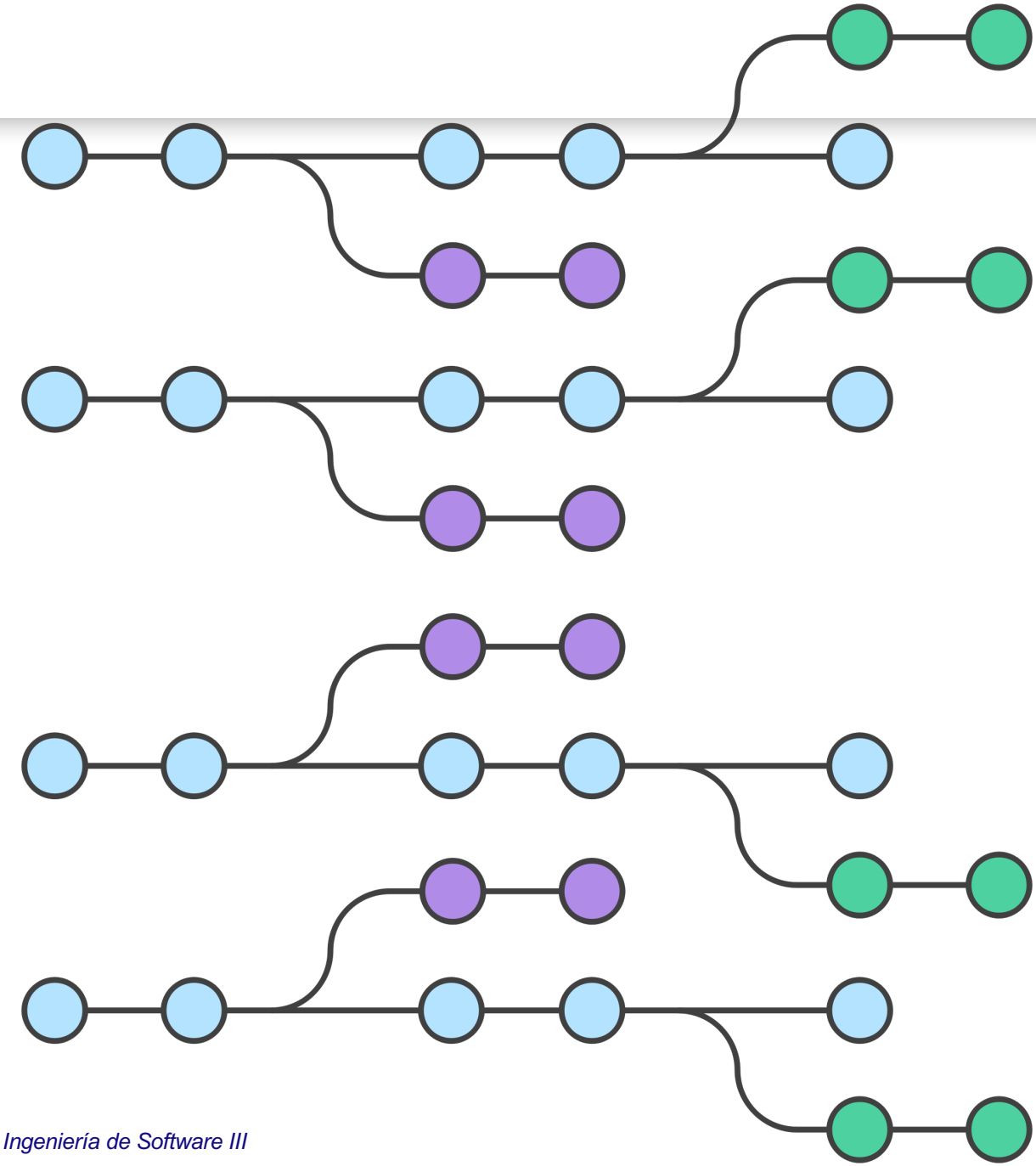


Ingeniería de Software III

Herramientas de Control de Versiones

Parte 1

Contenido

- Gestion de la configuración
  - Motivación
  - Definición
  - Actividades
- Control de Versiones
  - Conceptos
  - Terminología
  - Herramientas de versionado (VCS)
    - Funciones
  - Tipos de VCS
  - Ramas (Branching)
  - Fusión (Merge)

Motivación

- Software compuesto de varias partes cambiantes
  - Cambia constantemente durante su desarrollo y uso
  - Un sistema, varias versiones
    - Pueden haber diferentes versiones para clientes diferentes
    - Pueden haber diferentes versiones durante la etapa de desarrollo
    - Pueden haber diferentes versiones durante la vida de un sistema
- El software consiste de un conjunto de versiones que deben mantenerse y gestionarse
- Trabajo en equipo
  - Puede haber diferentes equipos involucrados en el desarrollo de las diferentes versiones

Gestión de la Configuración

- Gestión de la Configuración (CM)

"Políticas, herramientas y procesos para administrar los cambios del software"

  - Es necesario para identificar cambios y versiones de componentes incorporados en cada versión del sistema
  - Esencial para el desarrollo de todos los sistemas, en especial si el desarrollo lo lleva a cabo un equipo

"CM define como una organización construye y libera (builds and releases) sus productos, identifica y gestiona los cambios"

Actividades de CM

- Control de versiones
  - Seguimiento de las versiones de los componentes del sistema
  - Garantizar que los cambios hechos por diferentes desarrolladores no interfieran entre si
- Construcción del sistema
  - El proceso de ensamblar componentes, datos y librerías para crear un sistema ejecutable
- Release management
  - Preparar el software para ser liberado (release)
  - Seguimiento de versiones entregadas a los clientes
- Administración del cambio
  - Seguimiento de los pedidos de cambio de los clientes y desarrolladores,
  - Estimar el costo y el impacto de dichos cambios
  - Decidir si deben implementarse

Actividades de CM

- Control de versiones
  - Seguimiento de las versiones de los componentes del sistema
  - Garantizar que los cambios hechos por diferentes desarrolladores no interfieran entre si
- Construcción del sistema
  - El proceso de ensamblar componentes, datos y librerías para crear un sistema ejecutable
- Release management
  - Preparar el software para ser liberado (release)
  - Seguimiento de versiones entregadas a los clientes
- Administración del cambio
  - Seguimiento de los pedidos de cambio de los clientes y desarrolladores,
  - Estimar el costo y el impacto de dichos cambios
  - Decidir si deben implementarse

UNIVERSIDAD CATÓLICA DE CÓRDOBA
Universidad Jesuita

CONTROL DE VERSIONES

Control de Versiones

- Identificación y seguimiento de las diferentes versiones de los componentes de software o items de configuración y de los sistemas en los cuales son usados

"El proceso por el cuál se gestionan los codelines y las baselines"

Terminología

- Codeline
- Version
- Snapshot
- Baseline

Codeline - Definición

- Codeline
  - Secuencia de versiones
  - Versiones recientes derivan de versiones anteriores
  - Aplican a componentes de un sistema

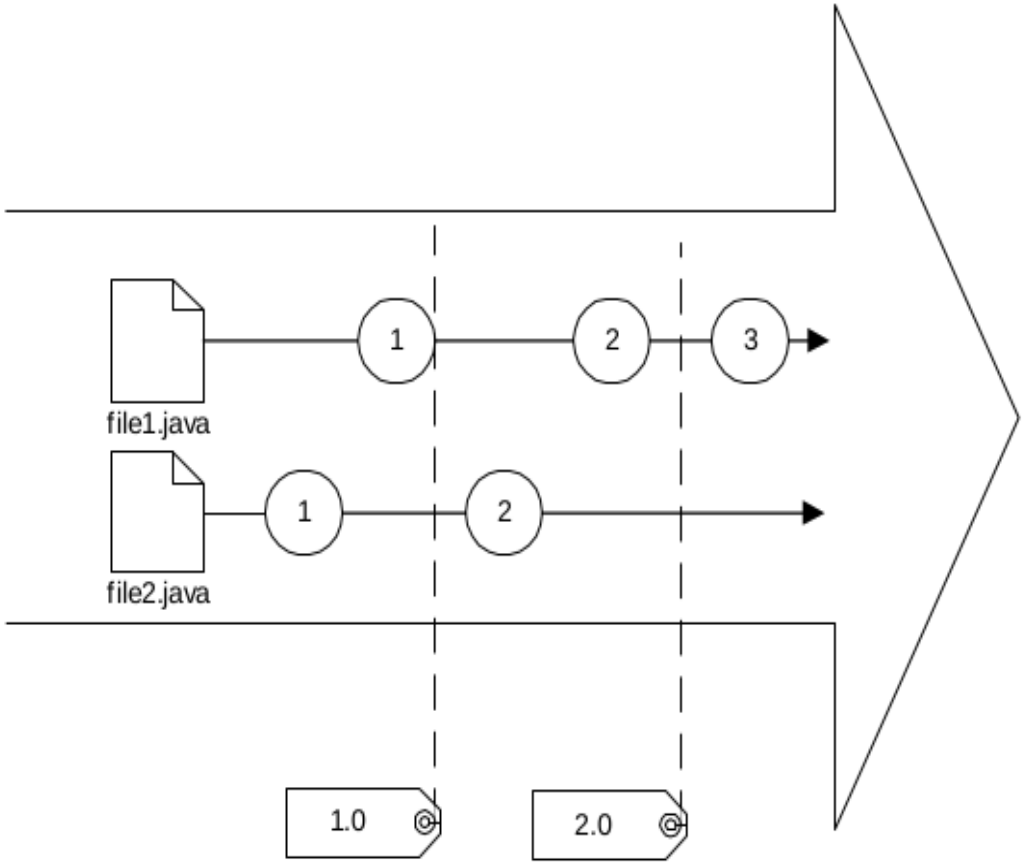
  "Conjunto de archivos fuente y otros ítems de configuración que conforman un componente de software a lo largo del tiempo mientras cambian"

- Cuando se cambia un item se crea una nueva revisión de ese ítem

  *"Un **codeline** contiene todas las revisiones de todos los **items** a lo largo de un camino evolutivo de un componente"*

Codeline – Version - Snaphot

- Snapshot -> contiene una revisión específica de cada ítem en un codeline
- Un **snapshot** de un **codeline** se dice que es una **version**
- Una **version** de puede identificar con una etiqueta



Codeline

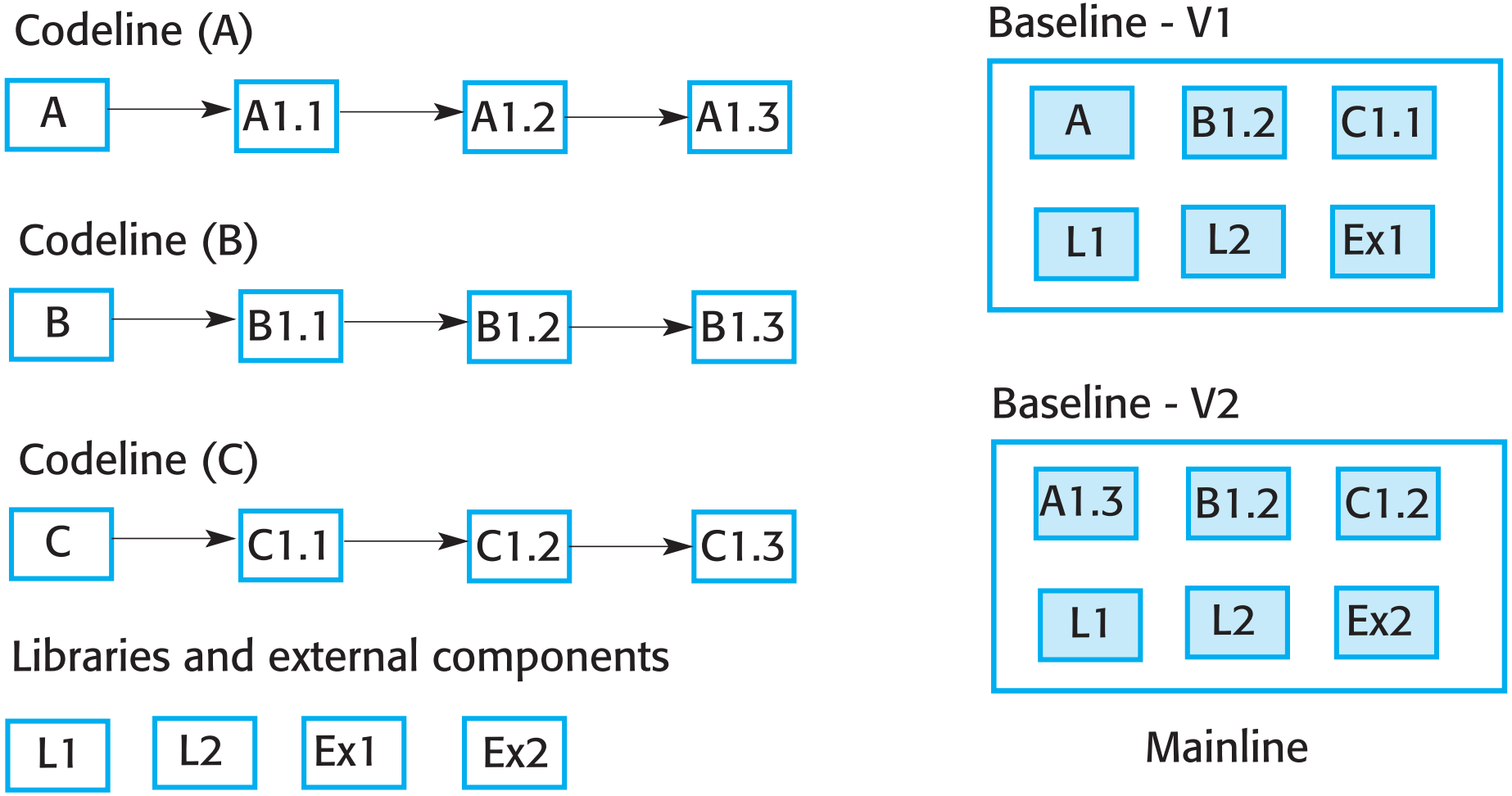
- Un producto puede ser construido a partir de
  - Un solo codeline
  - Más de un codeline

Baseline

- Baseline

  "*Definición específica de un sistema*"
- Especifica las versiones de los componentes incluidos en un sistema
  - *Librerías*
  - *Configuraciones*
  - *Entorno de ejecución*
  - *Etc*
- Sirven para recrear una versión específica de un sistema
  - Por ejemplo, para diferentes clientes, cuando se reporta un bug o se pide una nueva funcionalidad
- No se deben cambiar
- Sirven como base para construir nuevas versiones

Codelines y Baselines



UNIVERSIDAD CATÓLICA DE CÓRDOBA
Universidad Jesuita

Herramientas de Versionado

Herramientas de Control de Versiones (VCS)

- Motivación
- Definición
- Funcionalidades
- Caracteríticas
- Generaciones
- Branching models
- Desarrollo Open Source

Motivación de VCS

1) The amount of space to store the source code (whether on disk, tape or cards) may be several times that needed for any particular version. For example, there might be "customer," "system test," and "development" source libraries, with most modules represented by a different version in each.[1]

2) Fixes made to one version of a module sometimes fail to get made to other versions.

3) When changes occur it is difficult to tell exactly what changed and when.

4) When a customer has a problem it is hard to figure out what version he has.

1975

SCCS Paper

Sistema de Control de Versiones

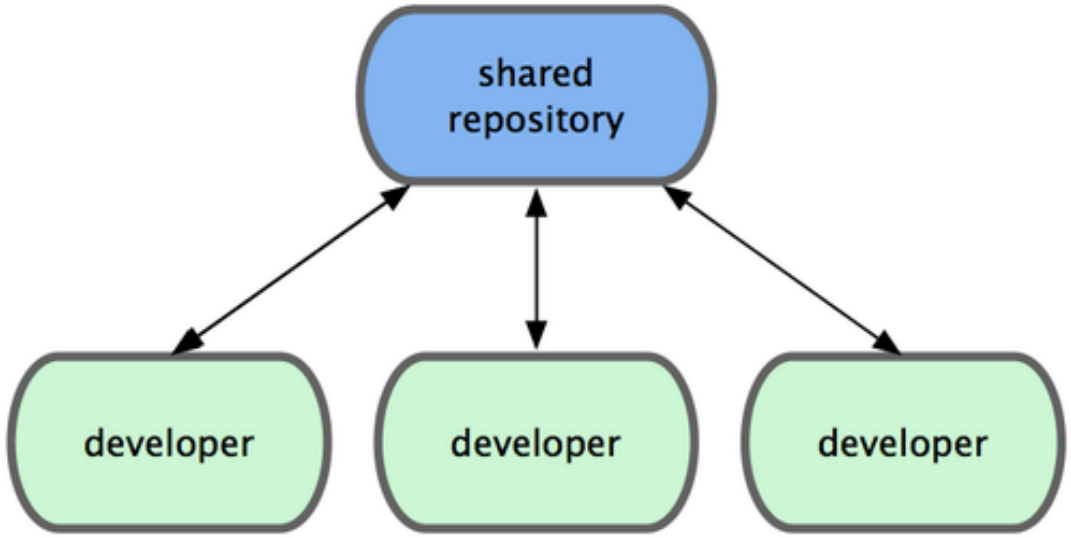
"Sistemas que identifican, almacenan y controlan el acceso a las diferentes versiones de los items de configuración"

VCS - Funcionalidad

- Identificación de Versiones y Releases
  - Crea versiones de los items cuando estos se envían al sistema
- Registro del historial de cambios
  - Mantiene registro de todos los cambios que se producen en cada item
- Soporte para el desarrollo Independiente
  - Permite el desarrollo concurrente de varios desarrolladores incluso sobre el mismo item
- Soporte de proyecto
  - Permite hacer check in y check out de grupos de items
- Gestión del almacenamiento
  - Maneja eficientemente el almacenamiento

VCS – Repositorio y Workspace

- Soporte de desarrollo independiente
  - Repositorio de proyecto público
  - Area de trabajo privada



Area de trabajo (workspace)

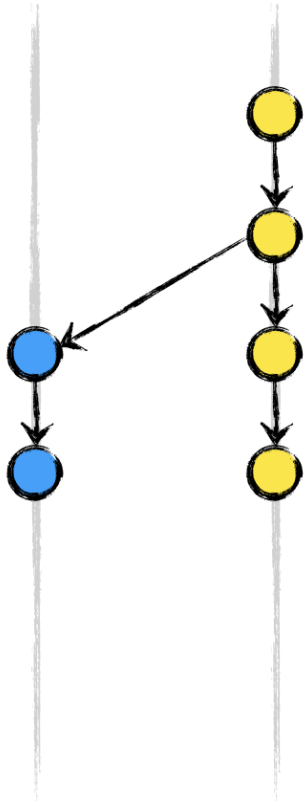
- Donde el desarrollador mantiene todos los items que necesita para realizar una tarea
  - Generalmente un directorio
- Contiene versiones específicas de los items
- Debería contener un mecanismo para construir ejecutables a partir de su contenido
  - Source code, source code for tests, libraries, scripts to build
- Puede ser manejado en el contexto de una IDE

Branching and merging

- En lugar de tener una sola secuencia de versiones que reflejen los cambios a un componente en el tiempo, suele haber varias secuencias independientes
  - Esto es normal durante el desarrollo del sistema dónde diferentes usuarios trabajan de manera independiente en diferentes versiones del código y por lo tanto cambia de manera diferente
  - También puede suceder que una versión del componente esté en operación, mientras que otra versión esté en desarrollo. En este caso puede ser necesario fixear bugs críticos en la versión en operación antes de que se entrega nueva versión
- En algún momento puede ser necesario crear una nueva versión de un componente que incluya todos los cambios que se hayan hecho de manera independiente, realizando una operación merge.
  - Si los cambios hechos involucran diferentes partes del código, las versiones de los componentes pueden mergearse de manera automática combinando los deltas que aplican a cada código
  - También puede ser necesaria la intervención de un desarrollador para resolver conflictos de merge

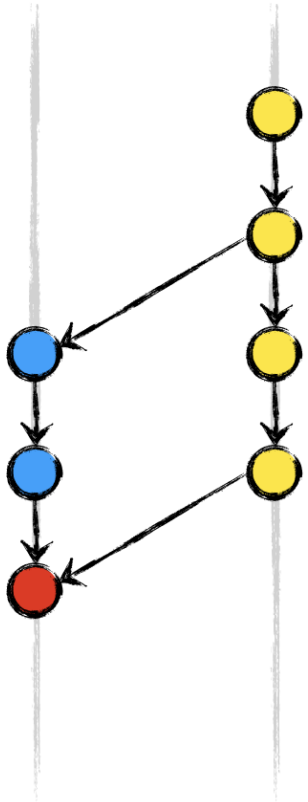
Merge vs Rebase

feature/login develop



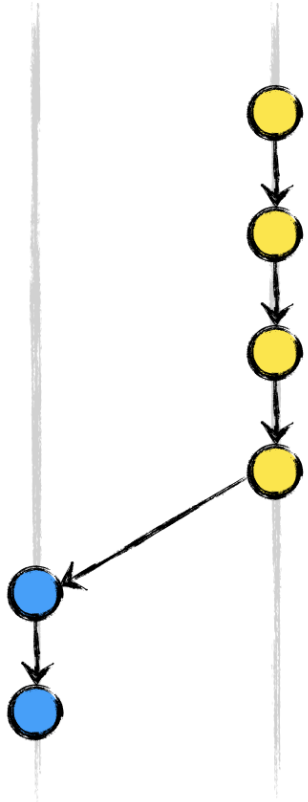
$ git merge develop

feature/login develop



$ git rebase develop

feature/login develop

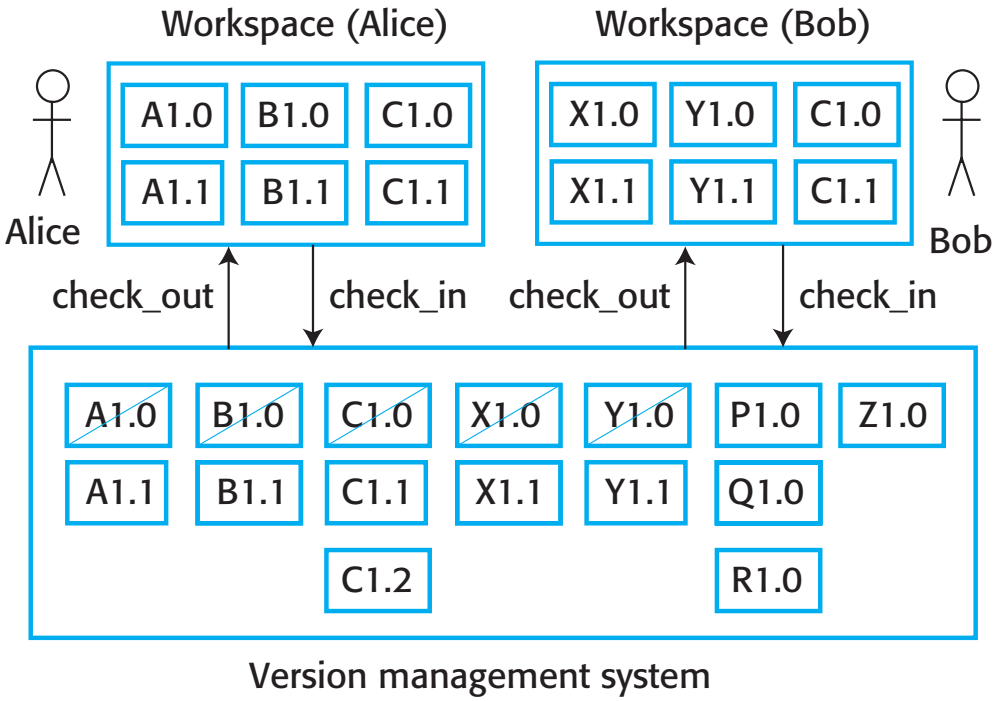


Caracterísitcas

- Caracterísitcas de VCS de acuerdo a
  - Arquitectura
    - Centralizado
    - Distribuido
  - Atomicidad
    - Operación por archivo
    - Operación por conjunto de archivos
  - Resolución de conflictos
    - Locking
    - Merge-before-commit
    - Commit-before-merge

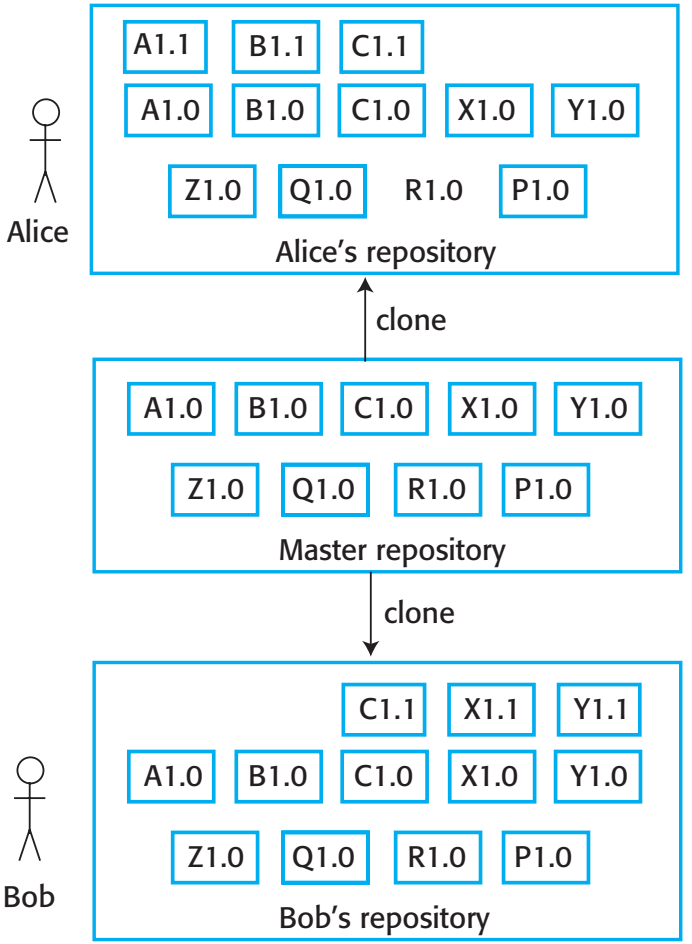
VCS - Centralizado

- Punto de fallo único
  - Si el repositorio falla detiene el trabajo
- Operación solo en modo conectado
  - Si el repositorio está offline no se puede trabajar
- En la primera generación, solo operaban de manera local
- En la segunda generación, se soporta modelo cliente-servidor



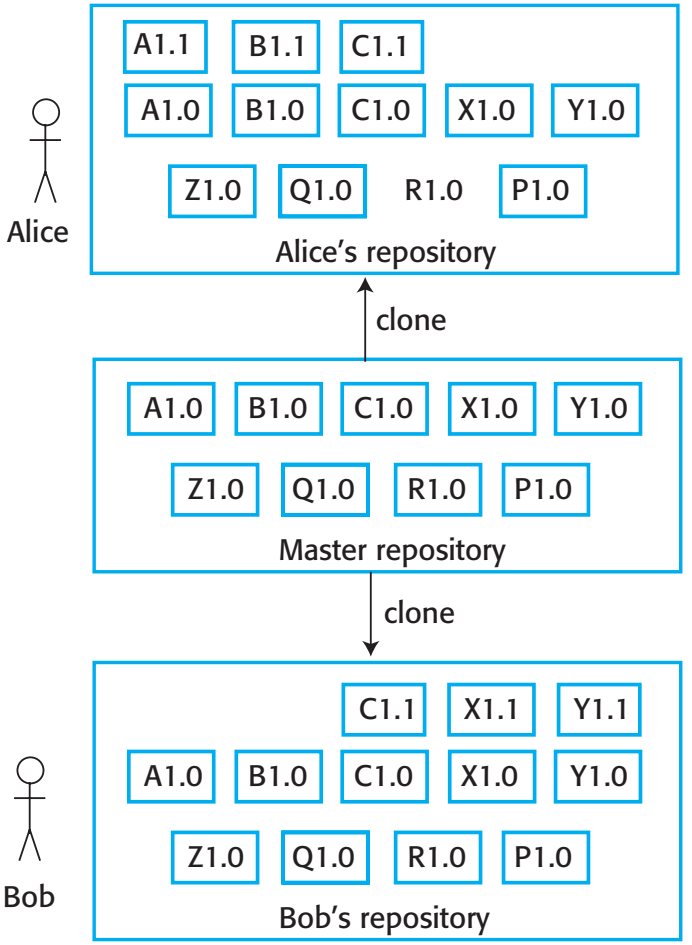
VCS - Distribuido

- Copia principal → servidor que mantiene el código producido por el equipo de desarrollo
- Desarrolador →
  - crea un clon
  - instalado en la máquina del desarrollador de manera local
  - Trabaja en los archivos y mantienen las nuevas versiones en su repositorio privado en su computadora
  - Hacen commit de los cambios y actualizan su repositorio privado
  - Hace push a un repositorio remoto de los cambios y por ejemplo sincronizar con el servidor principal del equipo

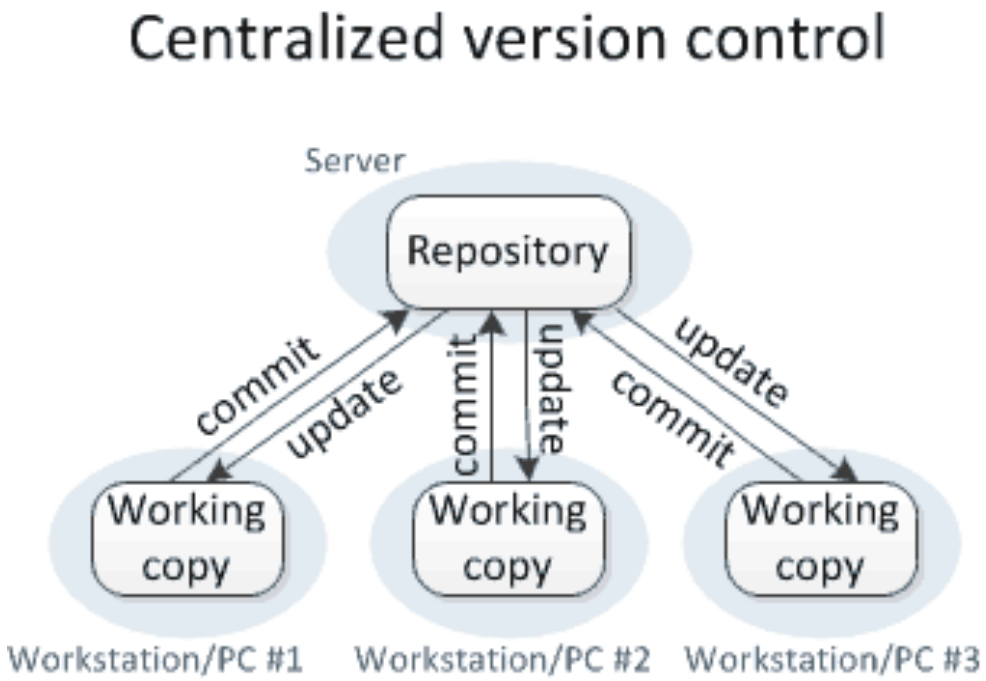


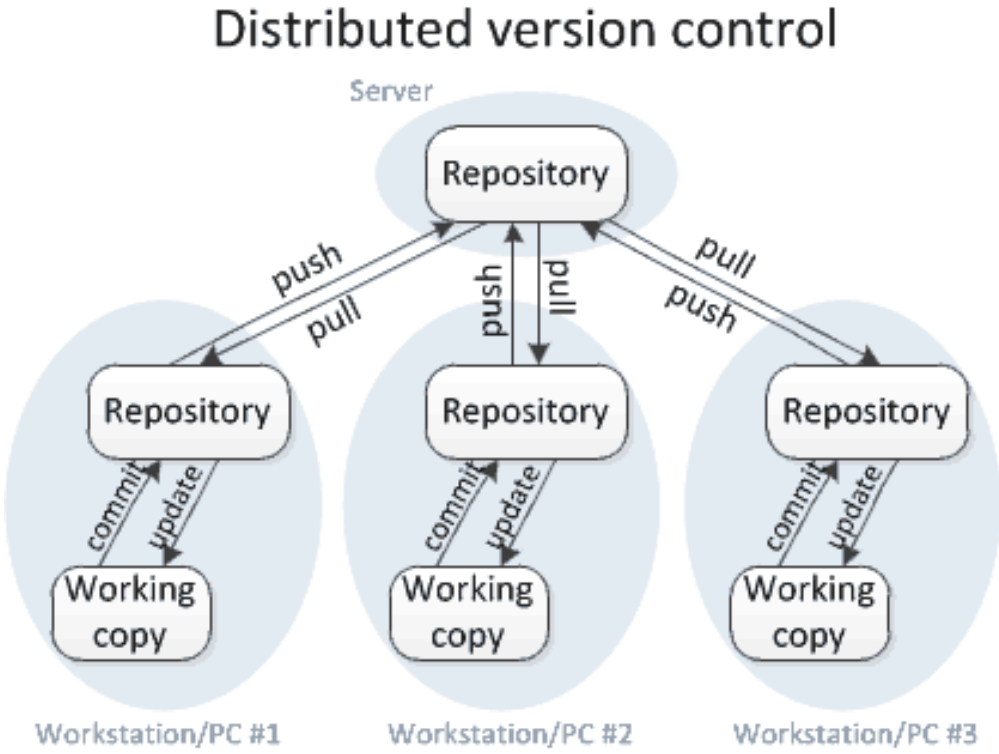
VCS - Distribuido

- Beneficios
  - Provee un mecanismo de backup
  - Existen varias copias del repositorio, se puede continuar el desarrollo incluso si el repositorio principal se corrompe
  - Permite el trabajo off-line, desconectado, sin conexión de red



Distributed vs Centralized flow





Atomicidad

- Operationes por archivo
  - En las primeras generaciones
  - Cada file tiene su propio master de cambios
  - Desventajas
    - Cambios que afectan varios files al mismo tiempo no se guardan de manera atómica
    - Los comentarios no se pueden asociar a un conjunto de archivos
- Operaciones por conjunto de archivos
  - Cambios que requieren la modificación de varios archivos se tratan como unidad
  - Historia y comentarios asociados a un cambio y no a archivos individuales
  - Es importante para migrar cambios o retirarlos

Resolución de conflictos

- Locking
  - Archivos read-only
  - Se tiene que solicitar una copia para escritura del archivo a modificar al VCS (adquiere el lock)
  - Solo un developer puede tener una copia para escritura
  - Cuando se hace check-in se libera el lock
- Flujo esperado
  - Develop1 adquiere el lock de un archive foo.c y comienza a modificarlo.
  - Developer2, trata de modificar el archive foo.c, es notificado que developer1 tiene el lock del archivo y no puede hacerle check out.
  - Developer2 está bloqueado y no puede continuar. Se va a tomar una taza de café?.
  - Develper1 termina los cambios en foo.c, hace check-in y libera el lock en foo.c.
  - Developer2 regresa de tomar el café hace check out de foo.c y adquiere el lock.
- Robar el lock?
- Primer mecanismo de resolución de conflictos

Resolución de conflictos

- Merge-before-commit
  - Avisa que el archivo ha sido modificado desde el checkout y hay que mergear los cambios antes de poder hacer commit
- Flujo esperado
  - Developer1 hace checkout del archivo foo.c y comienza a modificarlo
  - Developer2 hace checkout del archivo foo.c y comienza a modificarlo
  - Developer1 termina los cambios y hace checkin
  - Developer2 termina los cambios y cuando intenta hacer checkin, el VCS le dice que la version en el repositorio ha cambiado y que debe resolver los conflictos antes de proceder
  - Developer2 mergea los cambios que hizo con los cambios que hizo developer1 y que ya están en el repositorio
  - Los cambios de Developer1 y de Developer2 no se superponen
  - El merge es exitoso y el VCS le permite a Developer2 hacer commit de la version mergeada
- Segundo mecanismo de resolución de conflictos
- Empiricamente es raro que los cambios entren en conflicto si se los merges se realizan de manera frecuente

Resolución de conflictos

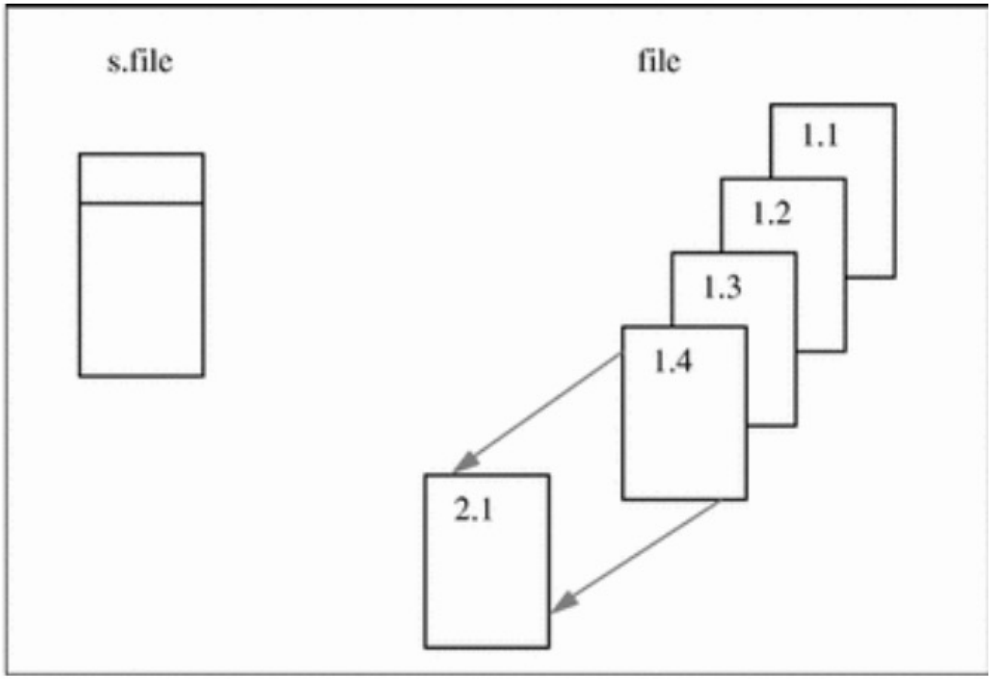
- Commit-before-merge
  - VCS nunca bloquea un commit
  - Si la version del repo ha cambiado desde el checkout se crea un nuevo branch
  - Los cambios pueden mergearse después
- Modelo de Desarrollo fluido
- Util para experimentación

Generaciones de VCS

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	CVS, SourceSafe, Subversion, Team Foundation Server
Third	Distributed	Changesets	Commit before merge	Bazaar, Git, Mercurial

Generaciones de VCS - SCCS

- Caracterísitcas
  - Inventó el concepto de control de versions
  - locking
  - Operación por archive
  - Centralizado
  - Deltas
- Ventajas
  - Interfaz de commando mejorada
    - Inventó terminología actual
  - Más rápido para acceder a versions más recientes
- Desventaja
  - Deltas por archivo y no por conjunto de archivos (imitando a sccs)
  - No networking



Generaciones de VCS - SCCS

- Caracterísitcas
  - Inventó el concepto de control de versions
  - locking
  - Operación por archive
  - Centralizado
  - Deltas
- Ventajas
  - Interfaz de commando mejorada
    - Inventó terminología actual
  - Más rápido para acceder a versions más recientes
- Desventaja
  - Deltas por archivo y no por conjunto de archivos (imitando a sccs)
  - No networking

Example session

```
$ echo "This is a one-liner file." >foo.txt   # Create our example file

$ admin -i foo.txt s.foo.txt            # Register it into SCCS

$ get -e s.foo.txt                      # Check out a writable copy of foo
Retrieved:
1.1
new delta 1.2
1 lines

echo "Adding a second line" >>foo.txt   # Modify contents

$ delta s.foo.txt                       # Check in the change
comments? Example change
No id keywords (cm7)
1.2
1 inserted
0 deleted
1 unchanged

$ get -e s.foo.txt                     # Check out an editable copy again
Retrieved:
1.2
2 lines
No id keywords (cm7)

$ unget s.foo.txt                      # Revert foo.txt to repository cont
1.2

$ prs s.foo.txt                          # Display changelog of foo.txt
D 1.2 07/12/26 10:24:16 esr 2 1 00001/00000/00001
MRs:
COMMENTS:
Example change

D 1.1 07/12/26 10:21:05 esr 1 0 00001/00000/00000
MRs:
COMMENTS:
date and time created 07/12/26 10:21:05 by esr

$ admin  -fb s.foo.txt                  # Enable branching in our sample m

$ get -b -e -r1.2 s.foo.txt             # Get writeable copy, creating a b
1.1
new delta 1.2.1.1
2 lines
```

Generaciones de VCS - SCCS

- Caracterísitcas
  - Inventó el concepto de control de versions
  - locking
  - Operación por archive
  - Centralizado
  - Deltas
- Ventajas
  - Interfaz de commando mejorada
    - Inventó terminología actual
  - Más rápido para acceder a versions más recientes
- Desventaja
  - Deltas por archivo y no por conjunto de archivos (imitando a sccs)
  - No networking

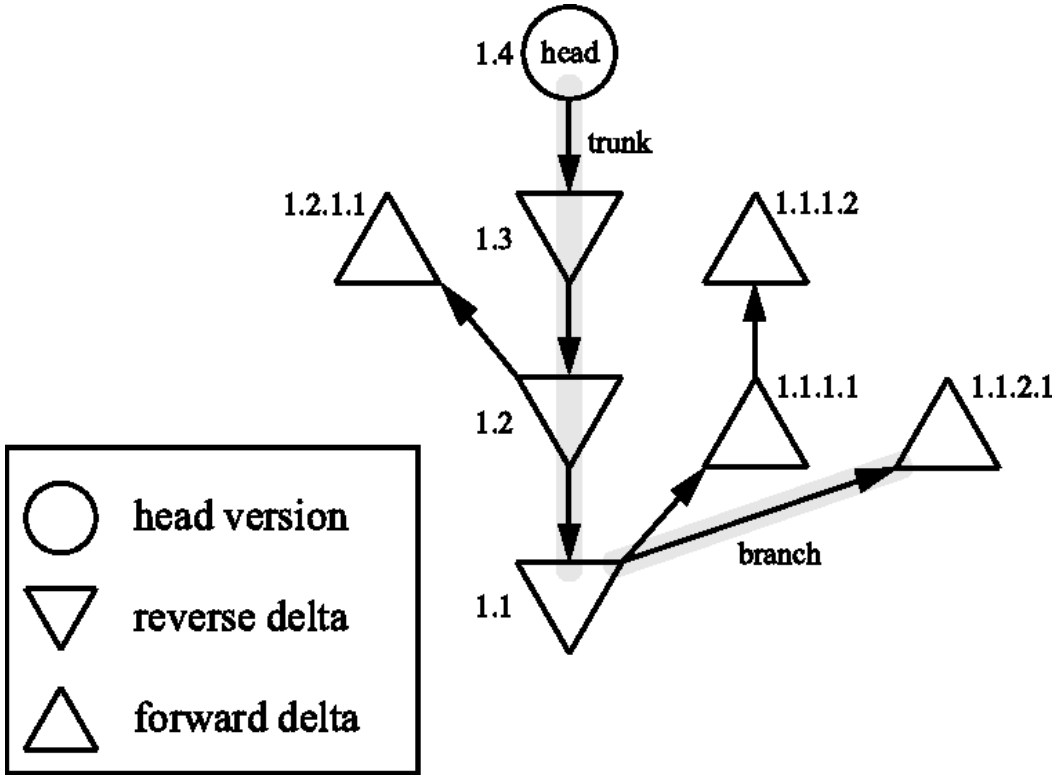
Keeping SIDs Consistent Across Files

With some care, it is possible to keep the SIDs consistent across sources composed of multiple files. The trick here is to edit all the files at once. The changes can then be made to whatever files are necessary. Check in all the files (even those not changed). This can be done fairly easily by specifying the SCCS subdirectory as the file name argument to both edit and delta:

```
$ sccs edit SCCS
.  .  .
$ sccs delta SCCS
```

Generaciones de VCS - RCS

- Caracterísiticas
  - locking
  - Operación por archivo
  - Centralizado
  - Deltas invertidos
- Ventajas
  - Liviano
  - Fácil de instalar y usar
- Desventaja
  - No soporta networking (esperable para la época)
  - Locking (esperable también)
  - Deltas por archivo y no por conjunto de archivos



Generaciones de VCS - RCS

- Caracterísiticas
  - locking
  - Operación por archivo
  - Centralizado
  - Deltas invertidos
- Ventajas
  - Liviano
  - Fácil de instalar y usar
- Desventaja
  - No soporta networking (esperable para la época)
  - Locking (esperable también)
  - Deltas por archivo y no por conjunto de archivos

Example session

```
$ echo "This is a one-liner file." >foo.txt   # Create our example file

$ ci -u foo.txt                              # Register it into RCS
RCS/foo.txt,v  <--  foo.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> a boring log message
>> .
initial revision: 1.1
done

$ co -l s.foo.txt                            # Check out a writable copy of foo.txt
RCS/foo.txt,v  -->  foo.txt
revision 1.1 (locked)
done

echo "Adding a second line" >>foo.txt   # Modify contents

$ ci -u foo.txt
RCS/foo.sh,v  <--  foo.sh
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Example change.
>> .
done

$ rcs -nTAG_ME:1.2  foo.txt              # Give r1.2 the symbolic name TAG.ME
RCS file: RCS/foo.txt,v
done

$ rlog foo.txt                           # Display the revision log

RCS file: RCS/foo.txt,v
Working file: foo.txt
head: 1.2
branch:
locks: strict
access list:
symbolic names:
        TAG_ME: 1.2
keyword substitution: kv
total revisions: 2;     selected revisions: 2
description:
a boring log message
----------------------------
revision 1.2
date: 2007/12/26 16:00:42;  author: esr;  state: Exp;  lines: +1 -0
Example change.
----------------------------
revision 1.1
date: 2007/12/26 15:59:40;  author: esr;  state: Exp;
Initial revision
```

Generaciones de VCS - Subversion

- Caracterísiticas
  - Merge-before-commit
  - Operación por conjunto de archivos
  - Centralizado
  - Client-server model
- Ventajas
  - Commits files-set based and atomic
  - Merge-before-commit (mejor que locks)
- Desventaja
  - El modelo centralizado imposibilita el trabajo desconectado

Example session

As with the CVS session, I use -m here to specify commit messages rather than having SVN run an editor to collect them.

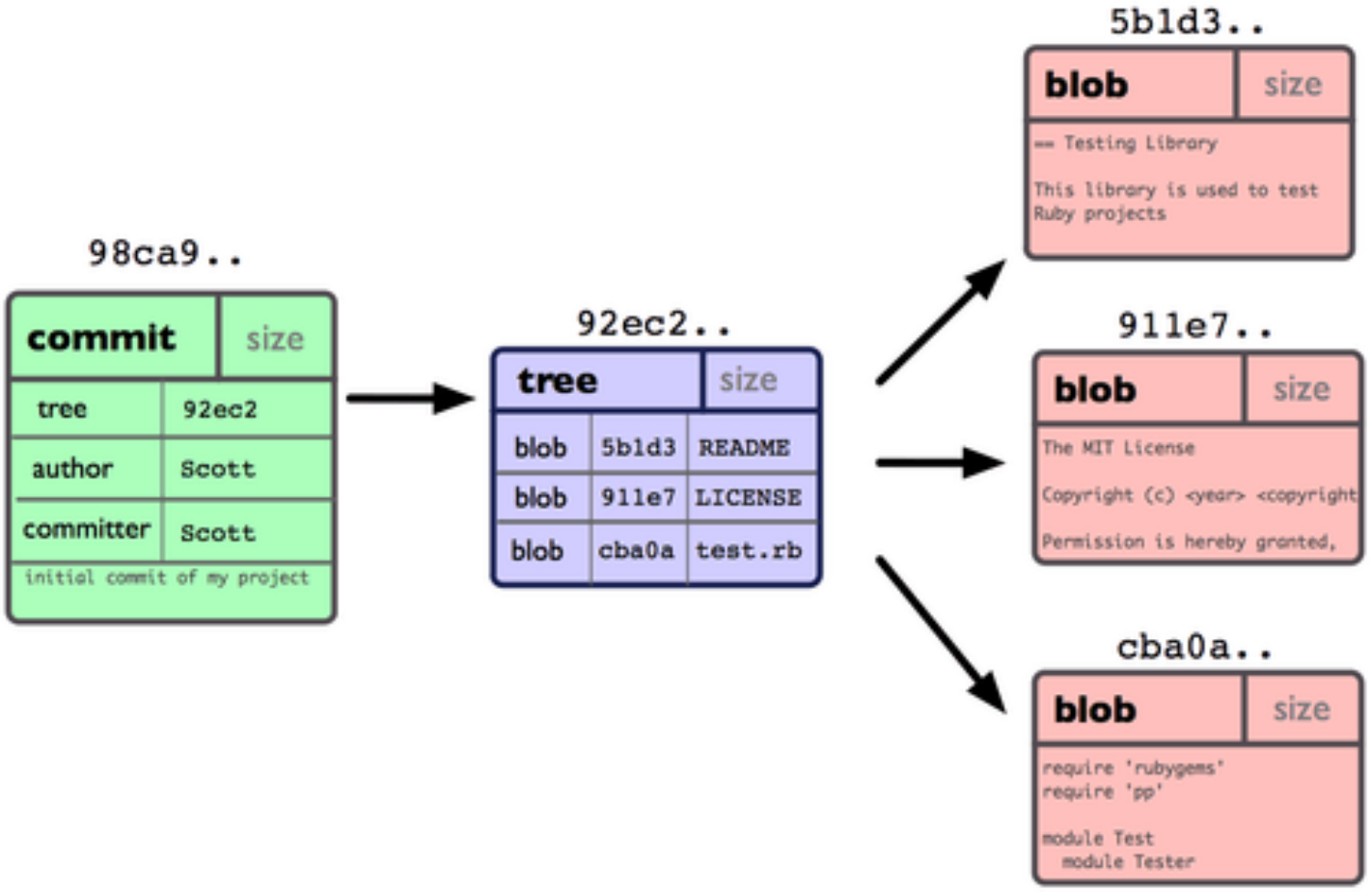
```
echo "This is a one-liner file." >foo.txt   # Create our example file

$ svnadmin create SVN            # Create empty repo in subdirectory SVN

$ svn checkout file://$PWD/SVN . # Check out a working copy here.
Checked out revision 0.

$ svn add foo.txt                # Schedule foo.txt to be added to the repo
A         foo.txt

$ svn -m "First commit" commit   # Commit the addition
Adding         foo.txt
Transmitting file data .
Committed revision 1.

$ svn status                     # No output means no local modifications

$ echo "Adding a second line" >>foo.txt

$ svn status                     # Now we see that foo.txt is locally modified
M       foo.txt

$ svn diff                       # We examine the differences
Index: foo.txt
===================================================================
--- foo.txt     (revision 1)
+++ foo.txt     (working copy)
@@ -1 +1,2 @@
 This is a one-liner file.
+Adding a second line

$ svn commit -m "Second commit"  # We commit the change
Sending        foo.txt
Transmitting file data .
Committed revision 2.

$ svn log foo.txt                # We examine the revision log
------------------------------------------------------------------------
r2 | esr | 2007-12-28 06:37:46 -0500 (Fri, 28 Dec 2007) | 1 line

Second commit
------------------------------------------------------------------------
r1 | esr | 2007-12-28 06:12:42 -0500 (Fri, 28 Dec 2007) | 1 line

First commit
------------------------------------------------------------------------

$ svn rename foo.txt bar.txt      # This is what a rename looks like
A         bar.txt
D         foo.txt

$ svn commit -m "Example rename"  # Commit the rename so it actually happens
Adding         bar.txt
Deleting       foo.txt

Committed revision 3.

$ svn log                         # History is preserved across the rename
------------------------------------------------------------------------
esr@snark:~/WWW/writings/version-control$ svn log bar.txt
------------------------------------------------------------------------
r3 | esr | 2007-12-28 06:48:40 -0500 (Fri, 28 Dec 2007) | 1 line

Example rename
------------------------------------------------------------------------
r2 | esr | 2007-12-28 06:37:46 -0500 (Fri, 28 Dec 2007) | 1 line

Second commit
------------------------------------------------------------------------
r1 | esr | 2007-12-28 06:12:42 -0500 (Fri, 28 Dec 2007) | 1 line

First commit
------------------------------------------------------------------------
```

Generaciones de VCS - Git

- Caracterísiticas
  - Commit-before-merge
  - Operación por conjunto de archivos
  - Distribuido
- Ventajas
  - Commits files-set based and atomic
  - Commit-before-merge
  - Posibilita el trabajo desconectadp

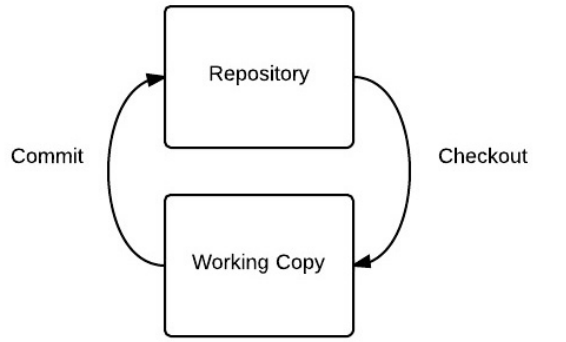
Generaciones de VCS - Git

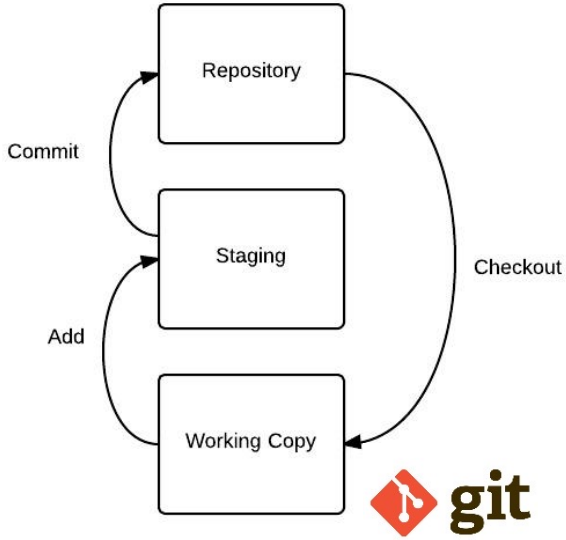
- Estructura

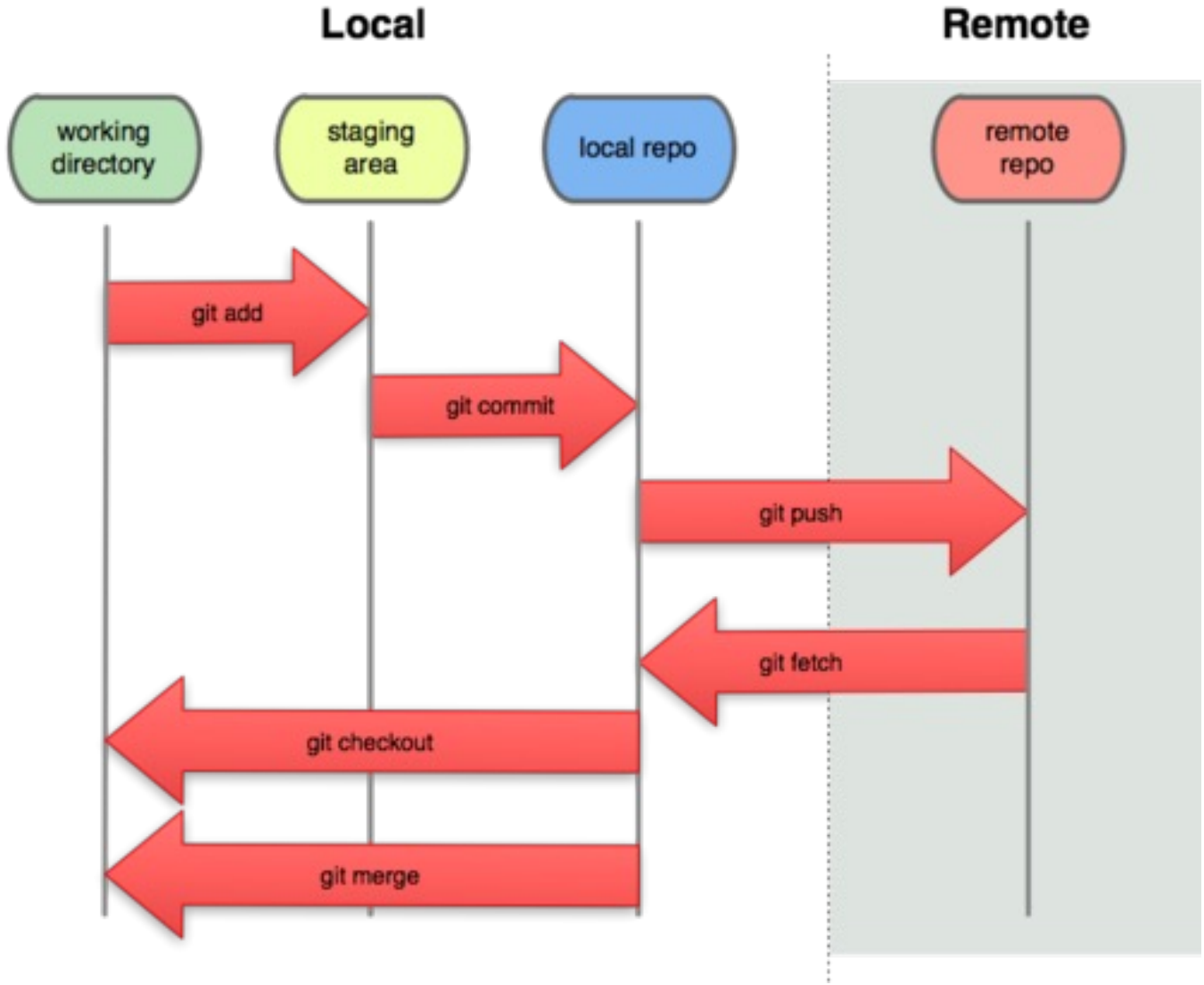


https://en.wikipedia.org/wiki/Merkle_tree

2-tree vs 3-tree architecture VCS

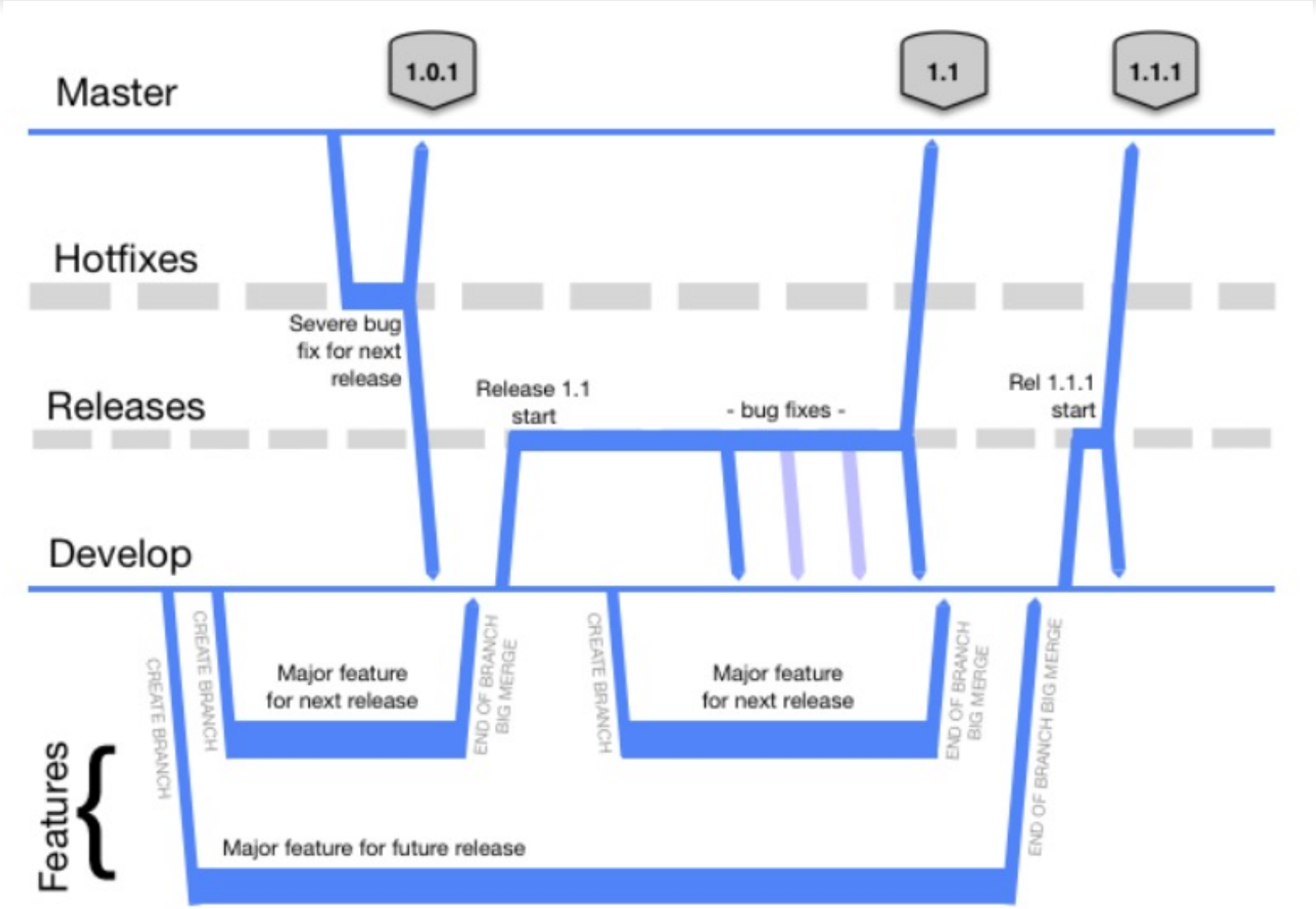






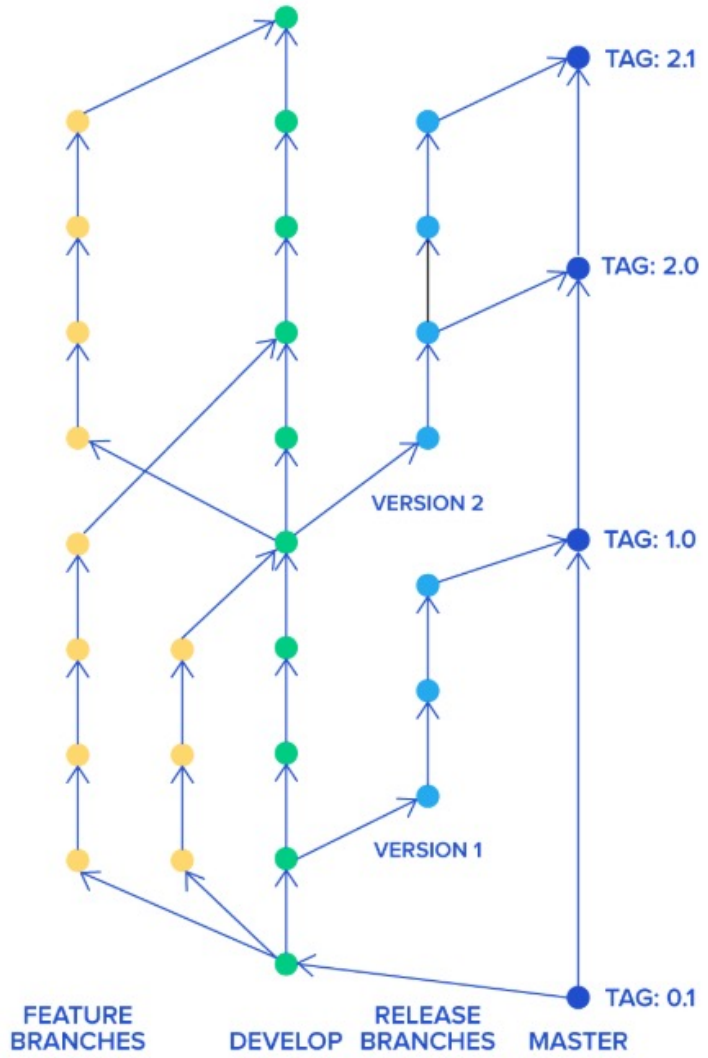
VCS Flows

Branching Models - Gitflow



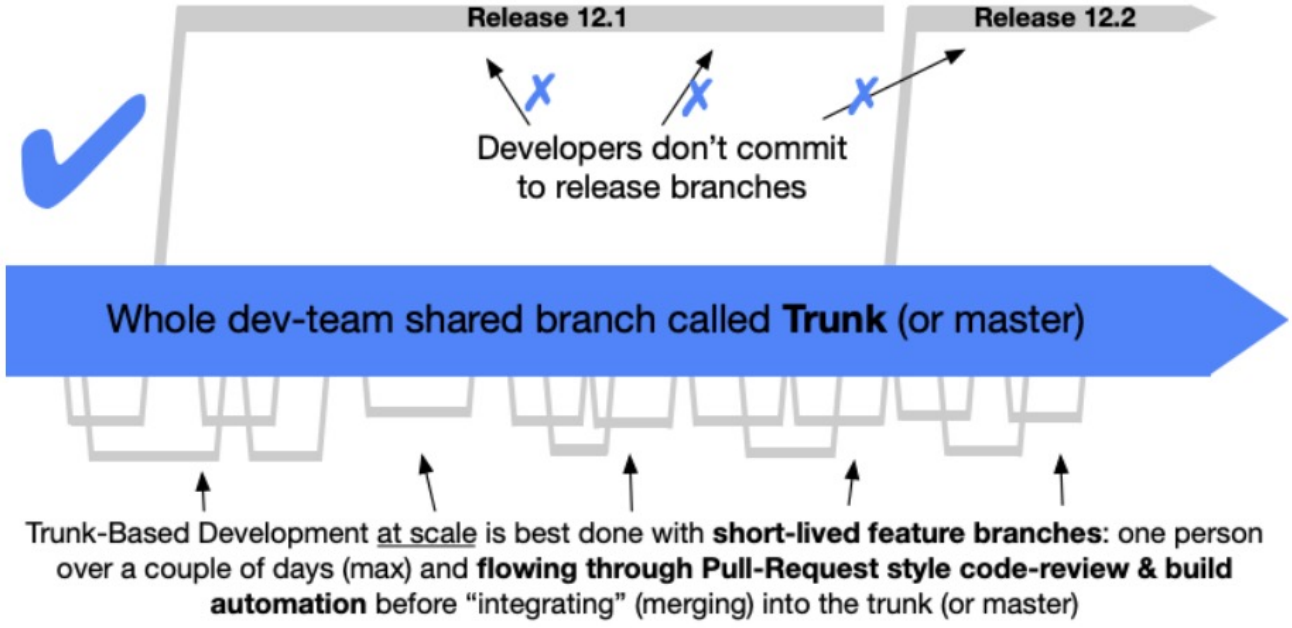
Branching Models - Gitflow

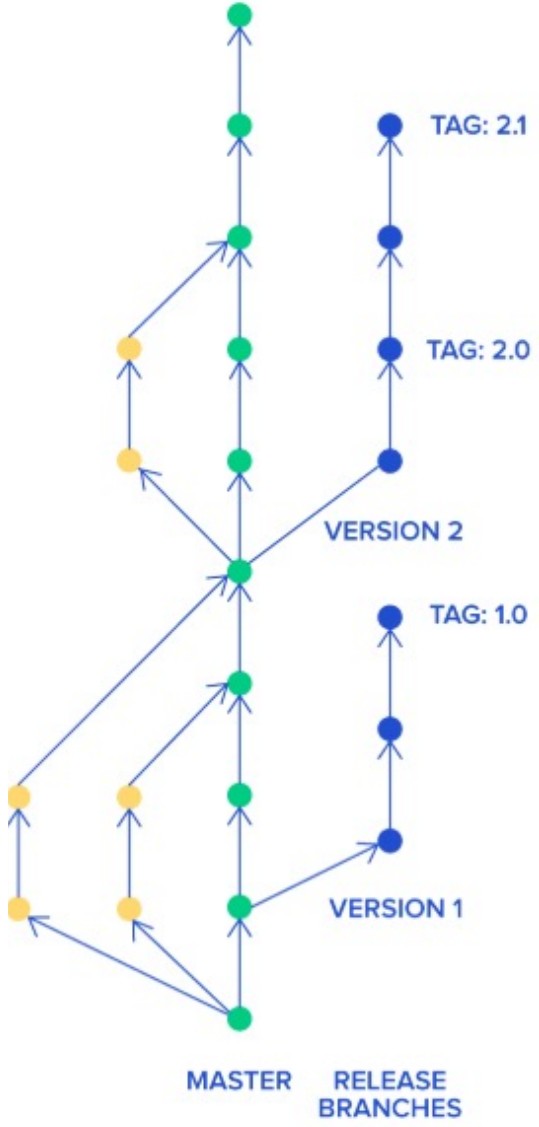
- Recomendado
  - Cuando el equipo no tiene experiencia
  - Muchos junior developers
  - Cuando el producto es estable
- No recomendado
  - En los start ups
  - Cuando se requiere iterar rápidamente
  - Cuando los desarrolladores son en su mayoría senior



Branching Models – Trunk based development

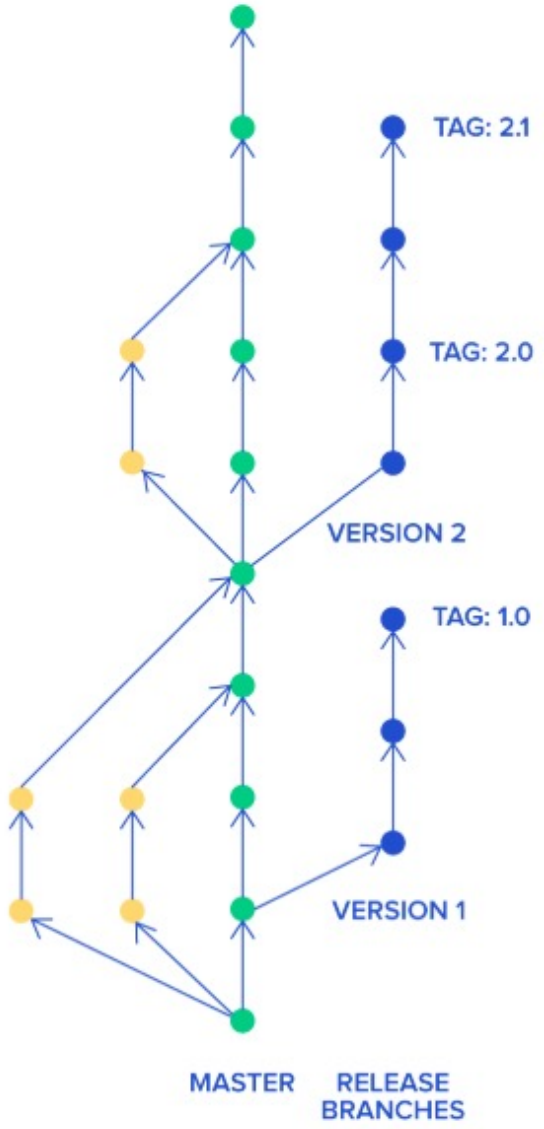
- Características
  - Más flexible
  - Escala major
  - Permite hacer integración continua CI



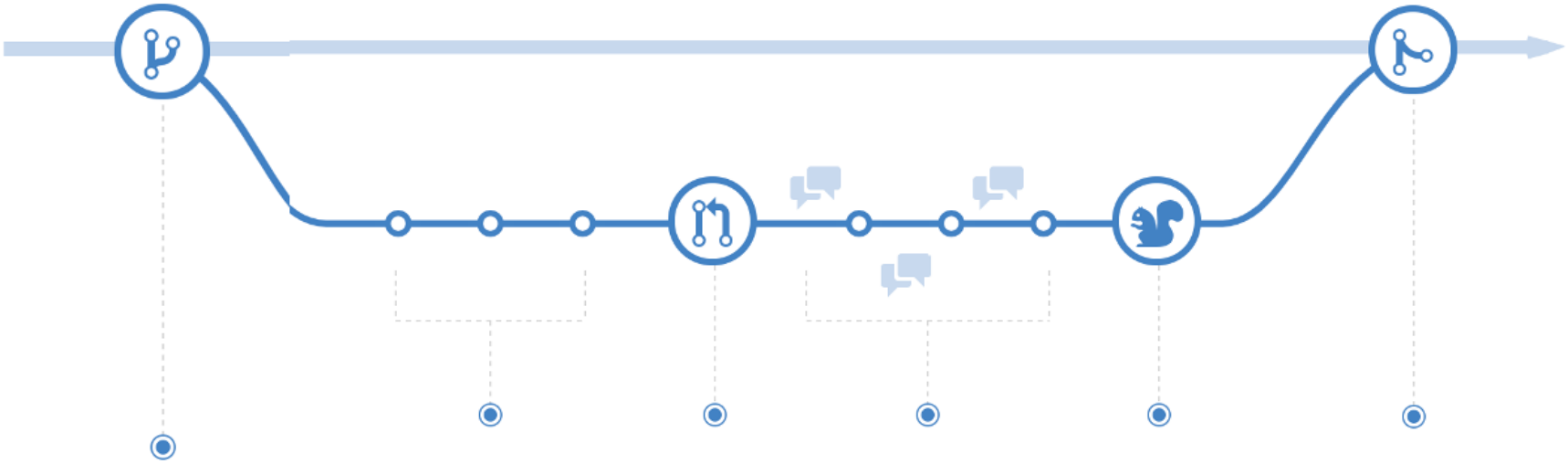


Branching Models – Trunk based development

- Recomendado
  - En los start ups
  - Cuando se requiere iterar rápidamente
  - Cuando los desarrolladores son en su mayoría senior
- No recomendado
  - Cuando el equipo no tiene experiencia
  - Muchos junior developers
  - Cuando el producto es estable



Github flow



https://guides.github.com/introduction/flow/

Preguntas?