



Introduccion a Git:

Un Sistema de control de versiones

¿Qué es Git?

Un sistema de control de versiones distribuido.

No depende de acceso a la red o un repositorio central.

Enfocado a la velocidad, uso practico y manejo de proyectos grandes.

Creado por Linus Torvalds, el creador del núcleo Linux.

Cualidades de Git

Diseñado para manejar proyectos grandes.

Es extremadamente rápido.

Autenticación criptográfica del historial.

Formato de archivo muy sencillo y compacto.

100% distribuido.

Se puede sincronizar por cualquier medio.

Centralizado vs Distribuido

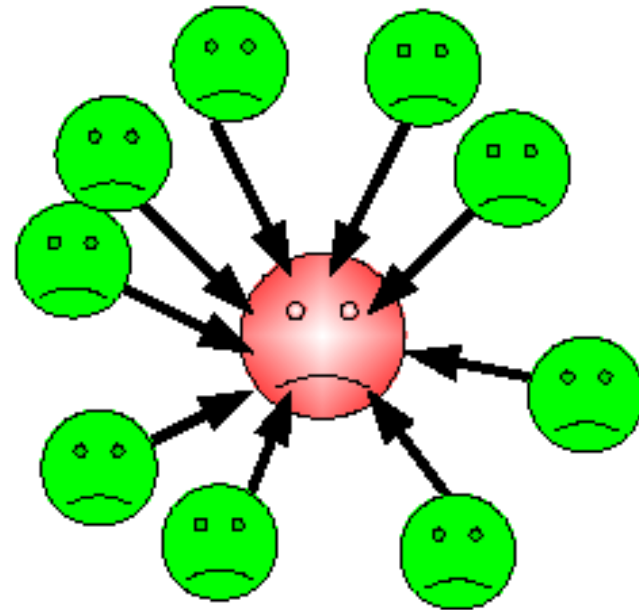
Centralizado

Todos deben conectarse a un servidor central.

No puedes trabajar sin conexión.

Políticas de escritura.

Los cambios afectan a todos.



Centralizado

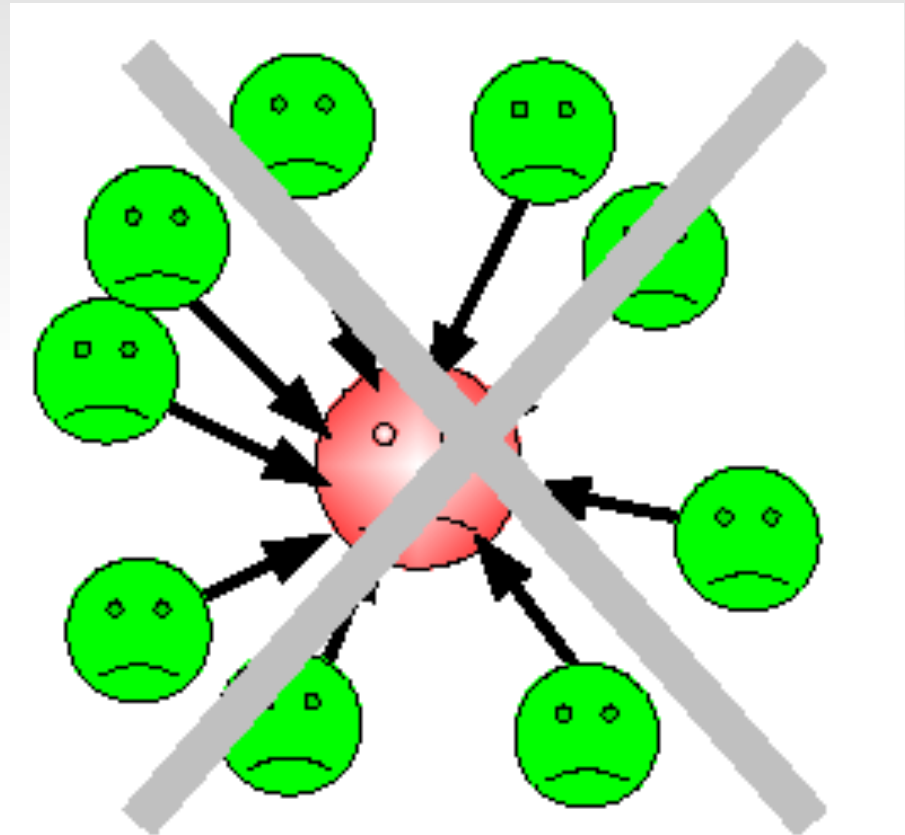
Crear ramas es muy costoso,
y tedioso.

Las ramas afectan a todos
porque son globales.

Necesitarias politicas de
nomenclaturas para no estorbar
a otros.

Casi no envias cambios. Te
esperas a que tengas algo
estable para subirlo al
repositorio.

Nunca haces algo estable a la
primera.



Distribuido

Todos son igual de importantes.

Se puede trabajar sin conexión.

Debes ganar confianza y confiar en otros.

Los cambios solo te afectan a ti.

Distribuido

Colaboración

Realiza cambios sin afectar a los demás.

Experimenta y ramifica el código a tu gusto.

Confía en tu información

sin tener que confiar en todos los demás.

sin tener que pedir permisos de escritura

sin depender de tu servidor o hosting del repositorio.

No hay un punto débil central, y la información se replica de forma natural.

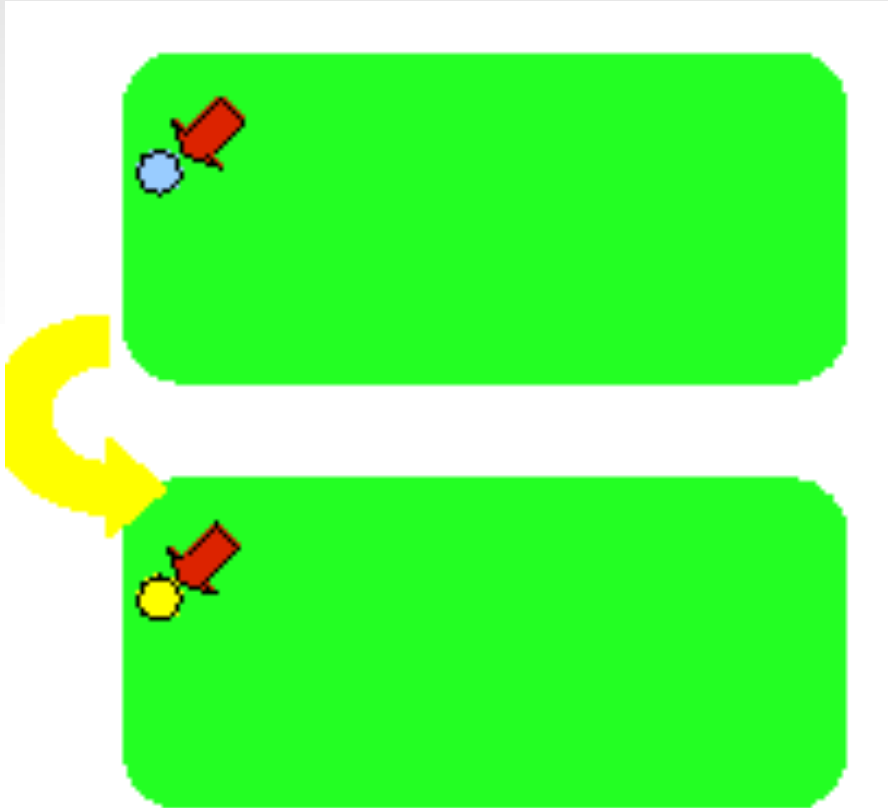
¿Cómo funciona?

Comenzamos con
nuestro proyecto en un
repositorio



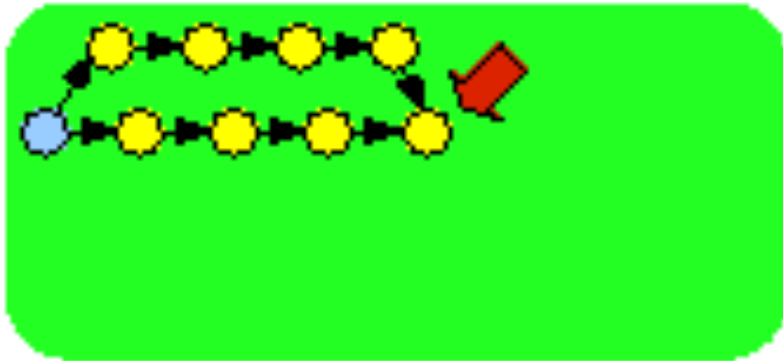
¿Cómo funciona?

Luego alguien hace una copia de nuestro proyecto



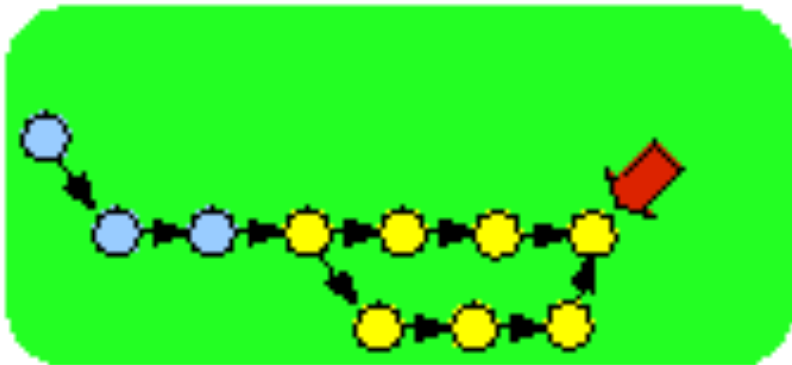
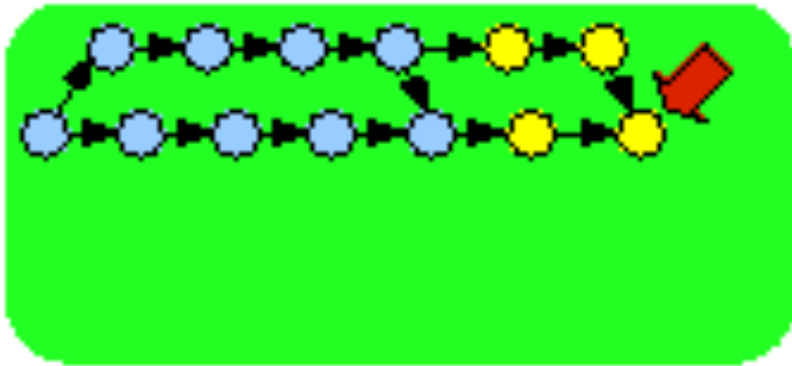
¿Cómo funciona?

Cada desarrollador
puede seguir trabajando
individualmente



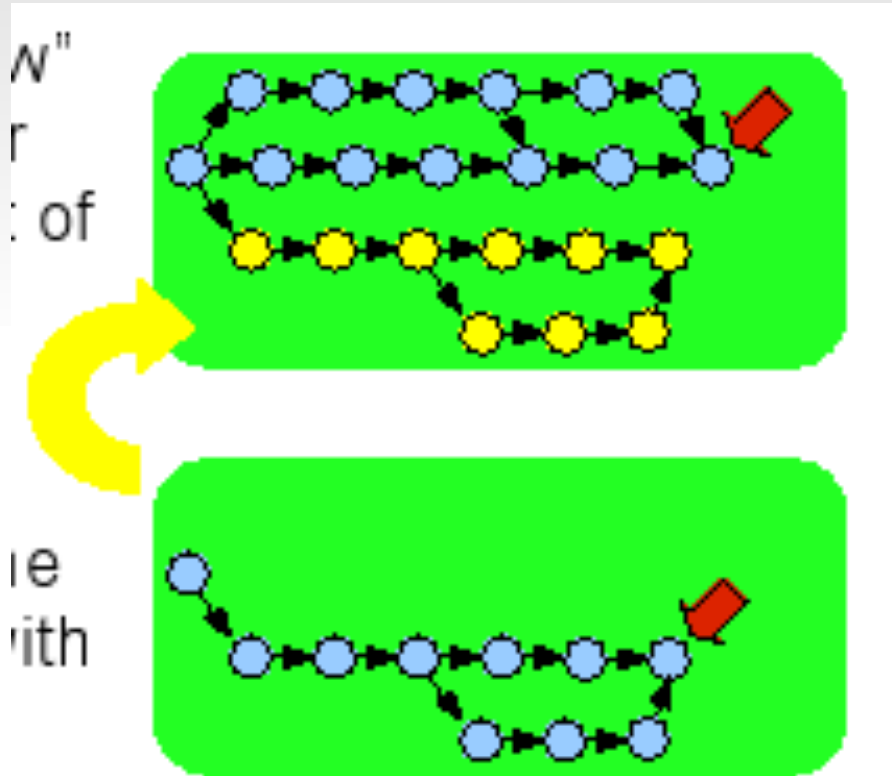
¿Cómo funciona?

Los cambios de cada uno no afectan al otro



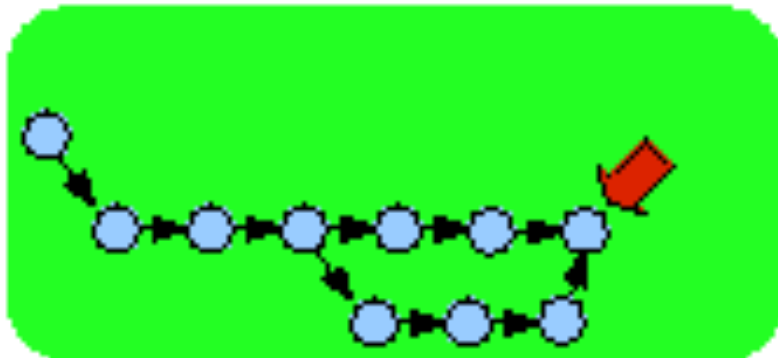
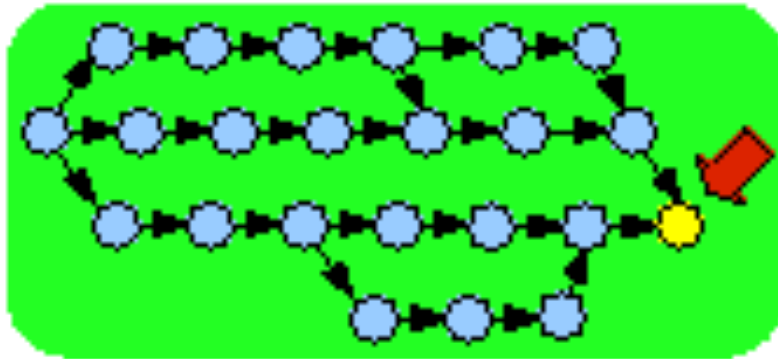
¿Cómo funciona?

Luego, el desarrollador principal incorpora los cambios del otro desarrollador

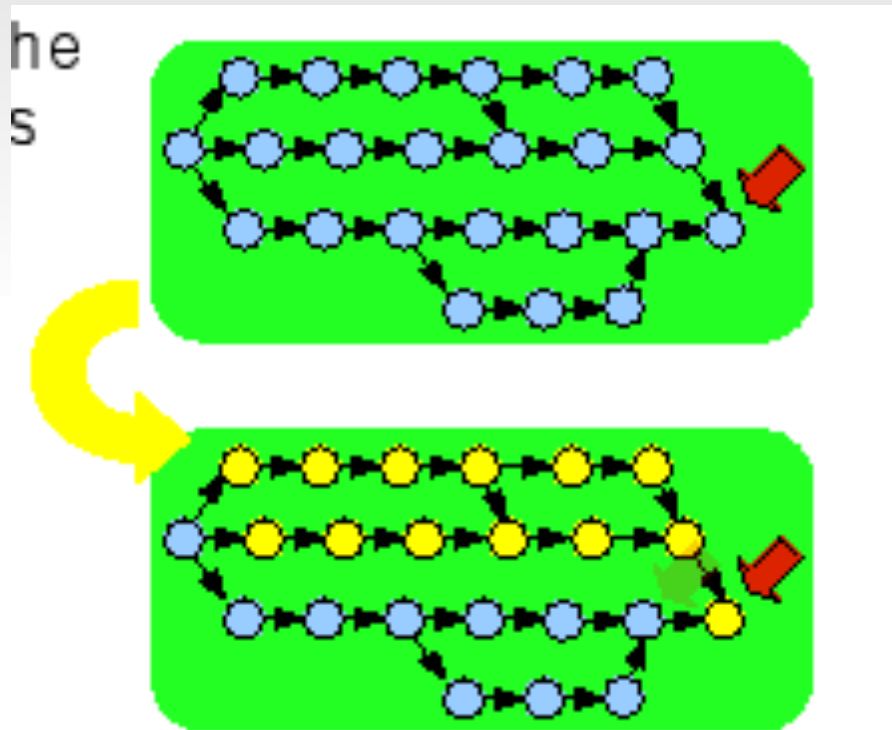


¿Cómo funciona?

Y los mezcla con su trabajo.



¿Cómo funciona?



Finalmente, el segundo desarrollador incorpora los nuevos cambios del primero, con todo integrado.

Confianza

Todos son tontos. Solo unos cuantos saben programar bien.

Solo acepto cambios de mis amigos de confianza.

Tus cambios son mas limpios.

Siempre habrá un repositorio de mayor confianza.

Ramas

Crear ramas del código es fácil y sencillo.

Crear ramas fomenta la experimentación y creatividad.

Las ramas solo te afectan a ti.

Normalmente se tienen 2, 3 o hasta 15 ramas o más.

Merges son muy sencillos de realizar.

Desempeño

La velocidad en el desempeño es muy importante.

No debe estorbar o frenar tu ritmo de trabajo.

Los merges deben ser rápidos y sencillos. Deben tomar menos de un segundo.

Contenido

Git controla contenido, no archivos.

Se puede borrar, renombrar, mover sin tener que avisarle a Git.

Y el historial se mantiene!

Puedes ver el historial de una función desde su inicio, aunque haya cambiado de un archivo a otro.

Git es para grandes

Linux kernel tiene más de 22,000 archivos.

Mostrar las diferencias de todo el kernel, 74,000 commits, toman aprox. 7 segundos.

El repositorio en CVS del Mozilla Project es de 3GB, en Subversion es de 12GB en formato fsfs. En Git es de 300 Mb.

¿Quienes usan Git?

El Linux kernel

Ruby on Rails

Perl

X.org / Cairo

Wine

Beryl

Gnome

Android

VLC

Obtener Git

Debian

```
aptitude install git-core
```

Fedora

```
yum install git
```

Gentoo

```
emerge dev-util/git
```

OpenSUSE

```
yum install git
```

Ubuntu

```
aptitude install git-core
```

Mac OS X

```
port install git-core
```

Usando Git

Crear un repositorio

```
cd project/
```

```
git init    #Inicializa el repositorio.
```

```
git add .   #Agrega todos los archivos al repositorio.
```

Realizar cambios

```
git status  #Consulta lo que ha cambiado
```

```
git commit -a -m "commit message" #Guarda todos tus cambios
```

Crear ramas

```
git branch experimento_uno
```

```
git checkout experimento_uno
```

Hacer merges

```
git checkout master
```

```
git merge experimento_uno
```

Revisar historial

```
git log
```

Usando Git

Regresar a la version anterior

```
git reset HEAD~1
```

Traer cambios de un repositorio externo:

```
#trae y mezcla los cambios de repositorio en un USB.
```

```
git pull /media/usbdisk/project.git
```

```
#agrega un repositorio externo para compartir cambios con un  
amigo
```

```
git remote add friend file:///net/friend/git/project
```

```
# Trae los cambios del repositorio externo sin mezclarlos
```

```
$ git fetch friend
```


Usando Git

Aplica parches recibidos en archivos .patch

```
git apply < ../p/foo.patch
```

```
git commit -a
```

Crea un repositorio vacío para un webserver.

```
mkdir my-repo.git
```

```
cd my-repo.git
```

```
git --bare init
```

```
chmod a+x hooks/post-update # this is needed for HTTP transport
```

Usando Git

Muestra las diferencias entre dos ramas

```
git diff origin..master
```

Crea un archivo .patch de los cambios realizados

```
git diff origin..master > my.patch
```

Obten un diffstat de los cambios hechos pero no guardados

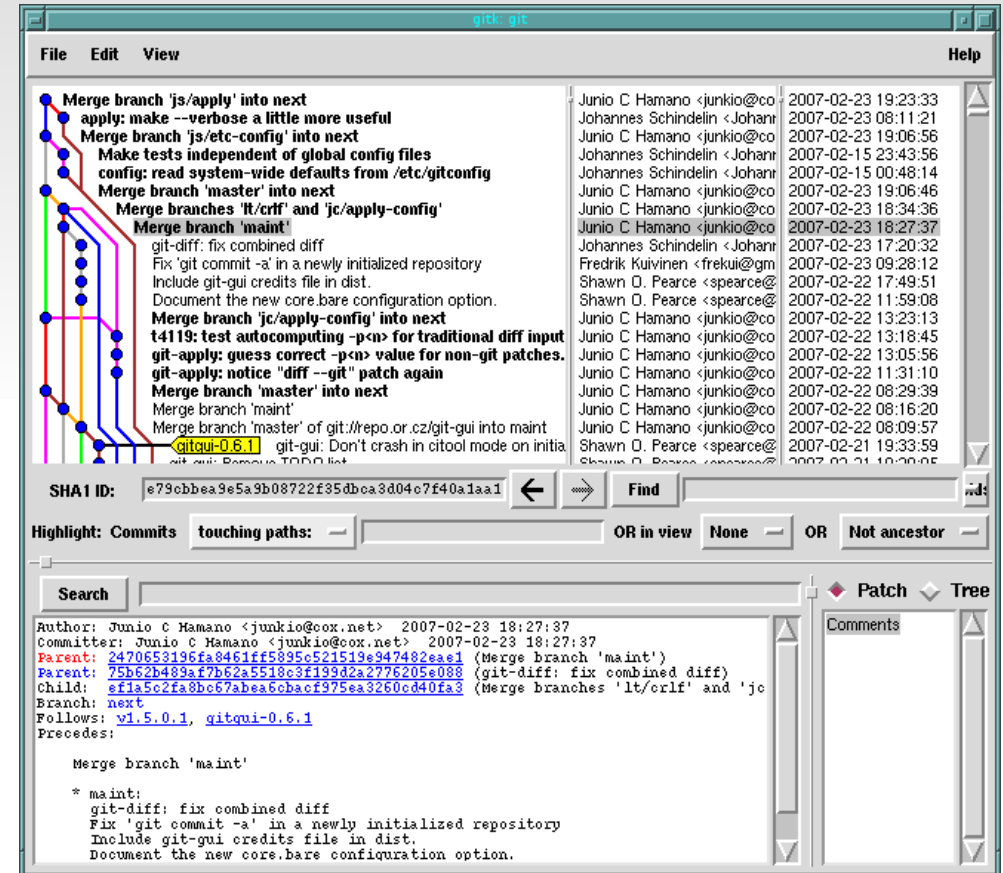
```
git diff --stat HEAD
```

Aplica el 4º ultimo parche de la rama “otra” a la rama actual.

```
git cherry-pick otra~3
```

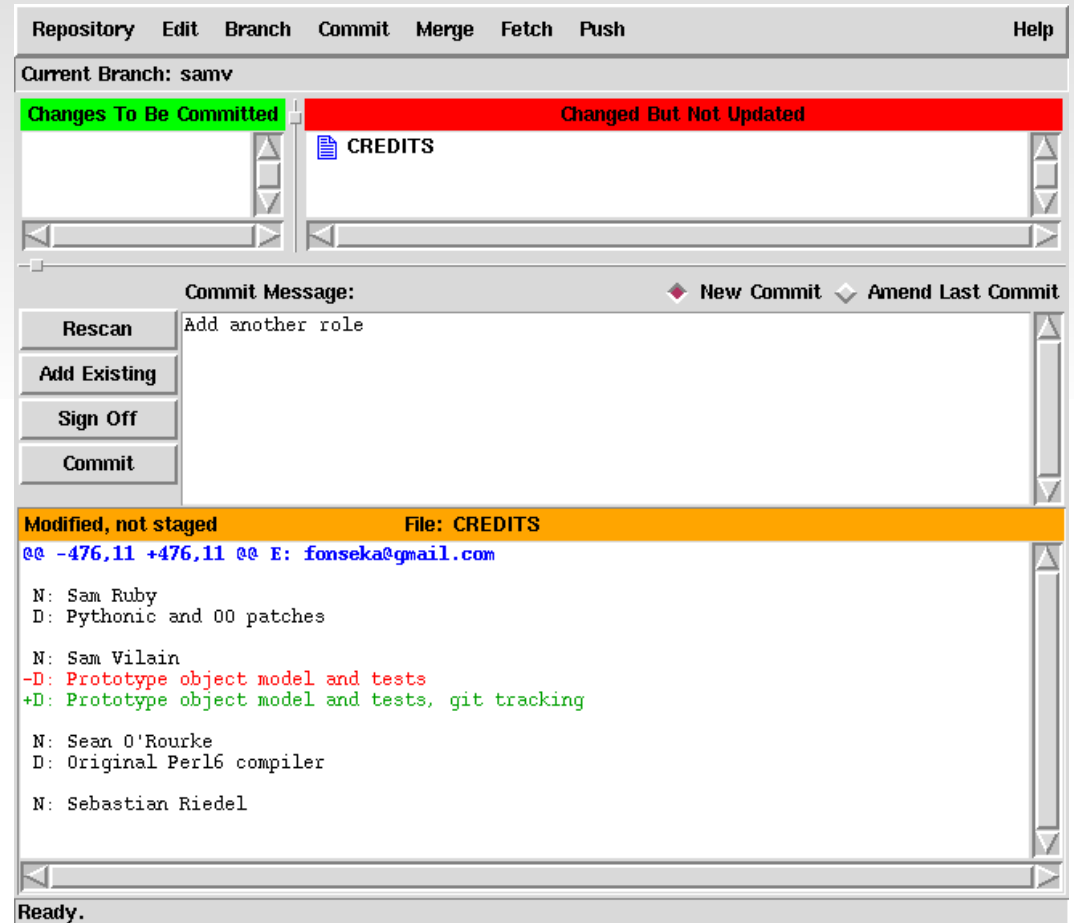
Herramientas

Gitk, una herramienta visual para ver el historial del repositorio.



Herramientas

Gittool, una interfaz visual para ver los cambios realizados y guardar los cambios.



Interfaces Web

gitweb implementación en Perl mantenida por Kay Sievers. Se usa en kernel.org

wit implementación en Python mantenida por Christian Meder.

gitarella implementación en Ruby mantenida por Diego Petten

gitphp implementación en PHP por Zack Bartel

cgkit – implementación en C por Lars Hjemli

Git y Subversion

SVN checkout

```
git svn checkout svn://myrepo/svn/trunk
```

SVN Update

```
git svn rebase
```

SVN Commit

```
git svn dcommit
```