Modulo GNU/Linux – Electiva Libre II

Universidad de Córdoba

Departamento de Ingeniería de Sistemas y Telecomunicaciones

Índice de contenido

1. Instalación de Paquetes en GNU/Linux Ubuntu y GNU/Linux Debian

Como administradores de un sistema GNU/Linux debemos actualizar e instalar los paquetes en nuestro sistema; para este propósito las distribuciones Debian y Ubuntu ofrecen herramientas que facilitan la instalación de software.

En Debian y Ubuntu, así como en otras distribuciones GNU/Linux el software viene en paquetes, Estos tienen un nombre y una versión. En Debian una nueva versión de un mismo paquete remplaza a otra ya existente, además un paquete puede tener referencia a otros paquetes, es decir que depende de estos.

Un paquete puede estar en uno de los siguientes estados: instalado, configurado y desinstalado. Un paquete tiene solo la configuración cuando este no esta instalado pero sus archivos de configuración si, es decir si tenemos instalado un paquete y lo removemos este pasa al estado de sólo la configuración, debido a que si volvemos a instalar el paquete recuperaremos la configuración.

Cuando se desea eliminar del sistema todo rastro de un paquete, es decir el paquete y sus dependencias. Entonces hay que purgarlo, borrando todo lo instalado por un paquete, incluyendo los ficheros de configuración.

Hay que tener en cuenta que no todas las distribuciones GNU/Linux tienen la misma extensión para sus paquetes, en el caso especifico de Debian y Ubuntu la extensión de los paquetes es **.deb**, si por ejemplo se trabaja con Red Hat la extensión de los paquetes seria **.rpm** y el comando a utilizar seria rpm en vez de dpkg.

Todos los paquetes dejan sus archivos de configuración en el directorio /etc, además en Debian todo archivo de configuración tiene su página de manual, es decir que si escribo en la consola la orden **man dpkg** existirá un manual para este paquete.

Ejemplo de como seria la sintaxis del nombre de un paquete:

NombrePaqute-NumeroVersion.deb

1.1 Comando dpkg

La herramienta principal en Debian es el comando **dpkg**, utilizado también en la distribución Ubuntu. Es una herramienta un poco compleja de utilizar para usuarios novatos teniendo en cuenta que Debian y Ubuntu poseen herramientas gráficas fáciles de utilizar para la instalación de paquetes, tales como el **gestor de paquetes Synaptic** y **Añadir y quitar aplicaciones** las cuales serán analizadas posteriormente.

Entre los parámetros que podemos utilizar con el comando dpkg encontramos:

- -i ó --install: Instalar paquetes.
- -r ó --remove: Desinstalar paquetes.
- -P --purge: Borrar todo lo instalado por un paquete, incluyendo los ficheros de configuración.
- -l ó --list: Listar los paquetes instalados y sus versiones.
- -s ó --status: Ver la información de estado de un paquete.

Si desea investigar mas a fondo sobre los parámetros y formas de uso del comando dpkg puede leer su página de manual, ya que el comando puede manejar directamente los ficheros .deb que tengamos.

Ejemplo de como instalar paquetes utilizando en comando dpkg:

El comando dpkg permite la instalación de paquetes en equipos que no tengan conexión a Internet, pero para hacer esto primero tenemos que descargar desde un equipo que tenga Internet los paquetes y las dependencias, en Ubuntu podemos hacerlo con los siguientes comandos:

```
$ sudo aptitude clean
$ sudo aptitude install -d nombre_paquete
```

Con el primer comando se borran esos paquetes similares existentes en el ordenador. El segundo comando descarga el paquete deseado y sus dependencias pero no lo instala.

Ahora vamos a **/var/cache/apt/archives**, luego copiamos los archivos a un medio magnético y los llevamos al equipo sin conexión, nos ubicamos en el directorio en donde se encuentre el paquete y los instalamos con el comando:

```
$ sudo dpkg -i nombre_paquete.deb
```

Hay que tener en cuenta que si el paquete tiene dependencias hay que instalarlas primero. Para no tener problema con las dependencias podemos ir al equipo con Internet y utilizando el gestor de paquetes Synaptic, buscamos el paquete que queremos, hacemos clic derecho sobre el, entramos en **Propiedades** y seleccionamos la pestaña de **Dependencias**. Ahí vemos los paquetes que necesitamos para instalar debidamente el paquete en el equipo sin internet.

Lo descrito anteriormente es para instalar paquetes en equipos que no tengan conexión a internet, pero la realidad es que es mucho mas fácil instalar o actualizar los paquetes en nuestra maquina si contamos con una buena conexión a internet.

En Debian y Ubuntu se pueden instalar paquetes ya preparados para ser instalados en el sistema los cuales tiene la extensión **.deb**, usando el comando dpkg también habrá que instalar manualmente las posibles dependencias del paquete.

A continuación instalaremos el paquete **xmms** esta aplicación es un reproductor de música mp3:

Para instalar un paquete con dpkg en Ubuntu y Debian es muy similar, podemos buscar la página oficial en donde están todos los paquetes con sus respectivas dependencias.

Para el caso de Ubuntu 8.04 podemos buscar en este sitio (http://packages.ubuntu.com/), los paquetes con sus dependencias o para el caso de Debian buscar en internet los paquetes necesarios (http://www.debian.org/distrib/packages) y sus dependencias para descargarlas en el disco local.

Primero descargamos un paquete .deb que sirva para ser instalado en nuestra maquina, desde internet descargue el paquete **xmms_1.2.10+20070601-1_amd64.deb**. Luego me desplazo a al directorio en donde descargue el paquete y ejecuto la siguiente orden:

```
pedro@pedrogs:~$ cd instaladores
pedro@pedrogs:~/instaladores$ sudo dpkg -i xmms_1.2.10+20070601-1_amd64.deb
```

Puede que muestre problemas de dependencia y el paquete quede roto, por falta de algún paquete o por el uso de una dependencia similar pero con otra versión.

Si vamos al menú **Aplicaciones --- Sonido y vídeo** encontramos instalado el reproductor xmms.

Si quiere desinstalar el paquete y eliminar los ficheros de configuración o dependencias se utiliza el siguiente comando:

```
pedro@pedrogs:~/instaladores$ sudo dpkg -P xmms
```

1.2 apt-get

Aunque se puede instalar programas de forma gráfica como veremos mas adelante, con este comando podemos instalar cualquier programa desde la terminal, en los sistemas Debian y Ubuntu. apt-get es el conjunto de herramientas ofrecida por Debian para que los usuarios no tengan que instalar paquetes manualmente.

APT (Advanced Packaging Tool), es un sistema de gestión de paquetes creado por el proyecto Debian para simplificar la complejidad de la instalación y eliminación de programas (paquetes) en los sistemas GNU/Linux. APT fue diseñado originariamente para trabajar con paquetes .deb, en lo sistemas Debian y distribuciones derivadas, pero ha sido modificado para trabajar con paquetes .rpm, con la herramienta apt-rpm, y para que funcione en otros sistemas operativos.

Esta herramienta baja los archivos a partir de la lista de repositorios en la cual se encuentran todos los paquetes existentes y las relaciones de dichos paquetes con sus dependencias. La herramienta apt-get requiere de la configuración del fichero plano **/etc/apt/sources.list** de donde toma la información sobre la ubicación (URL's) de los paquetes y sus dependencias para ser instalados.

El fichero /etc/apt/sources.list contiene la lista de los repositorios disponibles, de los paquetes de software candidatos a ser instalados, removidos o actualizados. De esta manera la herramienta APT administra el acceso a dichos paquetes, utilizando el fichero sources.list, siendo este de gran importancia para la administración de los sistemas Debian.

A continuación se muestran los parámetros de uso del comando apt-get:

- apt-get update: Actualiza la lista de paquetes del sistema con las nuevas versiones.
- apt-get install <paquete>: Instala un paquete.
- apt-get remove <paquete>: Desinstala un paquete.
- apt-cache search <paquete>: Busca un paquete entre todos los existentes.
- apt-get dist-upgrade: Actualiza la distribución a las últimas versiones.

Estructura del fichero /etc/apt/sources.list en Ubuntu 8.04:

```
pedro@pedrogs:~$ sudo gedit /etc/apt/sources.list
```

```
# deb cdrom:[Ubuntu 8.04 _Hardy Heron_ - Release amd64 (20080423)]/ hardy main restricted
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.

deb http://co.archive.ubuntu.com/ubuntu/ hardy main restricted
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy main restricted

## Major bug fix updates produced after the final release of the
## distribution.
deb http://co.archive.ubuntu.com/ubuntu/ hardy-updates main restricted
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy-updates main restricted

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## universe WILL NOT receive any review or updates from the Ubuntu security
## team.
deb http://co.archive.ubuntu.com/ubuntu/ hardy universe
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy universe
deb http://co.archive.ubuntu.com/ubuntu/ hardy-updates universe
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy-updates universe

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## multiverse WILL NOT receive any review or updates from the Ubuntu
## security team.
deb http://co.archive.ubuntu.com/ubuntu/ hardy multiverse
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy multiverse
deb http://co.archive.ubuntu.com/ubuntu/ hardy-updates multiverse
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy-updates multiverse

## Uncomment the following two lines to add software from the 'backports'
## repository.
## N.B. software from this repository may not have been tested as
## extensively as that contained in the main release, although it includes
## newer versions of some applications which may provide useful features.
## Also, please note that software in backports WILL NOT receive any review
## or updates from the Ubuntu security team.
deb http://co.archive.ubuntu.com/ubuntu/ hardy-backports main restricted universe multiverse
deb-src http://co.archive.ubuntu.com/ubuntu/ hardy-backports main restricted universe multiverse

## Uncomment the following two lines to add software from Canonical's
## 'partner' repository. This software is not part of Ubuntu, but is
## offered by Canonical and the respective vendors as a service to Ubuntu
## users.
# deb http://archive.canonical.com/ubuntu hardy partner
# deb-src http://archive.canonical.com/ubuntu hardy partner

deb http://security.ubuntu.com/ubuntu hardy-security main restricted
deb-src http://security.ubuntu.com/ubuntu hardy-security main restricted
deb http://security.ubuntu.com/ubuntu hardy-security universe
deb-src http://security.ubuntu.com/ubuntu hardy-security universe
deb http://security.ubuntu.com/ubuntu hardy-security multiverse
deb-src http://security.ubuntu.com/ubuntu hardy-security multiverse
```

Estructura del fichero /etc/apt/sources.list en Debian etch:

```
salasistema:/home/administrador# gedit /etc/apt/sources.list
```

```
#
# deb cdrom:[Debian GNU/Linux 4.0 r3 _Etch_ - Official amd64 DVD Binary-1 20080217-12:13]/ etch
contrib main

deb cdrom:[Debian GNU/Linux 4.0 r3 _Etch_ - Official amd64 DVD Binary-1 20080217-12:13]/ etch contrib
main

# Line commented out by installer because it failed to verify:
#deb http://security.debian.org/ etch/updates main contrib
# Line commented out by installer because it failed to verify:
#deb-src http://security.debian.org/ etch/updates main contrib
```

Con los comandos anteriores se editan estos ficheros para ser configurados. Por lo general en la configuración se quitan los comentarios (#) de las URL's en donde se encuentran los repositorios para tener acceso a estos, o se agregan direcciones con nuevos repositorios. Por ejemplo:

Ubuntu:

```
# deb http://archive.canonical.com/ubuntu hardy partner
# deb-src http://archive.canonical.com/ubuntu hardy partner
```

```
deb http://archive.canonical.com/ubuntu hardy partner
deb-src http://archive.canonical.com/ubuntu hardy partner
```

Debian

```
# Line commented out by installer because it failed to verify:
#deb http://security.debian.org/ etch/updates main contrib
# Line commented out by installer because it failed to verify:
#deb-src http://security.debian.org/ etch/updates main contrib
```

```
# Line commented out by installer because it failed to verify:
deb http://security.debian.org/ etch/updates main contrib
# Line commented out by installer because it failed to verify:
deb-src http://security.debian.org/ etch/updates main contrib
```

Para la Instalación del editor eclipse, escribimos en la consola el siguiente comando:

```
# apt-get install eclipse
```

Luego accedemos a la aplicación a través del menú **Aplicaciones --- Programación** para abrir el editor eclipse. Si queremos desinstalar la aplicación ejecutamos en la consola el siguiente comando.

```
# apt-get remove eclipse
```

1.3 aptitude

Con el comando aptitude también puede instalar aplicaciones (paquetes), en la distribución Ubuntu. Esta herramienta es similar a la utilizada anteriormente (apt-get), la diferencia que existe con la anterior, es que aptitude recuerda las dependencias de algún paquete instalado, es decir si se desea borrar un paquete (programa), aptitude borrará el programa junto con todas sus dependencias, siempre que estas no sean usadas por otros programas.

Si instala con apt-get o con el gestor de paquetes Synaptic, la desinstalación de algún paquete solo eliminara dicho paquete y no sus dependencias. La sintaxis de uso de este comando es la siguiente:

- Instalación de paquetes: **$ sudo aptitude install <paquetes>**
- Desinstala paquetes: **$ sudo aptitude remove <paquetes>**
- Desinstala los paquetes incluyendo los archivos de configuración: **$ sudo aptitude remove --purge <paquetes>**
- Actualiza la lista de paquetes disponibles: **$ sudo aptitude update**
- Actualiza el sistema con las actualizaciones de paquetes disponibles: **$ sudo aptitude upgrade**
- Obtener una lista de opciones del comando: **$ sudo aptitude help**

Ejemplo de la instalación del editor emacs. Ingresamos a la consola y digitamos el siguiente comando para instalar el editor emacs:

```
pedro@pedrogs:~$ sudo aptitude install emacs
```

Luego para acceder al editor instalado escribimos en la consola el nombre del mismo:

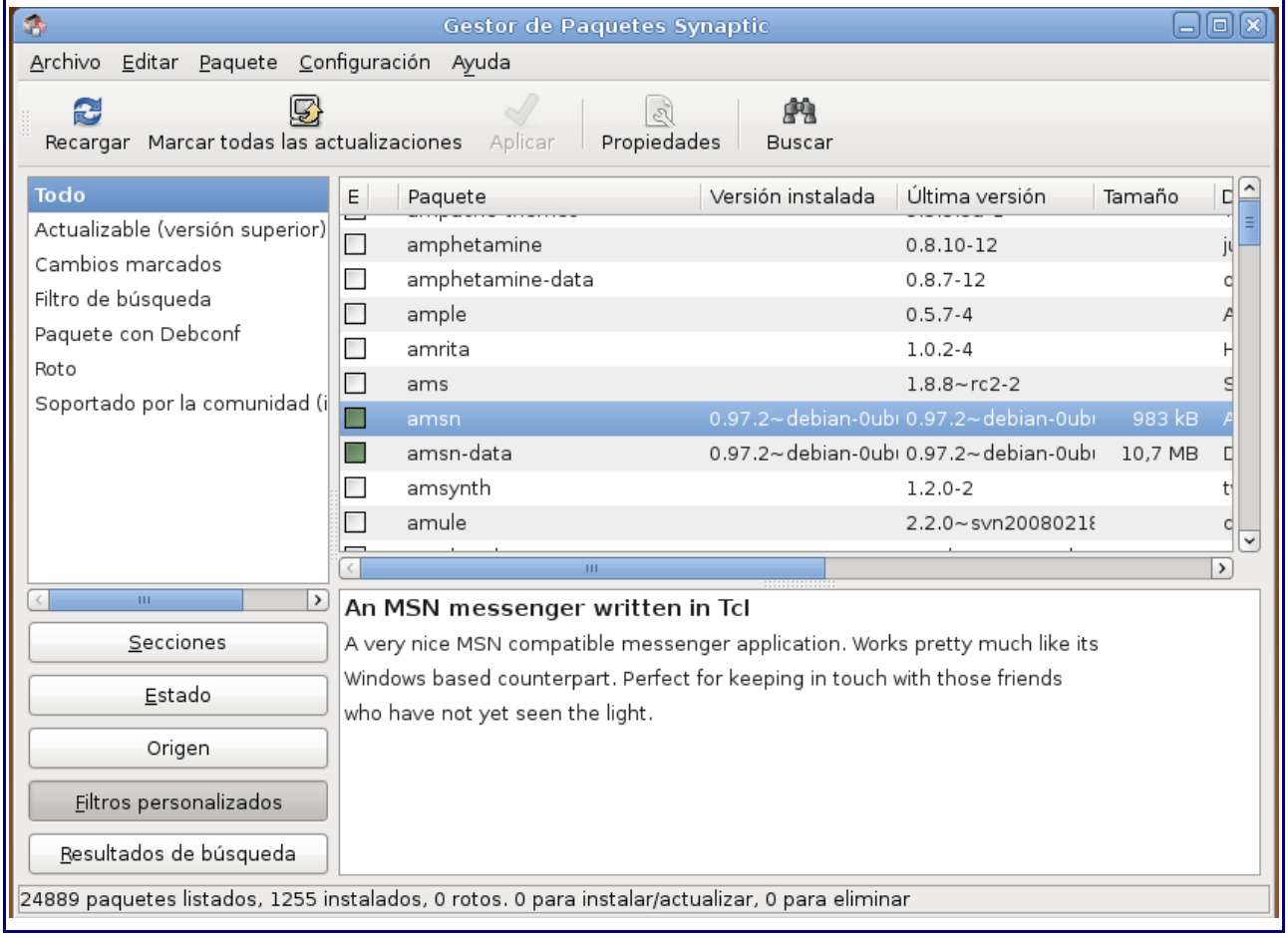
```
pedro@pedrogs:~$ emacs
```

Si deseamos desinstalar el paquete emacs incluyendo sus archivos de configuración escribimos en la consola en siguiente comando.

```
root@pedrogs:/home/pedro# aptitude remove --purge emacs
```

1.4 Gestor de paquetes Synaptic

Synaptic es una avanzada y potente herramienta gráfica para instalar o eliminar aplicaciones del sistema de una manera sencilla. Para ingresar a la aplicación tanto en Debian como en Ubuntu vamos al menú: **Sistema --- Administración --- Gestor de paquetes Synaptic**



Para instalar o desinstalar un paquete (programa), simplemente se selecciona el paquete y se oprime el **botón Aplicar**. En el siguiente enlace (http://www.vimeo.com/434596) pueden ver un video sobre la instalación de aplicaciones utilizando Synaptic.

1.5 Instalaciones comunes para todos los sistemas GNU/Linux (.tgz o .tar.gz)

En los sistemas operativos GNU/Linux existen ficheros con los ficheros fuentes de cualquier aplicación para ser compilados en nuestro sistema, estos archivos suelen tener las extensiones .tgz, .tar.gz y .tar.bz2.

Para la instalación de la aplicación **amsn**, a través de los ficheros fuentes debemos seguir los siguientes pasos:

- Descargar los fuentes del sitio web (http://www.amsn-project.net/linux-downloads.php), descargamos los fuentes del programa amsn SOURCE.
- Ingresamos a una determinada terminal y nos ubicamos dentro del directorio donde se encuentra el fichero descargado llamado amsn-0.97.2.tar.bz2.
- Descomprimir el fichero descargado amsn-0.97.2.tar.bz2, mediante la siguiente instrucción: # tar -xvvjf amsn-0.97.2.tar.bz2
- Una vez que descomprimimos y desempaquetamos el fichero con el comando anterior, obtenemos una carpeta con el nombre del fichero (amsn-0.97.2).
- La instalación de un aplicativo a partir del código fuente, requiere la configuración de las fuentes para la distribución, construcción de los ficheros con base a la distribución e instalación del aplicativo. Cabe aclarar que es necesario disponer de las cabeceras (linux-headears) de la distribución instalada.
- Ingresamos al directorio amsn-0.97.2, mediante el comando: # cd amsn-0.97.2

Cabe aclarar que esta aplicación posee ciertas dependencias. Para cada una de estas dependencias es útil, instalar las ultimas versiones de las mismas.:

tcl-dev
tk-dev
libpng-dev
libjpeg-dev

- Ejecutamos el fichero configure, mediante la instrucción: # ./configure, con el objetivo de identificar si faltan dependencias que ocasionen la no instalación del aplicativo. Al momento de la configuración encontramos los siguientes errores:

```
checking tcl build dir... configure: error: Unable to find Tcl directory or Tcl package is not tcl-dev.
checking tk build dir... configure: error: Unable to find Tk directory or Tk package is not tk-dev
configure: error: libpng is required
configure: error: libjpeg is required
```

- Al encontrar el error este, indica que debemos instalar el paquete **tcl-dev**, la ultima versión tcl8.5-dev, instalar el paquete **tk-dev**, la ultima versión tk8.5-dev, instalar el paquete libpngxx-dev, instalar el paquete libjpegxx-dev, donde xx indica un número de versión para la librería.

- Después de instalada cada una de las dependencias y la correcta configuración de las fuentes, mediante la instrucción ./configure, debemos compilar los ficheros mediante la instrucción make, la cual permite la compilación de los ficheros fuentes para la distribución sobre la cual se esta trabajando y una nueva versión del ejecutable del aplicativo.

```
samir@Personal:~/Desktop/amsn-0.97.2$ make
```

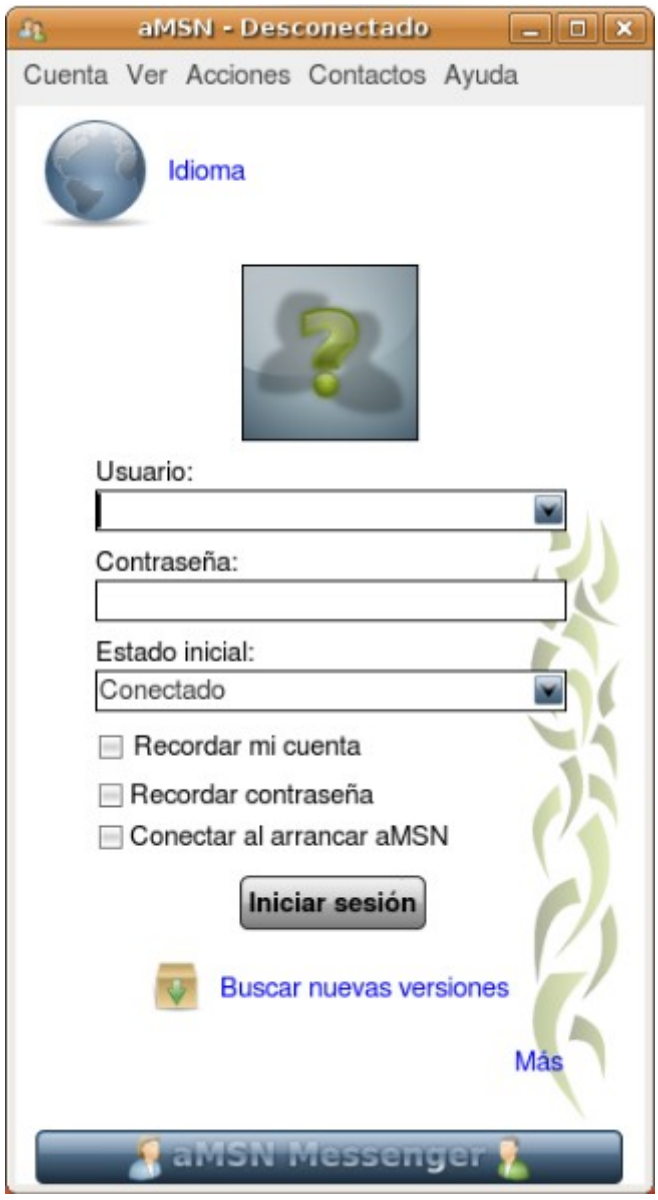
- Ahora para la instalación del aplicativo, ya construido debemos ejecutar en modo privilegiado el comando make install.

```
samir@Personal:~/Desktop/amsn-0.97.2$ sudo make install

rm -Rf /usr/share/amsn
mkdir -p /usr/share/amsn
mkdir -p /usr/bin
mkdir -p /usr/share/applications/
mkdir -p /usr/share/pixmaps/
find /usr/share/amsn -name '.svn' -print | xargs rm -Rf
ln -sf /usr/share/amsn/amsn /usr/bin/amsn
ln -sf /usr/share/amsn/amsn-remote /usr/bin/amsn-remote
ln -sf /usr/share/amsn/amsn-remote-CLI /usr/bin/amsn-remote-CLI
cp ./amsn.desktop /usr/share/applications/
ln -sf /usr/share/amsn/desktop-icons/48x48/apps/amsn.png /usr/share/pixmaps/
```

- Al ejecutar esta instrucción instalamos nuestro aplicativo, creamos directorios para el mismo, así como su acceso directo. Para ejecutar la aplicación amsn debemos ejecutar en la terminal el siguiente comando.

```
samir@Personal:~/Desktop/amsn-0.97.2$ amsn
```



1.6 Actividades

- Elaborar un cuadro comparativo con las semejanzas y diferencias entre las herramientas de instalación de paquetes: dpkg, aptitude y apt-get.

- Realizar la instalación de una aplicación en Debian y Ubuntu a partir de los ficheros fuentes (tgz ó .tar.gz). Documentar los pasos de la instalación en la maquina de prueba y especificar las dependencias de dicho paquete.

- Tomar tres paquetes diferentes y realizar la instalación con dpkg, apt-get y Synaptic, tanto en Debian como en Ubuntu. Documentar los pasos de la instalación en la maquina de prueba para cada paquete y especificar las dependencias de los paquetes.

- Que diferencia hay entre un fichero con extensión .tar.gz y .tar con base a la información de estos ficheros.

2. Redireccionamiento y pipelines

A medida que los usuarios aumentan sus conocimientos en el uso de los comandos de la shell bash en los sistemas GNU/Linux, se puede tener control de la entrada y la salida estándar en dichos sistemas. Esto permite que la ejecución de un comando no sea desplegada directamente en pantalla, sino que sirvan de entrada o de argumento a otro comando a ejecutarse, permitiendo la ejecución de varias órdenes en una sola línea de comando.

Cabe destacar que la siguiente descripción para las entradas y las salidas en el sistema:

Todas las ordenes que se digitadas por el teclado se conocen como:

- STDIN (**ST**AN**D**ARD**IN**PUT) ó stdin

Todas las salidas visualizadas o desplegadas en pantalla se conocen como:

- STDOUT (**ST**AN**D**ARD**OUT**PUT) ó stdout
- STDERR (**ST**AN**D**ARD**ERR**OR) ó stderr

Cuando se inicia una sesión en la shell bash se habilitan tres ficheros ó procesos correspondientes a los descriptores anteriores (stdin, stdout y stderr), los cuales son enumerados de la siguiente manera:

- El proceso **0** es la entrada estándar (**stdin**) y corresponde a lo digitado por el teclado.
- El proceso **1** es la salida estándar (**stdout**) y corresponde a lo mostrado en la pantalla.
- El proceso **2** es la salida de errores (**stderr**) y corresponde a los mensajes de error visualizados en pantalla.

Como se muestra en el recuadro anterior **stdin** es el fichero correspondiente a la entrada estándar, el cual se asigna al teclado, esta es la fuente de donde se toman los datos del proceso. La **stdout** es el fichero correspondiente a la salida estándar, esta se asigna normalmente a la pantalla y es a donde el proceso que se ejecuta envía los datos. **stderr** es el fichero en donde se escriben los mensajes de error, esta asignado a la pantalla y es donde el proceso que se ejecuto envía los mensajes de error.

Teniendo en cuenta lo anterior cuando escribimos un comando (programa) en la consola para que sea ejecutado normalmente la entrada estándar de dicho comando viene del

teclado y la salida estándar ó salida de error son visualizados por pantalla pero ambos son ficheros independientes. De esta manera podemos controlar que la salida de un proceso sea la entrada de otro y que a partir de una entrada se pueda modificar la salida.

2.1 Redireccionamiento

Cuando se redirecciona puede enviarse la salida de la ejecución de un determinado comando a otro dispositivo diferente a la pantalla, en este caso puede ser a un fichero.

Las formas más comunes de redireccionar son:

>: Comando > Archivo

Redirecciona la salida de la ejecución del comando hacia un archivo existente, si este no existe el archivo es creado. Si el archivo existe su contenido será modificado.

>>: Comando >> Archivo

Esta es otra forma de redireccionar la salida de la ejecución de un comando hacia un archivo existente pero añadiendo o anexando, a la información que ya existe en el archivo mencionado. Si este no existe el archivo será creado.

<: Comando < Archivo

La entrada estándar o argumentos del comando a ejecutarse se leen desde el archivo, en este caso no se producen cambios en el fichero de donde se tomo la entrada.

Ejemplos de manejo de redireccionamiento:

Si deseamos almacenar en un fichero plano el listado de todos los ficheros que se encuentran en el directorio de trabajo de un usuario del sistema con toda su información ejecutamos el siguiente comando:

```
pedro@pedrogs:~$ ls -al > misFicheros
```

El comando ls -al permite listar el contenido del directorio de trabajo del usuario pedro en el cual me encuentro y redirecciona la salida almacenando la información en el archivo de nombre misFicheros, luego si queremos ver su contenido simplemente se digitan alguno de estos dos comandos comando:

```
pedro@pedrogs:~$ cat misFicheros
```

```
pedro@pedrogs:~$ gedit misFicheros
```

Si quiero enviar el contenido del fichero que contiene la información de los usuarios creados en el sistema a un fichero de fácil acceso en mi directorio de trabajo utilizo el comando:

```
pedro@pedrogs:~$ cat /etc/passwd > misUsuarios
```

El comando cat /etc/passwd muestra la información de los usuarios del sistema y la salida se almacena en el directorio de trabajo del usuario pedro, en un archivo llamado misUsuarios, luego si queremos ver su contenido simplemente se digitan alguno de estos dos comandos:

```
pedro@pedrogs:~$ cat misUsuarios

pedro@pedrogs:~$ gedit misUsuarios
```

A continuación se escribirla una cadena de texto la cual se almacenara en un fichero llamado ingSistemas, como el fichero no existe este se crea en el directorio en el que me encuentro:

```
pedro@pedrogs:~$ echo "Ingeniería de Sistemas Linea Uno" > ingSistemas
```

El comando echo permite mostrar por pantalla una cadena de texto, en este caso la cadena de texto se almacena en el fichero ingSistemas. Entonces el contenido del fichero será:

```
pedro@pedrogs:~$ cat ingSistemas
Ingeniería de Sistemas Linea Uno
pedro@pedrogs:~$
```

Ahora si quiero anexar una nueva línea al fichero ingSistemas sin perder la información ya almacenada utilizo el siguiente comando:

```
pedro@pedrogs:~$ echo "Ingeniería de Sistemas Linea Dos" >> ingSistemas
```

Entonces el contenido del fichero será la información anterior mas la nueva información que se anexo o añadió, sin que se perdiera la información que existía en el fichero:

```
pedro@pedrogs:~$ cat ingSistemas
Ingeniería de Sistemas Línea Uno
Ingeniería de Sistemas Línea Dos
pedro@pedrogs:~$
```

Cuando se desea tener la información de un fichero en otro que ya exista sin que se borre la información del fichero hacia el cual se redirecciona la salida estándar, se utiliza el siguiente comando:

Tenemos el fichero asignarMAC el cual contiene la siguiente información:

```
pedro@pedrogs:~$ cat asignarMAC
****** Asignar Dirección MAC ******
pedro@pedrogs:~$ sudo ifdown eth0
pedro@pedrogs:~$ sudo ifconfig eth0 hw ether 00:13:8F:47:df:30
pedro@pedrogs:~$ sudo ifup eth0
pedro@pedrogs:~$
```

Entonces quiero agregarle a asignarMAC la información del fichero asignarIP, entonces se utiliza el comando:

```
pedro@pedrogs:~$ cat asignarIP >> asignarMAC
```

Si observamos el contenido del fichero asignarMAC se anexo el contenido del fichero asignarIP:

```
pedro@pedrogs:~$ cat asignarMAC
****** Asignar Dirección MAC ******
pedro@pedrogs:~$ sudo ifdown eth0
pedro@pedrogs:~$ sudo ifconfig eth0 hw ether 00:13:8F:47:df:30
pedro@pedrogs:~$ sudo ifup eth0


****** Asignar Dirección IP Universidad******
SSistemas
IP = 172.16.9.250
Mascara = 255.255.254.0
Puerta de enlace = 172.16.8.1
DNS = 172.16.14.12
pedro@pedrogs:~$
```

Para guardar la información de los usuarios del sistema en un fichero llamado usuarios, donde la entrada estándar del comando a ejecutar, será la información que contenga el fichero /etc/passwd:

```
pedro@pedrogs:~$ tee usuarios < /etc/passwd
```

El comando tee se utiliza para guardar la información que se toma del fichero /etc/passwd, en este caso la entrada del comando se toma del fichero.

2.2 Pipelines

Los pipelines también son conocidos como tuberías, las cuales proporcionan filtros que permiten redireccionar la salida de un comando como la entrada de otro comando. El símbolo utilizado para representar el pipe es el carácter **|**.

Ejemplos utilizando tuberías:

Para guardar la información de los grupos correspondiente a algún usuario especifico del sistema en un fichero, podemos usar el siguiente comando:

```
pedro@pedrogs:~$ cat /etc/group | grep "pedro" | tee grupoPedro
```

El comando **cat /etc/group** despliega la información de los grupos creados en el sistema, luego el comando **grep** busca todas las líneas del fichero donde aparece la palabra pedro el cual es el nombre del usuario y por ultimo con el comando **tee** guardamos la información desplegada en el fichero grupoPedro. Si queremos ver la información de dicho fichero ejecutamos el comando:

```
pedro@pedrogs:~$ cat grupoPedro
pedro@pedrogs:~$ gedit grupoPedro
```

Si un usuario desea contar el número de líneas que contiene un determinado fichero, que contiene la información de los ficheros con extensión .odt en su directorio de trabajo, puede utilizar el siguiente comando:

```
pedro@pedrogs:~$ find /home/pedro -name "*.odt" | tee salidaODT | wc -l
```

El comando **find /home/pedro -name "*.odt"** busca todos los ficheros con extensión .odt existentes dentro del directorio /home/pedro o cualquiera de los sub-directorios, luego la información de salida se almacena en el fichero salidaODT con el comando **tee salidaODT** por ultimo el comando **wc -l** cuenta cuantas líneas hay en el fichero salidaODT.

2.3 Actividades

- Utilizando la consola realizar y documentar dos ejemplos prácticos de redireccionamiento utilizando >& y >>&.
- Ejecutar un comando que liste el contenido del directorio de trabajo de un usuario, pero mostrando solamente los campos correspondientes al tamaño (KB, MB o GB)

y nombre respectivamente, de todos los ficheros que se encuentren en dicho directorio, el listado debe ser almacenado en un fichero llamado listadoUsuario.txt y mostrar el numero de líneas que tiene este fichero.

- Ejecutar un comando que almacene en un fichero llamado procesos.txt todos los procesos relacionados con el entorno gnome, solamente con la información correspondiente al ID del proceso y al nombre.

- Documentar que función realiza el siguiente comando ejecutado en la shell bash:

```
pedro@pedrogs:~$ find /home/pedro -atime -2 -name "*.odt" > archivos.txt | gedit archivos.txt
```

3. Creación de enlaces a ficheros

Dentro de una distribución GNU/Linux, son muchas las tareas realizadas por parte del usuario administrador, tales como instalación de paquetes, configuración de red, creación y eliminación de usuarios, manipulación de ficheros, etc.

La creación de ficheros por parte de los usuarios dentro de un distribución GNU/Linux, es una de las tareas realizadas con mayor cotidianidad. Sin embargo no solo es posible la creación también disponemos de opciones para eliminación, renombrado y por ultimo la creación de enlaces hacia estos. Por tal motivo un enlace se define como un vínculo que puede existir hacia un fichero.

Dentro de las distribuciones GNU/Linux (Redhat, Ubuntu, Debian, Suse, entre otras) se contemplan dos tipos de enlaces:

- Enlaces Simbólicos
- Enlaces duros o fuertes

Enlaces Simbólicos: Se define como la vinculación que existe hacia un fichero, de tal forma que es posible acceder al contenido de este desde una ubicación diferente al mismo. Cabe aclarar que este tipo de vinculación es débil, a razón que si el fichero al cual se le crea la vinculación es eliminado por completo, se pierde el enlace. Un enlace simbólico realiza el mismo objetivo que un acceso directo creado dentro de los sistemas operativos Windows.

Enlaces duros o fuertes: Este tipo de enlace es llamado enlace duro o fuerte, a razón de la creación de una copia en disco del fichero original al cual se le crea el enlace. Este tipo de enlace a diferencia de un enlace simbólico, no depende del fichero original. Es decir si el fichero al cual se le creo el enlace fuerte es eliminado, no se pierde la información hacia la cual apunta el enlace.

Para la creación de un enlace a través de una línea de comando, es necesario hacer uso del comando **ln**. La sintaxis general del comando es la siguiente:

```
ln <parámetros> <rutaDirectorio> <nombreElenlace>
```

- parámetros: **-s** Indica la creación de un enlace de tipo simbólico
- rutaDirectorio: Indica la ruta del fichero al cual se creara el enlace
- nombreElenlace: Indica el nombre que tendrá el enlace ha crear

3.1 Ejemplos utilizando enlaces

Para la creación de un enlace, es necesario ingresar a una determinada Terminal y en modo privilegiado se realizan los siguientes pasos:

- En las distribuciones Ubuntu o Debian con entorno de escritorio Gnome ingresamos a la terminal así: **Aplicaciones --- Accesorios --- Terminal**

Para la creación de un enlace simbólico empleamos el comando ln, de la siguiente forma (Ejemplo uno):

```
pedro@pedrogs:~$ ln -s /home/pedro/clasesHerencia.odt enlaceherencia
```

Como observamos el parámetro -s indica la creación de un enlace simbólico llamado enlaceherencia hacia el fichero clasesHerencia.odt, para identificar que en realidad ha sido creado correctamente el enlace, ejecutamos el comando ls -l y obtenemos:

```
pedro@pedrogs:~$ ls -l
total 616
drwx------  2 pedro pedro   4096 2008-11-22 18:31 amsn_received
-rw-r--r--  1 pedro pedro    150 2009-01-29 09:09 asignarIP
-rw-r--r--  1 pedro pedro    149 2009-01-29 09:09 asignarIP~
-rw-r--r--  1 pedro pedro    316 2009-01-29 09:12 asignarMAC
-rw-r--r--  1 pedro pedro    169 2009-01-29 09:04 asignarMAC~
drwxr-xr-x  5 pedro pedro   4096 2009-01-28 14:47 Catedra
-rw-r--r--  1 pedro pedro 105213 2008-11-27 15:56 clasesHerencia.odt
-rwxrwxrwx  1 pedro pedro 122856 2008-11-26 10:27 Concepto_Clases.pdf
-rw-r--r--  1 pedro pedro    268 2008-11-27 21:21 correos
-rw-r--r--  1 pedro pedro    245 2008-11-27 21:20 correos~
drwxrwxrwx  6 pedro pedro   4096 2008-11-26 10:25 Curso_SENA_Java_3
drwxr-xr-x  2 root  root    4096 2009-01-29 09:43 Desktop
drwxr-xr-x  2 pedro pedro   4096 2009-01-08 09:53 Documentos
lrwxrwxrwx  1 pedro pedro     30 2009-01-29 14:04 enlaceherencia ->
/home/pedro/clasesHerencia.odt
```

Como podemos apreciar al crear un enlace simbólico, este se identifica con una letra **l** al inicio de los permisos correspondientes y observamos que el nombre del enlace apunta hacia el fichero hacia el cual se ha creado el enlace.

Para la creación de un enlace simbólico empleamos el comando ln, de la siguiente forma (Ejemplo dos):

```
pedro@pedrogs:~$ ln -s /home/pedro/Imágenes/ enlaceimagenes
```

Como observamos el parámetro -s indica la creación de un enlace simbólico llamado enlaceimagenes hacia el directorio Imágenes, para identificar que en realidad ha sido creado correctamente el enlace, ejecutamos el comando ls -l y obtenemos:

```
pedro@pedrogs:~$ ls -l
total 636
drwx------  2 pedro pedro   4096 2008-11-22 18:31 amsn_received
-rw-r--r--  1 pedro pedro    150 2009-01-29 09:09 asignarIP
-rw-r--r--  1 pedro pedro    149 2009-01-29 09:09 asignarIP~
-rw-r--r--  1 pedro pedro    316 2009-01-29 09:12 asignarMAC
-rw-r--r--  1 pedro pedro    169 2009-01-29 09:04 asignarMAC~
drwxr-xr-x  5 pedro pedro   4096 2009-01-28 14:47 Catedra
-rw-r--r--  1 pedro pedro 105213 2008-11-27 15:56 clasesHerencia.odt
-rwxrwxrwx  1 pedro pedro 122856 2008-11-26 10:27 Concepto_Clases.pdf
-rw-r--r--  1 pedro pedro    268 2008-11-27 21:21 correos
-rw-r--r--  1 pedro pedro    245 2008-11-27 21:20 correos~
drwxrwxrwx  6 pedro pedro   4096 2008-11-26 10:25 Curso_SENA_Java_3
drwxr-xr-x  2 root  root    4096 2009-01-29 09:43 Desktop
drwxr-xr-x  2 pedro pedro   4096 2009-01-08 09:53 Documentos
-rw-r--r--  1 pedro pedro  18456 2009-01-29 14:11 ejemplos.odt
lrwxrwxrwx  1 pedro pedro     22 2009-01-29 14:12 enlaceimagenes ->
/home/pedro/Imágenes/
```

Para la creación de Enlaces fuertes o duros empleamos el comando ln, de la siguiente forma:

```
pedro@pedrogs:~$ ln /home/pedro/Concepto_Clases.pdf enlaceconcepto
```

Como se observa para la creación de este tipo de enlaces, no es necesario especificar algún parámetro, por lo demás basta seguir la sintaxis básica del comando.

Para identificar que en realidad ha sido creado correctamente el enlace, ejecutamos el comando ls -l y obtenemos:

```
pedro@pedrogs:~$ ls -l
total 764
drwx------  2 pedro pedro   4096 2008-11-22 18:31 amsn_received
-rw-r--r--  1 pedro pedro    150 2009-01-29 09:09 asignarIP
-rw-r--r--  1 pedro pedro    149 2009-01-29 09:09 asignarIP~
-rw-r--r--  1 pedro pedro    316 2009-01-29 09:12 asignarMAC
-rw-r--r--  1 pedro pedro    169 2009-01-29 09:04 asignarMAC~
drwxr-xr-x  5 pedro pedro   4096 2009-01-28 14:47 Catedra
-rw-r--r--  1 pedro pedro 105213 2008-11-27 15:56 clasesHerencia.odt
-rwxrwxrwx  2 pedro pedro 122856 2008-11-26 10:27 Concepto_Clases.pdf
-rw-r--r--  1 pedro pedro    268 2008-11-27 21:21 correos
-rw-r--r--  1 pedro pedro    245 2008-11-27 21:20 correos~
```

```
drwxrwxrwx  6 pedro pedro   4096 2008-11-26 10:25 Curso_SENA_Java_3
drwxr-xr-x  2 root  root    4096 2009-01-29 09:43 Desktop
drwxr-xr-x  2 pedro pedro   4096 2009-01-08 09:53 Documentos
-rw-r--r--  1 pedro pedro  22544 2009-01-29 14:16 ejemplos.odt
-rwxrwxrwx  2 pedro pedro 122856 2008-11-26 10:27 enlaceconcepto
```

Como observamos este tipo de enlace se diferencia del simbólico, al no mostrar a donde esta apuntado el enlace creado.

3.2 Actividades

- Explicar 2 ventajas y desventajas de la creación de enlaces simbólicos y duros.

- Tomar dos directorios cualesquiera pertenecientes al usuario en sesión. Crear dos enlaces de tipo simbólico con base a cada directorio, el nombre para cada enlace debe estar compuesto por el nombre de cada directorio y la fecha de creación del enlace.

  Ejemplo: nombredirectorio:imagenes
  nombredelenlace:imagenes-2009-03-01

- Crear dos enlaces duros sobre el directorio personal del usuario en sesión.

4. Niveles de ejecución (runlevel)

El proceso de arranque para un sistema operativo GNU/Linux, se desarrolla con base a la consecución de diversas fases o etapas, tales como búsqueda del MBR (sector de arranque), cargar del kernel y por ultimo iniciación de servicios. Este ultimo proceso es realizado por un programa llamado init, cuyo objetivo es dar inicio a un determinado nivel de ejecución (runlevel) sobre el cual trabajara la maquina o equipo.

Los niveles de ejecución o runlevels son distintos estados en los cuales puede iniciar un sistema operativo GNU/Linux. en la actualidad existen 7 niveles de ejecución sobre los que puede operar o trabajar un sistema operativo GNU/Linux. La siguiente tabla presenta cada nivel de ejecución y descripción del mismo:

Nivel de ejecución	Descripción
0	Nivel de ejecución conocido como **halt** se encarga de detener todos los procesos activos en el sistema, con el objetivo del correcto apagado del equipo.
1	Nivel de ejecución conocido como monousuario o single user, a razón de permitir la sesión de un único usuario por defecto inicia como usuarios root. Este nivel de ejecución es empleado para tareas de mantenimientos del sistema.
2	Nivel de ejecución multiusuario, sin soporte para red.
3	Nivel de ejecución multiusuario, con soporte para red.
4	Indefinido o sin uso.
5	Nivel de ejecución multiusuario, con capacidad gráfica (X window).
6	Nivel de ejecución de reinicio del sistema.

Cabe aclarar que cada uno de estos niveles de ejecución, dispone de un directorio especifico. Dichos niveles de ejecución se pueden encontrar dentro del directorio /etc. La siguiente tabla muestra el nivel de ejecución y su respectivo directorio de trabajo.

Nivel ejecución	Directorio
0	/etc/rc0.d
1	/etc/rc1.d
2	/etc/rc2.d
3	/etc/rc3.d
4	/etc/rc4.d
5	/etc/rc5.d
6	/etc/rc6.d

Cada directorio perteneciente a los distintos niveles de ejecución poseen distintos enlaces simbólicos a ficheros, utilizados para el inicio y parada de cada uno de los procesos ha ejecutar o detener al ingresar ha un nivel de ejecución. La creación de este tipo de enlace se realiza a partir de ficheros creados dentro del directorio /etc/init.d, es decir cada proceso o servicio ha iniciar o detener dentro de un nivel de ejecución debe poseer su fichero correspondiente dentro de este directorio. Por ejemplo:

```
samir@Personal:~$ cd /etc/rc0.d/
samir@Personal:/etc/rc0.d$ ls -l
total 4
lrwxrwxrwx 1 root root  13 2008-11-13 05:34 K01gdm -> ../init.d/gdm
lrwxrwxrwx 1 root root  17 2008-11-13 05:34 K01usplash -> ../init.d/usplash
lrwxrwxrwx 1 root root  16 2008-11-13 05:34 K16dhcdbd -> ../init.d/dhcdbd
lrwxrwxrwx 1 root root  16 2008-11-13 05:34 K20apport -> ../init.d/apport
lrwxrwxrwx 1 root root  20 2008-11-13 06:17 K20atieventsd -> ../init.d/atieventsd
lrwxrwxrwx 1 root root  22 2008-11-13 05:34 K20avahi-daemon -> ../init.d/avahi-daemon
lrwxrwxrwx 1 root root  20 2008-11-13 05:34 K25hwclock.sh -> ../init.d/hwclock.sh
lrwxrwxrwx 1 root root  13 2008-11-13 05:34 K39ufw -> ../init.d/ufw
lrwxrwxrwx 1 root root  20 2008-11-13 05:34 K50alsa-utils -> ../init.d/alsa-utils
lrwxrwxrwx 1 root root  26 2008-11-13 05:34 K59mountoverflowtmp ->
../init.d/mountoverflowtmp
lrwxrwxrwx 1 root root  21 2008-11-13 05:34 K99laptop-mode -> ../init.d/laptop-mode
-rw-r--r-- 1 root root 355 2008-04-19 00:05 README
lrwxrwxrwx 1 root root  41 2008-11-13 05:34 S01linux-restricted-modules-common ->
../init.d/linux-restricted-modules-common
```

Como apreciamos en el ejemplo anterior, dentro del directorio /etc/rc0.d, observamos distintos enlaces simbólicos a ficheros ubicados dentro del directorio /etc/init.d. K01gdm, S01linux-restricted-modules-common son ejemplos de estos enlaces.

Para la creación de este tipo de enlace dentro de un determinado nivel de ejecución se debe seguir ciertas convenciones al momento de estipular el nombre del enlace.

Si deseamos definir el inicio de un servicio o proceso para un nivel de ejecución debemos comenzar el nombre del enlace con la letra **S**, seguido de un valor numérico entero que indica el nivel de prioridad para ejecutar el servicio cuyo valor oscila de -20 a 20 y por ultimo el nombre relativo al servicio. Cabe aclarar que los valores de -20 a 0 para el nivel de prioridad son exclusivos para los procesos pertenecientes al superusuario (root). Ahora si el nombre del enlace inicia con la letra **K**, indica que el servicio sera detenido. Ejemplo:

```
lrwxrwxrwx 1 root root  41 2008-11-13 05:34 S01linux-restricted-modules-common ->
../init.d/linux-restricted-modules-common
```

En la creación de este enlace, apreciamos que el nombre del enlace S01linux-restricted-modules-common Cumple con la especificación para iniciar un servicio llamado linux-restricted-modules-common cuya prioridad es 01.

lrwxrwxrwx 1 root root 13 2008-11-13 05:34 K01gdm -> ../init.d/gdm

En la creación de este enlace, apreciamos que el nombre del enlace K01gdm Cumple con la especificación para detener un servicio llamado gdm cuya prioridad es 01.

4.1 Actividades

- ¿Explicar con sus palabras como se controla, qué servicios se deben iniciar al pasar el sistema a un determinado nivel de ejecución o runlevel?
- ¿Que fichero es necesario configurar en un sistema operativo GNU/Linux, con el objetivo de definir un nivel de ejecución predeterminado?
- ¿Explicar 3 formas de iniciar GNU/Linux en el runlevel single-user?

5. Introducción a la Programación ShellScript

Cada vez que el usuario del sistema GNU/Linux afianza sus conocimientos en el uso de la shell bash, observara que se pueden realizar varias operaciones al mismo tiempo pero existirá la necesidad de introducir varios comandos a la vez para lograr dicho objetivo. La programación Shell Script permite almacenar varias ordenes en un fichero para que sean interpretadas una a una, además, es apropiada para la administración de nuestro sistema.

Un Shell Script es un archivo de texto que contiene una serie de instrucciones, las cuales son un conjunto de comandos que ejecuta la shell bash de manera ordenada. Los Scripts no necesitan ser compilados, para esto existe un programa ayudante o interprete (shell bash), para su ejecución.

En conclusión un Script es un archivo plano que contiene comandos y contiene instrucciones para evaluar algunas condiciones. Si queremos escribir un programa Shell Script solo se requiere contar con un editor de textos como: vi, emacs, gedit, vim o cualquier otro.

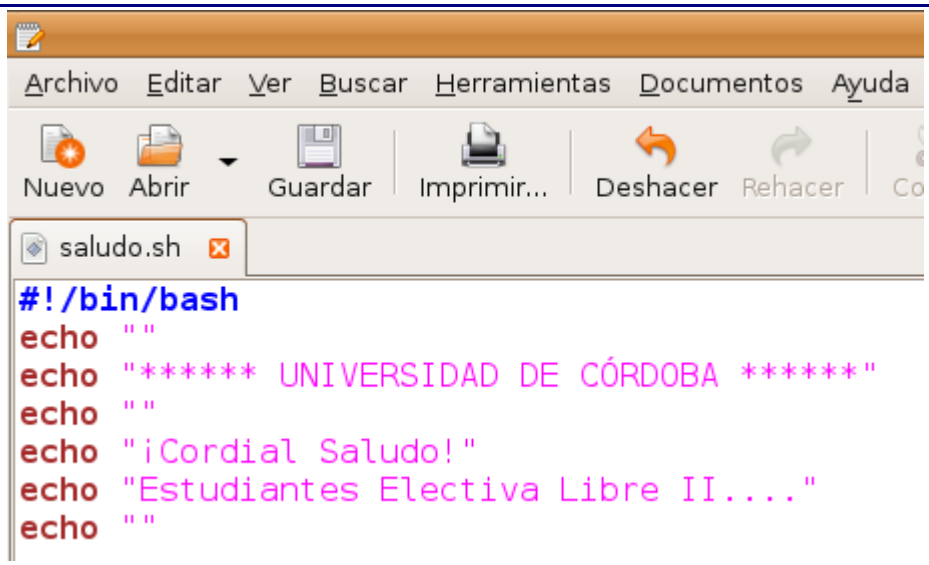
A continuación un ejemplo sencillo para crear nuestro primer Script, con el objetivo de entender mejor su utilidad, a medida que vamos aprendiendo mas de la programación ShellScript. Primero que todo aclararemos que en GNU/Linux los ficheros no necesitan tener una extensión, pero por convenio general, si queremos utilizaremos la extensión **.sh** para identificar nuestros Scripts entre el resto de archivos que pertenezcan al directorio donde se guarden.

Este primer programa o ShellScript, será un sencillo saludo de bienvenida utilizando el comando echo.

Primero que todo abrimos el editor gedit, del entorno de escritorio Gnome ejecutando el siguiente comando:

```
pedro@pedrogs:~$ gedit saludo.sh
```

También podemos acceder al editor a través del menú **Aplicaciones --- Accesorios --- Editor de textos** y escribimos las líneas de código que tendrá nuestro Script, como se ve en el siguiente gráfico.



Para ejecutarlo utilizo primeramente el comando bash, esta es la aplicación ayudante que interpreta los comandos del archivo, de esta manera reviso que el shellscript funciona correctamente:

```
pedro@pedrogs:~$ bash saludo.sh
```

Ahora modifico los permisos de acceso y le asigno los permisos de ejecución al fichero:

```
pedro@pedrogs:~$ chmod +x saludo.sh
```

Una vez agregado el permiso de ejecución, puedo ejecutarlo como un programa normal, gracias a la aplicación ayudante, que defino en la primera línea del script:

```
pedro@pedrogs:~$ ./saludo.sh
```

Para ejecutarlo como un programa normal del sistema, escribiendo únicamente su nombre. Puedo copiarlo en el directorio **/usr/local/bin** o /bin, definido en la variable de entorno $PHAT:

```
root@pedrogs:/home/pedro# cp saludo.sh /usr/local/bin
```

Una vez realizados todos los pasos anteriores puedo ejecutar el script como si se tratara de un comando normal del sistema, simplemente escribiendo su nombre en la consola:

```
pedro@pedrogs:~$ saludo.sh

****** UNIVERSIDAD DE CÓRDOBA ******

¡Cordial Saludo!
Estudiantes Electiva Libre II....

pedro@pedrogs:~$
```

5.1 Manejo de Variables

Una variable es un nombre al cual le asignamos un determinado valor, dicho valor puede ser numérico o una cadena de caracteres y pueden ser utilizadas en la programación ShellScript. A continuación veremos ejemplos que nos enseñen a utilizar las variables en nuestros scripts.

Creamos un script que se llame varuno, que tendrá el siguiente código:

```
#!/bin/bash
num1=20
num2=5
num3=10
resul=$[(num1+num3)/num2]
echo "El resultado es: $resul"
echo " "
exit 0
```

- La primera línea define la aplicación ayudante, la cual ejecutara las instrucciones del script.
- Las siguientes tres líneas muestran como se asigna el valor a las variables (num1, num2 y num3). Se debe tener en cuenta que no se pueden dejar espacios entre el nombre de la variable y el valor asignado (nomVar=valor).
- En la próxima línea se usa la expresión $[ ] para realizar la operación aritmética, la cual se asigna a la variable resul.
- Escribimos el resultado con la instrucción (echo "El resultado es: $resul"). Hay que tener en cuenta que para referirnos al nombre de una variable hay que colocar el signo **$** delante de su nombre ($nombreVariable).

A continuación probaremos con otro script al que llamaremos vardos, para asignar una cadena de texto a una variable:

```
#!/bin/bash
fraseUno="El nombre que escribió es: "
fraseDos="El usuario que esta usando la shell es: "
usuario=$(whoami)
echo "Escriba su el nombre:"
read nom
echo " "
echo $fraseUno $nom
echo $fraseDos $usuario
echo " "
exit 0
```

- La primera línea como se dijo anteriormente define la aplicación ayudante, la cual ejecutara las instrucciones del script.
- En la segunda y tercera línea se asigna una cadena de texto a las variables (fraseUno y fraseDos).
- La cuarta linea asigna a la variable usuario, el nombre del usuario que esta usando la shell.
- En las próximas dos líneas se pide que se escriba una cadena de texto y con el comando **read** podemos capturar la entrada del usuario en una variable llamada nom.
- Finalmente mandamos a mostrar el contenido de cada una de las variables con los comandos: **echo $fraseUno $nom** y echo **$fraseDos $usuario**.

Existen una serie de variables descritas por el sistema y serán a continuación:

- **$$:** Es el número de identificador del proceso del shell.
- **$?:** Resultado de la ejecución del comando. Cuando un programa termina su ejecución retorna un valor, si este es cero (0) el programa finalizo con éxito, de lo contrario arrojara un numero asociado a una causa especifica.
- **$0:** Permite obtener el nombre del script que se esta ejecutando o se ejecuto.
- **$1 hasta $9:** Corresponde desde el primer argumento hasta el noveno, con los que se ejecuta el comando.
- **$#:** Para tener el número de argumentos recibidos como entrada.

5.2 Manejo de argumentos (parámetros)

Los comandos que se ejecutan en una consola suelen tener argumentos o parámetros, tal como se muestra en el siguiente ejemplo:

```
pedro@pedrogs:~$ rm -R /home/pedro/prueba
```

Si observamos rm es el comando para borrar ficheros, -R es una opción del comando y la ruta del directorio /home/pedro/prueba es un argumento de dicho comando. A continuación veremos como pueden nuestros scripts tener opciones y argumentos igual a los demás comandos del sistema.

Escribimos un pequeño script con las siguientes líneas de código, al cual llamaremos argumentos:

```
#!/bin/bash
echo "Pasar argumentos para ser procesados..."
echo " "
echo "El nombre del Script es: $0"
echo "El primer argumento es: $1"
echo "El segundo argumento es: $2"
echo "El No. de argumentos es: $#"
exit 0
```

Después de probar el script, darle permisos de ejecución y guardarlo en el directorio /bin ó /usr/local/bin, miremos la salida:

```
pedro@pedrogs:~$ argumentos pedro juan
Pasar argumentos para ser procesados...

El nombre del Script es: /usr/local/bin/argumentos
El primer argumento es: pedro
El segundo argumento es: juan
El No. de argumentos es: 2
pedro@pedrogs:~$
```

- Podríamos decir que los argumentos son variables que ingresamos junto con el comando y estas pueden ser procesadas durante la ejecución del script. Por esta razón la variable $0 corresponde al nombre del script, la variable $1 al primer argumento que se pasa y la variable $2 al segundo y así sucesivamente tal como se menciono en el apartado anterior.

- La variable $# corresponde al numero de argumentos seguidos después del comando. De esta manera podemos manipular cada argumento como si fuera una variable in dependiente, ósea $1, $2 ... $n.

5.3 Operadores de comparación

Cuando escribamos un script es necesario evaluar algún tipo de condición para que este realice la operación que deseamos. Por tal motivo tenemos la siguiente tabla de operadores para comparar números y cadenas de caracteres en cualquier instrucción de nuestro script:

- -eq: Igual
- -ne: Diferente
- -lt: Menor que
- -gt: Mayor que
- -le: Menor o igual
- -ge: Mayor o igual

5.4 Manejo de condicionales (if)

Con la ayuda del condicional if podemos hacer que los scripts evalúen condiciones y se comporten de acuerdo a las necesidades de cualquier problema. La estructura es la siguiente:

```
if [ Expresión ]; then
        instrucción 1
        instrucción 2
        instrucción 3
        instrucción N
fi


if [ Expresión ]; then
        instrucción
        instrucción
        else
                instrucción
                instrucción
fi
```

```
if [ Expresión 1 ]; then
        instrucción
        instrucción
        elif [ Expresión 2 ]; then
                instrucción
                instrucción
        else
                instrucción
                instrucción
        fi
```

Para entender mejor el uso de los condicionales escribiremos un script que determine cual de los números pasados como argumentos de nuestro comando es mayor que el otro. Para este propósito creamos un script que llamaremos comparar:

```
#!/bin/bash
echo "****** COMPARAR DOS NUMEROS ******"
num1=$1
num2=$2
if [ $num1 -gt $num2 ]; then
        echo "$num1 es mayor que $num2"
        elif [ $num2 -gt $num1 ]; then
                echo "$num2 es mayor que $num1"
        elif [ $num1 -eq $num2 ]; then
                echo "$num1 es igual que $num2"
        else
                echo "No escribió ningún numero"
        fi
exit
```

- En las líneas 3 y 4 estamos asignando los valores de los dos argumentos que introduzcamos a las variables num1 y num2 respectivamente.
- En la línea 5 validamos si el primer argumento es mayor que el segundo y escribimos el respectivo mensaje en la línea 6.
- En la línea 7 validamos si el segundo argumento es mayor que el primero y escribimos el respectivo mensaje en la línea 8.
- En la línea 9 validamos si el primer argumento es igual que el segundo y escribimos el respectivo mensaje en la línea 10.

Luego ejecutamos el script para probarlo:

```
pedro@pedrogs:~$ bash comparar 22 11
****** COMPARAR DOS NUMEROS ******
```

```
22 es mayor que 11
pedro@pedrogs:~$
```

Realicemos otro ejemplo para listar los ficheros de un directorio del sistema, siempre que este exista, debemos pasar como mínimo un argumento (el directorio), entonces escribimos lo siguiente en nuestro script llamado listar:

```
#!/bin/bash
directorio=$1
if [ $# -ne 1 ]; then
        echo "Se necesita el parámetro <directorio>"
        echo "Uso: $0 <directorio>"
        exit
fi
if [ ! -e $directorio ]; then
        echo "El directorio $directorio No existe"
        else
                echo "Estos son los ficheros del directorio $directorio:"
                ls -l $directorio
                echo " "
fi
exit 0
```

- La línea 2 captura el argumento que se pasa, correspondiente al directorio a listar en la variable directorio.
- En la línea 3 se valida si el comando que se escribe (listar), esta acompañado del argumento que se requiere (directorio), de lo contrario en las líneas 4 y 5 se escribe el modo de uso correcto para el comando o script.
- La línea 8 valida si el directorio pasado como parámetro no existe, si esto es verdad en la línea 9 se escribe que el directorio no existe.
- Si ocurre lo contrario a lo descrito en el punto anterior línea 10 (else), entonces se escribe un comentario en la línea 11 y se lista el contenido del directorio con el comando ls -l como se muestra en la línea 12.

5.5 Trabajar con el case

En programación muchas veces es necesario comprobar el valor de una variable con case y validar si coincide con algún valor esperado, esto es posible solucionarlo con el if pero cuando son muchas las posibles opciones a evaluar es mejor la ayuda de case, evitando escribir tantos if. Su estructura de uso es la siguiente:

```
#!/bin/bash
case $variable in
        valor1)
                instrucción
                instrucción
                ;;
        valor2)
                instrucción
                instrucción
                ;;
        valor3)
                instrucción
                instrucción
                ;;
        *)
                instrucción
                instrucción
                ;;
esac
```

Para entender el funcionamiento del case escribiremos un script llamado menuop que contiene una lista de opciones y dependiendo la opción escogida el programa ejecutara las instrucciones especificas. El código del script es el siguiente:

```
#!/bin/bash
echo "*******************************************"
echo "*            MENÚ DE OPCIONES           *"
echo "*******************************************"
echo "1. Abrir el editor de textos"
echo "2. Listar los fichero del directorio actual"
echo "3. Mostrar saludo de Bienvenida"
echo "4. Salir del Programa"
echo "_____________________________________________"
echo " "
echo  "Seleccione una opción:"
read op
case $op in
        1)
                gedit
                exit 0
                ;;
        2)
                ls -l
                exit 0
                ;;
        3)
                saludo.sh
                exit 0
                ;;
```

```
        4)
                exit 0
                ;;
        *)
                echo "No selecciono ninguna opción"
                exit 1
                ;;
esac
```

- Desde la línea 2 hasta la 11 son comentarios que mostraran la apariencia de nuestro programa cuando se ejecute.
- La línea 12 (read op), permite capturar el valor de la opción a evaluar en el case.
- En la línea 13 (case $op in), se evalúa dicha opción y dependiendo cual sea el caso, ejecutara las instrucciones necesarias.
- Si toma la opción 1 se ejecutan las líneas 14 a la 17 y se abre el editor de textos del entorno gnome.
- Si toma la opción 2 se ejecutan las líneas 18 a la 21 y se listan los ficheros del directorio actual.
- Si toma la opción 3 se ejecutan las líneas 22 a la 25 y se abre el programa saludo.sh (este fue el primer script que se hizo).
- Si toma la opción 4 se ejecutan las líneas 26 a la 28 y se sale del programa.
- Si no escogió ninguna opción valida se ejecutan las líneas 29 a la 32.

5.6 Bucles (for y while)

Además de la sintaxis estudiada hasta ahora existen en la programación shellscript bloques de programa conocidos como bucles que permiten ejecutar ciclos de comandos siempre que se cumpla una condición. Los bucles tienen una función similar a la de cualquier otro lenguaje de programación, pero permiten que un script ejecute ciertas órdenes si se cumple una condición, permitiendo que el script haga lo que el programador desea. Los bucles mas utilizados son el for y el while.

5.6.1 For

```
for  variable in lista_elementos; do
        comandos
        comandos
done
```

- variable es el nombre que tendrá la variable que controlara el ciclo, esta variable no esta precedida de $.

- lista-elementos contiene la lista de elementos a recorrer, esta lista puede ser una variable que contenga dichos elementos y estará precedida de $.

- Los comandos se ejecutaran tantas veces dependiendo del número de elementos de la lista.

A continuación escribiremos un shellscript que recorre el directorio de trabajo del usuario y muestra en pantalla el listado de ficheros con extensión odt.

```
#!/bin/bash
echo "Ejemplo ciclo for..."
for i in $HOME/*.odt; do
        echo $i
done
```

El siguiente script borra los archivos de texto de un determinado directorio, el cual será pasado como parámetro por el usuario. Los ficheros se borraran si el usuario lo desea:

```
#!/bin/bash
echo "Ejemplo 2 ciclo for..."
directorio=$1
if [ $# -ne 1 ]; then
        echo "Se necesita el parámetro <directorio>"
        echo "Uso: <comando> <directorio>"
        exit
fi
for i in $directorio/*.txt; do
        rm -i $i
done
exit 0
```

- La línea 3 captura el directorio dentro del cual se recorrerá la lista de ficheros.

- En la línea 4 se verifica que el directorio en donde se borraran los ficheros haya sido pasado como argumento.

- La línea 9 recorre (for i in $directorio/*.txt; do) todos los ficheros con extensión .txt que hay dentro del directorio.

- La línea 10 eliminara cada uno de estos ficheros, siempre que el usuario confirme su eliminación.

5.6.2 While

Otro bucle utilizado en la programación shellscript es el ciclo while, este se utiliza para que se ejecuten una serie de comandos o instrucciones si se cumple una determinada condición. Su estructura de uso es la siguiente:

```
while  [ Expresión ]; do
        comandos
        comandos
done
```

Para entender mejor el funcionamiento del while realizaremos un pequeño script que repita varias veces un mensaje pasado como argumento. La cantidad de veces que se repetirá el mensaje también sea pasada como argumento.

```
#!/bin/bash
cantidad=$1
mensaje=$2
contador=1
if [ $# -ne 2 ]; then
        echo "Se necesita los parámetros <cantidad> <mensaje>"
        echo "Uso: $0 <cantidad> <mensaje>"
        exit
fi
while  [ $contador -le $cantidad ]; do
        echo $mensaje
        contador=`expr $contador + 1`
done
```

- En las líneas 2 y 3 se capturan los valores de los argumentos: cantidad de veces a repetir y el mensaje respectivamente.
- La línea 4 se inicia una variable contador en uno, para controlar el ciclo while.
- En la línea 5 se valida que se pasen los dos parámetros requeridos (la cantidad y el mensaje), de lo contrario en las líneas 6 y 7 se muestra la forma de uso del comando.
- La línea 10 evalúa de que el contador sea menor o igual que la cantidad de veces a repetir el mensaje.
- Si se cumple la condición anterior la línea 11 manda a escribir el mensaje tantas veces sea necesario. La línea 12 incrementa el contador en uno cada vez que se entra al ciclo.

5.7 Actividades

- Investigar sobre el uso del comando test, realizar varios ejemplos (mínimo 5) y documentarlos.
- Investigar sobre otras expresiones para evaluar condicionales, diferentes a las expuestas en el modulo (Tipos de comprobaciones entre ficheros y directorios).
- Estudiar el uso de funciones en la programación shellscript, realizar des ejemplos prácticos y documentarlos.
- Estudiar el ciclo until realizar dos ejemplos prácticos y documentarlos.
- Elaborar un pequeño shell script, que permita saber cuales son los servicios ejecutados dentro de un determinado runlevel.
- Realizar un script que liste la información de los archivos contenidos en los directorios de trabajo de cada uno de los usuarios normales creados en le sistema.
- Escribir un shellscript que almacén en un fichero plano la información referente a todos los paquetes instalados en tu sistema.

Enlaces de Interés

Unidad Uno:

http://www.debian.org/international/spanish/contrib/paqifaz.html
http://www.reloco.com.ar/linux/debian/principiantes.html
http://www.guia-ubuntu.org/index.php?title=A%C3%B1adir_aplicaciones
http://www.configurarequipos.com/doc735.html
http://es.wikipedia.org/wiki/Advanced_Packaging_Tool
http://es.wikipedia.org/wiki/Sources.list
http://portallinux.wordpress.com/2008/05/28/instalar-xmms-en-ubuntu-804/

Unidad Dos:

http://es.tldp.org/DEBIAN/%257Ejfs/debian/doc/es/debian-guide-es/html/node171.html
ftp://ftp.uniovi.es/pub/linux/docs/LuCaS/DEBIAN/%257Ejfs/debian/doc/es/debian-guide-es/html/node171.html
http://es.kioskea.net/faq/sujet-387-guia-de-uso-del-shell-para-principiantes#xxx-las-redirecciones-y-los-pipelines
http://www.inaoep.mx/~moises/S.O./cunx12.html

Unidad tres:

http://bioinformatiquillo.wordpress.com/2007/12/30/enlaces-en-linux/
http://andalinux.wordpress.com/2008/09/02/manejo-de-enlaces-simbolicos-en-linux/
http://www.ant.org.ar/cursos/curso_intro/x1811.html

Unidad cuatro:

http://www.linuxparatodos.net/portal/staticpages/index.php?page=niveles-ejecucion
http://www.linuxparasereshumanos.com/2007/09/20/los-runlevels-de-linux/

Unidad cinco:

http://www.elviajero.org/antoniux/
http://xinfo.sourceforge.net/documentos/bash-scripting/bash-script-2.0.html
http://www.e-ghost.deusto.es/docs/shellScriptin.html
http://www.monografias.com/trabajos50/ejercicios-shell-script/ejercicios-shell-script2.shtml