Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
Computación: Ciencia y Tecnología del Mundo Digital

Tarea Chica 1

Git y Línea de Comandos
Profesores Luis Ramírez y Vicente Domínguez
Enunciado: 19 de agosto de 2018

Indicaciones

- Fecha de entrega: hasta las 23:59 del día Lunes 27 de agosto de 2018.
- Debes entregar la tarea en tu repositorio git.
- Cada hora (o fracción) de atraso descontará 5 décimas de la nota que obtengas.
- La tarea es *individual*. La copia será sancionada con una nota $1{,}1$ en el la tarea, además de las sanciones disciplinarias correspondientes.

Objetivo

El objetivo de esta tarea es:

- Conocer el terminal de un sistema operativo Unix.
- Hacer uso de la herramienta *git*.
- Utilizar comandos básicos para manipular archivos desde el terminal.

Instrucciones

Para realizar esta tarea deberás familiarizarte con el terminal de Linux, por lo que debes realizar un tutorial en Codecademy **de forma obligatoria**. El curso presenta cuatro secciones, cada sección tiene primero una sesión interactiva y luego proyectos o pruebas. Solo debes completar las cuatro sesiones interactivas, ya que las otras son para las suscripciones pagadas. Si te interesa igualmente hacer los proyectos y pruebas, Codecademy te ofrece unos días de prueba que puedes usar para acceder a

esos recursos. Para iniciar sesión, utiliza una cuenta con un nombre de usuario que te identifique sin ambigüedades (*e.g.* **tu usuario UC o tu usuario de Github**).

Además, tendrás que familiarizarte con Git, por lo que deberás leer una Guía Introductoria que te servirá para entender los conceptos mínimos de este sistema de control de versiones.

Después de completar el tutorial y leer la guía, deberás realizar las actividades que se detallan a continuación.

Toda esta tarea la vamos a realizar usando la consola. Los usuarios de Linux y Mac pueden usar la *shell* de su sistema, mientras que los usuarios de Windows pueden usar Git Bash o Bash for Windows (Windows Subsystem for Linux).

Para las partes de la tarea en las que necesitemos ejecutar archivos de python, usaremos python 3.7.

Actividades

Actividad 1

Esta actividad será guiada, por lo que detallaremos cada uno de los comandos que tenemos que ejecutar. Lo primero que debemos hacer es abrir la consola de comandos y situarnos en la carpeta donde queramos crear nuestro repositorio local. Para ello usamos el siguiente comando:

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

Con este comando se crea una copia del repositorio en nuestro computador (recuerden que se les asignó un repositorio especial para la tarea. En caso de que no utilicen este repositorio y realicen la tarea en otro, se asumirá que no hicieron la tarea). Usamos el comando `ls` para ver que se creó una carpeta nueva. Entramos en ella usando el comando `cd`:

```
$ cd Tarea-Chicha-1-SUNOMBRE
```

Para ver qué hay dentro usamos nuevamente el comando `ls`, y deberíamos ver los archivos `nombre.md`, `actividad2.py`, las carpetas `resultados` y `ann` (paquete de Python). **No hay que ejecutar ni modificar nada.** Podemos usar el comando `git status` para verificar que el repositorio se encuentra sin cambios.

Nuestra primera modificación será en el archivo `nombre.md`. Usaremos la línea de comandos para editarlo con `nano`:

```
$ nano nombre.md
```

Escribimos nuestro usuario de Github en el editor y guardamos el archivo (`Ctrl + O` y luego `Enter`). Para salir usamos la combinacion `Ctrl + X`.

Si usamos `git status` nuevamente deberíamos ver que el archivo `nombre.md` fue modificado. Para ver qué cambios se hicieron (línea por línea) también podemos ejecutar `git diff`.

Ahora nos queda subir este cambio al repositorio remoto. Para eso debemos seguir tres pasos. El primero es agregar el archivo al *staging area*:

```
$ git add nombre.md
```

Igual que antes, usamos `git status` para ver si se agregó (esto lo haremos todo el tiempo). El segundo paso es crear el *commit*:

```
$ git commit –m "Terminada la actividad 1"
```

Y finalmente subimos los cambios al repositorio:

```
$ git push origin master
```

Si volvemos a usar `git status`, deberíamos ver que nuestro repositorio se encuentra sincronizado y sin cambios.

Actividad 2

Dentro de los archivos del repositorio, hay un programa en Python que supuestamente es capaz de generar código de manera inteligente. Se sabe que el proceso de generación de código es costoso, tiene hartas etapas y se crean archivos con información para *debugguear*. Sabemos además que el programa tiene algunos errores por lo que vamos a ejecutarlo de manera experimental para ver qué resulta.

Para ejecutar el programa desde la consola, podemos abrir Python y como parámetro entregarle el nombre del archivo. En Windows el comando `python` generalmente corre la última versión de Python que hayamos instalado. En la mayoría de las distribuciones Linux, el comando `python` ejecuta Python 2.7, por lo tanto debemos usar el comando `python3` para correr el programa:

```
$ python3 actividad2.pyc
```

Para revisar los cambios hechos por el programa en nuestro repositorio, vamos a ejecutar el comando `git status`. El programa creó un archivo `.py` en la carpeta `resultados`, junto con `LOGS.txt` y dos archivos temporales.

También, se creó una carpeta llamada `logs`, que dentro tiene varios archivos `.log` y también archivos temporales. Para revisar con más detalle los archivos en esta carpeta podemos usar:

```
$ cd logs
$ ls
```

Si volvemos a la carpeta principal y usamos `ls`:

```
$ cd ..
$ ls
```

Veremos que ahí también hay archivos temporales. Al parecer, uno de los problemas del programa es que no eliminó los archivos temporales.

Con `git status`, podemos ver que el archivo `nombre.md` también fue modificado, y ese archivo no se debería haber tocado. Este es un segundo error del programa.

En lo que queda de esta actividad vamos a solucionar los errores que causó el programa. Primero debemos eliminar todos los archivos temporales que se crearon, para ello usamos el comando `rm`:

```
$ rm *.tmp
$ rm */*.tmp
```

Con estos dos comandos eliminamos todos los archivos de extensión `.tmp` que se encuentran en la carpeta principal, y también todos los que se encuentran en las subcarpetas (recuerden que el símbolo `*` se usa como comodín). Si hacemos `git status`, podemos verificar que desparecieron del *working directory*.

El segundo problema a solucionar es la modificación en el archivo `nombre.md`. Para ello podemos usar un comando de git que deshace los cambios hechos en un archivo desde el último commit. En nuestro caso, el último commit fue el de la actividad 1, donde el archivo `nombre.md` solo contenía nuestro nombre. Para volverlo a ese estado usamos el siguiente comando:

```
$ git checkout nombre.md
```

Nuevamente si usamos `git status`, este archivo no debería aparecer en la lista de modificados.

Otro de los cambios detectados por Git y que no habíamos mencionado es una nueva carpeta `__pycache__` creada dentro del paquete `ann`. Estas carpetas contienen *bytecode* generado por el intérprete de Python al correr el programa por primera vez. Siempre que trabajemos con Python van a aparecer estas carpetas, pero no es algo que nos gustaría subir al repositorio remoto. Más bien es algo que nos gustaría que

Git ignorase. Para eso vamos a crear un archivo especial en el repositorio para que esas carpetas sean ignoradas.

El archivo que indica qué archivos y carpetas queremos ignorar se llama `.gitignore`. Vamos a crearlo y vamos a indicarle que no considere estas carpetas.

```
$ echo "__pycache__/" > .gitignore
```

Con el símbolo `>` estamos diciendo que el *output* del primer comando se guarde en el archivo `.gitignore`. El comando `echo` solo imprime lo que le entregamos como parámetro, de este modo estamos creando el archivo `.gitignore` que contiene la línea `__pycache__/` (esto le dice a Git que ignore todas las carpetas con ese nombre y su contenido). También podríamos haber usado `touch` y luego agregar la línea manualmente usando un editor como `nano`. Podemos verificar que el archivo se creó correctamente usando el comando `cat .gitignore`, que nos imprime su contenido.

Con `git status` podemos ver que ya no aparece la carpeta `__pycache__` como modificación del repositorio. Esto significa que funcionó. Lo único que queda es subir solo el archivo `.gitignore` al repositorio, para que todos los usuarios ignoren los mismos archivos:

```
$ git add .gitignore
$ git commit –m "Agregado .gitignore"
$ git push origin master
```

Con esto hemos terminado la actividad, pero si usamos `git status`, veremos que aún están los logs y el código generado como cambios en el repositorio. Esto lo revisaremos en la próxima actividad.

Actividad 3

A diferencia de las actividades anteriores, ahora deberás hacer uso de lo que has aprendido, sin ayuda.

Uno de nuestros intereses es subir el código generado por el programa al repositorio remoto, por lo tanto deberás crear un *commit* (**NO HACER PUSH**) que SOLO debe contener el archivo Python generado dentro de la carpeta `resultados`. Solo se debe crear el *commit*, sin subirlo al repositorio (eso lo haremos más adelante). Se debe usar una descripción clara para el *commit*.

Además, en esta oportunidad queremos subir los *logs* generados por el programa. Tendrás que crear otro *commit* que contenga exclusivamente los *logs* generados por el programa. Esto incluye los archivos `.log` dentro de la carpeta `logs` y también el

archivo `LOGS.txt` dentro de la carpeta `resultados`. Recuerda **no hacer push**, solo creen el *commit* y agreguen una descripción atingente.

Actividad 4

En la actividad anterior agregamos todos los archivos `.log` a un *commit*, pero no nos fijamos qué tan pesados eran estos archivos. Para revisar el tamaño de los archivos, podemos movernos a la carpeta `logs` y ejecutar `ls` para que nos liste los ficheros con más detalles (`-l`) y para que la información sea fácil de leer (`-h`):

```
$ cd logs
$ ls -lh
```

Si nos fijamos en el archivo `process2.log`, veremos que pesa cerca de 30 MB. En general no queremos tener archivos tan pesados en el repositorio y menos si se trata de un *log*.

Sin embargo, el *commit* ya está hecho y no hay manera de sacar un solo archivo del *commit*. La única manera es deshacer el último *commit*, quitar el archivo pesado y luego volver a crearlo. Comencemos deshaciendo el último *commit*:

```
$ cd ..
$ git reset HEAD^1
```

El comando `git reset HEAD^1` deshace todos los *commits* y quita los archivos del *staging area* que se agregaron después del penúltimo *commit*, por lo tanto es ideal para este caso, pues solo deshace el último *commit* y sin hacer cambios en los archivos. Notar que este comando solo afecta nuestro repositorio local, de modo que si ya subimos los *commits* al repositorio remoto (i.e. hicimos `git push`) entonces no nos servirá.

Ahora deberás usar el comando `rm` para eliminar el archivo pesado y una vez hecho esto volverán a crear el commit del mismo modo como hicieron al final de la Actividad 3. Por último tienes que hacer `git push` para subir los *commits* al repositorio remoto.

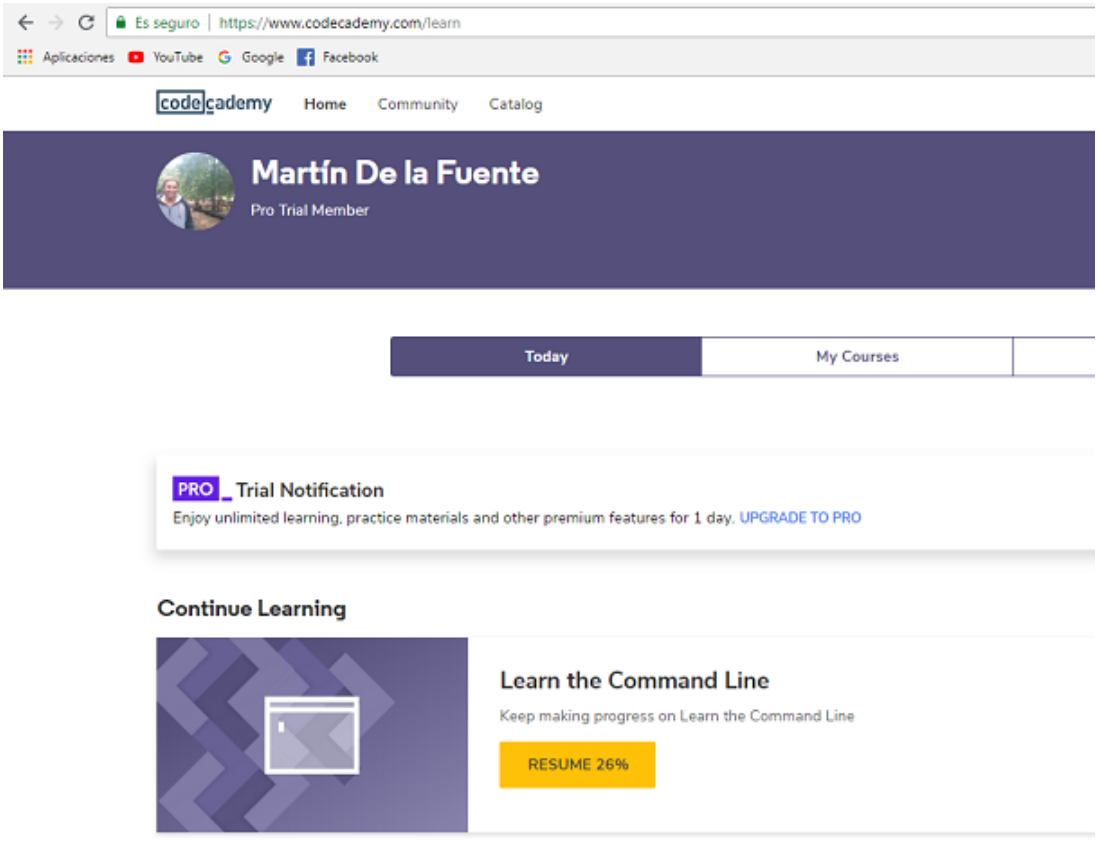
```
$ git push origin master
```

Actividad 5

En esta última actividad tendrás que sacar una *screenshot* de Codecademy **donde se vea su nombre de usuario** y que completaste al menos un 25 % del curso *Learn the command line* (por las cuatro sesiones interactivas). Debes guardarla como imagen en la carpeta principal de su repositorio, crear un *commit* con la descripción

correcta y subirla al repositorio remoto.

Un ejemplo de imagen donde se ve el nombre y el progreso es la siguiente:



Evaluación

- Para corregir, se revisarán todos los *commits* subidos al repositorio y que en cada uno se hayan subido y modificado exclusivamente los archivos indicados. Además, deben usarse descripciones correctas.
    - *Commit* 1: solo archivo `nombre.md` [0.5 ptos].
    - *Commit* 2: solo archivo `.gitignore` [1.5 ptos].
    - *Commit* 3: solo archivo de código generado por el programa [0.5 ptos].
    - *Commit* 4: solo archivos de *logging* [1.5 ptos].
    - *Commit* 5: solo imagen de Codecademy [2 ptos].
- Para la hora de entrega se revisará la hora del último **commit**.