INGENIERÍA UNIVERSIDAD DE LOS ANDES MÉRIDA VENEZUELA | DEPARTAMENTO DE COMPUTACIÓN ESCUELA DE SISTEMAS

Laboratorio de Ingeniería del Software
Práctica Nro. 3
Control de Versiones con Git

Prof. Solazver Solé
Prep. Yenifer Ramírez
Semestre B-2018

1. Git

Es un sistema de control de versiones, desarrollado por Linus Tolvards, ampliamente utilizado a nivel mundial por desarrolladores de software.

1.1. Conceptos básicos

- **Working directory:** directorio de trabajo dónde se inicia el repositorio.
- **Staging Area:** área de preparación, en esta área se encuentran todos los archivos que fueron modificados y están a punto de confirmarse.
- **.git directory:** es el directorio de archivos confirmados, es decir aquellos archivos que previamentemente estuvieron en el área de preparación.

1.2. Flujo de Trabajo

Una vez creado el repositorio local, algunas de las acciones mas comunes son:

- Modificar una serie de archivos en el directorio de trabajo.
- Enviar al área de preparación los archivos.
- Confirmar los cambios, lo que toma los archivos tal cual y como están en el área de preparación y almacena una copia instantánea de manera permanente en el directorio git.

1.3. Instalación de Git

Si no tenemos instalado Git en nuestro sistema operativo, escribimos en nuestra terminal (en debian o derivados):

```
root@pc:/# apt-get install git
```

1.4. Configuración por primera vez

Una vez ubicado en el directorio de trabajo. Realizamos los siguiente comandos:

```
usuario@pc:/# git init
usuario@pc:/# git config -global user.name "Tu Nombre"
usuario@pc:/# git config -global user.email "tucorreo@gmail.com"
usuario@pc:/# git config -global user.editor nano
```

1.5. Comandos básicos

Para realizar nuestras primeras acciones en el repositorio debemos tener un archivo, preferiblemente editable. Al tener algún archivo o directorio podemos utilizar los siguiente comands para observar el área de preparación, añadir un archivo al área de preparación y confirmar el archivo que se encuentra allí, mediante los siguiente comandos:

```
usuario@pc:/# git status
usuario@pc:/# git add archivo.txt
usuario@pc:/# git commit -m "Este es mi primer commit"
```

Otros comandos utiles que nos servirán:

- Para observar la lista de archivos sin seguimiento y en la zona de preparación:

```
usuario@pc:/# git status
```

- Para borrar un archivo que se encuentra en la zona de preparación:

```
usuario@pc:/# git reset HEAD archivo
```

- Para observar todos las confirmaciones realizadas(historial):

```
usuario@pc:/# git log -oneline -decorate -graph
```

- Para observar las diferencias entre el ultimo commit y lo modificado:

```
usuario@pc:/# git diff archivo
```

- Para observar las diferencias entre dos confirmaciones:

```
usuario@pc:/# git diff commit1(hash) commit2(hash)
```

- Para modicar el mensaje de un commit:

```
usuario@pc:/# git commit -amend
```

- Borrar archivos rastreados del repositorio y de nuestro directorio de trabajo de manera que no aparezcan la próxima vez como archivos no rastreados:

```
usuario@pc:/# git rm archivo
```

- Para renombrar archivos:

```
usuario@pc:/# git mv file from file to
```

1.6. Puntos de Confirmación

El comando **git log** nos permite observar todo el historial de confirmaciones de nuestro repositorio. Dentro de este historial encontramos 3 características importantes:

- **HEAD:** representa el punto de confirmación en donde estamos.
- **master:**rama inicial.
- **hash:** es el valor o id, que se la da a un punto de confirmación.

Para volver a un punto del historial de confirmaciones utilizamos el comando:

```
usuario@pc:/# git checkout hash
```

Si queremos volver a nuestra ultima confirmación:

```
usuario@pc:/# git checkout master
```

1.7. Etiquetas

Las etiquetas son puntos en el historial de confirmaciones que son importantes. Podemos agregar etiquetas a cualquier punto de la historia. Existen dos tipos de etiquetas:

- **Etiquetas ligeras:** es un checksum, de un commit guardado en un archivo.

```
usuario@pc:/# git tag nombre-etiqueta
```

- **Etiquetas anotadas:** se guardan en la base de datos de git como objetos enteros. Tienen checksum, el nombre del etiquetador, correo,fecha y mensaje.

```
usuario@pc:/# git tag -a v1.0 -m "mi version 1.0"
```

Podemos observar una etiqueta especifica con:

```
usuario@pc:/# git show tag-name
```

1.8. Ramas

- La rama principal es **master** y es un apuntador a una un confirmación.
- Las etiquetas también son apuntadores.
- A diferencia de las etiquetas, la rama master se mueve.
- Dos o mas ramas pueden apuntar a la misma confirmación o commit.
- Cuando se realiza un nuevo commit, este siempre es apuntado por la rama actual.
- Las ramas son utilizadas para crear nuevas características de nuestro proyecto, p.e. una nueva funcionalidad para nuestra aplicación, y una vez la funcionalidad ha sido probada, debemos funcionarla o agregarla a la rama principal.

Para crear una nueva rama usamos:

```
usuario@pc:/# git branch nombre-rama
```

Podemos observar en el historial la creacion de la nueva rama. Para ubicarnos en esa rama utilizamos el comando:

```
usuario@pc:/# git checkout nombre-rama
```

Para crear una rama y ubicarnos inmediatamente, podemos realizar modificaciones y confirmaciones en esta rama, y siempre estará apuntado por la rama actual.:

```
usuario@pc:/# git checkout -b nombre-rama
```

Con el siguiente comando podemos saber en que rama nos encontramos(HEAD):

```
usuario@pc:/# git branch
```

1.9. Fusiones

Una vez que se desarrolla una nueva funcionalidad en otra rama, es importante realizar la fusión. El comando utilizado para realizar la fusión es:

```
usuario@pc:/# git merge otra-rama
```

Para eliminar una de las ramas usamos. **Es eliminada si ya ha sido fusionada con la rama actual.**

```
usuario@pc:/# git branch -d nombre-rama
```

Si queremos borrar una rama incluso cuando no ha sido fusionada usamos:

```
usuario@pc:/# git branch -d nombre-rama
```

1.10. Conflictos

- Cuando hacemos una **fusión**, podemos generar conflictos. Esto pasa cuando modificamos **la misma linea del mismo archivo**.
- Cuando esto pasa, se genera un mensaje de conflicto.
- En caso de aceptar el conflicto, los archivos involucrados mostrar un mensaje, en el cual se indica el inicio de la rama actual HEAD, una linea de separación y el fin del conflicto.

2. GitHub

Github es un repositorio de git en la nube.

- Se le considera la red social de los programadores.
- Para mantener nuestros repositorios en la nube, github provee dos tipos de comunicacion: http y ssh.
- Para realizar esta actividad es necesario tener una cuenta en git-hub. [1]

2.1. Conceptos Básicos

- **Fork:** clon de un proyecto en la nube, en nuestra cuenta de github. Para clonar un repositorio usamos:

  ```
  usuario@pc:/# git clone http://..
  ```

  Podemos iniciar nuevas ramas y añadir funcionalidades. Para realizar confirmaciones en la nube usamos el siguiente comando:

[1] http://github.com/

```
usuario@pc:/# git push identificador-repositorio rama-nube
```

- **Pull Request:** es una petición que el propietario de un Fork de un repositorio hace al propietario del repositorio original para que este ultimo incorpore las confirmaciones (commits) que están en el Fork.

  1. Se debe enviar una solicitud.
  2. El dueño del repositorio decide si lo aprueba.
  3. Se añade o no, los nuevos cambios.

  Se dice que un pull request esta abierto cuando no se han aceptado los cambios.

2.2. Conexión a GitHub

1. Para mantener un repositorio en la nube debemos hacer las configuraciones globales de forma correspondientes con nuestros datos de github.
2. Crear un repositorio local en nuestro directorio de trabajo.
3. Crear un repositorio en la nube desde la interfaz de la github.
4. Para conectarnos lo primero que debemos hacer es añadir el repositorio en la nube en nuestra configuración.

Para añadir nuestro repositorio en la nube usamos, dónde origin es el identificador de nuestro repositorio en la nube:

```
usuario@pc:/# git remote add origin http:/github
```

Para subir los cambios la primera vez, en el repositorio origin en la rama master:

```
usuario@pc:/# git push -u origin master
```

Para bajar las confirmaciones del respositorio en la nube usamos:

```
usuario@pc:/# git pull origin master
```

Por ultimo podemos observar el estado de nuestro repositorio local con respecto al repositorio en la nube, revisando el historial de confirmaciones.

3. Actividades

3.1. Actividades Sugeridas

Actividades prácticas

- Genere un repositorio local con las configuraciones necesarias.

- Genere y modifique los archivos que considere necesarios.
- Realice una rama llamada test y añada los archivos necesarios.
- Modifique los archivos necesarios de tal forma que se genere un conflicto.
- Realice una fusión y resuelva el conflicto.
- Borre la rama creada en el repositorio anterior.

Actividades Teóricas

- Investigar acerca de los *work flows* conocidos para git.
- Organizar un documento donde muestre las características del work-flow: **git-flow**.
- Organice junto a su equipo de trabajo una simulación en un repositorio de github, el work-flow antes mencionado.

3.2. Actividades Obligatorias

- Envie al preparador por mensaje de telegram su usuario de GitHub.
- Después de recibir la invitación de su usuario a la organización **SolazverULA** en GitHub, cree el repositorio de su proyecto de semestre, de la materia Ingeniería del Software.

Recomendación: para completar el conocimiento adquirido con esta práctica realice el curso de de este enlace: `https://www.youtube.com/watch?v=zH3I1DZNovk&list=PLxOGU_ExE-4S0nYU4FNS3VdE6kfgtT2_k`