Taller de Git

Clase 1 - Comandos

Fernando López

LINTI
Facultad de Informática
Universidad Nacional de la Plata

16 de Octubre de 2013

Indice

Links

- `http://lihuen.linti.unlp.edu.ar/index.php?title=Git`
- `http://git-scm.com/book`

Git

- Sistema de control de versiones
- Distribuido
  - Cada usuario tiene toda la historia.
  - El comando git no tiene que consultar nada a ningún servidor.
  - Caso kernel.org

    "The code for the kernel - and for many other projects - is managed with the 'git' source code management system. And git does not allow the code to be modified by third parties without people knowing about it."

Instalación y configuración

- `apt-get install git`
- Cada commit queda registrado con nuestro nombre y e-mail:

```
git config --global user.name "Pipo Perez"
git config --global user.email "pperez@servidor.com"
```

- Resaltado con colores:

```
git config --global color.ui auto
```

- Nuestro editor favorito para escribir los commits:

```
git config --global core.editor vim
```

- Un comando muy útil:

```
git config --global alias.unstage 'reset HEAD --'
```

Primer repositorio

- Creá un directorio con el nombre `octocats`.
- Convertí ese directorio en un repositorio Git:

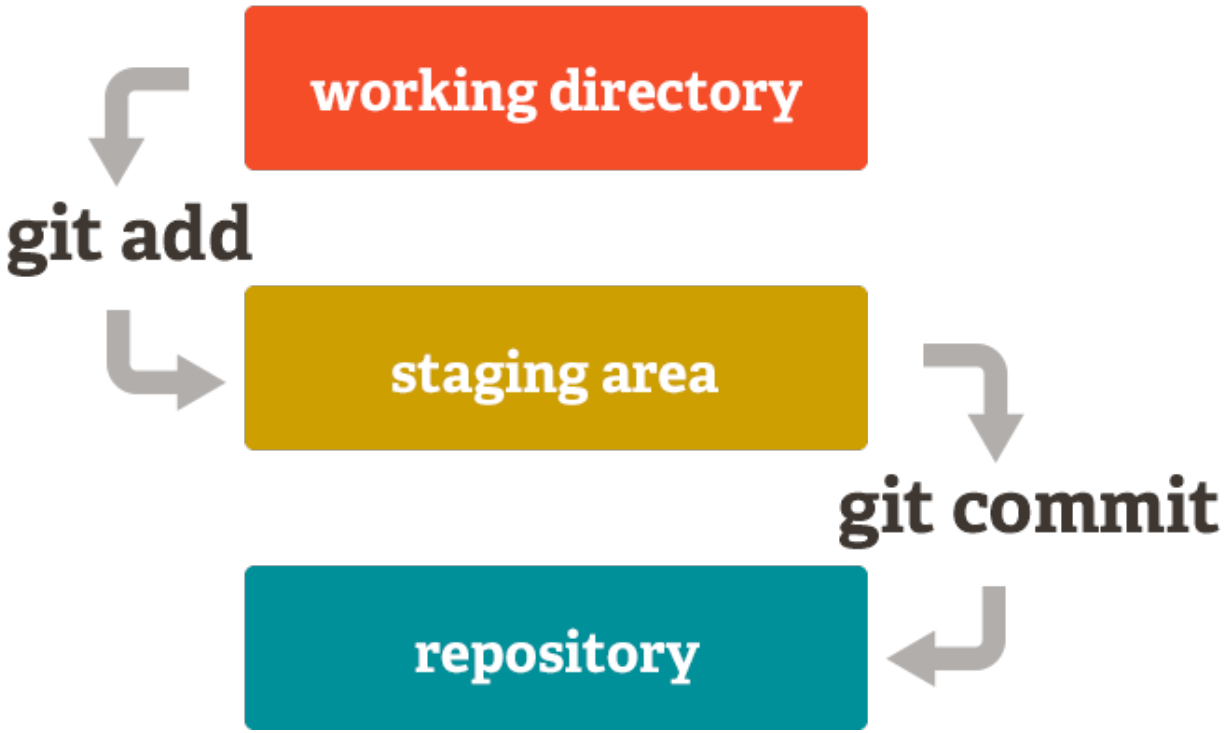
```
mkdir octocats
cd octocats
git init
git status
```

Primer repositorio
Las buenas prácticas

- En general cada repositorio tiene un archivo README.
- GitHub y GitLab lo renderizan.
- ¡Se convierte en la página principal del proyecto! `https://github.com/mozilla/pdf.js`
- Mínimamente tiene una descripción del proyecto, guía de instalación y links relevantes.
- Si el archivo se llama README.md puede tener formato con markdown.

```
vi README.md
git status
```

¿Untracked, Staged, qué?



Working directory, Staging area, repository


```
fernando@netbook:~/git/taller_git$ ls -a
.  ..  .git  .gitignore  .gitmodules
imagenes  Makefile  reveal.js  src
```

- taller_git → working directory
- taller_git/.git/index → staging area
- taller_git/.git → repository

Agregando contenido

- Creá un archivo de texto `favorito.txt` y escribí en él el nombre de tu octocat favorito (`http://octodex.github.com/`).
- Por ejemplo:
  `Jean-Luc Picat`
- `git status`
- `git add favorito.txt`
- `git status`
- `git add README.md`
- `git status`

Aplicar los cambios al repositorio

- `git commit`
-
```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Agregar la url

- Agregá en `favorito.txt` el link a la imagen del octocat, por ejemplo:
  `Jean-Luc Picat - http://octodex.github.com/images/`
- Hacé el commit de los cambios.
- `git status`
- Agregá en `favorito.txt` otro octocat con su link.
- Hacé el commit de los cambios.
- `git log`

Agregar la URL
Logs

- Logs → `git log`
- Cambios → `git whatchanged`
- Diferencia del último commit → `git show`
- Direrencia con el commit indicado → `git diff <hash>`

Logs
Alias útiles

- Grafo:

```
git config --global alias.lol 'log --graph --decorate
--pretty=oneline --abbrev-commit'
```

- Grafo con todos los branches:

```
git config --global alias.lola 'log --graph --decorate
--pretty=oneline --abbrev-commit --all --date=local'
```

Revertir cambios

- Los versionadores permiten revertir cambios.
- Todos los cambios que hayamos hecho con su correspondiente commit están en la historia del versionador.
- Incluso los archivos borrados.
- ```
  +----+----+
  A    B    C
               ^- Commit actual
  ```
- Todos los commits tienen un hash que los identifica.
- Para moverse entre commits se usa `git checkout <hash>`.
  ```

Revertir cambios
Ejemplo

- Volvemos al commit anterior:

```
git checkout <hash_del_commit_anterior>
```

- Vamos al último commit:

```
git checkout <hash_del_commit_más_nuevo>
```

- Se puede abreviar:

```
git checkout 2bb833b
```

Commits con nombre

- `HEAD` es el commit más reciente de la rama actual.
- `master` es la rama que se crea por defecto en git (en nuestro caso es la única).
-
```
+----+----+
A    B    C
          ^- (master, HEAD)
```
- `HEAD~1` es el commit anterior (B)
- `HEAD~2` es el anterior a ese, etc... (A)
- Vamos a hacer lo mismo que el slide pasado pero más fácil.

Commits con nombre
Ejemplo

- De nuevo vamos al commit anterior:

```
git checkout HEAD~1
```

- O lo que es lo mismo:

```
git checkout 2bb833b~1
```

- Volvemos al commit más nuevo (no podemos usar HEAD porque lo cambiamos):

```
git checkout master
```

Commits con nombre
Tags

- Las etiquetas indican un commit importante.
- Generalmente una versión publicada del producto.
- En git simplemente son un commit con nombre.
- Vamos a etiquetar el último commit como versión 1.0.

```
git checkout master
git tag 1.0
```

- Podemos ver los tags: `git tag`.
- Podemos usarlos para movernos entre commits:

```
git checkout HEAD~1
git checkout 1.0
```

¿Cómo trabajo con otra persona?

- Hasta ahora trabajamos de forma local.
- Podemos subir los repositorios a un servidor.
- Precisamos una cuenta en `https://gitlab.linti.unlp.edu.ar` o en `https://github.com`.

Protocolos

- Github y Gitlab admiten 2 protocolos:
  - SSH
  - HTTPS
- SSH es el más cómodo pero requiere una configuración previa.
- HTTPS es el único que funciona en la red 511-alumnos.

Configuración de SSH

Debemos hacer este procedimiento en los equipos que usemos para desarrollar.

- Crear un par de claves ssh para nuestro usuario: `ssh-keygen`.
- Podemos optar por ponerle password o no a la clave privada.
- Copiar nuestra clave **pública** en nuestro perfil de Github/Gitlab (todo menos la dirección de "e-mail" del final).

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA fernando@fernando
```

Subir nuestro código a un servidor

- Vía web crear un repositorio `octocats`.
- En nuestro repositorio agregar la url que generó Github/Gitlab como "remoto".

```
git remote add origin <url>
```

- Subir todos los cambios:

```
git push -u origin master
```

- El `-u` va solamente la primera vez, pone `origin master` por default para los `push` y `pull`.

Repositorios compartidos

- Pasamos a trabajar con otro repo.
- Podemos descargarlo con `clone`.
- Subimos cambios con `push`.
- Descargamos cambios con `pull`.
- Hagan `git clone` de:
  `git@gitlab.linti.unlp.edu.ar:lihuen/octocat_viewer.git`
  o de
  `git@github.com:fernandolopez/octocat_viewer.git`

Repositorios compartidos
Trabajando

- `mostrar.py` tiene varios errores, cada uno debe corregir 1 o 2 errores y subir los cambios al servidor.
- Se considera que alguien corrigió un error si hay un push que lo avale.

Ejemplos de status

Local:

- Untracked files
- Changes not staged for commit
- Changes to be commited

Con remotos:

- Your branch is **ahead** of 'origin/master' by 1 commit.
  $\rightarrow$ `push`
- Your branch is **behind** 'origin/master' by 1 commit...
  $\rightarrow$ `pull`
- Your branch and 'origin/master' **have diverged**, and have 1 and 1 different commit(s) each, respectively. $\rightarrow$ `pull` y después `push`

Licencia