Git. Repositorios remotos

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

Jorge Juan <jjchico@dte.us.es>

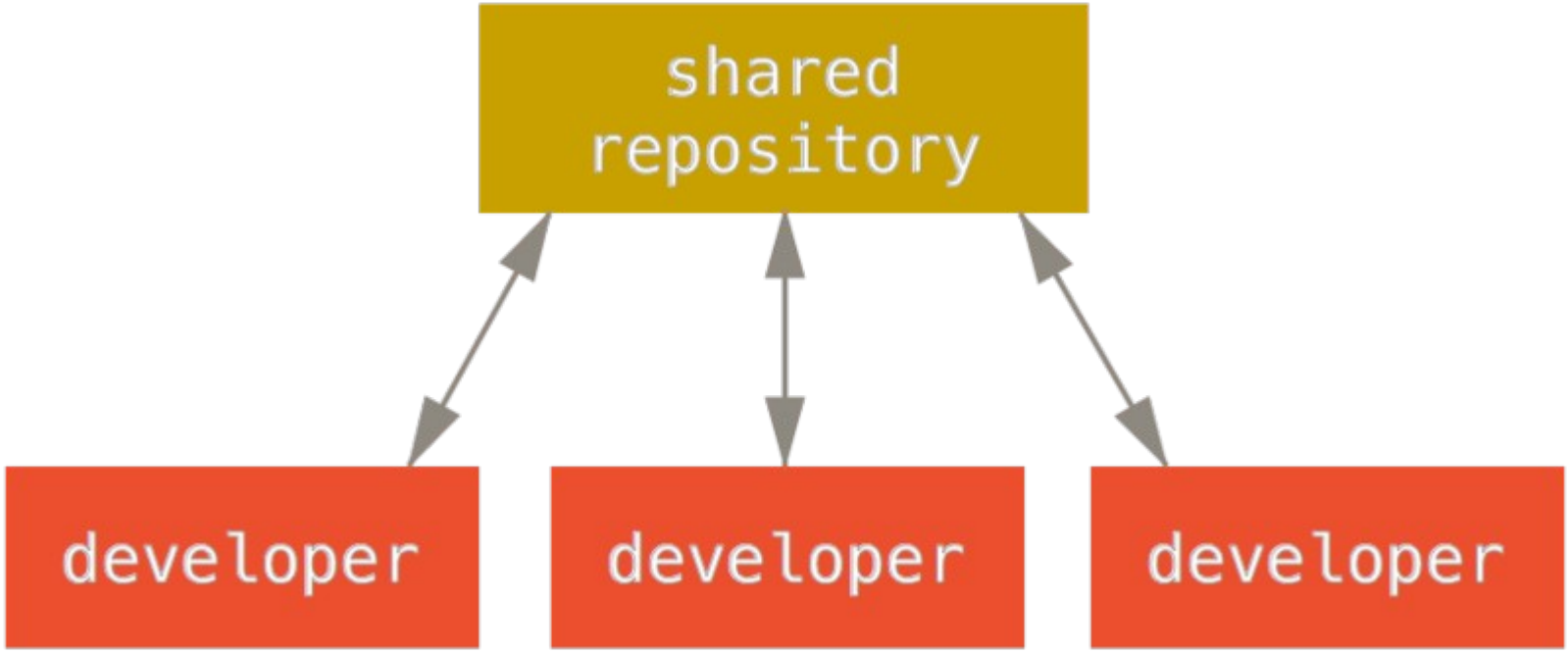
Repositorios remotos

- Referencia: Pro Git capítulo 2
- Contenidos
  - Flujos de trabajo
  - Remotos sobre `ssh'
  - Clonar
  - Seguir ramas remotas
  - Enviar y recibir

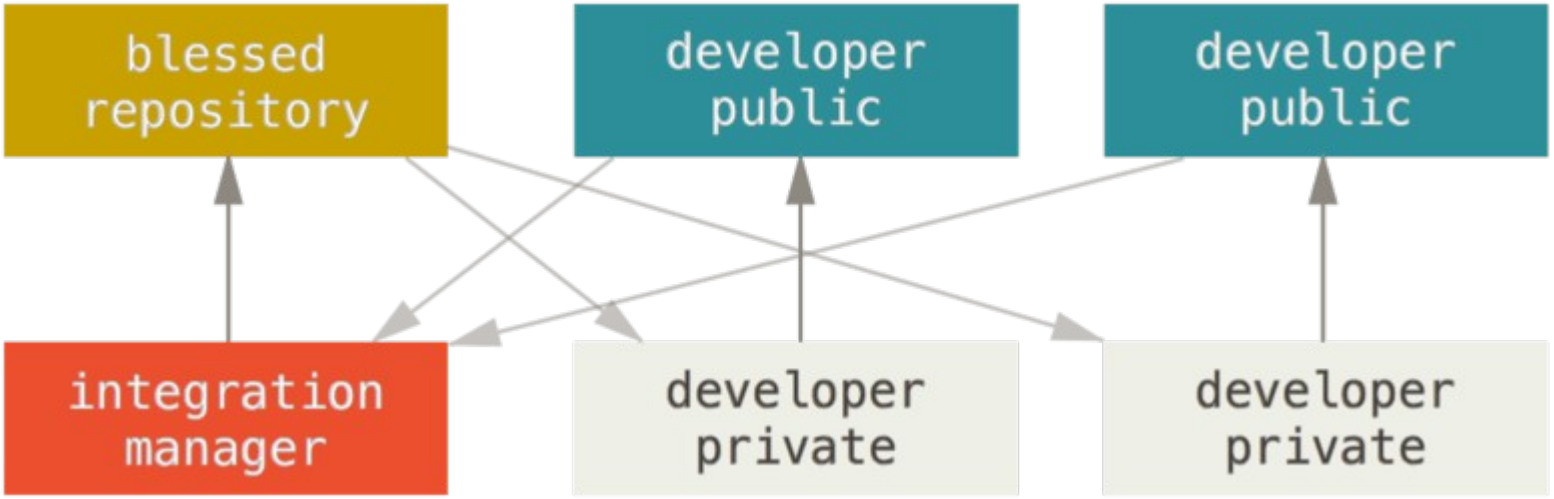
Competencias

- Comprender el concepto y operación de repositorios remotos en Git.
- Configuración del transporte SSH para rep. remotos.
- Clonar repositorios remotos y crear nuevos repositorios remotos para uso sobre SSH.
- Definir repositorios remotos y seguimiento de ramas.
- Sincronizar cambios hacia/desde rep. remotos.
- Desarrollar un proyecto con repositorios remotos empleando un flujo de trabajo simple
- Conocer diferentes flujos de trabajo usando repositorios remotos.

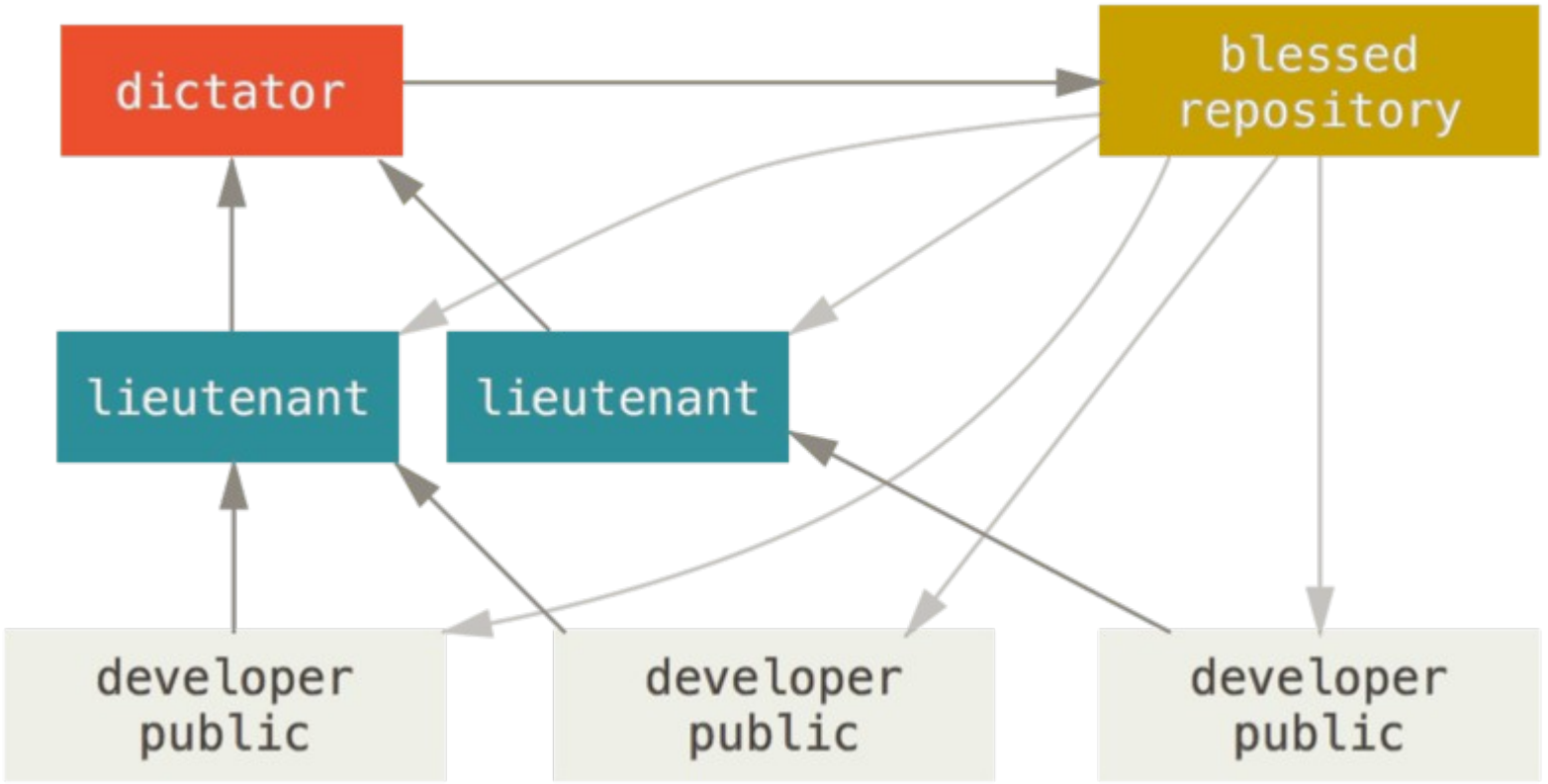
Flujos de trabajo
Proyecto centralizado



Flujos de trabajo
Gestor de integraciones



Flujos de trabajo
Dictador y tenientes



Repositorios remotos

- Cualquier repositorio local de Git puede actuar como repositorio remoto siempre que sea accesible.
- Múltiples protocolos de acceso:
  - ssh, http, https, git, local
- Un repositorio remoto puede contener todas o algunas de las ramas locales
- Las revisiones se sincronizan entre repositorios, normalmente sincronizando ramas completas
- Enviamos cambios (push)
  - Es necesario que el remoto no tenga cambios adicionales realizados por otros usurios.
- Recibimos cambios
  - Recibimos los cambios “aguas arriba” (upstream): fetch
  - Mezclamos los cambios con los cambios locales (merge)
  - fetch + merge = pull

Repositorios remotos

- Repositorio simple (bare):
  - No almacena archivos, sólo la base de datos
  - Pensado para ser usado sólo como repositorio remoto
- Protocolos y sus usos
  - SSH
    - Para enviar y recibir cambios
    - Puede configurarse para acceso mediante clave pública
    - Muy usado: estádar en sistema Linux
  - HTTP
    - Usado sobre todo para recibir cambios (repos públicos)
    - Basta poner la carpeta del repo en un servidor web
    - Estándar y fácil de configurar
  - Git
    - Sólo para recibir cambios
    - Similar a HTTP pero mucho más rápido

Crear repo remoto sobre ssh

- Cualquier cuenta de usuario accesible por “ssh” puede contener repositorios remotos.
- Un repositorio remoto no es más que una carpeta inicializada con un repositorio “simple” (bare)
  - **--bare**: sólo base de datos, sin archivos
  - **--shared**: usa permisos que facilitan colaboración de varios usuarios

```
(servidor) $ mkdir git
(servidor) $ mkdir git/proyecto.git
(servidor) $ cd git/proyecto.git
(servidor) $ git init --bare [--shared]
  Initialized empty Git repository in /home/<usuario>/git/proyecto.git/
```

Subir proyecto a repo remoto vacío

- Partimos de un proyecto local
  - Añadir dirección del repositorio remoto
  - Subir cambios (push)

```
$ git remote add origin <usuario>@<servidor>:git/proyecto.git
$ git push --set-upstream origin master
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 775 bytes | 0 bytes/s, done.
Total 9 (delta 0), reused 7 (delta 0)
To <usuario>@<servidor>:git/proyecto.git
 * [new branch]      master -> master
```

Obtener una copia local de un repo remoto (vacío o no)

- Basta “clonar” el repositorio remoto.
- Se creará un repositorio local y guardará la configuración del remoto del que se ha clonado

```
$ git clone <usuario>@<servidor>:git/proyecto.git
Cloning into 'proyecto'...
remote: Counting objects: 18, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 18 (delta 4), reused 0 (delta 0)
Receiving objects: 100% (18/18), done.
Resolving deltas: 100% (4/4), done.
```

Ejemplo. Repo remoto sobre ssh

```
# Información sobre remotos

$ git remote -v      # muestra repos remotos
    origin   <usuario>@<servidor>:git/proyecto.git (fetch)
    origin   <usuario>@<servidor>:git/proyecto.git (push)

# Listar ramas, incluyendo remotas

$ git branch -a      # muestra todas las ramas (locales y remotas)
* master
  remotes/origin/master

# Descripción detallada sobre un remoto

$ git remote show origin
  Fetch URL: <usuario>@<servidor>:git/proyecto.git
  Push  URL: <usuario>@<servidor>:git/proyecto.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Conexión mediante clave pública

```
# Generamos clave pública

$ ssh-keygen
...

# Copiamos clave pública al servidor

$ ssh-copy-id <usuario>@<servidor>
...

# Claves
#   ~/.ssh/id_dsa     - Clave privada: ¡nunca compartir!
#   ~/.ssh/id_dsa.pub - Clave privada: proporcionar al administrador del serv.
```

Seguimiento de ramas remotas

- Git distingue entre ramas remotas y locales
  - Ej: master, origin/master
- Una rama local puede (y suele) relacionarse con su correspondiente rama remota (upstream)
  - master → origin/master
  - devel → origin/devel
- Cuando una rama local se configurara para hacer “seguimiento” (track) de su correspondiente rama remota algunas tareas se simplifican:
  - “push” sin argumentos: envía cambios a la correspondiente rama remota que sigue.
  - “fetch” (“pull”) sin argumentos: recibe cambios (y los mezcla) de la correspondiente rama remota que sigue.
- “clone” establece automáticamente el seguimiento de ramas.

Recibir cambios

- fetch: recibe cambios en ramas remotas
- pull = fetch+merge: recibe y mezcla cambios
  - Puede fallar si hay conflictos
  - Se resuelven los conflictos y se hace la confirmación manualmente.

```
# Recibir cambios de un remoto y una rama determinada

$ git fetch origin dev
...

# Recibir cambios en el servidor en la rama actual (sin merge)

$ git fetch
...

# Recigir cambios y hacer mezcla (merge) de la rama actual

$ git pull
...
```

Enviar cambios

- Enviar cambios en una rama
  - La primera vez es necesario indicar el remoto
  - No es necesario indicar remotos si hay seguimiento de ramas

```
# Primera vez

$ git push --set-upstream origin dev
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 775 bytes | 0 bytes/s, done.
Total 9 (delta 0), reused 7 (delta 0)
To <usuario>@<servidor>:git/proyecto.git
 * [new branch]      dev -> dev

# Enviar cambios próximas ocasiones

$ git push
...
```

Seguimiento de ramas remotas

```
# Establecer rama remota y seguimiento al enviar una rama

$ git push --set-upstream origin dev
...

# Recibir una nueva rama, cambiar a ella (seguimiento automático)

$ git fetch origin dev
De <servidor>:git/lista_compra
 * branch            dev        -> FETCH_HEAD
$ git checkout dev
Branch dev set up to track remote branch dev from origin.
Switched to a new branch 'dev'

# Dejar de seguir una rama

$ git branch --unset-upstream dev

# Volver a seguir una rama

$ git branch --set-upstream-to=origin/dev dev
Branch dev set up to track remote branch dev from origin.
```

Seguimiento de ramas remotas

```
# Información de ramas locales y remotas y asociaciones

$ git branch -a -vv
* dev                    96b90a6 [origin/dev] aaaaaaaa
  master                 ba1ef40 [origin/master] bbbbbbbbbb
  tmp                    f58e796 ccccccccccccccccccc
  remotes/origin/dev     96b90a6 aaaaaaaa
  remotes/origin/master ba1ef40 bbbbbbbbbb

# Información de remotos

$ git remote show origin
* remote origin
  Fetch URL: <usuario>@<servidor>:git/proyecto.git
  Push  URL: <usuario>@<servidor>:git/proyecto.git
  HEAD branch: master
  Remote branches:
    dev    tracked
    master tracked
  Local branches configured for 'git pull':
    dev    merges with remote dev
    master merges with remote master
  Local refs configured for 'git push':
    dev    pushes to dev    (up to date)
    master pushes to master (up to date)
```

Borrar ramas remotas

- ¡Cuidado! No se debe borrar una rama remota usada por otros desarrolladores

```
$ git push origin --delete bug55

To <usuario>@<servidor>:git/proyecto.git
 - [deleted]         bug55
```

Flujos de trabajo simple (centralizado)

- Un desarrollador
  - El remoto actúa como copia de seguridad del proyecto
  - Contiene todas las ramas “importantes” del proyecto: principal (“master”), desarrollo (“dev”), etc.
  - El desarrollador puede usar ramas temporales sólo locales.
- Varios (pocos) desarrolladores
  - El remoto actúa como espacio de trabajo común, con permisos para todos: modelo centralizado.
  - Contiene ramas importantes del proyecto (principal -master-, desarrollo -dev-, etc.) y ramas específicas de cada desarrollador.
  - Un desarrollador actúa como gestor de integración y es el único que confirma cambios en “master” y/o “dev”.
  - Requiere disciplina y comunicación fluida por parte del equipo.

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