Escuela de Verano Complutense 2004
Bioinformática y biología computacional

Procedimientos elementales de manejo de ordenadores bajo LINUX

Daniel Mozos Muñoz
mozos@dacya.ucm.es

Bibliografía

- Bibliografía:
  - “Linux”, J. Tackett, S. Burnett, Prentice Hall, 2000
  - “Developing bioinformatics computing skills”, C. Gibas, P. Jambeck, O’Reilly 2001.
  - “UNIX. Programación avanzada”, F.M. Márquez, Ra-ma, 1996

- Sitios web:
  - www.linux.com
  - www.linux.org

- Man

ndice

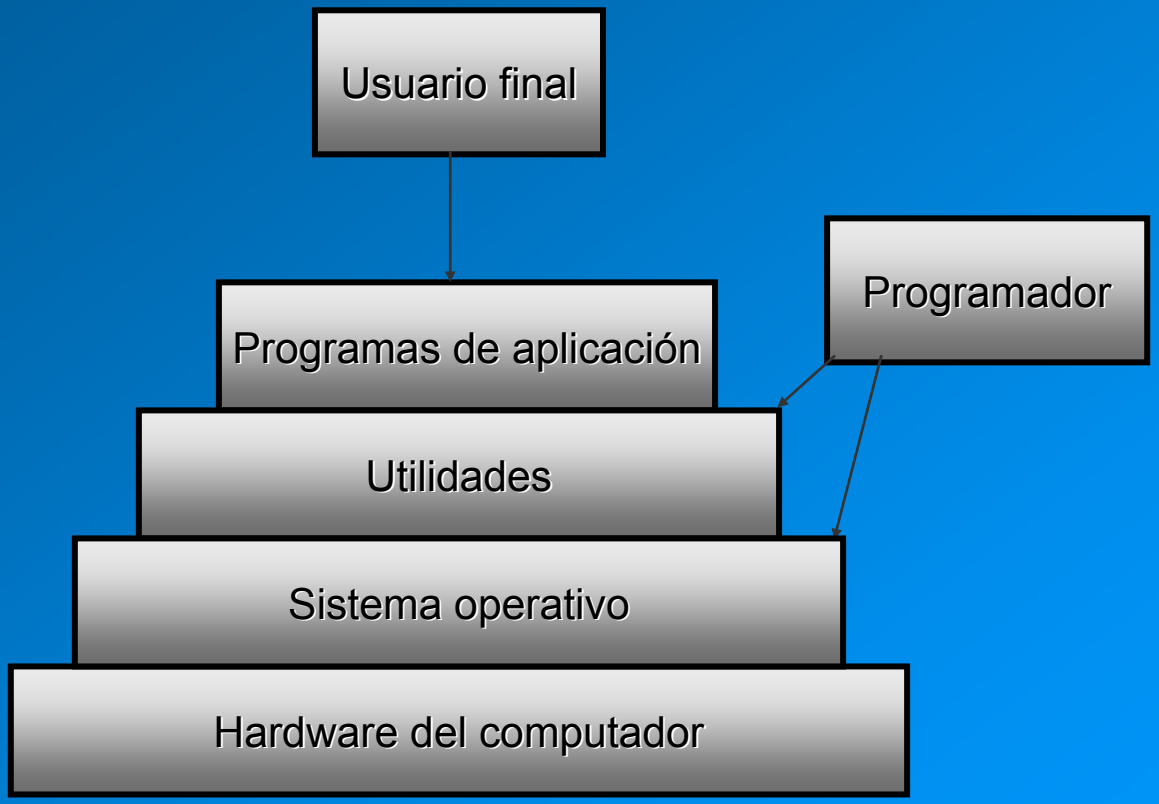
- Introducción
- ¿Qué es un sistema operativo?
- UNIX/LINUX
- Tipos de ficheros
- Características del sistema de ficheros
- Procesos
- El shell
- Seguridad

ntroducción

- Informática para bioinformáticos. Conocimientos útiles:
  - ¿Qué es un Computador?
  - ¿Qué es un Sistema Operativo?
  - ¿Qué es un lenguaje de programación?
  - ¿Qué es una red?
  - ¿Qué es una Base de datos?

ntroducción

- **¿Qué es un Computador?. Niveles de uso**



¿Qué es un Sistema Operativo?

- **Servicios del S.O.:**
  - **Creación de programas**. Disponibilidad de herramientas de ayuda a la programación: editores, depuradores.
  - **Ejecución de programas.** Si no existiese el S.O. el usuario debería encargarse de todo lo necesario para poder ejecutar un programa.
  - **Acceso a los dispositivos de E/S.** Cada dispositivo de E/S tiene características específicas que el S.O. conoce y evita que el usuario deba conocerlas.
  - **Acceso controlado a los archivos.** Evita que usuarios sin permiso accedan a ciertos archivos (permisos .r, .w). El S.O. conoce el formato que deben tener los archivos en los distintos tipos de medio de almacenamiento.

¿Qué es un Sistema Operativo?

- **Servicios del S.O.:**
  - **Acceso al sistema**. El S.O. controla qué usuarios pueden acceder al sistema y a cada uno de los recursos del mismo.
  - **Detección y respuesta a errores**. El S.O. se encarga de detectar los errores que se produzcan en el hardware y de tomar las decisiones adecuadas para que tengan el menor impacto negativo sobre el sistema.
  - **Contabilidad**. El S.O. debe tener estadísticas de utilización de los diversos recursos y supervisar los parámetros de rendimiento. Esto permitirá prever mejoras futuras y facturar a cada usuario.
  - **Asignación de recursos**. Debe decidir quién usa cada elemento del sistema (CPU, memoria, discos) en cada momento.

Unix/Linux

- **¿Por qué usar Unix o Linux en bioinformática?**
  - Tradición
  - Robusted
  - Seguridad
  - Sistema multiusuario
  - Disponibilidad de múltiples herramientas
  - Gratuidad de muchas herramientas

Unix/Linux

- **¿Qué necesito saber sobre Unix o Linux?**
  - Tipos de ficheros
  - Características del sistema de ficheros
  - Órdenes para trabajar con ficheros y directorios
  - Permisos
  - Procesos
  - El shell
  - Órdenes habituales del shell
  - Shell scripts
  - Seguridad

Tipos de ficheros

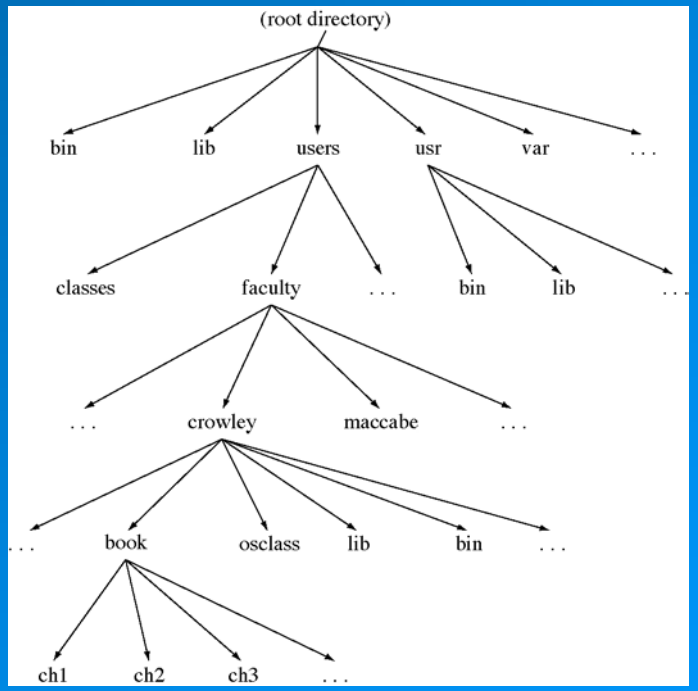
- Ficheros ordinarios
  - Pueden contener tanto datos como programas.
  - Tienen asociados permisos de acceso y ejecución.
- Directorios
  - Internamente similares a ficheros ordinarios, pero sirven para organizarlos sistemática y jerárquicamente.
  - Ej: /home/users/programas/pepe.c
  - Subdirectorios típicos en un sistema UNIX:
    - bin, dev, usr...
- Ficheros especiales o de dispositivo
  - Permiten la comunicación fácil con los dispositivos periféricos.
- Pipes
  - Permiten comunicar procesos entre sí.

Características del sistema de ficheros

- **El bloque de arranque** (boot). Puede contener el código de arranque, un pequeño programa que se encarga de buscar el sistema operativo y cargarlo en memoria para inicializarlo.
- **El superbloque**. Describe el estado de un sistema de ficheros. Contiene información acerca de su tamaño, nº total de ficheros que puede contener, espacio que queda libre, etc.
- **Lista de nodos índice** (nodos-i). Contiene una entrada por cada fichero, donde se guarda una descripción del mismo (situación del fichero en el disco, propietario, permisos de acceso, fecha de actualización, etc)
- **Bloques de datos**. En ellos se encuentra el contenido de los ficheros a los que se refiere la lista de nodos-i. Cada bloque se asigna totalmente a un fichero, aunque no lo ocupe totalmente.

Características del sistema de ficheros

- Organización jerárquica
  - Directorio raíz /
  - Permisos: rwx
    - Permisos para el usuario, el grupo, u otros.



Características del sistema de ficheros

- Particiones del disco
  - Divisiones del disco independientes unas de otras.
  - Posibilidad de tener varios S.O.s
  - Posibilidad de tener un sistema de ficheros independiente en cada partición, o uno ocupando varias particiones.

- Montaje y desmontaje de un sistema de ficheros
  - Permite ver a otro sistema de ficheros como parte de la jerarquía del sistema de ficheros en uso.
  - $ /etc/**mount** sistema_de_ficheros directorio
  - $ /etc/**unmount**

Características del sistema de ficheros

- Monitorización del sistema de ficheros
  - $ **df** [opciones] [sistema_de _ficheros]
  - Informa sobre el nivel de ocupación del sistema de ficheros
    - -t informa sobre el total de bloques ocupados.
    - -f informa sobre el total de bloques que hay en la lista de bloques libres.
    - -v informa sobre el porcentaje de bloques ocupados, así como el nº de bloques usados y libres
    - -i informa sobre el porcentaje de nodos-i ocupados, usados y libres.
  - $ **quod** [opciones] [sistema_de _ficheros]
    - Informa sobre el nivel de ocupación del disco que corresponde a cada usuario del sistema
  - $ **du** [opciones] [path_name]
    - Informa sobre el uso de disco que se realiza en un nodo de la jerarquía de directorios

Características del sistema de ficheros

- **Órdenes para trabajar con ficheros y directorios:**
  - ¿Cómo saber qué hace una orden y qué opciones tiene?
    - **man** nombre_orden
  - ¿Cómo saber en qué subdirectorio me encuentro?
    - **pwd**
      - » / users/classes
  - ¿Cómo ir a otro directorio?
    - **cd [path_name]**
      - » cd /users/faculty/crowley/bin
    - Caminos relativos:
      - » .. Indica el directorio padre
      - » . Indica el directorio actual
      - » Si no se indica nada, la orden cd, cambia al directorio de trabajo del usuario.
  - ¿Cómo ver qué archivos hay en un directorio?
    - **ls**
      - » Fi1 ej2 ej3

Características del sistema de ficheros

- **Órdenes para trabajar con ficheros y directorios:**
  - ¿Cómo realizar una copia de un fichero?
    - **cp** doc doc.bk
  - ¿Cómo realizar una copia de todos los ficheros de un directorio en otro directorio?
    - **cp** -r directorio_origen directorio_destino
  - ¿Cómo se borra un fichero?
    - **rm** docu.txt
    - **rm** -r nombredirectorio
  - Vinculación de ficheros
    - **ln** pepe /users/mozos/juan
    - Hace accesible el archivo pepe del directorio actual como archivo juan en el directorio /users/mozos
  - Creación y eliminación de directorios
    - **mkdir** nombre_directorio
    - **rmdir** nombre_directorio

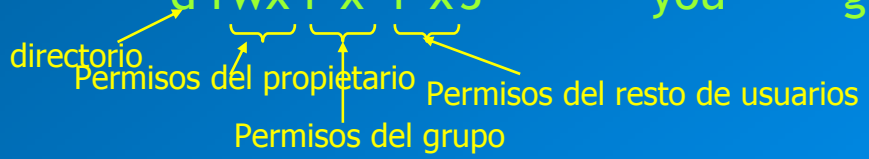
Características del sistema de ficheros

- **Órdenes para trabajar con ficheros y directorios:**
  - ¿Cómo ver el contenido de un fichero?
    - **cat** doc.bk
    - **more** doc.bk Permite ver el contenido de doc.bk formateado en páginas.
  - ¿Cómo imprimir un fichero?
    - **lp -p** impresora_destino nombre_fichero
      - » Imprime el contenido del fichero, sin formato.
    - **pr** nombre_ficheros
      - » Añade un cabecero a las páginas de un archivo (fecha, nombre del fichero, y nº de páginas)
    - **pr** nombre_fichero | **lp**
      - » Imprime nombre_fichero con cabecero

Características del sistema de ficheros

- **Permisos**
  - Unix divide a los usuarios de cualquier fichero en tres tipos, y asocia permisos independientes para cada uno de ellos:
    - El propietario del fichero
    - El grupo al que pertenece el propietario
    - Los otros usuarios
  - Ej:

```
$ ls -l memoria
d rwx r-x r-x 3 you group1 36 Apr 1 21:27 memoria
```



- Cambio de permisos mediante la orden **chmod**

```
$ chmod 700 memoria
$ ls -l memoria
d rwx --- --- 3 you group1 36 Apr 1 21:27 memoria
```

Procesos

- Un proceso es un programa en ejecución.
- En un sistema multitarea simultáneamente existen varios procesos en ejecución
- ¿Cómo saber qué procesos están activos?
  - **ps**

PID	TTY	TIME	COMMAND
3211	term/41	0:05	ksh
12345	term/41	0:01	ps
23455	term/41	1:20	ksh
9634	term/41	2:12	vi

- ¿Cómo eliminar un proceso que está en ejecución?
  - **kill** 9634
  - **Kill -9** 23455 elimina incondicionalmente un proceso.

- &, colocado al final de una orden, hace que el proceso se ejecute en modo subordinado y que se pueda seguir usando el sistema en modo interactivo.
- Planificación de procesos: at, batch
- Demonios: Procesos que no están conectados a un terminal, permiten realizar tareas útiles para el usuario o el sistema en modo subordinado.

El shell

- Es la parte del sistema con la que nos relacionamos y que gestiona los recursos del sistema.
- El indicador habitual de un shell UNIX es $.
- Su funcionamiento es:
  1. El shell solicita una orden mostrando su indicador $.
  2. Se teclea una orden.
  3. El shell procesa la línea de órdenes para determinar las acciones a realizar.
  4. Cuando finaliza, el shell vuelve al paso 1.

- Agrupación de órdenes:
  - Varias órdenes separadas por ; se ejecutan secuencialmente
  - Uso de <, > y |
  - Guiones de órdenes.

El shell

- **Órdenes habituales del shell**
  - Órdenes para manipular ficheros
    - Editores de texto:
      - »**vi**, **emacs**, **gedit**.
    - **split** permite dividir un fichero en otros más pequeños de un número dado de líneas.
    - **cut** permite mostrar partes de cada línea de fichero de entrada.
    - **paste -[opciones]** ficheros
      - »permite combinar varios ficheros en uno solo, uniendo las líneas de cada fichero, una por una, en una nueva línea del fichero final.
    - **join** -[opciones] fichero1, fichero2
      - »permite mezclar dos ficheros basándose en los contenidos. Se supone que los ficheros tienen carácter tabular y que están ordenados del mismo modo.
    - **sort** permite ordenar un fichero o varios y mezclarlos finalmente en uno único.
    - **grep** [patron] [ficheros]
      - »Busca el patrón en todos los ficheros indicados e imprime cada una de las líneas que lo contengan

El shell

- **Órdenes habituales del shell**
  - Órdenes para analizar ficheros
    - **cmp**
      - »Indica si dos ficheros son idénticos
    - **diff**
      - »Imprime aquellas líneas de dos ficheros que son diferentes
    - **wc** -[opciones] nombre_fichero(s)
      - »Permite contar cosas dentro de uno o varios ficheros, por defecto cuenta el número de líneas, palabras y caracteres.

      ```
      $ wc pepe.txt
       27 102 456 pepe.txt
      ```
    - **who**
      - »Indica qué usuarios están conectados en este momento
    - **cal**
      - »Imprime el calendario de cualquier mes
    - **date**
      - »Imprime la fecha y hora actuales

El shell

- **Guiones (scripts) de shell**
  - Permiten automatizar tareas, agrupando órdenes del shell en un fichero que podrá ejecutarse posteriormente de manera secuencial. Las distintas órdenes se separan con ;
  - Evita tener que teclear conjuntos de órdenes que se repiten regularmente y tener que esperar a que termine una orden para lanzar la siguiente.
  - Puede utilizar un lenguaje de programación similar a C, para programar bucles y condicionales.

Seguridad

- **Seguridad**
  - La seguridad es relativa
  - Usuarios
  - Passwords
  - Permisos
  - Cifrado de archivos

Escuela de Verano Complutense 2004
Bioinformática y biología computacional

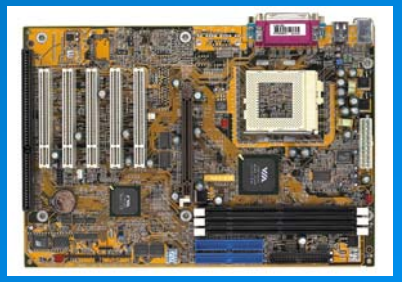
Principios básicos de programación

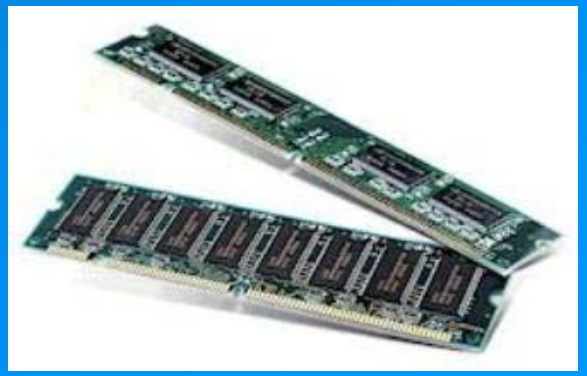
Daniel Mozos Muñoz
mozos@dacya.ucm.es

ndice

- Estructura de un computador
- ¿Qué es un programa?
- Desarrollo y ejecución de un programa
- Datos y operaciones sobre datos
- Control de flujo de un programa
- Entrada/salida de un programa
- Funciones
- Estrategias de programación

Estructura de un computador

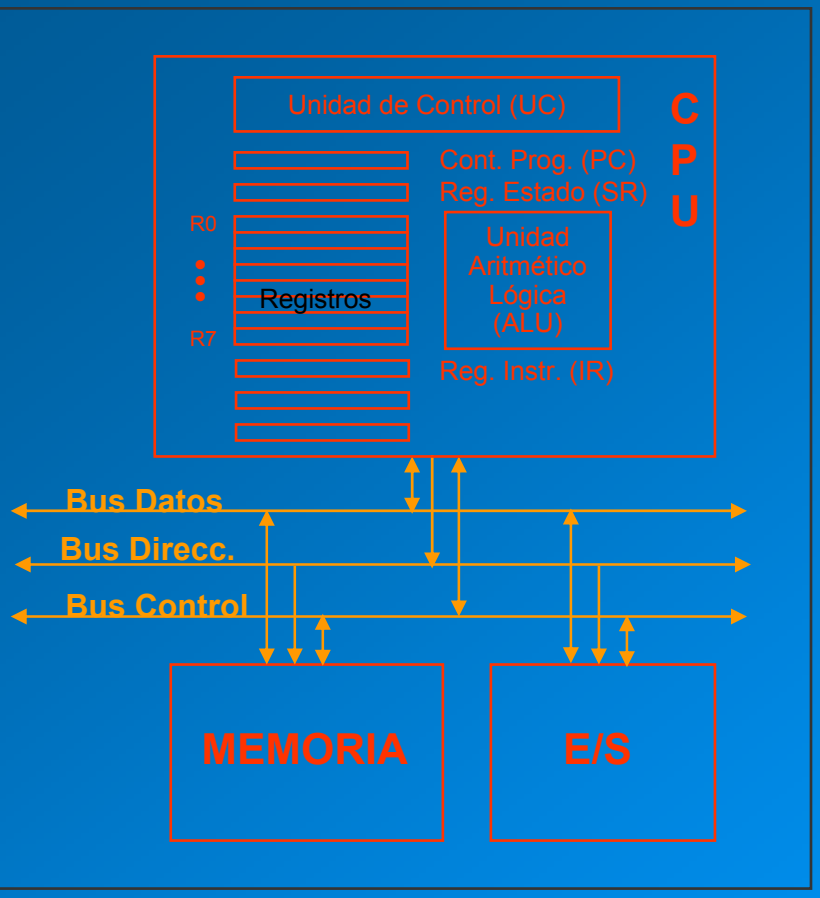




Estructura de un computador

- **Estructura de un computador**

Esquema



Módulos básicos

CPU (Unidad Central de Proceso)
- Ejecuta instrucciones

Unidad de Memoria
- Almacena las instrucciones y los datos

Unidad de E/S
- Transfiere información entre el computador y los dispositivos periféricos

Elementos de interconexión: BUSES

Bus de datos
- Para transferencia de datos entre la CPU y memoria o E/S

Bus de direcciones
- Para especificar la dirección de memoria o la dirección del registro de E/S

Bus de control
- Señales de control de la transferencia (reloj, lectura/escritura, etc.)

Qué es un programa?

- Un programa es una secuencia de órdenes que la electrónica del computador sabe interpretar.
- La interpretación de estas órdenes es secuencial.
- El computador sólo entiende niveles de voltaje.
- Lenguajes de programación:
  - Evitan tener que escribir los programas mediante secuencias de 0s y 1s.
  - Implican la existencia de compiladores y enlazadores
  - Todos suelen tener unos componentes básicos
- Todo lenguaje de programación viene descrito por:
  - Un léxico. Conjunto de palabras que pueden usarse en un programa
    - Palabras clave
    - Nombres de variables, constantes, subprogramas, etc
  - Una sintaxis. Formas válidas para mezclar los elementos léxicos
  - Una semántica. Significado de las expresiones definidas con una sintaxis correcta.

Desarrollo y ejecución de un programa

- Planteamiento del problema
- Diseño del algoritmo
- Representación de los datos a utilizar

} Sobre papel

- Codificación del programa
- Compilación
- Depuración

} Sobre computador

Desarrollo y ejecución de un programa

- Ejemplo:
  - Planteamiento del problema
    - Realizar un programa que pida al usuario que introduzca la nota de todos los alumnos de una clase y calcule la media. Si la media está por debajo de 4 el programa indicará por pantalla que los resultados son malos, si está por encima de 7 indicará que los resultados son buenos.
  - Diseño del algoritmo

Versión 1

```
Leer nota alumno 1
Guardar nota alumno 1
Acumular nota
Leer nota alumno 2
Guardar nota alumno 2
Acumular nota
…
Leer nota alumno n
Guardar nota alumno n
Acumular nota
Calcular media
Si media < 4 imprimir mensaje negativo
Si media > 7 imprimir mensaje positivo
```

Versión 2

```
Para cada alumno i {
        Leer nota alumno i
        Guardar nota alumno i
        Acumular nota
}
Calcular media
Si media < 4 imprimir mensaje negativo
Si media > 7 imprimir mensaje positivo
```

Desarrollo y ejecución de un programa

- Representación de los datos a utilizar
  - Definición de datos como variables o constantes.
  - Tamaño y tipo de datos
    - Influye sobre la memoria utilizada
  - Datos estructurados
    - Vectores. Conjunto de datos relacionados mediante índices.

  - Ejemplo:
    - Vector que almacena todas las notas: nota_alumno[i]
    - Variable que almacena la media: media.

Desarrollo y ejecución de un programa

❑ Codificación

- Supone elegir un lenguaje de programación (PERL)

```
# Ejemplo: Calcula la media de las notas de 40 alumnos

$media = 0;

for ($i = 0; $i< 40; $i++) {
            print "Introduce la nota del alumno $i: \n";
            $nota_alumno[$i] = <STDIN>;
            $media = $media + $nota_alumno[$i];
            }

$media = $media / 40;
print "Media = $media\n";

if ($media < 4) {print "Los resultados del examen han sido malos!!\n";}
if ($media > 7) {print "Los resultados del examen han sido buenos !!!\n";}

exit;
```

Desarrollo y ejecución de un programa

- Compilación y enlazado
  - La compilación es el proceso de pasar las órdenes en un lenguaje de alto nivel a lenguaje máquina.
  - Las órdenes más habituales del lenguaje máquina son:
    - Movimientos de datos entre una posición de memoria y un registro
    - Suma, resta, multiplciación o división de datos en registros
    - Desplazamientos
    - Comprobaciones de bifurcación: si el contenido de un registro es igual a un valor entonces bifurca
    - Saltos incondicionales.
  - El enlazado permite enlazar programas compilados independientemente determinando las posiciones relativas de los datos en memoria.

Desarrollo y ejecución de un programa

- Depurado
  - Posibles errores:
    - Al compilar pueden aparecer errores de sintaxis. Hay que corregirlos antes de poder ejecutar el programa.
    - Al ejecutar pueden presentarse errores porque el programa no funcione como se esperaba:
      - Uso de depuradores para buscar el error.
        - Permiten ejecución instrucción a instrucción.
        - Añadir breakpoints en el programa.
        - Visualizar el contenido de una variable en un determinado momento de la ejecución.

Escuela de Verano Complutense 2004
Bioinformática y biología computacional

Programación en PERL

Daniel Mozos Muñoz
mozos@dacya.ucm.es

Bibliografía

- Bibliografía:
  - “PERL 5. How-to”. M. Glover, A. Humphreys, E. Weiss, Waite Group Press 1996.
  - “PERL 5 al descubierto”, K. Husain, Prentice Hall, 1998
  - “Programming PERL”, L. Wall, T. Christiansen, R.L. Schwartz; O’Reilly, 1996
  - “Beginning PERL for bioinformatics”, James Tisdall, O’Reilly, October 2001

- Sitios web:
  - www.perl.com
  - www.perlreference.com

- Man
  - perl
  - perldata, perlop, perlre, perlfunc, etc.

ndice

- Introducción
- Uso de programas PERL
- Tipos de datos y operadores
- Sentencias de control
- Funciones
- Manejo básico de ficheros
- Identificación de patrones
- Operaciones básicas sobre cadenas
- Módulos de PERL

ntroducción

- **¿Qué es PERL?**
  - PERL (Practical Extraction and Report Language), es un lenguaje de programación interpretado. Fue inicialmente concebido para realizar lectura y manipulación de ficheros de texto de una manera más cómoda que empleando *awk*.
  - PERL tiene elementos de *awk*, *sed*, C y de algunos de los shells de UNIX (*bash*, *tcsh*, …).
  - Tanto la sintaxis como la funcionalidad de PERL son muy similares a las de los lenguajes de programación shell.
  - PERL no es un lenguaje interpretado en el más estricto de los sentidos, ya que los programas se leen completos y se almacenan en un formato intermedio, antes de su ejecución. Es decir, PERL no ejecuta los programas comando a comando (≠shell).

ntroducción

- **¿Para qué sirve PERL?**
  - Tratamiento de ficheros de texto (p.e. filtros de impresión).
  - Automatización de programas.
  - Comunicación entre procesos.
  - Programación de aplicaciones cliente-servidor.
  - Tareas de administración de UNIX.
  - Programación de CGIs (WWW).
  - Uso de interfaces gráficos (extensión Tk).
  - Identificación y manipulación de patrones.

ntroducción

- **Ventajas e inconvenientes**
  - Con relación a los shells:
    - **En general, los programas PERL se ejecutan más rápido. Aunque para programas pequeños tarda más, debido al tiempo de compilación.**
    - **Se realiza un chequeo sintáctico de todo el programa, antes de ejecutarse.**
    - **Es más potente y versátil.**
  - Con relación a C:
    - **La labor de programación es más fácil y rápida.**
      - **En particular, la manipulación de texto (p.e. *pattern-matching*) y la automatización de programas.**
    - **Los programas PERL son más lentos.**
    - **Es menos potente y versátil.**

Uso de programas PERL

- **¿Cómo instalar PERL?**
  - PERL es un lenguaje de libre distribución por lo que se puede descargar gratuitamente desde muchos lugares de la red.
  - El sitio central para investigar sobre PERL es:
    - http//www.perl.com
  - Podrás encontrar versiones para Unix, Linux, Win32, y Macintosh.
  - Puedes descargar la versión en código fuente o en binario, así como versiones beta de las futuras versiones de PERL.

Uso de programas PERL

- **¿Cómo crear y ejecutar un programa PERL?**
  - El proceso de creación y ejecución es muy similar al de los scripts shell.
    - Ejemplo:
      - » Editar un nuevo fichero llamado hola**.pl**, que contenga las siguientes líneas:

```
#!/usr/bin/perl
print “Hola a todos\n”;
```

      - » Cambiar los permisos del fichero (no siempre es necesario):

```
$> chmod +x hola.pl
```

      - » Ejecutar el programa:

```
$> perl hola.pl
```

Uso de programas PERL

- Comando **perl**
  - Opciones generales
    - Cuando se ejecuta el interprete **perl**, se dispone de múltiples opciones. Las más destacables son las que se muestran a continuación.

  - **perl [-c] [-d] [-v][-w] <fich_progr>**

    - » -c: chequea la sintaxis del programa sin ejecutarlo.
    - » -d: utiliza el depurador de código (debugger).
    - » -v: muestra la versión del interprete.
    - » -w: muestra mensajes de aviso.
  - Existen otras muchas opciones de ejecución, que se pueden ver en las paginas de introducción al perl del man

Tipos de datos y operadores

- **Literales**
  - Números
    - PERL almacena todos los datos numéricos como valores reales.
      - Ejemplos: 1, 4.5, 0xff (hexadecimal), O377 (octal), ...
  - Cadenas de caracteres (*strings*)
    - Existen dos maneras de definir cadenas de caracteres:
      - Sin interpolación de variable y sin caracteres especiales:
        - » Todos los caracteres son interpretados del mismo modo excepto la comilla simple (‘).

        Ejemplo:

        ‘cadena’
      - Con interpolación de variables y con caracteres especiales:
        - » Los caracteres que siguen al símbolo $ son interpretados como el nombre de una variable, y son substituidos por el valor de ésta.
        - » Los caracteres que siguen al símbolo \ son interpretados de manera análoga a C, como caracteres de control.

        Ejemplo:

        “cadena\n”

Tipos de datos y operadores

- Literales
  - Caracteres especiales

\n	Nueva línea
\t	Tabulador
\u	Poner en mayúsculas el próximo caracter
\l	Poner en minúsculas el próximo caracter

- Listas de elementos
  - Existen múltiples maneras de definir listas de elementos. Las listas de elementos están directamente relacionadas con las variables array que se tratarán con detalle más adelante.
    - Ejemplos: - (), (1,2,3), (1..4) # literales de array
      - (“clave1”, 2, “clave2”, 3),
      - (“clave1”=>2, “clave2” => 3)

Tipos de datos y operadores

- **Variables**
  - Escalares
    - Las variables escalares en PERL son similares a las variables del shell, tanto en sintaxis como en función.
    - Comienzan con el símbolo $
      - Definición de una variable:
      - $variable = valor;
      - Uso de una variable:
      - $variable1 = $variable2;

Asignación con **=**, no confundir con **==**

    - PERL no realiza chequeo de tipos, y por lo tanto no puede distinguir si se trata de un número o una cadena de caracteres. Si el valor de la variable no puede ser convertido, utiliza el valor por defecto (0 en el caso de números).
      - Ejemplo:
      - $a = 1;
      - $b = 3 * $a;

      - ATENCIÓN: por defecto PERL no avisa cuando realiza conversiones inadecuadas. Es necesario utilizar la opción de línea -w.

Tipos de datos y operadores

- Ejemplo1: Concatenar DNA

Comentario

Imprimir en pantalla

Unir cadenas

Salir del programa

```
#!/usr/bin/perl -w
# Concatenar DNA
# Guardar dos cadenas de DNA en las variables $DNA1 y $DNA2.
$DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';

# Mostrar en pantalla las dos cadenas
print "Estas son las dos cadenas de DNA:\n\n";

print $DNA1, "\n";
print $DNA2, "\n\n";

# Unir los dos fragmentos de DNA y guardarlo en la variable $DNA3, usando e
# operador .   Mostrarlo en pantalla.
$DNA3 = $DNA1 . $DNA2;

print "Aquí está la unión de las dos cadenas: \n $DNA3  \n\n";


exit;
```

Tipos de datos y operadores

- Ejemplo2: Concatenar DNA introducido por el usuario

Recoge datos de la entrada standard

Elimina el retorno de carro

```
#!/usr/bin/perl -w
# Concatenar DNA
# Guardar dos cadenas de DNA en las variables $DNA1 y $DNA2.
print "introduce una cadena de DNA:\n";
$DNA1 = <STDIN>;
chomp $DNA1;

print "introduce otra cadena de DNA:\n";
$DNA2 = <STDIN>;
chomp $DNA2;

# Mostrar en pantalla las dos cadenas
print "Estas son las dos cadenas de DNA: $DNA1 \n $DNA2 \n\n";

# Unir los dos fragmentos de DNA y guardarlo en la variable $DNA3,
# usando el operador "."   Mostrarlo en pantalla.
$DNA3 = $DNA1 . $DNA2;

print "Aquí está la unión de las dos cadenas: $DNA3 \n\n";

exit;
```

Tipos de datos y operadores

- Variables
  - Arrays
    - Existen dos tipos de arrays en PERL: indexado y asociativo.

Indice	Valores
0	uva
1	Manzana
2	pera

Clave	Valores
dia	lunes
mes	enero
año	1999

Los índices de arrays indexados comienzan en 0

» Cada elemento del array es una variable.

» Nota: El número de elementos que puede tener un array está limitado por el tamaño de la memoria del sistema.

Tipos de datos y operadores

- Variables
  - Arrays indexados
    - Definición de un array:
      - » Comienzan con el símbolo @

        @nombre = <literal de array>;

        Ejemplo:

        @mi_array = (1,2,3,4);

        @tu_array = (-1..5);

        @su_array = ($nombre, 1, 'CTG');
    - Acceso a un elemento del array:
      - » $valor_elemento = $array[indice]; # obtiene el valor
      - » $array[indice] = $valor_elemento; # almacena el valor
    - Manipulación del array como pila:
      - » push (@array, $elemento); # "mete" la variable elemento al final
      - » $elemento = pop (@array); # "saca" el último elemento del array
    - Manipulación del array como pila inversa:
      - » unshift (@array,$elemento); # "mete" la variable elemento al principio
      - » $elemento = shift (@array); # "saca" el primer elemento

Para referirnos a un elemento de un array se usa el símbolo $

Tipos de datos y operadores

- Variables
  - Arrays indexados
    - Obtención del número total de elementos de un array:
      - » $num_elementos = $#array; # devuelve el último índice
      - » $num_elementos = scalar @array; # número de elementos

    Invertir los elementos de un array:
      - » @z_invertido = reverse @z;

    Ordenar los elementos de un array alfabéticamente:
      - » @z_ordenado = sort @z;

Ejercicio 1:

1. Crear un array con el nombre de 4 ácidos nucleicos
2. Insertar uno más al final
3. Eliminar el primero
4. Invertirlo
5. Ordenarlo alfabéticamente
6. Imprimir el número de elementos

Imprimir el array después de cada paso

Tipos de datos y operadores

- Variables
  - Arrays asociativos (**hash**)
    - Comienzan con el símbolo %
    - Definición de un hash:
      - %nombre = literal_de_hash;
      - Ejemplo:
        %usuario = (“nombre” => “daniel”, “e-mail” => “dmozos”);
        %usuario = (“nombre”, “daniel”, “e-mail”, “dmozos”);
    - Acceso a un elemento del hash:
      - $valor_elemento = $hash{clave};
      - $hash{clave} = $valor_elemento;
      - Ejemplo:
        $usuario{e-mail} = “dmozos”;

Para referirnos a un elemento de un array se usa el símbolo $

Tipos de datos y operadores

- Variables
  - Arrays
    - Manejo especial de hashs:
      - » @claves = keys %hash # devuelve la lista de claves
      - » @valores = values %hash # devuelve la lista de valores
      - » @pares = each %hash # devuelve el siguiente elemento de la lista de pares (clave,valor)
      - » delete $hash{clave} # elimina un elemento del hash

Ejercicio 2:

1. Crear un array asociativo con el nombre de 4 ácidos nucleicos y su abreviatura
2. Insertar uno más
3. Eliminar uno de ellos usando su abreviatura
4. Imprimir los nombres de todos los ácidos
5. Imprimir todos las abreviaturas
6. Imprimir el número de elementos

Tipos de datos y operadores

- Operadores
  - Numéricos:
    - El conjunto de operadores numéricos es el estándar más algún que otro operador especial.
      - » Aritméticos: +, -, *, /, %, ++, --, **
      - » Bits: &, |, ^, >>, <<
      - » Lógicos: ||, &&
      - » Comparación: ==, !=, >, <, >=, <=, <=>

División
Módulo
Autoincremento
Autodecremento
Exponenciación
Desplazamiento a la derecha
Devuelve –1,0,1 Al comparar los argumentos

  - Cadenas de caracteres:
    - PERL tiene un conjunto completo de operadores para strings.
      - » Manipulación: ., x
      - » Comparación: eq, ne, lt, gt, le, ge, cmp

Concatenar
Copiar varias veces
Comparación:
-1 si menor
0 si igual
1 si mayor

Ejercicio 3:

1. Guardar en dos variables dos cadenas de DNA.
2. Crear una nueva variable uniendo las dos anteriores
3. Crear una nueva variable repitiendo tres veces la cadena 2

Tipos de datos y operadores

- Operadores
  - Identificación de patrones (pattern-matching):
    - Los operadores de identificación de patrones permiten realizar 2 funciones distintas:
      - » Búsqueda de un patrón en una cadena de caracteres:

        [cadena =~] [m]/expresión_regular/modificadores

        - Comportamiento: devuelve verdadero o falso si la expresión regular se encuentra dentro de la cadena o no.
        - Ejemplo:

          "Ejemplo de CadeNa" =~ m/[Cc]ade[Nn]a/i;

      - » Substitución de un patrón dentro de una cadena de caracteres:

        [cadena =~] [s]/expresión_regular/texto/modificadores

        - Comportamiento: cada vez que encuentra un patrón substituye la sub-cadena por el texto.
        - Ejemplo:

          $cadena = "Ejemplo de CadeNa";

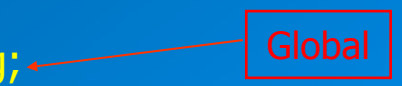
          $cadena =~ s/[Cc]ade[Nn]a/"cadena"/i;

          print $cadena;
    - El comportamiento del operador !~ es opuesto al del operador =~

Tipos de datos y operadores

- Ejemplo 3: Transcripción de ADN a ARN

```
#!/usr/bin/perl -w
#    Transcripción de ADN a ARN

# Cadena de DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Imprimir el DNA en pantalla
print "Esta es la cadena de partida DNA: $DNA \n\n";

# Transcribir el ADN a ARN sustituyendo las Ts por Us.
$RNA = $DNA;
$RNA =~ s/T/U/g;

# Imprimir el RNA en pantalla
print "Esta es la cadena de ARN:\n\n";

print "$RNA\n";

# Salir del programa
exit;
```



Tipos de datos y operadores

- Operadores
  - Chequeo de ficheros:
    - Existen múltiples operadores de ficheros que permiten chequear ciertas condiciones de los ficheros:
      - » Existencia y tamaño: -e, -z, -s
      - » Permisos: -r, -w, -x, -R, -W, -X
      - » Tipo: -f, -d, -l, -S, -p, -T, -B
      - » Otros: -M, -A

```
If (-e $nombre_fichero)
        {
        print "El fichero $nombre_fichero existe. \n";
        }
```

- » -e Chequea si el fichero existe
- » -z Chequea si el fichero tiene tamaño 0
- » -s Chequea si el fichero no tiene tamaño 0, y devuelve el tamaño.

Tipos de datos y operadores

- Operadores
  - Chequeo de ficheros:
    - » -r Chequea si el fichero tiene permiso de lectura
    - » -w Chequea si el fichero tiene permiso de escritura
    - » -x Chequea si el fichero tiene permiso de ejecución
    - » -f Chequea si el fichero es un fichero normal
    - » -d Chequea si el fichero es un directorio
    - » -l Chequea si el fichero es un enlace simbólico
    - » -S Chequea si el fichero es un socket
    - » -p Chequea si el fichero es un pipe con nombre
    - » -T Chequea si el fichero es de texto
    - » -B Chequea si el fichero es binario
    - » -M Indica la edad en días del fichero cuando el script comenzó
    - » -A Indica la edad en días del fichero desde el último acceso

Mas información sobre tipos de datos en: perldata

Mas información sobre operadores y precedencia en: perlop

Sentencias de control

- Bloques
  - Un bloque de PERL es muy similar a un bloque de C. Se trata de un conjunto de instrucciones delimitado por { }.
    - Ejemplo:

```
{ $a = 10;
  $b = 20; }
```

- Instrucciones de control
  - Condicionales
    - Sentencia **if:**
      - **if (expresion) Bloque**
    - Sentencia **if...else...:**
      - **if (expresion) Bloque else Bloque**
    - Sentencia **if... elsif... elsif... ... else...**
      - **if (expresion) Bloque elsif (expresion) Bloque... Else Bloque**
    - Sentencia **unless**
      - **unless (expresion) Bloque**

Sentencias de control

- Ejemplo:

```
if ($a < 10)
        { print “$a es menor que 10\n”; }
elsif ($a =10)
        { print “$a es igual a 10\n”; }
else
        { print “$a es mayor que 10\n”; }
```

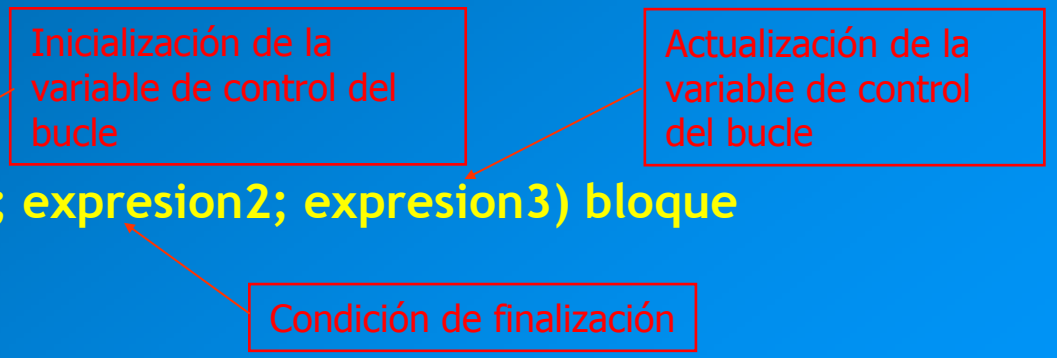
Sentencias de control

- Instrucciones de control
  - Condicionales múltiples
    - PERL no tiene sentencias para tomar decisiones múltiples, pero se pueden usar si se pone al comienzo del programa: use switch
    - La sintaxis es:

```
switch:{
    if (expresion1) bloque
    if (expresion2) bloque
    if (expresion3) bloque
    bloque
    }
```

Sentencias de control

- Instrucciones de control
  - Bucles
    - Sentencia **while**
      - **while (expresion) bloque**
    - Sentencia **for**
      - **for (expresion1; expresion2; expresion3) bloque**
    - Sentencia **foreach**
      - **foreach variable (lista) block**



Sentencias de control

- Ejemplos:

```
- for ($i = 0; $i < 20; $i = $i +2)
      { print $i * 2; }

- for (;;)
      { print “bucle infinito\n”; }

- foreach $i (1,2,3,4)
      { print $i;}

- while ($i < 10)
      {print $i++;}
```

Sentencias de control

- Instrucciones de control
  - Control de bucles
    - **next:** salta el resto del bucle, y pasa a analizar la condición.
    - **last:** hace que se finalice el bucle aunque no se cumpla la condición.

    - En bucles anidados tanto next como last se aplican al bucle más interno.

Ejercicio 4:

Usando la siguiente función:
@dna = split(‘’, $dna);
Que coloca cada una de las bases de la cadena de ADN almacenada en la variable $dna, en cada uno de los elementos del array @dna
Contar cuántas veces aparece cada una de las bases.

Mas información sobre la sintaxis de las sentencias de control en: perlsyn

Funciones

Funciones.

- Las funciones encapsulan un fragmento de código dándole un nombre, permiten pasarle parámetros, y devuelven resultados.

- ***Definición de una función***
  - Formato general:
    - **sub** nombre_función bloque

  - Parámetros de la función:
    - Los parámetros de una función son escalares. Si se le pasa un array lo convierte en sus elementos escalares.
    - Si se pasan varios argumentos, PERL los convierte también en sus elementos escalares.
    - El array predefinido **@_** contiene los parámetros de la función.
    - Por tanto, en $_[0] se encuentra el 1º parámetro, en $_[1] el 2º.

  - Resultado de la función:
    - Por defecto se toma como resultado el producido por la última instrucción. También es posible utilizar la función **return** como en C. En principio las modificaciones de los parámetros de llamada dentro de una función no se reflejan fuera de la misma

Funciones

Funciones.

- ***Uso de una función***
  - Sin parámetros:
    - Nombre_función;
  - Con parámetros:
    - Nombre_función lista_parámetros;
  - Ejemplo:

```
#!/usr/bin/perl –w
# Programa principal
$i = 1; $j = 2; $k= 3;
$resultado = mult3 ($i,$j,$k);
print "El resultado de multiplicar $i, $j y $k es: $resultado\n";
exit;

 Subrutina de mutiplicación
####
sub mult3 {
my ($a, $b, $c) = @_;
my $resul = $_[0] * $_[1] * $_[2];
return $resul;
}
```

Funciones

- Variables definidas con **my**

  Permite que el ámbito de validez de las variables sea sólo el del bloque en que está definida, por ejemplo, el cuerpo de una función.

- **use strict**
  - Obliga a que todas las variables usadas dentro de un programa tengan que estar definidas con my. En caso de que no lo estén da error.

- Argumentos en la línea de órdenes.
  - Cuando se llama a un programa se hace invocando su nombre seguido de una lista de argumentos, que son visibles dentro del programa.
  - @ARGV es el array que contiene todos los argumentos
  - $0 es una variable que contiene el nombre del programa invocado desde la línea de órdenes.

Funciones

- Paso de parámetros a funciones
  - Paso por valor
    - Es la opción que hemos visto hasta ahora
    - Los valores de los argumentos se copian y se pasan a la función. Lo que se haga con ellos dentro de la función no afecta a los parámetros originales del programa principal.
  - Paso por referencia
    - Es el método para pasar parámetros de diferentes tipos sin problemas.
    - Todo lo que se hace con los argumentos dentro de la subrutina afecta a los argumentos del programa principal.
    - Los argumentos por referencia van precedidos por \ que indica que lo que se pasa no es un valor sino una referencia (una dirección de memoria).
    - Dentro de la subrutina los argumentos que se encuentran en @_ se salvan como variables escalares independientemente del tipo de datos del argumento.
    - Para usar uno de estos argumentos hay que de-referenciarlo, colocando delante de la variable escalar el símbolo del tipo de variable requerido ($, @, %)
    - EJ: $$i, @$j. %$k.

Funciones

- Librerías de funciones
  - Permiten guardar colecciones de funciones en un fichero, que puede ser **incluido** posteriormente en un programa.
  - Se trata de no tener que insertar el código de todas las funciones que se vayan a utilizar dentro de todos los programas que las utilizan.
  - El fichero tendrá el nombre que se desee, con la extensión **.pm**
  - La última línea del fichero debe contener un único 1
  - Para incluir el fichero dentro de un programa, y que todas las funciones sean accesibles desde el programa se usa:
    - **use** nombrefichero (no hace falta poner la extensión .pm)

Mas información sobre funciones en: perlsub

Funciones

Ejercicio5:

Definir una función que calcule el factorial de un número de manera recursiva.

Ejercicio 6:

Diseñar un programa que pida al usuario por pantalla tres cadenas de DNA.
Copiar estas cadenas en tres arrays en los que cada base está en una posición del array.
Llamar a una función que recibe como parámetros los tres arrays y devuelve el número de cada tipo de base encontrada.

Funciones

- Funciones predefinidas
  - Para obtener información específica sobre una función consultar en: **perldoc -f *nombre_de_funcion***
  - Para obtener información de todas las funciones, ver perlfunc

Functions for SCALARs or strings

chomp, chop, chr, crypt, hex, index, lc, lcfirst, length, oct, ord, pack, q/STRING/, qq/STRING/, reverse, rindex, sprintf, substr, tr///, uc, ucfirst, y///

Regular expressions and pattern matching

m//, pos, quotemeta, s///, split, study, qr//

Numeric functions

abs, atan2, cos, exp, hex, int, log, oct, rand, sin, sqrt, srand

Functions for real @ARRAYs

pop, push, shift, splice, unshift

Functions for list data

grep, join, map, qw/STRING/, reverse, sort, unpack

Functions for real %HASHes

delete, each, exists, keys, values

Input and output functions

binmode, close, closedir, dbmclose, dbmopen, die, eof, fileno, flock, format, getc, print, printf, read, readdir, rewinddir, seek, seekdir, select, syscall, sysread, sysseek, syswrite, tell, telldir, truncate, warn, write

Funciones

- Funciones predefinidas

Functions for fixed length data or records

pack, read, syscall, sysread, syswrite, unpack, vec

Functions for filehandles, files, or directories

-X, chdir, chmod, chown, chroot, fcntl, glob, ioctl, link, lstat, mkdir, open, opendir, readlink, rename, rmdir, stat, symlink, sysopen, umask, unlink, utime

Keywords related to the control flow of your perl program

caller, continue, die, do, dump, eval, exit, goto, last, next, redo, return, sub, wantarray

Keywords related to scoping

caller, import, local, my, our, package, use

Miscellaneous functions

defined, dump, eval, formline, local, my, our, reset, scalar, undef, wantarray

Functions for processes and process groups

alarm, exec, fork, getpgrp, getppid, getpriority, kill, pipe, qx/STRING/, setpgrp, setpriority, sleep, system, times, wait, waitpid

Keywords related to perl modules

do, import, no, package, require, use

Funciones

- Funciones predefinidas

Keywords related to classes and object-orientedness

bless, dbmclose, dbmopen, package, ref, tie, tied, untie, use

Low-level socket functions

accept, bind, connect, getpeername, getsockname, getsockopt, listen, recv, send, setsockopt, shutdown, socket, socketpair

System V interprocess communication functions

msgctl, msgget, msgrcv, msgsnd, semctl, semget, semop, shmctl, shmget, shmread, shmwrite

Fetching user and group info

endgrent, endhostent, endnetent, endpwent, getgrent, getgrgid, getgrnam, getlogin, getpwent, getpwnam, getpwuid, setgrent, setpwent

Fetching network info

endprotoent, endservent, gethostbyaddr, gethostbyname, gethostent, getnetbyaddr, getnetbyname, getnetent, getprotobyname, getprotobynumber, getprotoent, getservbyname, getservbyport, getservent, sethostent, setnetent, setprotoent, setservent

Time-related functions

gmtime, localtime, time, times

Functions new in perl5

abs, bless, chomp, chr, exists, formline, glob, import, lc, lcfirst, map, my, no, our, prototype, qx, qw, readline, readpipe, ref, sub*, sysopen, tie, tied, uc, ucfirst, untie, use
* - sub was a keyword in perl4, but in perl5 it is an operator, which can be used in expressions.

Functions obsoleted in perl5

dbmclose, dbmopen

Manejo básico de ficheros

Lectura/Escritura

- Antes de utilizar un fichero para leer o escribir, es necesario abrirlo de manera adecuada mediante la función predefinida **open**.

  open (FILEHANDLE, nombre_de _fichero);
- Mediante el nombre de fichero se puede especificar si se va a abrir el fichero para:
  - Lectura: "nombre_fichero"
  - Escritura: ">nombre_fichero" si no existe el fichero, lo crea.
  - Ambas: "+>nombre_fichero"
  - Escritura, añadiendo al final: ">>nombre_fichero"
- Una vez terminadas las operaciones de lectura o escritura es necesario cerrar el fichero mediante la función **close (FILEHANDLE)**
- Para leer de un fichero se utiliza **<FILEHANDLE>**
  - La interpretación depende del contexto en el que lo utilicemos.

    @texto = < FILEHANDLE>; # devuelve la lista de todas las líneas del fichero

    $linea = < FILEHANDLE>; # devuelve una línea del fichero
- Para escribir en un fichero se emplea una de las funciones predefinidas **print**
- Para eliminar un fichero se usa **unlink**("nombre_fichero")

Manejo básico de ficheros

- **Lectura/Escritura**
  - Ejemplos:

```
open(PEPE,"juan.txt");
while(<PEPE>){
print "La linea $. Es: <$_>\n";
close (PEPE);
```

```
open(PEPE,"juan.txt");
@datos = <PEPE>;
print "El fichero contiene: @datos \n";
close (PEPE);
```

Indica el número de línea

Indica el contenido de una línea

```
open (DANIEL, ">daniel.txt");
print  DANIEL "Primera linea\n";
print  DANIEL "Segunda linea\n";
close (DANIEL);
```

```
open (DANIEL, ">daniel.txt");
print  DANIEL "Primera linea\n";
print  DANIEL "Segunda linea\n";
close (DANIEL);
unlink ("daniel.txt");
```

Manejo básico de ficheros

- Ficheros
  - Por defecto PERL utiliza un conjunto de identificadores para ficheros especiales:
    - STDIN: Entrada estándar.
    - STDOUT: Salida estándar.
    - STERR: Error .
    - ARGV: opciones de la línea de órdenes.

Ejercicio7:

Realizar un programa que pida al usuario el nombre de un fichero, lo abra y cuente cuántas palabras de cada tipo existen. Se puede utilizar como ejemplo el fichero lope.txt.

Ejercicio8:

Abrir el fichero dna1.txt.

Este fichero contiene 5 líneas de cabecero y el resto con DNA.

Contar cuántas veces aparece cada una de las bases.

Usando la subrutina que convierte codones en aminoácidos que encontrareis en el fichero subs.pl, convertir todo el DNA en aminoácidos y guardarlo en un fichero llamado amino.txt

Definir una subrutina que cuente cuántos aminoácidos de cada tipo hay.

Por pantalla solicitar al usuario un porcentaje y mostrar cuáles son los aminoácidos por encima de ese porcentaje.

Depurador de PERL

- Permite ver lo que el programa va haciendo instrucción a instrucción
- Llamada:
  - perl –d nombre_fichero.pl

<DB1> # símbolo del depurador

  - Ordenes del depurador:
    - h muestra todas las órdenes
    - h h muestra la página de ayuda completa
    - h orden explica lo que hace una orden
    - q salir del depurador
    - l lista código fuente, a partir de la posición en que nos encontremos o de la línea que se le indique
    - . Lista la línea a ejecutar
    - - lista la línea recién ejecutada
    - v [linea] lista una ventana alrededor de la linea indicada
    - f nombrefichero Lista el código fuente de nombrefichero

Depurador de PERL

- Ordenes del depurador:
  - b num_línea — coloca un punto de parada en la línea num_línea
  - B num_linea — quita el punto de parada de la línea num_línea
  - B * — quita todos los puntos de parada
  - s — ejecuta la siguiente sentencia
  - n — ejecuta la siguiente sentencia, si es una llamada a subrutina, la ejecuta toda.
  - r — vuelve de una subrutina
  - c num_linea — continua hasta num_linea
  - p expr — imprime la expresion
  - V — lista las variables.

Mas información sobre depuración de errores en: perldebtut y perldebug

dentificación de patrones

- Expresiones regulares
  - Una expresión regular, ER, es una cadena que describe un patrón.
  - Su utilidad es:
    - Buscar una cadena en otra
    - Extraer las partes deseadas de una cadena
    - Buscar y reemplazar unas cadenas por otras, dentro de cadenas.
  - Las ER aparecen entre /.
  - La ER más simple es una cadena de caracteres, tal cual:
    - Ej: /hola/ es una expresión regular
  - Ejemplo de uso
    - "Hola a todos" =~ /Hola/;

Operador que asocia la cadena de la izq. Con el patrón de la derecha. Devuelve TRUE si el patrón está dentro de la cadena.

  - Este operador puede utilizarse fácilmente en sentencias condicionales.
  - El operador !~ es el opuesto a =~

dentificación de patrones

- Expresiones regulares
  - La ER también puede sustituirse por una variable:
  - Ej:

```
$saludo = “Hola”;
if (“Hola a todos” =~ /$saludo/ {
        print “El patrón aparece\n”;
        }
else {
        print “El patrón no aparece\n”;
        }
```

  - Dentro de una ER las variables se interpolan igual que entre “.

dentificación de patrones

- Expresiones regulares
  - El delimitador / puede cambiarse por otro si se usa como operador:
    =~ m
  - Ej:

```
if (“Hola a todos” =~ m {Hola})  {
      print “El patrón aparece\n”;
      }
```

El delimitador es ahora {

  - Si se usa como delimitador “, el carácter / puede usarse en la ER como un carácter normal.

  - Si una ER aparece varias veces en la cadena donde se está buscando, PERL identifica la primera aparición.

dentificación de patrones

- Expresiones regulares
  - Metacaracteres. Son caracteres que no pueden usarse tal cual dentro de una ER:

    { } [ ] ( ) ^ $. | * + ? \

    / tampoco puede usarse cuando usamos // como delimitador de la ER.
  - Estos metacaracteres tendrán significados especiales y para usarlos debe colocarse delante \
  - Ej:

```
if ("el intervalo es [0,1)." =~  /\[0,1\)\. /  {
        print "El patrón aparece\n";
        }
```

dentificación de patrones

- Expresiones regulares
  - Anclajes
    - Sirven para determinar dónde debe buscarse el patrón.
      - ^: Sólo busca la ER al comienzo de la cadena de caracteres. Se coloca delante del patrón.
      - $: Sólo busca la ER al final de la cadena de caracteres, o antes de un carácter de nueva línea. Se coloca detrás del patrón.
    - Ej:

      "Hola a todos" =~ /^Hola/ -- Devuelve TRUE

      "Hola a todos" =~ /Hola$/ -- Devuelve FALSE

      "Hola a todos" =~ /^Hola$/ -- Devuelve FALSE. Sólo coincidiría con la cadena "Hola"

dentificación de patrones

- Expresiones regulares
  - Clases de caracteres. Permiten a un conjunto de caracteres, en vez de a uno solo, aparecer en un punto dado de una ER.
  - Las clases de caracteres se representan mediante corchetes [ ], donde lo que aparece dentro de los corchetes son los caracteres que se quiere hacer coincidir.
  - Ej:
    - /[bcr]at/; identificaría “bat”, “cat” o “rat”
    - /[yY][eE][sS]/; identificaría yes, sin preocuparse de mayúsculas y minúsculas.

dentificación de patrones

- Expresiones regulares
  - Existen una serie de caracteres con significado especial cuando están dentro de corchetes:
    - $ denota una variable escalar
    - \ se utiliza para secuencias de escape
    - - es un operador de rango, que permite expresar varios caracteres contiguos de un modo abreviado:
      - [0-9] cualquier número entre 0 y 9
      - Cuando – es el primer o último carácter en una clase se trata como un carácter normal.
    - ^ si está colocado en la primera posición de una clase, denota una clase de carácter negada, es decir coincide con cualquier carácter excepto los que están entre los corchetes.
      - Ej:
        - /[^0-9] identificaría cualquier carácter no numérico.
        - /[^a]at/ identificaría bat o cat, pero no at o aat
        - /[a^]at/ identificaría aat y ^at.

dentificación de patrones

- Expresiones regulares
  - Abreviaturas dentro de clases de caracteres
    - \d equivalente a [0-9]
    - \s equivalente a un espacio en blanco, [\t\r\n\f\]
    - \w equivalente a un carácter alfanumérico o _, [0-9a-zA-Z_]
    - \D es equivalente a negar \d, ^[0-9]
    - \S es equivalente a negar \s
    - \W es equivalente a negar \s
    - . Identifica cualquier carácter, excepto \n
  - Estas abreviaturas pueden usarse fuera de clases de caracteres, dentro de la ER.

dentificación de patrones

- Expresiones regulares
  - Ejemplos de expresiones con metacaracteres:
    - /.ar/ identifica a Zar, par, dar, 1ar, ...
    - /[ZM]ar/ identifica solamente a Zar y a Mar.
    - /a[0-9]/ identifica a a0, a1, a2, ...
    - /a\-[0-9]/ identifica a a-0, a-1, a-2, ...
    - /zar|par/ identifica a zar y a par.

Ejercicio 9:
Introducir por teclado una palabra y dar un mensaje en caso de que comience por vocal, después haya un número y a continuación cualquier letra.

dentificación de patrones

- Modificadores
  - Se colocan después del delimitador / final
    - i la operación de identificación no es sensible a mayúsculas y minúsculas.
    - s trata la cadena como una única línea. El “.” identifica cualquier carácter, incluso \n. ^ identifica sólo al comienzo de la cadena, y $ sólo al final.
    - m trata la cadena como un conjunto de múltiples líneas. El “.” identifica cualquier carácter excepto \n. ^ identifica al comienzo de cualquier línea, y $ al final de cualquier línea.
    - sm trata la cadena como una única línea, pero detecta múltiples líneas. El “.” identifica cualquier carácter, incluso \n. ^ identifica al comienzo de cualquier línea, y $ al final de cualquier línea.
    - g identifica la ER tantas veces como sea posible.
    - x permite colocar espacios en blanco y comentarios dentro de una ER sin considerarlos a la hora de la identificación. Se usa para aumentar la legibilidad.

dentificación de patrones

- Expresiones regulares
  - Metacaracter | (o lógica)
    - Se usa para crear ER, en las que hay varias alternativas:
    - Ej:
      - /cat|dog|bird/ identifica cat, dog o bird.
    - PERL trata de encontrar la ER lo antes posible, para ello probará cada una de las tres posibilidades del ejemplo con el primer carácter, antes de continuar con el segundo.
  - Metacaracteres de agrupamiento ( )
    - Permite tratar partes de una ER como una unidad.
    - Ej:
      - /(a|b)b/ identifica ab y bb
      - /(^a|b)c/ identifica ac al comienzo de la cadena o bc en cualquier lugar.

dentificación de patrones

- Expresiones regulares
  - Extracción de lo identificado
    - En $1 se guarda lo identificado con el primer agrupamiento, en $2 lo del segundo, etc.
    - Ej:

```
$time =~ /(\d\d):(\d\d):(\d\d)/;
$horas = $1;
$min = $2;
$seg = $3;
```

    - Ej2:

```
($horas, $min, $seg) = ($time =~ /(\d\d):(\d\d):(\d\d)/);
```

    - En caso de anidamiento de los agrupamientos en $1 se guarda lo correspondiente al paréntesis abierto más a la izquierda.
    - \1, \2, … también guarda lo último identificado, y además se puede usar dentro de una ER:
    - Ej:
      - /(\w\w)\1/ identificaría cualquier cadena que tuviese 2 caracteres consecutivos repetidos, como “titicaca”.

dentificación de patrones

- Expresiones regulares
  - Situación de lo identificado
    - PERL también guarda dos arrays con información de la posición de lo que se ha identificado: @- y @+.
    - $-[0] es la posición del comienzo de lo primero identificado y $+[0] la del final.
    - $-[n] es la posición de comienzo del agrupamiento identificado nº n y $+[n] la del final.

Ejercicio 10:
Introducir por teclado una palabra y dar un mensaje en caso de que contenga la sílaba pa, indicando en que posición dentro de la palabra se encuentra. Dar otro mensaje si contiene la sílaba pi indicando también su posición.

dentificación de patrones

- Expresiones regulares
  - Metacaracteres de repetición
  - Se colocan después del carácter, clase de carácter o agrupamiento que se quiere repetir
    - *: repetición del patrón anterior 0 o más veces.
    - +: repetición del patrón 1 o más veces.
    - ?: repetición del patrón 0 o 1 veces.
    - {n}: repetición del patrón n veces.
    - {n,}: repetición del patrón n o más veces.
    - {n,m}: repetición del patrón entre n y m veces.
    - Ejemplos:
      - /79*/ identifica a 7, a 79, a 799 ...
      - /79+/ identifica a 79, a 799 ...
      - /79?/ identifica a 7 y a 79
      - /79{2}/ identifica a 799
      - /79{2.}/ identifica a 799, a 7999, a 79999 ...
      - /79{2,4}/ identifica a 799, a 7999 y a 79999

Ejercicio 11:
Introducir por teclado una palabra y dar un mensaje en caso de que sea un nombre de variable legal en PERL.

dentificación de patrones

- Expresiones regulares
  - Metacaracteres de repetición
    - Estos metacaracteres tratan de identificar lo más posible de la cadena.
    - Si se desea que un metacaracter de repetición identifique lo menos posible, debe escribirse con una ? al final.
    - Ej:
      - a?? Trata de identificar a 0 o 1 vez, comenzando por 0 veces.

Ejercicio 12:

Para el siguiente código:

```
$x = "the cat in the hat";
$x =~ /^(.*)(cat)(.*)$/;
```

Determinar que se identifica en cada uno de los agrupamientos.

Hacer lo mismo con:

```
$x = "the cat in the hat";
$x =~ /^(.*)(at)(.*)$/;
```

Explicar lo ocurrido

dentificación de patrones

- Búsqueda y sustitución de textos: s///
  - La función de concordancia de patrones =~, se puede usar para modificar cadenas: s/patrón/reemplazo/[opciones]
    - Cada vez que se encuentra en la cadena el **patrón** se sustituye por **reemplazo**, este se interpreta literalmente.
    - Las opciones son las mismas que hemos estudiado para la identificación de patrones.
    - La opción más interesante es g, que permite modificar todas las apariciones del patrón. ~

dentificación de patrones

- Expresiones regulares

Ejercicio 13:

Escribir un código que pida al usuario un número y diga si es válido.
Debe identificar números enteros con o sin signo y números en notación científica o punto flotante: -12.23e-128.
Nota: Los números en punto flotante pueden tener el signo delante de la mantisa o no, pueden no tener parte decimal o no tener parte entera. La letra que indica el exponente puede ser mayúscula o no, y el exponente puede tener signo o no.

Ejercicio 14:

1. Crear un programa que abra el fichero GenBank.gb y haga lo siguiente:
2. Guarde en un fichero toda la información que no se corresponda con los datos de una secuencia.
3. Guarde en otro fichero los datos de secuencia.
4. Analice cada línea de secuencia e indique en qué líneas aparece tca y en qué posición dentro de la línea.

Operaciones básicas sobre cadenas

- pos $cadena
  - Devuelve la localización de la última concordancia del patrón en una cadena (hay que usar la opción g). $cadena es la cadena donde se ha buscado el patrón.
  - Ej:

```
$x = “hola pepe, Que tal?”;
while ($x =~ /(\w+)/g) {
      Print “La palabra $1, acaba en la posición “, pos $x, “\n”;
      }
```

- //g devuelve una lista de agrupamientos identificados
  - Ej:

```
@palabras = ($x =~ /(\w+)/g);  #devuelve: $palabras[0] = hola
                                          $palabras[1] = pepe
                                          $palabras[2] = Que
                                          $palabras[3] = tal?
```

Operaciones básicas sobre cadenas

- split /patrón/, cadena, limite
  - Divide la cadena en una lista de subcadenas y devuelve esa lista.
  - El patrón es el elemento delimitador que determina dónde se divide una cadena.
  - límite, cuando existe, determina cuantas subcadenas deben buscarse.
  - Ej:

```
$x = “hable con ella”;
@palabras = split /\s+/, $x;
```

El contenido del array es:

```
$palabras[0] = ‘hable’
$palabras[1] = ‘con’
$palabras[2] = ‘ella’
```

Operaciones básicas sobre cadenas

- chop ($cadena)
  - Elimina el último carácter de una cadena
- chomp($cadena)
  - Elimina todos los caracteres nueva línea al final de una cadena
- length($cadena)
  - Devuelve la longitud de una cadena
- lc($cadena)
  - Convierte la cadena a minúsculas
- uc($cadena)
  - Convierte la cadena a mayúsculas
- lcfirst($cadena)
  - Convierte el primer carácter de la cadena a minúsculas
- ucfirst($cadena)
  - Convierte el primer carácter de la cadena a mayúsculas

Operaciones básicas sobre cadenas

- . Une dos cadenas
- join($cadena_de_union, @lista)
  - Crea una cadena nueva uniendo todos los elementos del array lista, separados por la cadena de unión elegida
- index($cadena, $subcadena, [$desplazamiento])
  - Devuelve la posición de una subcadena en una cadena, realizando la búsqueda de izquierda a derecha, y comenzando por el principio de la cadena. Si se quiere comenzar la búsqueda desde otro punto se puede dar un desplazamiento.
- rindex($cadena, $subcadena, [$desplazamiento])
  - Igual que index pero realizando la búsqueda de derecha a izquierda.
- substr($fuente, $desplazamiento, $longitud)
  - extrae una subcadena de la cadena $fuente, comenzando en $desplazamiento y de longitud igual a $longitud.
- substr($fuente, $desplazamiento, $longitud) = $nueva_cadena
  - sustituye la subcadena especificada con $nueva_cadena

Si es negativo, el desplazamiento se toma desde la derecha de la cadena

Operaciones básicas sobre cadenas

- **Ejemplo:**

```
%finds = ();
$linea = 0;

print "\n Introduzca la palabra a buscar: \n";
$palabra = <STDIN>;
chop ($palabra);

print "Introduzca el fichero donde buscar:\n";
$anombre = <STDIN>;
chop($anombre);
open(IFILE, $anombre) or die "No se puede abrir $anombre $! \n";

while (<IFILE>) {
        $posicion = index($_, $palabra);
        if ($posicion >= 0) {
                $finds{"$linea"} = $posicion;
                }
        $linea++;
        }

close IFILE;

while (($clave,$valor) = each (%finds))  {
        print "Linea $clave : $valor \n";
        }
```

Indica el número del error que se ha producido

Imprime la cadena y aborta el programa

Módulos de PERL

- ¿Qué son?
  - Conjuntos de código que actúan como una biblioteca de llamadas a funciones.
- ¿Cómo usarlos?
  - **use** nombre_modulo
    - Incluye todo el contenido del módulo, como si formase parte de nuestro programa.
- ¿Cómo se llama a una función del módulo?
  - nombre_modulo::nombre_funcion()
- ¿Cómo averiguar qué módulos existen?
  - CPAN (Comprehensive Perl Archive Network)
  - ftp://ftp.rediris.es/mirror/CPAN/

Escuela de Verano Complutense 2004
Bioinformática y biología computacional

Apéndice

Daniel Mozos Muñoz
mozos@dacya.ucm.es

Referencias

- Referencias
  - Una referencia dura ("hard reference") es un puntero, escalar, a un dato de cualquier tipo.
  - Como los tipos de datos que conocemos arrays y hashes pueden contener escalares, podremos contruir arrays de arrays, arrays de hashes, hashes de arrays y hashes de hashes.
  - Para acceder al dato hay que de-referenciarlo.

- Creación de referencias
  - 1.- Operador \
    - Al colocar este operador delante de cualquier tipo de dato lo que se devuelve es una referencia a ese dato.

  - 2.- Creador de arrays anónimos
    - Se puede crear una referencia a un array anónimo usando corchetes

    $ref_array = [1, 2, ['a', 'b', 'c']];

    estos corchetes funcionan así sólo donde PERL espera una expresión.

  - 3.- Creador de hashes anónimos
    - Se puede crear una referencia a un hash anónimo usando llaves

    $ref_hash = {
    'Adan' => 'Eva',
    'Fernando' => 'Isabel',
    };

    estas llaves funcionan así sólo donde PERL espera una expresión.

  - 3.- Creador de subrutinas anónimas
    - Se puede crear una referencia a una subrutina anónima usando sub sin un nombre de subrutina.

    $ref_sub = sub { print "Hola \n"};

Referencias

- Uso de las referencias
  - 1.- Como en el paso de parámetros por referencia, poniendo delante de la variable que contiene la referencia el tipo del elemento referenciado.

    @$ref_array

    %$ref_hash
  - 2.- Uso de un BLOQUE como nombre de variable
    - Es igual al anterior pero usando un bloque en lugar de una variable. Dentro del BLOQUE puede ir cualquier expresión arbitraria.

    $xxx = @{$ref_array};
  - 3.- Uso del operador flecha ->
    - Sirve sólo para referencias a arrays y hashes cuando se va a acceder a elementos individuales.

    $ref_array->[1]

    $ref_hash->{}
    - Cuando se va a usar el operador -> entre índices con corchetes o llaves puede obviarse.

- Operador ref
  - Nos indica a que tipo de dato apunta una referencia.

Referencias

- Arrays de arrays
  - **Declaration of an ARRAY OF ARRAYS**

```
@AoA = (
        [ "fred", "barney" ],
        [ "george", "jane", "elroy" ],
        [ "homer", "marge", "bart" ],
        );
```

  - **Generation of an ARRAY OF ARRAYS**

```
# 1.- reading from file
while ( <> ) {
    push @AoA, [ split ]; }
# 2.- calling a function
for $i ( 1 .. 10 ) {
    @tmp = somefunc($i);
    $AoA[$i] = [ @tmp ];
}
# 3.- add to an existing row
push @{ $AoA[0] }, "wilma", "betty";
```

split sin parámetros divide la cadena por defecto con espacios en blanco

Referencias

- Arrays de arrays
  - **Access and Printing of an ARRAY OF ARRAYS**
    - # one element

```
$AoA[0][0] = "Fred";
```

    - # print the whole thing with refs

```
for $aref ( @AoA ) {
    print "\t @$aref  \n";
}
```

    - # print the whole thing with indices

```
for $i ( 0 .. $#AoA ) {
    print "\t @{$AoA[$i]}  \n";
}
```

    - # print the whole thing one at a time

```
for $i ( 0 .. $#AoA ) {
     for $j ( 0 .. $#{ $AoA[$i] } ) {
                print "el $i $j es $AoA[$i][$j]\n";
    }
}
```

Referencias

1 HASH OF ARRAYS

- **Declaration of a HASH OF ARRAYS**

```
%HoA = (
    flintstones => [ "fred", "barney" ],
    jetsons => [ "george", "jane", "elroy" ],
    simpsons => [ "homer", "marge", "bart" ],
);
```

- **Generation of a HASH OF ARRAYS**

```
# reading from file; more temps
# flintstones: fred barney wilma dino
while ( $line = <> ) {
($who, $rest) = split /:\s*/, $line, 2;
@fields = split ' ', $rest;
$HoA{$who} = [ @fields ];
}
# calling a function that returns a list
 for $group ( "simpsons", "jetsons", "flintstones" ) {
    $HoA{$group} = [ get_family($group) ];
}
# append new members to an existing family
push @{ $HoA{"flintstones"} }, "wilma", "betty";
```

Referencias

HASH de ARRAYS

- Acceso e impresión de un HASH de ARRAYS

```
# Un elemento
$HoA{‘flintstones’}[0] = "Fred";

# print the whole thing
foreach $family ( keys %HoA ) {
    print "$family: @{ $HoA{$family} }\n" ;
}

# print the whole thing with indices
foreach $family ( keys %HoA ) {
    print "family: ";
    foreach $i ( 0 .. $#{ $HoA{$family} } ) {
       print " $i = $HoA{$family}[$i]";
    }
    print "\n";
}
```

Referencias

ARRAYS OF HASHES

- Declaration of an ARRAY OF HASHES

```
@AoH = (
    {
            Lead => "fred",
            Friend => "barney",
     },
     {
            Lead => "george",
            Wife => "jane",
            Son => "elroy",
    },
    {
             Lead => "homer",
            Wife => "marge",
            Son => "bart",
    }
);
```

Referencias

ARRAYS OF HASHES

- Generation of an ARRAY OF HASHES

```
# reading from file
# format: LEAD=fred FRIEND=barney
while ( <> ) {
    $rec = {};
    for $field ( split ) {
                ($key, $value) = split /=/, $field;
                $rec->{$key} = $value;
    }
    push @AoH, $rec;
}

# calling a function that returns a key/value pair list, like
# "lead","fred","daughter","pebbles"
while ( %fields = getnextpairset() ) {
    push @AoH, { %fields };
}

# add key/value to an element
$AoH[0]{pet} = "dino";
$AoH[2]{pet} = "santa's little helper";
```

Referencias

ARRAYS OF HASHES

- Access and Printing of an ARRAY OF HASHES

```
# one element
$AoH[0]{lead} = "fred";
# print the whole thing with refs
for $href ( @AoH ) {
        print "{ ";
        for $role ( keys %$href ) {
            print "$role=$href->{$role} ";
        }
        print "}\n";
}

# print the whole thing with indices
for $i ( 0 .. $#AoH ) {
    print "$i is { ";
    for $role ( keys %{ $AoH[$i] } ) {
                print "$role=$AoH[$i]{$role} ";
    }
    print "}\n";
}

 # print the whole thing one at a time
for $i ( 0 .. $#AoH ) {
    for $role ( keys %{ $AoH[$i] } ) {
                print "elt $i $role is $AoH[$i]{$role}\n";
    }
}
```

Referencias

HASHES OF HASHES

- Declaration of a HASH OF HASHES

```
%HoH = (
    flintstones => {
                lead => "fred",
                pal => "barney",
    },
    jetsons => {
                lead => "george",
                wife => "jane",
                "his boy" => "elroy",
    },
    simpsons => {
                 lead => "homer",
                 wife => "marge",
                kid => "bart",
    },
);
```

Referencias

HASHES OF HASHES

- Generation of a HASH OF HASHES

```
# reading from file; more temps

# flintstones: lead=fred pal=barney wife=wilma pet=dino
while ( <> ) {
    next unless s/^(.*?):\s*//;
    $who = $1;
    $rec = {};
    $HoH{$who} = $rec;
    for $field ( split ) {
                ($key, $value) = split /=/, $field;
                $rec->{$key} = $value;
    }
}

# append new members to an existing family
%new_folks = (
    wife => "wilma",
    pet => "dino",
 );
 for $what (keys %new_folks) {
    $HoH{flintstones}{$what} = $new_folks{$what};
}
```

Referencias

HASHES OF HASHES

- Access and Printing of a HASH OF HASHES

```
# one element
$HoH{flintstones}{wife} = "wilma";

 # print the whole thing
foreach $family ( keys %HoH ) {
print "$family: { ";
foreach $role ( keys %{ $HoH{$family} } ) {
    print "$role=$HoH{$family}{$role} ";
    }
    print "}\n";
}
```

Referencias

Ejercicio 15:

En el subdirectorio GENBANK están almacenados una serie de ficheros con información genética. Cada fichero corresponde a un organismo.

1.- Queremos leer cada uno de esos ficheros y extraer de qué organismo se trata y cuántas bases de cada tipo contiene.

Esta información la vamos a guardar en una tabla que será un hash de hashes, donde las filas de la tabla se acceden con el nombre del organismo y las columnas con el de cada una de las cuatro bases.

2.- Realizar una serie de subrutinas que permitan, mostrar por pantalla un menú con varias opciones:

a) Mostrar todos los organismos por pantalla

b) Guardar toda la tabla en un fichero. Solicitar el nombre de fichero al usuario. El formato será:

ORGANISMO: NOMBREORGANISMO
nº de Adeninas =
nº de citosinas =
…

c) Mostrar el porcentaje de bases de un determinado tipo que existen en un determinado organismo (habrá que pedir al usuario qué base y que organismo). Mostar el resultado con un solo decimal.

d) Finalizar el programa

Referencias

Ejercicio 16:

1.- Implementar una función que multiplique dos matrices de enteros. Los parámetros que se le pasan son las referencias a las dos matrices.

2.- Implementar una función que muestre por pantalla una matriz colocando de menera alineada las filas y las columnas. Se le pasa como parámetro una referencia a la matriz

3.- Implementar una función que pida por pantalla las dimensiones de una matriz, y vaya solicitando al usuario que introduzca los datos para rellenar la matriz fila a fila, separando los diferentes elementos de cada fila con :

4.- Realizar un programa principal que solicite al usuario dos matrices las muestre por pantalla, las multiplique y muestre el resultado por pantalla.

Referencias

Ejercicio 17:

Queremos almacenar los artículos que leemos en una base de datos (BD), pero no nos gusta ninguna de las disponibles, por lo tanto vamos a construirla nosotros mismos.

La BD será una array de hashes. Cada hash consta de 5 campos: Título del artículo, autores, revista, fecha y el nombre del fichero donde guardamos un resumen del artículo.

El programa nos permitirá hacer una serie de cosas, elegidas por un menú:

1.- Introducir un nuevo elemento.

2.- Listar todos los artículos, especificando los 4 primeros campos.

3.- Buscar si existe un artículo dando una palabra clave del título.

4.- Buscar si existe un artículo dando el nombre de un autor

5.- Listar todos los artículos de una determinada revista