

## [Guía de iniciación a la shell de GNU/Linux](http://hwagm.elhacker.net/guia-linux/guia-linux.htm) (<http://hwagm.elhacker.net/guia-linux/guia-linux.htm>)

### **Prologo**

El presente articulo solo pretende ser una forma de acercarse de una forma rápida y cómoda en castellano a Linux, conforme he ido yo personalmente adquiriendo conocimientos sobre este fantástico sistema operativo.

También pretende ser una guía de mano de consulta para las posibles lagunas mentales que podamos tener a la hora de usar los distintos comandos.

Básicamente voy a tratar el uso de la terminal (shell) y los comandos que mas nos interesan a la hora de tratar con ficheros y configuraciones de dispositivos.

GNU/Linux, que es?

Un sistema operativo, bien, eso ya lo sabemos ¿pero que supone ?

Empecemos por su nombre o denominación. GNU/Linux proviene de la unión de GNU ( GNU is not Unix) y Linux, un sistema operativo pensado en primera instancia para uso recreativo por su desarrollador Linus Torvalds. El cual gracias a la colaboración de muchos desarrolladores y la posibilidad de portabilidad de aplicaciones gracias al proyecto GNU de la Free Software Foundation de Cambridge, es hoy el sistema potente y complejo pero cada vez mas accesible que conocemos.

Bueno si queremos saber sobre los principios de Linux y las diferencias que existen en comparación con Unix, hay multitud de artículos y papers que tratan sobre ello en internet. Vamos a lo que nos importa.

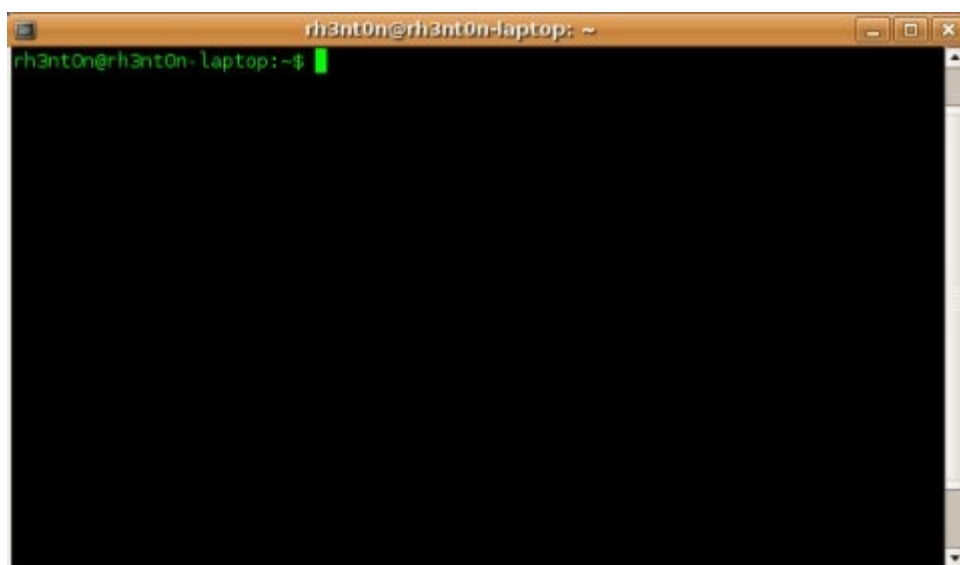
### **Capitulo primero.**

#### **Principios y Comandos Básicos.**

#### La Shell o interprete de comandos de GNU/Linux.

La Shell es el medio que tenemos para interactuar con la maquina con la que estamos trabajando y con su sistema operativo, literalmente interpreta ordenes.

Esto es una shell:



Hay muchos tipos de shell, la mas popular es 'bash' (bourne again shell) fue creada por J.Bourne, en principio creó 'bsh' después la mejoró, y la renombró añadiendo la 'a'

## La Shell se encarga de:

1º De interpretar las variables de entorno \*

2º Interpreta los metacaracteres,(caracteres comodines) '\*' '?'.

3º Maneja la entrada y salida standard de los comandos.

4º Busca alias, y los interpreta.

5º Interpreta las ordenes comparando si son comandos internos de la shell, o binarios ejecutables, mirando en \$PATH.

\* Las variables de entorno son porciones de memoria a las que se asigna un valor, por ejemplo una variable de entorno seria el propio 'bash' (\$BASH=/bin/bash), lo cual nos indica que cuando se llama a la variable \$BASH, la shell busca en las variables, e interpreta que tiene que ejecutar: /bin/bash. Esto es útil cuando veamos en otros capítulos, la iniciación a el shell-script.

### Principales diferencias con la línea de comandos de windows.

En la terminal de Linux (de ahora en adelante shell), el interprete de comandos distingue entre mayúsculas y minúsculas a la hora de interpretar los comandos, cosa que en la línea de comandos (de ahora en adelante cmd.exe) de Windows no ocurre:

#### Shell:

ls no es igual a LS

#### cmd.exe:

dir es igual a DIR

En la shell los modificadores de los comandos se representan con un guión '-', y en cmd.exe el modificador es '/'.

#### Shell:

ls -l

#### cmd.exe:

dir /all

En la shell no son estrictamente necesarias las extensiones de fichero, solo se usan para reconocer visualmente los archivos.

### El Prompt:

Hay muchos tipos de prompt, los mas usuales son '\$' para la terminal de usuario y '#' para la terminal de root.

Cuando abrimos una terminal, la única información en principio con la que contamos es el prompt, este nos indica el usuario con el que estamos accediendo a la shell, la maquina sobre la que esta corriendo la shell y el directorio donde nos encontramos, el cual lo normal sea nuestro directorio home.

ejemplos de prompt:

```
rh3nt0n@rh3nt0n-laptop:~$
```

```
root@rh3nt0n-laptop:/home/rh3nt0n#
```

El primer prompt nos indica que el usuario es rh3nt0n que esta en la maquina rh3nt0n-laptop en el directorio home de rh3nt0n (/home/rh3nt0n) y que la terminal es de usuario '\$'.

El segundo prompt nos indica que el usuario es root el cual esta en la maquina rh3nt0n-laptop, dentro del directorio /home/rh3nt0n y que la terminal es de root '#'.

### Principales diferencias entre usuario y root..

El usuario root, es el administrador del sistema es el amo y señor, es el que asigna permisos a otros usuarios, crea grupos, usuarios etc.. en fin tiene todos los derechos de hacer lo que quiera, es por esto que no es recomendable usar de continuo este usuario para el día a día de uso del sistema, ya que por un descuido o un comando mal efectuado, podemos causar graves daños al sistema o perder información de la cual no nos queramos desprender.

El usuario, depende de root para tareas de administración del sistema y configuración de

dispositivos. Pero para el trabajo con ficheros propios o navegación por internet y el sistema de archivos, el usuario por defecto no tendría ningún problema para usar el sistema.

En Linux no existen unidades, existen particiones del disco duro.

En Linux todo es un fichero, el monitor estar representado como fichero así como el teclado el mouse etc. Por ejemplo:

hda ---- representaría a todo el disco duro

hda1 ---- representaría a la partición 1

hda2..... representaría a la partición 2

El sistema de ficheros de Linux representa un árbol invertido del cual la raíz, es '/', y de la cual cuelgan el resto de directorios del sistema.

Cualquier fichero que comience por un punto '.' (eje: .fichero) es un fichero oculto.

Para cualquier usuario el símbolo de la ñ '~' en el prompt, representa su directorio home. a la hora de buscar ficheros existen los meta caracteres:

'\*' el asterisco representa a cualquier fichero o carácter

'?' la interrogación representa un carácter.

### Sistema de ficheros.

/<-----directorio raíz

|  
|\_\_\_\_\_ /bin <-----almacena binarios (ejecutables)  
|\_\_\_\_\_ /etc <-----contiene ficheros de configuración de programas.  
|\_\_\_\_\_ /media <----- dispositivos tipo CDROM, floppy  
|\_\_\_\_\_ /root <-----directorio home de el administrador del sistema  
|\_\_\_\_\_ /dev <-----contiene los ficheros de dispositivos tipo monitor, teclado etc  
|\_\_\_\_\_ /home <----- contiene los directorios home de los usuarios normales  
|\_\_\_\_\_ /sbin <-----contiene ficheros imprescindibles, accesible a root  
|\_\_\_\_\_ /tmp <-----ficheros temporales  
|\_\_\_\_\_ /var <-----ficheros variables eje: fichero.log  
|\_\_\_\_\_ /usr <-----tiene una estructura parecida a el directorio raíz.(explicado mas abajo)  
|\_\_\_\_\_ /lib <-----contiene librerías dinámicas.  
|\_\_\_\_\_ /boot <-----contiene ficheros de arranque de sistema  
|\_\_\_\_\_ /mnt <-----aquí se montan los dispositivos extraíbles.

/usr

|\_\_\_\_\_ /bin <-----binarios ejecutables de usuario  
|\_\_\_\_\_ /local <-----aquí van todas las compilaciones  
|\_\_\_\_\_ /src <-----aquí deberían de compilarse los programas de los usuarios.  
|\_\_\_\_\_ /share <-----ficheros de entorno gráfico.  
|\_\_\_\_\_ /include <-----archivos de cabecera que usan las aplicaciones.  
|\_\_\_\_\_ /lib <-----enlaces dinámicos a librerías.

### Permisos de los ficheros..

Los ficheros se protegen con permisos, los cuales pueden ser de lectura 'r', escritura 'w', y ejecución 'x'. Los cuales se asignan a los tres tipos de usuarios que puede tener un fichero, dueño, grupo. otros, y se representa de esta manera en la terminal:

drwxr-xr-x 2 root root 4096 2006-10-19 00:18 bin

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Columnas:

La primera es del tipo de fichero: 'd' si es directorio, '-' si es fichero regular, 'c' fichero de caracteres. 'b' fichero por bloques.

1. Permisos de los usuarios, se agrupan de tres en tres: [rwx] [r-x] [r-x], el primer grupo de permisos, corresponde a el dueño, el segundo a el grupo, y el tercero a 'otros'.
2. El numero de nombres que tiene el fichero.
3. Dueño
4. Grupo
5. Tamaño en bytes
6. Fecha y hora de la ultima modificación.
7. Nombre del fichero.

Cada permiso esta representado por una cifra en octal:

r	w	x
4	2	1

Por lo que si un fichero tiene los siguientes permisos: rwx rw- r--, su valor en octal sera 764. Esto nos servirá para cambiar a nuestro criterio los permisos de cada fichero del que seamos dueños.

### Comandos básicos para moverse en la Shell.

Para empezar aclaro algo importante, cada vez que se crea un directorio, este dentro de si mismo crea dos ficheros:

- '.' que representa el directorio actual.
- ..' que representa el directorio padre, o anterior.

```
rh3nt0n@rh3nt0n-laptop:~/guia$ ls -la nuevo/
total 8
drwxr-xr-x 2 rh3nt0n rh3nt0n 4096 2006-10-21 16:03.
drwxr-xr-x 3 rh3nt0n rh3nt0n 4096 2006-10-21 16:03..
```

Y los crea con unos permisos predeterminados por el sistema. en realidad, son enlaces a otros directorios como dije anteriormente y así consta en la primera columna 'd'.

Dicho esto pasamos a los comandos:

### Comandos para movernos por el Árbol de directorios..

**Comando ls:** abreviatura de list (listar)

sintaxis: ls [opciones] [fichero]

opciones: -l vemos todos los datos de los ficheros del directorio actual.

-ld vemos los permisos del directorio actual.

-ldi vemos el numero de Inodo del fichero en cuestión \*

-R muestra de forma recursiva, los ficheros que halla dentro de los directorios que se encuentren dentro de el directorio que estamos listando.

-IS lista los ficheros de mayor a menor en tamaño de bytes.

-ISr lo contrario que el anterior.

-t ordena los ficheros por fecha de modificación.

-tr forma inversa a la anterior opción.

**Comando pwd:** print working directory, imprime en la pantalla el directorio actual.

ejemplo:

```
rh3nt0n@rh3nt0n-laptop:~/guia$ pwd
/home/rh3nt0n/guia
```

Esto es muy útil, cuando estamos dentro de muchos directorios y no sabemos donde nos encontramos.

**Comando cd:** change directory. Cambia de directorio

Sintaxis: cd [directorio]

Formas útiles de usarlo:

cd sin mas, te lleva a tu directorio home.  
cd.. te lleva al directorio padre (directorio anterior)  
cd ~ igual que el primero, te lleva a tu directorio home.

**Comando mkdir:** make diretory. Hacer directorio.

Sintaxis: mkdir [directorio]

mkdir -p [directorio]/[directorio]/[directorio]/etc..

puedes crear tantos directorios como quieras, desde el directorio actual, o desde tu home poniendo la ruta absoluta \*\*

notas:

\* inodo es el numero de sector donde se encuentran los datos en el disco duro.

\*\* ruta absoluta, es la ruta desde el directorio home del usuario.

**Comando rmdir:** remove directory. Borrar directorio.

Sintaxis: rmdir [directorio]

rmdir -p [ruta de directorios ]

**Comando cat:** Muestra el contenido del archivo en cuestión, en pantalla.

Sintaxis: cat [fichero]

**Comando more:** igual que cat, pero muestra si el contenido de el fichero es grande, pantalla a pantalla.

Sintaxis: more [fichero]

**Comando head:** muestra la cabecera del archivo, por defecto las 10 primeras líneas.

opción: -n [numero] muestra el numero de líneas que especifiques

Sintaxis: head [opción] [fichero]

**Comando tail:** al contrario que head, este comando muestra el final de el archivo. Por defecto al igual que head muestra 10 líneas.

sintaxis: tail [opciones] [fichero]

opciones: -n numero de líneas especificado

-f esta opción, es muy interesante, ya que te muestra el incremento de líneas a tiempo real.

**Comando cp:** copy. Copiar.

sintaxis: cp [opciones] [fichero origen] [directorio destino]

opciones: -b hace back up del archivo si en el directorio donde copiamos existe un fichero con el mismo nombre y la renombra añadiendo al final ~.

-- no-dereference copia forzosamente si es un enlace simbólico.

-F fuerza la copia en caso de que exista el fichero. por lo tanto sobre escribe.

-i interactivo, te pregunta en caso de que pueda sobre escribir.

-l hace enlace duro. link.

-L te copia el contenido del fichero en caso de que intentes copia un enlace simbólico.

-r copia el contenido total de un fichero y ficheros hijos recursivamente.

-s crea un enlace simbólico, soft.

-u en caso de que halla un archivo con el mismo nombre, lo actualiza.update.

**Comando mv:** move. mueve o renombra el archivo o directorio.

sintaxis: mv [opciones] [origen] [destino]

opciones: -b si es un fichero, hace back-up. si es directorio, mueve el directorio y su contenido.

-f fuerza

-i interactivo

-u actualiza si el fichero movido existe en destino

**Comando ln:** link, enlace duro por defecto (sin opciones).

Sintaxis: In [opciones] [fichero]

opciones: -b hace back-up

-f fuerza

-i interactivo

-s enlace simbolico o soft

**Comando file:** muestra el tipo de archivo de que se trata.

sintaxis: file [opciones] [archivo]

opciones: -z muestra el tipo de archivo si es un comprimido con gz.

-L en caso de que sea enlace simbolico, te muestra los datos del fichero al que apunta.

**Comando rm:** remove. borra el fichero

sintaxis: rm [opciones] [fichero/s]

opciones: -i pregunta antes de borrar

-r borra de forma recursiva todo el contenido de un directorio

si es el caso. Muy peligroso efectuarlo, usadlo con cautela.

-f fuerza el borrado sin preguntar, mas peligroso aun que el anterior si se efectúan en

conjunto: rm -fr directorio ( el caos) sed prudentes al borrar archivos.

**Comando chmod:** change mode, cambiar permisos.

sintaxis: chmod [permisos] [ficheros]

Forma octal de cambiar permisos:

Como ya he explicado anteriormente el valor de cada permiso

( lectura, escritura y ejecución) se representa con números octales,

y se agrupan en conjuntos de tres, un conjunto para el dueño, otro grupo

para el grupo, y otro grupo para otros usuarios:

4 para 'r' ---> lectura

2 para 'w' --> escritura

1 para 'x' --> ejecución

Por lo cual si tenemos un fichero con estos permisos:

`rwX rw- r--`

Tendremos que el primer grupo tiene un valor de 7, resultante de la suma de 4+2+1, el segundo grupo tendrá un valor de 6 resultado de la suma de 4+2 y el tercer grupo tendrá un valor de 4 resultante del valor de 'r'.

Por lo tanto el valor en octal de los permisos de ese fichero en cuestión será de 764.

Para cambiar los permisos de ese archivo a:

`rw- r-- --X`

La orden seria la siguiente:

`chmod 641 fichero`

Forma abreviada de cambiar permisos:

La sintaxis seria la misma, pero variarían las opciones, para mi, es mas cómodo y menos laborioso que la anterior manera, pero de la forma anterior podemos sacar partido a el comando umask que veremos posteriormente.

Opciones:

u	dueño	r	lectura	+	dar permiso
g	grupo	w	escritura	-	quita permiso
o	otros	x	ejecución	=	igual

Por lo tanto una orden para dejar los permisos como la orden del otro modo seria:

chmod u+rw,g+r,o+x fichero

**Comando umask:** te dice la mascara de permisos del fichero en cuestión.

Sin especificar la nueva mascara te dice la mascara del directorio.

sintaxis: umask [mascara]

Si queremos que los archivos que creemos tengan un valor distinto a el valor por defecto habría que restar a 666 (por defecto rw-rw-rw-) el valor en octal que queramos para nuestros

archivos:

Si por ejemplo queremos que nuestros archivos se creen con permisos

r-- para dueño, r-- para grupo y r-- para otros, deberíamos restar 444

a 666 y nos daría 222, entonces el comando seria el que sigue:

umask 222

**Comando touch:** crea un fichero

sintaxis: touch [fichero]

**Comando which:** busca el fichero/s binario especificado/s.

Te devuelve la ruta absoluta donde se encuentra/n

sintaxis: which [fichero] [fichero]

**Comando id:** muestra nombre de usuario y grupo

**Comando su:** te permite cambiar al usuario que desees.

sintaxis: su usuario

password

[Comandos de ayuda.](#)

La terminal de Linux se caracteriza por su ayuda, sabiendo usarla, no se debería tener ningún problema a la hora de interactuar con ella.

Los mas útiles:

**Comando man:** manual, especificando el comando a usar te muestra la ayuda y opciones que tiene el comando

sintaxis: man [comando]

**Comando apropos:** propósito, te muestra los comandos relacionados con la palabra objeto.

sintaxis: apropos [palabra]

Esto no es un comando, en realidad es una opción que tienen todos los comandos en Linux, **la opción - - help.**

sintaxis: [comando] - - help

**Comando Whatis:** que es

rh3nt0n@rh3nt0n-laptop:~\$ whatis ls

ls (1) - list directory contents

**Comando uname:** este comando te devuelve información sobre el sistema que estas usando.

sintaxis: uname [opción]

rh3nt0n@rh3nt0n-laptop:~\$ uname -a

Linux rh3nt0n-laptop 2.6.15-27-386 #1 PREEMPT Sat Sep 16 01:51:59 UTC 2006 i686 GNU/Linux

opciones: -a da toda la información.

-r versión del kernel

**Comando date:** te devuelve la fecha actual, esto es usado en shell-script con opciones.

sintaxis: date + %[opcion]

sintaxis: date -s [formato de fecha] <----- cambia la hora y fecha del sistema  
recomiendo usar la opción - - help para este comando ya que tiene multitud de

opciones.

**Comando who:** quien, aporta información sobre quien esta usando el sistema.

sintaxis: who [opción]

opciones: -b system boot, hora y fecha de cuando arranco el sistema.

-H para ver en columnas quien esta y mas información.

-q cuantos usuarios

-r nivel de ejecución

**Comando cal:** devuelve el calendario mensual actual.

Especificando mes y año en numero te devuelve el calendario pedido.

sintaxis: cal [mes] [año]

opciones: -m cambia el formato del calendario a Europeo, empezando la semana por lunes.

-y devuelve el calendario de todo el año.

Hasta aquí llega la descripción de los comandos mas usados en la terminal, espero que os sea de ayuda.



## Capítulo segundo.

En este capítulo veremos algunos aspectos interesantes de la shell y el uso de los comandos, conoceremos aspectos de las variables del sistema, como añadirlas como editarlas, redirección de la entrada/salida estándar de los comandos, los filtros de búsqueda, iniciación y manejo de los procesos, entre alguna que otra utilidad.

Empecemos.

Antes de nada me gustaría mencionar un comando que a mi modo de ver es muy interesante, me refiero al comando 'alias', con este comando se puede dar un alias a un comando determinado.

Aquí vemos como le doy como alias a iwconfig 'red'

```
rh3nt0n@rh3nt0n-laptop:~$ alias red='iwconfig'
rh3nt0n@rh3nt0n-laptop:~$ red
lo no wireless extensions.
irda0 no wireless extensions.
eth1 no wireless extensions.
sit0 no wireless extensions.
```

Cabe resaltar que el alias espira cuando cerramos la shell donde la hemos creado, pero esto es fácilmente subsanable editando el archivo '.bashrc' de nuestro home, añadiendo el alias al final del archivo, este archivo se encarga de la configuración inicial de nuestra shell, y ejecuta automáticamente las ordenes que contiene al iniciar la shell. Esto es muy bueno, por ejemplo para tema de configuraciones de red etc.

Para eliminar un alias, solo hay que ejecutar el siguiente comando:

```
unalias -a alias_creado
```

### Concatenación de comandos en la shell.

Cuando estamos utilizando la shell, podemos hacer que los comandos o aplicaciones actúen en el 'back-ground' o en el 'front-ground'.

Que es esto? muy sencillo, al ser linux un sistema operativo multitarea, la shell puede trabajar con muchos procesos a la vez, aunque solo se puede ejecutar un proceso cada vez mientras los demás están preparados o dormidos. Este tema de los procesos lo veremos mas adelante.

Los procesos en back-ground, son los que se ejecutan invisiblemente para el usuario y dejan libre la shell para seguir trabajando con ella.

los procesos en front-ground, son justo lo contrario, son procesos que se ejecutan en primer plano y ocupan la shell el tiempo que están en funcionamiento.

También podemos unir una serie de ordenes, e interactuar con la salida de cada orden aprovechando esa salida para que sirva de argumento para otra orden esto lo entenderemos mejor viendo unos ejemplos:

### Modos de invocar ordenes:

orden &

El **umbersand '&'**, hace que esta orden se ejecute en back-ground y nos dejaría la shell libre para poder seguir usando otros comandos.

orden1 ; orden2

El punto y coma ';' hace que se ejecuten las ordenes una a una, independientemente de que una u otra funcione.

```
rh3nt0n@rh3nt0n-laptop:~$ cd .. ; ls
rh3nt0n
rh3nt0n@rh3nt0n-laptop:/home$
```

Como se puede observar, en el anterior comando, se ejecuta '`cd..`' y después se ejecuta '`ls`', sobre el directorio al que se sube con '`cd..`'

`orden1 | orden2`

El pipe '|' hace que la salida de la orden1, de argumentos a la orden 2. Ejemplo:

`dmesg | grep ipw`

Estos comandos los explicare mas adelante, la orden dmesg, da los argumentos a grep para que filtre el texto especificado .

`orden1 `orden2``

Estas comillas '`' ejecutan la orden2 que esta dentro de las comillas y pasan la salida de este como argumento a la orden1.

`apt-get install kernel-headers`uname -r``

Aquí el resultado de `uname -r` daría argumentos al comando `apt-get`

`orden1 && orden2`

En esta opción la orden2 se ejecutaría si orden1 termina sin errores.

`make && make install`

`orden1 || orden2`

En esta opción las ordenes irían ejecutándose, hasta que una de ellas se ejecute correctamente ,en ese momento se detiene.

### **Histórico de ordenes:**

Un buen comando de ayuda en referencia a comandos, y a trabajar con ellos en la shell, es el comando '`history`', este comando guarda en buffer de memoria los comandos que se han ido ejecutando en la shell y con el es posible conocer cuales se han efectuado, y citarlos por su número de orden.

```
rh3nt0n@rh3nt0n-laptop:/home$ history
```

```
1 sudo iwconfig eth0 essid j
```

```
.....
```

```
174 iwconfig
```

```
175 dmesg
```

```
176 lsusb
```

```
177 iwconfig
```

```
178 lsusb
```

```
179 dmesg
```

```
180 wget http://www.bluntmen.com/ipw2200-1.1.4-inject.patch
```

```
181 bogomips
```

```
182 yes
```

```
183 yes >/dev/null &
```

```
184 jobs
```

```
185 bg 2
```

```
186 fg 2
```

```
187 kill 2
```

```
188 jobs
```

```
189 top
```

```
190 cd .. ; ls
```

```
191 dmesg
```

```
192 history
```

La primera columna indica el número de comando, en el orden en el que se ha ejecutado.

La segunda evidentemente es el comando que se ha ejecutado, es una muy buena utilidad, ya

que por ejemplo en comandos largos haciendo un history y fijándote en el número de orden, con solo citar el número de la forma que explicare ahora, se ejecutaría el comando:

Si por ejemplo de la lista anterior queremos hacer el número 180, haríamos lo siguiente:

```
rh3nt0n@rh3nt0n-laptop:/home$ !180
wget http://www.bluntmen.com/ipw2200-1.1.4-inject.patch
--21:03:51-- http://www.bluntmen.com/ipw2200-1.1.4-inject.patch
=> `ipw2200-1.1.4-inject.patch'
Resolviendo www.bluntmen.com...
```

### Variables de entorno.

Las variables de entorno, son como explique en el anterior capítulo, porciones de memoria que el sistema coge para guardar valores específicos necesarios para el sistema. Si queremos saber las variables de entorno que hay en el sistema, solo tenéis que usar el comando **'set'**.

```
rh3nt0n@rh3nt0n-laptop:/home$
rh3nt0n@rh3nt0n-laptop:/home$ set | more
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_COMPLETION=/etc/bash_completion
BASH_COMPLETION_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_SOURCE=()
.....
```

Para ejecutar una variable de entorno solo hay que citarla de esta manera:

```
rh3nt0n@rh3nt0n-laptop:/home$ echo $HOSTNAME
rh3nt0n-laptop
```

las variables siempre se citan con el signo del dólar delante \$, y es de resaltar que las variables predeterminadas del sistema, suelen estar en mayúsculas .

Nosotros podemos crear variables de entorno, las cuales pueden ejecutarse en la shell donde las hemos creado, o exportarlas para poder usarlas en cualquier terminal de la sesión que esta en marcha, aunque estas variables una vez reiniciado el sistema se pierden, esto es útil en servidores en los cuales rara vez se reinicia el sistema.

Para crear una variable de entorno seria de esta forma:

nombre\_variable=concepto

Ejemplo:

```
rh3nt0n@rh3nt0n-laptop:/home$ edad=30
rh3nt0n@rh3nt0n-laptop:/home$ nombre=rh3nt0n
rh3nt0n@rh3nt0n-laptop:/home$ echo $edad
30
rh3nt0n@rh3nt0n-laptop:/home$ echo $nombre
rh3nt0n
rh3nt0n@rh3nt0n-laptop:/home$
```

Es recomendable a la hora de crear variables de entorno, que las que hagamos nosotros, las hagamos en minúsculas, así a la hora de borrar variables, no borraremos nada que no tengamos que borrar .

Las variables de entorno, cuando las crea un usuario, solo se ejecutan en la terminal donde se crean, para poder ejecutarlas en todas las terminales tendríamos que exportarlas:

### Comando 'export'

```
rh3nt0n@rh3nt0n-laptop:/home$ export edad
```

Esto haría que en la misma sesión en la que estamos, pero en otra terminal, se pueda ejecutar esa variable sin problemas.

Para borrar una variable determinada, solo habría que usar el comando 'unset' para ello:  
**comando 'unset'**

```
rh3nt0n@rh3nt0n-laptop:/home$ export edad
```

Hay variables como **\$PWD** que se van actualizando dependiendo de donde nos encontremos y automáticamente.

#### Entrecorillado de caracteres en la shell.

Hay ordenes que requieren de argumentos los cuales habitualmente dependiendo de la forma que se quieren utilizar se usan unas comillas u otras.

```
echo "$PATH "  
echo '$PATH '  
echo \ $PATH
```

En el primer caso, " " las comillas son denominadas débiles, el comando o el argumento lo usa la shell tal y como esta escrito y lo ejecuta si viene al caso.

En el segundo caso, ' ' las comillas son denominadas fuertes ,lo cual quiere decir que la shell no actúa dentro de ellas, por lo tanto se toma literalmente el argumento.

En el tercer caso, el back slash '\' se utiliza para escapar caracteres especiales como \$ para que la shell tome el argumento como en el segundo caso.

Las comillas `` se usan en el caso de las variables para que se pueda usar un comando asignado a una variable:

```
fecha:`date`
```

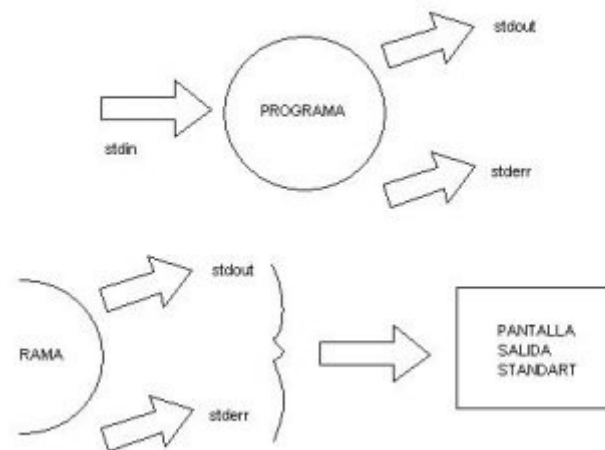
Esta orden ejecutaría el comando date, al llamar a la variable 'fecha'.

#### Redirección de la entrada/salida estándar.

La shell, toma información de la entrada estándar, (teclado) denominada **stdin**, cuando se ejecuta la orden dependiendo de como se ejecute, puede dar una salida valida o errónea, en caso de la salida valida la información por defecto la shell como salida estándar usa la pantalla, con lo cual nos muestra la información en la shell, en caso de que la ejecución del comando sea errónea, nos muestra la salida de error por defecto también por la pantalla y nos la muestra por la shell, aunque parezca en principio que tanto la salida estándar como la salida estándar de errores es lo mismo, no lo son, solo tienen por defecto asignado que enseñen la información por la pantalla .

Estas salidas, se pueden redireccionar, así mismo también la entrada estándar, o bien para que un comando tome los argumentos o sea la entrada de datos de un fichero o que la salida de un comando sea el argumento de otro, o que esa salida salga por la impresora, se imprima en un fichero etc .

Lo mejor es que lo veamos gráficamente:



### Formas de redireccionar la entrada estándar 'stdin'.

el signo '<' cambia la entrada estándar.

ejemplos:

mail rh3nt0n < carta.txt

El anterior comando tomaría como argumento para el comando mail, el contenido del fichero carta.txt.

comando < fichero

El anterior comando tomaría como argumento para el comando ejecutado, el contenido de fichero.

### Formas de redireccionar la salida standart 'stdout' .

El signo '>' redirecciona la salida estándar, creando el fichero si no existe y sobre escribiendo el archivo si existe.

**comando > fichero**

**comando > contenido**

**comando > argumentos**

**comando > /dev/dispositivo**

comando > stdout

El comando anterior, redireccionaria la salida del comando a el fichero [stdout](#)

La salida estándar de errores, también puede redireccionarse, ya que si el comando ejecutado, aun teniendo la salida estándar redireccionada a el sitio que sea que no sea la salida estándar, si ese comando falla, la salida de errores la muestra por pantalla.

Para que esto no ocurra si no es nuestro deseo, o bien capturar en un fichero las posibles salidas de error, se usa '2>' lo cual hace lo mismo que con la anterior opción hacia con la salida estándar.

comando 2> stderr

Este comando mostraría en caso de que la ejecución fuera correcta, por la pantalla, salida estándar de errores. y en caso de que hubiera algún error, la salida de error la mostraría en el fichero stderr.

En conjunción ,seria algo como esto:

comando > stdout 2> stderr

Esta sintaxis, redireccionaria la salida del comando en caso de que fuera correcto, a el archivo stdout, en caso de que fallara enviaría la salida de error a el fichero stderr.

Cuando no queremos que el archivo, a sabiendas de que existe, se sobrescriba, se utiliza el mayor que doble '>>' de esta forma:

```
echo " hola mundo " >> fichero.txt
```

El anterior comando añadiría el texto hola mundo a el fichero.txt sin sobrescribirlo. esto ocurre de igual manera con la salida de errores, pero de este modo:

```
eco "hola mundo" 2>> errores.txt
```

Como se podrá apreciar, el comando echo esta mal escrito, pero la salida de errores se redireccionaria a el fichero **errores.txt**, en lugar de mostrarla por la pantalla.

```
rh3nt0n@rh3nt0n-laptop:~$ eco "hola mundo" 2>> errores.txt
rh3nt0n@rh3nt0n-laptop:~$ cat errores.txt
bash: eco: orden no encontrada
rh3nt0n@rh3nt0n-laptop:~$
```

### Redirección a dispositivos 'especiales'

Se pueden redireccionar salidas muy extensas, o salidas de errores a el dispositivo **'/dev/null'**, para evitar colapsar de información la salida estándar, o simplemente por que no nos interese ver esa salida. Este dispositivo es denominado 'agujero negro' ya que lo que es redireccionado allí, no se puede recuperar.

#### Ejemplo de uso:

Aquí cabe resaltar el uso del comando **'time'**, el cual tiene la peculiaridad de mostrar su salida estándar, por la salida de errores. En el siguiente ejemplo ,este comando se encarga de cronometrar el tiempo que tarda en ejecutarse el comando junto al que se ejecuta, mostrando la salida por pantalla y enviando la salida del comando a ejecutar a el dispositivo **/dev/null**.

```
rh3nt0n@rh3nt0n-laptop:~$ time ls -R /etc > /dev/null
ls: /etc/cups/ssl: Permiso denegado
ls: /etc/ssl/private: Permiso denegado
real 0m0.382s
user 0m0.024s
sys 0m0.012s
rh3nt0n@rh3nt0n-laptop:~$
```

#### Los filtros de búsqueda.

Los filtros de búsqueda son una utilidad muy interesante a la hora de buscar palabras comandos etc de forma ordenada y desechando lo que no nos interesa . vamos a ver algunos comandos de búsqueda interesantes:

Comando **"sort"** sin opciones, ordena líneas de texto alfabéticamente aunque sean números.

#### opciones:

- n ordena números " 1 2 3 4 "
- t: define delimitador de campos
- +(n) n de número, para elegir que columna queremos ver primero.
- F no considera mayúsculas o minúsculas.
- r invierte la salida normal.

Comando **"grep"**, busca líneas de texto en un archivo o salida de comando especificada. sintaxis: **grep patrón fichero**

ejemplo de otro uso:

#### **dmesg argumento | grep**

opciones:

- v busca todas las líneas que no contienen el patrón
- i no considera mayúsculas de minúsculas .

-n numera las líneas que contienen el patrón especificado.

Comando "**wc**" 'word count' cuenta palabras la salida normal sin parámetros saca tres columnas:

sintaxis: **wc [opciones] fichero**

1º cantidad de líneas del fichero .

2º cantidad de palabras.

3º cantidad de caracteres.

opciones:

-l número de líneas únicamente

-w número de palabras únicamente

-c número de caracteres únicamente

Ejemplo de redirección con este comando:

```
ls /bin/!* | wc-l
```

### Iniciación a los procesos.

Los procesos no son mas que los programas que se están ejecutando, cada procesador puede ejecutar solo un proceso cada vez que actúa. existen multitud de estados de los procesos, pero a groso modo podríamos destacar estos cuatro:

Los procesos pueden estar en varios estados:

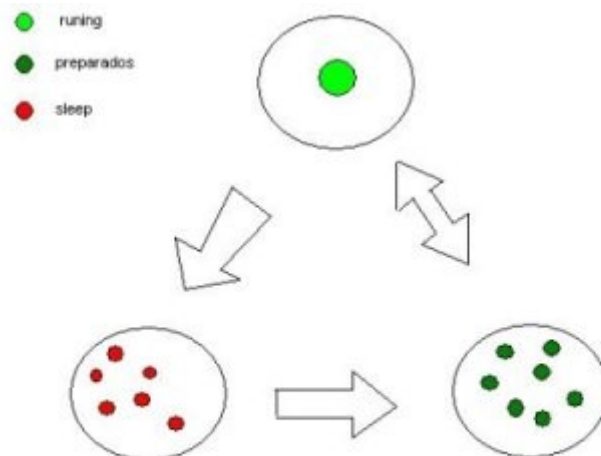
**Estado runing** - solo puede haber un proceso runing o funcionando.

**Estado preparado** - son procesos que están preparados para ser ejecutados por el procesador.

**Estado dormido** - son procesos que han acabado, pero pueden ser llamados otra vez a funcionar.

**Estado zombie** - son procesos como su propio nombre indica, que al no acabar bien, siguen en memoria sin borrarse.

Esto se aprecia mejor con el siguiente grafico:



### Comandos de información de procesos.

Comando "**ps**" este comando sirve para ver los procesos que se están ejecutando. Esto es muy útil para dar preferencias a algún proceso determinado, matar procesos, en definitiva conocer los PID de los procesos, para poder mandarles señales entre otras cosas .

La forma mas habitual de usar este comando es con las opciones **a** y **x**.

**a** para ver todos los procesos y **x** que muestre cuales se están ejecutando.

sintaxis: **ps ax**

ejemplo:

```
rh3nt0n@rh3nt0n-laptop:~$ ps ax
```

```
PID TTY STAT TIME COMMAND
```

```
1 ? S 0:01 init [2]
2 ? SN 0:00 [ksoftirqd/0]
3 ? S 0:00 [watchdog/0]
4 ? S< 0:00 [events/0]
5 ? S< 0:00 [khelper]
6 ? S< 0:00 [kthread]
8 ? S< 0:00 [kblockd/0]
9 ? S< 0:00 [kacpid]
10 ? S< 0:00 [kacpid-work-0]
162 ? S 0:00 [pdflush]
163 ? S 0:00 [pdflush]
165 ? S< 0:00 [aio/0]
164 ? S 0:00 [kswapd0]
766 ? .....
```

La primera columna de la salida del comando anterior muestra el PID del proceso ,todos los procesos tienen un PID del proceso, no es mas que un número identificador, para poder mandarle señal a cada proceso ejecutandose desde la consola.

La segunda columna indica la consola virtual donde se esta ejecutando el proceso, si no tiene aparece un signo de interrogacion '?'.  
La tercera columna indica en el estado en el que se encuentra el proceso:

**s ---- dormido**

**r ---- runing o corriendo**

**z ---- zombie**

**t ----- detenido**

comando útil:

```
ps ax | grep "proceso"
```

Busca y filtra el proceso buscado.

La cuarta columna, indica el tiempo que lleva ejecutándose el proceso.

Los procesos pueden recibir señales a través de la terminal, para interactuar con ellos .

Pueden trabajar como he explicado anteriormente en el front-ground, de forma visible en la terminal, y en el back-ground, de forma transparente, para nosotros desde la terminal.

Cuando un proceso recibe una señal desde la terminal, puede reaccionar de tres maneras diferentes:

1º puede ignorar la señal .

2º puede interpretar la señal .

3º puede aceptar que se encargue de el, el kernel .

### **Señales a los procesos.**

Comando "**kill**" en principio mata el proceso, con opciones puede hacer reiniciar procesos dormidos.

sintaxis: **kill -señal PID**

Señales mas importantes:

**señal -1** para que se reinicie un proceso, si no se puede, el sistema lo mata .

**señal -2** interrupción

**señal -3** para que el proceso acabe, pero sin errores .

**señal -9** para el proceso, y el proceso no puede ignorar esta orden.

**señal -15** es lo mismo que ejecutar kill sin opciones es como decirle al proceso 'cuando puedas cierra', esta señal puede ser ignorada por el proceso .

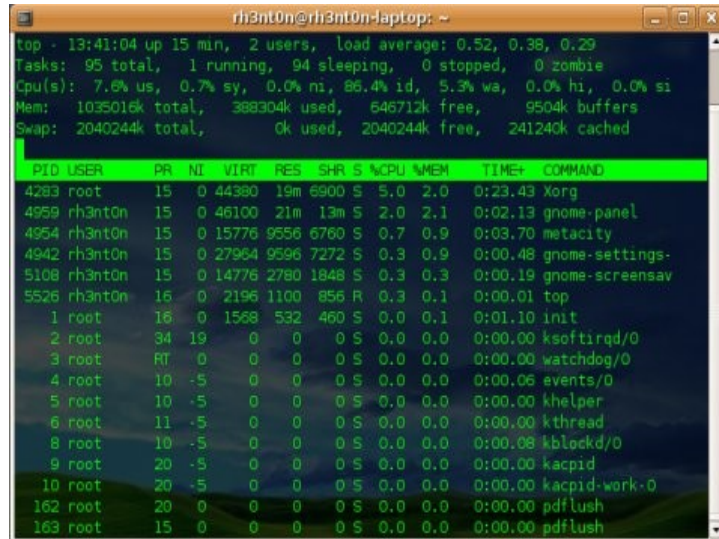


Comando **'nohup'** Ejecuta la orden ignorando las señales de colgar, sirve para que las ordenes sigan funcionando, aunque salgamos de la terminal.

Sintaxis: **nohup orden [argumento]**

Cabe resaltar que cuando la shell deja un comando funcionando y salimos de ella lo hereda init.

Para ver los procesos y los niveles de ejecución existe el comando **'top'**, este comando muestra de forma dinámica los procesos que se ejecutan en el sistema:



```
top - 13:41:04 up 15 min, 2 users, load average: 0.52, 0.38, 0.29
Tasks: 95 total, 1 running, 94 sleeping, 0 stopped, 0 zombie
Cpu(s): 7.6% us, 0.7% sy, 0.0% ni, 86.4% id, 5.3% wa, 0.0% hi, 0.0% si
Mem: 1035016k total, 388304k used, 646712k free, 9504k buffers
Swap: 2040244k total, 0k used, 2040244k free, 241240k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4283	root	15	0	44380	19m	6900	S	5.0	2.0	0:23.43	Xorg
4959	rh3nt0n	15	0	46100	21m	13m	S	2.0	2.1	0:02.13	gnome-panel
4954	rh3nt0n	15	0	15776	9556	6760	S	0.7	0.9	0:03.70	metacity
4942	rh3nt0n	15	0	27964	9596	7272	S	0.3	0.9	0:00.48	gnome-settings
5108	rh3nt0n	15	0	14776	2780	1848	S	0.3	0.3	0:00.19	gnome-screensav
5526	rh3nt0n	15	0	2196	1100	856	R	0.3	0.1	0:00.01	top
1	root	16	0	1568	532	460	S	0.0	0.1	0:01.10	init
2	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
4	root	10	-5	0	0	0	S	0.0	0.0	0:00.06	events/0
5	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khelper
6	root	11	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
8	root	10	-5	0	0	0	S	0.0	0.0	0:00.08	kblockd/0
9	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
10	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid-work-0
162	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
163	root	15	0	0	0	0	S	0.0	0.0	0:00.00	pdflush

Esta aplicación (parecida a el visor de procesos de Windows) nos es muy útil para interactuar con los procesos, a la hora de mandar señales, y dar prioridades a uno u otro proceso .

### Prioridades de los procesos.

Podemos asignar prioridades de ejecución a los procesos, si queremos que un programa tenga mas prioridad o menos, podemos bajarle o subirle el nivel de ejecución con el comando "nice".

Comando **"nice"**

Este comando asigna nivel de ejecución a los procesos, por defecto asigna el valor 10.

sintaxis: **nice [valor] orden [argumentos]**

Los valores de nivel de ejecución van de el +19 al -19 siendo el de menor valor '-19' el que mas prioridad tiene de ejecucion.

Los valores del -0 al -19 solo puede asignarlo el usuario root.

Comando **"renice"**.

Este comando repone el nivel de ejecución, a un procesos que este ejecutándose.

Sintaxis: **renice [valor] número PID**

Comando **"sleep"**.

Este comando duerme el proceso deseado 'x' segundos especificados.

Sintaxis: **sleep [tiempo] [opción] [orden]**

Opciones:

-s tiempo en segundos (por defecto)

-h tiempo en horas.

-d tiempo en días.

Comando **"jobs"**. Muestra los procesos que se ejecutan en el back-ground.

Para usar este comando y los siguientes, nos viene muy bien el comando anteriormente citado **"top"**, ya que debemos apoyarnos en el **IDE** de proceso **"PID"**, para poder interactuar con los

procesos en **back-ground**.

```
rh3nt0n@rh3nt0n-laptop:~$ yes > /dev/null &
[2] 6521
rh3nt0n@rh3nt0n-laptop:~$ jobs
[1]+  Stopped yes >/dev/null
[2]-  Running yes >/dev/null &
rh3nt0n@rh3nt0n-laptop:~$
```

Comando **"fg"**. Lleva un comando ejecutándose en back-ground, al front-ground.

Sintaxis: **fg [PID de proceso]**

```
rh3nt0n@rh3nt0n-laptop:~$ fg 2
yes >/dev/null
```

Comando **"bg"**. Relanza un proceso dormido, ejecutándolo en el back-ground.

Sintaxis: **bg [PID del proceso dormido]**

```
[2]+  Stopped yes >/dev/null
rh3nt0n@rh3nt0n-laptop:~$ jobs
[1]-  Stopped yes >/dev/null
[2]+  Stopped yes >/dev/null
rh3nt0n@rh3nt0n-laptop:~$ bg 2
[2]+  yes >/dev/null &
rh3nt0n@rh3nt0n-laptop:~$ jobs
[1]+  Stopped yes >/dev/null
[2]-  Running yes >/dev/null &
rh3nt0n@rh3nt0n-laptop:~$
```

En este punto, recuerdo que para matar un proceso conociendo su **PID** podemos usar lo siguiente:

Sintaxis: **kill "% PID"**

```
rh3nt0n@rh3nt0n-laptop:~$ kill %2
rh3nt0n@rh3nt0n-laptop:~$ jobs
[1]+  Stopped yes >/dev/null
[2]-  Terminado yes >/dev/null
rh3nt0n@rh3nt0n-laptop:~$
```

también recuerdo, como lanzar un proceso en el back-ground.

Sintaxis: **orden [argumentos] &**