Taller de Git y GitHub desde cero

Iván Martínez Ortiz

Facultad de Informática
Universidad Complutense

Por qué empecé a utilizar el control de versiones

GeneralBeca.java

Iván Martínez Ortiz

¿Qué es el Control de Versiones ?

- Gestión de ficheros a lo largo del tiempo
  - Evolución del trabajo
- Gestión del versionado de los ficheros
  - Si un archivo se corrompe o hemos cometido un fallo volvemos atrás
- Mecanismo para compartir ficheros
- Habitualmente tenemos nuestro propio mecanismo y modelo de trabajo
  - Versionado: Documento.docx, Documento_v2.docx
  - Herramientas: Dropbox, Adjunto correo.

¿Por qué un Sistema de Control de Versiones?

Iván Martínez Ortiz

¿Por qué un Sistema de Control de Versiones?

- Las metodologías/mecanismos particulares no escalan para proyectos de desarrollo
- Un SCV permite
  - Crear copias de seguridad y restaurarlas
  - Sincronizar (mantener al día) a los desarrolladores respecto a la última versión de desarrollo
  - Deshacer cambios
    - Tanto problemas puntuales, como problemas introducidos hace tiempo
  - Gestionar la autoría del código
  - Realizar pruebas (aisladas)
    - Simples o utilizando el mecanismo de branches/merges

Vocabulario de trabajo con los SCV

- Elementos básicos
  - **Repositorio**
    - Almacén de que guarda toda la información del proyecto.
    - Habitualmente tiene estructura de árbol.
  - **Servidor**
    - Máquina donde está alojado el Repositorio.
  - **Working Copy/Working Set** (Copia de trabajo)
    - Copia local donde el desarrollador trabaja.
  - **Trunk/Main/master** (Rama principal):
    - Localización dentro del repositorio que contiene la rama principal de desarrollo.

Vocabulario de trabajo con los SCV (II)

- Operaciones básicas
  - **Add**
    - Añade un archivo para que sea rastreado por el SCV.
  - **Revisión**
    - Versión de un archivo/directorio dentro del SCV
  - **Head**
    - Última versión del repositorio (completo o de una rama)
  - **Check out**
    - Creación de una copia de trabajo que rastrea un repositorio
  - **Check in / Commits**
    - Envío de cambios locales al repositorio
    - Como resultado cambia la versión del archivo(s)/repositorio
  - **Mensaje de Check in/log**
    - Todo Check in tiene asociado un mensaje que describe la finalidad del cambio
    - Puede estar asociado al un sistema de gestión de incidencias

Vocabulario de trabajo con los SCV (III)

- Operaciones básicas
  - **Log** (Historia)
    - Permite visualizar/revisar la lista de cambios de un archivo/repositorio
  - **Update/Syncronize/fetch&pull** (Actualizar)
    - Sincroniza la copia de trabajo con la última versión que existe en el repositorio.
  - **Revert/Reset** (Deshacer)
    - Permite deshacer los cambios realizados en la copia de trabajo y dejar el archivo/recurso en el último estado conocido del repositorio.

Vocabulario de trabajo con los SCV (IV)

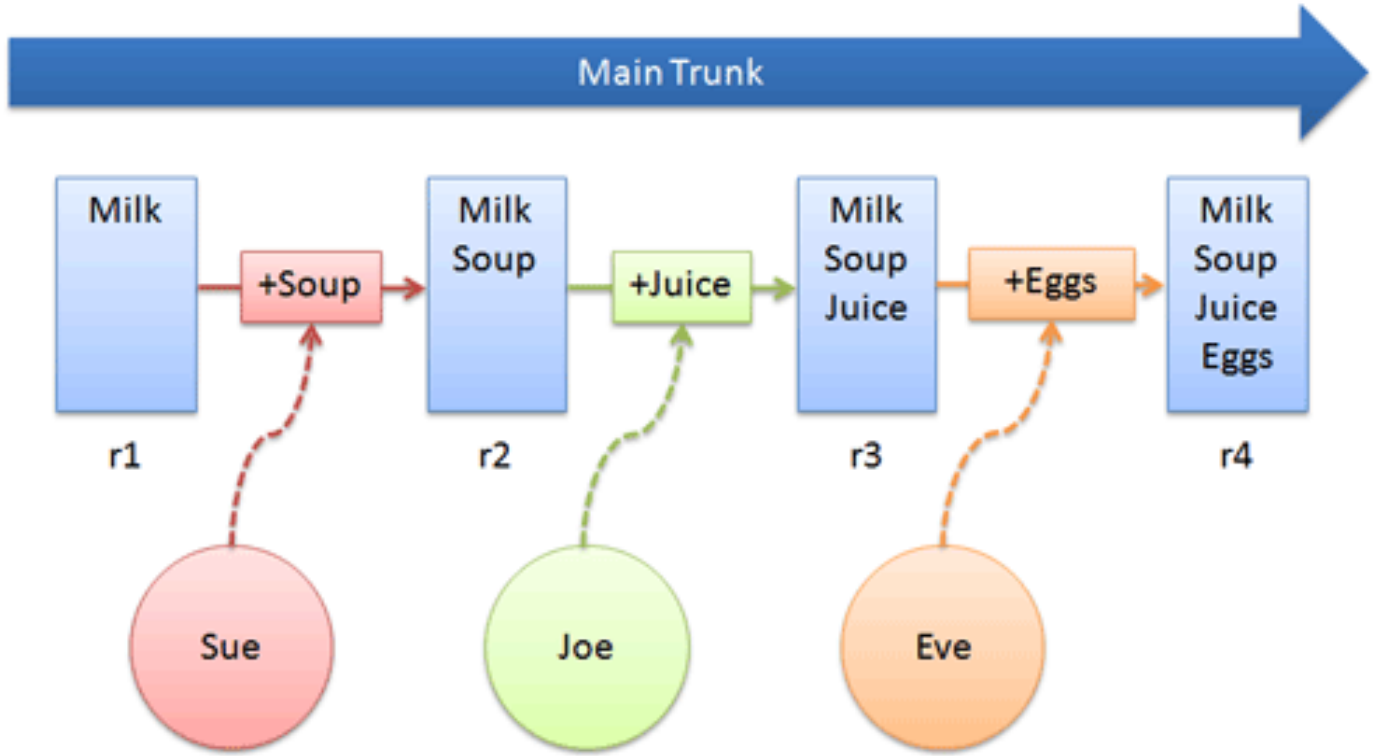
- Operaciones Avanzadas
  - **Branching** (ramas)
    - Permite crear una copia de un archivo/carpeta rastreada
    - Permite desarrollar en paralelo en otra "rama" pero dejando constancia de la relación que existe con la rama original.
  - **Diff/Change/Delta/** (Cambio)
    - Permite encontrar las diferencias entre dos versiones del repositorio.
    - Se puede generar un parche que permitiría pasar de una versión a otra.
  - **Merge/Patch**
    - Aplica los cambios de un archivo a otro
    - Utilizado habitualmente para mezclar branches
  - **Conflict** (Conflicto)
    - Problema que surge cuando varios desarrolladores modifican el mismo recurso y los cambios se solapan.

Tipos de Sistemas de Control de Versiones

- Centralizados
  - Existe un servidor centralizado que almacena el repositorio completo
  - La comunicación/colaboración entre desarrolladores se lleva a cabo (forzosamente) utilizando el repositorio centralizado
  - Son más simples de usar
  - Los modelos de trabajo son más restringidos

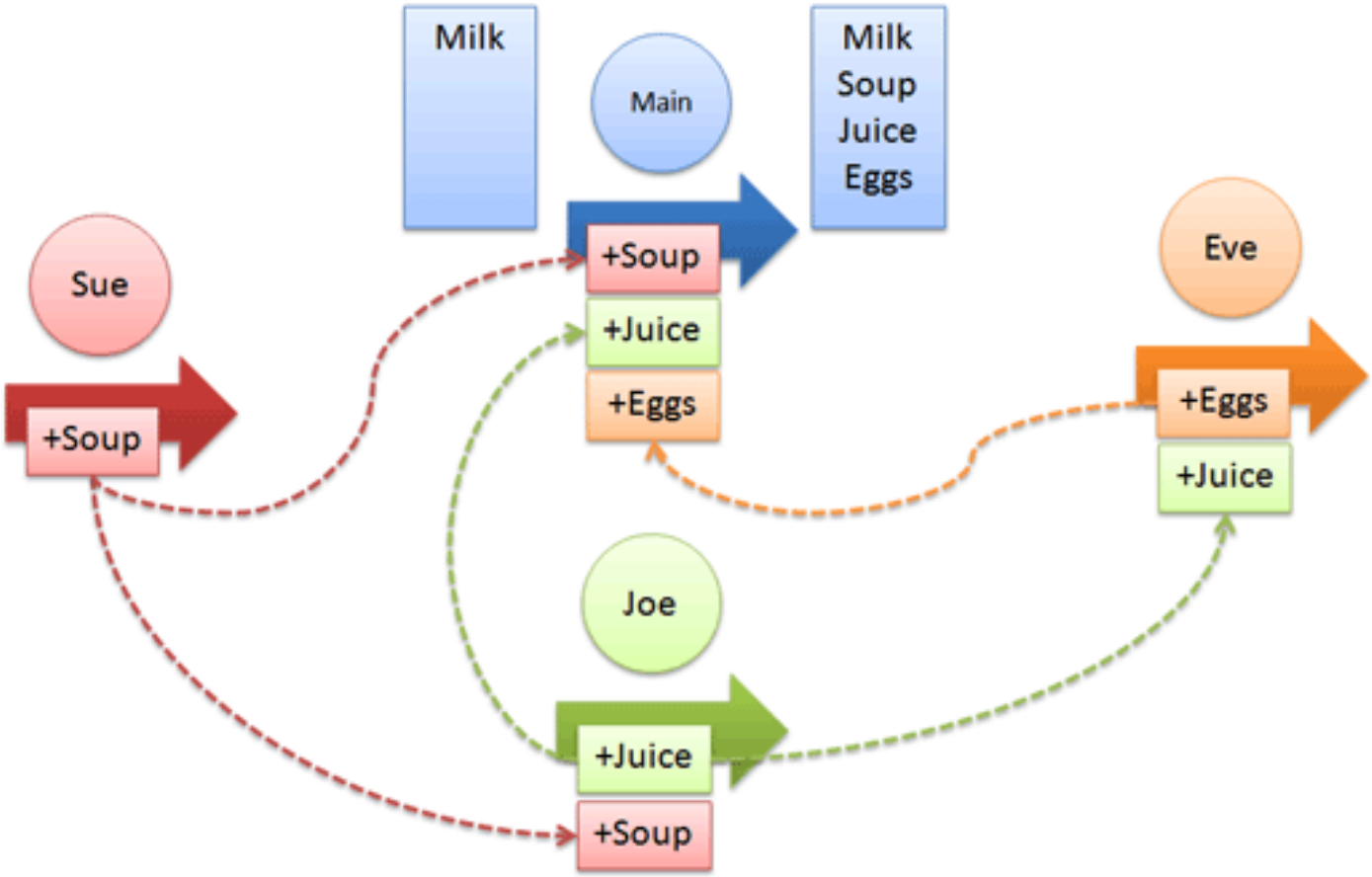
- Distribuidos
  - Cada desarrollador contiene una copia completa de todo el repositorio
  - Los mecanismos de comunicación/colaboración entre desarrolladores son más abiertos
  - Son (un poco) más difíciles de utilizar que los sistemas centralizados
  - Los modelos de trabajo son más flexibles
  - "Los branches/merges son más simples"

SCV Centralizado



Iván Martínez Ortiz

SCV Distribuido



Iván Martínez Ortiz

Herramientas SCV

- Centralizados
  - **Subversion** (SVN)
    - http://subversion.apache.org, http://subversion.tigris.org/
  - Concurrent Version System (CVS)
    - http://www.nonfnu.org/cvs/
  - Microsoft Visual Source Safe
  - Perforce
- Distribuidos
  - **Git**
    - http://git-scm.com
  - Mercurial
    - http://hg-scm.com
  - Bazaar, DARCS

Políticas de Control de Versiones

- Para aprovechar los SCV es necesario
  - Establecer una política para el control de versiones para los proyectos
    - Estructura del repositorio
    - Política para la rama principal
    - …
  - Documentar el desarrollo
    - Utilizando alguna herramienta de gestión de seguimiento: Trac
- Es interesante adoptar un modelo de trabajo que sea adecuado para el equipo de desarrollo
  - El modelo de trabajo del equipo de desarrollo puede influir en la elección del SCV a utilizar.

Metodología básica de trabajo

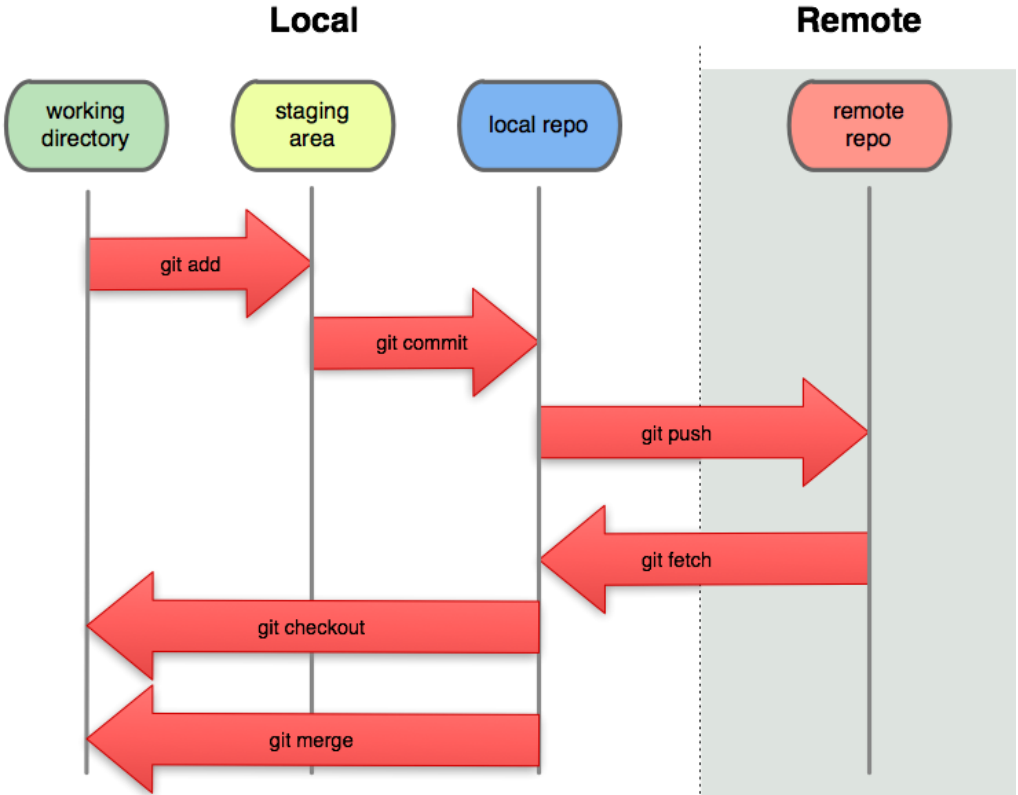
- La primera vez
  1. Creación del repositorio del proyecto (Opcional)
     - Importación inicial del código del proyecto (Opcional)
  2. Crear una copia de trabajo del repositorio
  3. Modificar la copia de trabajo
  4. Envío de cambios al repositorio
- Siguientes ocasiones
  1. Actualizar el repositorio
  2. Modificar la copia de trabajo
  3. Envío de cambios al repositorio

Por qué Git

- Branch locales "baratos"
  - Fáciles de crear y borrar
  - No tienen por qué ser públicos
  - Útiles para organizar el trabajo y los experimentos

Por qué Git

- Todo es local
  - Operaciones más rápidas
  - Puedes trabajar sin red
  - Todos los repositorios de los desarrolladores son iguales
    - En caso de emergencia puede servir de backup

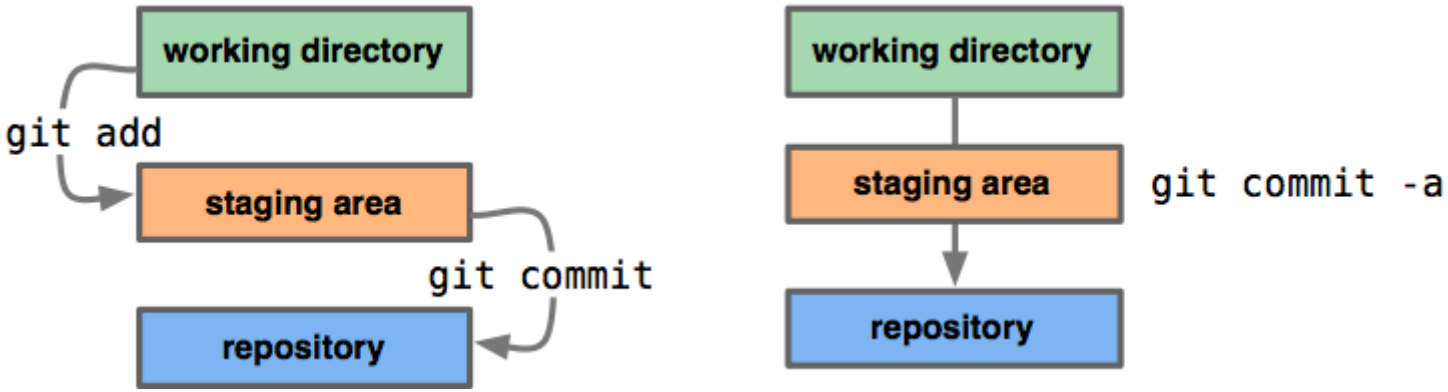


Por qué Git

- Git es rápido
  - Comparado con otras herramientas
- Git es pequeño
  - Pese a que es una copia de todo el repositorio
  - En algunos casos incluso comparándolo con svn
  - Notas
    - Si dos archivos son iguales sólo se guarda el contenido de 1.
    - El contenido de los archivos se guarda comprimido
    - Periódicamente se compactan los archivos
      - Se generan deltas entre las diferentes versiones de los archivos.

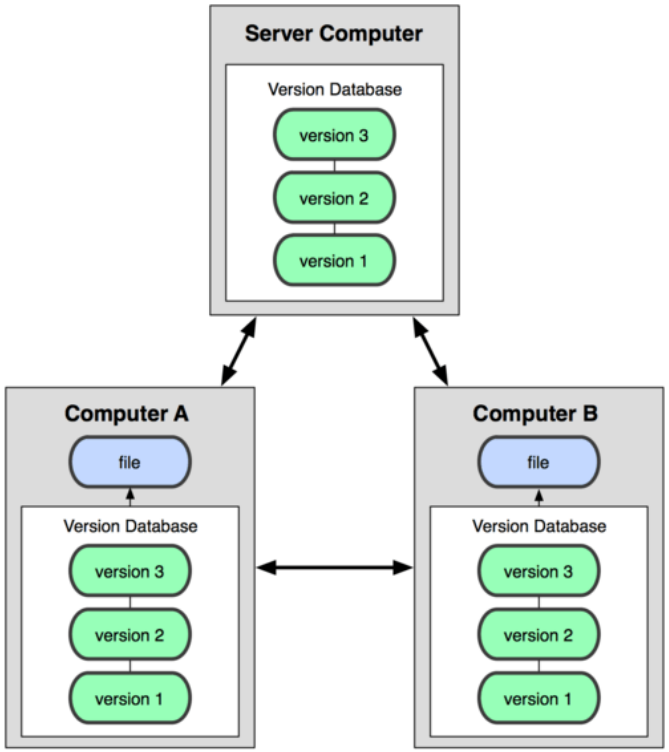
Por qué Git

- La "staging area" (área de ensayo)
  - También denominada índice.
  - Es la zona donde se añaden los cambios que se van a hacer commit.
    - NO es necesario añadir todos los archivos de la WC a la staging area
    - Es incluso posible añadir a la staging area modificaciones concretas dentro de 1 archivo (hunks).
      - Si hay 2 cambios en el archivo se puede hacer commit de 1 de los cambios en un primer paso y otro segundo commit en un segundo
  - Promociona una buena práctica de Git: **haz commit frecuentemente**
    - Que sean pequeños (si es posible)
    - Incluye sólo las modificaciones concretas que resuelvan el problema/tarea.



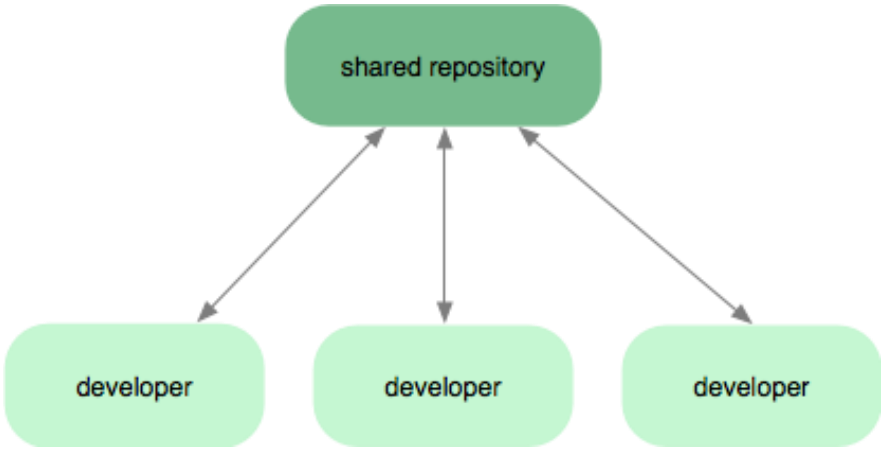
Por qué Git

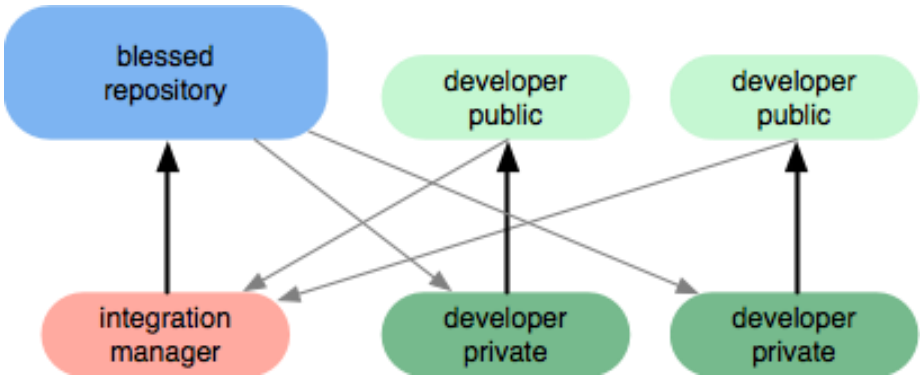
- Es distribuido
  - Todos los desarrolladores tienen una copia completa del repositorio
    - Pueden ser usadas como backups de emergencia
  - No es (demasiado) lento comparado con SVN
    - Teniendo en cuenta que con SVN sólo trabajamos con una rama a la vez.

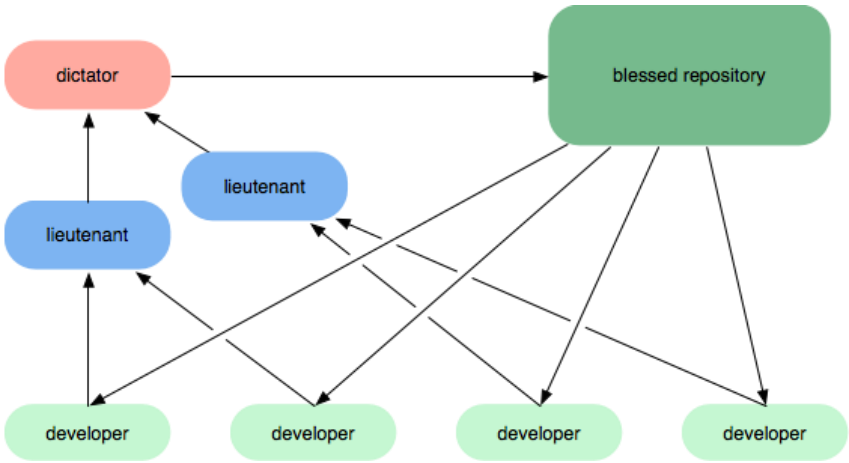


Por qué Git

- Permite múltiples flujos de trabajo





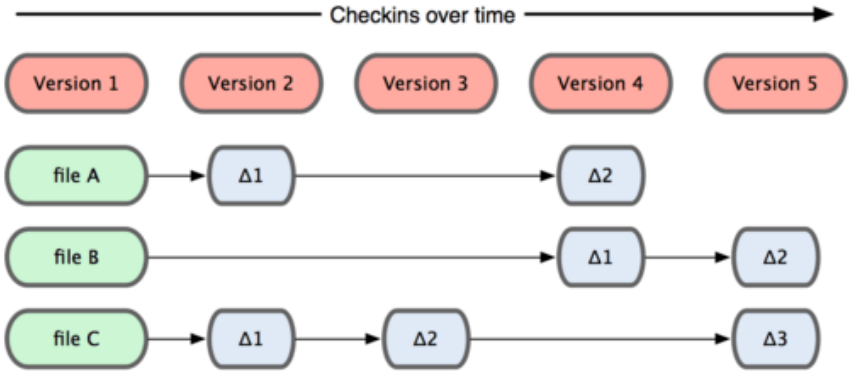


Por qué Git

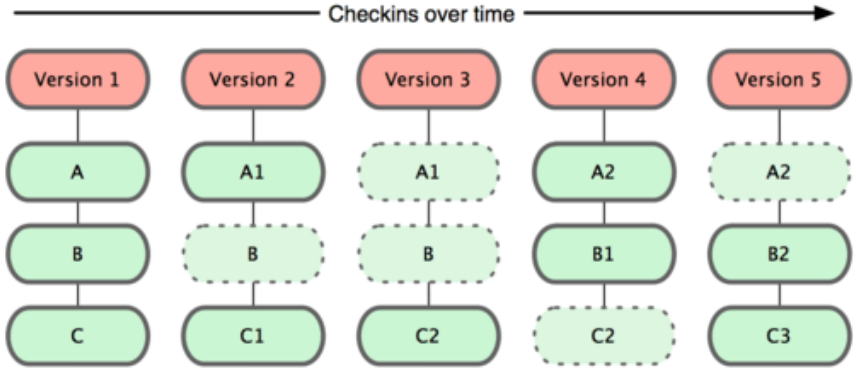
- GitHub y similares
  - Revolución en los proyectos de código libre
    - Mucho más simple colaborar y experimentar
    - Modelo Fork-PullRequest
  - Git gestionado
  - Además permite alojar la web del proyecto, crear una Wiki, discutir sobre el código o las contribuciones…
- Git es el nuevo estándar
  - En una gran cantidad de proyectos Open Source: Android, Apache (algunos), Debian, Drupal, ….
  - Cada vez hay más empresas que están migrando su código a Git
    - Hay productos "Enterprise" como JIRA y otros de Atlassian que soportan activamente Git.

Conceptos básicos

- Git gestiona el repositorio como instantáneas de su estado
  - SVN gestiona el repositorio llevando la cuenta de los cambios incrementales que ha habido.
  - Este hecho simplifica la gestión de branches



Modelo SVN



Modelo Git

Conceptos básicos

- La mayoría de operaciones son locales
  - En la máquina del desarrollador
  - Incluso para revisar la historia del repositorio
- Git tiene integridad
  - Todo en Git (archivos, carpetas, commits, etc.) tiene una firma asociada
    - SHA1: Bastante seguro respecto a colisiones
    - E.g.: 24b9da6552252987aa493b52f8696cd6d3b00373
- Git normalmente sólo añade datos al repositorio
  - Las operaciones de Git añaden datos dentro del repositorio del proyecto
  - Es posible deshacer fácilmente casi cualquier cambio realizado.

Instalación de GIT

- Básico
  - Linux: sudo apt-get/yum install git (git-cola, git-meld)
  - Windows ó instalar Cygwin+git
  - Mac
  - Les falta: una herramienta para poder hacer resolver conflictos o ver diferencias entre archivos
    - Perforce Visual Merge Tool (gratuita), Kdiff3
- Entornos gráficos
  - SourceTree (Windows / MacOSX) (Gratuita)
  - Smartgit (Multiplataforma) (free non-commercial)
  - MacOSX: GitX
  - Windows: TortoiseGIT
- Integrado en IDEs
  - Eclipse (Kepler ya tiene integrado GIT)
  - Xcode >= 4 ya tiene integrado GIT
  - Visual Studio 2012 (necesita plugin) 2013 ya lo tiene integrado

Configurando Git

- Consulta
  - git config --list
- Modificación
  - Configuración a nivel de proyecto
    - git config <param> <valor>.
    - Edita el archivo <proyecto>/git/config
  - Configuración a nivel global (usuario)
    - git config --global ….
    - Crea/Edita el archivo ~/.gitconfig
  - Configuración a nivel del sistema
    - git config --system ….
- Parámetros necesarios (globalmente)
  - user.name, user.email
- Parámetros interesantes
  - core.editor → Controla el editor utilizado en los mensajes de commit
  - merge.tool, mergetool.XXXX.path → Controla la herramienta externa utilizada para resolver conflictos.

Instalación extra

- Configurar Notepad++ para crear los mensajes de commit en MsysGit
  - Instalar Notepad++ (si todavía no lo tienes)
  - Modificar el PATH para que incluya la ruta al directorio de instalación del Notepad++ (opcional)
  - git config --global editor = 'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -nosession -noPlugin
- Cambiar la mergetool (gestor de conflictos)
  - Instalar Perforce Visual Merge Tool
  - Asegurarse que el instalador ha añadido al PATH la ruta al directorio de instalación
    - C:\Program Files\Perforce\
  - git config --global mergetool.p4merge.path 'p4merge.exe'
  - git config --global merge.tool p4merge

Ayuda

- La propia documentación de git
  - Son las man page de Linux pero incluyen muchos ejemplos
  - git help <comando>
  - git <comando> --help
  - E.g.: git help config
    - Es útil echarle un vistazo para ver que opciones hay de configuración.
- StackOverflow
  - La mayor parte de dudas que tengas sobre git ya están resueltas.
- Referencias al final de las trasparencias

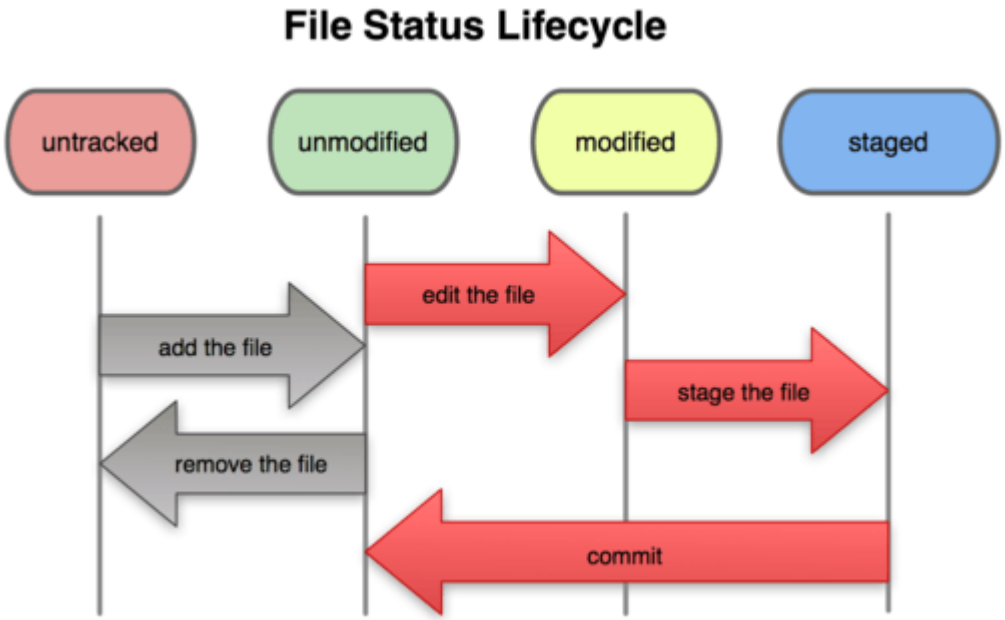
Creando un repositorio Git

- Creación de un repositorio a partir de código ya existente
  - cd <ruta proyecto>; git init
- Creación de proyecto en blanco
  - git init <ruta proyecto>
- Creación de un repositorio a compartir
  - git init --bare <ruta proyecto>
  - Se puede crear en la máquina de desarrollo y mover más adelante a un servidor compartido.

- Crea la carpeta .git en la raiz del proyecto
  - NO en el caso de --base
- Dentro se alojan todos archivos y carpetas internos que gestionan un repositorio de git

Estado de los archivos

- Commited : Gestionado por GIT
- Modificado: Gestionado por GIT pero modificado en la WC
- Staged: Marcado como modificado para incluirlo en el siguiente commit.
- Untracked: fuera de la gestión de Git



Gestión del staging area

- Verificar el estado de la staging area
  - git status → Cambios pendientes de commit
  - git diff → Muestra los cambios de archivos modificados pero NO añadidos al staging area
  - git diff --cached → Muestra los cambios de archivos modificados que SI están añadidos al staging area
- Añadir archivos a la staging area
  - git add <ruta archivo>
  - git add . # Añade todos los archivos nuevos o modificados
  - NOTA: si modificas el archivo añadido tendrás que volver a añadirlo ☺
  - git add -A # Añade todos los archivos modificados, nuevos o borrados
  - La opción –n muestra los cambios a realizar en la staging area pero no los realiza

Ignorando Archivos

- Ignorando archivos
  - Crear el archivo .gitignore en la carpeta del proyecto
  - Es posible tener más archivos .gitignore en otras subcarpetas.
- Ejemplos de archivos .gitignore
  - https://github.com/github/gitignore
- Git NO gestiona almacena carpetas vacías
  - Opción 1: Crear un archivo .gitignore ignorando todos los archivos '*'
  - Opción 2: Mantener carpetas vacías
- Edición avanzada del .gitignore
  - Ignorando ficheros en git: formas
  - Ignorando ficheros en git: prioridades
  - Ignorando ficheros en git: patrones
  - Ignorando ficheros en git: más patrones

Gestionando del staging area

- Eliminando archivos
  - Opción 1 (recomendada): git rm <archivo>
  - Opción 2: rm <archivo>; git rm [-f] <archivo>
  - Elimina el archivo tanto de la WC y anota en la staging area la eliminación.
- Crear una instantánea del repositorio
  - git commit [-m "Mensaje"]
  - Crea una instantánea en el repositorio teniendo en cuenta
    - El estado de la última instantánea realizada
    - El contenido de la staging area

Gestionando del staging area

- Renombrando
  - git mv <origen> <destino>
  - Es un resumen de: mv <origen> <destino>;git rm <origen>; git add <destino>

- Git se da cuenta de que estamos renombrando el archivo debido a la firma del archivo.

Gestionando del staging area

- Visualizar la historia de los commits
  - git log [-p] [-2]
    - -p: Visualiza los cambios realizados (diff) en los commit
    - -2, ó -N: límite del número de commits a visualizar.
  - Cuando se complica la estructura del repositorio mejor utilizar gitk o la interfaz gráfica
    - git log --pretty=format:"%h %s" --graph: proporciona representación textual si no se tiene a mano una interfaz gráfica

- Buscar el culpable
  - git blame <file>
    - Muestra el autor que ha modificado por última vez cada línea de un archivo.

Ups!, me he equivocado

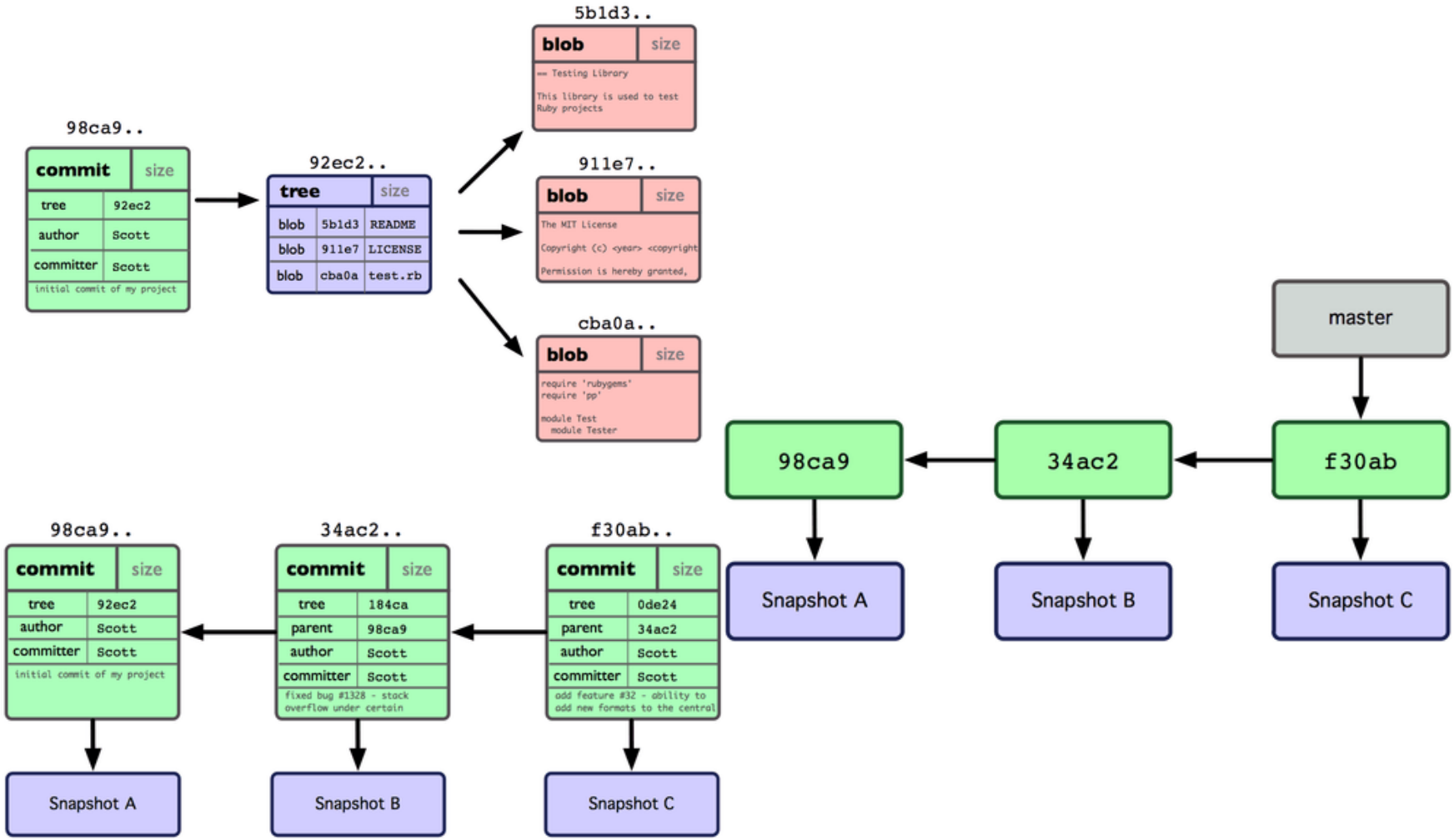
- La he liado en el mensaje del último commit
  - git commit --amend
- Eliminar un archivo del staging area sin perder las modificaciones
  - Si el archivo es nuevo
    - git rm --cached <archivo>
  - Si el archivo está modificado
    - git reset HEAD <archivo>
  - Útil por si no queremos hacer commit de este archivo.
- Deshacer los cambios en la copia de trabajo y volver al archivo original desde la última instantánea
  - git checkout -- <archivo>
- Deshacer el último commit
  - Como si no hubiera existido
  - git reset HEAD~1 # deshace el último commit del branch actual

Ups!, me he equivocado

- Deshacer el último commit (como si no hubiera existido)
  - Como si no hubiera existido
  - git reset HEAD~1 # deshace el último commit del branch actual
- Deshacer un commit (dejando constancia que se ha eliminado)
  - git revert <sha1 commit>
- ^ y ~ (especificando revisiones)
  - HEAD~1 → commit anterior al al último commit de la rama
  - HEAD^ → equivale a HEAD^1 y es el primer padre del último commit

Iván Martínez Ortiz

Estructura interna de un repositorio Git



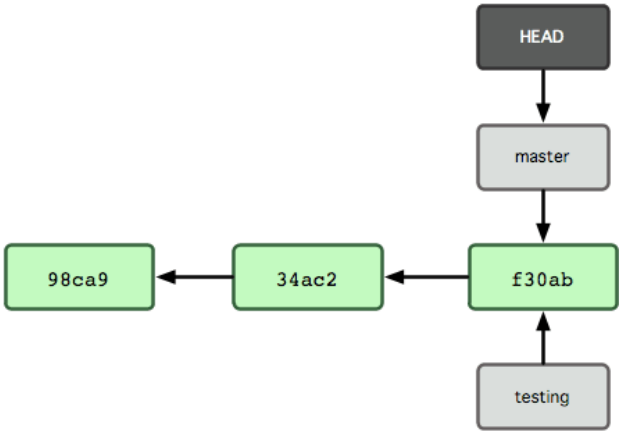
Branches en git

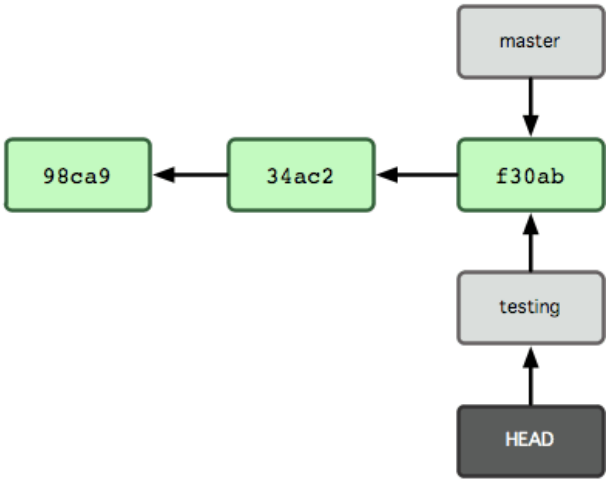
- Por defecto existe el branch **master**
  - Similar al trunk de SVN desde el punto de vista estructural.
  - NO ES IGUAL SEMÁNTICAMENTE AL trunk
    - El branch master se considera que contiene el código que se puede poner en producción.
  - El branch master es una referencia
- Listar branches / Averiguar branch actual
  - git branch [-v] –a
  - La referencia HEAD apunta al branch actual

Iván Martínez Ortiz

Branches en git

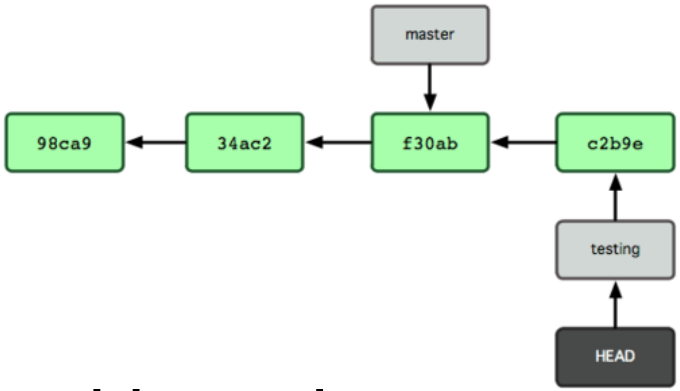
- Crear un branch (local)
  - git branch <nombre branch>
  - Crea un branch a partir del branch actual
- Pasar a trabajar a otro branch
  - git checkout <nombre branch>
- Los dos a la vez:
  - git checkout –b <nombre branch>



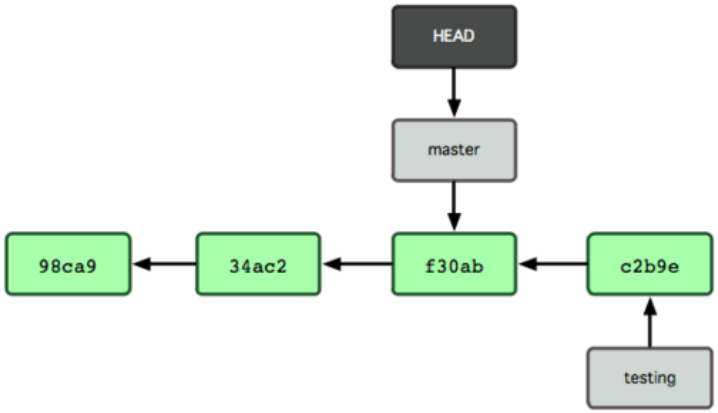


Branches en Git

- Al hacer commit se realizarán sobre el branch activo

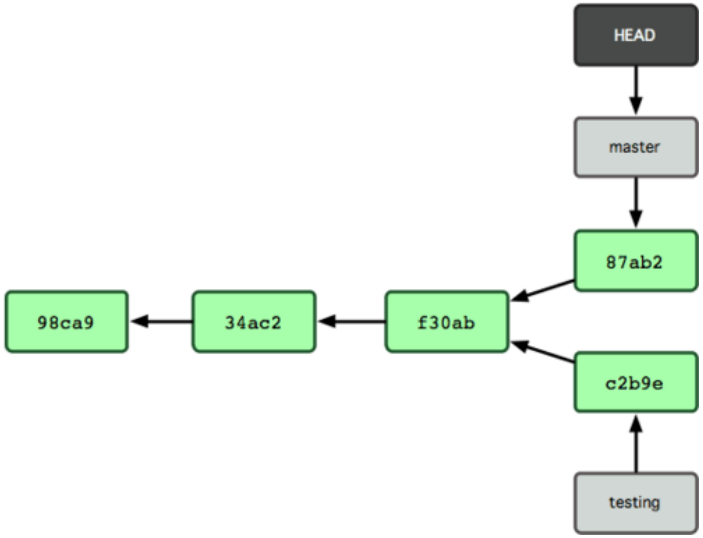


- Podemos volver al branch master cuando queramos



Branches en Git

- Al modificar el branch master la estructura del repositorio queda
  - La historia de los branches diverge
  - Es necesario hacer un merge (reconciliar) los cambios



Branches en Git

- Flujo de trabajo con branches
  - Crear un branch cuando tengo que hacer una tarea o quiero experimentar algo.
  - Trabajar sobre el branch (desarrollar, hacer pruebas)
  - Nos aseguramos que la copia de trabajo está limpia
    - No hay ningún cambio pendiente
  - Actualizamos nuestro branch de trabajo con los cambios que haya habido en master
  - Cuando estamos contentos con el trabajo hacemos un merge del trabajo en el branch master

Branches en Git

- ¿Cómo hacemos el merge?
  - Checkout del branch donde vamos a integrar los cambios
    - git checkout master
  - Integramos los cambios
    - git merge tarea
- Cuando se realizan los merges es posible que haya que resolver conflictos
  - Conflictos: modificaciones sobre un mismo archivo que git no sabe resolver.

Gestion de branches en Git

- ¿Cómo averiguo los branches que hay?
  - git branch [-a -v]
  - El branch activo aparece con un '*'
- ¿Cómo averiguo que branches NO están integrados con el branch activo?
  - git branch --no-merged
- ¿Cómo averiguo que branches SIestán integrados con el branch activo?
  - git branch --merged
- Una vez que un branch está integrado puedo eliminarlo (si quiero)
  - git branch -d <branch>

Git-flow

- Gestión del proyecto con branches de manera avanzada
  - Quizás demasiado
- Información detallada sobre git-flow
  - ¿Qué es git-flow?
  - Instalación de git-flow
  - La rama develop y el uso de features branches
  - Release branches
  - Hotfixes branches
  - Resumen y conclusiones

Iván Martínez Ortiz

Creando tags

- Tipos de tags en Git
  - Anotados → genera un objeto en el repositorio
  - Ligeros → Similar a los branches
  - En ambos casos los tags se crean en el repositorio local
- Crear un tag ligero a partir del último commit
  - git tag <nombre tag>
- Crear tags anotados a partir del último commit
  - git tag -a <nombre tag> [-m <mensaje>]
- Crear tag anotado y firmado (con GPG)
  - git tag -s <nombre tag> [-m <mensaje>]

Creando tags

- Crear tag de un commit pasado
  - Utilizar git log para averiguar el SHA1 del commit
  - git tag -a <nombre tag> [-m <mensaje>] <SHA1>
- ¿Cómo compartir un tag?
  - git push <remote> <nombre tag>
- ¿Cómo compartir todos los tags?
  - git push <remote> --tags

GitHub: Forks y Pull request

- Servicio de Git gestionado
  - Gratuito y de pago
- Conceptos importantes con Git en GitHub
  - Forks de repositorios en Github
  - Manteniendo forks al día
  - ¿Qué es un pull request?

Iván Martínez Ortiz

Servidor de Git propio

- Más simple si se parte de un servidor Linux
- Opciones de instalación
  - Usuario git + SSH
  - gitserver
  - Apache+gitserver
- Instalar algún otro software ….
  - Normalmente es necesario y complica el mantenimiento
    - Control grano fino sobre permisos de acceso a los repositorios.
    - Comentar los cambios en el código
  - Opciones más avanzas: instalar gitlab o gitorius.

Ejemplo de Servidor interno de Git

- Ubuntu Server 13.04 64bit
  - Instalación mínima como VM
  - Grupo de paquetes: OpenSSH Server
  - Paquetes necesarios: git
  - Paquetes extra: gitweb (sólo si queremos poder visualizar los repositorios a través de la web)
- Proceso de instalación
  - Crear usuario git

```
useradd -r -s /usr/bin/git-shell -d /var/lib/git
mkdir /var/lib/git
chown git:git -R /var/lib/git
chmod 755 /var/lib/git
```


Ejemplo de Servidor interno de Git

- Configuración de acceso SSH por clave pública para el usuario git

```
cd /var/lib/git
mkdir .ssh
touch .ssh/authorized_keys
chown git:git -R .ssh
chmod 700 .ssh
chmod 600 .ssh/authorized_keys
```

- Gestión de acceso de los desarrolladores
  - Debe hacerse utilizando una cuenta de administrador
    - El usuario git no puede abrir un terminal
  - Los desarrolladores deben generar una clave SSH

Servidor interno de Git

- Gestión de repositorios
  - Crear repositorios
  - Hacer backups de los repositorios
  - Debe hacerlo el usuario administrador
- Creación de un repositorio

```
cd /var/lib/git
git init --bare <repositorio>
chown git:git –R <repositorio>
```

```
cd /var/lib/git
git init --bare miRepo.git
chown git:git –R miRepo.git
```

  - El repositorio es accesible con la URL:
    - ssh://git@desarrollo.miempresa.es/var/lib/git/miRepo.git
    - ó git@desarrollo.miempresa.es:/var/lib/git/miRepo.git
- Backup de un repositorio
  - Crear un tar.gz de la carpeta del repositorio

Iván Martínez Ortiz

Ejemplo de Servidor interno de Git

- Dar acceso a un desarrollador
  - Copiar clave pública a /var/lib/git
    - E.g.: /var/lib/git/imartinez.pub
  - Convertir la clave si es necesario, deben tener la siguiente pinta

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAIEAlh86vznFOQmv+yPi2IR2E...
ssh-dsa RtxfiJt/YR7mpghbfSjHScPqBLntR9SgYDUSgvMTYJH882NBb...
```

    - En otro caso convertir

```
ssh-keygen –i –f /var/lib/imartinez.pub > /var/lib/imartinez.pub.ok
```

  - Añadir clave al archvivo /var/lib/.ssh/authorized_keys

```
cat /var/lib/imartinez.pub.ok >> /var/lib/git/.ssh/authorized_keys
```


Colaboración con git: remotes

- La colaboración entre desarrolladores se realiza a través de repositorios remotos
- Desde tu repositorio es posible acceder a otros repositorio para traerte cambios
- No es obligatorio un servidor git
  - Se podría utilizar simplemente un directorio en un disco compartido
  - Es recomendable utilizar el servidor SSH para evitar problemas
- Modelo de repositorios públicos por desarrollador y un blessed repositorio
  - Crear un repositorio "maestro"
  - Crear un repositorio por desarrollador.

Trabajando con remotes

- ¿Cómo me traigo mi repositorio publico a mi máquina?
  - git clone <url>
  - Automáticamente crea un remote llamado "origin"
- Puedo visualizar los remotes que hay en mi repositorio
  - git remote # Muestra el nombre del remote
  - git remote –v #Muestra la URL que se utilizo para crear el remote
- Puedo añadir más remote
  - De hecho tenemos que añadir el repositorio maestro
  - git remote add <nombre> <URL>
  - git remote add upstream ssh://git@git.miempresa.com/var/lib/git/maestro.git

Trabajando con remotes

- Los repositorios remotos también tienen alojados los branches
- Son referencias a branches en un repositorio remote
  - Tienen esta pinta: <remote>/<branch>
  - E.g.: origin/master, origin/development
- Cuando se clona un repositorio remoto se crea una branch local asociado al branch del master
  - E.g.: master → origin/master
  - Estos branches se denominan **tracking** branches
- ¿Cómo traerme un branch remoto?
  - git checkout --track <remote>/<branch>
  - git checkout –b <branch> <remote>/<branch>

Trabajando con remotes

- ¿Cómo me traigo cambios de algún repositorio remoto?
  - git fetch <nombre remote>
- ¿Cómo listo los branches que hay en un remote?
  - git ls-remote <remote>
- ¿Cómo aplico los cambios que ha en un remote?
  - git merge <remote>/<nombre branch>
- ¿Cómo **me traigo** los cambios y los aplico?
  - git pull [<remote>] [<nombre branch>]

Trabajando con remotes

- ¿Cómo **envio** cambios a un remote?
  - git push [<remote>] [<nombre branch>]

- ¿**Cómo "romper" un remote**?
  - O al menos molestar a la gente
  - git push --force [<remote>] [<branch>]
  - Use the Force, Luca

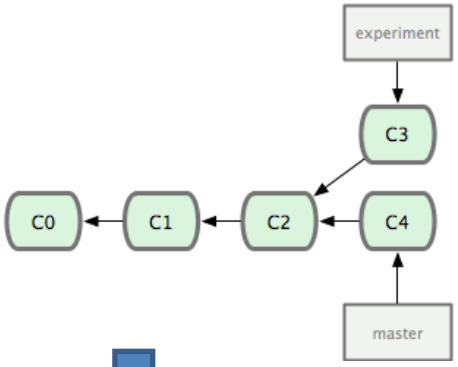
- ¿Cómo borro un branch remoto?
  - git push origin :<branch>

Trabajando con remotes

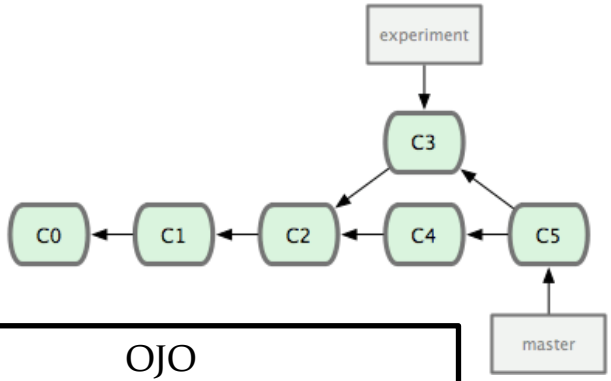
- ¿Puedo tener más de 1 remote?
  - Sí
- ¿Puedo colaborar con otro compañero sin pasar por el repositorio "maestro"?
  - Sí
  - Añades un remote que apunte al repositorio público de tu compañero y te traes el branch que quieras probar.

Git Rebase

- Es otra manera de integrar cambios de un branch en otro
- **ADVERTENCIA: Reescribe la historia del repositorio**
  - Si no se tiene cuidado se puede liar

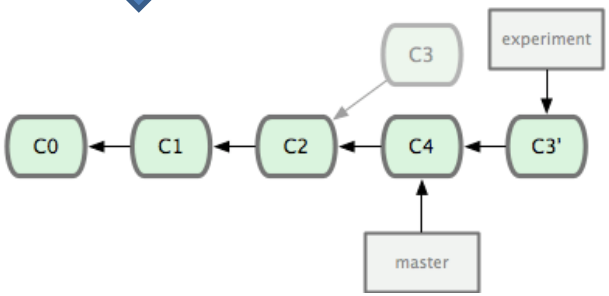



```
git checkout master
git merge experiments
```

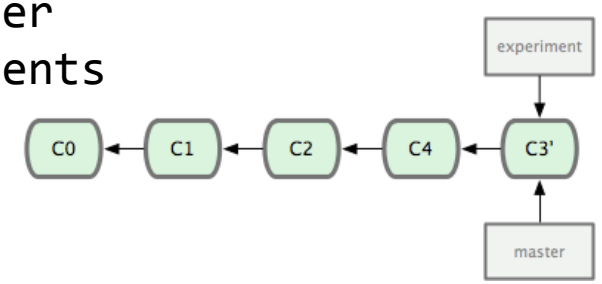



```
git checkout experiments
git rebase master
```

OJO
git rebase transplanta los cambios de una rama sobre la rama activa



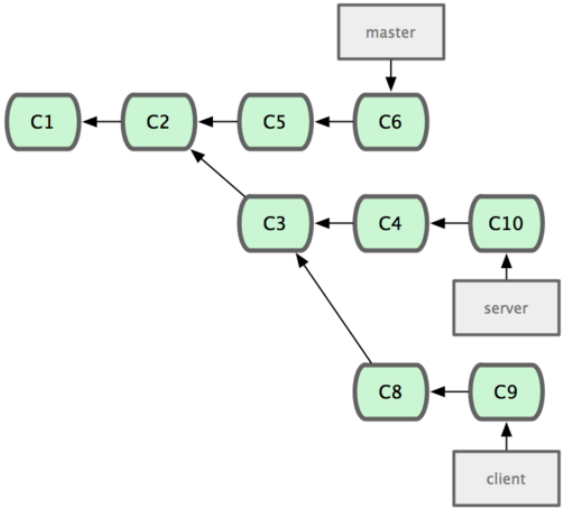

```
git checkout master
git merge experiments
```



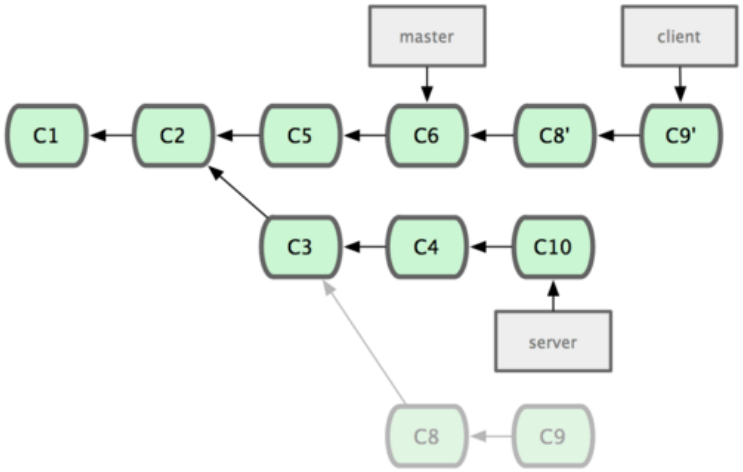
Git Rebase

- pull --rebase
  - Permite traer los cambios de un remote y aplicarlos pero utilizando un rebase en vez de un merge
  - Útil para no complicar la historia del repositorio y para abordar poco a poco la reconciliación con el branch del que hemos partido.

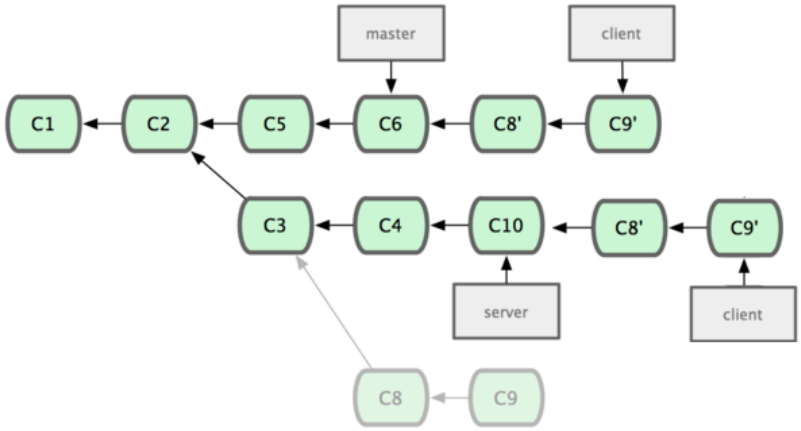
Git Rebase: Caso más avanzado




```
git rebase --onto master server client
```

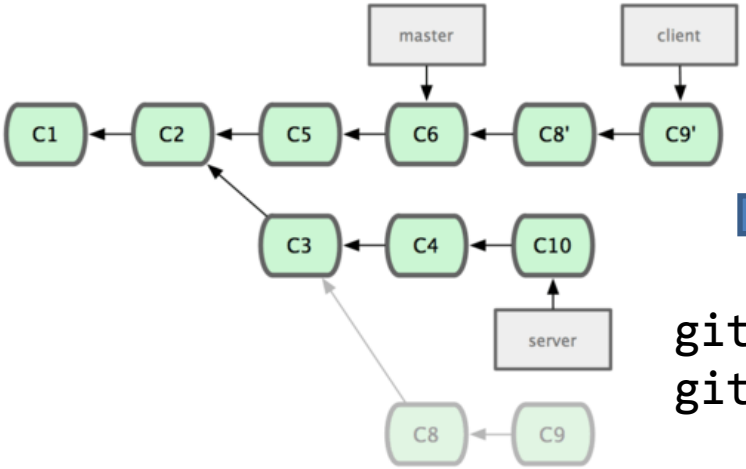



```
git checkout client
git rebase master
```

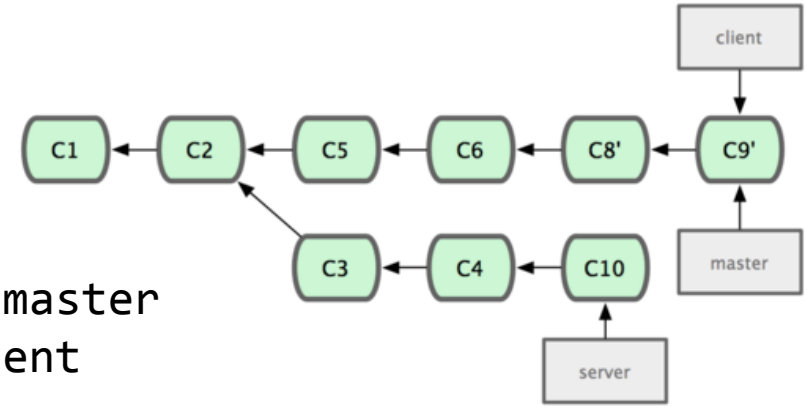


Iván Martínez Ortiz

Git Rebase: Caso más avanzado

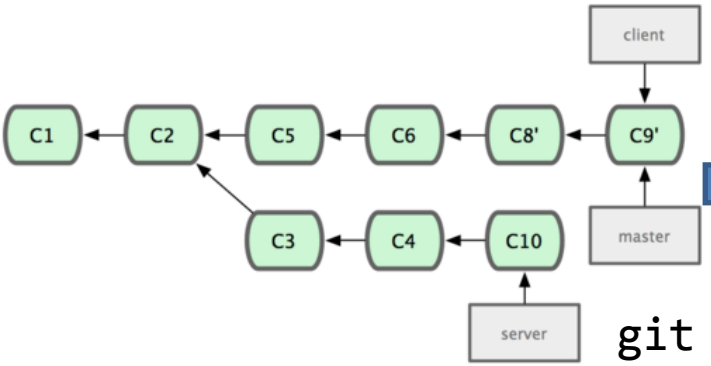


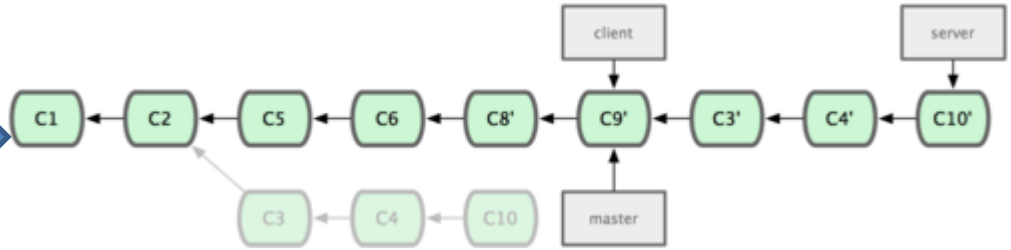

```
git checkout master
git merge client
```



Iván Martínez Ortiz

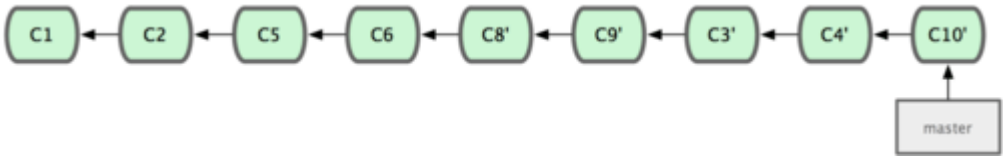
Git Rebase: Caso más avanzado






```
git checkout server
git rebase master
```

```
git checkout master
git rebase server
git -d client
git -d server
```



Referencias

- http://git-scm.com/
- Información básica
  - Pro Git (Free book)
  - Manual de git
    - Git tutorial
    - Everyday Git with 20 commands or so
    - Git User Manual
    - Git core tutorial
  - Git Pocket Guide (Acceso con IP UCM)
  - Version Control with Git, 2nd Edition (Acceso con IP UCM)
- Páginas para aprender GIT
  - http://speckyboy.com/2013/06/03/resources-for-learning-git/
  - http://www.gitguys.com/
  - http://teach.github.com/
  - http://gitimmersion.com/
  - http://sixrevisions.com/resources/git-tutorials-beginners/
  - http://www.webdesignerdepot.com/2009/03/intro-to-git-for-web-designers/

Referencias

- Tips para GIT
  - git-for-beginners-the-definitive-practical-guide
  - http://gitready.com/
  - Opciones del comando git add
  - Forzar un merge commit
  - Mantener carpetas vacías en el repositorio
  - Xcode y Git
  - Visual Git Reference (comandos intermedios)
  - 6 Motivos por los que Git no es un sistema de backups
  - Revisar cambios que se han añadido al index (staged)
- Conceptos avanzados de GIT
  - http://softwareswirl.blogspot.de/
  - Referencias, Github y pull requests
  - Git alias: creación de comandos parametrizados
  - Convertir repositorio SVN a GIT