Control de versiones con Git

Introducción

Un controlador de versiones es una herramienta que gestiona los cambios de un conjunto de archivos. Cada conjunto de cambios genera una nueva versión de los archivos. El controlador de versiones permite, además de la gestión de cambios, recuperar una versión vieja de los archivos o un archivo, así como resolver conflictos entre versiones.

Aunque su principal uso es para agilizar la colaboración en el desarrollo de proyectos de *software*, también puede utilizarse para otros fines (como estas notas) e inclusive para trabajar solo (como en una tesis o proyecto).

Especificamente, un controlador de versiones ofrece lo siguiente:

1. Nada de lo que es *comiteado* (commited, ahorita vemos que es eso) se perderá.
2. Lleva un registro de *quién* hizo *qué* cambios y *cuándo* los hizo.
3. Es *casi* imposible (nota el casi) sobreescribir los cambios de tu colaborador. El controlador de versiones notificará que hay un **conflicto** y pedirá que lo resuelvas antes de continuar.

En esta clase usaremos `git`, aunque debemos de notar que no es el único controlador de versiones que existe, entre los más populares se encuentran `bazaar`, `mercurial` y `subversion` (Aunque este último pertenece a una diferente clase de controladores de versiones).

Configurando tu git

Abre tu `docker`, cambia al usuario `jovyan` y ve a tu carpeta `home`. Ahí, teclea lo siguiente, para establecer el usuario, correo (que usará `Github` para identificarte por ejemplo), si colorea la salida, y que editor por default para resolver conflictos.

```
➜  git config --global user.name "<Tu nombre>"
➜  git config --global user.email "<Tu correo>"
➜  git config --global color.ui "auto"
```

Creando un repositorio

El **repositorio** es la carpeta donde `git` guarda y gestiona todas las versiones de los archivos.

Crea una carpeta en tu `$HOME` llamada `test`, ingresa a ella e inicializa el repositorio con `git init`. ¿Notas algún cambio? ¿Qué comando usarías? Hay una carpeta ahí ¿no la ves? ¿Cómo puedes ver una carpeta oculta?

La carpeta `.git` es la carpeta donde se guarda todo el historial, si la borras, toda la historia del repositorio se perderá.

Podemos verificar que todo esté bien, con el comando `status`.

```
➜  git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Llevando registro de los cambios a archivo

Crea un archivo llamado `hola.txt` de la siguiente manera (ALERTA: nuevos comandos de **CLI**):

```
➜ touch hola.txt
➜ echo "¡hola mundo!" > hola.tx
➜ git status
```

(Para ver el contenido de `hola.txt` utiliza el comando `cat hola.txt`).

El mensaje de `untracked files` significa que hay archivos en el repositorio de los cuales `git` no está llevando registro, para que `git` lo haga, debes de **agregarlos** (`add`):

```
➜ git add hola.txt
➜ git status
```

Ahora `git` sabe que debe de llevar registro de los cambios de `hola.txt`, pero aún se *comprometen* los cambios al repositorio (`Changes to be commited: ...`). Para *comitearlos*:

```
➜ git commit -m "Saludando"
```

Usamos la bandera `-m` para agregar un mensaje que nos ayude a recordar más tarde que se hizo y por qué.

Si ejecutamos

```
➜ git status
```

nos indica que todo está actualizado (`up to date`). Podemos ver la historia de cambio con `git log`.

Edita `hola.txt` usando emacs, luego, ejecuta `git status`

La parte clave es `no changes added to commit`. Hemos cambiado el archivo, pero aún no están "comprometidas" o guardadas en el repositorio.

Para ver que ha cambiado usamos lo siguiente

```
➜ git diff
```

Hagamos `commit` de estos cambios.

```
➜ git commit -m 'actualizamos hola.txt'
```

Pero `git` no nos dejará, ya que no lo agregamos antes al índice del repositorio. Agrégalo y repite el `commit` (¿Recuerdas como hacerlo con los tips de navegación?).

Git maneja este flujo:

Modificando los archivos --> `git add` --> en el área de *staging* (listos para *commit*) --> `git commit` --> Repositorio (guardados permanentemente en **tu** computadora.)

Modifica de nuevo `hola.txt`. Observa los cambios y agrégalo. ¿Qué sucede si vuelves a ejecutar `git diff`?

`Git` dice que no hay nada, ya que para `git` no hay diferencia entre el área de *staging* y el último *commit* (llamado `HEAD`). Para ver los cambios, ejecuta

```
➜ git diff --staged
```

esto muestra las diferencias entre los últimos cambios *comiteados* y lo que está en el área de *staging*. Ahora realiza el `commit`, verifica el estatus y revisa la historia.

Explorando el pasado

Podemos ver los cambios entre diferentes **revisiones**, podemos usar la siguiente notación: `HEAD~1`, `HEAD~2`, etc. como sigue:

```
➜ git diff HEAD~1 hola.txt
➜ git diff HEAD~2 hola.txt
```

También podemos utilizar el identificador único (el número enorme que aparece en el `git log`), inténtalo.

Modifiquemos de nuevo el archivo `hola.txt`. ¿Qué tal si nos equivocamos y queremos regresar los cambios? Podemos ejecutar el comando

```
➜ git checkout HEAD hola.txt
```

(Nota que `git` recomienda un *shortcut* para esta operación: `(use "git checkout -- <file>..." to discard changes in working directory)`)

Obviamente aquí podemos regresarnos las versiones que queramos, por lo que podemos utilizar el identificador único o `HEAD~2` por ejemplo.

> **Ejercicio**
> Recorre el log con `checkout`, observa como cambia el archivo, usando `cat`.

Por último, `git` tiene comandos `mv` y `rm` que deben de ser usados cuando queremos mover o borrar un archivo del repositorio, i.e. `git mv` y `git rm`.

> **Ejercicio**
> Crea un archivo `adios.txt`, comitealo, has cambios, comitea y luego bórralo. No olvides hacer el último `commit` también.

Antes de continuar con **Github** quiero recomendar que tengan a la mano la que mas les guste de estas ayudas visuales 1, 2, 3 o esta que es interactiva.

El concepto que no alcancé a ver aquí y que es muy poderoso es el de *stash*, pero pueden deducir su comportamiento de la última ayuda visual.

Github

Github es un repositorio central, con una interfaz web bonita, pero fuera de eso, nada impediría que trabajen ustedes con su computadora y la de sus amigos usando los comandos que en esta sección vamos a mostrar, ya que `git` es un sistema distribuito de control de versiones y **ningún nodo** tiene preferencia sobre los demás, pero por comodidad, podemos usar **Github** de tal manera.

El repositorio de la clase está en:

```
https://github.com/Skalas/Matematicas-computacionales-fall2020
```

Lo primero que hay que hacer es **forkear** este repositorio. La operación de `fork` creará una copia de repo de la clase en su usuario de github. Para obtener una copia de trabajo en su computadora deberán de **clonar** su repositorio:

```
→ git clone https://github.com/<usuario_git_hub>/Matematicas-computacionales-fall2020.git
```

Esto creará una carpeta `Matematicas-computacionales-fall2020` en su `$HOME`.

Ve a la carpeta `alumnos` y crea una carpeta cuyo nombre sea tu nombre de usuario en github. En esta carpeta realizarás las tareas, proyectos, etc.

Crea dentro de tu carpeta un archivo que se llame `prueba.txt`, agrégalo, verifica los cambios, has `commit`. El siguiente paso es **subir** los cambios al repositorio de github. Para hacerlo hacemos

```
➜ git push origin master
```

(Probablemente te pida tu usuario de github y contraseña.)

El verbo `push` es el encargado de la acción de subir los cambios. `origin` se refiere al repositorio del cual clonamos la copia local (en nuestra computadora) y `master` se refiere al `branch` (este es un tema avanzado).

Abre tu repositorio en github y observa que tus cambios ahí están. Ahora, vamos a simular que alguien está interactuando con tu repositorio. Desde el sitio web, ve a tu carpeta y has clic en el símbolo de `+` para agregar un archivo (verifica de nuevo que estés en tu carpeta). Llamalo `prueba2.txt` y agrega algo de texto. Para guardar los cambios aprieta el botón `Commit new file`.

En tu computadora, en `docker` en la carpeta `Matematicas-computacionales-fall2020`, para obtener los últimos cambios utiliza:

```
➜ git pull origin master
```

En tu carpeta debería de aparecer el archivo `prueba.txt`. Ahora quedan por contestar dos preguntas: (1) ¿Cómo actualizo el repositorio `Skalas/Matematicas-computacionales-fall2020` (para entregar tarea por ejemplo? y (2) ¿Cómo actualizo mi repositorio con los cambios del repositorio de `Skalas/Matematicas-computacionales-fall2020`?

Para ambas preguntas es necesario aclarar el concepto de los `remote`:
Si ejecutas

```
➜ git remote -v
origin  https://github.com/<usuario>/Matematicas-computacionales-fall2020.git (fetch)
origin  https://github.com/<usuario>/Matematicas-computacionales-fall2020.git (push)
```

Vamos a agregar como `remote` el repositorio `Skalas/Matematicas-computacionales-fall2020`:

```
➜ git remote add repo-clase
https://github.com/Skalas/Matematicas-computacionales-fall2020.git
```

Ejecuta de nuevo `git remote -v`

```
➜ git remote -v
origin    git@github.com:Skalas/Matematicas-computacionales-fall2020.git (fetch)
origin    git@github.com:Skalas/Matematicas-computacionales-fall2020.git (push)
repo-clase    https://github.com/Skalas/Matematicas-computacionales-fall2020.git (fetch)
repo-clase    https://github.com/Skalas/Matematicas-computacionales-fall2020.git (push)
```

Entonces, la respuesta a la primera pregunta es una técnica llamada `pull-request` que en mi conocimiento **solo** se puede realizar desde el navegador. Has click en `Pull Requests` y sigue las instrucciones de la pantalla.

Una vez **autorizado** el `pull-request` es necesario hacer

```
➜ git pull repo-clase master
➜ git push origin master
```

La primera instrucción baja los cambios del repositorio central, el segundo los sube a su repositorio particular.

Para responder la segunda respuesta usamos la misma cadena de comandos:

```
➜ git pull repo-clase master
➜ git push origin master
```

Nota que ahora no realizamos un `pull-request`.

¿Cuál debería de ser mi *workflow* de trabajo?

Te sientas en la compu, abres `docker`.
y:

```
➜ cd Matematicas-computacionales-fall2020
➜ git pull origin master
```

trabajas en varios archivos
(incluso cuando más adelante usemos
ipython notebook)
trabajas
editas
hackeas
Y conforme avances

```
➜ git status
➜ git add <archivos modificados o nuevos>
➜ git commit -m "<comentario acerca de que fue lo que se cambió>"
➜ git push origin master
➜ git status
```

Todo estará bien si

- `git status` respondió con:

```
Make sure the status says

On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
```

- Que en su repo estén sus cambios.

Cuando quieran enviar la tarea realizan el `pull-request`. Y habrán terminado si en el repositorio `Skalas/Matematicas-computacionales-fall2020` están sus cambios.

¡Has terminado! Ahora puedes irte a descansar y disfrutar del sol…