

Creación de un repositorio de software libre

Recomendaciones proyecto Laboratorio Software Libre Rivas

[**Desglose términos**](#_z3cyexa41mqj) **3**

[**Abstract**](#_2mwvv5a0dovk) **4**

[**Contexto - Software libre**](#_a0nbs0o87usa) **5**

[Tipos de licencias](#_dxlb1timrdtx) 5

[**Repositorios abiertos, comunidades**](#_yc5u586f8pfy) **7**

[Repositorio de software](#_yrara39ti2y5) 7

[Plataformas de repositorios de código abierto](#_jfagtq36f585) 7

[Ejemplos](#_rfxycrnzt9v) 7

[Sistemas de control de versiones](#_b3r1tx5jrghq) 8

[**Organización y administración**](#_pistt1qp3m91) **8**

[**Infraestructura - hardware**](#_xfcy3eqjscss) **9**

[Servidores propios](#_1o0fo03a2t1i) 9

[Servidores de terceros](#_x4t2digyx3bl) 9

[Solución mixta](#_6vt257y1zlqr) 10

[**Comparación soluciones software**](#_iip4n4wn0j3f) **11**

[Soluciones cloud privado](#_rxuvy0mdg9nx) 11

[Soluciones cloud público](#_wxcsx26gi69t) 11

[**WOW - forma de trabajar colaborativa**](#_7b711lst1quc) **11**

[QA - plataformas](#_bmvobz36p1t8) 11

[WIKIS - Documentación](#_nf1pohgoj9fh) 12

[Creación y gestión de un proyecto FOSS](#_b04pr6d6yn1d) 12

[**Problemática legal**](#_zh6c6ch0rcto) **14**

[**Conclusiones**](#_xbwx3v6n9vlc) **15**

[**Anexo I - Comparación Bitbucket & GitHub**](#_sw230tl4v27o) **16**

[**Anexo II - Guía buenas prácticas**](#_l6iwqjtp9hdg) **18**

|  |
| --- |
| **Versión documento** |
| **Versión** | 1.0.1 |
| **Fecha** | 9 Feb. 2018 |
| **Autor** | Ovidiu Mircea Moldovan |

#

# Desglose términos

|  |  |
| --- | --- |
| **SO** | Sistema Operativo |
| **Software** | Programas ejecutados por un sistema informático |
| **Open Software** | Software abierto |
| **Copyleft** | Se refiere a una práctica legal que consiste en el ejercicio del derecho de autor con el objetivo de propiciar el libre uso y distribución de una obra, exigiendo que los concesionarios preserven las mismas libertades al distribuir sus copias y derivados |
| **Licencia Software** | Contrato entre un licenciante y el licenciatario de un programa informático |
| **Cloud Computing** | Computación en la nube, paradigma que permite ofrecer servicios informáticos a través de una red (p.e. Internet) |
| **Hardware** | Las partes físicas tangibles de un sistema informático |
| ***FOSS* o *FLOSS*** | Software libre y de código abierto  |
| **IaaS, PaaS, SaaS** | Los acrónimos anteriores se refieren al tipo de acceso que se tiene a una sistema Cloud en forma de servicio: Infraestructura como Servicio, Plataforma como Servicio, Software como Servicio |
| **WOW** | Way of Working - Manera de trabajar |
| **QA** | Quality assurance - Garantía de calidad |
| **WIKI** | Un sitio web, cuyas páginas pueden ser editadas directamente desde el navegador, donde los mismos usuarios crean, modifican o eliminan contenidos que, generalmente, comparten |
| **Issue tracker** | Seguidor de problemas o hilo de discusión sobre problemas del proyecto |
| **Pull requests** | Hilo de discusión sobre los cambios y revisiones que están en progreso |
| **Server mirroring** | La duplicación de servidores es un proceso en la administración de redes a través del cual se crea continuamente una réplica exacta de un servidor en tiempo de ejecución. |

#

# Abstract

Este documento contiene información y recomendaciones orientadas a apoyar la puesta en marcha de una plataforma de repositorios de software libre (FLOSS) por parte del Ayuntamiento de Rivas dentro del marco del proyecto: “Laboratorio de Software Libre”.

En los primeros apartados se explicará el contexto que motivará las conclusiones finales. El anexo 2 contiene una lista de buenas prácticas para este tipo de proyectos.

Se debe tener en cuenta que este tipo de documentos sufren constantes cambios desde su inicio hasta su versión “X.0” que será la base para la puesta en marcha de la construcción de la plataforma.

#

# Contexto - Software libre

Nos referimos por el nombre de **“software”** al conjunto de componentes lógicos necesarios que hacen posible, por parte de los equipos informáticos, la realización de tareas específicas. En este conjunto hay que incluir los **Sistemas operativos** (que permiten el funcionamiento o la orquestación de la parte física), **programas para desarrollo de software** y **aplicaciones** con tareas específicas.

Todo software es desarrollado bajo un tipo de contrato o licencia concreto. Existen las siguientes **grandes categorías según su licencia** de uso:

* **El software propietario** no permite el uso o modificación son el permiso del dueño del mismo.
* **El software libre (FLOSS)** permite su distribución y cambio. Hay que tener en cuenta que **el software libre puede ser tanto gratuito como de pago** por tanto no se deben confundir.[[1]](#footnote-0)

## Tipos de licencias

En un principio se distingue entre el tipo del destinatario de la licencia entre **usuario final e distribuidores**. Aquí trataremos la distinción en función de los **derechos que cada autor se reserva** sobre su obra.[[2]](#footnote-1) Existe una tabla bastante completa que resume las principales característica de las licencias de software libre[[3]](#footnote-2).

1. **Licencias de software de código abierto permisivas**
	1. Se puede usar o modificar sin que ello conlleva ninguna obligación aparte de requerir la aplicación de la misma [licencia](https://es.wikipedia.org/wiki/Licencia) a todas las partes no modificadas y en cada archivo de licencia, así como los derechos de autor, patentes, marcas, y las comunicaciones originales de atribución de código redistribuido. Cada caso tiene pequeñas variaciones.
	2. Las más conocidas:
		1. [Academic Free License](https://es.wikipedia.org/wiki/Academic_Free_License) v.1.2. ;
		2. [Apache Software License](https://es.wikipedia.org/wiki/Apache_Software_License) v.1.1
		3. [Artistic License v.2.0](https://es.wikipedia.org/wiki/Artistic_License_v.2.0)
		4. [BSD License](https://es.wikipedia.org/wiki/Licencia_BSD).
		5. [MIT License](https://es.wikipedia.org/wiki/Licencia_MIT).
2. **Licencias de software de código abierto robustas fuertes**
	1. Disponen de **“copyleft” fuerte** ya que contienen una cláusula que obliga a que las obras derivadas o modificaciones que se realicen al software original se deban licenciar bajo los mismos términos y condiciones de la licencia original.
	2. Una lista de las principales:
		1. [EUPL](https://es.wikipedia.org/wiki/European_Union_Public_License)
		2. [Creative Commons](https://es.wikipedia.org/wiki/Licencias_Creative_Commons)
		3. [GPL](https://es.wikipedia.org/wiki/GNU_General_Public_License)
		4. Common Public License v.1.0.
		5. [GNU General Public License v.2.0](https://es.wikipedia.org/wiki/Licencia_p%C3%BAblica_general_de_GNU).
		6. [GNU General Public License v.3.0](https://es.wikipedia.org/wiki/Licencia_p%C3%BAblica_general_de_GNU).
		7. [Eclipse Public License](https://es.wikipedia.org/wiki/Eclipse_Public_License).
		8. OpenSSL License.
3. **Licencias de software de código abierto robustas débiles**
	1. Disponen un **“copyleft” débil** que implica que las modificaciones realizadas sobre el original se deben licencias bajo la misma licencia pero permite que las obras derivadas se puedan licenciar bajo otros términos y condiciones.
	2. Las más conocidas:
		1. [GNU Lesser General Public License v.2.1](https://es.wikipedia.org/wiki/Licencia_p%C3%BAblica_general_reducida_de_GNU).
		2. [Mozilla Public License](https://es.wikipedia.org/wiki/MPL)
		3. Open Source License.
		4. Apple Source License v.2.0
4. **Licencias de software de código cerrado**
	1. Se refieren al **software propietario o privativo.** *“En ellas los propietarios establecen los derechos de uso, distribución, redistribución, copia, modificación, cesión y en general cualquier otra consideración que se estime necesaria.”[[4]](#footnote-3)*
	2. Algunos ejemplos:
		1. ALUF, acuerdo de licencia de usuario final
		2. CLUF, contrato de licencia de usuario final
		3. EULA, el ALUF en el mundo anglosajón
5. **Licencia software de dominio público**
	1. Es la **licencia más laxa, casi inexistente**, que permite uso, copia modificación o redistribución con o sin ánimo de lucro.

#

# Repositorios abiertos, comunidades

## Repositorio de software

Un repositorio de software es un **lugar de almacenamiento** del cual pueden ser recuperados e instalados los paquetes de software en otros sistemas informáticos. Un repositorio remoto quita la necesidad de la presencia física para tal cometido permitiendo el acceso a través de las redes de datos al mismo (servidores en internet).

En general son utilizados por los creadores de software para compartir (y licenciar) paquetes, aplicaciones e incluso Sistemas Operativos. Pero se usan también para software de propósito general.

## Plataformas de repositorios de código abierto

Estas plataformas permiten **gestionar los repositorios de software,** también llamados de código en algunos casos. La función principal es la de permitir **compartir los recursos** presentes en el repositorio. Otras funciones importantes son las de:

* usar un **sistema de privilegios** (administrador, colaborador etc.)
* poder definir los r**epositorios públicos o privados** (en el caso de que se deseé)
* usar un **sistema de versiones**
* usar **filtros de malware & spam**
* usar sistemas de seguimiento de problemas (**Issue tracker**) y soluciones (**Pull requests**)
* usar foros de discusión o **lista de correos electrónicos** (o ambos)
* introducir automáticamente un **fichero de documentación** (readme) y un **fichero de licencia**, en caso de que falten

## Ejemplos

Una lista de plataformas que gestionan repositorios de **FLOSS** desde nivel local a europeo:

* <https://github.com/ctt-gob-es> (FLOSS Nacional)
* <http://www.juntadeandalucia.es/economiainnovacionyciencia/repositorioaall/index.jsf?select_locale=es> (FLOSS Andalucia)
* <https://joinup.ec.europa.eu/collection/open-source-observatory-osor/about> (OSOR - EU)
	+ <https://joinup.ec.europa.eu/page/competence-centres-open-source-software-oss#section-25> - Implementaciones en España
* https://github.com/AyuntamientoMadrid

## Sistemas de control de versiones

Existen sistemas automáticos, **herramientas**, que permiten la **gestión de los diversos cambios** (versiones) que se realizan sobre un código fuente o software concreto pudiéndose incluir aquí la documentación u otros tipos de ficheros. Los ejemplos más conocidos serían:

* [CVS](https://es.wikipedia.org/wiki/CVS),
* [Subversion](https://es.wikipedia.org/wiki/Subversion),
* [ClearCase](https://en.wikipedia.org/wiki/Rational_ClearCase),
* [Git](https://es.wikipedia.org/wiki/Git),
* [Mercurial](https://es.wikipedia.org/wiki/Mercurial),

Las principales características son la existencia de un **mecanismo de almacenamiento**, la posibilidad de **realizar cambios** sobre los elementos y disponer de un **registro histórico** de cada elemento que permita restaurar versiones antiguas.

Aunque todavía se usan varios de estos sistemas ya que su utilidad depende del cometido que se le da el más completo para los proyectos de propósito general es[**GIT**](https://git-scm.com/)[[5]](#footnote-4).

# Organización y administración

Todo proyecto software tiene unas fases por los que tiene que pasar a lo largo de su vida. En el caso de proyectos abierto y colaborativos hay que tener en cuenta además de lo habitual **una serie de criterios para su buen funcionamiento**. Algunas están detalladas [aquí](https://consugus.github.io/open-source-guide/how-to-contribute/#orient%C3%A1ndote-a-un-nuevo-proyecto)[[6]](#footnote-5).

Las principales figuras son **los autores y los dueños** (la organización - administradores finales), **los encargados y colaboradores** (que pueden no formar parte de la organización - administradores de la plataforma y contribuidores) y **los miembros de la comunidad** (tanto utilizadores como creadores de contenido). Los proyectos más grandes disponen de **grupos de trabajo o comisiones** que actúan como órgano de gestión de la plataforma.

Si no estamos hablando de software con licencia de dominio público hay que poner en marcha un sistema de documentación que cubra al menos **las reglas de conducta** (CODE\_OF\_CONDUCT), el **tipo de licencia** del código abierto y un fichero **README** que es un manual de instrucción como una presentación del proyecto.

La plataforma tienen que disponer, tal como ya se mencionó, de un **issue tracker, un pull requests** y al menos de **una lista de correos electrónicos.**

Aunque la plataforma no disponga de una **página para la documentación (wiki)** hay que asociar una propia o de terceros para el buen funcionamiento del proyecto.

Si se desea crear una comunidad necesariamente se tendrán que compartir parte de los privilegios de administración, manteniendo evidentemente el control último, con la misma. También es recomendable crear grupos de trabajo para dirigir los aspectos más importantes de la plataforma y luego para los proyectos más destacados.

# Infraestructura - hardware

Actualmente existen varios tipos de infraestructuras que pueden solucionar el problema del acceso remoto a recursos software. Aquí se detallarán las dos posibilidades principales que parecen más acordes al proyecto y su correcto funcionamiento. Por un lado está la posibilidad de poner en función **unos servidores propios**, infraestructura denominada **Cloud Privado** (IaaS), o alquilar las máquinas o **servicios de proveedores,** el llamado **Cloud Público** (PaaS, SaaS, etc.). Detallamos estos dos casos pero hay que tener en cuenta que existe la posibilidad de crear una mezcla de estas dos infraestructuras, el llamado **Cloud Híbrido**.

## Servidores propios

Los servidores propios son **maquinas que una organización gestiona internamente**. En el caso de usar tecnologías de virtualización este sistema se llama también Cloud privado. Puede tener o no acceso desde fuera con o sin restricciones.

Es importante recalcar que todo el peso de la gestión recae sobre la organización. Los **aspectos positivos** son:

* Control total sobre el hardware y software
* Acceso físico a las máquinas
* Sin dependencias de terceros en la gestión de los recursos

Los **aspectos menos positivos** son:

* Necesidad de implementar soluciones de seguridad (firewall, VPN y accesos, gestión usuarios etc.)
* Necesidad de mantener unas condiciones ambientales concretas
* Necesidad de contar con redundancia y alta disponibilidad
* Necesidad de disponer de una conexión de red suficientemente potente

## Servidores de terceros

En esta se hará referencia solo al conjunto de servicios **Cloud Públicos**. Estos **servicios son proporcionados por entidades** públicas o privadas, para varias organizaciones que comparten el entorno hardware pero no software. La gestión que es requerida en los servidores privados se reduce considerablemente.

Los puntos fuertes y débiles de este modelo se obtienen invirtiendo los enumerados en el apartado anterior. Además hay que tener en cuenta lo siguiente:

* 1. Los servicios ofrecidos suelen estar **distribuidos geográficamente** y disponen de **buenas conexión de red** (se puede elegir entre servidores localizados en el **espacio de la EU** u otros).
	2. La gestión por parte de la organización se puede reducir aún más si se buscan servicios **SaaS** como sería el caso de **GitHub** pero al mismo tiempo **se pierde el control** total, según la licencia que ofrecen estas plataformas, y se introducen límites al uso.

## Solución mixta

En la actualidad las debilidades que presentan los servidores privados se pueden superar optando por una solución mixta. Es decir, disponer de una plataforma de gestión de repositorios abiertos en servidores privados y de una copia actualizada, **mirroring**, en servidores de terceros.

#

# Comparación soluciones software

En este apartado se tratarán tanto las soluciones software para servidores privados como para los públicos.

## Soluciones cloud privado

Según se ha podido observar en los apartados anteriores este tipo de cloud tiene más desventajas para organizaciones pequeñas y medias.

Si se desea seguir por esta vía se recomienda buscar soluciones a los problemas enumerados desde el principio de la implantación.

Existe software para crear plataformas de gestión de repositorios públicos gratuitos como [GitLab](https://about.gitlab.com/community/) community (la versión gratuita) pero que tiene una lista menor de funcionalidades en comparación con los servicios profesionales ofrecidos por GitLab o los servicios especializados como GitHub, Bitbucket etc.

## Soluciones cloud público

En este apartado se enumeran varias **soluciones de tipo SaaS**, es decir software como servicio. **No se consideran los servicios tipo IaaS** ya que suponen añadir otro nivel de complejidad innecesario.

Aquí hay que mencionar a la fuerza dos servicios de gran interés ([Anexo I - Comparación Bitbucket & GitHub](#_sw230tl4v27o)).

Por un lado **Bitbucket** es un servicio gratuito para equipos de hasta 5 contribuyentes por proyecto que permite crear repositorios privados. Se recomienda usar en proyectos que no se van a publicar o no están en un estado de calidad suficiente.

Por otro lado **GitHub** es la elección de muchas organizaciones tanto del ámbito de lo público como de lo privado. Esto es así gracias a la gran lista de funcionalidades que ofrecen para las cuentas gratuitas. Hay que especificar algunas: sin limite de contribuyentes por proyecto, control de contribuyentes (commits) y equipos, creación de documentación (wiki) y gestión de tareas por proyectos (Kanban), issues tracker (seguimiento de incidencias), página gratuita del proyecto (<https://pages.github.com/>) etc.

# WOW - forma de trabajar colaborativa

## QA - plataformas

En ocasiones el éxito de una plataforma depende de la **calidad** que esta ofrece. Aquí hay que incluir también **la documentación y la seguridad.**

Para la calidad de código existen varias formas, dependiendo del alcance en lo relacionado a lenguajes de programación, SO o librerías de software que se quiera tener. Es muy recomendable contar con **herramientas de análisis sintáctico y lógico de código** (Por ejemplo Sonar). Otra forma muy recomendable de gestión de calidad de código es el sistema de revisión dentro de los mismos equipos, por ejemplo cada cambio (commit) subido a la rama común debe tener al menos una revisión de otro miembro del equipo u otros colaboradores (expertos, arquitectos, drivers etc.).

Por otro lado está la necesidad de **generar pruebas** que aseguren el correcto funcionamiento del mismo a todos los niveles (**Unit test, Function Test & System Test**) que recae sobre la propia organización. Estas pruebas se pueden automatizar gracias a plataformas como **Jenkins** que se encargan de orquestar su ejecución y poner de manifiesto los resultados finales.

## WIKIS - Documentación

Teniendo en cuenta que este proyecto está basado en la filosofía open se deberá contar además con una documentación adecuada y accesible (wikis).

La documentación debe estar presentes en al menos dos niveles:

* A nivel de proyecto software, es decir en cada aportación, a través del fichero Readme que tiene un formato de texto enriquecido para mejorar su legibilidad y si es posible a través de una wiki.
* Una wiki o documentación a nivel de plataforma que sea una guía para los usuarios además de una lista de respuestas a las preguntas más frecuentes (FAQ).

## Creación y gestión de un proyecto FOSS

Esta sección debe entender desde una perspectiva pragmática ya que tiene relación directa con l**a política y las condiciones que la organización** marcará para el uso del repositorio.

En general **los administradores** de este tipo de plataformas se reservan el derecho de realizar **las gestiones oportunas** para el correcto funcionamiento reservándose el derecho a ejercer de supra administradores.

**Los administradores o grupos de administración o de trabajo** están en la primera línea de la gestión de la plataforma y pueden estar formados por personas propias de la organización y **personas de la comunidad** que muestran su interés y colaboran activamente.

Existen guías que se basan en experiencias previas para poner en marcha comunidades de tipo Open[[7]](#footnote-6). Entre las recomendaciones encontramos:

* Un proyecto colaborativo acepta cambios de cualquiera (esto no excluye el control de la contribución).
* Cualquiera puede usar los proyectos publicados e incluso crear sus propios proyectos tomandolos como base (evidentemente se tiene en cuenta la licencia).
* La transparencias es importante y el hecho de tener el código a disposición de cualquiera es fundamental.
* Hacer a la gente sentirse bienvenida. Hay que explicar claramente como se puede usar el software así como la forma en que se puede contribuir (el fichero contributing).
* Documentar todo y no solo la parte técnica. Hay que usar el fichero README si mucha gente incurre en los mismos problemas (explicitando la solución) y tener una wiki abierta y actualizada.
* Hay que estar pendiente y ser comunicativo porque la gente ofrece feed-back. Se tienen que establecer diálogos para hacer a la gente sentirse parte del proyecto.
* Se tiene que crear un lugar de congregación para la comunidad (por ejemplo grupos Novagob junto a listas de correos, cuenta dedicada de Twitter, foros etc.).
* Hay que compartir la propiedad y responsabilidad de los proyectos (se tiene que disponer de ficheros de tipo: CONTRIBUTORS o AUTHORS en cada proyecto).
* El fichero README puede ser como la “constitución” de los proyectos.
* Mantener las conversaciones centradas en la acción y la resolución de asuntos.
* Identificar personas que pueden dirigir la orientación de los proyectos en un punto de bloqueo por diferencias de opiniones. Pueden ser gestores o mantenedores de los proyectos tanto personas físicas como grupos. Se pueden especificar en un fichero GOVERNANCE.

Otros consejos útiles para conseguir aumentar y dinamizar una comunidad se encuentran en [estos enlaces.](https://opensource.guide/)

#

# Problemática legal

Los tipos de licencias de software libre (FOSS) se han enumerado en un apartado anterior. En general los repositorios públicos promovidos por las administraciones se organizan bajo la licencia **GPLv3**[[8]](#footnote-7).

Aparte existen una serie de factores que se deben tener en cuenta:

1. El uso de **software de terceros puede condicionar el tipo de licencia que se ofrezca**. Es decir, no se puede usar una licencia menos restrictiva que la que usan las librerías, plugins etc. que forman parte del proyecto. Este caso solo aplica si este software se ofrece junto al código propio. Es por ello que en general se especifica una **lista de dependencias** o se usa **software de terceros con la misma licencia** que se aplicará sobre el proyecto o se **indica la dependencia** a dicho software (y la versión).
2. La Unión Europea dispone de un programa dedicado en exclusiva al FOSS para las administraciones públicas llamada[**Joinup**](https://joinup.ec.europa.eu/)**.**

*“Joinup es una plataforma de colaboración creada por la Comisión Europea y financiada por la Unión Europea a través de las soluciones de interoperabilidad para el programa de administraciones públicas, empresas y ciudadanos (ISA2). Ofrece varios servicios que tienen como objetivo ayudar a los profesionales del gobierno electrónico a compartir su experiencia entre ellos. También esperamos ayudarlos a encontrar, elegir, reutilizar, desarrollar e implementar soluciones de interoperabilidad.”[[9]](#footnote-8)*

La inclusión en el mismo está condicionado por el cumplimiento de **una serie de condiciones** disponibles en la misma plataforma.

1. Según el **artículo 16 del Real Decreto 4/2010** todo software que pasa a ser liberado por la administración pública tiene **por defecto** cobertura o licencia **“EUPL - European Union Public Licence”**[[10]](#footnote-9) (copyleft fuerte). El artículo 46 establece que las administraciones públicas deben mantener registros actualizados de las solicitudes de reutilización gratuita, en cooperación con un Centro de Transferencia de Tecnología creado y administrado por la Administración General del Estado y conforme a los principios establecidos por el Marco Nacional de Interoperabilidad.[[11]](#footnote-10)

**Nota:** Estas consideraciones **no cubren todo el espectro legal** relacionado con este tipo de plataformas ya que se centra en la problemática de las licencias.

# Conclusiones

El análisis del contexto relacionado a plataformas de repositorios de Open Software junto a los objetivos del proyecto dejan bastante clara la configuración que más encaja con las expectativas:

* Se recomienda la creación de una cuenta de GitHub Organization. Es gratuita (para repositorios privados y sin soporte dedicado) y dispone de funciones extras como la gestión de usuarios y niveles de acceso a los proyectos. Además hay que tener en cuenta [el gran número de administraciones](https://government.github.com/community/)[[12]](#footnote-11) tanto a nivel nacional como local que han apostado por esta solución.
* La licencia más usada en este tipo de plataformas es la GPLv3. Por tanto se recomienda esta por estar acorde a los objetivos del proyecto.
* Existen guías de buenas prácticas que ayudan al buen funcionamiento de un proyecto software colaborativo y abierto. En el [Anexo II - Guía buenas prácticas](#_l6iwqjtp9hdg) se pueden leer más detalles.
* A partir de estos datos se debe especificar **un documento con la “hoja de ruta”** para la implementación y finalmente el mantenimiento del proyecto.

#

#

# Anexo I - Comparación Bitbucket & GitHub

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nombre** | **Tipo** | **Control de versión** | **Wiki** | **Coste** | **Ventajas** | **Observaciones** |
| [BitBucket](https://bitbucket.org/) | Online | git, Mercurial | - | A partir de 5 licencias.Repositorios privados gratuitos. | Pull request - Solicitudes de extracción y revisiones de códigoRepositorios privados ilimitadosComparación de ramas e historial de commitsBitbucket Mac y el cliente de Windows llamado SourceTree; Aplicación de Android llamada BitBeakerIntegración con herramientas como Jira, Crucible, Bamboo, Jenkins | Recomendado para proyectos con equipos pequeños (<=5 componentes) públicos o privados. |
| GitHub | Online | git | si | Repositorios públicos gratuitos.Los repositorios privados son de pago. | Comentarios de código resaltadosRevisión de código colaborativoUn rastreador de problemas integrado dentro de su proyectoHitos y etiquetas dentro de los proyectosVistas de comparación de ramasAplicaciones nativas para escritorios de Windows y Mac, y también una aplicación de AndroidSoporte para más de 200 lenguajes de programación y formatos de datosPáginas Github, una característica para publicar y hospedar dentro de GithubSeguridad, como el uso de SSL, SSH y / o https para toda la transmisión de datos, y autenticación de dos factores para iniciar sesiónIntegración API para una fácil integración de herramientas de terceros e integración con una gran cantidad de otras herramientas y plataformas. Algunos ejemplos son Asana y Zendesk para el seguimiento de problemas / errores; CloudBees, Travis y CodeClimate para integración continua (CI); Hosting en la nube de AWS, Windows Azure, Google Cloud y Heroku.Resaltado de sintaxis | Recomendado para proyectos con equipos grandes dónde no importe que el código sea público.Ejemplos uso en la administración pública: https://government.github.com/community/ |

#

# Anexo II - Guía buenas prácticas

Esta guía pretende introducir **una serie de herramientas de desarrollo software** **agile** en el contexto de los servicios ofrecidos por la plataforma GitHub principalmente pero también alternativas o complementarias.

El trabajo en equipo conlleva la necesidad de disponer de **un espacio de trabajo común**. En la actualidad se utilizan los repositorios remotos que cuentan con un sistema de versiones incorporado, generalmente **git**.

GitHub ofrece repositorios gratuitos, públicos, para el trabajo en equipo.

1. Para las organizaciones existe **una capa de control extra** para gestionar el acceso y la otorgación de roles. Por tanto una vez creada una cuenta para la organización se tiene que crear un repositorio con las normas de la comunidad y luego repositorios nuevos para cada proyecto. Estos repositorios deben tener asignados todas las opciones que se permiten (**Issue tracker, pulls request control, wiki, project, vulnerability alerts, una página web del proyecto si se necesita o para un foro con discusiones sobre el mismo, los ficheros de licencia, readme, contributors etc.**)
2. Para cada proyecto se deben definir **un administrador**, o uno para todos si es posible, luego **una serie de contribuidores** (personas de fuera de la organización) y **colaboradores** (personas dentro de la organización) que deben tener otorgados los derechos necesarios. Estos últimos serán los encargados de crear código pero también la documentación. Pueden ser estos mismo los **encargados de mantener** el código y **arreglar** los errores que se encuentran u otras personas. Un esquema de la forma de trabajo se encuentra más abajo en la **fig.1**.
3. El uso de metodologías ágiles es recomendado. La forma de trabajo está representada en la **fig.3**.
4. Se recomienda **usar varias ramas** para el mismo repositorio disponiendo, por ejemplo, de una rama para **desarrollo** donde se subirán los cambios y una rama **master (producción)** o rama que tendrá la parte probada y versionada que se indicará que se puede usar. Un esquema de la forma de trabajo se encuentra más abajo en la **fig.2**. Debe existir una **persona o un grupo** que tenga la función de **revisar el código** antes de que este pueda ser introducido (merged) en la rama. Todo código que se haga público se debe encapsular en sus respectivas formas (por ejemplo en rpms o dpk u otros formatos) para facilitar su uso por parte de los usuarios.
5. El uso de **herramientas de unit, function & system test** es fundamental para garantizar la calidad del código. Hay que tener en cuenta que actualmente se entiende la forma de trabajo más eficiente en el desarrollo de software la **metodología TDD** es decir diseño dirigido por pruebas donde el código se escribe a la par que las pruebas básicas y de unidad que lo comprobarán. Se recomienda automatizar las pruebas en forma de scripts o plugins para **Jenkins**.

Un esquema con la forma de trabajo realizado por cada persona implicada:



**Fig.1 Contributors workflow GitHub (**[**fuente**](https://github.com/CoolProp/CoolProp/wiki/Contributing%3A-git-development-workflow)**)**

La forma de trabajo habitual en equipos que utilizan **git** como control de versiones:



**Fig.2 Interacción principales ramas (**[**fuente**](http://marcgg.com/blog/2016/02/22/git-flow-heroku-pipelines/)**)**

Los equipos, y cada componente de los mismos, tienen que seguir estos pasos para cada iteración:

****

**Fig.3 Ciclos de trabajo en metodologías ágiles (**[**fuente**](https://www.emaze.com/%40AOWZOLIL/agiles)**)**

1. https://www.tuyu.es/licencia-software/ [↑](#footnote-ref-0)
2. https://es.wikipedia.org/wiki/Licencia\_de\_software [↑](#footnote-ref-1)
3. https://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n\_de\_licencias\_de\_software\_libre [↑](#footnote-ref-2)
4. https://es.wikipedia.org/wiki/Licencia\_de\_software [↑](#footnote-ref-3)
5. https://git-scm.com/ [↑](#footnote-ref-4)
6. https://consugus.github.io/open-source-guide/how-to-contribute/#orient%C3%A1ndote-a-un-nuevo-proyecto [↑](#footnote-ref-5)
7. https://opensource.guide [↑](#footnote-ref-6)
8. https://github.com/AyuntamientoMadrid/transparencia/blob/master/LICENSE-AGPLv3.txt [↑](#footnote-ref-7)
9. https://joinup.ec.europa.eu/ [↑](#footnote-ref-8)
10. https://ec.europa.eu/info/european-union-public-licence\_es [↑](#footnote-ref-9)
11. http://www.europarl.europa.eu/document/activities/cont/201307/20130708ATT69346/20130708ATT69346EN.pdf [↑](#footnote-ref-10)
12. https://government.github.com/community/ [↑](#footnote-ref-11)