SISTEMES OPERATIUS
SESSIÓ 2

PRIMERA PART: Continuant amb l'entorn Unix

VARIABLES D'ENTORN

Els intèrprets de comandes permeten definir un seguit de variables i assignar valors a aquestes variables. Les variables són de tipus alfanumèric.

Els programes poden consultar el valor d'alguna variable i, en funció d'aquest, poden modificar el seu comportament. Per exemple, la variable `TERM` indica el tipus de terminal que esteu utilitzant. En funció del valor d'aquesta variable, un programa pot presentar els seus resultats de forma més o menys sofisticada (per exemple, utilitzant colors,...).

Per veure totes les variables definides actualment podeu utilitzar la comanda `set`. Si voleu veure el valor d'una única variable podeu utilitzar la comanda `echo` i el metacaràcter `$` (indica que fem referència a una variable). Per exemple, `echo $TERM`

Per definir o modificar el valor d'una variable cal fer servir l'operador **=**. Per exemple:

`prompt$` **`a=hola; echo $a`** Assigna a la variable `a` la paraula `hola`. Fixeu-vos que no hi ha cap espai entre el nom de la variable, el signe `=` i el valor. El metacaràcter **`;`** és un separador de comandes (equivalent a executar la primera comanda i a continuació la segona comanda).

Prova: executeu (i mireu les diferències) aquestes 4 comandes (**`echo $a`, `echo a`, `echo 'a'`, `echo '$a'`**)

`prompt$` **`b=(pri seg ter); echo ${b[*]}; echo ${b[1]}`** Inicialitza la variable `b` com un vector. El primer `echo` mostra el contingut sencer del vector i el segon mostra el contingut del segon element (la primera posició del vector és la 0).

A continuació es detalla el significat d'algunes de les variables més usuals:

`PATH`: Conté la llista de directoris on el *shell* busca els fitxers executables corresponents a les comandes. La cerca es realitza en l'ordre en què els directoris estan emmagatzemats a la variable; la cerca finalitza en el moment que es troba un directori que contingui l'executable; si no s'en troba cap, el *shell* mostra un missatge d'error com ara *Command not found*. Per afegir un directori a aquesta llista podeu executar **`PATH=$PATH:nou_directori`**

`TERM`: Indica el tipus del terminal en ús, per exemple `vt100`, `xterm`,...

`HOME`: Conté el directori d'usuari.

Per veure la llista completa de variables, feu servir **`env`**

Consideracions finals sobre variables

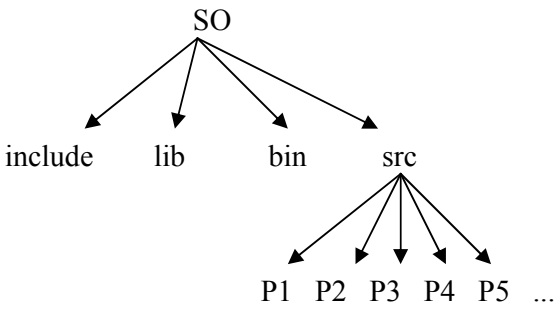
És habitual que, a l'entrar al sistema, ens interessi que les variables tinguin un valor diferent al valor per omissió. Per fer aquests canvis automàticament podem aprofitar el fitxer `~/.bashrc`. Les comandes que hi siguin presents s'executaran automàticament cada cop que iniciem un intèrpret de comandes `bash`.

Hi ha un fitxer s'aquest tipus per cada shell. Com vosaltres teniu definit per defecte el shell tcsh, podeu fer un fitxer `~/.tcshrc`. Si voleu, podeu modificar-lo (o crear-lo si no el teniu) i fer que executi `bash` a l'entrar de manera automàtica.

Si voleu que les variables siguin heretades pels processos creats des del *shell*, cal utilitzar la comanda `export`. Consulteu el manual del sistema.

Per esborrar una variable cal utilitzar la comanda `unset`.

Exercici: Crea aquest arbre de directoris a partir del teu HOME al servidor *albanta*:



D'ara endevant, aquesta serà l'estructura de directoris que farem servir a les pràctiques. Modifica el .bashrc per tal que *~/SO/bin* estigui sempre al teu path.

REDIRECCIÓ D'ENTRADA/SORTIDA

A Unix, els processos no llegeixen directament del teclat ni escriuen directament a pantalla, sinó que ho fan sobre uns dispositius intermitjos anomenats entrada estàndar (*stdin*) i sortida estàndar (*stdout*). Per omissió, aquests dispositius intermitjos fan referència al teclat i a la pantalla respectivament, però des de l'intèrpret de comandes podeu modificar aquesta associació (redireccionar els canals estàndar).

El metacaràcter < redirecciona l'entrada estàndar a un fitxer; > redirecciona la sortida estàndar a un fitxer, >> afegeix la sortida estàndar al contingut actual d'un fitxer.

A mode d'exemple, podeu executar la comanda `ls` redireccionant la seva sortida estàndar al fitxer `tmp` i després executar la comanda `wc` (compta el nombre de caràcters, paraules i línies) fent que la seva entrada estàndar sigui `tmp`.

```
prompt$ ls /bin > tmp
propmpt% ls
... tmp ...
prompt$ wc < tmp
     99    99    606
prompt$
```

Els missatges d'error de les comandes no s'escriuen a la sortida estàndar sinó que s'escriuen a un dispositiu anomenat canal d'error (*stderr*). Per omissió, aquest dispositiu està associat a la pantalla però el podem redireccionar al fitxer que vulgueu amb els metacaràcters `2>`. Per exemple, **`ls KKKK`** mostra un missatge d'error; **`ls KKKK 2> err`** redirecciona el missatge al fitxer `err`; finalment, **`ls -l KKKK 2> /dev/null`** es desfà del missatge d'error ja que `/dev/null` és un dispositiu que actua com una paperera: fa desaparèixer tota la informació que hi enviem.

SEGONA PART: compilació i muntatge

Aneu al directori **P2** penjant de **src** i crea allà el fitxer **main1.c** amb el següent contingut:

```
#include <stdio.h>
#include <string.h>

#define TRUE -1
#define FALSE 0

int string_a_enter(char *p)
{
        int i;
        int final=FALSE;

        res=0;
        while(!final) {
                if (p[i]=='\0') final=TRUE;
                        else {
                                res=res*10+(p[i]-'0');
                                i++;
                                }
                }
        return(res);
}

int main()
{
char buffer[256];
int enter;
char *frase="1234";

        enter=string_a_enter(frase)
        sprintf(buffer,"Cadena de caràcters original= %s, valor= %d\n", frase, enter);
        write(1,buffer,strlen(buffer));
}
```

Nota: el programa l'heu d'escriure vosaltres per practicar amb l'editor vi. Un cop hagueu escrit el programa, mireu com està indentat.
Passeu-li l'eina indent (**indent main1.c**) i mireu com ha quedat. Volem que tots els fitxers C que ens lliureu passin per aquesta eina (i us serà molt util per organitzar correctament el codi).

Compilant i muntant

Observeu que el programa té una rutina anomenada **string_a_enter**, que rep com a paràmetre un punter a caràcter (s'assumeix que és un string). Analitzeu que fa tant el programa principal com la rutina. Quan tingueu clar l'objectiu del main i la rutina procedirem a compilar-la i muntar-la (*link*).

Compte!, aquesta rutina té tres errors, part de l'exercici consisteix en trobar-los.

Per compilar, farem servir el compilador de C de GNU, feu **gcc -c main1.c**

Explicació de la comanda: el flag de compilació '-c' indica al compilador que només volem generar l'objecte del fitxer a compilar (*main1.c*). El nom del fitxer objecte serà el del fitxer font amb la extensió '.o' (en aquest cas *main1.o*). Al final d'aquest enunciat hi ha un apèndix on podeu trobar tots els flags de compilació de gcc.

En aquest exemple, tenim uns errors de compilació. Fixeu-vos que el compilador sempre indica on ha trobat l'error (número de línia) i una petita explicació de l'error. **Acostumeu-vos a llegir MOLT ACURADAMENT la explicació**, doncs amb una mica d'experiència podreu solucionar els errors ràpidament.

Trobeu els errors i solucioneu-los. N'hi ha dos errors en temps de compilació (que detecta el compilador) i un en temps d'execució (el compilador no el pot detectar)
Un cop heu solucionat els dos errors de compilació, la comanda gcc -c main1.c haurà generat el fitxer main1.o Per generar un executable, cal *linkar* (muntar) amb les llibreries de llenguatge i sistema corresponents. Es pot invocar el linker directament (el programa **ld** en el nostre entorn) o bé dir-li al compilador que ho faci. Hem escollit fer-ho d'aquesta segona manera.

Per tant, per *linkar*, feu

```
> gcc main1.o
```

invoca al *linker* per generar un executable. Si no se l'indica el contrari, l'executable rebrà el nom 'a.out'. (Comproveu-lo) Si volem assignar-li un nom a l'executable, s'ha d'indicar amb el flag '-o nom'

```
> gcc main1.o -o m1
```

generarà un executable amb el nom 'm1' (comproveu-lo)

Nota: el compilador permet fer la compilació i el muntatge en un únic pas, fent gcc -o m1 main.c tot i això l'hem explicat en dues comandes perquè veieu els passos lògics.

Executeu m1. Surt algun error? Si no, editeu el programa i dupliqueu les línies del main:

```
enter=string_a_enter(frase);
sprintf(buffer,"Cadena de caràcters original= %s, valor= %d\n", frase, enter);
write(1,buffer,strlen(buffer));
enter=string_a_enter(frase);
sprintf(buffer,"Cadena de caràcters original= %s, valor= %d\n", frase, enter);
write(1,buffer,strlen(buffer));
```

en principi hauria de fer dues vegades el mateix, no? compileu, *linkeu* i executeu. Busqueu l'error i arregleu-lo.

Fent mòduls

Copieu el **main1.c** al fitxer **aux.c**. Editeu el nou fitxer i deixeu només el codi de la rutina **string_a_enter**. Si compileu (gcc -c aux.c) veureu que dona error (analitzeu perquè abans de continuar llegint).
Ja haureu observat que falten les definicions de TRUE i FALSE. Es podrien incloure al propi fitxer aux.c, però el que farem és definir un fitxer de capçalera (*header*) **constants.h**, que contingui les definicions de TRUE i FALSE (i que més endavant ampliarem amb més constants).
Editeu aquest fitxer de manera que quedi així

```
>cat constants.h
#define TRUE -1
#define FALSE 0
```

Si ara obriu aux.c i afegiu com a primera línia *#include "constants.h"* i compileu, veureu que no dona cap error i genera un aux.o
Per poder generar ara el main1.c, heu de generar un fitxer de capçalera amb la definició de la rutina, de manera que quedi

```
>cat aux.h
int string_a_enter(char *p);
```

Si ara editem main1.c de manera que esborrem el codi de la rutina string_a_enter i afegim un #include "aux.h" podrem compilar

```
>gcc -c main1.c
>gcc -o m2 main1.o aux.o
```

Generarà un executable m2 (que farà el mateix que m1) però programat de manera modular. Executeu-lo.

Organització de directoris

Quan es fa un projecte mitjanament gran (no és el cas d'aquestes pràctiques, però no importa) es interessant organitzar la zona de treball. La idea és que tingueu els fitxers de *headers* al subdirectori include, els fonts a src (source), les llibreries a lib i els executables a bin. A més, a src mantindrem la zona de treball de cada sessió de laboratori.

Moveu els fitxers de capçalera al directori include (Nota: moure, NO copiar. No deixeu còpia dels .h al directori on són els .c, no és la idea). Si ara compilem de nou:

```
> gcc -c main1.c
```

Veurem que es queixa de no trobar els fitxers de capçalera. Per afegir un directori nou a la llista de directoris on gcc busca els includes, amb el flag -I*dir* (millor indiqueu el *path* des del vostre directori principal $HOME)

```
> gcc -I$HOME/SO/include -c main1.c
```

```
> gcc -I$HOME/SO/include -c aux.c
> gcc -o m2 main1.o aux.o
```

Com podeu veure, la feina de compilar i linkar els programes comença a resultar llarga. A més, calia tornar a compilar aux.c?. Si el programa consta de molts fitxers, no és qüestió de recompilar-los tots cada vegada, ni de portar una llista actualitzada de quins s'han modificat i quins no, o de picar totes les comandes necessàries per generar un executable. Per fer tot això de manera automàtica, hi ha una utilitat anomenada **make** (que veurem en una propera sessió)

Modifica la rutina **string_a_enter** de manera que faci el següent:

```
int string_a_enter(char *cadena, int *numero);
```

La rutina tradueix l'string *cadena*, a un enter *numero*. Aquesta nova rutina admet números negatius (en aquest cas, el primer caràcter de l'string és '-'). A més retorna un zero si la cadena és correcta (només caràcters numèrics, amb la possible excepció del primer, que pot ser el signe menys) Si la cadena és incorrecta retorna un -1.
Quan funcioni, modifica el programa principal de manera que l'string que es passi a la rutina **string_a_enter** sigui argument que estigui a la línia de comandes. El main imprimirà mateixa frase dels apartats anterior només si la cadena és correcta. Altrament imprimirà un missatge d'error.

```
>m3 1234
Cadena de caràcters original= 1234, valor= 1234
>m3 1x34
La cadena conté errors
```

(s'ha de fer servir argv i argc)

Muntant una llibreria pròpia

La tècnica de desenvolupar un programa en diversos fitxers permet la reutilització d'aquests fitxers per altres programes. Habitualment, els codis que es volen reutilitzar es solen agrupar en **biblioteques** de funcions (també anomenades llibreries degut a una mala traducció)

La eina **ar** permet gestionar llibreries (a l'apèndix teniu la llista d'opcions, feu man ar per més informació). Per crear la llibreria (que sol tenir la terminació **.a**) farem

ar -r lib*nomllibreria*.a llista_d_objectes_a_incloure

per exemple proveu,

```
> ar -r libaux.a aux.o
```

Fixeu-vos que el directori resultant és libaux.a, encara que hem parlat de lib*nom*.a, per tant la llibreria **aux** es guarda al fitxer **lib*aux*.a** (després veurem la importància d'això)

Hi han varies formes de veure el contingut d'una llibreria, però potser la més pràctica és fer servir nm -g llibreria. Per exemple, proveu **nm -g libaux.a**. Mireu el man per veure què fa nm i en especial el flag –g.

Per últim, caldria moure la llibreria al seu directori ($HOME/SO/lib). Feu un mv i porteu-la al seu lloc.

I com es compila amb una llibreria? Hi ha dos flags del gcc interessants: -L*directori*, que indica que es busquin llibreries al *directori* indicat i -l*nom*, que indica el *nom* de la llibreria a consultar.

Aquí és on intervé el nom, doncs la llibreria que hi ha a libaux.c és la llibreria aux, i com a tal s'ha d'indicar (proveu-lo):

```
> gcc -o m3 main1.o -L$HOME/SO/lib -laux
```

Apèndix

Flags del compilador de C

-c Elimina la fase de muntatge (*link*). No s'esborren els fitxers objecte produïts (sense aquesta opció, els fitxers objecte s'esborren)
-o *nom* Genera un executable anomenat *nom*. Sense aquesta opció l'executable rebrà el nom *a.out* Aquesta opció no té sentit combinada amb -c
-I*dir* Inclou el directori *dir* a la llista de directoris on buscar els fitxers d'include
-L*dir* Inclou el directori *dir* a la llista de directoris on buscar les llibreries
-l*nom* Munta amb la llibreria *libnom.a* (o *libnom.so*)

Opcions de la eina ar

-d esborra els fitxers indicats a la llibreria
-r reemplaça els fitxers indicats a la llibreria (i si no existeix, la crea)
-t imprimeix l'índex de la llibreria
-x extreu els fitxers indicats de la llibreria
-q inclou els fitxers indicats (i si no existeix, la crea). La diferència amb -r és que –q afegeix mentre que -r reemplaça si troba algun fitxer amb el mateix nom