ALGORITMOS Y PROGRAMACIÓN (LENGUAJE DE PROGRAMACION)

AYUDA 1

NOCIONES PRELIMINARES.

(Primavera 2003)

Índice.

1. - Objetivo.

El propósito de este documento es conocer y comenzar a utilizar los *recursos*, tanto físicos como lógicos, que se usarán habitualmente en las sesiones de clases de la asignatura. Con este fin presentaremos las características principales del sistema operativo **UNIX.**

2. - Introducción.

El término programación se asocia comúnmente con el de codificación, lo cual es un error bastante grave ya que la actividad de la programación implica una serie de etapas donde la codificación es solamente una de ellas. En líneas generales, el desarrollo de un programa para la resolución de un problema tiene las siguientes fases:

- *Definición del problema,* en la que se debe establecer con claridad qué es lo que se quiere resolver.
- *Diseño del algoritmo*, en la que se desarrolla la secuencia lógica de pasos para la resolución del problema, basada en la aplicación de una metodología de diseño y la utilización de una notación algorítmica.
- *Codificación,* se convierte el algoritmo en un programa escrito en un lenguaje de programación específico, en nuestro caso dicho lenguaje será el Pascal.
- *Ejecución y prueba,* para determinar el grado de corrección del programa construido.
- *Mantenimiento,* para reparar y/o actualizar el programa.

Estos cinco pasos constituyen lo que se llama ciclo de vida de un proyecto informático. Las dos primeras fases se desarrollarán en las clases de teoría, frente a las tres últimas que serán objeto de estudio en las prácticas.

Con el fin de ejecutar y probar los programas debemos trabajar sobre un entorno adecuado, esto es, sobre un *sistema informático*. Un sistema informático se divide en cuatro partes fundamentales: *Hardware* (HW), *Sistema Operativo* (S.O.), *Programas de Aplicación* y *Usuarios*.

El HW proporciona los recursos básicos del sistema informático: memoria, procesador y dispositivos de E/S .

El S.O. puede entenderse como un entorno para ejecutar programas, en este sentido actúa como una interfaz entre los programas y el hardware, de forma que el usuario no necesita conocer los detalles del manejo del hardware. Su objetivo principal consiste en incrementar la productividad, tanto de los usuarios como del HW , ofreciendo una máquina extendida que disminuye la diferencia entre lo que el programador desea y lo que la máquina es capaz de efectuar por sí sola.

Los compiladores, programas comerciales, bases de datos, editores de texto etc..., son los programas de aplicación que normalmente los usuarios utilizan dependiendo de sus necesidades.

Los usuarios no sólo son las personas que hacen uso de la computadora, sino todos aquellos entes que puedan obtener un servicio del mismo, por ejemplo otras computadoras y máquinas.

A continuación, presentaremos las características más relevantes del sistema informático sobre el que se trabajará a lo largo del curso.

3. - El Hardware.

Para la realización del trabajo se dispone de una computadora SUN Server VR80 y una terminal para cada uno de los puestos de trabajo. Sus características más relevantes son las siguientes:

- Procesador: 900 MHz.
- Memoria Principal: 4 GB
- Disco Duro: 80 Gb
- Monitor color de 17”

Las impresiones de los programas se realizarán en la impresora del modulo.

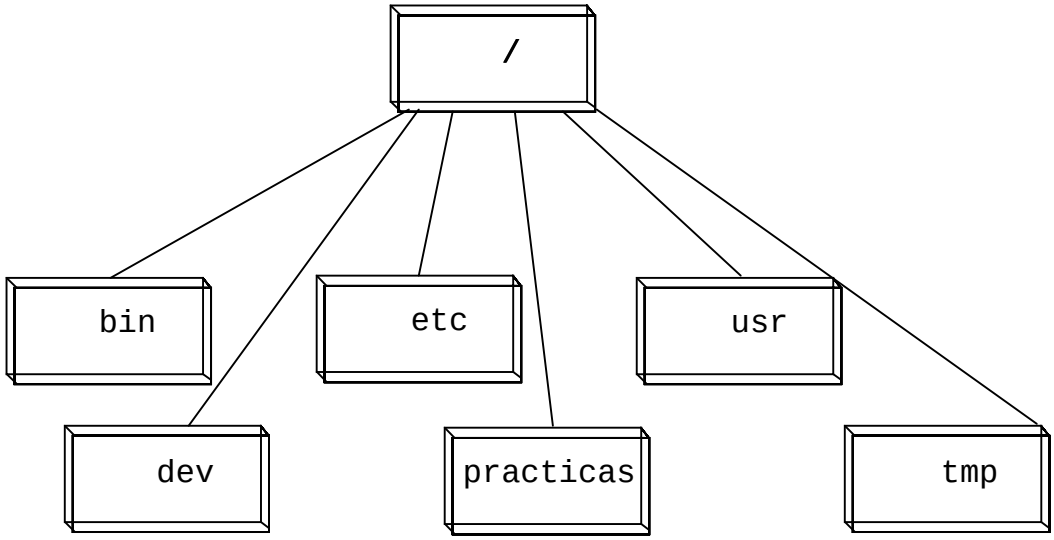
4. - El Sistema Operativo UNIX.

El sistema operativo es un conjunto de programas, cuyo objetivo final es ofrecer una interfaz entre el hardware (HW.) y los usuarios, de forma que el usuario no necesite conocer las peculiaridades de utilización de las diferentes componentes del HW. Su tarea consiste principalmente en gestionar los recursos del sistema informático (memoria, procesador, ...).

Además de los programas propios del SO (cuyo estudio sale del ámbito de la asignatura) uno de los programas de aplicación más importantes es el ***intérprete de órdenes*** (shell), puesto que hace las funciones de intermediario entre el usuario y el S.O.. El intérprete de órdenes se encarga de leer, interpretar y ejecutar las órdenes de los usuarios. Por lo tanto conociendo el conjunto de órdenes que sabe interpretar el intérprete de órdenes es posible acceder a todas las prestaciones que ofrece el sistema.

El sistema operativo UNIX se inició como un proyecto de investigación, convirtiéndose con el paso de los años en un producto ampliamente utilizado en el mundo académico, empresarial y gubernamental. Quizá entre otros uno de los motivos de su amplia difusión sea el hecho de que el código fuente haya estado disponible y en su mayoría escrito en el lenguaje de programación C.

El UNIX es un sistema que soporta la multiprogramación, esto significa que pueden haber varios programas ejecutándose simultáneamente. Además es un sistema interactivo, lo que proporciona una comunicación directa entre el usuario y el sistema (el mediador será el intérprete de órdenes) y multiusuario, lo que permite que varios usuarios trabajen simultáneamente.



La raíz del árbol de denomina directorio raíz y se representa con un / (slash). La raíz contiene, para este ejemplo, seis subdirectorios: bin, etc, prácticas, usr, tmp, dev.

Cuando un usuario accede al sistema se ubica automáticamente en su propio directorio, que denomina *directorio de trabajo*, a través de los comandos adecuados se puede desplazar hacia arriba o hacia abajo en el árbol de directorios, modificando el *directorio actual*. En la siguiente figura, el *usuario1*, al conectarse estaría en el directorio `usuario1`.

El modo de acceso inicial de un fichero temp se muestra por `ls`:

```
$ ls -l temp
-rw-rw-r--   1 alex   57 Dec  5 16:47  temp
```

La siguiente línea de mandato elimina todos los permisos de acceso para el grupo y el resto de usuarios:

```
$ chmod go-rw temp
$ ls -l temp
-rw-------   1 alex   57 Dec  5 16:47  temp
```

***Copia de ficheros:* `cp fich1 fich2`**

Si `fich2` no existe, se tendrán luego dos ficheros (`fich1` y `fich2`) con los mismos contenidos. Si `fich2` existe, el contenido de `fich2` es eliminado y pasa a contener una copia de `fich1.`

Ejemplos:

```
$ cp programa.p otro.p
```

el fichero `otro.p` pasa a tener el contenido de `programa.p`. El fichero `programa.p` no se modifica.

```
$ cp programa* ..
```

copia todos los ficheros que empiecen por la secuencia de caracteres `programa` al directorio superior.

***Contenido ascii de un fichero:* `cat fichero(s)`**

Vuelca en pantalla el contenido de uno o más ficheros de texto. Si hay más de un fichero, el resultado es la con`cat`enación de los contenidos de cada uno de ellos. Si el fichero (o los ficheros) no existe, devuelve un mensaje de error.

***Contenido ascii de un fichero, filtro more :* `more fichero(s)`**

Vuelca en pantalla el contenido de uno o más ficheros, adecuando su presentación al número de líneas de la terminal. Si hay más de un fichero, el resultado es la representación concatenada de los contenidos de cada uno de ellos. Si el fichero (o los ficheros) no existe, devuelve un mensaje de error.

***Consulta acerca del actual directorio:* `pwd`**

Indica en que directorio se encuentra el usuario en un momento dado.

***Creación de un nuevo directorio:* `mkdir directorio`**

Crea un directorio vacío de nombre `directorio`

Ejemplos:

```
$ mkdir dir1
$ ls
```

```
ejem.f
datos
dir1
practica1
```

crea el directorio `dir1` en el actual directorio (luego aparece en el listado con `ls`).

Eliminar un directorio: **`rmdir directorio`**

Elimina el directorio de nombre directorio sólo si éste está ya vacío (si contuviera algún fichero o directorio en su interior habría que eliminarlos previamente con las pertinentes llamadas a `rm` y/o `rmdir`).

```
$ rmdir ejemplo
```

4.5.- Otros comandos.

Redirección de E/S(>, >>, |)

Es posible desviar la salida de pantalla hacia un fichero cualquiera mediante los símbolos `>` y `>>`.

Ejemplos:

```
$ cat fich1 > fich2
```

vuelca el contenido de `fich1`, pero en lugar de hacerlo en pantalla, lo hace en `fich2`.

```
$ ls -l > prueba
```

en lugar de mostrar por pantalla el listado "largo" del directorio actual, lo vuelca al fichero `prueba.`

```
$ date > fich
$ ls >> fich
```

vuelca en `fich` la fecha actual (eso es lo que hace el comando `date`) y luego añade al final del fichero `fich` un listado del directorio.

```
$ cat fich1 fich2 > fich3
```

deja en `fich3` el contenido de `fich1` y `fich2.`

```
$ cat fich1 > fich3
$ cat fich2 >> fich 3
```

es equivalente al ejemplo anterior.

También es posible suministrar los datos de entrada de un programa desde un fichero, y no desde el teclado, mediante `<`.

```
$ ls > fich
$ wc < fich
```

cuenta el número de palabras (`wc` por `w`ord `c`ount) que hay en `fich`, y lo que hay en `fich` no es más que el resultado de hacer `ls`.