

Git

The fast version control system

Fco Javier Lucena
Consultor Informático

Granada, 23-Nov-2011

Git: The Fast Version Control System

Granada, 23 de Noviembre de 2011

- *git init*
- *Installation*
- *Setup*
- *First Step*
- *Working with remote repositories*
- *Branches and Merging*
- *Rebase*
- *Installing a Git server*
- *Online remote repositories*
- *Git Hosting Provider*
- *Graphical UI's for Git*

¿Qué es un sistema de control de versiones?

¿ Qué es una Versión?

“estado en el que se encuentra un proyecto en un momento determinado”

¿ un SCV “Casero” ?

- + practica1
- + practica1_Antonio
- + practica1_version1
- + practica1_version2__con_Cambios_de_Antonio
- + practica1_version2_la_mia
- + practica1_version3
- + practica1_final
- + practica1_final_error
- + practica1_final_corregida
- + practica1_ENTREGADA



¿Qué es un sistema de control de versiones?

“Software para gestionar el historial de versiones de un proyecto”

Ventajas

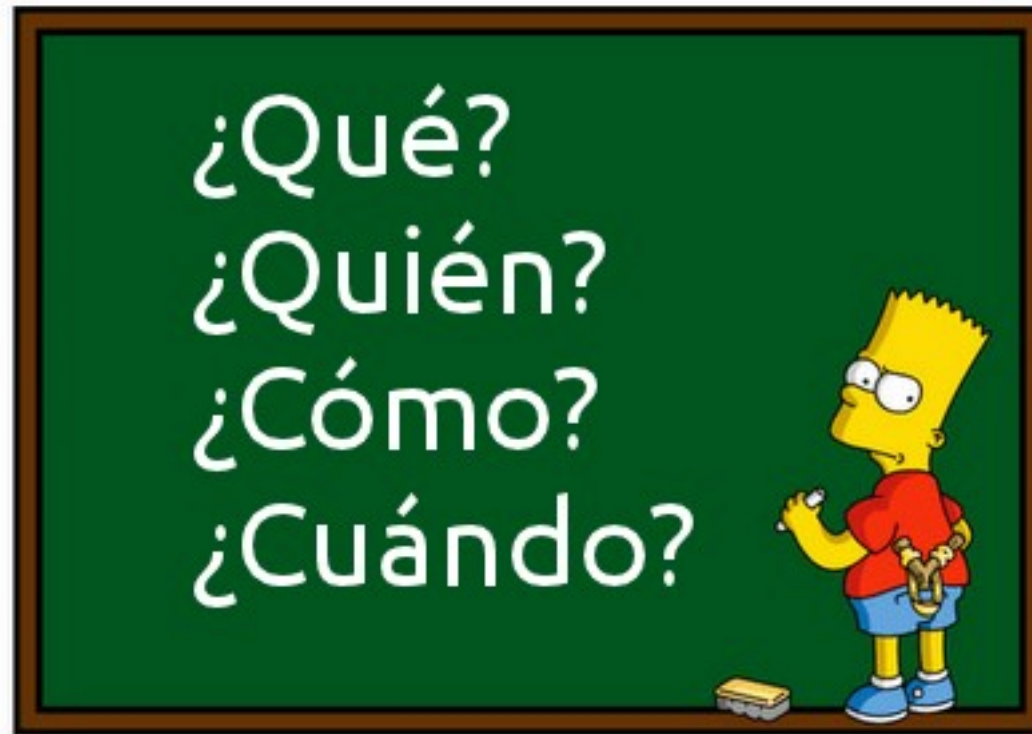
Copias de Seguridad



Deshacer Cambios



Historial de Cambios



Diferentes Versiones del Proyecto



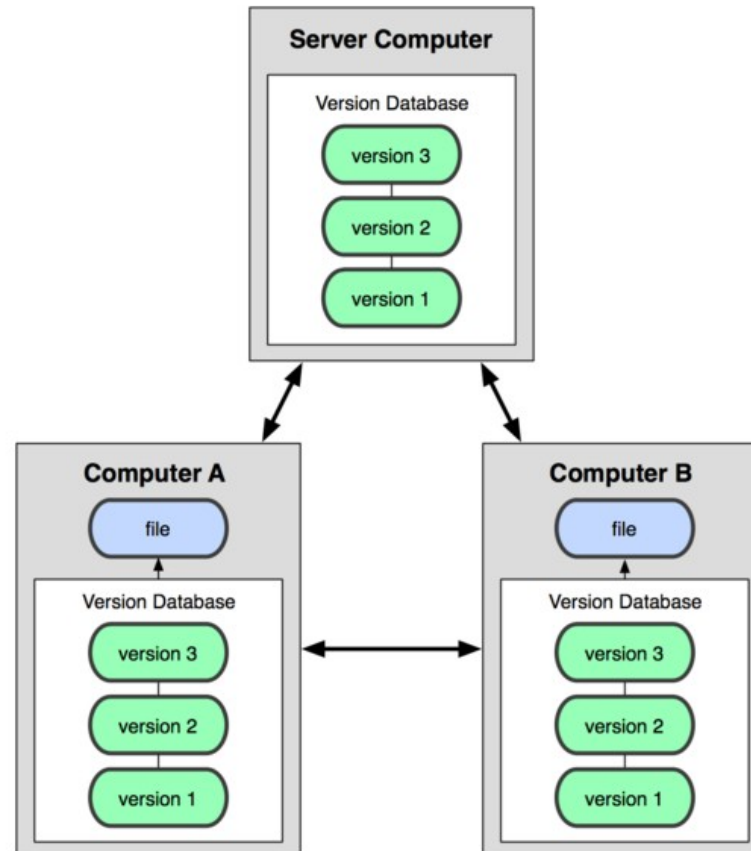


Git nació en 2005 de la mano de Linus Torvalds a raíz de su trabajo en el desarrollo del Kernel Linux

*** Objetivos**

- Velocidad
- Diseño simple
- Un fuerte apoyo para el desarrollo no lineal
(en miles de ramas en paralelo)
- Totalmente distribuido
- Capaz de manejar grandes proyectos como el kernel de Linux de manera eficiente
(la velocidad y tamaño de los datos)

Git se basa en un SCV **DISTRIBUIDO**



Integridad

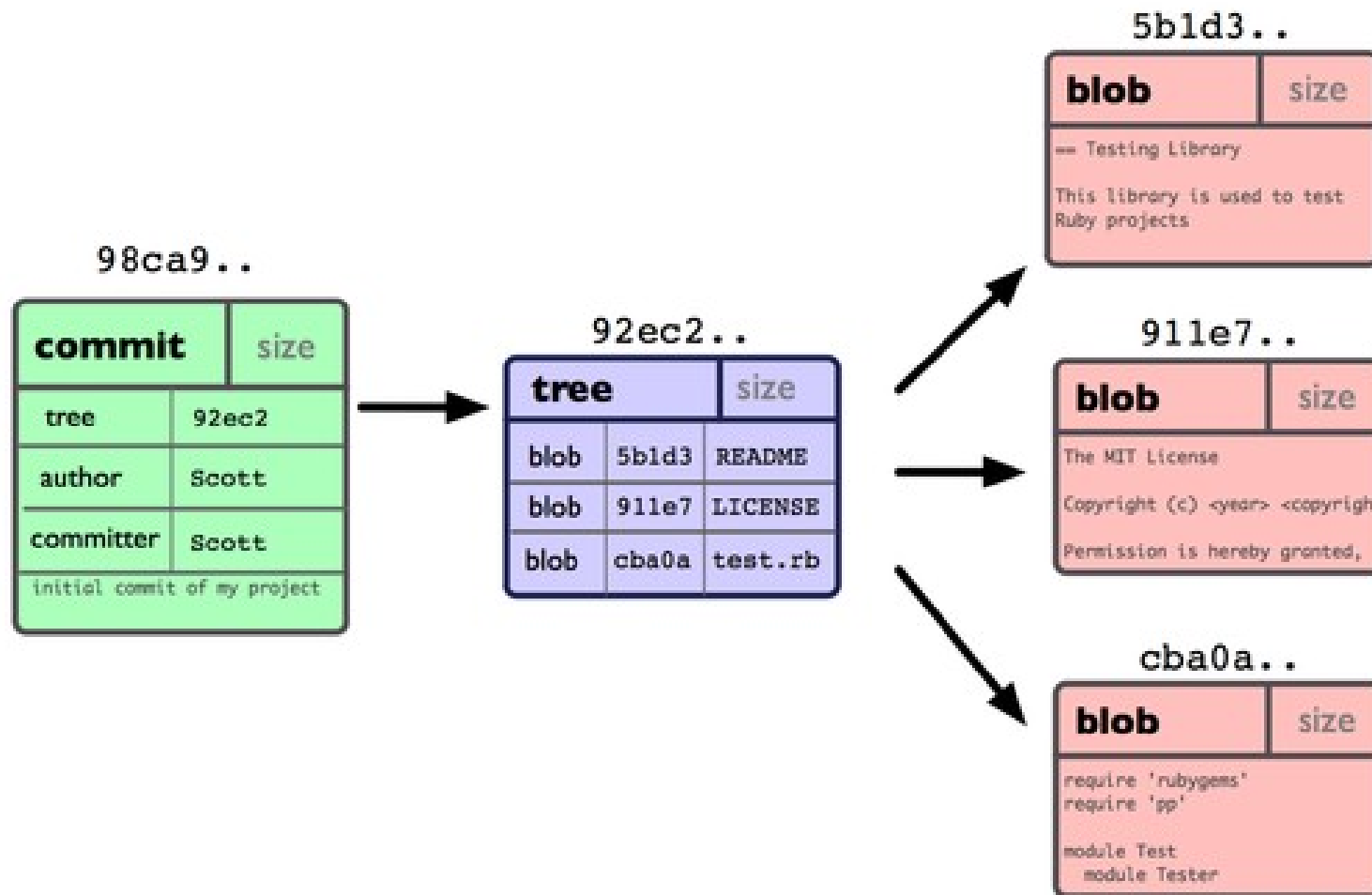
Todo es verificado antes de ser almacenado

Se identifica a partir de una suma de verificación.

Es imposible perder información al transmitirla sin que git lo pueda detectar

hash SHA-1:

24b9da6552252987aa493b52f8696cd6d3b00373



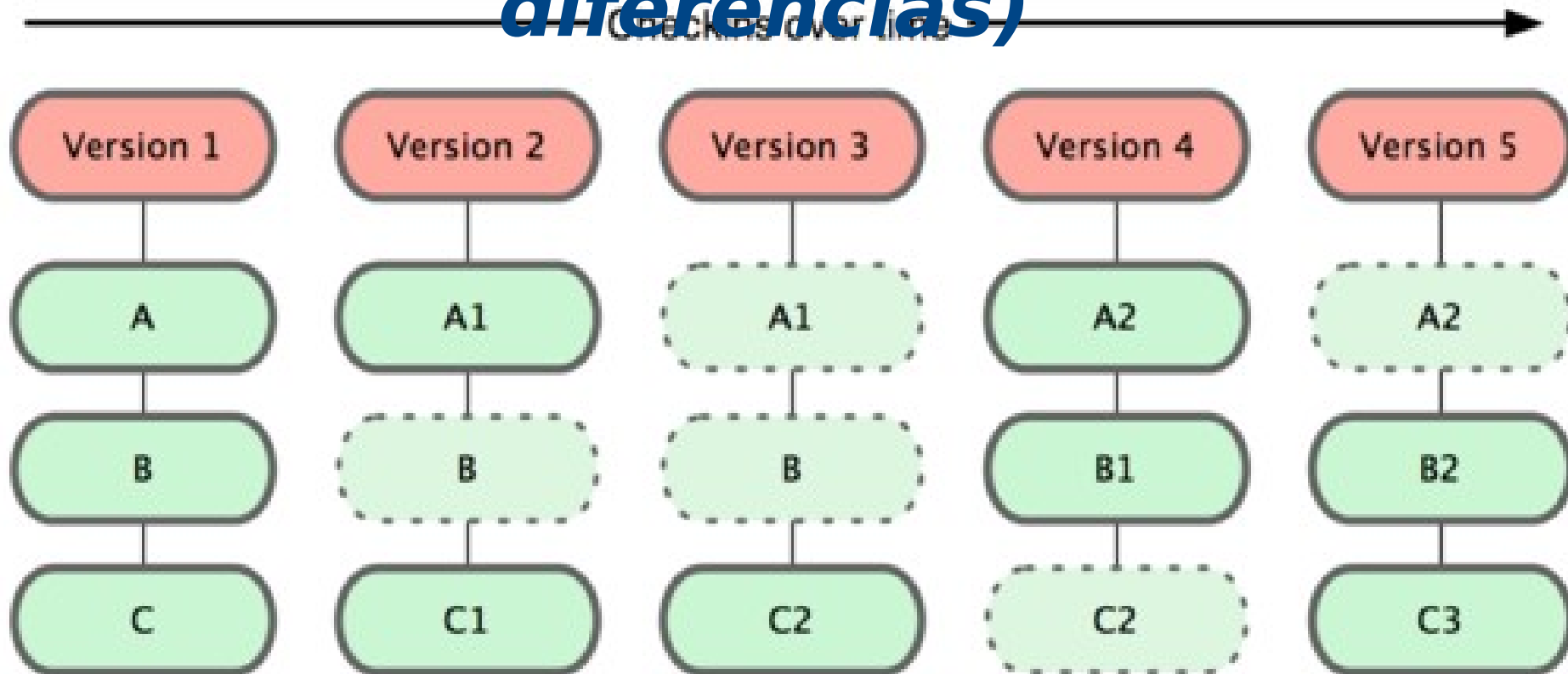
Instantáneas (No diferencias)

Git modela sus datos más como un **conjunto de instantáneas** de un mini sistema de archivos.

Al hacer un cambio Git hace una **foto** del aspecto de los archivos y guarda una **referencia** a esa instantánea.

Por eficiencia, si los archivos no se han modificado Git no almacena el archivo de nuevo, **sólo un enlace al archivo anterior idéntico que ya tiene almacenado**

Instantáneas (No diferencias)



Operaciones en local

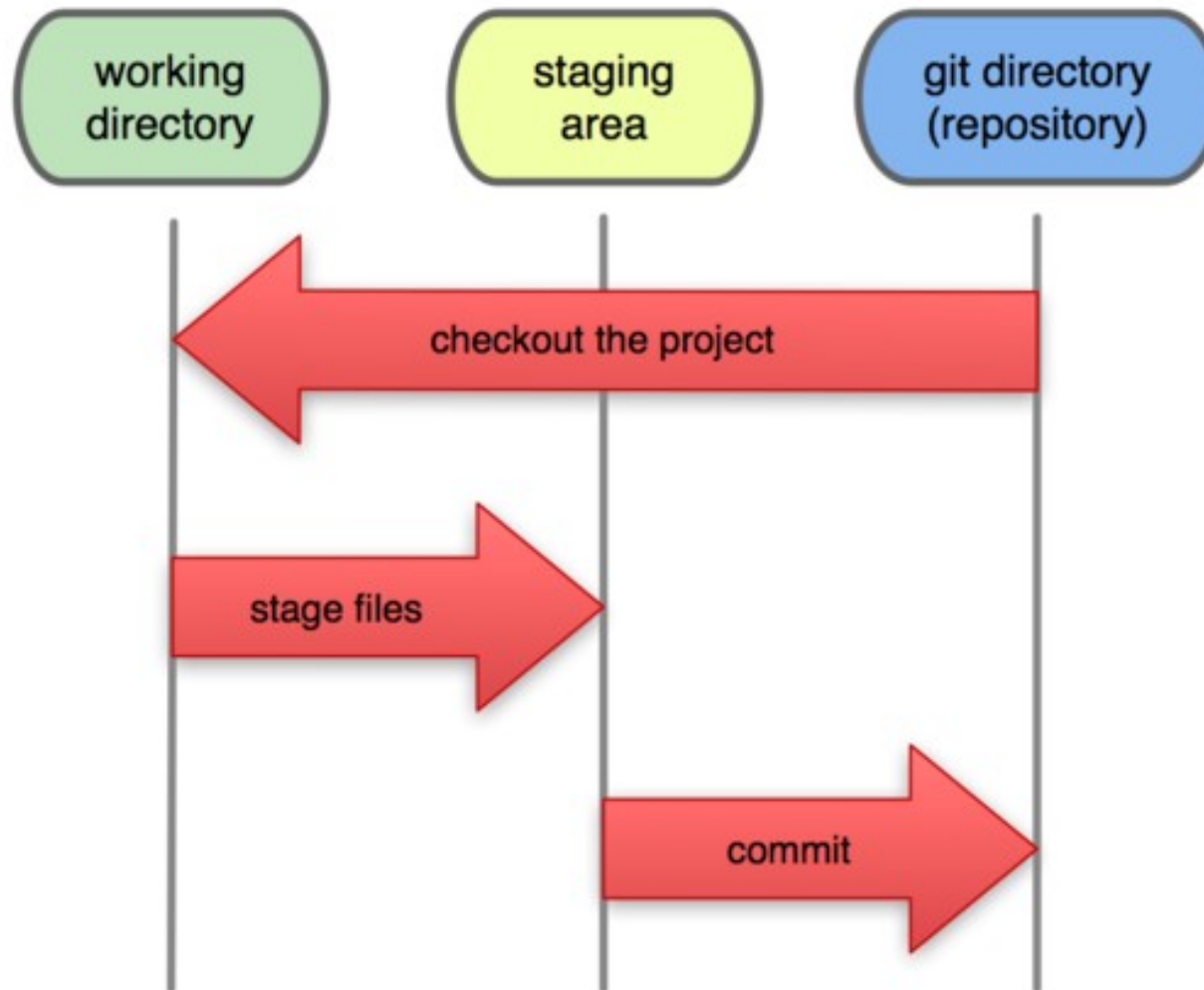
Commits **offline** !

Sin retrasos por operaciones de red

La historia del proyecto se encuentra en la DB local

Recuperar archivos de una versión de hace un mes haciendo un calculo de diferencias localmente.

Local Operations



Los tres estados

El área de preparación(**staging area**) es un archivo que almacena información sobre lo que irá en el próximo commit .Antes se le llamaba “el índice”.

El directorio (**repositorio**) es donde git almacena los *metadatos* y la *base de datos* de objetos para tu proyecto.

Los tres estados

Confirmado/no modificado(committed):

Los datos están almacenados de manera segura en el directorio.

Modificado(modified): se ha modificado el archivo pero todavía no se ha confirmado.

Preparado(staged): se ha marcado para confirmación un archivo modificado en su versión actual.

```
sudo apt-get install git-core gitk git-  
gui
```

Autocompletado

Descargar al directorio /home el fichero

<https://github.com/git/git/tree/master/contrib/completion>

Copiar al fichero .bashrc la linea:

```
source ~/.git-completion.bash
```

```
git + <tab>
```

Se almacena la configuración en el fichero **.gitconfig** del directorio **/home** del usuario

Configuración del **usuario**

git config --global user.name "Nombre Apellido"

git config --global user.email "email@gmail.com"

Colores para consola

git config --global color.status auto

git config --global color.branch auto

git config --list

El fichero **.gitignore** del directorio principal

```
# Compiled source #
#####
*.com
*.class
*.dll
*.exe
*.o
*.so

# Packages #
#####
# it's better to unpack these files and commit the raw source
# git has its own built in compression methods
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip

# Logs and databases #
#####
*.log
*.sql
*.sqlite

# OS generated files #
#####
.DS_Store*
ehthumbs.db
Icon?
Thumbs.db
```

cd ~/

mkdir repo1

cd repo1

mkdir datafiles

touch test01

touch test02

touch

datafiles/data.txt

ls > test01

git init

git add .

**git commit -m "Ficheros
iniciales"**

git log


```
echo "Cambiando fichero" > test01  
echo "nueva linea en test02" >  
test02
```

```
git diff
```

```
git commit -a -m "Los nuevos  
cambios"
```

```
echo "Nueva linea en test01 -A" > test01  
echo "Otra linea mas en test02 - B" > test02
```

```
git status  
git diff
```

```
git add . && git commit -m "Nuevos cambios -  
mensaje para el commit"
```

```
git commit --amend -m "He cambiado el mensaje  
del commit"
```

```
git log
```

gitk --all

cd ~/repo1

git clone --bare . ../repo-remoto.git

**#Mismo contenido que /.git en
repo1**

ls ~/repo-remoto.git

Enviar cambios a un repositorio remoto

```
cd ~/repo01
```

```
echo "Hello, hello. Turn your radio on" >  
test01
```

```
echo "Bye, bye. Turn your radio off" > test02
```

```
git commit -a -m "Algunos cambios"
```

```
# Push
```

```
git push ../repo-remoto.git
```

Cambios

**#desde el ultimo commit
git diff**

**#desde ayer
git diff "@{yesterday}"**

**#desde una versión concreta y 2 versiones
hacia atrás
git diff SHA1_HASH "master~2"**

**#Recuperar una versión concreta
git checkout SHA1_HASH**

git branch

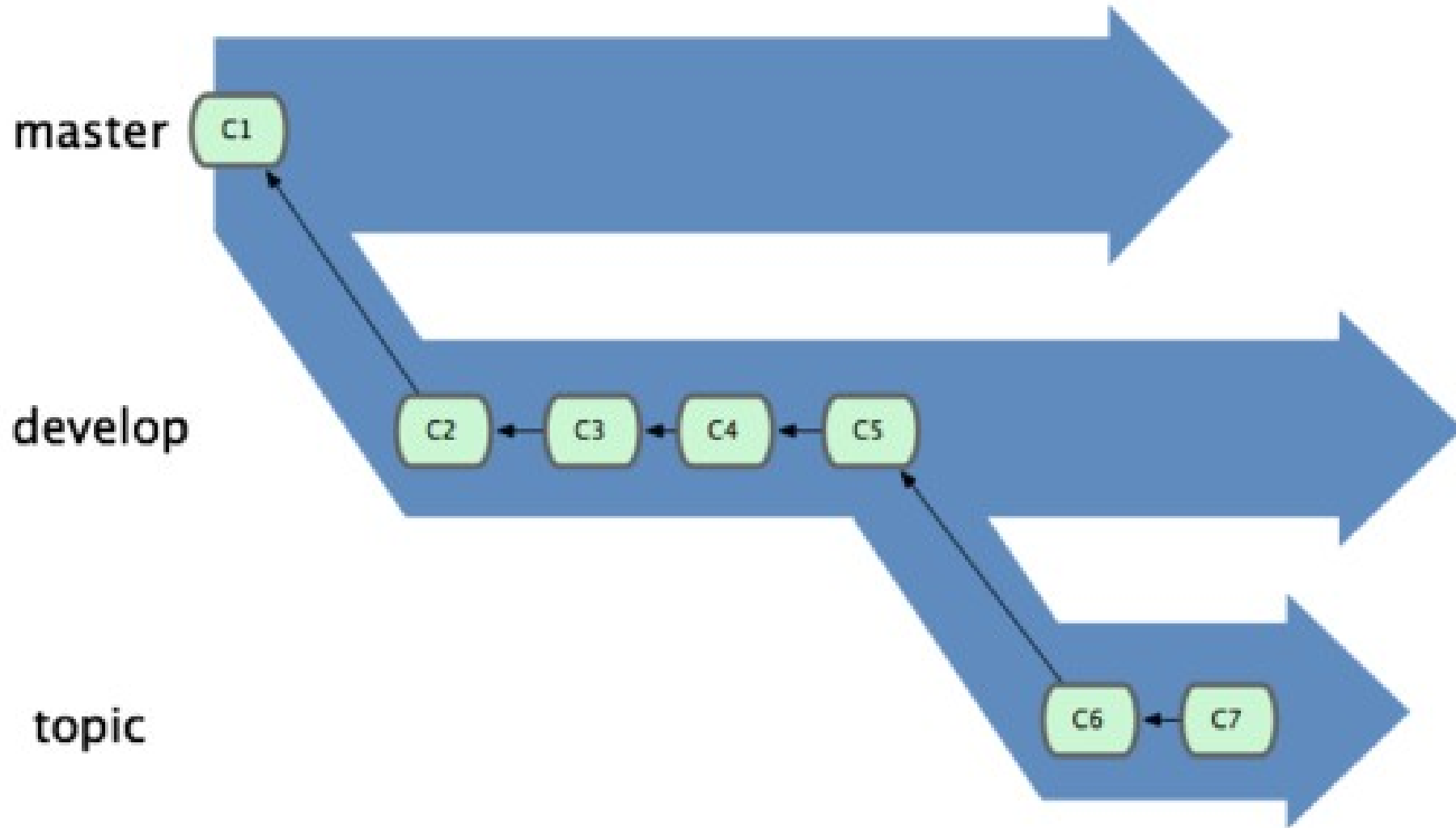
#incluye ramas remotas
git branch -a

git branch rama1

git checkout rama1

echo "En la rama1" > test01
git commit -a -m "Algunos cambios en la
nueva rama"

#volver a la rama principal
git checkout master
cat test01



git merge rama1

#resolver el conflicto a mano
git mergetool

<<<<<<< HEAD

Change in the first repository

=====

Change in the second repository

>>>>>>>

b29196692f5ebfd10d8a9ca1911c8b08127
c85f8

#eliminando rama
git branch -d probando

touch rebase.txt

git add . && git commit -m "rebase.txt añadido"

echo "primera linea" >> rebase.txt

git add . && git commit -m "contenido"

echo " otra linea mas" >> rebase.txt

git add . && git commit -m "mas contenido"

echo "Tercera linea" >> rebase.txt

git add . && git commit -m "Una tercera linea"

echo " esta es la ultima" >> rebase.txt

git add . && git commit -m "Ultima linea de código"

git log -stat

git log --pretty=format:"%h - %an, %ar : %s"

git log --pretty=oneline

git rebase -i HEAD~4

git merge rama1

#resolver el conflicto a mano
git mergetool

<<<<<<< HEAD

Change in the first repository

=====

Change in the second repository

>>>>>>>

b29196692f5ebfd10d8a9ca1911c8b08127
c85f8

#eliminando rama
git branch -d probando

Github (ssh key)

```
$ cd ~/.ssh (existe ??)
```

```
Ls
```

```
mkdir key_backup
```

```
cp id_rsa* key_backup
```

```
rm id_rsa*
```


```
ssh-keygen -t rsa -C "your_email@youremail.com"
```


```
$ ssh-keygen -t rsa -C "your_email@youremail.com"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/Users/your_user_directory  
/.ssh/id_rsa):<press enter>
```

```
Enter passphrase (empty for no passphrase):<enter a passphrase>  
Enter same passphrase again:<enter passphrase again>
```

```
Your identification has been saved in /Users/your_user_directory  
/.ssh/id_rsa.  
Your public key has been saved in /Users/your_user_directory  
/.ssh/id_rsa.pub.  
The key fingerprint is:  
01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db user_name@username.com  
The key's randomart image is:  
+--[ RSA 2048]-----+  
|      .+      +      |  
|      = o o .      |  
|      = * *        |  
|      o = +        |  
|      o S .        |  
|      o o =        |  
|      o . E        |  
+-----+-----+
```

Github

 SOCIAL CODING

 tran

[Explore G](#)

Create a New Repository

Project Name

Description (optional)

Homepage URL (optional)

Who has access to this repository? (You can change this later)

☒ **Anyone** ([learn more about public repos](#))

☐ **[Upgrade your plan to create more private repositories!](#)**

Create repository

Github

Global setup:

Set up git

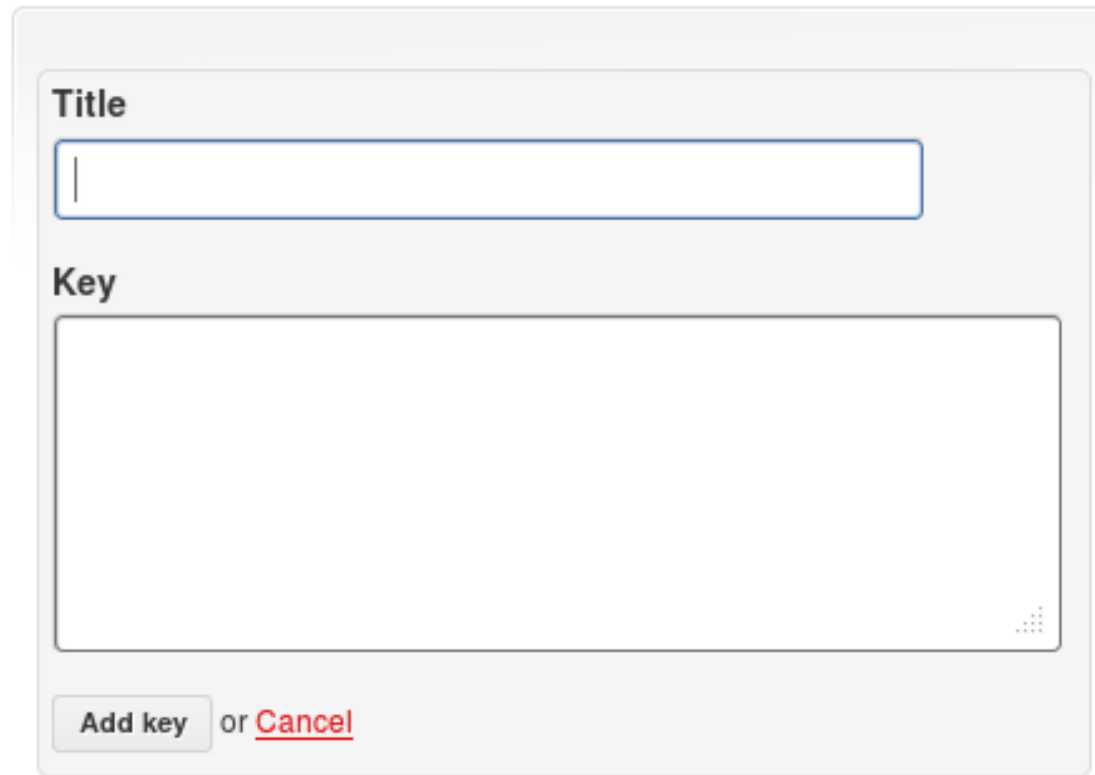
```
git config --global user.name "Fran Lucena"  
git config --global user.email fran.lucena@gmail.com
```

Next steps:

```
mkdir Taller-GIT  
cd Taller-GIT  
git init  
touch README  
git add README  
git commit -m 'first commit'  
git remote add origin git@github.com:franlu/Taller-GIT.git  
git push -u origin master
```

Existing Git Repo?

```
cd existing_git_repo  
git remote add origin git@github.com:franlu/Taller-GIT.git  
git push -u origin master
```

Title

Key

Add key or Cancel



*Av. Juan López de Peñalver, 21
Parque Tecnológico de
Andalucía
Málaga - España*

*(+34) 918 38 38 58
flucena@opentia.com
<http://www.opentia.com>*