Gestión de la configuración
Día 3

Curso 2024/25

Control de versiones

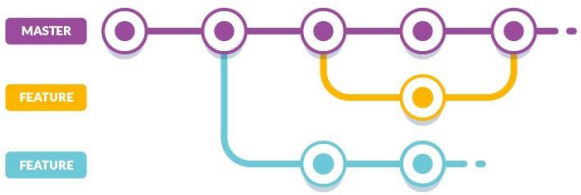
Git

Tareas de la Práctica

- Hecho
  - Documento inicial de control de cambios.
    - DP_ControldeCambios_GrupoN-v1.doc
  - Versión asociada a la revisión por pares.
    - DP_ControldeCambios_GrupoN-v2.doc
    - Creada con las solicitudes v1_1 aceptadas o rechazadas en v1_2:
      - DP_ControldeCambios_GrupoN-v1_1.pdf. *Solicitudes de cambio*
      - DP_ControldeCambios_GrupoN-v1_2.pdf. *Aceptación/rechazo solicitudes*
- Pendiente
  - Versión mejorada del proceso.
    - DP_ControldeCambios_GrupoN-v3.doc
- Presentación Git
  - Solo existirá: DP_ControldeCambios_GrupoN.doc

Control de versiones

- Objetivo: Seguir y mantener todas las versiones y modificaciones significativas en un documento.
  - Aunque el control de versiones se aplica sobre todo a código, sus principios, y frecuentemente sus herramientas, permiten su aplicación sobre cualquier tipo de documento.
- Alcanza su máximo sentido en desarrollos en equipo, de donde surgió. Aún así, su uso debe plantearse para cualquier desarrollo.

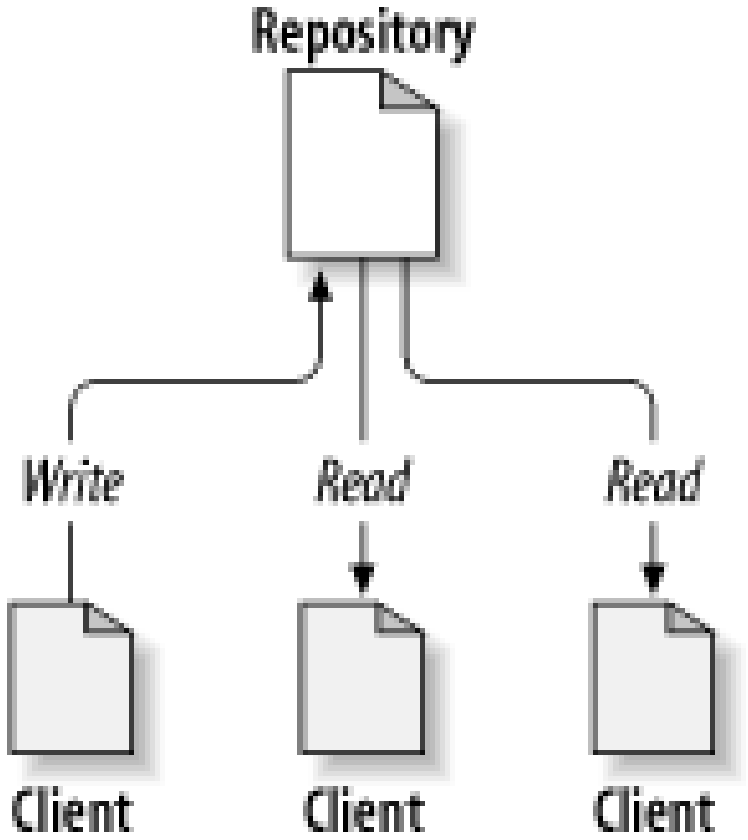


Herramientas

- El control de versiones es imprescindible en cualquier proyecto. Puede realizarse manualmente.
- Herramientas
  - CVS: Sistema centralizado original. Basado en fichero tiene fuertes limitaciones para el seguimiento de proyectos.
  - SVN: Sistema centralizado desarrollado tratando de resolver problemas en el CVS.
  - Git: Sistema distribuido desarrollado por Linus Torvalds usado en el mantenimiento del núcleo de linux. Puede usar repositorios svn.
  - Mercurial: Sistema distribuido implementado en Python originalmente para Linux.

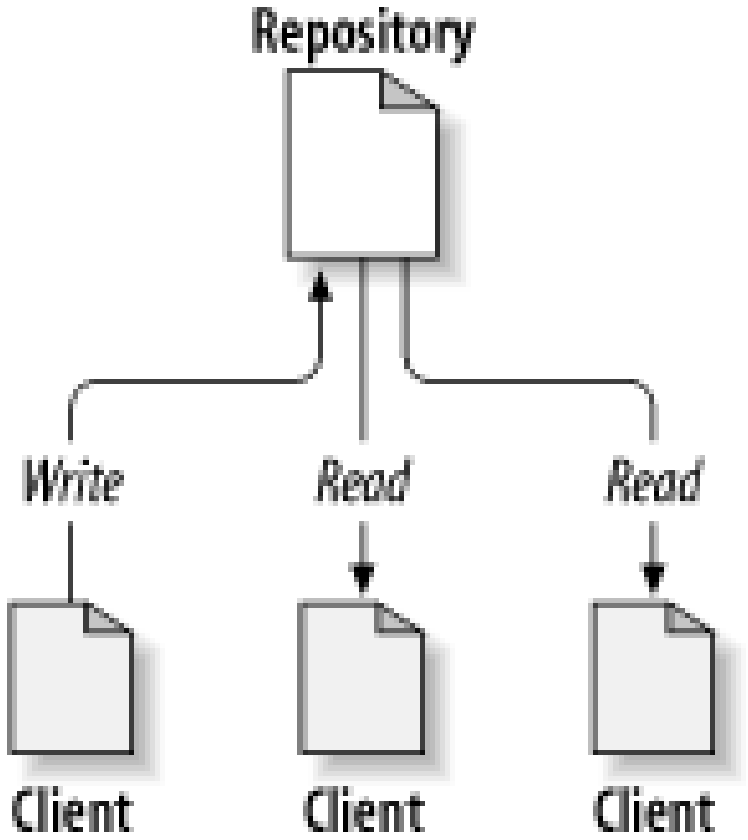
Repositorio

- Almacén "centralizado/distribuido" en el que compartir "toda" la información del proyecto
  - Permite el acceso de múltiples clientes, cada uno con su propia copia del repositorio
  - Información en árbol
  - Recuerda todos los cambios hechos sobre él. Incluidos los realizados sobre el propio árbol
  - El cliente puede ver estados previos del sistema



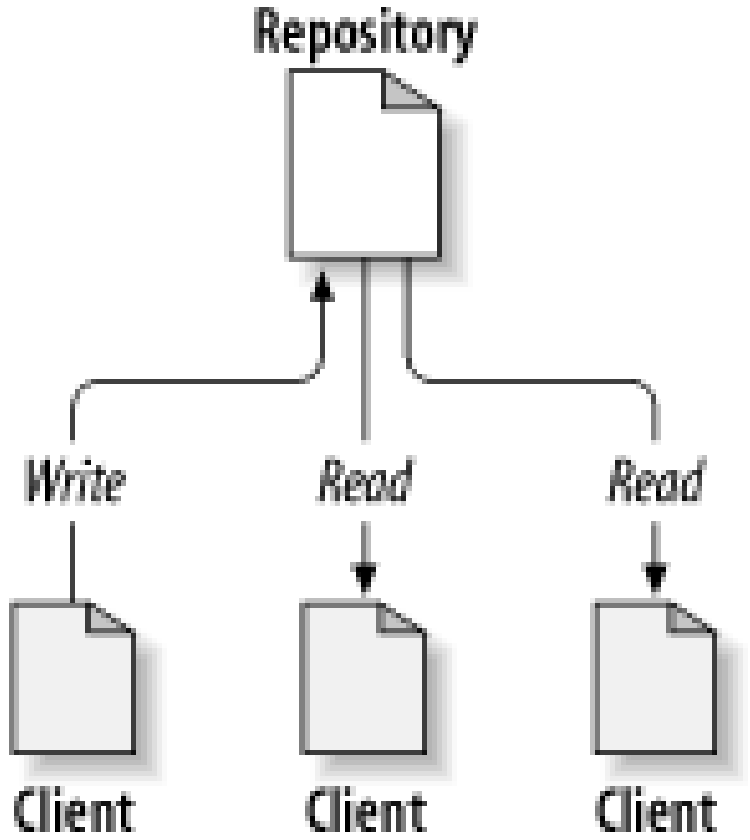
Repositorio

- Semejante al Drop-Box
  - La información está en remoto (repositorio) y local.
  - El repositorio guarda versiones de cada documento y es posible volver a ellas.
  - Cuando se modifica la copia en local se modifica en remoto.

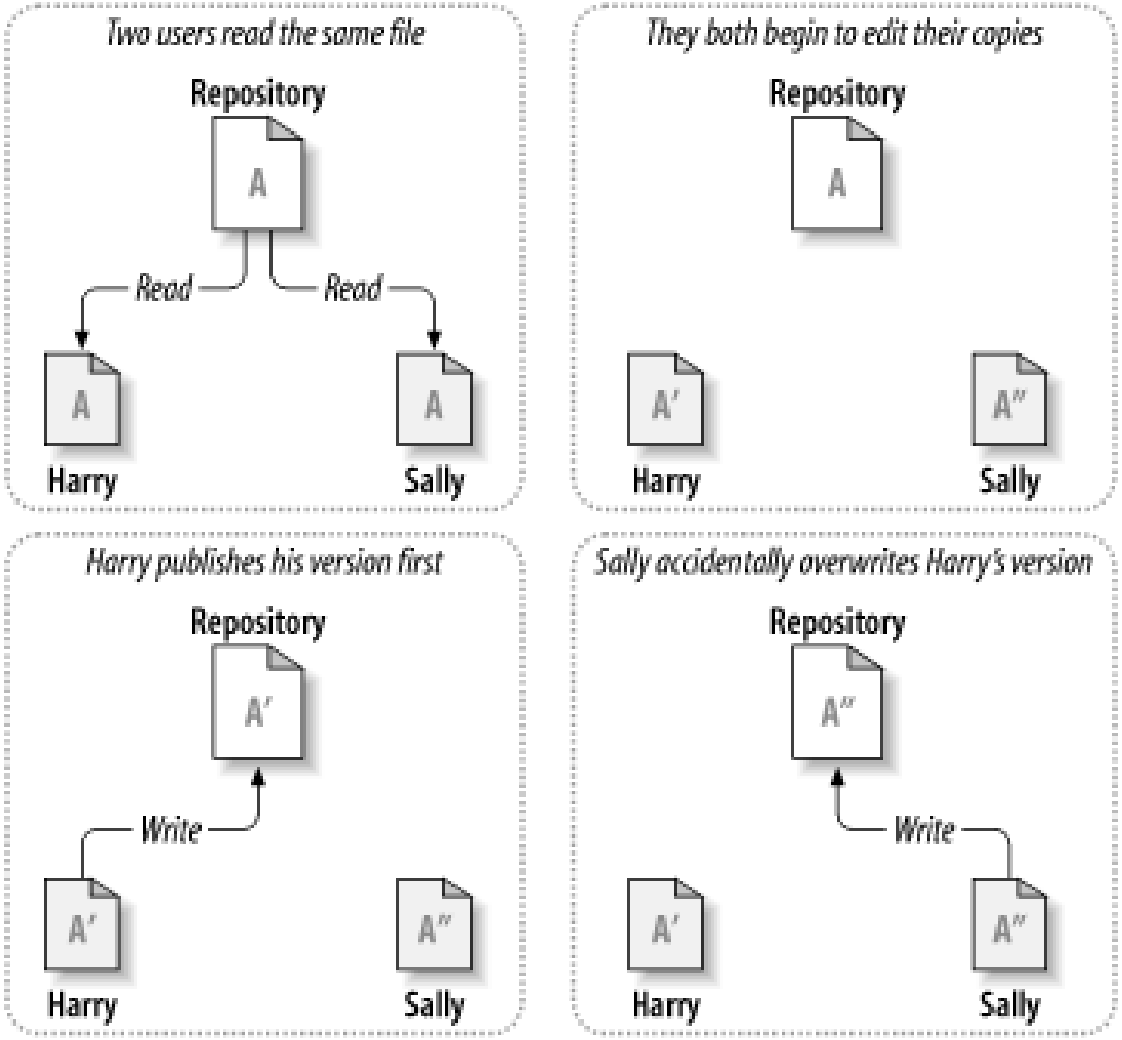


Repositorio

- Problemas en Drop-Box
  - Se guardan versiones intermedias.
  - Historia limitada
  - Los ficheros guardan versiones independientes.
  - ¿Cómo sabes que contenían otras versiones?. ¿En qué se diferencian de la actual?
  - ¿Qué ocurre si varios usuario cambian el mismo fichero?

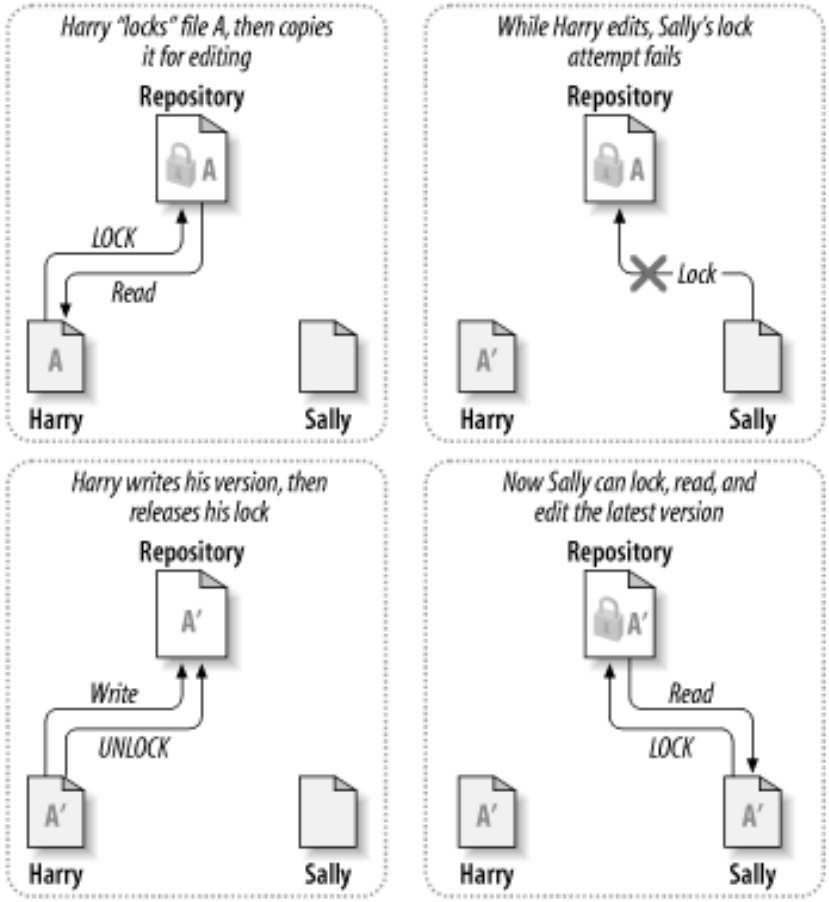


El problema a evitar

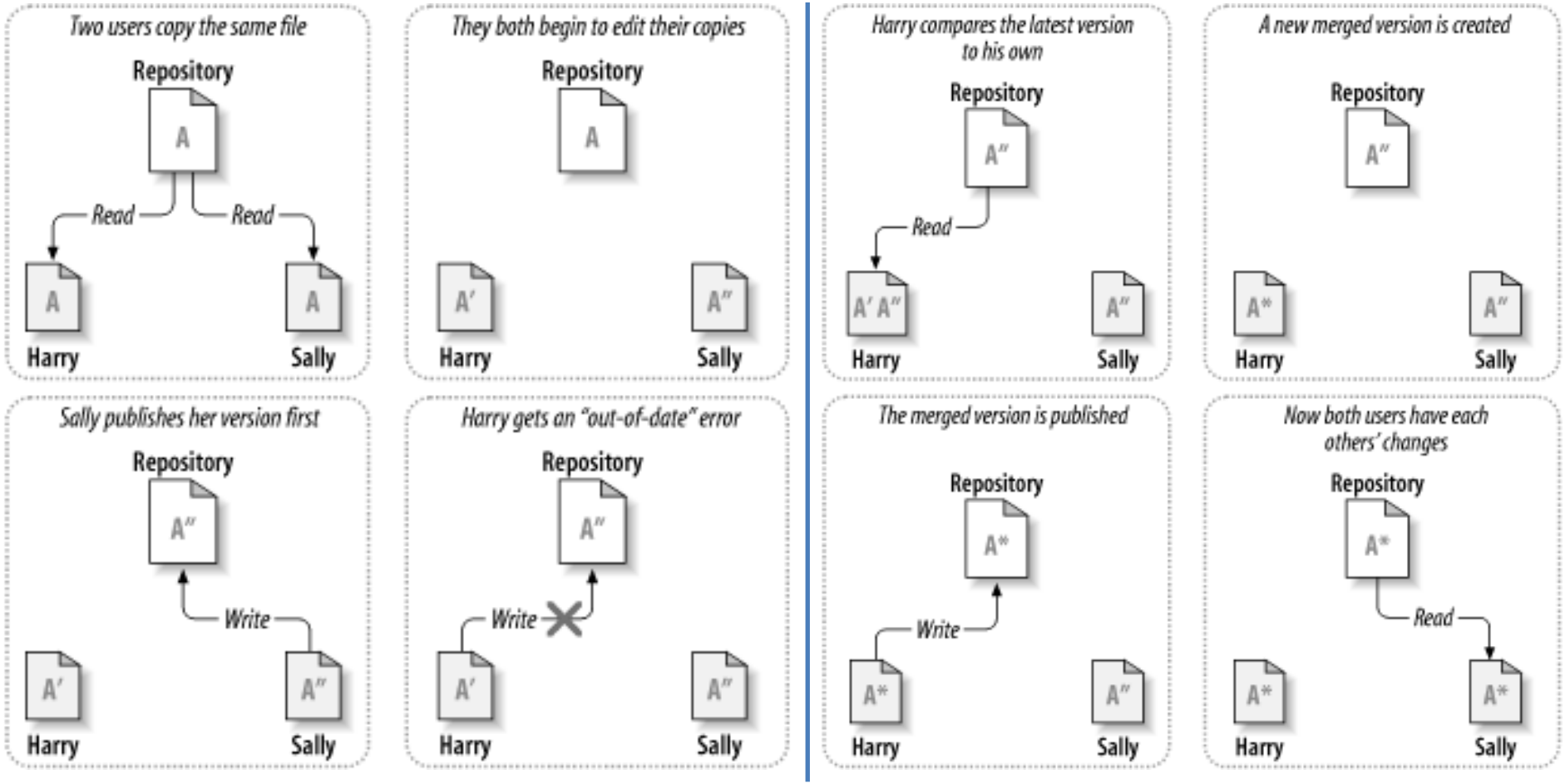


Solución: Bloqueo-modificación-desbloqueo

- **Bloqueos por tiempo indefinido**.
- **Serialización innecesaria**. Ej. No se pueden modificar dos métodos independientes en el mismo fichero.
- **Falsa sensación de seguridad**. Un código en varios ficheros A y B se puede modificar de forma incompatible



Solución: Copiar-modificar-mezclar



Git

Directorio de trabajo: Aquí tenemos los ficheros y directorios de nuestro proyecto. Es independiente del repositorio Git.

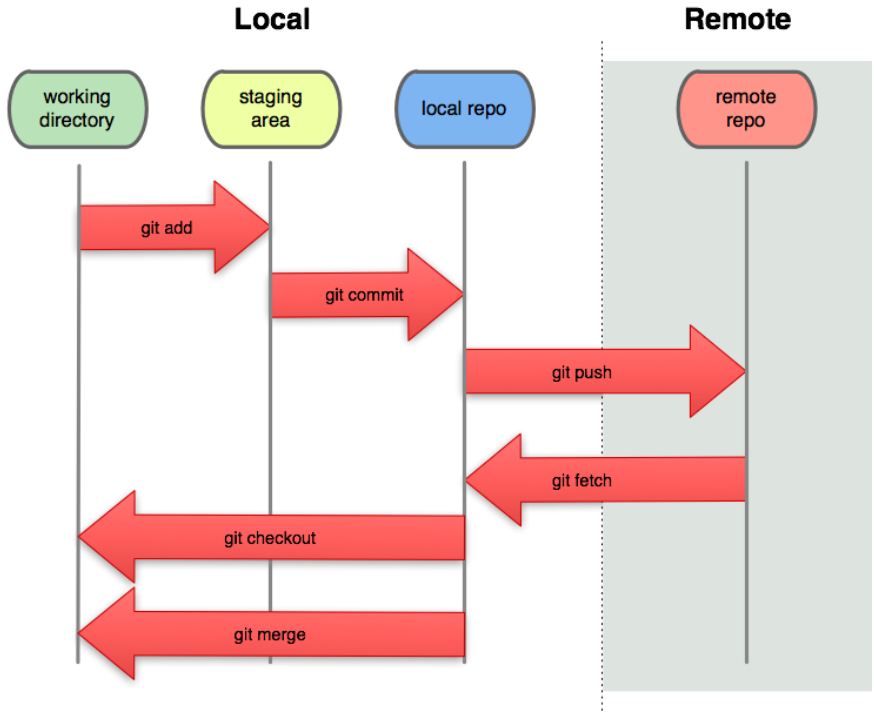
Repositorio: Aquí guardamos todos los ficheros del control de versiones y su historial. Inicializamos el repositorio con git init. Opera independientemente del directorio de trabajo. Dentro del repositorio tenemos:

- **Área de preparación:** Representa un punto intermedio entre el directorio de trabajo y la historia de commits. Representada por el fichero .git/index.
- **Historia de commits:** Un commit es una instantánea en un punto específico en el tiempo de nuestro proyecto. Cada commit tiene un hash único que lo identifica. La historia es la lista cronológica de commits.

Git: Ciclo de Trabajo

Las operaciones típicas que llevamos a cabo en un sistema de control de versiones son:

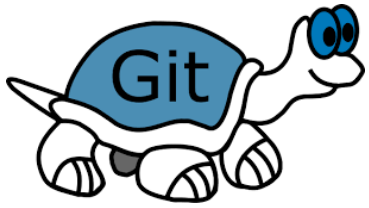
- Volver a una versión pasada
- Mezclar versiones
- Comparar versiones
- Enviar/traer



Github, Gitlab, Bitbucket son plataformas que nos ofrecen servicios de control de versiones en la nube.
Proveeen la posibilidad de gestionar repositorios públicos y privados.

¿Qué necesitamos?

- Git:
  - Herramienta de código abierto
  - Multiplataforma
  - https://git-scm.com
  - Podemos configurarlo: git config –list
  - Ayuda: git comando -h
- Cliente: TortoiseGit
  - Herramienta de código abierto
  - Basado en TortoiseSVN
  - Integrado con el explorador de Windows
  - Permite ejecutar todos los comandos Git desde su menú contextual



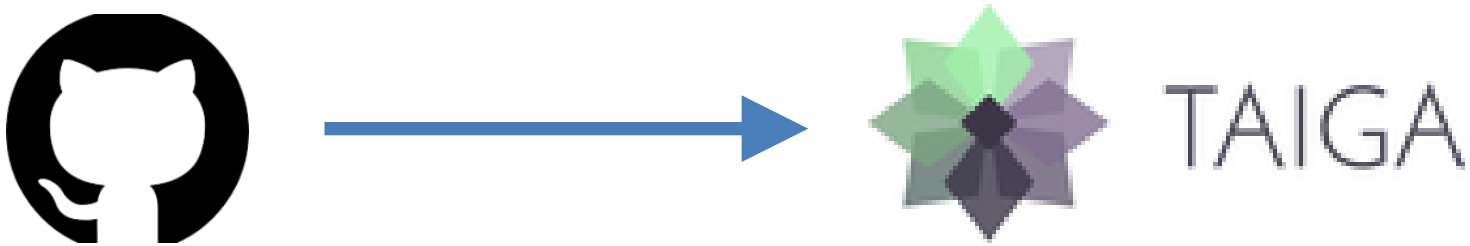
¿Qué necesitamos?

- Git:
  - Herramienta de código abierto
  - Multiplataforma
  - https://git-scm.com
- Cliente: GitHub Desktop
  - Herramienta de código abierto
  - Interface gráfico en Windows y Mac (otros sistemas Github CLI)
  - Permite conectarse con repositorios en GitHub, GitLab
- Acción: Cuenta en Github.

Conectando Taiga y Github

- Taiga: Settings-Integration-Github
- Github: Settings- Webhooks-Add webhook (application/json)

Ejemplo: Desde Github haced un commit referenciando historia de usuario #N con TG-N en los comentarios/descripción.



Flujo de trabajo

- Crear un repositorio
  - Desde un servidor remoto
    - Posicionarnos en el directorio
    - git clone url_servidor_remoto
  - En local
    - Posicionarnos en el directorio
    - **git init**
- Añadir directorios y ficheros
  - Crear una estructura según la filosofía que se desee
- Antes de hacer commit, tenemos que poner los elementos en la staging area:
  - git add . ó git add fichero
  - git commit –m “mensaje_commit”
- Historia de commits
  - git log
- Crear ramas
  - git brach nueva_rama
  - git switch nueva_rama
- Mezclar con rama principal
  - git switch master
  - git merge nueva_rama
- Tag
  - git tag –a v1.0 –m “Version…”
- Subir cambios al repositorio remoto
  - git push origin nombre_rama
- Otros:
  - git status
  - git diff
  - git reset <commit>

git help -g

Ramas y Etiquetas

- SVN: Una rama o branch es una copia del repositorio en un momento dado del desarrollo
  - La copia seguirá un desarrollo paralelo al tronco o la rama principal
  - En el futuro la rama puede abandonarse o volver a fusionarse con el tronco (**merge**)
  - Usamos deltas
- git: Una rama es un puntero a un determinado commit.
  - No se copian ficheros
  - Es muy rápido

  En cada commit, fichero se identifica por un hash.

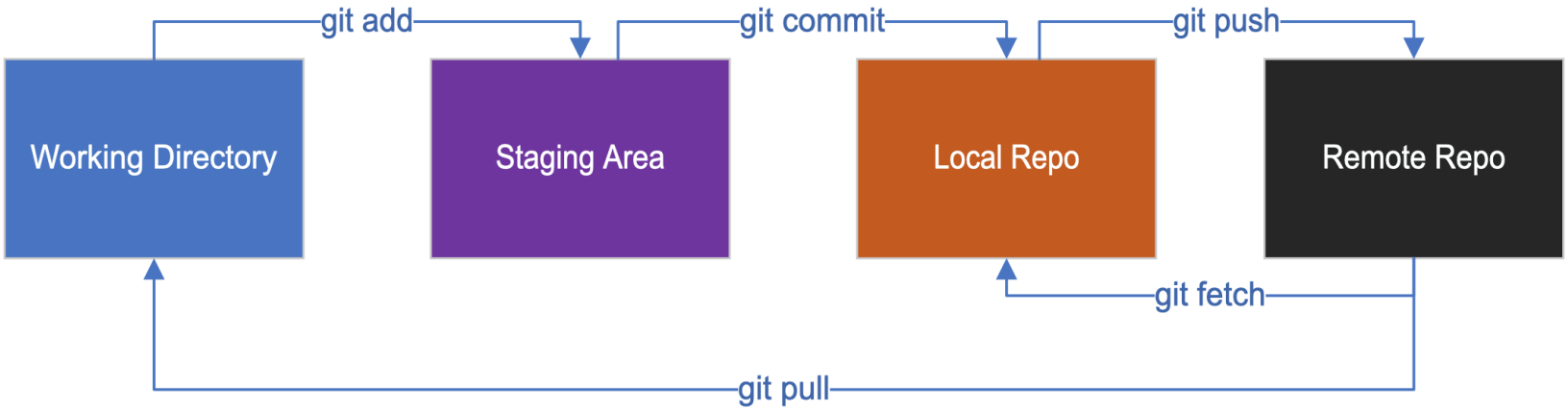
  Incluye metadatos sobre el autor, timestamp, mensaje, y referencia al estado del árbol raíz proyecto en ese momento.

Ramas y Etiquetas

- SVN: Etiquetas o Tags: Se corresponden con una copia del repositorio igual que una rama.
  - Una copia de Tag se marca con una etiqueta de referencia para indicar una versión especial. Release, estable, …
  - Una copia marcada **jamás se modifica.**
- Git: Etiquetas o Tags: Creamos una referencia a un commit específico.
  - Usamos git tag

Ciclo de trabajo

- Ciclo de trabajo
  - Pull
  - Commit
- Ciclo de trabajo con conflicto
  - Pull o fetch
  - Merge o rebase
  - Resolver los conflictos
  - Stage y Commit
  - Push



Flujo de trabajo Centralizado

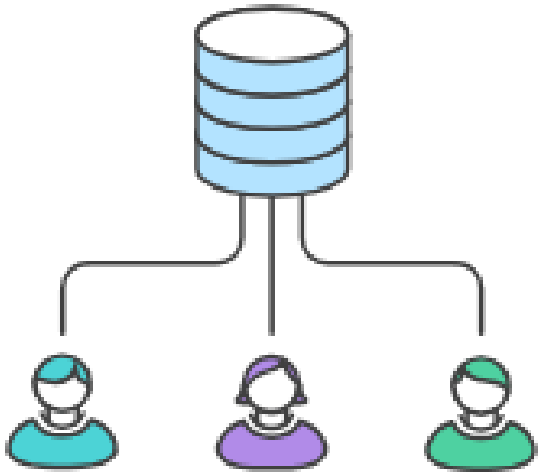
Repositorio Centralizado único

Todos hacen commits en el *main*.

main es la única rama.

Poca complejidad.

Utilizado por equipos pequeños o en proyectos legados que cambian de entorno (e.g. de SVN).



Pasos: Clonar repositorio central; Realizar cambios (add, commit,…), subir los cambios al repositorio central (git push origin main)
Si push falla, fetch del repositorio central y rebase commits locales por encima.

Flujo de trabajo: Feature Branch

Cada funcionalidad se desarrolla en una rama separada.

main no contiene versiones inestables.

Se mezclan en *main* a través de pull requests

Facilita la revisión y colaboración.

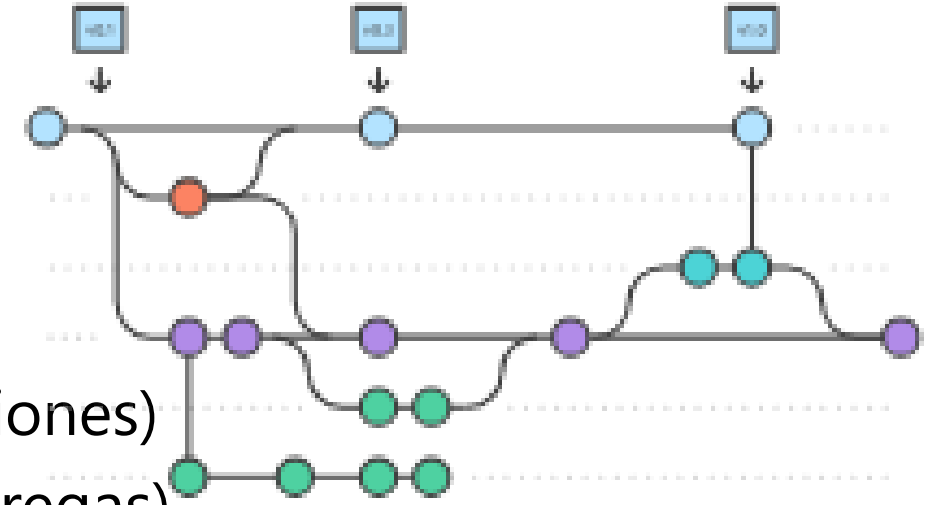
Popular en equipos que usan GitHub o GitLab.

Pasos: Crear una nueva rama; realizar cambios; subir la rama al repositorio remoto; pull request para integrar cambios en *main*.

Flujo de trabajo: Gitflow

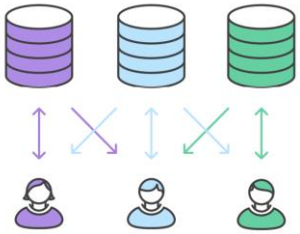
Múltiples ramas con roles específicos:

- Permanentes:
  - main (producción)
  - develop (desarrollo)
- Temporales:
  - feature/* (nuevas funciones)
  - release/* (finalizar entregas)
  - hotfix/* (urgentes)



Se usa en proyectos grandes, con entregas planificadas.

Flujo de trabajo: Fork



Cada miembro del equipo tiene una copia de su repositorio en el servidor (fork). Solo ese miembro puede hacer push, pero otros miembros pueden hacer pull.

Cada miembro hace un push a su repositorio remoto, pero solo el mantenedor del proyecto puede hacerlo al oficial.

Los cambios se envían como pull requests al repositorio principal para que el mantenedor conozca que hay una actualización disponible.

EL repositorio privado de cada miembro suele ser el *origin* y el repositorio principal suele denominarse *upstream*.

Popular en proyectos OS.

Crear Ramas y Etiquetas

- Crear una rama o branch del repositorio en un momento dado del desarrollo
  - Conflicto Harry-Sally
  - Refusionar rama con el tronco
- Crear una Etiqueta o Tag.
  - ~~Crear versiones~~
  - Una copia marcada **no debería modificarse.**

Github: Ejercicios

- Creamos cuenta en Github
- Creamos un repositorio de prueba
- Instalamos Github Desktop (GD). File/Options/Accounts
- Desde GD, clonamos nuestro repositorio de prueba. File/Clone
- En nuestro ordenador, modificamos/añadimos un fichero. Repository/Show in Explorer/…
- Hacemos commit desde GD
- Desde GD, push al servidor
- Veremos ese fichero en la página de Github
- Hacemos otro cambio en el mismo fichero en local, commit y push.
- Desde GD podemos revertir un commit: History, seleccionar el commit, revert changes (botón derecho),

Github: Ejercicios

- En GD, creamos una nueva rama. Branch/New
- En la main, cambiamos un fichero, commit a main y push
- En la nueva rama, añadimos un nuevo fichero.Commit a nueva rama y push
- Comprobamos que en el servidor vemos el nuevo fichero en la nueva rama.
- Vamos a mezclar la nueva rama y main.
  - Desde GD, seleccionar main como la rama actual.
  - Desde Branch, merge into current branch, y seleccionar la nueva rama.

  Todo bien? Y si hubiera conflicto? Simuladlo

  Podemos identificar los comandos git empleados? (Tip: Help/Show logs) ¿Podemos hacer lo mismo directamente desde git?

Estimaciones

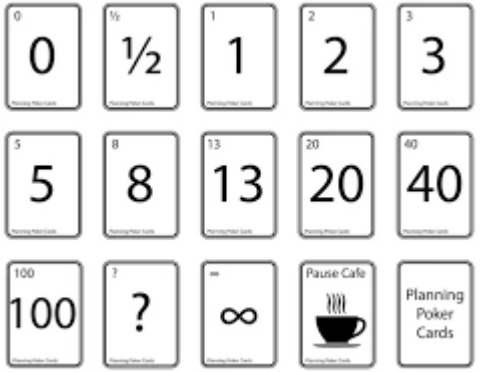
Poker SCRUM

Poker SCRUM

- Bibliografía
  - Henrik Kniberg. 2007. "*Scrum y XP desde las trincheras*".
    - http://www.proyectalis.com/wp-content/uploads/2008/02/scrum-y-xp-desde-las-trincheras.pdf
  - Juan Palacios. 2007. *Flexibiliad con SCRUM*.
    - http://www.scrummanager.net/files/flexibilidad_con_scrum.pdf
  - Ejemplo
    - http://www.crisp.se/planningpoker
  - Online: https://planningpokeronline.com

https://scrumpoker.online https://www.scrumpoker-online.org/es/

Poker SCRUM



- Estimaciones
  - Existen distintas técnicas. Se abordarán en "Gestión de Proxectos Informáticos"
    - Opinión de expertos
    - Descomposición:
      - Cuanto más pequeño y específico es un trabajo más fácil es hacer la estimación
      - Las tareas grandes se obtienen por la suma de las pequeñas en las que se descomponen. Bottom-Up y EDT

Poker SCRUM

- Utiliza la descomposición
- Obliga a todos los miembros del equipo a esforzarse en entender el problema.
  - Fuerza la participación activa de todos los miembros del equipo
    - Evita la "visión" del que mejor comprende el problema, del que tiene más capacidad de convencer al equipo o simplemente del que habla primero.
  - Permite detectar prematuramente discrepancias en la visión que tienen los miembros del equipo
    - Hace que surjan preguntas importantes sobre la comprensión del requisito de forma temprana.
- Involucra a todo el equipo
  - La información externa es orientativa pero lo realmente importante es la velocidad del equipo
    - El equipo conoce sus fortalezas y debilidades
  - Todo el equipo contribuye y comparte la decisión por lo que es más fácil el compromiso con la estimación.
- Se usa en metodologías ágiles para planificar de forma colaborativa el esfuerzo de las tareas, los sprints y refinar el backlog.

Poker SCRUM

¿Por qué lo necesitamos?

Reduce riesgos en la planificación del esfuerzo

Es ágil, rápido y ofrece resultados precisos.

Técnica colaborativa que tiene en cuenta diferentes perspectivas, evitando las opiniones dominantes y motivando la discusión y el consenso.

Puede funcionar en equipos distribuidos que usen herramientas online.

Poker SCRUM

- Baraja
  - Varias aproximaciones
    - Normalmente utilizan la serie de Fibonacci:
      - 1, 2, 3, 5, 8, 13, 21, 34. (No hay precisión en nº altos)
    - Añaden cartas con significado especial.
      - Ya está hecho, Ni idea, Café.
  - Nuestra Baraja
    - ½, 1, 2, 3, 5, 8, 13 (Son 2,5 semanas laborales)
    - Cartas especiales: Ni idea (?). Ya está hecho (0)

Poker SCRUM

- Antecedentes
  - Tenemos una lista de Cambios a Implementar.
  - En el coste se debe valorar
    - Esfuerzo de modificación de los documentos existentes.
    - Esfuerzo añadido al proyecto sobre la estimación inicial.
  - Concretar las unidades de las cartas en cada cambio (horas, días,…)
- Procedimiento
  - Tomamos la primera solicitud de cambio.
  - Todos los miembros ponen una carta OCULTA con su estimación.
  - Todas las cartas se revelan a la vez.
  - Resolver la votación.

Poker SCRUM

- Resolver la votación.
  - Acuerdo de convergencia
    - Hay convergencia si todas las votaciones se corresponden con 3 cartas consecutivas
    - Si no hay convergencia se discuten las posturas extremas y se vuelve a votar
  - Como tiempo final se propone el tiempo Pert.
    - Tiempo Pesimista (T_p): El más largo propuesto.
    - Tiempo optimista (T_o): El más corto propuesto.
    - Tiempo más Probable (T_{mp}): El más repetido.

$$T_M = \frac{T_p + 4 \cdot T_{mp} + T_o}{6}$$

	M1	M2	M3	M4	T_M
Vota 1					
Vota 2					

Poker SCRUM

- Ejemplos
  - Añadir el desbloqueo de una terminal de pago con la lectura de la huella digital.
  - Añadir a la página de web de la ETSE la funcionalidad que permita la reserva de locales para actividades extra-académicas.
  - Dotar a la web de la ETSE de un nivel de servicio 24x7 con una disponibilidad garantizada del 99%.

Proceso de Control de Cambios

OPTIMIZACIÓN

OPTIMIZAR PROCESOS

- Modificar el proceso para:
  - Utilizar git para controlar las versiones de los ficheros del proyecto.
    - Describir el repositorio.
    - Describir quién, cuándo y cómo se cambian los ficheros en el repositorio.
    - Describir quién, cuándo y cómo se etiqueta un estado del proyecto.
  - Usar poker Scrum para estimar el coste del cambio
    - Incluir la descripción detallada del proceso. Se asume que quién lee el proceso no sabe cómo se estima con poker Scrum.
    - Modificar las plantillas para que incluyan tablas de votaciones.
  - El análisis permite seleccionar que cambios de los actuales serán ejecutados y planificar su orden.
- Renombrar el documento.
  - DP_ControldeCambios_GrupoN-v3