Control de versiones

1. Resumen

Wolfman y Drácula han sido contratados como analistas de datos por Outbreak Missions (un equipo de apoyo rápido para los servicios de respuesta a brotes) para investigar un brote de enfermedad. Quieren poder trabajar en el desarrollo de un [informe](https://choleraoutbreak.org/book-page/appendix-7-outline-outbreak-situation-report)de [situación](https://choleraoutbreak.org/book-page/appendix-7-outline-outbreak-situation-report)reproducible al mismo tiempo, pero han tenido problemas para hacerlo en el pasado.

* Si se turnan, cada uno pasará mucho tiempo esperando a que el otro termine, pero
* si trabajan en sus propias copias y los cambios de correo electrónico de ida y vuelta, las cosas se perderán, se sobrescribirán o se duplicarán.

¡Un colega sugiere usar [el control de versiones](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004668)para gestionar su trabajo!

**El control de versiones es mejor que enviar archivos de ida y vuelta:**

* Nada de lo que se guarda en el control de versiones se pierde, a menos que trabajes muy, muy duro en ello.
* Dado que todas las versiones antiguas de los archivos se guardan, siempre es posible volver atrás en el tiempo para ver exactamente *quién escribió qué*en un día en particular, o *qué versión de un programa se utilizó*para generar un conjunto particular de resultados.
* Cuando varias personas colaboran en el mismo proyecto, es posible pasar por alto o sobrescribir accidentalmente los cambios de alguien. El sistema de control de versiones *notifica automáticamente a los usuarios cada vez que hay un conflicto*entre el trabajo de una persona y el de otra.

**El control de versiones es el cuaderno de laboratorio del mundo digital: es**lo que los profesionales utilizan para hacer un seguimiento de lo que han hecho y colaborar con otras personas. Todos los grandes proyectos de desarrollo de software dependen de él, y la mayoría de los programadores también lo utilizan para sus pequeños trabajos. Y **no es solo para software**: libros, documentos, pequeños conjuntos de datos y *cualquier cosa que cambie con el tiempo o necesite ser compartida*puede y debe almacenarse en un sistema de control de versiones.

1. Control de versiones automatizado

Comenzaremos explorando cómo generalmente se nos presenta el control de versiones para realizar un seguimiento de lo que una persona hizo y cuándo. Incluso si no está colaborando con otras personas,.

¿Te parece innecesario tener varias versiones casi idénticas del mismo documento? Posiblemente sí. Pero este sistema de control de versiones abre la posibilidad de volver a una versión específica en caso de que haya borrado algo que cree que ahora es esencial.

Los sistemas de control de versiones comienzan con una versión básica del documento y luego registran los cambios que realice en cada paso del camino. Puedes pensar en ello como un registro de tu progreso: puedes rebobinar para comenzar en el documento base y reproducir cada cambio que hiciste, llegando eventualmente a tu versión más reciente.

Una vez que piense en los cambios como separados del documento en sí, puede pensar en "reprover" diferentes conjuntos de cambios en el documento base, lo que en última instancia resulta en diferentes versiones de ese documento.

Un sistema de control de versiones es una herramienta que realiza un seguimiento de estos cambios para nosotros, creando efectivamente diferentes versiones de nuestros archivos.

Las características clave de los sistemas de control de versiones son:

Mantenga el historial completo de un archivo e inspeccione un archivo a lo largo de su vida útil.

Etiqueta una versión en particular para que puedas volver a ellas fácilmente.

Control de versiones en R

Para archivos similares a código, como .R y .Rmd, no podemos usar Google Docs. El software y la estrategia para el seguimiento de cambios en un proyecto dependen del tipo de archivo.

* La herramienta de historial de versiones de Google Docs es un software de control de versiones optimizado para [archivos de texto no plano](https://github.com/epiverse-trace/git-rstudio-basics/discussions/11)individuales como archivos Word/Doc que funcionan en línea.
* Gites el software de control de versiones optimizado para [archivos de texto sin formato](https://en.wikipedia.org/wiki/Plain_text)que funciona sin conexión. (Leer: "Qué no poner bajo control de versiones" en [G. Wilson et al. 2017)](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510#sec014)

De los archivos de datos:

Podemos usar Git para rastrear los cambios en archivos de datos (como .csv y .tsv). Sin embargo, si consideramos los archivos de datos como archivos sin procesar, que no deberían cambiar con el tiempo, es posible que no necesitemos usar Git con ellos. Analizaremos esto en el capítulo sobre Ignorar elementos.

Además, si considera que su archivo de datos es grande con respecto a su computadora, puede optar por usar:

un servicio de alojamiento de archivos como Google Drive y el paquete de R {googlesheets4} para importar datos,

un sistema de control de versiones diferente como Git Large File Storage (LFS), o

un formato de datos diferente como el formato parquet usando el paquete de R {arrow}.

1. Configuración de Git

Un sistema de control de versiones es una herramienta que realiza un seguimiento de estos cambios para nosotros, creando efectivamente diferentes versiones de nuestros archivos. Cada registro de estos cambios se **llama**[**confirmación**](https://epiverse-trace.github.io/git-rstudio-basics/instructor/reference.html#commit). Cada uno guarda metadatos útiles sobre ellos. En lugar de *guardar copias*con diferentes nombres de archivo, estamos *haciendo confirmaciones*en el mismo archivo. Las confirmaciones consecutivas generan un historial lineal de cambios.

El historial completo de las confirmaciones para un proyecto en particular y sus metadatos forman un [repositorio](https://epiverse-trace.github.io/git-rstudio-basics/instructor/reference.html#repository). Los repositorios se pueden mantener sincronizados en diferentes ordenadores, lo que facilita la colaboración entre diferentes personas.

Antes de crear nuestro primer repositorio, necesitamos configurar Git. ¡Así que abramos RStudio y preséntate a Git!

Ir a R

En la consola de R, Sustituya este fragmento (condigo de r del enlace) por su nombre y **el correo electrónico asociado a su cuenta de GitHub**.

Por favor, use su propio nombre y dirección de correo electrónico en lugar de los de Drácula. Este nombre de usuario y correo electrónico se asociarán con su actividad Git posterior, lo que significa que cualquier cambio enviado a [GitHub](https://github.com/), [BitBucket](https://bitbucket.org/), [GitLab](https://gitlab.com/)u otro servidor host de Git después de esta lección incluirá esta información.

Para esta lección, interactuaremos con [GitHub](https://github.com/)y, por lo tanto, la dirección de correo electrónico utilizada debe ser la misma que la utilizada al configurar su cuenta de GitHub. Si le preocupa la privacidad, revise [las instrucciones de GitHub para mantener su dirección de correo electrónico privada](https://help.github.com/articles/keeping-your-email-address-private/).

Configuracion token git

Para interactuar con GitHub, necesitamos incluir credenciales en la solicitud. Vamos a configurar un tipo de credencial llamada *Token de Acceso Personal (PAT).* Necesitamos esto para demostrar que somos un usuario específico de GitHub, permitido para hacer lo que sea que se nos pida. ([Bryan, 2021](https://happygitwithr.com/https-pat.html))

**Primero, vamos a crear tu token.**

Haz esto con usethis::create\_github\_token(). Esta función debería redirigirte a GitHub en tu navegador. Una vez allí, compruebe todas las opciones de la siguiente figura.

Revisar: Revisa los ámbitos; los recomendados son "repositorio", "flujo de trabajo" y "usuario". Al usar usethis::create\_github\_token(), se preseleccionarán todos los ámbitos recomendados

Copia tu token. Guárdalo para el siguiente paso.

Configuración token

Como mencionamos antes, el historial completo de commits de un proyecto en particular y sus metadatos conforman un repositorio. (¡Crearemos uno en el próximo episodio!)

Una rama es una instantánea de una versión de un repositorio. En ese sentido, un repositorio puede tener más de una rama. ¡¿Qué?! ¿Cómo es posible? Lo veremos en próximos episodios.

Git (2.28+) permite configurar el nombre de la rama creada al inicializar cualquier repositorio nuevo. Drácula decide usar esta función para configurarla como principal, de modo que coincida con el servicio en la nube que usará.

Ramas

Los cambios en el archivo fuente están asociados con una "rama". Para los nuevos estudiantes en esta lección, es suficiente saber que las ramas existen, y esta lección usa una rama.  
De forma predeterminada, Git creará una rama llamada master al crear un nuevo repositorio con git init (como se explica en el próximo episodio). Este término evoca la práctica racista de la esclavitud humana, y la comunidad de desarrollo de software ha adoptado un lenguaje más inclusivo.

En 2020, la mayoría de los servicios de alojamiento de código de Git adoptaron la rama "main" como predeterminada. Por ejemplo, cualquier repositorio nuevo que se abra en GitHub y GitLab usa "main" de forma predeterminada. Sin embargo, Git aún no ha implementado el mismo cambio. Por lo tanto, los repositorios locales deben configurarse manualmente para que tengan el mismo nombre de rama "main" que la mayoría de los servicios en la nube.

En las versiones de Git anteriores a la 2.28, el cambio se puede realizar a nivel de repositorio individual. El comando para ello se encuentra en el próximo episodio. Tenga en cuenta que si este valor no está definido en la configuración local de Git, el valor "init.defaultBranch" se establece de forma predeterminada en "master".

Con la configuraicon hecha: ¡Necesitamos ejecutar estos comandos **solo una vez**! Git usará esta configuración para cada proyecto, en su cuenta de usuario, en este ordenador.

Puntos clave:

* Utilice el paquete [usethis](https://usethis.r-lib.org)para configurar un nombre de usuario, dirección de correo electrónico y otras preferencias una vez por máquina.
* Use usethis::use\_git\_config() to configure Git in RStudio.
* Use usethis::git\_sitrep() para verificar configuración

1. Creación de un repositorio.

Relacionaremos *las acciones*de control de versiones que podemos realizar con *comandos*específicos de *git verb*. Estos verbos registrarán sus cambios entre Gitasociados a su carpeta.

Para iniciar git, necesitamos tener cerrados proyectos de Rstudio.

Primero, vamos a crear una nueva carpeta de proyecto para nuestro trabajo. Crea un nuevo proyecto como quieras. Aquí vamos a usar las funciones del paquete [usethis](https://usethis.r-lib.org).

Si utiliza el escritorio de RStudio, el proyecto se abre en una nueva sesión. De lo contrario, se cambia el directorio de trabajo y el proyecto activo:

Luego le decimos a Git que convierta nuestra carpeta en un repositorio: un lugar donde Git puede almacenar versiones de nuestros archivos: