Sistemas de Control de Versiones

1.1 Introducción

Elena García-Morato, Felipe Ortega, Enrique Soriano, Gorka Guardiola

GSyC, ETSIT. URJC.

Grupo de Innovación Docente Laboratorio de Ciencia de Datos para la Innovación de la Enseñanza (DSLAB-TI)

6 de julio, 2023

DSLAB-TI
DATA SCIENCE LABORATORY
FOR TEACHING INNOVATION

Universidad
Rey Juan Carlos

Sistemas de Control de Versiones

1.1 Introducción

Contenidos

1.1.1 El problema del control de versiones

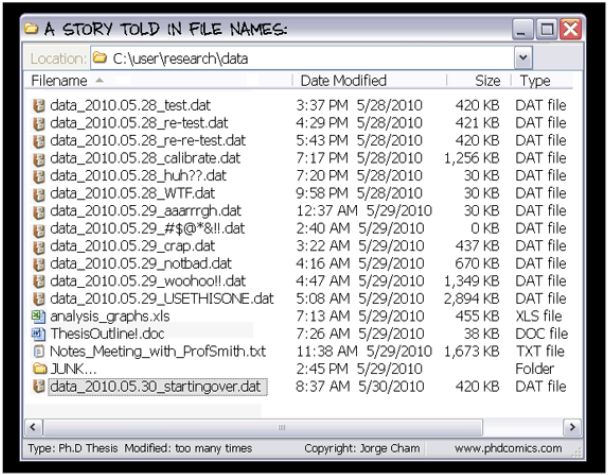
Equipos de desarrollo software

- Los proyectos de desarrollo de software involucran a un equipo con muchas personas.
- Además, con gran frecuencia cada miembro del equipo está en una ubicación diferente (sede de la empresa, teletrabajo, diferentes sucursales, contribuidores externos, etc.).
- El proceso de desarrollo de software es muy complejo:
  - Diferentes equipos (*hardware*) y sistemas operativos (*software*).
  - Instalación de dependencias y requisitos previos (paquetes, plug-ins, otros programas, etc.).
  - Múltiples tareas a resolver, muchas con dependencias cruzadas.
  - Varias personas trabajando en la misma tarea, editando los mismos archivos.

Equipos de desarrollo software

Desarrollador de software intentando arreglar un error crítico en el proyecto...

La situación que debemos evitar...



Fuente: https://phdcomics.com/comics/archive_print.php?comicid=1323.

Equipos de desarrollo software

- Debemos realizar muchos cambios sobre varios ficheros de todo tipo:
  - Código fuente.
  - Archivos de configuración.
  - Tareas automatizadas (compilación, enlazado, limpieza,...).
  - Archivos de datos.
  - Documentación y ejemplos de uso.
- Es muy fácil perder el control sobre lo que hemos cambiado, cuándo lo hemos modificado, qué combinación funcionaba, quién hizo cada acción, qué cambios son válidos y cuáles debemos descartar, etc.
- El desarrollo de software ya es suficientemente complicado por sí mismo como para añadir más caos al proceso.

Registro de cambios

- Idea principal: hacemos una modificación o unas pocas modificaciones que afectan a mismo elemento o funcionalidad del programa software y las confirmamos.
  - Los cambios deben estar claramente relacionados entre sí y tener un tamaño manejable.
  - El autor/a se responsabiliza del cambio confirmado.
  - Si la tarea es muy complicada la desgloso en tareas más simples.
  - Debemos acostumbrarnos a hacer cambios atómicos.
- Una vez que compruebo que las modificaciones son correctas entonces *confirmamos* la modificación (que llamaremos *commit*).
- Necesito un programa que vaya registrando las modificaciones que hacemos sobre el proyecto y vaya creando "fotos" del nuevo estado del proyecto tras cada modificación: un **Sistema de Control de Versiones (SCV)**.

Registro de cambios

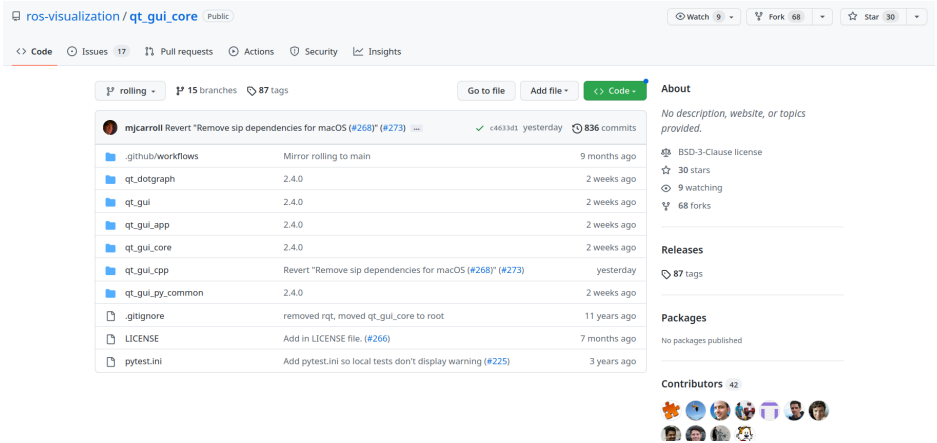


Figura 1: Repositorio del proyecto ROS-Visualization/Qt_GUI_core en GitHub.

Auditoría

- Un sistema de control de versiones me permite revisar o auditar el histórico de cambios que se han ido introduciendo en un proyecto software.
  - Quién o quiénes que introdujeron el cambio.
  - Quién o quiénes revisaron el cambio antes de aceptarlo (opcional, para proyectos muy críticos).
  - Qué archivos se modificaron y en qué consistió cada modificación.
- El historial de cambios mantiene en la mayoría de casos una *cronología* precisa. Es una especie de "máquina del tiempo", que me permite volver al estado del proyecto en cualquier punto del pasado.

Registro de cambios

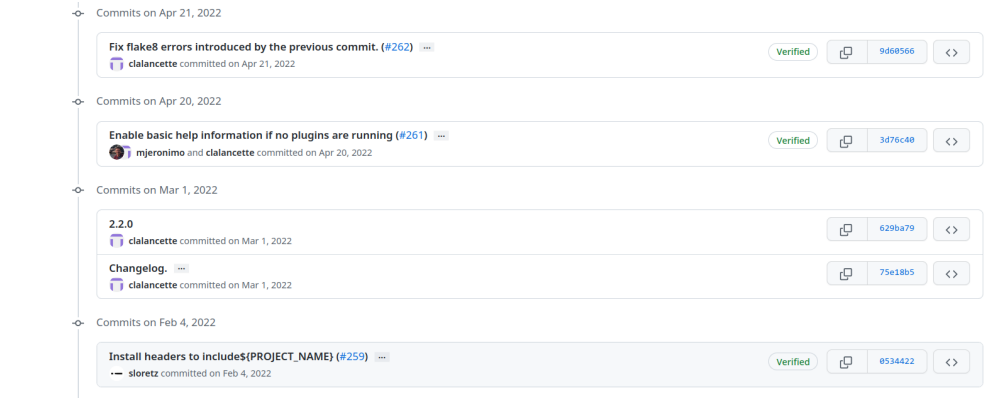


Figura 2: Detalle de una parte del registro de cambios en el repositorio del proyecto ROS-Visualization/Qt_GUI_core en GitHub.

Registro de cambios

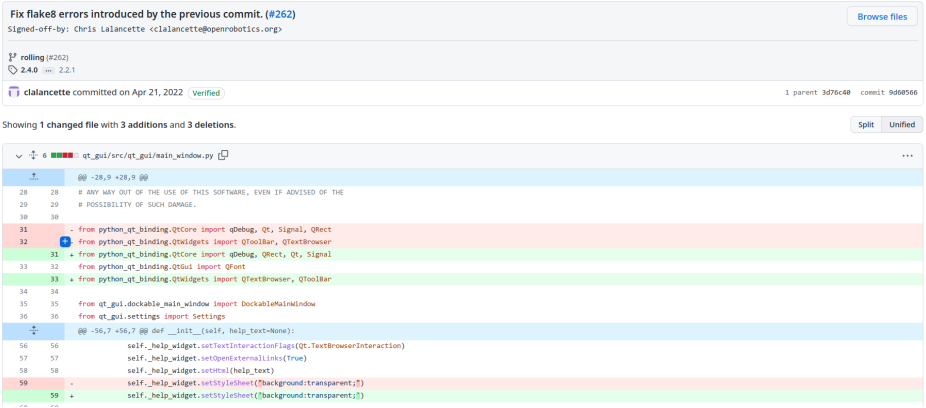


Figura 3: Cambios introducidos por un *commit* en un fichero del proyecto ROS-Visualization/Qt_GUI_core en GitHub.

Corrección de errores y petición de mejoras

- Es habitual que tengamos que solucionar errores o problemas en nuestro proyecto.
- Algunos aparecen conforme se añaden nuevas funciones y características.
- Otros surgen por cambios o mejoras que rompen cosas que ya funcionaban.
- También se pueden recibir comentarios y peticiones para introducción de nuevas mejoras y modificaciones.
- Un sistema aparte (ITS o *Issue Tracking System*) registra los avisos y las conversaciones asociadas a su procesamiento. Se pueden enlazar los cambios registrados a la resolución de cada error o los que responden a una petición de mejora específica.

Corrección de errores y petición de mejoras

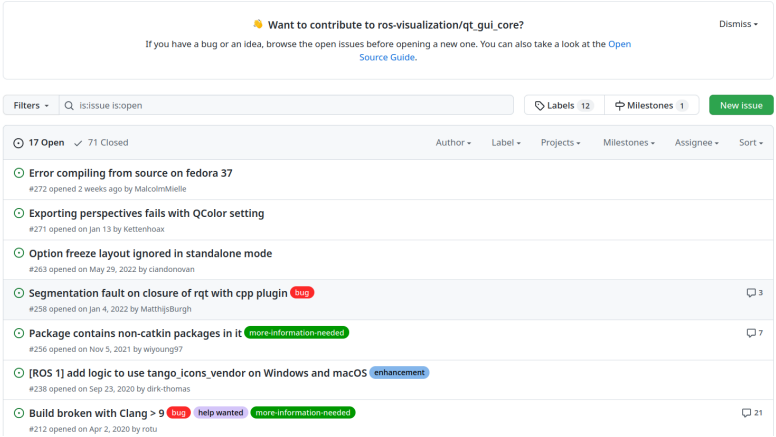


Figura 4: Sistema de informes de error y mejoras en el proyecto ROS-Visualization/Qt_GUI_core en GitHub.

Clonado, *fork* y contribuciones

- Es fácil clonar un proyecto para obtener una copia local del mismo y trabajar sobre ella.
- También es sencillo hacer un fork: se crea otra copia remota de mi proyecto que luego puedo clonar localmente para mantener los cambios que vaya haciendo.
- Por último, los programadores que trabajan con sus copias pueden solicitar la integración de sus cambios en el proyecto principal (lo que veremos como un *pull request*).

Fork de un proyecto

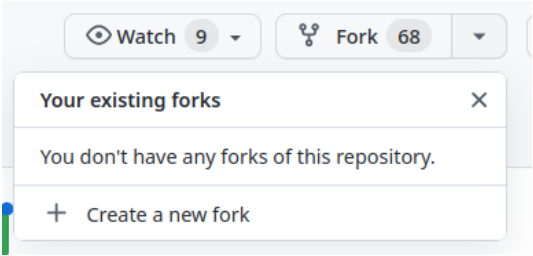


Figura 5: Detalle de la funcionalidad para crear *forks* de un proyecto en GitHub.

Versiones de lanzamiento o (*releases*)

- Cuando está lista una versión de lanzamiento (*release*) del software, con los archivos que la componen en un estado de desarrollo *maduro* y *probado*, se puede marcar ese punto y publicarlo.

- Se siguen algunas convenciones para identificar las versiones de software:

  - En muchas ocasiones (no siempre) se usan tres número separados por puntos, un esquema llamado *semantic versioning*.
  - Por ejemplo: `4.2.1`: 4 $\rightarrow$ `MAJOR`; 2 $\rightarrow$ `Minor`; 1 $\rightarrow$ `patch`.
  - Un nuevo número `MAJOR` indica cambios muy significativos (potencialmente pueden romper cosas o no ser compatibles con versiones anteriores). Un nuevo `MINOR` indica nuevas funcionalidades que no introducen riesgo. Un nuevo `PATCH` indica resolución de problemas.
  - La cifra `0` en posición `MAJOR` indica software no maduro o no estable aún (versión alfa o beta).

1.1.2 Sistemas de Control de Versiones (SCVs)

Propósito de un SCV

- Es un software que nos permite mantener un conjunto de archivos bajo control de versiones.
- En nuestro sistema de ficheros siempre "vemos" solo la versión más actualizada de los ficheros, por ejemplo, si hacemos un `ls -l` en la terminal Linux.
- Sin embargo, internamente el SCV mantiene un registro de todos los cambios efectuados en cada archivo, lo que permite ver el camino completo de evolución de mi proyecto (e incluso representarlo gráficamente).
- También me permite volver a puntos anteriores, auditar los cambios para descubrir quién los hizo o corregir errores introducidos y muchas otras funciones útiles.

Propósito de un SCV

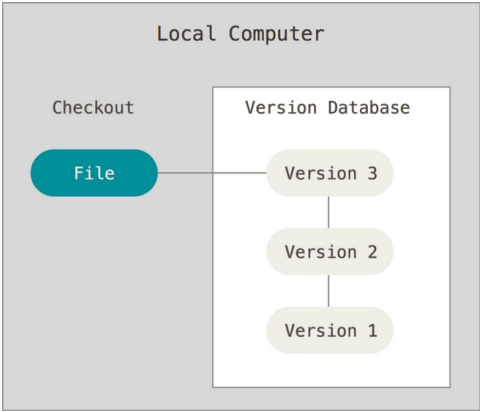


Figura 6: Esquema del comportamiento de un SCV local.

Fork de un proyecto

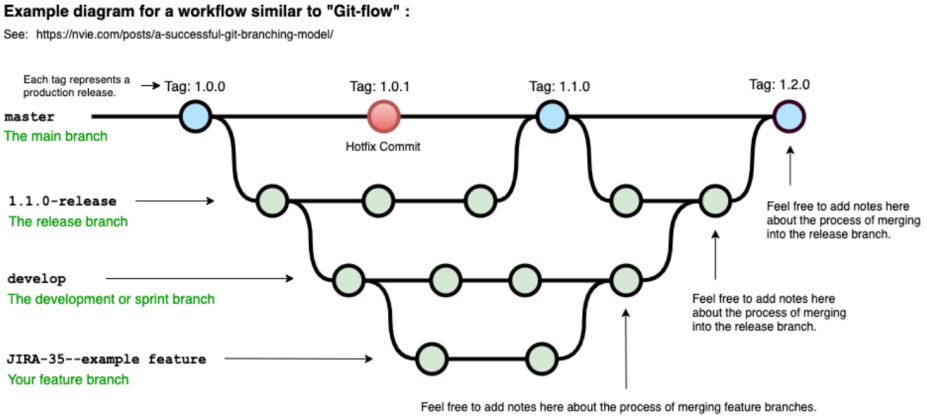


Figura 7: Ejemplo ficticio de cómo se podría representar un grafo de cambios en un SCV.

SCV centralizado vs. distribuido

- Los primeros SCV mantenían una copia **centralizada** (copia maestra) del proyecto.
- Todos los participantes tenían que trabajar sobre esa copia.
- Problema grave: si dos o más personas trabajan sobre el mismo archivo haciendo cambios diferentes e intentan actualizar la copia maestra con sus respectivas modificaciones no funciona.
- Los SCV **distribuidos** permiten que cada persona trabaje con su propia copia local del proyecto completo y haga cambios en ella. Incluso puedo trabajar *sin conexión a la red*.
- Cuando envío los cambios para que el resto los vean me preocupo de solucionar los posibles conflictos detectados, mezclando mis cambios (*merge*) y confirmándolos (*commit*).
- Más detalles en esta breve introducción (incluye vídeo).

Ejemplo de SCV: Git

- Git apareció en 2005 como respuesta de la comunidad que desarrollaba el kernel de Linux a la conversión en software no gratuito de la herramienta que usaban para gestionar el proyecto.
- Breve historia de Git.
- Algunos objetivos:
  - Velocidad.
  - Diseño sencillo.
  - Buen soporte para poder introducir muchos cambios en paralelo (desarrollo no lineal).
  - Totalmente distribuido.
  - Manejo eficiente de proyectos de gran tamaño (velocidad, transferencia de datos).

Otro ejemplo de SCV: Mercurial (hg)

- También surgió en 2005 como sistema distribuido de control de versiones en el seno de la comunidad de desarrollo del kernel Linux, solo unos días después del anuncio de git.
- Aunque no fue seleccionado como SCV oficial del kernel Linux, muchos proyectos y organizaciones lo utilizan actualmente, notablemente Meta (Facebook), el W3C y la Fundación Mozilla.
- Comparte con Git muchos de sus objetivos de diseño.
  - Escalabilidad y alto rendimiento.
  - Completamente descentralizado.
  - Soporte para archivos de texto plano y binarios.
  - Líneas de desarrollo paralelas y capacidad avanzada de integración de cambios.

1.1.3 Servicios SCV

GitLab

- https://gitlab.com/.
- Es un servicio web para alojamiento y desarrollo de proyectos software (denominado con frecuencia como forja).
- Incluye soporte para control de versiones mediante el SCV git.
- También incorpora herramientas para DevOps, como pruebas automáticas, integración continua, monitorización de rendimiento, etc.
- Tiene una versión libre (CE) y otra privativa (EE) con características exclusivas.

GitHub

- https://github.com/.
- Es una *forja* de desarrollo software que se ofrece como servicio web y utiliza el SCV git.
- Fundada en febrero de 2008, actualmente es propiedad de Microsoft, que compró la compañía en 2018 por un importe de +$7.000M.
- Hoy día es la plataforma más popular para desarrollo colaborativo de proyectos de software libre.

Lecturas sugeridas

Para saber más

- El sitio web `https://git-scm.com/` es una fuente de información de referencia sobre el control de versiones, en particular utilizando el software git.
  - Incluye un manual de referencia muy completo, que detalla tanto uso de comandos como esquemas de trabajo y buenas prácticas.
- En la misma web se ofrece gratuitamente acceso al libro de referencia "Pro Git" [1].
- El curso `https://happygitwithr.com/` de Jennifer Bryan introduce al usuario de R y RStudio cómo usar Git y GitHub en sus proyectos con este lenguaje.

Referencias I

[1] S. Chacon y B. Straub. *Pro Git*. The Expert's Voice. Apress, 2014.