

ENTORNOS DE DESARROLLO

IES Santiago Hernández
Curso 2017-2018
Ignacio Agudo Sancho

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones

- ⦿ Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.
- ⦿ Permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más.
- ⦿ Si rompemos o perdemos algo, lo podemos recuperar fácilmente.

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones

- ⦿ SCV locales:
- ⦿ Copiar los archivos a otro directorio.
 - Simple pero propenso a errores.
- ⦿ Mejora: BBDD locales con registros de las copias.
 - Guarda conjuntos de parches

4. Control de versiones

- ⦿ SCV centralizados:
- ⦿ Un único servidor contiene todos los archivos versionados.
- ⦿ Los clientes los descargan desde ese lugar central.
- ⦿ Válido durante muchos años.
- ⦿ Muchas ventajas frente a SCV locales:
 - Por ejemplo, todo el mundo puede saber (hasta cierto punto) en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un sistema centralizado (CVCS) que tener que lidiar con bases de datos locales en cada cliente.

4. Control de versiones

- SCV centralizados:

- Inconvenientes:

- Si ese servidor se cae durante una hora, entonces durante esa hora nadie puede colaborar o guardar cambios versionados de aquello en que están trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, pierdes absolutamente todo

4. Control de versiones

- SCV distribuidos (DVCS):
- Los clientes replican completamente el repositorio descargándose la última versión.
- Si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos

4. Control de versiones

- DVCS:
- Muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente simultáneamente dentro del mismo proyecto. Esto te permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones: Git

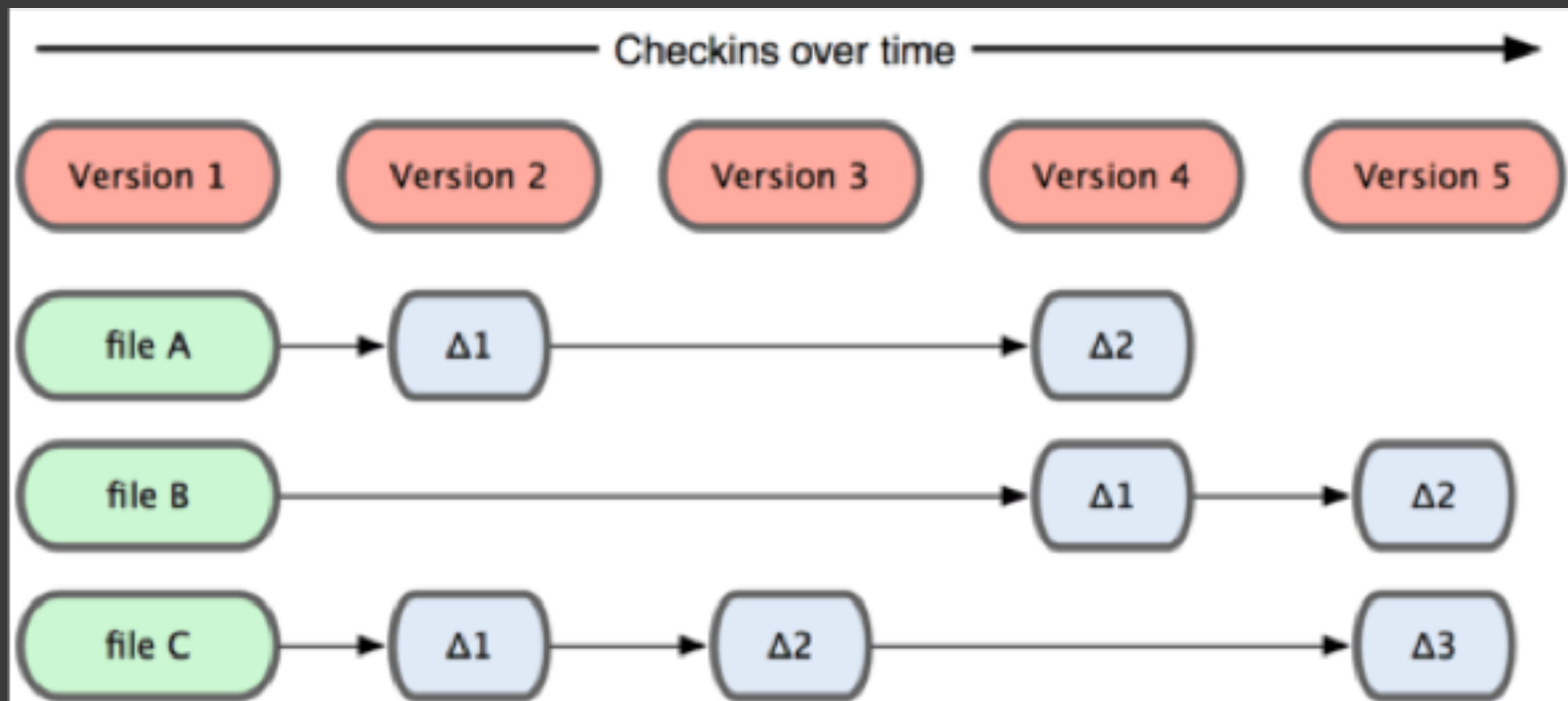
- ⦿ BitKeeper → Propietario para el núcleo de Linux
- ⦿ BitKeeper deja de ser ofrecida de manera gratuita
- ⦿ La comunidad de desarrollo de Linux decide desarrollar su propia herramienta basada en algunas de las lecciones aprendidas.
- ⦿ Objetivos:
 - Velocidad
 - Diseño sencillo
 - Apoyo al desarrollo no lineal (ramas paralelas)
 - Completamente distribuido
 - Capaz de manejar grandes proyectos de manera eficiente.

4. Control de versiones: Git

- Modela sus datos como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea.
- Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

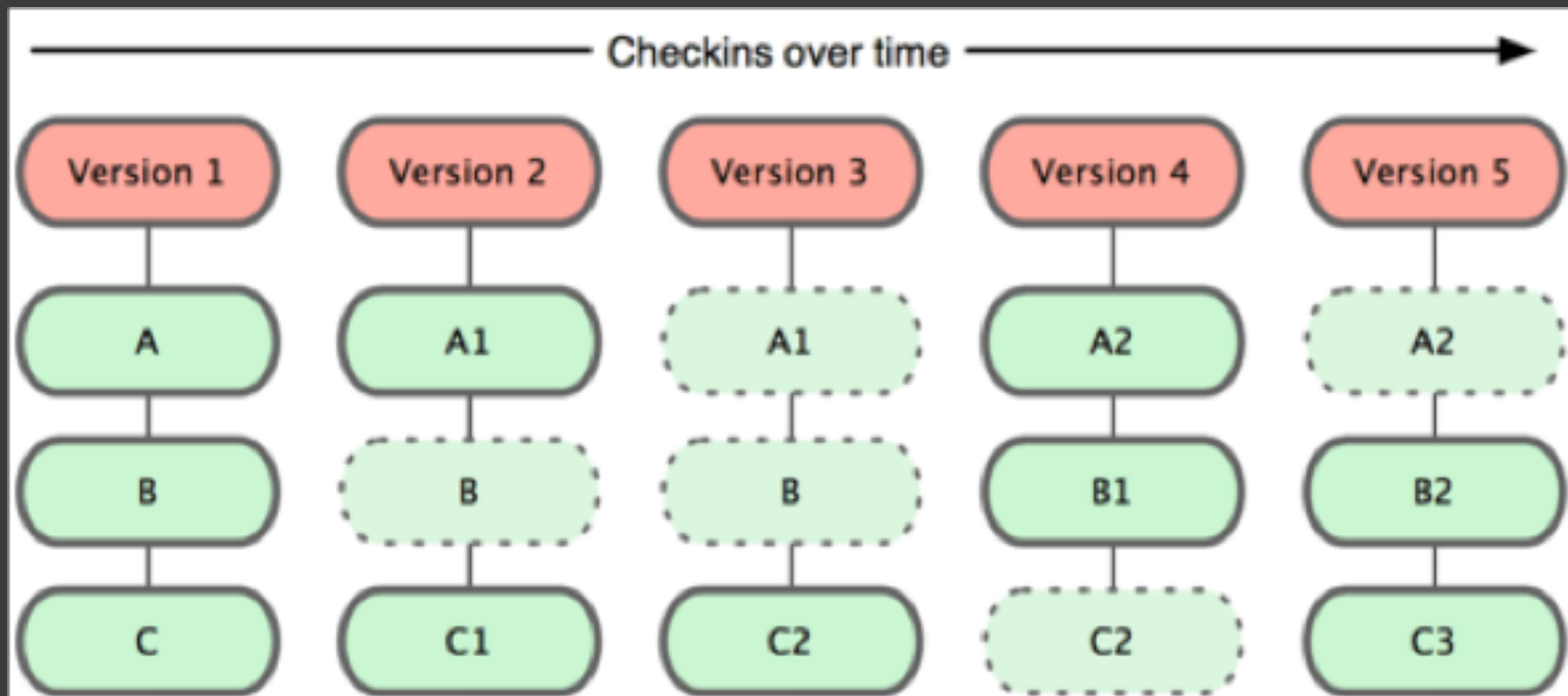
4. Control de versiones: Git

- Otros sistemas almacenan los datos como cambios de cada archivo respecto a una versión base.



4. Control de versiones: Git

- Git almacena la información como instantáneas del proyecto a lo largo del tiempo



4. Control de versiones: Git

⦿ GIT: Operaciones Locales

- No es necesario salir a la red
- Velocidad muy alta de las operaciones
- Lee de tu base de datos local
- Calcula diferencias entre ficheros en local
- No limita las acciones sin conexión

4. Control de versiones: Git

⦿ Integridad

- Asigna un identificador a los cambios (40 hex)
- Imposible cambiar cosas sin que Git lo sepa

⦿ Sólo añade información

- No se puede hacer algo que no se pueda deshacer
- Más aún cuando usas un repositorio para subir los datos

4. Control de versiones: Git

⦿ Operaciones locales

- Modificamos ficheros en nuestro directorio de trabajo.
- Preparamos los ficheros añadiéndolos al área de preparación.
- Confirmamos los cambios, almacenando una instantánea de manera permanente en nuestro directorio de Git.

4. Control de versiones: Git

⦿ Operaciones locales

- Directorio de trabajo: es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.
- Área de preparación: es un sencillo archivo, contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación.
- Directorio de Git: es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador.

4. Control de versiones: Git

⦿ Operaciones locales

- Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed).
- Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged).
- Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones: Git

⦿ GIT: Instalación

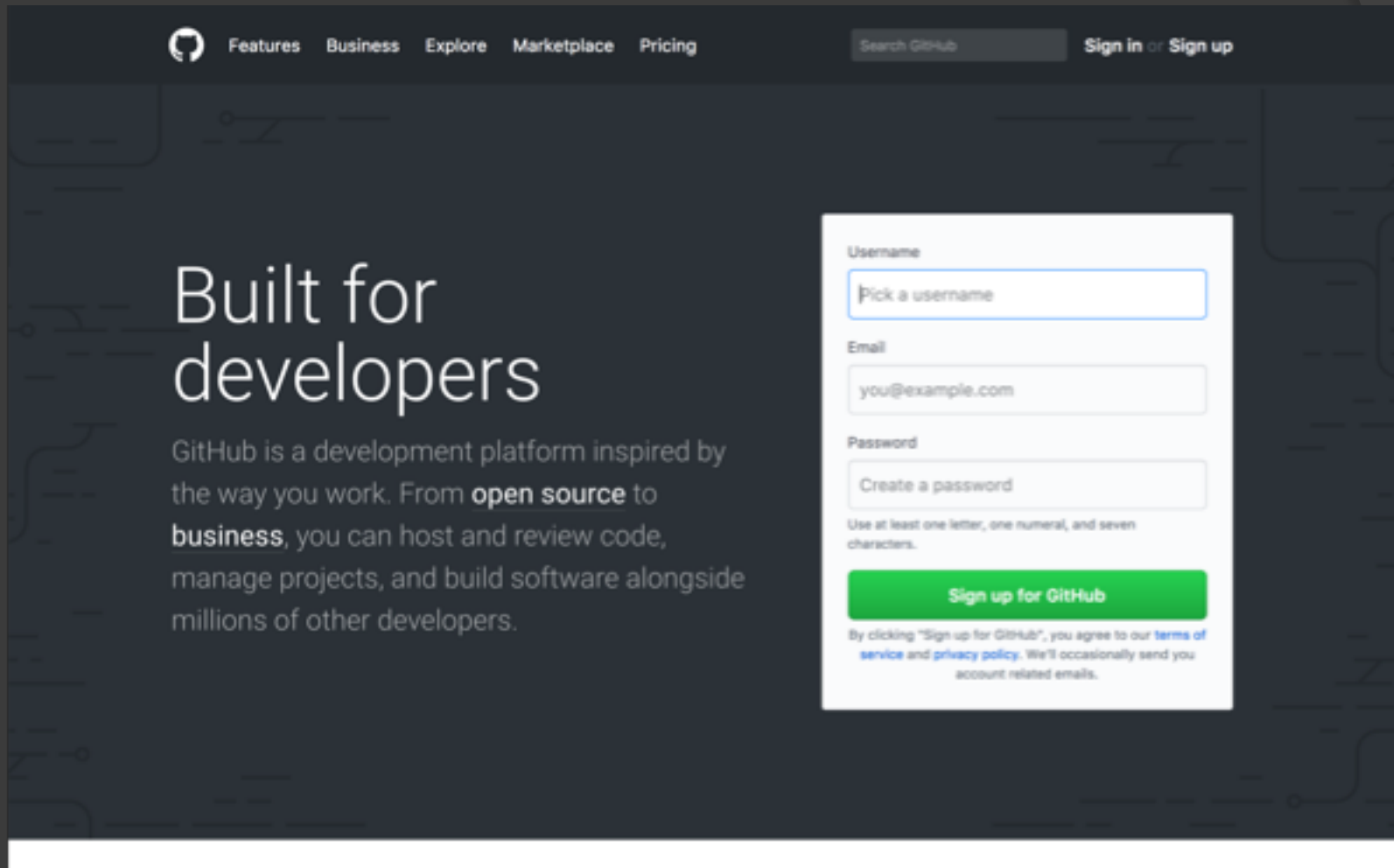
- Git es multiplataforma, y tiene maneras diferentes para instalar dependiendo del sistema operativo en el que lo queramos utilizar.
- Para instalar Git en Ubuntu, utilizaremos
 - \$ apt-get install git
- Si necesitamos permisos de usuario:
 - \$ sudo apt-get install git

4. Control de versiones: Git

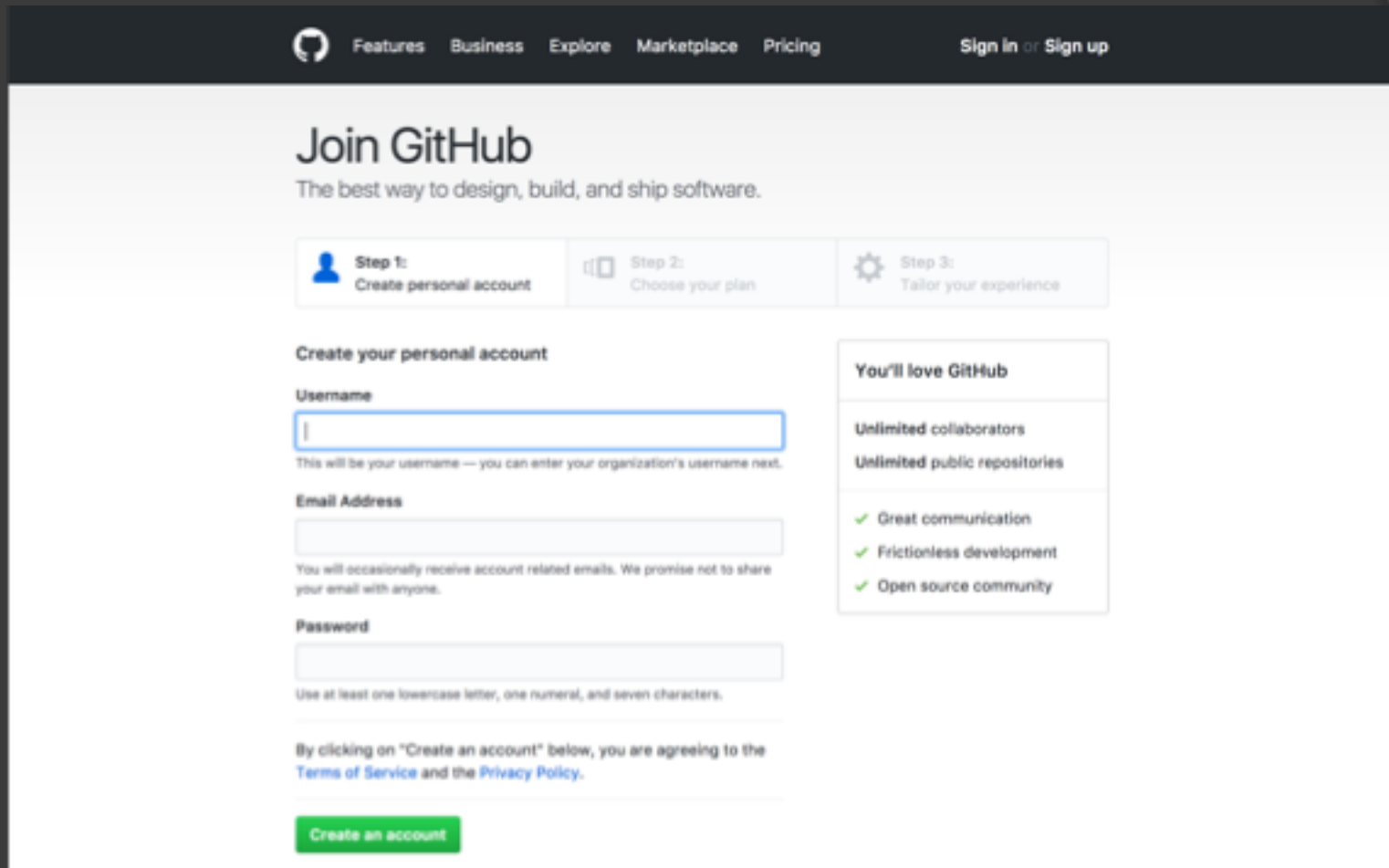
⦿ GIT: Configuración inicial

- Crear una cuenta en <https://github.com/>

4. Control de versiones: Git

A screenshot of the GitHub website's sign-up page. The background is dark with a faint, light-colored branching diagram. The top navigation bar is dark and contains the GitHub logo, links for 'Features', 'Business', 'Explore', 'Marketplace', and 'Pricing', a search bar labeled 'Search GitHub', and links for 'Sign in' or 'Sign up'. The main content area on the left has the heading 'Built for developers' in large white text, followed by a paragraph: 'GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside millions of other developers.' On the right, there is a white sign-up form with fields for 'Username' (placeholder: 'Pick a username'), 'Email' (placeholder: 'you@example.com'), and 'Password' (placeholder: 'Create a password'). Below the password field is a note: 'Use at least one letter, one numeral, and seven characters.' A prominent green button labeled 'Sign up for GitHub' is at the bottom of the form. Below the button, a small line of text states: 'By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.'

4. Control de versiones: Git



The screenshot shows the GitHub sign-up page. At the top is a dark navigation bar with the GitHub logo, links for Features, Business, Explore, Marketplace, and Pricing, and buttons for Sign in or Sign up. The main heading is 'Join GitHub' with the tagline 'The best way to design, build, and ship software.' Below this is a three-step progress bar: Step 1 (Create personal account), Step 2 (Choose your plan), and Step 3 (Tailor your experience). The 'Create your personal account' section contains three input fields: Username, Email Address, and Password, each with a descriptive note below it. To the right, a box titled 'You'll love GitHub' lists benefits: Unlimited collaborators, Unlimited public repositories, Great communication, Frictionless development, and Open source community. At the bottom, there is a link to the Terms of Service and Privacy Policy, and a green 'Create an account' button.

Features Business Explore Marketplace Pricing Sign in or Sign up

Join GitHub

The best way to design, build, and ship software.

Step 1: Create personal account

Step 2: Choose your plan

Step 3: Tailor your experience

Create your personal account

Username

This will be your username — you can enter your organization's username next.

Email Address

You will occasionally receive account related emails. We promise not to share your email with anyone.

Password

Use at least one lowercase letter, one numeral, and seven characters.


By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).

Create an account

You'll love GitHub

- Unlimited collaborators
- Unlimited public repositories
- Great communication
- Frictionless development
- Open source community

4. Control de versiones: Git



The screenshot shows the GitHub onboarding interface for a new user. At the top, there's a dark navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, Marketplace, and Explore. The main heading is "Welcome to GitHub" followed by a personalized message "You've taken your first step into a larger world, @NachoAS." Below this, a progress bar shows three steps: "Completed: Set up a personal account", "Step 2: Choose your plan" (the current step), and "Step 3: Tailor your experience". The "Choose your plan" section offers two options: "Unlimited public repositories for free." (selected) and "Unlimited private repositories for \$7/month." To the right, a box lists features included in both plans: Collaborative code review, Issue tracking, Open source community, Unlimited public repositories, and Join any organization. At the bottom, there are checkboxes for "Help me set up an organization next" and "Send me updates on GitHub news, offers, and events", followed by a green "Continue" button.

Search GitHub Pull requests Issues Marketplace Explore

Welcome to GitHub

You've taken your first step into a larger world, @NachoAS.

✓ Completed Set up a personal account

Step 2: Choose your plan

⚙ Step 3: Tailor your experience

Choose your personal plan

☒ Unlimited public repositories for free.

☐ Unlimited private repositories for \$7/month.

Don't worry, you can cancel or upgrade at any time.

☐ Help me set up an organization next
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees. [Learn more about organizations](#)

☐ Send me updates on GitHub news, offers, and events
Unsubscribe anytime in your email preferences. [Learn more](#)

Continue


Both plans include:


- ✓ Collaborative code review
- ✓ Issue tracking
- ✓ Open source community
- ✓ Unlimited public repositories
- ✓ Join any organization


4. Control de versiones: Git

Welcome to GitHub

You'll find endless opportunities to learn, code, and create, @NachoAS.

 **Completed**
Set up a personal account

 **Step 2:**
Choose your plan

 **Step 3:**
Tailor your experience

How would you describe your level of programming experience?

☐ Very experienced ☐ Somewhat experienced ☐ Totally new to programming

What do you plan to use GitHub for? (check all that apply)

☐ School projects ☐ Design ☐ Project Management
☐ Development ☐ Research ☐ Other (please specify)

Which is closest to how you would describe yourself?

☐ I'm a hobbyist ☐ I'm a professional ☐ I'm a student
☐ Other (please specify)

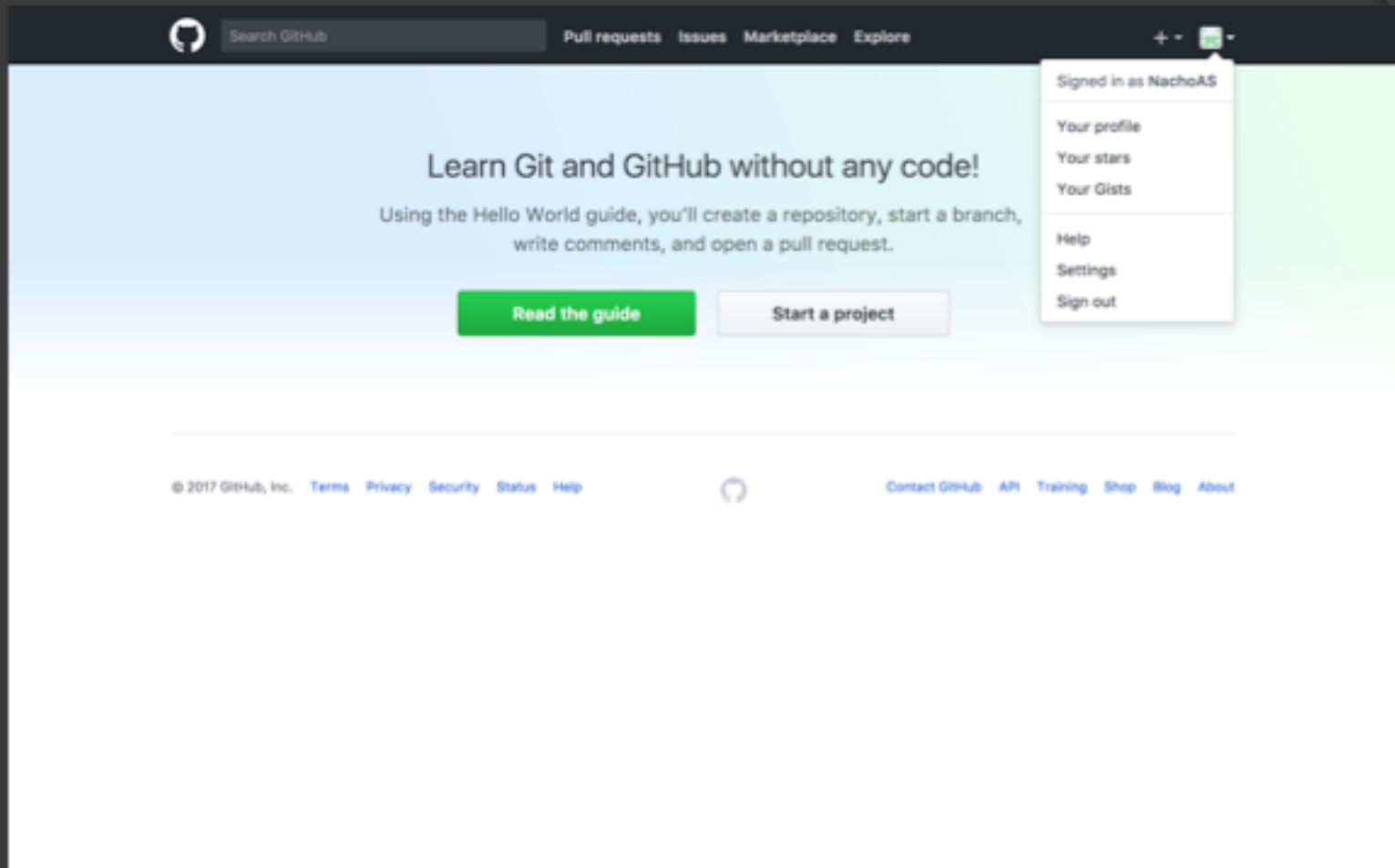
What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

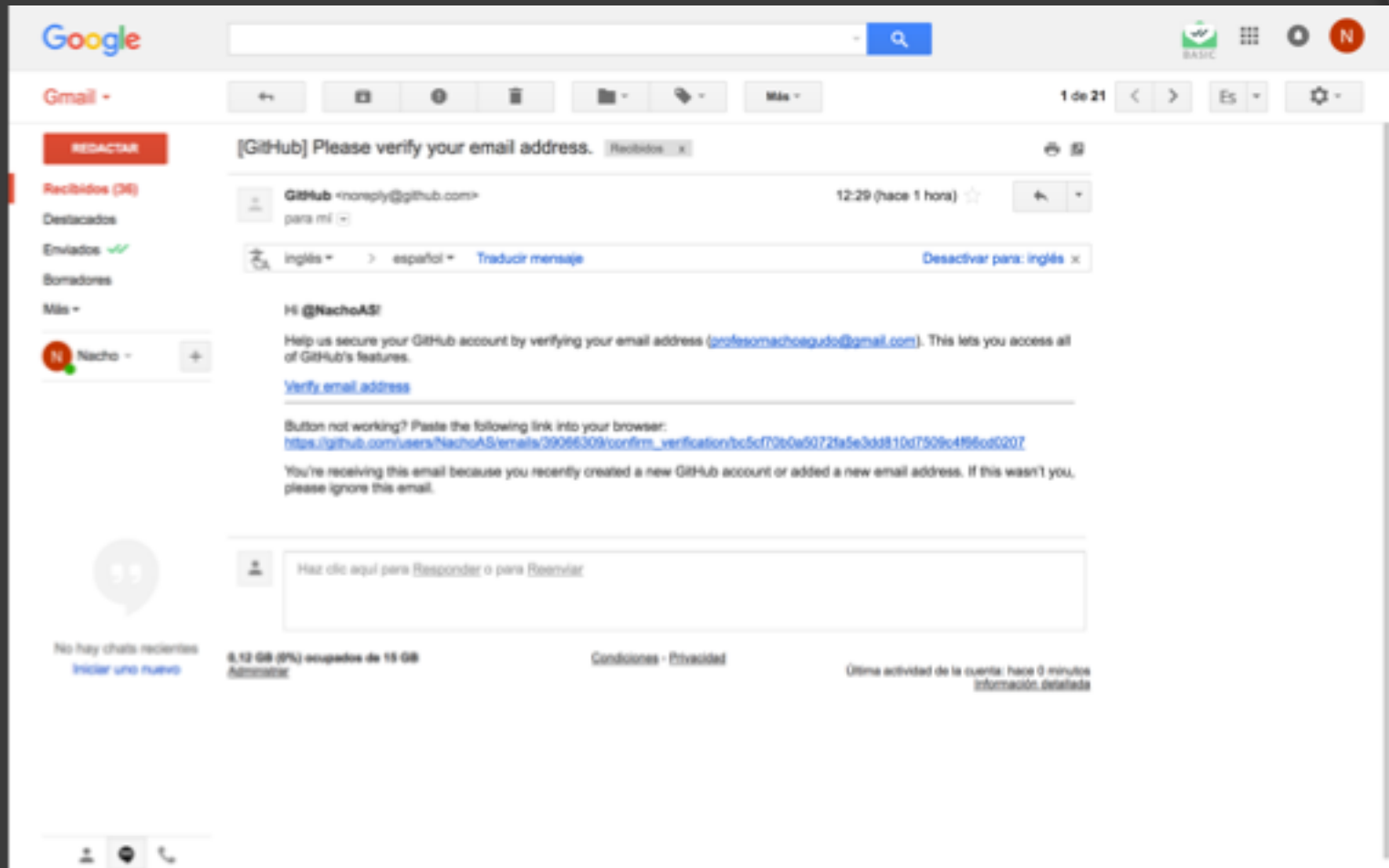
Submit

[skip this step](#)

4. Control de versiones: Git



4. Control de versiones: Git



4. Control de versiones: Git

- ⦿ Configurar Git en local:
- ⦿ Desde la línea de comandos:
 - \$ git config --global user.name “tuNombreDeUsuario”
 - \$ git config --global user.email [tuemail@tuemail.com](#)
- ⦿ Para ver nuestro fichero de configuración:
 - \$ git config --list

4. Control de versiones: Git

⦿ Primeros pasos con Git:

- Dudas:
 - `git help <comando>`
 - `git <comando> --help`
 - `man git-<comando>`
- Ejemplo:
 - `$ git help config`

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones: Git

● Cómo tener un proyecto Git:

- Coger un proyecto o un directorio existente e importarlo en Git (subirlo a Git).
- Clonar un repositorio Git existente desde otro servidor.

4. Control de versiones: Git

- Inicializar un repositorio en un directorio existente:
 - \$ git init
- Esto crea un nuevo subdirectorio llamado .git que contiene todos los archivos necesarios del repositorio —un esqueleto de un repositorio Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento.

4. Control de versiones: Git

- ⦿ Clonar un repositorio ya existente desde un servidor Git:
 - Para descargar a nuestro entorno local un repositorio ya existente, por ejemplo, para contribuir a un proyecto, utilizaremos
 - `$ git clone <url>`
 - Por ejemplo:
 - `$ git clone https://github.com/nacho/entornos.git`

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ **Git**
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - **Trabajando con Git**
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones: Git

- Empezar a controlar versiones de archivos:
- Añadir ficheros con cambios al stage:
 - `$ git add`
 - Podemos seleccionar qué ficheros queremos guardar:
 - `$ git add *.java`
 - `$ git add ruta/nombrefichero.extension`
 - Podemos decirle que añada todos los archivos que tienen cambios:
 - `$ git add .`

4. Control de versiones: Git

- Para ver el estado de nuestros ficheros podemos ejecutar :
 - `$ git status`
- Para ver las diferencias:
 - `$ git diff`
 - `$ git diff --staged`
- Para ignorar ficheros a subir, hay que crear un archivo llamado `.gitignore` donde meteremos reglas para los ficheros a evitar.

4. Control de versiones: Git

- El siguiente paso después de añadir nuestros ficheros al stage area sería hacer un commit.
- Con el commit grabamos los cambios en nuestro repositorio local.
- Realizamos un commit de la siguiente manera:
 - `$ git commit -m "Descripción de los cambios"`
 - Los mensajes del commit son muy importantes. Deberíamos ser capaces de entender qué se está haciendo y la cronología simplemente con estos mensajes.

4. Control de versiones: Git

- Finalmente, para subir los cambios confirmados en nuestro repositorio local, a otro repositorio, utilizaremos:
 - `$ git push`
 - `$ git push nombreRepositorio nombreRama`
 - Ej: `$ git push origin dev`
- Origin sería el repositorio que tenemos enlazado con un git clone, es decir, de donde hemos clonado nuestra copia local. Dev sería la rama de desarrollo donde vamos a “empujar” los cambios.

4. Control de versiones: Git

- ⦿ Crear una cuenta en Github
- ⦿ Clonar el repositorio creado en clase.
- ⦿ Modificar el fichero de usuarios.
- ⦿ Subir la modificación al repositorio original en Github.

4. Control de versiones: Git

- ⦿ Instalación de Git en Windows
 - <http://git-scm.com/download>
- ⦿ Crear un nuevo repositorio en nuestra cuenta de Github.
- ⦿ Subir el workspace del módulo de Programación al nuevo repositorio de Git.

4. Control de versiones: Git

- Ver el histórico de cambios de un repositorio:
 - Primero clonamos el siguiente repositorio para hacer pruebas en local:
 - `$ git clone git://github.com/schacon/simplegit-progit.git`
 - Y a continuación ejecutamos:
 - `$ git log`
 - Podemos usar `-p` para mostrar las diferencias introducidas en cada confirmación.
 - Podemos usar `-2` para mostrar las dos últimas entradas del histórico.
 - Añadiendo `--stat` indica una lista de los archivos modificados con información de cada uno
 - Para ver otro formato de salida podemos usar `--pretty`, a la que asignaremos un valor: `oneline`, `short`, `full`, `fuller`
 - Con `format` podemos especificar nuestro propio formato.

4. Control de versiones: Git

● Ver el histórico de cambios de un repositorio:

- `$ git log -p`
- `$ git log -p -2`
- `$ git log --stat`
- `$ git log --pretty=oneline`
- `$ git log --pretty=format:"%h - %an, %ar : %s"`
- `$ git log --since=2.weeks`
 - Este comando acepta muchos formatos. Puedes indicar una fecha concreta ("2017-01-15"), o relativa, como "2 years 1 day 3 minutes ago" ("hace 2 años, 1 día y 3 minutos").
 - También puedes filtrar la lista para que muestre sólo aquellas confirmaciones que cumplen ciertos criterios. La opción `--author` te permite filtrar por autor, y `--grep` te permite buscar palabras clave entre los mensajes de confirmación. (Ten en cuenta que si quieres aplicar ambas opciones simultáneamente, tienes que añadir `--all-match`, o el comando mostrará las confirmaciones que cumplan cualquiera de las dos, no necesariamente las dos a la vez.)

4. Control de versiones: Git

⦿ Deshaciendo cambios:

- Modificar la última confirmación
 - `$ git commit -m "primer commit"`
 - `$ git add fichero_olvidado`
 - `$ git commit --amend`
- Deshacer la última confirmación:
 - `$ git add <fichero>`
 - `$ git status`
 - `$ git reset HEAD <fichero>`

4. Control de versiones: Git

⦿ Deshaciendo cambios:

- Deshacer todos los cambios que hemos hecho en local sobre un archivo:
 - `$ git checkout -- <fichero>`

4. Control de versiones: Git

⦿ Deshaciendo cambios:

- Deshacer todos los cambios que hemos hecho en local sobre un archivo:
 - \$ git checkout -- <fichero>
- MUCHO CUIDADO CON ESTE COMANDO O
PODEMOS PERDER TODO EL TRABAJO
REALIZADO.

4. Control de versiones: Git

⦿ Creando etiquetas

- Mostrar la lista de etiquetas
 - `$ git tag`
- Crear una etiqueta:
 - `$ git tag -a v1.4 -m 'my version 1.4'`
- Ver los datos de la etiqueta
 - `git show v1.4`

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

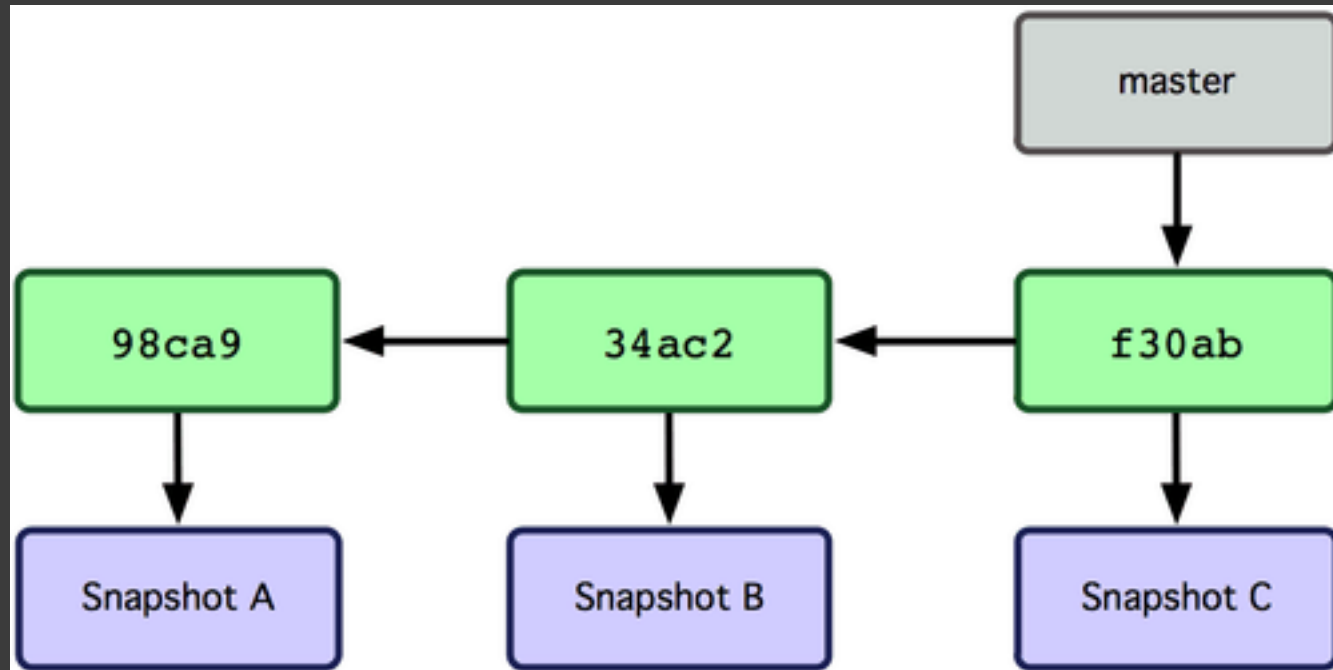
4. Control de versiones: Git

⦿ Utilizando Ramas

- Remember: Cada confirmación de cambios es como una foto del estado del proyecto.
- La rama por defecto de Git es la rama master.
- Con la primera confirmación de cambios que realicemos se creará la rama principal, apuntando a dicha confirmación.
- La rama master avanzará automáticamente para apuntar a la última confirmación que hayamos hecho.

4. Control de versiones: Git

● Utilizando Ramas



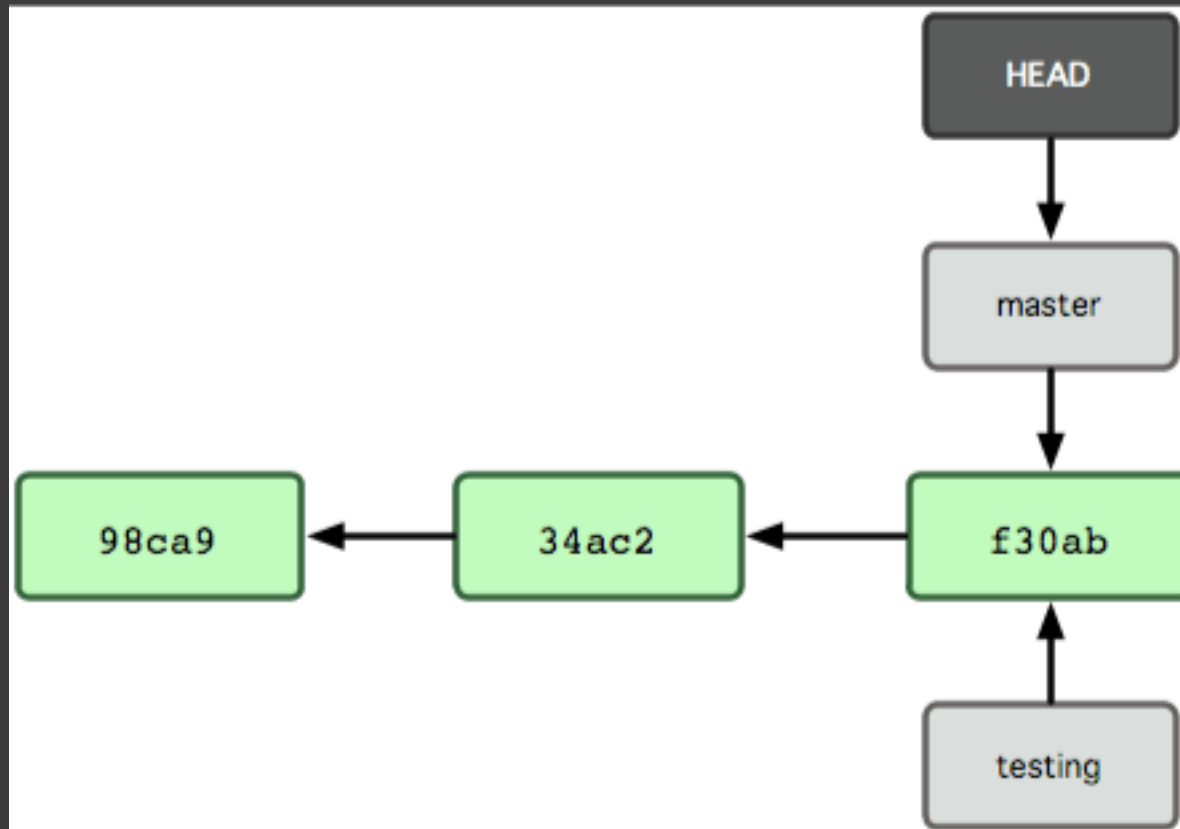
4. Control de versiones: Git

⦿ Utilizando Ramas

- Una rama es pues, un apuntador que apunta a una de esas confirmaciones.
- Creando una rama creamos un nuevo apuntador que podemos mover libremente.
- Por ejemplo, para no trabajar directamente en la rama master, creamos una nueva rama llamada testing:
 - `$ git branch testing`
- Nos creará un nuevo apuntador a la rama donde nos encontramos (luego ya nos moveremos)

4. Control de versiones: Git

● Utilizando Ramas



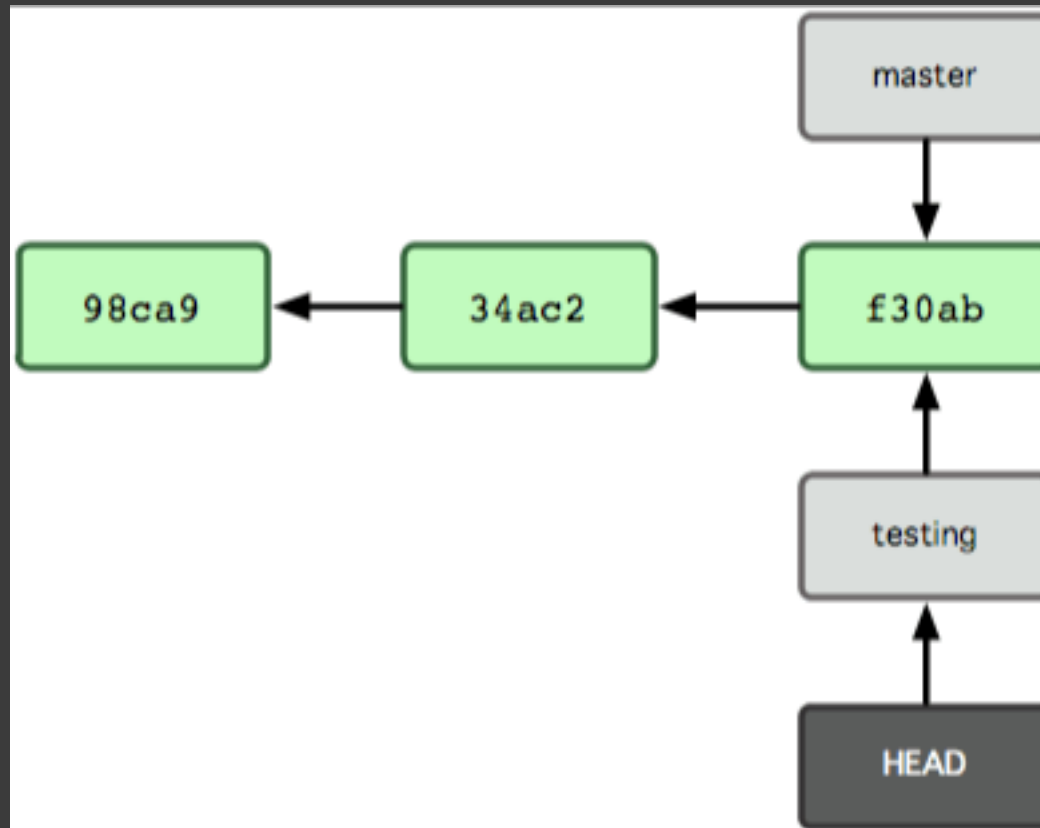
4. Control de versiones: Git

⦿ Utilizando Ramas

- Git usa un apuntador especial para saber en qué rama nos encontramos en cada momento. Ese apuntador se llama HEAD.
- HEAD apunta a la rama local en la que te encuentres en ese momento.
- Para saltar a otra rama utilizamos el comando checkout:
 - `$ git checkout testing`

4. Control de versiones: Git

● Utilizando Ramas



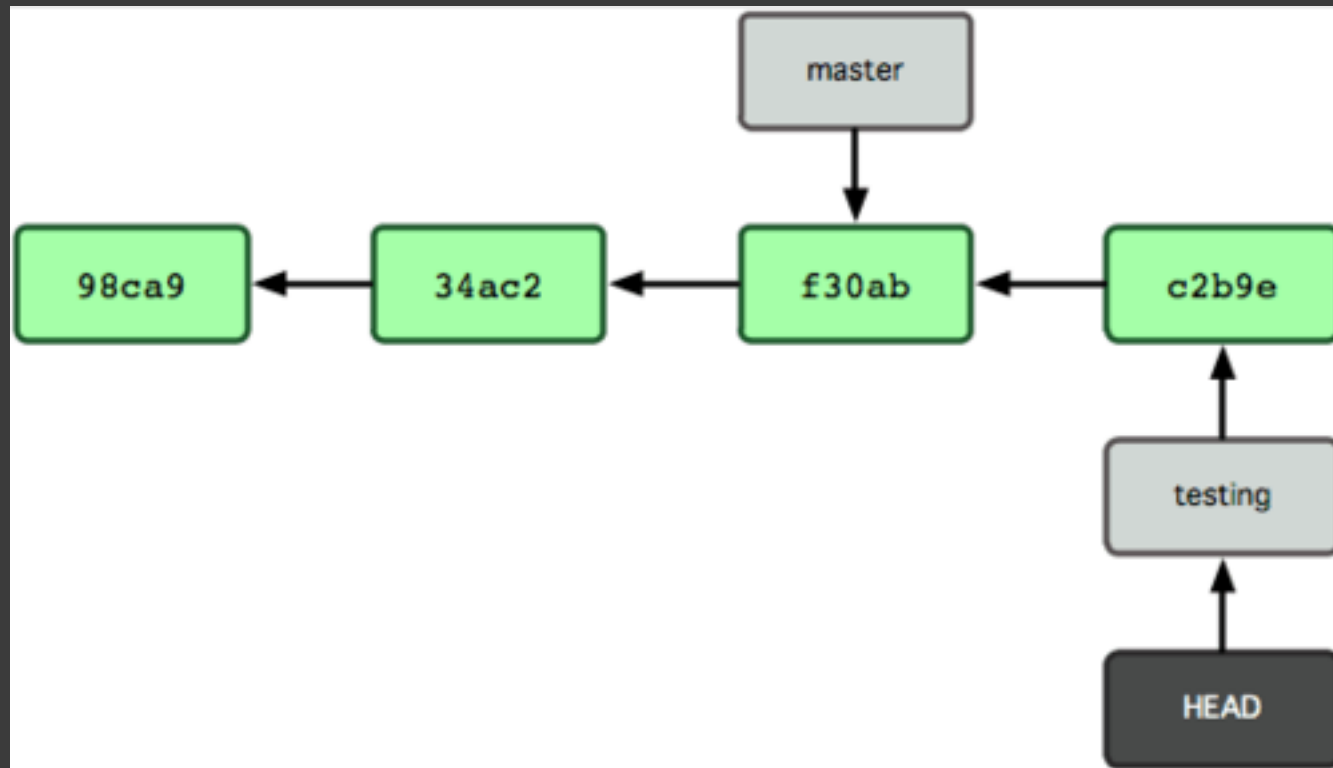
4. Control de versiones: Git

⦿ Utilizando Ramas

- Si ahora hacemos cambios y los subimos, pasará lo siguiente:
 - La rama testing avanza
 - La rama master permanece en la confirmación donde estaba cuando hicimos el checkout
 - Modificamos fichero
 - `git add fichero`
 - `git commit -m "he hecho un cambio"`

4. Control de versiones: Git

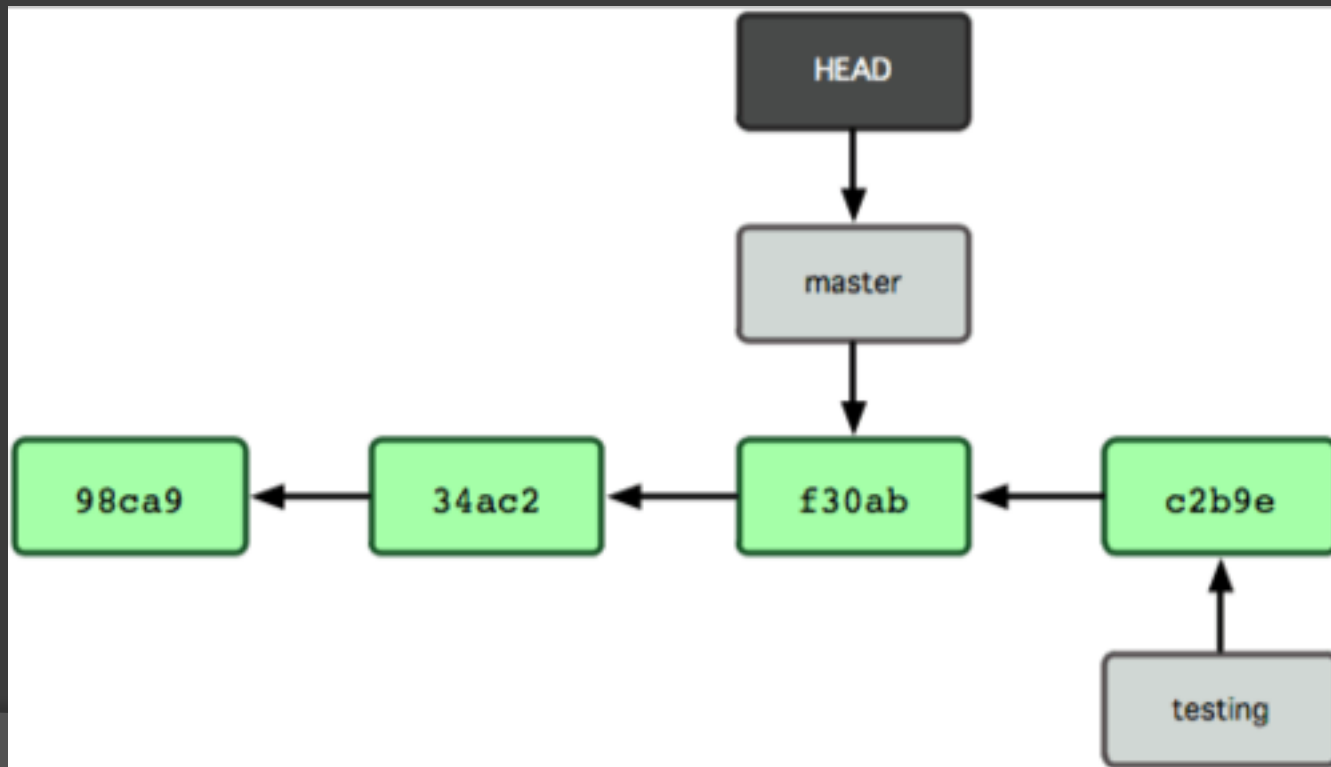
● Utilizando Ramas



4. Control de versiones: Git

● Utilizando Ramas

- Volvemos a la rama master con
 - \$ git checkout master



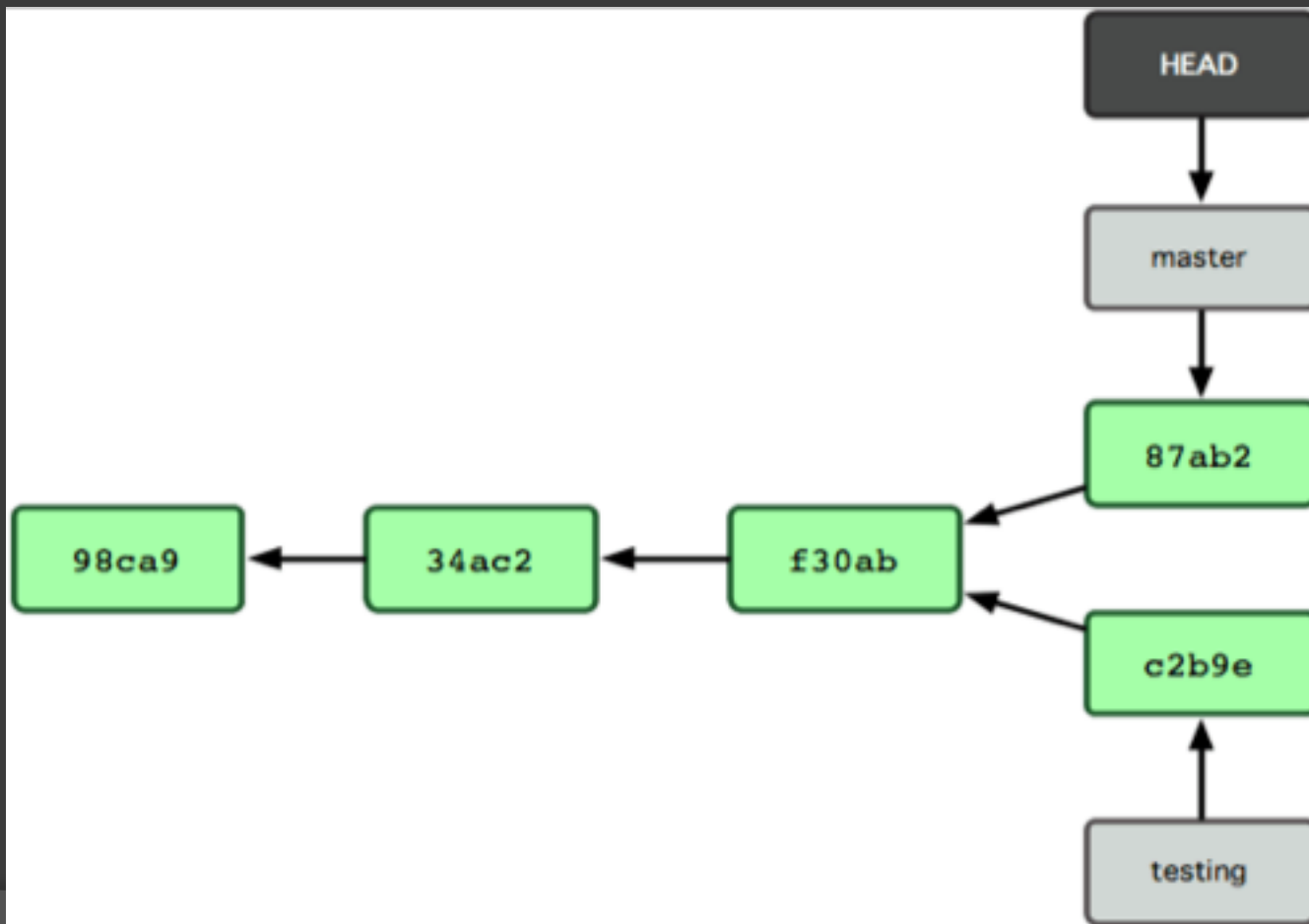
4. Control de versiones: Git

◉ Utilizando Ramas

- Al volver a la rama master hacemos dos cosas:
 - Cambiar el apuntador HEAD a la rama master.
 - Revertir los cambios de nuestro directorio de trabajo, dejándolos tal y como estaban en la última instantánea confirmada en dicha rama master.
 - Nos permite trabajar en otra dirección diferente.
- Si volvemos a hacer cambios en master, y los confirmamos, tenemos cambios aislados en ramas independientes entre las que podemos saltar.

4. Control de versiones: Git

● Utilizando Ramas



4. Control de versiones: Git

◉ Ramificar y fusionar

- Imagina que sigues los siguientes pasos:
 - Trabajas en un sitio web.
 - Creas una rama para un nuevo tema sobre el que quieres trabajar.
 - Realizas algo de trabajo en esa rama.
- En este momento, recibes una llamada avisándote de un problema crítico que has de resolver. Y sigues los siguientes pasos:
 - Vuelves a la rama de producción original.
 - Creas una nueva rama para el problema crítico y lo resuelves trabajando en ella.
 - Tras las pertinentes pruebas, fusionas (merge) esa rama y la envías (push) a la rama de producción.
 - Vuelves a la rama del tema en que andabas antes de la llamada y continuas tu trabajo.

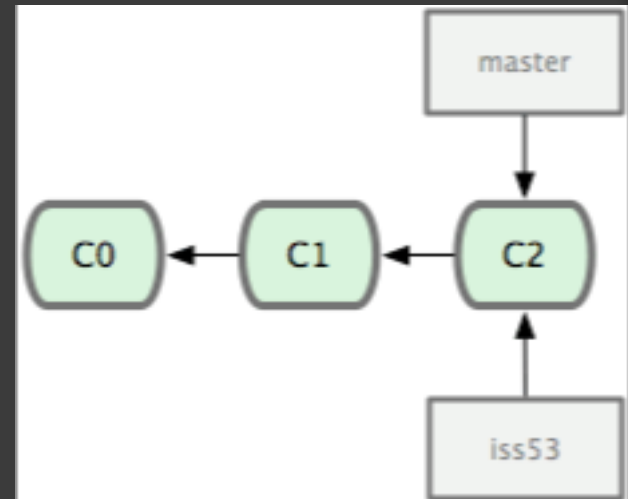
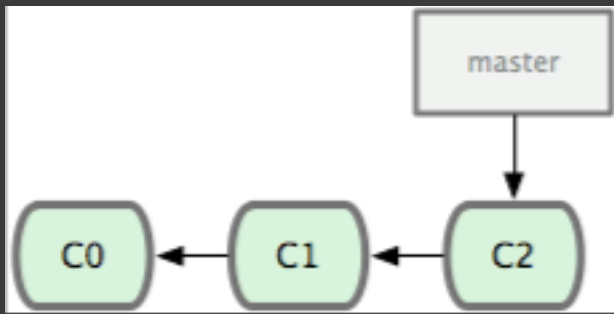
4. Control de versiones: Git

◉ Ramificar y fusionar

- Tenemos que trabajar en una nueva funcionalidad que es la número 53, así que crearemos una rama para esa funcionalidad:
 - `$ git branch iss53`
 - `$ git checkout iss53`
- Podríamos conseguir lo mismo con
 - `$ git checkout -b iss53`
- Así acabamos de crear y saltar a una nueva rama donde vamos a implementar esa nueva funcionalidad.

4. Control de versiones: Git

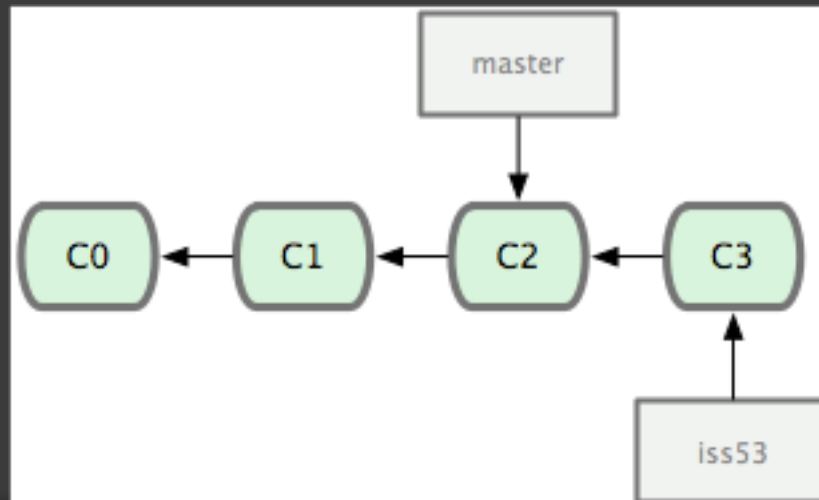
◉ Ramificar y fusionar



4. Control de versiones: Git

◉ Ramificar y fusionar

- Hacemos cambios para la creación de la nueva funcionalidad...
- Seguimos programando...
- Subimos los cambios...



4. Control de versiones: Git

◉ Ramificar y fusionar

- ¡¡¡¡URGENTE!!!!
- HAN DETECTADO UN PROBLEMA CON EL EMAIL Y HAY QUE SOLUCIONARLO...

• YA!!!

4. Control de versiones: Git

◉ Ramificar y fusionar

- Con Git no hay que mezclar el problema con la funcionalidad que tenemos a medio-implementar.
- Tampoco tenemos que quitar nuestros cambios para trabajar con lo que está en producción.
- Siempre y cuando tengamos los cambios confirmados, podremos saltar de rama.
- Por lo tanto:
 - `$git checkout master`

4. Control de versiones: Git

◉ Ramificar y fusionar

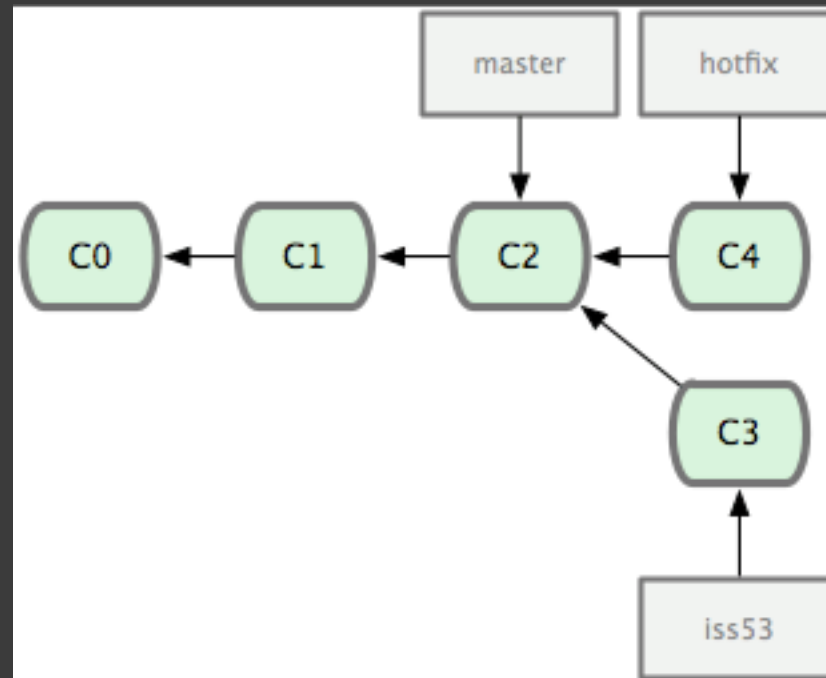
- Git revierte la carpeta de trabajo exactamente al estado en que estaba en la confirmación (commit) apuntada por la rama que activamos (checkout) en cada momento.
- Git añade, quita y modifica archivos automáticamente.

4. Control de versiones: Git

- ◉ Ramificar y fusionar
- ◉ Volvemos al problema urgente.
 - Creamos una rama nueva llamada hotfix sobre la que trabajaremos para resolver ese problema:
 - `$ git checkout -b hotfix`
 - Arreglamos el problema
 - `$ git add ficheros`
 - `$ git commit -m "solucion error email"`
- ◉ Y estamos trabajando con tres ramas:

4. Control de versiones: Git

◉ Ramificar y fusionar



4. Control de versiones: Git

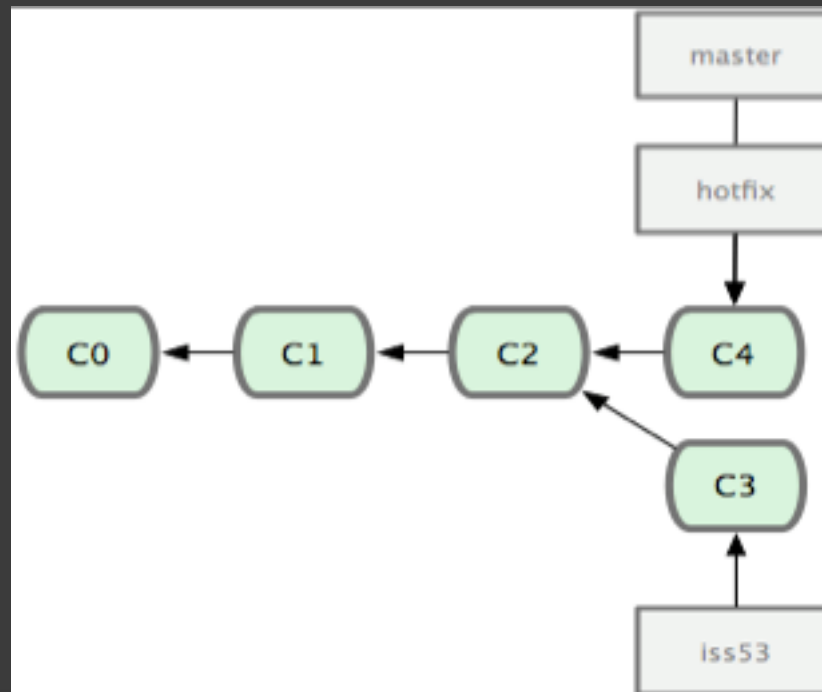
◉ Ramificar y fusionar

- Probaremos los cambios, nos aseguraremos de que todo funciona, y llegará el momento de añadir esos cambios a nuestra rama master, que es la que está en producción.
- Para eso, volvemos a la rama master:
 - `$ git checkout master`
 - `$ git merge hotfix`
 - “Fast forward” (avance rápido)
- Es un avance rápido porque hemos fusionado dos confirmaciones que eran contiguas, es decir, no había trabajos intermedios.

4. Control de versiones: Git

◉ Ramificar y fusionar

- Ahora los cambios están ya en la instantánea de la confirmación apuntada por la rama master:



4. Control de versiones: Git

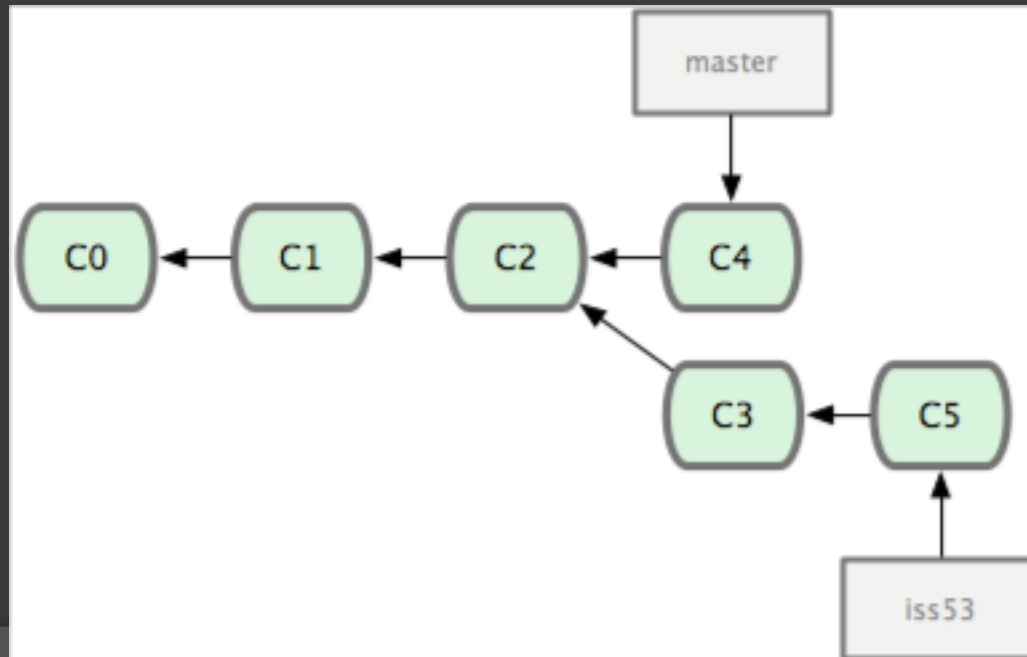
◉ Ramificar y fusionar

- Tras la fusión vemos que la rama master apunta al mismo sitio que la rama hotfix.
- Si ya hemos terminado con la rama hotfix, conviene borrarla:
 - `$ git branch -d hotfix`
 - La opción `-d` (delete) nos permite borrar una rama.
- Y una vez hecho esto, podemos volver al trabajo que estábamos realizando en la rama `iss53`
 - `$ git checkout iss53`

4. Control de versiones: Git

◉ Ramificar y fusionar

- Seguiremos trabajando en la rama `iss53`, confirmaremos nuestros cambios... y el esquema irá formándose de la siguiente manera:



4. Control de versiones: Git

◉ Ramificar y fusionar

- Ahora no tenemos los cambios de la rama master (problema del email solucionado) en nuestra rama iss53. Esto lo podemos resolver de dos maneras:
 - `$ git merge master` (fusionamos la rama master sobre nuestra rama iss53)
 - Esperamos hasta que decidamos llevar la rama iss53 a la rama master con los cambios finalizados.

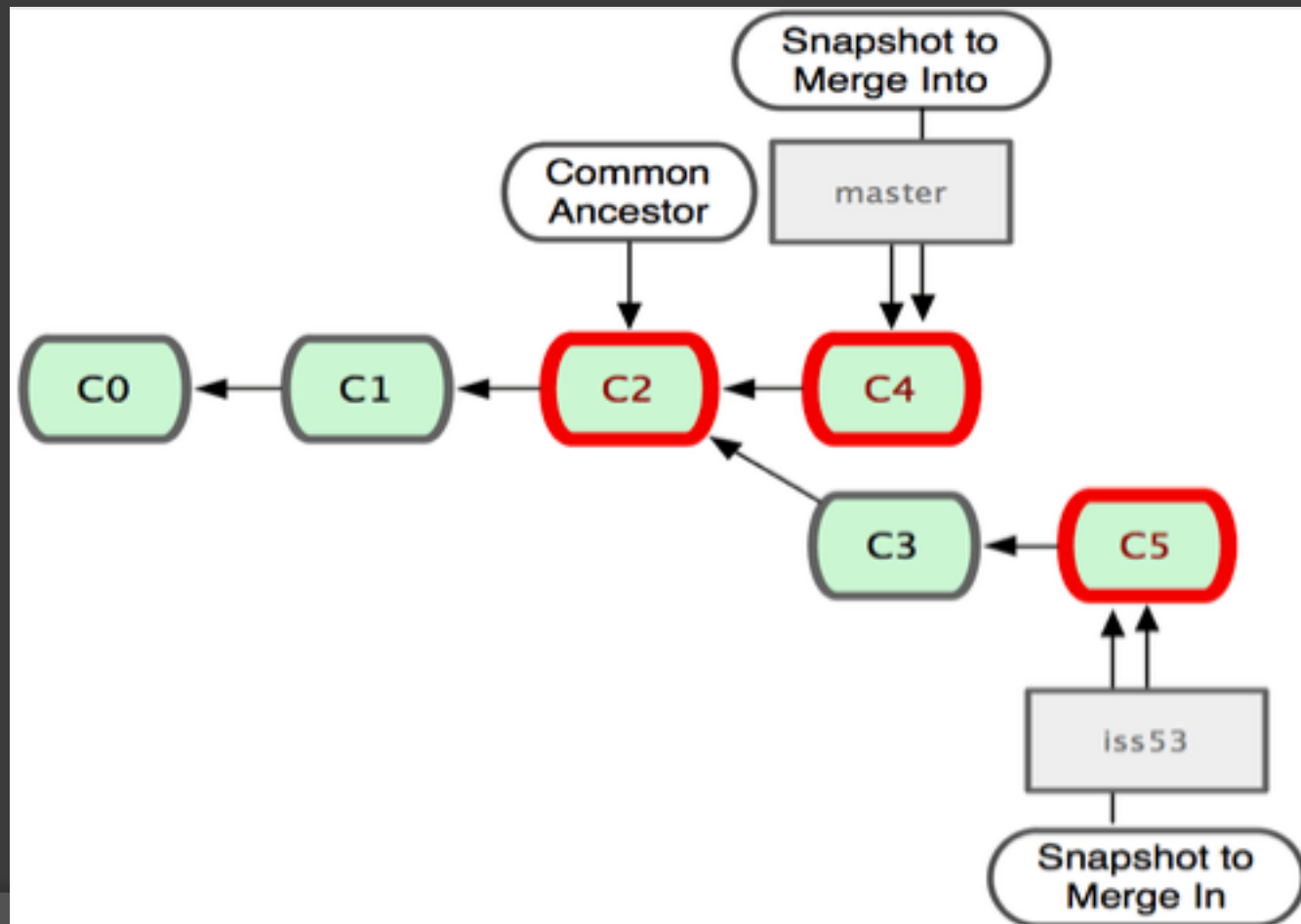
4. Control de versiones: Git

◉ Ramificar y fusionar

- Ya hemos terminado nuestro trabajo con el problema iss53.
- Lo queremos fusionar con la rama master:
 - `$ git checkout master`
 - `$ git merge iss53`
 - Merge made by recursive (ya no es un avance rápido)
- Git realizará una fusión a tres bandas, utilizando las dos instantáneas apuntadas por el extremo de cada una de las ramas y por el ancestro común a ambas dos.

4. Control de versiones: Git

◉ Ramificar y fusionar



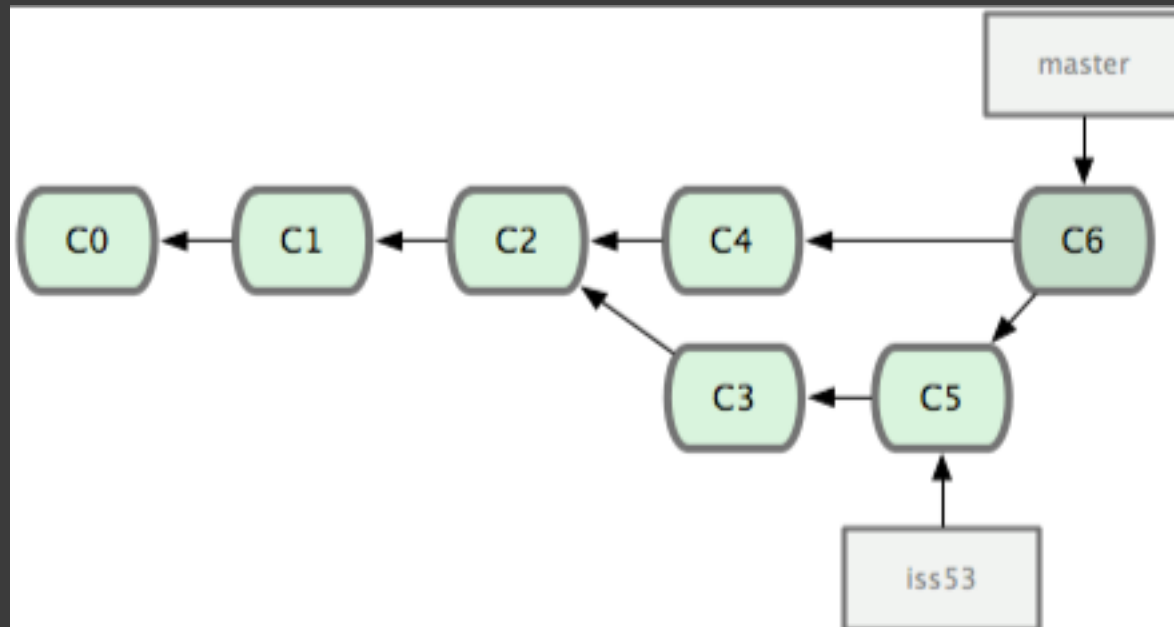
4. Control de versiones: Git

◉ Ramificar y fusionar

- Git crea una nueva instantánea (snapshot) resultante de la fusión a tres bandas; y crea automáticamente una nueva confirmación de cambios (commit) que apunta a ella.
- Nos referimos a este proceso como "fusión confirmada". Y se diferencia en que tiene más de un padre.
- Es el propio Git quien determina automáticamente el mejor ancestro común para realizar la fusión.
- Esto nos facilita mucho el trabajo de fusionar.

4. Control de versiones: Git

◉ Ramificar y fusionar



4. Control de versiones: Git

◉ Ramificar y fusionar

- Ahora que todo el trabajo está hecho, solamente nos quedaría eliminar la rama iss53:
 - `$ git branch -d iss53`

4. Control de versiones: Git

- ⦿ Pueden surgir problemas a la hora de fusionar.
 - La fusión no se completará y tendremos que resolver los conflictos de manera manual.
 - Con `$ git status` podemos ver qué ficheros tienen conflictos.
 - Git añade a los archivos conflictivos unos marcadores especiales de resolución de conflictos para ayudarnos a resolver la fusión.

4. Control de versiones: Git

- Pueden surgir problemas a la hora de fusionar.
 - El archivo conflictivo contendrá algo así:

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

- Dos bloques separados por =====
- Parte de arriba: Lo que hay en el HEAD
- Parte de abajo: Lo que hay en iss53

4. Control de versiones: Git

- ⦿ Pueden surgir problemas a la hora de fusionar.
 - Resolveremos los conflictos manualmente y tendremos que lanzar el `git add` para cada fichero modificado.
 - Así Git entenderá que el problema está resuelto.
 - En cualquier momento con `$ git status` veremos en qué estado se encuentran los conflictos existentes.

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones: Git

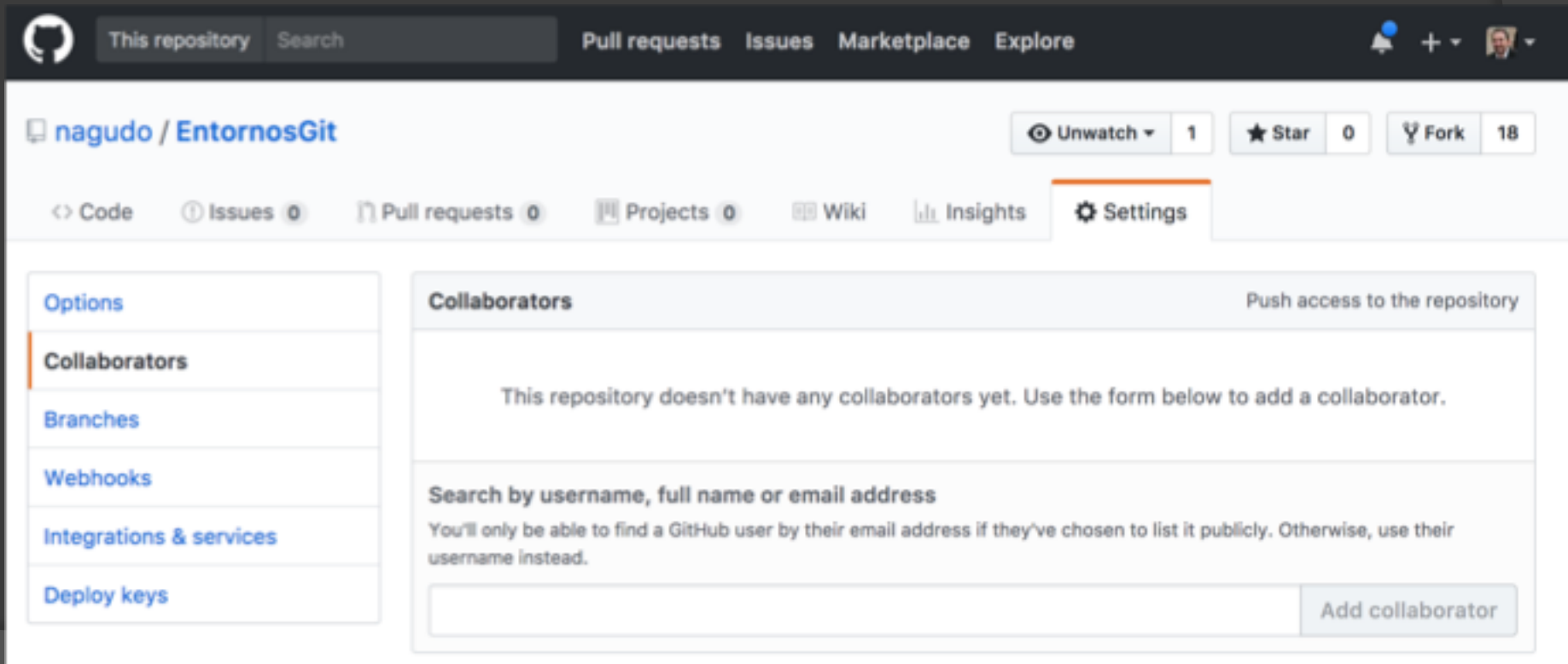
⦿ Colaborar en proyectos Git

- Además de colaborar con un proyecto agregándolo a nuestra cuenta con un Fork, podemos gestionar colaboradores en cada uno de nuestros repositorios para que participen en ellos.
- Esto se hace a través de invitaciones a los usuarios.

4. Control de versiones: Git

Colaborar en proyectos Git

- Tenemos que entrar en la configuración del repositorio, en el menú de colaboradores:



The screenshot shows the GitHub interface for the repository 'EntornosGit' by user 'nagudo'. The top navigation bar includes links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The repository's main navigation bar shows 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings' (which is highlighted). On the right, there are buttons for 'Unwatch', 'Star', and 'Fork'. The left sidebar contains a list of settings: 'Options', 'Collaborators' (selected), 'Branches', 'Webhooks', 'Integrations & services', and 'Deploy keys'. The 'Collaborators' section is titled 'Collaborators' and includes the subtitle 'Push access to the repository'. It contains a message: 'This repository doesn't have any collaborators yet. Use the form below to add a collaborator.' Below this is a search bar with the placeholder text 'Search by username, full name or email address' and a note: 'You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.' At the bottom right of the search bar is an 'Add collaborator' button.


4. Control de versiones: Git


Colaborar en proyectos Git


- Podemos buscar por nombre completo, nombre de usuario o dirección de email:


Search by username, full name or email address

You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

 nAchoA

 nachoalbores

 nachoalvarezdeleon


 nachoavalont

4. Control de versiones: Git

Colaborar en proyectos Git

- Cuando pulsamos en añadir colaborador, se envía una invitación a la persona en cuestión y se añade a la lista, a la espera de su aceptación:

CollaboratorsPush access to the repository

 Awaiting NachoAS's response

Copy invite link ▼Cancel invite

Search by username, full name or email address

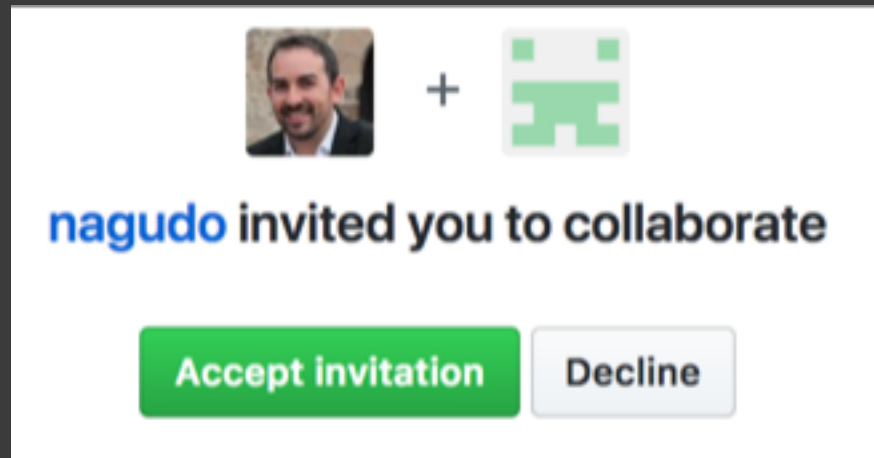
You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

Add collaborator

4. Control de versiones: Git

Colaborar en proyectos Git

- Cuando el colaborador abra la invitación y pulse en el enlace, le aparecerá algo similar a la siguiente imagen:




4. Control de versiones: Git

Colaborar en proyectos Git

- Y en el momento en el que acepte, al usuario propietario del repositorio le aparecerá confirmado en la lista de colaboradores:

Collaborators

Push access to the repository

 NachoAS ×

Search by username, full name or email address
You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

Add collaborator

4. Control de versiones

- ⦿ Qué es un SCV
- ⦿ Tipos de SCV
- ⦿ Git
 - Información
 - Instalación y configuración inicial
 - Cómo tener un proyecto Git
 - Trabajando con Git
 - Ramificar y fusionar
 - Colaboración en proyectos

4. Control de versiones: Git

● Práctica por grupos:

- Crear un nuevo repositorio con un proyecto real (calculadora) sin funcionalidades.
- Añadir como colaboradores a cada uno de los miembros, y también al profesor (usuario nagudo).
- Crear una funcionalidad cada uno de los miembros del grupo.
- CADA UNO SE CREA UNA RAMA PARA SU FUNCIONALIDAD.
- Cuando completemos nuestra funcionalidad, se fusionan las ramas (DESCARGAR SI HAY CAMBIOS ANTES DE SUBIR LOS NUESTROS)
- Probamos que todo funciona después de subir.
- DOCUMENTAR TODO.