Colaboración en equipo en Github

Github se ha convertido en la piedra angular de todo lo relacionado con el software de código abierto. Los desarrolladores lo adoran, colaboran en él y constantemente crean proyectos increibles . Además de alojar nuestro código, la atracción principal de GitHub es usarlo como una herramienta de colaboración. En este documento, exploremos algunas de las características más útiles de GitHub, especialmente para trabajar en equipo, ¡haciéndolo más eficiente, productivo y, lo más importante, divertido!.

Github y colaboración de software

Una cosa que me parece muy útil es integrar el Wiki de Github en el proyecto del código fuente principal.

Este tutorial asume que ya está familiarizado con Git, el sistema de control de versión distribuida de código abierto, creado por Linus Torvalds en 2005. . Además, ya deberías tener una cuenta de Github y conocer como realizar algunas funciones básicas, como crear un repositorio y enviar cambios a Github.

En el mundo de los proyectos de software, es inevitable que nos encontremos trabajando en equipo para entregar un proyecto. Para este tutorial sobre Github y la colaboración en equipo, exploraremos algunas de las herramientas más comunes que generalmente necesitamos cuando trabajamos con equipos de software. Las herramientas discutidas son:

1. **Adding Team Members (Agregando miembros al equipo) : Organización & Colaboradores**
2. **Pull Request: Sending & Merging**
3. **Bug Tracking: Problemas de github**
4. **Analytics: Graphs & Network**
5. **Project Management: Trello & Pivotal Tracker**
6. **Continuous Integration: Travis CI**
7. **Code Review: Line Commenting & URL queries**
8. **Documenting: Wiki &Hubot.**

Agregando miembros al equipo

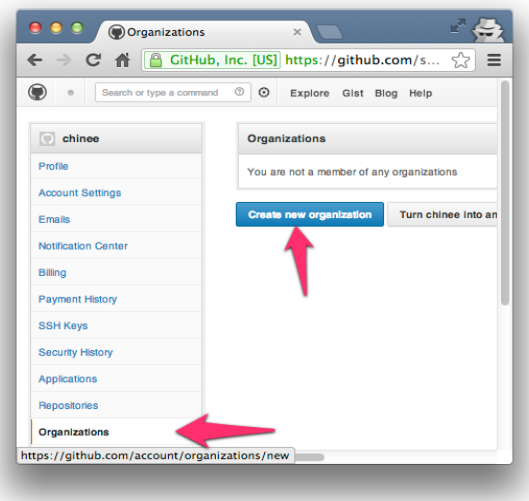
En general, hay dos formas de configurar Github para la colaboración en equipo:

Organizaciones: el propietario de la organización puede crear muchos equipos con diferentes niveles de permisos para varios repositorios.

Colaboradores: el propietario del repositorio puede agregar colaboradores con acceso de lectura + escritura para un único repositorio.

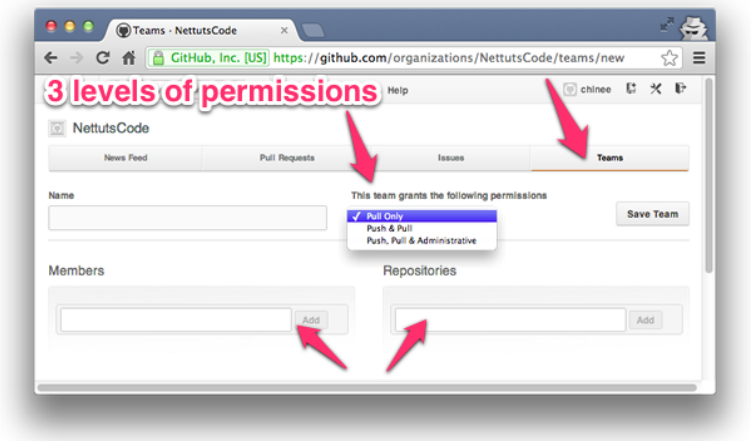
Organizaciones

Si supervisa varios equipos y desea establecer diferentes niveles de permisos para cada equipo con varios miembros y agregar cada miembro a diferentes repositorios, entonces la organización sería la mejor opción. Cualquier cuenta de usuario de Github puede crear organizaciones gratuitas para repositorios de código fuente abierto. Para crear una organización, simplemente vaya a la página de configuración de su organización:



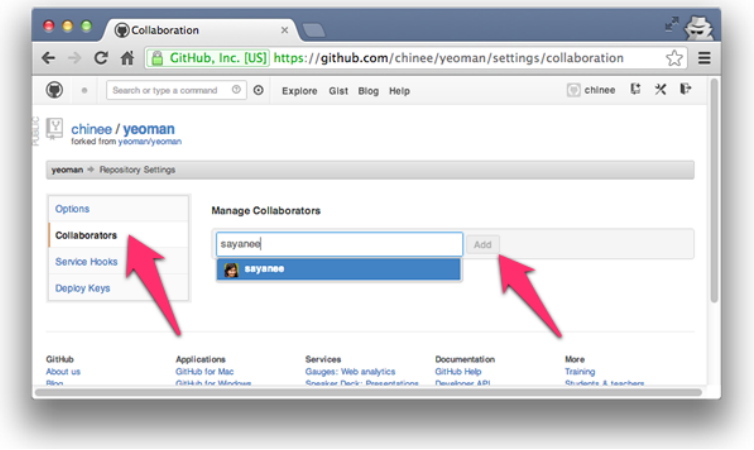
Para acceder a la página de equipos de su organización, simplemente vaya a http://github.com/organizations/[organization-name]/teams para verlos o incluso visite https://github.com/organizations/[organization- name] / teams / new para crear nuevos equipos con miembros de 3 niveles de permisos diferentes, como:

1. Solo Pull : Fetch and Merge con otro repositorio o una copia local. Acceso de solo lectura.
2. Push and Pull: (1) junto con updating del repositorio remoto. Acceso para leer + escribir.
3. Push, Pull & Administrative: (1), (2) junto con los derechos de información de facturación, creación de equipos y cancelación de cuentas de la organización. Acceso de lectura + escritura + administración.



Colaboradores

Los colaboradores se utilizan para otorgar acceso de lectura + escritura a un único repositorio propiedad de una cuenta personal. Para agregar Colaboradores, (otras cuentas personales de Github) simplemente vaya a: https://github.com/[username]/[repo-name]/settings/collaboration



Una vez hecho esto, cada colaborador verá un cambio en el estado de acceso en la página del repositorio. Después de tener acceso de escritura al repositorio, podemos hacer git clone trabajar en los cambios, hacer un git pull para combinar y fusionar cualquier cambio en el repositorio remoto y finalmente git push, para actualizar el repositorio remoto con nuestros propios cambios:

Pull request

Los pull request son una excelente manera de contribuir a un repositorio de forma independiente al bifurcarlo (hacer fork). Al final

del día, si lo deseamos, podemos enviar un pull request al propietario del repositorio para fusionar nuestros cambios de código. El pull request en sí mismo puede desatar discusiones sobre la calidad del código, características o incluso la estrategia general.

Veamos ahora los pasos básicos para un pull request.

Inicializando un Pull Request

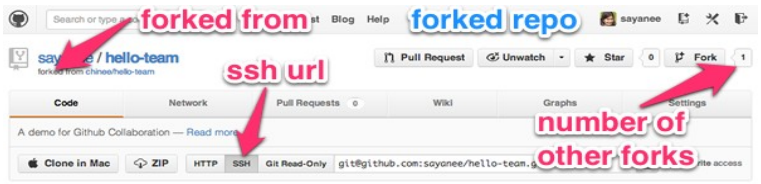
Hay dos modelos de pull request en Github:

Modelo Fork & Pull: se usa en un repositorio público para el que no tenemos acceso de hacer push.
Modelo de repositorio compartido (share Repository Model): se usa en un repositorio privado para el que tenemos acceso de hacer push. Hacer fork no es obligatorio es este caso.

Aquí vemos el flujo de trabajo entre dos usuarios (repo-owner y forked-repo-owner) para el modelo Fork y Pull:

Identifique el Repositorio de Github con el que desea contribuir y haga clic en el botón "Fork" para crear un clon del repositorio en su propia cuenta de Github:

Esto creará una copia exacta del repositorio en su propia cuenta.



Elije SSH URL para que le pida su contraseña clave SSH en lugar del nombre de usuario y la contraseña cada vez que presione git o git pull. A continuación, clonaremos este repositorio en nuestra máquina local:

```
$ git clone [ssh-url] [folder-name]
$ cd [folder-name]
```

En general, crearemos una nueva rama git para cada nueva característica. Esta es una buena práctica porque en el futuro si actualizamos la rama después de algunas discusiones, el pull request se actualizará automáticamente. Vamos a crear una nueva rama para hacer un cambio muy simple : modificar el archivo readme.md:

```
$ git checkout -b [new-feature]
```

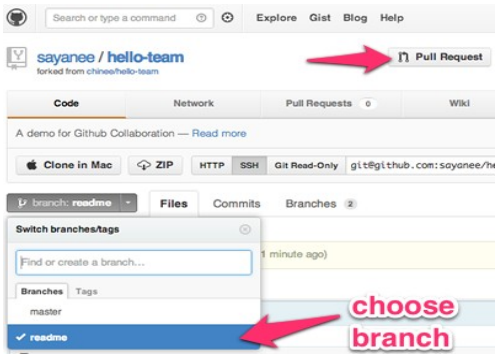
Después de hacer las adiciones pertinentes para crear las nuevas características solo confirmaremos los nuevos commit y checkout a la rama principal de git:

```
$ git add .
$ git commit -m "information added in readme"
$ git checkout master
```

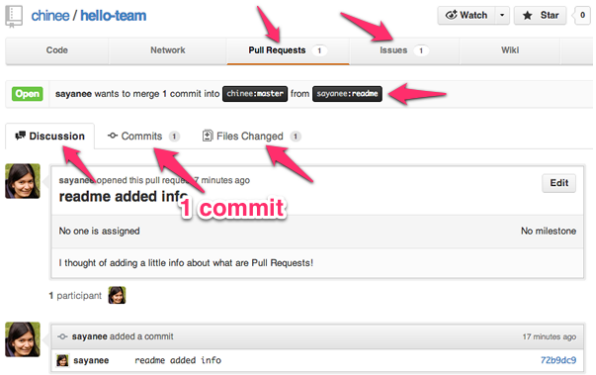
En este punto, hacemos un push de la rama al repositorio remoto. Para esto, primero verificaremos el nombre de la rama con la nueva característica así como los alias de repositorio remoto de git. Luego realizamos push a los cambios usando git push [git-remote-alias] [branch-name]:

```
$ git branch
* master
readme
$ git remote -v
origin git@github.com:[forked-repo-owner-username]/[repo-name].git (fetch)
origin git@github.com:[forked-repo-owner-username]/[repo-name].git (push)
$ git push origin readme
```

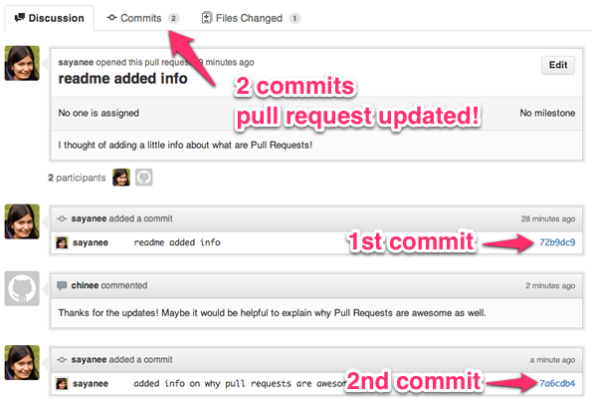
En nuestra página de repositorio donde se a realizado el fork , cambiaremos a la rama con la nueva característica y luego haremos clic en el botón "Pull Request".



Después de enviar el pull request, nos llevará directamente a la página del pull request del repositorio original. Veremos nuestro pull request tanto como un nuevo issue así como un nuevo pull request.



Después de todo estos cambios, es posible que el propietario del repositorio al que se le han hecho fork quiera agregar cambios a la nueva característica. En este caso, realizaremos el checkout en la misma rama en nuestra máquina local, haremos commit y realizaremos un push para devolverlo nuevamente a Github. ¡Cuando visitamos la página de pull request del repositorio original, se actualizará automáticamente!.



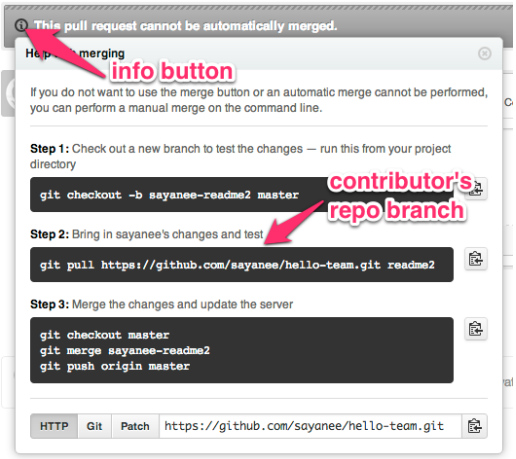
Merging (fusionando) un pull request

Si eres el propietario original del repositorio, hay dos formas de combinar un pull request entrante:

Fusionando directamente en Github: si nos estamos fusionando directamente en Github, entonces asegúrese de que no haya

conflictos y esté listo para fusionarse en la rama principal. El propietario original del repositorio simplemente puede hace clic en el botón verde "Merge Pull Request" para hacerlo:

Fusionarse en nuestras máquinas locales: en otras ocasiones, puede haber conflictos de fusión, y al hacer clic en el botón "info", Github tendrá instrucciones claras sobre cómo podemos fusionar la rama en nuestra máquina local al incorporar los cambios desde la rama del contribuyente:



Existen diferentes modelos de ramificación utilizados para el control de versiones en equipos de desarrollo de software. Aquí hay dos modelos de flujo de trabajo git populares: (1) Github workflow que tiene un modelo de ramificación simple y utiliza pull request y (2) Gitflow que tiene una ramificación más extensa. El modelo que finalmente se elija dependerá del equipo, el proyecto y la situación.

Bug Tracking (Seguimiento de errores)

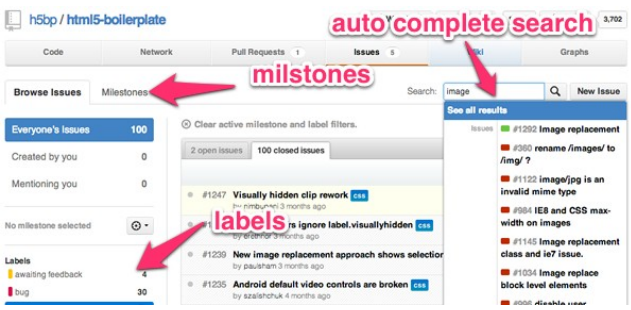
Los pull request son una excelente manera de contribuir a un repositorio de forma independiente al realizarle un fork.

En Github, el centro para el seguimiento de todos los errores son los **Issues.** A pesar de que son principalmente para el seguimiento de errores, también es útil utilizar Issues de las siguientes maneras:

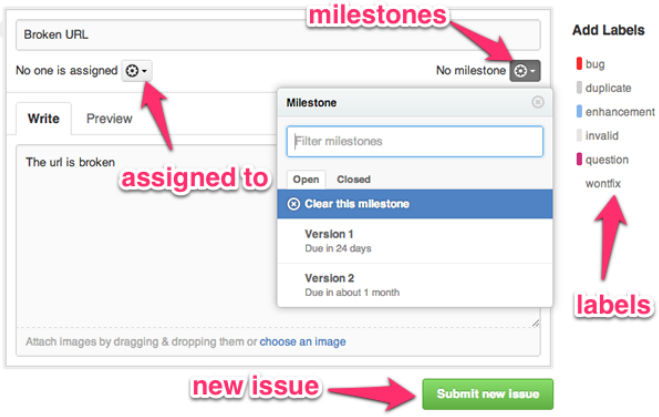
(Bugs) Errores: Cosas que obviamente están rotas y necesitan correcciones
(Features) Características: nuevas e increíbles ideas para implementar
(To do list) Lista de tareas: una lista de verificación de elementos para completar

Exploremos algunas de las características de Issues:

(Labels)Etiquetas: son categorías de colores para cada Issue. Son útiles para filtrar Issues en consecuencia.
(Milestone) Hitos: son categorías de fecha que se pueden asociar con cada Issue y son útiles para identificar en qué problemas se debe trabajar para la próxima versión. Además, dado que los hitos están conectados a Issue, automáticamente actualiza la barra de progreso al cerrar cada problema asociado.
(Search) Búsqueda: autocompleta la búsqueda de Issue e Milestone.



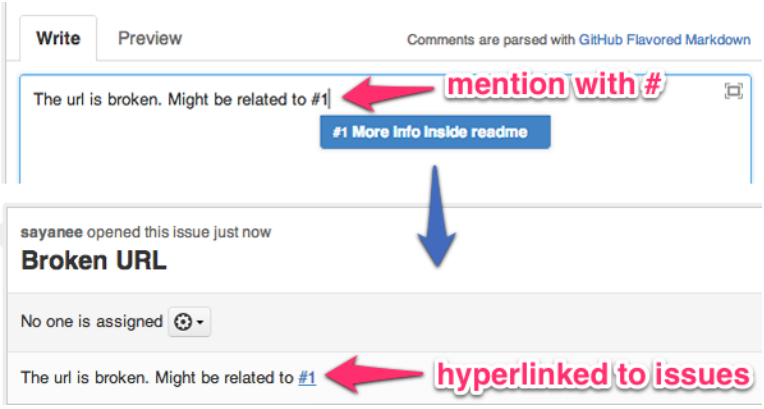
Assignment (Asignación): Cada problema se puede asignar a una persona a cargo para solucionar el problema. Es otra característica útil para ver en qué deberíamos estar trabajando.



(Auto-close) Cerrar automáticamente: Confirmar mensajes (commit) con Fixes/Fixed o Close/Closes/Closed # [issue-number] cerrará automáticamente el problema.

$ git add .
$ git commit -m "corrected url. fixes #2"
$ git push origin master

(**Mentions**)Menciones: cualquier persona también puede dejar una nota simplemente indicando # [issue-number] en sus mensajes. Debido a que los números del problema están hipervinculados, esto hace que sea realmente fácil mencionar problemas relacionados durante la discusión.



Analytics

Es claro que podemos unir estrechamente nuestra lista de tareas y actualizaciones a nuestras confirmaciones de código (commit).

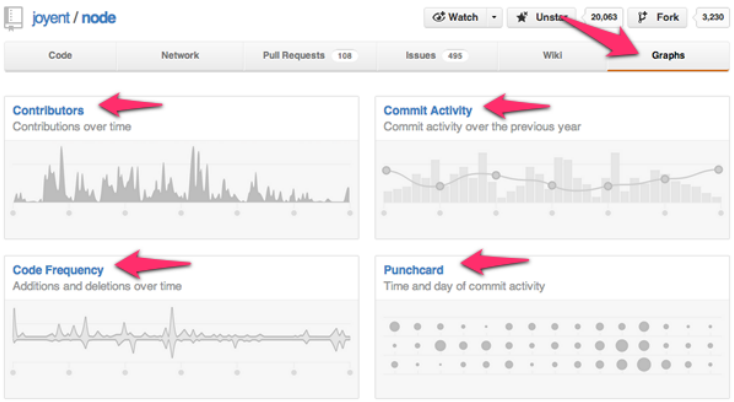
Hay dos herramientas que dan una idea de un repositorio: Graphs y Network. Github Graphs proporciona una visión de los

colaboradores y commits detrás de cada depósito de código, mientras que Github Network proporciona una visualización de cada contribuyente y sus confirmaciones en todos los repositorios donde se ha realizado un fork. Estos análisis y gráficos se vuelven muy poderosos, especialmente cuando se trabaja en equipo.

(Graphs) **Gráficos**

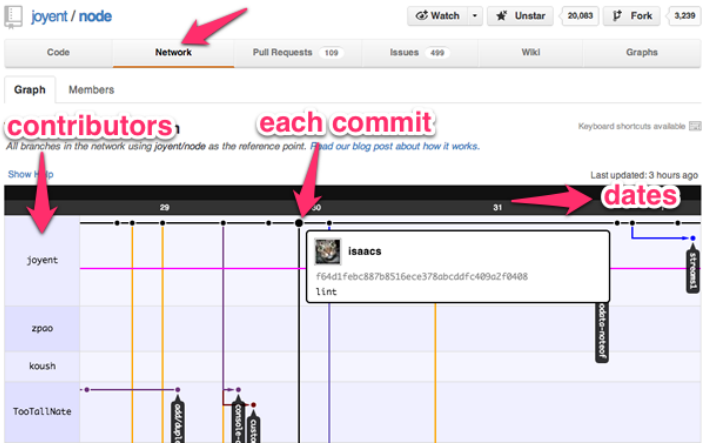
Los gráficos proporcionan análisis detallados como:

Colaboradores: ¿Quiénes fueron los contribuyentes? ¿Y cuántas líneas de código agregaron o eliminaron?
Actividad de commit: ¿En qué semanas tuvieron lugar los commit el año pasado?
Frecuencia de código: ¿Cuántas líneas de código se han subido (commits) a lo largo de todo el ciclo de vida del proyecto?
Punchcard: ¿en qué momentos del día suelen realizarse los commits?.



Network

Github Network es una poderosa herramienta que le permite ver los commits de cada colaborador y cómo se relacionan entre sí. Cuando miramos el visualizador en su totalidad, vemos cada commits en cada rama de cada repositorio que pertenece a una red.



Gestión de proyectos

Si bien Github Issues tiene capacidades de administración de proyectos con Issues and Milestones, algunos equipos pueden preferir otra herramienta debido a otras características o flujo de trabajo existente. En esta sección, veremos cómo podemos vincular Github con otras dos herramientas populares de administración de proyectos: Trello y Pivotal Tracker. Con los de servicio de Github, podemos automatizar la tarea de actualización con commits, issues y muchas otras actividades. Esta automatización ayuda no solo a ahorrar tiempo, sino también a aumentar la precisión de las actualizaciones para cualquier equipo de desarrollo de software.

Github y Trello

Trello proporciona una forma simple y visual de administrar tareas. Utilizando las metodologías de desarrollo ágil de software, las tarjetas Trello pueden emular un simple tablero Kanban virtual. Como ejemplo, crearemos automáticamente una tarjeta Trello cada vez que se realice un pull request utilizando los servicio Github (hooks). Repasemos los pasos!

Abre una cuenta en Trello si aún no tiene unaa y cree una nuevo tablero (board) de Trello.

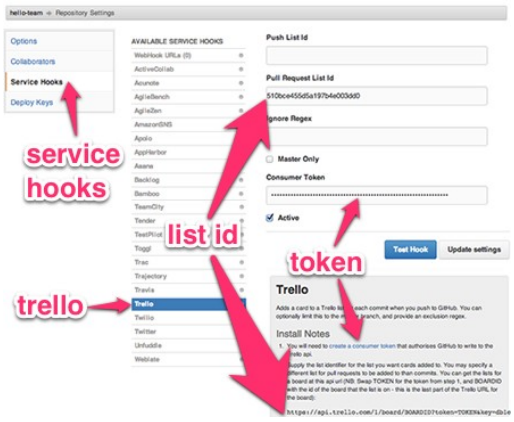
Ir al repositorio de Github> Configuración> Service Hooks> Trello

Consigue el TOKEN en Install Notes # 1 con el hipervínculo proporcionado para la autenticación.

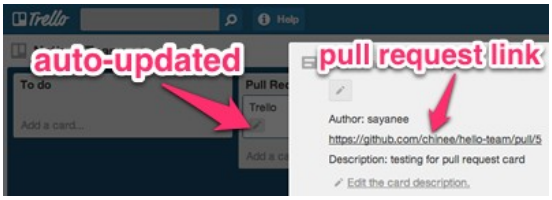
Debajo de Install Notes #2, utiliza la URL que se proporciona para generar una estructura con formato json que nos proporciona la lista id para cada tarjeta Trello. BOARDID forma parte de la URL cuando visitamos el tablero en https://trello.com/board/[BOARD-NAME]/[BOARDID].

TOKEN se puede crear con el hipervínculo proporcionado en Install Notes # 1.

Regresa a los servicios de Github, ingresa la identificación de la lista y el token. Realiza Active, Test Hook y estaremos listos para recibir actualizaciones automáticas cada vez que haya pull request.



La próxima vez que haya un pull request, la tarjeta Pull Request de Trello tendrá automáticamente un nuevo elemento.

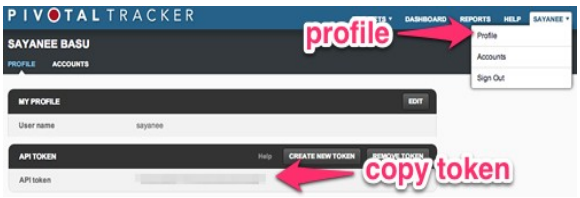


Github y Pivotal Tracker

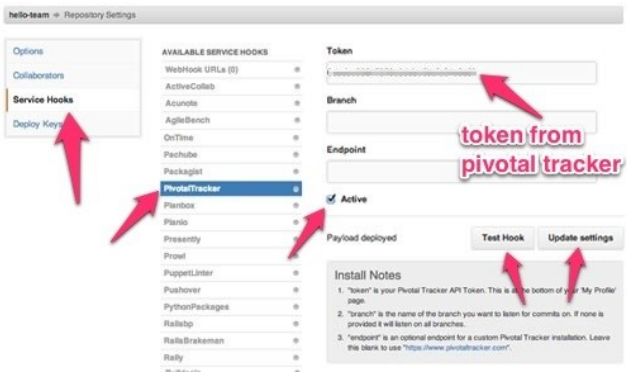
Pivotal Tracker es otra herramienta ágil de administración de proyectos ágil donde la planificación basada en historias permite al equipo colaborar fácilmente al reaccionar instantáneamente a varios cambios y avances del proyecto. Basado en el progreso actual del equipo, también puede crear gráficos para analizar la velocidad del equipo, la iteración, el lanzamiento, etc. En este breve ejemplo, entregaremos automáticamente una historia uniéndolo a un commit de Github.

Creamos un nuevo proyecto en el Pivotal Tracker con una nueva Story que debe entregarse.

Ir a Profile > API Token . Copia el token del API dado



Regresa al repositorio de Github> Settings> Services Hooks> Pivotal Tracker. Pega el token, marque Active y haga clic en Update Settings. Estamos todos listos para entregar automáticamente Pivotal Tracker Stories con Commits de Github!.



Finalmente, realizaremos commits a nuestros cambios y agregaremos la identificación del tracker con al mensaje de confirmación (commit) con el formato git commit -m "message [deliver # tracker_id]"

$ git add.

```
$ git commit -m "Github y Pivotal Tracker implementados [entrega # 43903595]"
$ git push
```

Ahora, cuando volvamos al Pivotal Tracker, veremos que la historia se ha entregado automáticamente con enlaces al commit de Github que muestra como el archivo cambia.

Con estos ejemplos de Trello y Pivotal Tracker, está claro que podemos unir estrechamente nuestra lista de tareas y actualizaciones a nuestros commits de código. Este es un gran ahorro de tiempo cuando se trabaja en equipo, y mejora la precisión al vincular las tareas a exactos commit. La buena noticia es que si ya usa otras herramientas de administración de proyectos, como Asana, Basecamp y otras, también puede crear Service Hooks de manera similar. Si no hay Service Hooks para su herramienta de administración de proyectos actual, puede incluso crear uno!.

Integración continua

La integración continua (CI) es una parte importante de todos los proyectos de desarrollo de software que trabajan con equipos. CI se asegura de que, cuando un desarrollador verifique su código, una compilación automatizada (incluidas las pruebas) detecte los errores de integración lo más rápido posible. Esto definitivamente reduce los errores de integración y hace que la iteración rápida sea mucho más eficiente. En este ejemplo, veremos cómo se puede usar Travis CI con Github para que CI detecte errores y recomiende fusiones cuando pase todas las pruebas.

Configuración de Travis CI

Utilizaremos un proyecto simple de "hello-world" para node.js con grunt.js como la herramienta de compilación para configurar un proyecto de Travis CI. Aquí están los archivos en el proyecto:

El archivo hello.js es el nodo proyecto . Aquí omitiremos deliberadamente un punto y coma para que no pase la herramienta de compilación grunt para el linting:

```
var http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World in Node!\n') // without semicolon, this will not pass linting
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

package.json denotes the dependencies:

```
{
  "name": "hello-team",
  "description": "A demo for github and travis ci for team collaboration",
  "author": "name <email@email.com>",
 "version": "0.0.1",
 "devDependencies": {
  "grunt": "~0.3.17"
},
"scripts": {
  "test": "grunt travis --verbose"
   }
}
```

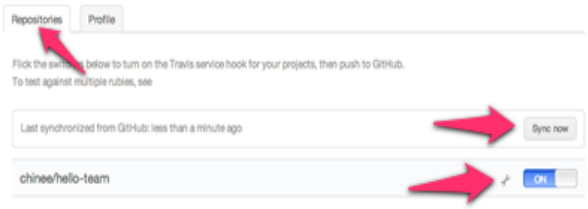
La herramienta de compilación grunt.js solo tiene una tarea (linting) solo por simplicidad:

```
module.exports = function(grunt) {
  grunt.initConfig({
  lint: {
  files: ['hello.js']
   }
});
grunt.registerTask('default', 'lint');
grunt.registerTask('travis', 'lint');
};
```

.travis.yml es un archivo de configuración de Travis que hará que Travis ejecute nuestras pruebas:

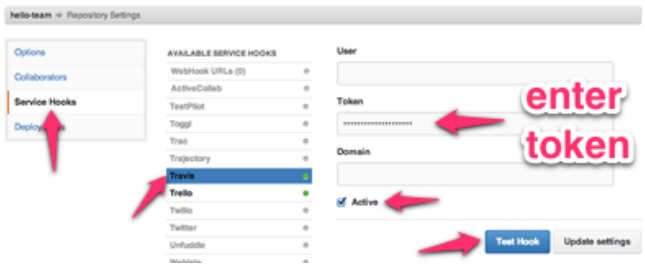
```
language: node_js
   node_js:
- 0.8
```

A continuación, inicie sesión en Travis con su cuenta de Github y active el enlace del repositorio debajo de la pestaña del repositorio.



Si el paso anterior no desencadena la construcción, entonces tendremos que configurar manualmente el servicio. Para configurarlo manualmente, copie el token debajo de la pestaña del perfil.

Vuelva al repositorio de Github para configurar los servicios de Github de Travis con el token.

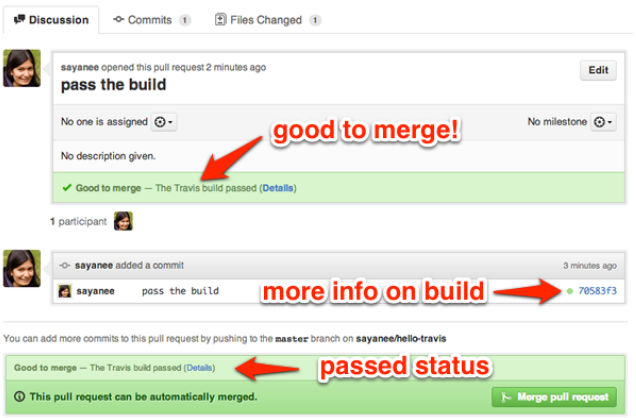


Por primera vez, necesitamos hacer un git push manual para activar la primera compilación de Travis y si todo está bien, simplemente visita http://travis-ci.org/[nombre de usuario]/[repo-name] para ver la compilación.

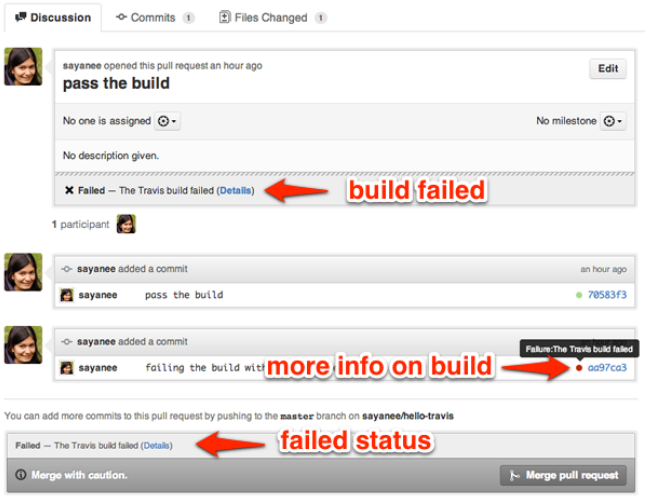
Travis CI con pull request

Anteriormente, sin ningún elemento CI en el proceso de un pull request, los pasos fueron algo como esto (1) envio de pull request extracción (2) fusión (merge) (3) prueba para ver si pasa o no. Con Travis CI conectado a los pull request, podremos invertir los pasos 2 y 3, lo que aumentará aún más la toma de decisiones rápida sobre si es bueno realizar una fusión o no con Travis, dándonos el estado con cada pull request. Veamos cómo hacer que eso suceda.

Envíe un pull request pasando una compilación y Travis hará su magia para hacerle saber que es bueno fusionarse incluso antes de la fusión.



Si el pull request falla la compilación, Travis también nos alertará.



Si hacemos clic en el botón de alerta roja, también se vinculará a la página de Travis para mostrarnos los detalles de la compilación.

Travis CI con Github es inmensamente útil para los equipos debido a compilaciones automáticas y notificaciones inmediatas.

Ciertamente hace que el ciclo de corrección de errores sea mucho más corto. Si se está utilizando Jenkins, otra herramienta común de CI, sí, también puede configurar los servicio de Github de manera muy similar.