Práctica 0: Git

Taller de Tecnologías de Producción de Software - Ruby

Cursada 2024

En esta práctica preliminar, repasaremos algunos conceptos y comandos básicos de Git, la herramienta de control de versiones que utilizaremos a lo largo de la cursada.

> Si ya estás familiarizado con los aspectos más generales de Git, podés saltear esta práctica, pero no sin antes **leer la siguiente sección** ("Aclaraciones generales") que explica algunas cuestiones comunes a todas las prácticas de la materia.

Aclaraciones generales

Estas son algunas cuestiones a tener en cuenta a lo largo de las prácticas de la materia:

1. Con el fin de unificar criterios, desde la cátedra sólo daremos soporte para Sistemas Operativos *nix (Unix-like). Principalmente, porque el desarrollo con Ruby en Windows suele traer problemas específicos de esa plataforma con los que se tiende a perder tiempo que no resulta redituable, y porque no tenemos experiencia solucionando esos problemas en esa plataforma.
2. Se asume que el ambiente donde se ejecutan los comandos es una shell Bash o compatible, por encontrarse presente en todas las distribuciones de Sistemas Operativos *nix.
3. Desde la cátedra no concebimos el desarrollo en Ruby sin utilizar una terminal, consola o linea de comandos. Es por eso que nuestras prácticas, explicaciones y contenidos complementarios serán en su gran mayoría basados en ese ambiente.
4. En los materiales de la cátedra, cuando estemos hablando de ejecutar comandos en una terminal vamos a denotar las líneas que tenés que ejecutar con un símbolo de prompt `$` si debés ejecutarlo con tu *user* o `#` si debés hacerlo con uno con privilegios de administrador (típicamente `root`).
5. A fin de facilitar la preparación del ambiente, en caso que no cuentes con una instalación disponible (física o virtual) de un Sistema Operativo *nix (GNU/Linux o macOS, por ejemplo), podés instalar una versión reciente de Ubuntu Linux en una máquina virtual en tu computadora con VirtualBox, que es una herramienta gratuita para virtualización de Sistemas Operativos.
6. Si no contás con un dispositivo en el que puedas instalar estas herramientas, podés probar un ambiente on line como repl.it en el que podrás realizar los ejercicios de la materia.

Requisitos

Para realizar esta práctica - y el resto de la materia también -, necesitás tener Git instalado en tu ambiente de desarrollo. Para saber si tenés Git instalado, podés ejecutar el siguiente comando en una terminal y ver su salida (resultado):

```
$ git --version
```

Si la salida del comando fue algo similar a esto, tenés instalado Git en tu sistema:

```
git version X.Y.Z
```

Si, por el contrario, obtuviste un error de que el comando no se encuentra quiere decir que Git no está instalado en el sistema. Cómo instalarlo variará según el ambiente de desarrollo que estés usando, pero si se trata de un Sistema Operativo basado en Debian como Ubuntu Linux, podés ejecutar el siguiente comando para instalar Git:

```
# apt update -qq && apt install -y git
```

Luego de eso, deberías poder ejecutar correctamente el comando `git --version` mencionado anteriormente.

Subcomandos Git

Git se maneja desde la línea de comandos mediante el uso de *subcomandos*. Cada tarea, orden o consulta que quieras hacer con Git la vas a poder realizar con el comando `git` y alguno de sus subcomandos, los cuales podés listar al ejecutar (solo) `git` o uno de los subcomandos más útiles que tiene Git: `git help`.

Cómo obtener ayuda

Git provee ayuda mediante las páginas del manual (o *man pages*) para cualquier subcomando que ofrezca, y para hacer aún más fácil la consulta de esa ayuda provee un subcomando especial que abre la página del manual del subcomando que quieras investigar: `git help`.

Ejercicios

1. Ejecutá `git` o `git help` en la línea de comandos y mirá los subcomandos que tenés disponibles.
2. Ejecutá el comando `git help help`. ¿Cuál fue el resultado?
3. Utilizá el subcomando `help` para conocer qué opción se puede pasar al subcomando `add` para que ignore errores al agregar archivos.
4. ¿Cuáles son los estados posibles en Git para un archivo? ¿Qué significa cada uno?

5. Cloná el repositorio de materiales de la materia: `https://github.com/TTPS-ruby/practicas.git`. Una vez finalizado, ¿cuál es el *hash* del último commit que hay en el repositorio que clonaste?

   > Tip: `git log`.

6. ¿Para qué se utilizan los siguientes subcomandos?

   - `init`
   - `status`
   - `log`
   - `fetch`
   - `merge`
   - `pull`
   - `commit`
   - `stash`
   - `push`
   - `rm`
   - `checkout`
   - `tag`

7. Utilizá el subcomando `log` para ver los commits que se han hecho en el repositorio, tomá cualquiera de ellos y copiá su *hash* (por ejemplo, `800dcba6c8bb2881d90dd39c285a81eabee5effa`), y luego utilizá el subcomando `checkout` para *viajar en el tiempo* (apuntar tu copia local) a ese commit. ¿Qué commits muestra ahora `git log`? ¿Qué ocurrió con los commits que no aparecen? ¿Qué dice el subcomando `status`?

8. Volvé al último commit de la rama principal (`master`) usando nuevamente el subcomando `checkout`. Corroborá que efectivamente haya ocurrido esto.

9. En un nuevo directorio, inicializá un repositorio Git, creá un archivo de texto en él y verificá el estado de tu espacio de trabajo con el subcomando `status`. ¿En qué estado está el archivo que creaste?

10. Agregá el nuevo archivo al *stage* de tu espacio de trabajo con el subcomand `add`, y luego verificiá su estado con el subcomando `status`. ¿En qué estado está el archivo que creaste?

11. Creá un nuevo *commit* en este repositorio con el subcomando `commit`. Recordá utilizar un mensaje descriptivo de los cambios que el *commit* incluye. Después de eso veriticá el estado de tu espacio de trabajo con el subcomando `status`. ¿Qué información obtuviste?

12. Para subir tus cambios locales a un repositorio remoto se utiliza el subcomando `push`. ¿Qué pasa si intentás subir el *commit* creado en el paso anterior? ¿Por qué? ¿Qué hace falta especificar?

13. Creá un directorio vacío en el raiz del proyecto clonado. ¿En qué estado aparece en el `git status`? ¿Por qué?
14. Creá un archivo vacío dentro del directorio que creaste en el ejercicio anterior y volvé a ejecutar el subcomando `status`. ¿Qué ocurre ahora? ¿Por qué?
15. Utilizá el subcomando `clean` para eliminar los archivos no versionados (*untracked*) y luego ejecutá `git status`. ¿Qué información muestra ahora?
16. Creá un repositorio en el servicio on line de hosting de Git de tu preferencia (los más conocidos son GitHub, GitLab y Bitbucket) y cloná el repositorio localmente, creá algunos archivos y directorios en el repositorio local, y subí tus cambios al remoto. Luego modificá desde la interfaz web que ofrecen estos servicios algún archivo de los que creaste, creá un *commit* desde allí mismo, y finalmente actualizá tu copia local de manera que puedas ver esos cambios. ¿Qué subcomando utilizaste para esto? ¿Qué hizo?