LENGUAJES DE CONTROL DE ÓRDENES
KORN SHELL

CONTENIDOS:

1. Introducción
2. Aspectos básicos
3. Variables y parámetros
4. Control de tareas
5. Operaciones aritméticas
6. El ambiente
7. Escritura de programas
8. Ejemplos

Korn Shell: bibliografía

- “The Korn Shell. User & Programming Manual”, Anatole Olczak, Addison-Wesley Publishing Company, 1992.
- Korn Shell Lenguaje de Programación y Comando. Jesús Alberto Vidal Cortes
- The New KornShell Command and Programming Language. Morris Bolksy & David Korn. Prentice Hall, 1995.
- www.kornshell.com

KSH: Introducción

- El Korn Shell (ksh) es un lenguaje de control y programación de tareas interactivo para entornos UNIX que:
  - Presenta un mejor rendimiento que los lenguajes tradicionales de control y programación de órdenes.
  - Compatible con otros lenguajes de control y programación de órdenes, por ejemplo el Bourne shell (bsh).
  - Soporta mayores facilidades de operaciones de Entrada/Salida.
  - Soporta tipos de datos y atributos.
  - Soporta vectores unidimensionales.
  - Soporta aritmética entera.
  - Proporciona facilidades para el manejo de cadenas de caracteres.
  - Soporta facilidades para el control de tareas.
  - Soporta funciones y “alias”
  - Etc.

KSH: 1. Introducción

- **Inicio de sesión.** cuando un usuario entra en el sistema se le ejecuta el intérprete de órdenes que tiene declarado en el fichero `/etc/passwd`, por ejemplo:
  - `lara:IWed4QkoRZ1d:101:12::/home/lara:/bin/ksh`
- **Cambio de intérprete de órdenes.** Simplemente tecleando el nombre, por ejemplo:
  - `$ ksh`
- **Retorno al intérprete de órdenes original.** Simplemente tecleando lo siguiente
  - `$ Ctl-d`

KSH: Introducción

- **Invocación separada de varios intérpretes de órdenes.** Cada intérprete posee un conjunto de variables de entorno que se definen para cada usuario. Estas se pueden almacenar en ficheros específicos:
  - Korn Shell (`ksh`): .kshrc
  - Bourne Shell (`bsh`): .profile
  - C Shell (`kshrc`): .login

KSH: 2. Conceptos básicos

- **Orden.** Se trata de una cadena de caracteres, organizada en palabras, en el que el carácter blanco se utiliza como separador entre palabras y en el que la primera palabra se interpreta como el nombre de un fichero ejecutable que contiene un programa.
  - `$ echo HOLA`
- **Continuación de orden.** Si una orden termina con el carácter "\", entonces la siguiente línea se interpreta como una continuación de la anterior.
  - `$ echo HOLA \`
  - `> JUAN`
- **Múltiples órdenes.** En una misma línea se pueden especificar varias órdenes empleando el carácter ";".
  - `$ echo HOLA; echo JUAN; echo MARIA`

KSH: Conceptos básicos

- **Ejecución de órdenes en segundo plano.** Toda orden que finalice con el carácter “&” el KSH la ejecutará, pero no esperará por su finalización, por tanto, mientras ésta se ejecuta se podrá ordenar la ejecución de otra.
  - `$ ls /usr &`

- **Tuberías (pipes).** Se trata de un mecanismo que permite la comunicación entre órdenes, de forma que la salidas que unas producen son las entradas de otras. Para establecer una tubería de comunicación entre órdenes se ha de especificar el carácter “|”.
  - `$ ls | wc -l`

KSH: Conceptos básicos

- **Retorno.** Cuando una orden (programa) se ejecuta ésta devuelve un valor entero, el valor 0 se suele interpretar como ejecución con éxito, un valor distinto de cero se suele interpretar como ejecución con error.

- **Ejecución condicional.** Si dos órdenes están separadas por "&&", si la primera devuelve el valor 0 entonces la segunda se ejecuta.
  - `$ ls temp && echo temp existe`

- **Ejecución condicional.** Si dos órdenes están separadas por "||", si la primera devuelve el valor distinto de 0 entonces la segunda se ejecuta.
  - `$ ls temp || echo temp no existe`
  - `$ ls temp && echo temp existe || echo temp no\ >existe`

KSH: Conceptos básicos

- **Agrupación de órdenes.** las órdenes encerrados con “{}” se ejecutan combinando sus salidas. Para que la construcción sea correcta debe haber un espacio en blanco después del símbolo “{“ y antes del símbolo “}”. las órdenes deberán ir separados por “;” y el último dela línea deberá ir terminado también por “;”.
  - `$ { echo El contenido de temp: ; cat temp ; } | nl`

KSH: Conceptos básicos

- **Redirección de entrada y salida.** Mediante los símbolos “>” y “>>” podemos redirigir el flujo de salida de una orden.
  - `$ echo Hola > prueba`
  - `$ echo Hola >> prueba`
  - `$ > tmp`

KSH: Conceptos básicos

- **La opción “noclobber”.** Podemos prevenir la escritura de ficheros mediante redireccionamientos mediante esta opción.
  - `$ set -o noclobber`
  - `$ set +o noclobber`

- **El operador “>|”.** Podemos forzar la escritura en un fichero incluso estando activa la opción “noclobber”
  - `$ ls >| temp`

KSH: Conceptos básicos

- **Redirección de entrada y salida.** Mediante el símbolo “<” podemos redirigir el canal de entrada de una orden.
  - `$ mail Juan Maria < mensaje`

- **Cierre de los canales de entrada y salida.** Mediante los operadores “<&-” y “>&-” podemos cerrar los canales de entrada y salida respectivamente de una orden.
  - `$ cat temp | wc -l <&-`

KSH: Conceptos básicos

- **Descriptores de ficheros.**
  - 0 Canal de entrada estándar
  - 1 Canal de salida estándar
  - 2 Canal de error estándar
  - 3-9 Descriptores disponibles

- **Redirección de cualquier canal.** Con el operador “n>” podemos redireccionar el canal n a un archivo, siendo “n” un descriptor de archivo válido.
  - `$ ls tmp t.out 2> ls.salida_error`

KSH: Conceptos básicos

- **Redirección de cualquier canal.** Con el operador “>&n” podemos redireccionar la salida de una orden al archivo especificado por el descriptor n, por tanto “n” debe ser un descriptor de archivo válido.
  - `$ echo Este mensaje va al canal de error >&2`
  - `$ ls tmp t.out >&2 2>ls.salida_error`
  - `$ { echo Esto va al canal de salida >&1 ; \`
  - `echo Esto va al canal de error >&2 ; }`
  - `$ { echo Esto va al canal de salida >&1 ; \`
  - `echo Esto va al canal de error >&2 ; } > salida`

KSH: Conceptos básicos

- **Redirección de canales.** Con el operador “n>&m”produce una salida donde los contenidos de los archivos referenciados por los descriptores n y m están anexados, por tanto “n” y “m”deben ser descriptores de archivos válidos.
  - `$ { echo Esto va al canal de salida >&1 ; \`
  - `echo Esto va al canal de error >&2 ; } >sal 2>&1`

KSH: Conceptos básicos

- **Metacaracteres.** Mediante estos caracteres especiales podemos formar listas de nombres de archivos que encajen con un patrón determinado:
  - “*” Cero o cualquier cadena de 1 o más caracteres
  - “?” Cualquier carácter
  - “[]” Cualquier carácter o rango de caracteres especificado
  - “!” Se usa con “[]” y significa negación, o sea, que no coincida con el carácter o rango de caracteres especificado.
  - “.” El punto inicial se emplea para los archivos ocultos y por ello debe se especificado explícitamente.

KSH: Conceptos básicos

- **Ejemplos de uso de metacaracteres.**
  - `$ ls *ab*`
  - `$ ls ??`
  - `$ ls ???*`
  - `$ ls [am]*[1-9]`
  - `$ ls [!a]*`
  - `$ ls .[a-h]`
  - `$ rm *.[!ab]`

KSH: Conceptos básicos

- **Operadores de patrones complejos.** El KSH permite buscar patrones formados por cadenas de caracteres:
  - ?(patrón) Cero o una ocurrencia de patrón.
  - *(patrón) Cero, una o más ocurrencias de patrón
  - +(patrón) Una o más ocurrencias de patrón
  - @(patrón) Una ocurrencia de patrón
  - !(patrón) Cualquier excepto en las que aparezca patrón

- **Lista de patrones.** Se pueden especificar múltiples patrones utilizando el carácter “|” como separador.

KSH: Conceptos básicos

- **Ejemplos de uso de operadores con patrones complejos.**
  - `*(A|i)`
  - `s?(?|??)`
  - `1?([0-9])`
  - `m+(iss)*`
  - `@([AC]la)*`

- **Desactivando el uso de metacaracteres o de opradores de patrones.**
  - `$ set -o noglob`
  - `$ set -f`

KSH: Conceptos básicos

- **Ejecución de órdenes.** Mediante la expresión $(*orden*) nos referimos a la salida producida por la ejecución de la orden:
  - `$ echo La fecha de hoy es $(date)`
  - `echo $(who -q) están en sesión`
  - `echo Hay $(who | wc -l) usarios en sesión`

- **Operaciones aritméticas.** Mediante la expresión $((expresion_aritmética)) podemos referirnos al resultado de la operación.
  - $ echo $((8-3))

KSH: Conceptos básicos

- **El operador “ ~”.** Con este operador se referencia atributos de entorno referidos a rutas de archivos:
  - “~” sinónimo del contenido de la variable `HOME`
  - “~user” sinónimo del contenido de la variable `HOME` del usuario user
  - “~-” sinónimo del contenido de la variable `OLDPWD`
  - “~+” sinónimo del contenido de la variable `PWD`

KSH: 3. Variables y Parámetros

- **Variable.** Un variable se define cuando se declara o cuando se le asigna un valor. Los nombres de variables deben empezar por un carácter alfabético (a-Z) al que le puede seguir cualquier carácter alfanumérico (a-Z,0-9). Existen variables cuyos nombres están compuesto por sólo número o caracteres especiales (!,@,#,%,*,?,$) que son de uso interno del KSH.
  - `$ X`
  - `$ typeset B`
  - `$ Z=abc`
  - `$typset K1 = hola`

- **Acceso al valor de la variable.** Mediante el operador “$”
  - `$ echo $Z`
  - `$ cd $HOME`

KSH: variables y parámetros

- **Atributos de variables.** Mediante la palabra clave “typeset” podemos establecer valores y/o atributos a las variables.
  - ***typeset*** *-atributo variable = valor*
  - ***typeset*** *-atributo variable*

- **Atributos soportados por el KSH.**
  - ***typeset*** *-i var*
  - ***typeset*** *-l var*
  - ***typeset*** *-L var*
  - ***typeset*** *-LZn var*
  - ***typeset*** *-r var*
  - ***typeset*** *-R var*
  - ***typset*** *-RZn var*
  - ***typeset*** *-t var*
  - ***typeset*** *-u var*
  - ***typeset*** *-x var*
  - ***typeset*** *-Z var*

KSH: variables y parámetros

- **Desactivación de atributos.** Todos los atributos de una variable pueden ser desactivados, excepto el atributo de sólo lectura.
  - ***typeset*** *+atributo variable*
- **Múltiples atributos.** Podemos establecer varios atributos de una variable mediante una sola línea de órdenes:
  - `$ typeset -ix TMOUT = 300`
- **Comprobación de los atributos de las variables.** Podemos listar conocer qué variables poseen un determinado atributo
  - ***typeset*** *-atributo*
  - ***typeset*** *+atributo*

KSH: variables y parámetros

- **Asignación de valores a variables.** Ésta se puede realizar de distintas formas
  - *variable = literal* `$ X = HOME`
  - *variable1 = $variable2* `$ X = $HOME`
  - *variable1 = $(orden)* `$ X = $(echo $HOME)`
  - *variable1 = ‘orden’* `$ X = ‘echo $HOME)`
  - *variable1 = $(<archivo)* `$ X = $(<prueba)`

- **Eliminación de variables.** Podemos eliminar definiciones de variables mediante la opción `unset`. Eliminar una opción no es lo mismo que asignar el valor nulo al contenido de una variable
  - ***unset*** *variable*

KSH: variables y parámetros

- **Parámetros especiales.** El KSH hace uso de algunos parámetros de forma automática
  - **?** Código de salida de la última orden ejecutada
  - **$** Identificador de proceso del intérprete actual
  - **ERRNO** Código de error de la última llamada al sistema

- **Ejemplos:**
  - `$ print $?`
  - `$ print $$`
  - `$ cat tmp.out`
  - `tmp.out: No such file or directory`
  - `$ print $ERRNO`

KSH: variables y parámetros

- **Expansión de variables.** El KSH soporta el acceso y modificación del contenido de las variables mediante un conjunto de operadores
  - *$variable* `$ CA=hola; CA=$CAlifornia`
  - *${variable}* `$ CA=${CA}lifornia`
  - *${#variable}* `$ print ${#CA}`
  - *${variable:-literal}* `$ ${CA:-ab}`
  - *${variable:=literal}* `$ ${CA:=ab}`
  - *${variable:+literal}* `$ ${CA:+hola}`
  - *${variable:?}* `$ ${CA:?}`
  - *${variable:?literal}* `$ ${CA:?hola}`
  - *${variable#patrón}* `$ ${CA#ho}`
  - *${variable##patron}* `$ ${CA##la}`
  - *${variable%patrón}* `$ ${CA%ho}`
  - *${variable%%patrón}* `$ $(CA%%la}`

KSH: variables y parámetros

- **Vectores de variables.** El KSH soporta vectores de dimensión máxima de 512 elementos. Los índices de un vector empiezan en 0 y acaban en su dimensión menos uno.
  - *variable**[0]**=valor0 variable**[1]**=valor1 .... variable**[n]**=valorn*
  - ***set** -A variable valor0 valor1 ... Valorn*
  - ***typeset** variable**[0]**=valor0 variable**[1]**=valor1 .... variable**[n]**=valorn*

- **Acceso y modificación al contenido de vectores.**
  - ${vector}, $vector
  - ${vector[n]}
  - ${vector[*]}, ${vector[@]}
  - ${#vector[*]}, ${#vector[@]}
  - ${#vector[n]}

KSH: variables y parámetros

- **Atributos.** Cómo las variables ordinarias, el KSH permite establecer atributos de vectores y éstos son los mismos que los definidos para las variables ordinarias. Los atributos se aplican a todos los elementos del vector.
  - ***typeset*** *-atributo variable[0]=valor0 .... variable[n]=valorn*
  - ***typeset*** *-atributo vector*

- **Reasignación.** Mediante la misma sintaxis de definición y asignación de valores a vectores podemos reasignar los contenidos de los elementos de un vector. Además el KSH permite modificar el contenido de sólo algunos elementos de un vector
  - ***set +A*** *variable valo0 valor1 ...*

KSH: variables y parámetros

- **Las comillas simples**. Se utiliza para obviar el significado especial de los caracteres especiales ($,*,?,\,”,etc.) y realizar asignaciones que contienen espacios en blanco`$ CA = 'hola Juan'`
  - `$ echo '$HOME'`
- **Dobles comillas.** Igual que la comillas simples, excepto que no anulan el significado de los caracteres especiales: $, ` y \.
  - $ CA=“$HOME:`pwd`”
- **La Comillas: ``.** Se utilizan para asignar la salida producida por la ejecución de una orden
  - $ CA=`date`

KSH: 4. Control de tareas

- **Control de tareas.** El KSH posee un conjunto de facilidades que permiten el control de la ejecución de las tareas, para ello es necesario que esté activada la opción “monitor”
  - `$ set -o monitor`
  - `$ set -m`

- **¿Cómo saber si la opción monitor está activa?**
  - `$ set -o | grep monitor`

KSH: control de tareas

- **Órdenes para el control de tareas.**
  - **Ctl-z** Detener la tarea actual
  - **jobs** Visualiza el estado de todas las tareas
  - **jobs** %n Visualiza el estado de la tarea n
  - **jobs -p** Visualiza los identificadores de proceso de todas las tareas.
  - **bg** Pasa a segundo plano la tarea actual detenida
  - **bg** %n Pasa a segundo plano la tarea n detenida
  - **fg** Pasa a modo interactivo la tarea actual que está en segundo plano
  - **fg** %n Pasa a modo interactivo la tarea n que está en segundo plano

KSH: control de tareas

- **Órdenes para el control de tareas.**
  - **Kill** %n — Finaliza la tarea n
  - **kill -l** — Visualiza los nombres de señales válidas
  - **kill** -signal %n — Envía la señal especificada a la tarea n
  - **stty tostop** — Anula la salidas de las tareas en segundo plano
  - **stty -tostop** — Permite que las salidas de las tareas de segundo plano
  - **wait** — Espera por la finalización de la tarea actual en segundo plano
  - **wait** %n — Espera por la finalización de la tarea n en segundo plano

KSH: control de tareas

- **Formas de nombrar a las tareas.** Existen varías formas de referirnos a una tarea
  - **%n** Tarea n
  - **%+ , %%** Tarea actual
  - **%-** Tarea previa
  - **%cadena** Tarea cuyo nombre empieza por cadena
  - **%?** Tarea cuyo nombre contiene a cadena

  - **Ejemplo:**
  - `$ jobs -l`
  - `[3] + 466 Stopped split -5000 hugefile`
  - Entonces formas de nombrar a la tarea split: %3, %+,%%, 466, %split, %?spl

KSH: 5. Operaciones aritméticas

- **Especificación de operaciones aritméticas.** Cualquier operación de las soportadas por el KSH puede realizarse utilizando las siguientes sintaxis alternativas:
  - ***let*** *“operación-aritmética”*
  - ***((*** *operación_aritmética****))***

  - **Ejemplos:** las siguientes expresiones son equivalentes
    - $ let “X=X+1”
    - $ ((X=X+1)

  - **Declaración de variables enteras.** Mediante la cláusula **typset** podemos definir variables enteras
  - ***typeset*** *-i variable=valor-numérico*

KSH: operaciones aritméticas

- Formato de constantes numéricas.
  - *número*
  - *base#número*

  - **Ejemplos:**
  - `$ typeset -i2 X=5`
  - `$ typeset -i X=2#101`

KSH: operaciones aritméticas

- **Operadores aritméticos (por orden de precedencia).**
  - \- Evalúa el valor negativo de una expresión
  - ! Negación lógica
  - ~ Negación binaria
  - *, /, % Multiplicación, divisón, resto
  - +,- Suma, resta
  - <<, >> Desplazamiento a la izq., desplazamiento a la derch.
  - <=, < Menor que, menor que
  - >=, < Mayor o igual que, mayor que
  - == Igual que
  - != Distinto que
  - &And binario
  - ^ OR_exclusivo
  - | OR binario

KSH: operaciones aritméticas

- **Operadores aritméticos (por orden de precedencia).**
  - && Operador relacional AND
  - || Operador relacional OR
  - = Asignación
  - *=, /=, %= multiplación y asignación, división y asignación, resto y asignación
  - +=, -= Suma y asignación, resta y asignación
  - <<=, >>= Desplazamiento izq. Y asignación, desplazamiento derecha y asignación
  - &=, ^=, |= And binario y asignación, OR-exclusivo y asignación, OR binario y asignación
  - (...) Especificación de operación y precedencia

KSH: 6 El ambiente

- **Establecimiento de las variables de ambiente.** Cuando se invoca al KSH, éste con el objeto de establecer el ambiente del usuario que le ha invocado accede a los siguientes archivos:
  - /etc/.profile
  - $HOME/.profile

- **Algunas variables de ambiente:**
  - **CDPATH**
  - **COLUMNS**
  - **EDITOR**
  - **ENV**
  - **HISTFILE**
  - **HISTSIZE**
  - **HOME**
  - **IFS**
  - **MAIL**
  - **MAILCHECK**
  - **PATH**
  - **PS1, ..., PS4**
  - **SHELL**
  - **TERM**
  - **TMOUT**
  - **VISUAL**

KSH: el ambiente

- **Opciones de ambiente.** El KSH permite variar ciertas formas de su funcionamiento, para ello debemos utilizar la orden **set**. Por ejemplo:
  - **set -a, set -o allexport**
  - **set -o bgnice**
  - **set -o emacs, set -o gmacs**
  - **set -m, set -o monitor**
  - **set -n, set -o noexec**
  - **set -o noclobber**
  - **set -f, set -o noglob**
  - **set -o vi**
  - **set -x, set -o xtrace**

KSH: el ambiente

- **Alias.** Se trata de una facilidad soportada por el KSH mediante la cual podemos definir nuevos nombres de órdenes. Para usar esta facilidad disponemos de las siguientes órdenes:
  - ***alias*** *nuevo_nombre=orden*
  - ***alias***
  - ***unalias*** *nuevo_nombre*

- **Ejemplos:**
  - `$ alias p=printf`
  - `$ p Hola`
  - `Hola`
  - `$ unalias p`
  - `$ p Hola`

KSH: el ambiente

- **Subshells.** Se trata de procesos hijos del intérprete actual y que ejecutan a otro intérprete de órdenes. Éstos heredan las variables de entorno del intérprete padre y una vez finalizada su ejecución, las variable cosnervan el valor que tenían en el momento de la creación del subshell. Para generar un subshell se ha de utilizar la siguiente sintaxis:
  - ***(orden)***

- **Ejemplo:**
- `$ PRUEBA=Hola`
- `$ (printf $PRUEBA)`
- `$ (PRUEBA=Adios; printf $PRUEBA )`
- `$ printf PRUEBA`

KSH: 7. Escritura de programas

- **Programas escritos en KSH.** El KSH puede ejecutar archivos (“scripts”) que contienen un conjunto de órdenes. El archivo ha de tener permiso de ejecución.
  - `$ echo echo Mi primer programa > programa`
  - `$ chmod 0755 programa`
  - `$ programa`
  - `$ ksh programa`

- **Parámetros posicionales.** Son variables que se crean automáticamente y que contienen las distintas subcadenas que componen una orden.
  - `$ ptest HOLA JUAN`
- En este ejemplo, $0 es “ptest”, $1 es HOLA y $2 es JUAN
- Estas variables no admiten sentencias que impliquen la modificación de sus contenidos. La única manera de modificarlos es mediante la sentencia ***shift.***

KSH: escritura de programas

- **La orden [[...]]** Con ésta podemos evaluar expresiones condicionales e utilizar el resultado de dicha evaluación. Su sintaxis es:
  - ***[[expresión_condicional]]***

  - **La orden [[...]] para operar con cadenas de caracteres.**
    - **[[-n** cadena **]]**
    - **[[-o** opcion **]]**
    - **[[-z** cadena **]]**
    - **[[** cadena1 = cadena2 **]]**
    - **[[** cadena1 **!=** cadena2 **]]**
    - **[[** cadena1 **<** cadena2 **]]**
    - **[[** cadena1 **>** cadena2 **]]**
    - **[[** cadena1 **=** patron **]]**
    - **[[** cadena1 **!=** patron **]]**

  - **La orden [[...]] para operar con enteros.**
  - **[[** expr1 **-eq** expr2 **]]**
  - **[[** expr1 **-ne** expr2 **]]**
  - **[[** expr1 **-lq** expr2 **]]**
  - **[[** expr1 **-lt** expr2 **]]**
  - **[[** expr1 **-ge** expr2 **]]**
  - **[[** expr1 **-gt** expr2 **]]**

KSH: escritura de programas

- **La orden [[...]] para operar con cadenas de caracteres.**
  - **[[-a** fich **]]**
  - **[[-d** dir **]]**
  - **[[-f** fich **]]**
  - **[[-L** fich **]]**
  - **[[-O** fich **]]**
  - **[[-G** fich **]]**
  - **[[-r** fich **]]**
  - **[[-s** fich **]]**
  - **[[-S** fich **]]**
  - **[[-u** fich **]]**
  - **[[-w** fich **]]**
  - **[[-x** fich **]]**
  - **[[** fich1 **-ef** fich2 **]]**
  - **[[** fich1 **-nt** fich2 **]]**
  - **[[** fich1 **-ot** fich2 **]]**

- **Expresiones complejas con la orden [[...]]**
  - **[[** expr1 **&&** expr2 **]]**
  - **[[** expr1 **||** expr2 **]]**
  - **[[!**expr **]]**
  - **[[(**expr**)]]**

KSH: escritura de programas

- **Sentencia “case”.**
  - ***case*** *valor* ***in***
    - *patron1)* *orden*
    - *orden;;*
    - *patron2)* *orden*
    - *orden;;*
    - *.....*
    - *patronN)* *orden*
    - *orden;;*
  - ***esac***

KSH: escritura de programas

- **Ejemplo de sentencia case.**
  - `case` *`$1`* `in`
    - `-@([a-z]))` `echo carácter minuscula;;`
    - `-@([A-Z]))` `echo carácter mayuscula;;`
    - `@([1-9])([0-9]))` `echo numero entero`
  - `esac`

KSH: escritura de programas

- **Sentencia “for”.**
  - ***for*** *variable* ***in*** *palabra1 palabra2 .... palabraN*
  - ***do***
    - *orden*
  - ***done***

- **Variante:**
  - ***for*** *variable*
  - ***do***
    - *orden*
  - ***done***

KSH: escritura de programas

- **Ejemplo de sentencia “for”**
- `integer ITERACION=0`
- `for X in A B C D`
- `do`
  - `printf “$ITERACION $X”`
  - `((ITERACION+=1))`
- `done`

KSH: escritura de programas

- **Sentencia “if”.**
  - ***if*** *orden1*
  - ***then***
    - *orden*
  - ***else***
    - *orden*
  - ***fi***

- `USERS=$(who | wc l)`
- `if ((USERS == 1))`
- `then`
  - `echo Hay 1 usuario`
- `else`
  - `echo hay mas de 1 usuario`
- `fi`

KSH: escritura de programas

- **Sentencia “elif”.**
  - ***if*** *orden1*
  - ***then***
    - *orden*
  - ***elif*** *orden2*
  - ***then***
    - *orden*
  - ***elif*** *orden3*
  - ***then***
    - *orden*
  - ***else***
    - *orden*
  - ***fi***

KSH: escritura de programas

- **Ejemplo elif**
- `USERS=$(who | wc l)`
- `if ((USERS == 1))`
- `then`
  - `echo Hay 1 usuario`
- `elif ((USERS == 2))`
- `then`
  - `echo hay 2 usuarios`
- `else`
  - `echo hay 3 o mas usuarios`
- `fi`

KSH: escritura de programas

- **Sentecia “while”.**
  - ***while*** *orden1*
  - ***do***
    - *órdenes*
  - ***done***

- **Ejemplo**
  - `while (($# != 0))`
  - `do`
    - `printf $1`
    - `shift`
  - `done`

KSH: escritura de programas

- **Sentecia “until”.**
  - ***until*** *orden1*
  - ***do***
    - *órdenes*
  - ***done***

- **Ejemplo**
  - `until (($# == 0))`
  - `do`
    - `printf $1`
    - `shift`
  - `done`

KSH: escritura de programas

- **Sentecia “select”.**
  - ***select*** *variable* ***in*** *palabra1 palabra2 palabra3 ... palabran*
  - ***do***
    - *órdenes*
  - ***done***

- **Ejemplo**
  - `select i in op1 op2 op3`
  - `do`
    - `if [[ $i = op[1-3] ]]`
    - `then`
    - `printf”Ha seleccionado $REPLY: $i”`
    - `fi`
  - `done`

KSH: escritura de programas

- **Sentencia “continue”.** Con esta sentencia forzamos la finalización de la iteración actual en bucles for, while y until.

```
- while (($# != 0 ))
- do
    • if  [[ $1 = +([A-z]) ]]
    • then
        - printf “$1: argumento invalido”
        - shift
    • else
        - printf “$1: argumento invalido”
        - shift
        - continue
    • fi
- done
```

KSH: escritura de programas

- **Sentecia “break”.** Con esta sentencia forzamos la salida de un bucle for, while, until. Admite dos formas
  - ***break***
  - ***break*** *n*
  - `for i in 1 2 3`
  - `do`
    - `for j in 5 6`
    - `do`
      - `if (( i == 3 && j == 5 ))`
      - `then`
        - `» break 2`
      - `else`
        - `» printf “$i$j`
      - `fi`
    - `done`
  - `done`

8. Ejemplo 1: Diseñar un script que simule el comando cat

```
#!/bin/ksh
# Ejemplo1 de programa escrito para KSH: visualiza archivos
#
# Uso: kcat arch1 arch2 ... archn
#
# comprobacion de invocacion correcta
if (($# < 1))
then
    printf"Invocacion incorrecta"
printf "use kcat arch1 arch2 ... archn"
    exit 1
fi
```

Ejemplo 1: Diseñar un script que simule el comando cat

```
# Bucle para visualizar cada uno de los archivos
# Mientras queden archivos por visualizar
while (($# > 0))
do
    # El fichero existe?
    if [[ ! $1 ]]
    then
        printf"$1: archivo no existente o no accesible"
    else
        # Se abre el fichero y se hace canal de entrada
        exec 0<$1
        # Bucle de lectura de cada linea del archivo
        while read LINE
        do
            print $LINE
        done
    fi
    # Se prepara para procesar el siguiente archivo
    shift
done
```

Ejemplo 2: Diseñar un script que visualice el último componente de una ruta

```
#!/bin/ksh
# Ejemplo2 de programa escrito para KSH
# Visualiza el ultimo componente de un pathname
#
# Uso: kbasename pathname
#
# comprobacion de invocacion correcta

if (($#  == 0 || $# > 2))
then
    print "Invocacion incorrecta, use kbasename pathname"
    exit 1
fi
```

Ejemplo 2: Diseñar un script que visualice el último componente de una ruta

```
#
# Asignamos a BASE todo menos el utlimo componente
BASE=${1##*/}
#
# Comprobación si se especifico pathname
if (($# > 1))
then
    # Se especifico y se visualiza
    print ${BASE%$2}
else
    # No se especifico
    print $BASE
fi
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
#!/bin/ksh
# Ejemplo 3: programa escrito para KSH:
# Busca patrones de caracteres  en archivos
#
# Uso: kgrep [opciones] patron archivos
#
# Comprobacion de invocacion correcta
if (( $# < 2 ))
then
    print "Invocacion incorrecta: uso kgrep [opciones] patron archivos"
    exit 1
fi
#
# Se declaran variables
CFLAG=IFLAG=LFLAG=NFLAG=SFLAG=VFLAG=
Integer LNUM=0 COUNT=0 TOT_COUNT=0
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
# Desabilitamos generacion de nombres de archivos mediante # metacaracters
set -f
#
while true
do
    print "Argumento: $1"
     case $1 in
          -b* ) print "opcion b no soportada" ;;
          -c* ) CFLAG=1 ;;
          -i*  ) IFLAG=1 ;;
          -l*  ) LFLAG=1 ;;
          -n* ) NFLAG=1 ;;
          -s* ) SFLAG=1 ;;
          -v* ) VFLAG=1 ;;
          -*  ) print "$0: opcion desconocida $1"
                exit;;
           * ) PATRON=$1
                shift
                break ;;
        esac;
        shift
done
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
#
# Preparamos opciones de no visualizacion
#
NOPRINTF=$VFLAG$CFLAG$LFLAG
V_NOPRINTF=$CFLAG$LFLAG
#
# Preparamos patrones de comparacion  para mayusculas y minusculas
typeset -u MAYUS_PATRON=$PATRON
typeset -l MINUS_PATRON=$PATRON
#
# Bucle para procesar archivos
print "PATRON: $PATRON"
for FILE
do
    # Se hace FILE archivo de entrada del programa
    print "Procesando $FILE"
    exec 0<$FILE
    #
    # Se procesa cada linea de FILE
    while read -r LINE
    do
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
# Incrementamos contador de líneas
((LNUM+=1))
# Buscamos el patron en LINE
case $LINE in
      #  PATRON esta en LINE?
      *$PATRON* )
      # Si no se especificaron opciones de no # visualización
      if  [[ $VFLAG = "" ]]
       then
            [[ $NOPRINT = "" ]] && print -r  "${NFLAG:+LNUM:}$LINE"
            ((COUNT+=1))
      fi ;;
    #
    # Si se especifico opción -i
    # Entonces valen mayúsculas y minúsculas
    *$MAYUS_PATRON* | *$MINUS_PATRON* )
    if [[ $IFLAG != ""  ]]
    then
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
                if  [[ $VFLAG = "" ]]
                 then
                                [[$NOPRINT = ""]] && printf  -r
"${NFLAG:+LNUM:}$LINE"
                                ((COUNT+=1))
                fi
        else
                 if  [[ $VFLAG != "" ]]
                  then
                                [[ $V_NOPRINT = "" ]] && printf -r
"${NFLAG:+LNUM:}$LINE"
                                ((COUNT+=1))
                fi
        fi ;;
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
                # Si se especifico opcion -v
                # Entonces se visualizan las lineas que no contienen PATRON
                !(*$PATRON*) )
                        if  [[ $VFLAG != "" ]]
                        then
                                [[ $V_NOPRINT = "" ]] &&  printf -r
    "${NFLAG:+LNUM:}$LINE"
                                ((COUNT+=1))
                        fi  ;;
        esac
    done
#
# Si se especifico opcion -l
# Entonces visualizamos el nombre del archivo
if [[ $LFLAG != "" ]] && ((COUNT))
then
        print $FILE
fi
```

Ejemplo 3: Diseñar un script que busque patrones dentro de un archivo

```
# Incrementamos el contador de busquedas con éxito
TOT_COUNT+=COUNT
#
# Ponemos a cero el contador de lineas y de busquedas con éxito
LNUM=0
COUNT=0
done
#
# Si se especifico opcion -c
# Entonces visualizamos el numero total de comapraciones con éxito
if [[ $CFLAG != "" ]]
then
        print $TOT_COUNT
fi
#
# Se termina indicando procesamiento con éxito
exit 0
```