the original Hacker

creado por EUGENIA BAHIT

Jugando con la inteligencia

número 9

Woman Eyes creado por Mourad Mokrane - Silueta de Mujer creado por Leonardo B. Cunha

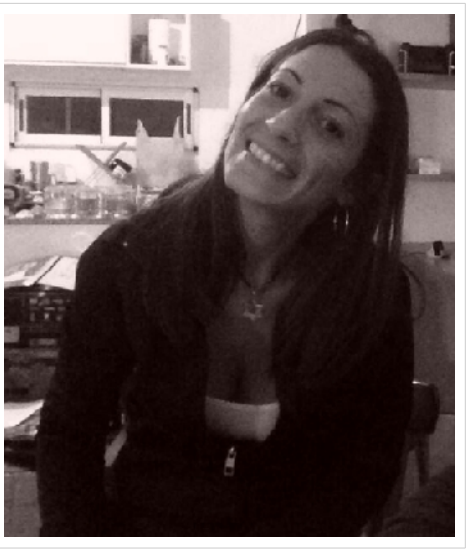
AUTHORS AGAINST DRM

SIN DRM

GNU
run free run GNU

THE ORIGINAL HACKER
SOFTWARE LIBRE, HACKING y PROGRAMACIÓN, EN UN PROYECTO DE

EUGENIA BAHIT



@eugeniabahit

GLAMP HACKER Y PROGRAMADORA EXTREMA

Hacker especializada en Programación eXtrema e Ingeniería Inversa de código sobre GNU/Linux, Apache, MySQL, Python y PHP. EUGENIABAHIT.COM

Docente e instructora de tecnologías GLAMP CURSOS.EUGENIABAHIT.COM CURSOSDEPROGRAMACIONADISTANCIA.COM

Miembro de la Free Software Foundation FSF.ORG y The Linux Foundation LINUXFOUNDATION.ORG.

Creadora de python-printr, Europio Engine, JackTheStripper. Vim contributor. Fundadora de Hackers n' Developers Magazine y responsable editorial hasta octubre '13.

MATERIAL DE
LIBRE DISTRIBUCIÓN
HECHO EN LA REPÚBLICA ARGENTINA

Buenos Aires
Septiembre 2014

ÍNDICE DE LA EDICIÓN NRO9

Donar

Guía para ser un buen estudiante de sistemas: las 12 reglas del aprendiz de Hacker

*— Eugenia Bahit agradece a **Hugo** (@huguidugui) por la **revisión ortográfica** de este artículo*

Ser un buen estudiante no implica ser «visto» como un buen estudiante. Ser un buen estudiante, solo requiere saber administrar las energías invertidas en aprender.

Muchas personas podrían creer que un «buen estudiante» es aquel que «cae bien al docente», pero nada más lejos de la realidad.

Cuando hablamos de «buen estudiante» no nos referimos a aquel que según terceras personas resultaría ser un buen alumno, sino a «**aquel que sabe cómo mejor invertir sus energías en el tiempo del que dispone para sacar el máximo provecho de lo que desea aprender**».

Aquí podrán encontrar una serie de **normas o preceptos que los ayudarán a ser mejores estudiantes**.

Regla №1: no estudies estando cansado, traumatizado o mal humorado

Neurológicamente, el cerebro absorbe y asimila mejor la información que le llega estando en buen estado anímico. Toda la información que ingrese al cerebro bajo una situación poco agradable (ya sea cansancio, estrés, mal humor u otra emoción negativa), por asociación indirecta, será enlazada a «algo negativo». De esta forma, si acabas de recibir un telegrama de despido de tu trabajo y éste, te generase -más allá de la sorpresa y lo inesperado- un gran malestar, si inmediatamente te explicasen qué es un algoritmo, tu cerebro, indirectamente, asociaría algoritmo a sorpresa desagradable.

Regla №2: estudia con tu cerebro pero «mentalízate» con tu espíritu

Esta regla viene de la mano de la anterior. No basta con solo no estar de mal humor a la hora de estudiar. Es sumamente importante que sea capaz de generarte a ti mismo, buenas sensaciones antes de entrar a la clase. Al momento de comprender y asimilar lo nuevo, utilizarás la razón, pero te mentalizarás previamente para llenarte de un espíritu positivo antes de comenzar a emplear la razón. Por los mismos motivos que los mencionados en la regla №1, esto ayudará a asimilar mejor la nueva información.

Regla Nº3: no faltes a tus clases y sé puntual

La hora de ir a clases debe ser sagrada, irrenunciable e impostergable. Pues cada clase es tu única oportunidad de conectar con tu maestro y aprender de él o ella. Si supieses que tu siguiente clase será tu última clase ¿llegarías tarde o cancelarías la clase? Seguramente no. Y eso es porque realmente valoras el hecho de poder aprender de tus maestros.

Regla Nº4: en la clase, concéntrate solo en la clase

Esto significa que debes tener tus 5 sentidos puestos en la clase. En el aula, apaga el móvil y siéntate en un lugar donde nada pueda distraerte. Ni hablar de ¡no utilizar reproductores de música! Si grabarás la clase, asegúrate de hacerlo con un dispositivo que no pueda distraerte. Si funciona a pilas/batería, asegúrate que estén cargadas. Si requiere de cinta o unidad de almacenamiento asegúrate de que tenga espacio suficiente, etc.

Si la clase es *online*, busca un lugar tranquilo y silencioso. No escuches radio, TV, ni música mientras cursas. Si cursas en casa estando en familia, avisa a tu cónyuge o padres (según el caso) que necesitarás estar un tiempo a solas para tomar una clase *online*, de manera que nadie te distraiga con risas, anécdotas, etc. Si cursas en tu lugar de trabajo, asegúrate de que se te asigne un lugar aislado para tomar tu clase y que nadie te moleste en ese momento. Si alguna de las anteriores circunstancias no te es posible cumplir, es preferible optar por cursos presenciales ya que hacerlo a distancia no será para nada productivo.

Regla Nº5: siempre prefiere ejercitar lo aprendido que leer o teorizar

A no ser que tu maestro te indique que debes leer algo en particular, tu primera reacción, siempre debe ser ejercitar aquello de lo que hayas hablado en clase. La psicopedagogía diferencia entre el 'saber' y el 'saber hacer' considerando que solo cuando se llega a este último se ha aprendido realmente[1]. Nadie aprende a caminar teorizando sobre los pasos que algún día deberá dar. Simplemente, una vez comprendido qué es lo que se debe hacer, se lo intenta hacer repetidamente hasta que logre hacerse de forma natural.

Regla Nº6: mantén fluidez y continuidad

De forma tal, que entre una clase y otra no quede tiempo muerto. Si has logrado estudiar todo lo que debías estudiar en menos tiempo del esperado, deberás administrar mejor tu tiempo. Si te ha sobrado el tiempo, repite y revisa tus ejercicios o relee lo que el docente te indicó, pero utiliza el tiempo sobrante en asimilar mejor lo que hayas aprendido. Si estás estudiando con ayuda de terceras personas, mezclar mecanismos autodidactas puede ser contraproducente. Entonces, cuando te sobre tiempo, no intentes incorporar nuevos conocimientos ya que correrás el riesgo de mal interpretar lo que ya hayas aprendido.

1 Según taxonomía cognitiva de Bloom.

Regla Nº7: confía en tus maestros

La confianza en tus maestros es la clave fundamental para aprender lo que tus maestros te enseñen. Si no tienes confianza profesional en tus maestros, estarás invirtiendo tus energías en cuestionar lo que tus maestros te enseñen y terminarás no aprendiendo nada. Si no confías en tu maestro, busca uno en el que confíes. Si no confías en la institución en la que estudias, no intentes buscar un maestro en quien confiar dentro de esa institución. Búscalo fuera.
Pero ten en cuenta que si confías en tu maestro, confiarás en lo que te enseñe. Y si confías en lo que te enseña, solo deberás ejercitarlo para haberlo aprendido.

Regla Nº8: pregunta todo lo que no entiendas sin interrumpir a tu maestro

Cuando estudies a solas, anota todas las dudas que tengas para luego consultarlas a tu maestro. Cuando estés en la clase y tu maestro se encuentre explicando un tema, apunta todo lo que no entiendas y cuando tu maestro pregunte a la clase si se ha entendido lo que explicó, pide permiso para preguntar lo que hayas apuntado y pregúntalo cuando tu maestro te autorice a hacerlo. Nunca interrumpas a tu maestro mientras explica un tema. Hacerlo podría distraer tu atención, la suya o la de tus compañeros y lograr perder el hilo conductor.

Regla Nº9: no leas el material técnico como un cuento o una novela

El material técnico no posee un principio un nudo y un desenlace. Por eso, no debe ser leído como una obra literaria más. Cuando te enfrentes a material técnico, lo primero que debes leer es el índice de contenidos e ir hacia aquello que te interese incorporar (a no ser que tu maestro ya te haya dado las indicaciones del caso). Lo segundo que debes hacer, si encuentras código fuente, tratar de explicártelo a ti mismo antes de leer el texto. Lo tercero, es leer el texto que rodea al código fuente. A continuación, copiarás y ejecutarás el código fuente del libro tal y como aparece. Luego, tratarás de modificarlo y finalmente, si aplica, tratarás de implementarlo en algún contexto. Solo después de todo esto, adelantarás la lectura, incluso aunque hayas estado 2 horas en la misma carilla.

Regla Nº10: cuando tu maestro te corrija, aprende, no te justifiques

Por lo general, conocer los motivos que llevaron a cometer un error no requiere de explicación. Un buen maestro es capaz de conocer los «porqué» del error. Si inviertes tus energías en explicar a tu maestro por qué te has equivocado, estarás desperdiciando la oportunidad de aprender. Si cuando tu maestro te corrige, solo te concentras en comprender exactamente qué es lo que tu maestro te está diciendo, aprovecharás mejor el tiempo ya que podrás emplear el tiempo disponible en preguntar lo que aún no hayas entendido; en ejercitarlo frente a tu maestro; o en ratificar con tu maestro que lo has entendido realmente. Si en cambio te preocupas por excusarte del error cometido, solo habrás invertido el tiempo en eso y no habrás aprendido nada.

Regla Nº11: no pretendas saber todo en una sola clase

Para aprender de verdad debes ser paciente. Si en una sola clase intentas evacuar todas las dudas que te hayan surgido a lo largo de tu carrera, no harás más que saturarte de información y por consiguiente, conocerás las cosas superficialmente sin llegar a saber nada. Intenta no abarcar más de un tema por clase para poder alcanzar un mejor nivel de experiencia en el mismo.

Regla Nº12: nunca des por sentado que has entendido antes de que finalice la explicación

Cuando preguntas algo a tu maestro y éste comienza a explicarte el tema, es probable que con las primeras palabras que escuches creas haber comprendido todo por completo. Lo que en realidad sucede es que un buen maestro comenzará a darte la explicación citando aquello con lo que sabe ya tienes un dominio. Por eso crearás haberlo entendido todo. Si en ese momento te cierras a continuar escuchando o peor aún, interrumpes a tu maestro y le impides continuar la explicación, más allá de ser visto como un irrespetuoso estarás actuando como necio y tú mismo te cerrarás la puerta a aprender aquello que te generaba dudas. Un buen maestro no es escueto en sus explicaciones pero tampoco es «hablador». De un buen maestro puedes aprender hasta de su última palabra, pero solo si eres un buen estudiante.

ANEXO I: Consideraciones generales que mejoran la experiencia como estudiante

Ceñirse a las pautas dictadas por el docente

Cuando era pequeña, recuerdo haber sido regañada por mi maestra de segundo grado tras haber entregado «tarea de más». Cuando se lo comenté a mi mamá, ella, mostrándose ofendida dijo: ***«pero ¿está loca esa mujer? ¿cómo va a retar a una criatura por tener iniciativa? Tendría que haberte felicitado»****. Sentía por mi maestra un gran respeto y un enorme cariño así que me resultaba imposible creer que ella se pudiese equivocar. Pero por mi mamá sentía mucho más respeto, un incomparable amor y algo que no sentía por mi maestra: admiración. Así que tampoco era posible que se equivocara. Entonces, no quedaba más remedio que tratar de compatibilizar ambas «opiniones», la de mi mamá con la de la maestra. Y a pesar de mi escasa experiencia, lo intenté.*

Lo cierto era que siempre sentía que «la tarea me quedaba corta», la finalizaba muy rápido y me aburría mucho el resto del tiempo. Así que pensé que si mi mamá decía que estaba bien y la maestra, que estaba mal, ***compatibilizar ambas sería como mezclar colores, hacer un bello rosa, mezclando un hermoso rojo con blanco.***

Así que la siguiente vez, al terminar la tarea que mi maestra me había encomendado, me levanté de mi pupitre y le dije: «Seño', ya terminé pero me aburro mucho si no hago nada ¿me puede dar más tarea, por favor?». Y así fue, como Ana María, mi maestra de primero y segundo grado, me «nombró» su «Ayudante oficial». Y ***mi nueva tarea fue mucho más divertida que la que yo pretendía hacer.*** *Desde ese momento y hasta el último día de clases, se me había encomendado colaborar en la preparación de ejercicios para mis compañeros y ayudar a aquellos a quienes se les complicara resolverlos, cada vez que me sobrara tiempo libre.*

La historia anterior, sin dudas, marcó mi vida en muchos aspectos y el más relevante a nivel profesional, ha sido en mi aspecto como docente -y también como alumna-.

Frecuentemente, la mayoría de las personas cree que «hacer más de lo que se les pide» es algo elogioso. Incluso muchos consideran que «hacer algo MEJOR de lo que se les pide» es aún más digno de grandes elogios. Sin embargo, imagina esta situación:

> *Vas al aeropuerto y pides comprar un pasaje a Haití que tan solo cuesta un €1. Cuando la empleada te entrega el pasaje, ves que el destino es Cancún y que el valor del pasaje es de €1000. Le reclamas a la cajera y ella te responde: «no se preocupe por el precio, se lo regalo para que pueda viajar a un mejor lugar que Haití». Lo que la cajera no tuvo en cuenta, es que tú quieres ir a Haití porque es el único lugar del mundo que aún te queda sin conocer. No importa para ti qué lugar es mejor o peor, pues para ti, lo mejor para cumplir tus objetivos de conocer el mundo, es ir al único lugar al que jamás has ido.*

Es evidente, que cuando uno pide algo, espera que la otra persona haga lo que se le pide sin modificaciones, pues todo lo que se pide, se pide por un motivo. Entonces ¿con qué fin harías lo contrario? Incluso aunque imagines que hacer lo que no se te pide es «la mejor alternativa», el riesgo de estar equivocándote es altísimo.

En lo personal y profesional, la experiencia vivida en la niñez me llevó a aprender que:

- Como alumna, sincerarnos con nuestro maestro y decirle la forma en la que nos sentimos con respecto a sus pautas, es beneficioso para nuestro aprendizaje y para la satisfacción de nuestro espíritu.

- Como docente, es tan importante escuchar a nuestros alumnos como tener en cuenta sus emociones y que se debe ser flexible a la hora de fijar las pautas de ejercitación.

Un buen maestro, sabe con exactitud como guiar a sus alumnos y por consiguiente, cuando encomienda determinada tarea, lo hace con un único fin: lograr un aprendizaje significativo en sus alumnos. Por ese motivo, es tan importante seguir sus pautas a la hora de cumplir con los ejercicios propuestos y ser honestos a la hora de expresar cómo nos hemos sentido.

Acepta que los fallos en tus códigos e incluso los del sistema, forman parte de la tarea de programar

No me canso de explicar esto a mis alumnos y de hacer los máximos esfuerzos para que lo comprendan. **Cuando intentamos ejecutar nuestra aplicación y falla, puede deberse** -como es sabido-, **tanto a un error nuestro en el código como a una característica del sistema que debe ser modificada.**

Frente a un fallo, podemos reaccionar con la misma ignorancia e inmadurez de un usuario, blasfemando por los cuatro costados lo inservible que es el sistema, lo injusta que es la vida y la forma en la que ésta apesta o podemos afrontar la situación como lo que es en realidad: un problema inesperado que como profesionales estamos capacitados para resolver.

En este sentido, es importante comprender dos cosas:

- La primera, es que si nos dedicamos al desarrollo de sistemas, debemos entender que la Ingeniería es la ciencia encargada de resolver problemas existentes aplicando la rama de estudio que le es competencia. De esta forma, la Ingeniería de Software es la ciencia encargada de resolver problemas existentes mediante la aplicación informática. Por lo tanto, en Ingeniería, **los errores no son más que problemas y los problemas son los que hacen que la Ingeniería tenga sentido**. Sin problemas, no hay Ingeniería que aplicar.

- La segunda, es que los errores -que son en realidad problemas- deben tomarse como algo no solo esperable sino además, deseable. **Es necesario que los sistemas fallen**, tanto los propios como aquellos sobre los cuáles éstos se sustentan, porque de lo contrario, no habría forma de saber si lo creado funciona verdaderamente o solo lo aparenta. De hecho, **la técnica de programación más compleja y avanzada que existe, se basa en «hacer fallar la aplicación»**. Me refiero a TDD.

> *Si asumes que los errores son deseables y que resolverlos ES tu tarea propiamente dicha, podrás disfrutar de cada segundo de estudio y no conocerás los límites. **Podrás atravesar la barrera del conocimiento y alcanzar metas inimaginables.***

Pero, **si te enfrentas a los errores con inmadurez**, no lograrás más que amargura y **te estarás cerrando las puertas al aprendizaje**.

Sobre el reporte del bug #56883 en Apache 2.4 y otros conflictos sobre Ubuntu 14.04

*— Eugenia Bahit agradece a **Hugo** (@huguidugui) por la **revisión ortográfica** de este artículo*

Recorrido sobre los problemas más frecuentes hallados en la versión 2.4 de Apache compilada por Ubuntu 14.04 y sus posibles soluciones.

El pasado jueves 21 de agosto estaba dictando clases, cuando casi literalmente «explotó», producto de un ataque de nervios generado por los innumerables «martes 13» que vengo atravesando con la versión 14.04 LTS de Ubuntu.

Algunos de mis alumnos decidieron probar esta nueva versión cuando yo aún no terminé de investigarla, pues los cambios a los que me he tenido que enfrentar, muy poco me han agradado y eso, ha retrasado mis *testeos*. Entre Ubuntu 14.04 y Apache 2.4 los cambios implementados, en mi opinión profesional, no hacen «uno bueno» y traen más problemas que soluciones:

- que si el *Web Site* no se almacena en `/var/www`;
- que si el *VirtualHost* no posee la extensión `.conf`;
- y ahora, que si se activa `MultiViews` falla `ModRewrite`.

¿Por qué si no va a /var/www da un Forbiden?

Porque en el archivo de configuración de Apache que viene en la compilación para Ubuntu 14.04 (localizado en `/etc/apache2/apache2.conf`), por defecto, se encuentran denegadas todas las solicitudes:

```
<Directory />
    ...
    Require all denied
</Directory>
```

Pero son concedidas al directorio `/var/www`:

```
<Directory /var/www/>
    ...
    Require all granted
</Directory>
```

¿Es esto una cuestión de seguridad?
Sí lo es, no me explico cómo es posible que por defecto se permita entonces un **listado del directorio raíz**, **una de las peores prácticas de seguridad**:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    ...
</Directory>
```

Si se quiere conservar /var/www como lugar de almacenaje para los *Web Sites*, bastará con agregar un nuevo `<Directory>` y así solucionar «ese maldito *Forbiden*». Sino, solo modificar la ruta `/var/www` por la que se desee:

```
<Directory /nuevo/directorio/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

Vale aclarar que **prohibir el listado de directorios será una buena práctica**. Pero **¡ojo!** que ahora Apache requiere la asignación del signo + cuando se emplee además un signo - y de no hacerlo, fallará:

```
Either all Options must start with + or -, or no Option may
```

Para que no falle, **siempre que se agregue un -** (desactivar) a una opción, **se deberá agregar un +** (activar) a las opciones que no se estén desactivando:

```
<Directory /nuevo/directorio/>
    Options -Indexes +FollowSymLinks
    ...
</Directory>
```

¿Por qué el VirtualHost debe llevar la extensión .conf?

Sinceramente, esta me resulta una de las políticas menos justificables de la historia. **La «extensión» de un archivo** no es más que parte del nombre del archivo y **no representa en absoluto el tipo MIME del archivo**, como el sistema operativo privativo de las *ventanitas* ha querido imponer.

Entonces ¿para qué agregar un `.conf` a los archivos? Todo archivo que esté dentro de `sites-enabled` debería ser cargado como *VirtualHost* porque ¿quién colocaría allí un archivo que no fuese un *VirtualHost* habilitado?

Sin embargo, `apache2.conf` en el paquete compilado por Ubuntu 14.04, decide establecer que «solo» los archivos con extensión .conf sean cargados como sitios habilitados:

```
# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf
```

¡Qué chiste! **Quien colocase en `sites-enabled` un archivo que no pertenezca a un *VirtualHost* habilitado, estaría siendo desordenado e incoherente**. Es por ello, que una buena práctica sería cargar «todos» los archivos que estuviesen en ese directorio:

```
IncludeOptional sites-enabled/*
```

De esa forma, **la extensión de los *VirtualHost* ya no sería un problema**.

Ya no se trata de una mala decisión, se trata de un bug...

Y esta vez no es responsabilidad de Ubuntu. Sucede que en la versión 2.4 de Apache y al menos hasta la 2.4.10 **la activación de la opción `MultiViews` genera un fallo en `ModRewrite` cuando la URI contiene parte del nombre de un archivo existente**.

> ***MultiViews*** *es una opción que permite lo que se denomina «negociación de contenido» a través del módulo* ***Negotiation***

Cuando se solicite a través de la URI un archivo inexistente, `ModNegotiation` permitirá a Apache efectuar una búsqueda por patrón y servir aquel archivo que primero coincida con dicha búsqueda. De esta forma, si se solicitase el archivo `/foo` en la URI, se serviría el primer archivo cuyo nombre coincidiera con la palabra `foo`, pudiendo tratarse de `foo.php`, `foo.png`, etc. Pero siempre será el archivo que primero coincida en la búsqueda.

Cuando se trabaja con el módulo `Rewrite` activado y se implementa un sistema de URLs virtuales (conocidas entre los usuarios y profesionales del *marketing* como *«friendly URLs»*) toda solicitud efectuada a través de la URI, es primero manejada por `ModRewrite` y solo cuando esto no sea posible, tomaría el mando `ModNegotiation`, pues para eso ha sido creado: para tratar de «solucionar el problema» cuando todo lo

demás falle. De esta forma, si se posee una regla de reescritura como la siguiente:

```
RewriteRule ^ whatever.php
```

Toda solicitud (absolutamente «toda») debería ser redirigida por `ModRewrite` al archivo `whatever.php` y esto, es lo que sucedía hasta la aparición de la versión 2.4 de Apache.

En esta nueva versión, cuando `/archivo-que-no-existe` es solicitado, `whatever.php` es servido. Sin embargo, al solicitar `/si-existo` el archivo `si-existo.php` es servido al igual que la petición literal `/otro-archivo-que-existe.php` se sirve de forma directa. Un error garrafal que si bien no compromete la seguridad de forma directa, podría ponerla en riesgo considerando que la implementación de reglas de reescritura suele utilizarse para enmascarar las verdaderas *URIs*.

Esto mismo es lo que reporté como *bug* a Apache:
https://issues.apache.org/bugzilla/show_bug.cgi?id=56883

Al momento de escribir este artículo, el reporte es muy reciente y por lo tanto, no se pueden esgrimir conjeturas de ningún tipo sobre el tratamiento que se le haya dado al mismo.

Dado que a diferencia de los dos casos anteriores, aquí se trata de un *bug*, **no existe solución posible hasta que no se implemente una a nivel del *core* de Apache**. No obstante, **puede implementarse una medida provisoria** para impedir que el *bug* se refleje en nuestros *hosts*. Lo que debe hacerse en **desactivar la opción `MultiViews`**. Lamentablemente, la desactivación de esta opción, **impedirá la negociación de contenido** y frente a reglas tales como:

```
RewriteRule !(^static) whatever.php
```

La solicitud de `/static/img/foo` **«NO»** sería negociada como `/static/img/foo.png` provocando entonces, un error 404 Not Found. Vale aclarar que **al desactivar la opción `MultiViews`** en un directorio, **no será posible activarla en ninguno de sus subdirectorios**, de manera tal que la activación en un caso como el siguiente, sería absolutamente ignorada por Apache:

```
<Directory /foo/>
    Options -Indexes +FollowSymLinks -MultiViews
</Directory>

<Directory /foo/bar>
    # La activación de la opción MultiViews aquí, sería ignorada
    # ya que bar es subdirectorio de foo
    Options +MultiViews
</Directory>
```

NOVEDADES SOBRE JACKTHESTRIPPER Y EUROPIOCODE

Servidores Seguros en un solo paso con JackTheStripper

> ***JackTheStripper**, el deployer para servidores que en un solo paso **instala, configura y asegura un servidor Web completo** sobre **Ubuntu 12.04**, ahora se encuentra disponible para múltiples distribuciones y versiones gracias al aporte de **Jason Soto**.*

Las distribuciones y versiones soportadas al momento son las siguientes:

Versiones adaptadas por **Jason Soto**:

- **Ubuntu 14.04 LTS**
- **CentOS 6**
- **CentOS 7**
- **Debian 6**
- **Debian 7**

Versiones oficiales:

- **Ubuntu 12.04 LTS:** Página | Descarga | Código Fuente
- Manual del Usuario (apto para todas las versiones y distribuciones)
- Videotutorial

EuropioCode y una nueva librería para Python

> ***EuropioCode**, el **codificador de caracteres** basado en modelos permisivos y empleado por el núcleo EuropioEngine, ahora se encuentra disponible (aunque aún en desarrollo) para **Python 2.7** y **Python 3.x** siendo apto tanto para aplicaciones Web como Desktop e incluso en el Scripting.*

Documentación:

- Manual completo de EuropioCode (incluye notas para el desarrollador y manual del usuario)

Librerías disponibles:

- **JavaScript** (versión estable): para envío codificado de formularios desde el *Front-End*
- **PHP** (versión estable): codificación, decodificación y funciones de *reversing*, limpieza y depuración de *strings*
- **Python** (versión en desarrollo): codificación y decodificación de *strings*

TUI: Text User Interfaces en GNU/Bash y Python

— *Eugenia Bahit agradece a **Hugo** (@huguidugui) por la **revisión ortográfica** de este artículo*

La creación de interfeces de texto parece ser algo olvidado en este siglo. Sin embargo, en el mundo de los servidores donde los entornos gráficos no dejarán de ser una molestia innecesaria, las TUI no parecen tener fecha de caducidad.

Mucho se habla (y hablo) de las GUI (*Graphical User Interfaces*) pero de las **TUI** (*Text User Interfaces*) nos olvidamos de su existencia hasta que aparece alguien que nos lo recuerda y eso, fue lo que sucedió días atrás en *Twitter* cuando un usuario me preguntaba cómo crearlas.

A raíz de lo anterior y tras haberle comentado que antiguamente, empleaba `dialog` para GNU/Bash y `ncurses` en C y Python. *Dialog*, siempre me había resultado simple y sencillo de utilizar y jamás me había dejado con «sensación de insatisfacción» ya que tenía todo que un programador podría necesitar para la TUI de su *Script* o aplicación y entonces me surgió la pregunta **¿por qué no utilizaba `dialog` en Python?**

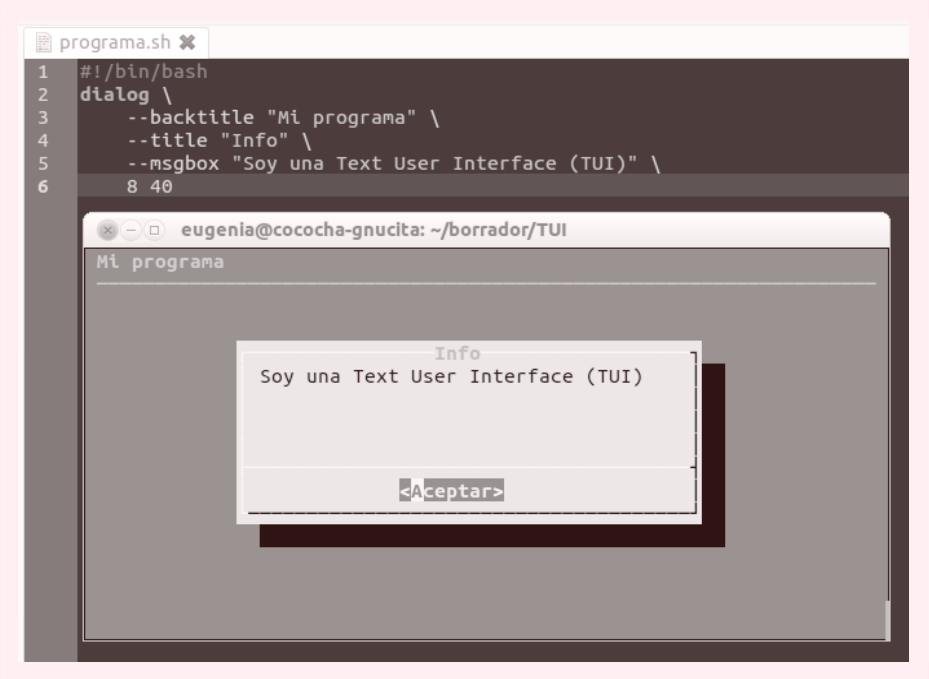
Y así fue que me surgieron unas ganas tremendas de escribir una guía sobre este tema del cual en más de 15 años no he escrito nunca. ¡Espero que la disfruten!

Quiero comenzar primero por introducirlos en el mundo `dialog` de GNU/Bash para que puedan ir familiarizándose con esta herramienta. El **sitio Web oficial** de `dialog` es http://invisible-island.net/dialog/

> *Dialog comenzó en 1994 de la mano del programador **Savio Lam** pero desde 1999 es escrito y mantenido por **Thomas E. Dickey**, creador de **Lynx**, **ncurses** y **xterm** -entre muchas otras herramientas- quien lamentablemente **decidió cambiar la original licencia GPL** del programa por **LGPL**, permitiendo así que cualquier persona pudiese crear mejoras privativas alejándolas así del alcance de la comunidad y de su consecuente estudio.*

¿Qué es dialog?

Dialog es una herramienta que provee de pseudo interfaces gráficas (o interfaces gráficas textuales) para nuestras aplicaciones de consola.



Entre los tipos de «artilugios» disponibles, se encuentran los cuadros de diálogo (si/no), calendarios, entradas de texto, listas desplegables, etc.

¿Cómo se obtiene?

Dialog puede instalarse directamente desde repositorios. En **Debian** y derivados:

```
# apt-get install dialog
```

Casos de Uso

El uso más frecuente de las TUI es en el *Scripting* y programas de consola en general. Mediante el comando `man dialog` se puede acceder a toda la ayuda del programa y obtener así, detalles claros sobre su sintaxis y argumentos, lo que nos podría dar una idea más amplia de sus posibles usos.

La sintaxis básica para utilizar `dialog` es:

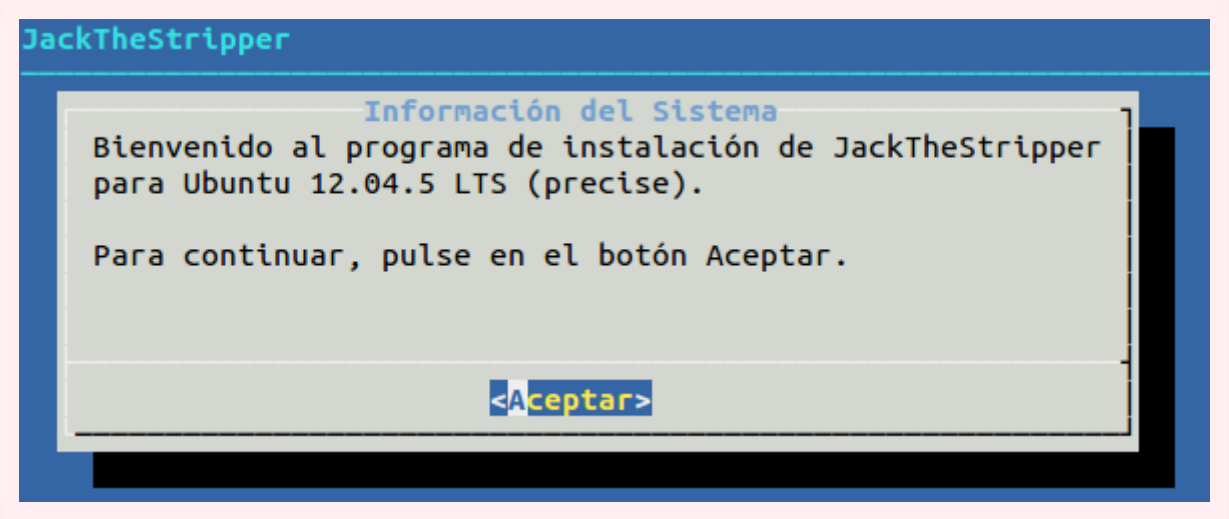
```
dialog --opciones-comunes --tipo-de-diálogo "Contenido" alto ancho --opciones-del-diálogo
```

®2013, 2014 Eugenia Bahit – www.eugeniabahit.com – Bajo Licencia Creative Commons BY-NC-SA

Caja de diálogo normal: Mostrando la información del sistema operativo

```
#!/bin/bash
# Archivo: programa.sh

system_info=`lsb_release -ds`
system_codename=`lsb_release -cs`

read -d '' msg_bienvenida << EOF
Bienvenido al programa de instalación de JackTheStripper para $system_info
($system_codename).

Para continuar, pulse en el botón Aceptar.
EOF

dialog \
    --backtitle "JackTheStripper" \
    --title "Información del Sistema" \
    --msgbox "$msg_bienvenida" \
    10 60
```

El código anterior producirá un diálogo como el siguiente:



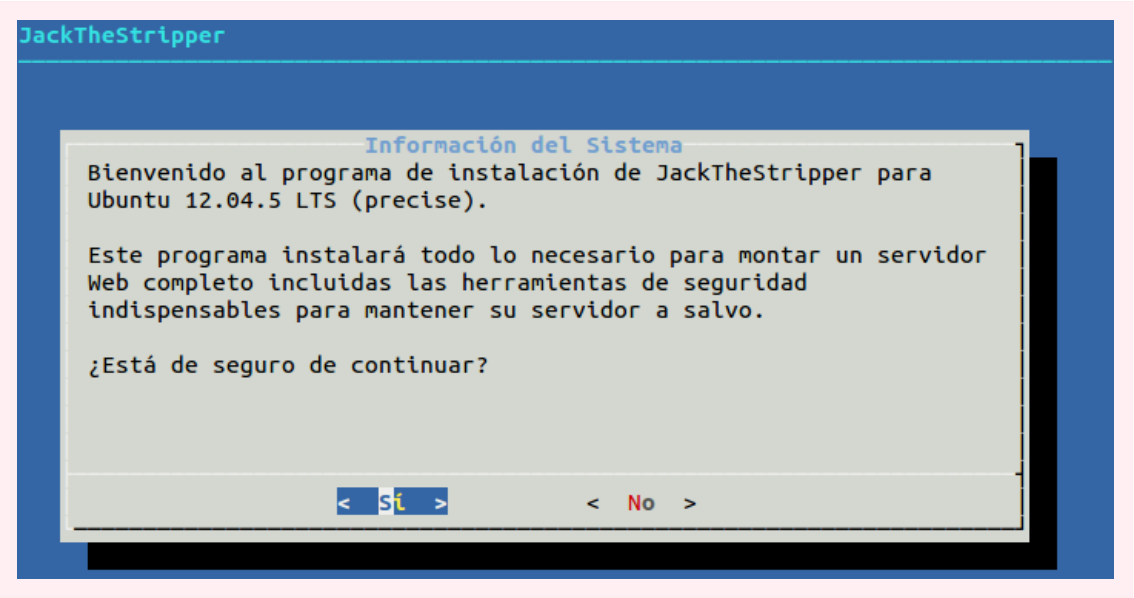
Si deseas probar el código anterior (y los siguientes) y aún no sabes cómo hacerlo, abre un terminal y procura seguir estos pasos:

1. Crear un archivo: `touch programa.sh`
2. Asígnale permisos de ejecución: `chmod +x programa.sh`
3. Abre el archivo anterior con cualquier editor de textos, copia el código y pégalo en el archivo programa.sh.
4. Tras guardar el archivo, ejecútalo desde la terminal escribiendo: `./programa.sh`

Diálogo de confirmación: Confirmando antes de actuar

```
#!/bin/bash
system_info=`lsb_release -ds`
system_codename=`lsb_release -cs`

read -d '' msg_bienvenida << EOF
Bienvenido al programa de instalación de JackTheStripper para $system_info
($system_codename).

Este programa instalará todo lo necesario para montar un servidor Web completo incluidas las
herramientas de seguridad indispensables para mantener su servidor a salvo.

¿Está de seguro de continuar?
EOF

dialog \
    --backtitle "JackTheStripper" \
    --title "Información del Sistema" \
    --yesno "$msg_bienvenida" \
    15 70
```

El código anterior producirá un diálogo como el siguiente:



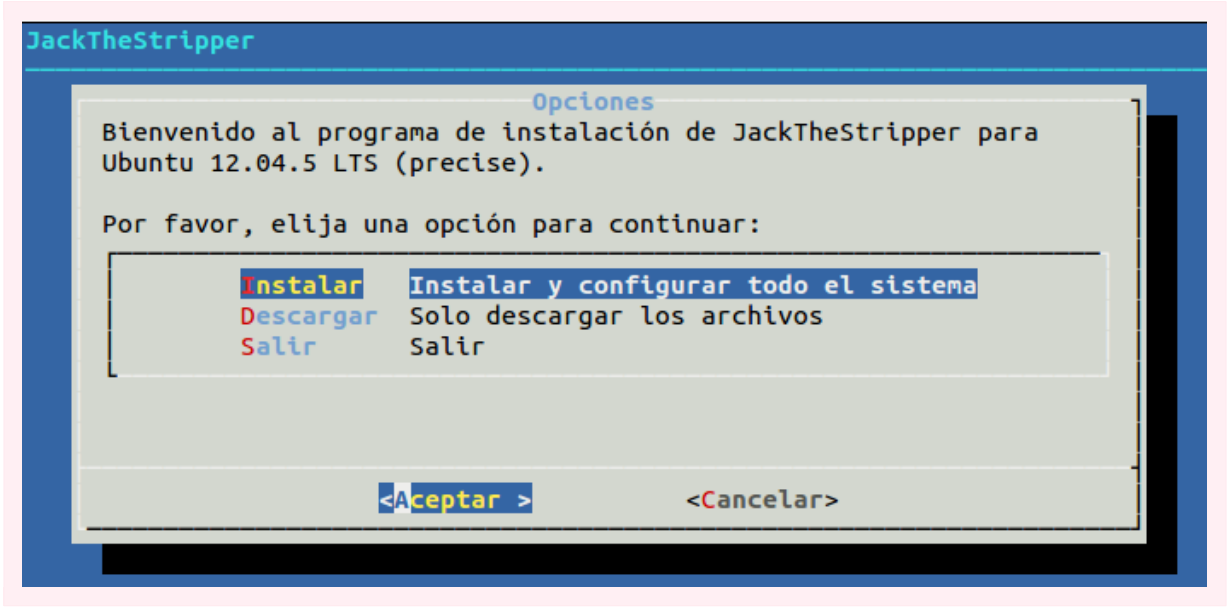
La opción elegida por el usuario, será capturada y representada como: 0=>Sí, 1=>No, 255=>cualquier tecla de escape que canceló la operación. Un ejemplo:

```
respuesta=$?

case $respuesta in
    0) cd cd deploymyserver; sudo ./dms.sh;;  # ejecuta el archivo de instalación
    1) rm -R deploymyserver;;  # Elimina los archivos descargados
    255) echo "Bye!";;   # No hace nada e imprime un mensaje en pantalla
esac
```

Menú: Eligiendo opciones de un menú

```
#!/bin/bash
system_info=`lsb_release -ds`
system_codename=`lsb_release -cs`

read -d '' msg_bienvenida << EOF
Bienvenido al programa de instalación de JackTheStripper para $system_info
($system_codename).

Por favor, elija una opción para continuar:
EOF

dialog \
    --backtitle "JackTheStripper" \
    --title "Opciones" \
    --menu "$msg_bienvenida" \
    15 70 3 \
    Instalar "Instalar y configurar todo el sistema" \
    Descargar "Solo descargar los archivos" \
    Salir "Salir"
```

El código anterior producirá un diálogo como el siguiente:



El número **3** corresponde a la cantidad de opciones del menú que serán visibles (resaltado en rojo en el código). Las siguientes líneas corresponden a cada uno de los ítem del menú, siendo la *string* de la izquierda, el valor que será almacenado por `dialog`. Puede capturarse dicho valor almacenándolo en un archivo temporal:

```
dialog \
    ...
    Instalar "Instalar y configurar todo el sistema" \
    Descargar "Solo descargar los archivos" \
    Salir "Salir" 2> /tmp/opcion-elegida

cat /tmp/opcion-elegida
```

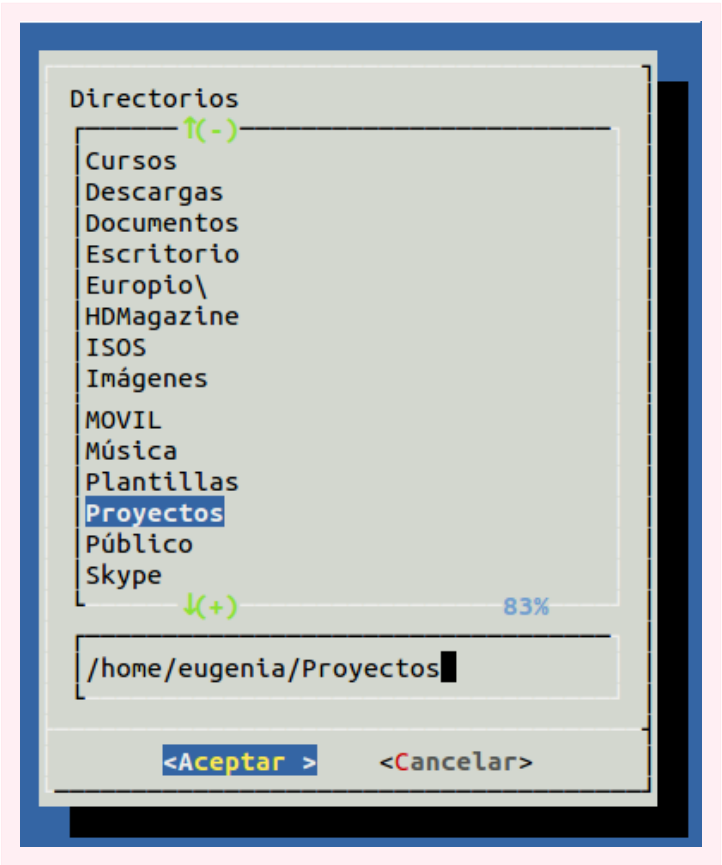
O también se podría capturar en una variable, agregando la opción común `--stdout`, aunque no es la forma más recomendada por su autor:

```
salida=$(dialog --stdout \
    --backtitle "JackTheStripper" \
    --title "Opciones" \
    --menu "$msg_bienvenida" \
    15 70 3 \
    Instalar "Instalar y configurar todo el sistema" \
    Descargar "Solo descargar los archivos" \
    Salir "Salir")

echo $salida
```

Selector de archivos y directorios: Eligiendo un directorio de instalación

```
#!/bin/bash
directorio=$(dialog \
    --stdout \
    --backtitle "Europio Engine Installer" \
    --dselect $HOME/ \  # utilizar --fselect en vez de –dselect para seleccionar archivos
    15 40)

cd $directorio
```

El código anterior producirá un diálogo como el siguiente:



Lo mismo pero desde Python

Lógicamente que los diálogos vistos anteriormente, son solo una mera introducción a la larga lista de artilugios provistos por `dialog`. **Vivek Gite** escribió una excelente **guía sobre *Shell Scripting*[2]** en la cual dedica un **capítulo sobre `dialog`[3]** que te puede resultar muy interesante. Pero teniendo ya una visión general sobre `dialog` y sus usos, hablar de la librería **`pythondialog`** será mucho más sencillo.

PythonDialog (pythondialog.sourceforge.net) puede instalarse mediante un simple «pip», disponiendo de las versiones para Python 2 y 3 de forma separada:

```
eugenia@cococha-gnucita:~$ pip search pythondialog
pythondialog              - A Python interface to the UNIX dialog utility ...
python2-pythondialog      - A Python interface to the UNIX dialog ... (Python 2 backport)
eugenia@cococha-gnucita:~$
eugenia@cococha-gnucita:~$ sudo pip install python2-pythondialog
```

Emplear esta librería es tan sencillo como:

1. importar la clase `Dialog` del módulo;
2. crear una instancia de la clase y comenzar a crear cuadros de diálogo.

Los cuadros de diálogo se crean mediante métodos del objeto `Dialog`, cuyos nombres equivalen a los diversos tipos de diálogos provistos por `dialog`. Los argumentos de opción que antes se pasaban con `--opcion` ahora se pasan como claves (*keyword args*). De esta forma, para crear un cuadro de diálogo normal, bastaría con las siguientes instrucciones:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from dialog import Dialog


dialog = Dialog()
dialog.msgbox(title=u"Título", text=u"Mensaje del cuadro de diálogo")
respuesta = dialog.yesno(title=u"Título", text=u"Está seguro de continuar?")

if respuesta == 'ok':
    print "Está seguro"
elif respuesta == 'cancel':
    print "No. No estaba seguro"
```

Claro que no todo es color de rosa y algunas dificultades se pueden presentar. La más notable de ellas es para los hispanoparlantes, el problema eterno con los caracteres no ASCII en Python. No basta con definir la codificación de caracteres como UTF-8 al inicio. Pero se pueden evitar todos los conflictos empleando `cadena.decode('utf-8')` como se muestra a continuación:

2 http://bash.cyberciti.biz/guide/Main_Page

3 http://bash.cyberciti.biz/guide/Bash_display_dialog_boxes

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from dialog import Dialog


dialog = Dialog()
dialog.msgbox(title=u"Título", text=u"Mensaje del cuadro de diálogo")
respuesta = dialog.yesno(title=u"Título",
    text="¿Está seguro de continuar?".decode('utf-8'))

if respuesta == 'ok':
    print "Está seguro"
else:
    print "No. No estaba seguro"
```

Luego, las opciones comunes pueden «setearse» mediante métodos, aunque en versiones recientes no son del todo intuitivos. Por ejemplo:

```
dialog = Dialog()
dialog.set_background_title("Título del programa")
```

Es la versión moderna de:

```
dialog = Dialog()
dialog.add_persistent_args(["--backtitle", "Título del programa"])
```

Ninguno de los dos es demasiado intuitivo pero tal vez, en la forma correspondiente a las versiones anteriores, con solo recordar un método, bastaba para definir la lista con el argumento original y su valor.

El sitio Web oficial es http://pythondialog.sourceforge.net/ pero no posee más que unos pocos ejemplos. Sin embargo, siempre podemos hacer un:

```
eugenia@cococha-gnucita:~$ python
...
>>> from dialog import Dialog
>>> help(Dialog)
```

Lo anterior, nos permitirá acceder a una documentación muy completa sobre esta librería. ¿Mi recomendación? Prueba primero «jugar» con `dialog` en GNU/Bash. Entiéndelo, disfrútalo y encuentra una utilidad. Recién allí, comienza a experimentar desde Python pero ya, con una idea más concreta y una necesidad de uso real. De esa forma, lograrás mejores resultados.

Python Web sin frameworks: sobre las sesiones y el acceso restringido

— *Eugenia Bahit agradece a **Hugo** (@huguidugui) por la **revisión ortográfica** de este artículo*

Beaker es un middleware para WSGI que permite, entre otras características, efectuar un manejo de sesiones avanzado en aplicaciones Web creadas con Python. Su implementación es verdaderamente sencilla y se logra en pocos pasos.

Donar

Poco a poco, la ingeniería de Software tras haber ganado terreno en el ámbito Web, se está volcando más por Python y cada vez son más los programadores que le pierden el miedo a lo desconocido y se animan a hacer verdadera ingeniería creando aplicaciones sin recurrir a *Django* o a otros *frameworks* de similares características.

En el desarrollo de aplicaciones Web siempre ha sido la restricción de acceso y el manejo de usuarios, un gran hito para los recién llegados. En lenguajes como PHP existe no solo un soporte nativo para el manejo de **«sesiones de usuario»** sino que además, abunda la bibliografía al respecto. Pero en lenguajes como Python, la documentación pareciera reflejar que el manejo de sesiones de usuarios solo sería posible empleando *frameworks*. Y esto no es así.

Pero para profundizar aún más, es necesario primero, comprender de qué se habla en realidad cuando hablamos de «sesiones». Y para ello, recomiendo leer el artículo **«Sesiones ¿qué son realmente y en qué consisten? (desmitificando el concepto de sesiones)»** publicado en la edición Nº6 de The Original Hacker. Una versión del artículo en PDF puede obtenerse desde la siguiente URL: http://library.originalhacker.org/search/sesiones

Sesiones en Python

Como bien comencé diciendo, Python no tiene un soporte «nativo» para la implementación de sesiones y el manejo de *cookies* suele ser algo engorroso con WSGI. Esto, convierte al trabajo de desarrollar un manejador de sesiones en una tarea larga y compleja de emprender, aunque no imposible. Aquí, abarcaremos el manejo de sesiones empleando la librería **Beaker** (http://beaker.readthedocs.org/en/latest/).

Primeros pasos

Se utilice Beaker, cualquier otra librería o se desarrolle una propia, las herramientas con las cuáles debe contarse son:

Una base de datos con al menos un usuario de prueba en una tabla de usuarios

```
-- Reemplazar foobar por el nombre de la base de datos a utilizar
USE foobar;

-- Estructura base de la tabla usuarios
CREATE TABLE usuarios (
    usuario_id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(25) NOT NULL,
    password VARCHAR(32) NOT NULL
) ENGINE=InnoDB;

-- Usuario de prueba
INSERT INTO usuarios
            (username, password)
     VALUES ('prueba', 'c893bad68927b457dbed39460e6afd62');
```

c893bad68927b457dbed39460e6afd62 es el ***hash* MD5** de la clave **prueba**.

Luego, necesitaremos un **formulario de inicio de sesión**:

```
<form method='POST' action='/usuarios/validar'>
  Usuario: <input type='text' name='u' id='username'><br>
  Clave: <input type='password' name='p' id='password'>
  <button type='submit'>Ingresar</button>
</form>
```

Por consiguiente, un módulo de usuarios (**`usuarios.py`**) con un **recurso público encargado de mostrar el formulario**:

```
def ingresar():
    with open('/ruta/a/form_login.html', 'r') as archivo:
        return archivo.read()
```

Una **función validar** (en el módulo de usuarios) encargada de verificar que los datos ingresados en el formulario anterior, coincidan con el de un usuario en la base de datos y en caso afirmativo, invoque al iniciador de sesión o de lo contrario, muestre nuevamente el formulario:

```
from hashlib import hashlib

from core.postdata import POSTDataHandler, POSTDataCleaner
from core.dblayer import run_query
from core.sessions import iniciar  # TODO
```

```
@POSTDataHandler
def validar(_POST):  # application deberá enviarle environ como parámetro
    usuario = POSTDataCleaner().sanitize_string(_POST['u'])
    clave = hashlib.md5(_POST['p']).hexdigest()

    sql = """SELECT usuario_id FROM usuarios
              WHERE username = '%s'
                AND password = '%s'""" % (usuario, clave)
    resultados = run_query(sql)

    return iniciar() if len(resultados) == 1 else ingresar()
```

El archivo de sesiones (**sessions.py**) deberá hospedarse a nivel del *core* y contar con al menos las funciones para iniciar una sesión, destruirla y verificar una sesión activa (completaremos las funciones más adelante):

```
def iniciar():
    pass


def destruir():
    pass


def verificar():
    pass
```

Implementando Beaker

Ante todo, para poder utilizar Beaker será necesario instalarlo desde PyPi:

```
sudo pip install beaker
```

Luego, serán necesarios algunos cambios en nuestro controlador y en la función `application`. El primer paso será **importar el *middleware***:

```
from beaker.middleware import SessionMiddleware
```

A continuación, **la función `application` cambiará de nombre** (puede elegirse cualquier nombre) ya que deberá ser envuelta por el *middleware* y éste, ser quien responda ante la llamada de WSGI:

```
def ~~application~~(environ, start_response):
def start(environ, start_response):
```

Finalmente, **el *middleware* envolverá a la ex función `application`** y éste será quien actúe ante la invocación de WSGI:

```
# Configuraciones necesarias para el Middleware
beaker_dic = {
    'session.type': 'file',
    'session.cookie_expires': True,
    'session.auto': True,
    'session.data_dir': '/tmp/sessions',
}


application = SessionMiddleware(start, beaker_dic)
```

El archivo de sesiones

Ahora completaremos las funciones definidas en el archivo de sesiones. Las **variables de sesión** generadas por *Beaker*, se encontrarán disponibles en la **clave `beaker.session` del diccionario `environ`**. La función de inicialización de sesión se encargará de crear variables de sesión que nos ayuden a identificar si un usuario se encuentra *«logueado»* en nuestro sistema. Crearemos 2 variables de sesión: `logged` (*booleana*) y `username` (*string*). Para lograrlo, será necesario contar con el diccionario `environ` por lo cual, primero, modificaremos la función `validar()` de `usuarios.py` (para que nos pase este parámetro):

```
@POSTDataHandler
def validar(_POST, environ):
    usuario = POSTDataCleaner().sanitize_string(_POST['u'])
    #...
    return iniciar(environ, usuario) if len(resultados) == 1 else ingresar()
```

Y ahora sí, completaremos la **función `iniciar`** creando las nuevas variables de sesión:

```
def iniciar(environ, username):
    environ['beaker.session']['logged'] = True
    environ['beaker.session']['username'] =  username
```

Para **destruir una sesión** bastará con invocar al método `delete()` -creado por Beaker-:

```
def destruir(environ):
    environ['beaker.session'].delete()
```

Y para finalizar este archivo, **la función que verifica la sesión** será la «frutilla del postre» :)
Esta función se encargará de buscar las variables de sesión dentro de `beaker.session` en `environ`. De encontrarla nada malo podría pasar pero de no encontrarla, deberá destruir «las eventuales variables de sesión residuales que puedan quedar» e invocar a la función de solicitud de datos del módulo de usuarios. Por esto motivo, la **función verificar** será un decorador que envuelva a aquellas funciones cuyo acceso se

encuentre restringido. Veamos cómo lo logrará:

```
from usuarios import ingresar


def verificar(funcion_restringida):

    def wrapper(*args, **kwargs):
        # Comienzo diciendo que el usuario no está logueado hasta comprobarlo
        usuario_logueado = False

        # Obtengo el diccionario environ (suponiendo es el primer argumento)
        environ = args[0]

        # Verifico si las variables de sesión están disponibles
        existe_var_logged = 'logged' in environ['beaker.session']
        existe_var_uname = 'username' in environ['beaker.session']
        # Si están disponibles, modifico el valor de usuario_logueado con el de logged
        if existe_var_logged and existe_var_uname:
            # obtengo el valor de la variable logged
            usuario_logueado = environ['beaker.session']['logged']

        # Si el usuario está logueado, devuelvo la función decorada
        if usuario_logueado:
            return funcion_restringida(*args, **kwargs)
        # Sino, destruyo residuales y devuelvo la función para loguearse
        else:
            destruir()
            return ingresar()

    return wrapper
```

Switcheando solicitudes en el controlador

Una vez preparados los módulos de usuarios y sesiones y adaptado el controlador, habrá que *switchear* las URL teniendo en cuenta la siguiente tabla de referencia:

Acción/Funcionalidad	URL	Función a invocar (parámetros)
Ingresar al sistema (muestra formulario de *logueo*)	`/usuarios/ingresar`	`usuarios.ingresar()`
Verificar los datos del *loguin* (*action* del formulario anterior)	`/usuarios/validar`	`usuarios.validar(environ, environ)`
Finalizar una sesión	`/usuarios/salir`	`sessions.destruir(environ)`

Restringir el acceso

Para restringir el acceso a una función determinada solo bastará con decorarla con la función verificar del módulo de sesiones:

```
from core.sessions import verificar

@verificar
def funcion_con_acceso_restringido():
    pass
```

ORIGINAL
COPY

The Original Hacker #_9_

Copyright 2013 - Eugenia Bahit
Creative Commons BY-NC-SA
SafeCreative Work:__409272207688__
safecreative.org/work/__409272207688

safecreative
1 409272 207688
INFO ABOUT RIGHTS