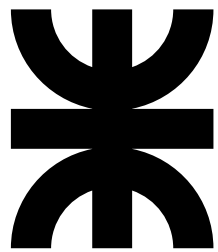
Programación en Linux embebido

La shell de Linux

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2017 –

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

¿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.

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

¿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.

Programas de shell

- Se puede programar rápidamente y de forma simple
- Disponible en la mayoría de las instalaciones del SO Linux
- Los programas de la shell se llaman *scripts* (interpretados en tiempo de ejecución)

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

¿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.

Programas de shell

- Se puede programar rápidamente y de forma simple
- Disponible en la mayoría de las instalaciones del SO Linux
- Los programas de la shell se llaman *scripts* (interpretados en tiempo de ejecución)

POSIX 1003.2 indica las especificaciones mínimas de una shell

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

¿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.

Programas de shell

- Se puede programar rápidamente y de forma simple
- Disponible en la mayoría de las instalaciones del SO Linux
- Los programas de la shell se llaman *scripts* (interpretados en tiempo de ejecución)

POSIX 1003.2 indica las especificaciones mínimas de una shell

Filosofía Linux

Utilidades simples y pequeñas que se utilizan como un eslabón de una cadena de comandos (`$ ls -la | more`)

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

¿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.

Programas de shell

- Se puede programar rápidamente y de forma simple
- Disponible en la mayoría de las instalaciones del SO Linux
- Los programas de la shell se llaman *scripts* (interpretados en tiempo de ejecución)

POSIX 1003.2 indica las especificaciones mínimas de una shell

Filosofía Linux

Utilidades simples y pequeñas que se utilizan como un eslabón de una cadena de comandos (`$ ls -la | more`)
KISS: Keep It Small and Simple...o...

La shell de Linux

Introducción

Orignalmente Linux no disponía de interfaz gráfica

¿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.

Programas de shell

- Se puede programar rápidamente y de forma simple
- Disponible en la mayoría de las instalaciones del SO Linux
- Los programas de la shell se llaman *scripts* (interpretados en tiempo de ejecución)

POSIX 1003.2 indica las especificaciones mínimas de una shell

Filosofía Linux

Utilidades simples y pequeñas que se utilizan como un eslabón de una cadena de comandos (`$ ls -la | more`)
KISS: Keep It Small and Simple. . . o. . . (Keep It Simple Stupid :)

La shell de Linux

Usuarios y comandos

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

La shell de Linux

Usuarios y comandos

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

Usuario

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Nro. de usuario, UID. `whoami`
- Los usuarios se organizan en grupos. Nro. de grupo, GID. `groups`. `id`
- Usuario **root** (superusuario o admin), cuenta con todos los privilegios

La shell de Linux

Usuarios y comandos

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

Usuario

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Nro. de usuario, UID. `whoami`
- Los usuarios se organizan en grupos. Nro. de grupo, GID. `groups`. `id`
- Usuario **root** (superusuario o admin), cuenta con todos los privilegios

Comandos

- Archivo binarios o ejecutables localizados en el sistema. `which`. Variable `PATH`
- Auto-completar `TAB`

La shell de Linux

Usuarios y comandos

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

Usuario

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Nro. de usuario, UID. `whoami`
- Los usuarios se organizan en grupos. Nro. de grupo, GID. `groups`. `id`
- Usuario **root** (superusuario o admin), cuenta con todos los privilegios

Comandos

- Archivo binarios o ejecutables localizados en el sistema. `which`. Variable `PATH`
- Auto-completar `TAB`

- `/bin`: binarios utilizados por el sistema de arranque/buteo
- `/usr/bin`: binarios de usuario
- `/usr/local/bin`: binarios locales, programas específicos de la instalación

La shell de Linux

Variables de entorno y. . . más comandos

Variables de entorno

- Variables inicializadas desde el entorno
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `$HOME`, `$PATH`, `$PS1`, `$PS2`, `$IFS`, `$0`, `$#`, `$$`

La shell de Linux

Variables de entorno y. . . más comandos

Variables de entorno

- Variables inicializadas desde el entorno
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `$HOME`, `$PATH`, `$PS1`, `$PS2`, `$IFS`, `$0`, `$#`, `$$`

Más sobre comandos:

- Opciones de los comandos, letra seguido de '-'. O bien '--' (`--help`, `--version`)
- Comandos `echo`, `printenv`, `man`, `apropos`
- **bash**: prompt. Aparece en la línea de comandos indicando que está a la espera de órdenes (`$ echo $SHELL`, `$ bash --version`)
- `Manpages`: manual en línea (RTFM). `man man`, `man 1 printf`, `man 3 printf`

La shell de Linux

Variables de entorno y. . . más comandos

Variables de entorno

- Variables inicializadas desde el entorno
- Escritas en mayúsculas para distinguirlas de otras variables (usuario)
- Ej.: `$HOME`, `$PATH`, `$PS1`, `$PS2`, `$IFS`, `$0`, `$#`, `$$`

Más sobre comandos:

- Opciones de los comandos, letra seguido de '-'. O bien '--' (`--help`, `--version`)
- Comandos `echo`, `printenv`, `man`, `apropos`
- **bash**: prompt. Aparece en la línea de comandos indicando que está a la espera de órdenes (`$ echo $SHELL`, `$ bash --version`)
- `Manpages`: manual en línea (RTFM). `man man`, `man 1 printf`, `man 3 printf`

Variable `$PATH` y comando `export` (`$ man export`)

La shell de Linux

Páginas de manuales (manpages)

Cuenta con diferentes secciones: `$ man man`

La shell de Linux

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)

La shell de Linux

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'`

La shell de Linux
Sistema de archivos y permisos

- Estructura de archivos en árbol. Archivos tipo directorio.

La shell de Linux
Sistema de archivos y permisos

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)

La shell de Linux
Sistema de archivos y permisos

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comandos `cd`, `pwd`

La shell de Linux

Sistema de archivos y permisos

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comandos `cd`, `pwd`
- Camino absoluto (comenzando en `/`), y camino relativo (comenzando en `./` o `../`)

La shell de Linux

Sistema de archivos y permisos

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comandos `cd`, `pwd`
- Camino absoluto (comenzando en `/`), y camino relativo (comenzando en `./` o `../`)
- Comandos `ls`, `touch`, `rm`, `cd`, `mkdir`, `rmdir`, `cp`, `mv`. (`$ which cd`)

La shell de Linux

Sistema de archivos y permisos

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comandos `cd`, `pwd`
- Camino absoluto (comenzando en `/`), y camino relativo (comenzando en `./` o `../`)
- Comandos `ls`, `touch`, `rm`, `cd`, `mkdir`, `rmdir`, `cp`, `mv`. (`$ which cd`)
- Atributos de archivos (`$ ls -l /`)


-rw-rw-r-- 1 gfpp gfpp 11 ago 4 15:25 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

Alterar permisos de archivos. Comando `chmod`

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

Alterar permisos de archivos. Comando `chmod`

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

Alterar permisos de archivos. Comando `chmod`

- `$ chmod u-r hola.txt`.
- `$ cat hola.txt`
- `$ chmod o+x hola_bash.sh` (Crear script Shell)

Alterar permisos de archivos. Comando `chmod`

- `$ chmod u-r hola.txt`.
- `$ cat hola.txt`
- `$ chmod o+x hola_bash.sh` (Crear script Shell)

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
```

La shell de Linux

Redirección de entrada y salida. Pipes

Redirección de salida

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

La shell de Linux

Redirección de entrada y salida. Pipes

Redirección de salida

- `$ 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

La shell de Linux

Redirección de entrada y salida. Pipes

Redirección de salida

- `$ 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
- `$ kill -HUP 1234 > killout.txt 2> killerr.txt`

La shell de Linux

Redirección de entrada y salida. Pipes

Redirección de salida

- `$ 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
- `$ kill -HUP 1234 > killout.txt 2> killerr.txt`
- `$ kill -1 1234 > killouterr.txt 2>&1`

La shell de Linux

Redirección de entrada y salida. Pipes

Redirección de salida

- `$ 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
- `$ kill -HUP 1234 > killout.txt 2> killerr.txt`
- `$ kill -1 1234 > killouterr.txt 2>&1`
- `$ kill -1 1234 > /dev/null 2>&1`

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

La shell de Linux

Redirección de entrada y salida. Pipes

Redirección de salida

- `$ 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
- `$ kill -HUP 1234 > killout.txt 2> killerr.txt`
- `$ kill -1 1234 > killouterr.txt 2>&1`
- `$ kill -1 1234 > /dev/null 2>&1`

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

Redirección de entrada

- `$ more < lsoutput.txt`

La shell de Linux

Redirección de entrada y salida. Pipes

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

La shell de Linux

Redirección de entrada y salida. Pipes

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

Si no se utiliza pipes, se necesitan varios pasos

1. `$ ps > psout.txt`
2. `$ sort psout.txt > pssort.txt`

La shell de Linux

Redirección de entrada y salida. Pipes

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

Si no se utiliza pipes, se necesitan varios pasos

1. `$ ps > psout.txt`
2. `$ sort psout.txt > pssort.txt`

Conectando los procesos mediante pipes

- `$ ps | sort > pssort.txt`

La shell de Linux

Redirección de entrada y salida. Pipes

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

Si no se utiliza pipes, se necesitan varios pasos

1. `$ ps > psout.txt`
2. `$ sort psout.txt > pssort.txt`

Conectando los procesos mediante pipes

- `$ ps | sort > pssort.txt`

Otros ejemplos

- `$ ps aux | sort`
- `$ ps aux | sort | more`

La shell de Linux

Procesos

- En un sistema monotarea se utiliza generalmente el término programa.
- En un sistema multitarea se utiliza el término proceso, indicando que el mismo está arrancado y en funcionamiento.
- Un programa puede dar lugar a varios procesos.
- Un proceso puede estar detenido (dormido), pero existe información del estado del mismo.
- Cada proceso tiene un nro. que lo identifica, PID, PPID
- Comandos `pstree`, `ps`, `top`/`htop`, `kill` (señal 15 y 9)
- `$ pstree -p`, `$ ps axf`

La shell de Linux

Procesos

- En un sistema monotarea se utiliza generalmente el término programa.
- En un sistema multitarea se utiliza el término proceso, indicando que el mismo está arrancado y en funcionamiento.
- Un programa puede dar lugar a varios procesos.
- Un proceso puede estar detenido (dormido), pero existe información del estado del mismo.
- Cada proceso tiene un nro. que lo identifica, PID, PPID
- Comandos `pstree`, `ps`, `top`/`htop`, `kill` (señal 15 y 9)
- `$ pstree -p`, `$ ps axf`

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

Programas de shell

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

Programas de shell

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

Programa interactivo: se tiene una gran cantidad de archivos C y desea examinar los archivos para ver si contienen la cadena POSIX

```
$ for file in *
> do
> if grep -l POSIX $file
> then
> more $file
> fi
> done
```

Programas de shell

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

Programa interactivo: se tiene una gran cantidad de archivos C y desea examinar los archivos para ver si contienen la cadena POSIX

```
$ for file in *
> do
> if grep -l POSIX $file
> then
> more $file
> fi
> done
```

(cambio del prompt de $ a >)

La shell de Linux

Scripts de bash

```
#!/bin/sh

# first
# This file looks through all the files in the current
# directory for the string POSIX, and then prints the names of
# those files to the standard output.

for file in *
do
    if grep -q POSIX $file
    then
        echo $file
    fi
done

exit 0
```

La shell de Linux

Scripts de bash

```
#!/bin/sh

# first
# This file looks through all the files in the current
# directory for the string POSIX, and then prints the names of
# those files to the standard output.

for file in *
do
    if grep -q POSIX $file
    then
        echo $file
    fi
done

exit 0
```

- Los comentarios comienzan con `#`
- Línea especial `#!/bin/bash`
- Comando `exit` devuelve un valor de salida
- Archivo sin extensión (`$ file script_name`, `$ file /bin/bash`)

La shell de Linux
Scripts de bash

- `$ /bin/bash first`

La shell de Linux
Scripts de bash

- `$ /bin/bash first`
- `$ chmod +x first`

La shell de Linux
Scripts de bash

- `$ /bin/bash first`
- `$ chmod +x first`
- Ejecutar (error): `$ first`

La shell de Linux
Scripts de bash

- `$ /bin/bash first`
- `$ chmod +x first`
- Ejecutar (error): `$ first`
- `$ PATH=$PATH:.`

La shell de Linux
Scripts de bash

- `$ /bin/bash first`
- `$ chmod +x first`
- Ejecutar (error): `$ first`
- `$ PATH=$PATH:.`
- `$ first`

La shell de Linux
Scripts de bash

- `$ /bin/bash first`
- `$ chmod +x first`
- Ejecutar (error): `$ first`
- `$ PATH=$PATH:.`
- `$ first`

O bien, lo que es más común:
`$ ./first`

La shell como lenguaje de programación

- Lenguaje de programación fácil de aprender
- Se puede hacer pruebas de forma interactiva y luego ir modifcando el script

La shell como lenguaje de programación

- Lenguaje de programación fácil de aprender
- Se puede hacer pruebas de forma interactiva y luego ir modifcando el script

Sintaxis de la shell:

- Variables: string, numbers, environments, and parameters
- Conditions: shell Booleans
- Program controls: `if`, `elif`, `for`, `while`, `util`, `case`
- Listas
- Funciones
- Comandos build-in: `break`, `continue`, `exit`, etc.
- etc.

La shell como lenguaje de programación

- Lenguaje de programación fácil de aprender
- Se puede hacer pruebas de forma interactiva y luego ir modifcando el script

Sintaxis de la shell:

- Variables: string, numbers, environments, and parameters
- Conditions: shell Booleans
- Program controls: `if`, `elif`, `for`, `while`, `util`, `case`
- Listas
- Funciones
- Comandos build-in: `break`, `continue`, `exit`, etc.
- etc.

La shell de Linux

Scripts de bash – variables

- No se declaran. Se les asigna valor directamente
- Por defecto se consideran y son almacenadas como strings, aún cuando se le asignen valores numéricos
- Linux es case-sensitive `foo` es una variable diferente de `Foo`, y ambas diferentes de `FOO`
- Dentro de la shell se tiene acceso al valor de la variable precediendola por `$`

La shell de Linux

Scripts de bash – variables

- No se declaran. Se les asigna valor directamente
- Por defecto se consideran y son almacenadas como strings, aún cuando se le asignen valores numéricos
- Linux es case-sensitive `foo` es una variable diferente de `Foo`, y ambas diferentes de `FOO`
- Dentro de la shell se tiene acceso al valor de la variable precediendola por `$`

Probar:

1. `$ VAR=Hello`
2. `$ echo $VAR`

La shell de Linux

Scripts de bash – variables

- No se declaran. Se les asigna valor directamente
- Por defecto se consideran y son almacenadas como strings, aún cuando se le asignen valores numéricos
- Linux es case-sensitive `foo` es una variable diferente de `Foo`, y ambas diferentes de `FOO`
- Dentro de la shell se tiene acceso al valor de la variable precediendola por `$`

Probar:

1. `$ VAR=Hello`
2. `$ echo $VAR`
3. `$ VAR="Shell VAR"`
4. `$ echo $VAR`

La shell de Linux

Scripts de bash – variables

- No se declaran. Se les asigna valor directamente
- Por defecto se consideran y son almacenadas como strings, aún cuando se le asignen valores numéricos
- Linux es case-sensitive `foo` es una variable diferente de `Foo`, y ambas diferentes de `FOO`
- Dentro de la shell se tiene acceso al valor de la variable precediendola por `$`

Probar:

1. `$ VAR=Hello`
2. `$ echo $VAR`
3. `$ VAR="Shell VAR"`
4. `$ echo $VAR`
5. `$ VAR=7+5`
6. `$ echo $VAR`

La shell de Linux

Scripts de bash – variables

- No se declaran. Se les asigna valor directamente
- Por defecto se consideran y son almacenadas como strings, aún cuando se le asignen valores numéricos
- Linux es case-sensitive `foo` es una variable diferente de `Foo`, y ambas diferentes de `FOO`
- Dentro de la shell se tiene acceso al valor de la variable precediendola por `$`

Probar:

1. `$ VAR=Hello`
2. `$ echo $VAR`
3. `$ VAR="Shell VAR"`
4. `$ echo $VAR`
5. `$ VAR=7+5`
6. `$ echo $VAR`
7. `$ read VAR`
8. `$ echo $VAR`

La shell de Linux

Scripts de bash – variables

- No se declaran. Se les asigna valor directamente
- Por defecto se consideran y son almacenadas como strings, aún cuando se le asignen valores numéricos
- Linux es case-sensitive `foo` es una variable diferente de `Foo`, y ambas diferentes de `FOO`
- Dentro de la shell se tiene acceso al valor de la variable precediendola por `$`

Probar:

1. `$ VAR=Hello`
2. `$ echo $VAR`
3. `$ VAR="Shell VAR"`
4. `$ echo $VAR`
5. `$ VAR=7+5`
6. `$ echo $VAR`
7. `$ read VAR`
8. `$ echo $VAR`

Ver ejemplo con comillas simples, dobles y '\'. Ver comando `read`

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

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

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)
- `$PS2`: prompt secundario (`>`)

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)
- `$PS2`: prompt secundario (`>`)
- `$IFS`: campo de separación de entrada. Lista de caracteres que se utilizan para separar palabras cuando se lee entrada en la shell, generalmente es un espacio, tab o nueva línea

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)
- `$PS2`: prompt secundario (`>`)
- `$IFS`: campo de separación de entrada. Lista de caracteres que se utilizan para separar palabras cuando se lee entrada en la shell, generalmente es un espacio, tab o nueva línea
- `$0`: nombre del script de shell

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)
- `$PS2`: prompt secundario (`>`)
- `$IFS`: campo de separación de entrada. Lista de caracteres que se utilizan para separar palabras cuando se lee entrada en la shell, generalmente es un espacio, tab o nueva línea
- `$0`: nombre del script de shell
- `$#`: cantidad de parámetros pasados

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)
- `$PS2`: prompt secundario (`>`)
- `$IFS`: campo de separación de entrada. Lista de caracteres que se utilizan para separar palabras cuando se lee entrada en la shell, generalmente es un espacio, tab o nueva línea
- `$0`: nombre del script de shell
- `$#`: cantidad de parámetros pasados
- `$$`: ID del proceso del script del shell, generalmente utilizado dentro del script para generar nombres de archivos temporarios, p.e. `/tmp/tmpfile_$$`

La shell de Linux

Scripts de bash – variables de entorno

Cuando se ejecuta un script de shell, existen algunas variables inicializadas desde el entorno (generalmente escritas en mayúsculas)

- `$HOME`: directorio home del usuario actual
- `$PATH`: lista de directorios separados por `:` para buscar los comandos
- `$PS1`: prompt de comandos (`$`)
- `$PS2`: prompt secundario (`>`)
- `$IFS`: campo de separación de entrada. Lista de caracteres que se utilizan para separar palabras cuando se lee entrada en la shell, generalmente es un espacio, tab o nueva línea
- `$0`: nombre del script de shell
- `$#`: cantidad de parámetros pasados
- `$$`: ID del proceso del script del shell, generalmente utilizado dentro del script para generar nombres de archivos temporarios, p.e. `/tmp/tmpfile_$$`

Script: `try_var`

La shell de Linux

Scripts de bash – variables de entorno

Variables de parámetros:

- `$1`, `$2`, ...: parámetros dados al script
- `$*`: lista de todos los parámetros en una única variable, separados por el primer carácter de la variable de entorno `IFS`
- `$@`: variación sutil de `$*`, no utiliza la variable de entorno `IFS`, los parámetros están juntos aún si `IFS` está vacía

La shell de Linux

Scripts de bash – variables de entorno

Variables de parámetros:

- `$1`, `$2`, ...: parámetros dados al script
- `$*`: lista de todos los parámetros en una única variable, separados por el primer carácter de la variable de entorno `IFS`
- `$@`: variación sutil de `$*`, no utiliza la variable de entorno `IFS`, los parámetros están juntos aún si `IFS` está vacía

Probar:

1. `$ IFS=''`
2. `$ set foo bar bam`
3. `$ echo "$@"`
4. `$ echo "$*"`
5. `$ unset IFS`
6. `$ echo "$*"`

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`
- El comando `[` puede parecer extraño, pero presenta una sintaxis clara y ordenada

La shell de Linux

Scripts de bash – condiciones

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`
- El comando `[` puede parecer extraño, pero presenta una sintaxis clara y ordenada
- Generalmente el comando `[` se cierra con `]`

La shell de Linux

Scripts de bash – condiciones

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`
- El comando `[` puede parecer extraño, pero presenta una sintaxis clara y ordenada
- Generalmente el comando `[` se cierra con `]`

Ejemplo: verificar si existe un archivo

```
if test -f main.c
then
...
fi
```

La shell de Linux

Scripts de bash – condiciones

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`
- El comando `[` puede parecer extraño, pero presenta una sintaxis clara y ordenada
- Generalmente el comando `[` se cierra con `]`

Ejemplo: verificar si existe un archivo

```
if test -f main.c
then
...
fi
```

```
if [ -f main.c ]
then
...
fi
```

La shell de Linux

Scripts de bash – condiciones

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`
- El comando `[` puede parecer extraño, pero presenta una sintaxis clara y ordenada
- Generalmente el comando `[` se cierra con `]`

Ejemplo: verificar si existe un archivo

```
if test -f main.c
then
...
fi
```

```
if [ -f main.c ]
then
...
fi
```

```
if [ -f main.c ]; then
...
fi
```

(en la misma línea)

Comandos `test` y `[`

- Los comandos `test` y `[` son sinónimos
- La mayoría de los scripts utilizan mucho los comandos `test` o `[`
- El comando `[` puede parecer extraño, pero presenta una sintaxis clara y ordenada
- Generalmente el comando `[` se cierra con `]`

Ejemplo: verificar si existe un archivo

```
if test -f main.c
then
...
fi
```

```
if [ -f main.c ]
then
...
fi
```

```
if [ -f main.c ]; then
...
fi
```

(en la misma línea)

Tipos de condiciones (tres categorías):

1. comparación de cadenas
2. comparación aritmética
3. condiciones de archivos

String comparison	**result**
`string1 = string2`	True if the strings are equal
`string1 != string2`	True if the strings are not equal
`-n string`	True if the string is not null
`-z string`	True if the string is null (an empty string)
Arithmetic comparison	**result**
`expr1 -eq expr`	True if the expressions are equal
`expr1 -ne expr`	True if the expressions are not equal
`expr1 -gt expr`	True if `expr1` is greater than `expr2`
`expr1 -ge expr`	True if `expr1` is greater than or equal to `expr2`
`expr1 -lt expr`	True if `expr1` is less than `expr2`
`expr1 -le expr`	True if `expr1` is less than or equal to `expr2`
`! expr`	True if the expression is false, and vice versa

La shell de Linux

Scripts de bash – condiciones

File conditional	result
`-d file`	True if the file is a directory
`-f file`	True if the file is a regular file
`-g file`	True if `set-group-id` is set on `file`
`-r file`	True if the file is readable
`-s file`	True if the file has nonzero size
`-u file`	True if `set-user-id` is set on `file`
`-w file`	True if the file is writable
`-x file`	True if the file is executable

La shell de Linux

Scripts de bash – condiciones

File conditional	result
`-d file`	True if the file is a directory
`-f file`	True if the file is a regular file
`-g file`	True if `set-group-id` is set on `file`
`-r file`	True if the file is readable
`-s file`	True if the file has nonzero size
`-u file`	True if `set-user-id` is set on `file`
`-w file`	True if the file is writable
`-x file`	True if the file is executable

Script ejemplo: `condition`

La shell de Linux

Scripts de bash – estructuras de control

- `if`
- `elif`
- `for`
- `while`

La shell de Linux

Scripts de bash – estructuras de control

- `if`
- `elif`
- `for`
- `while`

(ver scripts ejemplos)