Exposición Empty

Sistemas de control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o SVC (del inglés System Version Control).

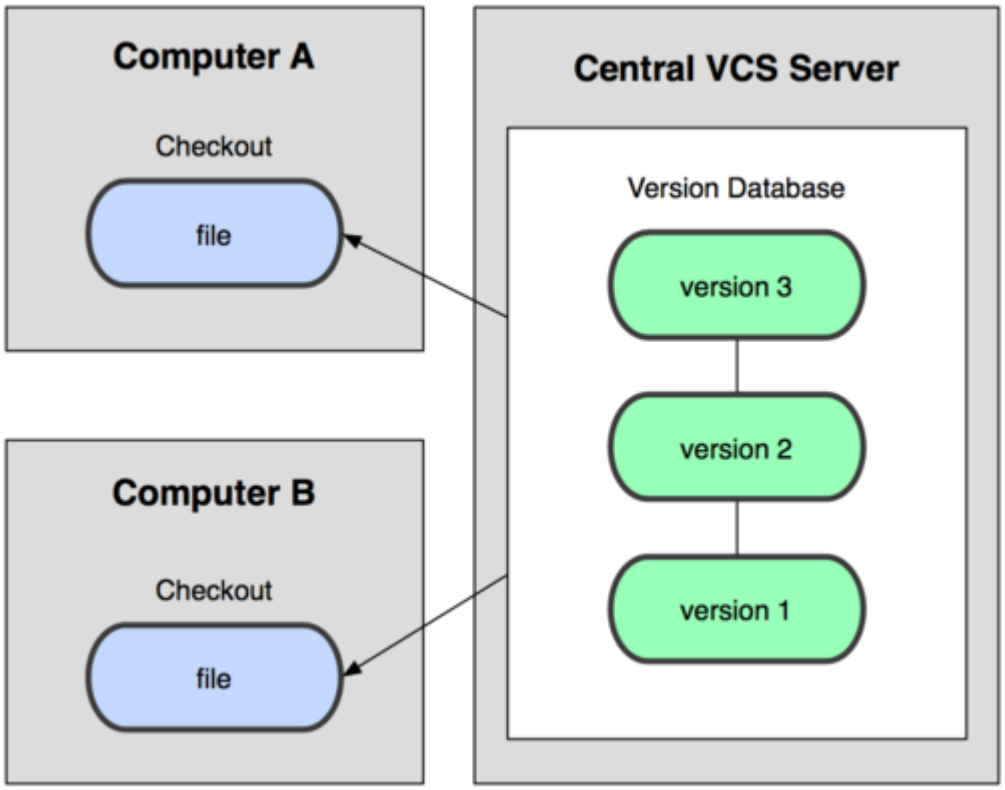
Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico). Ejemplos de este tipo de herramientas son entre otros: CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar , Plastic SCM, Git, Mercurial, Perforce.

Centralizados

Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS y Subversion.

Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para afrontar este problema los programadores desarrollaron hace tiempo VCS locales que contenían una simple base de datos, en la que se llevaba el registro de todos los cambios realizados a los archivos.



Historia

Uno de los sistemas más antiguos es RCS (Revision Control System), que se escribió en C y se introdujo en 1982. RCS forma parte de la primera generación de VCS. Es un conjunto de comandos de Unix que mantiene conjuntos de parches (lo que significa diferencias entre archivos) y hace posible recuperar versiones anteriores de archivos. El VCS local no es lo suficientemente bueno para los equipos. Esto nos lleva a la próxima generación de VCS que se llama Sistemas de control de versiones centralizado. Herramientas como CVS (introducida en 1990), Perforce (introducida en 1995) y Subversion (introducida en 2000) son una parte importante de esta generación. En estos sistemas tenemos un servidor central que contiene todos los archivos versionados y los desarrolladores pueden conectarse a ese servidor. Este enfoque es mejor. El administrador

puede monitorear a las personas y los miembros del equipo pueden ver lo que han hecho otros miembros. Pero tiene una desventaja seria que es el punto único de falla. Si el servidor deja de funcionar o el disco duro en el que se encuentra la base de datos central se corrompe, nos metemos en problemas. Para lidiar con este problema, intervienen los sistemas de control de versiones distribuidos. Bazaar, Mercurial, Git y Darcs son las herramientas más notables de esta generación que se introdujeron alrededor de 2003 a 2005. En los VCS distribuidos, cada usuario refleja el repositorio, incluido su historial completo. . Dicho esto, significa que cada clon es una copia de seguridad completa de todos los datos. Como sabemos, Linux Kernel es uno de los mayores proyectos de código abierto en el mundo de la ingeniería de software. En 2002, el proyecto Linux Kernel comenzó a usar un VCS llamado BitKeeper. En 2005, la comunidad de desarrolladores de Linux Kernel y BitKeeper no estaban en buenos términos y se revocó su licencia. Fue así como la comunidad de desarrolladores de Linux Kernel empezó a desarrollar su propio sistema de control de versiones basado en su experiencia con BitKeeper y así nació Git

Comandos

```
git clone <https://link-con-nombre-del-repositorio>
```

Git clone es un comando para descargarte el código fuente existente desde un repositorio remoto (como Github, por ejemplo). En otras palabras, Git clone básicamente realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en tu ordenador.

```
git branch <nombre-de-la-rama>
```

Las ramas (branch) son altamente importantes en el mundo de Git. Usando ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crearlas, listarlas y eliminarlas.

```
git checkout <nombre-de-la-rama>
```

Este es también uno de los comandos más utilizados en Git. Para trabajar en una rama, primero tienes que cambiarte a ella. Usaremos git checkout principalmente para cambiarte de una rama a otra. También lo podemos usar para chequear archivos y commits.

```
git status
```

El comando de git status nos da toda la información necesaria sobre la rama actual.

```
git add <archivo>
```

Cuando creamos, modificamos o eliminamos un archivo, estos cambios suceden en local y no se incluirán en el siguiente commit (a menos que cambiemos la configuración).

```
git commit -m "mensaje de confirmación"
```

Este sea quizás el comando más utilizado de Git. Una vez que se llega a cierto punto en el desarrollo, queremos guardar nuestros cambios (quizás después de una tarea o asunto específico).

```
git push <nombre-remoto> <nombre-de-tu-rama>
```

Después de haber confirmado tus cambios, el siguiente paso que quieres dar es enviar tus cambios al servidor remoto. Git push envía tus commits al repositorio remoto.

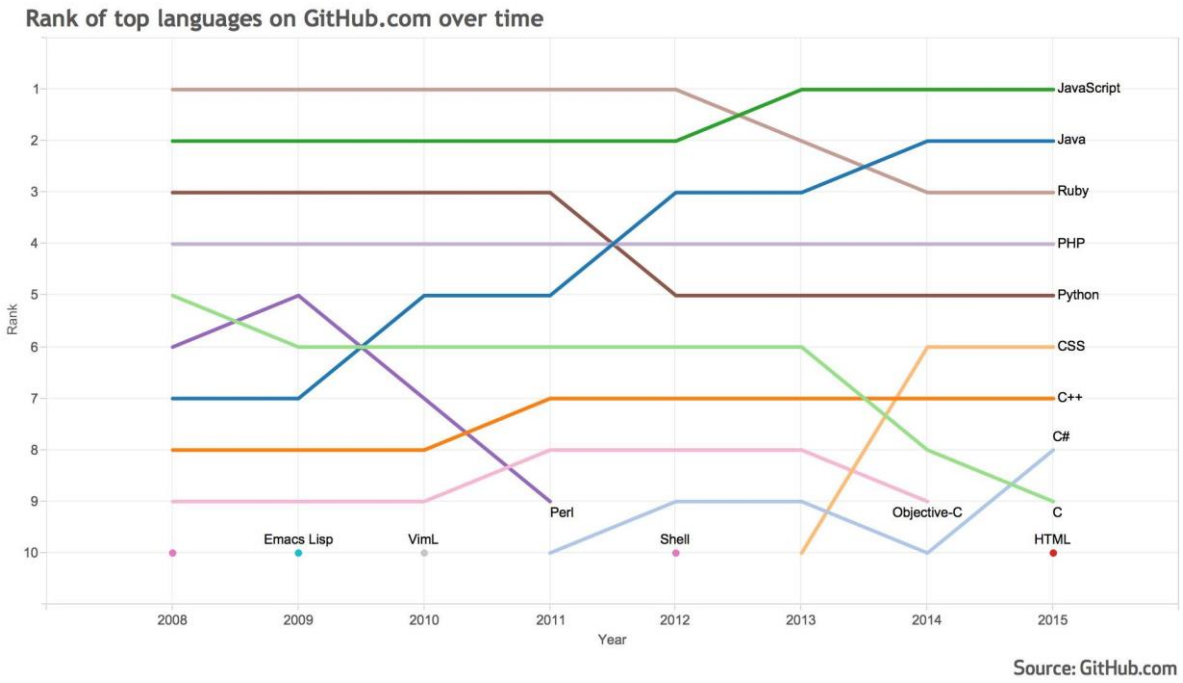
```
git pull <nombre-remoto>
```

El comando git pull se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del git fetch y del git merge lo cual significa que cundo usemos el git pull recogeremos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge).

```
git revert 3321844
```

A veces, necesitaremos deshacer los cambios que hemos hecho. Hay varias maneras para deshacer nuestros cambios en local y/o en remoto (dependiendo de lo que necesitemos), pero necesitaremos utilizar cuidadosamente estos comandos para evitar borrados no deseados.

Lenguajes más guardados en GitHub



Referencias:

https://aulasoftwarelibre.github.io/taller-de-git/cvs/

https://altenwald.org/2009/01/12/sistemas-de-control-de-versiones-centralizados-o-distribuidos/

https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones

https://medium.com/@mehran.hrajabi98/a-brief-history-of-version-control-systems-vcss-5881f07ba0e1

https://www.freecodecamp.org/espanol/news/10-comandos-de-git-que-todo-desarrollador-deberia-saber/

https://github.blog/2015-08-19-language-trends-on-github/