra.

\*\* Una representación visual de estos conceptos, de https://www.atlassian.com/git/tutorials/using-branches/git-merge\*\*

\*\* Una representación visual de estos conceptos, de https://www.atlassian.com/git/tutorials/using-branches/git-merge\*\*

Clonar: hacer una copia de un repositorio de Git existente. Si acaba de ingresar a un proyecto que ha sido rastreado con control de versiones, debe clonar el repositorio para obtener acceso y crear una versión local de todos los archivos del repositorio y todos los cambios registrados.

Bifurcación: una copia personal de un repositorio que ha tomado de otra persona. Si alguien está trabajando en un proyecto interesante y quieres jugar con él, puedes bifurcar su repositorio y luego, cuando realizas cambios, las ediciones se registran en tu repositorio, no en el de ellos.

Mejores prácticas

Puede llevar algún tiempo acostumbrarse a trabajar con software de control de versiones como Git, pero hay algunas cosas que debe tener en cuenta para ayudar a establecer buenos hábitos que lo ayudarán en el futuro.

Una de esas cosas es realizar compromisos intencionados. Cada confirmación solo debe abordar un único problema. De esta manera, si necesita identificar cuándo cambió una determinada línea de código, solo hay un lugar donde buscar para identificar el cambio y puede ver fácilmente cómo revertir el código.

Del mismo modo, asegurarse de escribir mensajes informativos en cada confirmación es un hábito útil. Si cada mensaje es preciso en lo que se estaba cambiando, cualquiera puede examinar el archivo comprometido e identificar el propósito de su cambio. Además, si está buscando una edición específica que realizó en el pasado, puede escanear fácilmente todas sus confirmaciones para identificar los cambios relacionados con la edición deseada.

¡No querrás adquirir el mismo hábito que tiene XKCD!

Por último, tenga en cuenta la versión de los archivos en los que está trabajando. Compruebe con frecuencia que está actualizado con el repositorio actual tirando con frecuencia. Además, no guarde sus archivos editados; una vez que haya enviado sus archivos (¡y escrito ese mensaje útil!), Debe enviar esos cambios al repositorio común. Si ha terminado de editar una sección de código y planea pasar a un problema no relacionado, ¡debe compartir esa edición con sus colaboradores!

Un resumen de las principales prácticas recomendadas que debe tener en cuenta al trabajar con el control de versiones.

Un resumen de las principales prácticas recomendadas que debe tener en cuenta al trabajar con el control de versiones.

Resumen

Ahora que hemos cubierto qué es el control de versiones y algunos de los beneficios, debería poder comprender por qué tenemos tres lecciones completas dedicadas al control de versiones y su instalación. Analizamos qué son Git y GitHub, y luego cubrimos gran parte del vocabulario de uso común (¡y a veces confuso!) Inherente al trabajo de control de versiones. Luego repasamos rápidamente algunas de las mejores prácticas para usar Git, ¡pero la mejor manera de familiarizarnos con todo esto es usarlo! ¡Con suerte, sientes que tienes un mejor manejo del funcionamiento de Git que la gente de este cómic de XKCD! Así que pasemos a la siguiente lección e instálelo.