Repaso programación Shell

Comandos para manipular archivos.

- **cat:** Muestra contenido y crea con redirección archivos
- **sort:** Ordena en modo ascendente un fichero. Concatenable.
    - sort (opciones) archivo
        - -r : Ordena en sentido inverso
        - -n : Ordena de forma numérica
        - -t : Usa el carácter especificado entre comillas como delimitador
        - -k : Selecciona los campos indicados (uno, un rango, varios separados por comas)
- **grep:** Busca y devuelve todas las líneas que tiene un patrón (texto) determinado.
    - grep (opciones) “patrón” archivo
        - -i : se vuelve case insensitive
        - -v : Muestra las líneas que no contienen el patrón
        - “^patrón” : Muestra lineas que empiezan por el patrón
        - “patrón$” : Muestra lineas que terminan por el patrón
- **diff:** Compara ficheros de texto
    - diff (opciones) fichero-desde fichero-hasta
- **uniq:** Elimina lineas duplicadas de un fichero ordenado
    - sort archivo | uniq (opciones)
    - -c : Genera un informe de salida precedida por un contador de líneas repetidas
- **cut:** Muestra columnas delimitadas de los ficheros
    - cut (opciones) ficheros
        - -d : Usa el carácter especificado entre comillas como delimitador
        - -f : Muestra los campos indicados (uno, un rango, varios separados por comas)
        - -c : Muestra los caracteres que se especifiquen de cada columna.
- **find:** Busca ficheros que cumplan unas condiciones
    - find [ruta] (opciones)
        - **-iname** (nombre)
        - -type d,f

Comandos y variables variados

id – Muestra información sobre el usuario
env – Muestra las variables de entorno

Las siguientes variables se invocan con echo
$**PATH** – Rutas para el interprete de ordenes
$**PWD** – Muestra la ruta absoluta del directorio donde nos encontramos
$**HOME –** Ruta del directorio personal.
$**SHELL –** Ruta absoluta del shell que estamos ejecutando.
$**HOSTNAME** – Nombre asignado al equipo.
$**USER –** Nombre de usuario.
$**LOGNAME** – Muestra cual es nuestro usuario.
$**(($RANDOM%X))** – Genera un número (X) aleatorio.

Paso de parámetros

$0 : Devuelve el nombre del script.
$1, 2…9 : Devuelve el 1º, 2 … 9º parámetro pasado al script.
$* : Devuelve todos los parametros separados por un espacio.
$? : Devuelve si el último comando se ha ejecutado (da valor 0) o no correctamente.
$# : Devuelve el número de parámetros que se han pasado.
Shift: Aumenta en 1 el número de parametro pasado al script
$(comando) : Nos permite asignale a una variable el valor de un comando.

Operadores variados

Aritméticos: Para realizar operaciones. Se emplean con el comando **let**. Para operar con las variables no es necesario incluir el simbolo $

- +, -, *, :, %

Relacionales:

- Para comparar cadenas alfanuméricas. Colocar valores entre comillas.
  - =, !=, <, >
- Para comparar valores numéricos. Colocar un guión (-) delante.
  - eq, ne, lt, gt, le, ge

Condicionales:

- Para comprobar si existe. Usados con IF.
  - -d : verdadero si existe y es directorio.
  - -f : verdadero si existe y es fichero normal.
  - -r : verdadero si existe y tiene permisos de lectura.
  - -w : verdadero si existe y tiene permisos de escritura.
  - -x : verdadero si existe y tiene permisos de ejecución.

Lógicos:

- ! : NO cumple con la condición especificada.
- && : Cumple con las condiciones representadas en la expresion.
- || : Cumple con alguna de las condiciones representadas en la expresion.

Estructuras condicionales

Cláusula IF: Principal en shell. Cada “expresión” produce resultado de cierto o falso. Si la expresion es cierto ejecuta su correspondiente comando y termina. Si no, pasa al siguiente comando y se repite. Si se llega al último comando termina.
Se pueden usar los operadores && ó || para comparar dos expresiones con operadores relacionales.

```
if [ expresión1 ]; then
   (comando1)
elif [ expresión2 ]; then
   (comando2)
else
   (Último comando)
fi
```

Comando TEST: Prueba condiciones y devuelve cierto o falso. Se emplea con el comando IF. Permite comparar una o varias condiciones: [cond. Valor **-a/-o** cond. Valor] ó
[cond. Valor] **&&/||** [cond. Valor]

```
if [ cond. Valor ]; then
   (comando1)
elif [ cond. Valor ]; then
   (comando2)
else
   (Último comando)
fi
```

Estructuras repetitivas

Estructura WHILE: Repite algo hasta que se cumple la condición. Se traduce como “mientras...”

```
while [ expresión ]; do
  (comandos a repetir)
done
```

Estructura UNTIL: Repite algo hasta que se cumple la condición. Se traduce como “hasta…”

```
until [ expresión ]; do
  (comandos a repetir)
done
```

Estructura FOR: Permite repetir un comando una cantidad de veces definida.

```
for i in $( seq 1 1 10 ); do
  (comandos a repetir)
done
```

Estructuras defensivas

Fichero como parámetro: Evita que se pasen como parámetros mas o menos ficheros de los indicados y muestra un mensaje de error.

```
if [ $# (relac.num.) (valor deseado) ]; then
  echo “mensaje de error”
else
  (resto del código)
fi
```

Acotar rango de valores: Evita que se exceda el rango de valores del script.

```
if [ (variable) (relac.num.) (val.des) ] || [ (variable) (relac.num.) (val.des) ]; then
  echo “mensaje error”
else
  (resto de código)
fi
```

Precaución

- Ojo con los espacios
- Poner el $ cuando se llama a una variable.
- Repasar el nombre de las variables.
- Dentro de los corchetes todo va separado.
- Si no queremos ver el resultado de un comando lo redirigimos a /dev/null