¡Denme un shell y moveré al mundo! o ¿Por qué la linea de comandos no es una interfaz anticuada?

Pablo L. De Nápoli

GlugCEN - Grupo de Usuarios de Software Libre
de la Facultad de Ciencias Exactas y Naturales - Universidad de Buenos Aires
SoLAR - Software Libre Argentina

25 de abril de 2015

Motivación de la Charla

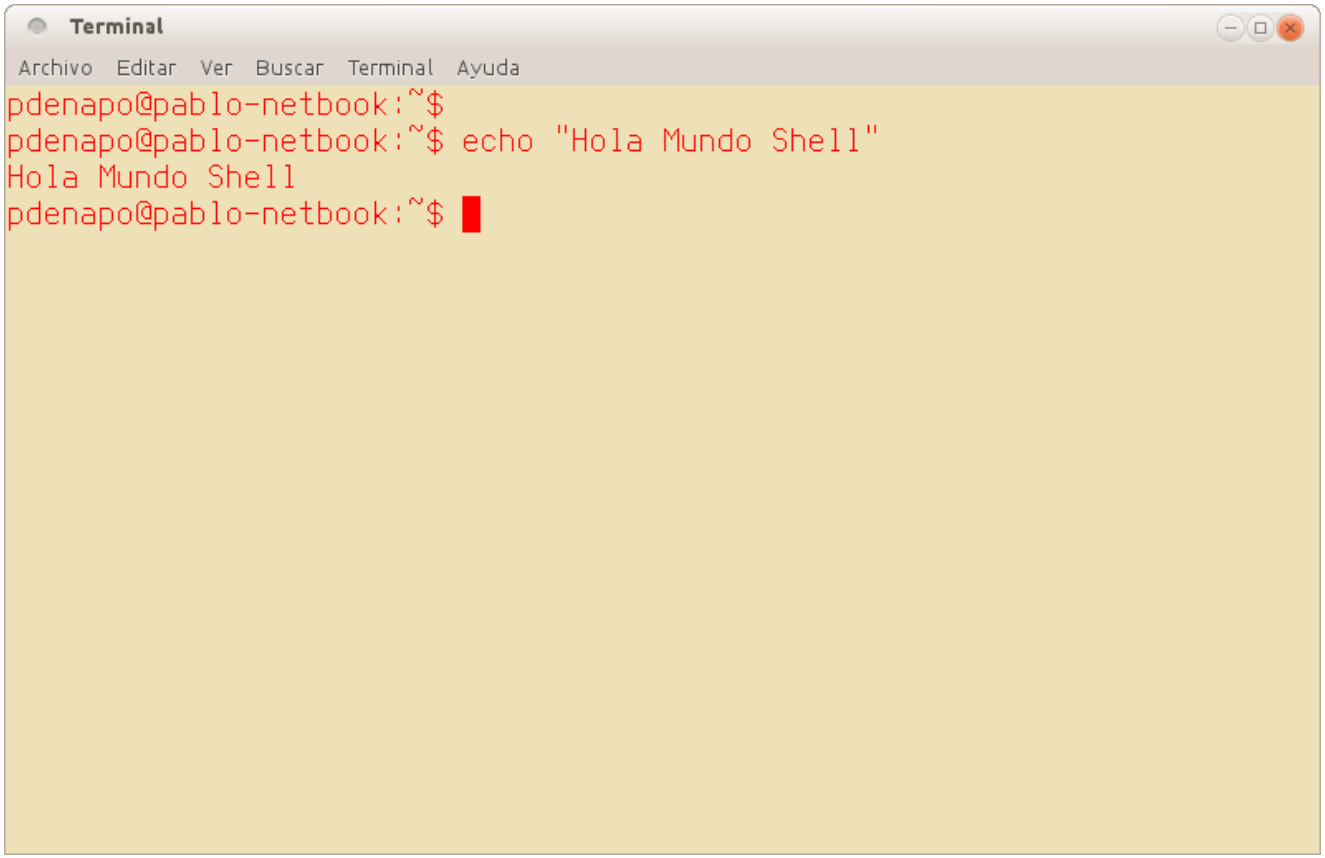
- El shell o intérprete de comandos es un progama fundamental del sistema operativo. Su misión es tomar órdenes del usuario y ejecutarlas.
- Cuando se creó el sistema Unix (hacia 1970), era la única interfaz disponible.
- Actualmente disponemos de muchas interfaces gráficas muy amigables para el usuario (como KDE, Gnome, etc.), y muchos usuarios tienden a pensar que el intérprete de comandos es una interfaz anticuada. Creen que es como usar el MS-DOS.
- En esta charla, trataré de convencerlos, con ejemplos prácticos, de qué esto no es así.

Parte I

¿Qué es el Shell (intérprete de comandos) de Unix / GNU Linux ?

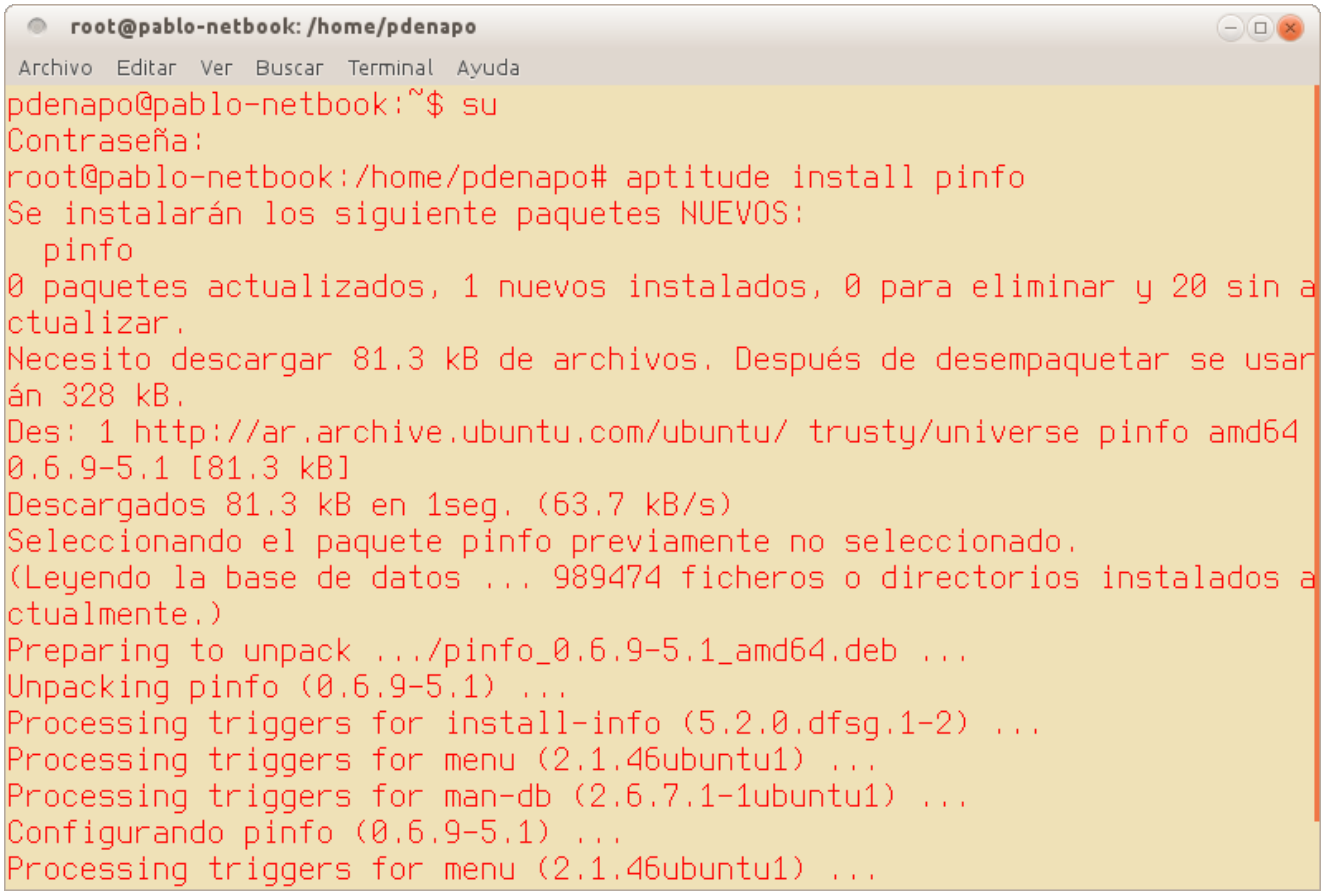
¡Hola mundo del Shell !

Abramos una terminal desde nuestro entorno gráfico. Aparecerá el prompt (apuntador) $ que indica que el Shell espera una orden.
Podemos entonces introducir y ejecutar un comando. Veamos por ejemplo la orden echo que imprime un mensaje:



Ejecución de comandos como root

El prompt es $ para un usuario común, sin privilegios; y # para el usuario root (administrador del sistema). Para poder realizar ciertas tareas administrativas, como por ejemplo instalar programas, se necesita convertirse en root, mediante los comandos su o sudo.



El Shell o Los Shells

- Existen en realidad un montón de intérpretes de comandos diferentes. El más popular es Bash (Bourne again shell) del proyecto GNU.

  En esta charla trabajaremos con él.

- Otras opciones son csh, dash, ksh, tcsh, zsh, etc.

Algunos Comandos de Uso Frecuente

ls	listar archivos
rm	borrar archivos
mv	mover o renombrar archivos
cp	copiar archivos
ln	crear un enlace a un archivo
cat	imprimir el contenido de un archivo
cd	cambiar el directorio actual
pwd	imprimir el directorio actual
mkdir	crear un directorio
rmdir	borrar un directorio
man	mostrar la página de manual de un comando
info / pinfo	mostrar las páginas de ayuda en formato info
ps / top / htop	ver los procesos que se están ejecutando
kill	matar a un proceso que se está ejecutando
exit	salir del shell actual

Comandos Internos y Externos

Existen dos clases de comandos:

- Comandos internos: Son comandos que el shell sabe ejecutar directamente. Por ejemplo: exit, echo, cd
- Comandos externos: Son realmente programas que se ejecutan. Esos programas están contenidos en un archivo (Pueden ser en formato binario, o en lenguaje que es interpretado por un intérprete. Esto es transparente para el usuario.)

Ejemplo 1: El comando ls, es realmente un programa contenido en el archivo /bin/ls, en formato binario.

Ejemplo 2: El comando xzless que sive para ver algunos archivos comprimidos, es realmente un programa en el lenguaje del shell Bash (que a su vez ejecuta a los programas xz y less).

Sintaxis de los comandos

Un comando en un sistema Unix, tiene generalmente la siguiente sintaxis:

```
[ruta de búsqueda/]comando [opciones] [argumentos]
```

La ruta de búsqueda indica en qué directorio está el archivo que contiene el programa (comando externo) que queremos ejecutar. Si no se especifica, se lo busca en los directorios indicados en la variable de entorno PATH.
Las opciones que modifican el comportamiento del comando comienzan por - o –.
Ejemplos con el comando ls (listar archivos):

```
ls -lrt
```

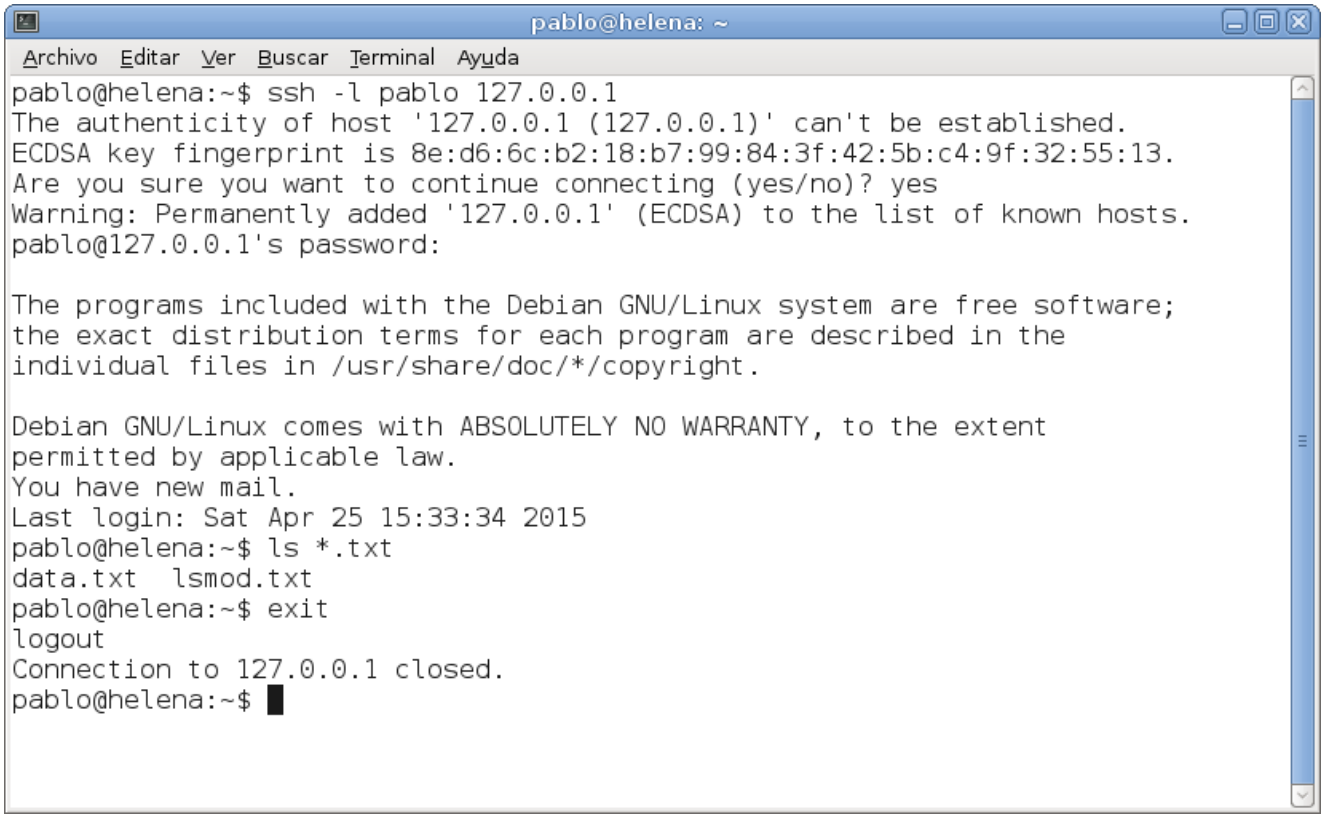
```
ls --reverse *.txt
```

```
/bin/ls --color
```

Algunos programas útiles en la consola

nano, joe, vi, emacs	editores de texto
mc (Midnight Commander)	manejador de archivos
wget	descargar un archivo de internet
lynx, links	navegador web modo texto
lftp	cliente ftp en la consola
ssh (Secure Shell)	ejecución remota de un shell
mpg321 / orpheus	reproducir archivos de sonido en mp3
alsamixer	control del mezclador de sonido

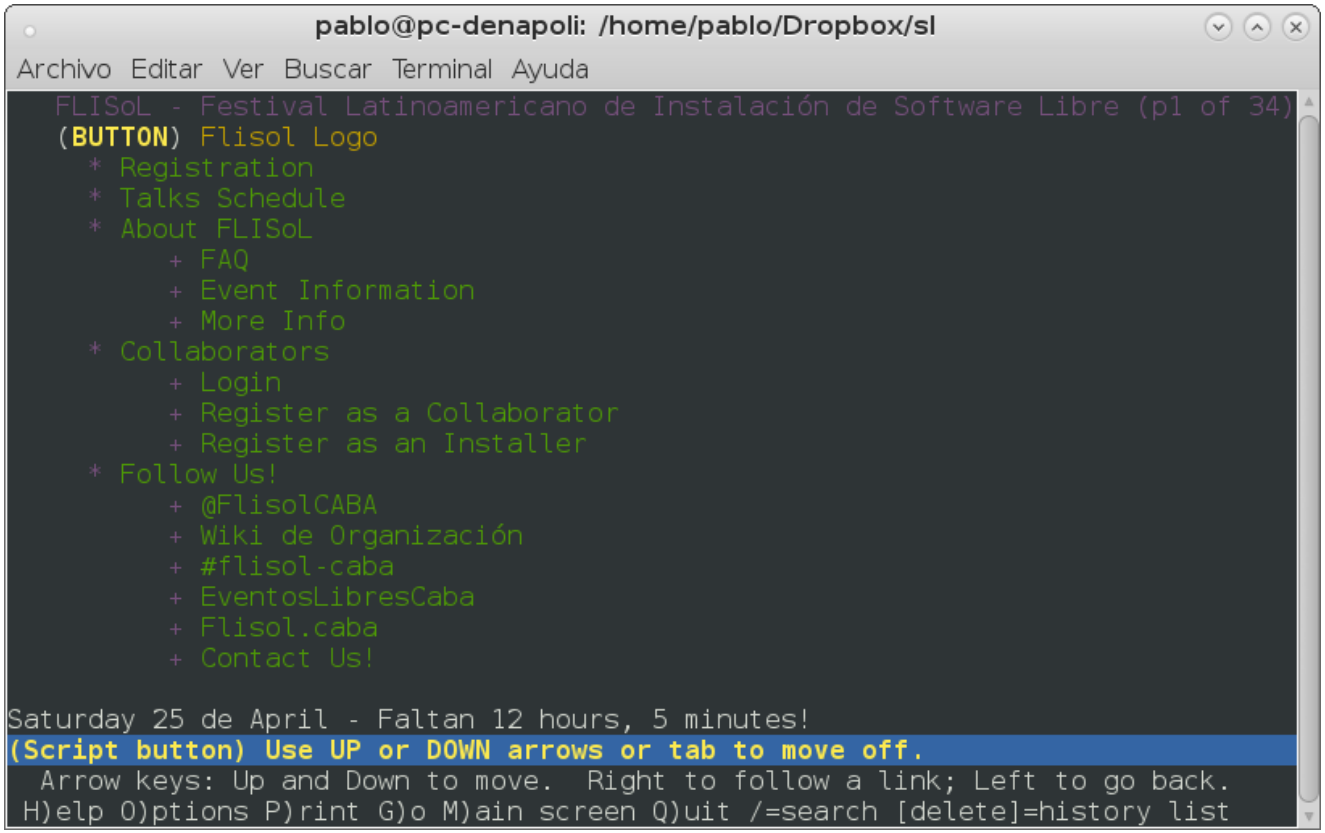
Ejemplo: ejecución de un shell remoto por ssh



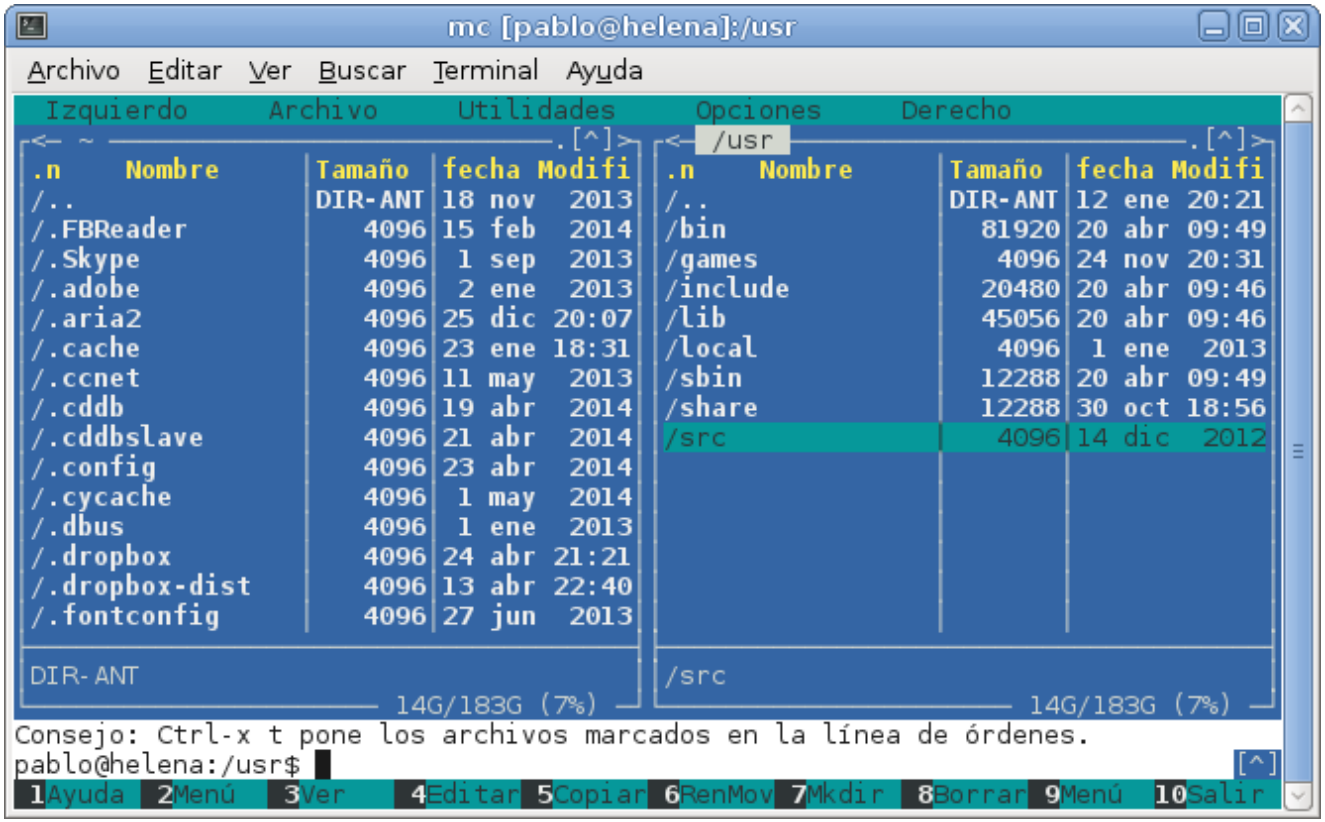
(acá hice trampa pues 127.0.0.1 es mi máquina local, pero funcionaría exactamente igual con la ip o nombre de una máquina remota)

Ejemplo: Navegando la página del Flisol con Lynx

```
lynx http://flisol.usla.org.ar/sede/CABA/
```



Ejemplo: Manejando archivos con el Midnight Commander



Para nostálgicos del Norton Commander...

Parte II

Los programas pueden hablar con otros programas

Algunas palabras sobre el diseño de un sistema Unix

- El sistema Unix fue diseñado como un conjunto de pequeñas piezas simples. Idealmente, cada programa debe concentrarse en realizar una sóla tarea, y ¡realizarla bien!
- Los programas pueden interactuar entre sí mediante las facilidades provistas por el sistema.
- Como veremos, las interfaces orientadas a la linea de comandos permiten que los programas se comuniquen entre sí.
- En esta parte de la charla, describiremos dos de las facilidades provistas por el shell para ello: la redirección y las tuberías.
- Para conocer más sobre la filosofía de Unix, les recomiendo el libro The Art of Unix Programming por Eric Steven Raymond.

La redirección

Normalmente un comando de Unix lee datos del teclado (standard input) y escribe sus resultados en la pantalla (standard output).

La redirección es una facilidad del shell, que nos permite redirigir la entrada o la salida de un comando hacia un archivo. Veamos algunos ejemplos:

Como vimos antes, el comando echo normalmente muestra un mensaje en la pantalla. Pero si hacemos:

```
echo "Hola mundo!"> mi_arhivo.txt
```

estaremos enviando la salida a un archivo. La redirección de la salida se indica con el símbolo $>$. Podemos comprobarlo con el comando:

```
cat mi_arhivo.txt
```

Redirección de la entrada y la salida

La redirección de la entrada se indica con el símbolo $<$, y la de la salida con el símbolo $>$.

Un ejemplo: el comando inconv se usa para convertir un archivo de una codificación a otra. Por ejemplo tenemos un archivo viejo en la codificación latin1, y lo queremos convertir a utf8. Lo hacemos mediante el comando:

```
iconv -f latin1 -t utf8 < viejo.txt > nuevo.txt
```

Tuberías

Una tubería es un mecanismo mediante el cuál la salida de un programa sirve como entrada para otro. En el shell, se indica mediante el símbolo | .
Veamos algunos ejemplos:

```
ls *.sh | sort --reverse
```

Mejor usar el comando equivalente:

```
ls -1r *.sh
```

```
ls  | more
```

```
ls  | less
```

```
ls | sort --reverse | less
```

Parte III

El Shell como lenguaje de programación

El shell como lenguaje de programación

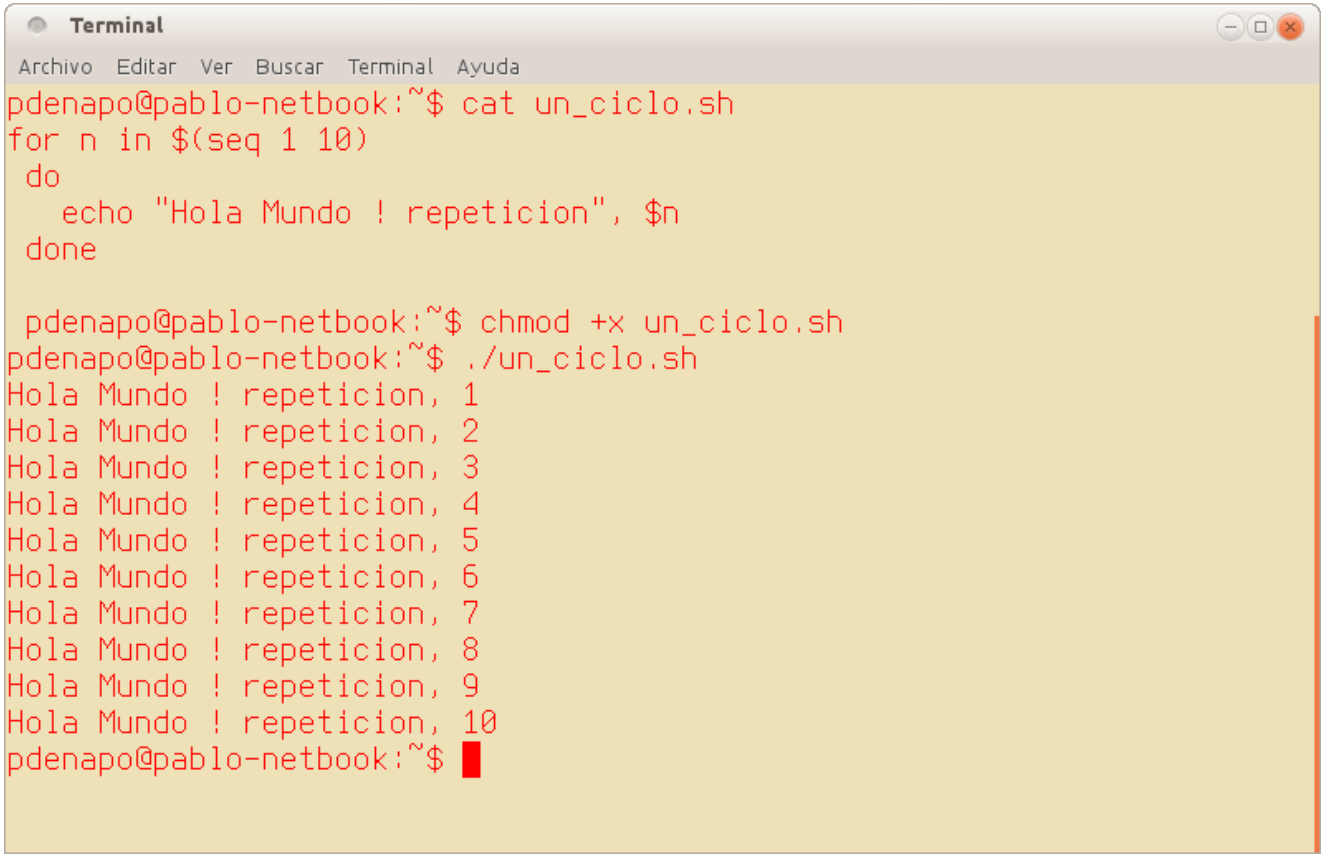
- El shell posee realmente un lenguaje de programación con variables, sentencias condicionales y ciclos, estructuras de datos como arrays, definición de funciones, etc.
- En este sentido Bash es mucho más potente que el primitivo intérprete de comandos del MS-DOS (COMMAND.COM). ¡Es posible escribir programas completos y útiles en el lenguaje del shell!.
- En esta charla, ilustraremos como hacer ésto (y porqué podríamos querer hacerlo) con algunos ejemplos simples.
- Los programas en el lenguaje del shell se denominan habitualmente shell scripts.

El shell como lenguaje de programación (2)

- Los shell scripts son archivos de texto que podemos crear usando cualquier editor de texto. Por ejemplo yo suelo usar joe (un editor de texto en la consola) o pluma (en modo gráfico). Otras opciones populares son: nano, gedit, vi, emacs, etc.
- Por ejemplo el siguiente es un shell script que imprime 10 veces el mensaje hola mundo:

```
for n in $(seq 1 10)
 do
    echo "Hola Mundo ! repeticion", $n
 done
```

El shell como lenguaje de programación (3)



Generando una secuencia de números

Expliquemos mejor como funciona este ejemplo: el comando seq genera una secuencia de números.

```
$ seq 1 5
1
2
3
4
5
```

Con la sintaxis usada en el script, podemos capturar la salida en una variable:

```
$ s=$(seq 1 5)
$ echo $s
1 2 3 4 5
```

Parte IV

Un par de ejemplos de uso del Shell

Necesito hacer siempre lo mismo...

Supongamos que necesitamos realizar alguna tarea en forma repetida, pero que no queremos acordarnos los detalles de cómo hacerlo.

Por ejemplo, mi madre necesita leer un libro en formato pdf usando el programa atril. Pero no quiero que ella tenga que recordar donde está el archivo, o con qué programa tiene qué abrirlo. Entonces creé un shell script como el que muestro:

```
#!/bin/bash
/usr/bin/atril "/home/mama/que_archivo.pdf"
```

La primera línea le dice al sistema cuál es el intérprete de comandos con el cuál el archivo será ejecutado: en este caso el shell bash.
La segunda linea es una orden que le dice al shell que ejecute el programa atril, pasándole como parámetro la localización del archivo.

Después creé un archivo .Desktop para asignarle un ícono y un lugar en el menú de la interfaz gráfica para que este archivo sea ejecutado cuando mi madre hace click en el ícono.

Bajar una galería de imágenes

Queremos bajar una galería de imágenes en jpg. Investigando un poco, vemos que sus direcciones URL se forman por números que van variando. Entonces utilizamos el comando seq para generar la secuencia de números, y wget para bajarlas.

```
donde="http://www.all-free-photos.com/images/grece"
for n in $(seq  205 300)
do
  archivo="PI73${n}-hr.jpg"
  wget $donde/$archivo
  tipo=$(file -ib $archivo | cut -f 1 -d\;)
  if [ "$tipo" == "text/html" ]; then
     echo "$archivo no es una imagen, la borro!"
     rm $archivo
  fi
done
```

Explicando los trucos del ejemplo

En el ejemplo anterior, usamos el comando file, que devuelve el tipo MIME de un archivo, para detectar cuando un archivo jpg es en realidad un documento html, y borrarlo.

```
$ file -ib verdadera_imagen.jpg
image/jpeg; charset=binary
$ cat falsa_imagen.jpg
<html>
<body>
Falsa imagen!
</body>
</html>
$ file -ib falsa_imagen.jpg
text/html; charset=us-ascii
```

Explicando los trucos del ejemplo

En el script anterior, usamos una tubería con el comando cut para quedarnos con la primer parte de la salida del comando file hasta el ;.

```
$ file -ib verdadera_imagen.jpg | cut -f 1 -d\;
image/jpeg
$ file -ib falsa_imagen.jpg | cut -f 1 -d\;
text/html
```

La salida se captura en la variable tipo mediante la linea

```
tipo=$(file -ib $archivo | cut -f 1 -d\;)
```

Parte V

Otros usos típicos del Shell

Ejecución de programas en el arranque del sistema

Podemos escribir un script que se ejecute cuando la máquina arranque, para que se inicie cierto servicio.

Algunos casos en los que tuve que hacerlo fueron:

- Cuando tenía Speedy como proveedor de Internet, necesité escribir un script para que la máquina se conectara automáticamente a internet.
- En un servidor instalé el programa Rhodecode (similar a Github para Mercurial). Escribí un script para que el programa se ejecute en el arranque del servidor.

La forma exacta de hacer esto, depende del sistema de arranque que use nuestra distribución de GNU/Linux. Tradicionalmente se empleaba System V init, pero las distribuciones mas modernas usan systemd o upstart.

Personalizando nuestra sesión de Bash

Similarmente, cuando Bash arranca ejecuta una serie de archivos como /etc/bash.bashrc (para todos los usuarios), /.bashrc (para un usuario particular), etc. Uno puede querer modificarlos para, por ejemplo, establecer variables de entorno (como PATH) que afectan el comportamiento de los programas.

Ejemplo: Supongamos que hemos instalado el programa axiom en el directorio /usr/local/axiom/bin, y queremos que todos los usuarios puedan ejecutarlo. Entonces la orden

```
export PATH=$PATH:/usr/local/axiom/bin
```

colocada en el archivo /etc/bash.bashrc, añadirá el directorio /usr/local/axiom/bin a la ruta de búsqueda.

¿Cómo le indico cómo hacer algo a alguien?

Otro uso muy útil del shell, es cuando uno quiere explicale a otro cómo exactamente hacer algo ...

Con las interfaces gráficas, esto es muy difícil de hacer: distintos sistemas GNU/Linux pueden usar interfaces gráficas diferentes y tener un aspecto visual muy diferente. Entonces ¿Cómo te digo exactamente donde tener que hacer click? ¿Cuál es el ícono? ¿Donde encontrarlo? ¡Es una pesadilla!

Usando el shell, podemos darle a alguien instrucciones exactas.

Un ejemplo:

Las instrucciones para instalar Axiom (desde las fuentes) en Ubuntu, en su página web http://www.axiom-developer.org/axiom-website/download.html son:

```
echo 0 >/proc/sys/kernel/randomize_va_space
apt-get install m4 libxpm-dev libxt-dev libx11-dev
apt-get install libxext-dev gettext git-core texlive gawk
texlive-fonts-extra
git clone git://github.com/daly/axiom.git
cd axiom
export AXIOM=‘pwd‘/mnt/ubuntu
export PATH=$AXIOM/bin:$PATH
make
```

Para aprender más...

- Bash Guide for Beginners por Machtelt Garrels

  http://www.tldp.org/LDP/Bash-Beginners-Guide/html/

- Advanced Bash-Scripting Guide:
  An in-depth exploration of the art of shell scripting
  por Mendel Cooper

  http://www.tldp.org/LDP/abs/html/

- Bash Reference Manual

  https://www.gnu.org/software/bash/manual/bash.html

The End

¡MUCHAS GRACIAS POR VENIR!

¿PREGUNTAS? ¿COMENTARIOS?