Git y Gitlab Fujo de trabajo

Implementacion de entornos de desarrollo con control de versiones.

Lo primero que hay que tener en cuenta es que el flujo de trabajo con git involucra a todos los integrantes del equipo, por lo que se requiere que esten en buena disposición y dispuestos a cambiar sus paradigmas, así mismo que entiendan en que consiste trabajar con git.

Acerca del control de versiones

¿Qué es el control de versiones, y por qué debería ser importarte?

El 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.

Si eres diseñador gráfico o web, y quieres mantener cada versión de una imagen o diseño (algo que sin duda quieres), un sistema de control de versiones (Version Control System o VCS en inglés) es una elección muy sabia. Te 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. Usar un VCS también significa generalmente que si fastidias o pierdes archivos, puedes recuperarlos fácilmente. Además, obtienes todos estos beneficios a un coste muy bajo. (fuente: http://git-scm.com/book/es/Empezando-Acerca-del-control-de-versiones)

GIT

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. 3 Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux. (fuente: wikipedia)

GitLab

Es un aplicación para administrar proyectos con el control de versiones Git, basada en Ruby on Rails es software libre. Tiene un gran parecido tanto visual como funcionalmente a lo que es GitHub. En el podemos observar algunas de sus fucniones como manejar distintos proyectos, manejar usuarios, ver commits, ver archivos, ver ramas, ver gráficos, asignar tareas, ver meger, editar archivos, Wiki, entre otras.

Requisitos programador.

- Disposicion al cambio
- Manejo de GIT
- Manejo de github o gitlab.
- Conocimiento de Markdown.

Requerimientos de software

- cliente ssh
- git
- Gitlab
- (opcionales)
  - Meld
  - Git GUI

Pre-Configuracion

- Instalar git
- Instalar cliente ssh
- Configurar git

```
    git config --global user.name "USERNAME"
    git config --global user.email "EMAIL@EMAIL.com"
    git config --global color.ui true
```

- Crear llave ssh

```
    ssh-keygen
```

- Ingresar a gitlab
- Inscribir llave ssh en gitlab

.

.

.

.

.

.

Flujo de trabajo FORK

1. Hacer un FORK del repositorio a tu cuenta.
2. Clonar el repositorio a tu maquina.

```
git clone git@REPO.git
```

3. Agregar upstream remoto.

```
git remote add upstream git@REPO.git

# Esto ahora te permitirá que hacer un pull de cambios del origen localmente y combi
narlos, así:
git fetch upstream
git merge upstream/master
```

4. Crear una BRANCH con titulo detallado sobre lo que vas a trabajar y cambiate a ella.

```
git checkout -b mejora_345
```

5. Trabajar
6. Agregar los cambios

```
git add FILE
# o
git add .
```

7. Haz COMMIT

```
git commit --am "COMENTARIO DETALLADO"
```

8. Haz un PUSH

```
git push origin mejora_345
```

9. Crea un PULL REQUEST

.

.

.

.

Comandos Basicos

- Crea un repositorio

```
git init
```

- Crea un copia en gitlab (clone)

```
git clone git@github.com:sbadia/vagrant-gitlab.git
```

- Agregar archivos al stage area (archivos listo para hacer commit)

```
git add <filename>
# o
git add .
```

- Crear un commit

```
git commit -m "Commit message"
```

- Añadir remoto

```
git remote add origin <server>
```

- Enviar cambios a un repositorio remoto

```
git push origin master
```

- Crear una rama

```
git branch <branch_name>
```

- Cambiar a una rama

```
git checkout <branch_name>
```

- Volver a la rama principal

```
git checkout master
```

- Crear una rama y cambiarse a ella

```
git checkout -b feature_x
```

.

.

- Borrar una rama

```
git branch -d <branch_name>
```

- Subir una rama al repositorio

```
git push origin <branch_name>
```

- Recibir repositorios remotos

```
git fetch <remote_name>
```

- Fusionar otra rama con la rama activa

```
git merge <branch>
```

- Actualizar repositorio local con los ultimos commits en el remoto

```
git pull
```

- Revisar cambios entre ramas

```
git diff <source_branch> <target_branch>
```