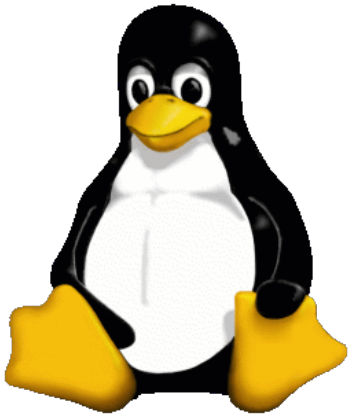


Introducción a UNIX®
Unidad 4

Daniel Millán & Nicolás Muzi

Abril 2022, San Rafael, Argentina

CONICET

MoΩCCAi
MODELADO COMPUTACIONAL EN CIENCIAS APLICADAS E INGENIERÍA

UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE CIENCIAS APLICADAS A LA INDUSTRIA

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

Intérpretes de órdenes de Unix

Scripts

Los temas que se cubrirán son:

1. Intérpretes de órdenes (='*commands*') ***shells***
2. Archivos de órdenes/procesamiento por guiones (***scripts***)
3. Variables de la *shell* y de entorno ($pepe, $PATH, …)
4. Guiones de intérpretes de órdenes (***shell scripts***) sencillos
5. Guiones de inicio en Unix
6. Guiones de inicio de Bash [wiki]
7. Misceláneos

Objetivos

- Conocer las principales características de una *shell*
- Diferenciar una *terminal* de una *shell*.
- Diferenciar las variables locales a una *shell* de las variables de entorno del SO.
- Comprender la diferencia entre un archivo de texto con secuencias de órdenes y un *script*.
- Comprender la utilidad de utilizar *scripts*.
- Manejar órdenes simples de *shell scripting*.

1. Intérpretes de órdenes

shells

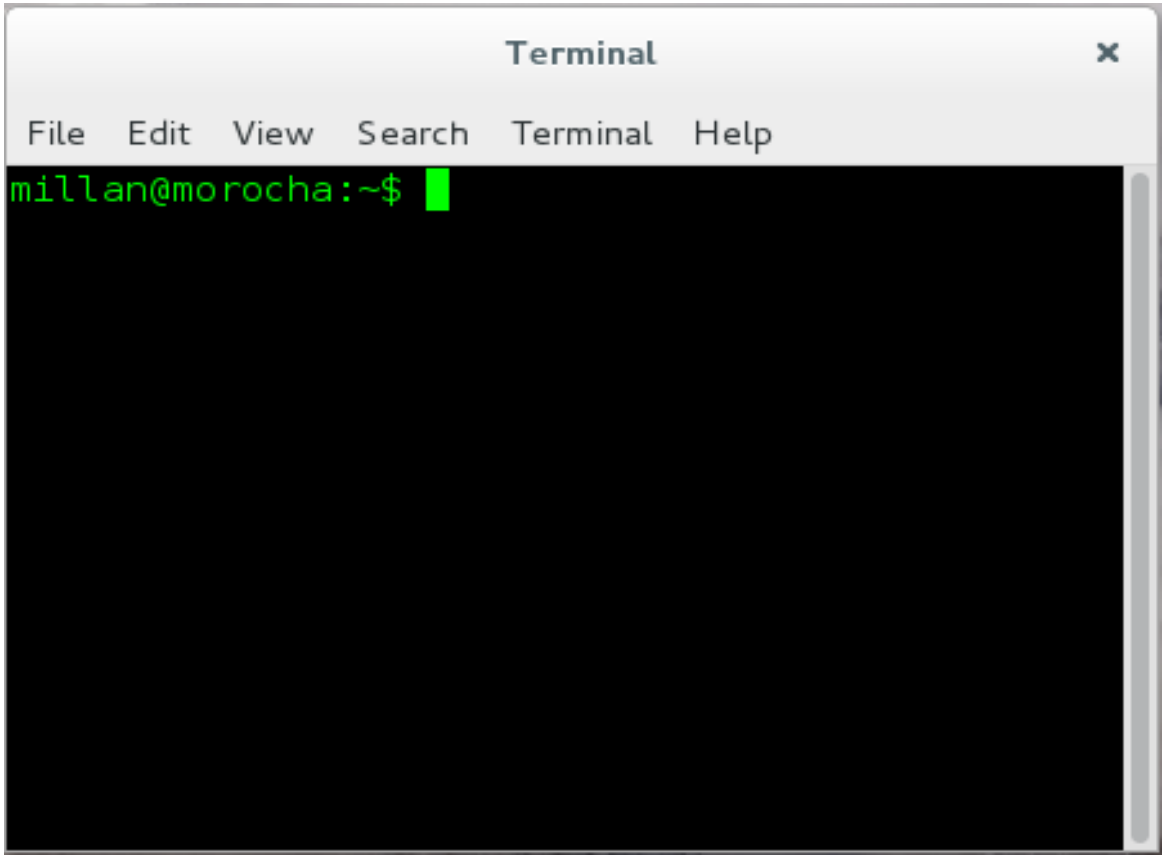
- Una ***shell*** es un programa que lee y ejecuta órdenes del usuario para acceder a servicios del *SO*.
- Las ***shells*** suelen contar con características como:
  - ➢ control de trabajo
  - ➢ redirección de entrada y salida
  - ➢ un lenguaje de órdenes para escribir ***scripts*** de ***shell***
- Hay muchos 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* Unix
- ***sh*** es compatible con ***bash*** (*Bourne again shell*, 1989) y ***ksh*** (KornShell 1980).

1. Intérpretes de comandos

shells

- Discutir:

¿Cual 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 de "guiones de órdenes" suelen ser identificados por el sistema a través del siguiente encabezamiento en el contenido del archivo, conocido como ***shebang*** (*hash-bang* o *sharpbang*): **#!**

  No confundir shebang con "she bangs" que es un tema de Ricky Martin!
- A continuación de estos caracteres 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 como si ejecutara un programa binario.

  #!/bin/bash #!/bin/ksh #!/bin/csh
- Sirve para identificar el formato del fichero a modo de «número mágico».
- Los archivos de *guiones* de órdenes se identifican por la extensión «***.sh***».

2. Archivos de órdenes por guiones *scripts*

- El *shebang* es 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*"

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

- Cuando se ejecute:

```
$ ./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* 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 esa *shell* puede acceder a ellas.
- El comando ***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

3. Variables de la *Shell* y de entorno

- Las variables de entorno forman un conjunto de valores dinámicos que normalmente afectan como funcionan los procesos en una PC.

Ejemplo: la variable de entorno $PAGER es utilizada por el comando *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⏎
- ➢ ¿Que sucede si ejecuta $ export PAGER=less⏎?

- Para averiguar qué variables de entorno son utilizados por un comando en particular, consulte el manual para 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, ...

3. Variables de la *Shell* y de entorno

- 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 el 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]\$\[\e[0m\] '⏎
- 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 en la terminal 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*).

4. Guiones de órdenes sencillos

shell scripts

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

```
#!/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 **número**.

```
$ ./simple.sh
```

5. Guiones de inicio en Unix

- **¿Qué diferencia hay entre bin y sbin?** https://eslinux.com/foro/1542/diferencia-sbin-bin/
  Jerarquía estándar del sistema de archivos (FHS, del inglés Filesystem Hierarchy Standard) de Linux.
- Breve resumen de las definiciones de los diferentes directorios **bin** y **sbin**:
  - **/bin** contiene ejecutables que son necesarios en el modo de un solo usuario para levantar o reparar el sistema.
  - **/sbin** contiene los comandos para bootear el sistema, pero usualmente no son ejecutados por usuarios normales.
  - **/usr/bin** es el directorio principal para los archivos ejecutables. La mayoría de los programas que son ejecutados por usuarios normales los cuales no son necesarios para bootear o reparar el sistema y no son instalados localmente deben ser ubicados en este directorio.
  - **/usr/local/bin** contiene los programas que no son parte del sistema operativo, usualmente programas que se compilaron localmente desde la fuente o se descargaron sus ejecutables y son usualmente utilizados por usuarios normales. ¿Algún programa que se le ocurra?
  - **/usr/local/sbin** similar al directorio /usr/local/bin con la diferencia que son programas para administración del sistema o que ejecuta solo el administrador/super usuario.
- Muchos asocian la **'s'** que las diferencia alegando que refiere a **"system"**, pero según la documentación de ***man***, podemos pensar que tiene que ver con una cuestión de permisos, y asociar esa **'s'** al usario ***root*** o **super-user**.

5. Guiones de inicio en Unix

- Cuando se inicia sesión en una *shell*, esta ejecuta una amplia secuencia de comandos de inicio del sistema.
  - Normalmente llamando:
    /etc/profile en **sh**, **bash** y **ksh**
    /etc/.login en **csh**
- A continuación busque en su *home* el *guión/script* de inicio personal
  - *.profile* en **sh** y **ksh**
  - .bashrc en **bash**
  - *.cshrc* en **csh** y **tcsh**

5. Guiones de inicio en Unix

- Por esta razón su *script* de inicio es un buen lugar para definir variables de entorno como PATH, LD_LIBRARY_PATH, etc, así como **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 *.bashrc* (o *.profile*)

  export PATH=$PATH:~/bin
- Luego de modificar el archivo *.bashrc* se deben importar los cambios a la shell de trabajo actual

  $ source ~/.bashrc⏎

- **Ejemplo:** modifique el ***.bashrc*** de su cuenta *alumnoXY* en el "server mecanica" para que el *prompt* de la terminal ($PS1) se muestre en color rojo e indique nombre de la PC y del usuario.

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 los siguientes comandos/órdenes o asignaciones que pueden ser utiles 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*** editor de texto liviano
  - ***su*** Cambia 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).