Cómo configurar un repositorio para ser alojado en nuestros servidores Git privados y público (GitHub)

No podemos, nunca, confiar todo en los servicios de alojamiento público "gratuito" como GitHub o GitLab: aunque mantengamos una copia de nuestro código y proyectos en nuestro propio ordenador de mesa y/o portátil **siempre debemos tener nuestros respectivos servidores Git para nuestro uso privado, es nuestro derecho.**

Introducción

En varios artículos anteriores hemos expuesto cómo trabajar con Git, el sistema de control de versiones, recomendamos leerlos si usted lector o lectora se está iniciando en este sector de trabajo colaborativo remoto:

- **Git**: en este artículo hablamos de la historia del protocolo Git y sus comandos básicos, un poco largo el artículo, lo escribimos al estilo de publicación para un futuro libro al respecto.
- **Git**: Los tutoriales rápidos son muy prácticos pero generalmente está escritos en idioma inglés, por eso realizamos la traducción de uno de ellos, publicado por la empresa de alojamiento de VPS DigitalOcean (al momento de escribir estas líneas le pagamos a esa empresa para alojar esta página web).
- **Git**: también tradujimos cómo configurar nuestro propio servidor Git privado, aunque es muy parco el trabajo original, hemos agregado sustanciales mejoras al mismo en nuestro idioma.
- **SSH**: obligatoriamente necesitamos crear llaves privadas y públicas y eso está detallado en esta entrada; además una traducción completa sobre cómo manejar dichas -muy importantes- claves para nosotros.

Objetivos

- Clonaremos el repositorio de un proyecto existente y alojado en GitHub por vía HTTPS con el programa git.
- Con el proyecto clonado mejoramos el mismo agregando y modificando ficheros.
- Ya en nuestro equipo, renombraremos el identificador que apunta hacia donde se aloja dicho proyecto *en el protocolo Git*, además de cambiar el protocolo de conexión de **HTTPS**

a **git**.
- Agregaremos un nombre identificador de ubicación hacia nuestro servidor Git privado en otro equipo ubicado dentro de nuestra LAN (equipo real, la idea es sacarlo hacia otro disco duro físico).
- *Acometer* y preparar los cambios realizados, guardar en ambos repositorios (privado y público).

Inicio

Lo anterior suena muy sencillo pero *«del dicho al hecho hay enorme trecho»;* además quiero establecer ciertos términos importantes, "terminología" básica.

En el libro Pro Git versión 2, el cual se encuentra traducido al castellano, emplean unos términos que modificamos ligeramente para facilitar su comprensión en nuestro idioma:

- **Proyecto**: son todos los archivos y subcarpetas que componen cualquier trabajo que hagamos (dibujos para una campaña publicitaria, planos de una fábrica, un software para facturación y control de inventario, etcétera) y que necesitemos dar seguimiento y respaldo en varios repositorios distintos.

- **Repositorio**: es un fichero con extensión **.git** que es manejado por la aplicación del mismo nombre y que recoge los ficheros y subcarpetas de un **proyecto** y sus cambios a través del tiempo (esto es muchísimo más complejo *pero quedemos con ese concepto bien sencillo*).

- Ficheros o archivos **sin seguimiento** "*untracked*", son archivos que:
    - Estaban antes de iniciar un repositorio git.
    - Son archivos nuevos que agregamos después de clonar un repositorio git.
    - Existen también ficheros sin seguimiento de manera explícita: *los archivos ignorados* (temporales o de respaldo: .tmp, .bak, etc.)

- Ficheros o archivos **sin modificación (con seguimiento)** "*unmodified*", son archivos que:
    - Se encontraban en el repositorio cuando lo clonamos y no hemos tocado para nada.

    - Los que agregamos al seguimiento o rastreo en el punto anterior y que ahora aparecen -obvio- como archivos "nuevos" "*new"* (incluso si los modificamos seguirán apareciendo como nuevos)

- Ficheros o archivos **modificados (con seguimiento)** "*modified*": todos los del punto anterior a los cuales hayamos agregado o quitado algún carácter o conjunto de caracteres, simplemente los llamaremos "modificados".

- Ficheros o archivos **acometidos** o "*staged*": acá viene las diferencias en nuestro idioma. El verbo **acometer** es reconocido como intentar por la fuerza, *pero en la ciencia de la ingeniería civil significa la unión o suministros, como por ejemplo en la acometida eléctrica o acometida telefónica, etc.*
    - Nuestra elección de tal término es para diferenciar claramente a qué nos referimos: guardar nuestros **modificados** y/o **nuevos** a nuestro repositorio local, en nuestro disco duro. En el libro Pro Git versión 2 lo traducen como *confirmados o confirmación pero, sinceramente, **¿confirmar qué?***
    - En cambio cuando decimos **acometidos** ya sabremos que están **preparados** para ser "guardados" en nuestros diferentes repositorios, privados o públicos; veamos el próximo paso.

- Cuando los **preparados** los **empujamos** "*push*" a el o los repositorios, pues tendremos guardada (*más bien sincronizada*, pero no tocaremos ese tema en este artículo) nuestra **acometida** o ***acometidas***; así ya estarán listos para que los otros participantes del proyecto puedan "copiar" nuestros aportes. Todos nuestros ficheros vuelven a estar "sin modificación" y listos para repetir el ciclo.

- En el siguiente gráfico que hicimos con **Inkscape** tomamos como base traducción del libro nombrado anteriormente pero con nuestros términos agregados.

[SVG: Ciclo de vida de los archivos mediante Git (formato SVG)]

Ciclo de vida de los archivos mediante Git (formato SVG)

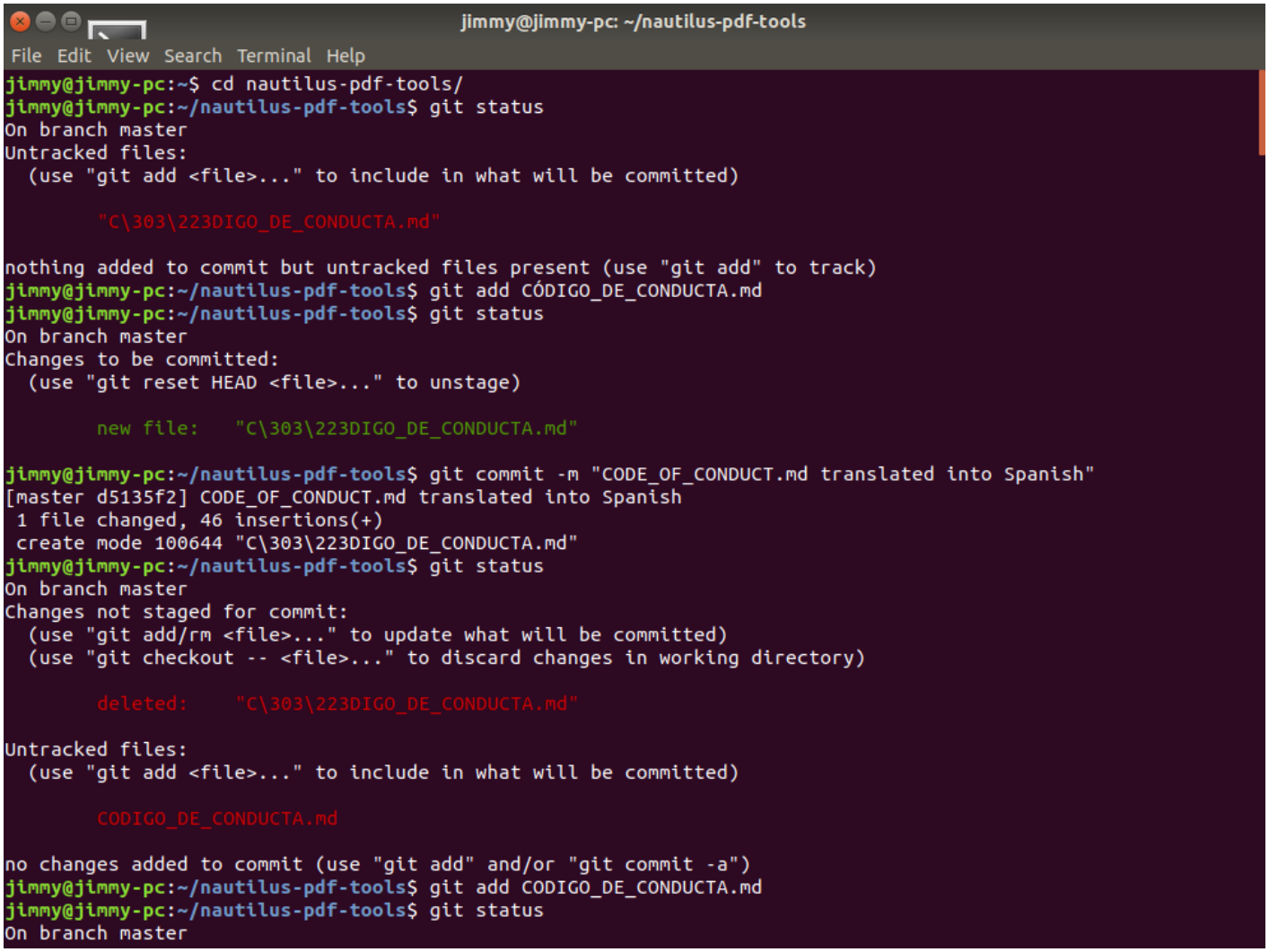
Clonado de repositorio desde GitHub

El repositorio que elegí para este artículo es la bifurcación ("*fork*") del excelente programa **nautilis-pdf-tools** cuyo autor es el señor Lorenzo Carbonell (@atareao en Twitter), software que utilizamos por acá para preparar los ejemplares de la Gaceta Oficial de la República Bolivariana de Venezuela. Dicho esto, abramos una ventana terminal, vamos a nuestra carpeta de usuario y descargamos una copia de la bifurcación:

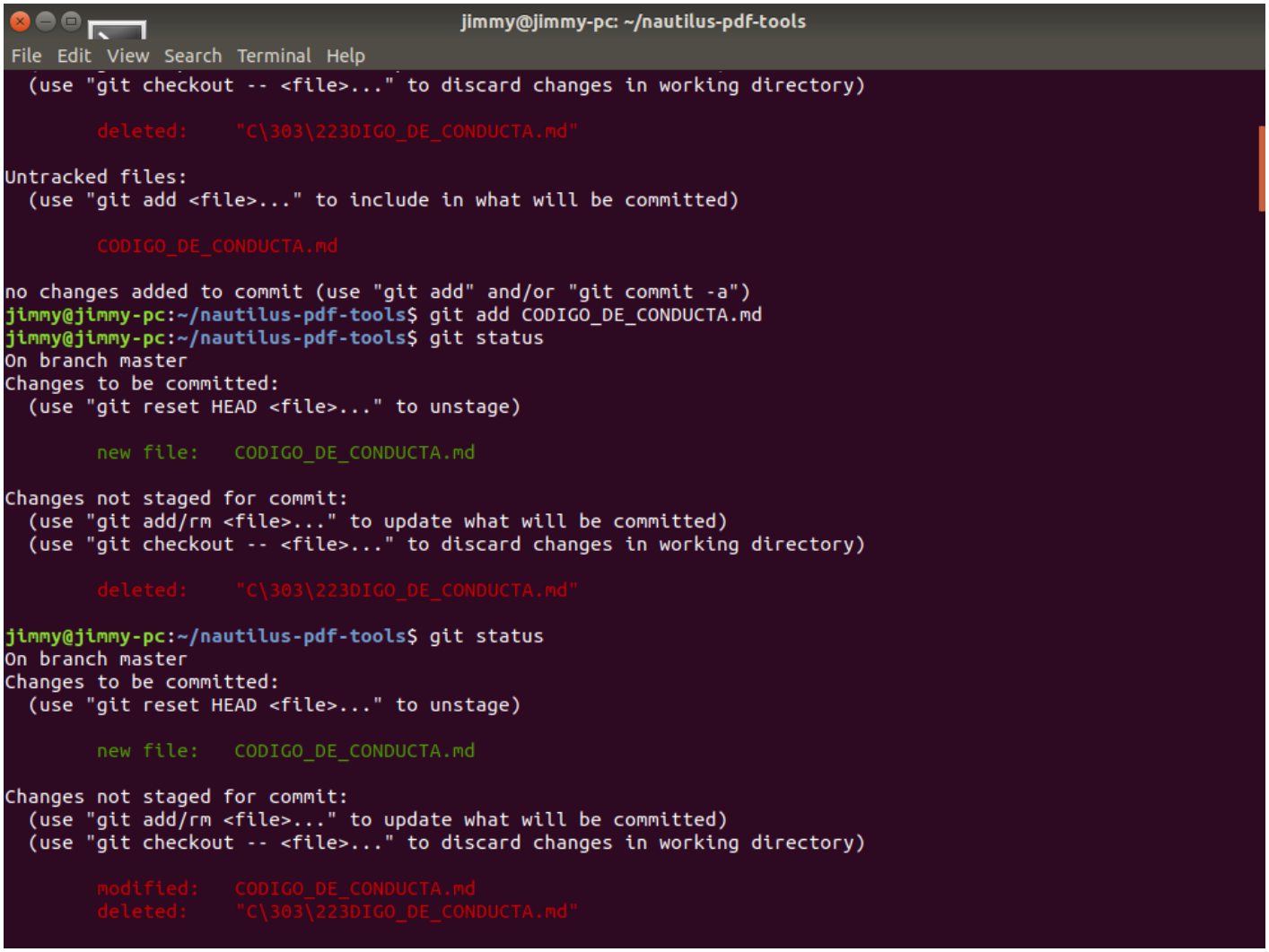
```
$ cd $ git clone https://github.com/ks7000/nautilus-pdf-tools.git
```

Agregando y modificando ficheros

Con la misma terminal de comandos abierta, esencialmente lo que hicimos fue entrar en la carpeta del proyecto, agregar un fichero con la traducción del archivo existente CODE_OF_CONDUCT.md pero con un acento en su nombre. Todo esto lo hicimos a propósito para que vean cómo se manejan las **acometidas** en Git, luego las podrán ver en GitHub; acá dejamos dos capturas de pantallas con lo que hicimos ***no explicaremos más ya que no es el tema principal para esta entrada.***



Archivo nuevo en proyecto nautilus-pdf-tools



Archivo renombrado y modificado en proyecto nautilus-pdf-tools

Remoción y agregado del nombre de servidor Git

De manera predeterminada la palabra clave **"origin"** es agregada a cada nuevo repositorio git y, curiosamente, pocos y pocas se preocupan por cambiar este nombre. Este nombre es el identificador con la URL que indica a cual servidor Git deberemos empujar nuestras acometidas para así estar disponible para el resto de los programadores y programadoras (y si el servidor es público, como el caso de GitHub, para el resto del mundo).

```
git remove origin git remote -v
```

Con la primera línea lo eliminamos y con la segunda línea consultamos en qué estado queda nuestro repositorio local: apuntando hacia ningún servidor Git.

Dos aclaratorias adicionales:

- Con el comando **git** clone bien podemos aprovechar de cambiar el nombre (por ejemplo, si lo descargamos de GitHub, pues usamos el nombre **github**).
- Si hacemos lo del punto anterior, de todas maneras no podremos cambiar su método de conexión, HTTPS en nuestro caso; por ello mejor removemos todo en un solo paso.

Añadiendo un nuevo origen con un nuevo protocolo

Servidor Git público: GitHub

Que en realidad vamos a establecer el mismo servidor Git, **GitHub**, *pero cambiando el protocolo de comunicación a git con SSH* (con la segunda línea visualizaremos el estado):

```
git remote add github git@github.com/ks7000/nautilus-pdf-tools.git
git remote -v
```

¡Atención! En nuestra cuenta en GitHub necesitaremos agregar nuestra llave pública especialmente creada para GitHub.

Creando un par de llaves para nuestra cuenta GitHub

Aunque ya escribimos una artículo al respecto, no explicamos allá sobre cómo agregar una *frase de contraseña* y especificar un nombre diferente para el archivo (lo coloreamos):

```
ssh-keygen -t rsa -C "Para empujar a GitHub" -f github_llaves
```

Si desea establezac una *frase de contraseña* que ustedes gusten a fin de proteger dicha llave privada (pero será preguntada luego al conectarse) y será guardado en un fichero cuyo nombre será **github_llaves** (noten también que agregamos un comentario con el parámetro "**-C** ").

Ahora movemos dichos archivos a la carpeta **~/.ssh** (carpeta **ssh** oculta en nuestro directorio de usuario):

```
mv github_llaves ~/.ssh
mv github_llaves.pub ~/.ssh
```

Ahora debemos modificar el archivo de configuración, usaremos nano:

```
nano ~/.ssh/config
```

Y agregaremos al final el siguiente contenido:

```
Host github.com
HostName github.com
IdentityFile ~/.ssh/github_llaves
User git
```

Esto quiere decir que cuando conectemos con github.com el usuario **git** utilizará las llaves llamadas **github_llaves**, guardamos el fichero y salimos del editor de texto y tecleamos lo siguiente:

```
cat ~/.ssh/github_llaves.pub
```

¡Advertencia! Asegúrense de que tenga la extensión **.pub** que significa que es la llave pública, nunca, en ningún caso entreguen su llave privada a nada ni nadie. En el archivo de

configuración que editamos, al contrario, indicaremos que utilice la llave privada para "descifrar" nuestra llave pública cuando nos conectemos por medio del usuario **git**.

En el contenido que visualizarán por pantalla se mostrará nuestro comentario a la llave, noten bien

Agregando solamente nuestra llave pública a GitHub

- Deberemos iniciar sesión en nuestra cuenta GitHub, si aún no lo hemos hecho (contraseñas, factores de autenticación, etcétera).
- Luego iremos a la sección de configuración de llaves: https://github.com/settings/keys y haremos clic en agregar nuestra **llave pública**.
- Solicitarán nuestra contraseña en GitHub, enviarán correo electrónico avisando, entre otras medidas de seguridad: **este paso es crucial, debemos siempre estar atentos que no se agreguen credenciales que nosotros no hayamos autorizado. Ojo.**

Agregando la huella digital de GitHub

Para saber si realmente estamos conectando nuestro repositorio a GitHub debemos saber la *huella digital* (no confundir con la *huella dactilar* de nosotros los humanos) de dicho sitio web. Para este caso particular podemos acudir, por HTTPS a este enlace y visualizarlo, anotarlos y confirmar que son los mismos cuando realicemos la primera conexión... Sí, esto es un tanto anticuado, y es positivo, existe un método mejor, más exacto y rápido.

En realidad esto lo podremos realizar con cualquier servidor SSH que tengamos que conectar

```
ssh-keyscan -p 22 -4 -H github.com >> ~/.ssh/known_hosts
```

- Usamos el comando **ssh-keyscan**.
- El parámetro **-p** permite especificar el puerto, generalmente es el 22.
- El parámetro **-4** solo busca direcciones IPv4, acá en Venezuela por ahora no tenemos direcciones IPv6.
- El parámetro **-H** consulta todos los anfitriones que tenga **github.com**, podemos dejar un espacio y escribir más dominios, los que necesitemos conectar.
- Con " **>>** " agregaremos la(s) respuestas a un fichero, que en nuestra cuenta de usuario está ubicado en **~/.ssh**
- Recordar que pueden haber varios servidores SSH, cada uno con su dirección IP propia y que serán agregados sin mayor dilación a nuestro fichero de anfitriones conocidos.

Servidor Git privado

Creando nuestro Servidor Git privado

A ello dedicamos un artículo completo, traducción de lo publicado en DigitalOcean, acá colocaremos de una manera resumida:

- Usaremos otra llave distinta llamada **id_rsa.pub** para conectarnos a una máquina real con la dirección IP 192.168.1.27 y lo enviaremos por SSH con nuestra contraseña de usuario **git** *en esa máquina remota de nuestra LAN.*

```
$ cat ~/.ssh/id_rsa.pub | ssh git
@192.168.1.27 "cat >> ~/.ssh/authorized_keys"
```

También debemos crear un repositorio vacio con el nombre **nautilus-pdf-tools** *pero con el nombre de usuario git en Acer:*

```
ssh git@192.168.1.27 "git init --bare nautilius-pdf-tools.git"
```

Agregando nuestro servidor Git privado a nuestro repositorio local

Al igual que hicimos con GitHub, deberemos agregar un nombre, un protocolo y URL, como nuestro caso el ordenador auxiliar local es marca Acer usaremos ese nombre distintivo:

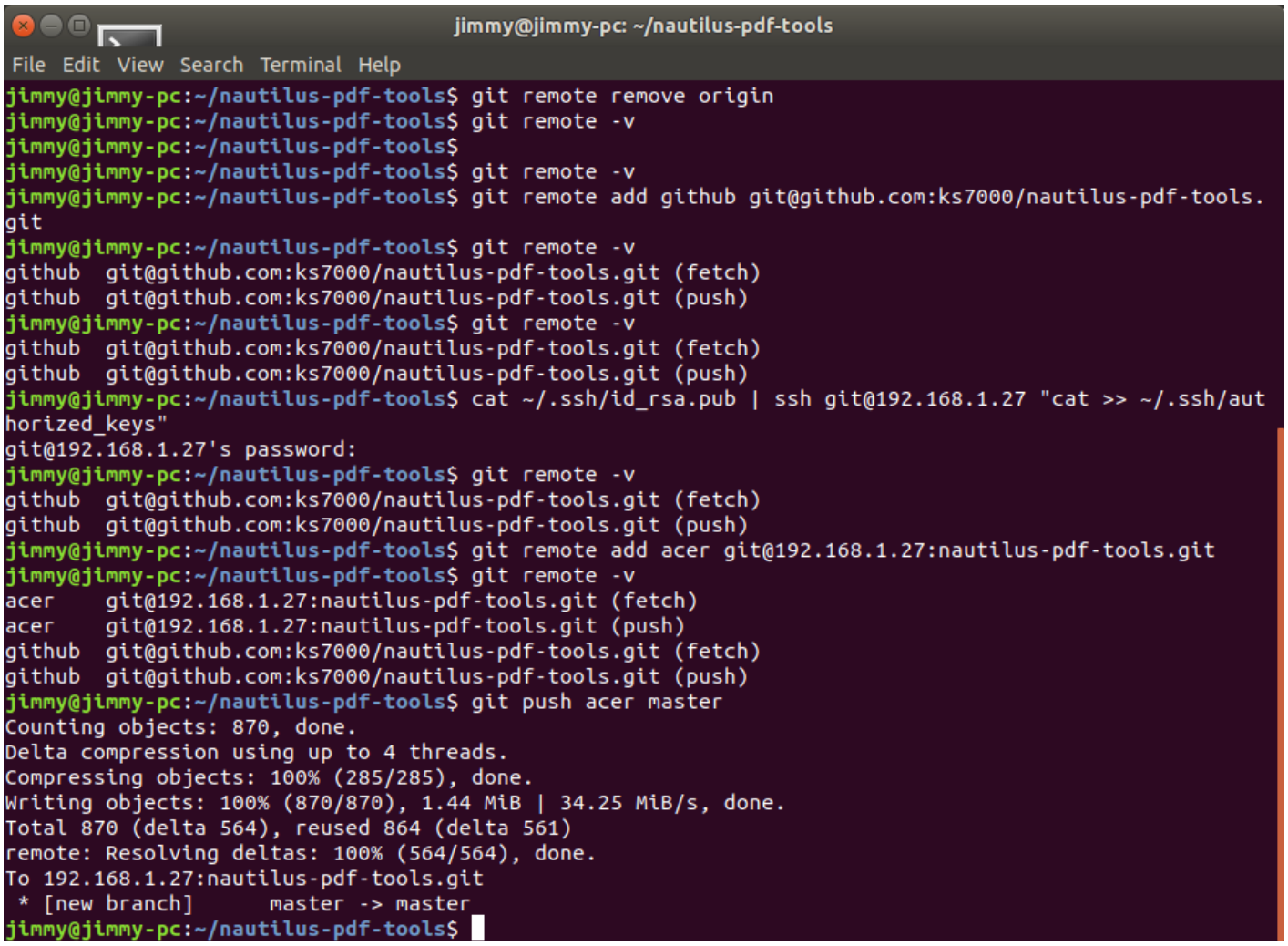
```
git remote add acer git@192.168.1.27:nautilus-pdf-tools.git
git remote -v
```

De esta manera, al ejecutar la segunda línea, observaremos el siguiente resultado:

```
acer git@192.168.1.27:nautilus-pdf-tools.git (fetch)
acer git@192.168.1.27:nautilus-pdf-tools.git (push)
```

```
github git@github.com:ks7000/nautilus-pdf-tools.git (fetch)
github git@github.com:ks7000/nautilus-pdf-tools.git (push)
```

Lo mejor de todo esto es que aunque nuestra explicación es larga para que puedan aprender, en la ventana terminal son unas cuantas líneas de código resumidas en esta captura de pantalla:



«git remote remove origin» y «git remote add»

Acometiendo y empujando

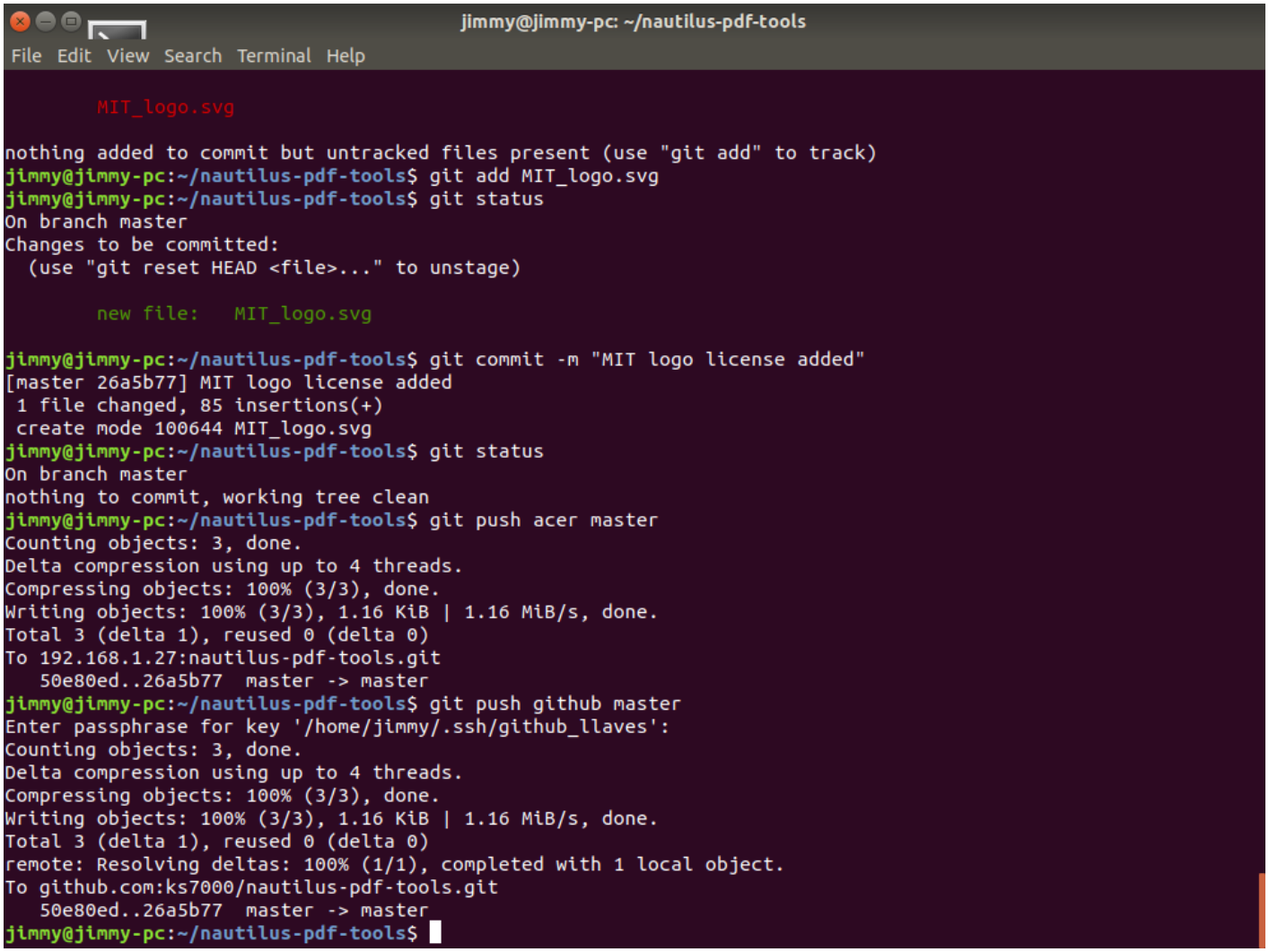
Como una prueba final agregamos el logotipo del Instituto Tecnológico de Massachusetts, donde crearoon la licencia que rige este proyecto. una vez tenemos todo listo para *empujar* a los dos repositorios, procedemos:

```
git push acer master
```

Lo cual hará sin problema, mostrando los resultados, luego:

```
git push github master
```

Si establecimos una *frase de contraseña* para crear la clave pública para GitHub, introduzcan dicho valor y vean el resultado gráficamente:



«git push acer master» y «git push github master»

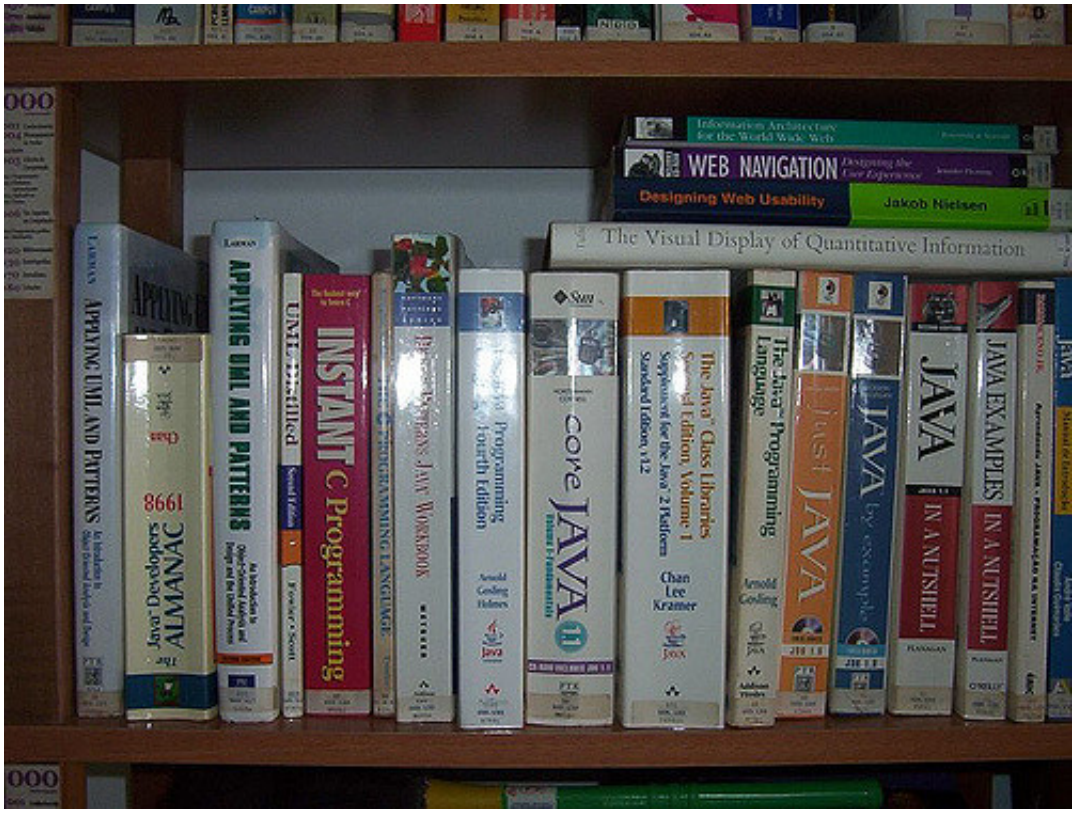
Visualización en GitHub

Si navegan a nuestra página de acometidas en GitHub, para el proyecto en cuestión, podrán ver para los días sábado 12 y domingo 13 de octubre de 2019, los diversos cambios que hicimos escribiendo este artículo, ¡disfrutamos mucho de ello!

Conclusión

Hemos aprendido a llevar un proyecto, controlado por **Git** en dos repositorios fuera de nuestro entorno de programación; **¿posibles usos?**

- Aún cuando no tengamos conexión a Internet podremos sacar fuera de nuestro ordenadores personales nuestro trabajo, una protección adicional.
- Dado el caso bloqueen, vendan o quiebre GitHub siempre podremos seguir trabajando con nuestro proyecto, al menos si conectamos en nuestra LAN a los colaboradores y colaboradoras (ponemos antenas marcas Ubiquiti en un rango de 12 kilómetros de nuestras oficinas).
- Podremos arrendar un VPS para colocarlo con nuestro repositorio privado en Internet y para mayor facilidad podremos adquirir un dominio para crear un subdominio con nuestro repositorio: en este caso tendríamos tres servidores Git con nuestros proyectos («los ramanes lo hacen todo por triplicado»).
- ¿Algún inconveniente? Recordemos bien que con este esquema nuestros compañeros y compañeras de código siempre se conectarán a GitHub y ese seguirá siendo el repositorio principal, cuando subamos a nuestro repositorio privado **nunca encontraremos "choques de código", es decir, que dos personas hayan modificado el mismo archivo o ficheros mientras nosotros también.** Esos casos quedarán para nuestros futuros artículos y ¿quién sabe? algún día escribiremos y venderemos un manual rápido para Git ¿Qué opinan?



Fuentes consultadas

En idioma castellano

- «».
- «».
- «».

En idioma francés

- «».
- «».

- «».

En idioma inglés

- «SSH Key: “Permissions 0644 for 'id_rsa.pub' are too open.” on mac».
- «SSH Host Key Protection».
- «Configure Jenkins to log into Github account».

Crédito de la imagen Gerd Altmann, trabajo, licencia de uso:Pixabay