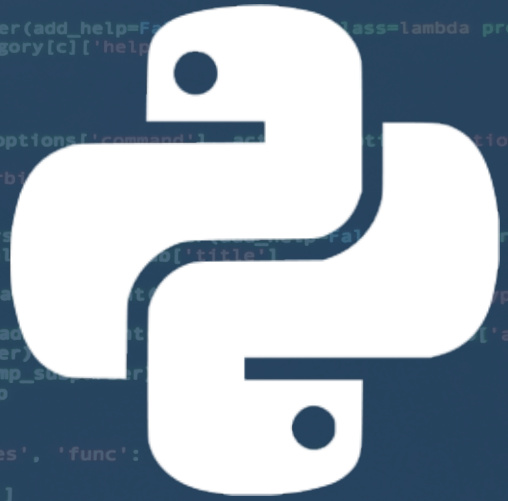


CONCEPTOS INICIALES

PROGRAMACIÓN III

Abdel G. Martínez L.

CONCEPTO Y DEFINICIÓN

- Lenguaje de programación interpretado
- Creado por Guido van Rossum
- Administrado por PSF
- Su nombre proviene de Monty Python
- Multiparadigma: orientación a objetos, a eventos, funcional, estructurada.
- Dinámicamente y fuertemente tipado
- Conteo de referencias

CARACTERÍSTICAS

- Sintaxis sencilla y elegante
- Ideal para desarrollo de prototipos
- Viene con una amplia librería
- El modo interactivo es útil para pruebas cortas
- Es fácil de extender usando módulos creados en C y C++
- Puede ser embebido en aplicaciones
- Se ejecuta en múltiples plataformas y sistemas operativos
- Es software libre

¿QUIÉN USA PYTHON?

- Google
- PBS
- NASA
- Yahoo
- Dropbox
- Eventbrite
- Mozilla
- Quora
- Pinterest
- Slideshare
- SurveyMonkey
- Red Hat
- Oracle
- Linode
- Cisco
- Yelp

ZEN DE PYTHON

- Bello es mejor que feo
- Explícito es mejor que implícito
- Simple es mejor que complejo
- Complejo es mejor que complicado
- Plano es mejor que anidado
- Disperso es mejor que denso
- La legibilidad cuenta
- Los casos especiales no son tan espaciales para quebrantar reglas

ZEN DE PYTHON

- Aunque lo práctico gana a la pureza
- Los errores nunca deberían dejarse pasar silenciosamente
- A menos que hayan sido silenciados explícitamente
- Frente a la ambigüedad, rechaza la oportunidad de adivinar
- Debería haber una, y preferiblemente una, manera obvia de hacerlo
- Esa manera no es obvia al inicio a menos que seas holandés
- Ahora es mejor que nunca
- Aunque nunca es a menudo mejor que ya mismo

ZEN DE PYTHON

- Si la implementación es difícil de explicar, es una mala idea
- Si la implementación es fácil de explicar, puede que sea buena idea
- Los espacios de nombre son una gran idea, hagamos más de eso

VERSIONES

- Creado en 1989
- **Python 1.0** lanzado en 1994
- **Python 2.0** lanzado en 2000
- **Python 3.0** lanzado en 2008
- Existe una dualidad de versiones:
  - **Python 2.7.10** lanzado en 2015
  - **Python 3.4.3** lanzado en 2015
  - Entonces, *¿qué aprendo?*

DIFERENCIAS ENTRE PYTHON 2.7.X Y 3.X

- Para imprimir se usa `print`, que es una función no una sentencia
- Se utilizan vistas e iteradores en lugar de vistas
- Las reglas de ordenamiento han sido simplificadas
- Existe un único tipo de entero: `int`
- La división de dos enteros da como resultado un flotante
- Todo el texto es Unicode; Unicode codificado es data binaria
- ¿Por qué todavía usan Python 2.7?
  - *Por las librerías. Algunas no están portadas a Python 3.*

MANEJO DE REPOSITORIOS GIT

PROGRAMACIÓN III

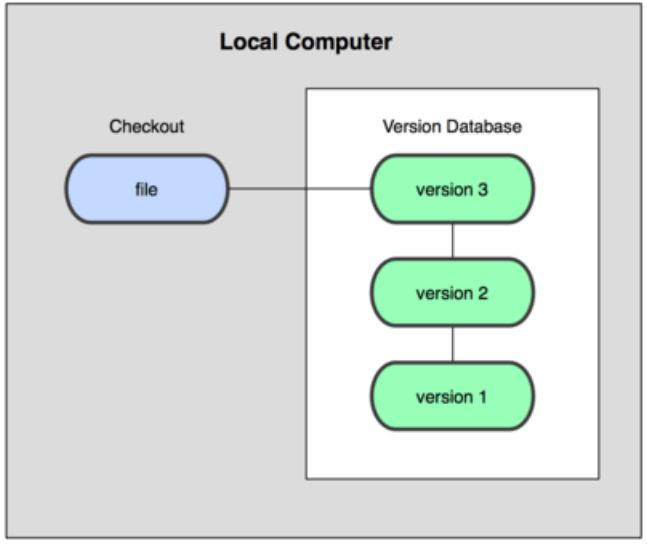
Abdel G. Martínez L.

ACERCA DEL CONTROL DE VERSIONES

- Sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo
- Es útil para cualquier profesión, no únicamente programadores
- Permite revertir archivos específicos a su estado anterior, o bien todo el proyecto entero
- Compara cambios a lo largo del tiempo, permitiendo ver quién modificó por última vez un archivo
- Su coste de implementación es muy bajo

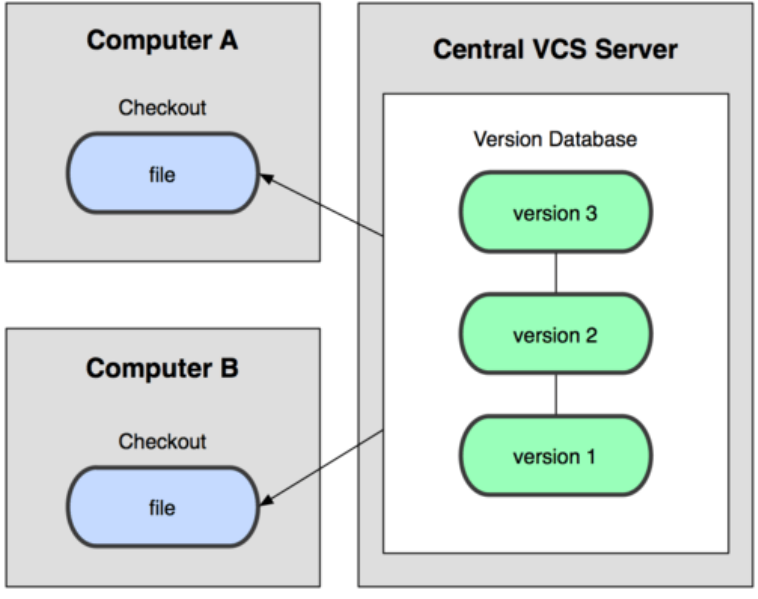
CONTROL DE VERSIONES LOCAL

- El método más primitivo, utilizado por mucha gente, es copiar los archivos a otro directorio
- Este enfoque es muy común por su simpleza, pero está propenso a múltiples errores: olvidar la ubicación del directorio, sobreescribir archivos no deseados



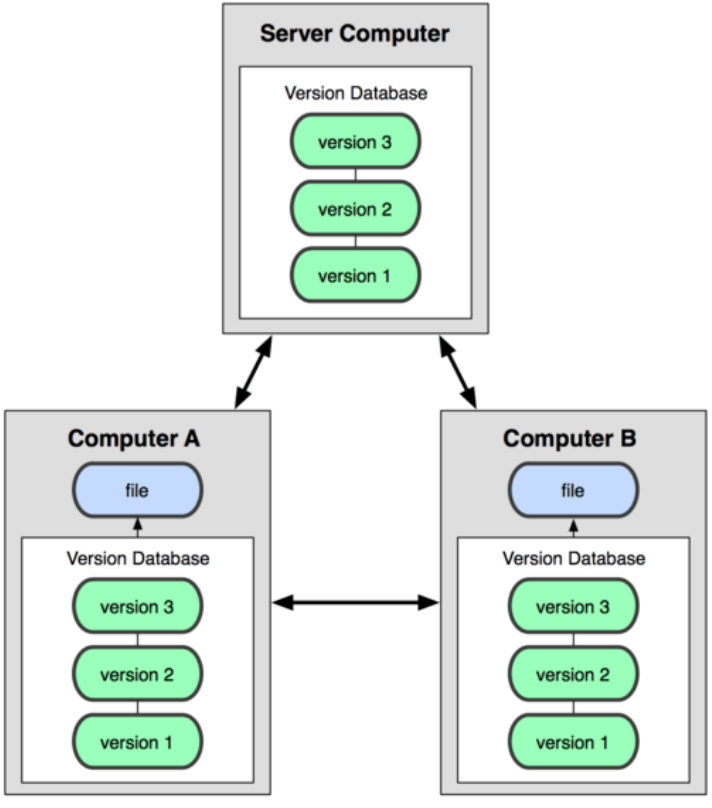
CONTROL DE VERSIONES CENTRALIZADO

- Nacen por la necesidad de colaborar en otros sistemas
- El servidor contiene todos los archivos centralizados y los clientes descargan del servidor
- Su mayor desventaja es la alta dependencia al servidor
- Ejemplos: CVS, Subversion.



CONTROL DE VERSIONES DISTRIBUIDO

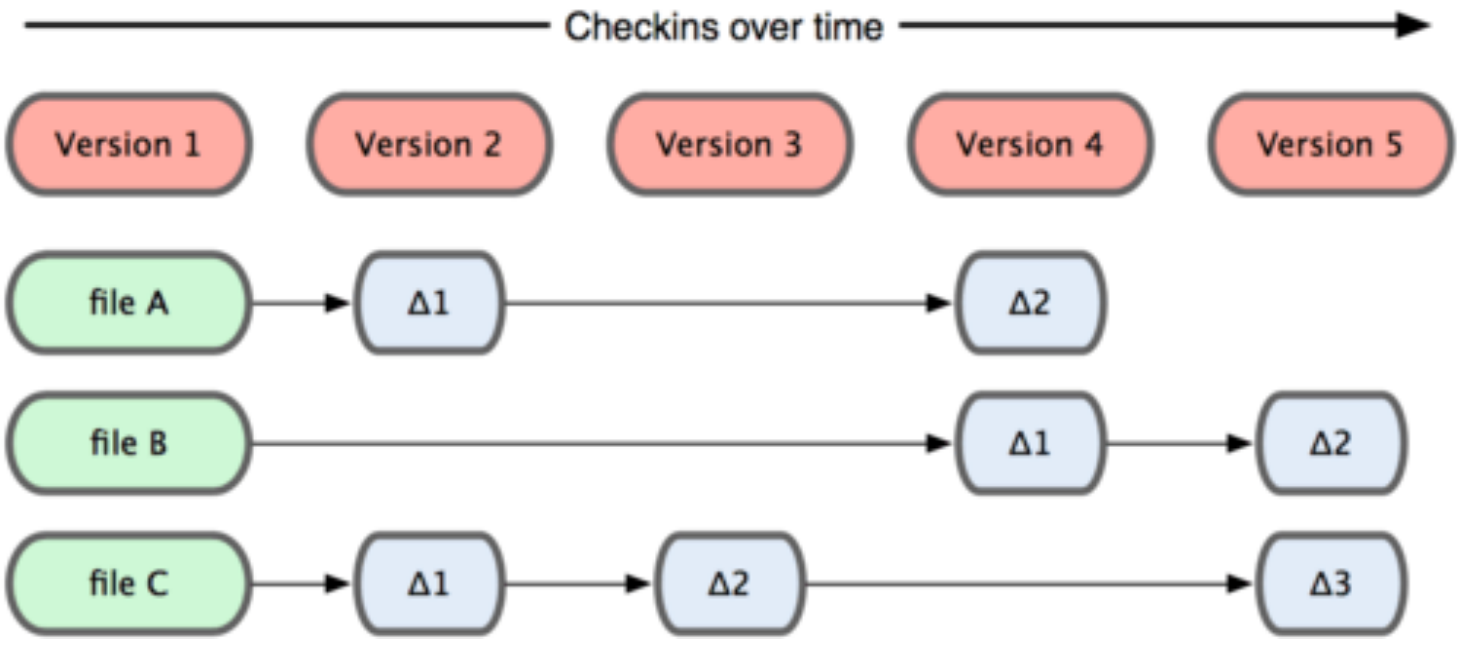
- No sólo descargan la última versión de los archivos, replican completamente el repositorio
- Se elimina la dependencia total al servidor primario
- Se puede trabajar de forma simultánea en distintos grupos
- Ejemplos: Git, Mercurial



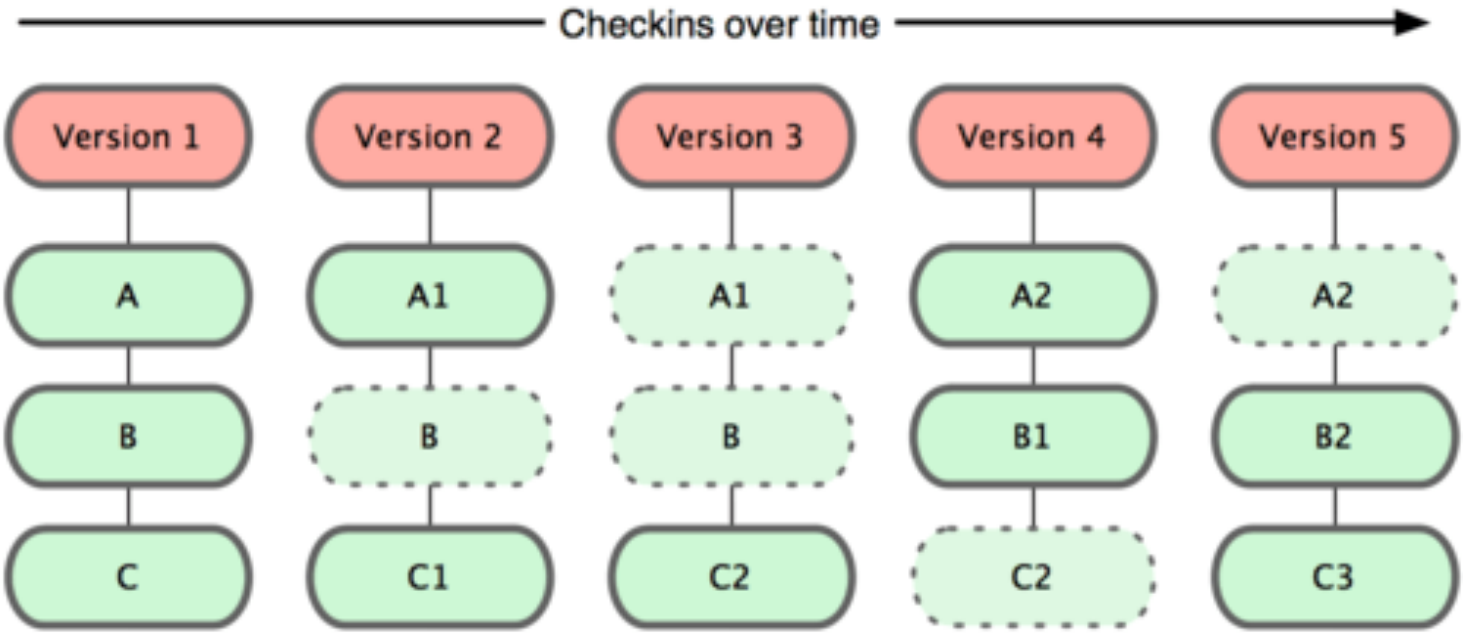
BREVE HISTORIA DE GIT

- Comenzó de la mala relación entre la comunidad desarrolladora del núcleo de Linux y la compañía que desarrollaba BitKeeper
- Linus Torvalds, impulsó a desarrollar su propia herramienta basada en la experiencia del uso de BitKeeper
- Git ha evolucionado y madurado, desde 2005, para ser fácil de usar, ser tremendamente rápido, eficiente con grandes proyectos y tener un sistema de ramificación increíble para el desarrollo no lineal

FUNDAMENTOS DE GIT



FUNDAMENTOS DE GIT



GIT USA OPERACIONES LOCALES

- La mayoría de las operaciones necesitan archivos y recursos locales
- No se necesita información de ningún otro ordenador de la red
- Como toda la historia del proyecto es local, parece inmediato
- Otros controles de versiones necesitan conexión al servidor

GIT TIENE INTEGRIDAD

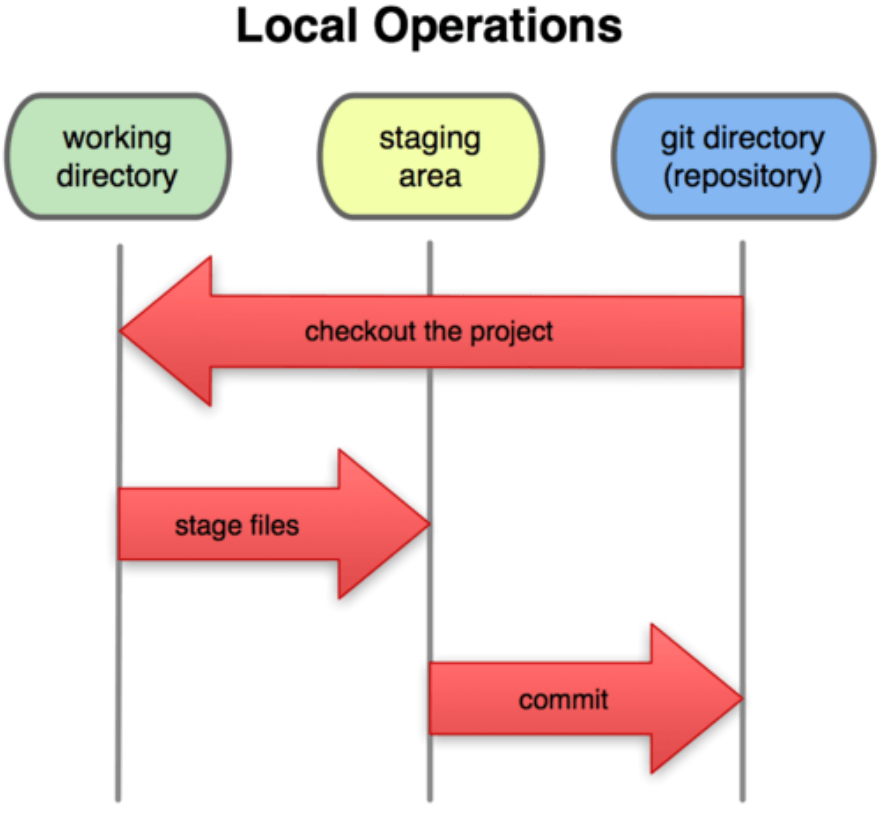
- Verifica todo mediante una suma de comprobación (checksum)
- Es imposible cambiar contenidos sin que Git se entere
- Esto forma parte de su funcionalidad y su filosofía
- El mecanismo utilizado por Git es el hash SHA-1:
  - Cadena de 40 caracteres hexadecimales (0-9 y a-f)
  - Calculado por contenido del archivo o estructura de directorio

  `da39a3ee5e6b4b0d3255bfef95601890afd80709`
- Git guarda todo por el valor hash de su contenido

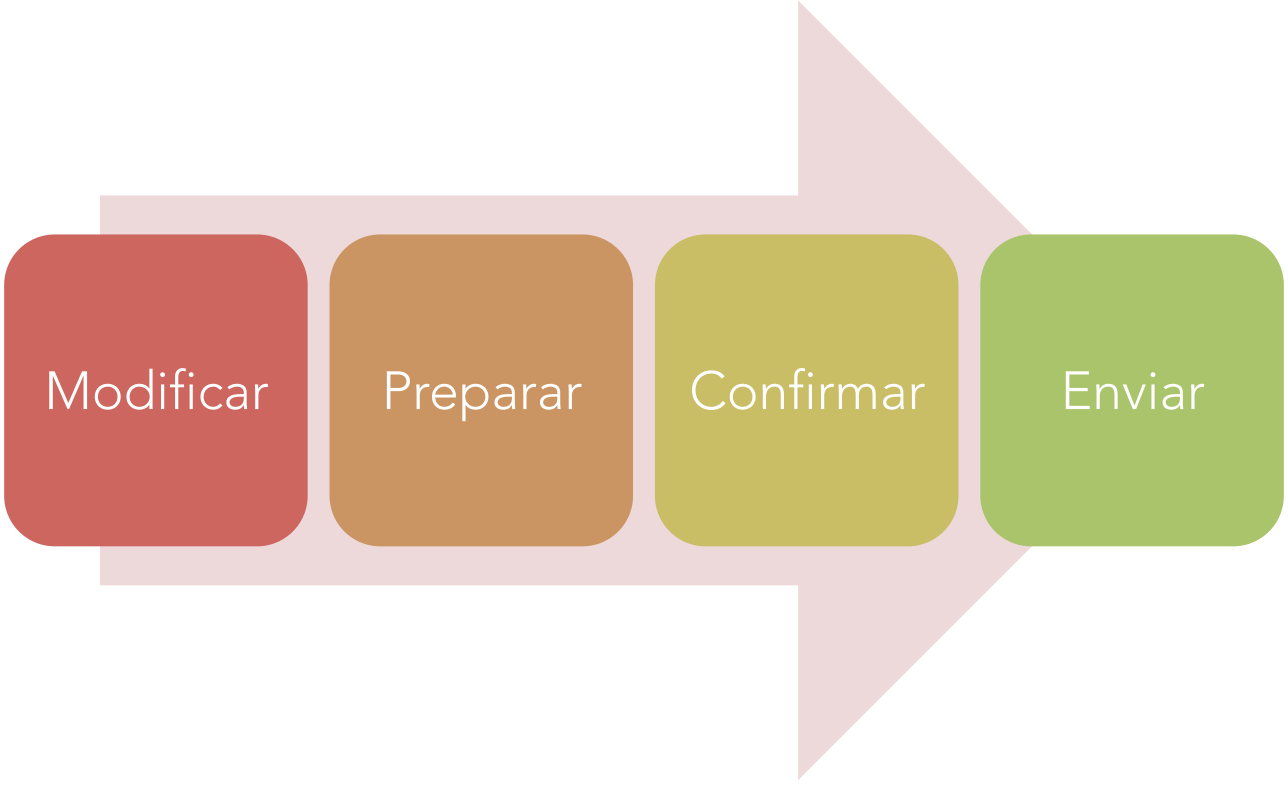
GIT ES SEGURO

- Las acciones realizadas sólo añaden información a la base de datos
- Es difícil conseguir que el sistema haga algo que no se pueda deshacer, o que borre información

LOS TRES ESTADOS DE GIT



FLUJO DE TRABAJO DE GIT



INSTALACIÓN

- *Windows*
  - msysGit (http://msysgit.github.com/)
- *Mac OS X*
  - Git Installer (http://sourceforge.net/projects/git-osx-installer)
  - Usando el comando `brew install git`
- *GNU/Linux*
  - En Fedora se instala usando `dnf install git-core`
  - En Ubuntu se instala usando `apt-get install git`

COMANDOS BÁSICOS - CREACIÓN

- Clonación de repositorio existente

```
$ git clone https://domain.com/user/repo.git
```

- Creación de un nuevo repositorio local

```
$ git init
```

COMANDOS BÁSICOS - CONFIGURACIÓN

- Establecer usuario, correo, editor de texto y diferenciador

```
$ git config --global user.name "Carlos Martínez"
$ git config --global user.email me@example.com
$ git config --global core.editor emacs
$ git config --global merge.tool vimdiff
```

- Comprobar la configuración actual

```
$ git config --list
```

COMANDOS BÁSICOS – CAMBIOS LOCALES

- Ver los archivos cambiados en el directorio de trabajo

```
$ git status
```

- Ver los cambios en los archivos monitorizados

```
$ git diff
```

- Agregar todos los cambios actuales en el siguiente *commit*

```
$ git add .
```

COMANDOS BÁSICOS – CAMBIOS LOCALES

- Agregar algunos cambios en <archivo> en el siguiente *commit*

```
$ git add <archivo>
```

- Entregar todos los cambios locales en los archivos monitorizados

```
$ git commit -a
```

- Cambiar el último *commit*

```
$ git commit --amend
```

COMANDOS BÁSICOS – HISTÓRICO

- Mostrar todos los *commits*, iniciando con el más nuevo

```
$ git log
```

- Mostrar los cambios en el tiempo de un archivo específico

```
$ git log –p <archivo>
```

- Ver quién cambió qué y cuándo en un archivo específico

```
$ git blame <archivo>
```

COMANDOS BÁSICOS – RAMAS Y ETIQUETAS

- Lista todas las ramas existentes

```
$ git branch -av
```

- Cambiar rama en el HEAD

```
$ git checkout <rama>
```

- Crear una nueva rama basada en el contenido en el HEAD

```
$ git branch <nueva-rama>
```

COMANDOS BÁSICOS – RAMAS Y ETIQUETAS

- Eliminar una rama local

```
$ git branch -d <rama>
```

- Clonar una rama remota

```
$ git checkout --track -b <nombre> <rep>/<bra>
```

- Marcar el commit actual con una etiqueta

```
$ git tag <etiqueta>
```

COMANDOS BÁSICOS – ACTUALIZAR

- Listar todos los sitios remotos configurados

```
$ git remote -v
```

- Muestra información de un repositorio remoto

```
$ git remote show <remoto>
```

- Agrega un nuevo repositorio remoto

```
$ git remote add <nombre-corto> <url>
```

COMANDOS BÁSICOS – ACTUALIZAR

- Descarga todos los cambios de <remoto>, sin integrar en HEAD

```
$ git pull <remoto> <rama>
```

- Publica los cambios locales en un repositorio remoto

```
$ git push <remoto> <rama>
```

- Elimina una rama de un repositorio remoto

```
$ git branch -dr <remoto>/<rama>
```

COMANDOS BÁSICOS – ACTUALIZAR

- Publica todas las etiquetas

```
$ git push --tags
```

COMANDOS BÁSICOS – MEZCLAR

- Mezcla la <rama> en el HEAD actual

```
$ git merge <rama>
```

- Rebase de HEAD actual en <rama>

```
$ git rebase <rama>
```

- Abortar un rebase

```
$ git rebase --abort
```

COMANDOS BÁSICOS – MEZCLAR

- Continua un rebase luego de resolver conflictos

```
$ git rebase --continue
```

- Resuelve el conflicto entre archivos

```
$ git mergetool
$ git add <archivo>
$ git rm <archivo>
```

COMANDOS BÁSICOS – DESHACER

- Descartar todos los cambios locales en el directorio de trabajo

```
$ git reset --hard HEAD
```

- Descartar los cambios locales en un archivo específico

```
$ git checkout HEAD <file>
```

- Revertir un *commit*

```
$ git revert <commit>
```

COMANDOS BÁSICOS – DESHACER

- Resetear el puntero del HEAD a un *commit* anterior
  - *Descartando todos los cambios*

    ```
    $ git reset --hard <commit>
    ```

  - *Preservando los cambios como cambios no enviados*

    ```
    $ git reset <commit>
    ```

  - *Preservando cambios locales sin commit*

    ```
    $ git reset --keep <commit>
    ```

COMANDOS BÁSICOS – AYUDA GENERAL

- Obtener ayuda desde la línea de comandos

```
$ git help <comando>
```

MEJORES PRÁCTICAS

- Hacer *commit* de cambios relacionados
- Hacer *commit* más seguidos
- No hacer *commit* de trabajo medio-terminado
- Probar el código antes de hacer *commit*
- Escribir buen mensaje en los *commit*
- Un control de versiones no es sinónimo de sistema de respaldos
- Utilizar ramas
- Seguir un flujo de trabajo

OTRAS REFERENCIAS

- **Git – la guía sencilla**

  http://rogerdudler.github.io/git-guide/index.es.html

- **Try Git**

  https://try.github.io/