Git. Repositorios locales

PGPI
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan <jjchico@dte.us.es>

Introducción a Git

- Referencia: Pro Git capítulo 2
- Contenidos
  - Instalación
  - Configuración básica de git
  - Crear un repositorio (repo)
  - Añadir archivos a seguimiento
  - Actualizar repo (commit)
  - Estado e historia
  - Ramas: crear, cambiar, mezclar
  - Etiquetas

Introducción a Git

- Linus Torvalds empezó en 2002 a utilizar un DVCS propietario llamado Bitkeeper para controlar el kernel de Linux
- En 2005 rompieron su relación y, como la alternativa (CVS) no le complacía, decidieron crear la suya propia con lo que habían aprendido como usuarios.
- Buscaban estos objetivos:
  - Velocidad y diseño sencillo
  - Soporte desarrollo no lineal (miles de ramas)
  - Completamente distribuido
  - Manejo eficiente de grandes proyectos de software

Instalación de Git

- Linux. Disponible en todas las distribuciones
  - Ej. **Ubuntu**: git, gitg (opcional pero recomendada), ...
    $ sudo apt-get install git gitg
- Si la versión de tu Ubuntu está muy desactualizada puedes añadir el repositorio actualizado oficial e instalar el versión más actual:

  $ sudo add-apt-repository ppa:git-core/ppa
  $ sudo apt-get update
  $ sudo apt-get install git
- Windows
  - https://gitforwindows.org/
- Otros
  - http://git-scm.com/download

Configuración básica

```
$ git config --global user.name "Jorge Juan"
$ git config --global user.email jjchico@gmail.com
$ git config --global push.default simple

$ git config --list
user.name=Jorge Juan
user.email=jjchico@gmail.com
push.default=simple
```

```
$ git config --local [...]
```

Ayuda

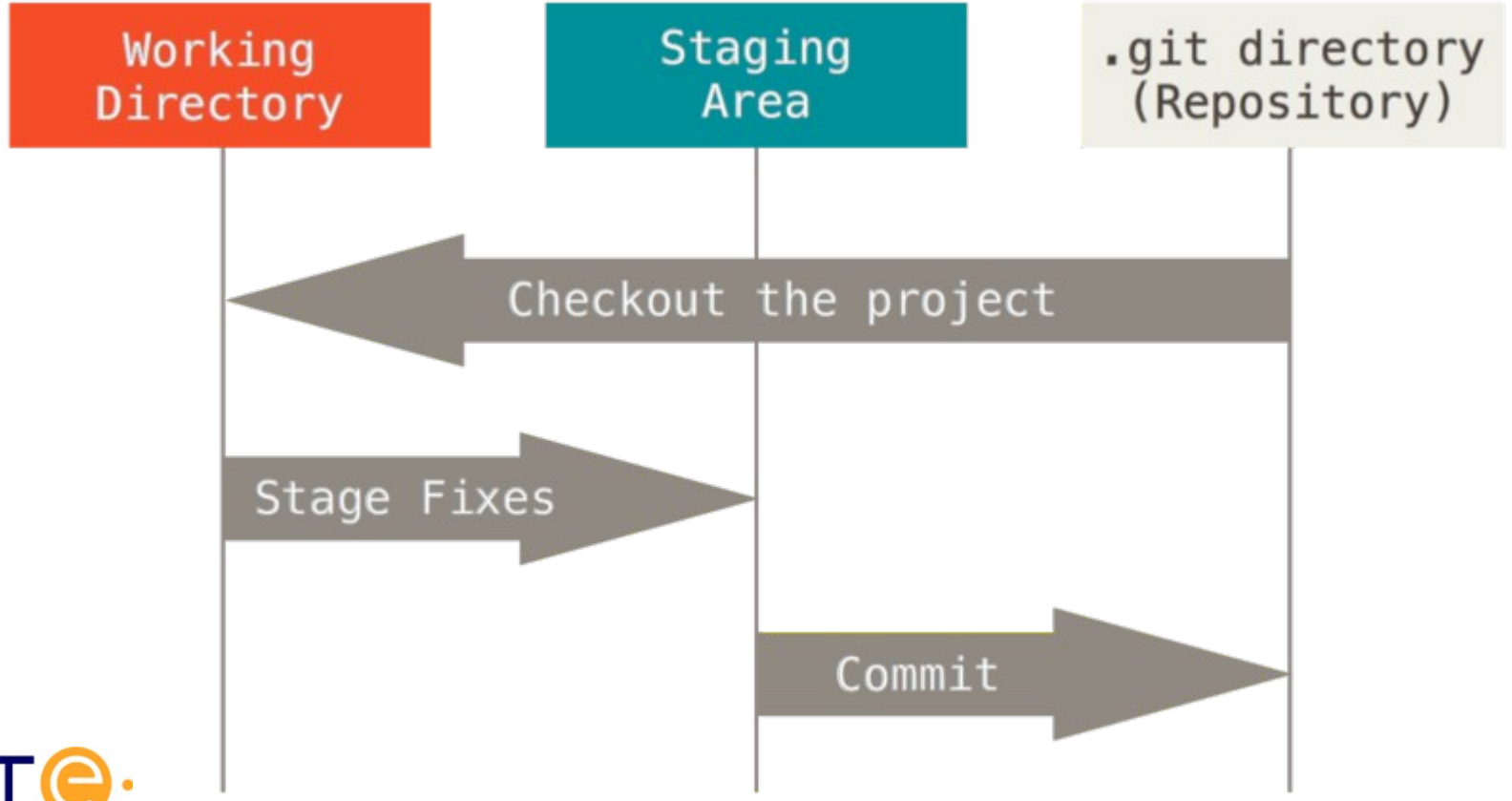
```
# Ayuda general y páginas de ayuda
$ git help
...

# Ayuda sobre funciones específicas
$ git help commit
...
$ git help log
...

# Listado de guías disponible
$ git help -g
...

# Tutorial
$ git help tutorial
...
```

Crear repositorio (repo)

```
$ mkdir proyecto
$ cd proyecto
$ git init
Initialized empty Git repository in /home/jjchico/git/proyecto/.git/
```

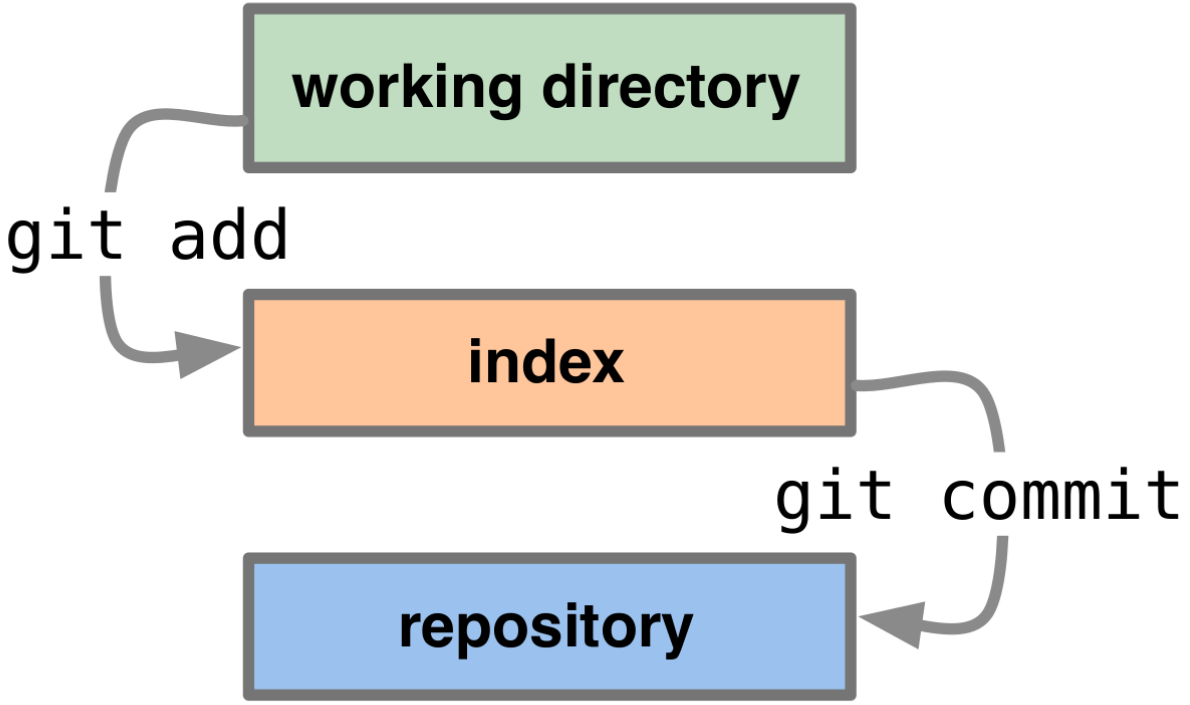
Flujo de trabajo en Git (1)

- Git mantiene 3 secciones en un proyecto:
  - El directorio de trabajo, dónde podemos modificar ficheros.
  - Área de preparación o índice, dónde se prepara la imagen completa antes de ser confirmada.
  - Repositorio, dónde se confirman los cambios preparados.



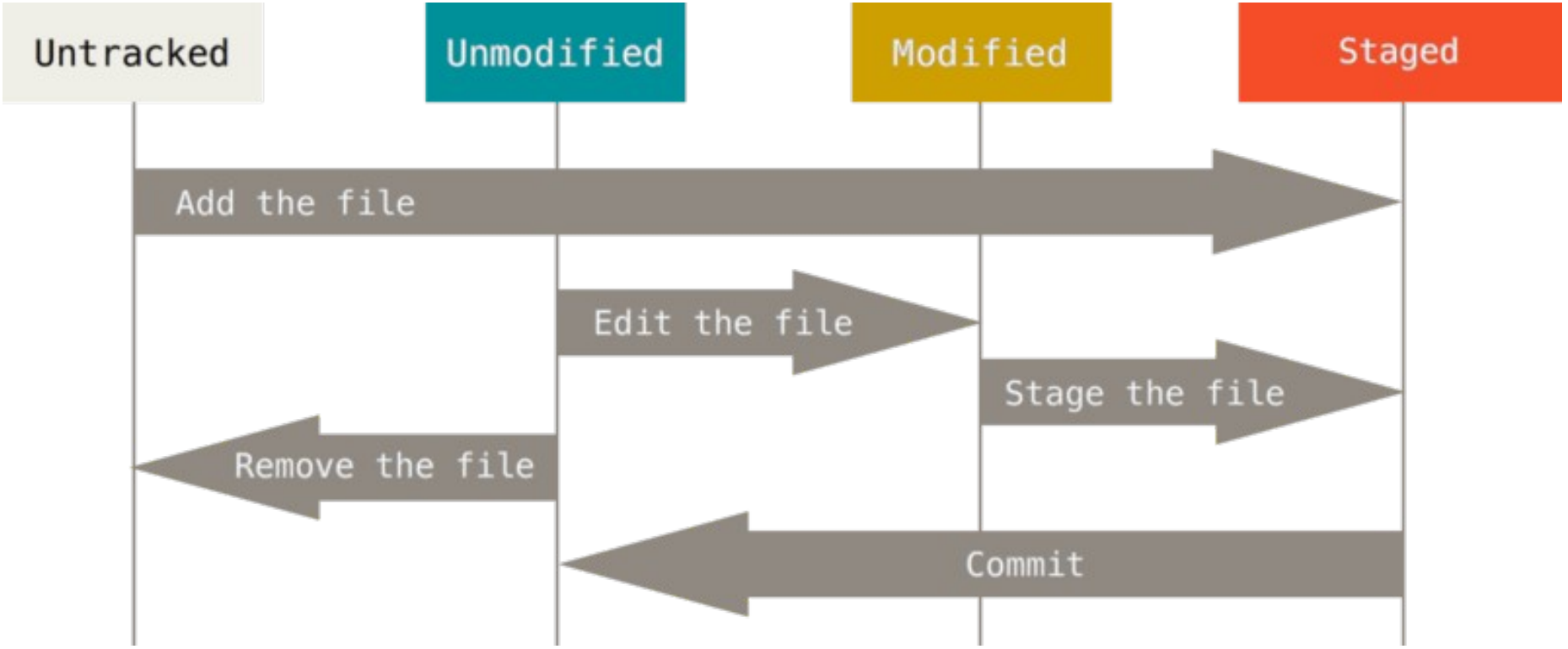
Flujo de trabajo en Git (y 2)

- Cada vez que se realizan cambias en un fichero hay que añadirlo al índice (área de preparación) con “git add”.
- Cuando todos los cambios están preparados toca hacer el “git commit” para registrar una revisión nueva.



Estado de los archivos

- “git status” muestra el estado de los archivos modificados y eliminados, es interesante hacerlo antes de confirmar.
- “git rm” y “git mv” permiten borrar y renombrar ficheros.



Confirmaciones (*commit*) (1)

```
$ vi lista.txt       # nano lista.txt, gedit lista.txt &
...

$ git add lista.txt
```

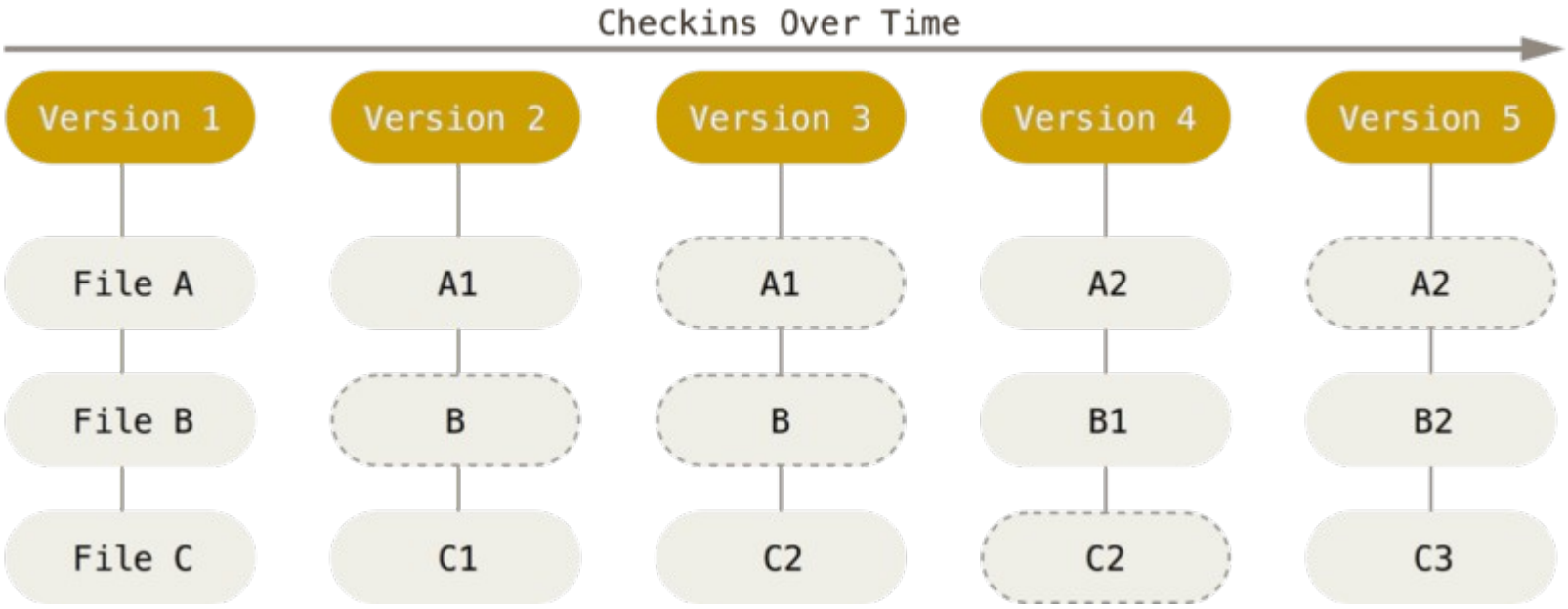
```
$ git commit     # interactivo
...

$ git commit -m "mensaje"

$ git commit -a -m "mensaje"  # añade archivos modificados al commit
```

Confirmaciones (*commit*) (2)

- ¿Qué?
  - Un conjunto de cambios en uno o más archivos del proyecto.
- ¿Qué incluir?
  - Cambios relativos a una funcionalidad dada.
  - Independientes de otros cambios.
  - ¿Se puede deshacer fácilmente?
- ¿Cuándo?
  - En algún hito del desarrollo, parcial o definitivo.
- ¿Dónde?
  - En la rama principal.
  - En una rama de desarrollo.
  - En una rama de mantenimiento (solución de errores).

Confirmaciones (*commit*) (3)

- Git usa un mecanismo de copias instantáneas, de forma que en cada confirmación (*commit*) mantiene un lista completa de archivos cómo están en ese momento.
- Los ficheros iguales son simples enlaces al original.
- Hace que sea muy eficiente en sus operaciones.
- Otros CVS guardan deltas de cada fichero modificado.



Archivo .gitignore

- Define patrones de nombres de rutas (archivos o carpetas) ignoradas por Git.
- Evita los mensajes del tipo “archivo sin seguimiento”.
- Casos útiles:
  - Archivos temporales
  - Código objeto o generado
  - Copias de seguridad de editores
  - Datos locales
- El archivo “.gitignore” se añade al repositorio y se hace seguimiento como cualquier otro archivo del proyecto.

```
# Archivo .gitignore
*~
*.o
*.tmp
local/*
```

Estado e histórico

```
$ git status          # estado de carpeta de trabaja e índice
...

$ git log             # registro de revisiones (en la rama actual)
...

$ git log --all       # registro de revisiones (en todas las ramas)
...

$ git log -p          # añade conjuntos de cambios (patch)
...

$ git show <rev>      # detalles de una revisión en particular
...

$ git log --pretty --graph
...

$ git log --all --decorate --graph --pretty=oneline
...

$ git log <rev1>..<rev2>      # Cambios en <rev2> que no están en <rev1>
...

$ git reflog          # registro de acciones realizadas
...
```

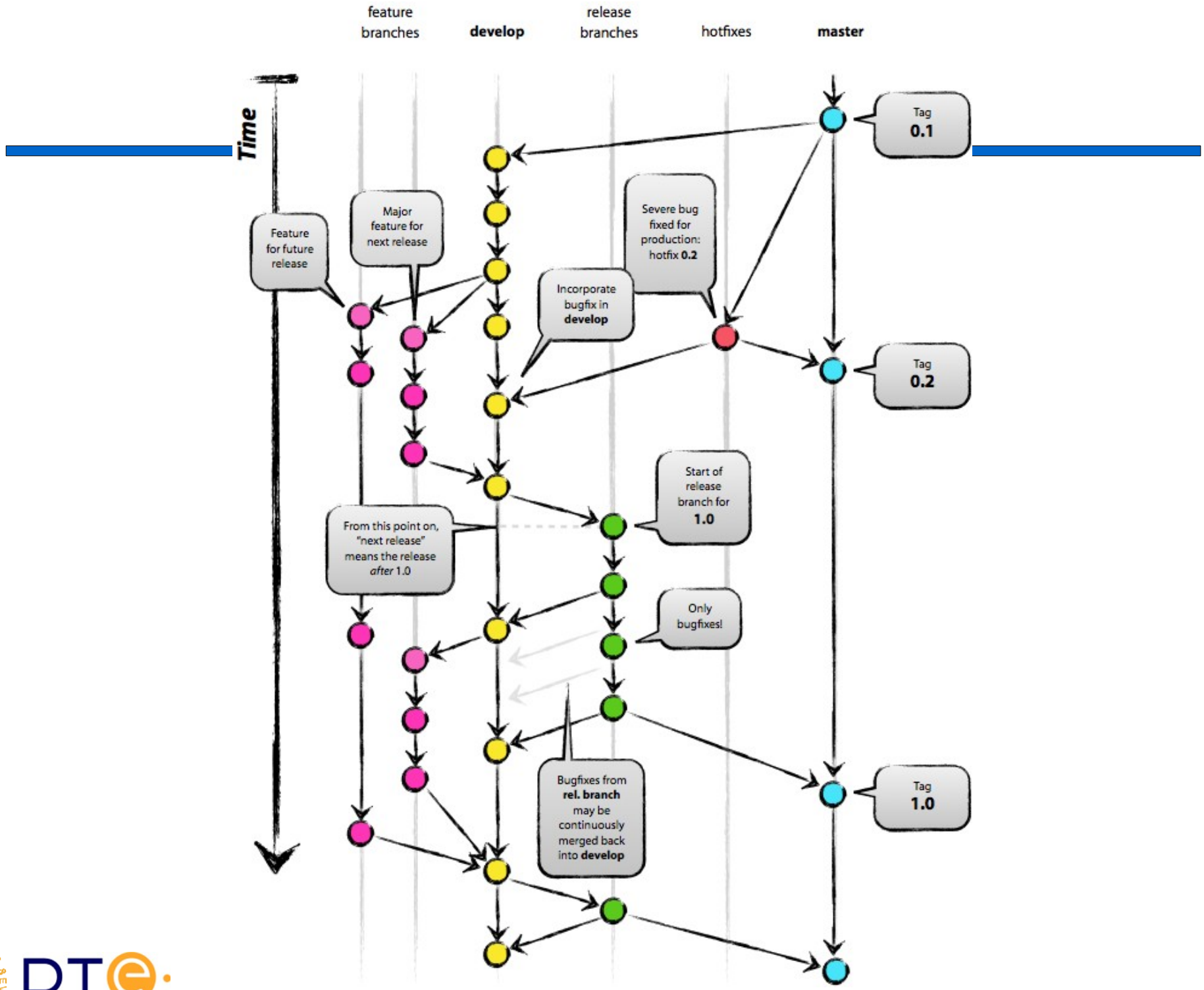
Etiquetas

- Se aplican a revisiones concretas
- Marcan hitos en el desarrollo y facilitan recuperar el estado del desarrollo en ese momento:
  - Versiones públicas, propuestas

```
$ git tag issue99 -m "Resuelto problema issue99"   # etiqueta simple

$ git tag -a meeting12 -m "Reunión 2012"       # etiqueta anotada

$ git tag -s meeting12 -m "Reunión 2012"       # etiqueta firmada

$ git tag -v meeting12                         # comprobar firma

$ git tag                          # lista todas las etiquetas

$ git tag -l 'meeting*'            # lista etiquetas por patrón

$ git log --decorate               # incluye nombres de etiquetas
```

Ramas

- Líneas de desarrollo independientes
- Git es muy eficiente manejando ramas
- Facilitan implementar modelos de desarrollo
- Operaciones
  - Crear ramas (branch)
  - Cambiar a una rama (checkout)
  - Unir ramas (merge+commit)
  - “Trasplantar” ramas (rebase)


feature branches
develop
release branches
hotfixes
master
Time
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.1
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Modelo de desarrollo simple

- Rama principal: “master”
  - Es funcional: compila, pasa los bancos de prueba, etc.
  - No contiene errores conocidos
- Rama de desarrollo: “dev”
  - Contiene modificaciones en curso.
  - Cada confirmación añade una funcionalidad concreta, parcial o completamente.
  - Cuando se alcanza un hito y el código es correcto, se fusiona en “master”, posiblemente añadiendo un número de versión.
- Ramas de correcciones menores: “bugX”
  - Derivan de “master” para corregir errores o añadir pequeña funcionalidad.
  - Se fusionan con “master”, posiblemente añadiendo un número de versión menor.
  - Se borran una vez fusionadas.

Ramas. Crear

```
$ git branch issue99     # creamos nueva rama
$ git branch             # listamos ramas
  issue99
* master

$ git checkout issue99   # cambiamos a la nueva rama

# editar algunos archivos

$ git commit -a -m "nueva rama issue99"

$ git checkout master    # volvemos a la rama master

# editar algunos archivos

$ git commit -a -m "algunos cambios adicionales en master"
```

Ramas. Unir

- Al unir dos ramas los archivos incorporan los cambios realizados en ambas ramas.
- Git intenta “filtrar” cambios repetidos: si hacemos el mismo cambio en dos ramas, sólo aparecerá una vez en la unión.
- Cambios diferentes sobre las mismas líneas producen conflictos
  - Basta editar los archivos con conflictos y confirmar los cambios como en una confirmación normal.
- El resultado de la mezcla puede ser:
  - La combinación de dos ramas
  - El avance de la rama más retrasada (fast-forward)
    - Puede obligarse la combinación de las ramas.

Ramas. Unir

- Al unir dos ramas los archivos incorporan los cambios realizados en ambas ramas.
- Git intenta “filtrar” cambios repetidos: si hacemos el mismo cambio en dos ramas, sólo aparecerá una vez en la unión.
- Cambios diferentes sobre las mismas líneas producen conflictos

```
$ git checkout master    # estamos en master

$ git merge issue99      # mezclamos de issue99 en master

#### si hay conflictos ####

# editamos archivos con conflictos y los resolvemos

$ git commit -a -m "mezcla de issue99 y solución de conflictos"

$ git branch -d issue99  # borramos rama issue99
```

GUI

```
$ gitg
...

$ git gui
...

$ gitk --all
...
```

Moverse por la historia

```
# Recuperar archivos en su estado en una revisión anterior
$ git checkout 54de61

# Recuperar archivos de una etiqueta
$ git checkout v1.0

# Cambiar a una rama
$ git checkout <rama>

# Cambiar a una revisión creando una rama en ese punto
$ git checkout f7b177 -b prueba
```

Referenciar confirmaciones

- **git help gitrevisions**
- Número de revisión (SHA-1): completo o parcial
- Nombre de la etiqueta
- Nombre de la rama: referencia extremo de la rama
- Otras referencias
  - HEAD: revisión actual.
  - FETCH_HEAD: revisión del último “fetch”.

Referenciar confirmaciones

- Modificadores
  - <rev>~n: ancestro n-ésima generación (sólo primer padre)
  - <rev>^n: n-ésima referencia padre
  - <ref>@{n}: n-ésimo valor previos de <ref>
  - <ref>@{<tiempo>}: valor de <ref> en <tiempo>
  - :/<texto>: revisión anterior cuyo mensaje contiene <texto>
- Rangos
  - <rev>: todos los ancestros de <rev>
  - <rev1>..<rev2>
    - Cambios accesibles desde <rev2> que no son accesibles desde <rev1> (cambios en <rev2> desde que divergió de <rev1>).
  - <rev1>...<rev2>
    - Cambios accesibles desde <rev2> o <rev1> pero que no son accesibles desde los dos (cambios en <rev1> y <rev2> desde que divergieron).

Referenciar confirmaciones. Ejemplos

```
# Recuperar archivos Tres revisiones antes de la actual
$ git checkout HEAD~3

# Comparar cambios revisión anterior a master con rama tmp
$ git diff master~ tmp

# Comparar cambios segundo padre (tras merge)
$ git diff HEAD^2

# Diferencias entre dos padres (tras merge)
$ git diff HEAD^1 HEAD^2

# Cambios en la rama "test" que no están en "master"
$ git diff master..test

# Cambios en la rama actual y la remota desde que divergieron
$ git fetch origin master
$ git diff HEAD...FETCH_HEAD

# Lista de cambios desde ayer a las 6pm hasta hoy a las 12pm en la
# rama tmp
$ git log 'tmp@{yesterday 18:00}'..'tmp@{today 12:00}'
```

Comparar cambios

```
# Cambios respecto al índice (o última revisión)
$ git diff
...

# Cambios respecto a una revisión anterior (rama, etc.)
$ git diff 62b19c
...

$ git diff HEAD
...

$ git diff v1.0
...

# Cambios entre dos revisiones (versiones, ramas, etc.)
$ git diff jorge v1.0
...

# Cambios en el índice respecto a otra revisión (HEAD por defecto)
$ git diff --cached [<commit>]

# Cambios que se aplicarán en la siguiente confirmación
$ git diff --cached

# Cambios en archivos concretos
$ git diff [...] -- <archivos>
...
```

Añadir cambios interactivamente

- Modo interactivo de “add”
  - Permite seleccionar sólo algunos cambios de uno o varios archivos.
  - Permite confirmar sólo aquellos cambios relativos a una tareas concreta: mejor organización de confirmaciones.
- Interfaz interactiva en el terminal. Permite:
  - Seleccionar partes a incluir en el índice (patch)
  - Ver actualizaciones en el índice (diff)
  - Eliminar actualizaciones del índice (revert)

Añadir cambios interactivamente

```
$ git add --interactive fruta.txt
           staged     unstaged path
  1:    unchanged        +2/-0 fruta.txt

*** Commands ***
  1: status   2: update   3: revert   4: add untracked
  5: patch    6: diff     7: quit     8: help
What now> p
           staged     unstaged path
  1:    unchanged        +2/-0 fruta.txt
Patch update>> 1
           staged     unstaged path
* 1:    unchanged        +2/-0 fruta.txt
Patch update>> [enter]
diff --git a/fruta.txt b/fruta.txt
index 5d90e31..4b685ea 100644
--- a/fruta.txt
+++ b/fruta.txt
@@ -1,6 +1,8 @@
 aguacate
 fresas
+manzanas
 melón
+naranjas
 patatas
 sandía
 tomates
```

```
Stage this hunk [y,n,q,a,d,s,e,?]? s
Split into 2 hunks.
@@ -1,3 +1,4 @@
 aguacate
 fresas
+manzanas
 melón
Stage this hunk [y,n,q,a,d,j,J,g,/,e,?]? y
@@ -3,4 +4,5 @@
 melón
+naranjas
 patatas
 sandía
 tomates
Stage this hunk [y,n,q,a,d,K,g,/,e,?]? q

*** Commands ***
  1: status   2: update   3: revert   4: add untracked
  5: patch    6: diff     7: quit     8: help
What now> q

$ git commit -m "Manzanas"
[master a7d431a] Manzanas
 1 file changed, 1 insertion(+)

$ git commit -am "Naranjas"
[master fb9df4f] Naranjas
 1 file changed, 1 insertion(+)
```

Manipular confirmaciones

- Sin cambiar la historia
  - Recuperar versiones de archivos de una confirmación anterior (checkout --)
  - Actualizar el índice con archivos de una confirmación anterior (reset --)
  - Deshacer los cambios de una confirmación anterior (revert)
  - Importar cambios de otra rama (cherry-pick)
- ¡Cambiando la historia! ¡¡¡Nunca en ramas o revisiones que hayan sido compartidas!!!
  - Corregir la última confirmación (commit --amend)
  - Mover una rama a una confirmación diferente (reset)
  - Trasplantar una rama (rebase)
  - Combinar varias confirmaciones (squash) (rebase)
  - Eliminar confirmaciones (rebase)
  - Editar confirmaciones (rebase)
  - Dividir confirmaciones

Recuperar archivos de confirmaciones anteriores

- Recuperar archivos de una confirmación anterior
  - Descartar los cambios en un archivo en la carpeta de trabajo.
  - Tomar un archivo de otra rama, versión, etc.
- Actualizar el índice con archivos de una confirmación anterior
  - Descartar archivos del índice (stage)

```
# git checkout [<commit>] -- <archivos>   # HEAD si no <commit>

# Descartar cambios en interface.h
$ git checkout -- interface.h

# Tomar un archivo de la versión 1.0
$ git checkout v1.0 -- log.c

# Descartar cambios programados
$ git reset HEAD -- interface.h
```

Deshacer cambios sin cambiar la historia

- Aplica el cambio inverso a una confirmación anterior, para deshacer sus efectos.
- Deshace cambios sin alterar la historia.

```
# Deshacer los cambios de una revisión anterior
$ git revert fe77              # sin conflictos
(editar comentario)

$ git revert bug89             # con conflictos
(editar archivos, etc.)

$ git revert --continue        # continúa el revert
...
$ git revert --abort           # vuelve al estado antes del revert
```

Importar cambios de otra rama

- Aplicar en la rama actual una confirmación de otra rama
- Ej: aplicar correcciones a la rama principal antes de fusionar otras ramas.

```
$ git checkout dev
...
$ git commit -m "Bug fix"
[dev 7f6c364] Bug fix
 1 file changed, 1 deletion(-)

$ git checkout master

$ git cherry-pick 7f6c364
[master ca1a425] Bug fix
 Date: Fri Oct 12 01:46:40 2018 +0200
 1 file changed, 1 deletion(-)
```

Corregir la última confirmación

- Útil para corregir una confirmación que se acaba de hacer.
  - Edición adicional de archivos
  - Cambiar datos del autor
  - Etc.

```
# Corregir la última confirmación
(editar archivos, actualizar índice, etc.)
$ git commit --amend [--reset-author]
...
```

Mover una rama a una posición diferente

- Descartar cambios en la rama
- Corregir errores: hemos trabajado en la rama equivocada

```
# Mover una rama a una revisión anterior. Opciones:
#   --soft: mantiene los archivos actuales y el índice
#   --hard: actualiza los archivos y el índice (¡cuidado!)
#   --mixed: actualiza el índice pero no los archivos (predeterminado)

# Descartar las 3 últimas confirmaciones
$ git reset HEAD~3

# Devolver una rama al estado anterior
$ git reset --hard HEAD@{1}

# Localizar una revisión (posiblemente perdida) e iniciar una rama ahí
$ git reflog
...
$ git checkout HEAD@{5} -b recup
```

Tras un 'reset', las revisiones que no queden ligadas a ninguna rama dejarán de aparecer en la historia ('git log') y se borrarán con el tiempo. Estas revisiones pueden localizarse con 'reflog' y recuperarse con otro 'reset' o con 'checkout'.

Trasplantar una rama

- Trasladar una rama como si se hubiera iniciado desde una revisión diferente. Casos de uso:
  - Incorporar cambios de una rama principal
  - Hacer depender los cambios de una versión determinada para facilitar la integración de cambios

```
# Trasladar la rama actual (tmp) al extremo de la rama master
$ git checkout tmp
$ git rebase master

# Trasladar la rama actual a una versión determinada
$ git rebase v1.0

# Continuar el 'rebase' tras solucionar conflictos
$ git rebase --continue

# Abortar un rebase en marcha
$ git rebase --abort
```

Editar las confirmaciones de una rama

- “rebase” tiene un modo interactivo que permite reubicar una rama y a la vez:
  - Eliminar confirmaciones
  - Unir confirmaciones
  - Editar confirmaciones
- Es la herramienta más poderosa de Git a la hora de reescribir la historia.
- Múltiples casos de uso:
  - Unir confirmaciones que tratan de la misma tarea, simplificando la historia.
  - Eliminar confirmaciones erróneas.
  - Dividir confirmaciones anteriores: crear una historia del desarrollo más lógica.

Combinar confirmaciones (squash)

- Combina varias revisiones en una
  - tenemos muchas revisiones poco interesantes
  - necesitamos simplificar la historia antes de compartir nuestra rama

```
# Iniciamos un 'rebase' interactivo desde la última revisión que
# queremos conservar. Editamos revisiones desde HEAD~3 a HEAD
$ git rebase -i HEAD~4
...
```

```
pick 1d28e40 ddddddddddddddddddddddd
pick 31177b7 ccccccccccccccccccccc
pick 3fd2ac2 bbbbbbbbbbbbbbbbbbbbb
pick 846efbd aaaaaaaaaaaaaaaaaaaaa

# Rebase 994c564..846efbd onto 994c564
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
```

Combinar confirmaciones (squash)

```
pick 1d28e40 ddddddddddddddddddddddd
pick 31177b7 ccccccccccccccccccccc
pick 3fd2ac2 bbbbbbbbbbbbbbbbbbbbb
pick 846efbd aaaaaaaaaaaaaaaaaaaaa
...
```

```
pick 1d28e40 ddddddddddddddddddddddd
s 31177b7 ccccccccccccccccccccc
s 3fd2ac2 bbbbbbbbbbbbbbbbbbbbb
s 846efbd aaaaaaaaaaaaaaaaaaaaa
...
```

```
... (editamos nueva revisión combinada) ...
...
Successfully rebased and updated refs/heads/tmp.
$ git log
...
```

Dividir una confirmación

- Hemos hecho una confirmación con demasiados cambios y queremos separarlos en varias confirmaciones
  - Cambiar base de la rama a la confirmación “anterior” a la que queremos dividir (rebase). Usar modo interactivo.
    - No necesario si editamos la última confirmación.
  - Marcar la confirmación a dividir para editar (“edit”).
  - Volver una confirmación atrás sin modificar los archivos (reset)
  - Añadir cambios de forma selectiva (add, add -i)
  - Confirmar cada cambio
  - Continuar el rebase

Dividir una confirmación

```
$ git log --pretty=oneline
94524e (HEAD -> dev) Nata y garbanzos
7d3de9 Cordero con naranja y fresas
ce496c Ponemos sandía
cfd1bc Inicial

$ git rebase --interactive ce496c
[marcar commit 7d3dw9 para editar]

Detenido en 7d3de9b...  Cordero con naranja y fresas

$ git reset HEAD^
Cambios fuera del área de stage tras el reset:
M     carne.txt
M     fruta.txt

$ git add carne.txt
$ git add --interactive fruta.txt
[añadir sólo parche que contiene "naranja"]

# Comprobamos cambios en el índice
$ git diff --cached
...
```

```
$ git commit -m "Cordero con naranja"
[HEAD desacoplado 722ee38] Cordero con naranja
 2 files changed, 2 insertions(+), 1 deletion(-)

# Comprobamos cambios pendienes
$ git diff
...

$ git commit -am "Fresas"
[HEAD desacoplado cc13b70] Fresas
 1 file changed, 1 insertion(+)

$ git rebase --continue
Successfully rebased and updated refs/heads/dev.

$ git log --pretty=oneline
f54d67 (HEAD -> dev) Nata y garbanzos
cc13b7 Fresas
722ee3 Cordero con naranja
ce496c Ponemos sandía
cfd1bc Inicial
```

merge vs rebase + ff

- Muchos proyectos prefieren hacer un “rebase” de las ramas de desarrollo a la rama “master” antes de mezclar los cambios:
  - Se resuelven los conflictos antes de mezclar las ramas.
  - La mezcla posterior resulta en un “fast-forward”.
  - La historia de la rama “master” es siempre lineal
- También es frecuente combinar todas o casi todas las confirmaciones de la rama de desarrollo durante el rebase:
  - Simplifica la historia de la rama “master”.
  - Se pueden conseguir confirmaciones más fáciles de editar en el futuro: corrección de errores, etc.

Referencias

- Scott Chacon. “Pro Git”. http://git-scm.com/book
- “Control de Versiones”. Wikipedia. http://es.wikipedia.org/wiki/Control_de_versiones
- Vincent Driessen. “A successful Git branching model”. http://nvie.com/posts/a-successful-git-branching-model
- Bryan O'Sullivan. “Mercurial: The Definitive Guide”. http://hgbook.red-bean.com/
- “Git vs Mercurial”. WikiVS. http://www.wikivs.com/wiki/Git_vs_Mercurial