



*Facultad de Ingeniería
Universidad de Buenos
Aires*

75.08 Sistemas Operativos
Lic. Ing. Osvaldo Clúa
Lic. Adrián Muccio

Shell Scripting II

Unix

Funciones del Shell

- Intérprete de comandos: Modo de ejecución
 - Foreground: con/sin shell hijo (.)
 - Background (&) ← Se puede traer a primer plano con el comando fg seguido del pid. Ej:
> fg 2041
 - Asignación (`) ← La salida std de un comando puede ser el término derecho de una asignación. Ej:
> a=`ls`

Unix

Funciones del Shell

- Redireccionamiento de entrada std

> cat <archivo.input

- Redireccionamiento de salida std

> cat archivo.input > archivo.output

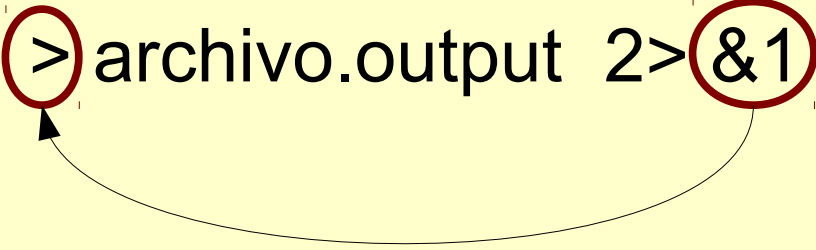
- Redireccionamiento de error std

> cat archivo.input 2> archivo.error

Unix

- Redireccionamiento concurrente de salida std y error

> proceso.sh > archivo.output 2> &1



Referencia a std output

Unix

- Todos los procesos nacen con 3 archivos abiertos
- Cada proceso tiene asociada una tabla con los descriptores de los archivos que utiliza llamada *File Descriptor Table*.
- Las 3 primeras entradas son:
 - 0: *Std Input*
 - 1: *Std Output*
 - 2: *Std Error*

Unix

- Cuando el Shell recibe el control su primera acción es realizar los redireccionamientos de entrada/salida
- Según el siguiente ejemplo:

```
> cat archivo.1 > otro.archivo
```

Si *otro.archivo* no existe, lo crea.

Si *otro.archivo* existe, lo trunca.

Unix

Funciones del Shell

- `|`: *Pipeline*
- Es un redireccionamiento especial donde la salida de un comando se redirecciona como la entrada std de otro. Ejemplo:

```
> cut -c1-10,20-30 archivo* | fgrep 'HOY' | sort -u | more
```

cut también permite tomar campos. Ejemplo:

```
> cut -f3-5 -d',' archivo # toma como separador de campos el caracter .
```

- ```
> paso1.sh archivos* | tee log | paso2.sh > salida
```

# Unix

## Funciones del Shell

- Variables de Ambiente
- El shell permite el uso de variables de ambiente como parte del scripting
- Realiza la expansión del valor
- A menos que se evite en forma explícita, eso se denomina *Protección del Shell*



# Unix

## Funciones del Shell

- Caracteres Comodines
- El Shell interpreta en forma especial ciertos caracteres para la expansión de nombres de archivos.
- A menos que se evite en forma explícita.

# Unix

- Supongamos que el contenido del directorio corriente es:

```
archivo
Archivo
Archivo1
Archivo1.dat
Archivo1.datos
Archivo2
Archivo2.txt
Archivo3.txt
Archivo23.txt
Mi_Script.sh
```

# Unix

?: el Shell lo reemplaza por cualquier caracter

```
> ls Archivo?
```

```
Archivo1
```

```
Archivo2
```

aquí vemos que no aparecen Archivo ni archivo

```
> ls ?rchivo?
```

```
Archivo1
```

```
Archivo2
```

# Unix

\*: el Shell lo reemplaza por cualquier caracter en cualquier cantidad de ocurrencias (incluye la ocurrencia nula)

```
> ls ?rchivo*
archivo
Archivo
Archivo1
Archivo1.dat
Archivo1.datos
Archivo2
Archivo2.txt
Archivo23.txt
Archivo3.txt
```

# Unix

`[]`: define conjunto, el shell lo reemplaza por algún carácter que pertenezca al conjunto

```
> ls *rchivo[23].*
```

```
Archivo2.txt
Archivo3.txt
```

El conjunto puede ser definido en forma explícita enumerando los caracteres que lo componen

```
> ls *rchivo[2-3].*
```

```
Archivo2.txt
Archivo3.txt
```

El conjunto puede ser definido en forma explícita por medio de un rango

# Unix

!: niega el conjunto

```
> ls [!A]*
```

```
archivo
Mi_Script.sh
```

# Unix

## Protección del Shell

- “: Evitan la expansión de los caracteres comodines, por lo tanto son interpretados por el comando en forma literal. Ejemplo:

```
> var="existe_novedad?"
```

```
> ls -l "$var" "novedad[1]"
```

```
existe_novedad?
```

```
novedad[1]
```

# Unix

## Protección del Shell

- \: Evita toda acción del Shell sobre el caracter inmediato posterior. Ejemplo:

```
> ls -l existe_novedad\? novedad\[1\]
```

```
existe_novedad?
```

```
novedad[1]
```



# Unix

## Protección del Shell

- ': Evita toda acción del Shell hasta la próxima '.  
Ejemplo:

```
> var="existe_novedad?"
```

```
> ls -l '$var' 'novedad[1]'
```

```
novedad[1]
```

No muestra existe\_novedad? porque al ls solo le llegan los 4 caracteres del string \$var

# Unix

## Agrupamiento

- Tanto “ como ' producen el efecto de agrupamiento de parámetros. Ejemplo:

```
> ls -l “existe_novedad? Novedad[1]”
```

No va a listar el nombre de los archivos porque al comando ls le llega un solo parámetro de 26 caracteres con valor:

```
existe_novedad? Novedad[1]
```

# Unix

## Agrupamiento

- Tanto “ como ' producen el efecto de agrupamiento de parámetros. Ejemplo:

```
> ls -l “existe_novedad? Novedad[1]”
```

No va a listar el nombre de los archivos porque al comando ls le llega un solo parámetro de 26 caracteres con valor:

```
existe_novedad? Novedad[1]
```

# Unix

## Ejercicio

- En `/etc/passwd` se encuentra la informacion de todos los usuarios del sistema
- Cada línea del archivo se compone de:  
`user:X:user_id:group_id:Nombre:home:shell`
- Ejemplo de línea del `/ect/passwd`:  
`lalujan:4Mcbn2/PcSwrl:528:501::/home/lalujan:/bin/bash`

# Unix

## Ejercicio

- Se desea listar los id y nombres de todos los usuarios del sistema (ordenados alfabéticamente por nombre)

# Unix

## Construyendo Shell Scripts

- Parámetros de dentro del script:
  - \$0: Nombre con que fue invocado el script
  - \$1: Parámetro posicional 1
  - ..\$9: Parámetro posicional 9
  - \$@: Lista de los parámetros (excepto parámetro 0)
  - \$#: Cantidad de parámetros (excepto parámetro 0)

# Unix

## Construyendo Shell Scripts

- Estructuras de control
- El Shell soporta estructuras de control, como por ejemplo:

- if

if [cond]

then

fi

Ej: if [\$A = 'PRIMERO']

Las condiciones se pueden realicionar mediante -a, -o

# Unix

- for: su compartamiento por defecto es iterar por los elementos de una lista. Ejemplo:  
for i in “elemento1 elemento2 elemento3”  
do  
done
- En caso de querer iterar en por las líneas de un archivo es posible modificar el comportamiento de la siguiente forma:



# Unix

IFS='

' #variable que indica el separador de registro

for i in `cat archivo.input`

do

echo \$i #Muestra cada línea del archivo

done

# Unix

- Operaciones aritméticas (+, -, \*, /)
- Existen distintas posibilidades, por ejemplo:

```
a=`expr $a + 1`
```

```
let a=a+1
```

```
a=`echo $a + 1 | bc`
```

# Unix

## Otras utilidades del Shell

- [ -f archivo ] : Verdadero si existe archivo
- [ -r archivo ] : Verdadero si existe archivo y tiene permiso de lectura para el usuario
- [ -w archivo ] : Verdadero si existe archivo y tiene permiso de escritura para el usuario

# Unix

- Ejercicio CONVERSION DE TEMPERATURAS

Cree un programa de shell usando las siguientes formulas que efectúen la conversión de temperaturas.

Dados grados Celsius, los convierte a Fahrenheit y viceversa.

$$C = 5 \cdot (F - 32) / 9$$

$$F = 9 \cdot C / 5 + 32$$

# Unix

- Manejo de Patrones

```
$ archivo=./usr/apps/bin/cmd.exe
```

```
$ echo ${archivo%/*}
```

```
./usr/apps/bin # le saco desde la última /
```

```
$ echo ${archivo%%/*}
```

```
. # le saco desde la primera /
```

```
$ echo ${archivo#*/}
```

```
usr/apps/bin/cmd.exe # le saco hasta la primera /
```

```
$ echo ${archivo##*/}
```

```
cmd.exe # le saco hasta la última /
```

# Unix

- Verificación de existencia

a='hola a todos'

\$ echo \${b:-no existe}

no existe

\$ echo \${a:-no existe}

hola a todos

\$ echo \${a:+si existe}

si existe