Charla Introductoria a GIT

"the stupid content tracker"

Linus Torvalds

Junio 2021

Rafael Ignacio Zurita <rafa@fi.uncoma.edu.ar>

Advertencia: Estos slides traen ejemplos.

No copiar (ctrl+c) y pegar en un shell o terminal los comandos aquí presentes.

Algunos no funcionarán, porque al copiar y pegar tambien van caracteres “ocultos” (no visibles pero que están en el pdf) que luego interfieren en el shell.

Sucedió en vivo :)

Conviene “escribirlos” manualmente al trabajar.

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits Ejemplo 1 – practica 1
- Delorean – historial Ejemplo 2 – practica 2
- Ramas Ejemplo 3 – practica 3
- Flujo de trabajo Ejemplo 4 - practica 4
- Github – gitlab – bitbucket Ejemlpo 5 - practica 5
- Caracteristicas Avanzadas

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits Ejemplo 1 – practica 1
- Delorean – historial Ejemplo 2 – practica 2
- Ramas Ejemplo 3 – practica 3
- Flujo de trabajo Ejemplo 4 - practica 4
- Github – gitlab – bitbucket Ejemlpo 5 - practica 5
- Caracteristicas Avanzadas

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

1983 Stallaman se va del MIT, y junto a otros hackers comienzan el proyecto GNU y la FSF
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un sacadito de Finlandia escribe un pequeño kernel llamado Linux en un par de semanas, para PC.
Hoy en día lo usan tres mil millones de personas en sus smartphones. Otro tanto en Pcs, laptops y servidores
Desde entonces miles de hackers, empresas y desarrolladores colaboran alrededor del planeta
-2000 Hasta ese año el flujo de desarrollo se hacia por mail: usando diff y patch, dos herramientas de GNU
El compilador de C era GCC del proyecto GNU
2000 Se empezó a utilizar bitkeeper para el proyecto Linux, herramienta no libre, aunque tenía un cliente gratis
2000 y algo Un hacker cancherito quiso hacer ingeniería inversa, para intentar desarrollar uno libre
La empresa bitkeeper decide quitar la versión gratuita del cliente. Bardo total
2005 Torvalds se calienta y desarrolla en un par de meses git.
Hoy en dia lo usan probablemente miles de millones de personas (github sólo eportó 100 millones de repositorios)
2010- Aparecen servicios web para almacenar remotamente y publicamente proyectos gestionados con git
(github/gitlab/bitbucket)

Mi historia con git: arranco por acá, entre el bardo y la transición:
usé mucho tiempo diff, patch y mail manualmente
Costó acostumbrarme
Trabajé con git para el kernel Linux, Ofono y Jlime. Entre 208 y 2012

Git es un sistema de control de versiones, creado para gestionar el desarrollo del proyecto Linux,
distribuido, basado en snapshots explícitos, llamados cambios permanentes (commits)

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

~1983:

1983 Stallman y gente del MIT, junto a otros hackers comienzan el proyecto GNU y la FSF
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un sacadito de Finlandia escribe un pequeño kernel llamado Linux en un par de semanas, para PC
Hoy en día lo usan tres mil millones de personas en sus smartphones. Otro tanto en Pcs, laptops
Desde entonces miles de hackers, empresas y decenas de millones colaboran para él en el planeta
-2000 Hasta ese año el flujo de desarrollo se hacía por mail. usando diff y patch, dos herramientas de
El compilador de C era GCC del proyecto GNU
2000 Se empezó a utilizar bitkeeper para el proyecto Linux, herramienta no libre, aunque tenía un clien
2000 y algo Un hacker cancherito quiso hacer ingeniería inversa, para intentar desarrollar uno libre
La empresa bitkeeper decide quitar la versión gratuita del clien
2005 Torvalds se calienta y desarrolla en un par de meses git.
Hoy en dia lo usan probablemente miles de millones de persona de repositorios)
2010- Aparecen servicios web para almacenar remotamente y publicame
(github/gitlab/bitbucket)

MIT chau

Mi historia con git: arranco por acá, entre el bardo y la transición:
usé mucho tiempo diff, patch y mail manualmente
Costó acostumbrarme
Trabajé con git para el kernel Linux, Ofono y Jlime. Entre 208 y 2012

Git es un sistema de control de versiones, creado para el proyecto Linux,
distribuido, basado en snapshots explícitos anentes (commits)

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

~1990:

1983 Stallman y otros hackers comienzan el proyecto GNU y la FSF

Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel

1991

Desde

-2000

El com

2000

2000 y

2005

Hoy en día lo usan probablemente miles de millones de personas (github sólo eportó 100 millones de repositorios)

2010- Aparecen servicios web para almacenar remotamente y publicar proyectos gestionados con git

(github/gitlab/bitbucket)

Free Unix!

Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it away free(1) to everyone who can use it. Contributions of time, money, programs and equipment are greatly needed.

To begin with, GNU will be a kernel plus all the utilities needed to write and run C programs: editor, shell, C compiler, linker, assembler, and a few other things. After this we will add a text formatter, a YACC, an Empire game, a spreadsheet, and hundreds of other things. We hope to supply, eventually, everything useful that normally comes with a Unix system, and anything else useful, including on-line and hardcopy documentation.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer filenames, file version numbers, a crashproof file system, filename completion perhaps, terminal-independent display support,

Pero Falta el kernel!

ansición:

almente

Costó acostumbrarme

Trabajé con git para el kernel Linux, Ofono y Jlime. Entre 208 y 2012

Git es un sistema de control de versiones, creado para gestionar el desarrollo del proyecto Linux,
distribuido, basado en snapshots explícitos, llamados cambios permanentes (commits)

%15

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

1991:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki


Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones.  This has been brewing
since april, and is starting to get ready.  I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want.  Any suggestions
are welcome, but I won't promise I'll implement them :-)

                Linus (torvalds@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

%17

Charla Introductoria a GIT

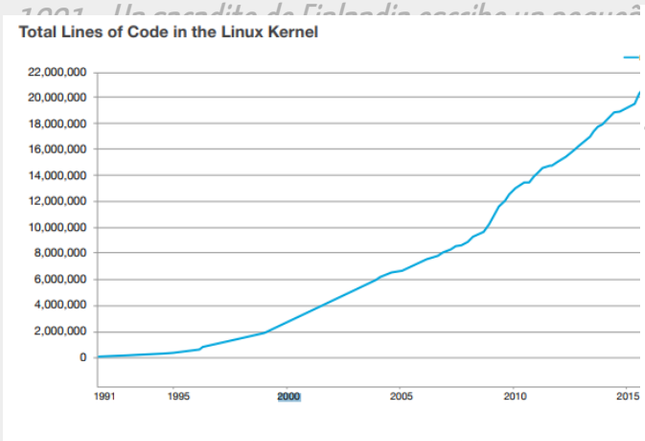
Historia: *"te lo resumo así nomás"*

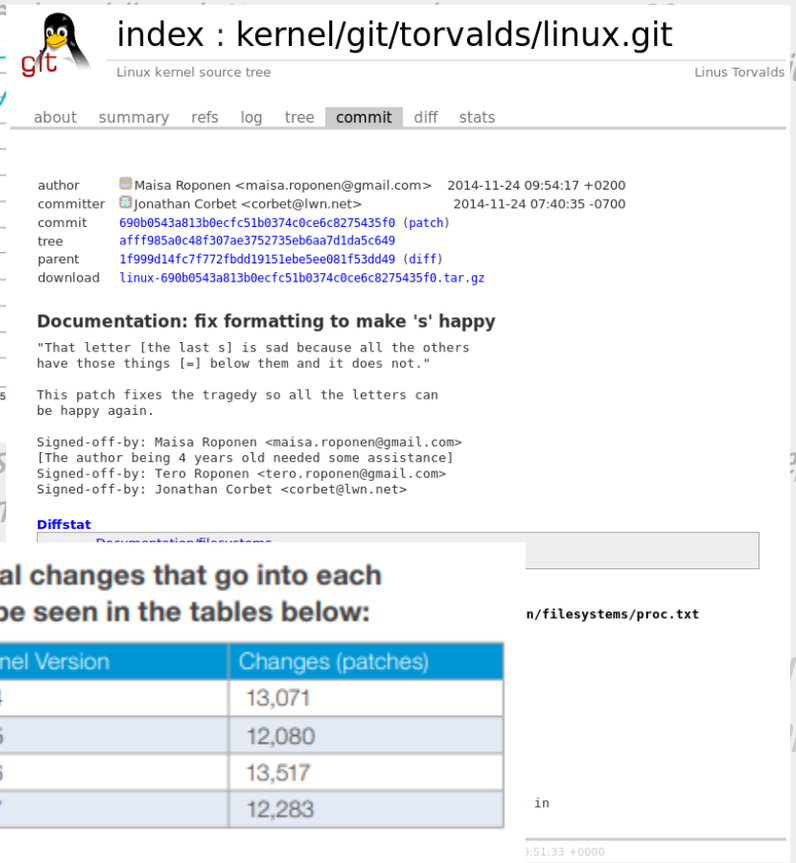
~2000: diff patch mail

1983 Stallman va del MIT junto a otros hackers comienza el proyecto GNU y la FSF

Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel

1991 Un estudiante de Finlandia escribe un pequeño [illegible]





Hoy en dia lo usan probablemente miles

2010- Aparecen servicios web para almacenar rep

One other result is that the number of individual changes that go into each kernel release is large and increasing, as can be seen in the tables below:

Kernel Version	Changes (patches)
3.19	12,461
4.0	10,346
4.1	11,916
4.2	13,694

Kernel Version	Changes (patches)
4.4	13,071
4.5	12,080
4.6	13,517
4.7	12,283

%20

```
[ Linux-kernel added back into the cc, because I actually think this is
  important. ]

On Tue, 21 Dec 2004, Jesper Juhl wrote:
>
> Should I just stop attemting to make these trivial cleanups/fixes/whatever
> patches? are they more noice than gain? am I being a pain to more skilled
> people on lkml or can you all live with my, sometimes quite ignorant,
> patches?
> I do try to learn from the feedback I get, and I like to think that my
> patches are gradually getting a bit better, but if I'm more of a bother
> than a help I might as well stop.

To me, the biggest thing with small patches is not necessarily the patch
itself. I think that much more important than the patch is the fact that
people get used to the notion that they can change the kernel - not just
on an intellectual level ("I understand that the GPL means that I have the
right to change my kernel"), but on a more practical level ("Hey, I did
that small change").

And whether it ends up being the right thing or not, that's how everybody
starts out. It's simply not possible to "get into" the kernel without
starting out small, and making mistakes. So I very much encourage it, even
if I often don't have the time to actually worry about small patches, and
I try to get suckers^H^H^H^H^H^Hother developers like Rusty to try to
acts as quality control and a "gathering place".

Btw, this is why even "trivial patches" really do take time - they often
have trivial mistakes in them, and it's not just because there are more
inexperienced people doing them - most of _my_ mistakes tend to be at the
truly idiotic level, just because it "looked obvious", and then there's
something that I miss.

So at one level I absolutely _hate_ trivial patches: they take time and
effort to merge, and individually the patch itself is often not really
obviously "worth it". But at the same time, I think the trivial patches
are among the most important ones - exactly because they are the "entry"
patches for every new developer.

I just try really hard to find somebody else to worry about them ;)

(It's not a thankful job, btw, exactly because it _looks_ so trivial. It's
easy to point to 99 patches that are absolutely obvious, and complain
about the fact that they haven't been merged. But they take time to merge
exactly because of that one patch that _did_ look obvious, but wasn't.
And actually, it's usually not 99:1, it's usually more like 10:1 or
something).

So please don't stop. Yes, those trivial patches _are_ a bother. Damn,
they are _horrible_. But at the same time, the devil is in the detail, and
they are needed in the long run. Both the patches themselves, and the
people that grew up on them.

                Linus
-
```

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

~2000: usamos Bitkeeper

1983 Stal... MIT junto a ... proyect... ES...
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un sacadito de Finlandia escribe un pequeño kernel llamado Linux en un par de semanas, para ...
Hoy en día lo usan tres mil millones de personas en sus smartphones. Otro tanto en Pcs, lap...
Desde entonces miles de hackers, empresas y desarrolladores colaboran alrededor del planeta
-2000 Hasta ese año el flujo de desarrollo se hacia por mail: usando diff y patch, dos herramientas de...
El compilador de C era GCC del proyecto GNU
2000 Se empezó a utilizar bitkeeper pa... el proyecto Linux, herramienta no libre, aunque tenía un cli...
2000 y algo ... ibre
... cliente. ... total

~2005: BARDO TOTAL

2005 Torvalds se calienta y desarrolla en un par de meses git.
Hoy en día lo usan probablemente miles de millones de personas (github sólo eportó 100 m...
2010- Aparecen servicios web para almacenar remotamente y publicamente proyectos gestionados c...
(github/gitlab/bitbucket)

"He'd have been a hero to me. It's unquestionably true that BitKeeper has advanced the state of SCM technology. Anybody who argues against that just doesn't know what the hell he is talking about. But I'd have loved even an 'almost-as- good' open-source SCM, because that would obviously just be a good idea.

"But that's not what Tridge did. He didn't write a 'better SCM than BK'. He didn't even try - it wasn't his goal. He just wanted to see what the protocols and data was, without actually producing any replacement for the (inevitable) problems he caused and knew about.

"He didn't create something new and impressive. He just tore down something new (and impressive) because he could, and rather than helping others, he screwed people over. And you expect me to respect that kind of behaviour?

"Anobody (sic) that compares that to Open Office (or even samba, which Tridge did write) is an idiot. Open office and samba are constructive projects that actually do something useful, and are technically advanced quite regardless of

Git es un sistema de control de versiones, creado para gestionar el desarro...
distribuido, basado en snapshots explícitos, llamados cambios permanentes (commits)

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

2005: " Put... que los remil par.. "

1983 Stallman inicia el proyecto GNU y la FSF
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un s
H
Desde en
-2000 Ha
El compil
2000 Se e
2000 y alg
2005 Tor
H
2010- Apa

Git es u

...on git: arranco por acá, entre el bardo y la transición:
usé mucho tiempo diff, patch y mail manualmente
...umbrarme
...git para el kernel Linux, Ofono y Jlime. Entre 208 y 2012

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

2005: " Put… que los remil par.. "

1983 [illegible] de MIT, y junto a otros hackers comienzan [illegible] proyecto GNU y la [illegible]
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un sacadito de Finlandia escribe un pequeño kernel llamado Linux en un par de semanas, para PC.
Hoy en día lo usan tres mil millones de personas en sus smartphones. Otro tanto en Pcs, laptops y servidores
Desde entonces miles de hackers, empresas y desarrolladores colaboran alrededor del planeta
-2000 Hasta ese año el flujo de desarrollo se hacia por mail: usando diff y patch, dos herramientas de GNU
El compilador de C era GCC del proyecto GNU
2000 Se empezó a utilizar bitkeeper para el proyecto Linux, herramienta no libre, aunque tenía un cliente gratis
2000 y algo [illegible] desarrollar uno libre
La empresa bitkeeper [illegible] total

… dos meses later …

2005 Torvalds se calienta y desarrolla en un par de meses git.
Hoy en dia lo usan probablemente miles de millones de personas (github sólo eportó 100 millones de reposit
2010- Aparecen servicios web para almacenar remotamente y publicamente proyectos gestionados con git
(github, gitlab, bucket

"bueh, acá tá, se llama

Git es un sistema de control de versiones, creado para gestionar el desarrollo del proyecto Linux,
distribuido, basado en snapshots explícitos llamados cambios permanentes (commits)

the stupid content tracker, o git" (que rima con shit)

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

1983 Stallaman se va del MIT, y junto a otros hackers comienzan el proyecto GNU y la FSF
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un sacadito de Finlandia escribe un pequeño kernel llamado Linux en un par de semanas, para PC.
Hoy en día lo usan tres mil millones de personas en sus smartphones. Otro tanto en Pcs, laptops y servidores
Desde entonces miles de hackers, empresas y desarrolladores colaboran alrededor del planeta
-2000 Hasta ese año el flujo de desarrollo se hacia por mail: usando diff y patch, dos herramientas de GNU
El compilador de C era GCC del proyecto GNU
2000 Se empezó a utilizar bitkeeper para el proyecto Linux, herramienta no libre, aunque tenía un cliente gratis
2000 y algo Un hacker cancherito quiso hacer ingeniería inversa, para intentar desarrollar uno libre
La empresa bitkeeper decide quitar la versión gratuita del cliente. Bardo total
2005 Torvalds se calienta y desarrolla en un par de meses git.
Hoy en dia lo usan probablemente miles de millones de personas (github sólo eportó 100 millones de repositorios)
2010- Aparecen servicios web para almacenar remotamente y publicamente proyectos gestionados con git (github/gitlab/bitbucket)

Mi historia con git: arranco por acá, entre el bardo y la transición:
usé mucho tiempo diff, patch y mail manualmente
Costó acostumbrarme
Trabajé con git para el kernel Linux, Ofono y Jlime. Entre 2008 y 2012

git es un sistema de control de versiones, creado para gestionar el desarrollo del proyecto Linux, distribuido, basado en snapshots explícitos, llamados cambios permanentes (commits)

Charla Introductoria a GIT

Historia: *"te lo resumo así nomás"*

1983 Stallman y varios hackers se juntan a crear hackers comunidad en el proyecto GNU y la FSF
Desarrollan todo lo necesario para un sistema UNIX pero faltaba el kernel
1991 Un sacadito de Finlandia escribe un pequeño kernel llamado Linux en un par de semanas, para PC.
Hoy en día lo usan tres mil millones de personas en sus smartphones. Otro tanto en PCs, tantos servidores
Desde entonces miles de hackers, empresas y desarrolladores colaboran alrededor del planeta
-2000 Hasta ese año el flujo de desarrollo se hacía por mail: usando diff y patch, dos herramientas de GNU
El compilador de C era GCC del proyecto GNU
2000 Se empezó a utilizar bitkeeper para el proyecto Linux, herramienta no libre, aunque tenía un cliente gratis
2000 y algo Un hacker cancherito quiso hacer ingeniería inversa para intentar usar bitkeeper libre
La empresa bitkeeper decide quitar la versión gratuita del cliente. Boooom!
2005 Torvalds se calienta y desarrolla en un par de meses git.
Hoy en dia lo usan probablemente miles de millones de personas (github sólo eportó 100 millones de repositorios)
2010- Aparecen servicios web para almacenar remotamente y publicar los repositorios (github/gitlab/bitbucket)

POR LO TANTO:

GIT no es GITHUB,

ni GITLAB ni BITBUCKET

ni SOURCEFORGE

Mi historia con git: arranco por acá, entre el bardo y la transición:
usé mucho tiempo diff, patch y mail manualmente
Costó acostumbrarme
Trabajé con git para el kernel Linux, Ofono y Jlime. Entre 208 y 2012

Git es un sistema de control de versiones, creado para gestionar el desarrollo del proyecto Linux,
distribuido, basado en snapshots explícitos, llamados cambios permanentes (commits)

%30

Charla Introductoria a GIT

Contenido:

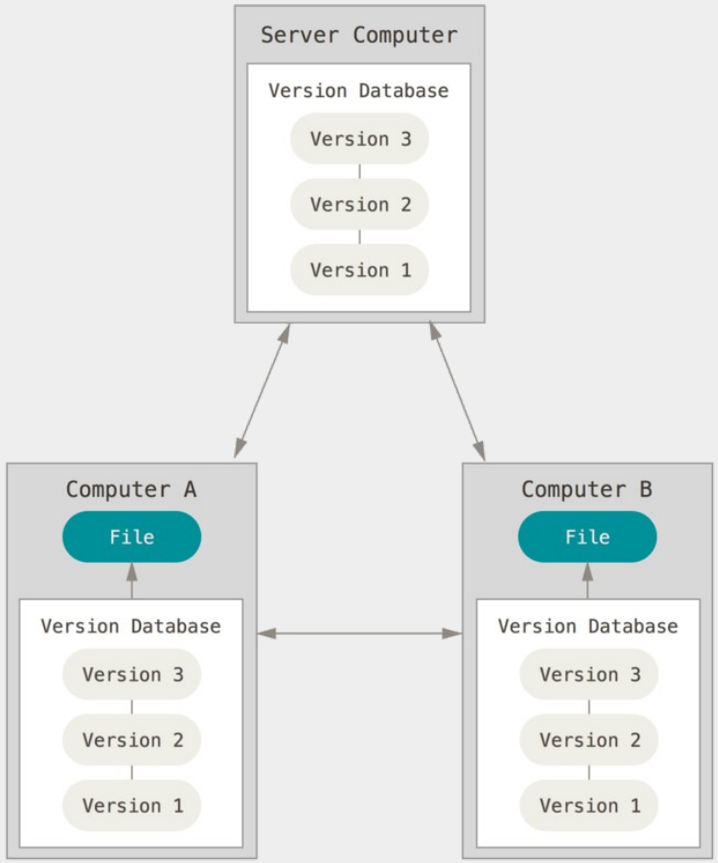
- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Caracteristicas

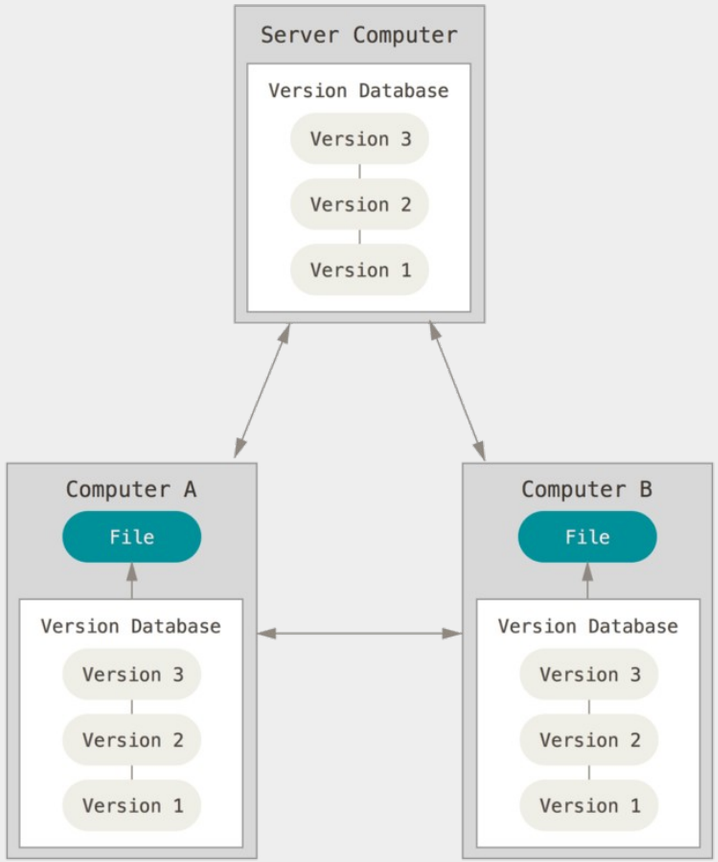
- Revisar el pasado



Charla Introductoria a GIT

Caracteristicas

- Revisar el pasado
- Desarrollo distribuido

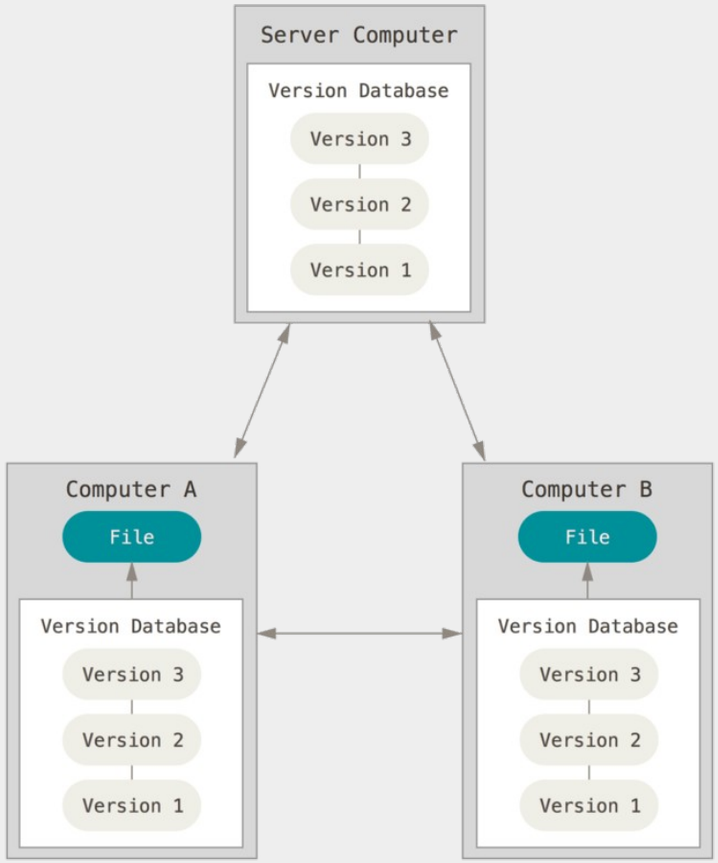


%35

Charla Introductoria a GIT

Caracteristicas

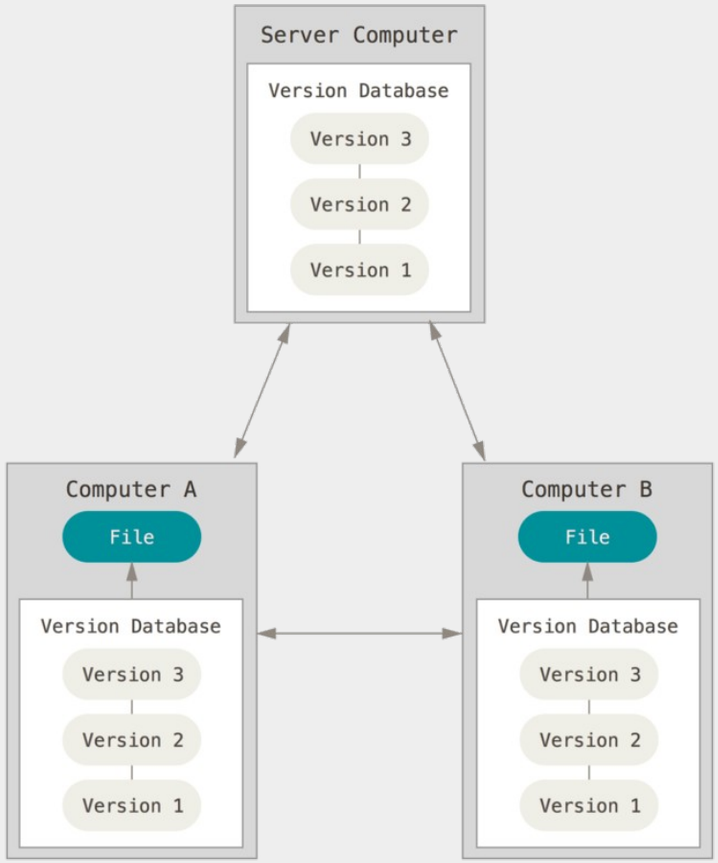
- Revisar el pasado
- Desarrollo distribuido
- Ligero



Charla Introductoria a GIT

Caracteristicas

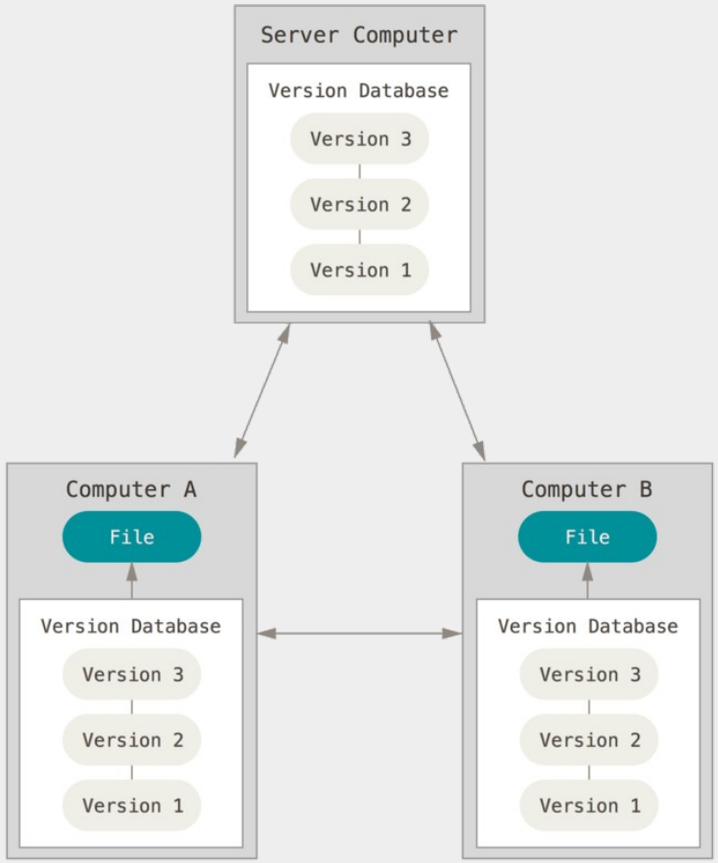
- Revisar el pasado
- Desarrollo distribuido
- Ligero
- Escalable



Charla Introductoria a GIT

Caracteristicas

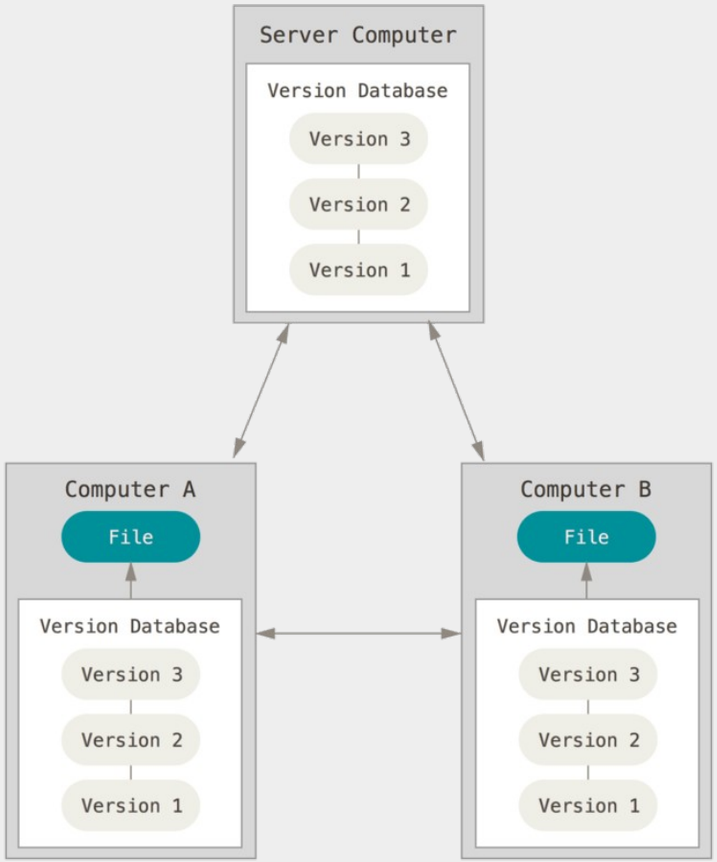
- Revisar el pasado
- Desarrollo distribuido
- Ligero
- Escalable
- Una copia contiene todo el historial



Charla Introductoria a GIT

Caracteristicas

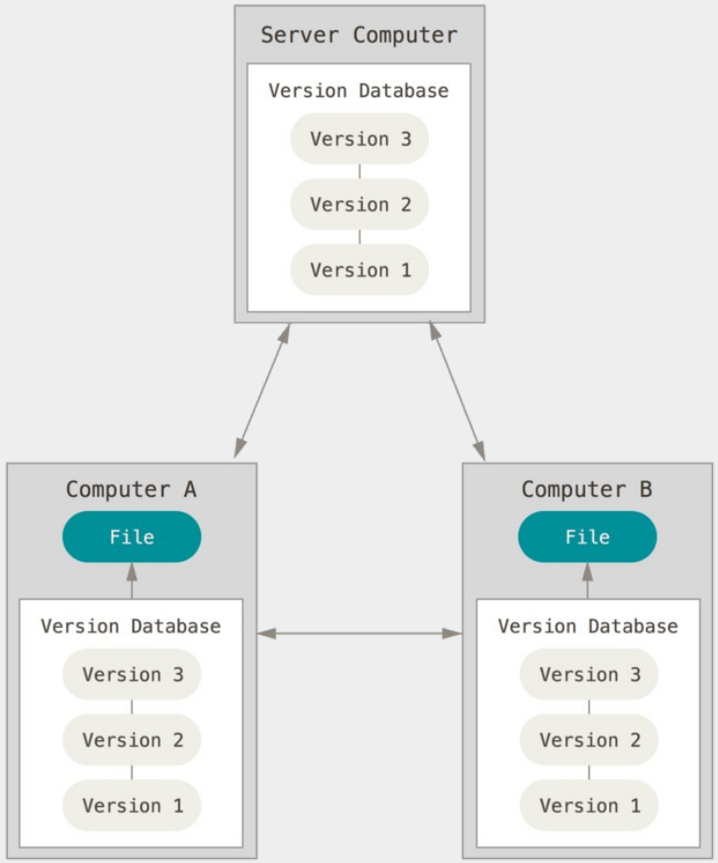
- Revisar el pasado
- Desarrollo distribuido
- Ligero
- Escalable
- Una copia contiene todo el historial
- Acceso remoto por ssh, http, y git



Charla Introductoria a GIT

Caracteristicas

- Revisar el pasado
- Desarrollo distribuido
- Ligero
- Escalable
- Una copia contiene todo el historial
- Acceso remoto por ssh, http, y git
- Busqueda binaria en una regresión

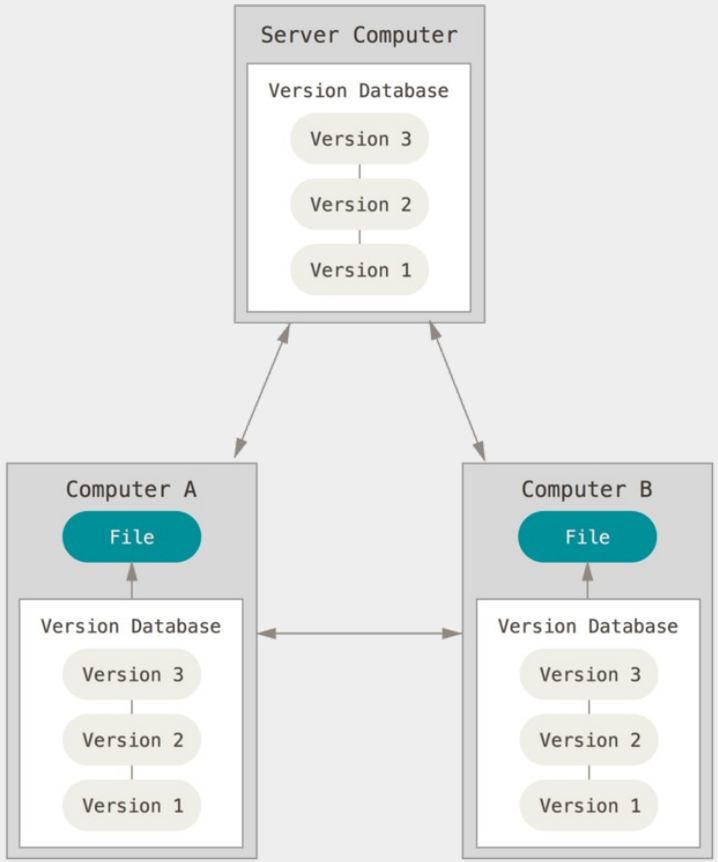


%35

Charla Introductoria a GIT

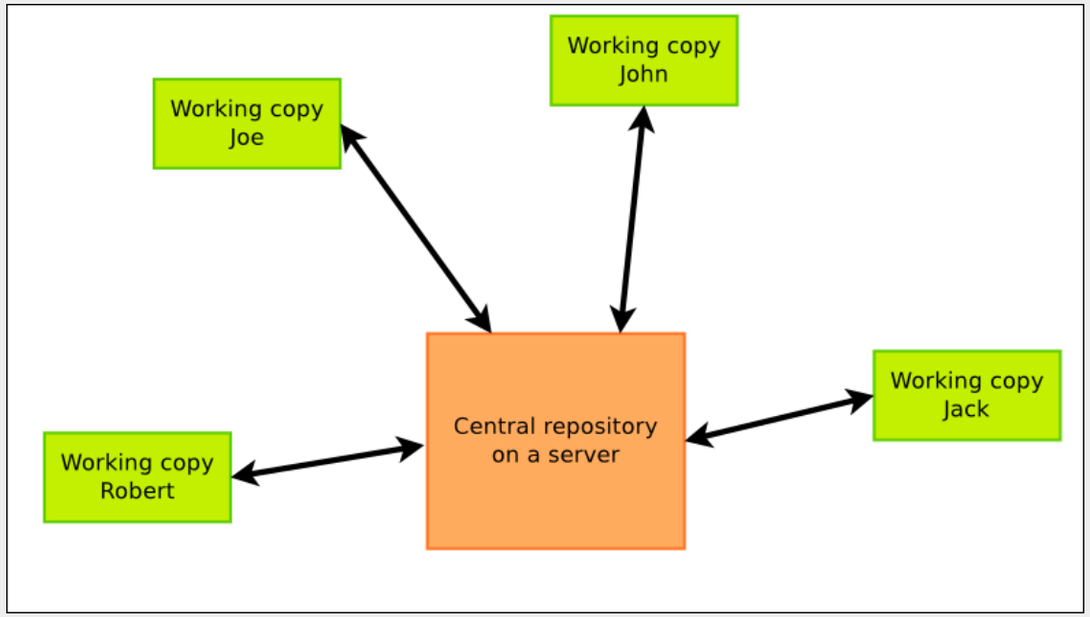
Caracteristicas

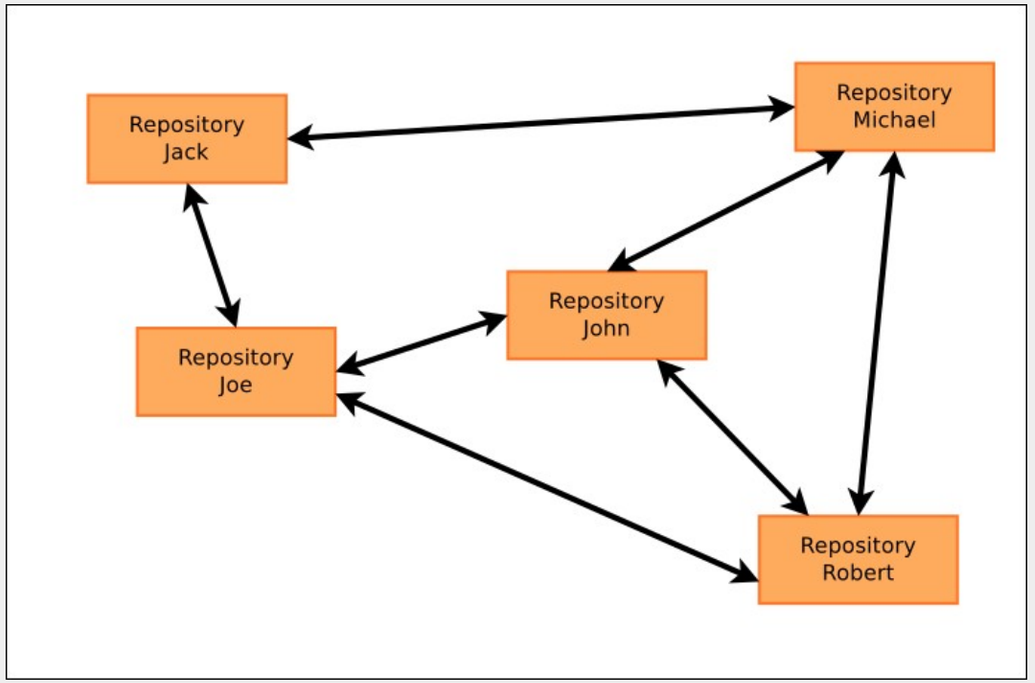
- Posibilita ir al pasado y verificar por qué, quien, y cuando se incorporó una porción de código
- Desarrollo distribuido (paralelo)
- Ligero (mayormente todo trabajo es local)
- Escalable (ej: el proyecto Linux tiene ~30 millones de lineas de codigo, en ~27 mil archivos de codigo fuente en C, y trabajaron ~30 mil desarrolladores)
- Una copia contiene todo el historial
- Soporta acceso remoto por ssh, http, y git
- Permite realizar busqueda binaria en una regresión



Charla Introductoria a GIT

Git es un sistema de control de versiones distribuido





Instalación – Configuración

1. Instalación

```
en Linux:    apt-get install git
en Windows:  video de.. (instala git y un ambiente de desarrollo UNIX)
```

2. Configuración (datos del desarrollador)

```
git config --global user.name "Rafael Ignacio Zurita"
git config --global user.email rafa@fi.uncoma.edu.ar
```

Hay 3 tipos de configuraciones:
1.A nivel del proyecto repo/.git/config
2.A nivel del usuario ~/.gitconfig
3.A nivel global del sistema /*etc*/gitconfig

3. Obteniendo ayuda

```
# en Linux
man git add     # pagina de manual de git-add
man git commit  # pagina de manual de git-commit
apropos git     # encuentra todas las paginas de manual sobre git
```

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Iniciando un repositorio – commits – Ejemplo 1

1. Crear un repositorio vacio (ver sección github tambien)

```
mkdir repo
cd repo/
git init .
```

2. Primer commit

```
# creamos un archivo (ej: echo "hola mundo" > LEAME)
git add LEAME
git commit -m "agregando el archivo inicial"
```

3. Verificamos el cambio

```
git log
```

Si necesitamos un repositorio remoto publico y trabajar con él localmente:

```
# En el servidor remoto publico:
git init --bare repo.git
# En nuestra pc local:
git clone ssh://user@pse.fi.uncoma.edu.ar/home/user/repo.git

cd repo/
... etc ...
# Luego de crear cambios localmente
# los empujamos al servidor remoto público
git push
```

Antes de git commit conviene ejecutar

```
git status
```

y verificar lo que estamos por confirmar

Charla Introductoria a GIT

En este ejemplo, Juan y Pedro trabajan juntos sobre un mismo repositorio. Para esto, pedro, crea primero un repositorio compartido para trabajar con Juan.

El repositorio se “crea” en la Maquina A. Así que pedro, desde la PC de su casa **(Maquina B)**, primero se conecta a la a **pse.fi.uncoma.edu.ar (Maquina A)** así:

```
ssh git@pse.fi.uncoma.edu.ar
```

Una vez “loggeado” crea el repositorio compartido así (esto lo ejecuta dentro de la **pse.fi.uncoma.edu.ar (Maquina A)**:

```
git init –bare repojuanypedro.git
```

Listo, repo creado en Maquina A. pedro cierra la sesión en **pse.fi.uncoma.edu.ar (Maquina A)** así:

```
exit [ENTER]
```

Juan, desde la PC de su casa **(Maquina C),** clona el nuevo repositorio así:

```
git clone ssh://git@pse.fi.uncoma.edu.ar/home/git/repojuanypedro.git
```

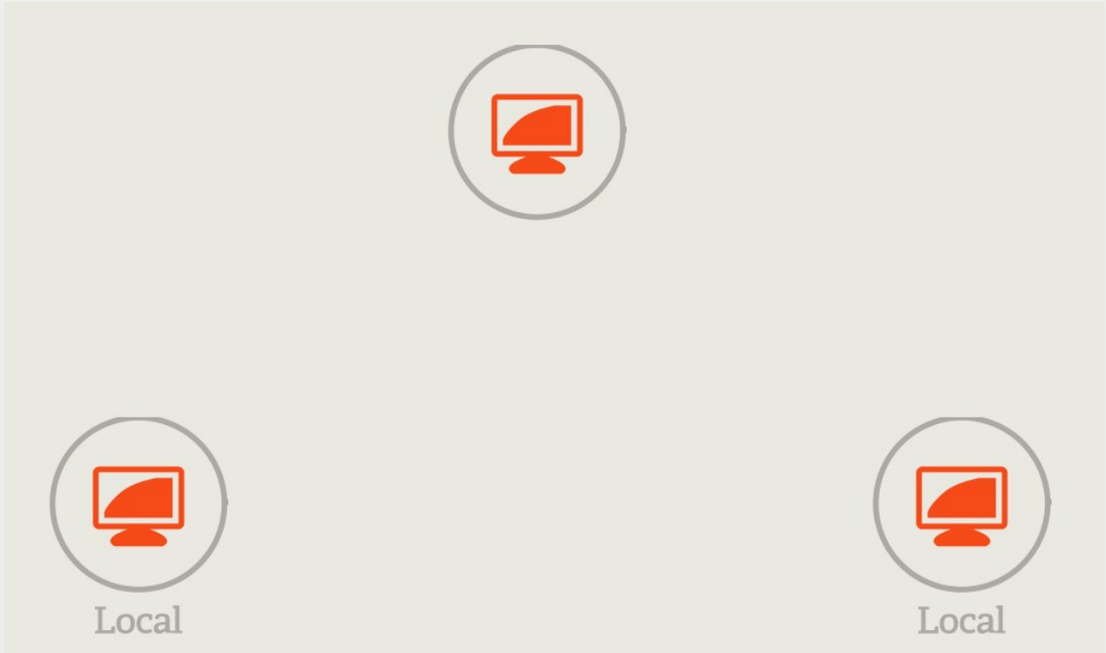
Listo, ahora Juan puede ingresar al directorio repojuanypedro/ y trabajar. Su primer commit podría ser así:

```
cd repojuanypedro/
# juan copia un archivo a esta carpeta o crea uno nuevo, supongamos que se llama main.c
git add main.c
git commit -m “codigo fuente principal”
git push
```

Este ultimo git push llevó los cambios de Juan al repositorio remoto publico **(Maquina A)**.

CONTINUA en la siguiente página…..

Maquina (A)
Simple PC Linux o servidor visible en internet (Ejemplo: **pse.fi.uncoma.edu.ar**). Tiene un usuario llamado **git**



Maquina (B)
PC Linux de Pedro. La tiene en su casa.
Usa localmente un usuario llamado **pedro**.

Maquina (C)
PC Linux de Juan. La tiene en su casa.
Usa localmente un usuario llamado **juan**.

Charla Introductoria a GIT

Ahora pedro, desde la PC de su casa **(Maquina B),** clona el nuevo repositorio así:

```
git clone ssh://git@pse.fi.uncoma.edu.ar/home/git/repojuanypedro.git
```

Listo, como lo hizo luego de un commit y push de Juan, vendrá a su máquina B el commit de Juan tambien.

Pedro trabaja en la PC de su casa con el repositorio:

```
cd repojuanypedro/
# pedro copia un archivo a esta carpeta o crea uno nuevo, supongamos que se llama utils.c
git add utils.c
git commit -m “codigo de biblioteca”
git push
```

Este ultimo git push llevó los cambios al repositorio remoto publico **(Maquina A)**.

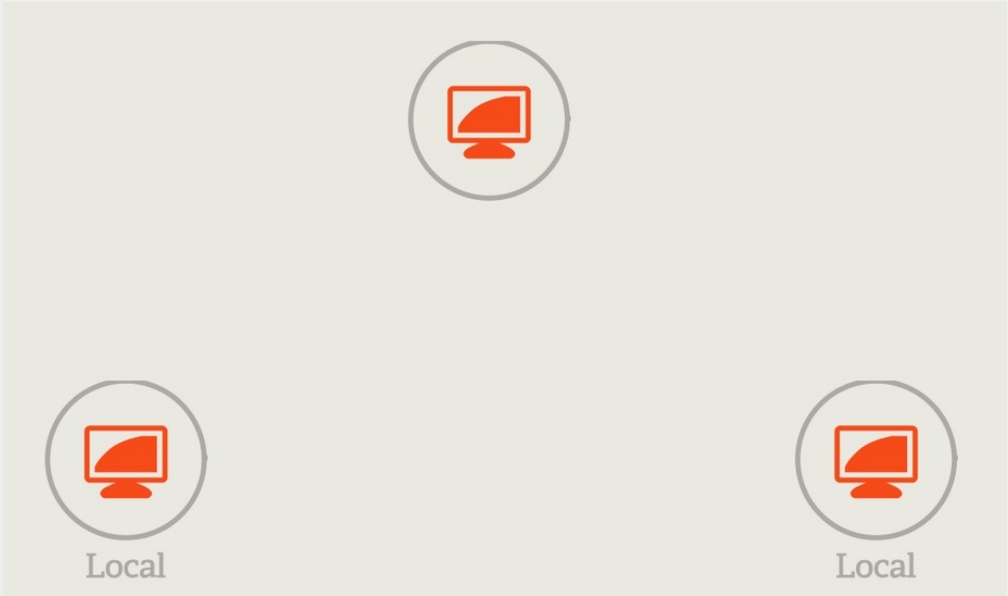
Ahora el repositorio publico y remoto **(Maquina A)** tiene dos commits, uno de Juan y uno de Pedro.

Si Juan hace un nuevo cambio y vuelve a intentar un push, git se “quejará”, y le dirá que su repositorio local en Maquina B es diferente al remoto. Entonces, lo que hace Juan es un git pull. Esto traerá el ultimo commit de Pedro y generará un commit nuevo automaticamente, para fusionar el commit que Juan tenía antes del push y el de Pedro que ya está publicado en Maquina A. Luego del git pull y lograr la fusión Juan puede volver a intentar el git push para emitir los cambios. Si esto le parece confuso, haga una pruebita :)

De esta manera, Juan y Pedro trabajan en conjunto con una configuración minima sobre un mismo repositorio.

NOTA: Ambos, Juan y Pedro, están utilizando la máquina A con el mismo usuario git. Esto puede no ser del todo recomandable. Aún así, los commits tendrán como autor a Juan y Pedro dependiendo de quien fue el autor, ya que esos commits se establecen en las Pcs de Juan y Pedro localmente.

Maquina (A)
Simple PC Linux o servidor, visible en internet (Ejemplo: **pse.fi.uncoma.edu.ar**). Tiene un usuario llamado **git**



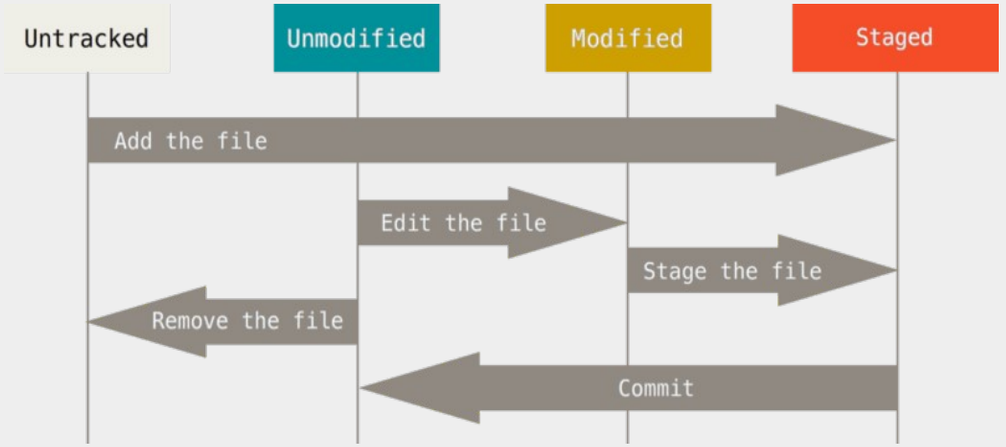
Maquina (B)
PC Linux de Pedro. La tiene en su casa.
Usa localmente un usuario llamado **pedro**.

Maquina (C)
PC Linux de Juan. La tiene en su casa.
Usa localmente un usuario llamado **juan**.

Charla Introductoria a GIT

Iniciando repositorio - commits

1. Varios cambios permanentes




```
# creamos los archivos README main.c LICENSE y Makefile
git add README main.c
git add LICENSE
git commit -m "segunda version"

# creamos o modificamos varios archivos más
git add utils.c utils.h
git add include/    # ATENCION: agrega a staging todos los archivos
                    #           dentro del directorio recursivamente
git commit -m "tercera version"
```

2. Verificamos los cambios

```
git log
```

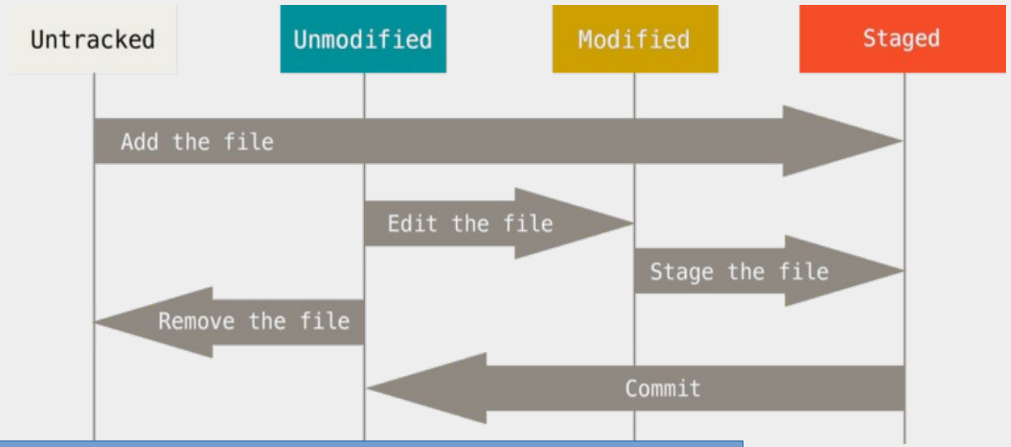
Charla Introductoria a GIT

Iniciando repositorio - commits

git presta atención a todos los archivos locales:

Los que pertenecen al repo
Los que no pertenecen
Los que pertenecen y fueron modificados

```
# Verificamos las areas y cambios pendientes
git status
git diff
```



Charla Introductoria a GIT

Commits y su descipción

```
git commit -m "…"
```

```
net: fix rds_iovec page count overflow
```

Pero, en proyectos importantes
la descripción debería estar compuesta de (ejemplo):

```
net: fix rds_iovec page count overflow

As reported by Thomas Pollet, the rdma page counting can overflow.  We
get the rdma sizes in 64-bit unsigned entities, but then limit it to
UINT_MAX bytes and shift them down to pages (so with a possible "+1" for
an unaligned address).

So each individual page count fits comfortably in an 'unsigned int' (not
even close to overflowing into signed), but as they are added up, they
might end up resulting in a signed return value. Which would be wrong.

Catch the case of tot_pages turning negative, and return the appropriate
error code.

Reported-by: Thomas Pollet <thomas.pollet@gmail.com>
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
Signed-off-by: Andy Grover <andy.grover@oracle.com>
Signed-off-by: David S. Miller <davem@davemloft.net>
```

Las descripciones completas para commits se pueden realizar ejecutando simplemente:

```
git commit
# ejecutará un editor predeterminado (vi / nano / ed en Linux)
# para redactar todas estas descripciones manualmente
```

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1

Ejemplo 2 – practica 2

Ejemplo 3 – practica 3

Ejemplo 4 - practica 4

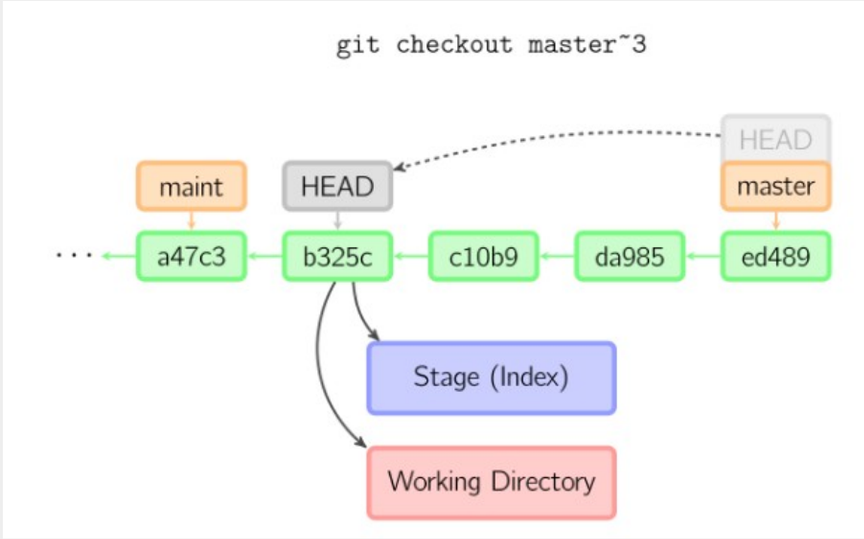
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Delorean – historial

1. Leer el historial de cambios (commits)

```
git log    # ver todo el historial de cambios (commits)
git log -p # ver todo el historial de parches (commits patchs)
git log foo.c # ver todos los commits que afectan a este archivo
git log hash/tag..hash/tag # ver todo el historial entre dos commits
git show hash/tag # ver el cambio en forma de parche
```



2. Viajar en el tiempo y trabajar con una versión anterior – Ejemplo 2

```
git log # busco el hash que me interesa
git checkout hash # viajamos a una versión anterior del proyecto
git branch -a  # lista las ramas y donde estamos
# aquí podemos compilar el software, probarlo, recuperar un archivo, ejemplo:
cp LEAME /tmp  # copio el archivo LEAME antiguo al directorio /tmp (lo traigo al presente)
git checkout master # volvemos a la rama master (presente del proyecto)
   o
git checkout -f master  # volvemos a master y eliminamos las modificaciones realizadas a archivos del pasado
```

Charla Introductoria a GIT

Git tips

```
git mv archivo nuevo_nombre_o_lugar  # mueve el archivo (o lo renombra)
git rm archivo # elimina un archivo del repositorio y fisicamente
git commit -m "quitamos tal y renombramos aquel" # Es necesario confirmar el cambio

# ATENCiÓN: si por un descuido movemos o borramos un archivo sin usar git habrá
#           que ejecutar en algun momento los comandos anteriores igualmente (puede quejarse,
#           habrá que preguntar a google como subsanar)

git checkout .   # elimina cambios en los archivos que pertenecen al repo
git reset        # quita de stage todo lo que se agregó con git add
git reset HEAD~1 # elimina el ultimo commit (no recomendado: pregunte por qué)

git tag -a v0.2 -m 'version alfa funcional'  # crea una etiqueta al commit actual
# Las etiquetas son utiles: cada vez que alguien quiera usar el software en un "momento funcional"
# puede simplemente ir a esa versión con git checkout v0.2

# NO almacenar en repositorios git archivos binarios grandes, o binarios que se modifican todo el tiempo
# (harán al historial del repositorio muy muy pesado: dificil de clonar por ejemplo).
```

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

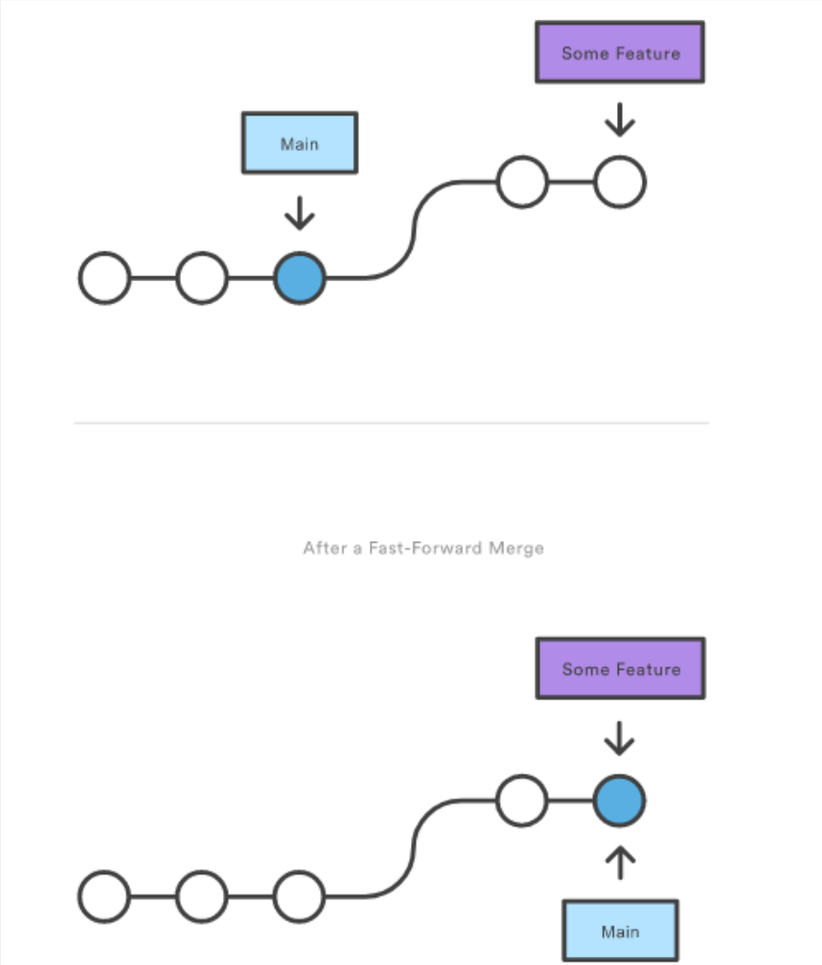
Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Ramas

Las ramas es probablemente la característica “mas potente y útil” de git.

- “livianas/ligeras”

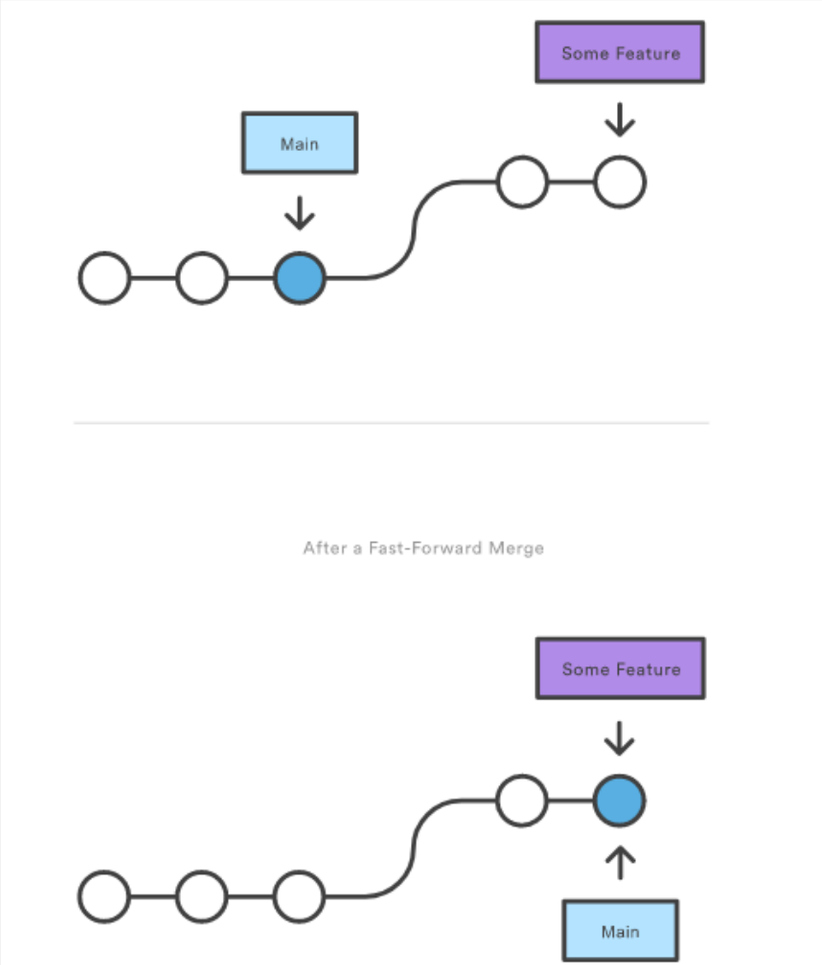


Charla Introductoria a GIT

Ramas

Las ramas es probablemente la característica “mas potente y útil” de git.

- “livianas/ligeras”
- Fáciles de usar

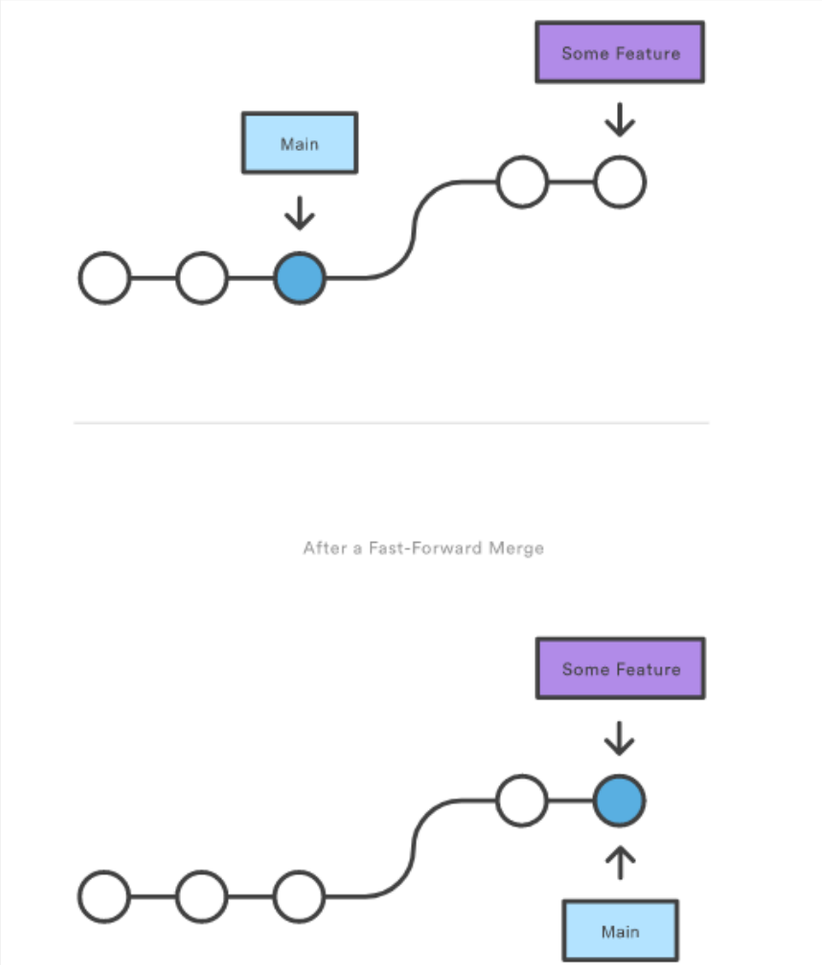


Charla Introductoria a GIT

Ramas

Las ramas es probablemente la característica “mas potente y útil” de git.

- “livianas/ligeras”
- Fáciles de usar
- Generalmente arrancan en master

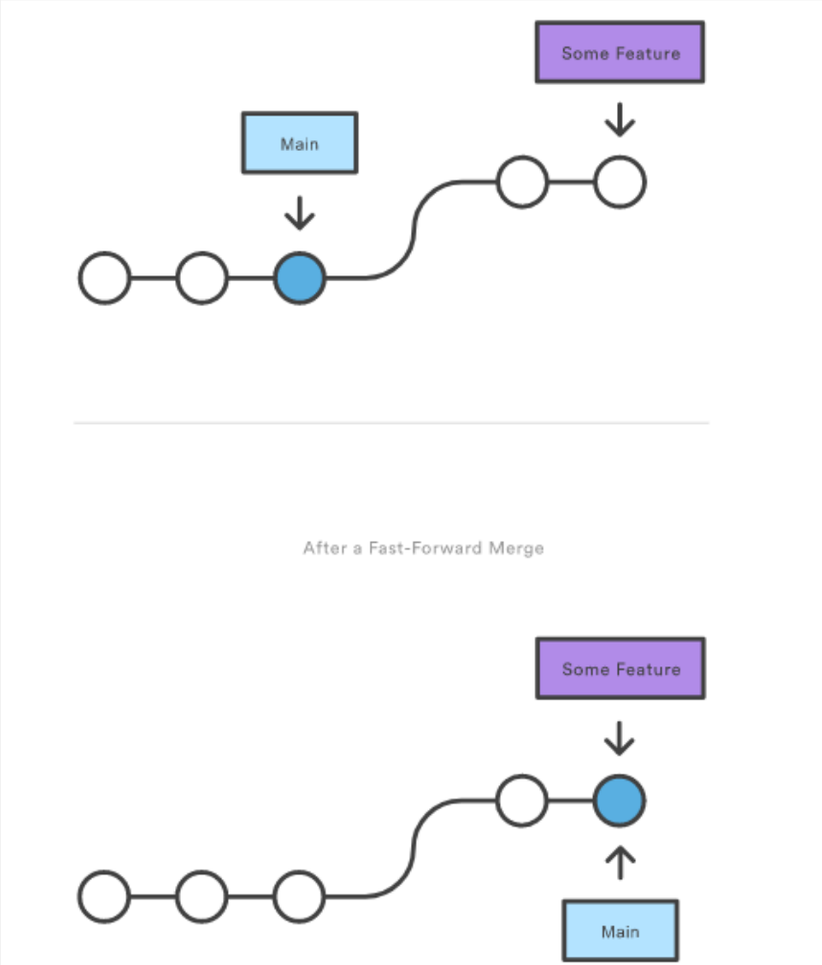


Charla Introductoria a GIT

Ramas

Las ramas es probablemente la característica “mas potente y útil” de git.

- “livianas/ligeras”
- Fáciles de usar
- Generalmente arrancan en master
- Probar ideas, corregir un bug, mejorar un código

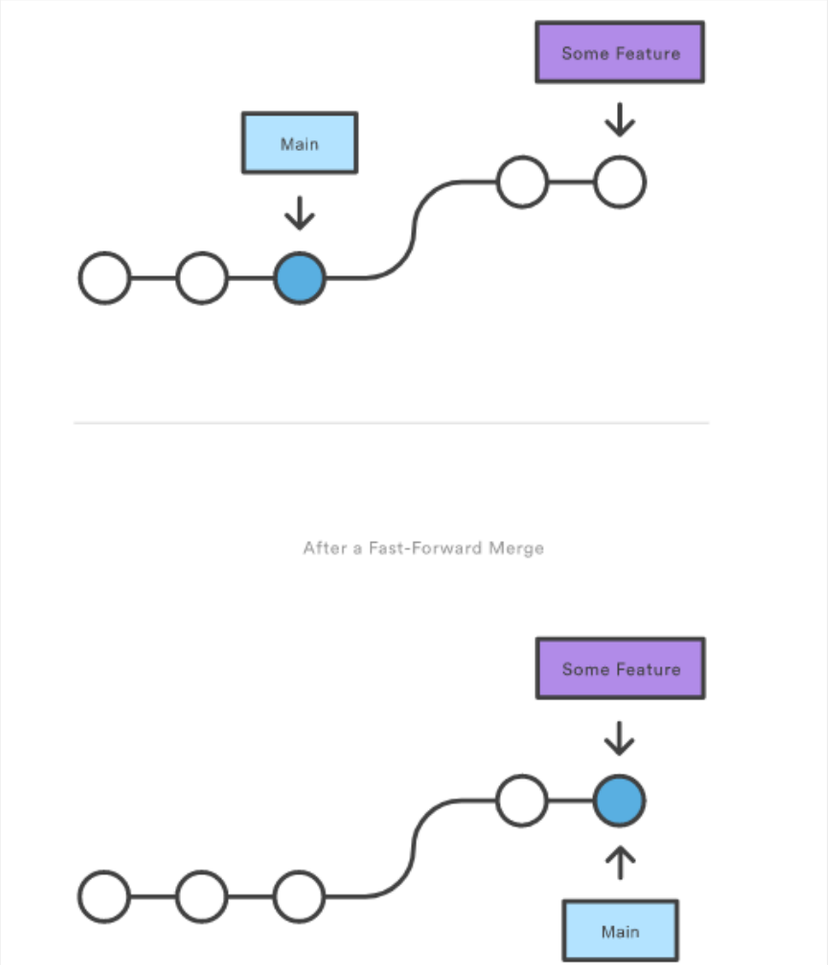


Charla Introductoria a GIT

Ramas

Las ramas es probablemente la característica “mas potente y útil” de git.

- “livianas/ligeras”
- Fáciles de usar
- Generalmente arrancan en master
- Probar ideas, corregir un bug, mejorar un código
- Se comparten facilmente con los demás desarrolladores



Charla Introductoria a GIT

Ramas

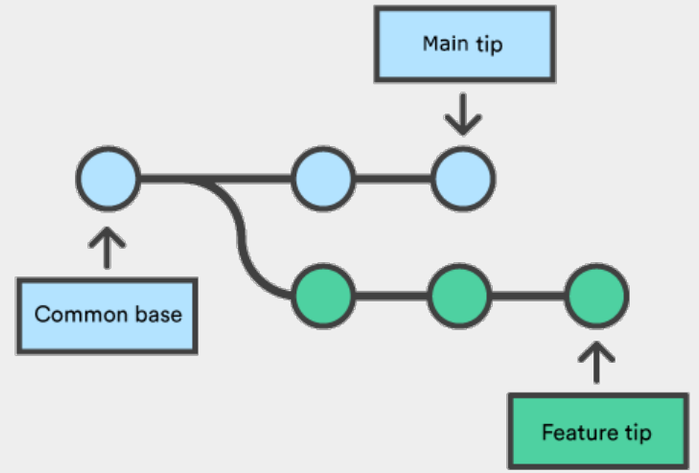
Las ramas es probablemente la característica “mas potente y útil” de git.

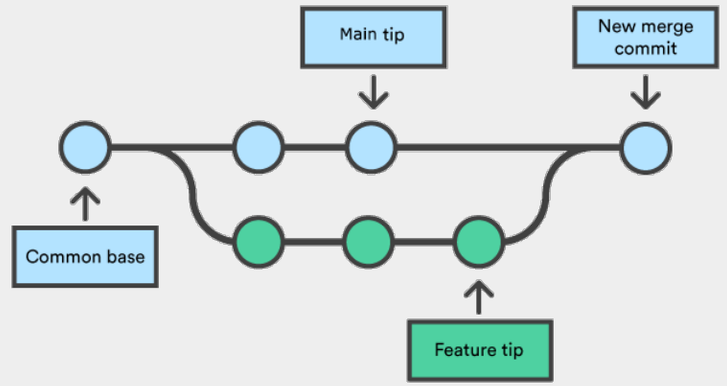
Generalmente se crea una rama a partir del codigo “estable” o “principal” (master)
y se desarrollan ahí ideas o correcciones al proyecto. Una rama por idea.

Cuando se logra una idea y parece util, o se soluciona algun problema que tenia el proyecto “estable”,
se puede compartir la rama con otros desarrolladores.

Las ramas se crean y se utilizan localmente, son “ligeras”.
En otros sistemas son complejas de crear y mantener, pero en git no.

Lamentablemente, para la gran mayoria de los proyectos en sitios publicos como
Github, gitlab, bitbucket, no se usan muy seguido. Hay una o mas razones.





Charla Introductoria a GIT

Ramas

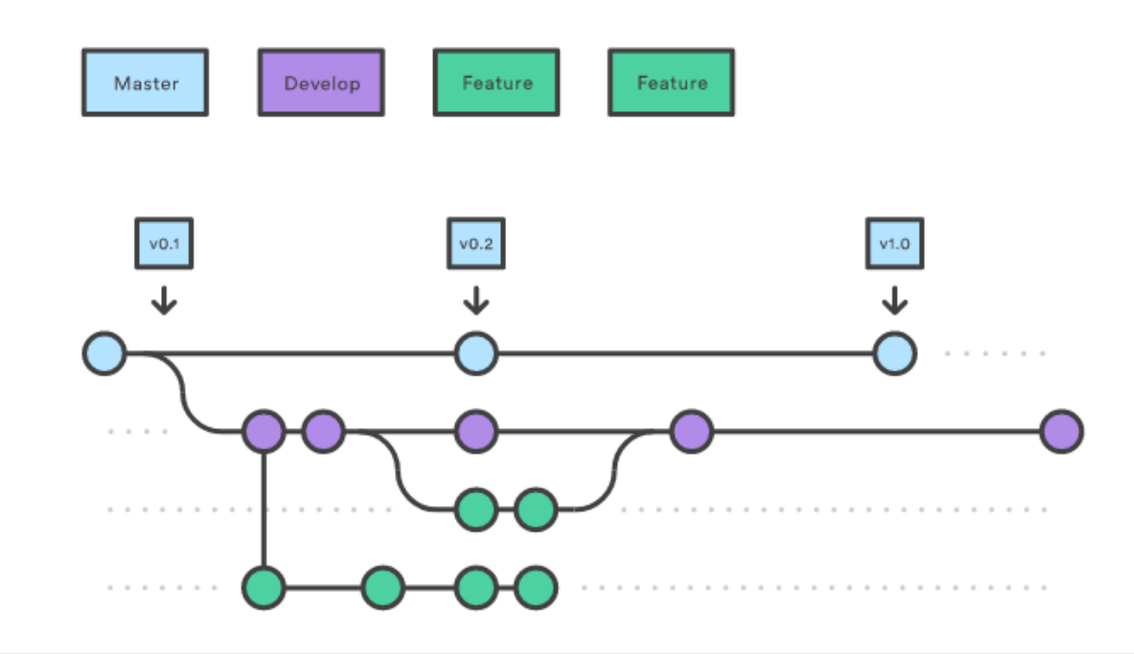
Las ramas es probablemente la característica “mas potente y útil” de git.

Generalmente se crea una rama a partir del codigo “estable” o “principal” (master)
y se desarrollan ahí ideas o correcciones al proyecto. Una rama por idea.

Cuando se logra una idea y parece util, o se soluciona algun problema que tenia el proyecto “est
se puede compartir la rama con otros desarrolladores.

Las ramas se crean y se utilizan localmente, son “ligeras”.
En otros sistemas son complejas de crear y mantener, pero en git no.

Lamentablemente, para la gran mayoria de los proyectos en sitios publicos como
Github, gitlab, bitbucket, no se usan muy seguido. Hay una o mas razones.



Charla Introductoria a GIT

Ramas

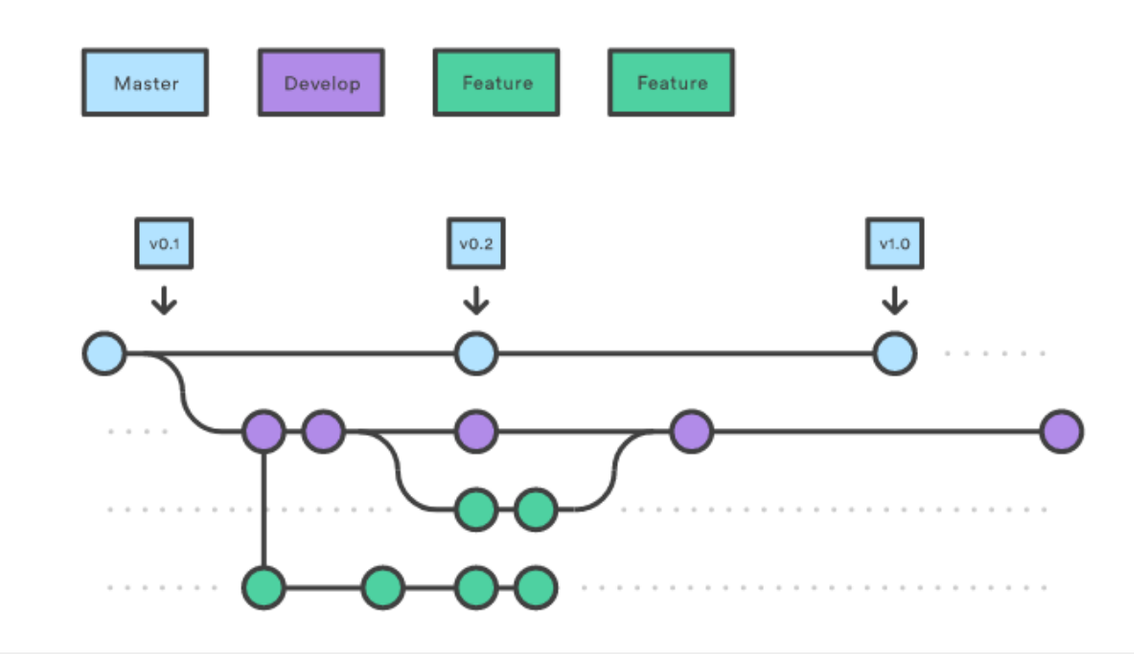
Las ramas es probablemente la característica “mas potente y útil” de git.

Generalmente se crea una rama a partir del codigo “estable” o “principal” (master)
y se desarrollan ahí ideas o correcciones al proyecto. Una rama por idea.

Cuando se logra una idea y parece util, o se soluciona algun problema que tenia el proyecto “est
se puede compartir la rama con otros desarrolladores.

Las ramas se crean y se utilizan localmente, son “ligeras”.
En otros sistemas son complejas de crear y mantener, pero en git no.

Lamentablemente, para la gran mayoria de los proyectos en sitios publicos como
Github, gitlab, bitbucket, no se usan muy seguido. Hay una o mas razones.

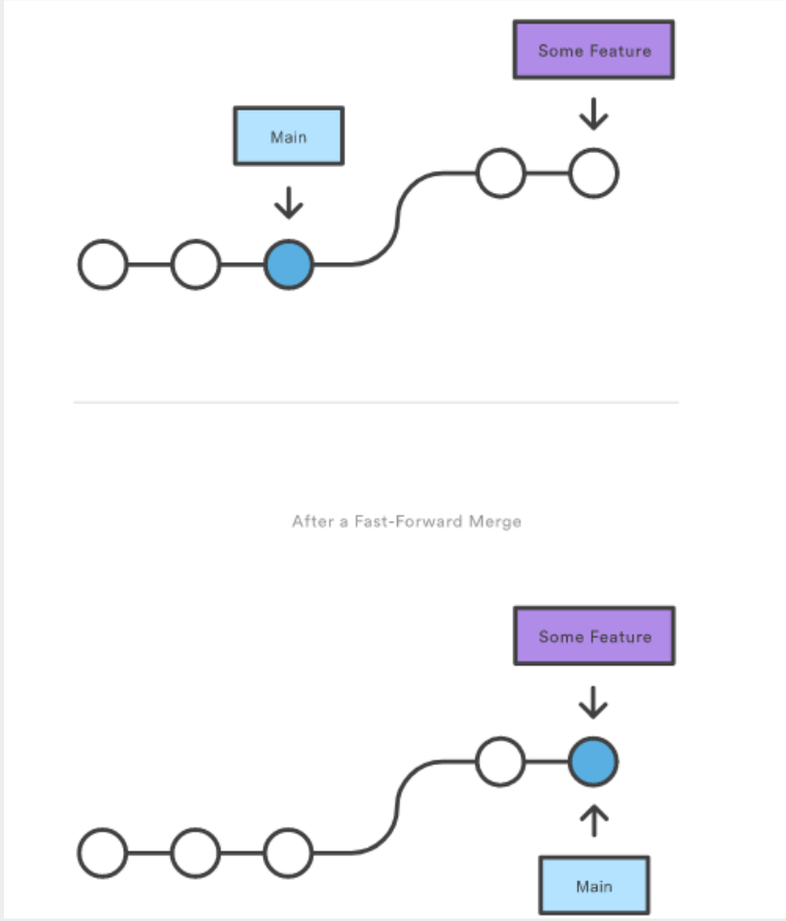


Charla Introductoria a GIT

Ramas – Ejemplo 3

- Creamos una rama,
- desarrollamos una idea,
- la integramos a master,
- borramos la rama.

```
cd repotest/
git checkout -b  nueva-idea master    # esto ejecuta git branch nueva-idea
                                      #              git checkout nueva-idea
git commit -m "…"     # o varios commits

# estamos contentos? Entonces fusionamos con master
git checkout master     # pregunte que pasa si en la rama anterior quedan cosas sueltas
git merge nueva-idea

# borramos la rama
git branch -d nueva-idea
```

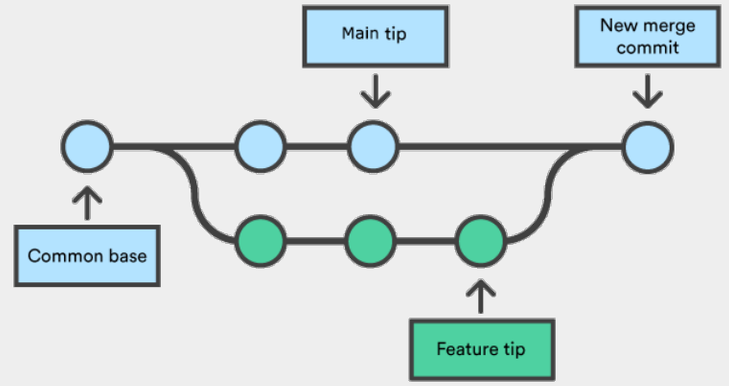


- Practica 3: crear dos ramas, crear un commit en cada una, fusionar con master. Hacer push al remoto.

Charla Introductoria a GIT

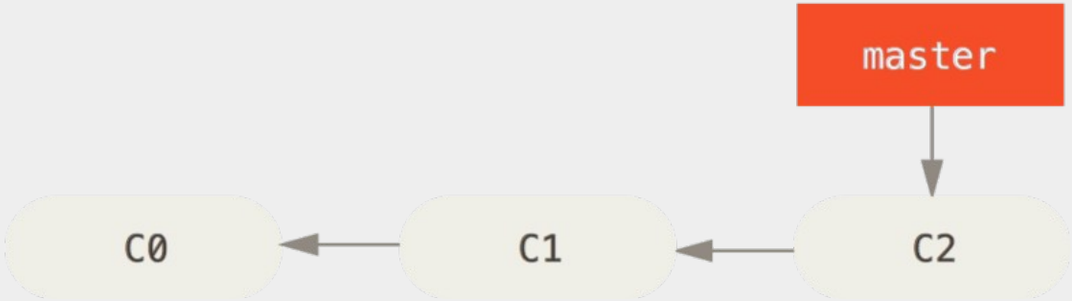
Ramas – Tip

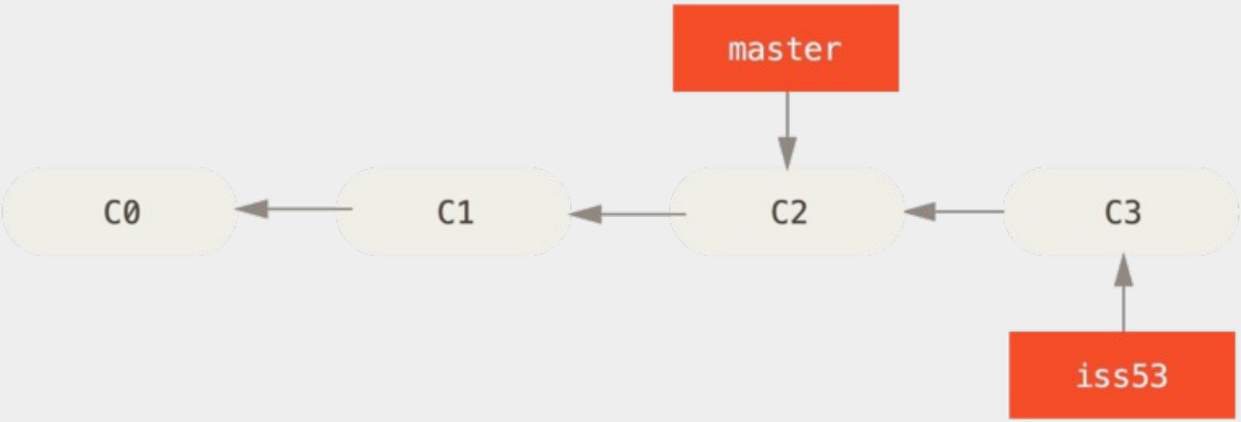
```
git log -graph --oneline
```

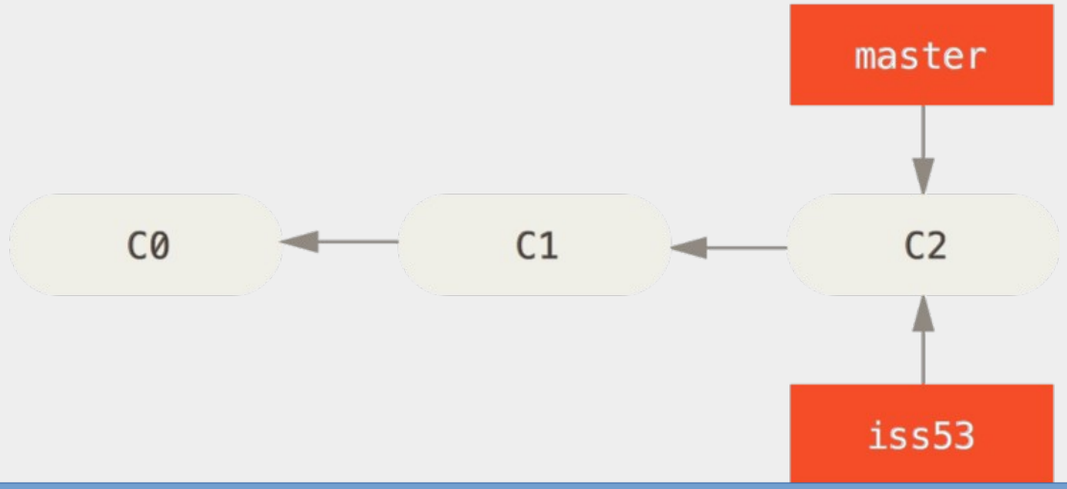


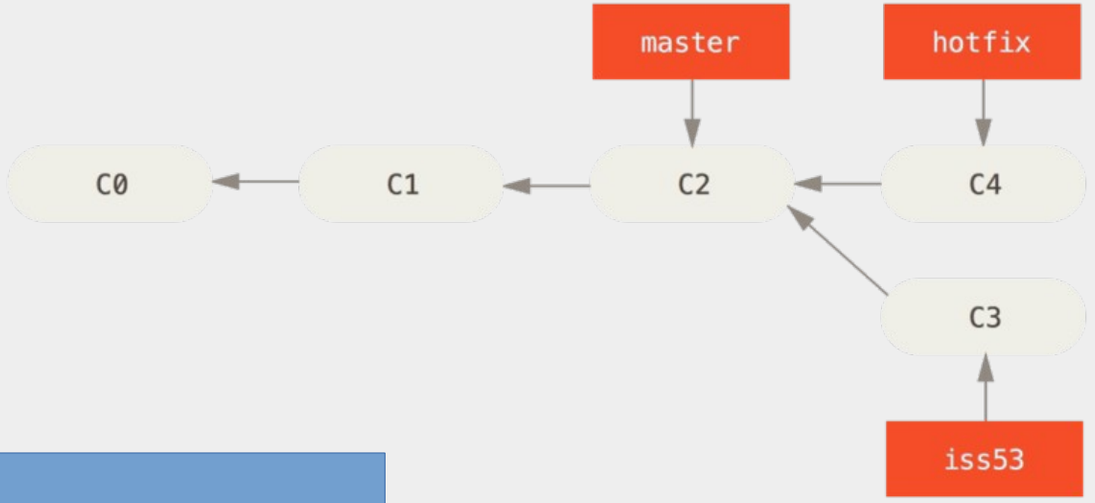
Charla Introductoria a GIT

Ramas : Git merge puede crear commits al proyecto al fusionar ramas



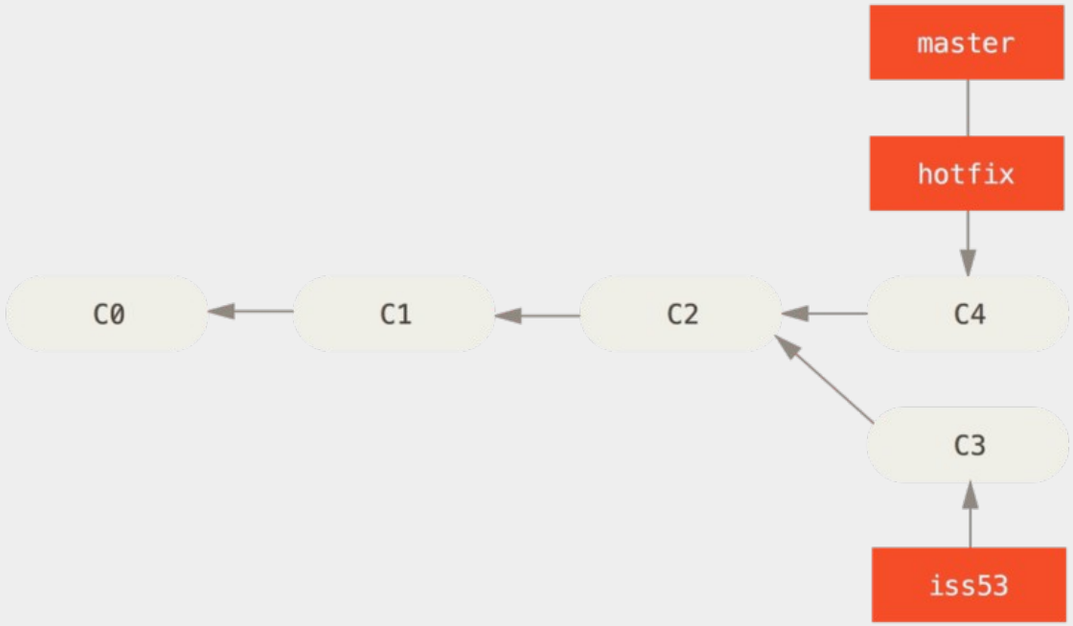


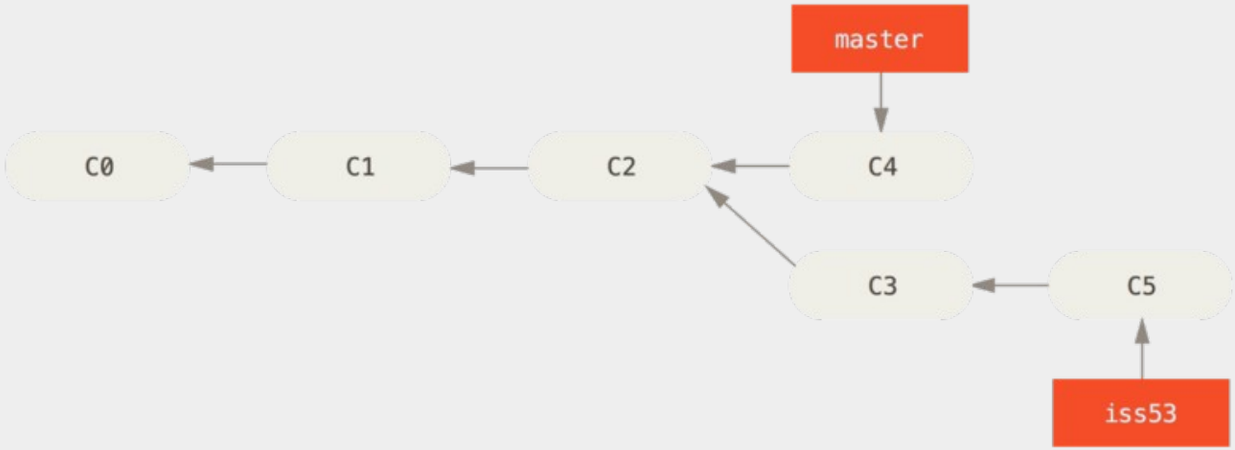


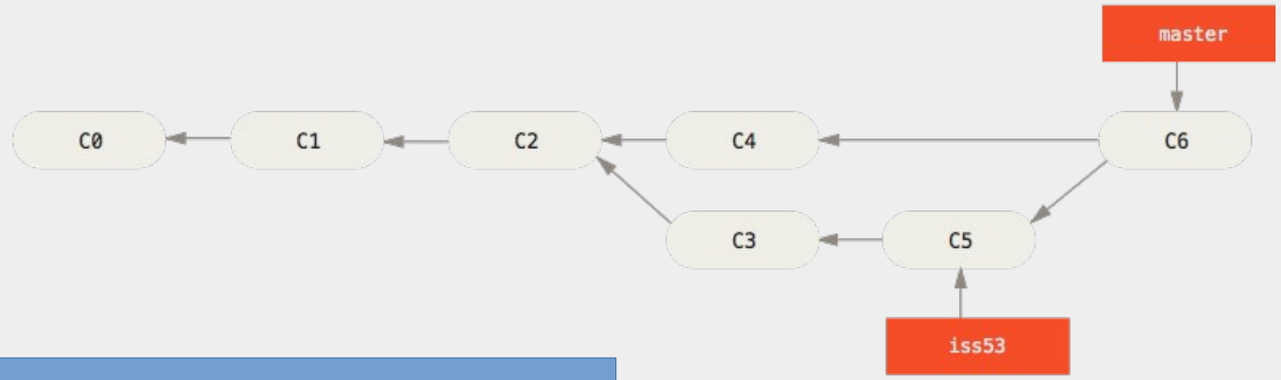


Charla Introductoria a GIT

Ramas : Git merge puede crear commits al proyecto al fusionar ramas

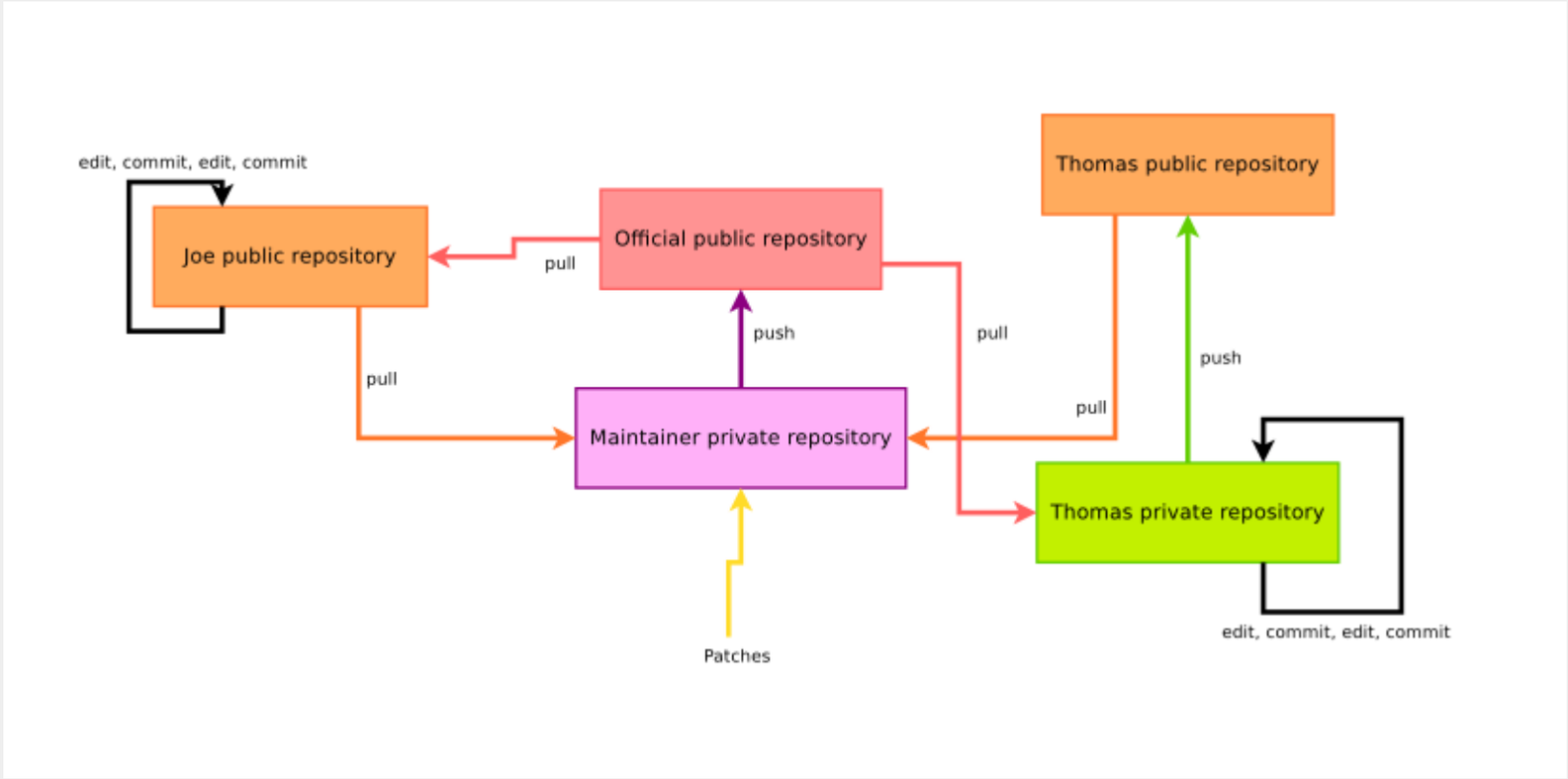






Charla Introductoria a GIT

Ramas: ejemplo
En este ejemplo creamos una rama, desarrollamos una caracteristica (commits), publicamos la rama publicamente.
Luego que el desarrollador apruebe (merge) los cambios en el proyecto principal borramos la rama publicamente.



Charla Introductoria a GIT

Contenido:

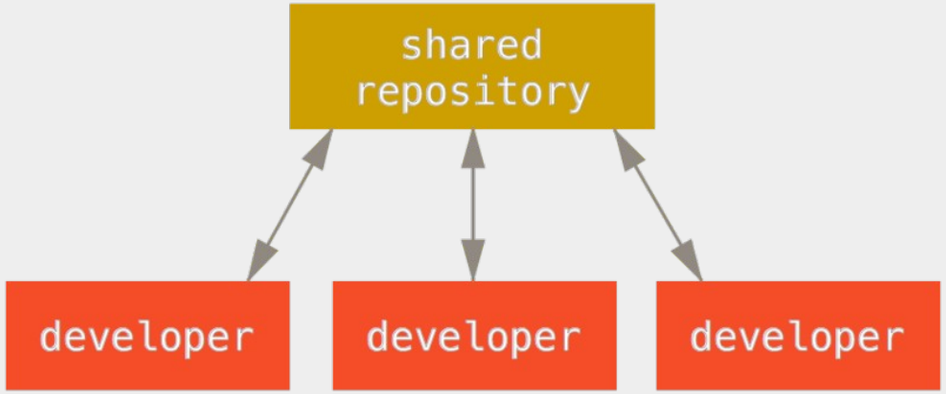
- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Flujos de trabajo

Varios desarrolladores, un unico repositorio, una unica rama

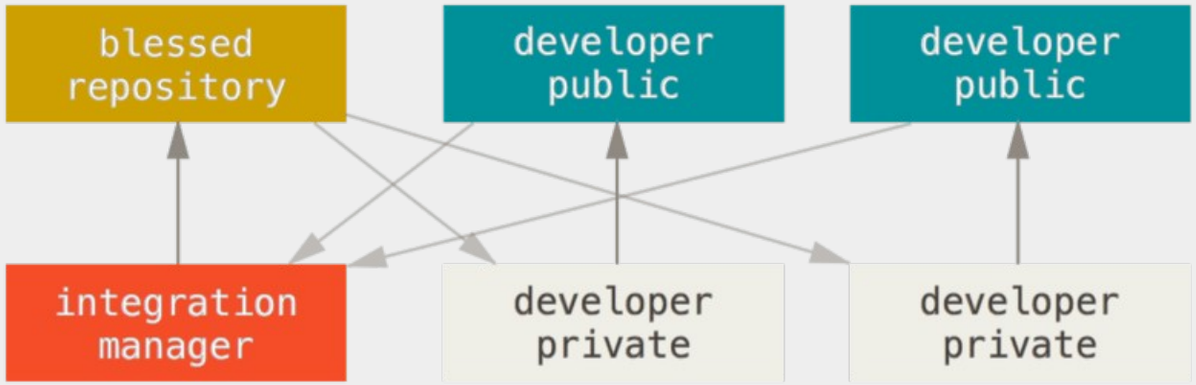


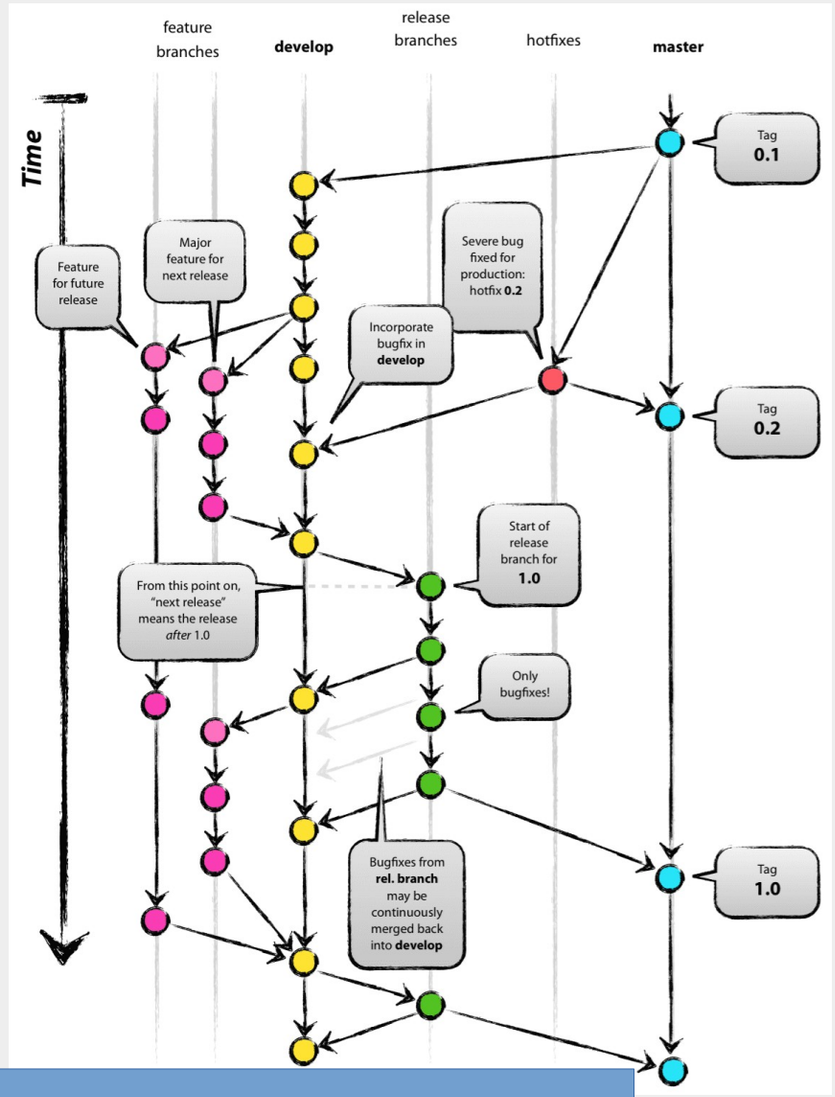
- Cómodo para cuando el proyecto es pequeño
- Git pull hará merges automaticamente
- Si hay conflictos no resolubles automaticamente se deben resolver manualmente
- No escala de manera ordenada (muchos conflictos)
- Para escalar: crear ramas, y uno o mas desarrolladores (pocos) se encargan unicamente de master

Charla Introductoria a GIT

Flujos de trabajo

Varios desarrolladores, cada uno con su repositorio publico

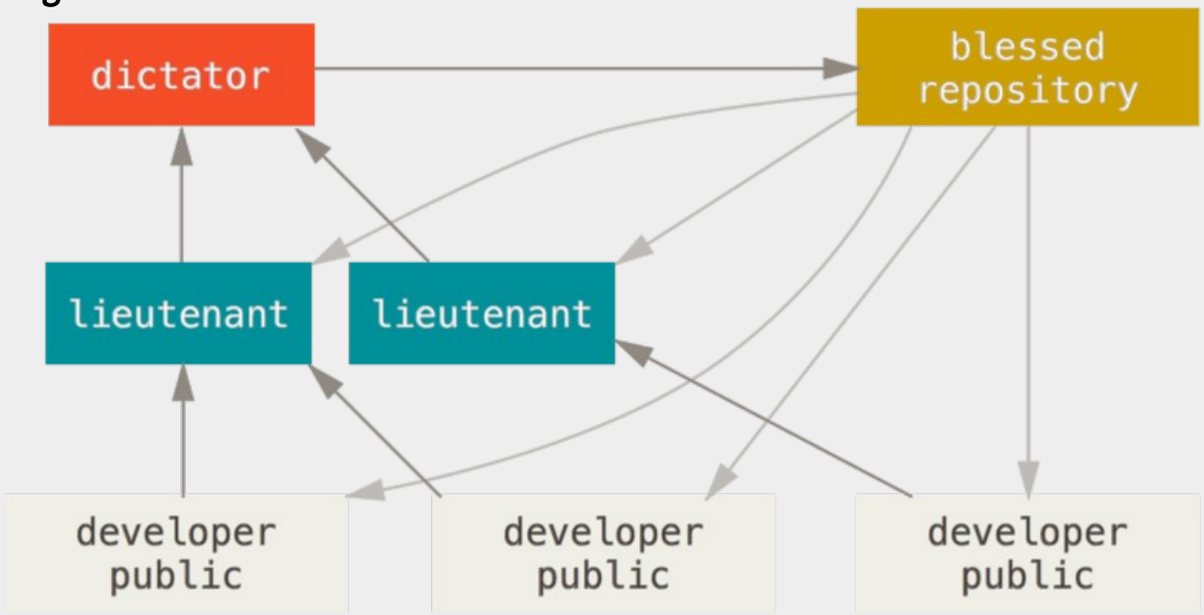


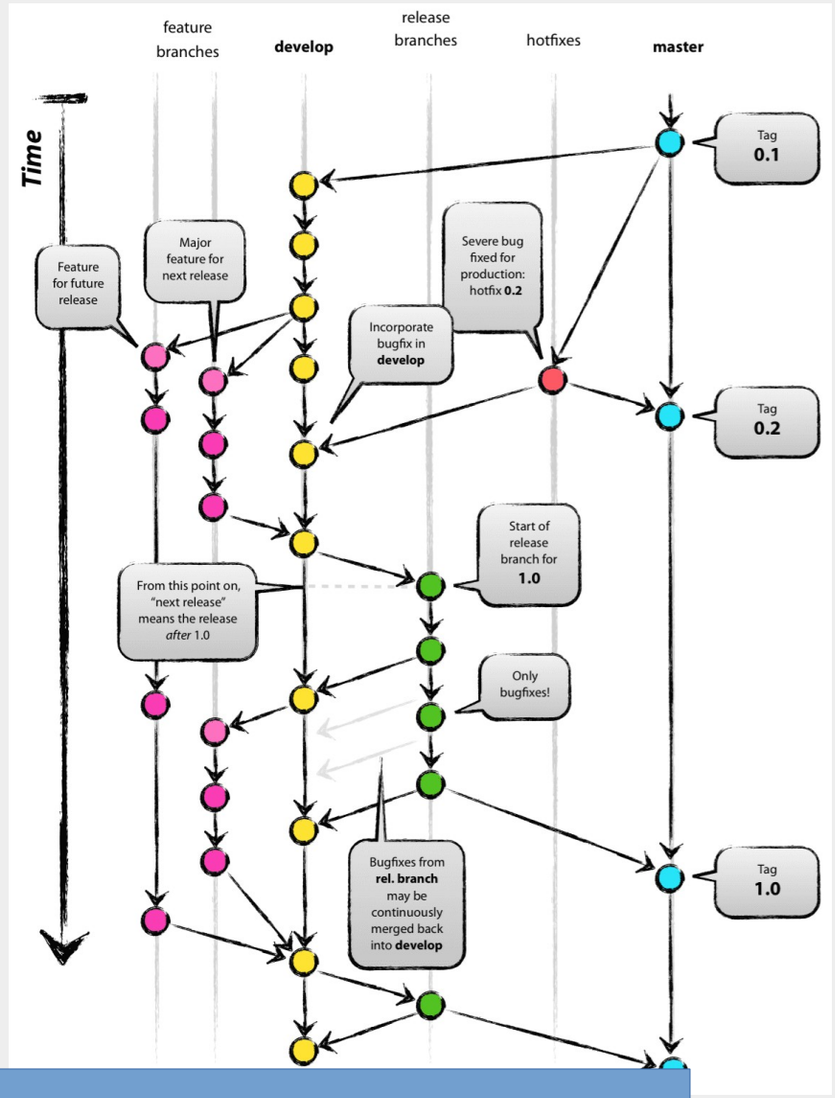


Charla Introductoria a GIT

Flujos de trabajo

Varios desarrolladores, los principales con repositorios publicos,
Los demás participan enviando parches por mail.
Hay una jerarquia que acepta, o no, a mayor nivel.
Organización en el kernel Linux



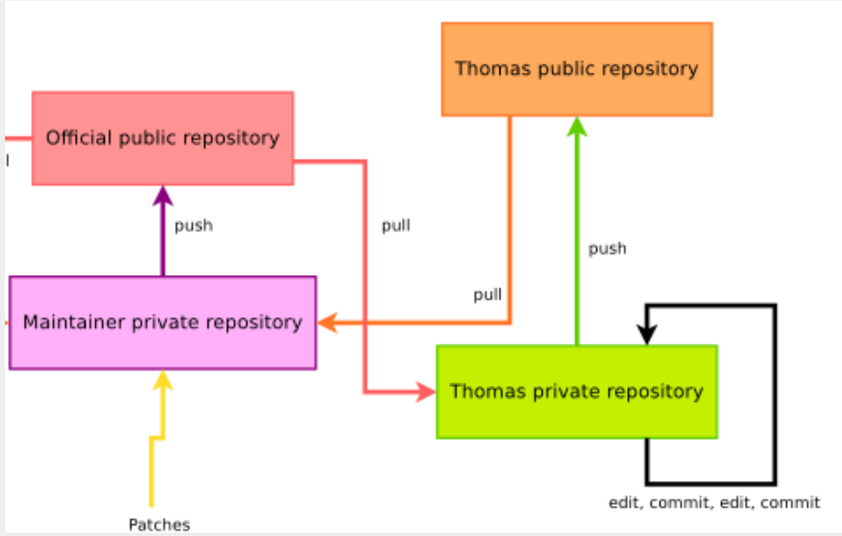


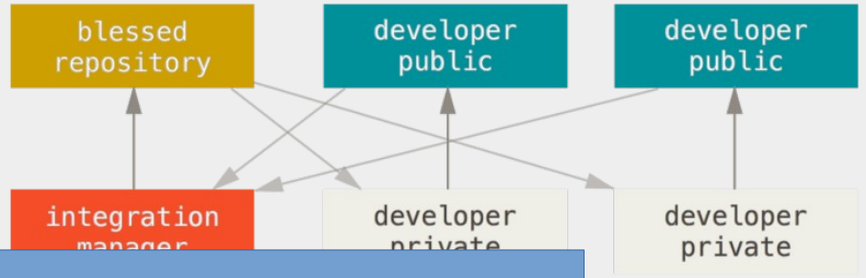
Charla Introductoria a GIT

Flujos de trabajo - Varios desarrolladores, cada uno con su repositorio publico – Ejemplo 4

Creamos una rama, desarrollamos una caracteristica (commits), publicamos la rama.
Luego que el desarrollador apruebe (merge) los cambios en el proyecto principal borramos la rama publicamente.

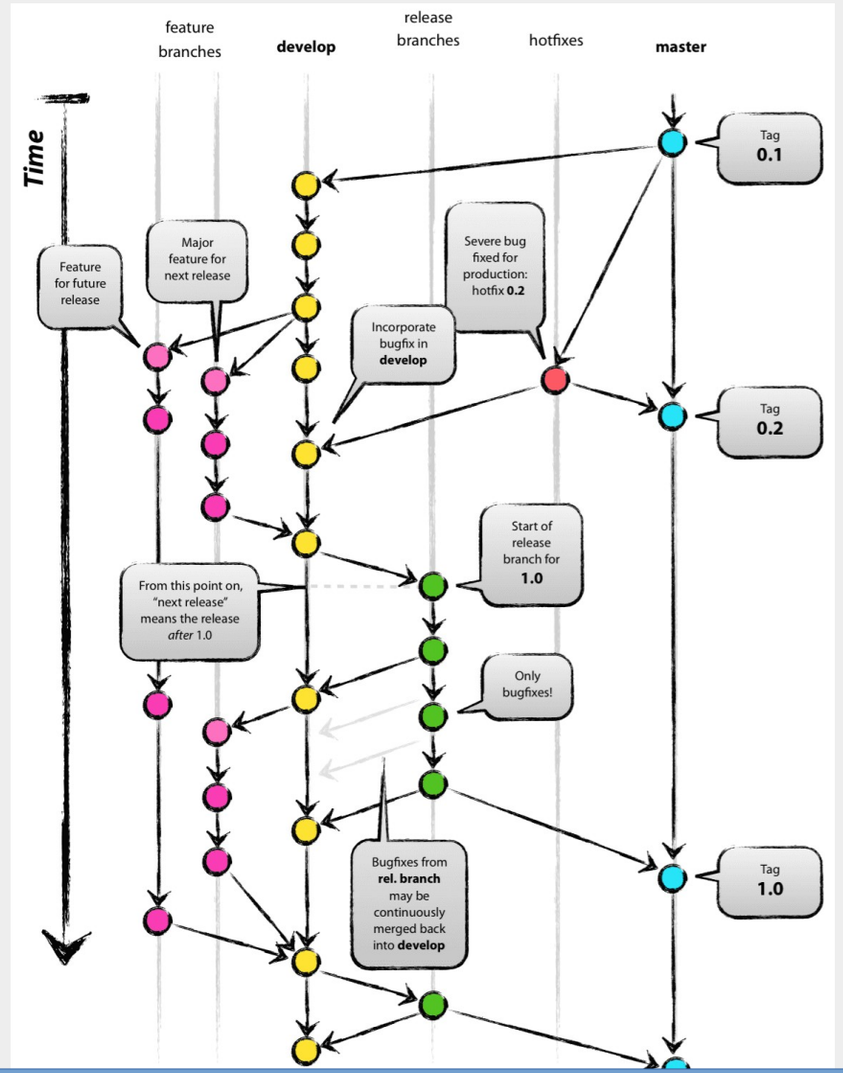
```
# donde dice user reemplace con su usuario remoto
git clone ssh://user@pse.fi.uncoma.edu.ar:/home/git/repotest.git

git clone --bare repotest/
# Se genera repotest.git ademas de repotest
# Enviamos repotest.git a nuestro home en un servidor publico
scp -r repotest.git/ user@pse.fi.uncoma.edu.ar:/home/user/
cd repotest/  # seguimos local
git checkout -b idea master
git add...; git commit -m...
git push user@pse.fi.uncoma.edu.ar:/home/user/repotest.git idea:ideapub
# creamos un pull request
git request-pull master ssh://pse.fi.uncoma.edu.ar:/home/user/repotest.git idea:ideapub
# enviamos el texto de salida por mail al desarrollador principal

# El desarrollador principal crea una rama en su pc:
git checkout -b idea-user master
git pull user@pse.fi.uncoma.edu.ar:/home/user/repotest.git ideapub
git checkout master
git merge idea-user
git push
```





Charla Introductoria a GIT

Flujos de trabajo



Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online
- Interfaz web de gestión. Aplicaciones móviles

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online
- Interfaz web de gestión. Aplicaciones móviles
- Permisos y seguridad sencillo

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online
- Interfaz web de gestión. Aplicaciones móviles
- Permisos y seguridad sencillo
- Se trabaja localmente con git como visto en la charla

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online
- Interfaz web de gestión. Aplicaciones móviles
- Permisos y seguridad sencillo
- Se trabaja localmente con git como visto en la charla
- Gestion de “problemas” (issues), wiki para documentación, gestión de proyectos

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online
- Interfaz web de gestión. Aplicaciones móviles
- Permisos y seguridad sencillo
- Se trabaja localmente con git como visto en la charla
- Gestion de “problemas” (issues), wiki para documentación, gestión de proyectos
- Interfaces visuales del avance del código

Charla Introductoria a GIT

Github, gitlab, bitbucket

- Repositorio online
- Interfaz web de gestión. Aplicaciones móviles
- Permisos y seguridad sencillo
- Se trabaja localmente con git como visto en la charla
- Gestion de “problemas” (issues), wiki para documentación, gestión de proyectos
- Interfaces visuales del avance del código
- Util para divulgar proyectos

Charla Introductoria a GIT

Github, gitlab, bitbucket

Estos servicios web gratuitos permiten tener un "**repositorio online**". Se puede pagar por un servicio empresarial.
Util cuando no se trabaja con máquinas publicamente visibles en internet

Los proyectos pueden crearse, borrarse, desde la **interfaz web de gestión. Aplicaciones móviles** para gestión.

Manejo de los **permisos y seguridad** (colaboradores, etc) sencillo. En un servidor privado se debe administrar el sistemas de archivos y usuarios manualmente.

Luego de configurar el proyecto, y dar visibilidad, los integrantes pueden **trabajar localmente con git como visto en la charla**

Proveen **gestion de "problemas" (issues)**, con foros de discusión, wiki para documentacion con markdown, varias cosas utiles mas.

Los managers "gustan" las **interfaces visuales** del avance de los proyectos (git log es para nerds)

Utiles como plataforma de **divulgación de proyectos**: se puede llegar a una gran cantidad de usuarios y desarrolladores

Charla Introductoria a GIT

Contenido:

- Historia
- Caracteristicas – Instalación - Configuración
- Iniciando repositorio - commits
- Delorean – historial
- Ramas
- Flujo de trabajo
- Github – gitlab – bitbucket
- Caracteristicas Avanzadas

Ejemplo 1 – practica 1
Ejemplo 2 – practica 2
Ejemplo 3 – practica 3
Ejemplo 4 - practica 4
Ejemlpo 5 - practica 5

Charla Introductoria a GIT

Características avanzadas: Busqueda binaria de bugs. (regresión)

Uno detecta un bug, e intuye que está ahí desde hace mucho tiempo (tal vez miles de commits).

Se desea enontrar el commit que introdujo el bug, y por supuesto, conocer cuál es el bug.

Se realiza un viaje a un commit antiguo, hasta un punto en el historial del proyecto donde al probar el software, funciona bien (el bug no está presente).

Se le indica a git que ese commit es “bueno”, y el ultimo commit del proyecto es “malo”.

Git encuentra entonces el commit que está a la mitad del historial entre el commit “bueno” y “malo”.

Se comprueba si el software funciona bien en ese commit (el bug no está presente).
- Si el software no tiene el bug, se marca ese commit como “bueno”.
- Si el software tiene el bug, se marca el bug como “malo”

```
git bisect start
git bisect good commit-hash
git bisect bad commit-hash
```

Git vuelve a realizar una busqueda binaria (commit del medio en el historial, entre el commit “bueno” y “malo”).

Esta busqueda se repite hasta aislar el commit que introdujo el bug. La busqueda se hace rapidamente ya que la busqueda es binaria.

Charla Introductoria a GIT

Algunas referencias útiles:

- apropos git – man git-subcomando
- Libro Pro Git. Autores: Scott Chacon, Ben Straub, Apress. Disponible gratuitamente (creative commons) aquí:
  https://git-scm.com/book/es/v2
- Arrancando git en gitlab: https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html
- Crear un repo en github: https://docs.github.com/en/get-started/quickstart/create-a-repo
- 4 pasos para empezar con git y bitbucket:
  https://bitbucket.org/product/es/guides/basics/four-starting-steps#step-1-put-your-code-in-bitbucket