Curs 20-21

Unitat 1. Desenvolupament de programari.

Butlletí de pràctiques

Entorns de desenvolupament

IES Jaume II El Just
Tavernes de la Valldigna

Continguts

Introducció

En aquest butlletí pràctic veurem algunes eines que ens seran d'utilitat en unitats posteriors, així com alguns procediments d'ús habitual quan treballem en l'edició i compilació de programes. Llegiu-lo amb atenció, i aneu seguint amb l'ordinador els diferens passos i exemples que s'hi indiquen.

Pràctica 1. L'intèrpret d'ordres i el sistema de fitxers

La *shell* del sistema o *intèrpret d'ordres* és la part del sistema operatiu que ens permet interactuar amb ell a través d'ordres, de forma completament textual, a través d'un terminal.

Als sistemes GNU/Linux existeixen diverses *shells*: *sh*, *bash*, *csh*, *ksh*… per defecte, la shell en Ubuntu és *bash*.

Per tal d'accedir a la shell des d'un entorn gràfic, ho fem a través de diverses aplicacions, anomenades *emuladors de terminal*. Cada entorn gràfic utilitza els seus propis emuladors de terminal. Així, Gnome ofereix *Gnome Terminal*, Mate ofereix *Mate Terminal*, KDE amb *Konsole*… Al nostre cas, utilitzarem un emuldor de terminal anomenat *Tilix*, instal·lat per defecte al programari de JustiX.

En programació, tot i que els entorns de desenvolupament moderns ens permeten crear i executar projectes de forma gràfica, sovint resulta més còmode realitzar xicotetes modificacions del codi i realitzar el procés de compilació i construcció de d'aplicacions des de la línia d'ordres. Tot i que en sistemes operatius veureu a fons aquesta part del sistema, en aquesta pràctica anem a veure una xicoteta introducció per moure'ns pel sistema de directoris, crear una estrucutra, i llegir fitxers. Més endavant veurem com editar-los, també des de la línia d'ordres.

Accés a la terminal

Per tal d'accedir a la terminal, ho podem fer de diverses formes:

1. **A través del menú o el *dock* del sistema**: Generalment en l'apartat d'*Eines del sistema*.

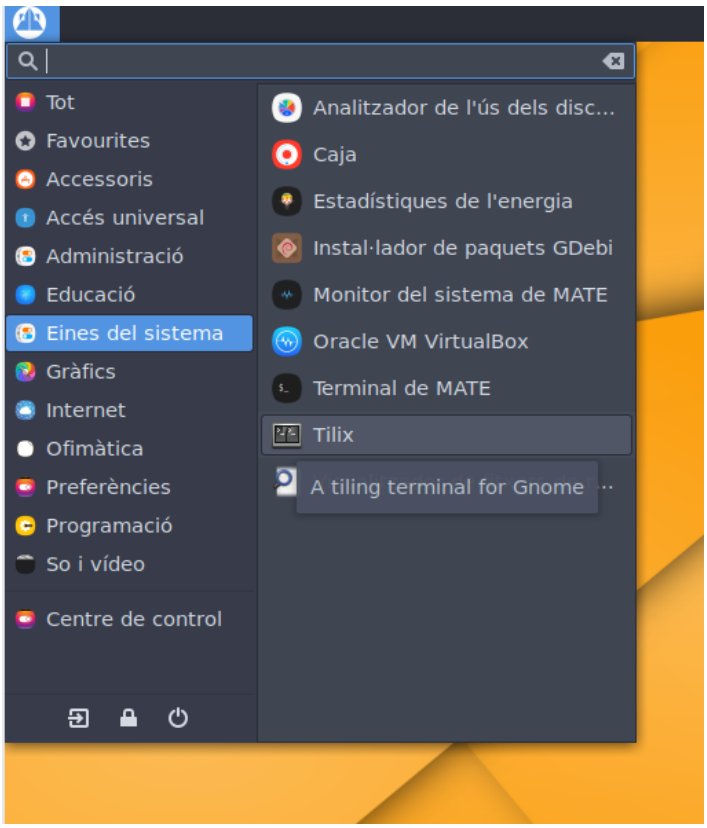

Figura 1: Accés a l'emulador de terminal a través del menú del sisema

Al cas de JustiX, teniu un accés directe al *Dock*, la barra que apareix a la part inferior de la pantalla.

2. **A través d'estalvis de teclat**: Generalment amb la combinació de tecles `Ctrl`+`Alt`+`T` o bé amb `Alt`+`F2` i escrivim el nom de l'aplicació (Tilix).

Una vegada hem acedit a l'emulador de terminal, veurem el *prompt* del sistema, amb un aspecte semblant al següent:

```
usuari@ordinador:ruta$
```

Per exemple:

```
profe@profeDAM:~$
```

Com veiem, apareix el **nom de l'usuari** i de l'**ordinador** al què està connectat. Tot seguit veiem el *path* o *camí* en l'arbre de directoris on ens trobem. En aquest cas, veiem el símbol especial ~, que representa el directori `home` o d'*inici* de l'usuari. Finalment, el símbol `$` ens indica que estem executant `bash` com a un usuari del sistema (l'altra opció sería el símbol # que indicaria que estem com a administradors)

Bé, Ara ja podem començar a interactuar amb el sistema. El mecanisme és tan fàcil com **escriure l'ordre** amb els paràmetres que necessitem, i polsar la tecla **intro**.

Començant amb les ordres

On estem

Si utilitzem l'ordre `pwd`, el prompt ens diu el directori en el què estem actualment:

```
1 profe@profeDAM:~$ pwd
2 /home/profe
```

Com veiem, ens indica que ens trobem a una carpeta anomenada `home`, i dins d'ella, una carpeta amb el nostre nom d'usuari. El primer símbol que apareix és la `/`, o *arrel del sistema*, a partir del qual s'expandeix tot l'arbre de directoris. El treball d'aqueta pràctica el realitzarem a partir d'aquesta mateixa carpeta d'usuari.

Veient el contingut dels directoris

L'ordre `ls`, abreviatura de *list*, ens mostra el *llistat* de fitxers i directoris del directori on ens trobem:

```
1 $ ls
2 Baixades   Escriptori  Música           Plantilles  Vídeos
3 Documents  Imatges     NetBeansProjects  Públic
```

Com veiem, si no hem creat res més, ens mostra les carpetes i els fitxers que tenim al nostre *home*.

També podem indicar-li a `ls` de quina carpeta volem que ens mostre els fitxers. Per exemple, si escrivim:

```
1 $ ls /
2 bin    dev   initrd.img      lib64       mnt   root  snap      sys  var
3 boot   etc   initrd.img.old  lost+found  opt   run   srv       tmp
     vmlinuz
4 cdrom  home  lib             media       proc  sbin  swapfile  usr
     vmlinuz.old
```

Ens mostra el contingut de la carpeta *arrel* del sistema. Com veieu, a l'arrel apareixen diversos directoris i fitxers, comuns en moltes distribucions GNU/Linux. Aquesta estructura segueix un estàndard conegut com FHS *FHS (Filesystem Hierarchy Standard)*, que defineix el contingut que ha d'haver en aquestes carpetes. En unitats posteriors, tornarem a reprendre aquest estàndard.

Per altra banda, l'ordre `ls` accepta també diverses *opcions*. Les opcions, generalment indicades amb un `-` al davant, i separades per espais, serveixen per tal de modificar el comportament de l'ordre. Una opció bastant habitual és utilitzar `ls -l`, per tal d'obtenir un llistat en *format llarg*, que ens

indica, a més del nom de carpetes i directoris, informació addicional, com els permissos, l'usuari i grup propietari, el tamany i la data de la última modificació.

Movent-nos per l'arbre de directoris

Una altra ordre interessant és l'ordre `cd` (*change directory*), que com el nom indica, serveix per canviar el directori actual. Per exemple si escrivim:

```
$ cd Baixades
```

Accedirem a la carpeta `Baixades`

Si ara escrivim:

```
$ cd /
```

Accedirem a l'*arrel* del sistema. Per tal d'accedir al nostre home, podem fer-ho de dos formes:

- Indicant la ruta completa:

  ```
  $ cd /home/el_meu_usuari
  ```

- Amb l'abreviatura ~:

  ```
  $ cd ~
  ```

Com hem comentat, l'estrucutra de directoris té forma d'*arbre*. Això vol dir, que existeix un directori, anomenat *arrel*, del que *naixen* tots els directoris, i de cada directori, *creixen* més directoris, *ramificant* l'estructura. Quan ens trobem a un directori concret, el directori immediatament anterior en l'estrucutura de directoris s'anomena *directori pare* (i per analogia, els directoris continguts dins un directori *directoris fills*). Tant el *directori pare* com el *directori actual* dins de cada directori tenen noms especials: `..` per al directori pare i `.` per al directori actual (més endavant li veurem més utilitats a aquesta abreviatura).

A efectes pràctics, si tenim la següent estructura de directoris:

```
moduls/
|-- EDD
|-- FOL
|-- ISO
|-- LMI
\-- PRG
```

Si ens trobem dins el directori EDD, per tornar al directori moduls, farem:

```
$ cd ..
```

Un error freqüent sol ser no posar l'espai entre `cd` i `...` Tingueu-ho en compte.

Creació, modificació i esborrat de carpetes

Per tal de crear un directori, fem ús de l'ordre `mkdir` (*make directory*), seguida del nom del directori que volem.

Per exemple, des del nostre *home* escrivim:

```
$ mkdir moduls
```

per crear la carpeta *moduls* en la carpeta on estem. Si ara volem crear més carpetes dins d'aquesta, podem fer-ho de dos formes:

1. Accedint a la carpeta moduls i creant dins d'ella altres carpetes

```
$ cd moduls
$ mkdir EDD
```

2. Creant les carpetes a partir del directori *pare*, especificant la ruta:

```
$ mkdir moduls/EDD
```

Per altra banda, és interessant tindre en compte el següent:

- Podem crear diversos directoris en una maeixa ordre mkdir, indicant-los tots com a arguments separats per espai:

  ```
  $ mkdir moduls/EDD moduls/PRG moduls/LMI moduls/ISO moduls/FOL
  ```

- Podem crear una estructura completa amb l'opció `-p` (de *path*). Per exemple, en lloc de fer: `bash $ mkdir moduls $ mkdir moduls/EDD $ mkdir moduls/EDD/Practica1 $ mkdir moduls/EDD/Practica1/part1`

  Podem fer directament:

  ```
  $ mkdir -p moduls/EDD/Practica1/part1
  ```

  Per a que ens cree l'estructura completa, sense haver de crear pas a pas els directoris intermitjos.

Per altra banda, podem fer ús del que s'anomena *expansió de claus*, que ens permet generar múltiples expressions d'una. PEr exemple, podem fer:

```
$ mkdir -p moduls/EDD/Practica1/{part1,part2,part3}
```

Que ens generaría la següent estrucutra de directoris:

```
moduls/
\-- EDD
    \-- Practica1
```

```
4           |-- part1
5           |-- part2
6           \-- part3
```

Com veieu, quan tenim una miqueta de pràctica, resulta bastant més ràpid que crear estructures de carpetes mitjançant el navegador de fitxers.

Anem a passar ara a **esborrar directoris**. Per a això, farem ús de l'ordre `rmdir`, seguida del nom de la carpeta a eliminar. Cal tindre en compte que **per tal d'eliminar un directori, aquest ha d'estar buit**. En cas contrari, se'ns mostrarà el següent error:

```
1 $ rmdir /tmp/moduls/EDD/Practica1/
2 rmdir: failed to remove '/tmp/moduls/EDD/Practica1/': Directory not
     empty
```

En aquest cas, caldria esborrar les carpetes `part1`, `part2` i `part3` per poder esborrar el directori `Practica1`. Proveu a fer-ho.

Altra opció, és utilitzar l'ordre `rm`, que serveix per eliminar tant directoris com fitxers, i indicar-li l'argument `-r` (*esborrat recursiu*), de la següent forma:

```
1 $ rm -r /tmp/moduls/EDD/Practica1/
```

I per acabar aquesta secció, es queda modificar o renomenar el nom dels directoris. Per a això, farem ús de l'ordre `mv`(*move*), amb la sintaxi següent:

```
1 $ mv fitxer_o_directori_original fitxer_o_directori_amb_el_nom_nou
```

Per exemple, per renomenar el directori `part1` com a `part01`, farem:

```
1 $ mv moduls/EDD/Practica1/part1/ moduls/EDD/Practica1/part01/
```

Consultant el contingut dels fitxers

L'últim apartat d'aquesta primera pràctica, serà consultar el contingut de fitxers. Como que encara no hem creat cap fitxer, anem a fer ús d'un fitxer del sistema, com per exemple puga ser el fitxer `/etc/apt/sources.list`,on s'indiquen les diferents fonts de programari que tenim configurades al sistema.

Per tal de **consultar el contingut** del fitxer, fem ús de l'ordre `cat`:

```
1 $ cat /etc/apt/sources.list
```

Com veureu, es tracta d'un fitxer un tant llarg, i per llegir-lo tot, ens haurem de desplaçar cap amunt amb la barra de desplaçament lateral. Si volem evitar això, i que ens mostre el resultat *paginat*, farem ús de l'ordre `more` en lloc de `cat`:

```
$ more /etc/apt/sources.list
```

Amb açò veurem tantes línies del fitxer com ens càpiguen en pantalla i podrem desplaçar-nos cap avall amb la tecla `intro`, per avançar una línia, o la tecla `espai`, per avançar tantes línies més com càpiguen.

El que no podem fer amb `more` és *tornar cap amunt* en el fitxer. Si volem poder moure'ns amunt i avall pel fitxer, podem fer ús de l'ordre `less`, de la mateixa manera:

```
$ less /etc/apt/sources.list
```

Amb aquesta ordre, a més de l'`espai` i l'`intro`, podem utilitzar els *cursors* amunt i avall per desplaçar-nos pel document. **Per tal de poder eixir del que ens mostra `less`, hem de prémer la tecla `q`.**

Altres ordres interessants per consultar fitxers són `head` i `tail`, que ens permeten consultar les primeres i les últimes línies dels fitxers. Podeu consultar-ne l'ajuda (`man head` o `man tail`) per veure les diverses opcions que tenen.

Taula ressum de les ordres tractades

Ordre	Significat
`pwd`	Mostra el directori on ens trobem actualment
`ls`	Mostra el contingut d'un directori
`ls -l`	Mostra el contingut d'un directori en format llarg
`cd directori_o_ruta`	Accedim al directori indicat
`cd ~`	Accedim al nostre *home*
`cd ..`	Accedim al directori pare de l'actual
`mkdir directori`	Crea un directori
`mkdir -p ruta`	Crea la ruta completa, incloent subdirectoris
`rmdir directori`	Elimina un directori buit
`rm fitxer`	Elimina fitxer
`rm -r directori`	Elimina un directori recursivament
`mv origen destí`	Mou/Renomena un fitxer o directori
`cat fitxer`	Mostra el contingut d'un fitxer
`more fitxer`	Mostra el contingut d'un fitxer paginat

Ordre	Significat
`less fitxer`	Mostra el contingut d'un fitxer navegable
`head fitxer`	Mostra les (10) primeres línies del fitex
`tail fitxer`	Mostra les (10) últimes del fitex

Pràctica 2. Edició de fitxers de text: vim

Quan treballem des de la línia d'ordres, sovint cal editar fitxers, bé per tal de realitzar alguna modificació en un programa o directament, per programar a través de la consola.

En unitats posteriors veurem eines per editar els nostres programes dins un entorn gràfic, però ara anem a fer ús d'un editor de fitxers en mode text: **vim**. Tot i que al principi pot resultar una miqueta costós i *dur*, quan ens acostumem a ell solem ser més productius, ja que ens permet editar fitxers sense moure pràcticament els dits del teclat. A més, en algunes ocasions, com quan ens connectem remotament a un ordinador, no disposem d'eines gràfiques per a l'edició de text, i aquest tipus d'editors ens poden ser de gran ajuda.

Aquesta pràctica es basa en el *webinar Vim, manual de uso básico*[1], però d'una forma més simplificada.

Introducció a vim

Vim és una versió millorada de l'editor *vi*, el primer editor de text a pantalla completa de sistemes Unix.

Si al nostre ordinador no tenim instal·lat *vim*, podem fer-ho amb:

```
$ sudo apt-get update
$ sudo apt-get install vim
```

Quan ja disposem del *vim* instal·lat al sistema, per tal d'accedir només haurem d'invocar l'ordre `vim` des de la terminal. Si no indiquem res, se'ns mostrarà informació sobre algunes ordres bàsiques i sobre la versió. El més habitual, per contra, serà invocar *vim* passant-li el nom de fitxer que volem editar o crear:

```
$ vim fitxer
```

[1] https://openwebinars.net/blog/vim-manual-de-uso-basico/

Si `fitxer` existeix, l'obrirà per editar-lo, i si no, obrirà l'editor en blanc, i guardarà el fitxer quan indiquem l'ordre de guardar.

Modes de Vim

Vim té tres modes principals de treball:

- El **mode ordres**, que ens permet navegar pel document i introduïr ordres a executar dins el propi fitxer (buscar, reemplaçar, guardar). En aquest mode **no s'interpreten les tecles del teclar pels seus caràcters, sinò per les funcions o ordres assignades a cada tecla**. Aquestes ordres seran combinacions de lletres sensible a les majúscules, i podran anar precedides pel nombre de vegades que desitgem repetir l'acció a executar. **Per accedir a aquest mode des de qualsevol altre, polsarem la tecla `Esc` (escape)**.
- El **mode ex**, que ens permet manipular els fitxers (guardar, eixir, etc.). Per accedir a aquest mètode escrivim dos punts **:** seguit pel nom de l'ordre `ex` que volem executar. Després d'executar l'ordre, torna automàticament al mode comando.
- El **mode inserció**, que serveix per afegir text al fitxer. Per tal d'accedir a aquest mode, des del *mode d'ordres*, polsem la tecla `i` (*de insert*) o bé la tecla d'inserció `Ins`. Per tal de tornar de nou al mode d'ordres, utilitzarem la tecla `Esc`.

El mode d'ordres

Les ordres de més utilitat en el mode d'ordres són:

Ordre	Significat
h	Desplaçament cap a l'esquerra (igual que el cursor)
l	Desplaçament cap a la dreata (igual que el cursor)
j	Desplaçament cap avall (igual que el cursor avall)
k	Desplaçament cap amunt (igual queel cursos cap amunt)
w	Desplaça una paraula a la dreta
b	Desplaça una paraula a l'esquerra
0	Mou el cursor al principi de la línia actual
$	Mou el cursos al final de la línia actual
i	Permet **passar a mode d'inserció**, i començar a escriure text en la posició actual del cursor

Ordre	Significat
I	Permet **passar a mode d'inserció**, i començar a escriure text a principi de la línia en què ens trobem
O	Permet **passar a mode d'inserció**, afegint una línia en blanc davant d'on estem, per començar a escriure
o	Permet **passar a mode d'inserció**, afegint una línia en blanc després d'on estem, per començar a escriure
dd	Retalla la línia actual, i la deixa al portapapers
P	Apega el contingut del portapapers en la línia d'abans d'on ens trobem
p	Apega el contingut del portapapers en la línia de després d'on ens trobem
u	Desfà l'última ordre
n	Troba la següent coincidència en una búsqueda
n	Troba la coincidència anterior en una búsqueda
:w	Guarda el fitxer actual. Amb `:w Fitxer` el guarda amb altre nom
:q	Ix de *Vim* si no hi ha canvis pendents al fitxer
:q!	Ix de *Vim* encara que hi hagen canvis pendents al fitxer
:wq	Guarda els canvis i ix de *Vim*
:*número*	Va directament al número de línia indicat

Buscar i reemplaçar text

Per tal de buscar i reemplaçar text en vim, fem ús de `/`, que porta el cursor directament a la ubicació que li indiquem. A més, també podem indicar fent ús de `%s` que reemplace text. De forma ressumida, tenim:

Ordre	Significat
/*patró*	Busca el patró en el fitxer a mesura que escrivim
:/*patró*	Busca el patró en el text. Cal que donem a la tecla *intro* per comencar a buscar

Ordre	Significat
:%s/*patró*/*text*	Busca el patró i el reemplaça pel text indicat, %s indica que busca en tot el document
:%s/*patró*/*text*/gc	Com l'anterior, però ara ens demanarà confirmació

Quan busquem un text per reemplaçar-lo amb `gc` ens demana confirmació, donant diverses opcions:

- `y`: Confirma l'acció
- `n`: No fa la substitució i segueix buscant
- `a`: Confirmem l'acció per a totes le següents coincidències
- `q`: Aturem la búsqueda
- `l`: Confirmem la substitució i parem la búsqueda, tornant al mode editor
- `Ctrl+e`: Avancem cap avall en el document per localitzar el context de la coincidència
- `Ctrl+y`: Retrocedim per localitzar el context de la coincidència

Exercici pràctic

Per tal de *trencar la mà* amb el vim, podeu descarregar algun text o crear-lo vosaltres mateixos, i anar provant les ordres indicades més amunt.

Com a exemple, des de la línia d'ordres podeu executar:

```
$ wget http://tiny.cc/hpckcz -O text.txt
```

I al fitxer `text.txt` disposareu d'una còpia del Quixot en mode text.

Pràctica 3. Llenguatges de programació

A la teoría hem parlat de diversos llenguatges de programació. Tot i que aneu a tindre un mòdul dedicat íntegrament a la programació, anem a veure alguns programes d'exemple en diversos llenguatges, perquè vegeu les diferències principals entre cadascun d'ells.

Llenguatges interpretats: Python i nodejs

Anem a veure un exemple clàssic, sobre com mostar un text per pantalla en diversos llenguatges. Anem a començar en dos llenguatges interpretats: *Python* i *Javascript*, a través de nodejs.

Hello World! en python

Editeu un fitxer de text amb *Vim*, al que anomenarem `hello.py` i el següent contingut:

```
1 # Hola mon en python
2
3 print "Hello World! des de Python"
```

Alguns detalls d'interès:

- L'extensió dels fitxers en python és `.py`.
- La primera línia és un comentari, que comença amb `#`. Amb els comentaris solem documentar els programes, i ens serveixen per donar-nos informació quan llegim el programa, però no interfereixen per a res en l'execució.
- Amb python, per escriure per pantalla, simplement indiquem l'ordre `print`, seguida de la cadena de text a mostrar, tancada entre cometes dobles.

Ara eixim de l'editor i anem a executar el programa. Com que *Python* és un llenguatge interpretat, necessitarem invocar l'intèrpet. Així que simplement farem des de la línia d'ordres (i des de la mateixa carpeta on tinguem el fitxer `hello.py`):

```
1 $ python hello.py
```

I veurem com ens mostra el missatge `Hello World! des de python` per la consola.

Com veiem, no s'ha generat cap altre fiter addicional, ja que *Python* directament interpreta el codi dei fitxer i l'executa.

Hello World! en nodejs

Editem ara un nou fitxer, de nom `hello.js` amb el següent contingut:

```
1 // Hola mon en nodejs/javascript
2
3 console.log("Hello World! des de Javascript");
```

Com veiem, és un codi relativament semblant al de python amb les següents peculiaritats:

- Els comentaris ara s'escriuen amb dues barres (`//`) al principi, en lloc del símbol `#`. Javascript, igual que C, C++ i Java, fa ús d'aquestes barres quan el comentari ocupa una línia, o bé si va a ocupar més d'una línia, es pot indicar entre `/*` i `*/`.
- Ara, per escriure per la consola, en lloc de l'ordre *print*, s'utilitza la funció `console.log`, a la que li passem, entre parèntesi, i entre cometes el missatge a mostrar.

Per tal d'executar aquest programa, necessitarem *nodejs*, invocant-lo des de la línia d'ordres com feiem amb python:

```
$ nodejs hello.js
```

Per obtindre doncs el resultat `Hello World! des de Javascript` per consola.

Llenguatges compilats: C

Anem ara a veure com faríem un programa que realitzara la mateixa funcioanalitat en *C*. Escrivim en un fitxer `hello.c` el següent codi:

```
/* Hola mon en C */

#include <stdio.h>

int main(void)
{
  printf("Hello World! des de C!");
}
```

Veiem que fa variat una miqueta respecte als exemples que hem vist anteriorment. En aquest cas:

- Els comentaris s'afigen igual que en javascript, bé començant amb `//` o com és el cas, entre `/*` i `*/`.
- Hi ha una línia nova, `#include <stdio.h>`, que és una llibrería que ens proporciona la funció `printf`.
- El codi principal del programa està inclòs dins una funció anomenada `main`. La forma de declarar esta funció és amb la línia `int main(void)` (ja veureu el significad de cada cosa amb més profunditat més endavant).
- El contingut de la funció `main`, el que es coneix com *cos de la funció* es troba comprés en un bloc de codi delimitat pels símbols `{` i `}`.

Anem ara a llançar l'aplicació. Si recordem, *C* no és un llenguatge interpretat, sinò compilat. Això vol dir que no podem llançar l'aplicació directament sense haver passat prèviament per un procés de compilació.

El compilador per a *C* és el `gcc`, i s'usa de la següent forma: Des de la línia d'ordres, i en el mateix directori on tenim el fitxer `hello.c`, escriurem:

```
$ gcc hello.c -o hello
```

Tot i que aparentment, no sembla que haja fet res, si fem un `ls` del directori on estem, veurem que tenim un fitxer nou anomenat `hello`, i de color diferent a la resta (el que indica que és un fitxer

executable)

La sintaxi de l'ordre anterior és `gcc fitxer_original_en_C -o fitxer_executable`. És a dir, li passem com a argument el fitxer original en C, i amb l'opció `-o`, li indiquem quin nom volem que tinga el fitxer executable. En cas que no indiquem aquesta opció, el nom per defecte sería `a.out`.

Ara ja només ens queda executar el programa directament:

```
$ ./hello
```

Tinguem en compte que hem d'utilitzar al principi `./`, i no podem invocar directament l'executable amb el seu nom. Amb açò li diem que busque l'executable en el directori actual, ja que en cas contrari, els executables només els busca en els directoris on *se suposa* han d'extar aquesta (*/usr/bin*, */usr/sbin/*, etc)

Llenguatges compilats i interpretats: Java

Com a apartat final, anem a veure el cas especial de Java. Escrivim un nou fitxer de text amb el següent contingut:

```
// Hola mon en Java

class HolaMon {
  static public void main( String args[] ) {
    System.out.println( "Hello World! des de Java!" );
  }
}
```

Respecte del codi, de moment, podem comentar:

- Els comentaris s'expressen igual que en C, C++ o Javascript (`//` i `/**/`)
- També tenim una funció `main`, que conté el codi executable principal (bàsicament la crida a la funció que imprimeix el missatge). Igual que en C, el cos d'aquesta funció es troba entre *claudàtors* `{...}`. El **`public static void`** de davant ho veurem més clar quan introduïm conceptes d'orientació a objectes.
- Fixem-nos també, que aquesta funció *main*, es troba dins altre *bloc de codi {..}*. Aquest bloc, té davant la línia **`class`** `HolaMon`, el que indica que estem definint el que es coneix com una *classe*. Més endavant veurem que Java és un llenguatge puramet obtectats a objecte.
- Un aspecte important a tenir en compte és que aquesta *classe* s'ha d'anomenar exactament igual que el nom del fitxer, sense l'extensió `.java`.

Anem a realitzar ara la *compilació* d'aquest programa. Per a això, farem ús del *Compilador de Java* (`javac`), inclòs al JDK (*Java Development Kit). Per a això, des del directori on tenim el fitxer `.java` escrivim:

```
$ javac Hello.java
```

Com veurem, aparentment, tampoc crea res, però si veiem el contingut del directori, veurem que tenim un fitxer nou `HolaMon.class`, aparentment també executable. Si intentem llançar-lo igual que amb el programa en C, obtindrem un error:

```
$ ./HolaMon.class
bash: ./HolaMon.class: Permission denied
```

Anem a veure una ordre nova de l'intèrpret d'ordres: l'ordre `file`, que ens indica el tipus de fitxer que és el fitxer que li passem com a argument. Així, podem preguntar pels fitxers anteriors:

- Per al fitxer `hello` compilat de *C*, ens diu que és un executable de 64 bits:

```
$ file hello
hello: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
    dynamically linked, interpreter /lib64/l, for GNU/Linux 3.2.0,
    BuildID[sha1]=9c9022147e874630813b5b4e0cad2f9ac59a867e, not stripped
```

- Per al fitxer de *Python* o qualsevol altre font, ens indicarà, evidentment que es tracta de fitxers de text:

```
$ file hello.py
hello.py: ASCII text
```

I veiem ara què ens diu per al fitxer que acabem de generar:

```
$ file HolaMon.class
HolaMon.class: compiled Java class data, version 55.0
```

Com veiem, ens indica que és un fitxer de classe compilat en Java, però no diu que siga executable. Recordeu que Java necessitava d'una compilació prèvia per generar llenguatge *bytecode*, en lloc de llenguatge màquina. Amb `javac` el que hem fet és generar aquest *bytecode*, que podrà se executat dins la màquina virtual de java (JVM). Per a això, farem ús de l'ordre `java` passant-li *el nom de la classe* que hem definit, que és igual que el nom del fitxer, sense l'extensió `.java`:

```
$ java HolaMon
Hello World! des de Java!
```

Com a activitat, podeu anar *trastejant* amb els exemples, i provar de canvia el missatge que us mostra per pantalla.

Pràctica 4. El llenguatge Markdown

Markdown es defineix com un **llenguatge de marques lleuger**, amb la finalitat d'escriure utilitzant un format de text pla, fàcil d'escriure i de llegir, i que puga convertir-se a altres formats, com HTML, epub o pdf.

A la web (https://markdown.es/)[https://markdown.es/] disposeu de molta informació i un vídeo bastant bo sobre aquest llenguatge i algunes eines que us faciliten la vida a l'hora de generar documents.

Amb Markdown podem formatar el text amb lletres cursives, negretes, capçaleres o enllaços utilitzant únicament text pla, el que fa que l'escriptura siga més simple i eficient, ja que ens evita haver d'estar pensant en el format i ens permet centrar-nos només en el contingut. La seua facilitat d'us i sintaxi clara, junt amb que permet incloure i formatar codi font, fa que siga le llenguatge per excel·lència en llocs com Gihtub per tal de documentar els projectes que allotja.

Veiem un ressum de la sintaxi principal:

- **Capçaleres**

```
# Capçalera de primer nivell
## Capçalera de segon nivell nivell
### Capçalera de tercer nivell nivell
...
```

- **Formatació de text**

```
*Text en cursiva* (compte amb no posar espais darrere el primer * d'inici o davant el * del final)
**Text en negreta** (compte també amb els espais)
***Text en cursiva i negreta***
```

- **Llistes**

```
* Item 1 en llista desordenada
* Item 2 en llista desordenada
* ...
```

```
1. Item 1 en llista ordenada
2. Item 2 en llista ordenada
* ...
```

- **Taules:** Les taules han de tindre una capçalera i un cos, i la seua sintaxi és la següent:

```
| Camp 1 capçalera | Camp 2 capçalera |
|------------------|------------------|
```

```
| Valor            | Valor            |
| Altre valor | Altre valor |
```

Cal tindre en compte que podem afegir tants camps com volguem. A més, la línia que separa la capçalera del cos `|---|---|` és obligatòria, però no és necessari que tinga tants caràcters com tinguen les capçaleres.

- **Fragments de codi**

Els fragments de codi els podem posar bé de forma `inline`, escrivint en la mateixa línia el codi que volem escriure tantat entre `'accents oberts'`, o bé si es tracta d'un fragment de codi, podem fer:

```
'''
línia 1
línia 2
etc
'''
```

Per a més informació podeu consultar:

- La guía de markdown en espanyol: https://markdown.es/
- La sintaxi estesa de markdown a la web: https://www.markdownguide.org/extended-syntax