Sistema de control de versiones GIT

Daniel Walther Berns

Departamento de Ingeniería Electrónica - Facultad de Ingeniería

Universidad Nacional de la Patagonia San Juan Bosco

daniel.w.berns@gmail.com

November 26, 2020

Overview

1. GIT: WTF?
   - Preguntas y ejemplos
2. GIT: comandos y ejemplos de uso
   - Git y Github
     - Computadora con Linux

¿Qué queremos?

1. Queremos programar la placa Raspberry Pi: En realidad, queremos programar la placa Raspberry Pi "embebida" o "incorporada" en un sistema más grande. La única seguridad que tenemos es que la placa va a estar conectada a Internet (ver Internet de las cosas), sin teclado, ratón, o monitor.

¿Qué queremos?

1. Queremos programar la placa Raspberry Pi: En realidad, queremos programar la placa Raspberry Pi "embebida" o "incorporada" en un sistema más grande. La única seguridad que tenemos es que la placa va a estar conectada a Internet (ver Internet de las cosas), sin teclado, ratón, o monitor.
2. Queremos trabajar con un servidor "en la nube", donde tenemos un sitio web propio o donde ejecutamos un programa que realiza algún cálculo o proceso durante mucho tiempo (varios días).

¿Cómo hacemos?

1. ¿Cómo hacemos para instalar o actualizar el software instalada en estos sistemas sin acceso físico?

¿Cómo hacemos?

1. ¿Cómo hacemos para instalar o actualizar el software instalada en estos sistemas sin acceso físico?
2. ¿Cómo hacemos para desarrollar software para estos sistemas?

¿Cómo hacemos?

1. ¿Cómo hacemos para instalar o actualizar el software instalada en estos sistemas sin acceso físico?
2. ¿Cómo hacemos para desarrollar software para estos sistemas?
3. ¿Cómo podemos desarrollar el software en nuestra PC y transferirlo con rapidez a la computadora donde se ejecutará?

Una propuesta

- Podemos desarrollar en una PC o notebook
- después copiar nuestro trabajo a un servidor
- y finalmente copiar desde el servidor a la computadora de trabajo.

Las razones de la propuesta

- Podemos desarrollar en una PC o notebook para trabajar cómodos, tal vez varias personas en equipo,

Las razones de la propuesta

- Podemos desarrollar en una PC o notebook para trabajar cómodos, tal vez varias personas en equipo,
- después copiar nuestro trabajo a un repositorio de software en un sitio web conocido,

Las razones de la propuesta

- Podemos desarrollar en una PC o notebook para trabajar cómodos, tal vez varias personas en equipo,
- después copiar nuestro trabajo a un repositorio de software en un sitio web conocido,
- y finalmente copiar desde el servidor al sistema embebido o servidor de Internet, con una actualización automatizada.

Preguntas

- ¿Para qué un sitio web con un repositorio?
- ¿Para qué varias personas en equipo? "Yo trabajo solo".
- ¿Actualización automatizada?
- ¿Como evitamos problemas con las versiones del software?

Respuestas

¿Para qué un sitio web con un repositorio?

El servidor nos permite independizar el software de las computadoras donde desarrollamos el software. Los sistemas conectados a Internet tienen una dirección IP, al igual que las computadoras donde se desarrolla el software. Para las actualizaciones, nuestros sistemas embebidos o sitios web necesitan una dirección IP conocida, la del repositorio.
Las personas que desarrollan el software pueden cambiar, sus computadoras pueden cambiar, pero el software está siempre en el repositorio donde puede ser recuperado y actualizado.

Respuestas

¿Para qué varias personas en equipo? "Yo trabajo solo".

Aún cuando una persona trabaje sola en un momento determinado, no trabaja para si misma. El hecho de guardar el software en un repositorio facilita la entrega del software a quien haya solicitado el desarrollo. Además, el repostorio puede servir también como lugar para un sistema de manejo de asuntos relacionados al software, donde se va construyendo la historia de anuncios, modificaciones, problemas y características del software.

Respuestas

"Yo trabajo solo" no existe en el movimiento Open Source

El software y hardware producido en forma Open Source (libre y gratuito para usar y modificar) se construye mediante la colaboración de los usuarios que encuentran y corrigen errores. Aún cuando no trabajemos en un proyecto Open Source, el trabajo en equipo permite producir sistemas más confiables en un lapso de tiempo menor. El empleo de un servidor donde siempre están todas las versiones del software desarrollado permite una colaboración más fluída.

Respuestas

¿Actualización automatizada (Raspberry PI)?

Las Raspberry Pi son computadoras muy potentes, capaces de ejecutar programas sofisticados. Nosotros vamos a emplear Raspbian, una variante de Debian (una distribucion de Linux). Uno de los servicios de Linux se denomina crond, y es capaz de ejecutar un comando de nuestra elección en un horario también de nuestra elección. Así cada Raspberry Pi puede conectarse al servidor y verificar si tiene que actualizar su software, por ejemplo a medianoche. En otras palabras, nosotros solamente debemos actualizar el servidor cuando sea necesario y todas las placas que programemos se conectarán y se actualizarán.

Respuestas

¿Como evitamos problemas con las versiones del software y del hardware?

Cuando se desarrolla un sistema industrial, es normal que haya diferentes versiones de software y hardware. Además, cuando varias personas trabajando en el mismo proyecto, se pueden generar dos modificaciones diferentes del mismo archivo por accidente. Al tener todo los archivos almacenados en un servidor, ahí se puede verificar si existen estos tipos de problemas.

Respuestas

1. GIT es un sistema de control de versiones, formado por un servidor y un cliente, que implementa todas estas características que hemos planteado.
2. Si comenzamos a trabajar en un proyecto que ya está en funcionamiento, primero usamos el cliente para copiar la información del servidor a nuestra computadora.
3. Editamos el software en nuestra computadora, y cuando terminamos empleamos el cliente para indicar que archivos van al servidor.
4. Si aparece algún conflicto, el cliente nos avisa con quien tenemos que negociar la versión final del archivo en cuestión o nos permite abrir dos ramas de desarrollo distintas.

Github y Bitbucket

`https://github.com/`

`https://bitbucket.org/`

Github

Servidor GIT, con repositorios públicos gratuitos, otras opciones pagas.

Bitbucket

Servidor GIT, con repositorios públicos y privados gratuitos, otras opciones pagas.

Crear cuentas de usuarios en uno de los dos servidores

Es necesario crear una cuenta de usuario en uno de los dos sitios, a elección de cada uno.

¿Que vamos a aprender?

1. Como manejamos proyectos usando git y github.
2. Como contribuimos a proyectos dirigidos por otras personas.

¿Es necesario instalar algo?

1. Raspbian ya tiene preinstalado git, por lo tanto en la Raspberry Pi no hay nada que instalar por el momento.
2. En Debian, Ubuntu podemos escribir

   ```
   sudo apt install git
   ```
3. En Fedora, podemos escribir

   ```
   sudo dnf install git
   ```
4. En Windows, descargar el instalador

   ```
   https://git-scm.com/download/win
   ```
5. Otras alternativas para Windows

   ```
   http://www.jamessturtevant.com/posts/
       5-Ways-to-install-git-on-Windows/
   ```

PythonAnywhere

1. En PythonAnywhere, podemos abrir una terminal en nuestro navegador. Esta terminal nos da acceso a una máquina virtual con Linux en PythonAnywhere.
2. En la terminal disponemos de git, por lo tanto no necesitamos instalarlo.

¿Hay documentación para leer?

`https://git-scm.com/doc`

Repositorio GIT

Un repositorio GIT es un almacenamiento virtual para proyectos de software. Permite guardar distintas versiones de archivos, que pueden ser accedidos en cualquier momento.

Configuración

En una computadora con Linux, abramos una terminal y ejecutemos los siguientes comandos.

1. Configuramos estos valores para simplificar algunos comandos.

   ```
   > git config --global user.name "Mi Nombre"
   > git config --global user.email "mi_cuenta@ejemplo.org"
   ```

   Omitimos `--global` si tenemos varios repositorios en una computadora y deseamos configurar solamente uno.

Configuración

En una computadora con Linux, abramos una terminal y ejecutemos los siguientes comandos.

1. Configuramos estos valores para simplificar algunos comandos.

   ```
   > git config --global user.name "Mi Nombre"
   > git config --global user.email "mi_cuenta@ejemplo.org"
   ```

   Omitimos `--global` si tenemos varios repositorios en una computadora y deseamos configurar solamente uno.

2. Esto es muy importante

   ```
   > git config --global core.editor nano
   ```

   El editor por defecto es vim. Con esta configuración lo cambiamos por nano. Si quieren, prueben ejecutar vim.

Como crear repositorios

1. En una computadora con Linux, para crear un repositorio local ejecutamos en la terminal los tres comandos siguientes

```
> mkdir proyecto
> cd proyecto
> git init
```

Como crear repositorios

1. En una computadora con Linux, para crear un repositorio local ejecutamos en la terminal los tres comandos siguientes

   ```
   > mkdir proyecto
   > cd proyecto
   > git init
   ```

2. En una computadora con Linux, para copiar un repositorio existente en github ejecutamos en la terminal los comandos

   ```
   > git clone <repo-url>
   ```

   donde `<repo-url>` es la dirección web del repositorio deseado. Por ejemplo, una dirección de un repositorio en Github es

   `https://github.com/DanielBerns/Image-Recognition.git`

Como crear repositorios, explicación

1. El comando `git init` crea un directorio denominado `.git` dentro del directorio proyecto, donde se guardan datos para manejar las versiones de los archivos incluidos en el repositorio.

```
mkdir proyecto
cd proyecto
git init
```

2. El comando `git clone` copia un repositorio desde un servidor. Este comando crea un directorio con el nombre del repositorio, y dentro de este directorio aparece el directorio `.git` correspondiente y los archivos incluidos en el repositorio.

```
git clone https://github.com/DanielBerns/Image-Recognition
```

Directorio controlado

1. El directorio controlado de nuestro repositorio es aquel donde encontramos al directorio `.git`
2. Por ejemplo, si ejecutamos

   ```
   git clone https://github.com/DanielBerns/Image-Recognition
   ```

   git creará un directorio con el nombre `Image-Recognition` que es el directorio controlado del repositorio `Image-Recognition`, porque cuando ejecutamos

   ```
   cd Image-Recognition
   ls -a
   ```

   vamos a ver que aparece el directorio `.git` en el listado de archivos.

Diferencia entre git init y git clone

1. El comando `git init` inicializa un repositorio en forma local, sin usar ningún servidor remoto.
2. El comando `git clone` copia un repositorio de un servidor remoto a la computadora local.
3. Un repositorio local creado con `git init` puede conectarse a un servidor remoto, pero no es necesario hacerlo. En caso de usar un repositorio local, podemos manejar las distintas versiones de nuestro software pero sin compartirlo con otras persnas. En este caso, tendremos que usar un método alternativo para pasar nuestro código al sistema embebido (que veremos mas adelante, en el capítulo "SSH").

Después de git init, como conectarnos a un servidor remoto

Supongamos que hemos creado un repositorio local con `git init`, y deseamos "conectarlo" a un servidor remoto.

1. Creamos un repositorio en el servidor remoto, empleando la interfaz web de Github. Supongamos que la dirección que github nos da como resultado es

   `https://github.com/DanielBerns/cia.git`

   Recordemos el usuario y la contraseña de nuestra cuenta en Github porque la necesitaremos en el siguiente paso.

2. En la terminal local, agregamos el repositorio remoto al local

   ```
   git remote add origin
       https://github.com/DanielBerns/cia.git
   ```

   Para esto necesitamos estar en el directorio controlado local.

Como agregar y enviar archivos al repositorio en el servidor

1. Dentro del directorio controlado (el directorio del proyecto), podemos crear los archivos que deseemos. Una suposición implícita es que estos son archivos de texto, que contienen programas escritos en python o c (por ejemplo). Se supone que no son archivos binarios de gran magnitud.
2. Una vez que hemos escrito un archivo (supongamos `first_example.py`) ejecutemos el comando

   ```
   git status
   ```

   que nos mostrará un mensaje donde nos indica que existe un archivo modificado (`first_example.py`) y que sus cambios no están registrados. También nos indica que podemos ejecutar el comando

   ```
   git add first_example.py
   ```

   Este comando prepara lo que se denomina "staging area", o sea el listado de archivos que van a ser incluidos en una nueva versión.

Como agregar y enviar archivos al repositorio en el servidor

1. Una vez que agregamos el archivo, tenemos que ejecutar el comando

   `git commit -m "creating first_example.py"`

   donde lo que escribimos después de `-m` es un mensaje que aparecerá en github como comentario "adherido" a los archivos incluidos en la "staging area".
2. Finalmente, ejecutamos

   `git push`

   Después de este comando, podemos verificar mediante la interfaz web de Github que los archivos se han actualizado en el servidor.

Como deshacer cambios

1. Podemos usar el comando `git checkout` para deshacer cambios, y volver a versiones anteriores.
2. Suponemos que hemos ejecutado al menos un `git push`.
3. Ejecutando

   ```
   git checkout
   ```

   hacemos que desaparezca cualquier cambio hecho a los archivos incluidos en nuestro directorio de trabajo, después del último `git push`.

Como deshacer cambios a un archivo

1. Supongamos que tenemos un archivo `README.md`
2. Además, este archivo ha sido incluido en el repositorio antes del último `git push`.
3. Ejecutando

```
git checkout README.md
```

hacemos que desaparezca cualquier cambio hecho al archivo `README.md` después del último `git push`.

Ramas (branches)

1. Un repositorio tiene una o más ramas (branches).
2. Supongamos que hemos estado trabajando en un programa durante varias semanas, y por fin se lo mostramos a un grupo de usuarios.
3. Los usuarios necesitan dos cosas: explicaciones sobre como usar el programa, y que corrijamos los errores que van apareciendo.
4. Para poder explicar el uso del programa, necesitamos mantener el código tal cual fue entregado. Para poder corregir errores, necesitamos modificar el código.
5. Para resolver esta contradicción, tenemos las distintas ramas.
6. Cuando creamos un repositorio, tenemos una rama llamada main o master.

Ramas (branches)

1. Un repositorio tiene una o más ramas (branches).
2. Cuando creamos un repositorio, tenemos una rama llamada main o master.
3. Cuando necesitemos modificar el código y a la vez mantener una versión anterior, podemos crear una rama nueva con un nombre propio.

Crear una rama: esto es magia

Comando para crear una rama y "copiar archivos"

```
git checkout -b nombre_rama
```

Al crear una rama se crea una copia nueva.

Cambiar de ramas

Comando para cambiar de rama (¡atención!)

```
git checkout main           # volvemos a la rama main
git checkout nombre_rama    # volvemos a la rama nombre_rama
```