Capacitación Git
Módulo 3

Commits

por Gonzalo Barro Gil, fuente https://git-scm.com/book/es/v2

https://github.com/GonzaloBarroGil/git-course

¿Qué es un "commit"?

- Vimos previamente que para confirmar los cambios agregados a al area de preparación creábamos un commit, que significa "cometer" algo; todo lo que veníamos haciendo o preparando previamente, lo "cometemos" o confirmamos con esta acción, no importa cual fue el proceso intermedio.

```
git add .
git commit
```

ó

```
git commit -m "... mensaje ..."
git init
```

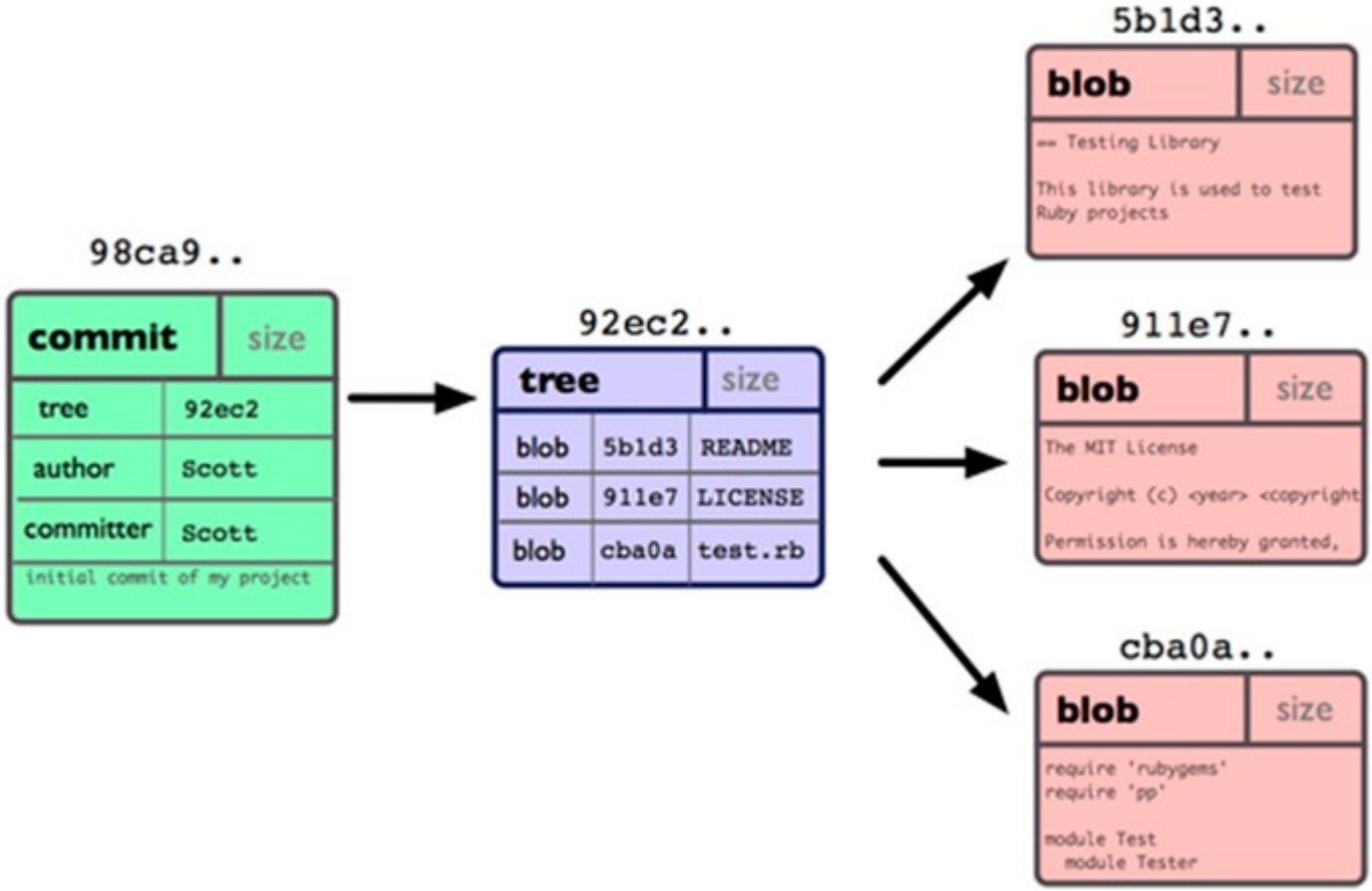
- ¿Por qué no importa el proceso intermedio desde la última confirmación o inicialización? Precisamente porque es el **commit** el punto específico que nos va definir un estado general del repositorio, una versión, una "foto", un "snapshot". Ese **commit** va a contener la copia de todos los archivos y carpetas al momento de ser "cometido".

Componentes del commit

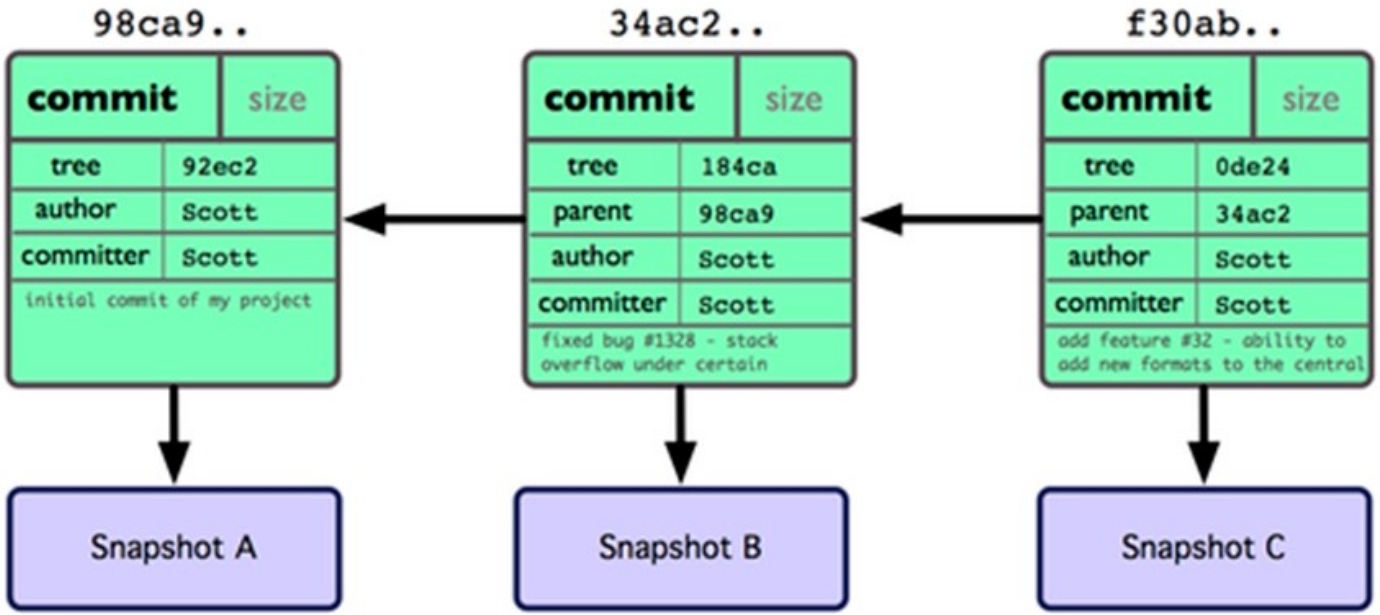
Identificador: hash SHA-1 de 40 caracteres hexadecimales.

- **tree**: objeto que representa al directorio principal con su contenido.
- **parent**: objeto commit padre (si no es el commit inicial).
- **author**: se obtiene de los valores de author.name o user.name.
- **committer**: se obtiene de los valores de committer.name o user.name.
- Mensaje: cadena de caracteres que representa las tareas resueltas en la confirmación. Debe ser semántico, descriptivo pero conciso, preferiblemente una oración simple, por ejemplo: "agregar título al documento".

Contenido del commit



Cadena de commits



- Cada commit (excepto el inicial) tiene una referencia al id de su padre, no al de su hijo, y como dijimos antes representa una versión o estado en un momento dado.
- De esta manera se define no solo la relación entre un commit y otro, sino además un orden de versiones que representa la evoución o **historia** del proyecto.

Historia de commits

- La historia, "bitácora" o log se puede consultar con:

```
git log
```

- Esto nos mostrará la informción relativa a cada commit, comenzando por el último (recordar que las referencias son hacia las precedencias o padres) en la forma:

```
commit id del commit
Author: user.name <user.email>
Date: Fecha y hora del commit

       texto del mensaje
```

- Modificadores:
  - --all (commits de todas las ramas)
  - --decorate (información del HEAD, ramas contenedoras locales y remotas)
  - --oneline (una línea por commit)
  - --pretty (formato oneline, short, full o fuller)
  - --graph (representación gráfica de ramas)

Modificar un commit

- Una confirmación, plasmada en el objeto **commit**, puede ser modificada posteriormente por medio del modificador de reconfirmación:

```
git commit --amend
```

- El resultado de esto será:
  - El identificador SHA-1 será modificado, es decir será un objeto commit nuevo.
  - El commit incluirá ahora los tree y blob con las modificaciones adicionales.
  - La referencia al commit padre será la misma, pese a ser un nuevo commit.
  - Nos abrirá el editor de mensajes para mantener o modificar el mensaje anterior. Esto se puede evitar con el modificador de no edición:

```
git commit --amend --no-edit
```

  - Esto confirmará directamente dejando el mensaje anterior.

Verificar estados

- Estos estados se pueden verificar con

```
git status
```

- Como primera medida nos informará en qué branch estamos. Por ejemplo:

```
On branch master
...
```

- Seguidamente nos agrupará por estado todos los cambios:
  - No rastreados:

```
Untracked files:
...
```

  - Preparados:

```
Changes to be committed:
...
```

  - Modificados:

```
Changes not staged for commit:
...
```

  - No modificados:

```
nothing to commit, working directory clean
```

Deshacer cambios

HEAD: *apuntador o índice de cabecera. Indica cuál es el commit y branch actual, a partir de donde se recrea el área de trabajo.*

- **Sacar cambios del área de preparación (Stage)**:

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  modified:    archivo
```

```
git reset HEAD archivo
```

  - Esto revierte los cambios sacándolos del stage, dejándolos como cambios pendientes sobre la última versión a la que apunta el HEAD.

Deshacer cambios

- **Deshacer cambios sobre un archivo**:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be
  committed)
  (use "git checkout -- <file>..." to discard changes in
  working directory)

  modified:    archivo
```

```
git checkout -- archivo
```

- Esto dejará el archivo en el mismo estado que estaba en el último commit apuntado por el HEAD.

Deshacer un commit

```
git reset --soft HEAD~1
```

- Esto moverá el HEAD al commit padre del actual, pero dejará el **stage** tal como estaba antes del commit actual.

```
git reset --mixed HEAD~1
```

- Esto moverá el HEAD al commit padre del actual, pero dejará los cambios posteriores sin preparar.

```
git reset --hard HEAD~1
```

- Esto moverá el HEAD al commit padre del actual y dejará el área de trabajo limpia.

Etiquetado

- Etiquetar commit actual:

```
git tag v1.0.1
```

  - anotaciones y mensajes:

```
git tag -a v1.0.1 -m "parches de seguridad"
```

- Etiquetar commit anterior

```
git tag -a v1.0.0 a996393
```

- Listar etiquetas:

```
git tag
```

- Mostrar información de etiqueta:

```
git tag v1.0.1
```

Alias

- Para algunos comandos que efectuamos frecuentemente se puede definir una sintaxis abreviada. Esta debe ser fácil de memorizar y no deber reemplazar un método existente:

  - comando simple:

```
git config --global alias.c commit
git c

git config --global alias.ch checkout
git ch
```

  - comando compuesto:

```
git config --global alias.cm "commit -m"
git cm "mensaje del commit"

git config --global alias.adog "log --all --decorate --oneline
--graph"
git adog
```