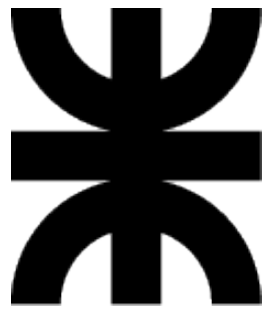
Informática II
La shell de Linux

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2021 –

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 indica las especificaciones mínimas de una shell[1]:

[1] POSIX: Portable Operating System Interface, familia de estándares del IEEE-CS

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 indica las especificaciones mínimas de una shell[1]:

- Intérprete de comandos
- Programas de utilidad

[1] POSIX: Portable Operating System Interface, familia de estándares del IEEE-CS

Introducción

Filosofía Unix

- Todo es un archivo.

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.
- Guardar los datos y configuración en archivos de texto plano. Los archivos de texto son una interfaz universal, son fáciles de crear, guardar y mover a otro sistema.

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.
- Guardar los datos y configuración en archivos de texto plano. Los archivos de texto son una interfaz universal, son fáciles de crear, guardar y mover a otro sistema.
- Encadenar programas para realizar tareas complejas. Utilizar las tuberías (pipes) y filtros de la shell para encadenar pequeñas utilidades que realizan una tarea a la vez.

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.
- Guardar los datos y configuración en archivos de texto plano. Los archivos de texto son una interfaz universal, son fáciles de crear, guardar y mover a otro sistema.
- Encadenar programas para realizar tareas complejas. Utilizar las tuberías (pipes) y filtros de la shell para encadenar pequeñas utilidades que realizan una tarea a la vez.
- Elegir portabilidad sobre eficiencia.

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.
- Guardar los datos y configuración en archivos de texto plano. Los archivos de texto son una interfaz universal, son fáciles de crear, guardar y mover a otro sistema.
- Encadenar programas para realizar tareas complejas. Utilizar las tuberías (pipes) y filtros de la shell para encadenar pequeñas utilidades que realizan una tarea a la vez.
- Elegir portabilidad sobre eficiencia.

Por ejemplo:

```
> ls -la | more
```

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.
- Guardar los datos y configuración en archivos de texto plano. Los archivos de texto son una interfaz universal, son fáciles de crear, guardar y mover a otro sistema.
- Encadenar programas para realizar tareas complejas. Utilizar las tuberías (pipes) y filtros de la shell para encadenar pequeñas utilidades que realizan una tarea a la vez.
- Elegir portabilidad sobre eficiencia.

Por ejemplo:

```
> ls -la | more
```

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

Introducción

Filosofía Unix

- Todo es un archivo.
- Lo pequeño es bello.
- Escribir programas simples que hagan una sola cosa y lo haga bien. Escribir programas que trabajen en conjunto. Escribir programas que manejen flujos de caractéres, dado que es una interfaz universal.
- Guardar los datos y configuración en archivos de texto plano. Los archivos de texto son una interfaz universal, son fáciles de crear, guardar y mover a otro sistema.
- Encadenar programas para realizar tareas complejas. Utilizar las tuberías (pipes) y filtros de la shell para encadenar pequeñas utilidades que realizan una tarea a la vez.
- Elegir portabilidad sobre eficiencia.

Por ejemplo:

```
> ls -la | more
```

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

Introducción - Ejemplo de script de shell

Programa de la shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar los comandos en un archivo que luego se invoca como un programa

Introducción - Ejemplo de script de shell

Programa de la shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar los comandos en un archivo que luego se invoca como un programa

Archivo `hola1.sh`

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

Introducción - Ejemplo de script de shell

Programa de la shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar los comandos en un archivo que luego se invoca como un programa

Archivo `hola1.sh`

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

Ejecutar

```
> bash hola1.sh
```

Introducción - Ejemplo de script de shell

Programa de la shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar los comandos en un archivo que luego se invoca como un programa

Archivo `hola1.sh`

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

Ejecutar

```
> bash hola1.sh
```

Archivo `hola2.sh`

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

Introducción - Ejemplo de script de shell

Programa de la shell

1. Secuencia de comandos ejecutados de forma interactiva
2. Guardar los comandos en un archivo que luego se invoca como un programa

Archivo `hola1.sh`

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

Ejecutar

```
> bash hola1.sh
```

Archivo `hola2.sh`

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

- Puede ejecutarse luego de darle permisos de ejecución (`> ./hola2.sh`)

Introducción - Ejemplo de script de shell

Archivo `files.sh`

```
1 #!/bin/bash
2
3 for file in *; do
4   echo -n "Nombre de archivo: "
5   echo ${file}
6 done
7
8 exit 0
```

Introducción - Ejemplo de script de shell

Archivo `files.sh`

```
1 #!/bin/bash
2
3 for file in *; do
4   echo -n "Nombre de archivo: "
5   echo ${file}
6 done
7
8 exit 0
```

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

Usuarios y grupos

Algunos comandos:

- `whoami`, `groups`, `id`, `clear` (atajo `Ctrl+L` en `bash`)

Usuarios y grupos

Algunos comandos:

- `whoami`, `groups`, `id`, `clear` (atajo `Ctrl+L` en `bash`)
- Archivo binarios o ejecutables localizados en el sistema. `which` Ej. `which whoami`.

Usuarios y grupos

Algunos comandos:

- `whoami`, `groups`, `id`, `clear` (atajo `Ctrl+L` en `bash`)
- Archivo binarios o ejecutables localizados en el sistema. `which` Ej. `which whoami`.
- ¿Dónde busca la shell los binarios de comandos?

Usuarios y grupos

Algunos comandos:

- `whoami`, `groups`, `id`, `clear` (atajo `Ctrl+L` en `bash`)
- Archivo binarios o ejecutables localizados en el sistema. `which`
  Ej. `which whoami`.
- ¿Dónde busca la shell los binarios de comandos?
  Ver variable de entorno `PATH`

Usuarios y grupos

Algunos comandos:

- `whoami`, `groups`, `id`, `clear` (atajo `Ctrl+L` en `bash`)
- Archivo binarios o ejecutables localizados en el sistema. `which`
  Ej. `which whoami`.
- ¿Dónde busca la shell los binarios de comandos?
  Ver variable de entorno `PATH`

Utilizar la función de auto-completar `TAB` y `doble-TAB`

Usuarios y grupos

Algunos comandos:

- `whoami`, `groups`, `id`, `clear` (atajo `Ctrl+L` en `bash`)
- Archivo binarios o ejecutables localizados en el sistema. `which`
  Ej. `which whoami`.
- ¿Dónde busca la shell los binarios de comandos?
  Ver variable de entorno `PATH`

Utilizar la función de auto-completar `TAB` y `doble-TAB`

Shell prompt: aparece en la línea de comandos indicando que está a la espera de órdenes

```
> echo $PATH
> bash --version
```

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.

- 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).

Usuarios y grupos

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

- 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).
- Los usuarios se organizan en grupos, identificados también por un números, GID.

Usuarios y grupos

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

- 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).
- Los usuarios se organizan en grupos, identificados también por un números, GID.
  - Ej.: para acceder a dispositivos `tty` hay que estar en el grupo `dialout`

Usuarios y grupos

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

- 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).
- Los usuarios se organizan en grupos, identificados también por un números, GID.
  - Ej.: para acceder a dispositivos `tty` hay que estar en el grupo `dialout`
  - Algunos grupos: `adm`, `sudo`, `plugdev`, `docker`

Usuarios y grupos

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

- 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).
- Los usuarios se organizan en grupos, identificados también por un números, GID.
  - Ej.: para acceder a dispositivos `tty` hay que estar en el grupo `dialout`
  - Algunos grupos: `adm`, `sudo`, `plugdev`, `docker`

Superusuario

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

Usuarios y grupos

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

- 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).
- Los usuarios se organizan en grupos, identificados también por un números, GID.
  - Ej.: para acceder a dispositivos `tty` hay que estar en el grupo `dialout`
  - Algunos grupos: `adm`, `sudo`, `plugdev`, `docker`

Superusuario

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

Usuarios y grupos

Por cada usuario existente se tiene una línea en el archivo del sistema `/etc/passwd`

```
> cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
gfpp:x:1000:1000:Gonzalo Perez Paina,,,:/home/gfpp:/bin/bash
```

Usuarios y grupos

Por cada usuario existente se tiene una línea en el archivo del sistema `/etc/passwd`

```
> cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
gfpp:x:1000:1000:Gonzalo Perez Paina,,,:/home/gfpp:/bin/bash
```

Contiene la siguiente información:

- *ID del grupo*: identificador numérico de group (ID) del primero de los grupos a los cuales el usuario es miembro
- *Directorio home*: el directorio donde se encuentra el usuario luego del login
- *Shell*: nombre del programa que se ejecuta como interprete de comandos de usuarios

Usuarios y grupos

Por cada usuario existente se tiene una línea en el archivo del sistema `/etc/passwd`

```
> cat /etc/group
root:x:0:
adm:x:4:syslog,gfpp
gfpp:x:1000:
```

Usuarios y grupos

Por cada usuario existente se tiene una línea en el archivo del sistema `/etc/passwd`

```
> cat /etc/group
root:x:0:
adm:x:4:syslog,gfpp
gfpp:x:1000:
```

Contiene la siguiente información:

- *Nombre del grupo*: nombre (único) del grupo
- *ID del grupo*: ID numérico asociado a ese grupo
- *Lista de usuarios*: lista separada por coma de nombres de usuarios que son miembros de este grupo

Variables de entorno

- Variables inicializadas desde el entorno (shell)

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell

Variables de entorno

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

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `SHELL`, `?`

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `SHELL`, `?`
- Imprimir valor de variable de entorno: `echo $HOME`

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `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)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `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)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `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
- `$PS1`: prompt de comandos

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `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
- `$PS1`: prompt de comandos
- `$PS2`: prompt secundario

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `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
- `$PS1`: prompt de comandos
- `$PS2`: prompt secundario
- `$0`: nombre del script de shell

Variables de entorno

- Variables inicializadas desde el entorno (shell)
- Son de gran utilidad en la programación de script de la shell
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `HOME`, `PATH`, `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
- `$PS1`: prompt de comandos
- `$PS2`: prompt secundario
- `$0`: nombre del script de shell
- `$#`: cantidad de parámetros pasados

Más comandos ☺

- Comandos `echo`, `printenv`, `man`, `apropos`, `cat`, `grep`, `head`, `tail`

Más comandos ☺

- Comandos `echo`, `printenv`, `man`, `apropos`, `cat`, `grep`, `head`, `tail`
- Opciones de los comandos, letra seguido de '`-`'. O bien '`--`' (`--help`, `--version`)

Más comandos ☺

- Comandos `echo`, `printenv`, `man`, `apropos`, `cat`, `grep`, `head`, `tail`
- Opciones de los comandos, letra seguido de '`-`'. O bien '`--`' (`--help`, `--version`)
- Manpages: manual en línea (RTFM).

```
> man printenv
> man 1 printf
> man 3 printf
> 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`

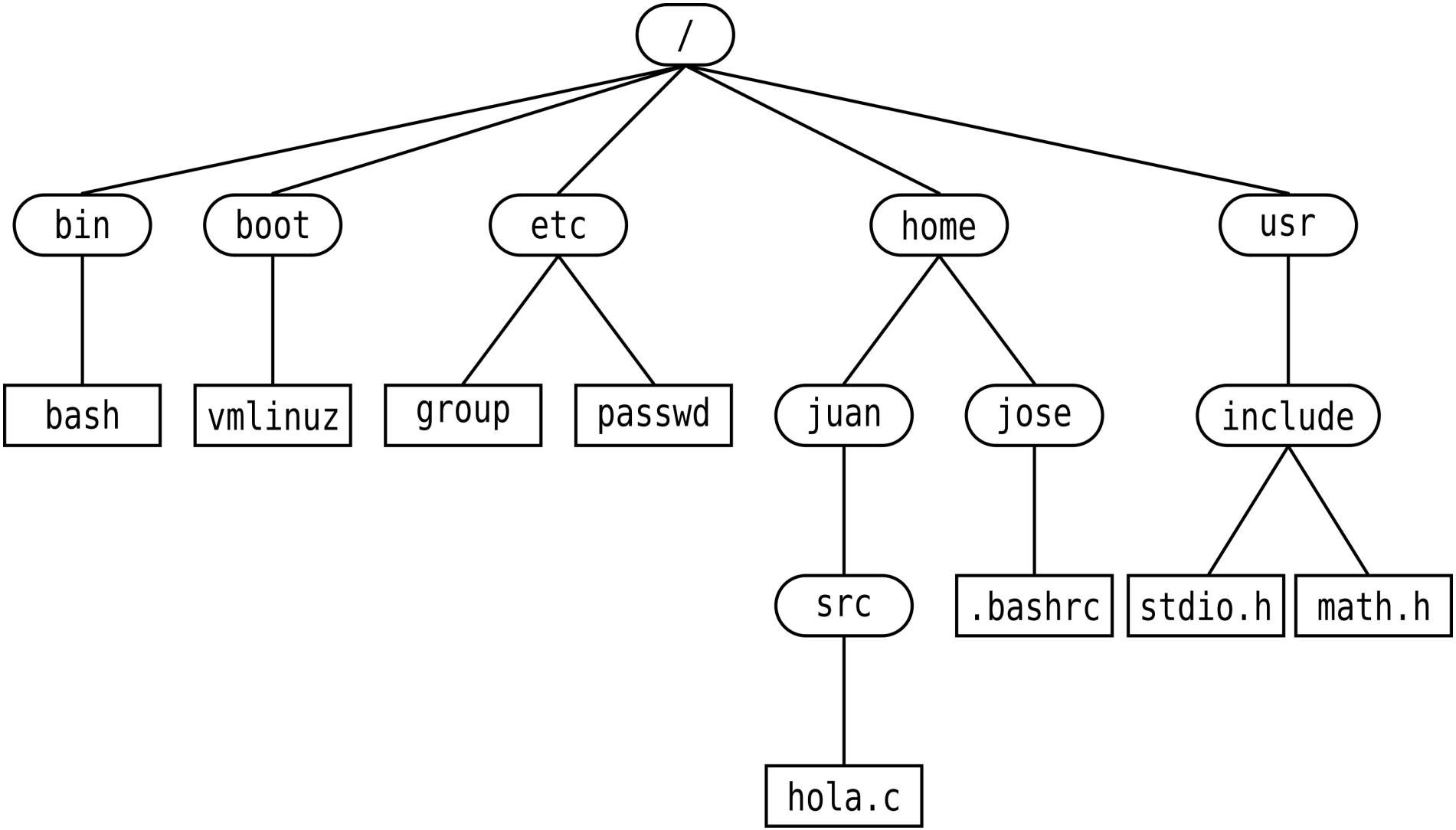
Algunas son:

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
> man -a printf
> man -k '^printf'
```

Sistema de archivos y permisos



Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos “tipo directorio”. Comando `ls`

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos “tipo directorio”. Comando `ls`

- Directorios (raíz/root)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos “tipo directorio”. Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos “tipo directorio”. Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: ’.’ (punto)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos "tipo directorio". Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: *'.'* (punto)
- Directorio anterior o padre: *'..'* (doble punto)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos “tipo directorio”. Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: *’.’* (punto)
- Directorio anterior o padre: *’..’* (doble punto)
- Camino absoluto (comienza en `/`)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos "tipo directorio". Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: *'.'* (punto)
- Directorio anterior o padre: *'..'* (doble punto)
- Camino absoluto (comienza en `/`)
- Camino relativo (comienza en `./` o `../`)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos "tipo directorio". Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: *'.'* (punto)
- Directorio anterior o padre: *'..'* (doble punto)
- Camino absoluto (comienza en `/`)
- Camino relativo (comienza en `./` o `../`)

Ver archivos ocultos: `ls -a`

- Directorio `/home`. Variable de entorno `HOME` (`ls $HOME`). (`cd`, `pwd`)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos "tipo directorio". Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: *'.'* (punto)
- Directorio anterior o padre: *'..'* (doble punto)
- Camino absoluto (comienza en `/`)
- Camino relativo (comienza en `./` o `../`)

Ver archivos ocultos: `ls -a`

- Directorio `/home`. Variable de entorno `HOME` (`ls $HOME`). (`cd`, `pwd`)
- Atributos de archivos (`ls -l /`)

Sistema de archivos y permisos

Estructura de archivos en árbol. Archivos "tipo directorio". Comando `ls`

- Directorios (raíz/root)
- Camino/ruta/path
- Directorio actual: *'.'* (punto)
- Directorio anterior o padre: *'..'* (doble punto)
- Camino absoluto (comienza en `/`)
- Camino relativo (comienza en `./` o `../`)

Ver archivos ocultos: `ls -a`

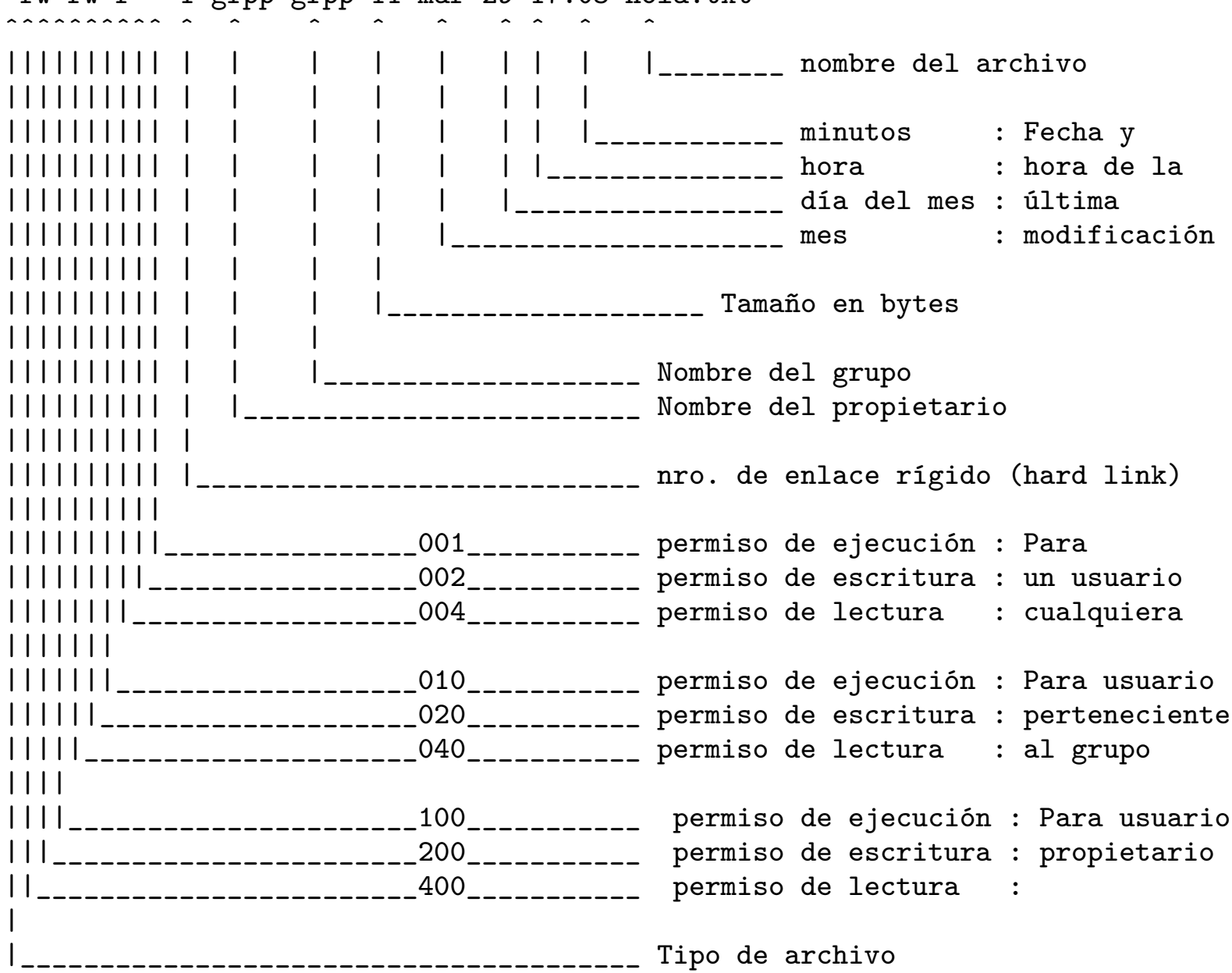
- Directorio `/home`. Variable de entorno `HOME` (`ls $HOME`). (`cd`, `pwd`)
- Atributos de archivos (`ls -l /`)

Comandos

`mkdir`, `rmdir`, `cp`
`mv`, `rm`, `touch`. (`which cd`)

Hacer:

```
> cd
> touch hola.txt
> ls -l hola.txt
```


-rw-rw-r-- 1 gfpp gfpp 11 mar 29 17:03 hola.txt
nombre del archivo
minutos : Fecha y
hora : hora de la
día del mes : última
mes : modificación
Tamaño en bytes
Nombre del grupo
Nombre del propietario
nro. de enlace rígido (hard link)
001 permiso de ejecución : Para
002 permiso de escritura : un usuario
004 permiso de lectura : cualquiera
010 permiso de ejecución : Para usuario
020 permiso de escritura : perteneciente
040 permiso de lectura : al grupo
100 permiso de ejecución : Para usuario
200 permiso de escritura : propietario
400 permiso de lectura :
Tipo de archivo

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (-)

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (`-`)
2. Archivos directorios (`d`)

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (`-`)
2. Archivos directorios (`d`)
3. Archivos especiales

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (`-`)
2. Archivos directorios (`d`)
3. Archivos especiales
   - Archivos de bloque (`b`)

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (`-`)
2. Archivos directorios (`d`)
3. Archivos especiales
   - Archivos de bloque (`b`)
   - Archivos de caracteres (`c`)

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (`-`)
2. Archivos directorios (`d`)
3. Archivos especiales
   - Archivos de bloque (`b`)
   - Archivos de caracteres (`c`)
   - Enlaces simbólicos (`l`)

Sistema de archivos y permisos

Tipos de archivos en Linux

1. Archivos regulares (`-`)
2. Archivos directorios (`d`)
3. Archivos especiales
   - Archivos de bloque (`b`)
   - Archivos de caracteres (`c`)
   - Enlaces simbólicos (`l`)

Probar:

```
> ls -l |  grep ^-
> ls -l |  grep ^d
> cd /dev
> ls -l |  grep ^b
> ls -l |  grep ^c
> ls -l |  grep ^l
```

Sistema de archivos y permisos

Alterar permisos de archivos – comando `chmod`

- `> chmod u-r hola.txt`.

Sistema de archivos y permisos

Alterar permisos de archivos – comando `chmod`

- `> chmod u-r hola.txt`.
- `> cat hola.txt`

Sistema de archivos y permisos

Alterar permisos de archivos – comando `chmod`

- `> chmod u-r hola.txt.`
- `> cat hola.txt`
- `> chmod -x a.out`

Sistema de archivos y permisos

Alterar permisos de archivos – comando `chmod`

- `> chmod u-r hola.txt.`
- `> cat hola.txt`
- `> chmod -x a.out`
- `> chmod o+x a.out`

Sistema de archivos y permisos

Alterar permisos de archivos – comando `chmod`

- `> chmod u-r hola.txt.`
- `> cat hola.txt`
- `> chmod -x a.out`
- `> chmod o+x a.out`

Notación numérica de los permisos

```
r w x | - w x | r - x          Equivale a un permiso 735
4 2 1 | 0 2 1 | 4 0 1          4+2+1, 0+2+1, 4+0+1 = 7,3,5
```

Más comandos ☺

- `find`, `cal`
- `uname` (`-n`, `-v`, `-r`, `-m`, `-a`)
- `lshw`, `lsusb`, `lspci` (`-tv`)
- `uptime`
- `df -h`
- `dmesg`

(Consultar las páginas de manuales para ver la función de cada comando)

Más comandos ☺

- `find`, `cal`
- `uname` (`-n`, `-v`, `-r`, `-m`, `-a`)
- `lshw`, `lsusb`, `lspci` (`-tv`)
- `uptime`
- `df -h`
- `dmesg`

(Consultar las páginas de manuales para ver la función de cada comando)

Ejemplos:

```
> find / -name ‘uname -r‘
> find / -name ‘uname -r‘ 2> /dev/null
```

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de salida y error (`stdout` y `stderr`)

- `ls -ld /tmp /tnt`

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de salida y error (`stdout` y `stderr`)

- `ls -ld /tmp /tnt`
- `ls -ld /tmp /tnt >/dev/null`

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de salida y error (`stdout` y `stderr`)

- `ls -ld /tmp /tnt`
- `ls -ld /tmp /tnt >/dev/null`
- `ls -ld /tmp /tnt 2>/dev/null`

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de salida y error (`stdout` y `stderr`)

- `ls -ld /tmp /tnt`
- `ls -ld /tmp /tnt >/dev/null`
- `ls -ld /tmp /tnt 2>/dev/null`

Descriptor de archivo 0: entrada estándar, 1: salida estándar, 2: salida de error estándar

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de salida y error (`stdout` y `stderr`)

- `ls -ld /tmp /tnt`
- `ls -ld /tmp /tnt >/dev/null`
- `ls -ld /tmp /tnt 2>/dev/null`

Descriptor de archivo 0: entrada estándar, 1: salida estándar, 2: salida de error estándar

Redirección de entrada (`stdin`) [`> ls -lR /usr/include > lsoutput.txt`]

- `> more < lsoutput.txt`

Redirección de entrada, salida y error

Redirección de salida (`stdout`)

- `> ls -l > lsoutput.txt`
  Redirecciona la salida del comando `ls` al archivo `lsoutput.txt`
- `> ps >> lsoutput.txt`
  Agrega la salida del comando `ps` al archivo

Redirección de salida y error (`stdout` y `stderr`)

- `ls -ld /tmp /tnt`
- `ls -ld /tmp /tnt >/dev/null`
- `ls -ld /tmp /tnt 2>/dev/null`

Descriptor de archivo 0: entrada estándar, 1: salida estándar, 2: salida de error estándar

Redirección de entrada (`stdin`) [`> ls -lR /usr/include > lsoutput.txt`]

- `> more < lsoutput.txt`

Ver ejemplo de código fuente con `stdout` y `stderr` (`sqrt.c`)

Otro uso de la redirección de entrada salida

Ver forma de participar de `aceptaelreto.com`

Otro uso de la redirección de entrada salida

Ver forma de participar de `aceptaelreto.com`

```
> ./a.out < entrada > salida
> diff salida salida.ok
```

Otro uso de la redirección de entrada salida

Ver forma de participar de `aceptaelreto.com`

```
> ./a.out < entrada > salida
> diff salida salida.ok
```

Opciones de `diff`:

- `-s`: informa si los archivos son iguales
- `-q`: informa si los archivos son diferentes
- `-c`: formato de salida de contexto
- `-u`: formato de salida unificado
- `-i`: ignora las diferencias entre mayúsculas y minúsculas

(`> man diff`)

Pipes

Se puede conectar procesos utilizando el operador pipe, `|`.
Ejemplo: utilizar `sort` para ordenar de forma inversa la salida de `ls`

```
> ls -1 | sort -r
```

(Ver el manual de `ls` y `sort`)

Pipes

Se puede conectar procesos utilizando el operador pipe, `|`.
Ejemplo: utilizar `sort` para ordenar de forma inversa la salida de `ls`
`> ls -1 | sort -r`
(Ver el manual de `ls` y `sort`)

Si no se utiliza pipes, se necesitan varios pasos

1. `> ls -1 > ls.txt`
2. `> sort -r ls.txt > lsrev.txt`

Pipes

Se puede conectar procesos utilizando el operador pipe, `|`.
Ejemplo: utilizar `sort` para ordenar de forma inversa la salida de `ls`
`> ls -1 | sort -r`
(Ver el manual de `ls` y `sort`)

Si no se utiliza pipes, se necesitan varios pasos

1. `> ls -1 > ls.txt`
2. `> sort -r ls.txt > lsrev.txt`

Conectando los procesos mediante pipes

- `> ls -1 | sort -r > lsrev.txt`

Pipes

Se puede conectar procesos utilizando el operador pipe, `|`.
Ejemplo: utilizar `sort` para ordenar de forma inversa la salida de `ls`
`> ls -1 | sort -r`
(Ver el manual de `ls` y `sort`)

Si no se utiliza pipes, se necesitan varios pasos

1. `> ls -1 > ls.txt`
2. `> sort -r ls.txt > lsrev.txt`

Conectando los procesos mediante pipes

- `> ls -1 | sort -r > lsrev.txt`

Otros ejemplos

- `> ps aux | sort`
- `> ps aux | sort | more`

Procesos

Un proceso es una instancia de un programa en ejecución

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el kernel:

- carga el código del programa en memoria virtual,

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el kernel:

- carga el código del programa en memoria virtual,
- reserva espacio para las variables del programa, e

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el kernel:

- carga el código del programa en memoria virtual,
- reserva espacio para las variables del programa, e
- inicializa estructuras de datos propias del kernel para guardar información del proceso (PID, estado, IDs de usuario y grupo, etc.).

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el kernel:

- carga el código del programa en memoria virtual,
- reserva espacio para las variables del programa, e
- inicializa estructuras de datos propias del kernel para guardar información del proceso (PID, estado, IDs de usuario y grupo, etc.).

El estándar UNIX (IEEE Std 1003.1-2004) define un proceso como:

> *"an address space with one or more threads executing within that address space, and the required system resources for those threads"*

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

heap : área desde la cual un programa puede asignar memoria adicional de forma dinámica

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

heap : área desde la cual un programa puede asignar memoria adicional de forma dinámica

stack : porción de memoria que crece o se encoje a medida que las funciones se llaman y retornan; utilizada para almacenar las variables locales e información de enlace de las funciones

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

heap : área desde la cual un programa puede asignar memoria adicional de forma dinámica

stack : porción de memoria que crece o se encoje a medida que las funciones se llaman y retornan; utilizada para almacenar las variables locales e información de enlace de las funciones

- Cada proceso tiene asociado un identificador único, PID (entero positivo entre 2 y 32768)
- Cada proceso tiene asociado un identificador del proceso padre, PPID
- El nro. de proceso 1 está reservado para el proceso `init`

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

heap : área desde la cual un programa puede asignar memoria adicional de forma dinámica

stack : porción de memoria que crece o se encoje a medida que las funciones se llaman y retornan; utilizada para almacenar las variables locales e información de enlace de las funciones

- Cada proceso tiene asociado un identificador único, PID (entero positivo entre 2 y 32768)
- Cada proceso tiene asociado un identificador del proceso padre, PPID
- El nro. de proceso 1 está reservado para el proceso `init`

Ver comandos: `ps`, `top`, `htop`, `pstree`, `ps aux` (`man ps`)

Procesos

Atributos de los procesos

- `PID`: Valor numérico que identifica al proceso
- `TTY`: Terminal asociada al proceso
- `STAT`: Estado del proceso
- `TIME`: Tiempo de CPU consumido por el proceso
- `COMMAND`: Comandos y argumentos utilizados

Procesos

Atributos de los procesos

- `PID`: Valor numérico que identifica al proceso
- `TTY`: Terminal asociada al proceso
- `STAT`: Estado del proceso
- `TIME`: Tiempo de CPU consumido por el proceso
- `COMMAND`: Comandos y argumentos utilizados

Algunos estados posible:

- `S`: Sleeping. Esperando un evento (señal o entrada)
- `R`: Running. En la cola para ser ejecutado
- `T`: Stopped, por la shell o debbuger
- `N`: Tarea de baja prioridad
- `Z`: Proceso zombie
- `<`: Tarea de alta prioridad

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Combinación de teclas:

- `Ctrl+C`: interrumpe la ejecución del proceso
- `Ctrl+Z`: suspende la ejecución del proceso

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Combinación de teclas:

- `Ctrl+C`: interrumpe la ejecución del proceso
- `Ctrl+Z`: suspende la ejecución del proceso

Probar lanzar procesos en segundo plano (gedit, gnome-calculator, evince, etc.). Ver comando `jobs`.

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Combinación de teclas:

- `Ctrl+C`: interrumpe la ejecución del proceso
- `Ctrl+Z`: suspende la ejecución del proceso

Probar lanzar procesos en segundo plano (gedit, gnome-calculator, evince, etc.). Ver comando `jobs`.

Probar: Ejecutar el comando `yes` y detener con `Ctrl+C`. Luego ejecutar con redirección (`yes >/dev/null`) y:

- Suspender, `Ctrl+Z`
- Enviar a 2do plano, `bg`; y a 1er plano, `fg`
- Detener, `Ctrl+C`

Procesos demonio – daemon

Un demonio (daemon) es un proceso de propósitos especiales creado para tareas de administración del sistema. Se distingue por las siguientes características:

- Arranca cuando se inicia el sistema (buteo) y permanece en existencia hasta que se apaga
- Corre en segundo plano (background) y no tiene terminal de control desde la cual se puede leer entrada o sobre la cual mostrar la salida

Procesos demonio – daemon

Un demonio (daemon) es un proceso de propósitos especiales creado para tareas de administración del sistema. Se distingue por las siguientes características:

- Arranca cuando se inicia el sistema (buteo) y permanece en existencia hasta que se apaga
- Corre en segundo plano (background) y no tiene terminal de control desde la cual se puede leer entrada o sobre la cual mostrar la salida

Algunos ejemplos de procesos *"demonios"* son:

- `syslogd`: guarda logs (registro) del sistema
- `httpd`: servidor de páginas web
- `sshd`: servidor de SSH (Secure Shell)
- `cupsd`: sistema de impresión de UNIX
- Etc.

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones
- Ejemplo de señal: *caracter de interrupción* Ctrl-C

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones
- Ejemplo de señal: *caracter de interrupción* Ctrl-C

Ver comando `kill` (`man 7 signal`)

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones
- Ejemplo de señal: *caracter de interrupción* Ctrl-C

Ver comando `kill` (`man 7 signal`)

Terminar/matar un proceso:

1. Identificar el proceso utilizando el comando `ps`
2. Enviarle una señal con el comando `kill`

Señales en lenguaje C

```c
#include <stdio.h>
#include <unistd.h>

int main(void)
{
  printf("Iniciando bucle...\n");
  while(1)
  {
    printf("Corriendo.\n");
    sleep(1);
  }
  printf("Terminando bucle...\n");

  return 0;
}
```

Señales en lenguaje C

```c
#include <stdio.h>
#include <unistd.h>


unsigned int flag = 1;
void handler(int );

int main(void) {

  printf("Iniciando bucle...\n");
  while(flag)
  {
    printf("Corriendo.\n");
    sleep(1);
  }
  printf("Terminando bucle...\n");

  return 0;
}

void handler(int sig)
{ flag = 0; }
```

Señales en lenguaje C

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

unsigned int flag = 1;
void handler(int );

int main(void) {
  signal(SIGINT, handler);
  printf("Iniciando bucle...\n");
  while(flag)
  {
    printf("Corriendo.\n");
    sleep(1);
  }
  printf("Terminando bucle...\n");

  return 0;
}

void handler(int sig) // manejador de la señal
{ flag = 0; }
```