Que es Git?

Git es un sistema de control de versiones distribuido, diseñado por Linus Torvalds. Está pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Git está optimizado para guardar cambios de forma incremental.

Permite contar con un historial, regresar a una versión anterior y agregar funcionalidades.

Lleva un registro de los cambios que otras personas realicen en los archivos.

Git fue diseñado para operar en un entorno Linux. Actualmente, es multiplataforma, es decir, es compatible con Linux, MacOS y Windows. En la máquina local se encuentra Git, se utiliza bajo la terminal o línea de comandos y tiene comandos como merge, pull, add, commit y rebase, entre otros.

Para qué proyectos sirve Git

Con Git se obtiene una mayor eficiencia usando archivos de texto plano, ya que con archivos binarios no puede guardar solo los cambios, sino que debe volver a grabar el archivo completo ante cada modificación, por mínima que sea, lo que hace que incremente demasiado el tamaño del repositorio.

“Guardar archivos binarios en el repositorio de Git no es una buena práctica, únicamente deberían guardarse archivos pequeños (como logos) que no sufran casi modificaciones durante la vida del proyecto. Los binarios deben guardarse en un CDN”.

Características de Git

Git almacena la información como un conjunto de archivos.

No existen cambios, corrupción en archivos o cualquier alteración sin que Git lo sepa.

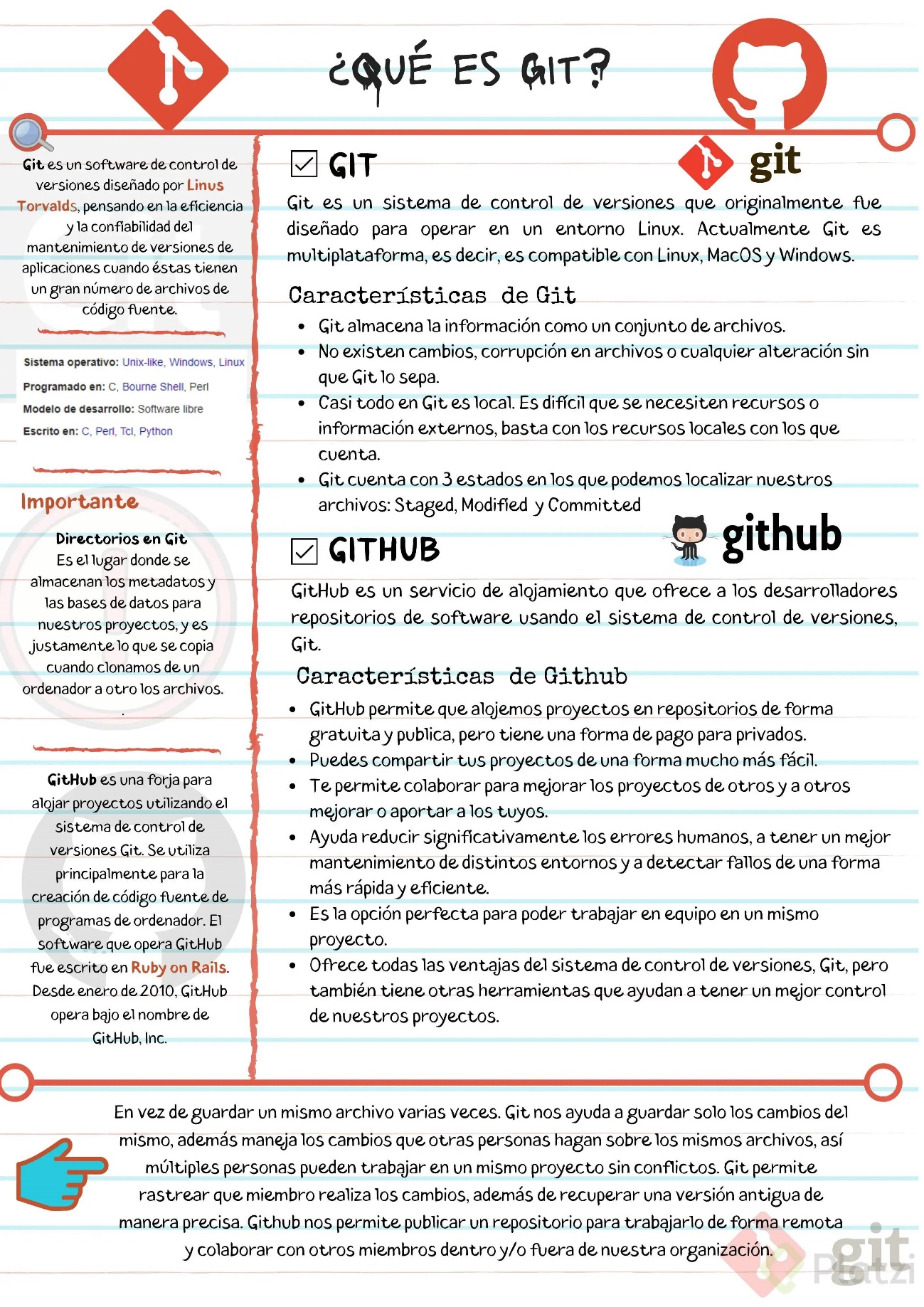
Casi todo en Git es local. Es difícil que se necesiten recursos o información externos, basta con los recursos locales con los que cuenta.

Git cuenta con 3 estados en los que es posible localizar archivos: Staged, Modified y Committed.

¿Qué es un sistema de control de versiones?

El SCV o VCS (por sus siglas en inglés) es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas llevar el historial del ciclo de vida de un proyecto, comparar cambios a lo largo del tiempo, ver quién los realizó o revertir el proyecto entero a un estado anterior.

Cualquier tipo de archivo que se encuentre en un ordenador puede ponerse bajo control de versiones.



¿Qué es Github?

Github es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se emplea principalmente para la creación de código fuente de programas de computadora.

Puede considerarse a Github como la red social de código para los programadores y en muchos casos es visto como un curriculum vitae, pues aquí se guarda el portafolio de proyectos de programación.

Características de Github

GitHub permite alojar proyectos en repositorios de forma gratuita y pública, pero tiene una forma de pago para privados.

Puedes compartir fácilmente tus proyectos.

Permite colaborar para mejorar los proyectos de otros y a otros mejorar o aportar a los tuyos.

Ayuda a reducir significativamente los errores humanos, a tener un mejor mantenimiento de distintos entornos y a detectar fallos de una forma más rápida y eficiente.

Es la opción perfecta para poder trabajar en equipo en un mismo proyecto.

Ofrece todas las ventajas del sistema de control de versiones Git, pero también tiene otras herramientas que ayudan a tener un mejor control de los proyectos.

Editores de código, archivos binarios y de texto plano

Un editor de código o IDE es una herramienta que nos brinda muchas ayudas para escribir código, algo así como un bloc de notas muy avanzado. Los editores más populares son VSCode, Sublime Text y Atom, pero no es obligatorio usar alguno de estos para programar. Conoce más a fondo sobre qué es un IDE.

Tipos de archivos y sus diferencias:

Archivos de Texto (.txt): Texto plano normal y sin nada especial. Lo vemos igual sin importar dónde lo abramos, ya sea con el bloc de notas o con editores de texto avanzados.

Archivos RTF (.rtf): Podemos guardar texto con diferentes tamaños, estilos y colores. Pero si lo abrimos desde un editor de código, vamos a ver que es mucho más complejo que solo el texto plano. Esto es porque debe guardar todos los estilos del texto y, para esto, usa un código especial un poco difícil de entender y muy diferente a los textos con estilos especiales al que estamos acostumbrados.

Archivos de Word (.docx): Podemos guardar imágenes y texto con diferentes tamaños, estilos o colores. Al abrirlo desde un editor de código podemos ver que es código binario, muy difícil de entender y muy diferente al texto al que estamos acostumbrados. Esto es porque Word está optimizado para entender este código especial y representarlo gráficamente.

Recuerda que debes habilitar la opción de ver la extensión de los archivos, de lo contrario, solo podrás ver su nombre. La forma de hacerlo en Windows es Vista > Mostrar u ocultar > Extensiones de nombre de archivo.

Conceptos importantes de Git

Bug: Error en el código

Repository: Donde se almacena todo el proyecto, el cual puede vivir tanto en local como en remoto. El repositorio guarda un historial de versiones y, más importante, de la relación de cada versión con la anterior para que pueda hacerse el árbol de versiones con las diferentes ramas.

Fork: Si en algún momento queremos contribuir al proyecto de otra persona, o si queremos utilizar el proyecto de otro como el punto de partida del nuestro. Esto se conoce como “fork”.

Clone: Una vez se decide hacer un fork , hasta ese momento sólo existe en GitHub. Para poder trabajar en el proyecto, toca clonar el repositorio elegido al computador personal.

Branch: Es una bifurcación del proyecto que se está realizando para anexar una nueva funcionalidad o corregir un bug.

Master: Rama donde se almacena la última versión estable del proyecto que se está realizando. La rama master es la que está en producción en cada momento (o casi) y debería estar libre de bugs. Así, si esta rama está en producción, sirve como referente para hacer nuevas funcionalidades y/o arreglar bugs de última hora.

Commit: consiste en subir cosas a la versión local del repositorio. De esta manera se puede trabajar en la rama de forma local sin tener que modificar ninguna versión en remoto ni tener que tener la última versión remota, cosa muy útil en grandes desarrollos trabajados por varias personas.

Push: Consiste en enviar todo lo que se ha confirmado con un commit al repositorio remoto. Aquí es donde se une nuestro trabajo con el de los demás.

Checkout: Acción de descargarse una rama del repositorio GIT local (sí, GIT tiene su propio repositorio en local para poder ir haciendo commits) o remoto.

Fetch: Actualiza el repositorio local bajando datos del repositorio remoto al repositorio local sin actualizarlo, es decir, se guarda una copia del repositorio remoto en el local.

Merge: La acción de merge es la continuación natural del fetch. El merge permite unir la copia del repositorio remoto con tu repositorio local, mezclando los diferentes códigos.

Pull: Consiste en la unión del fetch y del merge, esto es, recoge la información del repositorio remoto y luego mezcla el trabajo en local con esta.

Diff: Se utiliza para mostrar los cambios entre dos versiones del mismo archivo.

**Introducción a la terminal y línea de comandos**

La línea de comandos nos permite interactuar con nuestro computador sin necesidad de utilizar una interfaz gráfica. Sin embargo, los computadores emplean distintos sistemas de archivos y manejan diferentes comandos, dependiendo del sistema operativo que utilicen.

**Diferencias entre la estructura de archivos de Windows, Mac o Linux.**

La ruta principal en Windows es C:\, en UNIX es solo /.

Windows no hace diferencia entre mayúsculas y minúsculas pero UNIX sí.

Recuerda que GitBash usa la ruta /c para dirigirse a C:\ (o /d para dirigirse a D:\) en Windows. Por lo tanto, la ruta del usuario con el que estás trabajando es /c/Users/Nombre de tu usuario

**Comandos básicos en la terminal:**

pwd: Nos muestra la ruta de carpetas en la que te encuentras ahora mismo.

mkdir: Nos permite crear carpetas (por ejemplo, mkdir Carpeta-Importante).

touch: Nos permite crear archivos (por ejemplo, touch archivo.txt).

rm: Nos permite borrar un archivo o carpeta (por ejemplo, rm archivo.txt). Mucho cuidado con este comando, puedes borrar todo tu disco duro.

cat: Ver el contenido de un archivo (por ejemplo, cat nombre-archivo.txt).

ls: Nos permite cambiar ver los archivos de la carpeta donde estamos ahora mismo. Podemos usar uno o más argumentos para ver más información sobre estos archivos (los argumentos pueden ser -- + el nombre del argumento o - + una sola letra o shortcut por cada argumento).

- ls -a: Mostrar todos los archivos, incluso los ocultos.

- ls -l: Ver todos los archivos como una lista.

cd: Nos permite navegar entre carpetas.

- cd /: Ir a la ruta principal:

- cd o cd ~: Ir a la ruta de tu usuario

- cd carpeta/subcarpeta: Navegar a una ruta dentro de la carpeta donde estamos ahora mismo.

- cd .. (cd + dos puntos): Regresar una carpeta hacia atrás.

- Si quieres referirte al directorio en el que te encuentras ahora mismo puedes usar cd . (cd + un punto).

history: Ver los últimos comandos que ejecutamos y un número especial con el que podemos repetir su ejecución.

! + número: Ejecutar algún comando con el número que nos muestra el comando history (por ejemplo, !72).

clear: Para limpiar la terminal. También podemos usar los atajos de teclado Ctrl + L o Command + L.

Todos estos comandos tiene una función de autocompletado, o sea, puedes escribir la primera parte y presionar la tecla Tab para que la terminal nos muestre todas las posibles carpetas o comandos que podemos ejecutar. Si presionas la tecla Arriba puedes ver el último comando que ejecutamos.

Recuerda que podemos descubrir todos los argumentos de un comando con el argumento --help (por ejemplo, cat --help).