Introducción a Unix: Unidad 4

Curso basado en uno propuesto por William Knottenbelt, UK, 2001

Daniel Millán, Nora Moyano & Evelin Giaroli

Facultad de Ciencias Aplicadas a la Industria, UNCuyo.

Junio de 2017

Intérpretes de órdenes de Unix

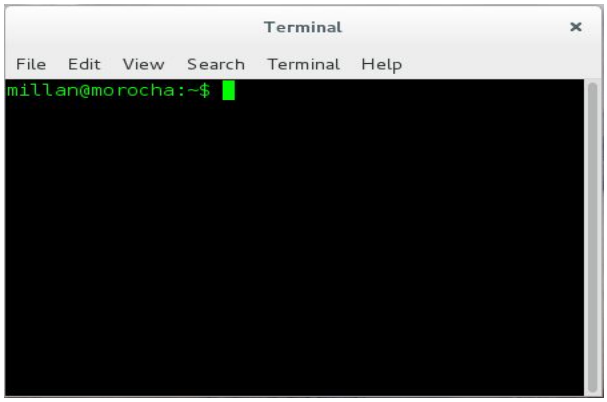
Contenido

1. Intérpretes de órdenes - ***Shell***

Una ***shell*** es un programa que lee y ejecuta órdenes del usuario para acceder a servicios del *SO*. Es un programa que funciona de *escudo* entre el usuario y el *SO*.

- Las ***shells*** suelen contar con características como:
  - ➢ control de trabajo
  - ➢ redirección de entrada y salida
  - ➢ un lenguaje de órdenes para escribir guiones (***scripts***) de ***shell***
- Hay varios tipos de ***shells*** disponibles en los sistemas Unix
  - ➢ por ejemplo: `sh, bash, csh, ksh, tcsh, ...`
- Cada uno de ellos posee un "lenguaje" de órdenes diferente.
- Aquí vamos a comentar el lenguaje de órdenes de la ***Bourne shell "sh"*** (Stephen Bourne de Bell Labs, 1977), ya que está disponible en casi todos los *SO* tipo Unix.
- ***sh*** es compatible con ***bash*** (*Bourne again shell*, 1989) y ***ksh*** (KornShell 1980).

- Discutir: ¿Cuál es la diferencia entre una *terminal* y una *shell*?



2. Archivos de órdenes por guiones - ***Scripts***

- Un *script* de *shell* es simplemente un archivo de texto ordinario que contiene una serie de órdenes en un lenguaje de una *shell* de Unix.
- En español cada vez es más aceptado el término guión para referirse a un *script*.
- Los archivos guión suelen ser identificados por el sistema a través del siguiente encabezamiento en el contenido del archivo, lo cual es conocido como ***shebang*** **(***hash-bang* o *sharpbang*): **#!**

No confundir shebang con "she bangs" que es una canción de Ricky Martin!

- A continuación del *shebang* se indica la ruta completa al intérprete de las órdenes contenidas en el mismo. Este método estándar permite que el usuario pueda ejecutar un programa interpretado (ascii) como si ejecutara un programa binario (compilado).

```
#!/bin/bash     #!/bin/ksh     #!/bin/csh
```

- Sirve para identificar el formato del fichero a modo de «número mágico».
- A veces los archivos de *guiones* del intérprete de órdenes de Bourne se identifican por la extensión «***.sh***».
- Además el *shebang* puede ser necesario en algunos casos, como en *scripts* hechos en AWK, donde el parámetro `-f` indica cuál es el *script* que tiene que cargarse.
- Ejemplo: si creamos un archivo "*programa.awk*" cuyo contenido se muestra a continuación

```
#!/usr/bin/awk -f
BEGIN{
print "parámetros:"
for (i = 1; i < ARGC; i++)
print "   " i ": " ARGV[i]}
```

Cuando se ejecute el *script* "*programa.awk*"

```
$./programa.awk argumento1 argumento2⏎
```

el *SO* tras leer la línea del *shebang* ejecutará:

```
$/usr/bin/awk -f ./programa.awk argumento1 argumento2
parámetros:
  1: argumento1
  2: argumento2
```

3. Variables de la *Shell* y de entorno

- Una *shell* le permite definir variables (como la mayoría de los lenguajes de programación). Una variable es un fragmento de información al que se le da un nombre.
- Una vez asignado un valor a una variable, se puede acceder a su valor anteponiendo **$** al nombre:

```
$pepe='Hola don Pepe Argento'⏎
$echo $pepe⏎
Hola don Pepe Argento
```

- Las variables creadas dentro de una *shell* son locales, por lo que sólo la *shell* puede acceder a ellas.
- La orden ***set*** muestra una lista de todas las variables definidas actualmente en la *shell*.
- Si desea que una variable sea accesible a otros programas fuera de la *shell*, esta se puede exportar al entorno ("*environment*"):

```
$export pepe
```
➔Ver ejercicio en clase *hola.c*

- Las **variables de entorno** forman un conjunto de valores dinámicos que normalmente afectan cómo funcionan los procesos en una *PC.*
- Ejemplo:
  - La variable de entorno `$PAGER` es utilizada por la orden **man** (y otros) para mostrar las páginas de ayuda.
  - Si cambiamos la forma en que se muestra la información, digamos:

    ```
    $export PAGER=cat⏎
    ```

    Para observar el efecto podemos ejecutar:

    ```
    $man man⏎
    ```

    Esto imprimirá en la salida estándar todo el contenido de la ayuda sobre **man**.
  - Ahora intente

```
$export PAGER=more⏎
$export PAGER=less⏎
```

- Para conocer qué variables de entorno son utilizados por una orden en particular, consulte el manual para dicha orden (*man prog*).
- `$EDITOR` especifica el editor a ser utilizado por defecto, p.ej. comúnmente ***nano***, ***vi*** o ***emacs***, o algún otro editor que prefiera.
- `$HOME` indica la ubicación del directorio de usuario.
- `$PWD, $OLDPWD, $SHELL, $TERM, ...`
- La variable de entorno `$PS1` define la cadena de caracteres del indicador principal de la *shell*. Para ello se incorporan mecanismos que permiten especificar partes comunes de la *shell*.
- Por ejemplo, en *bash* se puede utilizar ***\h*** para la máquina actual, ***\w*** para mostrar el camino completo del directorio de trabajo actual, ***\d*** para la fecha, ***\t*** para el tiempo, ***\u*** para el usuario y así sucesivamente (ver *man bash*, `PROMPTING`).

  $export `PS1='(\h) \w>'` ⏎ (o si queremos que nos muestre colores)

  $export `PS1='\[\e[1;32m\][\u@\h \W]\$\[\e[0m\]'` ⏎

  The shell prompt (or command line) is where one types commands. When accessing the system through a text-based terminal, the shell is the main way of accessing programs and doing work on the system.

  Prompt's colors:

```
# Black           \e[0;30m
# Blue            \e[0;34m
# Green           \e[0;32m
# Cyan            \e[0;36m
# Red             \e[0;31m
# Purple          \e[0;35m
# Brown           \e[0;33m
# Gray            \e[0;37m
# Dark Gray       \e[1;30m
# Light Blue      \e[1;34m
# Light Green     \e[1;32m
# Light Cyan      \e[1;36m
# Light Red       \e[1;31m
# Light Purple    \e[1;35m
# Yellow          \e[1;33m
# White           \e[1;37m
```

CONTROL COMMANDS AVAILABLE FOR PS1 PROMPT:

`\d` - the date in "Weekday Month Date" format (e.g., "Tue May 26")

`\e` - an ASCII escape character (033)

`\h` - the hostname up to the first .

`\H` - the full hostname

`\j` - the number of jobs currently run in background

`\l` - the basename of the shells terminal device name

`\n` - newline

`\r` - carriage return

`\s` - the name of the shell, the basename of $0 (the portion following the final slash)

`\t` - the current time in 24-hour HH:MM:SS format

`\T` - the current time in 12-hour HH:MM:SS format

`\@` - the current time in 12-hour am/pm format

`\A` - the current time in 24-hour HH:MM format

`\u` - the username of the current user

`\v` - the version of bash (e.g., 4.00)

`\V` - the release of bash, version + patch level (e.g., 4.00.0)

`\w` - Complete path of current working directory

`\W` - the basename of the current working directory

`\!` - the history number of this command

`\#` - the command number of this command

`\$` - if the effective UID is 0, a #, otherwise a `e$`

`\nnn` - the character corresponding to the octal number `nnn`

`\\` - a backslash

`\[` - begin a sequence of non-printing characters, which could be used to embed a terminal control sequence into the prompt

`\]` - end a sequence of non-printing characters

- Otra variable importante es `$PATH`. `$PATH` es una lista de directorios que la *shell* utiliza para localizar archivos ejecutables.
- Por ejemplo si `PATH` se establece como:

  `/bin:/usr/bin:/usr/local/bin:.` ('**.**' indica el directorio actual **¿y?**)
- Si escribimos `ls`, la shell buscará `/bin/ls`, `/usr/bin/ls`, etc.
- Existen otras variables importantes como `$LD_LIBRARY_PATH`, la cual es una lista de directorios utilizada por la *shell* para buscar bibliotecas, e.g. al compilar programas de C

(*libDINAMICA.so, libSTATICA.a*).

- La orden ldd permite conocer las librerías dinámicas utilizadas por un dado programa

```
$ldd hola
```

4. Guiones de órdenes sencillos - *shell scripts*

- Consideremos el siguiente guión de órdenes contenido en un archivo de texto llamado *simple*

```
#!/bin/sh
# Esto es un comentario
echo "El número de argumentos es $#"
echo "Los argumentos son $*"
echo "La primera es $1"
echo "Mi número de proceso es $$"
echo "Introduzca un número en el teclado:"
# NO utilice acentos para definir variables!
read numero
echo "El número que ha introducido es el $numero"
```

- **#!** es el *shebang*, *sh* es el tipo de *shell* utilizado, los argumentos de entrada son **$1**, **$2**, etc, en tanto que **$*** indica todos los argumentos, **$#** es el número de argumentos, **$$** es el PID del proceso, y **read** es una orden que asigna la entrada del teclado a la variable **numero**.

```
$./simple
```

5. Guiones de inicio en Unix

- Cuando se inicia sesión en una *shell*, ésta ejecuta una amplia secuencia de órdenes de inicio del sistema (normalmente llamando */etc/profile* en **sh**, **bash** y **ksh** y */etc/.login* en **csh**).
- A continuación buscará en su *home* el *guión/script* de inicio personal (*.profile* en **sh**, **bash** y **ksh** y *.cshrc* en **csh** y **tcsh**).
- Por esta razón su *script* de inicio es un buen lugar para definir variables de entorno como PATH, LD_LIBRARY_PATH, etc; **alias** de órdenes y funciones propias.
- Por ejemplo en **bash** para incluir el directorio ~/bin en su PATH se debe agregar la siguiente línea en su archivo *.profile*

```
export PATH=$PATH:~/bin
```

- Luego de modificar el archivo .profile se deben importar los cambios a la shell de trabajo actual

```
$source ~/.profile⏎
```

6. Guiones de inicio de *Bash* [wiki]

- Cuando *Bash* arranca, ejecuta las órdenes que se encuentran en diferentes guiones dependiendo de donde se llame esta *shell*:
  - ➢ Cuando se invoca a *Bash* como un *shell* interactivo al inicio de una sesión (**login shell**), en primer lugar lee y ejecuta órdenes desde el archivo `/etc/profile`, si existe. Después, busca `~/.bash_profile`, `~/.bash_login`, y `~/.profile`, en este orden, y lee y ejecuta las órdenes desde el primero que existe y es legible.
  - ➢ Cuando un *shell* interactivo que NO es un *login shell* arranca, *Bash* lee y ejecuta órdenes desde `~/.bashrc`, si existiese (una terminal).

7. Misceláneos

- Investigue sobre las siguientes órdenes o asignaciones que pueden ser útiles en *shell scripting*
  - ➢ **`nohup`** permite ejecutar un programa en segundo plano tal que es inmune frente a rupturas de comunicación y cierre de la *shell.*
  - ➢ **`/dev/null`** (periférico nulo) utilizado para redirigir la salida de un flujo de datos.
  - ➢ **`nice`** ajusta la prioridad de un proceso de -20 a 19.
  - ➢ **`make`** es una herramienta de gestión de dependencias (ver *Makefile*).
- Otros
  - ➢ **`nano`** es un editor de texto liviano
  - ➢ ***`su`*** permite cambiar de usuario.
  - ➢ **`id`** muestra datos de identificación del usuario.
  - ➢ **`df`** muestra el espacio libre de los discos/dispositivos.
  - ➢ **`du`** muestra el espacio usado por el disco/dispositivo o un directorio.
  - ➢ **`join`** cruza la información de dos archivos y muestra las partes que se repiten (**`split`** divide un archivo).