

Informática II

La Shell de Linux (1/4)

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Introducción

¿Qué es una Shell?

Es un programa que sirve de interfaz entre el usuario y el SO Linux. Permite introducir comandos y que el SO los ejecute.

Introducción

¿Qué es una Shell?

Es un programa que sirve de interfaz entre el usuario y el SO Linux. Permite introducir comandos y que el SO los ejecute.

Originalmente Linux no disponía de interfaz gráfica

Introducción

¿Qué es una Shell?

Es un programa que sirve de interfaz entre el usuario y el SO Linux. Permite introducir comandos y que el SO los ejecute.

Originalmente Linux no disponía de interfaz gráfica

Programas de Shell

- ▶ Se puede programar rápidamente y de forma simple
- ▶ Disponible en la mayoría de las instalaciones del SO Linux
- ▶ Programas de Shell: *scripts* (interpretados en tiempo de ejecución)

Introducción

¿Qué es una Shell?

Es un programa que sirve de interfaz entre el usuario y el SO Linux. Permite introducir comandos y que el SO los ejecute.

Originalmente Linux no disponía de interfaz gráfica

Programas de Shell

- ▶ Se puede programar rápidamente y de forma simple
- ▶ Disponible en la mayoría de las instalaciones del SO Linux
- ▶ Programas de Shell: *scripts* (interpretados en tiempo de ejecución)

([POSIX.2](#), [IEEE Std 1003.2-1992](#): especificaciones mínimas de una Shell – POSIX: Portable Operating System Interface)

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell
- ▶ Elegir portabilidad sobre eficiencia

(https://en.wikipedia.org/wiki/Unix_philosophy)

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell
- ▶ Elegir portabilidad sobre eficiencia

(https://en.wikipedia.org/wiki/Unix_philosophy)

Algunos ejemplos:

```
$> dmesg | more
```

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell
- ▶ Elegir portabilidad sobre eficiencia

(https://en.wikipedia.org/wiki/Unix_philosophy)

Algunos ejemplos:

```
$> dmesg | more
```

```
$> dmesg | tail
```

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell
- ▶ Elegir portabilidad sobre eficiencia

(https://en.wikipedia.org/wiki/Unix_philosophy)

Algunos ejemplos:

```
$> dmesg | more
```

```
$> dmesg | tail
```

```
$> ls -l / | grep '^d' | wc -l
```

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell
- ▶ Elegir portabilidad sobre eficiencia

(https://en.wikipedia.org/wiki/Unix_philosophy)

Algunos ejemplos:

```
$> dmesg | more
```

```
$> dmesg | tail
```

```
$> ls -l / | grep '^d' | wc -l
```

KISS: Keep It Small and Simple...o...

Introducción – Filosofía Unix

- ▶ Lo pequeño es bello
- ▶ Escribir programas simples que hagan una sola cosa y lo haga bien
- ▶ Escribir programas que manejen flujos de caracteres (interfaz universal)
- ▶ Escribir programas que trabajen en conjunto
- ▶ Todo es un archivo
- ▶ Encadenar programas para realizar tareas complejas utilizando tuberías (pipes) y filtros de la Shell
- ▶ Elegir portabilidad sobre eficiencia

(https://en.wikipedia.org/wiki/Unix_philosophy)

Algunos ejemplos:

```
$> dmesg | more
```

```
$> dmesg | tail
```

```
$> ls -l / | grep '^d' | wc -l
```



KISS: Keep It Small and Simple... o... (Keep It Simple, Stupid! ☺)

Shell prompt

Aparece en la línea de comandos indicando que está a la espera de órdenes



Shell prompt

Aparece en la línea de comandos indicando que está a la espera de órdenes

- ▶ Utilizar la función de auto-completar TAB y doble-TAB
- ▶ Historial de comandos: teclas de flechas ( y ).
Ver comando `history` (archivo `~/.bash_history`)

Shell prompt



Aparece en la línea de comandos indicando que está a la espera de órdenes

- ▶ Utilizar la función de auto-completar TAB y doble-TAB
- ▶ Historial de comandos: teclas de flechas ( y ).
Ver comando `history` (archivo `~/.bash_history`)

Se usarán los símbolos `$>` para indicar el prompt.

Shell prompt

Aparece en la línea de comandos indicando que está a la espera de órdenes



- ▶ Utilizar la función de auto-completar TAB y doble-TAB
- ▶ Historial de comandos: teclas de flechas ( y ).
Ver comando `history` (archivo `~/.bash_history`)

Se usarán los símbolos `$>` para indicar el prompt.

-
- ▶ Otros comandos: `id`, `clear` (atajo `Ctrl+L` en `bash`)

Shell prompt

Aparece en la línea de comandos indicando que está a la espera de órdenes

- ▶ Utilizar la función de auto-completar TAB y doble-TAB
- ▶ Historial de comandos: teclas de flechas ( y ).
Ver comando `history` (archivo `~/.bash_history`)

Se usarán los símbolos `$>` para indicar el prompt.

-
- ▶ Otros comandos: `id`, `clear` (atajo `Ctrl+L` en `bash`)
 - ▶ La mayoría de los comandos son archivo ejecutables localizados en el sistema.
Ver comando `which` (`$> which whoami`)

Ejemplo de script de Shell

Programa de la Shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar comandos en un archivo e invocar como un programa

Ejemplo de script de Shell

Programa de la Shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar comandos en un archivo e invocar como un programa

Archivo `hola1.sh`

```
1 echo "Hola mundo"
```

Ejecutar:

```
$> bash hola1.sh
```


Ejemplo de script de Shell

Programa de la Shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar comandos en un archivo e invocar como un programa

Archivo hola1.sh

```
1 echo "Hola mundo"
```

Archivo hola2.sh

```
1 #!/bin/bash
2 echo "Hola mundo"
3 exit 0
```

Ejecutar:

```
$> bash hola1.sh
```

O bien:

```
$> ./hola1.sh
```

(luego de darle permiso de ejecución)

Ejemplo de script de Shell

Programa de la Shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar comandos en un archivo e invocar como un programa

Archivo hola1.sh

```
1 echo "Hola mundo"
```

Archivo hola2.sh

```
1 #!/bin/bash
2 echo "Hola mundo"
3 exit 0
```

Ejecutar:

```
$> bash hola1.sh
```

O bien:

```
$> ./hola1.sh
```

(luego de darle permiso de ejecución)

- ▶ Los comentarios comienzan con #
- ▶ Línea especial #!/bin/bash
- ▶ El comando `exit` devuelve un valor de salida

Usuarios y grupos

Linux es un SO tipo Unix multiplataforma, multitarea y multiusuario.

Usuarios y grupos

Linux es un SO tipo Unix multiplataforma, multitarea y multiusuario.

Usuarios

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Cada usuario tiene un *nombre de usuario* único y su correspondiente *número de usuario*, UID (user identifier). Comando `whoami`.

(cada usuario tiene una línea en el archivo del sistema `/etc/passwd`)

Usuarios y grupos

Linux es un SO tipo Unix multiplataforma, multitarea y multiusuario.

Usuarios

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Cada usuario tiene un *nombre de usuario* único y su correspondiente *número de usuario*, UID (user identifier). Comando `whoami`.

(cada usuario tiene una línea en el archivo del sistema `/etc/passwd`)

Grupos

- Los usuarios se organizan en grupos, identificados también por un número, GID (group identifier). Comando `groups`.

Usuarios y grupos

Linux es un SO tipo Unix multiplataforma, multitarea y multiusuario.

Usuarios

- ▶ Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Cada usuario tiene un *nombre de usuario* único y su correspondiente *número de usuario*, UID (user identifier). Comando `whoami`.

(cada usuario tiene una línea en el archivo del sistema `/etc/passwd`)

Grupos

- ▶ Los usuarios se organizan en grupos, identificados también por un número, GID (group identifier). Comando `groups`.
- ▶ Se utilizan para asignar privilegios de acceso, p.e. para acceder a dispositivos `tty` hay que estar en el grupo `dialout`.

(cada usuario tiene una línea en el archivo del sistema `/etc/group`)

Usuarios y grupos

Superusuario

- ▶ Tiene privilegios sobre todo el sistema.

Usuarios y grupos

Superusuario

- ▶ Tiene privilegios sobre todo el sistema.
- ▶ Nombre de usuario `root`, `UID = 0`.

Usuarios y grupos

Superusuario

- ▶ Tiene privilegios sobre todo el sistema.
- ▶ Nombre de usuario `root`, `UID = 0`.
- ▶ Utilizado por el administrador del sistema para tareas administrativas.

- ▶ ¿Dónde busca la Shell los binarios de comandos?

► ¿Dónde busca la Shell los binarios de comandos?

► Ver variable de entorno PATH

```
$> echo $PATH
```

```
$> echo $PATH | tr ":" "\n"
```

- ▶ ¿Dónde busca la Shell los binarios de comandos?
- ▶ Ver variable de entorno PATH

```
$> echo $PATH
$> echo $PATH | tr ":" "\n"
```
- ▶ ¿Qué es una variable de entorno?

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ `HOME`: directorio home del usuario actual

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ **HOME**: directorio home del usuario actual
- ▶ **PATH**: lista de directorios separados por `:` para buscar los comandos

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ `HOME`: directorio home del usuario actual
- ▶ `PATH`: lista de directorios separados por `:` para buscar los comandos
- ▶ `SHELL`: Shell del usuario

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ `HOME`: directorio home del usuario actual
- ▶ `PATH`: lista de directorios separados por `:` para buscar los comandos
- ▶ `SHELL`: Shell del usuario
- ▶ `USER`: nombre del usuario actual (logueado)

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ `HOME`: directorio home del usuario actual
- ▶ `PATH`: lista de directorios separados por `:` para buscar los comandos
- ▶ `SHELL`: Shell del usuario
- ▶ `USER`: nombre del usuario actual (logueado)
- ▶ `EDITOR`: editor de preferencia del usuario

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ `HOME`: directorio home del usuario actual
- ▶ `PATH`: lista de directorios separados por `:` para buscar los comandos
- ▶ `SHELL`: Shell del usuario
- ▶ `USER`: nombre del usuario actual (logueado)
- ▶ `EDITOR`: editor de preferencia del usuario
- ▶ `?`: valor devuelto por el último comando ejecutado

Variables de entorno

- ▶ Variables inicializadas desde el entorno (Shell)
- ▶ Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- ▶ Son de gran utilidad en la programación de script de la Shell
- ▶ Imprimir valor de variable de entorno: `echo $HOME`

Algunas variables de entorno:

- ▶ `HOME`: directorio home del usuario actual
- ▶ `PATH`: lista de directorios separados por `:` para buscar los comandos
- ▶ `SHELL`: Shell del usuario
- ▶ `USER`: nombre del usuario actual (logueado)
- ▶ `EDITOR`: editor de preferencia del usuario
- ▶ `?`: valor devuelto por el último comando ejecutado

(\$> `man 7 environ`)

Más comandos ☺

- Comandos `printenv`, `grep`
`$> printenv | grep PATH`

Más comandos ☺

- ▶ Comandos `printenv`, `grep`

```
$> printenv | grep PATH
```

- ▶ Comandos `cat`, `head`, `tail`

```
$> cat num.txt | head
```

```
$> cat num.txt | tail
```

```
$> cat num.txt | grep 10
```

Más comandos ☺

- ▶ Comandos `printenv`, `grep`
`$> printenv | grep PATH`
- ▶ Comandos `cat`, `head`, `tail`
`$> cat num.txt | head`
`$> cat num.txt | tail`
`$> cat num.txt | grep 10`
- ▶ Opciones de los comandos:
 - ▶ letra seguido de '-'
 - ▶ o bien '--' (`--help`, `--version`)

Más comandos ☺

- ▶ Comandos `printenv`, `grep`
`$> printenv | grep PATH`
- ▶ Comandos `cat`, `head`, `tail`
`$> cat num.txt | head`
`$> cat num.txt | tail`
`$> cat num.txt | grep 10`
- ▶ Opciones de los comandos:
 - ▶ letra seguido de '-'
 - ▶ o bien '--' (`--help`, `--version`)
- ▶ Comandos `whatis`, `apropos`, `man`

Más comandos ☺

- ▶ Comandos `printenv`, `grep`
`$> printenv | grep PATH`
- ▶ Comandos `cat`, `head`, `tail`
`$> cat num.txt | head`
`$> cat num.txt | tail`
`$> cat num.txt | grep 10`
- ▶ Opciones de los comandos:
 - ▶ letra seguido de '-'
 - ▶ o bien '--' (`--help`, `--version`)
- ▶ Comandos `whatis`, `apropos`, `man`
- ▶ **Manpages**: manual en línea (RTFM).
`$> man echo`
`$> man whatis`
`$> man man`

Páginas de manuales (manpages)

Cuenta con diferentes secciones: `$> man man`

Páginas de manuales (manpages)

Cuenta con diferentes secciones: `$> man man`

Algunas son:

1. Executable programs or Shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in `/dev`)
5. File formats and conventions eg. `/etc/passwd`
6. Games
7. Miscellaneous (including macro packages and conventions)
8. System administration commands (usually only for root)

Páginas de manuales (manpages)

Cuenta con diferentes secciones: `$> man man`

Algunas son:

1. Executable programs or Shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in `/dev`)
5. File formats and conventions eg. `/etc/passwd`
6. Games
7. Miscellaneous (including macro packages and conventions)
8. System administration commands (usually only for root)

Ejemplos:

```
$> man 1 printf
```

```
$> man 3 printf
```