Git Internals

Jesús Espino García

jesus.espino@kaleidos.net
@jespinog

Kaleidos
beautiful code

19 de Abril de 2013

¿Por qué?

- La interfaz de git es de bajo nivel.
- Conocer git bien da mucho poder.
- El poder está ahí aunque no lo conozcamos.
- Un gran poder conlleva una gran responsabilidad.

Conceptos básicos

- Porcelain (Porcelana).
- Plumbing (Cañerias).
- Objetos
- Referencias
- Head
- Working copy

Contenido de .git

- Ficheros.
  - HEAD
  - index
  - config
- Directorios.
  - objects
  - refs
  - hooks
  - info

Objetos

- Bloque de datos almacenado en git
- Referenciado por el sha1 de su contenido
- Almacenados en el directorio `.git/objects/` (o en packs).
- Hay 4 tipos de objetos en git (blob, tree, commit, tag).

blobs

- Será el nodo hoja de nuestros árboles.
- Será equivalente (normalmente) a nuestros ficheros.

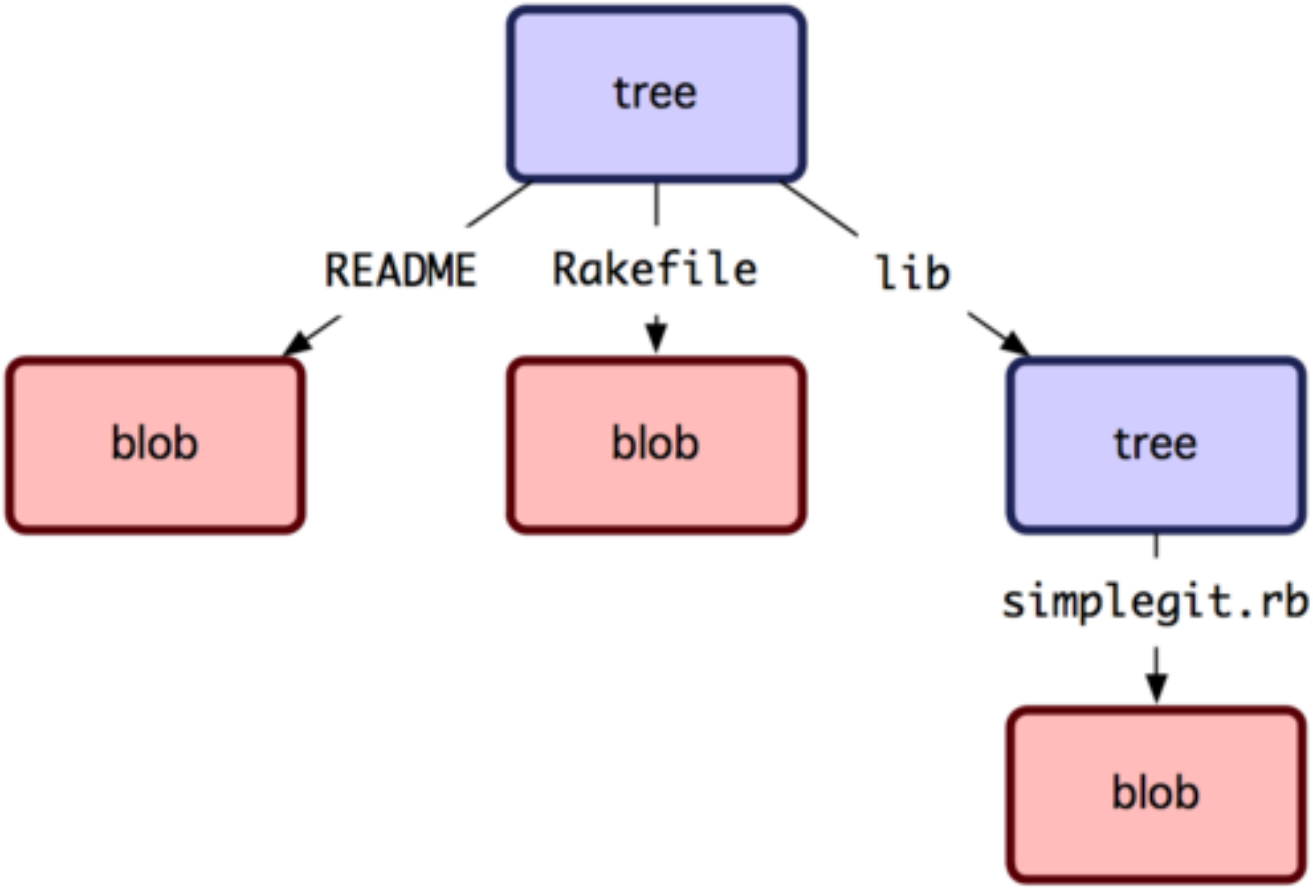
Ejemplo de almacenar blob

- `git init repo; cd repo`
- `echo 'version 1' | git hash-object -w --stdin`
- `find .git/objects -type f`
- `git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30`
- `echo 'version 2' | git hash-object -w --stdin`
- `find .git/objects -type f`
- `git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a`

trees

- Es un directorio de referencias a blob y otros trees.
- Almacena referencias (sha1 de objetos) y metadatos.

diagrama de ejemplo de trees



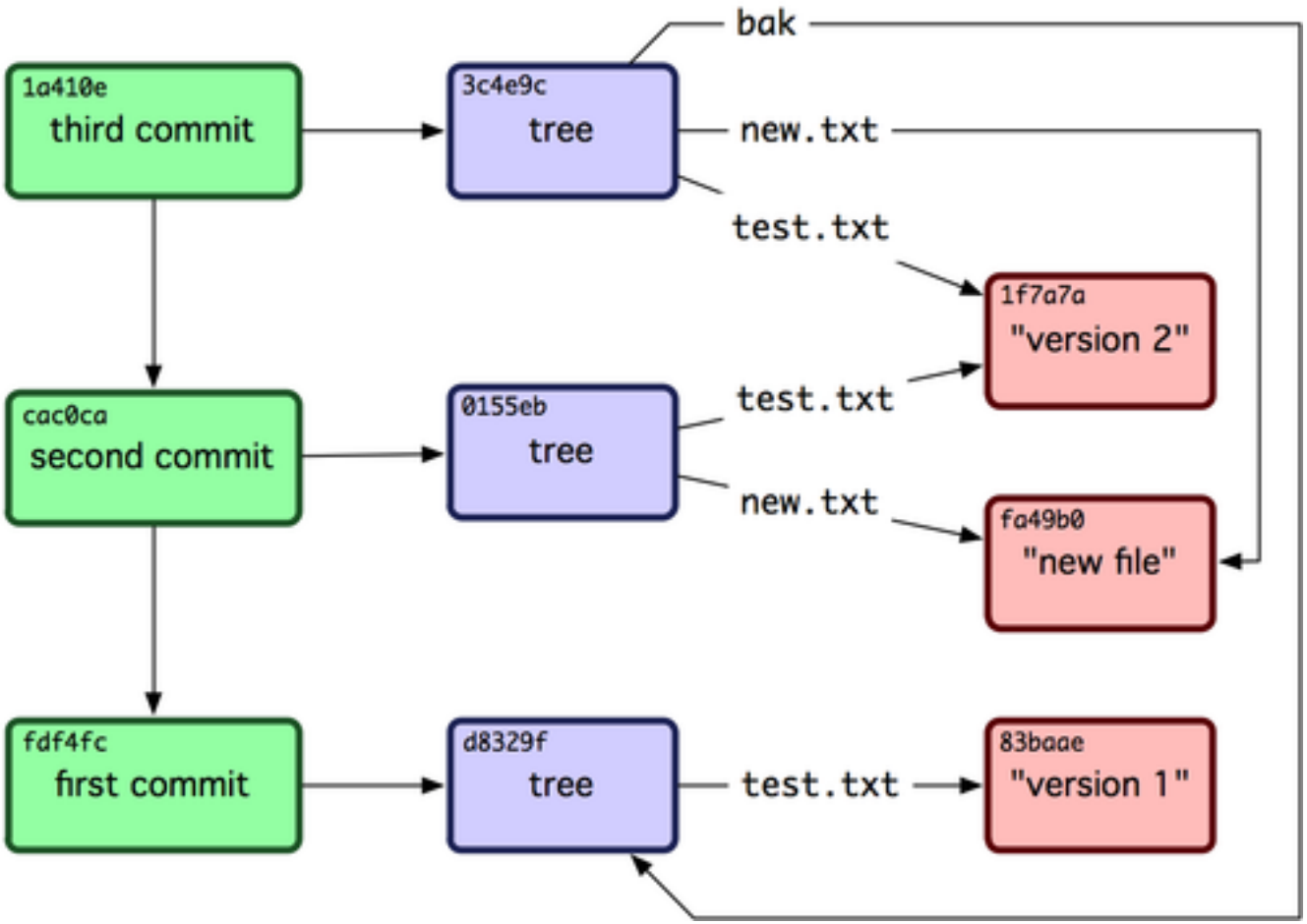
Ejemplo de almacenar tree

- `git init repo; cd repo`
- `git update-index --add --cacheinfo 100644 \ 83baae61804e65cc73a7201a7252750c76066a30 test.txt`
- `git write-tree`
- `find .git/objects -type f`
- `git cat-file -p d8329fc1cc938780ffdd9f94e0d364e0ea74f579`

commits

- Almacena una referencia a un tree.
- Almacena una referencia a su commit padre.
- Almacena metadatos del commit (autor, fecha, mensaje...)

diagrama de ejemplo de commits



Ejemplo de commit

- `git init repo; cd repo`
- `echo "Version 1" > fichero.txt; git add fichero.txt`
- `git commit -m "Version 1"`
- `find .git/objects -type f`
- `git cat-file -p HEAD`
- `git cat-file -p b7c0dba424af1e98a3570f8125476126129e5c32`
- `git cat-file -p fb8247c7b27ae4cad9e7e3e66ba95126658ea7c2`

tags

- Almacena una referencia a un commit.
- Almacena metadatos del commit (autor, fecha, nombre...)

Objects storage

- Se añade una cabecera con el tipo de objeto y la longitud del mismo.
- Se concatena con los datos que se van a almacenar
- Se calcula su sha1 que se utilizará como nombre del objeto.
- Se comprime con zlib.
- Se almacena en `.git/objects/XX/XXXXXX...`

Ejemplo de lectura directa

- `git init repo; cd repo`
- `echo "Version 1" > fichero.txt; git add fichero.txt`
- `git commit -m "Version 1"`
- `find .git/objects -type f`
- `git cat-file -p HEAD`
- `git cat-file -p b7c0dba424af1e98a3570f8125476126129e5c32`
- `git cat-file -p fb8247c7b27ae4cad9e7e3e66ba95126658ea7c2`
- `cat .git/objects/fb/8247c7b27ae4cad9e7e3e66ba95126658ea7c2 \ | zlib-flate -uncompress`

Packfiles

- Paquetes de objetos.
- Periódicamente git empaqueta los objetos en packs (gc).
- Se almacenan en `.git/objects/pack/`.
- Hay un listado de packs en `.git/objects/info/packs`.
- Cada pack tiene su índice en `.git/objects/pack/`.

Referencias

- Están en `.git/refs` principalmente.
- Son punteros a objetos.
- Contienen el id del objeto al que apuntan.

branchs

- Están en `.git/refs/heads`

HEAD

- Está en `.git/HEAD`
- Es una referencia simbólica que apunta a la referencia de la rama actual.

tags

- Están en `.git/refs/tags`
- Son referencias a commits o a objetos tag.
- Los tags simples son una referencia directa a un commit.
- Los tags con anotaciones son referencias a un objeto tag que apunta a un commit.

remotes

- Están en `.git/refs/remotes`
- Contiene las referencias de mis remotes.
- Se actualizan cuando hago push o fetch.

Reflog

- Log de cambios de las referencias
- Cada referencia tiene su log de cambios en `.git/logs/`

Refspects

- Forma de definir la relación entre las referencias de diferentes origins
- Tienen el formato `[+]<src>:<dst>`
- Opcionalmente se pone un `+` para actualizar la referencia cuando no hay fast-forward.
- Las referencias pueden tener `*` para definir "todo lo que haya en un directorio"
- No se permite el uso de `*` para selecciones parciales de referencias.
- Ejemplo: `+refs/heads/*:refs/remotes/origin/*`

Pulling with refspects

- Ejemplo: `git fetch origin master:refs/remotes/origin/mymaster`
- Descarga la referencia master del origin a `refs/remotes/origin/mymaster` en local

Pushing with refspects

- Ejemplo: `git push origin master:refs/heads/qa/master`
- Envía la referencia master local al `refs/heads/qa/master` en origin

Borrando referencias

- Ejemplo: `git push origin :topic`
- Borra la referencia topic en origin

init

Crea un .git con datos básicos.

- Un fichero config por defecto.
- Los directorios refs, objects e info.
- Un `HEAD` apuntando a la referencia master.
- Y poco más

add

- Añade el fichero a la base de datos de Objetos.
- Añade el fichero al index.

commit

- Añade el tree apuntando al árbol de ficheros al que apunte el index a la base de datos de objetos.
- Añade el commit apunte al tree recién añadido a la base de datos de objetos.
- Modifica el `HEAD`.
- Modifica la referencia de la rama actual.

checkout

- Compara el tree del commit actual y el tree del commit destino.
- Extrae los objetos diferentes entre ambos a la working copy.
- Si existen ficheros modificados intenta hacer el merge y conservar las modificaciones.
- Modifica el `HEAD`.

reset

- Modifica la referencia a la que apunte `HEAD` y la apunta al commit que le digas.
- Depende del tipo de reset modifica el index para ajustarlo al nuevo commit.
- Depende del tipo de reset modifica la working copy para ajustarlo al nuevo commit.

rebase

- Se posiciona en la rama destino.
- Añade los commits de la rama origen uno a uno en la rama destino.
- Si hay conflicto los resuelve (automática o manualmente) modificando el commit.
- Actualiza la referencia de ambas ramas para apuntar al último commit.

merge

- Pregunta por el origen común de las ramas a mergear.
- Calcula las diferencias que genera cada rama.
- Intenta mezclar las diferencias en un nuevo commit.
- Si hay conflicto los resuelve (automática o manualmente).
- Genera un objeto commit (con varios parents) con las diferencias de ambos aplicadas.
- Actualiza la referencia de la rama destino al nuevo commit.

stash

- Almacena todos los blobs y trees de mis cambios actuales.
- Genera y almacena un commit a partir de los cambios actuales.
- Actualiza la referencia `.git/refs/stash`.
- Esta acción añade una entrada al reflog de la referencia stash.
- Cuando hacemos un `stash pop` aplicamos el commit y eliminamos la ultima entrada del reflog.

¿De qué no hemos hablado?

- Más comandos de porcelana.
- Comandos de plumbing.
- Protocolos de transferencia.
- Mantenimiento y recuperación de datos.
- fsck
- Other files internal format (index, packs, pack-idx...)

Referencias

- http://git-scm.com/ - Web oficial de git.
- http://git-scm.com/book - ProGit (El libro de Git).
- Documentation/technical - Technical doc in the git respoitory.
- http://gitguys.com/ - Página sobre git.
- http://github.com/ - Servicio de git por excelencia.
- http://bitbucket.org/ - Servicio de git de repositorios privados gratis.

Dudas

. . .