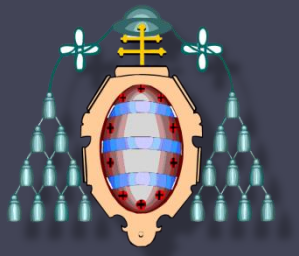
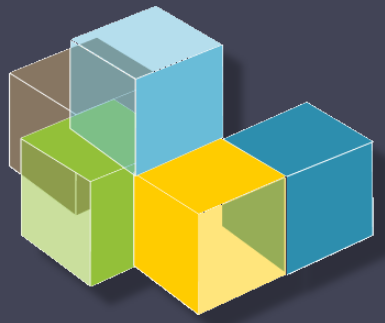
ES
Español



Universidad de Oviedo

Escuela de Ingeniería Informática



ARQUITECTURA DEL SOFTWARE

Arquitectura del Software

Clases de teoría 1
Esquema de la clase

Escuela de Ingeniería Informática

2016-17

Jose Emilio Labra Gayo

Esquema

- Definiciones de arquitectura básicas
  - Stakeholders, atributos de calidad, restricciones
- Herramientas de construcción
  - Control de versiones
    - Git básico
  - Automatización de la construcción
    - Maven básico
      - Estructura del código fuente
      - Tareas: clean, compile, test, site
  - Test-driven development
    - Pruebas unitarias y pruebas de integración
    - Integración continua

¿Qué es arquitectura del software?

Estructura básica del sistema

"Decisiones de diseño principales del sistema"

Si hay que cambiarlas ⇒ Coste elevado

¿Cómo se diseña?

Solución de compromiso entre…

Creatividad
Divertido
Arriesgado
Puede ofrecer soluciones nuevas
Puede ser innecesario

Método
Eficiente en terrenos familiares
Resultado predecible
No siempre es lo mejor
Técnicas de calidad contrastada

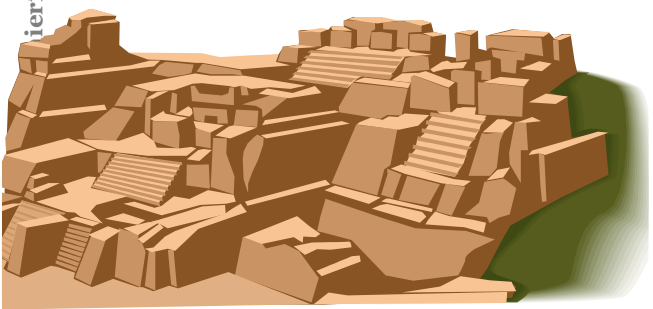
Arquitecto

Arquitecto del software

La disciplina evoluciona

Arquitecto debe conocer:

- Avances en técnicas de construcción
- Estilos y patrones
  - Mejor herramienta = experiencia (*no silver bullet*)
    - Experiencia propia
    - Experiencia de la comunidad



Arquitecto

Modelado de arquitectura

Stakeholders

Atributos de calidad

Restricciones

Experiencia de la comunidad

Principios
Patrones
Estilos
Antipatrones

Arquitecto

Arquitectura

Tecnología

Atributos de calidad

- Especifica lo bien que un sistema se comporta respecto a alguna medida
  - También conocidos como requisitos no-funcionales
    - No se refieren a un comportamiento específico
    - La mayoría terminan en "-idad"
      - Seguridad, Desplegabilidad, Modificabilidad, Accesibilidad, Fiabilidad, Escalabilidad, usabilidad, ...
    - No todos terminan en "-idad":
      - Rendimiento, Tolerancia a fallos, Precio, ...
  - ISO 25010: lista algunos requisitos no-funcionales

Restricciones

- Decisiones de diseño pre-especificadas
  - Técnicas u organizativas
- Reducen el espacio de posibles arquitecturas en las cuales buscar una solución
  - Muy poco software tiene libertad total
  - Ejemplos:
    - Marcos de aplicaciones (frameworks)
    - Lenguajes de programación, ...
  - Las restricciones vienen impuestas

El arquitecto toma decisiones de diseño para alcanzar atributos de calidad dentro de unas restricciones

Desarrollo de software

Desarrollo de software

- Algunos atributos de calidad
- Gestionabilidad
  - Capacidad para gestionar el software que se construye
    - Diferentes versiones, configuraciones, etc.
    - Dependencias
    - Ciclo de vida (construcción, compilación, empaquetado)
- Testabilidad
  - Poder realizar pruebas del sistema

Herramientas

- Control de versiones (git…)
- Automatización de la construcción (maven…)
  - Estructura de directorios
  - Dependencias
  - Ciclo de vida del proyecto
    - Compilación, enlazado, empaquetado, etc.

Sistemas de control de versiones

- Centralizados
  - Un repositorio centralizado de todo el código
  - Ejemplos: CVS, Subversion,...
- Distribuidos
  - Cada usuario tiene su propio repositorio
  - Ejemplos: mercurial, git, ...

Construcción de software

- Gestión de configuraciones
  - ¿Dónde ponemos el software?
  - Control de versiones
- Automatización de la construcción
  - Estructura de directorios
  - Ciclo de vida del proyecto
    - Dependencias, Compilación, enlazado, empaquetado, etc.

Sistemas de control de versiones

- Centralizados
  - Un repositorio centralizado de todo el código
  - Ejemplos: CVS, Subversion,...
- Distribuidos
  - Cada usuario tiene su propio repositorio
  - Ejemplos: mercurial, git, ...

Git

Diseñado por Linus Torvalds (Linux), 2005

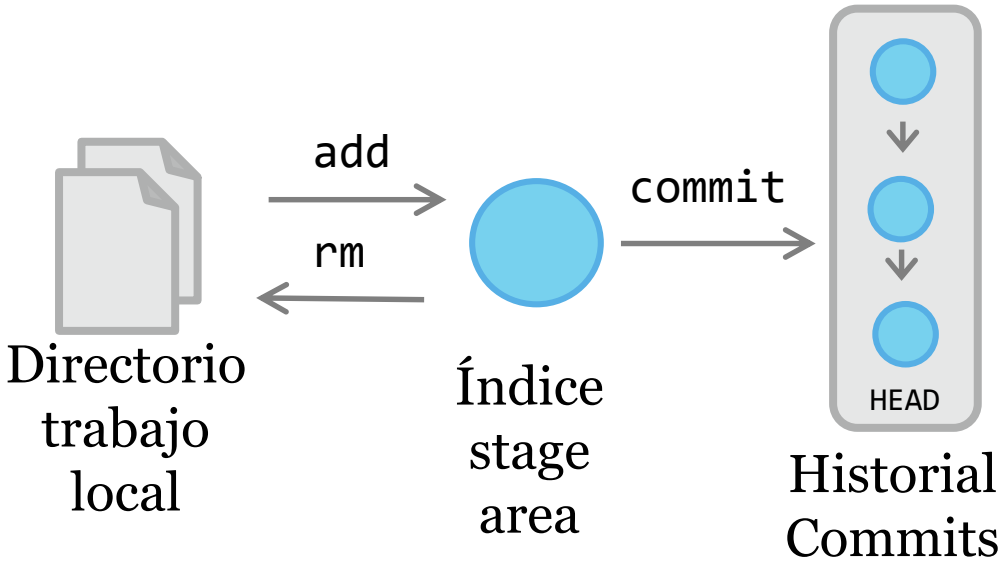
Objetivos:

- Aplicaciones con gran nº de archivos de código
  - Eficiencia
- Trabajo distribuido
- Cada desarrollador tiene su propio repositorio
  - Copia local de todo el historial de cambios
  - Es posible realizar commit's incluso sin conexión
- Desarrollo no lineal (ramificaciones)

Componentes locales

3 componentes locales:

- Directorio de trabajo local
- Índice: área de ensayo (stage). A veces también caché.
- Historial: Almacena versiones ó commits
  - HEAD (versión más reciente)



Ramas

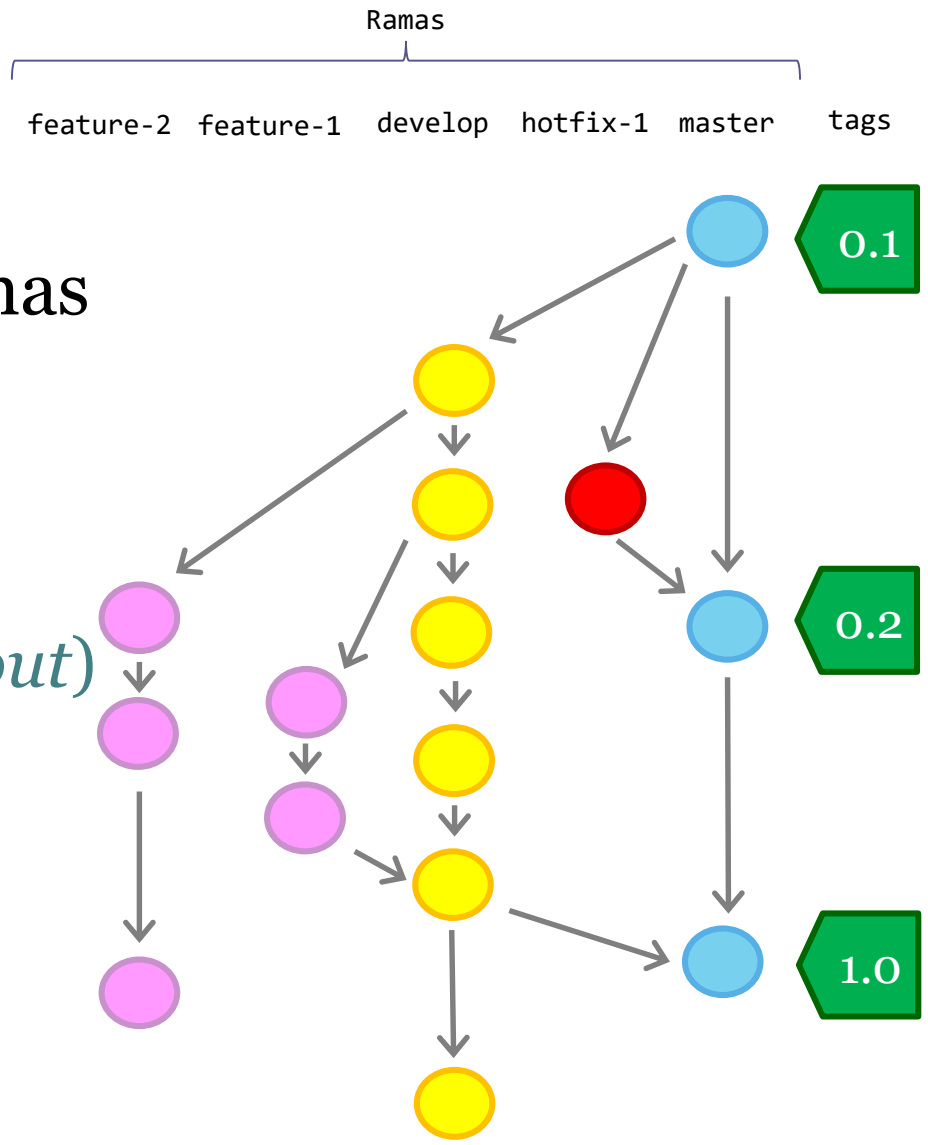
Git facilita gestión de ramas

- master = rama inicial

Operaciones:

- Crear ramas (*branch*)
- Cambiar a ramas (*checkout*)
- Combinar (*merge*)
- Etiquetar (*tag*)

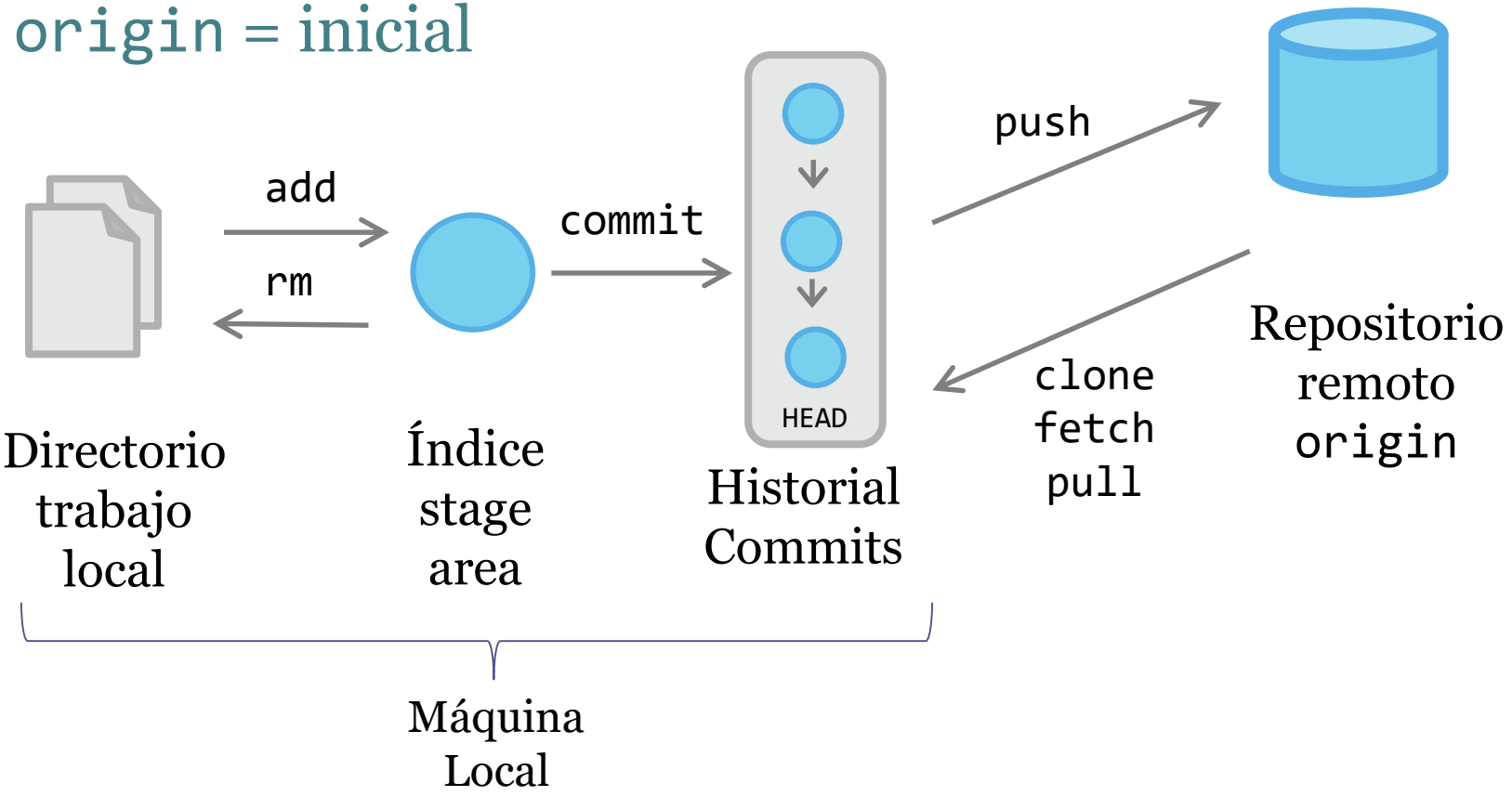
Múltiples estilos de ramificación



Repositorios remotos

Se pueden conectar con repositorios remotos

`origin` = inicial



Funcionamiento básico

```
init
clone
config
add
commit
status
log
diff
```

init - Crear repositorios

`git init`

Transforma el directorio actual en repositorio Git

Se crea directorio .git

Variantes:

`git init <directorio>`

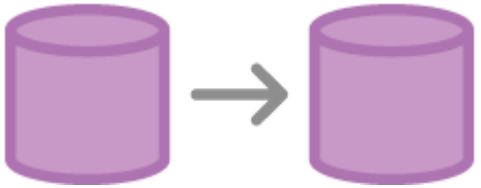
Crea un repositorio vacío en el directorio especificado

`git init --bare <directorio>`

Inicializa repositorio Git pero omite directorio de trabajo

NOTA: Normalmente, esta instrucción sólo se realiza una vez

clone - Clonar repositorios



```
git clone <repo>
```

Clonar el repositorio <repo> en la máquina local

<repo> puede estar en una máquina remota

Ejemplo:

```
git clone https://github.com/Arquisoft/ObservaTerra0.git
```

NOTA: Al igual que init, esta instrucción sólo se realiza una vez

config - Configurar git

```
git config --global user.name <name>
git config --global user.email <email>
```

- Declara el nombre/email de usuario
- Otras opciones de configuración:
  - merge.tool, core.editor, ...
- Ficheros de configuración:
  - <repo>/.git/config -- Específicos de repositorio
  - ~/.git/config -- Globales

add - Añadir al índice

```
git add <fichero>
git add <dir>
```

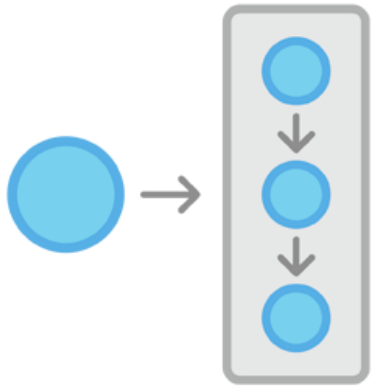
Añade fichero o directorio al índice

Variantes

`git add --all` = Añade/borra ficheros

El índice ó escenario almacena copias de los ficheros antes de ser incluidos en el historial

commit - Añadir al historial



```
git commit
git commit -m "mensaje"
```

- Añade los ficheros del índice al historial
  - Crea una nueva instantánea "snapshot" del proyecto
  - Cada instantánea tiene un identificador SHA1
    - Puede recuperarse posteriormente
    - Pueden asignarse etiquetas para facilitar su gestión

NOTA: Conviene excluir de control de versiones algunos ficheros
Ejemplos: binarios (*.class), temporales, configuración (.settings), privados (claves de Bases de datos...), etc.
Se incluyen en fichero: `.gitignore`

status - Observar índice

```
git status
```

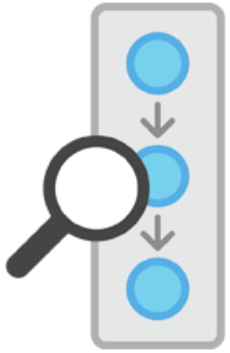
Muestra ficheros *staged*, *unstaged* y *untracked*

staged = en índice pero no en historial

unstaged = modificados pero no añadidos a índice

untracked = en directorio de trabajo

log - Observar historial



`git log`

Muestra historial de cambios

Variantes

`git log --oneline`	Resumen en 1 línea
`git log --stat`	Estadísticas
`git log -p`	Camino completo con diff
`git log --autor="expr"`	Commits de un autor
`git log --grep="expr"`	Busca commits

`git log --graph --decorate --online`

Muestra grafo de cambios

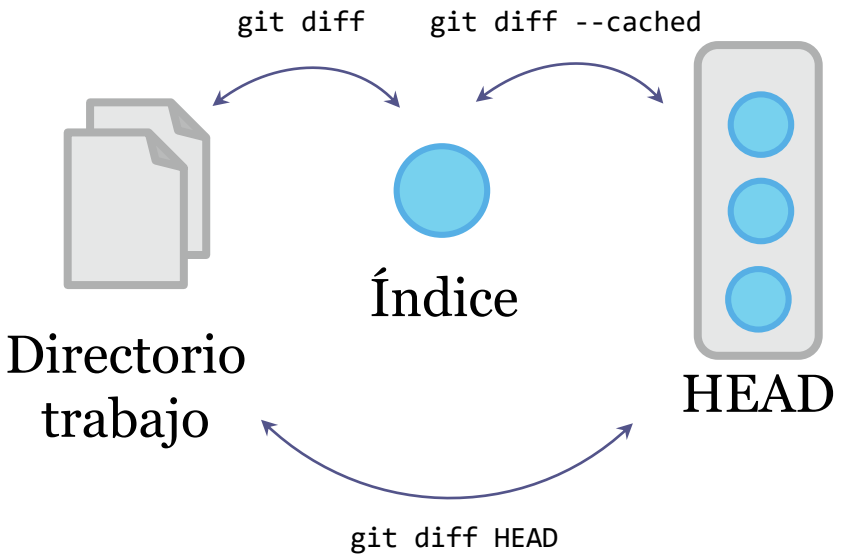
diff - Mostrar diferencias

`git diff`
- Dir. trabajo vs índice

`git diff --cached`
- Índice vs commit

`git diff HEAD`
- Dir. trabajo vs commit

Algunas opciones:
- --color-words
- --stat



Deshaciendo cambios

Comandos para deshacer cambios

- checkout
- revert
- reset
- clean

checkout - Cambiar

Cambia directorio de trabajo

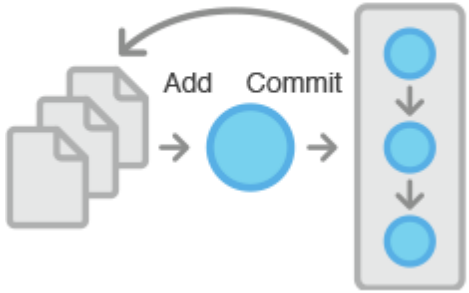
`git checkout <c>`

Cambiar a commit <c>

Se pasa a estado "*detached HEAD*"

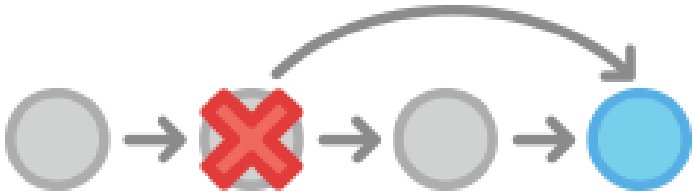
`git checkout <c> <f>`

Recupera fichero <f> de commit <c>



NOTA:
`checkout` también se utiliza para cambiar a diferentes ramas

revert - Recuperar



`git revert <c>` Recupera commit `<c>`

- Añade la versión recuperada al historial
- Operación "segura"
  - permite rastrear cambios en historial

reset - Deshacer

Deshacer cambios

Operación no segura

`git reset`	Deshace cambios en índice
`git reset --hard`	Deshace cambios en índice y directorio trabajo
`git reset <c>`	Deshacer cambios y recuperar commit `<c>`

NOTA: Es peligroso hacer reset en repositorios ya publicados
Es mejor utilizar `revert`

clean - Limpiar

Borrar ficheros locales

`git clean -f` Borra ficheros *untracked*

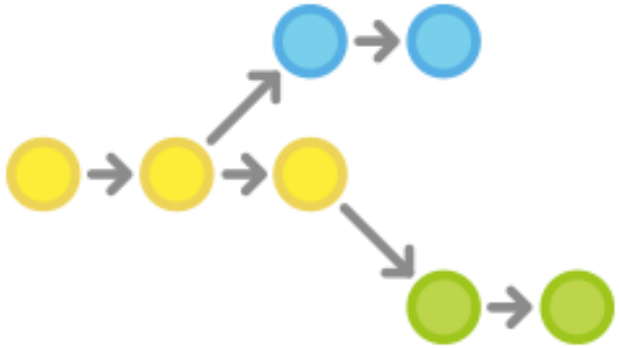
NOTA: Peligroso (se pueden perder cambios locales)

`git clean -n` Muestra qué ficheros se borrarían

Ramas

```
branch
checkout
merge
```

branch - Ramas



Gestión de ramas

`git branch` Muestra las ramas existentes

`git branch <r>` Crear la rama `<r>`

`git branch -d <r>` Borrar rama `<r>`

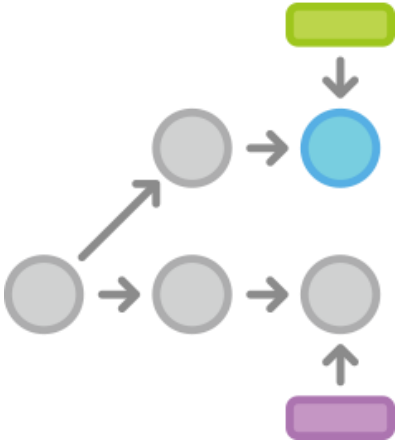
Segura (no borra si no hay mezclas pendientes)

`git branch -D <r>` Borrar rama `<r>`

Insegura (borra una rama y sus commits)

`git branch -m <r>` Renombrar rama actual a `<r>`

checkout - Cambiar



Cambiar a una rama

`git checkout <r>`

Cambia a la rama existente `<r>`

`git checkout master` Cambia a rama master

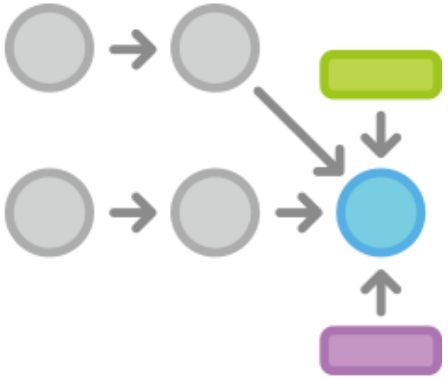
`git checkout -b <r>`

Crear rama `<r>` y cambia a ella

Equivalente a

```
git branch <r>
git checkout <r>
```

merge - Combinar



Combinar dos ramas

`git merge <r>`	Mezclar rama actual con `<r>`
`git merge --no-ff <r>`	Mezclar generando commit de mezcla (más seguro)

2 tipos de combinación (se verá en seminario)

- Merge fast-forward
- 3-way merge

Repositorios remotos

```
remote
fetch
pull
push
```

remote - Conectar repositorios

`git remote`

Ver repositorios externos

`git remote add <nombre> <uri>`

Crear conexión de nombre `<nombre>` a `<uri>`

`git remote rm <nombre>`

Borrar conexión `<nombre>`

`git rename <anterior> <nuevo>`

Renombrar conexión `<anterior>` a `<nuevo>`

NOTAS: `git clone` crea automáticamente una conexión llamada `origin`
Es posible tener conexiones a más de un repositorio externo

fetch - traer

Traer elementos de repositorio remoto

- Permite descargar ramas externas
- Operación segura: no mezcla con ficheros locales

`git fetch <remote>`

- Descargar todas las ramas del repositorio <remote>

`git fetch <remote> <rama>`

- Descargar la rama `<rama>` de repositorio `<remote>`

NOTA: Asigna FETCH_HEAD a cabeza de rama traída
Convenio para nombrar ramas: `<remoto>/<rama>`
Ejemplo: `origin/master`

pull - traer y mezclar

```
git pull <remoto>
```

Trae un repositorio remoto y lo mezcla

Equivale a:

```
git fetch
 +
git merge FETCH_HEAD
```

push - enviar

`git push <remoto> <rama>`

Enviar commits de repositorio local a remoto

Variantes

`git push <remoto> --all`

Enviar todas las ramas

Si hay cambios en repositorio remoto, muestra un error (non-fast-forward).
Solución:

1. Traer (pull) cambios y mezclar con repositorio local
2. Volver a enviar (push)

NOTA: También puede usarse opción: `--force` (no recomendado)