

GIT COMO PLATAFORMA DE APOYO DOCENTE

Hoy, para un puesto de trabajo como desarrollador debe utilizar Git. Como freelance es importante conocer Git para el control de versiones de sus desarrollos.

Con un SCV si detecta un cambio que está generando un error puede revertirlo y volver a un estado anterior. Además, crea ramas para probar cosas y si funcionan estos cambios, los fusiona con la rama principal.

GitHub, GitLab o Git es una plataforma centrada en el desarrollo de software colaborativo, pero una persona, estudiante o profesional que utilice un dispositivo informático es un "trabajador del conocimiento". Por tanto, puede beneficiarse de esta red social.

Los puntos presentados son básicos, y otros específicos, sofisticados, u "oscuros".

El libro no es largo, pero si aprende los puntos aquí tratados, sabrá sobre configuración, administración de usuarios y grupos, accesos y rutas, ramas y sobre aspectos de seguridad en el servidor.

Lo que necesita saber de →



JORGE DOMÍNGUEZ CHÁVEZ

Publicado por



Git como plataforma de apoyo docente
JORGE DOMÍNGUEZ CHÁVEZ
UNIVERSIDAD POLITÉCNICA TERRITORIAL DE ARAGUA



Venezuela, 2020

Copyright © Jorge Domínguez Chávez.

ORCID: 0000-0002-5018-3242

Esta obra se distribuye licencia Creative Commons:



<http://creativecommonsvenezuela.org.ve>

Reconocimiento:

- Atribución: Permite a otros copiar, distribuir, exhibir y realizar su trabajo con Derechos de Autor y trabajos derivados basados en ella – pero sólo si ellos dan créditos de la manera en que usted lo solicite.
- Compartir igual: Permite que otros distribuyan trabajos derivados sólo bajo una licencia idéntica a la que rige el trabajo original.
- Adaptar: mezclar, transformar y crear a partir de este material.

Siempre que lo haga bajo las condiciones siguientes:

- Reconocimiento: reconocer plenamente la autoría de Jorge Domínguez Chávez, proporcionar un enlace a la licencia e indicar si se ha realizado cambios. Puede hacerlo de cualquier manera razonable, pero no una que sugiera que tiene el apoyo del autor o lo recibe por el uso que hace.
- No Comercial: no puede utilizar el material para una finalidad comercial o lucrativa.

© 2020, Domínguez Chávez, Jorge

ISBN 9789806366091



Publicado por IEASS, Editores

ieass@gmail.com

Venezuela, 2020

La portada y contenido de este libro fue editado y maquetado en TeXstudio 2.12.16

Índice general

Resumen	I
Terminología	II
1. Introducción	IV
1.1. Introducción a git	V
1.2. Los tres estados	VI
1.3. Definición, clasificación y funcionamiento	VI
1.4. Clasificación	VII
1.4.1. Locales	VII
1.4.2. Centralizados	VII
1.4.3. Distribuidos	VIII
2. GIT como plataforma de apoyo docente	1
2.1. Aprendizaje del estudiante	1
2.2. Soporte del curso	3
3. GitHub	5
3.1. Creación y configuración de la cuenta	5
3.2. Acceso SSH	6
3.3. Tu icono	8
3.4. Tus direcciones de correo	9
3.5. Autenticación de dos pasos	9
3.6. Mantenimiento de un proyecto	10
3.6.1. Creación de un repositorio	10
3.6.2. Añadir colaboradores	12
3.6.3. Gestión de los Pull Requests	13
3.6.4. Notificaciones por correo electrónico	13
3.6.5. Colaboración en el Pull Request	14
3.6.6. Referencias de Pull Request	15
3.6.7. Pull Requests sobre Pull Requests	17
3.6.8. Menciones y notificaciones	17
3.6.9. Página de notificaciones	18
3.6.10. NOTIFICACIONES WEB	19
3.6.11. NOTIFICACIONES POR CORREO	20
3.6.12. Archivos especiales	20
3.7. Administración del proyecto	21
3.7.1. Cambiar la rama predeterminada	21

3.7.2.	Rama predeterminada	22
3.7.3.	Transferencia de un proyecto	22
3.7.4.	Transferir	22
3.8.	Gestión de una organización	22
3.9.	Conceptos básicos	22
3.10.	Equipos	23
3.11.	Auditorías	25
4.	Antecedentes	26
4.1.	Académico	26
4.2.	Tecnológico	27
4.3.	La unión	28
4.4.	Git colaborativo	29
4.5.	GitHub no es sólo para desarrolladores de software	29
5.	Operaciones básicas con repositorios locales	31
5.1.	Configuración local	31
5.2.	Operaciones básicas con repositorios remotos	34
5.3.	Escenario colaborativo basado en control de acceso	36
5.4.	Escenario colaborativo basado en merge requests (pull requests en GitHub)	38
5.4.1.	origin y upstream	39
5.5.	Integrando un repositorio remoto	39
5.5.1.	Comandos Git para Cree ramas	40
5.5.2.	Fusionando el contenido de una rama en otra	40
5.5.3.	Comandos Git para deshacer los cambios: revert	40
5.6.	Recomendaciones	41
6.	Ejercicios selectos	43
	Conclusiones	44
	Bibliografía	45

Índice de figuras

1.1. Modelo de datos de Git	V
1.2. Directorio de trabajo, área de preparación, y directorio de Git	VI
1.3. Sistema de control de versiones local	VII
1.4. Sistema de control de versiones centralizado	VIII
1.5. Sistema de control de versiones distribuido	IX
2.1. Flujo de trabajo en grupo utilizando Git-Hub/Git	2
2.2. Informes proporcionados por la plataforma GitHub	4
3.1. Formulario para darse de alta en GitHub.	6
3.2. Enlace " Account settings"	7
3.3. Enlace " SSH keys"	7
3.4. Enlace " Profile"	8
3.5. Recortar tu icono	8
3.6. Añadiendo direcciones de correo	9
3.7. 2FA dentro de Security.	10
3.8. Desplegable "New repository".	11
3.9. Formulario para crear repositorio.	11
3.10. Enlace a ajustes del repositorio.	12
3.11. Colaboradores del repositorio.	13
3.12. Notificación por correo de nuevo Pull Request.	13
3.13. Las respuestas a correos se incluyen en el hilo de discusión.	14
3.14. Botón Merge e instrucciones para fusionar manualmente un Pull Request.	15
3.15. Cambio manual de la rama o del fork en un pull request.	17
3.16. Empieza tecleando @ para mencionar a alguien.	18
3.17. Quitar suscripción de un pull request o incidencia.	18
3.18. Opciones de Notification center.	19
3.19. Centro de notificaciones.	19
3.20. Apertura de un Pull Request cuando existe el archivo CONTRIBUTING.	21
3.21. Cambio de la rama predeterminada del proyecto.	21
3.22. Transferir propiedad de un proyecto.	22
3.23. El menú "New organization".	23
3.24. Página de la organización.	24
3.25. Página de equipos.	24
3.26. Log de auditoría.	25
5.1. addcommit	31

5.2.	addsshkey	34
5.3.	clone	35
5.4.	addmembers	37
5.5.	unprotectbranch	37
5.6.	allowpush	37
5.7.	colaboración	38
5.8.	upstream	39

Resumen

Los sistemas de la administración de la enseñanza (LMS) son de amplio uso tanto entre los profesores como entre los estudiantes para asistir a las actividades no presenciales de la clase; si bien el esfuerzo necesario para conformar las actividades relacionadas a un curso no es trivial, al brindar no únicamente la capacidad de recoger la interacción, sino que generar automáticamente calificaciones, resultan muy atractivos para los profesores.

El sistema de control distribuido de versiones Git se ha convertido en una herramienta esencial para manejar proyectos de software. Uno de los motivos de la creciente popularidad de Git es el éxito de GitHub, una plataforma Web de desarrollo colaborativo basada en Git.

GitHub ofrece toda la funcionalidad de Git e integra diversas herramientas de control de acceso, colaboración, trazabilidad, gestión de tareas y control de proyectos.

Recientemente, profesores dentro y fuera del mundo académico relacionado con la Informática han comenzado a usar GitHub en sus cursos. Esta contribución presenta una experiencia docente desarrollada en unidades curriculares relacionadas con la ingeniería de software en la que GitHub se ha utilizado como la herramienta académica básica para el desarrollo de la parte práctica.

La intención del autor con este texto es presentar la justificación para el elegir esta plataforma en particular para muchas de las unidades curriculares en el Plan Nacional de Formación en Informática del Departamento de Informática de la Universidad Politécnica Territorial del estado Aragua; si bien la UPT Aragua tiene una amplia experiencia de 18 años en e-learning, primero con A-tutor y posteriormente con Moodle.

Este aporte se centra en motivar la utilización de Git, explicar su implementación, evaluar los beneficios y riesgos potenciales que conllevan a identificar nuevos retos.

Terminología

Repositorio (" repository")	El repositorio almacena los datos actualizados e históricos de cambios.
Revisión (" revision")	Es una versión determinada de la información que se gestiona. Hay sistemas que identifican las revisiones con un contador. Hay otros, que las identifican por un código de detección de modificaciones (git usa SHA1).
Etiqueta (" tag")	Los tags identifican las revisiones importantes en el proyecto. Se usan tags para identificar el contenido de las versiones publicadas del proyecto.
Rama (" branch")	Un conjunto de archivos puede ser ramificado o bifurcado en un punto en el tiempo de manera que, a partir de ese momento, dos copias de esos archivos se desarrollen a velocidades diferentes o de forma independiente el uno del otro.
Cambio (" change")	Un cambio (o diff, o delta) es una modificación de un documento bajo el control de versiones. La granularidad de la modificación es un cambio que varía entre los sistemas de control de versiones.
Desplegar (" checkout")	Crea una copia de trabajo local desde el repositorio. Un usuario puede especificar una revisión en concreto u obtener la última. El término 'checkout' se utiliza como un sustantivo para describir la copia de trabajo.
Confirmar (" commit")	Escribe o mezcla los cambios realizados en la copia de trabajo del repositorio. Los términos 'commit' y 'checkin' también se utilizan como sustantivos para la nueva revisión que se crea como resultado de confirmar.
Conflicto (" conflict")	Un conflicto se produce cuando diferentes partes realizan cambios en el mismo documento, y el sistema es incapaz de conciliar los cambios. Un usuario debe resolver el conflicto mediante la integración de los cambios, o por la selección de un cambio en favor del otro.
Cabeza (" head")	También se llama tip (punta) y se refiere a la última confirmación, ya sea en el tronco ('trunk') o en una rama ('branch'). El tronco y cada rama tienen su propia cabeza, aunque HEAD se utiliza a veces libremente para referirse al tronco.

Tronco ("trunk")	La única línea de desarrollo que no es una rama (llamada línea base, línea principal o máster).
Fusionar, integrar, mezclar ("merge")	<p>Una fusión o integración es una operación donde se aplican dos tipos de cambios en uno o más archivos. Algunos escenarios son los siguientes:</p> <ul style="list-style-type: none"> ■ Un usuario, trabajando en un conjunto de archivos, actualiza o sincroniza su copia de trabajo con los cambios realizados y confirmados por otros usuarios en el repositorio. ■ Un usuario intenta confirmar archivos que han sido actualizado por otros usuarios desde el último despliegue ('checkout'), y el software de control de versiones integra automáticamente los archivos (por lo general, después de preguntarle al usuario si procede con la integración automática, y en algunos casos sólo la hace si la fusión puede ser clara y razonablemente resuelta). ■ Un conjunto de archivos se bifurca, un problema que existía antes de la ramificación se trabaja en una nueva rama, y la solución se combina luego en la otra rama. ■ Se crea una rama, el código de los archivos es independiente editado, y la rama actualizada se incorpora más tarde en un único tronco unificado.

Capítulo 1

Introducción

Los sistemas de control de versiones (SCV) son una herramienta esencial para manejar proyectos de gestión de software y desde hace algunos años se han introducido en la enseñanza como herramienta de apoyo docente. De hecho, cada vez se encuentran trabajos que analizan su uso en ese contexto.

Los SCV son herramientas informáticas de uso cotidiano en el desarrollo profesional de software desde hace años, y proporcionan funcionalidades claves para el desarrollo y culminación de proyectos como es el control de cambios en el código, su reversibilidad y la posibilidad de trabajo colaborativo en el desarrollo del código.

Además, los SCV facilitan tener en paralelo varias versiones o ramas del proyecto. Las ramas se utilizan para desarrollar funcionalidades aisladas de los cambios en otras partes del proyecto que posteriormente pueden integrarse en la rama principal.

Los SCV se clasifican en dos grandes familias: centralizados y distribuidos. Los SCV centralizados son sistemas cliente-servidor donde existe un repositorio canónico en el servidor que contiene toda la información de los cambios mientras que los clientes sólo tienen copias de trabajo; mientras que en los CVS distribuidos cada usuario tiene su propio repositorio y puede intercambiar y mezclar revisiones con los distintos repositorios.

Desde 2010, la tendencia es utilizar cada vez más SCV distribuidos, en particular Git.

Los SCV se están incorporando a la enseñanza por diferentes motivos: ayudan a desarrollar el trabajo en equipo [1], facilitan la retroalimentación de los desarrolladores y/o estudiantes [4], establecen escenarios de desarrollo realistas [3] e, incluso, se utilizan como herramienta de supervisión [12]. Si bien su uso no está exento de retos. Aunque los estudiantes y desarrolladores pueden utilizarlo como un sistema de entrega más, sin aprovechar en su completa funcionalidad [2], o usarlo de forma ineficiente [5].

Desde la perspectiva académica existe el riesgo de un aumento de las tareas de gestión docente [6] o que la curva de aprendizaje por parte de los estudiantes sea tan elevada que afecte al normal desarrollo de un curso [7].

Esta contribución presenta la experiencia docente desarrollada en las diferentes unidades curriculares en el grado de Ingeniería Informática gestionado por el Departamento de Informática de la Universidad Politécnica Territorial del estado Aragua. En la unidades curriculares como Arquitectura del computador, Algorítmica y programación, así como en Paradigmas de programación se emplea para entregar las tareas y para el seguimiento en el desarrollo de la parte práctica y de la veracidad y autenticación de los proyectos finales.

Adicionalmente, en este momento de pandemia y alejamiento social, donde la educación de ha visto en la necesidad de implementar actividades académicas vía internet, en lugar de utilizar directamente Git desde la línea de comandos, se viene utilizado la plataforma y las herramientas de escritorio proporcionadas por GitHub [8] para afrontar los retos y riesgos mencionados.

GitHub es un servicio comercial de alojamiento en la Web de repositorios remotos Git y está considerado como la plataforma de alojamiento de repositorios remotos más popular. A principios de 2014 el número de usuarios de GitHub se estimaba n más de 3,4 millones y el número de repositorios en 16,7 millones [9].

En abril de 2015, el número de usuarios es más de 9,4 millones y el número de repositorios alcanza los 22,4 millones [8]. Nuestra contribución se centra en motivar esta experiencia, explicar cómo utilizar esta herramienta en los cursos, así como evaluar sus beneficios, riesgos y retos.

1.1. Introducción a git

Git es un sistema de control de versiones distribuido que se diferencia del resto en el modo en que modela sus datos. La mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos, mientras que Git modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos.

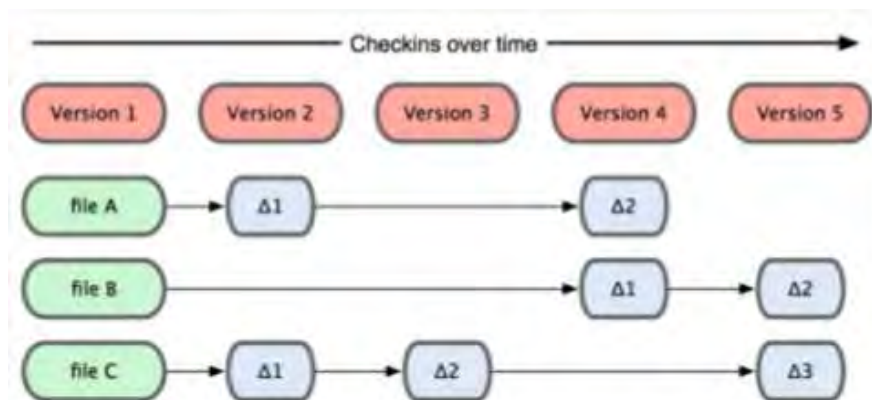


Figura 1.4 Modelo de datos de los sistemas distribuidos tradicionales

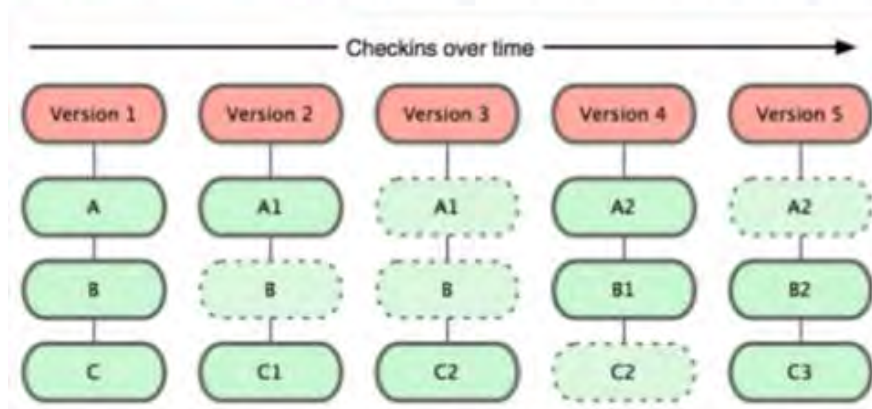


Figura 1.1: Modelo de datos de Git

1.2. Los tres estados

Git tiene tres estados principales en los que se encuentran sus archivos: confirmado (committed), modificado (modified) y preparado (staged). Confirmado significa que los datos están almacenados de manera segura en la base de datos local. Modificado significa que ha cambiado un archivo pero todavía no lo ha confirmado a la base de datos. Preparado significa que ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.



Figura 1.2: Directorio de trabajo, área de preparación, y directorio de Git

1.3. Definición, clasificación y funcionamiento

Se llama control de versiones a 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 que se encuentra dicho producto en un momento dado de su desarrollo. Aunque un sistema de control de versiones puede realizarse de forma manual, es aconsejable disponer de herramientas que faciliten la gestión, dando así lugar a los llamados sistemas de control de versiones o SVC (del inglés System Version Control).

Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (para algún cliente específico). Tipos de herramientas son: CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, Plastic SCM, Git, Mercurial, Perforce.

1.4. Clasificación

Los sistemas de control de versiones se clasifican atendiendo a la arquitectura utilizada para el almacenamiento del código, en:

1.4.1. Locales

Los cambios son guardados localmente y no se comparten. Esta arquitectura es la antecesora de las dos siguientes.

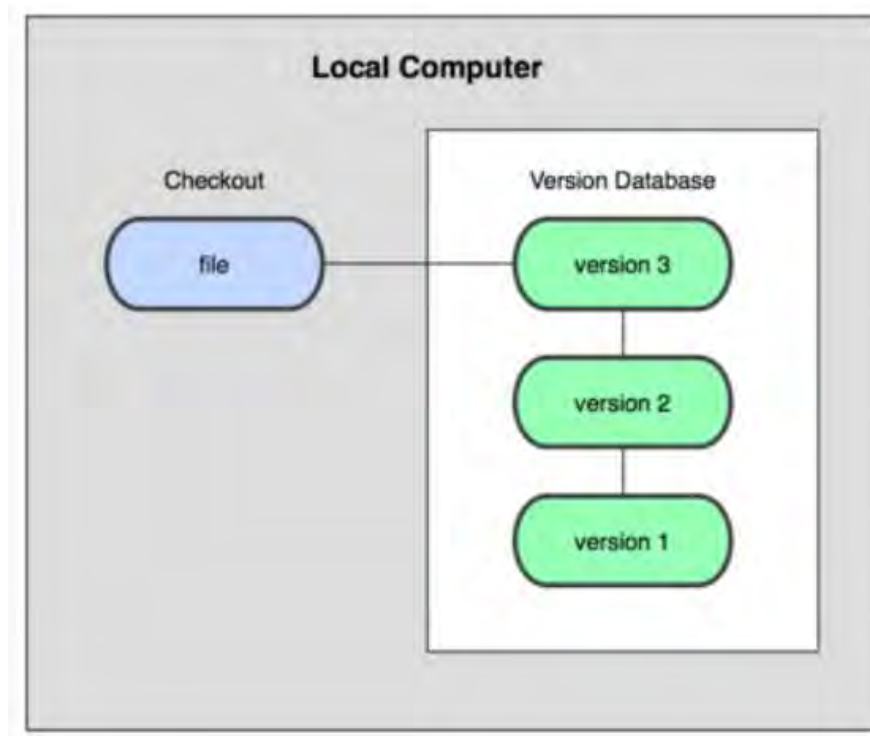


Figura 1.3: Sistema de control de versiones local

1.4.2. Centralizados

Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS y Subversion.

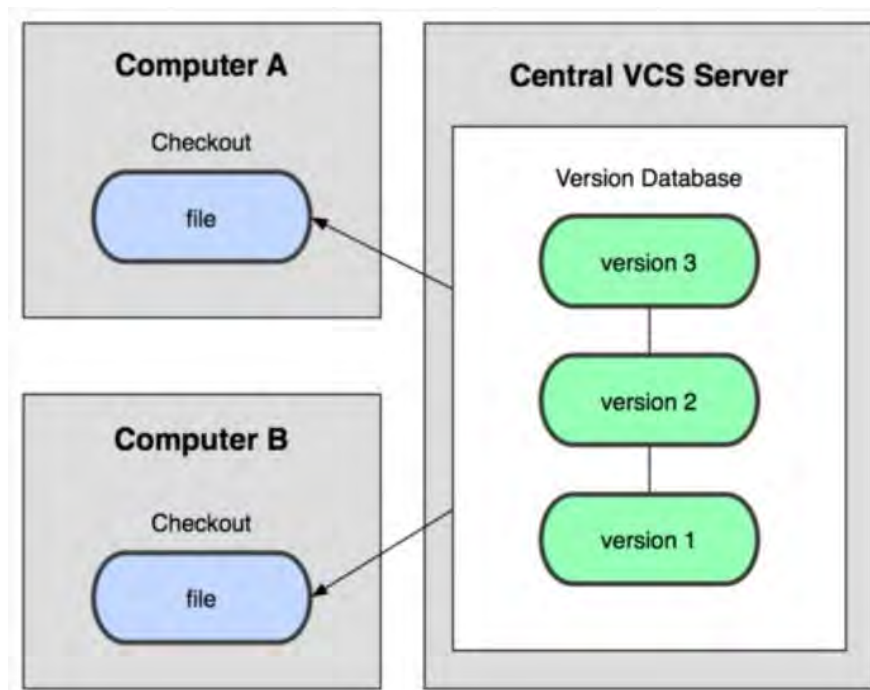


Figura 1.4: Sistema de control de versiones centralizado

1.4.3. Distribuidos

Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos si existe permiso para ello. Es frecuente el uso de un repositorio, que está normalmente disponible, como punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial.

Ventajas de sistemas distribuidos

- No es necesario estar conectado para guardar cambios.
- Posibilidad de continuar trabajando si el repositorio remoto no está accesible.
- El repositorio central está más libre de ramas de pruebas.
- Se necesitan menos recursos para el repositorio remoto.
- Más flexibles al gestionar cada repositorio personal como se quiera.

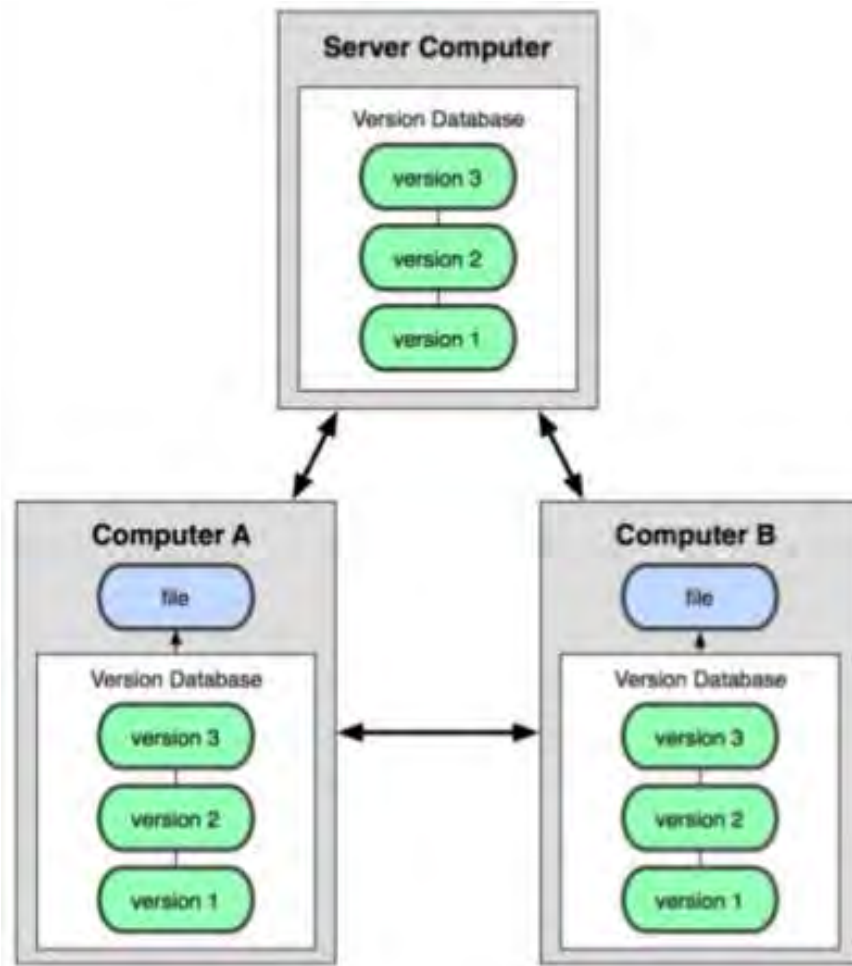


Figura 1.5: Sistema de control de versiones distribuido

Capítulo 2

GIT como plataforma de apoyo docente

En este apartado se presentan las herramientas y funcionalidades de GitHub a ser utilizadas en las diferentes unidades curriculares del PNFI en el Departamento de Informática de la UPT Aragua distinguiendo entre las derivadas de Git y las exclusivas de GitHub.

Las herramientas y funcionalidades se presentan agrupadas bajo tres criterios: a) El primero atiende a su uso por parte del estudiante en su aprendizaje, b) El segundo se fija en su papel como soporte del curso y c) el tercero describe aspectos relacionados con tecnologías, licencias y precios.

2.1. Aprendizaje del estudiante

Esta categoría incluye las herramientas y funcionalidades relacionadas con actividades de comunicación, productividad y participación de los estudiantes. Se tiene:

- Diario de actividades. El histórico de los cambios en un repositorio hospedado en GitHub es accesible vía Web, accediendo a los cambios producidos con indicación de quién, cuándo, qué ha cambiado y su origen. Toda la información contenida es navegable.
- Trabajo fuera de línea. Implícito por soportar Git pero restringido a los recursos gestionados en los repositorios por Git. No hay soporte fuera de línea a las herramientas de trabajo colaborativo de GitHub.
- Trabajo en grupo. El uso de Git y GitHub implementa diferentes flujos de trabajo en entornos educativos [10]. Esto se ha organizado según se describe en la Figura 2.1.

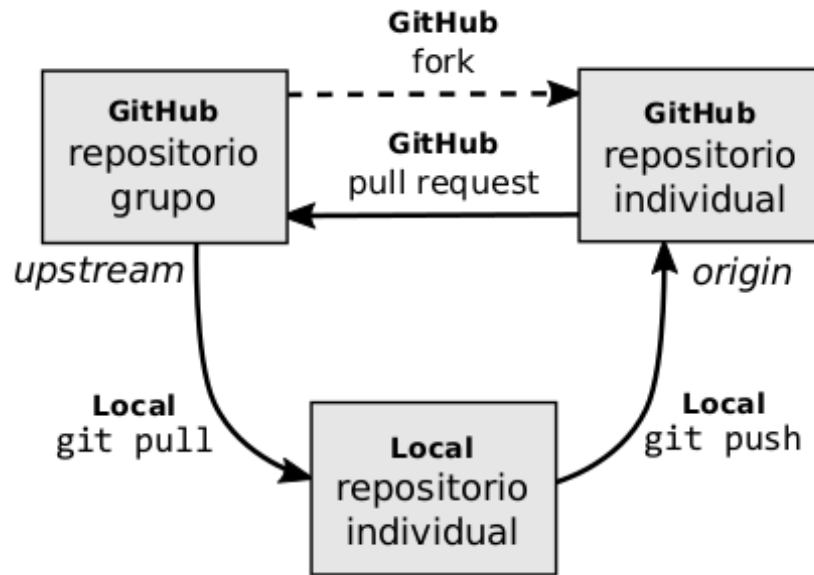


Figura 2.1: Flujo de trabajo en grupo utilizando Git-Hub/Git

La relevancia del trabajo en grupo merece el detallar su implementación.

1. Primero, preparar los repositorios remotos compartidos de los grupos y los repositorios remotos y locales de los estudiantes. GitHub actúa como la máquina remota donde se crean los repositorios remotos compartidos.
2. Segundo, cada estudiante clona en GitHub el repositorio remoto compartido de su grupo para crear su propio repositorio remoto.
3. Tercero, crear por parte de cada estudiante un clon local de su repositorio remoto utilizando Git. Al repositorio local se le añade como remoto el repositorio remoto del grupo. El repositorio remoto del estudiante se referencia localmente en Git como origin y el del grupo como upstream.
4. Una vez preparada la infraestructura, el flujo de trabajo esperado de cada estudiante ejecuta los siguientes pasos:
 - integra los cambios del repositorio del grupo en su repositorio local (`git pull -rebase upstream`), realiza los cambios pertinentes, envía dichos cambios a su repositorio individual remoto (`git push -force origin`) y,
 - finalmente, solicita vía un pull request que los cambios de su repositorio remoto se integren en el repositorio remoto del grupo. Dentro del grupo hay un estudiante con el rol de gestor del repositorio remoto. Este estudiante decide si se integra o no el cambio en el repositorio del grupo.

La única herramienta de GitHub considerada de aprendizaje no asociada con Git y de la cual se tiene constancia de su uso por parte de los estudiantes son los micro foros de discusión. Los estudiantes que han iniciado sesión en GitHub agregan vía Web comentarios a cualquier cambio y a los issues asociados a un repositorio público.

También permite al propietario del repositorio (como, un profesor) moderar dichos comentarios. Esta herramienta se utilizará principalmente en la interacción profesor-estudiante durante los seguimientos y entregas de los proyectos.

Otras herramientas como el potente motor de búsqueda facetado que restringe la búsqueda de usuarios a repositorios, rutas, lenguajes de programación, asuntos, y fechas de creación y mezcla, puede haber sido utilizado por los estudiantes pero no existe forma de cuantificar su uso.

2.2. Soporte del curso

Dentro de esta categoría, se incluyen las herramientas y funcionalidades relacionadas con actividades de soporte del curso, incluyendo aspectos administrativos, de desarrollo del contenido y de realización del curso.

Las derivadas de Git son:

- **Integridad.** Tal como se ha comentado en una sección anterior, Git incorpora una autenticación criptográfica del historial de cambios de un repositorio. Esta característica implica que un estudiante no puede modificar un cambio integrado en la historia del repositorio para añadir o borrar un fichero o alterar su contenido. En consecuencia, el histórico de cambios es una imagen fiel de la actividad del estudiante o del grupo.
- **Compartir material de prácticas.** El material de prácticas del curso así como todo el código que necesitan los estudiantes está almacenado en repositorios públicos. Se ha enfatizado a los estudiantes que deben clonar el material de prácticas. De esta manera, ya sea por nuevo material disponible o para corregir errores detectados, pueden aprovechar Git para incorporar los cambios.

Las herramientas y funcionalidades exclusivas que ofrece GitHub son:

- **Transparencia.** [11] señala que la disponibilidad en GitHub de pistas visibles como el volumen de actividad, la actividad a lo largo del tiempo, a relevancia del flujo de cambios, y la información detallada son herramientas útiles para resolver problemas de coordinación y comunicación.
Aplicada a la gestión del curso, la transparencia ha resuelto problemas de coordinación y comunicación entre los profesores y los estudiantes, y entre los estudiantes entre sí.
- **Publicación de contenido.** GitHub genera automáticamente un sumario en HTML del contenido del repositorio o de una carpeta si existe un fichero denominado README o README.md (.md es la extensión que corresponde al lenguaje de marcado ligero Markdown [6]).
- **Edición en línea de contenido.** GitHub lo edita vía un editor Web. Según el tipo de extensión, como .md, facilita su previsualización.
- **Seguimiento de los estudiantes.** Es posible acceder a diferentes visiones de la actividad de los estudiantes durante el curso en forma de diferentes tipos de informes (ver Figura 2.2: a) informes globales de actividad, b) de seguimiento del trabajo en equipo, c) informes comparativos de actividad y d) informes individuales de actividad.

- Plataforma de entrega con realimentación. La combinación del uso de pull requests como herramienta de entrega junto con los micro foros de discusión controlan la entrega y realimenta al estudiante con el resultado. Este procedimiento es uno de los recomendados por GitHub cuando se utiliza como plataforma de entrega en la docencia [12].

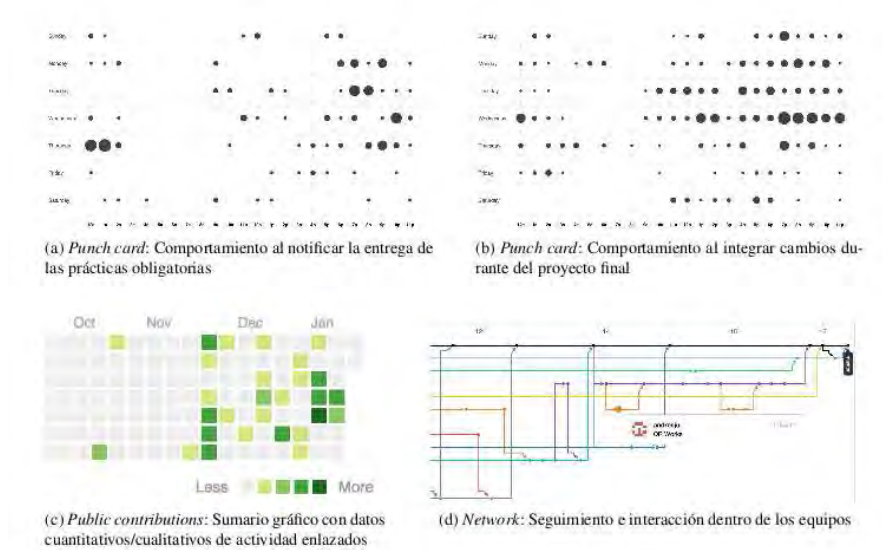


Figura 2.2: Informes proporcionados por la plataforma GitHub

Capítulo 3

GitHub

GitHub es el mayor proveedor de alojamiento de repositorios Git, y es el punto de encuentro para que millones de desarrolladores colaboren en el desarrollo de sus proyectos. Un gran porcentaje de los repositorios Git se almacenan en GitHub, y muchos proyectos de código abierto lo utilizan para hospedar su Git, realizar su seguimiento de fallos, hacer revisiones de código y otras cosas. Por tanto, aunque no sea parte directa del proyecto de código abierto de Git, es muy probable que durante su uso profesional de Git necesite interactuar con GitHub en algún momento.

Este capítulo trata del uso eficaz de GitHub. Trata de cómo crear y gestionar una cuenta, crear y gestionar repositorios Git, también los flujos de trabajo (workflows) habituales para participar en proyectos y para aceptar nuevos participantes en los suyos, la interfaz de programación de GitHub (API) y muchos otros pequeños trucos que harán, en general, la vida más fácil.

3.1. Creación y configuración de la cuenta

Lo primero que necesita es una cuenta de usuario gratuita. Simplemente visite <https://github.com>, elija un nombre de usuario que no esté en uso, proporcione un correo y una contraseña, y presione el botón verde grande "Sign up for GitHub".

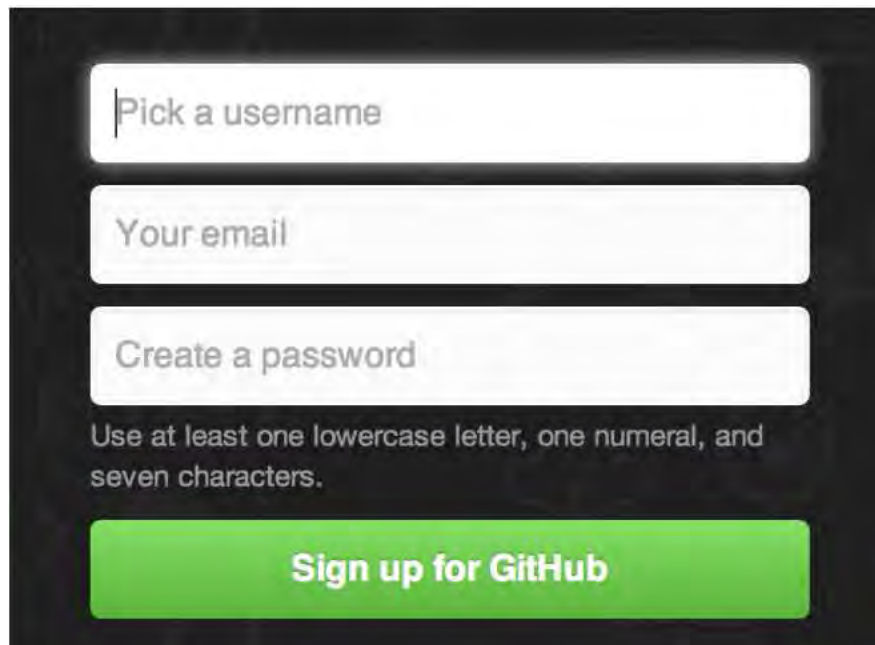
The image shows the GitHub sign-up form on a dark background. It consists of three white input fields stacked vertically. The first field is labeled 'Pick a username', the second 'Your email', and the third 'Create a password'. Below the third field, there is a line of text: 'Use at least one lowercase letter, one numeral, and seven characters.' At the bottom of the form is a large green button with the text 'Sign up for GitHub' in white.

Figura 3.1: Formulario para darse de alta en GitHub.

Lo siguiente que verá es la página de precios según planes, pero lo puede ignorar por el momento. GitHub le enviará un correo para verificar la dirección suministrada. Confirmar su dirección es importante.

GitHub proporciona toda su funcionalidad en cuentas gratuitas, puede tener tanto proyectos públicos como privados ilimitados. La única limitación es que cada uno de sus proyectos privados tenga sólo un máximo de tres colaboradores. Los planes de pago de GitHub permiten tener algunas herramientas extra, pero esto es algo fuera del alcance de este libro.

Si presiona el logo del gato con patas de pulpo en la parte superior izquierda de la pantalla llega a su escritorio principal. Ahora está listo para usar GitHub.

3.2. Acceso SSH

Acceda a los repositorios Git utilizando el protocolo `https://`, identificándose con el usuario y la contraseña correcto. Sin embargo, para simplificar el clonado de proyectos públicos, no necesita crear la cuenta. Es decir, la cuenta sólo la necesita cuando bifurca (fork) proyectos y enviará sus cambios más tarde.

Si prefiere usar SSH, debe configurar una clave pública. Si no la tiene, vea cómo generarla en generar su clave pública SSH. Abra el panel de control de la cuenta utilizando el enlace de la parte superior derecha de la ventana:

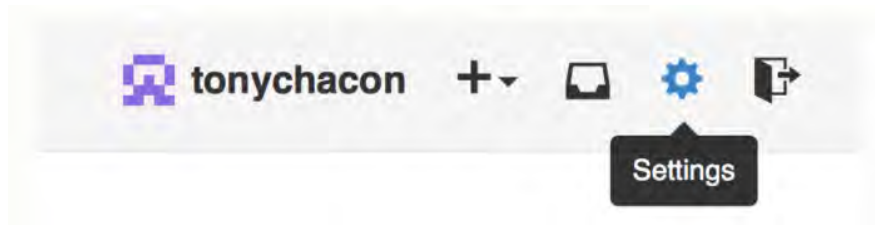


Figura 3.2: Enlace "Account settings" .

Seleccione en el lado izquierdo la opción "SSH keys" .

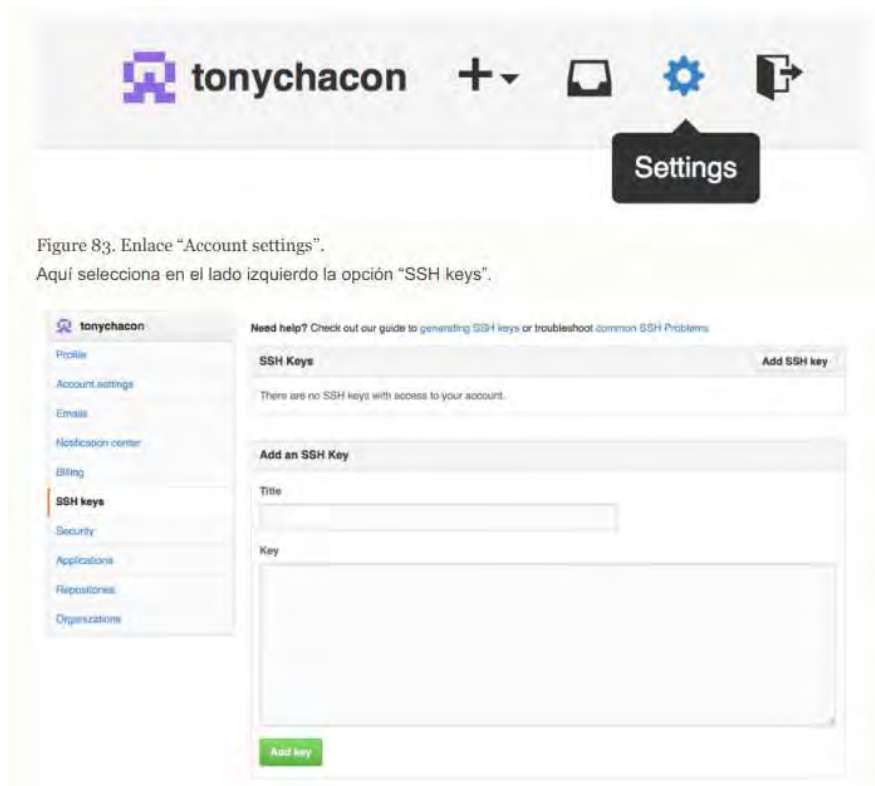


Figura 3.3: Enlace " SSH keys" .

Presione sobre "Add an SSH key", proporcione un nombre y pegue los contenidos del archivo `/.ssh/id_rsa.pub` (o donde haya definido su clave pública) en el área de texto y presione "Add key" .

Asegúrese de dar a su clave un nombre fácil de recordar. Puede añadir claves diferentes, con nombres como "Clave Portátil" o "Cuenta de trabajo", de modo que si tiene que revocar alguna clave más tarde, resultará fácil saber cuál es.

3.3. Tu icono

También, puede reemplazar el icono (avatar) generado con la imagen de su elección. En primer lugar seleccione la opción " Profile" (arriba de la opción " SSH keys") y presione " Upload new picture" .

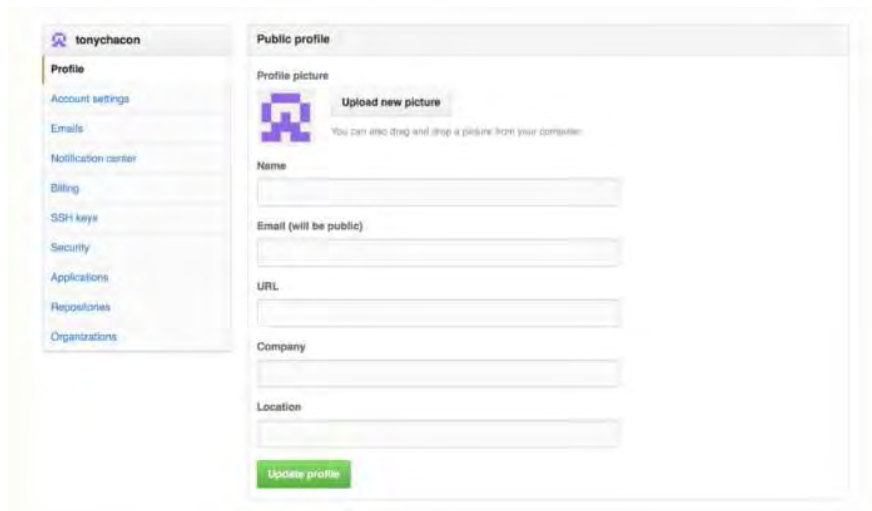


Figura 3.4: Enlace " Profile" .

Elija una copia del logo de Git que tenga en el disco duro, tendrá la opción de recortarlo al subirlo.

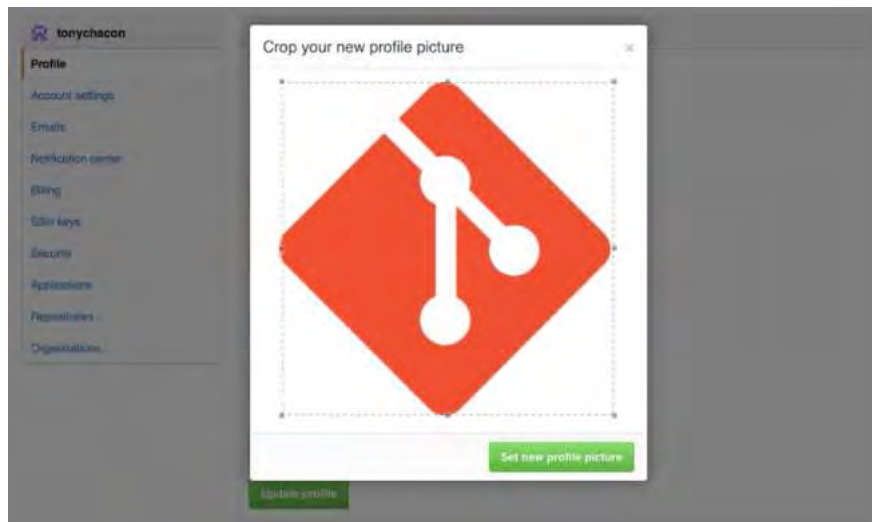


Figura 3.5: Recortar tu icono

Desde ahora, quien vea su perfil o sus contribuciones en los repositorios, verá su nuevo icono junto a su nombre.

Si por casualidad tiene su icono en el popular servicio Gravatar (conocido por su uso en las cuentas de Wordpress), éste será detectado y no tendrá que hacer este paso, si no lo desea.

3.4. Tus direcciones de correo

La forma en que GitHub identifica sus contribuciones a Git es mediante la dirección de correo electrónico. Si tiene varias direcciones diferentes en sus contribuciones (commits) y quiere que GitHub sepa que son de su cuenta, necesita añadirlas en el apartado Emails de la sección de administración.

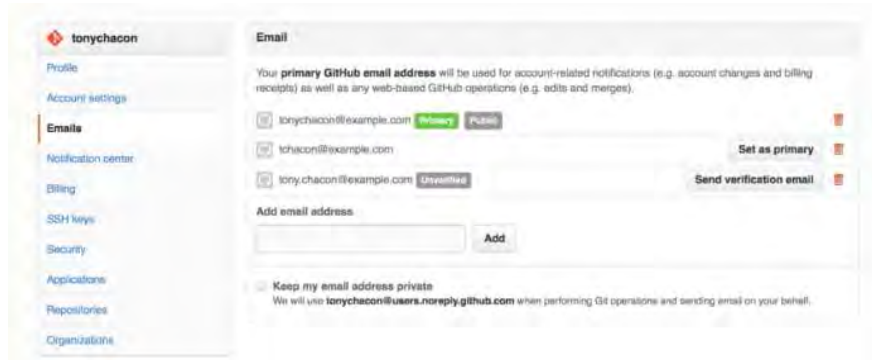


Figura 3.6: Añadiendo direcciones de correo

En Añadiendo direcciones de correo verá los diferentes estados posibles. La dirección inicial se verifica y utiliza como dirección principal, lo que significa que es donde va a recibir cualquier notificación. La siguiente dirección se puede verificar y ponerla como dirección principal, si decide cambiarla. La última dirección no está verificada, lo que significa que no puede usarla como principal. Pero si GitHub ve un commit con esa dirección, la identificará asociándola a su usuario.

3.5. Autenticación de dos pasos

Finalmente, y para mayor seguridad, configure la autenticación de dos pasos o "2FA". Este tipo de autenticación reduce el riesgo de que comprometan la cuenta. Al activarla, GitHub le solicita identificarse de dos formas, de manera que si una de ellas resulta comprometida, el atacante no conseguirá acceso a su cuenta.

Encuentra la configuración de "2FA" en la opción Security de los ajustes de la cuenta.

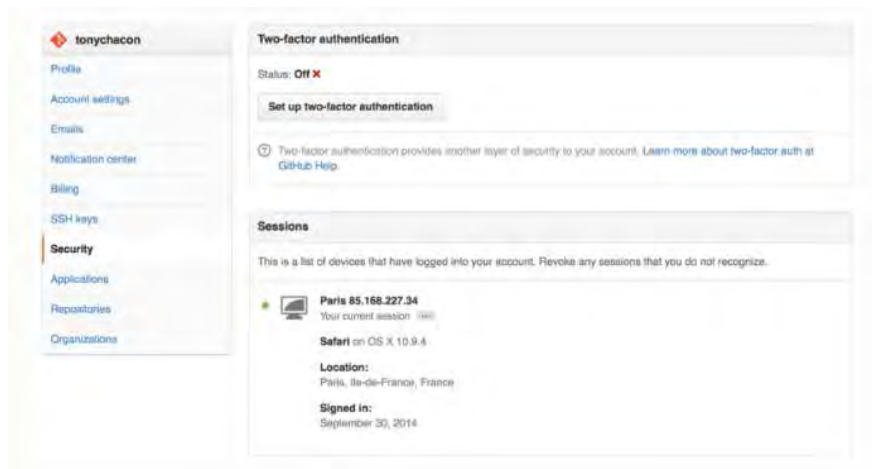


Figura 3.7: 2FA dentro de Security.

Si presiona el botón "Set up two-factor authentication", te saldrá una página de configuración donde podrás elegir un generador de códigos en una aplicación de móvil (es decir, códigos de un solo uso) o bien podrás elegir que te envíen un SMS cada vez que necesites entrar.

Cuando configures este método de autenticación, tu cuenta será un poco más segura ya que tendrás que proporcionar un código junto a tu contraseña cada vez que accedas a GitHub.

3.6. Mantenimiento de un proyecto

Ahora que ya sabes cómo ayudar a un proyecto, veamos el otro lado: cómo puede crear, administrar y mantener tu propio proyecto.

3.6.1. Creación de un repositorio

Vamos a crear un nuevo repositorio para compartir nuestro código en él. Comienza presionando el botón "New repository" en el lado derecho de tu página principal, o bien desde el botón + en la barra de botones cercano a tu nombre de usuario, tal como se ve en Desplegable "New repository"

..

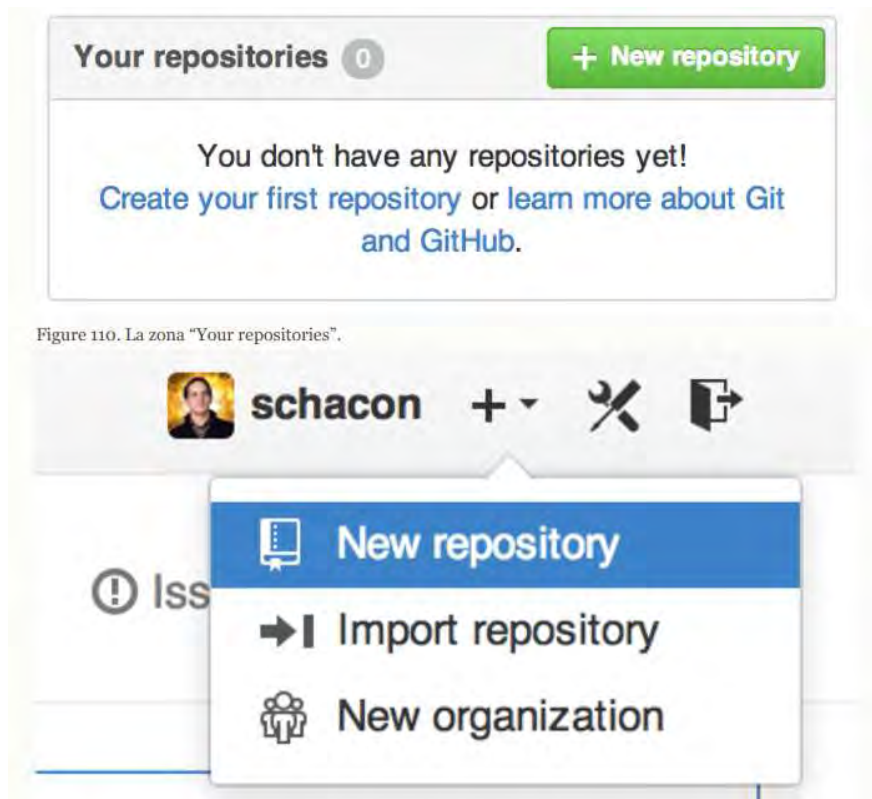


Figure 110. La zona "Your repositories".

Figura 3.8: Desplegable "New repository".

Esto te llevará al formulario para crear un nuevo repositorio:

Figura 3.9: Formulario para crear repositorio.

Todo lo que tiene que hacer aquí es darle un nombre al proyecto; el resto de campos es opcional. Por ahora, presione en el botón "Create Repository" y listo: se crea el repositorio en GitHub, con el nombre <usuario>/<proyecto>.

Dado que no tiene todavía contenido, GitHub te muestra instrucciones para crear el repositorio Git, o para conectarlo a un proyecto Git existente.

Ahora que el proyecto está alojado en GitHub, puede dar la URL a cualquiera con quien quieras compartirlo. Cada proyecto en GitHub es accesible mediante HTTP como `https://github.com/<usuario>/<proyecto>`, y también con SSH con la dirección `git@github.com:<usuario>/<proyecto>`. Git puede obtener y enviar cambios en ambas URL, ya que tienen control de acceso basado en las credenciales del usuario.

Suele ser preferible compartir la URL de tipo HTTP de los proyectos públicos, puesto que así el usuario no necesitará una cuenta GitHub para clonar el proyecto. Si das la dirección SSH, los usuarios necesitarán una cuenta GitHub y subir una clave SSH para acceder. Además, la URL HTTP es exactamente la misma que usamos para ver la página web del proyecto.

3.6.2. Añadir colaboradores

Si estás trabajando con otras personas y quiere darle acceso de escritura, necesitarás añadirlas como "colaboradores". Si Ben, Jeff y Louise se crean cuentas en GitHub, y quiere darles acceso de escritura a tu repositorio, los tiene que añadir al proyecto. Al hacerlo le darás permiso de "push", que significa que tendrán tanto acceso de lectura como de escritura en el proyecto y en el repositorio Git.

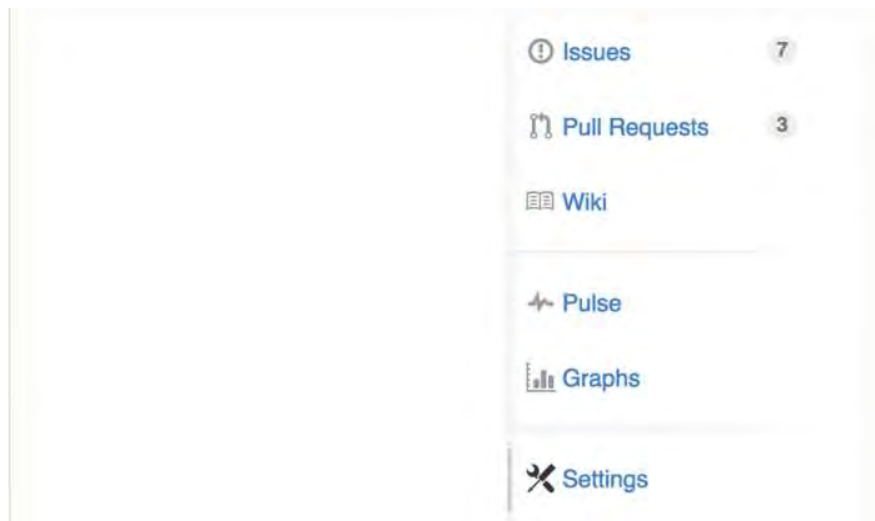


Figura 3.10: Enlace a ajustes del repositorio.

seleccione "Collaborators" del menú del lado izquierdo. Simplemente, teclea el usuario en la caja y presione en "Add collaborator." puede repetir esto las veces que necesites para dar acceso a otras personas. Recuerda que si el proyecto está en un repositorio privado gratuito, solo podrás dar accesos a tres colaboradores. Si necesita quitar un acceso, presione en la "X" del lado derecho del usuario.

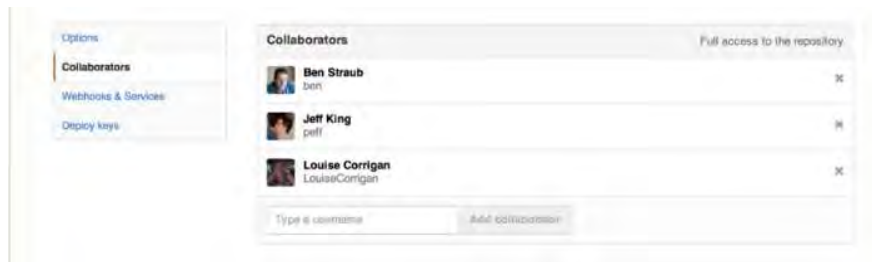


Figura 3.11: Colaboradores del repositorio.

3.6.3. Gestión de los Pull Requests

Ahora que tiene un proyecto con algo de código, y probablemente algunos colaboradores con acceso de escritura, veamos qué pasa cuando alguien te hace un Pull Request.

Los Pull Requests pueden venir de una rama en una bifurcación del repositorio, o pueden venir de una rama del mismo repositorio. La única diferencia es que, en el primer caso procede de gente que no tiene acceso de escritura a tu proyecto y quiere integrar en el tuyo cambios interesantes, mientras que en el segundo caso procede de gente con acceso de escritura al repositorio.

En los siguientes ejemplos, supondremos que eres "tonychacon" y has creado un nuevo proyecto para Arduino llamado "fade".

3.6.4. Notificaciones por correo electrónico

Cuando alguien realiza un cambio en el código y te crea un Pull Request, debes recibir una notificación por correo electrónico avisándote, con un aspecto similar a Notificación por correo de nuevo Pull Request..



Figura 3.12: Notificación por correo de nuevo Pull Request.

Hay algunas cosas a destacar en este correo. En primer lugar, te dará un pequeño diffstat (es decir,

una lista de archivos cambiados y en qué medida). Además, trae un enlace al Pull Request y algunas URL que puede usar desde la línea de comandos.

Si observas la línea que dice `git pull <url>patch-1`, es una forma simple de fusionar una rama remota sin tener que añadirla localmente. Lo vimos esto rápidamente en Recuperando ramas remotas. Si lo deseas, puede crear y cambiar a una rama y luego ejecutar el comando para fusionar los cambios del Pull Request.

Las otras URL interesantes son las de `.diff` y `.patch`, que como su nombre lo indica, proporcionan "diff unificados" y formatos de parche del Pull Request. Técnicamente, podrías fusionar con algo como:

```
1 | $ curl http://github.com/tonychacon/fade/pull/1.patch | git am
```

3.6.5. Colaboración en el Pull Request

Como hemos visto en El Flujo de Trabajo en GitHub, puede participar en una discusión con la persona que generó el Pull Request. puede comentar líneas concretas de código, comentar commits completos o comentar el Pull Request en sí mismo, utilizando donde quieras el formato Markdown.

Cada vez que alguien comenta, recibirás nuevas notificaciones por correo, lo que te permite vigilar todo lo que pasa. Cada correo tendrá un enlace a la actividad que ha tenido lugar y, además, puede responder al comentario simplemente contestando al correo.

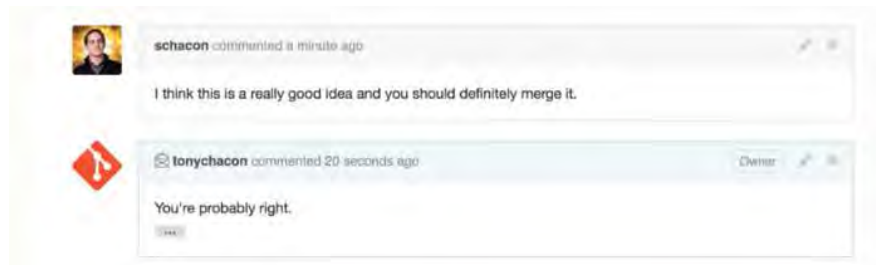


Figura 3.13: Las respuestas a correos se incluyen en el hilo de discusión.

Una vez que el código está como quiere y deseas fusionarlo, puede copiar el código y fusionarlo localmente, mediante la sintaxis ya conocida de `git pull <url><branch>`, o bien añadiendo el fork como nuevo remoto, bajándotelo y luego fusionándolo.

Si la fusión es trivial, también puede presionar el botón "Merge" en GitHub. Esto realizará una fusión "sin avance rápido", creando un commit de fusión incluso si era posible una fusión con avance rápido. Esto significa que cada vez que pulses el botón Merge, se creará un commit de fusión. Como verás en Botón Merge e instrucciones para fusionar manualmente un Pull Request., GitHub te da toda esta información si presiones en el enlace de ayuda.

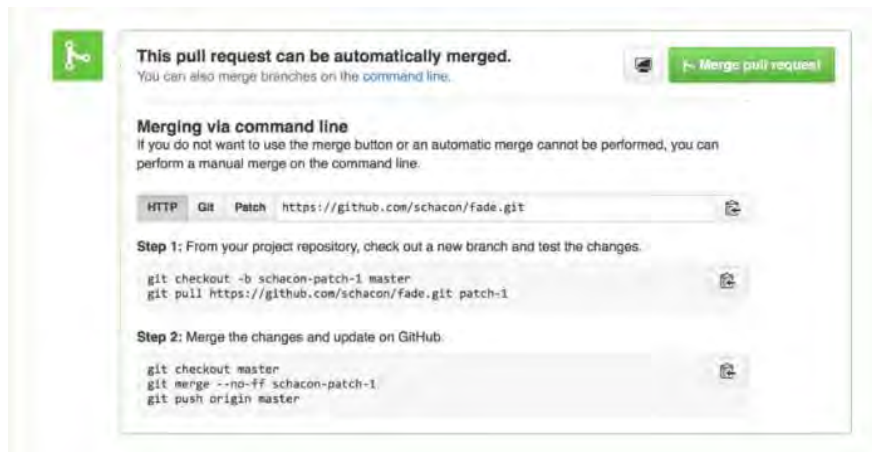


Figura 3.14: Botón Merge e instrucciones para fusionar manualmente un Pull Request.

Si decides que no quiere fusionar, también puede cerrar el Pull Request y la persona que lo creó será notificada.

3.6.6. Referencias de Pull Request

Si tiene muchos Pull Request y no quiere añadir un montón de remotos o hacer muchos cada vez, hay un pequeño truco que GitHub te permite. Es un poco avanzado y lo veremos en detalle después en Las especificaciones para hacer referencia a (refspec), pero puede ser bastante útil.

En GitHub tenemos que las ramas de Pull Request son una especie de pseudo-ramas del servidor. De forma predeterminada no las obtendrás cuando hagas un clonado, pero hay una forma algo oscura de acceder a ellos.

Para demostrarlo, usaremos un comando de bajo nivel (conocido como de "fontanería", sabremos más sobre esto en Fontanería y porcelana) llamado `ls-remote`. Este comando no se suele usar en el día a día de Git pero es útil para ver las referencias presentes en el servidor.

Si ejecutamos este comando sobre el repositorio "blink" que hemos estado usando antes, obtendremos una lista de ramas, etiquetas y otras referencias del repositorio.

```

1 $ git ls-remote https://github.com/schacon/blink
2 10d539600d86723087810ec636870a504f4fee4d HEAD
3 10d539600d86723087810ec636870a504f4fee4d refs/heads/master
4 6a83107c62950be9453aac297bb0193fd743cd6e refs/pull/1/head
5 afe83c2d1a70674c9505cc1d8b7d380d5e076ed3 refs/pull/1/merge
6 3c8d735ee16296c242be7a9742ebfbc2665adec1 refs/pull/2/head
7 15c9f4f80973a2758462ab2066b6ad9fe8dcf03d refs/pull/2/merge
8 a5a7751a33b7e86c5e9bb07b26001bb17d775d1a refs/pull/4/head
9 31a45fc257e8433c8d8804e3e848cf61c9d3166c refs/pull/4/merge

```

Por supuesto, si estás en tu repositorio y tecleas `git ls-remote origin` podrás ver algo similar pero para el remoto etiquetado como `origin`.

Si el repositorio está en GitHub y tiene Pull Requests abiertos, tendrás estas referencias con el prefijo `refs/pull`. Básicamente, son ramas, pero ya que no están bajo `refs/heads/`, no las obtendrás normalmente cuando clones o te bajas el repositorio del servidor, ya que el proceso de obtención las ignora.

Hay dos referencias por cada Pull Request, la que termina en /head apunta exactamente al último commit de la rama del Pull Request. Así si alguien abre un Pull Request en el repositorio y su rama se llama bug-fix apuntando al commit a5a775, en nuestro repositorio no tendremos una rama bug-fix (puesto que está en el fork) pero tendremos el pull/<pr#>/head apuntando a a5a775. Esto significa que podemos obtener fácilmente cada Pull Request sin tener que añadir un montón de remotos.

Ahora puede obtenerlo directamente.

```
1 $ git fetch origin refs/pull/958/head
2 From https://github.com/libgit2/libgit2
3 * branch refs/pull/958/head -> FETCH_HEAD
```

Esto dice a Git, " Conecta al remoto origin y descarga la referencia llamada refs/pull/958/head." Git obedece y descarga todo lo necesario para construir esa referencia, y deja un puntero al commit que quiere bajo `.git/FETCH_HEAD`. puede realizar operaciones como `git merge FETCH_HEAD` aunque el mensaje del commit será un poco confuso. Además, si estás revisando un montón de Pull Requests, se convertirá en algo tedioso.

Hay también una forma de obtener todos los Pull Requests, y mantenerlos actualizados cada vez que conectas al remoto. Para ello abre el archivo `.git/config` y busca la línea origin. Será similar a esto:

```
1 [remote "origin"]
2   url = https://github.com/libgit2/libgit2
3   fetch = +refs/heads/*:refs/remotes/origin/*
```

La línea que comienza con `fetch =` es un " refspec." Es una forma de mapear nombres del remoto con nombres de tu copia local. Este caso concreto dice a Git, que "las cosas en el remoto bajo refs/heads deben ir en mi repositorio bajo refs/remotes/origin." puede modificar esta sección añadiendo otra refspec:

```
1 [remote "origin"]
2   url = https://github.com/libgit2/libgit2.git
3   fetch = +refs/heads/*:refs/remotes/origin/*
4   fetch = +refs/pull/*:refs/remotes/origin/pr/*
```

Con esta última línea decimos a Git, " Todas las referencias del tipo refs/pull/123/head deben guardarse localmente como refs/remotes/origin/pr/123." Ahora, si guardas el archivo y ejecutas un `git fetch` tendremos:

```
1 $ git fetch
2 # ...
3 * [new ref] refs/pull/1/head -> origin/pr/1
4 * [new ref] refs/pull/2/head -> origin/pr/2
5 * [new ref] refs/pull/4/head -> origin/pr/4
6 # ...
```

Ya tiene todos los Pull Request en local de forma parecida a las ramas; son solo-lectura y se actualizan cada vez que haces un fetch. Pero hace muy fácil probar el código de un Pull Request en local:

```
1 $ git checkout pr/2
2 Checking out files: 100% (3769/3769), done.
3 Branch pr/2 set up to track remote branch pr/2 from origin.
4 Switched to a new branch 'pr/2'
```


La referencia `refs/pull/#/merge` de GitHub representa el "commit" que resultaría si presionamos el botón "merge". Esto te permite probar la fusión del Pull Request sin llegar a presionar dicho botón.

3.6.7. Pull Requests sobre Pull Requests

No solamente se puede abrir Pull Requests en la rama master, también se pueden abrir sobre cualquier rama de la red. De hecho, puede poner como objetivo otro Pull Request.

Si ves que un Pull Request va en la buena dirección y tiene una idea para hacer un cambio que depende de él, o bien no estás seguro de que sea una buena idea, o no tiene acceso de escritura en la rama objetivo, puede abrir un Pull Request directamente.

Cuando vas a abrir el Pull Request, hay una caja en la parte superior de la página que especifica qué rama quiere usar y desde qué rama quiere hacer la petición. Si presionamos el botón "Edit" en el lado derecho de la caja, puede cambiar no solo las ramas sino también la bifurcación.

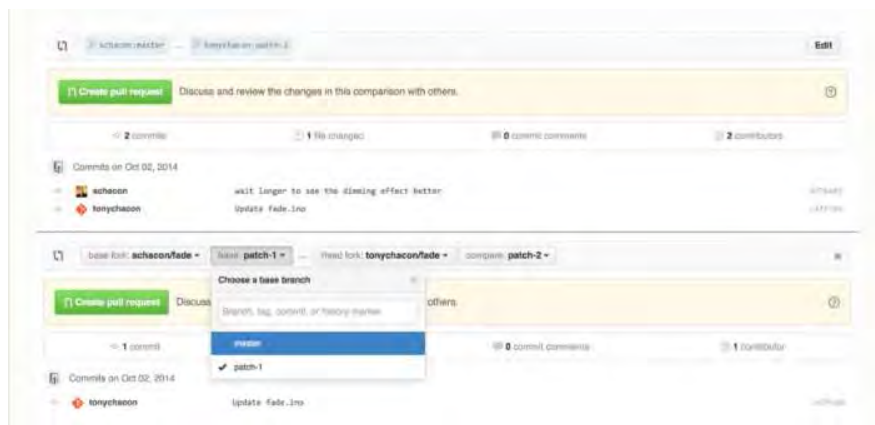


Figura 3.15: Cambio manual de la rama o del fork en un pull request.

Aquí puede fácilmente especificar la fusión de tu nueva rama en otro Pull Request o en otra bifurcación del proyecto.

3.6.8. Menciones y notificaciones

GitHub tiene un sistema de notificaciones que resulta útil cuando necesita pedir ayuda, o necesita la opinión de otros usuarios o equipos concretos.

En cualquier comentario, si comienzas una palabra anteponiendo el carácter @, intentará auto-completar nombres de usuario de personas que sean colaboradores o responsables en el proyecto.

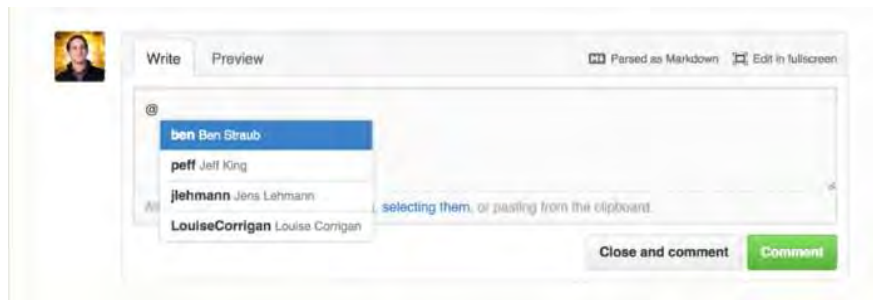


Figura 3.16: Empieza tecleando @ para mencionar a alguien.

También puede mencionar a un usuario que no esté en la lista desplegable, pero normalmente el autocompletado lo hará más rápido.

Una vez que envías un comentario con mención a un usuario, el usuario citado recibirá una notificación. Es decir, es una forma de implicar más gente en una conversación. Esto es muy común en los Pull Requests para invitar a terceros a que participen en la revisión de una incidencia o un Pull Request.

Si alguien es mencionado en un Pull Request o incidencia, quedará además "suscrito" y recibirá desde este momento las notificaciones que genere su actividad. Del mismo modo, el usuario que crea la incidencia o el Pull Request queda automáticamente "suscrito" para recibir las notificaciones, disponiendo todos de un botón "Unsubscribe" para dejar de recibirlas.

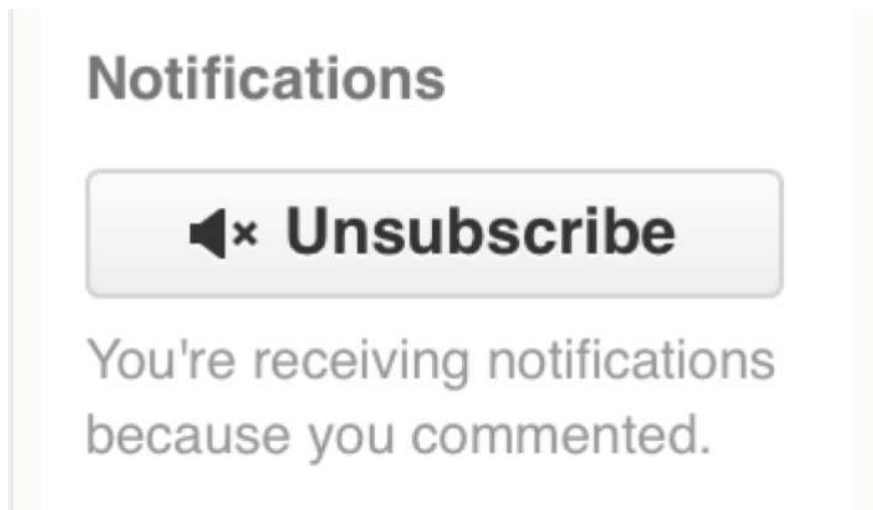


Figura 3.17: Quitar suscripción de un pull request o incidencia.

3.6.9. Página de notificaciones

Cuando decimos "notificaciones", nos referimos a una forma por la que GitHub intenta contactar contigo cuando tienen lugar eventos, y éstas pueden ser configuradas de diferentes formas. Si te vas al enlace "Notification center" de la página de ajustes, verás las diferentes opciones disponibles.

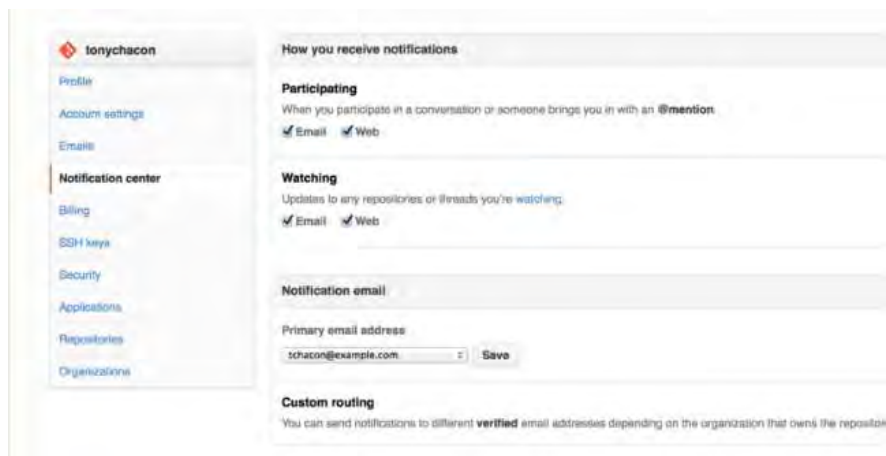


Figura 3.18: Opciones de Notification center.

Para cada tipo, puede elegir tener notificaciones de "Email" o de "Web", y puede elegir tener una de ellas, ambas o ninguna.

3.6.10. NOTIFICACIONES WEB

Las notificaciones web se muestran en la página de Github. Si las tiene activas verás un pequeño punto azul sobre el icono de Notificaciones en la parte superior de la pantalla, en Centro de notificaciones..

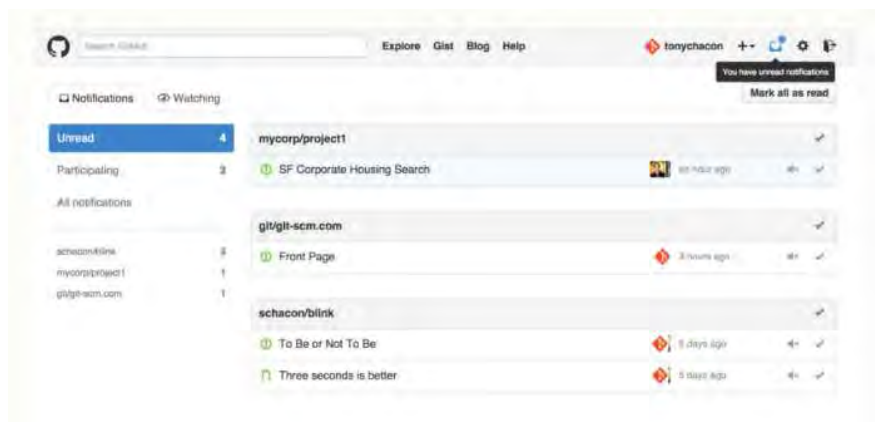


Figura 3.19: Centro de notificaciones.

Si presione s en él, verás una lista de todos los elementos sobre los que se te notifica, agrupados por proyecto. puede filtrar para un proyecto específico presione ndo en su nombre en el lado izquierdo. También puede reconocer (marcar como leída) una notificación presione ndo en el icono de check en una notificación, o reconocerlas todas presione ndo en el icono de check de todo el grupo. Hay también un botón "mute" para silenciarlas, que puede presione r para no recibir nuevas notificaciones de ese elemento en el futuro.

Todas estas características son útiles para manejar un gran número de notificaciones. Muchos usuarios avanzados de GitHub suelen desactivar las notificaciones por correo y manejarlas todas mediante esta pantalla.

3.6.11. NOTIFICACIONES POR CORREO

Las notificaciones por correo electrónico son la otra manera de gestionar notificaciones con GitHub. Si las tiene activas, recibirás los correos de cada notificación. Vimos ya algún ejemplo en Comentarios enviados en notificaciones de correo y Notificación por correo de nuevo Pull Request.. Los correos también serán agrupados correctamente en conversaciones, con lo que estará bien que uses un cliente de correo que maneje las conversaciones.

En las cabeceras de estos correos se incluyen también algunos metadatos, que serán útiles para crear filtros y reglas adecuados.

Por ejemplo, si miramos las cabeceras de los correos enviados a Tony en el correo visto en Notificación por correo de nuevo Pull Request., veremos que se envió la siguiente información:

```
1 | To: tonychacon/fade <fade@noreply.github.com>
2 | Message-ID: <tonychacon/fade/pull/1@github.com>
3 | Subject: [fade] Wait longer to see the dimming effect better (#1)
4 | X-GitHub-Recipient: tonychacon
5 | List-ID: tonychacon/fade <fade.tonychacon.github.com>
6 | List-Archive: https://github.com/tonychacon/fade
7 | List-Post: <mailto:reply+i-4XXX@reply.github.com>
8 | List-Unsubscribe: <mailto:unsub+i-XXX@reply.github.com>, ...
9 | X-GitHub-Recipient-Address: tchacon@example.com
```

Vemos en primer lugar que la información de la cabecera Message-Id nos da los datos que necesitamos para identificar usuario, proyecto y demás en formato <usuario>/<proyecto>/<tipo>/<id>. Si se tratase de una incidencia, la palabra " pull" habría sido reemplazada por " issues" .

Las cabeceras List-Post y List-Unsubscribe permiten a clientes de correo capaces de interpretarlas, ayudarnos a solicitar dejar de recibir nuevas notificaciones de ese tema. Esto es similar a presione r el botón " mute" que vimos en la versión web, o en " Unsubscribe" en la página de la incidencia o el Pull Request.

También merece la pena señalar que si tiene activadas las notificaciones tanto en la web como por correo, y marcas como leído el correo en la web también se marcará como leído, siempre que permitas las imágenes en el cliente de correo.

3.6.12. Archivos especiales

Hay dos archivos especiales que GitHub detecta y maneja si están presentes en el repositorio.

■ README

En primer lugar, tiene el archivo README, que puede estar en varios formatos. Con el nombre README, README.md, README.asciidoc y alguno más. Cuando GitHub detecta su presencia en el proyecto, lo muestra en la página principal, con el renderizado que corresponda a su formato.

En muchos casos este archivo se usa para mostrar información relevante a cualquiera que sea nuevo en el proyecto o repositorio. Esto incluye normalmente cosas como:

- Para qué es el proyecto
- Cómo se configura y se instala
- Ejemplo de uso

- Licencia del código del proyecto
- Cómo participar en su desarrollo

Puesto que GitHub hace un renderizado del archivo, puede incluir imágenes o enlaces en él para facilitar su comprensión.

■ CONTRIBUTING

El otro archivo que GitHub reconoce es CONTRIBUTING. Si tiene uno con ese nombre y cualquier extensión, GitHub mostrará algo como Apertura de un Pull Request cuando existe el archivo CONTRIBUTING. cuando se intente abrir un Pull Request.

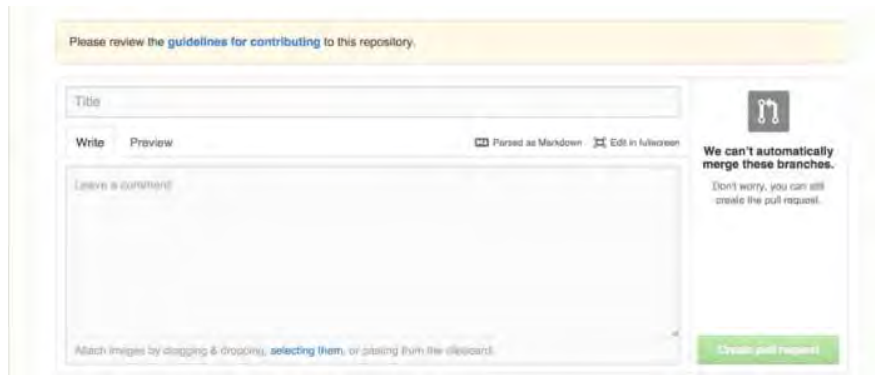


Figura 3.20: Apertura de un Pull Request cuando existe el archivo CONTRIBUTING.

La idea es que indique cosas a considerar a la hora de recibir un Pull Request. Los usuarios lo deben leer a modo de guía sobre cómo abrir la petición.

3.7. Administración del proyecto

Por lo general, no hay muchas cosas que administrar en un proyecto concreto, pero sí un par de cosas que pueden ser interesantes.

3.7.1. Cambiar la rama predeterminada

Si usa una rama predeterminada que no sea " master" , por ejemplo para que sea objetivo de los Pull Requests, puede cambiarla en las opciones de configuración del repositorio, en donde pone " Options" .



Figura 3.21: Cambio de la rama predeterminada del proyecto.

3.7.2. Rama predeterminada

Simplemente, cambie la rama predeterminada en la lista desplegable, y ésta será la elegida para la mayoría de las operaciones, así mismo será la que sea visible al principio (” checked-out”) cuando alguien clona el repositorio.

3.7.3. Transferencia de un proyecto

Si quiere transferir la propiedad de un proyecto a otro usuario u organización en GitHub, hay una opción para ello al final de ” Options” llamada ” Transfer ownership” .

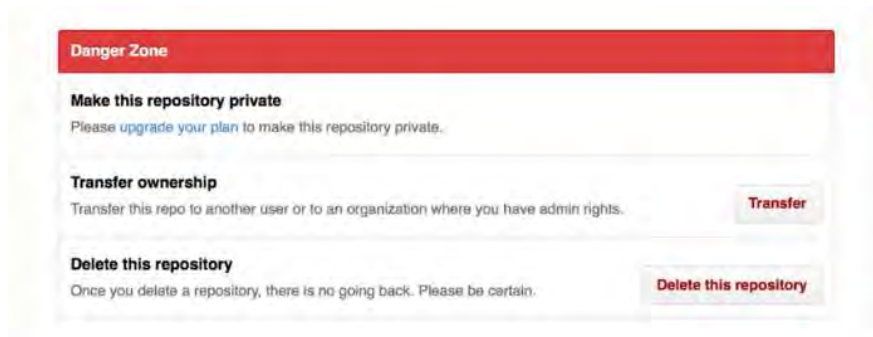


Figura 3.22: Transferir propiedad de un proyecto.

3.7.4. Transferir

Esto es útil si abandona el proyecto y quiere que alguien continúe, o bien se ha vuelto muy grande y prefieres que se gestione desde una organización.

Esta transferencia, supone un cambio de URL. Para evitar que nadie se pierda, genera una redirección web en la URL antigua. Esta redirección funciona también con las operaciones de clonado o de copia desde Git.

3.8. Gestión de una organización

Además de las cuentas de usuario, GitHub tiene Organizaciones. Al igual que las cuentas de usuario, las cuentas de organización tienen un espacio donde se guardarán los proyectos, pero en otras cosas son diferentes. Estas cuentas representan un grupo de gente que comparte la propiedad de los proyectos, y además se pueden gestionar estos miembros en subgrupos. Normalmente, estas cuentas se usan en equipos de desarrollo de código abierto (por ejemplo, un grupo para ” perl” o para ” rails”) o empresas (como sería ” google” o ” twitter”).

3.9. Conceptos básicos

Crear una organización es fácil: presione en el icono ” +” en el lado superior derecho y seleccione ” New organization” .

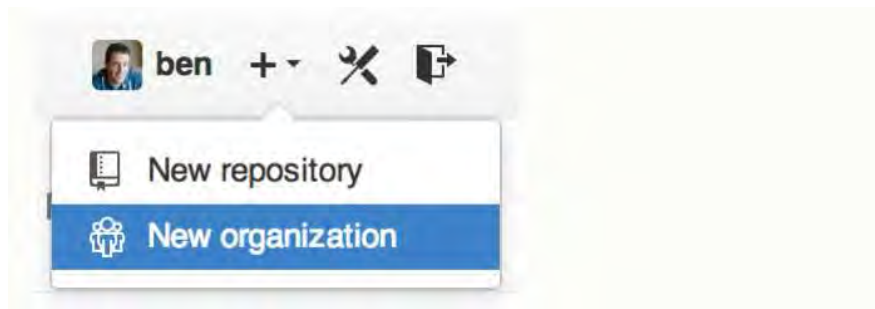


Figura 3.23: El menú “New organization”.

En primer lugar tiene que decidir el nombre de la organización y una dirección de correo que será el punto principal de contacto del grupo. A continuación, invite a otros usuarios a que se unan como co-propietarios de la cuenta.

Siga estos pasos y será propietario de un grupo nuevo. A diferencia de las cuentas personales, las organizaciones son gratuitas siempre que los repositorios sean de código abierto (y por tanto, públicos).

Como propietario de la organización, cuando bifurca un repositorio podrá hacerlo a su elección en el espacio de la organización. Cuando crea nuevos repositorios puede también elige el espacio donde se crearán: la organización o cuenta personal. Automáticamente, además, quedá como vigilante (watcher) de los repositorios creados en la organización.

Al igual que en “Tu icono”, puede subir uno para personalizar la organización, que aparecerá entre otros sitios en la página principal de la misma, que muestra los repositorios y puede ser visitada por publico en general.

Veamos algunas cosas que son diferentes con una cuenta de organización.

3.10. Equipos

Las organizaciones se asocian con individuos mediante los equipos, que son agrupaciones de cuentas de usuario y repositorios dentro de la organización, y los accesos que tienen esas personas sobre cada repositorio.

Si su empresa tiene tres repositorios: frontend, backend y deployscripts; y requiere que los desarrolladores de web tengan acceso a frontend y tal vez a backend, y que las personas de operaciones accedan a backend y deployscripts. Los equipos hacen fácil esta organización, sin tener que gestionar a los colaboradores en cada repositorio individual.

La página de la organización muestra un panel simple con todos los repositorios, usuarios y equipos asociados a ella.

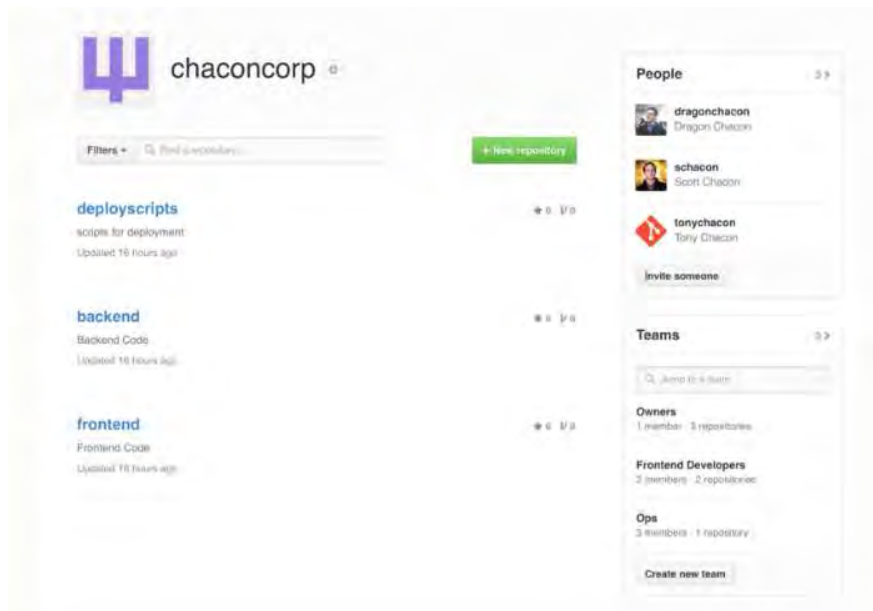


Figura 3.24: Página de la organización.

Para gestionar sus equipos, presione "Teams" en la barra del lado derecho en Página de la organización. Esto le lleva a una página en la que añade los miembros y repositorios del equipo o gestiona los ajustes y niveles de acceso del mismo. Cada equipo puede tener acceso de sólo lectura, de escritura o administrativo al repositorio. Cambie el nivel presionando el botón "Settings" en Página de equipos.

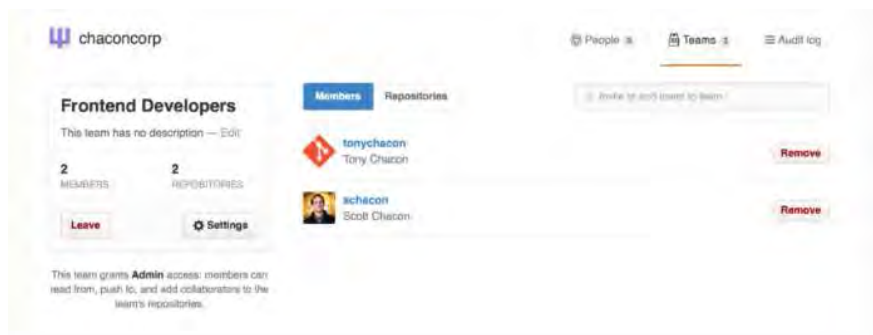


Figura 3.25: Página de equipos.

Cuando invita un usuario a un equipo, este usuario recibirá un correo con una invitación.

Además, hay menciones de equipo (como @acmecorp/frontend) que sirven para que todos los miembros del equipo sean suscritos con un único hilo. Esto es útil si involucra a un equipo en algo y no tiene claro a quién en concreto preguntar.

Un usuario puede pertenecer a cuantos equipos requiera, por lo que no use únicamente para temas de control de acceso a repositorios, sino que puede usarlos para formar equipos especializados y dispares como ux, css, refactoring, legal, etc.

3.11. Auditorías

Las organizaciones pueden también dar a los propietarios acceso a la información sobre la misma; incluso ir a la opción Audit Log y ver los eventos que han sucedido, quién hizo, por qué y dónde.

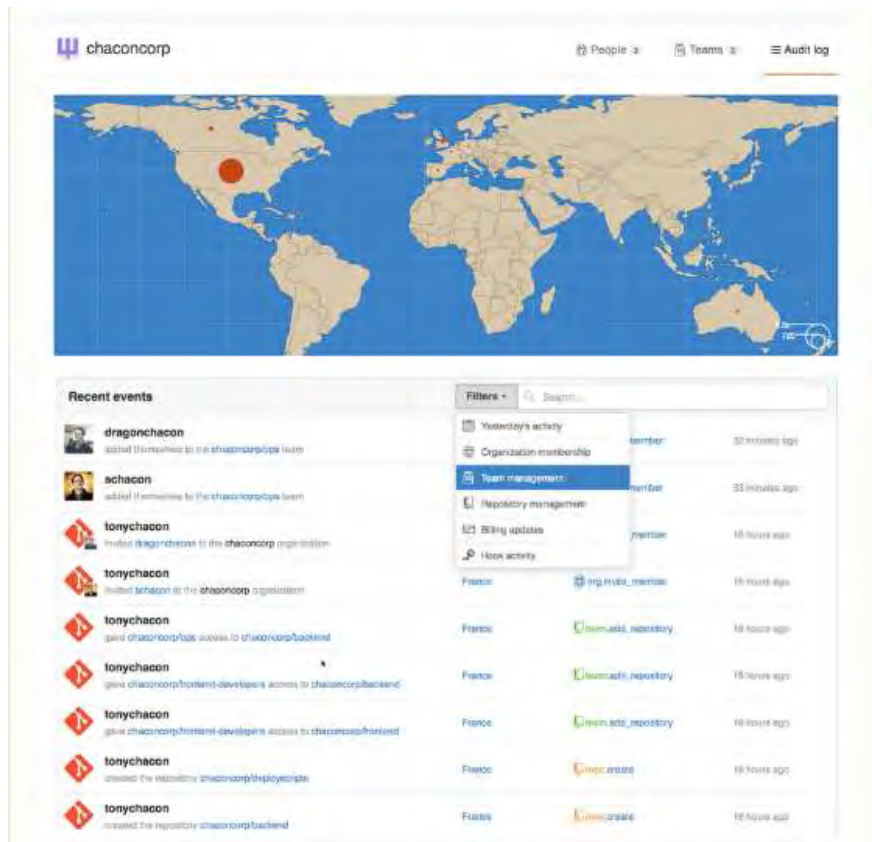


Figura 3.26: Log de auditoría.

También puede filtrar por tipo de evento, por lugares o por personas concretas.

Capítulo 4

Antecedentes

4.1. Académico

Los temas aquí descritos se ubican dentro de la enseñanza de las materias de la carrera de Ingeniería en Informática, impartida por el Departamento de Informática de la Universidad Politécnica Territorial del estado Aragua. La Coordinación de los Laboratorios de Informática del Departamento de Informática del UPT Aragua ofrece a sus docentes diversas herramientas de gestión de la enseñanza (LMS); actualmente ofrece la plataforma Moodle.

El plan de estudios de la carrera indica, a grosso modo, que el estudiante obtendrá las bases para administrar un sistema informático, así como para diseñar y desarrollar software y sistemas (administrativo, web, operativo); así como administrar redes informáticas.

Dichos objetivos son formalistas y mínimos, por lo cual dentro de cualquiera de los cursos del PNFI deben incluirse los siguientes objetivos adicionales:

- El estudiante conocerá el desarrollo histórico de los sistemas informáticos, lo que le llevará a comprender la razón de ser y el funcionamiento general de los diversos componentes de los sistemas informáticos actuales.
- Aplicará el conocimiento obtenido sobre el funcionamiento general de los sistemas informáticos, obteniendo así el mejor provecho de la computadora:
 - Al emplearla como usuario final.
 - Al administrarla.
 - Al programarla.
- El estudiante conocerá las principales herramientas que ofrecen los sistemas informáticos libres para su desarrollo, supervisión y administración.

La experiencia que se aborda a continuación, pues, se enmarca en la intersección de varios de estos objetivos: de los formales, el diseño y desarrollo de software administrativo, web y operativo; de los adicionales, el conocer las herramientas que ofrecen los sistemas operativos libres para el desarrollo, supervisión y administración de sistemas informáticos.

Cabe mencionar que, por razones que claramente exceden el ámbito del presente trabajo, el autor es un firme impulsor del software libre en lo personal y en lo profesional; esto lleva a que, más allá de únicamente influir las decisiones de software utilizado, y posiblemente contraviniendo tradiciones

del área ingenieril, considera las dimensiones éticas y sociales de los principios que transmite a los estudiantes — incluso en la elección de herramientas y métodos de colaboración.

4.2. Tecnológico

El desarrollo de software, a cualquier escala, siempre ha requerido de la coordinación de esfuerzos entre desarrolladores, y resulta natural que herramientas con éste fin han sido desarrolladas desde muy temprano en la historia de la computación. El modelo de desarrollo que se seguía hasta la década de 1960 era muy diferente, pero conforme se popularizó el uso interactivo de las computadoras mediante los sistemas de acceso compartido y las terminales interactivas, esta necesidad se hizo patente.

Los grandes proyectos de software libre iniciados a principios de los noventa (particularmente, los sistemas operativos Linux, FreeBSD, NetBSD y OpenBSD, y una gran cantidad de aplicaciones) iniciaron su desarrollo empleando CVS como punto modal.

La década de los noventa fue testigo de un crecimiento vertiginoso tanto en los proyectos de software libre como en la capacidad de los equipos de cómputo y la universalidad del acceso a red (puede argumentarse que el primero se debió a la conjunción de los otros dos). El modelo de uso de CVS resultó insuficiente; carecía de capacidad para representar acciones como la eliminación o el cambio de nombre de un archivo e imponía un alto costo a trabajar con archivos que no fueran de texto plano.

Entre el 2000 y 2004 se desarrolló Subversion; éste sistema seguía el mismo modelo e interacción de CVS, corrigiendo estas y otras debilidades [13].

Subversion creció en pocos años y se convirtió en una de las principales herramientas de desarrollo en el ámbito del software libre.

Sin embargo, los SCVs mencionados operan de forma centralizada. Con el crecimiento de algunos proyectos a escalas de desarrollo inimaginables para el momento (8,000 desarrolladores en 20 años, 15 millones de líneas de código y más de 37,000 archivos [14], fueron necesarios algunos cambios. En 2002 Linus Torvalds tomó la controvertida decisión de dejar CVS por un SCV distribuido y gratuito, pero no libre — Un sistema en que no existiera una copia maestra o servidor central, sino que cada copia del proyecto guarde toda la historia y estado, con sincronización entre ramas relacionadas. Dado que no había un SCV libre considerado modelo distribuido, Torvalds adoptó BitKeeper.

Si bien la adopción de BitKeeper fue difícil y un punto recurrente de fricción con los puristas de la libertad del software es indudable que impulsó la velocidad en el desarrollo de Linux, que había decrecido por varios años [15]. BitKeeper ofrecía una licencia gratuita (no libre) para los desarrolladores de software libre, siempre y cuando no se utilice para competir con BitKeeper mismo.

Para 2005, se suscitó una discusión acerca de si la funcionalidad que Andrew Tridgell¹ estaba agregando a Linux consistía una violación de la licencia [17], lo cual llevó a que Torvalds iniciara el desarrollo de un SCV distribuido, al que llamó Git².

En el mismo periodo de tiempo se desarrollaron varios SCVs distribuidos libres, como Monotone,

¹Andrew "Tridge" Tridgell es un programador australiano, residente en Canberra. Nació en Sídney, Andrew es el autor inicial del servidor de ficheros Samba, y co-inventor del algoritmo rsync. Es conocido por sus análisis de protocolos propietarios y algoritmos, para hacer implementaciones libres compatibles con estos.

²En slang británico, Git denomina a una mala persona; bromeando, Torvalds dice, "soy un bastardo egoísta, por lo que denomino a todos mis proyectos en referencia a mí mismo. Primero Linux, y ahora Git". [16]

Darcs, Mercurial o Bazaar. Además de otras alternativas propietarias, como Team Foundation Server de Microsoft. Sin embargo, Git ha resultado preferido sobre los demás, y puede verse como "lingua franca", no únicamente entre los desarrolladores de software libre, sino en el mundo de la programación en general; ha concentrado una clara mayoría de proyectos entre los SCVs distribuidos, y atraer a una gran cantidad de proyectos que tenían incluso más de veinte años de historia en los SCVs centralizados [18].

De forma paralela, desde fines de los noventa, aparecieron las forjas, sitios Web dedicados al hospedaje de proyectos de software libre, los cuales ofrecen recursos administrados como un espacio Web, seguidor de fallos, listas de correo — y un SCV.

Según la información disponible en el sitio SourceForge hospeda al día de hoy más de 500,000 proyectos y tiene "varios millones" de usuarios registrados. Sin embargo, el flujo de interacción en que está basado es poco amigable, lo cual lo hace apto únicamente para usuarios que ya son profesionales del desarrollo de software.³

En 2008, y ya viendo el rápido crecimiento en la adopción de Git para proyectos de todo tamaño, nació GitHub: Una forja con un flujo de trabajo simplificado, y fuertemente centrado en el modelo de desarrollo de Git. Apenas tres años más tarde se transformó en la forja con mayor actividad en el mundo del software libre [19]. A la fecha del presente texto, según la información disponible en el sitio Web, hospeda más de 68 millones de proyectos.

Si bien GitHub se ha vuelto nodal para el desarrollo de cantidad de proyectos libres, causa una cierta disonancia cognitiva que GitHub mismo no es libre: El software con el cual opera el sitio Web, y que integra las diferentes herramientas que lo componen, es un desarrollo propietario.

Hay otros servicios, como GitLab, que ofrece un modelo de colaboración y una semántica perfectamente mapeable con GitHub. Resultaría más coherente con los principios personales de uso y promoción del software libre que se mencionaron al cierre de la sección anterior, el desarrollar la experiencia que se presenta en GitLab o alguna plataforma similar; la decisión de hacerlo en GitHub no fue tomada a la ligera, y se centra en la importancia que dicho sitio tiene para el desarrollo de software en general. Hoy, prácticamente todos los proyectos libres de desarrollo están alojados en GitHub (bien sea que provea su espacio principal de desarrollo o sea elegido como un almacenamiento de respaldo o conveniencia).

4.3. La unión

El uso de un SCV como repositorio para la entrega de trabajos en clase no es una idea novedosa ni es la primera vez que se documenta. Hay dos trabajos que apuntan en el mismo sentido que éste; la principal diferencia tanto en el método como en los resultados, radica en la década que ha pasado desde su publicación Reid [20] y Milentijevic [21]. Ambos documentan experiencias centradas en el uso de un SCV centralizado.

El trabajo de Reid implementa un repositorio por estudiante, empleando CVS. Esto impone un límite que resulta determinante: la parte fundamental de los SCV es la colaboración. Si cada estudiante tiene un repositorio personal, realizar trabajos en equipo presenta la disyuntiva de si multiplicar los commits (requerir que cada uno de los participantes deposite el mismo trabajo en su depósito perso-

³Aunque la cantidad de proyectos hospedados en SourceForge es muy grande, incluso en sus años de mayor actividad resultaba tan poco atractivo al uso casual que se ganó el mote de SourceForget, "fuentes olvidadas", por la gran cantidad de proyectos inactivos hospedados.

nal) o indicar de alguna manera al docente que un sólo proyecto ampara a varios estudiantes. Reid incluso menciona como uno de los puntos de tensión para el uso educativo de CVS a la "necesidad de prevenir que un estudiante vea el repositorio del otro".

Milentijevic relata una experiencia sobre una base tecnológica que puede sustentarse ya sea en Subversion o en CVS, con un módulo que presente al conjunto de repositorios sobre una interfaz Web. La separación en múltiples repositorios se efectúa a nivel proyecto: según avance la materia, cada proyecto se presentará a los estudiantes, quienes tendrán que obtener un nuevo repositorio; el ciclo de vida se separa en actividades de inicialización, realización de la tarea, y evaluación de la tarea; la primera de estas etapas es la que más actividades demanda, lo cual habla del trabajo burocrático adicional requerido para éste. Esto permite la flexibilidad de tareas con equipos de diferentes conformaciones, incluso dando la tarea de supervisor a uno de los estudiantes.

GitHub tiene con un sitio comunitario⁴ orientado al contacto entre docentes y estudiantes para discutir la tecnología en la educación. La opción está estructurada en la creación de organizaciones y su operación dentro de salones GitHub; el flujo de trabajo que presenta es específico a esa configuración, y no refleja la colaboración sobre un flujo de trabajo en el espacio profesional.

La mera existencia de la comunidad GitHub para la educación es suficiente para estar seguros de que hay muchas otras experiencias en este sentido.

4.4. Git colaborativo

Con la implantación del trabajo remoto⁵ total o parcial, las organizaciones se enfrentan al reto de mantener a su equipo conectado y colaborando en todo momento, sin importar distancias geográficas. Git es una de las soluciones tecnológicas que la educación y/o empresa necesitan para afrontar con éxito un paradigma complejo, en el que la productividad y la rentabilidad se presentan como elementos clave.

Existen muchos beneficios que el estar aislado tienen sobre el trabajo académico y/o profesional y/o personal, por lo que Git es un servicio diseñado para cubrir las diversas necesidades en el ámbito de las comunicaciones de las universidades, empresas y administraciones públicas y privadas. Necesidades que en este momento, a consecuencia del cambio de paradigma provocado por la crisis de la covid-19, cobran mayor importancia.

La experiencia de más de 17 años del Departamento de Informática de la UPT Aragua trabajando con herramientas open source ofrece servicios flexibles, modulares y adaptados a las necesidades de cada tipo de estudiante y/o usuario. Un conjunto de servicios basado en Software Libre que minimiza los costos de inversión y operación.

4.5. GitHub no es sólo para desarrolladores de software

A pesar de que GitHub, GitLab o Git es una plataforma centrada en el desarrollo de software colaborativo, no es necesario ser un desarrollador para utilizarla. Cualquier persona, estudiante o profesional que utilice un computador, tablet, laptop y/o teléfono inteligente a diario es tenido en cuenta como un "trabajador del conocimiento". Por tanto, puede beneficiarse de esta red social para seguir aprendiendo cosas sobre este entorno.

⁴<https://education.github.community/>

⁵Teletrabajo

Hay muchas razones para utilizar GitHub aunque no seas desarrollador. Considerar GitHub como una simple herramienta de desarrollo es un error. Es cierto que fundamentalmente se basa en codificación, lenguajes y compiladores de sistemas informáticos. También tiene características muy similares a otras redes sociales como Facebook o Twitter.

Capítulo 5

Operaciones básicas con repositorios locales

5.1. Configuración local

Cada commit va firmado con nuestro email y nombre.

```
1 | $ git config --global user.email "jodocha@upta.edu.ve"
2 | $ git config --global user.name "Jorge Dominguez Chavez"
3 | Configuración almacenada en ~/.gitconfig
```

- git init Crea un repositorio local en la carpeta de proyecto.

Cree carpeta para pruebas denominada holagit y acceda a ella.

```
1 | $ git init
2 | Initialized empty Git repository in Reinitialized existing Git
   repository in /home/jorge/.git/
```

- Working dir, Staging Area y Repo stagingarea.svg

Otra forma de verlo.



Figura 5.1: addcommit

- Cree un archivo de ejemplo llamado README.md

```
1 | # Hola Git
```

Este es mi primer archivo para control de versiones.

- git status Muestre las diferencias entre el Working dir, la Staging Area y el repo.

```
1 $ git status
2 On branch master
3
4 No commits yet
5
6 Untracked files:
7   (use "git add <file>..." to include in what will be committed)
8
9       README.md
10
11 nothing added to commit but untracked files present (use "git add"
    to track)
```

- git add Añada desde Working dir a Staging area (index).

La sintaxis es git add files

Algunos ejemplos:

```
1 $ git add README.md
2 $ git add .
3 $ git add --all
```

Vea los cambios con git status para corroborar cómo ha pasado a la Staging area y está listo para pasarlo al repositorio.

```
1 $ git status
2 On branch master
3
4 No commits yet
5
6 Changes to be committed:
7   (use "git rm --cached <file>..." to unstage)
8
9       new file:   README.md
```

- git commit Añada desde Staging Area al repo.

La sintaxis es git commit -m commitText:

```
1 $ git commit -m "README.md creado"
2 [master (root-commit) 1f86692] README.md creado
3 1 file changed, 3 insertions(+)
4  create mode 100644 README.md
```

Vea el estado con git status

```
1 $ git status
2 On branch master
3 nothing to commit, working tree clean
```

- git blame Quién ha cambiado un archivo y cuándo


```

1 $ git blame README.md
2 ^1f86692 (Jorge Dominguez Chavez 2018-11-27 17:39:47 +0000 1) #
   Hola Git
3 ^1f86692 (Jorge Dominguez Chavez 2018-11-27 17:39:47 +0000 2)
4 ^1f86692 (Jorge Dominguez Chavez 2018-11-27 17:39:47 +0000 3) Este
   es mi primer archivo para control de versiones.
5 \item git log

```

Muestra los comentarios comenzando por el más reciente.

```

1 $ git log
2 $ git log --oneline

```

Versión compacta.

Más información sobre la presentación de logs en: <https://stackoverflow.com/questions/1441010/the-shortest-possible-output-from-git-log-containing-author-and-date>

```

1 $ git log
2 commit 1f8669231adc380d96261f91495018ba2791b0b3 (HEAD -> master)
3 Author: Jorge Dominguez Chavez <archivo@upta.edu.ve>
4 Date: Tue Nov 27 17:39:47 2018 +0000
5
6     README.md creado
7
8 $ git log --oneline
9 1f86692 (HEAD -> master) README.md creado
10 Versión normal
11 Versión compacta

```

- Ejercicio Añada una línea nueva al archivo README.md con el texto que aparece a continuación.

```

1 # Aprende latín
2
3 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
   eiusmod tempor incididunt ut labore et dolore magna aliqua.

```

Añada los cambios a la Staging area.

Añada los cambios al repositorio con el mensaje Nuevo epígrafe.

Muestre el log.

Obtenga el estado del repositorio.

Debería obtener algo similar a lo siguiente

```

1 $ git log
2 commit 706c9ab817425ea714b758cbcf7934dcf0a6ecb4 (HEAD -> master)
3 Author: Jorge Dominguez Chavez <archivo@upta.edu.ve>
4 Date: Tue Nov 27 17:55:13 2018 +0000
5
6     Nuevo epígrafe
7
8 commit 1f8669231adc380d96261f91495018ba2791b0b3
9 Author: Jorge Dominguez Chavez <archivo@upta.edu.ve>

```

```

10 Date:    Tue Nov 27 17:39:47 2018 +0000
11
12     README.md creado
13
14 $ git status
15 On branch master
16 nothing to commit, working tree clean
17 Mostrando los cambios
18 El nuevo commit aparece el primero
19 Mostrando el estado

```

5.2. Operaciones básicas con repositorios remotos

- Configuración de claves SSH en gitlabdoc.upta.com.ve

Las claves SSH evitan introducir login y pass en cada operación de subida al repositorio remoto (push)

En GitLab, cada usuario tiene que modificar sus Settings en Profile Settings \rightarrow SSK Keys \rightarrow Add SSH Key:

- Cree SSH Key Desde una terminal ejecute:

```
1 | $ ssh-keygen
```

Esto genera dos archivos: Clave privada: id_rsa

Clave pública: id_rsa.pub

- Actualizar SSH Key en GitLab Copie el contenido de la clave pública id_rsa.pub en GitLab y dele un nombre

```
1 | $ cat ~/.ssh/id_rsa.pub
```

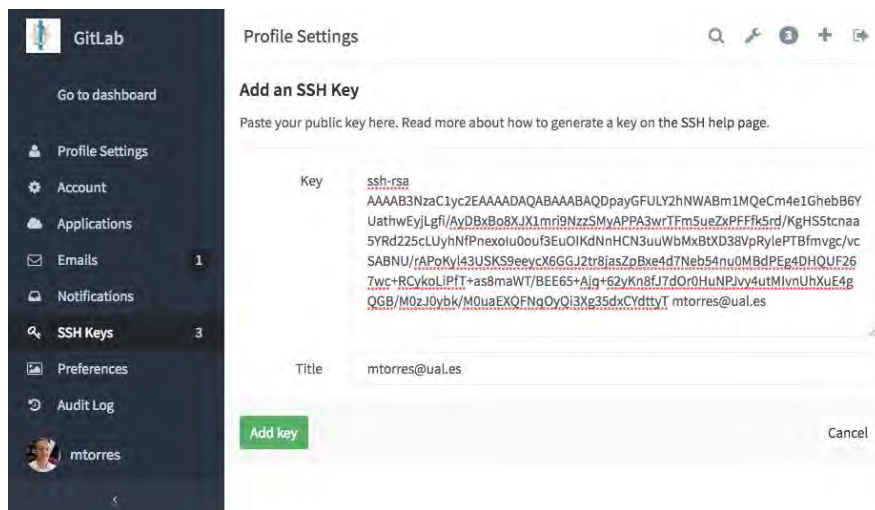


Figura 5.2: addsshkey

- `git clone` Clone un repo remoto (bare) en su equipo

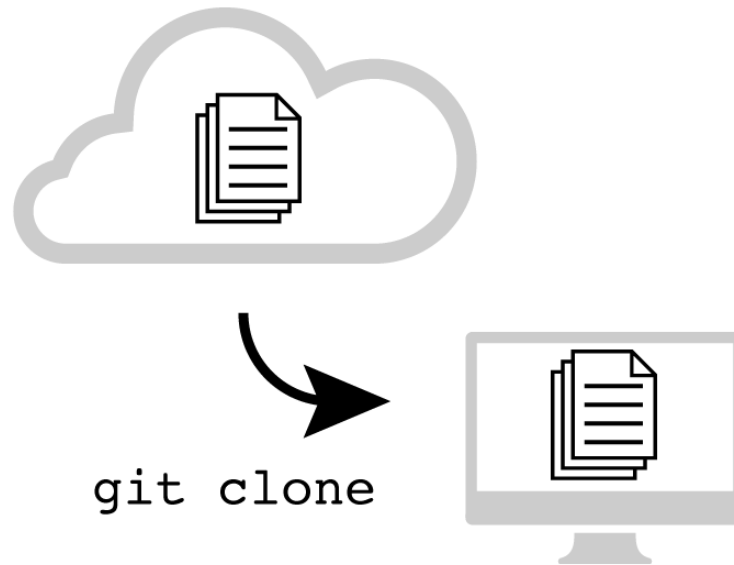


Figura 5.3: clone

Si el repositorio clonado es suyo podrá propagar los cambios del repositorio local al remoto. En cambio, si no es suyo sólo podrá subir cambios al repositorio remoto si está autorizado a ello.

Merge requests y Pull requests

Una forma de colaboración consiste en crear una copia del proyecto remoto. Esta operación se conoce como Fork. Al hacer un fork crea un repositorio remoto de nuestra propiedad, el cual es copia del repositorio original.

Para solicitar que nuestros cambios sean considerados para pasar al proyecto original realice un Merge request en GitLab (Pull request en GitHub). Esto inicia un proceso de revisión que finaliza con la aceptación, rechazo (o ignorando) de los cambios por parte del propietario del repositorio remoto original.

- Ejemplo Cree repo prueba en GitLab.

Clone repo remoto en la carpeta del usuario que esté usando `git clone` URL con el protocolo SSH.

Actualmente, hay un detalle sobre la identificación del certificado que usa `gitlabdoc.ual.es` al clonar con el protocolo mediante HTTPS. Si recibe este mensaje de error:

```
1 | server certificate verification failed. CAfile: /etc/ssl/certs/ca-  
   | certificates.crt CRLfile: none
```

puede salir del paso desactivando la verificación del certificado con este comando:

```
1 | $ git config --global http.sslverify false
```

Entrar al directorio del nuevo proyecto y Cree archivo README.md con el texto que aparece a continuación

```
1 # Git
2
3 Git is a free and open source distributed version control system
  designed to handle everything from small to very large projects
  with speed and efficiency.
```

Añada los cambios a la staging area.

Añada los cambios al repositorio con el mensaje Commit inicial

Al mostrar los cambios de aparecer algo similar a esto:

```
1 $ git log
2 commit 6aabc98f79034f3ea505b33bead4fa807e5ee59d (HEAD -> master)
3 Author: Jorge Dominguez Chavez <archivo@upta.edu.ve>
4 Date:   Tue Nov 27 18:14:20 2018 +0000
```

Commit inicial

- git push Envíe los cambios del repositorio local a un repo remoto

```
1 |git push remoto ramaLocal
```

En el ejemplo ejecute `git push origin master`

origin es el alias dado al repositorio remoto. master es la rama que contiene los commits que requiere subir. Al hacer push se suben todos los commits pendientes a sincronizarse con el repositorio remoto.

Tras hacer el push se debe obtener algo similar a esto:

```
1 $ git push origin master
2 Enumerating objects: 3, done.
3 Counting objects: 100% (3/3), done.
4 Delta compression using up to 2 threads
5 Compressing objects: 100% (2/2), done.
6 Writing objects: 100% (3/3), 337 bytes | 168.00 KiB/s, done.
7 Total 3 (delta 0), reused 0 (delta 0)
8 To gitlabdoc.ual.es:mtorres/prueba.git
9 * [new branch]      master -> master
```

Repositorio remoto al que se suben los cambios (origin).

Cree la rama master en el repositorio remoto que antes no estaba.

En el archivo `.git/config` de la carpeta de trabajo está la URL que corresponde a origin.

Si tiene dos repositorios remotos a sincronizar, añada una segunda URL en la sección de `[remote "origin"]`

5.3. Escenario colaborativo basado en control de acceso

El propietario del proyecto añade nuevos miembros configurando sus privilegios.

En GitLab, en propiedades del proyecto añade miembros al proyecto indicando sus privilegios (guest, reporter, developer, master) en función de las operaciones permitidas.

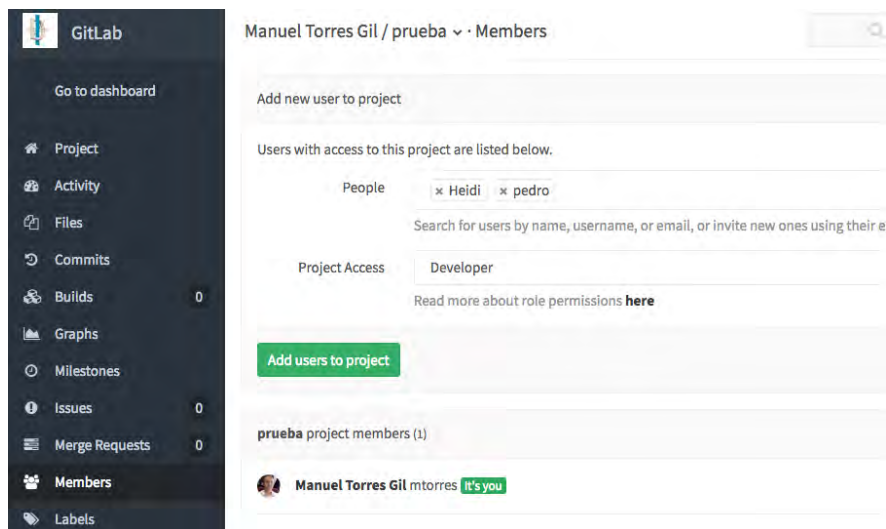


Figura 5.4: addmembers

A continuación, especifique las ramas donde estarán autorizados a subir cambios (Desproteger ramas).

En GitLab, en las propiedades del proyecto seleccione **Settings** \rightarrow **Protected Branches**

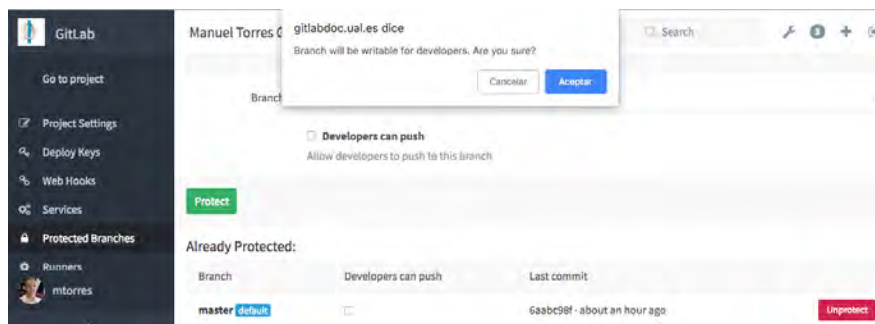


Figura 5.5: unprotectbranch

Por último, seleccione la rama e indique los desarrolladores que pueden realizar push

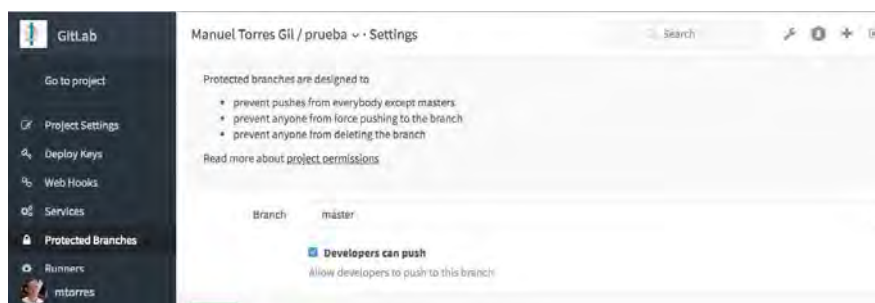


Figura 5.6: allowpush

5.4. Escenario colaborativo basado en merge requests (pull requests en GitHub)

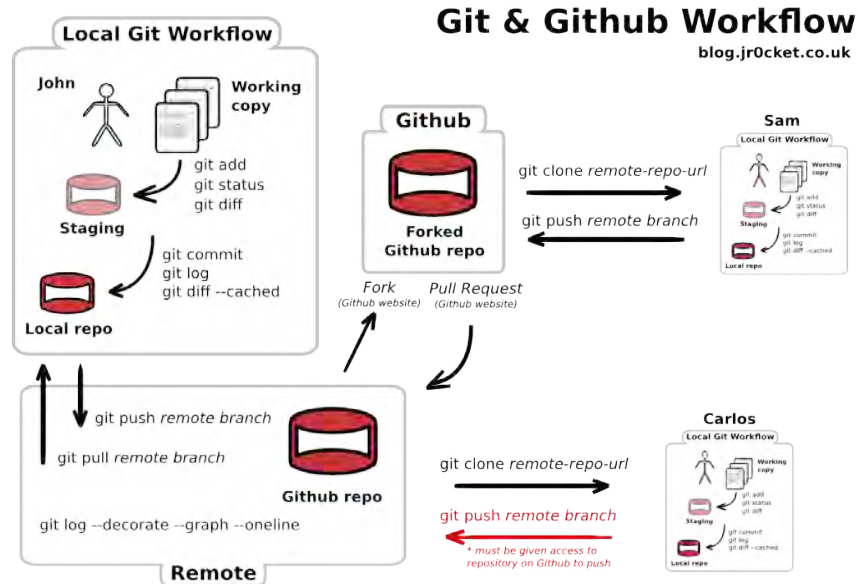


Figura 5.7: colaboración

5.4.1. origin y upstream

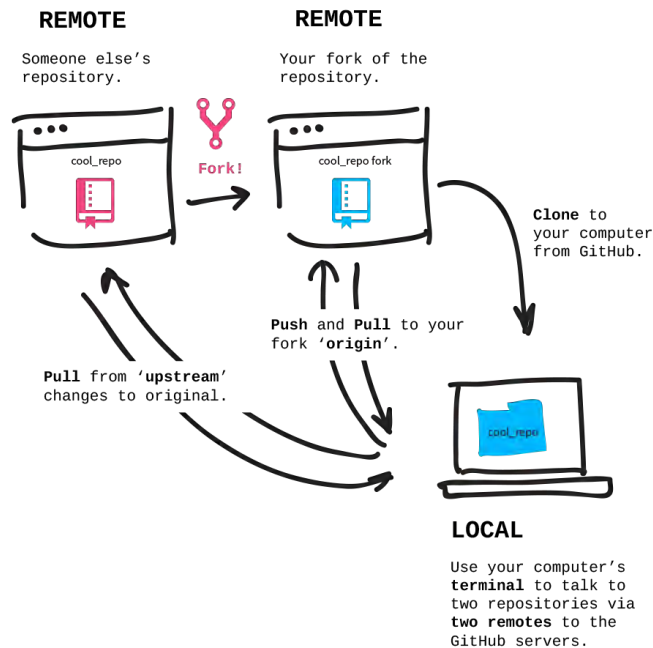


Figura 5.8: upstream

5.5. Integrando un repositorio remoto

Para integrar un repositorio local con un repositorio remoto ejecute el siguiente comando Git:

```
1 | git remote add origin <URL o PATH del repositorio remoto>
```

Ejemplo:

```
1 | git remote add origin https://github.com/j2logo/tutorial-flask.git
```

La palabra origin no es obligatoria. Es el nombre para identificar al repositorio remoto (puede agregar tantos repositorios remotos como requiera).

También puede hacer un checkout o clonar un repositorio remoto directamente. Esto ahorrará el paso de Crear el repositorio local, ya que se crea automáticamente y quedará conectado al repositorio remoto. Para ello ejecute:

```
1 | git clone <URL o PATH del repositorio remoto>
```

Ejemplo:

```
1 | git clone https://github.com/j2logo/tutorial-flask.git
```

Recuerde configurar el nombre y el email con el que firmará los commits.

Comandos Git para confirmar los cambios: add y commit

Una vez que quiera confirmar los cambios en su repositorio local hay dos pasos a seguir. En primer lugar, debe indicar qué archivos contienen los cambios a confirmar, para lo que usa el comando add:

```
1 | git add <nombre_archivo>
```

Para incluir todos los archivos con cambios, ejecute:

```
1 | git add .
```

Una vez que ha indicado los cambios a confirmar, ejecute un commit:

```
1 | git commit -m "Mensaje del commit"
```

Integrando los cambios en tu repositorio remoto

Cuando sus cambios en el repositorio local estén listos para ser integrados en el repositorio remoto, ejecute el comando:

```
1 | git push origin <nombre_rama_local>
```

Recuerde que origin es el nombre del repositorio remoto.

5.5.1. Comandos Git para Cree ramas

Las ramas nos permiten trabajar en distintas funcionalidades y en versiones a la vez de nuestra aplicación sin que los cambios afecten de unas a otras. Para saber más acerca de las ramas, verifique la entrada en la que explico cómo trabajar con ramas.

Para crear una rama y situars en ella ejecute:

```
1 | git checkout -b <nombre_rama>
```

Si requiere cambiarse a otra rama existente:

```
1 | git checkout <nombre_otra_rama>
```

Recuerde borrar una rama cuando ya no requiera trabajar más en ella:

```
1 | git branch -d <nombre_rama>
```

5.5.2. Fusionando el contenido de una rama en otra

Una vez que el trabajo en una rama secundaria de su repositorio local ha terminado, debe integrar los cambios en una rama principal. Por ejemplo, de una rama feature a la rama dev. Para ello, ejecute un merge de la rama feature desde dev de la siguiente manera:

```
1 | # Sitúate en dev
2 | git checkout dev
3 | # Ahora fusiona la rama feature-1
4 | git merge feature-1
```

El comando para integrar los cambios del repositorio remoto a su repositorio local es:

```
1 | git pull
```

5.5.3. Comandos Git para deshacer los cambios: revert

Es posible que en alguna ocasión los cambios realizados no sirvan y requiera volver a la versión anterior de un archivo. En ese caso, ejecute:

```
1 | git checkout -- <nombre_archivo>
```


Para deshacer los cambios locales y commits y traer la última versión estable del repositorio remoto, ejecute:

```
1 | git fetch origin
2 | git reset --hard <nombre_rama_remota>
```

Ejemplo:

```
1 | git reset --hard origin/dev
```

5.6. Recomendaciones

Estos son los principales comandos Git a utilizar al hora de trabajar en su repositorio. No obstante, se presentan una serie de recomendaciones y trucos para gestionar su repositorio sin pierdas el trabajo realizado:

- Si es posible, utilice este flujo para gestionar las ramas de su repositorio.
- Añada el archivo .gitignore al crear su repositorio local para no ensuciar el repositorio con archivos innecesarios.
- No incluya en su repositorio, los archivos de configuración de su proyecto que crea el IDE de desarrollo.
- Ejecute commits con regularidad en su repositorio local (siempre y cuando el código esté limpio de fallos).
- Sólo integre su repositorio local con el remoto (haciendo un push) cuando el código esté realmente listo para compartir y libre de errores.

Para evitar conflictos en el repositorio, intente el flujo siguiente:

```
1 | # Clono de un repositorio remoto: Github, Gitlab, ...
2 | # Previamente he configurado este repositorio
3 | git clone <URL_repositorio_remoto>
4 | # Establezco los parámetros de mi usuario
5 | git config user.name "J2logo"
6 | git config user.email juanjo@j2logo.com
7 | # Creo la rama dev a partir de master
8 | git checkout -b dev
9 | # Pusheo la rama dev en el repositorio remoto
10 | git push origin dev
11 | # Cuando tengo que desarrollar algo, creo la
12 | # rama correspondiente
13 | git checkout -b feature/feature_1
14 | # Voy haciendo commits sobre la rama feature_1
15 | # hasta que la funcionalidad está terminada
16 | git commit -m "Msg 1"
17 | git commit -m "Msg 2"
18 | # Cuando la funcionalidad está lista, la paso
19 | # a la rama dev. Para ello, me sitúo en dev
20 | git checkout dev
21 | # Actualizo la rama dev con posibles cambios
```

```
22 # en el repositorio remoto
23 git pull origin dev
24 # Ahora sí, integro los cambios de la rama
25 # feature_1 en la rama dev
26 git merge --no-ff feature/feature_1
27 # Pusheo la rama dev para que el resto del
28 # equipo pueda ver los cambios
29 git push origin dev
30 # Borro la rama feature_1 porque ya no
31 # me va a hacer falta
32 git branch -d feature/feature_1
33 ...
```

Capítulo 6

Ejercicios selectos

- A.- Ejercicio: Haz un fork del repositorio de DECIDE para trabajar en el. En esta clase, vamos a introducir una nueva característica dentro del código de Decide. Concretamente, se pide que modifiquemos el booth para que muestre exclusivamente voting.name a mayor tamaño de letras. Para esto, crearemos una nueva rama con nombre improveBooth.

En la nueva rama creada vamos a realizar las modificaciones necesarias dentro del fichero `decide/decide/booth/templates/booth/booth.html` y añade esa información en el `readme.md`. Haz un commit de estos cambios.

Corrige el último commit para que el nombre de la web sea Decide-10-11 en lugar de Decide. Subimos esta rama al repositorio en Github.

Haz un merge de la rama `improveBooth` en master. Asegúrate que tu compañero ha clonado el repositorio según el Ejercicio (b) antes de seguir y da permiso de escritura a tu compañero. Sube los cambios de la rama master al repositorio de Github. Consulta como ha quedado el log del repositorio y añádelo al README. Vuelve a subir los cambios del README al repositorio.

- B.- Ejercicio: Realiza una operación de cherrypicking con el primer commit.

Aborta la operación en el último paso si da lugar a conflictos.

Clona el repositorio creado por un compañero en el ejercicio (a).

Modifica el método `greeting` del booth de la siguiente forma (Poner decide donde decía Decide-10-11). Cuando tu compañero termine el ejercicio (a) Sube los cambios al repositorio en Github.

- C.- Ejercicio: Deshaz el commit que introdujo el cambio.

Haz que git borre el seguimiento (ya efectuado) de la carpeta `vagrant` y añádelo al `gitignore`. Actualiza los índices del repositorio sin hacer update.

Conclusiones

La plataforma GitHub (u otra similar) aplicada a la docencia no debe ser considerada como un mero servicio de alojamiento de repositorios de Git en la Web con funcionalidades extra. En la experiencia docente descrita en este trabajo hemos comenzado a vislumbrar su potencial papel como herramienta de aprendizaje y de gestión de la enseñanza. Es necesario profundizar en el análisis de esta plataforma como herramienta docente para dimensionar adecuadamente su papel. Futuras investigaciones deben abordar aspectos cuantitativos y cualitativos relacionados, por ejemplo, con la satisfacción de alumnos y profesores, los resultados académicos, y la evolución de la carga docente. En cualquier caso, la aparición de plataformas como GitHub plantea nuevos retos a la docencia como, por ejemplo, su integración con las plataformas educativas existentes, el desarrollo de herramientas de apoyo orientadas a la gestión educativa, la gestión de nuevos retos relacionados con la propiedad intelectual y la ética profesional, y el desarrollo de buenas prácticas para un correcto aprovechamiento de esta nueva generación de herramientas de docencia.

Bibliografía

- [1] ANDREW MENEELY Y LAURIE WILLIAMS., *On preparing students for distributed software development with a synchronous, collaborative development platform*. En *40th ACM technical symposium on Computer science education*, página 529, New York, New York, USA, Marzo 2009. ACM Request Permissions, ISBN 9781605581835..
- [2] MICHAEL COCHEZ, VILLE ISOMÖTTÖNEN, VILLE TIRRONEN Y JONNE ITKONEN., *How Do Computer Science Students Use Distributed Version Control Systems?* En Vadim Ermolayev, HeinrichCayr, Mykola Nikitchenko, Aleksander Spivakovsky y Grygoriy Zholtkevych (editores): *Metadata and Semantic Research*, páginas 210–228. Springer International Publishing, 2013, ISBN 978-3-319-03997-8
- [3] KEN T N HARTNESS., *Eclipse and CVS for group projects*. *Journal of Computing Sciences in Colleges*, 21(4):217–222, Abril 2006
- [4] OREN LAADAN, JASON NIEH Y NICOLAS VIENNOT., *Teaching operating systems using virtual appliances and distributed version control*. En *the 41st ACM technical symposium*, página 480, New York, New York, USA, 2010. ACM Press, ISBN 9781450300063
- [5] VILLE ISOMÖTTÖNEN Y MICHAEL COCHEZ., *Challenges and Confusions in Learning Version Control with Git*. En *Information and Communication Technologies in Education, Research, and Industrial Applications*, páginas 178–193. Springer International Publishing, Junio 2014, ISBN 978-3-319-13205-1.
- [6] KAREN L REID Y GREGORY V WILSON., *Learning by doing: introducing version control as a way to manage student assignments*. *SIGCSE*, páginas 272–276, 2005.
- [7] IVAN MILENTIJEVIC, VLADIMIR CIRIC Y OLIVER VOJINOVIC., *Version control in project-based learning*. *Computers & Education*, 50(4):1331–1338, Mayo 2008
- [8] GITHUB. PRESS, 2015., <https://github.com/about/press>.
- [9] MARISA WHITAKER. GITHUB CO-FOUNDER CHRIS WANSTRATH SHARES HIS STORY, ABRIL 2014., <http://magazine.uc.edu/content/magazine/favorites/webonly/wanstrath.html>.
- [10] ZHIGUANG XU., *Using Git to Manage Capstone Software Projects* . En *7th International Multi-Conference on Computing in the Global Information Technology*, 2012.
- [11] LAURA DABBISH, COLLEEN STUART, JASON TSAY Y JIM HERBSLEB., *Social coding in GitHub*. En *ACM 2012 conference on Computer Supported Cooperative Work*, páginas 1277–1286, New York, New York, USA, 2012. ACM Press, ISBN 9781450310864.

- [12] PAUL SAWERS., *GitHub Wants Schools to Collaborate Around code*, Febrero 2014. <http://thenextweb.com/insider/2014/02/11/github-wants-schools-collaborate-code/>.
- [13] COLLINS-SUSSMAN, B., FITZPATRICK, B., & PILATO, M. (2004)., *Version control with subversion*. O'Reilly Media, Inc.
- [14] BROCKMEIER, K. (2012)., *Counting Contributions: Who Wrote Linux 3.2?* *Linux.com*, News for the Open Source Professional. <https://www.linux.com/learn/counting-contributions-who-wrote-linux-32>
- [15] HENSON, V., & GARZIK, J. (2002)., *Bitkeeper for kernel developers*. In *Ottawa Linux Symposium* (p. 197).
- [16] MCMILLAN, R. (2005)., *After controversy, Torvalds begins work on "git"*. IDG News Service. https://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/
- [17] BARR, J. (2005)., *Bitkeeper and Linux: The end of the road?* *Linux.com*, News for the Open Source Professional. Recuperado de <https://www.linux.com/news/bitkeeper-and-linux-end-road>
- [18] DE ALWIS, B., SILLITO, J. (2009)., *Why are software projects moving from centralized to decentralized version control systems?*. In *Proceedings of the 2009 ICSE Workshop on cooperative and human aspects on software engineering* (pp. 36-39). IEEE Computer Society.
- [19] FINLEY, K. (2011), *GitHub has surpassed Sourceforge and Google Code in popularity*. ReadWrite.com. <http://readwrite.com/2011/06/02/github-has-passed-sourceforge/>
- [20] REID, K. L., & WILSON, G. V. (2005)., *Learning by doing: introducing version control as a way to manage student assignments*. *ACM SIGCSE Bulletin* (Vol. 37, No. 1, pp.272-276). ACM.
- [21] MILENTIJEVIC, I., CIRIC, V., & VOJINOVIC, O. (2008)., *Version control in project-based learning*. *Computers & Education*, 50(4), 1331-1338.

Acerca del autor

Graduado en Física, Facultad de Ciencias, Universidad Nacional Autónoma de México (UNAM), Doctor en Ciencias de la Computación, Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas (IIMAS) UNAM. Especialista en Economía Matemática, Centro de Investigación y Docencia Económica, CIDE, México. Cursante del Doctorado en Minería de Datos, Modelos y Sistemas Expertos por la Universidad de Illinois en Urbana-Champaigns (USA).

Ha sido profesor en la Facultad de Química y en la Facultad de Ingeniería, UNAM. Fue profesor en el Departamento de Ciencias Básicas, Universidad de Las Américas en Cholula, (UDLAP), Puebla, México y, durante seis años, profesor visitante en la Universidade Federal de Rio Grande do Sul (Brasil), profesor y jefe de sistemas postgrados de agronomía y veterinaria, Universidad Central de Venezuela (UCV), actualmente es profesor del Departamento de informática y del Departamento de Postgrado, Universidad Politécnica Territorial de Aragua (UPT Aragua), Venezuela.

Expositor y conferencista a nivel nacional e internacional.

Es asesor en mejora de procesos, gestión de proyectos, desarrollo de software corporativo en los sectores de servicios, banca, industria y gobierno.

El Dr. Domínguez es un especialista reconocido en base de datos, desarrollo de software y servidores en el área del software libre, así como un experto en LINUX DEBIAN.

En la actualidad orienta su trabajo a la creación y desarrollo de equipos de software de alto desempeño. Autor de múltiples artículos y libros sobre la materia.

