Taller de git

SEMANA 3

Objetivos de hoy

- Poder ver cambios locales
- Poder ver cambios en la historia
- Volver en la historia
- Crear ramas
- Integrar ramas

Antes de empezar:

- Creen un par de commits en su repositorio
- Agreguen cambios a su working directory
- Agreguen algunos de estos cambios al staging area

Consultando la historia

- Ya que tenemos nuestros commits hechos, es importante saber:
  - ¿Qué se hizo?
  - ¿Cuándo se hizo?
  - ¿Quién lo hizo?
- Para ello usamos el comando:
  - git log
- El resultado muestra:
  - Todos los commits
  - Orden cronológico inverso
  - Responde las 3 preguntas de arriba*
  - Nos muestra el SHA del commit
  - * Siempre y cuando el autor nos haya dejado un mensaje de commit apropiado

Banderas útiles

- -p muestra la diferencia entre 2 commits
- --stat muestra un resumen de los cambios
- --pretty=<options> muestra cada cambio en una línea
  - Options = oneline, short, full, fuller
- --pretty=format:"formato"
- --graph muestra el grafo de commits del proyecto
- --abbrev-commit muestra sha's cortos
- --relative-date nos dice cuanto ha pasado desde el commit

- Git log puede tomar un hash como parámetro y empezará desde ese ese hash

Tabla de formatos

Option	Description of Output
`%H`	Commit hash
`%h`	Abbreviated commit hash
`%T`	Tree hash
`%t`	Abbreviated tree hash
`%P`	Parent hashes
`%p`	Abbreviated parent hashes
`%an`	Author name
`%ae`	Author e-mail
`%ad`	Author date (format respects the --date=option)
`%ar`	Author date, relative
`%cn`	Committer name
`%ce`	Committer email
`%cd`	Committer date

- Actividad:

- Definan un formato con por lo menos 3 elementos y ejecuten el comando para ver la historia de esa manera

Opciones de filtrado

Option	Description
`-(n)`	Show only the last n commits
`--since, --after`	Limit the commits to those made after the specified date.
`--until, --before`	Limit the commits to those made before the specified date.
`--author`	Only show commits in which the author entry matches the specified string.
`--committer`	Only show commits in which the committer entry matches the specified string.
`--grep`	Only show commits with a commit message containing the string
`-S`	Only show commits adding or removing code matching the string

Consultando cambios

- Hemos visto que git status nos dice qué archivos hemos cambiado pero no qué cambiamos
- En ocasiones es muy útil saber los cambios exactos que hemos hecho antes de hacer un add o un commit
- El comando para esto es
  - git diff

Git diff

Parámetros	Resultado
Sin parámetros	Compara el commit contra el working directory
--cached	Compara el commit contra el staging área
--staged	
<rama>	Compara la rama actual con la rama especificada
<commit>	Compara el commit especificado con el actual

Veremos más opciones conforme avancemos con ramas, selectores y etiquetas

Git diff en una herramienta externa

- Se puede configurar una herramienta externa para hacer las comparaciones de manera similar a como configuramos el editor
- Usualmente debemos de crear un script que llame al editor como queramos, ya que git le pasará 7 parámetros:
  - path old-file old-hex old-mode new-file new-hex new-mode
- Un ejemplo de script:
  - /usr/bin/opendiff "$2" "$5" -merge "$1"
- El nombre de la configuración es:
  - diff.external
- Algunas herramientas están mencionadas en:
  - https://www.git-tower.com/blog/diff-tools-windows/

Volviendo en la historia

- En ocasiones es útil volver a un determinado punto en la historia.
- Para hacer esto necesitamos el SHA del commit al que queremos volver

- git checkout <SHA>
- git checkout <rama>
- git checkout <tag>

Si hay cambios en staging área o working directory, git podría rechazar el checkout para evitar perder los cambios

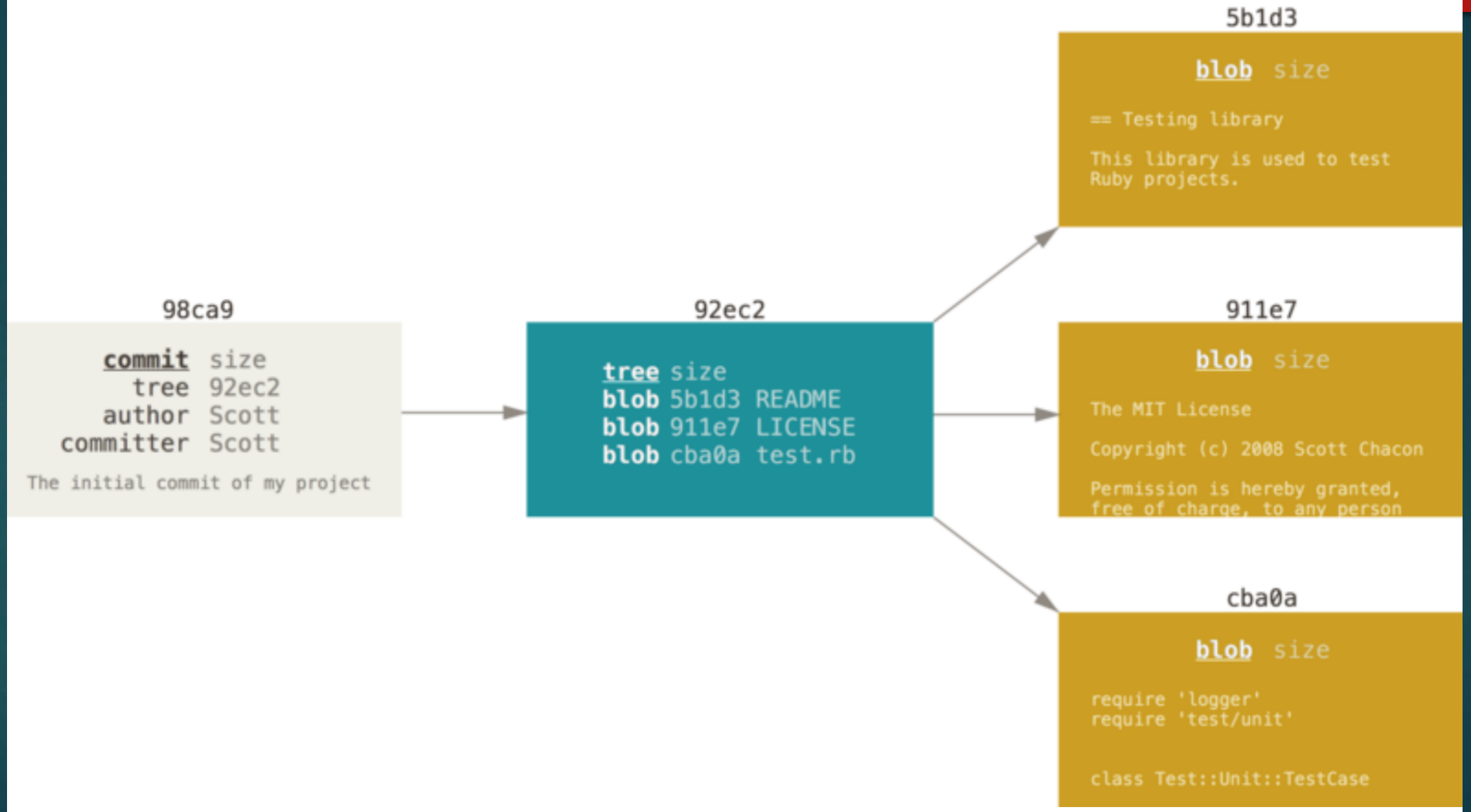
Checkout cambiara el apuntador a HEAD, por lo que para hacer un commit, si usamos un tag o un sha, debemos de regresar a master o crear otra rama

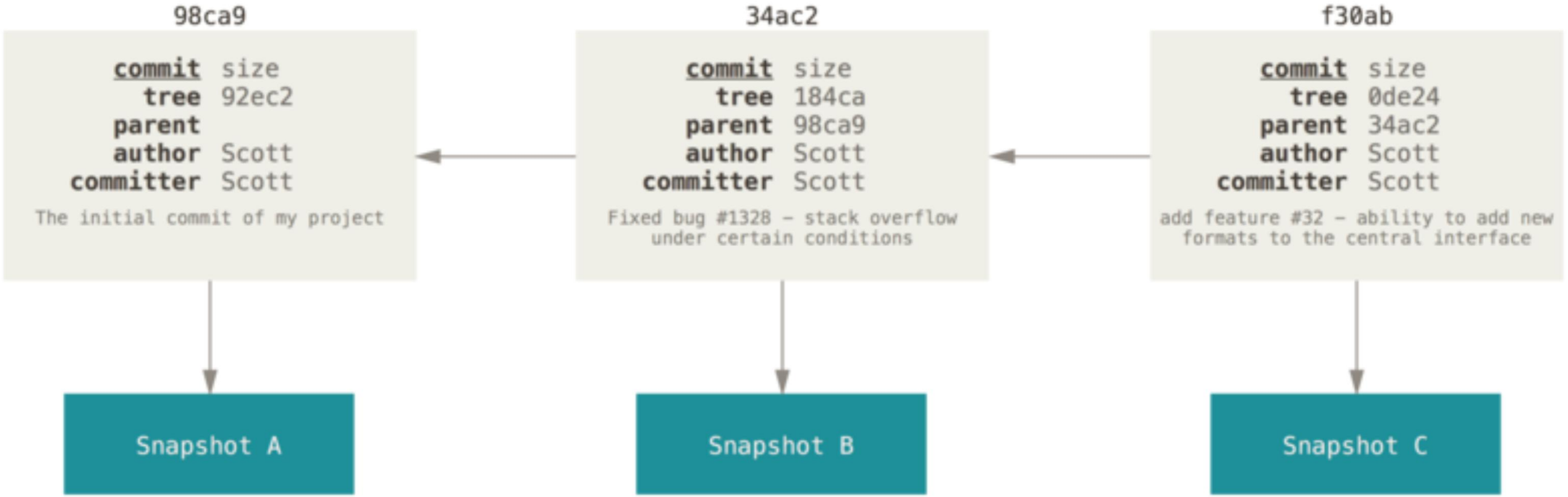
Deshaciendo cambios

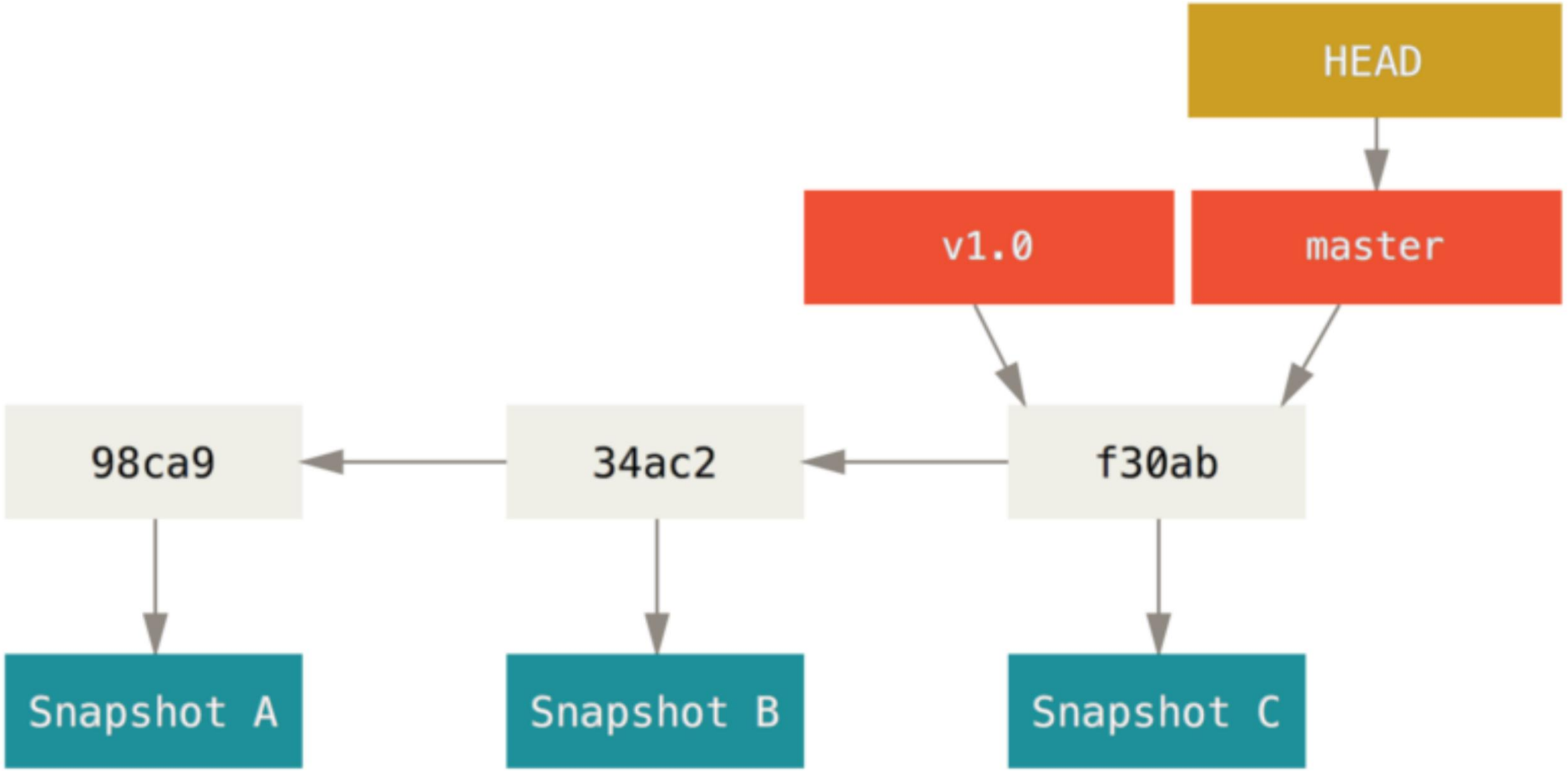
- Si debemos de deshacer un cambio en la historia, podemos pedirle a git que lo deshaga
- Git NO borrará el commit del cambio, en lugar de ello, hará un nuevo commit con el cambio opuesto al introducido por el commit
- Para ejecutar esto, debemos correr el siguiente comando:
  - git revert <SHA>
  - Opcionalmente, podemos usar –n para que el commit no se haga de forma automática
- Nota: El comando solo servirá si el commit tiene 1 padre. Si no, se debe especificar que padre usar con -m

¿Qué son las ramas?

REPASO RÁPIDO DEL FUNCIONAMIENTO DE GIT


98ca9
commit size
tree 92ec2
author Scott
committer Scott
The initial commit of my project
92ec2
tree size
blob 5b1d3 README
blob 911e7 LICENSE
blob cba0a test.rb
5b1d3
blob size
== Testing library
This library is used to test
Ruby projects.
911e7
blob size
The MIT License
Copyright (c) 2008 Scott Chacon
Permission is hereby granted,
free of charge, to any person
cba0a
blob size
require 'logger'
require 'test/unit'
class Test::Unit::TestCase


98ca9
commit size
tree 92ec2
parent
author Scott
committer Scott
The initial commit of my project
34ac2
commit size
tree 184ca
parent 98ca9
author Scott
committer Scott
Fixed bug #1328 - stack overflow under certain conditions
f30ab
commit size
tree 0de24
parent 34ac2
author Scott
committer Scott
add feature #32 - ability to add new formats to the central interface
Snapshot A
Snapshot B
Snapshot C


HEAD
v1.0
master
98ca9
34ac2
f30ab
Snapshot A
Snapshot B
Snapshot C

¿Por qué crear ramas?

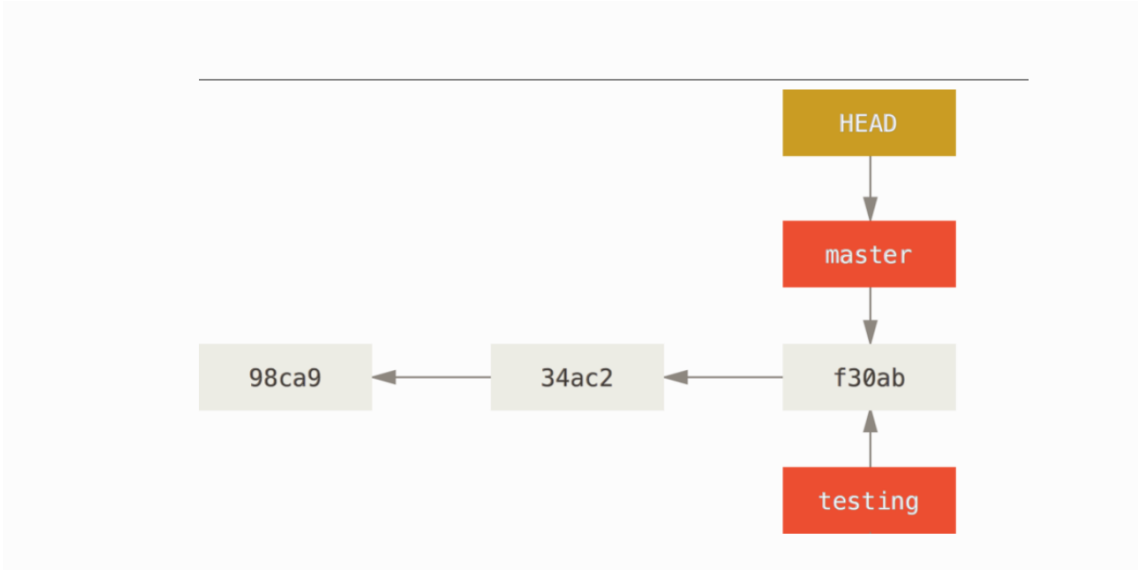
- Para mantener un orden del proyecto
- Para trabajar colaborativamente de manera eficiente
- Para tener diferentes niveles de estabilidad
- Para no romper algo que ya está funcionando
- Para poder arreglar problemas de producción, sin necesidad de volver a la historia o exponer cambios nuevos

Creando ramas

- Para crear una rama debemos de ejecutar el siguiente comando:
  - git branch <nombre>
- Para ver las ramas que tenemos podemos correr
  - git branch –l
- Un * indicará en que rama nos encontramos. Git status también nos facilita esa información.
- Nota: La rama se creará en el lugar donde esté HEAD

Cambiando de rama

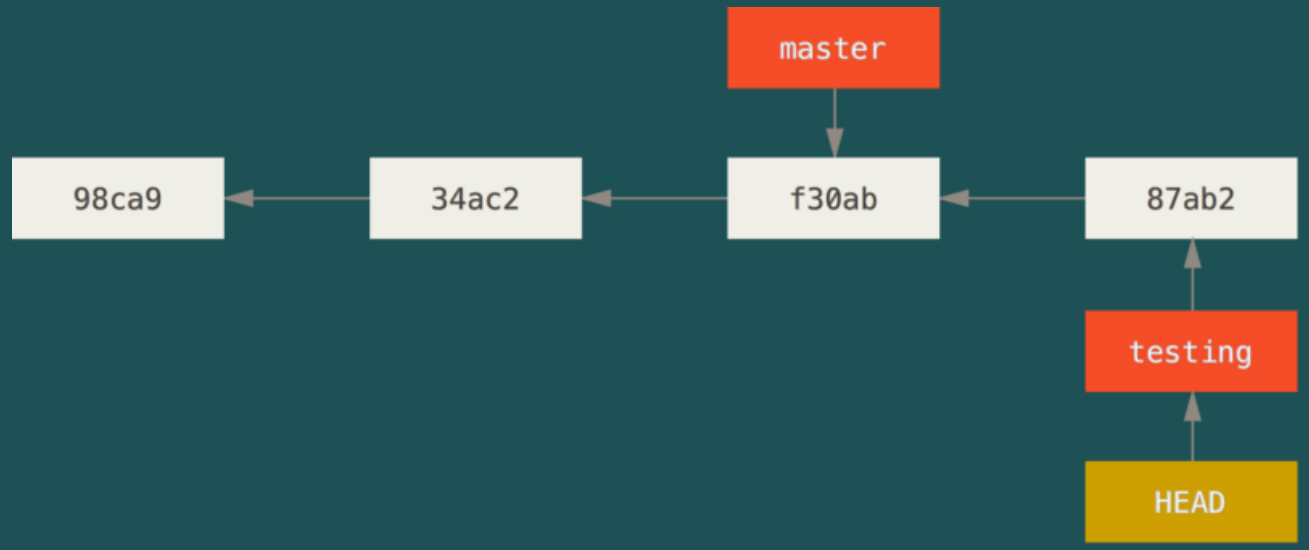
- Al igual que al volver en la historia, podemos cambiar de rama con el comando git checkout.
- Git podrá rechazar el comando si hubiera cambios que no se aplican de manera limpia en la rama a la que vamos
- * Si desean crear una rama y cambiar inmediatamente a ella, se puede correr el comando:
  - git checkout –b <rama>
- Actividad:
  1. Creen una nueva rama
  2. Cambien a esa rama
  3. Vuelvan varios commit atrás
  4. Creen otra rama en ese punto
  5. Comparen el contenido de su rama actual con master



GIT CHECKOUT EN DIAGRAMAS

Integrando ramas (básico)

- Cuando tenemos una rama es esperado que ésta tenga algunos commits que master no tenga
- Estos cambios deberán volver a master en algún momento
- Para volver, debemos integrar los cambios de testing en master



Pasos

- Rama A = La rama de la que integramos
- Rama B = La rama de la que integramos

1. Hacer todos los cambios que necesitemos en nuestra rama A y ponerlos en el repositorio (.git)
2. Si nuestra rama A está detrás de B, traer los cambios de A en B
3. Volver a B y traer los cambios de A

- El comando para integrar cambios de otra rama es
  - git merge <rama>

Si aún hay tiempo

- Generando un conflicto
- Formas de integración: Recursive vs Fast forward
- Consultando el estatus de las ramas
- --merged --no-merged
- Cómo trabajar con git
- Git remoto