



Capgemini  
CONSULTING. TECHNOLOGY. OUTSOURCING



# Píldora de Git - UV (GIM)

---

Cristóbal Belda Pérez

# Índice

- ¿Qué es Git y qué es GitHub?
- Pasos previos
- Nomenclatura y conceptos básicos
- Flujo de trabajo
- Trabajar con GitHub
- Caso práctico
- Extras (usos, alternativas, ...)
- Links de interés

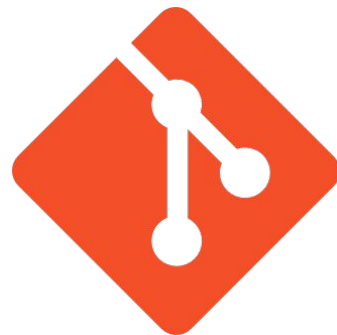
¿Git & GitHub?

---

# ¿Qué es Git y qué es GitHub?

**Git** es un *software de control de versiones* (SCM) diseñado pensando en la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

*Grosso modo*, podemos decir que nos permite conservar nuestro código fuente de forma segura, así como agilizar y clarificar nuestra forma de trabajar en él. Esto se acentúa cuando, además, trabajamos en un equipo.



# ¿Qué es Git y qué es GitHub?

**GitHub** es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones (SCM) Git. Brinda herramientas muy útiles para el trabajo en equipo dentro un proyecto.



# Pasos previos

---

# Pasos previos

Instalación de herramientas:

## Git

<https://git-scm.com/download/>



## VSCode (opcional)

<https://code.visualstudio.com/download>



# Pasos previos

Creación de cuentas:

**GitHub**

<https://github.com/>





# Nomenclatura y Conceptos

---

# Nomenclatura y Conceptos

## Comandos de consola

Usaremos el cliente de consola Git para realizar las operaciones y los comandos necesarios.

Una vez instalado el cliente (las distros de Linux lo suelen llevar integrado), puedes usar las distintas opciones (actualizar cambios, copiar repositorio en tu máquina, ir a una versión del repositorio, ...) escribiendo en consola:

```
$ git [option]
```

# Nomenclatura y Conceptos

## **Repositorio (*repo*)**

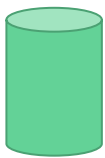
Un *repositorio* es la unidad básica. La forma más fácil de imaginarlo es como la carpeta donde se encuentra nuestro proyecto.

En él tendremos nuestro código, podremos hacer una copia local (*repositorio local*) desde la cual podremos actualizar y subir cambios al *repositorio remoto*, y un largo *etc.*

# Nomenclatura y Conceptos

## Repositorio (*repo*)

*remote*



`github.com/cristobal/mi-proyecto`



`local/directory/mi-proyecto`

*local*

# Nomenclatura y Conceptos

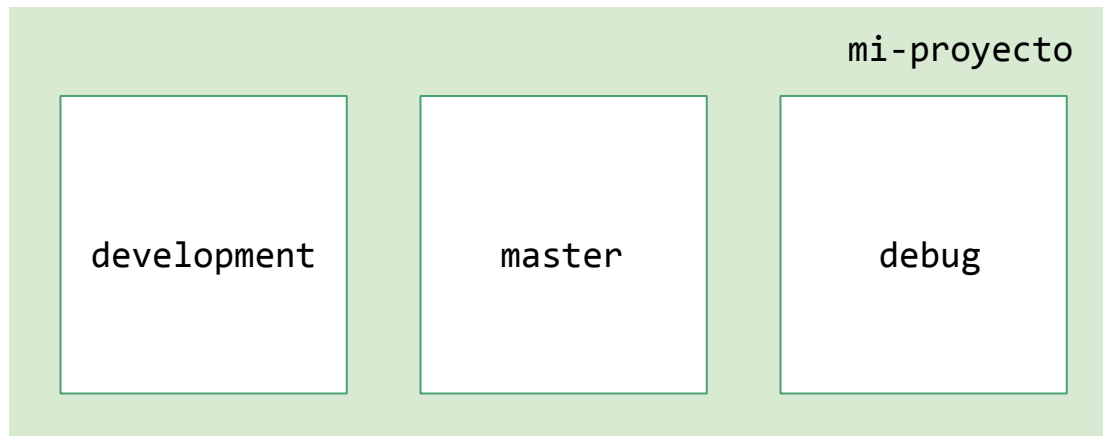
## Rama (*branch*)

Una rama es una versión paralela del repositorio. Por defecto, la versión principal del repositorio se aloja en la rama *master*. En ella “debería” ofrecerse una versión estable del proyecto. Las distintas ramas almacenan también en el repositorio sin afectar las unas a las otras.

Un ejemplo de uso sería la resolución de un *bug* del proyecto o utilizar una rama para el desarrollo y dejar la *master* para sacar versiones nuevas y estables.

# Nomenclatura y Conceptos

## Rama (*branch*)



# Nomenclatura y Conceptos

## Rama (*branch*)



# Nomenclatura y Conceptos

## Clone

Cuando nos clonamos un repositorio, significa que estamos “copiando” un *repositorio remoto* en nuestra máquina, obteniendo así un *repositorio local* que siempre estará apuntando al remoto.

```
$ git clone https://github.com/cristobal/mi-proyecto
```

Para ver dónde apunta nuestro *repositorio local* podemos ejecutar:

```
$ git remote show origin
```



# Nomenclatura y Conceptos

## Commit

Entendemos *commit* como un conjunto de cambios que se hacen al repositorio.

Imaginemos que modificamos una línea de código de un archivo de nuestro repositorio. La forma de actualizarlo es hacer un *commit* de ese cambio.

Antes de hacerlo, hemos de ver qué cambios han surgido en nuestro *repo local* comparándolo con el *repo remoto*. Después añadimos los cambios y finalmente hacemos el *commit* (empaquetamos los cambios).

# Nomenclatura y Conceptos

## Commit

Comprobar si hay diferencias entre *local* y *remoto*:

```
$ git status
```

```
λ git status
On branch ocp-3.6
Your branch is up-to-date with 'origin/ocp-3.6'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   cicd-template.yaml

no changes added to commit (use "git add" and/or "git commit -a")
```

# Nomenclatura y Conceptos

## Commit

Añadir (todos) mis cambios para ser *commiteados* y chequear:

```
$ git add .
```

```
$ git status
```

```
λ git status
On branch ocp-3.6
Your branch is up-to-date with 'origin/ocp-3.6'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   cicd-template.yaml
```

# Nomenclatura y Conceptos

## Commit

Finalmente  
empaquetamos los  
cambios añadidos en  
un *commit*.

```
$ git commit -m  
"info sobre los  
cambios"
```

```
λ git commit -m "actualizacion de mis cambios locales"  
[ocp-3.6 4a79020] actualizacion de mis cambios locales  
Committer: Belda Perez <crisobal.belda-perez@capgemini.com>  
Your name and email address were configured automatically based  
on your username and hostname. Please check that they are accurate.  
You can suppress this message by setting them explicitly. Run the  
following command and follow the instructions in your editor to edit  
your configuration file:
```

```
git config --global --edit
```

After doing this, you may fix the identity used for this commit with:

```
git commit --amend --reset-author
```

```
1 file changed, 1 insertion(+), 12 deletions(-)
```

# Nomenclatura y Conceptos

## Push

Cuando hacemos un *push* significa que estamos “empujando” el paquete de cambios (el *commit*) al repositorio remoto.

En el caso de que estemos en la rama *ejemplo* del *repo local*, los cambios serán actualizados en la rama *ejemplo* del *repo remoto*.

```
$ git push
```

# Nomenclatura y Conceptos

## Pull

Para actualizar tu *repo local* al *commit* más nuevo. Se ejecuta el *pull* en el directorio de trabajo para “bajar y fusionar” los cambios remotos.

Si el repositorio local está actualizado, la consola mostrará un

“Already up-to-date”

```
$ git pull
```

# Nomenclatura y Conceptos

## Push / Pull

Se puede dar el caso en el que existan **conflictos**. Esto ocurre cuando se intenta hacer un *push* a un repo que no tenemos actualizado en local. Para ello, deberíamos hacer un *pull* del cambio que no tenemos en local, generar un *commit* de nuevo y hacer un *push* de él (habiendo solucionado los conflictos).

# Nomenclatura y Conceptos

## Push / Pull

`$ git push`



**Caso 1:** Todo ha ido bien. El repo se actualiza.

**Caso 2:**  $\nexists$  conflictos. Pero otrx compañerx ha hecho *push* antes que tú. Esto implica:

1. `$ git pull`
2. `$ git commit -m "merge with latest changes"`
3. `$ git push`

**Caso 3:**  $\exists$  conflictos. El código que pretendes “empujar” el repo entra en contradicción con el del último *commit*.

1. `$ git pull`
2. `-- resolver conflictos (visibles en tu IDE) --`
3. `$ git add .`
4. `$ git commit -m "solved merge conflicts"`
5. `$ git push`



# Nomenclatura y Conceptos

## Fork

Un fork es una copia personal en tu cuenta del repositorio de otro usuario. Los forks nos permiten hacer cambios libremente de un proyecto sin afectar el original.

Esta copia permanecerá adjunta al original permitiendo remitir los cambios a éste mediante un mecanismo llamado *pull-request*.

También es posible mantener tu fork *up-to-date* actualizándose con los últimos cambios del original.

# Nomenclatura y Conceptos

## **Pull-request**

Un pull request es un conjunto de cambios propuestos a un repositorio por un usuario y aceptado o rechazado por otro usuario colaborador de ese repositorio.

Cada pull request tiene su propio hilo de discusión.

# Nomenclatura y Conceptos

## Merge

Hacer un *merge* implica compilar los cambios de una *branch* (en el mismo repositorio o en un fork) y aplicarlos en otro.

Suele suceder normalmente cuando se hace un *pull request* (los cambios propuestos del *fork* se “mergean” en el repositorio original).

# Nomenclatura y Conceptos

## Init

Este comando crea un repositorio Git vacío, lo que se traduce en un directorio `./git` con sus subdirectorios y archivos de plantillas.

```
$ git init
```

# Nomenclatura y Conceptos

## **.gitignore**

Podría darse el caso de que no quisiéramos subir todos nuestros archivos locales al repositorio remoto. Por ejemplo, si se trata de algo que **nunca** se va a ver modificado, o una carpeta con dependencias instaladas del proyecto, etc.

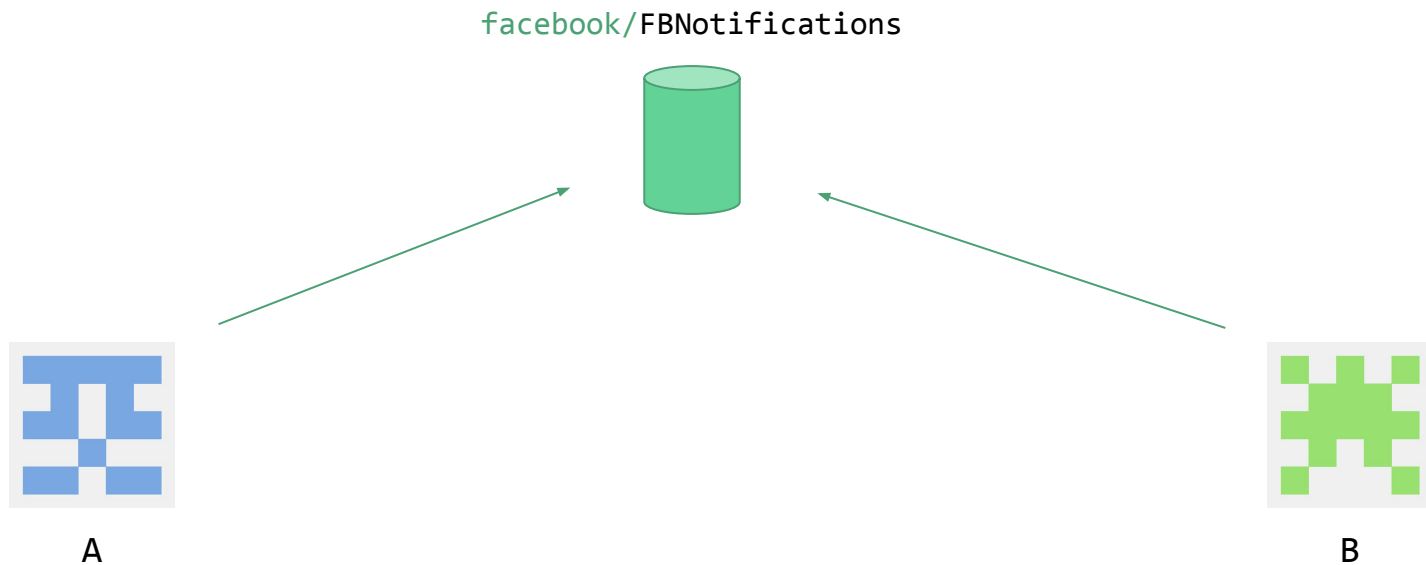
El archivo donde reflejamos esto se llama **.gitignore** (sí, con un '.' delante) y, generalmente se añade en el directorio root del repositorio. [Aquí](#) tenéis ejemplos de este archivo según el lenguaje en el que esté implementado vuestro proyecto/repositorio.

# Flujo de trabajo

---

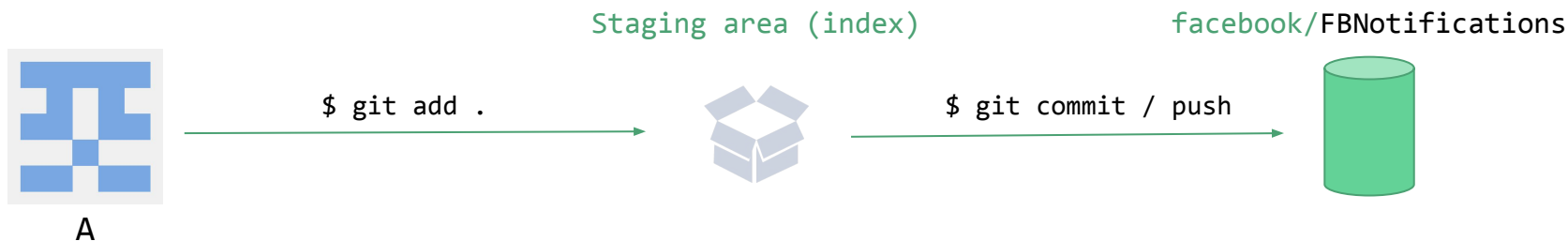
# Flujo de trabajo: Ejemplo

2 desarrolladorxs (A y B) están colaborando en un proyecto open-source de Facebook llamado *FBNotifications*.



# Flujo de trabajo: Ejemplo

**A** está segurísimx de que sus últimos cambios en local son correctos por lo que procede a añadirlos a un *commit* y hacer un *push* directamente al repositorio.





# Flujo de trabajo: Ejemplo

Al pasar los tests del *commit* del repositorio, el manager de lxs desarrolladores descubre que hay **un bug causado por un bucle infinito** en el código de A, por lo que le pide a B que lo solucione.



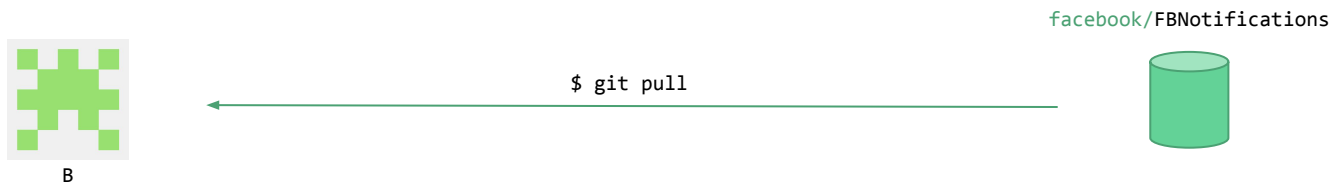
## Flujo de trabajo: Ejemplo

Al pasar los tests del *commit* del repositorio, el manager de lxs desarrolladores descubre que hay un **bug causado por un bucle infinito** en el código de A, por lo que le pide a B que lo solucione.

# ¿Qué haría B?



# Flujo de trabajo: “Solución” de B



-- Se deshace el bucle infinito --



# Flujo de trabajo: Solución (posterior) del Manager

Se instala en el equipo una forma nueva de trabajar:

**Siempre y cuando se deba hacer cambios en el repositorio, se hará mediante *pull request*.**

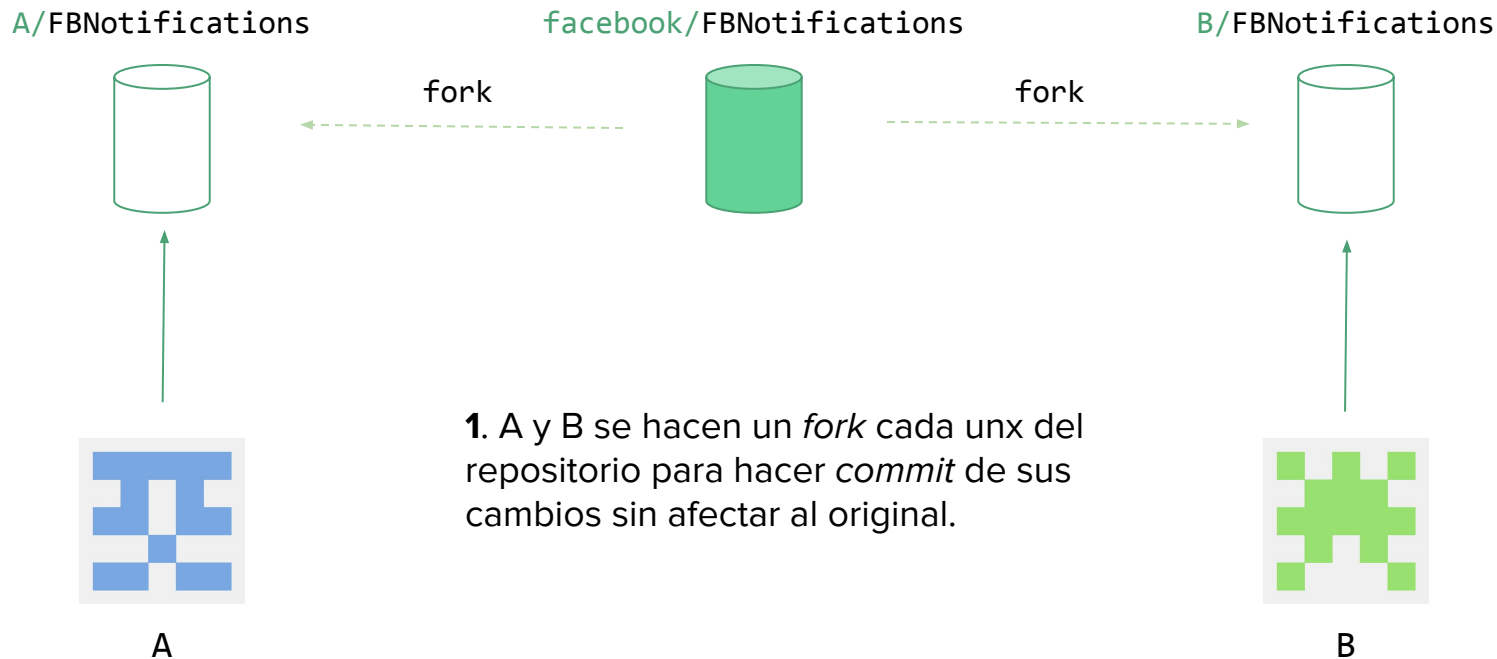
Flujo de trabajo: Solución (posterior) del Manager

Se instala en el equipo una forma nueva de trabajar:

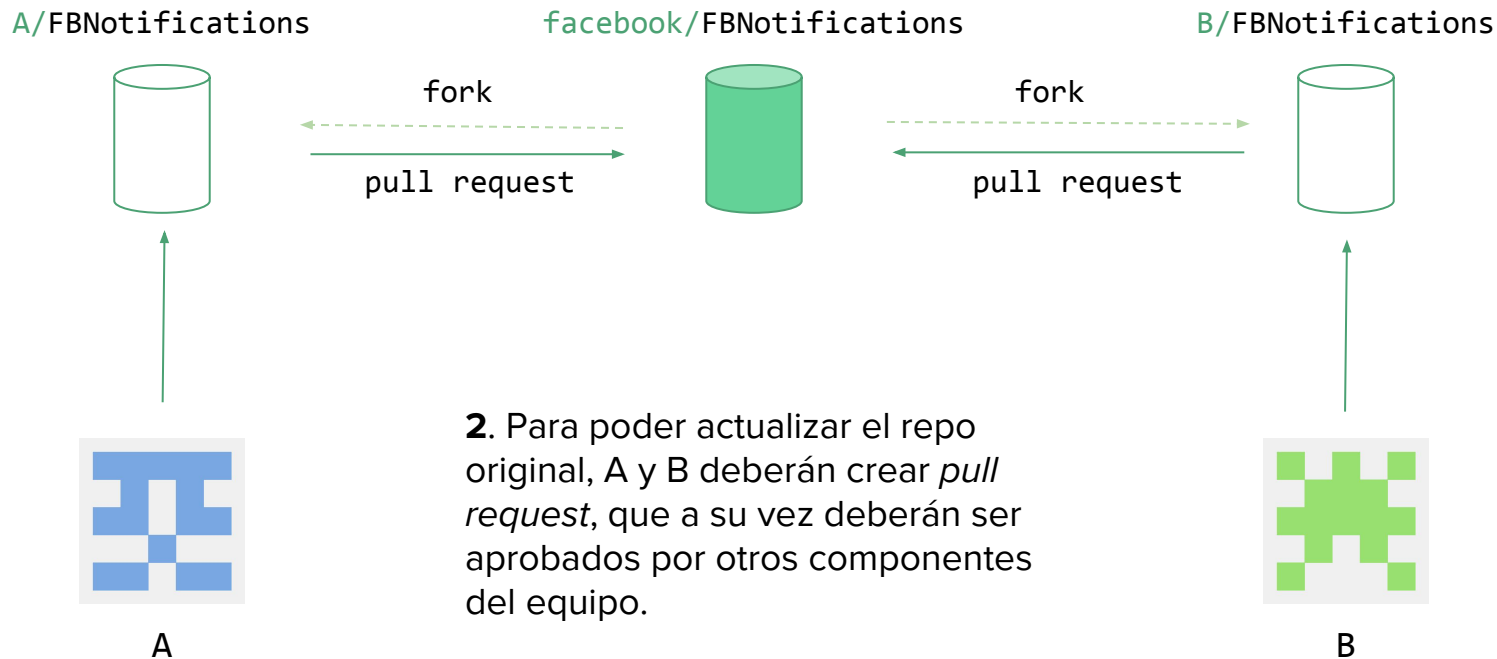
Siempre y cuando se deba hacer cambios en el repositorio, se hará mediante *pull request*.

¿Qué deberán  
hacer A y B?

# Flujo de trabajo: Solución (posterior) del Manager



# Flujo de trabajo: Solución (posterior) del Manager



# Trabajar con GitHub

---



# GitHub: Sistemas de documentación

## **README.md**

Este archivo es “renderizado” por el propio GitHub. Utiliza el lenguaje *markdown* (.md) de escritura. En estos archivos se suele encontrar información sobre el repositorio, forma de instalar y ejecutar el proyecto una vez clonado, problemas encontrados, etc.

# GitHub: Sistemas de documentación

## README.md

[Code](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Settings](#) [Insights](#)


Devonfw newsletter [Edit](#)

[Add topics](#)


35 commits 1 branch 0 releases 4 contributors

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find file](#) [Clone or download](#)


This branch is 39 commits behind devonfw:master. [Pull request](#) [Compare](#)

 **cbeldacap**


 very few typo correction Latest commit f711fee on 9 Mar

 2017


 very few typo correction 7 months ago

 \_layouts


 Fixed broken links 7 months ago

 img


 This should be it 7 months ago

 .gitignore


 Added Colophon & .gitignore file 7 months ago

 LICENSE

 First commit 7 months ago

 README.md


 First test with Jekhyl theme 7 months ago

 \_config.yml

 First test with Jekhyl theme 7 months ago

README.md

## we-dev-on



This is the Devonfw newsletter. **we-dev-on** is a reader-supported publication dedicated to producing the best coverage of project info, opinions, news, events, tips & tricks from within the devonfw & OASP communities all over Capgemini APPS2.

# GitHub: Sistemas de documentación

## Wiki del repositorio

La *wiki* de un repositorio la utilizamos para desarrollar una documentación mucho más amplia sobre el proyecto. En ella podríamos incluso añadir desde la planificación y el diseño hasta una demostración gráfica de pruebas funcionales, un resultado de carga de peticiones, etc.

Puede contener documentos tanto en *markdown* como en *asciidoc* (utilizados para el mismo fin).

# GitHub: Incidencias

Las incidencias (o comúnmente denominadas *issues*) pueden ser resultado de quejas, problemas surgidos, propuestas de mejora o adaptación, etc.

Dentro del repositorio de GitHub podemos encontrar una pestaña superior en la que pone “Issues”. Es allí donde podemos abrirlos, cerrarlos, discutir sobre ellos en el mismo hilo de la incidencia, añadir *flags* indicando el tipo de incidencia, hacer referencia a ellos cuando hacemos un commit (podemos nombrarlos en el mensaje del *commit* añadiendo el número de *issue* con:

```
$ git commit -m “issue #345 solved”
```

), etc.

# Caso práctico

---

# Caso práctico

Equipos en **grupos de 3 desarrolladorxs** (A, B y C).



A



B



C

# Caso práctico

Equipos en **grupos de 3 desarrolladorxs** (A, B y C).

1. **A** crea un repositorio en GitHub (por ejemplo, “*pildora-git*”)
2. **A** añade como *collaborators* a **B** y **C**
3. **Todxs** hacen un *clone* del repositorio remoto (`$ git clone ...`)
4. **B** crea un archivo `index.html` en el repositorio y hace `add/commit/push` al repositorio remoto usando el contenido del archivo alojado en github:  
<https://raw.githubusercontent.com/cbeldacap/pildora-git/master/ejemplo/index.html> ) y hace un `push`.
5. **C** edita la línea 5 del archivo (`<h1>`) cambiando el contenido.
6. **A** edita la misma línea con contenido distinto al de **C**.

# Caso práctico

Equipos en **grupos de 3 desarrolladorxs** (A, B y C).

7. **A** hace `git add . / git commit -m "..."/ git push` al repo.
8. **C** hace `add/commit/push` al repo y encuentra que otro usuario (**A**) ha actualizado el conflicto en el push.
9. Cuando **C** hace un pull encuentra que hay conflicto en una de las líneas de código (Ayuda). ¿Qué ocurre en nuestro código en estos casos?

```
<<<<<<< HEAD
--Tu commit--
=====
--Último commit del repo--
>>>>>>> bf454eff1b2ea242ea0570389bc75c1ade6b7fa0
```



# Caso práctico

Equipos en **grupos de 3 desarrolladorxs** (A, B y C).

10. **C** hace `add/commit/push` de nuevo generando una versión definitiva.
11. **C** crea la nueva rama “`add-button`”. (`$ git checkout -b add-button`)
12. **A** y **B** hacen `pull` de los últimos cambios y, después de hacer `check-out` a la nueva rama, **B** añade un botón al `<body>` del archivo y sube los cambios al *repo*.
13. **C** hace `pull` de los cambios y en la misma rama `add-button` añade un segundo botón por debajo del que había añadido **B**.

# Caso práctico

Equipos en **grupos de 3 desarrolladorxs** (A, B y C).

14. Finalmente, **A** (después de tener totalmente actualizado su repositorio local), hace un `pull request` desde la rama `add-button` a la rama `master` (éste deberá ser aprobado). Ahora las dos ramas del repositorio deberían ser iguales.
15. **Todxs** hacen un `pull` con los últimos cambios.
16. **B** crea un issue en GitHub señalando que, después de todo el lío, el cliente ha decidido que sólo quiere un botón. Asigna como “encargadx” a **C**.
17. **C** elimina el último botón creado por **B** y hace un `commit` (y un `push`) nombrando el issue que está solucionando: `$ git commit -m “issue #1 solved”`

# Caso práctico

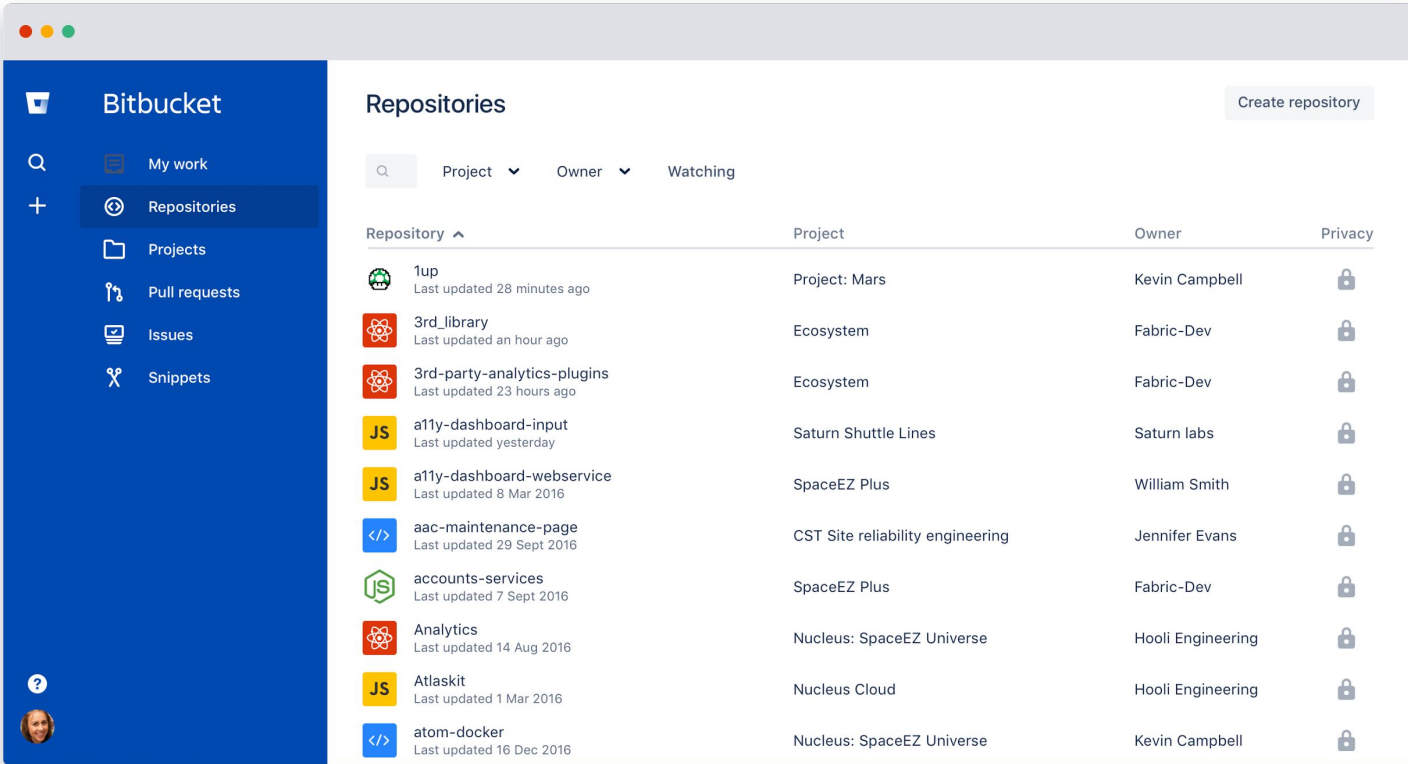
Equipos en **grupos de 3 desarrolladorxs** (A, B y C).

18. **A** y **B** hacen un fork del repositorio (hacerlo desde GitHub).
19. Probar a hacer cambios (da igual el *collaborator*) en un fork y crear un `pull request` desde GitHub.
20. Aceptar el `pull request` (da igual el *collaborator*).

Extras (usos, alternativas, ...)

---

# Extras: Alternativas a GitHub (BitBucket)



The screenshot displays the Bitbucket web interface. On the left is a blue sidebar with navigation links: 'My work', 'Repositories' (selected), 'Projects', 'Pull requests', 'Issues', and 'Snippets'. The main area is titled 'Repositories' and features a search bar and filters for 'Project', 'Owner', and 'Watching'. A 'Create repository' button is in the top right. Below is a table of repositories with columns for Repository, Project, Owner, and Privacy.

Repository	Project	Owner	Privacy
1up Last updated 28 minutes ago	Project: Mars	Kevin Campbell	🔒
3rd_library Last updated an hour ago	Ecosystem	Fabric-Dev	🔒
3rd-party-analytics-plugins Last updated 23 hours ago	Ecosystem	Fabric-Dev	🔒
a11y-dashboard-input Last updated yesterday	Saturn Shuttle Lines	Saturn labs	🔒
a11y-dashboard-webservice Last updated 8 Mar 2016	SpaceEZ Plus	William Smith	🔒
aac-maintenance-page Last updated 29 Sept 2016	CST Site reliability engineering	Jennifer Evans	🔒
accounts-services Last updated 7 Sept 2016	SpaceEZ Plus	Fabric-Dev	🔒
Analytics Last updated 14 Aug 2016	Nucleus: SpaceEZ Universe	Hooli Engineering	🔒
Atlaskit Last updated 1 Mar 2016	Nucleus Cloud	Hooli Engineering	🔒
atom-docker Last updated 16 Dec 2016	Nucleus: SpaceEZ Universe	Kevin Campbell	🔒

# Extras: Alternativas a GitHub (GitLab)

The screenshot displays the GitLab web interface. On the left is a dark sidebar with the GitLab logo and navigation links: Go to dashboard, Project, Activity, Builds (0), Milestones, Issues (0), Merge Requests (0), Members, Labels, Wiki, Forks, Settings, and Profile stivesso. The main content area shows the user 'Steve ESSO' and the project 'newfeature'. A blue banner at the top states 'Project 'newfeature' was successfully created.' Below this is a large light blue circle with the letter 'N'. The project name 'newfeature' is followed by a lock icon and the description 'New feature Module'. A 'Star' button shows 0 stars. Below the star button is a text input field for SSH keys with a dropdown menu showing 'git@' and 'mygitsystem :stivesso/newfeature.g'. To the right of the input field are icons for cloning the repository. The main content area also contains the text 'The repository for this project is empty' and instructions on how to push files using command line instructions. At the bottom, there is a section titled 'Command line instructions'.

GitLab

Steve ESSO / newfeature

Search

Project 'newfeature' was successfully created.

Project

Activity

Builds 0

Milestones

Issues 0

Merge Requests 0

Members

Labels

Wiki

Forks

Settings

Profile stivesso

N

newfeature

New feature Module

Star 0

SSH git@ mygitsystem :stivesso/newfeature.g


The repository for this project is empty


If you already have files you can push them using command line instructions below.

Otherwise you can start with [adding README](#) file to this project.

Command line instructions



# Extras: Alternativas a GitHub (Gogs)

[Dashboard](#) [Issues](#) [Pull Requests](#) [Explore](#)



## Gogs

A painless self-hosted Git service.


 Boston, MA  <https://gogs.io>

[New Repository](#)

[gogs](#)


Gogs is a painless self-hosted Git service.

Updated 8 hours ago

★ 12  7


[git-module](#)

Updated 5 months ago

★ 1  0

[go-gogs-client](#)

Updated 6 months ago

★ 1  0

People

1 >



[Invite Someone](#)

Teams

2 >

Owners

1 members · 3 repositories

member

0 members · 0 repositories

[Create New Team](#)

© 2017 Gogs Version: 0.9.160.0219 Page: 13ms Template: 2ms

    English [Website](#) Go 1.7.5

# Extras: Alternativas a GitHub (Gerrit)

[All](#) | [My](#) | [Admin](#) | [Documentation](#) | [Open](#) | [Merged](#) | [Abandoned](#)

Roman Birg <romanbirg@gmail.com> | [Settings](#) | [Sign Out](#)

status: open [Search](#)

☆ Change I2d15c15b: Gerrit tutorial String

Change-Id: I2d15c15b07735dbf2d059d7a2e3eb21f330f2e9e

Owner: [Roman Birg](#)

Project: [AOKP/packages\\_apps\\_ROMControl](#)

Branch: [ics](#)

Topic:

Uploaded: Apr 7, 2012 10:55 AM

Updated: Apr 7, 2012 10:55 AM

Status: Review in Progress

Gerrit tutorial String

Notice in the first line it is just a breif description.  
Below you can make specific notes on your commit that will show up in gerrit.

Signed-off-by: Roman Birg <romanbirg@gmail.com>

[Permalink](#)

- Need Verified
- Need Code-Review

[Add Reviewer](#)

► Dependencies

Old Version History: [Base](#)

▼ Patch Set 1 2d15c15b07735dbf2d059d7a2e3eb21f330f2e9e

Author: [Roman Birg](#) <romanbirg@gmail.com> Apr 7, 2012 10:48 AM

Committer: [Roman Birg](#) <romanbirg@gmail.com> Apr 7, 2012 10:48 AM

Parent(s): 2413721b512a22cfff33adc8f83e4513657cdd030f change temp file location name to .aokp to avoid conflicts

Download: [checkout](#) | [pull](#) | [cherry-pick](#) | [patch](#) | [Anonymous HTTP](#) | [SSH](#) | [HTTP](#)  
git fetch http://gerrit.sudoservers.com:8080/AOKP/packages\_apps\_ROMControl refs/changes/45/45/1 && git cherry-pick FETCH\_HEAD

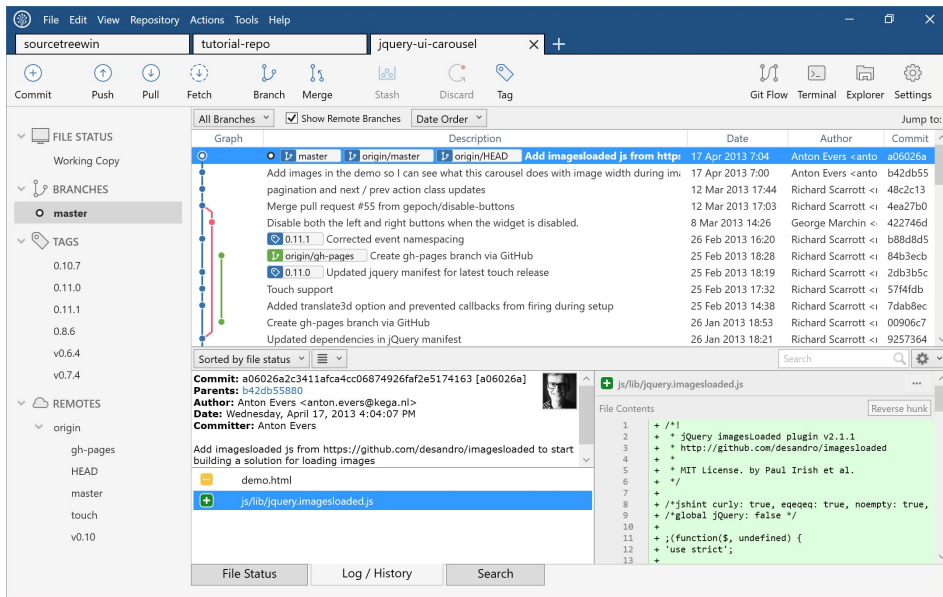
[Review](#) [Abandon Change](#) [Diff All Side-by-Side](#) [Diff All Unified](#)

	File Path	Comments	Size	Diff	Reviewed
►	<a href="#">Commit Message</a>			<a href="#">Side-by-Side</a> <a href="#">Unified</a>	
M	<a href="#">res/values/strings.xml</a>		+3, -1	<a href="#">Side-by-Side</a> <a href="#">Unified</a>	
			+3, -1		



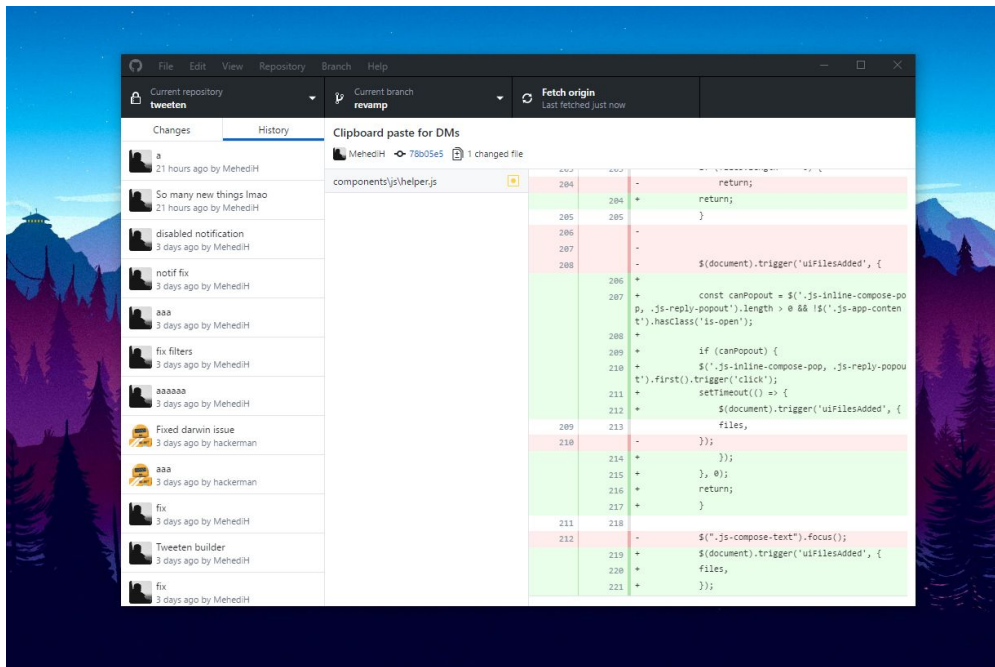
# Extras: UI

Existen herramientas con *User Interface* para trabajar con GitHub y similares. Una de las más conocidas es *SourceTree*.



# Extras: UI

Para GitHub existe también una versión desktop:



## Extras: *Cookbook* (repo de GitHub como servidor)

1. Crear un repositorio en GitHub con el nombre:

`[username].github.io`

2. Clonar el repositorio en local

3. Mete los ficheros que quieras servir y haz *commit/push* a la master

4. Espera un momento (poco, pero indefinido) y visita:

[https://\[username\].github.io](https://[username].github.io)

# ¡Muchas gracias!

---

Mail Capgemini: [cbeldape@capgemini.com](mailto:cbeldape@capgemini.com)

Mail personal: [cbeldaperez@gmail.com](mailto:cbeldaperez@gmail.com)

# Links de interés

Git y GitHub: <http://wpmallorca.com/2013/02/12/pero-que-es-github/>

GitHub: <http://conociendogithub.readthedocs.io/en/latest/data/introduccion/>

Git, la guía sencilla: <http://rogerdudler.github.io/git-guide/index.es.html>

GitHub glossary: <https://help.github.com/articles/github-glossary/>

Getting a Git repository:

<https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository>

GitHub desktop: <https://desktop.github.com/>

# Links de interés

Cmder: <http://cmder.net/>

Documentación Git oficial: <https://git-scm.com/docs/>

Uso GitHub:

<https://www.coursera.org/learn/nube-ios/lecture/pCojB/ejemplo-de-uso-de-github>

Makigas ¿qué es Git?: <https://www.youtube.com/watch?v=jSJ8xhKtfP4>

Resolución de problemas con Git:

[https://github.com/oslugr/curso-git/blob/master/texto/solucion\\_problemas.md](https://github.com/oslugr/curso-git/blob/master/texto/solucion_problemas.md)

# Links de interés

Branching and merging:

<https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Procedimientos-B%C3%A1sicos-para-Ramificar-y-Fusionar>