# Introducción a los Repositorios de Código Distribuido

*Alteración de Commits*

Diego Madariaga

# Contenidos de la clase

1. Precauciones al alterar el historial
2. Uso de git reset
3. Uso de git cherry-pick
4. Uso de git revert

# 1.

# Precauciones al alterar el historial

# ¿Por qué modificar commits?

- Arreglar un problema antes de que genere un gran impacto
- Desagrupar un gran cambio en una serie de cambios pequeños
- Agrupar cambios en un commit más grande
- Reordenar commits
- Remover código incluido accidentalmente en un commit

# ¿Por qué modificar commits?

- En general, el historial de commits puede ser modificado si hay una buena razón para hacerlo

# FIlosofías respecto a alterar historial

- “Historial realista”: Nada se altera
  - Provee detalles “arqueológicos” del desarrollo
  - Explicación de aparición y solución de errores
  - Permite analizar el trabajo de equipos de desarrollo

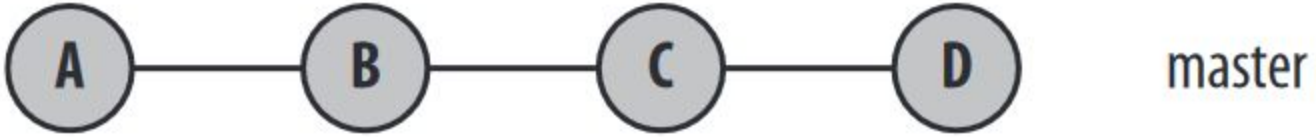
- “Historia ideal”: El historial es limpiado
  - Se pierden pasos intermedios en el desarrollo
  - Muestra de forma clara los pasos secuenciales en el desarrollo
  - Se evitan versiones intermedias inestables

# Precauciones al alterar el historial

- Como regla general: Hay libertad para alterar o mejorar el historial mientras nadie tenga una copia de dicho historial
  - Repositorio completo
  - Rama dentro de un repositorio distribuido

# Ejemplo de alteración del historial

Historial publicado



# Ejemplo de alteración del historial

- Se agregan nuevos cambios de forma local (historial no publicado)

# Ejemplo de alteración del historial

- Se agregan nuevos cambios de forma local (historial no publicado)
- Podemos hacer cualquier alteración en el historial entre W y Z, con el fin de tener un mejor historial en ese intervalo

# Ejemplo de alteración del historial

- Se agregan nuevos cambios de forma local (historial no publicado)
- Podemos hacer cualquier alteración en el historial entre W y Z, con el fin de tener un mejor historial en ese intervalo:
- Modificación, combinación, reordenamiento, agregación de commits

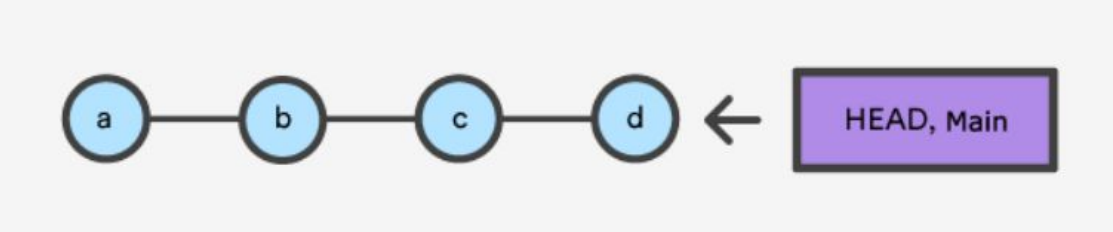
# 2.

# Uso de git reset

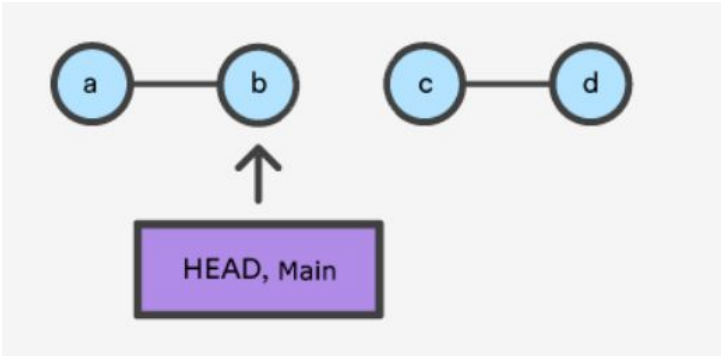
# git reset

- Comando para cambiar la referencia HEAD a un cierto commit
- Se utiliza para deshacer cambios
- Es considerado un comando “destructivo”

# git reset




```
git reset b
```



# git reset

- Tiene 3 opciones principales
  - git reset --soft *commit*
  - git reset --mixed *commit*
  - git reset --hard *commit*

# git reset --soft *commit*

- Cambia la referencia HEAD hacia *commit*.
- El contenido del índice y el directorio de trabajo no se alteran
- Es la versión de git reset con menos efectos

# git reset --mixed *commit*

- Cambia la referencia HEAD hacia *commit*.
- El contenido del índice es modificado, sincronizandose con el Tree referenciado por *commit*
- El directorio de trabajo no se altera
- Es la versión por default de git reset

# git reset --hard *commit*

- Cambia la referencia HEAD hacia *commit*.
- El contenido del índice y el directorio de trabajo son modificados, sincronizandose con el Tree referenciado por *commit*
- La estructura de directorios es restaurada completamente, por lo que cualquier revierte cualquier modificación y se eliminan los  archivos nuevos

# git reset (resumen)

| Opción | HEAD | Índice | Directorio de trabajo |
|---|---|---|---|
| `--soft` | ✓ | | |
| `--mixed` | ✓ | ✓ | |
| `--head` | ✓ | ✓ | ✓ |

# 3.
# Uso de git cherry-pick

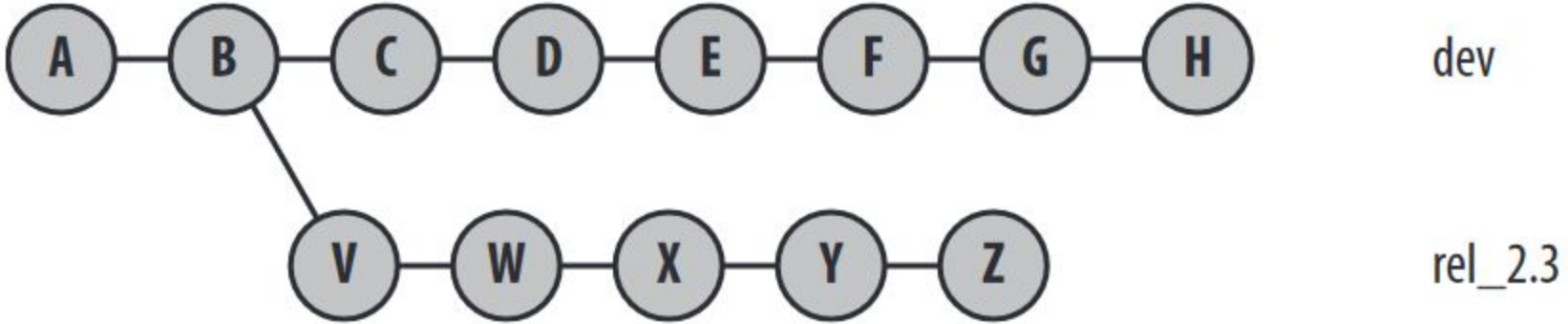
# git cherry-pick *commit*

- Comando para aplicar los cambios introducidos en *commit* en la rama actual
- Estrictamente hablando, no altera el historial, más bien agrega algo al historial (ya que introduce un commit)
- Se utiliza para introducir cambios desde una rama de desarrollo a una rama de mantención

# git cherry-pick *commit*

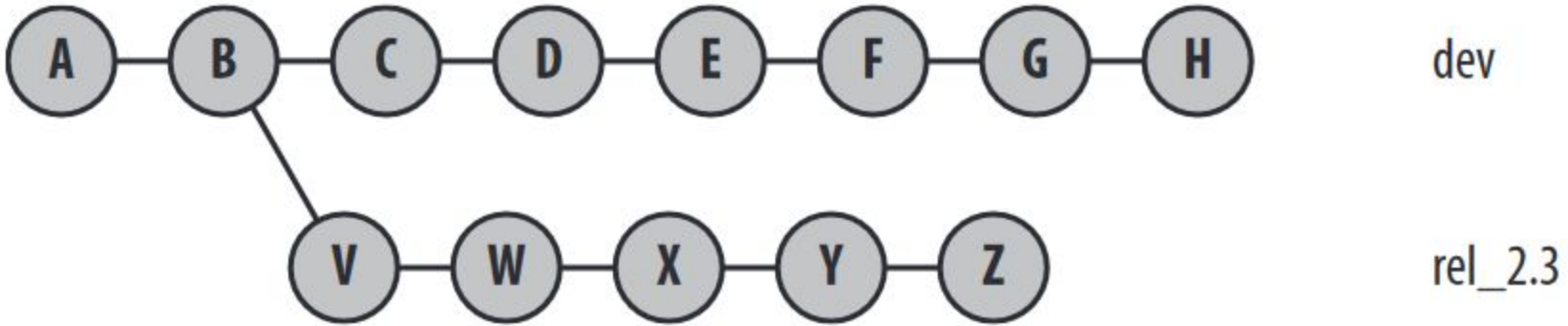
**dev:** Rama de desarrollo

**rel_2.3:** Rama de mantención de la versión 2.3



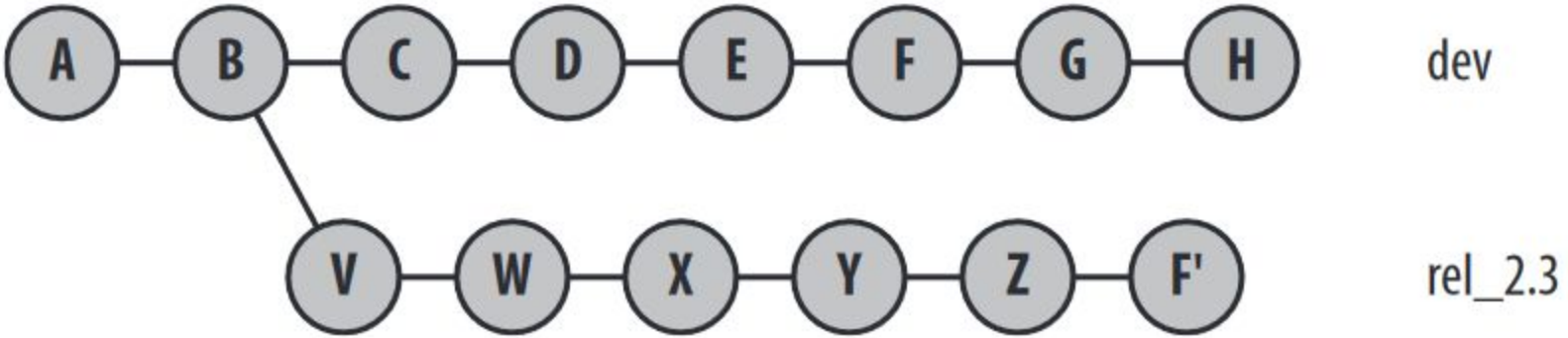
# git cherry-pick *commit*

- Se arregla un bug en F de la rama dev.
- El error también está presente en rel_2.3
- La solución del bug en F, se introduce en rel_2.3
    - `git cherry-pick F`



# git cherry-pick *commit*

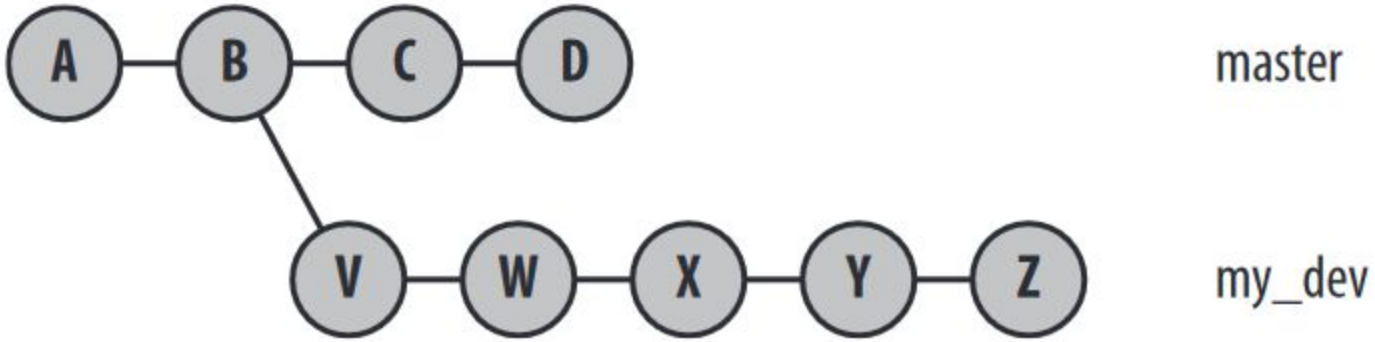
- Se arregla un bug en F de la rama dev.
- El error también está presente en rel_2.3
- La solución del bug en F, se introduce en rel_2.3
    - `git cherry-pick F`



# git cherry-pick *commit*

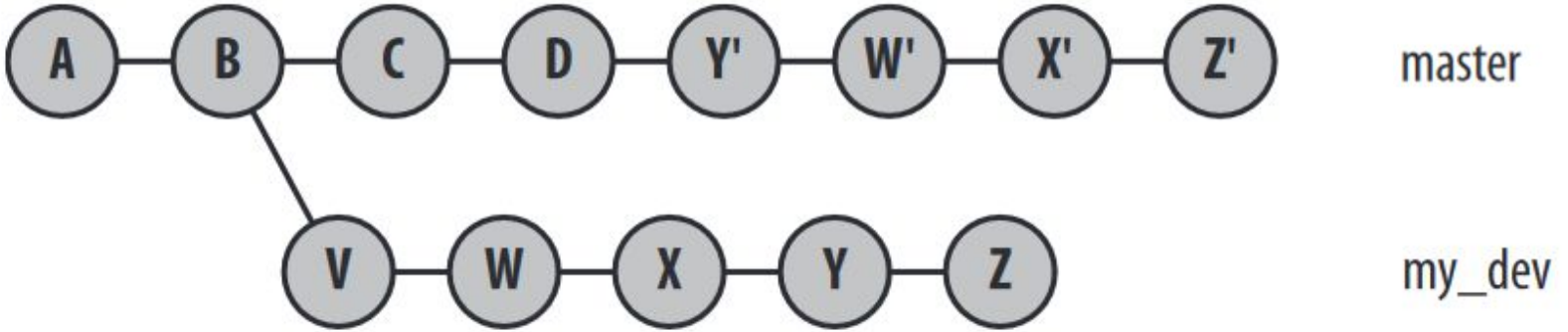
**my_dev:** Rama de desarrollo personal

- Se quieren enviar los cambios de my_dev a master pero en un orden particular (Y, W, X, Z)



# git cherry-pick *commit*

```
git checkout master
git cherry-pick Y
git cherry-pick W
git cherry-pick X
git cherry-pick Z
```



# 4.
# Uso de git revert

# git revert *commit*

- Comando similar a git cherry-pick, pero aplica lo “inverso” a los cambios de *commit*
- La idea es revertir los cambios hechos en un determinado commit (generalmente muy atrás en el historial)
- Al igual que git cherry-pick no altera el historial, más bien agrega algo al historial (ya que introduce un commit)

# git revert *commit*

- En el historial de un repositorio, por alguna razón, el commit D en rama master es defectuoso
  - Una solución sería revertir el commit manualmente, especificando el por qué en el mensaje de commit
  - Una solución más simple: `git revert D`

# git revert *commit*

- En el historial de un repositorio, por alguna razón, el commit D en rama master es defectuoso
  - Una solución sería revertir el commit manualmente, especificando el por qué en el mensaje de commit
  - Una solución más simple: `git revert D`

# Introducción a los Repositorios de Código Distribuido

*Alteración de Commits*

Diego Madariaga