Git

Jesus Rivera

15 de marzo de 2022

Control de Versiones

Se llama control de versiones a los métodos y herramientas disponibles para controlar todo lo referente a los cambios en el tiempo de un archivo. Difícilmente un archivo de código o un documento de texto está terminado con la primera escritura; necesita cambios o revisiones para corregir errores o modificar su contenido. A medida que el documento evoluciona, podemos mantener un historial de cambios o dejar que evolucione sin guardar ningún registro. El control de versiones es un método estándar para mantener esta historia que puede ser útil en el futuro. En documentos sencillos como un ensayo o un pequeño programa los cambios no son algo esencial, pero en la escritura de un libro o un programa con centenares de páginas o lineas de código es esencial mantener la historia por ejemplo para poder corregir errores.

¿ Por qué utilizar un control de versiones (distribuido)?

Un control de versiones nos permite atacar los siguientes problemas

- Trabajar en los mismos archivos desde varios lugares, por ejemplo, desde nuestra laptop, nuestra computadora del trabajo o nuestra computadora de escritorio.
- Nos permite rastrear los cambios en nuestros archivos, así podemos volver a alguna versión anterior.
- Se pueden fijar versiones especificas de nuestros archivos, por ejemplo las distintas versiones de un articulo.

- Nos permite trabajar de forma no lineal, haciendo cambios en paralelo.
- Nos permite trabajar con colegas, al poder compartir nuestro trabajo o para colaborar sobre un mismo documento.

Uno de los controladores de versiones más usados en la actualidad es *git*
.

¿ Qué es Git?

Git es un proyecto de código abierto para el control de versiones distribuido creado en el 2005 por Linus Torvalds. Git surge como alternativa al controlador de versiones *BitKeeper* utilizado por la comunidad de desarrollo del kernel de linux hasta ese año, al venirse abajo la relación entre estos. Al crear este control de version se buscaba un sistema que fuera **distribuido**, **veloz**, **de diseño sencillo**, con soporte para **desarrollo no lineal** y con la capacidad de **manejar grandes proyectos** como el Kernel de Linux. Desde su nacimiento en el 2005, Git ha evolucionado para ser un sistema de gran popularidad con gran rendimiento, seguro y flexible.

¿ Como funciona Git?

A diferencia de otros sistemas de control de versiones git, no guarda diferencias de los archivos, si no, copias instantáneas. Esto es cada vez que se confirma un cambio o se guarda el estado de nuestro proyecto en Git, se toma una 'fotografía' del estado actual de *todos* los archivos y guarda esta 'fotografía'. Para ser eficiente los archivos sin modificar no se almacenan de nuevos, solo se guarda una referencia al archivo idéntico anterior.

Gran parte de las operaciones de git trabajan de forma local, es decir solo requieren los recursos y archivos de nuestra computadora. Esto significa que podemos trabajar sin necesidad de estar conectados a a alguna VPN, además de tener a nuestro alcance toda la historia de nuestros archivos.

Los tres estados de git

Basicamente git maneja los archivos en tres estados

- Modificado, es el estado en el que se encuentra nuestros archivos modificados pero aún no han sido confirmados.

- Confirmado, en esta estado nuestro datos ya se han almacenado de forma segura en nuestra base de datos local.
- Preparado, en este estado nuestros archivos modificados se encuentran marcados para la proxima confirmación.

Un proyecto de git cuenta con tres secciones,

- El directorio de trabajo, es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que su uso o modificación.
- El área de preparación,es un archivo que almacena información acerca de lo que va a ir en tu próxima confirmación.
- El directorio de git, es donde se almacenan los meta-datos y la base de datos de objetos para el proyecto. Es la parte más importante de Git, y es lo que se copia cuando se clona un repositorio desde otra computadora.

El flujo de trabajo básico en git es el siguiente

1. Se modifican los archivos en el directorio de trabajo.
2. Se agregan los archivos al área de preparación.
3. Se confirman los cambios.

Github, Gitlab

Aun que git resuelve varios problemas al ser un controlador de versiones, sigue siendo un sistema que corre de forma local en nuestra computadora, una solución son plataformas como github, gitlab o bitbucket, que nos permiten tener nuestros repositorios de git en linea, de esta forma podemos compartirlos y trabajar con otras personas.

Otras de las ventajas que ofrecen estas plataformas son crear proyectos, agregar a nuestros colaboradores a nuestros proyectos, asignar tareas, permisos, etc.

Utilizando git

Esta sección explica como se utiliza git de forma basica para poder utilizar los comandos es necesario instalar git, lo cual se puede ver en documentación de git.

Repositorios de git

Un repositorio o repo es en un lugar donde se guardan nuestros archivos estos pueden ser locales, es decir, se guardan en nuestra computadora, o remotos, es decir, se guardan en algún servidor por ejemplo en GitLab.

Para crear un repositorio local es necesario ejecutar el comando

```
$ git init
```

Para copiar un repositorio remoto es necesario ejecutar el comando

```
$ git clone https://gitlab.com/hunahpu18/git.git:
```

Registrando cambios

Una vez que hemos creado o clonado un repositorio y hemos modificado o creado nuestros archivos, es necesario agregarlos a el área de preparación usando el comando

```
$ git add <nombre del archivo>
```

Una vez que hemos agregado todos los archivos al área de preparación debemos aceptar los cambios, esto se hace usando el comando

```
$ git commit -m "Mensaje que describe el commit:"
```

Si estamos trabajando con repos remotas entonces es necesario enviar los cambios al servidor para hacer esto usamos el comando

```
$ git push origin master
```

Actualizando repositorios

Si estamos trabajando con repos remotas entonces es necesario actualizar los cambios que pudieron hacer nuestros colaboradores o que realizamos en otra computadora para esto podemos actualizar nuestro repositorio local usando el comando

```
$ git pull
```

Ignorando Archivos

Dentro de git podemos ignorar archivos, para ello es necesario crear un archivo de nombre *.gitignore* dentro de nuestro repo y dentro de el agregar las extensiones o nombres de archivos que queremos ignorar por ejemplo en el caso de latex, queremos ignorar los archivos auxiliares

```
## .gitignore
*.aux
*.log
*.toc
*.out
*.synctex.gz
```

Referencias

[1] *Become a git guru*. URL: https://www.atlassian.com/git/tutorials. (revisado: 15.03.2022).

[2] Sam Livingston-Gray. *Think Like (a) Git*. URL: http://think-like-a-git.net/. (revisado: 15.03.2022).

[3] Jesus Rivera. *Git*. URL: https://gitlab.com/hunahpu18/git/-/blob/master/.gitignore. (revisado: 15.03.2022).

[4] Ben Straub Scott Chacon. *Pro Git*. URL: https://git-scm.com/book/en/v2. (revisado: 15.03.2022).

[5] *Using Git and GitHub*. URL: https://www.overleaf.com/learn/how-to/Using_Git_and_GitHub. (revisado: 15.03.2022).