Asignatura: Entornos de programación
Lenguajes de guiones (*Scripting languages*)
Lenguaje de órdenes de Windows NT/2000/XP

En este tema se introducen los lenguajes de "guiones" en el contexto de los lenguajes de programación en general. Como caso concreto se introduce el lenguaje de órdenes de MS-DOS/Windows.

Los lenguajes de guiones no son un elemento propio de los entornos de programación. La razón de dedicarles un tema en esta asignatura es porque se necesita manejar algún lenguaje de este tipo para realizar parte de los ejercicios prácticos, y que los alumnos pueden no conocerlo suficientemente al matricularse de esta asignatura.

La descripción del lenguaje de órdenes de MS-DOS/Windows sólo muestra una parte de sus posibilidades. Los elementos que se mencionan son suficientes para realizar las prácticas de la asignatura.

1. Grupos de lenguajes de programación

Desde cierto punto de vista se pueden clasificar los lenguajes de programación en:

1. Lenguajes para programación de sistemas (en inglés: *System programming languages*)
   - En general son rigurosos, seguros y eficientes
   - Facilitan y/o exigen cierta disciplina de programación
   - Son adecuados para desarrollar programas complicados
   - Suelen tener buenos mecanismos para definir tipos de datos
   - Suelen procesarse mediante compiladores
2. Lenguajes de guiones o *scripts* (en inglés: *Scripting languages*)
   - En general son permisivos, menos seguros y menos eficientes que los anteriores
   - No exigen mucha disciplina para usarlos
   - Son adecuados para desarrollar programas sencillos
   - Suelen permitir el uso de variables no tipadas
   - Suelen procesarse mediante intérpretes

A veces es difícil determinar a qué categoría corresponde un lenguaje de programación en particular, ya que las características indicadas pueden darse en mayor o menor grado. A continuación se indican ejemplos de cada una de estas clases de lenguajes.

- Lenguajes para programación de sistemas
  - COBOL, FORTRAN, C/C++/C#
  - Pascal, Modula-2, Oberon, Ada
  - Java
  - Lisp, Haskell, Smalltalk, Eiffel

- Lenguajes de guiones (*scripts*)
  - Lenguajes de órdenes (*command languages*, *shell languages*)
  - Rexx, Tcl, Perl, Pyton, Ruby
  - VBScript, JavaScript

Dentro de los lenguajes de guiones, los lenguajes de órdenes constituyen un grupo particular. Podría decirse que los lenguajes de órdenes tienen como objetivo principal gobernar la ejecución de otros programas y automatizar así operaciones complejas combinando programas ya existentes. El resto de los lenguajes de guiones vienen a ser lenguajes de programación de uso general o especializado y que no requieren la existencia de otros programas para construir aplicaciones con ellos.

2. Lenguajes de órdenes

Se denominan también lenguajes de "*shell*". Poseen las siguientes características:

- Suelen estar asociados a algún Sistema Operativo
- Pueden usarse de modo interactivo y no interactivo
  - En el modo interactivo el usuario introduce las órdenes una a una, y se ejecutan inmediatamente
  - En el modo no interactivo se dispone de un guión de órdenes preparadas de antemano y permite la automatización de operaciones
- Ofrecen las siguientes funciones
  - Ejecutar programas
  - Usar/configurar dispositivos y servicios del S.O.
  - Manipular ficheros y grupos de ficheros
  - Elementos básicos de programación
    - Secuencias, alternativas y bucles
    - Variables
    - Subprogramas
    - Detección de errores, etc.

El nombre de "lenguajes de órdenes" se debe a que el guión (*script*) se plantea básicamente como una secuencia de órdenes que se van ejecutando sucesivamente. Históricamente surgieron como un mecanismo para gobernar el funcionamiento en "batch" (sin interacción con el usuario) de los primeros computadores y sistemas operativos. Con la aparición de UNIX se popularizó el uso interactivo de los computadores y el empleo de estructuras generales de programación en los lenguajes de órdenes. En UNIX se introdujo también el término "shell" para designar el intérprete de órdenes que permitía gobernar el funcionamiento del computador invocando los servicios del sistema operaivo.

Los lenguajes de órdenes no suelen tener un nombre propio. El nombre del lenguaje se corresponde con el nombre del intérprete o "shell" que lo procesa. Como ejemplos de intérpretes de lenguajes de órdenes se pueden citar los siguientes:

- Para los sistemas operativos MS-DOS o Windows:
  - COMMAND.COM (en MS-DOS y Win9x - 16 bits)
  - CMD.EXE (en WinNT/2000/XP - 32 bits)
- Para los sistemas operativos UNIX y Linux
  - sh (shell de Bourne, estándar)
  - csh, tcsh (C-shell, algo irregular)
  - ksh (shell de Korn para el UNIX de ATT)
  - bash (shell de Bourne actualizado, introducido en Linux)

3. El lenguaje de órdenes de WinNT

Está disponible en la familia de sistemas Windows NT/2000/XP. Es una extensión del lenguaje de órdenes de MS-DOS. Un guión de órdenes es un fichero de texto con la extensión `.BAT` o `.CMD`. El intérprete (*shell*) de este lenguaje es un programa llamado `cmd.exe`.

3.1 Sintaxis general

- Por defecto, una orden por línea
  - `orden argumentos...`
  - en el nombre de la orden no se distingue entre mayúsculas y minúsculas (ORDEN = Orden = orden)
- El intérprete distingue dos clases de órdenes
  - órdenes internas (`DIR`, `COPY`, ...) - las ejecuta el propio intérprete, en general no devuelven indicación de error
  - órdenes externas (programas ejecutables u otros guiones) - se ejecutan por separado, devuelven indicación de error
- Líneas de comentarios
  - `REM texto del comentario`
  - `:: texto del comentario`
- Etiquetas - pemiten programar saltos mediante la orden `GOTO`
  - `:nombre`
  - `GOTO nombre`
- Los argumentos actuales con los que se invoca una orden son cadenas de texto, con los siguientes formatos:
  - `valor` - sin espacios en blanco
  - `/xxx` - por convenio, se interpreta como una opción y no como un dato
  - `"valor ...\" ..."` - permite incluir espacios en blancos y comillas (") literales
- Órdenes compuestas - permiten escribir varias órdenes como una sola o una orden en varias líneas
  - `orden ... & orden ...` (secuencia de órdenes, se ejecutan todas)
  - `orden ... && orden ...` (secuencia condicional, sólo continúa mientras haya éxito)
  - `orden ... || orden ...` (secuencia condicional, sólo continúa mientras haya fallo)
  - `( órdenes ... )` (orden compuesta, puede ocupar varias líneas)
  - `^& ^|` (se usa ^ como escape para introducir un `&` o `|` literal)

3.2 Ejemplo: imprimir serie de Fibonacci

Aunque los lenguajes de órdenes no se usan habitalmente para realizar tareas de cálculo, el caso es que tienen posibilidad de hacerlo. Para tener una idea de cómo es un guión de órdenes aquí se presenta un ejemplo que permite imprimir los términos de la serie de Fibonacci hasta un límite dado como argumento. El significado de las órdenes utilizadas se irá viendo en los siguientes apartados. El guión estará almacenado en un fichero de texto llamado `fibonacci.bat`:

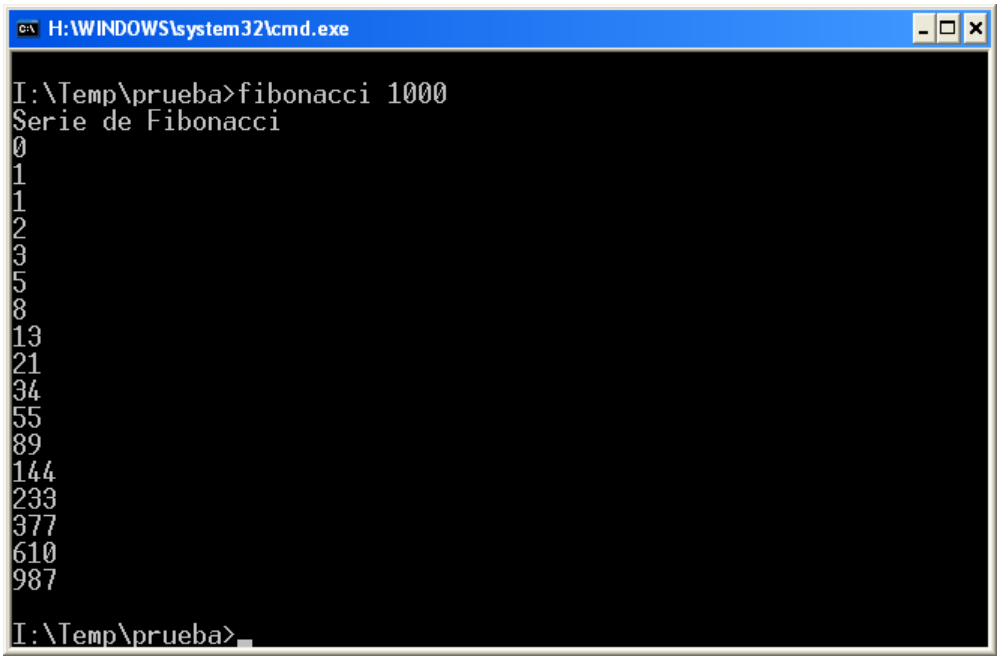
```
@echo off
rem Imprimir la serie de Fibonacci hasta un límite dado como argumento

:: comprobar si hay argumento
if  not .%1.==..  goto iniciar
  echo -- Faltan argumentos
  echo -- Uso:  %0  limite
  exit /B

:iniciar
echo Serie de Fibonacci
set x1=0
echo %x1%
set x2=1
echo %x2%

:repetir
  set x0=%x1%
  set x1=%x2%
  set /A x2=x0+x1
  if  %x2% GTR %1  exit /B
  echo %x2%
goto repetir
```

Para ejecutar el guión de ejemplo hay que abrir una ventana de órdenes (Microsoft la llama "Simbolo del sistema"), escribir en ella el nombre del fichero y pulsar la tecla Intro (*Enter*). Ejemplo:



3.3 Ayuda en línea

Se puede obtener información de ayuda acerca de las órdenes de CMD mediante:

- `HELP` (presenta la lista de las órdenes disponibles - órdenes internas y librería de órdenes externas de CMD)
- `HELP orden` (presenta una descripción de la orden indicada)
- `orden /?` (equivale a `HELP orden`)

3.4 Orden ECHO

En realidad son dos órdenes distintas con el mismo nombre:

- Escribir resultados
  - `ECHO resultado` (escribe el texto indicado)
  - `ECHO.` (escribe una línea en blanco - no hay espacio entre ECHO y el punto)
- Control de la traza de ejecución (escribir cada orden antes de ejecutarla)
  - `ECHO ON/OFF` (habilita o suprime la traza)
  - `ECHO` (sin argumentos, informa del estado de traza)
  - `@orden` (la @ al comienzo de la línea suprime la traza de la orden, aunque esté activo `ECHO ON`)

3.5 Orden EXIT

- Termina la ejecución del guión de órdenes
  - `EXIT` (termina la ejecución del intérprete de órdenes `cmd.exe`)
  - `EXIT /B` (termina la ejecución del guión en curso)
  - `EXIT /B nivelerror` (termina el guión en curso y devuelve el código de error indicado)

3.6 Ejecución de programas u otros guiones

- Orden de ejecución, implícita
  - `nombre argumentos...`
    - ejecuta el fichero `nombre`, o bien `nombre.bat/cmd/exe/com`
    - busca el fichero en el directorio actual y en el PATH
    - la ejecución termina con una indicación o nivel de error
    - si el fichero es un *script* (`.bat`, `.cmd`), no retorna
- Orden de ejecución, explícita
  - `CALL nombre argumentos...`
    - como la anterior, pero siempre retorna
- Uso de un *shell* secundario
  - `CMD /C orden ...`
    - invoca una copia del *shell* para que ejecute la orden y retorne
    - no devuelve indicación de error (el código de retorno es el del propio CMD, y no el de la orden ejecutada)
- Ejecución en una ventana separada
  - `START orden argumentos...`
    - invoca la orden en una nueva ventana
    - el guión en curso continúa inmediatamente sin esperar que termine la orden, por lo tanto no hay indicación de error
    - si la orden es otro *script*, su ventana continuará abierta al terminar
  - `START /WAIT orden argumentos...`
    - como la anterior, pero se espera a que termine la orden y se cierre su ventana antes de continuar el guión en curso

El código de retorno que sirve como indicador de nivel de error es un valor numérico no negativo que se designa simbólicamente como ERRORLEVEL. Por convenio se asume que el valor 0 indica que no hubo error, y valores distintos de cero indican que sí hubo error. El valor particular de ERRORLEVEL permite distinguir entre varios tipos de error.

3.7 Acceso a los argumentos o parámetros

Los argumentos de una orden se escriben a continuación del nombre de la orden, separando unos de otros por espacios en blanco. Dentro del guión se pueden usar los argumentos de la siguiente manera:

- `%1, %2, ... %9` designan los argumentos desde el primero al noveno
- los argumentos posteriores al noveno no pueden nombrarse directamente
- `%*` designa la serie de todos los argumentos , incluso si hay más de 9
- `%0` designa el nombre de la orden
- `%%` designa literalmente un carácter `%`

- `SHIFT`
  - desplaza los argumentos una posición hacia el comienzo (`%0 ← %1 ← %2 ← %3 ← %4 ...`, se pierde el antiguo `%0`)
  - opera con todos los argumentos, incluso si hay más de 9
  - no afecta a `%*`, que sigue designando la serie de argumentos originales (de 1 en adelante)

3.8 Redirección de los canales estándar de Entrada/Salida

- Canales estándar
  - entrada estándar (0)
  - salida estándar (1)
  - salida de error estándar (2)
- Redirección de la E/S
  - `orden < entrada.txt` (ejecuta la `orden` tomando la entrada del fichero `entrada.txt`)
  - `orden > salida.txt` (ejecuta la `orden` enviando la salida al fichero `salida.txt`)
  - `orden1 | orden2` (usa la salida de `orden1` como entrada para `orden2`)
  - `... >> salida.txt` (añade la salida al fichero `salida.txt`)
  - `... 2> salida.txt` (envía la salida de error al fichero `salida.txt`)
  - `... 2>&1` (envía la salida de error al canal de salida normal)
  - `... 1>&2` (envía la salida normal al canal de salida de error)
  - `orden1 < entrada.txt | orden2 | orden3 >> salida.txt` (las redirecciones pueden combinarse)

3.9 Entorno o contexto de ejecución (variables, directorio actual)

- Las órdenes se ejecutan en un contexto que incluye, entre otros elementos:
  - un conjunto de variables de entorno - cada variable tiene un nombre y un valor
  - un directorio actual, que se toma como punto de referencia para los ficheros a los que no se designa de manera absoluta
- Manejo de las variables de entorno
  - `SET variable=valor` (¡ojo!, sin dejar espacio)
  - `SET variable=` (suprime la variable)
  - `%variable%` (recupera el valor de la variable)
  - `SET prefijo` (presenta las variables cuyo nombre empieza por ese prefijo)
  - `SET` (presenta todas las variables)

- Cálculos aritméticos
  - `SET /A variable = expresión` (interpreta el valor a asignar como una expresión numérica, que se evalúa)
    - la expresión puede contener paréntesis, operadores, constantes numéricas enteras y nombres de variables de entorno
    - operadores: `+ - * /` ... etc.
    - las referencias a variables de entorno no necesitan los "%" (`var` equivale a `%var%`)
    - `variable += expresión` (se admiten operadores de asignación como en C: `+= -= *= /=` ...)
- Variable `PATH`
  - contiene una lista de directorios separados por punto y coma (`;`) en los que se buscan los programas o guiones a ejecutar
  - `PATH lista-de-directorios` (equivale a `SET PATH=lista-de-directorios`)
  - `PATH` (presenta el valor de `PATH` - equivale a `ECHO %PATH%`)
- Cambio de directorio, permanente
  - `CD directorio` (cambia el directorio actual)
  - `x:` (cambia la unidad actual, cada unidad de disco tiene su propio directorio actual)
  - `CD` (presenta el valor del directorio actual)
- Cambio de directorio, temporal
  - `PUSHD directorio` (cambia el directorio actual)
  - `POPD` (restaura el anterior directorio actual)
  - `PUSHD/POPD` pueden anidarse
- Entorno de ámbito limitado
  - `SETLOCAL` (inicia ámbito local)
  - `SET/CD/PATH` (cambios locales)
  - fin del guión (termina el ámbito local)
  - `ENDLOCAL` (termina el ámbito local)
  - `SETLOCAL/ENDLOCAL` pueden anidarse

3.10 Estructuras de control: órdenes condicionales

En un guión de órdenes se pueden programar acciones condicionales mediante los siguientes mecanismos:

- Estructuras de control: IF-THEN-ELSE
  - `IF condición orden`
  - `IF condición ( orden1 ) ELSE orden2`

- Las condiciones pueden escribirse como:
  - `valor1 == valor2` (comparación de valores de texto)
  - `valor1 operador valor2` (comparación de valores numéricos enteros o valores de texto)
    - el operador puede ser `EQU, NEQ, LSS, LEQ, GTR, GEQ`
  - `/I valor1 operador valor2` (comparar sin distinguir entre mayúsculas y minúsculas)
  - `EXIST nombre` (existe un fichero o directorio con ese nombre)
  - `EXIST directorio\nul` (existe el `directorio`)
  - `ERRORLEVEL valor` (nivel-de-error ≥ valor)
  - `DEFINED variable` (existe una variable de entorno con ese nombre)
  - `NOT condición` (negación de la `condición`)

También se pueden programar acciones condicionales mediantes saltos. El siguiente esquema de código equivale a un IF-THEN-ELSE

```
IF NOT condición GOTO es-falso
   ... acción si se cumple la condición (then) ...
   GOTO fin-condición
:es-falso
   ... acción si no se cumple la condición (else) ...
:fin-condición
```

3.11 Estructuras de control: bucles

La orden FOR permite programar bucles controlados por una variable de índice:

- `FOR %%v IN (lista-de-valores) DO acción-con-%%v`
  - el nombre de la variable es una letra
  - repite la acción por cada valor de la lista, cambiando la variable
  - los valores de la lista se escriben separados por espacios en blanco
  - los valores de la lista pueden contener comodines (*wildcards*) - se repite por cada fichero que se ajuste al patrón
  - si la acción es un nombre de guión el bucle no se repite, ya que no hay retorno
- `FOR %%v IN (lista-de-valores) DO CALL acción...`
  - esta es la forma correcta de repetir la ejecución de un guión - evita salir del bucle prematuramente
- `FOR /D %%v IN (lista) DO ...`
  `FOR /R %%v IN (lista) DO ...`
  - repite para cada directorio (/D) o árbol de subdirectorios (/R)
- `FOR /L %%v IN (inicio, paso, fin) DO ...`
  - bucle con contador numérico

Otras formas de bucle (WHILE, bucles indefinidos, ...) se deben programar mediante saltos. El siguiente esquema de código equivale a un bucle WHILE:

```
:inicio-bucle
   IF NOT condición GOTO fin-bucle
   ... acción del bucle
   GOTO inicio-bucle
:fin-bucle
```

3.12 Manejo de ficheros

Las operaciones con ficheros o grupos de ficheros son uno de los objetivos fundamentales de los lenguajes de órdenes. Los ficheros tienen un nombre y una extensión (que por convenio designa el tipo del fichero), y se organizan en directorios, que puede estar jerarquizados. Los directorios tienen nombre y no suelen tener extensión (si se les pone extensión, no significa nada). En MS-DOS y Windows cada unidad de disco se designa por una letra, y tiene su propio sistema de ficheros.

- La notación para designar ficheros y grupos de ficheros es:
  - `unidad:\directorio\...\nombre.ext` (nombre completo de un fichero o directorio)
  - si se omite la unidad o los primeros directorios se toma como referencia la unidad o directorio actual
  - `.` (referencia al directorio actual)
  - `..` (referencia al directorio padre)
  - `?*` (comodines o patrones: ? = cualquier carácter; * = cualquier texto)
  - si un nombre de fichero o directorio contiene espacios en blanco (no se recomienda), entonces debe escribirse entre comillas
- Listar los ficheros o directorios
  - `DIR patrón` (presenta todos los ficheros y directorios)
  - `DIR` (equivale a `DIR *`)
  - `DIR /opciones...` (especifica cómo se organiza la lista - ociones: `/S /P /W /B /O` ... etc.)
- Listar el contenido de un fichero de texto
  - `TYPE fichero(s)` (admite patrones)
- Cambiar el nombre y/o la extensión de un fichero o directorio
  - `RENAME fichero nuevo-nombre` (admite patrones)
  - `REN` (abreviatura de RENAME)
  - `nombre.*, *.ext` (como nuevo patrón, conserva la extensión o el nombre originales)
- Eliminar ficheros
  - `DEL patrón` (**¡ojo!**: `DEL *` elimina todo)

- Copiar o mover ficheros
  - `COPY origen destino` (copia sólo ficheros - admite patrones)
    - (destino puede ser un fichero o un directorio)
  - `COPY origen` (por defecto, destino = `.`)
  - `XCOPY origen destino` (copia ficheros o directorios)
  - `XCOPY origen` (por defecto, destino = `.`)
  - `MOVE origen destino` (mueve ficheros o directorios)
  - `MOVE origen` (destino = .)
- Crear y eliminar directorios
  - `MKDIR directorio` (crea un directorio)
  - `RMDIR directorio` (elimina el directorio, que debe estar vacío)
  - `MD, RD` (abreviaturas de MKDIR y RMDIR, respectivamente)

3.13 "Expansión" de parámetros y variables

Con mucha frecuencia los parámetros de un guión o las variables de índice de un FOR contienen referencias a ficheros o directorios. El lenguaje de órdenes permite extraer partes seleccionadas (directorio, nombre, extensión,...) y construir nuevas designaciones a partir de otras. Microsoft denomina "expansión de parámetros" a este mecanismo.

- Extracción de partes de una referencia a fichero o directorio (u otra información del mismo):
  - `%~`*`XXN`* (*`XX`*: una o más letras de código, *`N`*: número del parámetro [1..9] o letra de la variable índice de FOR)
    - lista de códigos:
      - f - nombre completo
      - d - unidad de disco
      - p - directorio(s)
      - n - sólo el nombre
      - x - extensión
      - a - atributos del fichero
      - t - fecha y hora del fichero
      - z - tamaño del fichero
    - si se aplica a una variable índice, su letra de nombre no debe coincidir con ningún código posible
    - si se combinan varias letras de código los valores se reordenan de manera prefijada
  - `%~`*`N`* (si se omiten las letras de código, simplemente se quitan las comillas, si las hubiera)

- Búsqueda en la lista de directorios contenida en el PATH u otra variable de entorno
  - `%~$PATH:N` (devuelve lo primero que encuentre que se ajuste al valor del parámetro)
    - puede usarse cualquier otra variable de entorno, no sólo PATH
- Expansión de variables (sustitución de texto)
  - `%var:antes=después%` (recupera el valor de la variable con las sustituciones indicadas)
    - no modifica el valor interno de la variable

3.14 Ayuda para la interacción

Los lenguajes de órdenes no suelen disponer de mecanismos adecuados para construir programas interactivos. De todas maneras ofrecen ciertas facilidades para la ejecución en modo interactivo, por consola, dando manualmente las órdenes una a una.

- `CLS` (borra la pantalla o ventana de órdenes)
- `COLOR xy` (cambia los colores de la pantalla: x = color de fondo, y = color del texto)
  - `xy` son dos dígitos hexadecimales
  - para consultar los códigos de color, dar `COLOR /?`
- `COLOR` (sin parámetro, restaura los colores por defecto - texto blanco en fondo negro)
- `TITLE título` (establece el texto que aparece en la barra de título de la ventana de órdenes)
- `PROMPT texto-guía` (establece el texto de petición de nueva orden)
  - el texto guía puede contener texto nomal combinado con códigos especiales:
    - `$P` (directorio actual)
    - `$G` (signo mayor que >)
    - `$N` (letra de la unidad de disco actual)
    - `$$` (literalmente un $)
    - para consultar otros códigos, dar `PROMPT /?`
- `PROMPT` (sin parámetros, restaura el valor por defecto `$P$G`)
- `PAUSE` (presenta un mensaje de pausa y espera hasta que se pulsa una tecla)
- `MORE` (copia la entrada estándar a la salida estándar, haciendo pausas cada P líneas - P = líneas de la pantalla)
- `MORE fichero` (toma la entrada del fichero indicado, similar a `TYPE fichero` pero con pausas)
- `SET /P variable=texto-guía` (se presenta el texto-guía y se asigna a la variable el valor que se introduzca por consola)
  - si se introduce un valor vacío, la variable no cambia