Sistemas de Control de Versiones

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

Jorge Juan <jjchico@dte.us.es>, Enrique Ostúa <ostua@dte.us.es> 2013-19

Contenidos teóricos

- ¿Qué es? Objetivos del control de versiones
- Modelos generales: centralizado vs distribuido
- Eventos básicos
- Eventos avanzados
- Flujos de desarrollo

Organización

- 8 sesiones
  - 1. Introducción
  - 2. Repositorios locales
  - 3. Repositorios remotos
  - 4. Ejercicio repositorios remotos
  - 5. Servicios de alojamiento
  - 6. Ejercicio servicios de alojamiento
  - 7. Introducción a los servicios CI/CD
  - 8. Proyecto final

¿Qué es el control de versiones?

- ¿Control de versiones o control de revisiones?

Algoritmo diff

- Control de revisiones: registro de cambios en archivos
- Principalmente archivos de texto (código)
- Múltiples cambios
- Múltiples archivos
- Múltiples orígenes
- ...

Ejemplo diff/patch

```
$ cat lista.txt
Patatas
Melones
Sandías
Manzanas
Setas

$ cat lista2.txt
Patatas
Melones
Sandías
Naranjas
Manzanas
Setas

$ cat lista3.txt
Patatas
Melones
Sandías
Manzanas
Setas
Zanahorias
```

```
$ vi lista.txt
... (lista2.txt, lista3.txt) ...
$ cp lista.txt lista-orig.txt
$ diff -u lista.txt lista2.txt > diff2.txt
$ diff -u lista.txt lista3.txt > diff3.txt
...
$ patch lista.txt diff2.txt
...
$ patch lista.txt diff3.txt
...
$ cat diff2.txt diff3.txt | patch lista.txt
```

Ejemplo diff/patch

```
$ cat lista.txt
Patatas
Melones
Sandías
Manzanas
Setas

$ cat lista2.txt
Patatas
Melones
Sandías
Naranjas
Manzanas
Setas

$ cat lista4.txt
Patatas
Melones
Sandías
Limones
Manzanas
Setas
```

```
$ diff -u lista.txt lista4.txt > diff4.txt

$ patch lista.txt diff2.txt
patching file lista.txt

$ patch lista.txt diff4.txt
patching file lista.txt
Hunk #1 FAILED at 1.
1 out of 1 hunk FAILED -- saving rejects to file
lista.txt.rej

$ vi lista.txt.rej
...

$ man patch
...
```

Control de versiones manual

- Versiones gestionadas con recursos generales
  - Carpetas
  - copiar archivos
  - registros de cambios manuales (log)
  - etc.
- Distribución de cambios mediante diff/patch
  - Correo electrónico
  - Listas de correo
- Inconvenientes
  - Poco eficiente. Muchas copias inútiles
  - Volver atrás en la historia
  - Gestionar múltiples colaboradores
  - Saber quién hace qué (blame)

Sistemas de control de versiones

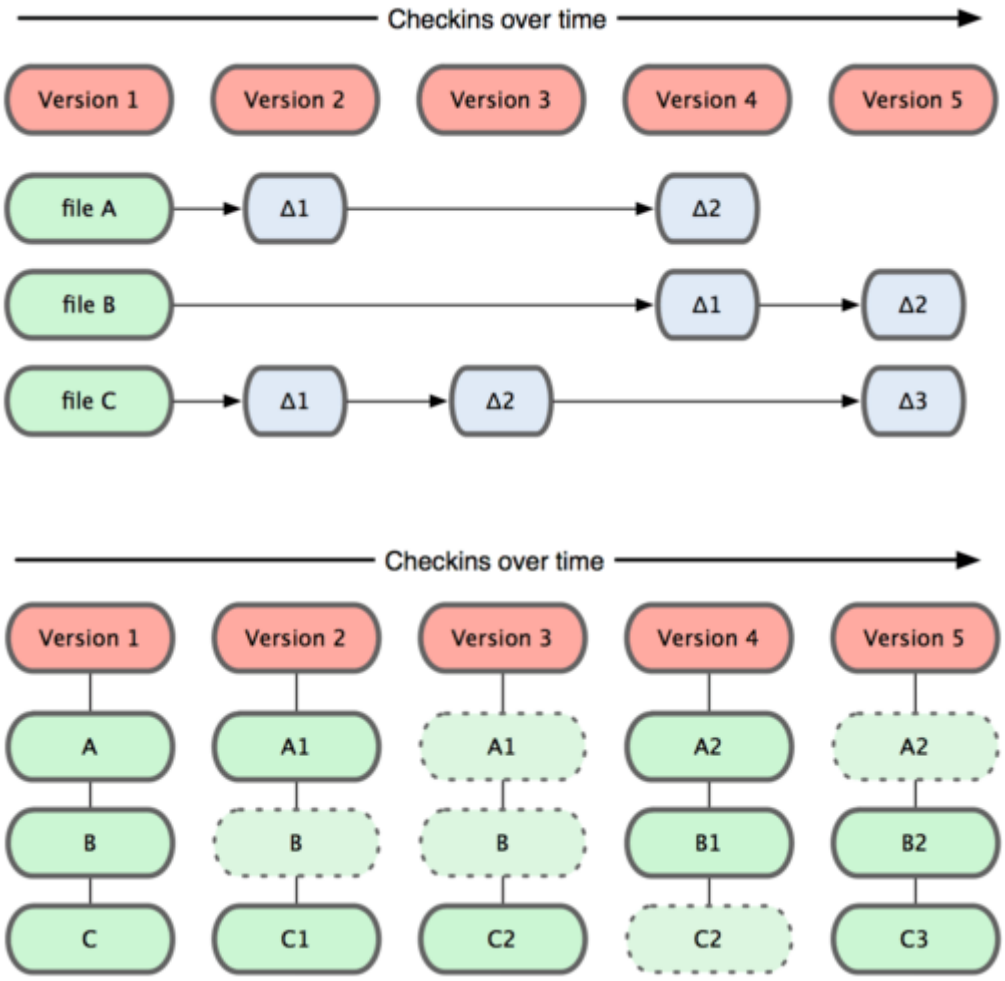
- Registrar cambios con facilidad
- Gestión eficiente de los cambios (diff/patch)
- Volver a un estado anterior
- Colaborar con otros desarrolladores

Ejemplo git

```
$ cd git-example

$ cat lista.txt
Patatas
Melones
Sandías
Manzanas
Setas
```

```
$ git init
...
$ git commit
...
$ git add lista.txt

$ git commit

$ git log
...
$ vi lista.txt
...
$ git add lista.txt
$ git commit -m "Añadido Naranjas"
...
$ git log -p
...
$ vi lista.txt
...
$ git commit -a -m "Añadido Zanahorias"
$ git log --pretty --graph
...
```

Ejemplo git

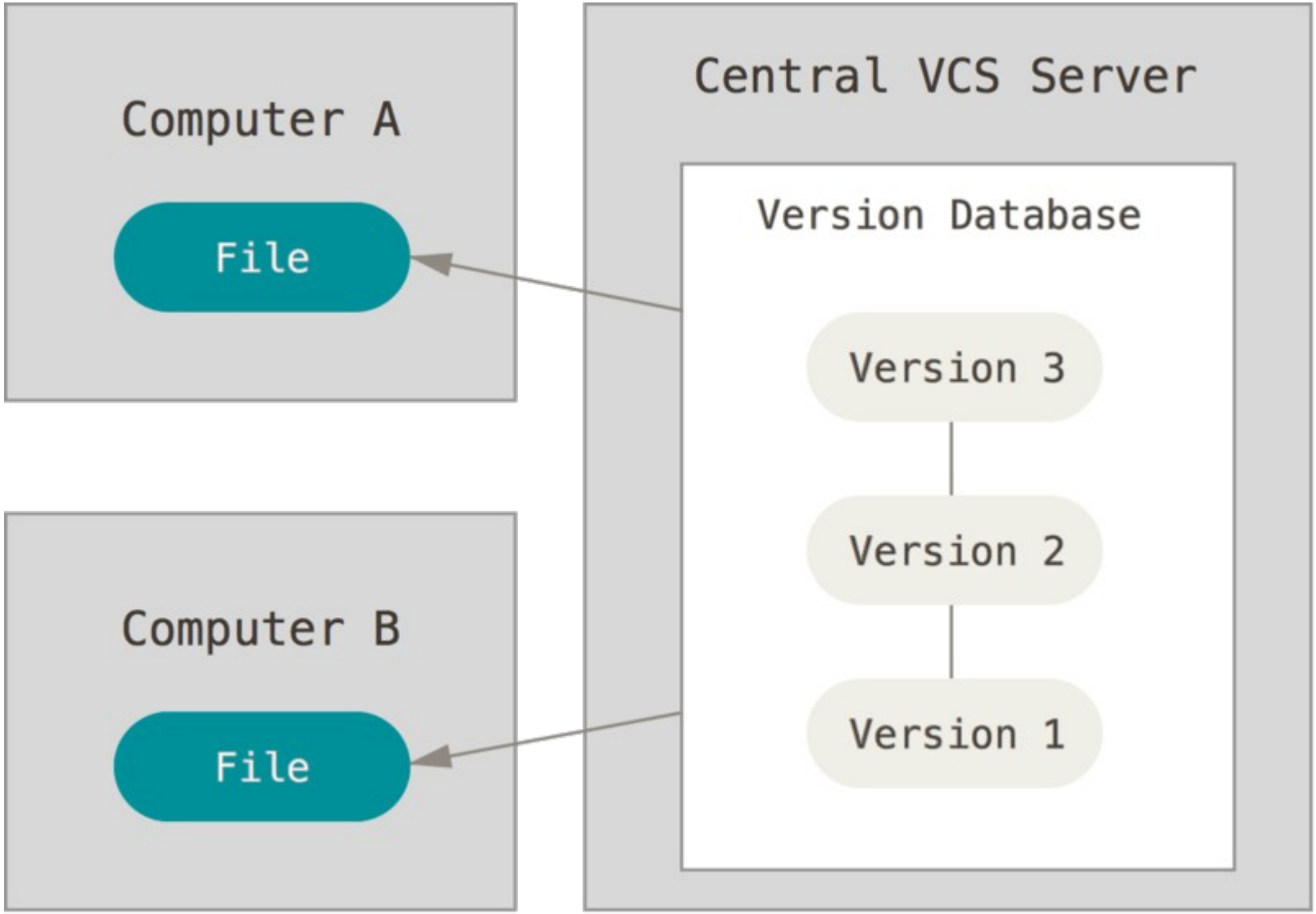

Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
file A
Δ1
Δ2
file B
Δ1
Δ2
file C
Δ1
Δ2
Δ3
Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

Evolución de los SCV

- Modelo de datos locales
  - Datos en el disco duro local
  - Uno o pocos desarrolladores
- Modelo cliente-servidor (centralizados)
  - Repositorio central
  - Los cambios se envían a través de la red
  - Ej: CVS, Subversion.
- Modelo cliente-servidor (distribuidos)
  - Cada desarrollador tiene un repositorio local
  - Unos repositorios pueden sincronizarse con otros
  - Incluye el modelo centralizado como caso particular
  - Ej: Git, Mercurial, Bazaar.

http://en.wikipedia.org/wiki/List_of_revision_control_software

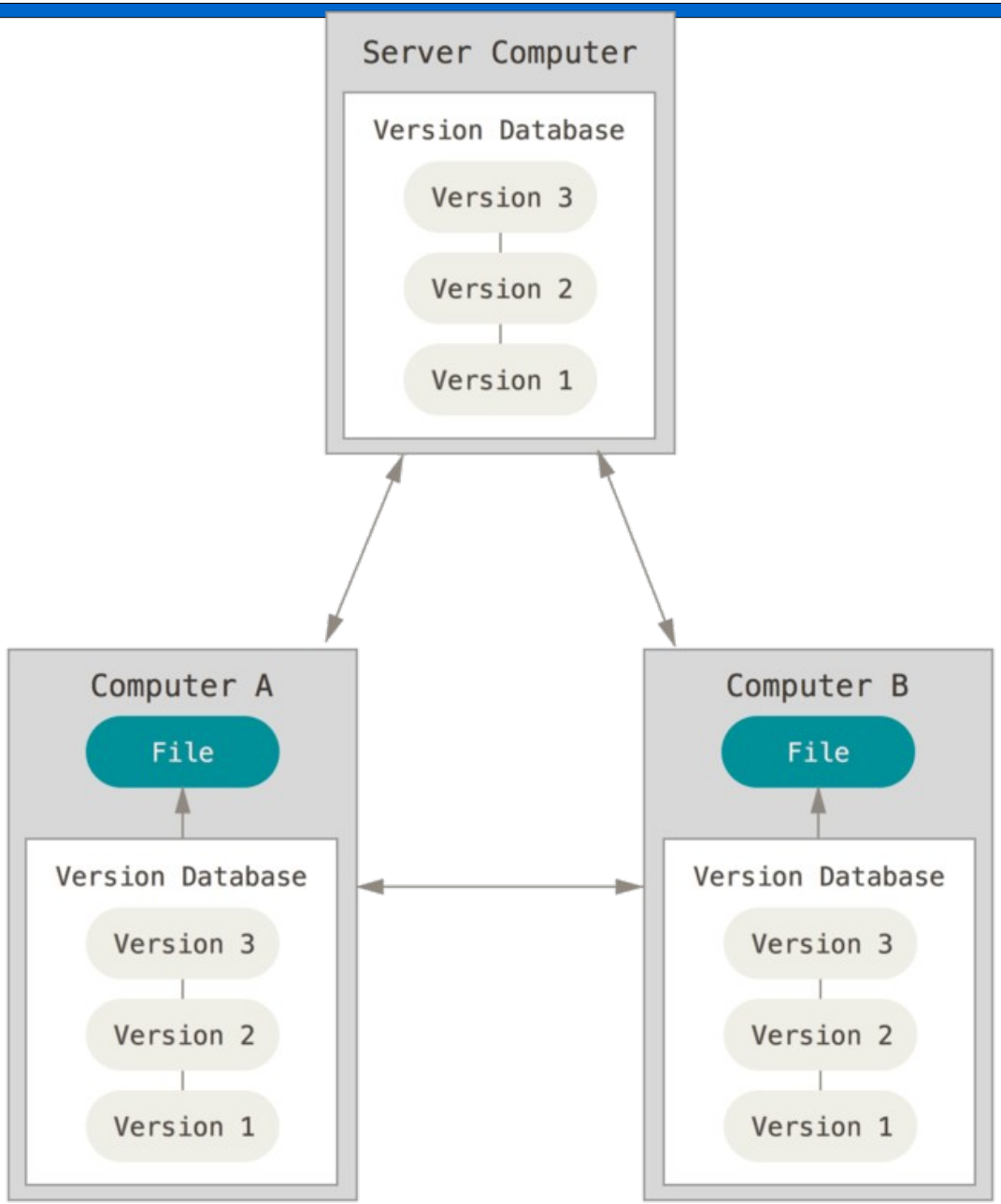
Modelo centralizado



Modelo centralizado

- Todo el mundo sabe, hasta cierto punto, en qué está trabajando el resto.
- Hay una copia principal centralizada, facilita la administración.
- Permite limitar fácilmente a que tiene acceso cada uno.
- Uno punto de fallo, caída del servidor implica que no se puede seguir colaborando.
- Las copias locales no son copias completas del servidor.

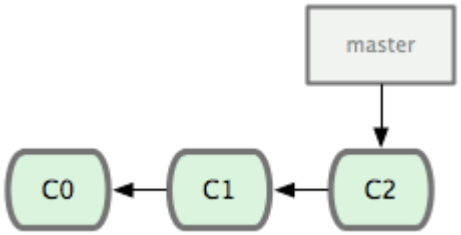
Modelo distribuido

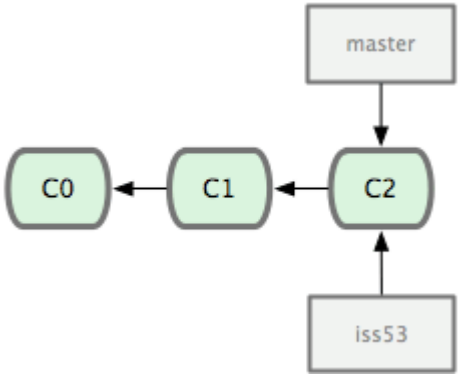


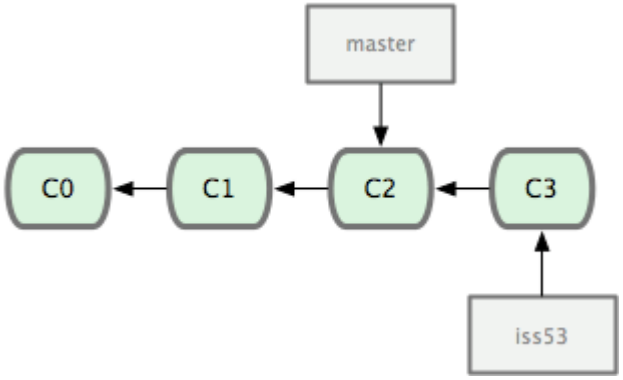
Modelo distribuido

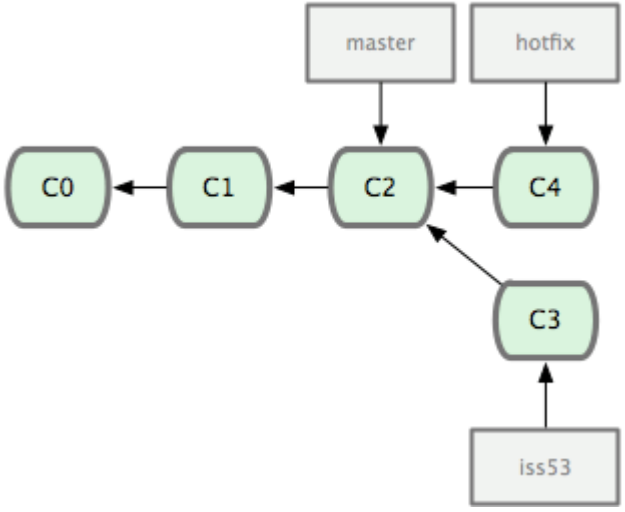
- No hay una copia principal
- Operaciones rápidas: trabajan sobre copia local
- Cada copia local actúa como una copia de seguridad
- Permite múltiples repositorios “centrales”
- Permite múltiples modelos de desarrollo (p.ej. centralizado o jerárquico)
- Permite trabajar “desconectado”
- Permite gestionar partes privadas del proyecto

Ramas

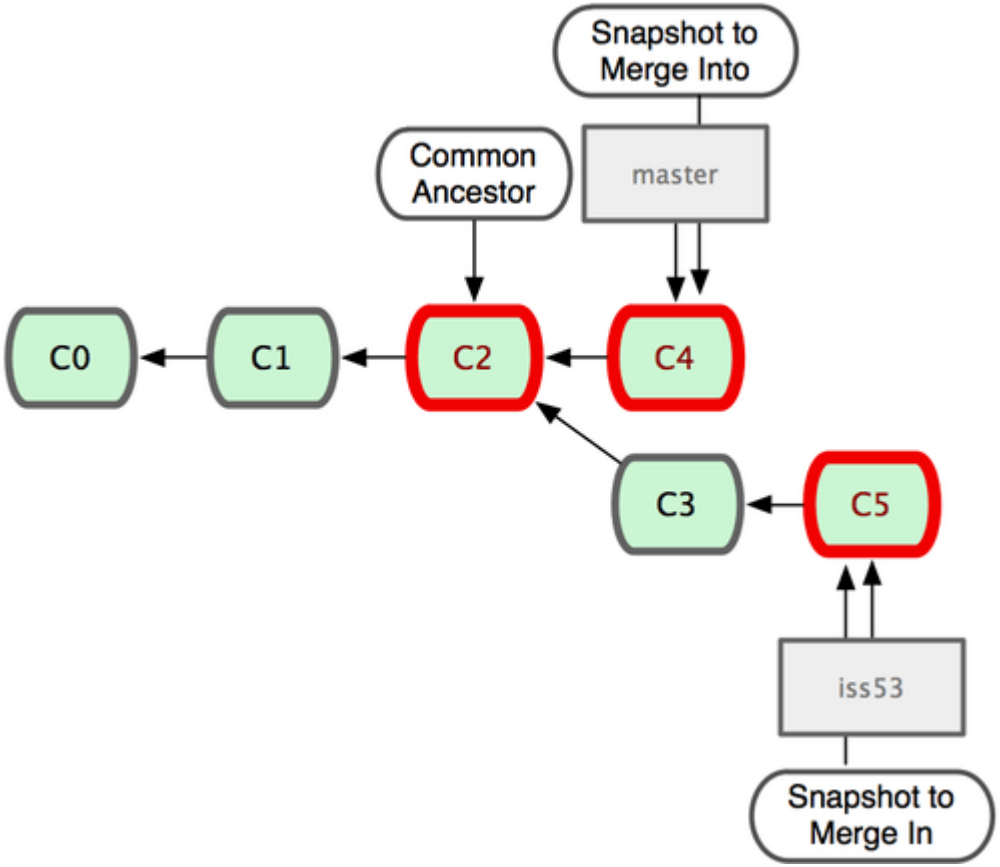

master
C0
C1
C2


master
C0
C1
C2
iss53

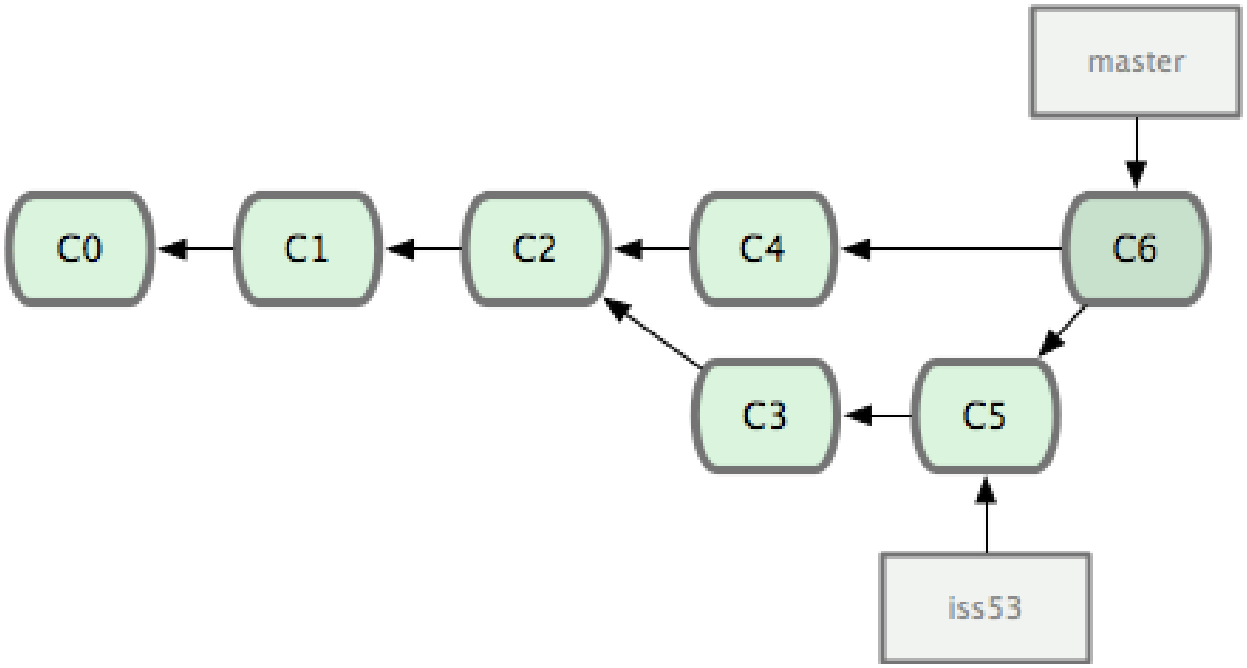

master
C0
C1
C2
C3
iss53


master
hotfix
C0
C1
C2
C4
C3
iss53

Ramas

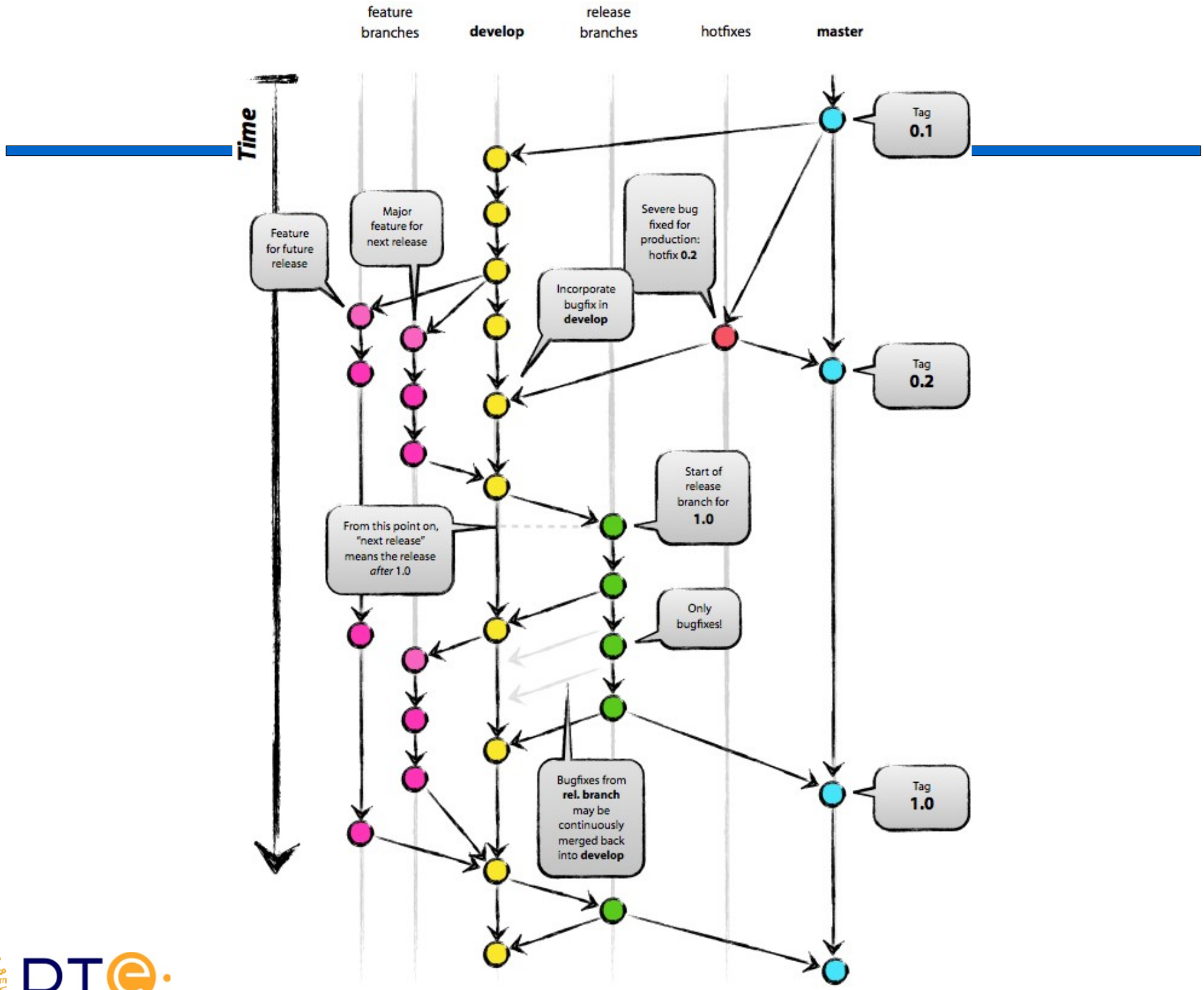

Snapshot to Merge Into
Common Ancestor
master
C0
C1
C2
C4
C3
C5
iss53
Snapshot to Merge In

Ramas



Ramas

- Ramas de largo recorrido
  - Principal (master)
  - Versiones
  - Desarrollo
- Ramas puntuales
  - Reparaciones (hotfix)
  - Nuevas funcionalidades
  - Pruebas
  - ...


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

Referencias

- “Control de Versiones”. Wikipedia. http://es.wikipedia.org/wiki/Control_de_versiones
- Scott Chacon. “Pro Git”. http://git-scm.com/book
- 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