

SEXTA SEMANA	1
Miembros del grupo de estudios	1
Materiales de estudio	1
Algunos criterios.....	2
TEMA 4 - Programación Básica del Shell	4
2.3.- Las variables \$*, \$@, \$#	4
2.4.- Expansión de variables usando llaves	4

SEXTA SEMANA

Miembros del grupo de estudios

María, Fabricio, Diego --> **Argentina**

Jorge, Asterix --> **Chile**

Alejandro --> **Brasil**

Gerardo --> **España**

Con la reciente incorporación de:

Miguel --> **Venezuela**

Reinaldo --> **España**



Materiales de estudio

(Si tienen alguno más para recomendar, lo agregan)

[Apuntes bash 0.pdf / Apuntes bash 1.pdf / bash.pdf / Comandos.pdf](#)

[\(GRUPO BASH\) PRIMERA SEMANA "COMODINES"](#)

[\(GRUPO BASH\) SEGUNDA SEMANA "REDIRECCIONES Y PIPES"](#)

*** [\(GRUPO BASH\) TERCERA SEMANA "CARACTERES ESPECIALES Y ENTRECORNILLADO"](#) *** **NUEVO!!!**

*** [\(GRUPO BASH\) CUARTA SEMANA "COMBINACIONES DE TECLAS"](#) *** **NUEVO!!!**

*** [\(GRUPO BASH\) QUINTA SEMANA "PARÁMETROS POSICIONALES. VARIABLES LOCALES Y GLOBALES"](#) *** **NUEVO!!!**

[Ejemplos de scripts - Ubuntu Forum](#)

[Un comando cada día - Ubuntu Forum](#)

[Bash scripting de supervivencia](#)

[Taller de programación shell](#)

[Shell Scripting](#)

[Google Docs](#)







[Redireccionamiento de Entrada-Salida](#)

[Redireccionamiento y tuberías](#)

[Combinaciones de teclas](#)

Algunos criterios

- Podemos reutilizar los iconos para señalar algunas cuestiones (copiar y pegar).

	Referencia al material de estudio
	Desafío
	Script
	Atención!
	Mi Reino por un caballo!!!
	Una duda...

	Premio "Rex Tux Scripting"
	Consejo
	Cuidado!!!
	Comando

- Delimitar las intervenciones con barras horizontales.
- Firmar con el nombre resaltado con color.

María, Fabricio, Diego, Jorge, Alejandro, Asterix, Gerardo, Miguel

- Fuente Courier New para los scripts y comandos. Aplicar sangrado de párrafo (ahora está en Formato-->Estilo de Párrafo).
- Si no visualizan correctamente las fuentes verdana y courier, instalen los paquetes:

```
sudo aptitude install msttcorefonts ttf-mscorefonts-installer
```

- **Poner onda con el formato** para facilitar la legibilidad del documento y el trabajo de publicación posterior.



TEMA 4 - Programación Básica del Shell

2.3.- Las variables \$*, \$@, \$#

2.4.- Expansión de variables usando llaves

Mi estudio de este tema:

Las variables predefinidas

1.- En el shell tenemos: **VARIABLES**

- **DE ENTORNO**
- **DEFINIDAS POR EL USUARIO**
- **PARÁMETROS POSICIONALES**
- **PREDEFINIDAS.**

Estas últimas están formadas por **un carácter especial precedido por el \$**

2.- **Variable \$*** : cadena de caracteres con todos los parámetros posicionales del shell activo excepto el nombre del programa shell.

3.- **Variable \$@** : idem a \$*. Se diferencian cuando se expanden con comillas dobles.

4.- **Variable "\$@"** (expandida con comillas dobles) Una sola cadena, cada componente separado del otro según el IFS.

Si IFS="s" entonces "\$*" dará **\$1s\$2s...**

Si IFS= no definida entonces "\$@" dará **\$1 \$2**

5.- **Variable "\$@"** (expandida con comillas dobles) Se expande en tantas cadenas de caracteres como parámetros posicionales haya. Equivale a **"\$1" "\$2"...**

6.- **Variable \$#** Da el número de parámetros posicionales excluido el nombre del programa shell. Sirve especialmente para verificar si el número de argumentos es el correcto.

7.- Por definición de variable, si al utilizar el valor de una variable el nombre de la variable es seguido de un carácter que sea otra letra, número o el símbolo _ tendremos que utilizar los símbolos **{ }** alrededor del nombre de la variable.

8.- **Variable \$?** Cuando una orden termina le devuelve al sistema un código de finalización que se almacena en la variable **\$?** Estos son:

0 → terminó correcto

1 → se produjo un error

9.- **UNIX**: Una línea leída de un fichero es idéntica a una línea leída desde el teclado: una serie de caracteres terminados por un carácter de retorno de carro.

10.- Es buena práctica acostumbrarse a usar **letras mayúsculas** para los nombres de las variables. Le dan más claridad a los scripts, más fácil lectura de uno.

11.- En mi /home tengo una carpeta bin especial para guardar los scripts, en lugar de repartirlos por todas partes: /home/jorval/bin Además la dirección de esa carpeta está en el PATH

Jorval

Algunos ejemplos:

Ejemplo 1.-

```
#!/bin/bash
# Ejemplo de script que recibe parámetros y los imprime

echo "El script $0 recibe $# argumentos: $*"
echo "El script $0 recibe $# argumentos: $@"
```

```
jorval@jorval-laptop:~$ source recibe hola adios
El script bash recibe 2 argumentos: hola adios
El script bash recibe 2 argumentos: hola adios
```

Ejemplo 2.-

```
#!/bin/bash
# Ejemplo de script que recibe parámetros y los imprime

echo "El script $0 recibe $# argumentos: $*"
echo "El script $0 recibe $# argumentos: $@"

function CuentaArgumentos {
    echo "Recibidos $# argumentos"
}

CuentaArgumentos "$*"
CuentaArgumentos "$@"
```

```
jorval@jorval-laptop:~$ source recibel hola adios
El script bash recibe 2 argumentos: hola adios
El script bash recibe 2 argumentos: hola adios
Recibidos 1 argumentos
Recibidos 2 argumentos
jorval@jorval-laptop:~$
```

Ejemplo 3.-

Mismo script anterior, pero con entrecomillado

```
jorval@jorval-laptop:~$ source recibel "El perro" "La casa"
```

```
El script bash recibe 2 argumentos: El perro La casa
El script bash recibe 2 argumentos: El perro La casa
Recibidos 1 argumentos
Recibidos 2 argumentos
jorval@jorval-laptop:~$
```

Jorval

Muy buena idea, **Jorge**, lo del punto 11.

¿Para incorporar el directorio a PATH usaste el procedimiento que nos dejó Asterix la semana pasada?

Alejandro

Alejandro: No, en mi sistema ese directorio viene creado por defecto y solo tenía un ejecutable de compiz-check. Después me dí cuenta que PATH lo tiene incorporado desde un principio y ahí estimé que ese directorio es precisamente para los scripts igual que /bin.

Jorval

Mirá qué interesante. Nunca había visto eso. Tú estás usando Ubuntu 8.04 LTS, no recuerdo que existiese ese directorio en home cuando usé esa versión ¿Está oculto?

Alejandro

No está oculto, siempre ha estado ahí a la vista.

Jorval

Respecto a la duda que surgió la semana pasada sobre el método de modificación de la variable PATH que propuso **Asterix**, y al directorio /home/bin de **Jorge**, resultó esto:

```
~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
~$ PATH=$PATH:/home/alejandro/bin
```

Grupo de estudios: Programación Bash Script

Sexta semana: del 12 al 18 de noviembre de 2009

```
~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games:/home/alejandro/bin
```

- Vean que listé el estado de mi variable PATH;
- Cada directorio de la variable está separado por ":";
- Entonces ahí queda claro el procedimiento y la sintaxis:
"PATH=\$PATH:/nuevo_directorio". Es decir la variable va a tomar el valor que ya tenía más "/nuevo directorio";
- El método es totalmente seguro y sólo agrega un nuevo directorio a la variable;
- Al listar nuevamente, se puede apreciar el resultado. Ahora puedo ir dejando mis scripts ahí, y ejecutarlos **como si fueran un comando más**.
- Supongo, aunque no puedo asegurar nada, que otro usuario que no sea yo, tendrá que usar sudo ¿será?

Ah, y me faltó decir que hay que darles permiso de ejecución a los scripts, para que funcione todo bien.

```
~$ chmod +x nombre_script
```

Alejandro

Hago lo mismo que Alejandro y me funciona... pero el problema es que cuando cierro la consola y abro una nueva, la variable \$PATH volvió a estar como antes.

¿A alguien le pasa lo mismo?

Fabricio

Que raro lo que les pasa, les copio mi PATH y creo que siempre ha estado así

```
jorval@jorval-laptop:~$ echo $PATH
/home/jorval/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin:/usr/games
jorval@jorval-laptop:~$
```

Jorval

Encontré la solución, para modificar la variable \$PATH de manera permanente, hay que editar el archivo de configuración de la shell, en nuestro caso tendríamos que editar el archivo ~/.bashrc y al final del archivo escribir

```
export PATH=$PATH:/home/user/scripts
```

(recuerden cambiar **user** por su usuario y **scripts** por el nombre que quieren que se llame la carpeta)

A continuación les dejo un script que hace precisamente lo que dije antes

```
echo 'export PATH=$PATH:/home/user/scripts' >> /home/user/.bashrc
```

Al tema de los permisos lo solucioné de la siguiente manera, **en el archivo ~/.bashrc agregué la siguiente línea:**

```
alias perm='chmod -R +x ~/scripts'
```

y entonces cuando quiero ejecutar un script que acabo de crear hago lo siguiente: (suponiendo que **asd** es un script que se encuentra en ~/scripts)

```
fb91@fb91-desktop:~$ perm
fb91@fb91-desktop:~$ asd
hola
```

Fabricio

Pero claro!!! Sí es lo que todos tendríamos que haber sabido a esta altura!!! La definición de la variable vale para esa sesión, si quiero que sea válida fuera de esa sesión, tengo que exportarla. **Cita textual** del libro guía:



Las variables de entorno que definimos dentro de una sesión de Bash no están disponibles para los subprocesos que lanza Bash (sólo las puede usar Bash y nuestros comandos interactivos y scripts) a no ser que las exportemos con el comando export.

Muy bueno el descubrimiento de adónde hay que configurar eso, **Fabricio!** Porque sin eso era lo mismo que nada.

Lo de los permisos, esperá que lo pienso un poco...

Alejandro

A raíz de lo que posteo **Fabricio**, estaba leyendo en la **página 36 del libro** guía sobre los **alias**. Y ahí dice que si quiero saber qué alias tengo definidos, tengo que usar el comando alias sin parámetros. Hice eso, y me tiró un alias que, claro, yo nunca había definido:

```
alias ls='ls --color=auto'
```

Busqué en el man de **ls**, para saber de qué se trataba, y encontré esto:

```
--color[=cuándo]
    Especifica si emplear color para distinguir
tipos de ficheros.
    Los colores se especifican mediante la
variable de entorno
    LS_COLORS. Para información acerca de cómo
definir esta vari-
    able, consulte dircolors(1). cuándo puede
omitirse, o ser uno
    de:

    none No emplear color en absoluto. Esto es lo
predeterminado.

    auto Emplear color solamente si la salida
estándar es una ter-
    minal.

    always Emplear color siempre. Especificar --color
y no cuándo es
    equivalente a --color=always.
```

Eso quiere decir que cuando ejecutamos el comando "ls", en realidad estamos ejecutando un alias.

Off Topic, como dirían en Ubuntu Forum, pero bueno, lo quería comentar. Sobre todo, porque el tema "alias" nos lo salteamos.

Y sigo pensando sobre los permisos...

Alejandro

Cuando escribo **perm** en definitiva estoy escribiendo `chmod -R +x ~/scripts` (ya que **perm** es un alias que definí anteriormente) y lo que hace el `chmod` es dar permisos de manera recursiva (por eso el `-R`) a la carpeta `~/scripts`

Fabricio

Perdonen un poco la tardanza. He aquí algunas dudas



a) En los ejemplos dados en el manual bash, los script los ejecutan sin anteponer nada, solo con el nombre; pero a mi no me dan los mismos resultados. Obtengo los mismos resultados si antepongo `./` o bien anteponiendo `source`

Para el siguiente ejemplo de la página 52

```
function CuentaArgumentos {
    echo "Recibidos $# argumentos"
}
```

```
CuentaArgumentos $*  
CuentaArgumentos $@
```

Al ejecutarlo en la terminal (ejemplo 1)

```
$ recibe "el perro" "la casa"  
La función bash  
Recibe los argumentos el perro la casa
```

Al ejecutarlo en la terminal (ejemplo 2)

```
./recibe "el perro" "la casa"  
Recibidos 4 argumentos  
Recibidos 4 argumentos
```

¿Existe un error en la sintaxis del manual?

b) Otra duda, al agregar una vía a mi PATH (ej. /home/asterix/script), me resulta. Pero al parecer ¿hay que primero crear la carpeta?. Ya que al listar el contenido no me aparece dicha carpeta.

c) En la página 54 "Expansión de variables" el ejemplo:

```
$ nombre=Fernando  
$ apellido=Lopez  
$ echo "${nombre}_${apellido}"
```

Me dá el mismo resultado que si fuera

```
$ nombre=Fernando  
$ apellido=Lopez  
$ echo "$nombre $apellido"
```

¿Existirá alguna razón especial por el que no se pueda usar la segunda forma?

Asterix

Asterix: Si te fijas en mis ejemplos, en los tres puse "source" para ejecutar los scripts. Son los misterios de Linux, pero lo bueno es que si no funciona de una manera, pruebas de otra y listo. Siempre que no estés actuando como root, cosa que evito permanentemente. Eso es lo que hago yo por lo menos. Saludos.

Jorval

Planteo cuatro situaciones que veo en mi terminal:

```
~$ bash nombre_script  
~$ source nombre_script
```

Ejecutan el script aunque este no tenga permisos de ejecución.

```
~$ ./nombre_script
```

Ejecuta el script **si tiene permisos de ejecución**.

Si quiero ejecutarlos directamente, con el nombre del script como si fuera un comando, sólo es posible cuando están en un directorio que está en la variable **PATH**. Pero tengo que darles permisos de ejecución como explicó Fabricio.

Alguien tiene fundamentos teóricos como para explicar cuál es la diferencia entre los tres primeros procedimientos? Al último ya lo estuvimos comentando en esta semana.

Asterix: Sí, creo que primero hay que crear la carpeta. Aunque, seguramente si la creas después, es lo mismo. Pero hay que crearla, el procedimiento de incluirla en la variable PATH, no la crea, sólo la añade a la lista de directorios cuyos ficheros serán considerados como comandos ejecutables.

Ah, y guarda cuando guardan un script con extensión ".sh", por dar un ejemplo. Para ejecutarlo, hay que colocar la extensión. No puedo ejecutar "script.sh" con la orden "script" y listo.

Alejandro

Archivos de configuración de Bash

Lo que nos sucede con la ejecución de los scripts que creamos tiene algo que ver con esto de los archivos de configuración de Bash. Les dejo esta página que complementa y aclara en parte el Manual [Hacer clic aquí](#).

Verifiqué mi sistema y de los cuatro archivos principales que ahí aparecen a mí me falta ~/.bash.profile y por eso, quizás, tengo que ponerles a mis script el comando source.

Jorval

Asterix:

Estaba probando el script de la **página 52**. Probé ejecutarlo directamente con el nombre (ya que lo tengo en un directorio de PATH), probé ejecutándolo con "source", "bash" y "./", y me da siempre el mismo resultado. Cuenta la cantidad de parámetros. Si están entrecomilladas las variables, el resultado varía entre lo que devuelve una línea de la función y la otra, pero no cambia nada entre una ejecución y la otra.

Por otro lado, viendo lo que te devuelve la terminal en el primer ejemplo, pareciera ser que fuera otro script el que está funcionando. Porque, "\$*" y "\$@" toman los argumentos y se los pasan a la función; pero en la función lo que actúa es "\$#" que lo que hace es contar, nunca debería devolverte las palabras que se usaron como argumentos.

Estoy equivocado?

No será que tienes dos script con el mismo nombre en distintos directorios, y por eso varía cuál de ellos es ejecutado según la forma de ejecutarlo?

Ese "otro" script o función debe tener una línea que dice algo así:

```
echo "La función bash Recibe los argumentos" $* (o $@)
```

Por las dudas, no definiste previamente una función de nombre "Recibe"? Debería haber sido exportada para que esté influyendo en este script. Coloca "export" en la terminal a ver si no sale algo.

Pero no puede ser, porque no está precedida por un signo \$. No sé che, qué extraño...

Alejandro

Y por último:

```
~$ echo "$nombre $apellido"  
fernando lopez  
  
~$ echo "$nombre_ $apellido"  
lopez  
  
~$ echo "${nombre}_ $apellido"  
fernando_lopez
```

No dan el mismo resultado:

1. Toma el espacio en blanco.
2. No encuentra la variable "\$nombre_", porque la variable que fue definida es: "\$nombre" (sin guión bajo)
3. Considera la cadena "nombre" como el nombre de la variable y excluye el guión bajo del nombre de la variable, pero lo imprime en la salida estándar.

PD: Interesante el material, **Jorge**.

En mi home tampoco figura el fichero **.bash_profile**.

El material que propuso **Jorge**, dice:

```
"Cuando se crea una cuenta de usuario en un sistema Ubuntu, se crean con ella los ficheros .bashrc y .bash_profile. Esos dos ficheros se crean mediante la copia desde unos ficheros plantilla con los mismos nombres, y que están ubicados en el directorio /etc/skel."
```

Mi directorio **/etc/skel** tiene estos ficheros ocultos:

```
~$ ls -a /etc/skel
.  ..  .bash_logout  .bashrc  .profile
```

El contenido de **.profile** es:

```
~$ cat /etc/skel/.profile
# ~/.profile: executed by the command interpreter for login
shells.
# This file is not read by bash(1), if ~/.bash_profile or
~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the
umask
# for ssh logins, install and configure the libpam-umask
package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

Ahí, en la línea que remarqué creo que explica por qué no tenemos ese fichero: *"Este fichero no será leído por Bash si ~/.bash_profile or ~/.bash_login existen"*.

No, en realidad no, porque tampoco tengo esos ficheros... está **.bash_logout**, pero no **.bash_login**...

Entonces, debe ser cuestión de copiar y renombrar el fichero **.profile**, si fuera necesario contar con ese fichero de configuración de bash.

¿Y vieron estas líneas? Tienen que ver con el **~/bin de Jorge**:

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

En castellano:

Si existe el directorio `"/home/usuario/bin"`, entonces agregar al comienzo de la variable PATH lo siguiente: `"/home/usuario/bin:"`

Eso quiere decir que, si este fichero está en funcionamiento, es decir, está en nuestra home; solamente hay que crear en nuestra home un directorio `~/bin` y este se agrega automáticamente a PATH.

Alejandro

Alejandro he vuelto a realizar el ejercicio varias veces hoy, y ahora si me imprime como a tí

```
$ nombre=Fernando
$ apellido=Lopez
$ echo "${nombre}_${apellido}"
$ Fernando_Lopez
```

Me dá el mismo resultado que si fuera

```
$ nombre=Fernando
$ apellido=Lopez
$ echo "$nombre $apellido"
$ Fernando Lopez
```

No sé que habrá pasado ayer, seguramente habré hecho algo.

Con respecto al ejercicio de la pág 52, pido disculpas. Había un error en la sintáxis que he corregido y ahora si funciona tal cual el libro. Creo que ayer no fue mi día.

En el contenido de mi home, veo un archivo oculto `._recibe.swp`. ¿Tendrá algo que ver con el ejemplo de recibe?. Además tengo `.bash_logout`, `.bashrc` entre otros.

Asterix

Archivos de configuración de Bash **(continuación)**

Respecto a los ALIAS, les dejo esta página que complementa y aclara en parte el Manual [Hacer clic AQUI](#)

Aquí también, al igual que con los script, parece conveniente tener en el `/home` un archivo especial para los alias, tal como explica el artículo que indico arriba. (Estoy pensando que no fue bueno saltarnos la Parte 3 del Manual)

Jorval

Me pareció buena idea incluirlo en el documento.

Definiendo y usando alias

"Otra característica muy interesante de la shell bash es la posibilidad de definir alias en algunos comandos. Un alias es una cadena que la shell reconoce y que nos facilita el uso de ciertos comandos. Algunos de los alias más comunes usados en Linux son para los comandos cp y rm, forzando a que se ejecuten automáticamente en su modo interactivo, lo que nos muestra una confirmación antes de borrar o copiar un archivo. Los alias se pueden definir desde la consola como se muestra a continuación (si quieres que sea permanente debes modificar el archivo `~/.bashrc`):

```
$ alias cp="cp -i"
$ alias rm="rm -i"
```

Una vez asignados los alias la consola los reemplaza automáticamente cuando los usas, como si los tecleases enteros (escribes "cp" y la consola hace como si hubieras escrito "cp -i"). Para simplificar el uso de los alias (y minimizar el número de cambios de `~/.bashrc`), el archivo `~/.bashrc` tiene por defecto las siguientes entradas comentadas:

```
# if [ -f ~/.bash_aliases ]; then
#   . ~/.bash_aliases
# fi
```

Si les quitas el símbolo numeral (o almohadilla), las entradas dejarán de estar comentadas, y el archivo hará que todos los alias se guarden en `~/.bash_aliases`, que se creará en el directorio `/home`."

Alejandro

Me puse a mirar el dichoso fichero `~/.bashrc` y aparecen una serie de líneas (entre muchas otras) que sería interesante analizar cómo funcionan y para qué. No sé qué opinan ustedes...

```
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval
"$$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
```

```
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

- Por empezar, en mi "~/.bashrc" las líneas de las que hablaba el artículo que reproducimos anteriormente no están comentadas. Pero en mi home no se creó "~/.bash_aliases", como dice en el artículo **¿Tendré que crearlo yo?** Y otra cosa que me desconcierta: **¿qué significa el punto antes del comodín tilde "~" en el if?**
- Aparecen una serie de alias sin comentar que, efectivamente, si ejecuto el comando alias sin argumentos en la terminal, aparecen listados. Y otra serie de alias que están comentados pero que se podrían des-comentar.

Alejandro

Alejandro: Yo nunca seré Administrador y creo que nunca crearé un script para hacer trabajos con mis archivos. Tampoco estoy interesado en moverme por la terminal sin emplear el ratón - dichoso ratón - mi único y principal interés es poder leer los scripts que se encuentran repartidos por todas partes, pero también creo que aún no estamos listos para ello, pues aún no pasamos esos operadores en cadena y especialmente las sentencias de control de flujo. Por lo que sugiero no adelantarnos, pues empezaremos a rasguñar por todas partes y en forma desordenada.

Mi archivo ~/.bashrc es igual al tuyo. Creo que al igual como sucedió con /home/usuario/bin deberíamos crear en nuestro /home/usuario un directorio .bash_aliases

Respecto al punto antes del comodín tilde ~ en if, en alguna parte lei que ese punto es lo mismo que el comando source. Cuando lo encuentre lo pongo.

Jorval

A mi parecer, ya hemos avanzado lo suficiente como para entender algunas líneas de los scripts repartidos en nuestro pc. Eso en lo personal me gratifica bastante. Ni se imaginan la alegría al ver que me funcionó el script comentado en la quinta semana. Y justamente eso es lo que más hago, revisar cuanto script tenga posibilidad de ver y tratar de entenderlo, ahora los entiendo bastante más de cuando comencé. Bueno cada uno tiene su método.

Con respecto al contenido de ~/.bashrc

alias ls='ls --color=auto' Al ejecutarlo en consola en realidad ejecuta ls --color=auto, y al estar descomentado lo que hace es mostrar el contenido de una carpeta (listar), agrupando los archivos y directorios con un color definido

#alias dir='dir --color=auto' Hace lo mismo que el anterior con algunas salvedades; antepone un \ antes de un espacio si el nombre es compuesto, y al estar comentado no agrupa los archivos y directorios por color. Para hacerlo hay que descomentarlo.

#alias vdir='vdir --color=auto' Hace lo mismo que ls -l (lista los directorios con los atributos de cada carpeta y archivo); al estar comentado no agrupa los archivos y directorios por color. Para hacerlo hay que descomentarlo.

alias grep='grep --color=auto' El comando grep permite buscar un patrón de coincidencias en archivos de texto, mostrando la línea entera en dónde se encontró esta coincidencia. Es bastante configurable y también permite expresiones regulares para indicar este patrón.

alias fgrep='fgrep --color=auto' El comando fgrep es similar a grep, pero con tres diferencias principales: se puede utilizar para buscar varios objetivos al mismo tiempo, no permite utilizar expresiones regulares para buscar patrones y es más rápida que grep. Cuando se busca en un archivo grande o en varios pequeños, la diferencia de velocidad puede ser significativa.

Con fgrep se pueden buscar las líneas que contengan uno cualquiera de varios objetivos alternativos. Por ejemplo, la siguiente orden busca las entradas en el archivo recetas.txt que contengan las palabras "pollo" o "pavo".

```
$ fgrep "chevrolet  
> mazda" camionetas.txt
```

alias egrep='egrep --color=auto' egrep es el componente más potente de la familia de órdenes grep. Al igual que fgrep, se puede utilizar para buscar múltiples objetivos. Lo mismo que grep, permite usar expresiones regulares para especificar los objetivos, pero proporciona un conjunto más completo y potente de expresiones regulares que grep. egrep acepta todas las expresiones regulares básicas reconocidas por grep.

```
egrep "asterix|alejandro|jorval" agenda.txt
```
