Andalucía lab CENTRO DE INNOVACIÓN TURÍSTICA

Junta de Andalucía
Consejería de Turismo, Regeneración, Justicia y Administración Local

UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional

PROGRAMA DE ATRACCIÓN DE TALENTO TECNOLÓGICO PARA EL SECTOR TURÍSTICO ANDALUZ

2020

Introducción a GIT

FORMACIÓN TURISMO Y TECNOLOGÍA

UTE conformada por:

ideanto comunicación y tecnología

arelance Building Digital Talent

Introducción a GIT.

Control de versiones.

- Herramientas y utilidades que permiten tener una gestión de todos los cambios que se hacen tanto en la estructura de directorios como en el contenido de cada uno de los ficheros.
- Cuando un equipo trabaja con los mismos archivos, en tales casos resulta imprescindible mantener el control sobre los cambios que se realizan, así como tener la posibilidad de saber qué cambios han habido, quién ha hecho dichos cambios y cuándo
- Muy útil a la hora de gestionar conflictos, en aquellas situaciones en las que los cambios realizados por dos personas son incompatibles y es necesario tomar una decisión sobre cuál de los cambios es que debe permanecer.
- Si por el motivo que sea, los últimos cambios sobre alguno de los ficheros tiene efectos indeseados, gracias al control de versiones es posible revertir dichos cambios y volver a un estado consistente.

Sistemas de control de versiones

- Un sistema de control de versiones permite la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado.
- Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o SVC (System Version Control).
- Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Ejemplos de este tipo de herramientas son entre otros: CVS, Subversion, SourceSafe, Bazaar, Git, Mercurial, etc.

Sistemas de control de versiones - Terminología

- Repositorio:

  El repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios

- Revisión:

  Una revisión es una versión determinada de la información que se gestiona. Dependiendo del sistema de versiones, es posible que utilicen contadores (por ejemplo, subversion) mientras que hay otros que utilizan hashes (por ejemplo, git utiliza SHA1).

- Tag (etiqueta):

  Los tags permiten identificar revisiones importantes en el proyecto. Por ejemplo se suelen usar tags para identificar el contenido de las versiones publicadas del proyecto.

- Branch (rama):

  Se refiere a un conjunto de archivos que ha sido ramificado o bifurcado en un punto en el tiempo, de tal manera que a partir de ese momento, existen dos copias de dichos archivos que serán desarrolladas y modificadas de forma independiente.

Sistemas de control de versiones - Terminología

- Change/Diff (cambio):

  Representa una modificación específica en un documento bajo el control de versiones.

- Checkout (despliegue):

  Se trata de la operación que le permite a un usuario crear una copia local del repositorio. Un usuario puede especificar una revisión en concreto u obtener la última revisión disponible (head).

- Commit (Confirmación):
  - Operación que permite confirmar o mezclar los cambios realizados en la copia de trabajo local con el repositorio, dando lugar a la creación de una nueva revisión con los cambios realizados en la copia de trabajo local. Los términos 'commit' y 'checkin' también se pueden utilizar para describir la nueva revisión que se crea como resultado de confirmar.

- Conflict (Conflicto):

  Se trata de una situación que se presenta cuando existen varios “commit” sobre un mismo recurso y el sistema de versiones no puede conciliar los cambios. En éste caso, el usuario debe resolver el conflicto manualmente, ya sea integrando los cambios o descartando una versión y permitiendo que la otra pueda sea guardada.

Sistemas de control de versiones - Terminología

- Head (Cabeza):

  Se trata de la última confirmación, ya sea en el trunk o en una de las rama del repositorio. El trunk y cada rama tienen su propia cabeza, aunque HEAD se utiliza normalmente para referirse al trunk.

- Trunk (Tronco):

  Se trata de la única línea de desarrollo que no es una rama, dependiendo de la terminología de cada sistema de control de versiones, se le llama también línea base, línea principal o máster.

- Merge (Mezcla):

  Una mezcla es la operación mediante la cual se aplican dos o varios cambios de fuentes diferentes sobre el mismo archivo y dichos cambios son mezclados, siempre y cuando no se produzcan conflictos. Se trata de una operación muy habitual en equipos de trabajo.

Sistemas de control de versiones - Clasificación

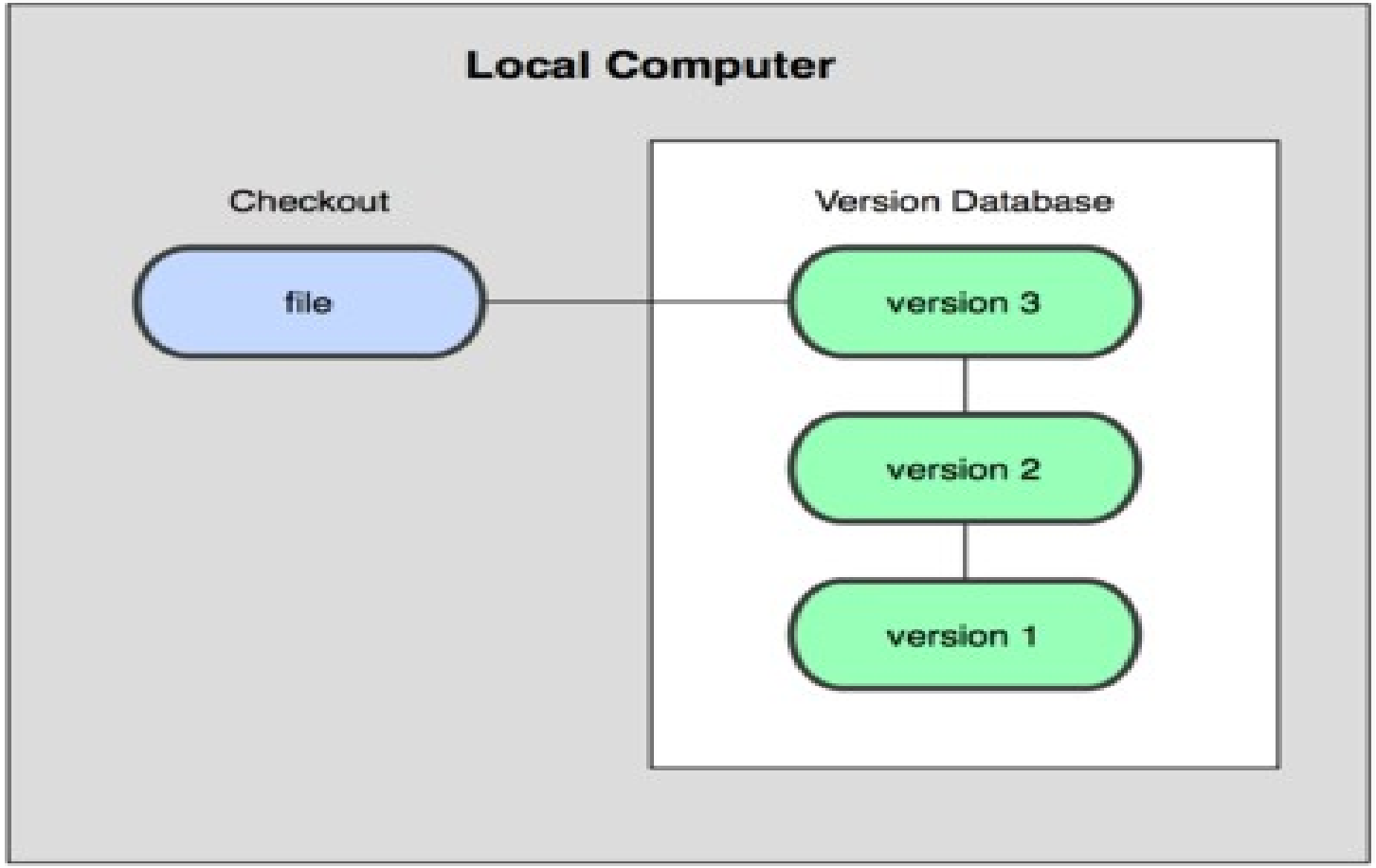
- Es posible clasificar los sistemas de control de versiones por su arquitectura para el almacenamiento de ficheros: locales, centralizados y distribuidos.
  - Locales.
  - Los cambios son guardados localmente y no se comparten con nadie.

  - Centralizados.
  - Existe un repositorio centralizado con toda la estructura de ficheros. En éste caso, es necesario contar con los permisos necesarios para realizar cambios importantes sobre el repositorio, como por ejemplo crear ramas o alterar la estructura del repositorio. Este modelo es típico en sistemas de control de versiones como CVS y Subversion

  - Distribuidos.
  - Cada usuario tiene su propio repositorio local y es responsable del mismo, sin embargo, los repositorios de cada usuario se pueden intercambiar y mezclar. Es frecuente el uso de un repositorio que actúa como punto de sincronización de los distintos repositorios locales. Este modelo es típico en sistemas de control de versiones como Git y Mercurial.

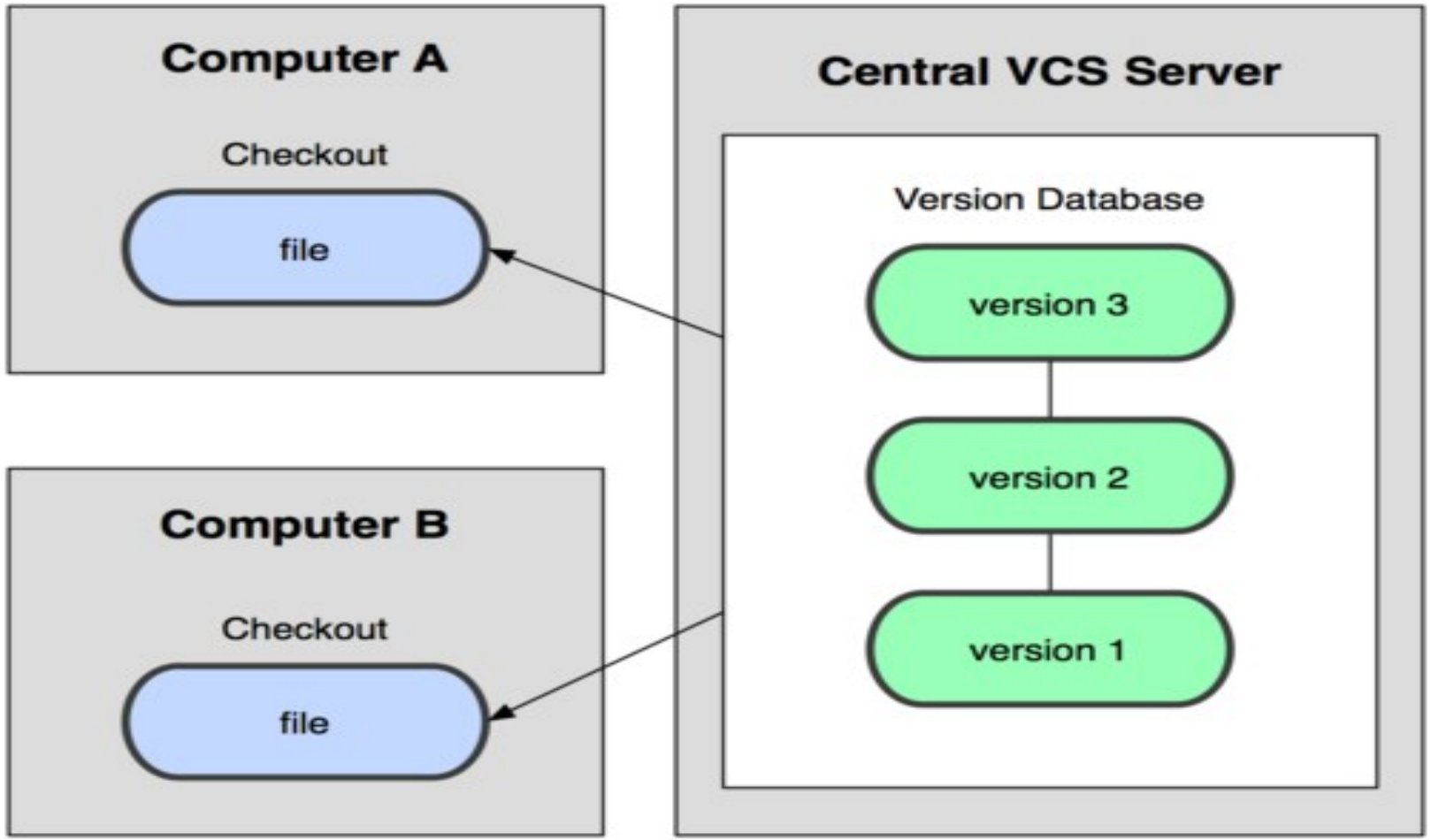
Sistemas de control de versiones - Clasificación

- Sistemas de control de versiones locales.



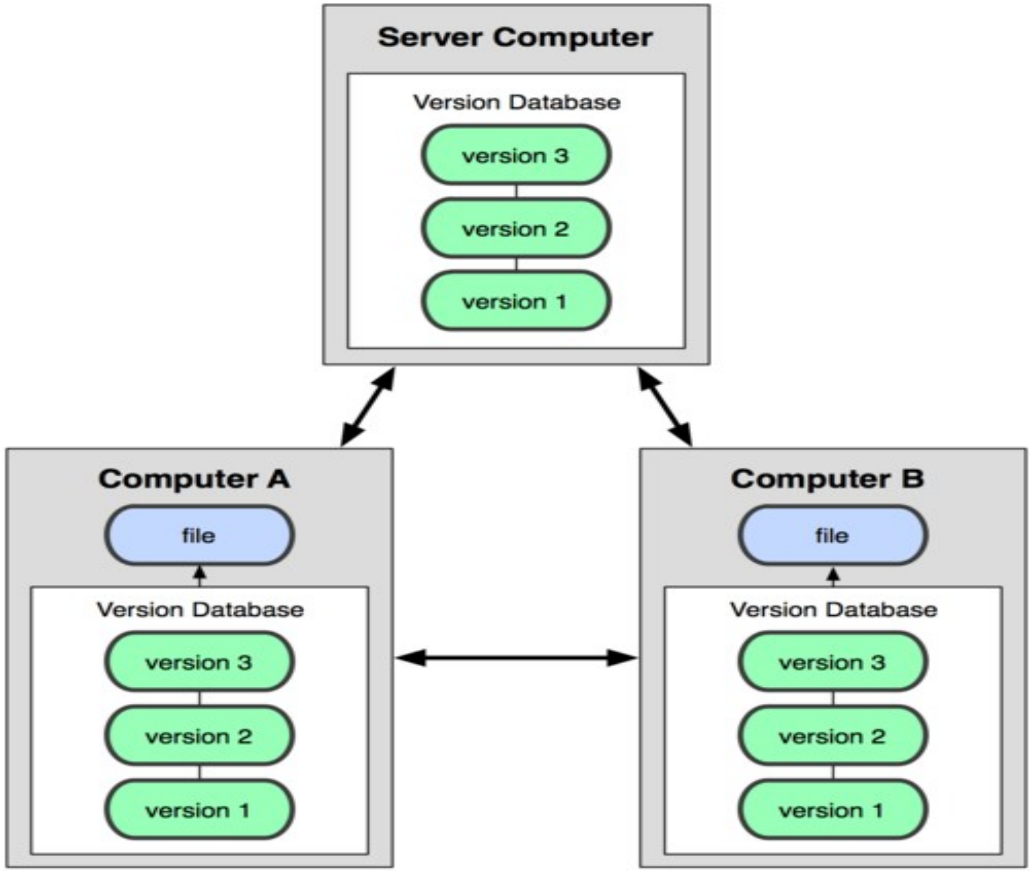
Sistemas de control de versiones - Clasificación

- Sistemas de control de versiones centralizados.



Sistemas de control de versiones - Clasificación

- Sistemas de control de versiones distribuidos.



Sistemas de control de versiones - GIT

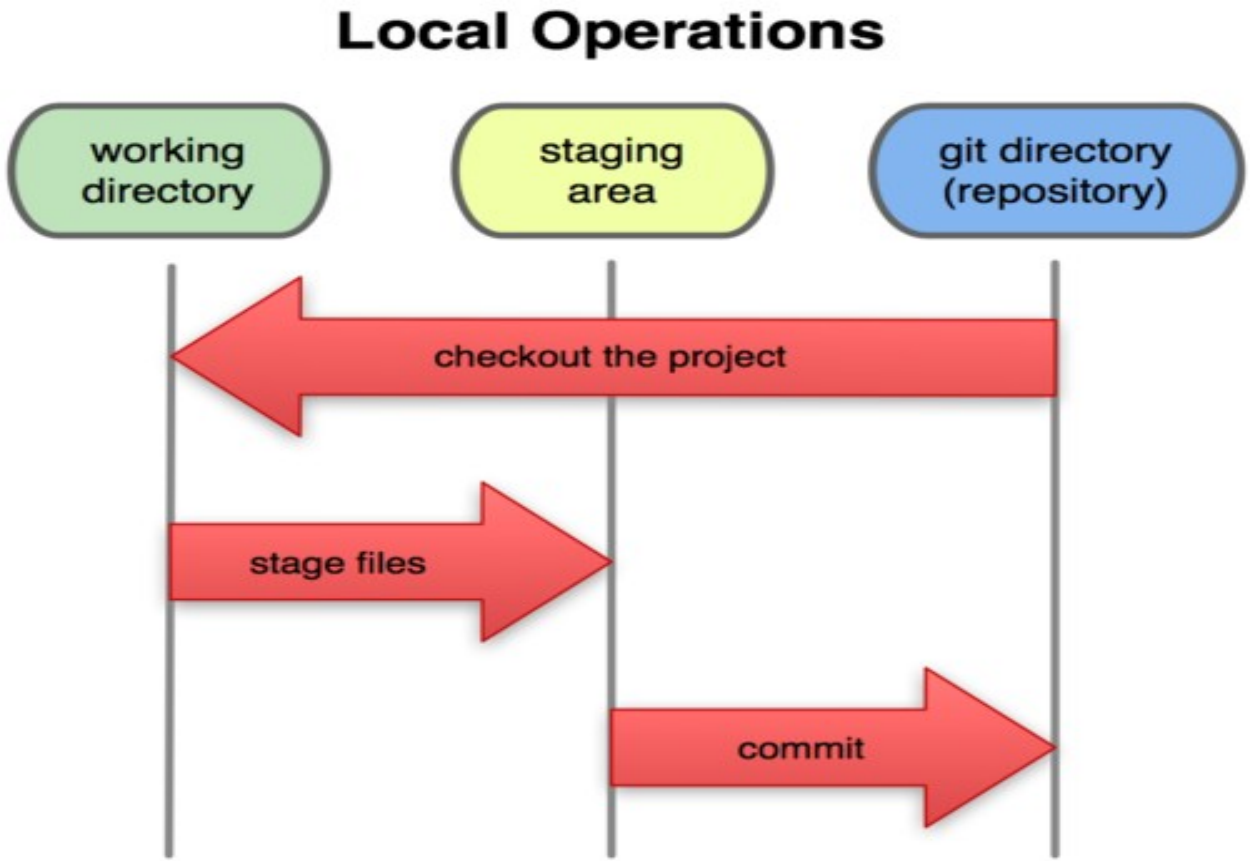
- GIT es un SVC distribuido que se caracteriza por ser rápido, fiable y escalable. A diferencia de otros sistemas de control de versiones, Git modela sus datos como un conjunto de instantáneas de cada repositorio.
- La mayoría de las operaciones son locales y opcionalmente se pueden compartir con otros repositorios por medio de operaciones “commit” y “push”
- Git lleva un control del checksum SHA1 de cada archivo o commit y habitualmente sólo añade datos, por tanto, todo queda registrado y todo es recuperable.
- A diferencia de otros SVC, GIT tiene tres zonas diferentes en las que se gestionan los estados del repositorio local.

Estados en GIT

- Todos los ficheros almacenados en el repositorio local pueden tener tres estados diferentes: confirmado (committed), modificado (modified), y preparado (staged). Todos los ficheros en el repositorio tienen uno de estos tres estados en un momento dado.
- Confirmado significa que los datos están almacenados de manera segura en la base de datos local.
- Modificado significa que se ha modificado el archivo pero todavía no se ha confirmado en la base de datos local del repositorio.
- Preparado significa que el fichero ha sido marcado para que sea incluido en la próxima revisión que se generará tras ejecutar una confirmación (commit).

Estados y secciones en GIT

- Partiendo de los estados indicados anteriormente, existen tres secciones principales de un proyecto de Git: El directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area)



Instalación de GIT en sistemas basados en GNU/Linux

- Crear un repositorio en un directorio concreto.
  - git init
- Configuración global:
  - git config --global user.name "Usuario Nombre"
  - git config --global user.email usuario@example.com
  - git config --global core.editor nano
  - git config --global merge.tool vimdiff
  - git config --list
  - git config user.name
  - git config merge.tool

Uso básico de GIT

Creación y commit de un fichero en el repositorio.

- touch fichero1; touch fichero2; git add .
- git commit -m “Creación de ficheros.”
- git status
- Modificar cualquiera de los ficheros creados anteriormente y ejecutar:
- git status
- git add .
- git commit -m “Fichero modificado.”

- Diferencias entre workdir y staging
  - Modificar el fichero “fichero1” y luego: git add fichero1 ; git status
  - Modificar nuevamente el fichero “fichero1: Y luego: git status
    - Changes to be committed:
      - (use "git reset HEAD <file>..." to unstage)
      - modified: fichero1
    - Changes not staged for commit:
      - (use "git add <file>..." to update what will be committed)
      - modified: fichero1

Uso básico de GIT

- El fichero modificado aparece dos veces. El primero está preparado para ser confirmado y está almacenado en la zona de staging. El segundo indica que está modificado otra vez en la zona de trabajo (workdir).
- En este punto es posible hacer los cambios por separado.
  - git commit -m "Primer cambio adicionado"
  - git status
  - git add . ; git status

- Realizar más pruebas modificando varios ficheros y añadiéndolos uno a uno.

Uso básico de GIT

- La orden “git add .” es mucho más rápido que tener que ir añadiéndolos uno por uno. El problema es que, si no se tiene cuidado, se puede terminar por añadir archivos innecesarios o con información sensible.
- Por lo general se debe evitar añadir archivos que se hayan generado como producto de la compilación del proyecto, los que generen los entornos de desarrollo (archivos de configuración y temporales) y aquellos que contentan información sensible, como contraseñas o tokens de autenticación. Por ejemplo, en un proyecto Java, los archivos “.class” no deben incluirse, solo los que contengan código fuente.
- Para indicarle a git que debe ignorar un archivo, se puede crear un fichero llamado “.gitignore”, bien en la raíz del proyecto o en los subdirectorios que se desee. Dicho fichero puede contener patrones, uno en cada línea, que especifiquen los archivos que deben ignorarse.

Uso básico de GIT

- El formato del fichero “.gitignore” debe ser como el siguiente:
  - # .gitignore
  - dir1/ # ignora todo lo que contenga el directorio dir1
  - !dir1/info.txt # El operador ! excluye del ignore a dir1/info.txt (sí se guardaría)
  - dir2/*.txt # ignora todos los archivos txt que hay en el directorio dir2
  - dir3/**/*.txt # ignora todos los archivos txt que hay en el dir3 y sus subdirectorios
  - *.o # ignora todos los archivos con extensión .o en todos los directorios
- Crear una estructura de directorios que cumplan las reglas anteriores y comprobar que efectivamente, los ficheros que cumplen con los formatos anteriores son ignorados.
  - Si un fichero ya ha sido añadido, por ejemplo con un “git add .”, las reglas del “gitignore” serán omitidas. Para eliminar un fichero de la gestión de GIT se debe ejecutar: git rm --cached FICHERO
  - Ver las plantillas de gitignore más comunes: https://github.com/github/gitignore

Uso básico de GIT

- Aunque en ocasiones basta simplemente con ignorar ficheros aplicando un conjunto de patrones en directorios concretos, es posible ignorar ficheros o directorios partiendo de patrones de manera global.
  - git config --global core.excludesfile /var/gitrepository/.gitignore_global

- Historial de GIT.
  - git log
  - git log --oneline
  - git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

- La creación de alias permite asignar atajos a comandos que son demasiado largos o que se utilizan con frecuencia. Para ello, es necesario editar el fichero $HOME/.gitconfig e incluir las opciones del alias:
  - [alias]
  - hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph –date=short
  - GLOBAL:
    git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"

Uso básico de GIT

- Recuperación de versiones:
  - git checkout “número de versión”
  - git checkout <número de versión> -b “número de versión”
  - git checkout master
- Etiquetas en versiones: Es mucho más simple etiquetar versiones concretas para acceder a ellas posteriormente en lugar de utilizar el hash que las identifica.
  - git tag version1
  - git checkout master
  - git tag version2
  - git tag
  - git hist master --all
  - git tag -d nombre_etiqueta

Uso básico de GIT

- Visualizando cambios: Para ver los cambios que se han realizado en el código se utiliza el comando “diff”. El comando mostrará los cambios que no han sido añadidos aún, es decir, todos los cambios que se han hecho antes de usar la instrucción “git add”. Después se puede indicar un parámetro y dará los cambios entre la versión indicada y el estado actual. O para comparar dos versiones entre sí, se indica la más antigua y la más nueva.
  - git diff master
  - git diff tag1 tag2
  - git diff tag1 master

Uso básico de GIT

- Deshacer cambios: Para deshacer cambios se puede utilizar el comando “checkout” sobre el fichero que se ha modificado para deshacer la última modificación.
  - git status
  - git checkout ficheromodificado
  - git status
- Si el fichero ya ha sido añadido al “staging” se puede deshacer el cambio con “reset”
  - git add fichero2
  - git status
  - git reset HEAD fichero2
  - git status

Uso básico de GIT

- En el caso de que ya se haya realizado un commit sobre un fichero y se quiera deshacer dicho cambio, se puede utilizar el comando “revert”.
  - git add fichero
  - git commit -m “Commit equivocado”
  - git revert HEAD --no-edit
- Cuando se debe añadir un cambio a un commit que se acaba de realizar se puede utilizar la opción “-a”. Por ejemplo, cuando se hace un commit y posteriormente hay que añadir más ficheros a esa revisión.
  - git add .
  - git commit -m “Commit 1”
  - touch ficheronuevo
  - git add ficheronuevo
  - git commit -a -m “Añadir ficheros nuevos”
- Los ficheros pueden moverse, eliminarse y añadirse dentro de un repositorio con los comandos “mv”, “rm” y “add” (explicado antes).
  - git add fichero.txt ; git rm fichero.txt ; git mv fichero,txt directorio

Uso básico de GIT: Gestión de ramas y de conflictos.

- Crear una nueva rama:
  - git branch
  - git branch nuevarama
  - git checkout nuevarama
- Es posible realizar commits sobre diferentes ramas y moverse entre ellas.
  - touch ficheronuevo ; git add ficheronuevo ; git commit
  - git checkout master ; git checkout hola
- Es posible fusionar los cambios de una rama con otra utilizando “git merge”
  - git checkout nuevarama
  - git merge master
- De esa forma se puede trabajar en una rama secundaria incorporando los cambios de la rama principal o de otra rama.

Uso básico de GIT: Gestión de ramas y de conflictos.

- Un conflicto en GIT ocurre cuando una fusión entre dos ramas no se puede resolver.
  - git checkout master ; nano ficherotestconflict (editar fichero)
  - git add fichero1 ; git commit -a -m “Commit en master”
  - git checkout nuevarama ; nano ficherotestconflict (editar fichero)
  - git add fichero1 ; git commit -a -m “Commit en merge”
  - git merge master
- En el ejemplo anterior hay un conflicto ya que se ha creado el mismo fichero en ambas ramas. Es necesario resolver los conflictos manualmente y posteriormente, hacer un “add” y “commit”.