COMANDOS

Sessió 1:

man: accedeix a manuals on-line

ls: mostra contingut del directori

- **-l:** List in long format.
- **-a:** Include directory entries whose names begin with a dot (.). -> mostra arxius ocults

alias: define un nombre alternativo a un comando.

- #alias ls='ls -la' (-l i -a junts)

mkdir: crea directorio

rmdir: elimina directorio vacío

mv: canvia nom del fitxer o el mou a un altre directori

- **-i:** Cause mv to write a prompt to standard error before moving a file that would overwrite an existing file. If the response from the standard input begins with the character 'y' or 'Y', the move is attempted. (The -i option over-rides any previous -f or -n options.)

cp: copia fitxers i directoris

- **-i:** Cause cp to write a prompt to the standard error output before copying a file that would overwrite an existing file. If the response from the standard input begins with the character 'y' or 'Y', the file copy is attempted. (The -i option overrides any previous -n option.)

rm: borra fitxers o directoris

- **-i:** Request confirmation before attempting to remove each file, regardless of the file's permissions, or whether or not the standard input device is aterminal. The -i option overrides any previous -f

options.

echo: visualitza un text (pot ser una variable d'entorn)

less: mostra fitxer en un temany apte per terminal

cat: concatena ficheros y los muestra en su salida estándar

grep: busca text (o patrons de text) en fitxers

- #grep hola test test1 test2 test3 test4

gedit: editor (sublime mola més)

env: ejecuta un comando en un entorno modificado, si no se le pasa comando, muestra el entorno

chmod: modifica els permisos d'accés a un fitxer

which: localitza un comando

help: ofrece información sobre comandos internos de la Shell

- usar con: help, export, cd, alias
- export NOMBRE_VARIABLE=valor (sin espacios).

en man bash:

(nota: el carácter "/" sirve para buscar patrones en las páginas del man. Utilízalo para encontrar directamente la descripción de estas variables)

HOME: (`echo &HOME`) The home directory of the current user; the default argument for the cd builtin command. The value of this variable is also used when performing tilde expansion.
PATH: (`echo &PATH`) The search path for commands. It is a colon-separated list of directories in which the shell looks for commands. A zero-length (null) directory name in the value of PATH indicates the current directory. A null directory name may appear as two adjacent colons, or as an initial or trailing colon. The default path is system-dependent, and is set by the administrator who installs bash.
PWD: (`echo &PWD`) The current working directory as set by the cd command.

Sessió 2:

make: Utilidad para automatizar el proceso de compilación/linkaje de un programa o grupo de programas

gcc: compilador de C en GNU

- **-c:** Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file. By default, the object file name for a source file is made by replacing the suffix .c, .i, .s, etc., with .o. Unrecognized input files, not requiring compilation or assembly, are ignored.
- **-o:** (`-o file`) Place output in file file. This applies to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code. If -o is not specified, the default is to put an executable file in a.out, the object file for source.suffix in source.o, its assembler file in source.s, a precompiled header file in source.suffix.gch, and all preprocessed C source on standard output.
- **-I:** includes
- **-L, -l:** libraries

print: conversión de formato almacenándola en un búffer

- printf, fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf, vdprintf, vsprintf, vsnprintf - formatted output conversion

```
 #include <stdio.h>
 int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int dprintf(int fd, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
```

atoi: convierte un string a numero entero
indent: indentación de ficheros fuente

Sessió 3: processos

```
int main (int argc,char *argv[])
{
    ...
    if ((pid=fork())<0)  error_y_exit("Error en fork",1);
    ...
}
void error_y_exit(char *msg,int exit_status)
{
    perror(msg);
    exit(exit_status);
}
```

> Además, en caso de que el error sea crítico, como por ejemplo que falle un fork o un execlp, tiene que terminar la ejecución del programa.

getpid: `pid_t getpid(void); pid_t getppid(void);`

- retorna pid del procés que executa
- llibreries: `#include <sys/types.h>, #include <unistd.h>`
- RETURN VALUE: (mai error)

> getpid() -> returns the process ID of the calling process.
> getppid() -> returns the process ID of the parent of the calling process.

fork: `pid_t fork(void);`

> crea un proceso nuevo, hijo del que ejecuta el fork
> llibreries: `#include <unistd.h>`
> The child's parent process ID is the same as the parent's process ID.
> The child does not inherit timers from its parent (alarm(2)…).
> The termination signal of the child is always SIGCHLD.
> RETURN VALUE:
> - Success: the PID of the child process is returned in the parent and 0 is returned in the child.
> - Failure: -1 is returned in the parent no child process is created and errno is set appropriately.

exit: `void exit(int status);`

> acaba el procés que executa la crida
> llibreries: `#include <stdlib.h>`
> exit() causes normal process termination and the value of status is returned to the parent
> RETURN VALUE: The exit() function does not return.

execlp: extern char **environ;

> ```
> int execl(const char *path, const char *arg, .../* (char *) NULL */);
> int execlp(const char *file, const char *arg, .../* (char *) NULL */);
> int execle(const char *path, const char *arg, .../*, (char *) NULL, char *
> const envp[]*/);
> int execv(const char *path, char *const argv[]);
> int execvp(const char *file, char *const argv[]);
> int execvpe(const char *file, char *const argv[], char *const envp[]);
> ```
>
> En **path** debemos pasar la ruta del ejecutable
> **file:** Busca el archivo ejecutable en todos los directorios especificados por PATH.
> **RETURN VALUE:** Esta función retorna -1 en caso de error, en caso contrario no retorna.
> No retorna ya que hemos sustituido el contexto del programa por el de otro.
>
> Debemos pasar los argumentos para el nuevo programa a ejecutarse en *arg.
> Se heredan los descriptores de ficheros abiertos
> Todas las señales pasaran a la acción por defecto.
> Ejecuta un programa en el contexto del mismo proceso.
> Llibreries: `#include <unistd.h>`
> The exec() family of functions replaces the current process image with a new process image.
> RETURN VALUE: The exec() functions return only if an error has occurred.

> The return value is -1, and errno is set to indicate the error.

perror: `void perror(const char *s);`

> escribe un mensaje del último error
> `llibreries: #include <errno.h>`

ps: (comando)

> devuelve la información de los procesos
> **-a:** Select all processes associated with a terminal.
> **-u:** (userlist) Select by effective user ID or name. This selects the processes whose effective user name or ID is in userlist.
> **-o:** (format) User-defined format. format is a single argument in the form of a blank-separated or comma-separated list, which offers a way to specify individual output columns.

proc: (directoy) proc - process information pseudo-filesystem

> `/proc/[pid]/cmdline` :
> This read-only file holds the complete command line for the process, unless the process is a zombie. In the latter case, there is nothing in thisfile: that is, a read on this file will return 0 characters. The command-line arguments appear in this file as a set of strings separated by null bytes ('\0'), with a further null byte after the last string.
>
> `/proc/[pid]/cwd`
> This is a symbolic link to the current working directory of the process. To find out the current working directory of process 20, for instance, you can do this: `$ cd /proc/20/cwd; /bin/pwd`
>
> `/proc/[pid]/environ`
> This file contains the environment for the process. The entries are separated by null bytes ('\0'), and there may be a null byte at the end. Thus, to print out the environment of process 1, you would do: `$ strings /proc/1/environ`
>
> `/proc/[pid]/exe` Under Linux 2.2 and later, this file is a symbolic link containing the actual pathname of the executed command.
>
> `/proc/[pid]/stat`
> Status information about the process.
> (1) pid %d The process ID.
> (2) comm %s The filename of the executable, parentheses.
> (3) state %c:
> R Running
> S Sleeping in an interruptible wait
> D Waiting in uninterruptible disk sleep
> Z Zombie
> T Stopped (on a signal) or trace stopped
> t Tracing stop
> W Paging
> X, x Dead
> K Wakekill
> W Waking

P Parked
(4) ppid %d The PID of the parent of this process.
(5) pgrp %d The process group ID of the process.
(6) session %d The session ID of the process.
(7) tty_nr %d The controlling terminal of the process.
(8) tpgid %d The ID of the foreground process group of process' controlling terminal.
(9) flags %u The kernel flags word of the process.
(10) minflt %lu …
(11) cminflt %lu …
(12) majflt %lu …
(13) cmajflt %lu …
(14) utime %lu: Amount of time that this process has
been scheduled in user mode, measured in
clock ticks.
(15) stime %lu: Amount of time that this process has
been scheduled in kernel mode, measured
in clock ticks.
(16) cutime %ld: Amount of time that this process's
waited-for children have been scheduled
in user mode, measured in clock ticks
(17) cstime %ld: Amount of time that this process's
waited-for children have been scheduled
in kernel mode, measured in clock ticks.
(18) priority %ld
(19) nice %ld The nice value
(20) num_threads %ld Number of threads in this process
(21) itrealvalue %ld
The time in jiffies before the next
SIGALRM is sent to the process due to an
interval timer.
(22) starttime %llu
The time the process started after sys-
tem boot.
(23) vsize %lu Virtual memory size in bytes.
(24) rss %ld Resident Set Size: number of pages the process has in real memory.
(25) rsslim %lu Current soft limit in bytes on the rss of the process
(26) startcode %lu The address above which program text can run.
(27) endcode %lu The address below which program text can run.
(28) startstack %lu The address of the start of the stack.
(29) kstkesp %lu The current value of ESP (stack pointer)
(30) kstkeip %lu The current EIP (instruction pointer).
(31) signal %lu The bitmap of pending signals, displayed as a decimal number.
(32) blocked %lu The bitmap of blocked signals, displayed as a decimal number.
(33) sigignore %lu The bitmap of ignored signals, displayed as a decimal number
(34) sigcatch %lu The bitmap of caught signals, displayed as a decimal number.
(35) wchan %lu This is the "channel" in which the process is waiting.
…

Sessió 4: Signals

The child does not inherit timers from its parent
Acciones que pueden realizar los signals:
- Ignorar evento (no todos son ignorables)
- Realizar acciñón por defecto
- Ejecutar una función que el proceso haya definido (sigaction)
Funciones de tratamiento de signal:

```
void nombre_funcion(int numero_de_signal_recibido);
```

Quan es rep un signal, el sistema passa a executar el tractament. Si el tractament és una funció, aquesta rep num del signal com a parametre. -> una funció pot tractar diferents signals.

sigaction: Reprograma la acción asociada a un evento concreto.

```
#include <signal.h>;
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

signum: specifies the signal and can be any valid signal (sigkill i sigstop no)
act: If act is non-NULL, the new action for signal signum is installed from act.
oldact: If oldact is non-NULL, the previous action is saved in oldact.
sigaction() system call: change the action taken by a process on receipt of a specific signal.

```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
};
```

SA_RESETHAND: Restore the signal action to the default upon entry to the signal handler. This flag is meaningful only when establishing a signal handler.

kill: Envía un evento a un proceso (kill - send a signal to a process)

kill -KILL
(comando)
< pid > […]: Send signal to every listed.
-< signal >
The default signal for kill is TERM.

sigsuspend: Bloquea el proceso que la ejecuta hasta que recibe un signal (no ignorado).

```
#include <signal.h>
int sigsuspend(const sigset_t *mask);
```

Description: sigsuspend() temporarily replaces the signal mask of the calling process with the mask given by

mask (variable) and then suspends the process until delivery of a signal whosIf pid is positive, then signal sig is sent to the process with the ID specified by pid.

If pid equals 0, then sig is sent to every process in the process group of the calling process.
If pid equals -1, then sig is sent to every process for which the calling process has permission to send signals, except for process 1 (init), but see below.
If pid is less than -1, then sig is sent to every process in the process group whose ID is -pid.
e action is to invoke a signal handler or to terminate a process.
- signal terminates: not return
- signal mask restored to the state before the call
(NOT: SIGKILL or SIGSTOP)
**Return value: **sigsuspend() always returns -1 (with errno)

sigprocmask: Permite modificar la máscara de signals bloqueados del proceso

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```
(how)SIG_BLOCK: The set of blocked signals is the union of the current set and the set argument.
(how)SIG_UNBLOCK: The signals in set are removed from the current set of blocked signals. It is permissible to attempt to unblock a signal which is not blocked.
(how)SIG_SETMASK: The set of blocked signals is set to the argument set.

mask = vector amb processos marcats.
normalment fem: `int sigprocmask(SIG_BLOCK, mask, NULL);`

alarm: programa el envío de un signal SIGALRM al cabo de n segundos.

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```
If seconds is zero, any pending alarm is canceled.
In any event any previously set alarm() is canceled
RETURN VALUE: number of seconds remaining until any previously scheduled alarm or 0 if there was no previously scheduled alarm.

sleep: función de C que bloquea proceso durante el tiempo que pasa por su parámetro.

```
sleep NUMBER[SUFFIX]...
```
suffix: s (seconds), m (minutes), h (hours), d (days)

kill: Envía un evento concreto a un proceso
(syscall codi)

```
#include <sys/types.h> #include <signal.h>
int kill(pid_t pid, int sig);
```
- pid > 0: sig sent to process with pid
- pid == 0: sig sent to every process in the process group of the calling process
- pid == -1: sig sent to every process which the calling process has permission to send signals
- pid < -1: sig sent to every process in the process group whose ID is -pid.
- sig == 0: no signal sent (error checking stil performed)
RETURN VALUE: sucess -> 0, error -> errno (no signal sent)

ps: mostra informació sobre els processos del sistema

> **-o** pid: format
> **s:** Display signal format
> Here are the different keywords that may be used to control the output format (e.g. with option -o) or to sort the selected processes with the GNU-style –sort option. For example: ps -eo pid,user,args –sort user This version of ps tries to recognize most of the keywords used in other implementations of ps. The following user-defined format specifiers may contain spaces: args, cmd, comm, command, fname, ucmd, ucomm, lstart, bsdstart, start.
> **cmd:** command with all its arguments as a string. Modifications to the arguments may be shown. The output in this column may contain spaces. A process marked < defunct > is partly dead, waiting to be fully destroyed by its parent. Sometimes the process args will be unavailable; when this happens, ps will instead print the executable name in brackets. (alias cmd, command).
> **time:** cumulative CPU time, "[DD-]HH:MM:SS" format. (alias cputime).

waitpid: espera finalització d'un procés

> #include <sys/types.h>, #include <sys/wait.h>``
> pid_t wait(int *status); pid_t waitpid(pid_t pid, int *status, int options);`
> wait(&status) = waitpid(-1, &status, 0) <- equivalents

```
exit(0) //fin hijo
int status;
waitpid(&status); //padre espera y status = x (exit(x))
if (status == 0)...
else ...
```

> The value of pid can be -> wait for…
> - pid < -1 any child process (group == id)
> - pid == -1 any child process
> - pid == 0 any child process (group ID == that of the calling process)
> - pid > 0 process with that pid
> RETURN VALUE
> wait(): success -> PID terminated child , error -> -1
> waitpid(): success -> returns the PID of the child whose state has changed; if WNOHANG was specified and one or more child(ren) specified by pid exist, but have not yet changed state, then 0 is returned. (WNOHANG: return immediately if no child has exited), error -> -1
> Each of these calls sets errno to an appropriate value in the case of an error.

SIGNALS:

> #include <signal.h>
> sigalrm (alarma, temporizador)
> sigchld (fi de un procés fill)
> sigusr1 / sigusr2 (sense significat previ)

Sessió 5: Memòria

gcc: compilador de c

> **-static:** On systems that support dynamic linking, this overrides -pie and prevents linking with the shared libraries. On other systems, this option has no effect.

nm: comando que muestra la tabla de símbolos del programa (variables globales y funciones)

> `nm + ejecutable`

objdump: `objdump [options] <input object files>`

> comando que muestra información sobre el fichero objeto (.o)
> **-d:** Display the assembler mnemonics for the machine instructions from objfile. This option only disassembles those sections which are expected to contain instructions.

`/proc/[pid]/maps:` A file containing the currently mapped memory regions and their access permissions.

malloc: función de C que valida una región de memoria lógica (reserva)

> `(void *)malloc(size_t size);` <- sizeof(type)
> `#include <stdlib.h>`
> RETURN VALUES: successful: return pointer to allocated memory, error: NULL to pointer and set errno.

free: función de C que libera una región de memoria lógica

> `void free(void *ptr);` <- punter que retorna el malloc
> `#include <stdlib.h>`
> RETURN VALUES: The free(…) function does not return a value.

sbrk: llamada al sistema que modifica el tamaño de la sección de datos.

> `void *sbrk(int incr);` <- sizeof(type)
> `#include <unistd.h>`
> RETURN VALUES: successful: pointer to the base of the new storage, error: -1 with errno set to indicate why the allocation failed.