Nivell d'usuari

Remo Suppi Boldrito

PID_00238575

Universitat Oberta de Catalunya

Índex

Introducció

Com ja s'ha vist en el mòdul anterior, **GNU/Linux** és un dels termes emprats per a referir-se a la combinació del nucli (*kernel*) –equivalent des del punt de vista de prestacions i funcionalitat (i en alguns casos superior) a Unix– denominat **Linux**, i d'eines de sistema GNU, tot sota llicència GPL (Llicència Pública de GNU) i una altra sèrie de llicències lliures.

Malgrat que Linux és, en sentit estricte, el sistema operatiu, part fonamental de la interacció entre el nucli i l'usuari (o els programes d'aplicació), es maneja usualment amb les eines GNU, com per exemple l'intèrpret d'ordres o instruccions Bash, que permet la comunicació amb el nucli mitjançant un conjunt complet d'ordres i instruccions. Hi ha altres nuclis disponibles per al projecte GNU, com el Hurd (http://es.wikipedia.org/wiki/Hurd), que molts desenvolupadors consideren que és l'autèntic nucli del projecte GNU. És interessant observar el mapa del nucli (http://commons.wikimedia.org/wiki/File:Linux_kernel_map.png) o la seva versió interactiva (http://www.makelinux.net/kernel_map/) en què es demostra la complexitat que té un sistema d'aquestes característiques.

En aquest mòdul veurem conceptes de nivell bàsic per a aprendre des de l'inici els diferents conceptes de GNU/Linux, i anirem avançant fins a veure aspectes d'inicialització i configuracions per a adequar el sistema operatiu a les nostres necessitats.

1. Introducció al sistema GNU/Linux

El sistema GNU/Linux és un sistema multitasca, és a dir, permet l'execució de centenars de tasques al mateix temps independentment de la quantitat de *cores* o CPU (processadors, de l'anglès *Central Process Unit*) de què disposi per a executar-se, i utilitza una característica comuna de tots els sistemes operatius moderns anomenada *multiprogramació*. Aquesta característica permet executar una tasca durant un determinat temps, suspendre-la, passar a la següent i així successivament, i quan s'arriba al final, tornar a començar per la primera sense afectar el seu comportament o execució.

Normalment aquesta execució es denomina *round robin*, ja que distribueix un quàntum de temps (que oscil·la entre 15 mil·lisegons i 150 mil·lisegons, depenent de l'operatiu) per a cada tasca en espera, i torna a començar per la primera quan s'arriba a l'última de la cua. Si el sistema té més d'un *core* o processador, GNU/Linux té capacitat per a distribuir aquestes tasques en els diferents elements de còmput, i obtenir les conseqüents millores en les prestacions. Pot semblar que 15 mil·lisegons (= 0,015 segons) és un temps petit, però per exemple processadors actuals com l'i3-370M (2 *cores*, 4 *threads*, de l'any 2010) s'obtenen en executar el *benchmark* Dhrystone 30.380 MIPS (milions d'instruccions per segon), la qual cosa amb un simple compte ens indica que en 15 mil·lisegons podria executar un mínim de 455 MIPS! S'ha de tenir en compte que el càlcul de MIPS depèn de molts factors i en aquest exemple es consideren els valors obtinguts de l'execució del programa Dhrystone utilitzat freqüentment per a comparar la potència de còmput dels processadors. A més, GNU/Linux és un sistema operatiu multiusuari que permet que més d'un usuari alhora pugui estar treballant amb el sistema, i que aquest amb el seu treball no pugui afectar cap de les tasques dels altres usuaris en el sistema.

> GNU/LINUX és un sistema multitasca i multiusuari que permet (fins i tot amb un sol processador) atendre les necessitats simultànies de múltiples usuaris, i utilitza una tècnica habitual dels sistemes operatius moderns denominada *multiprogramació*.

Tots els sistemes Unix, i GNU/Linux no és una excepció, consideren dos tipus d'usuaris diferenciats: el **superusuari** (anomenat **root**), que té tots els permisos sobre el sistema, i la resta dels usuaris, que disposen d'un directori de treball (*home*), del qual tenen tots els permisos, però en la resta del sistema el que poden fer està en funció de la seva importància i nivell de seguretat per al sistema mateix. Generalment, en qualsevol d'aquests sistemes ***nix** (Unix, Linux...) un usuari pot "mirar" (i en alguns casos executar) tot allò que no impliqui infor-

mació confidencial, però té generalment restringides la resta d'accions. Generalment, tots els usuaris tenen el seu directori de treball (*home*) dins del directori */home*; el directori de l'usuari *root* el trobem dins del directori */root*.

El segon concepte interessant en els sistemes *nix és que es pot treballar interactivament amb dos modes diferenciats: en mode text i en mode gràfic (i en aquest últim es pot obrir una aplicació en una finestra especial anomenada *terminal*, que permet treballar en mode text dins del mode gràfic). Normalment, els modes gràfics són els més utilitzats en sistemes d'escriptori o d'usuari domèstic, mentre que els de text són adequats per a servidors. No obstant això, com que no s'imposa cap restricció, es pot canviar fàcilment d'un a l'altre amb una seqüència de tecles (generalment Crtl+Alt+F1 a F6 per a passar a mode text –sis possibles terminals de text disponibles concurrents– i Crtl+Alt+F7 per a retornar a mode gràfic; considerar que totes les sessions són simultànies per la qual cosa el que estigui fent en el terminal 1 s'executa simultàniament amb el que faci en el terminal 2 o en mode gràfic) o fins i tot estar en mode gràfic i desenvolupar codi en mode text sobre un terminal, o connectat amb un terminal a una altra màquina o al disc d'una altra màquina.

El tercer concepte interessant és que la interacció entre l'usuari i el nucli es fa per mitjà d'un intèrpret d'ordres anomenat *shell*, que pot ser escollit per l'usuari entre uns quants. Aquests *shells* permeten l'execució d'ordres de manera interactiva (una cada vegada) o seqüencial per mitjà de petits programes anomenats *shell scripts* (seqüència d'ordres, que inclouen sentències de control i variables en un arxiu de text ASCII, que el *shell* interpretarà seqüencialment), que són molt potents i poden arribar a ser molt complexes. Òbviament, igual com altres sistemes operatius, GNU/Linux en la seva interfície gràfica suporta la interacció gràfica sobre les diferents accions del sistema, i permet executar *shell scripts* com si es tractés d'altres programes.

Dos conceptes interessants més en els sistemes *nix són la idea de tasca d'usuari o del sistema operatiu i l'estructura del sistema d'arxius. Quant al primer concepte, una tasca és una activitat que ha de fer el sistema operatiu, que pot ser l'execució d'una instrucció, una ordre, editar un arxiu, etc. Per a això, el sistema operatiu ha d'executar un programa adequat per a fer aquesta tasca, que normalment es denomina *programa executable*, ja que conté les instruccions màquina per a fer-la. Quan el programa executable es carrega en memòria s'anomena *procés* o programa en execució, ja que conté, a més del programa executable, totes les estructures de dades perquè es pugui fer aquesta tasca, i s'allibera tota la memòria quan finalitza l'execució.

Aquest procés es pot suspendre, bloquejar, o se'n pot continuar l'execució d'acord amb les necessitats de l'usuari i amb el que ordeni el sistema operatiu. Una derivació d'aquest concepte de procés és que en els processadors moderns un procés pot ser dividit entre diverses subtasques (anomenats fils d'execució o *threads*).

Quins avantatges té un programa *multithread* (multifil)? Que el codi que executa cada fil el defineix el programador, i per això es pot tenir un fil atenent una lectura de disc i un altre fent un refresc d'una imatge en pantalla simultàniament dins del mateix procés. Fins i tot tenint un sol processador, aquest tipus de programació és més eficient que no si fa una subtasca primer i una altra després.

Nota

Els fils d'execució o *threads* poden ser executats de manera independent per diferents processadors.

Finalment, els sistemes *nix disposen d'una estructura d'arxius estàndard en què s'ubiquen els arxius del sistema amb independència total dels dispositius físics. És a dir, a partir d'una arrel (anomenada *root* i definida per la barra /) s'ubiquen els diferents directoris en funció dels seus objectius, i aquí (com ja hem comentat) cada usuari disposa d'un directori de treball propi, generalment en el directori */home/nom-usuari*, en què el propietari té capacitat de decisió total, mentre que no és així en la resta de l'arbre (el superusuari té el seu directori en */root*).

2. Conceptes i ordres bàsiques

Entrant en una mica més de detall, en aquest apartat discutirem les idees bàsiques i les instruccions necessàries per a "moure'ns" en el sistema. La manera més simple és per mitjà d'ordres en l'intèrpret d'ordres (i ja veureu com és el més eficient quan s'adquireix una mica de pràctica, encara que al principi pugui semblar antiquat o complicat, o totes dues coses). Aquest mètode de treball ens permetrà treballar ràpidament amb qualsevol màquina de manera local o remota, ja que hi ha repetició i text predictiu de les instruccions, i es poden executar ordres molt complexes o programar *shell scripts* simplement amb un terminal de text. S'ha de tenir en compte que una interfície gràfica és summament útil per a un usuari novell, però extremadament ineficient per a un usuari avançat.

La majoria de les ordres, o instruccions, que veurem en aquest apartat, formen part de l'estàndard (normes IEEE POSIX) i són comunes a tots els sistemes GNU/Linux i Unix. Encara que cada distribució té les seves aplicacions d'administració i gestió pròpies, generalment totes les accions que s'hi fan també es poden fer amb les ordres que veurem. A partir d'aquestes ordres, podrem manipular gairebé tots els aspectes del sistema i moure'ns eficientment. En aquest apartat tenim per objectiu aprendre a utilitzar correctament aquestes ordres i a navegar per qualsevol sistema basat en GNU/Linux independentment de quina distribució usem. Cada una de les ordres del sistema sol tenir multitud de paràmetres diferents. Amb la utilització dels paràmetres podem, amb una mateixa ordre, executar accions diferents, encara que totes siguin d'un mateix estil. En aquest document no especificarem els diferents paràmetres de cada una de les ordres que veurem, ja que es pot consultar el manual inclòs en tot sistema *nix amb l'ordre `man <nom_ordre>`. S'ha de comentar que si no se sap el nom de l'ordre es pot utilitzar la instrucció `apropos acció`, que ens farà una llista de totes les ordres, i en la qual la paraula passada com a acció surt en l'especificació de l'ordre. Per exemple, si posem `apropos copy` ens donarà:

```
cp (1) - copy files and directories
cpgr (8) - copy with locking the given file to the password...
cpio (1) - copy files to and from archives
cppw (8) - copy with locking the given file to the password...
dd (1) - convert and copy a file
...
```

Això indica les instruccions que permeten copiar algun element. El paràmetre d'una ordre està precedit per un espai o moltes vegades per un guionet, com per exemple:

```
cp -dpR /home/juan /usr/local/backup
```

Això permet fer una còpia de suport dels arxius de */home/juan* en el directori */usr/local/backup*, i amb *-d* indiquem que copiï els enllaços simbòlics tal com són, en lloc de copiar els arxius als quals apunten; amb *-p* preserva els permisos, l'usuari i el grup de l'arxiu per copiar, i amb *-R* copia els directoris recursivament.

2.1. Usuari i grups

Nota

Hi ha diferents tipus d'instal·lacions d'un sistema operatiu en funció de l'objectiu o paper que complirà i les aplicacions que tindrà instal·lades: servidor, escriptori, ofimàtica, lleure, passarel·la, tallafocs, etc. Un **servidor** és aquella màquina que conté programes que s'encarreguen de proporcionar algun tipus de servei (com servir pàgines web, deixar que els usuaris es connectin remotament, etc.), que en la majoria dels casos estan vinculats a una xarxa de comunicacions, però no necessàriament.

Com hem comentat en la introducció, tots els sistemes operatius *nix són multiusuari i multitasca. Per aquest motiu és molt important que el sistema operatiu mateix incorpori mecanismes per a manipular i controlar correctament els usuaris: el sistema d'accés al sistema i identificació (conegut com a *login*), els programes que pot executar, els mecanismes de seguretat per a protegir el maquinari de l'ordinador, protecció per als fitxers dels usuaris, etc.

Per a identificar els usuaris davant del sistema operatiu, generalment s'utilitza una política de noms estàndard, que sol ser posar com a nom d'entrada la primera inicial del nom de l'usuari seguit del seu cognom. Els sistemes *nix organitzen tota aquesta informació per usuaris i grups i cal recordar que es diferencien majúscules i minúscules i, per tant, *abc* és diferent de *ABC*. Per a entrar a treballar interactivament amb el sistema, ens demanarà un *login* i una clau d'accés, la contrasenya.

L'inici de sessió (*login*) sol ser un nom que identifica de manera inequívoca l'usuari, i si bé hi ha altres mètodes d'identificació, per exemple, mitjançant certificats digitals, el primer és el mètode més habitual. Per a validar la connexió se sol·licita una paraula que només coneix l'usuari, que s'anomena *contrasenya* (*password*, en anglès). La contrasenya ha de ser una combinació de lletres, nombres i caràcters especials, i no ha de ser cap paraula de diccionari o similars, perquè pot representar un problema de seguretat important.

Exemple

Per exemple, si posem una contrasenya com .MBAqcytvav34 podria semblar impossible de recordar fàcilment, però és un punt i les primeres lletres del tango *Mi Buenos Aires querido cuando yo te vuelva a ver* i l'any que es va escriure, i forma una paraula clau que serà molt més complexa d'esbrinar que una paraula que sigui al diccionari, ja sigui escrita en l'ordre normal o al revés.

Actualment, en els sistemes GNU/Linux es pot seleccionar dos tipus de xifratge possibles per a les contrasenyes d'usuari. El primer és el que s'utilitza des dels inicis de Unix és el 3DES (que no és recomanable), en què la contrasenya es guarda en un arxiu (*etc/shadow*). Un dels problemes principals d'aquest mètode és que utilitza la funció *crypt*, la qual per mitjà de diccionaris o llistes de paraules pot fer el procés invers i obtenir el valor original. Si bé aquest mètode

utilitza un valor aleatori anomenat *salt* que permet codificar una mateixa clau de 4.096 maneres diferents, hi ha mètodes basats en taules que permeten tenir totes les combinacions.

Els sistemes Linux, gairebé en la totalitat de les versions actuals i que utilitzen *glibc2*, utilitzen un mecanisme diferent basat en una funció resum (*hash*). La funció *hash* criptogràfica és una funció resum d'un bloc arbitrari de dades i retorna una cadena de bits de grandària fixa (anomenada *digest*) que serà el valor xifrat, de manera que qualsevol canvi en les dades canviarà el valor del *digest*. La funció *hash* criptogràfica té quatre propietats principals: a) és fàcil de calcular per a qualsevol tipus de dades d'entrada, b) és impossible o extremadament complex/inviable generar el missatge original a partir d'un valor obtingut per la funció *hash*, c) no és factible modificar un missatge sense canviar el valor *hash* o *digest*, d) no és factible trobar dos missatges diferents amb el mateix valor *hash*.

A més s'utilitza el *salt* per a evitar per comparació i, sabent l'algorisme de *hash*, poder obtenir la paraula que va generar el resum o *digest*.

Les definicions de l'algorisme i com es generen els *passwd* estan definits en el sistema PAM (*pluggable authentication modules*), que és un mecanisme que inclouen tots els Linux per a gestionar l'autenticació de tasques i serveis, la qual cosa permet modelar polítiques de seguretat personalitzades per als diferents serveis. La configuració de *passwd* en la distribució Debian, per exemple, serà en l'arxiu */etc/pam.d/common-password*, en què tindrem una línia que indica:

```
password [success=1 default=ignore] pam_unix.so obscure sha512
```

On *sha512* indica que l'algorisme de *hash* serà *secure hash algorithm* de 512 bits, i *obscure* indica comprovacions extres sobre la paraula clau. El valor del *hash* (*digest*) es desarà en l'arxiu */etc/shadow* (en el segon camp de la línia que pertany a l'usuari corresponent) amb el format següent: idsalt$valor_hash. En què *id* identifica l'algorisme utilitzat i en què poden prendre els valors següents: 1(MD5), 2a (Blowfish, solament en algunes distribucions), 5 (SHA256), 6 (SHA-512). Per la qual cosa 5salt$digest és una contrasenya codificada SHA-256 i 6salt$digest n'és una en SHA-512. El *salt* representa un màxim de setze caràcters següents entre els símbols "$" a continuació de l'id i la grandària de la cadena xifrada. La grandària d'aquesta cadena es fixa en MD5=22 bytes, SHA-256=43 bytes i SHA-512=86 bytes, en què tots els caràcters de la contrasenya són representatius i no solament els vuit primers com en DES. Per a generar un *hash* d'una paraula de contrasenya hem d'utilitzar directament la tecla d'ordre `passwd`. Si es vol generar un valor *hash* però canviar l'algoritme, es pot utilitzar la tecla d'ordre:

```
mkpasswd -m sha-512 nteumyvist -S 666999666
```

en la qual estic posant la clau nteumyvist amb un *salt* 666999666 i generarà la cadena següent:

```
$6$666999666$O0KEfCSEe/H6DrFPLSfsM4d7phA5ySumsgnl5pI/YaOR80NpnCP3LjKneBBEUeVGUBouG6jdPwihi4a5UNLPn/
```

`mkpasswd -m help` donarà tots els algorismes disponibles.

Els grups d'usuaris són un conjunt d'usuaris amb accés al sistema que comparteixen unes mateixes característiques, de manera que ens és útil agrupar-los per a poder donar-los una sèrie de permisos especials en el sistema. Un usuari ha de pertànyer, almenys, a un grup, encara que pot ser de més d'un. El sistema també utilitza tot aquest mecanisme d'usuaris i grups per a gestionar els servidors d'aplicacions instal·lats i altres mecanismes. Per aquesta raó, a més dels usuaris reals, en un sistema hi haurà usuaris i grups que no tenen accés interactiu però estan vinculats a altres tasques que s'han de fer en l'operatiu.

Exemple

Per exemple, l'usuari Debian-Exim és un usuari definit per a les funcionalitats del gestor de correu Exim (quan es trobi instal·lat aquest paquet) que té ID d'usuari (101) i de grup (105), però que no es pot connectar interactivament (tal com ho indica el /bin/false de la definició de l'usuari en l'arxiu /etc/passwd).

```
Debian-exim:x:101:105::/var/spool/exim4:/bin/false.
```

Com ja hem explicat, en tot sistema operatiu hi ha un superusuari (*root*) que té privilegis màxims per a efectuar qualsevol operació sobre el sistema. És necessari que existeixi, ja que serà qui s'encarregarà de tota l'administració i gestió de servidors, grups, etc. Aquest usuari no s'ha d'utilitzar per a treballar normalment en el sistema. Només haurem d'entrar com a *root* quan sigui realment necessari (actualment, la majoria de les distribucions no permeten entrar com a *root*, sinó entrar com a usuari normal i després "elevar-se" com a *root* per mitjà de la instrucció `su` o executar instruccions amb l'ordre `sudo`) i utilitzarem altres usuaris per al treball normal. D'aquesta manera, mai no podrem danyar el sistema amb operacions errònies o provant programes que poden ser maliciosos, etc.

Tota la informació d'usuaris i grups es troba als arxius següents:

- /etc/passwd: informació (nom, directori *home*, etc.) de l'usuari.
- /etc/group: informació sobre els grups d'usuaris.
- /etc/shadow: contrasenyes xifrades (valor *hash*, generalment) dels usuaris i configuració per a la validació, canvi, etc.

Tot i que realment sigui possible deshabilitar el mecanisme d'autentificació basat en */etc/shadow* (bàsicament perquè encara hi ha un petit nombre d'aplicacions/serveis que no el suporten), **no** és recomenable fer-ho ja que és un arxiu que només pot llegir *root;* per tant, la resta d'usuaris no tindrà accés

als valors *hash* i es redueix així la probabilitat que un usuari normal pugui utilitzar eines de força bruta (taules, diccionaris, etc.) per a esbrinar les contrasenyes d'altres usuaris o de *root*.

Tots aquests fitxers estan organitzats per línies, cada una de les quals identifica un usuari o grup (depenent del fitxer). En cada línia hi ha diversos camps separats pel caràcter ":", i és important saber què són aquests camps, per la qual cosa els explorarem amb una mica més de detall:

Nota

Informació d'usuaris i grups:
- /etc/passwd: informació dels usuaris.
- /etc/group: informació sobre els grups d'usuaris.
- /etc/shadow: valor *hash* de les contrasenyes dels usuaris i configuració per a la validació, canvi, etc.

1) *<u>/etc/passwd</u>*. Per exemple, *games:x:5:60:games:/usr/games:/bin/sh*.

- Camp 1. Login: el nom de l'usuari. No hi pot haver dos noms iguals, encara que algun pot coincidir amb un grup del sistema.

- Camp 2. Contrasenya xifrada: si no s'utilitza el fitxer *shadow*, les contrasenyes xifrades s'emmagatzemen en aquest camp. Si utilitzem el fitxer *shadow*, tots els usuaris existents han d'estar també en el *shadow* i en aquest camp s'identifica amb el caràcter *x*.

- Camp 3. ID d'usuari: número d'identificació de l'usuari. És el número amb el qual el sistema identifica l'usuari. El 0 és el reservat per al *root*.

- Camp 4. ID del grup: el número de grup al qual pertany l'usuari. Com que un usuari pot pertànyer a més d'un grup, aquest grup es denomina *primari*.

- Camp 5. Comentaris: camp reservat per a introduir els comentaris sobre l'usuari. Se sol utilitzar per a posar el nom complet o algun tipus d'identificació personal.

- Camp 6. Directori d'usuari: el directori de l'usuari, que és on pot deixar tots els seus fitxers. Se solen posar en una carpeta del sistema (generalment */home/login-usuari*).

- Camp 7. Intèrpret d'ordres: un intèrpret d'ordres (*shell*) és un programa que s'encarrega de llegir tot el que escrivim amb el teclat i executar els programes o ordres que indiquem. En el món GNU/linux el més utilitzat és el *Bash* (GNU Bourne again shell), si bé n'hi ha altres (*csh*, *ksh*, etc.). Si en aquest camp escrivim */bin/false*, no permetrem que l'usuari executi cap ordre en el sistema, encara que estigui donat d'alta.

2) <u>/etc/group</u>. Per exemple, *netdev:x:110:Debian*.

- Camp 1. Nom del grup.

- Camp 2. Contrasenya xifrada: la contrasenya d'un grup s'utilitza per a permetre que els usuaris d'un determinat grup es puguin canviar a un altre

o per a executar alguns programes amb permisos d'un altre grup (sempre que es disposi de la contrasenya).

- Camp 3. ID de grup: número d'identificació del grup. És el número amb el qual el sistema identifica internament els grups. El 0 està reservat per al grup de *root* (els administradors).

- Camp 4. Llista d'usuaris: els noms dels usuaris que pertanyen al grup, separats per comes. Encara que tots els usuaris han de pertànyer a un grup determinat (especificat en el quart camp del fitxer *passwd*), aquest camp es pot utilitzar perquè usuaris d'altres grups també disposin dels mateixos permisos que té l'usuari a qui s'està fent referència.

3) /etc/shadow. Per exemple,
root:1oE9iKzko$n9imGERnPDayiyat0XQjm.:14409:0:99999:7:::

- Camp 1. Login: ha de ser el mateix nom que s'usa en el fitxer */etc/passwd*.

- Camp 2. Valor *hash* de la contrasenya.

- Camp 3. Dies que han passat, des de l'1/1/1970, fins que la contrasenya ha estat canviada per última vegada.

- Camp 4. Dies que han de passar fins que la contrasenya es pugui canviar.

- Camp 5. Dies que han de passar fins que la contrasenya s'hagi de canviar.

- Camp 6. Dies abans que caduqui la contrasenya s'avisarà l'usuari que l'ha de canviar.

- Camp 7. Dies que poden passar després que la contrasenya caduqui abans de deshabilitar el compte de l'usuari (si no es canvia la contrasenya).

- Camp 8. Dies, des de l'1/1/1970, que el compte és vàlid, i passats aquests dies serà deshabilitat.

- Camp 9. Reservat.

Quan un usuari entra en el sistema, se situa en el seu directori personal i s'executa l'intèrpret d'ordres (*shell*) configurat en */etc/passwd* en la línia corresponent a l'usuari. D'aquesta manera, ja pot començar a treballar. Només el *root* del sistema (o els usuaris del seu grup) tenen permís per a manipular la informació dels usuaris i grups, donar-los d'alta, de baixa, etc. Cada ordre per a manejar els usuaris té diversos paràmetres diferents per a gestionar tots els camps que hem vist anteriorment.

A tall d'exemple, podem esmentar:

- **adduser:** ens serveix bàsicament per a afegir un nou usuari al sistema. La manera com s'afegeixi (si no és que li especifiquem) es podrà configurar en el fitxer */etc/adduser.conf*. Admet un conjunt d'opcions diferents per a especificar el directori d'usuari, l'intèrpret que utilitzarà, etc.

- **useradd:** crea un nou usuari o canvia la configuració per defecte. Aquesta ordre i l'anterior ens poden servir per a fer les mateixes accions, encara que amb diferents paràmetres.

- **usermod:** amb aquesta ordre podem modificar la majoria dels camps que es troben en els fitxers *passwd* i *shadow*, com el directori personal, l'intèrpret, la caducitat de la contrasenya, etc.

- **chfn:** canvia la informació personal de l'usuari, continguda en el camp de comentaris del fitxer *passwd* (campos).

- **chsh:** canvia l'intèrpret d'ordres de l'usuari.

- **deluser:** elimina un usuari del sistema, i esborra tots els seus fitxers segons els paràmetres que li passem, fa còpia de seguretat o no, etc. La configuració que s'utilitzarà per defecte amb aquesta ordre s'especifica en el fitxer */etc/deluser.conf*.

- **userdel:** similar a l'ordre anterior i amb diferents paràmetres/condicions.

- **passwd:** serveix per a canviar la contrasenya d'un usuari, la informació de caducitat, o per a bloquejar o desbloquejar un determinat compte.

- **addgroup:** permet afegir un grup al sistema.

- **groupadd:** el mateix que l'ordre anterior, però amb diferents paràmetres.

- **groupmod:** permet modificar la informació (nom i GID) d'un grup determinat.

- **delgroup:** elimina un grup determinat. Si algun usuari el té com a primari, no es podrà eliminar.

- **groupdel:** similar a l'ordre anterior, però amb paràmetres/condicions diferents.

- **gpasswd:** serveix per a canviar la contrasenya del grup. Per a saber quin usuari som, podem utilitzar l'ordre `whoami`, que ens mostrarà el nostre nom d'entrada.

Nota

Ordres bàsiques de gestió d'usuaris i grups:

- useradd
- usermod
- chfn
- chsh
- deluser
- userdel
- passwd
- addgroup
- grouppadd
- groupmodd
- delgroup
- groupdel
- gpasswd
- groups

- **groups:** serveix per a saber a quins grups pertanyem, i `id` ens mostrarà l'usuari i els grups.

També és interessant poder convertir-nos en un altre usuari sense sortir de la sessió (ordre `login` o `su`) o canviar-nos de grup amb l'ordre `newgrp`. Aquesta última ordre s'ha d'utilitzar només quan no es pertany al grup en qüestió i se sap la seva contrasenya (que ha d'estar activada en el fitxer group). Si només necessitem els permisos del grup en qüestió per a executar una ordre determinada, també podem utilitzar `sg`.

Com veiem, en GNU/Linux tenim més d'una manera per a executar una acció determinada. Aquesta és la tònica general que se segueix en el sistema: podem editar directament els fitxers i modificar-los, utilitzar algunes de les ordres que hi ha, crear-les nosaltres mateixos, etc. En definitiva, tenim la possibilitat de seleccionar quina és l'opció que més ens satisfà.

D'altra banda, i com dèiem anteriorment, GNU/Linux és un sistema operatiu multiusuari, per la qual cosa en un mateix moment hi pot haver diversos usuaris connectats al sistema de manera simultània. Per a esbrinar qui són, es pot utilitzar l'ordre `who`, que ens mostra la llista d'usuaris dins del sistema; `w`, a més, ens mostra què és el que estan fent. Ens podem comunicar amb un altre usuari utilitzant l'ordre `write`, amb la qual apareix el missatge que hem escrit a la pantalla de l'usuari indicat o `wall`, que escriu el contingut del fitxer que hem especificat en tots els usuaris dins del sistema. Per a activar o desactivar l'opció de rebre missatges, tenim l'ordre `mesg`. També podem fer un xat personal amb algun usuari a partir de l'ordre `talk`.

2.2. El sistema de fitxers i la jerarquia

Tot sistema operatiu necessita desar multitud d'arxius: configuració del sistema, registre d'activitats, d'usuaris, etc. Hi ha diferents sistemes d'arxius caracteritzats per la seva estructura, fiabilitat, arquitectura, rendiment, etc., i GNU/Linux és capaç de llegir i escriure arxius en la gairebé totalitat dels sistemes d'arxius, encara que té els seus sistemes propis optimitzats per a les seves funcionalitats, com per exemple ext3 o ReiserFS.

L'*ext4* és una millora compatible amb *ext3* que, al seu torn, és una evolució de l'*ext2* que és el més típic i estès. El seu rendiment és molt bo, incorpora tot tipus de mecanismes de seguretat i adaptació, és molt fiable i incorpora una tecnologia denominada *journaling*, que permet recuperar fàcilment errors en el sistema quan, per exemple, hi ha un tall de llum o l'ordinador sofreix

una parada no prevista. *ext4* suporta volums de fins a 1024 PiB (PiB pebibyte = 2^{50} bytes ≈ 1.000.000.000.000.000 bytes), millora l'ús de la CPU i el temps de lectura i escriptura..

ReiserFS és un altre dels sistemes utilitzats en Linux que incorpora noves tecnologies de disseny, que li permeten obtenir prestacions i utilització de l'espai lliure més adequades. Qualsevol d'aquests tipus de sistemes d'arxius pot ser seleccionat en el procés d'instal·lació, i és recomanable *ext3* per a la majoria de les instal·lacions.

Una característica molt important de tots els sistemes *nix és que tots els dispositius del sistema es poden tractar com si fossin arxius (independència del dispositiu físic). És a dir, hi ha el procediment de "muntar el dispositiu" per mitjà de l'ordre `mount` en un directori del sistema i després, accedint a aquest directori, s'accedeix al dispositiu (fins i tot si el dispositiu és remot), per la qual cosa no hi ha una identificació del dispositiu físic per a treballar amb els fitxers com en altres sistemes operatius, com A:, C:, etc.).

El sistema de fitxers *ext2/3/4* ha estat dissenyat per a gestionar de manera òptima fitxers petits (els més comuns) i de manera acceptable els fitxers grans (per exemple, arxius multimèdia) si bé es poden configurar els paràmetres del sistema de fitxers per a optimitzar el treball amb aquest tipus d'arxius. Com hem esmentat anteriorment, el sistema d'arxius parteix d'una arrel indicada amb /, i les carpetes (directoris) i subcarpetes (subdirectoris) s'organitzen a partir d'aquesta, de manera que presenta una organització jeràrquica (visualitzada amb l'ordre `tree -L 1` a Debian) com:

```
/
├── bin
├── boot
├── dev
├── etc
├── home
├── initrd.img -> /boot/initrd.img-3.16.0-4-amd64
├── lib
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
```

```
├── usr
├── var
└── vmlinuz -> boot/vmlinuz-3.16.0-4-amd64
```

Aquí es mostra el primer nivell de directoris a partir del /. Tots són directoris, excepte els que figuren amb el caràcter "->", que són enllaços a arxius (l'arxiu original es troba a la dreta de la fletxa).

2.3. Directoris del sistema

En les distribucions GNU/Linux se segueix l'estàndard FHS [FHS], i tenim com a directoris en el directori arrel (els més importants):

- **/bin:** ordres bàsiques per a tots els usuaris del sistema.
- **/boot:** arxius necessaris per a l'arrencada del sistema.
- **/dev:** dispositius del sistema.
- **/etc:** arxius de configuració del sistema i de les aplicacions que hi ha instal·lades.
- **/home:** directoris personals dels usuaris.
- **/lib**, **lib64:** biblioteques essencials per al nucli del sistema i els seus mòduls.
- **/mnt:** punt de muntatge temporal per a dispositius.
- **/proc:** processos i variables del nucli del sistema.
- **/root:** directori personal per al *root* del sistema.
- **/run:** dades de variables durant l'execució (informació sobre el sistema, processos i usuaris que estan actius al sistema).
- **/sbin:** instruccions especials per al *root* del sistema.
- **/sys:** informació sobre els dispositius connectats al sistema.
- **/tmp:** arxius temporals.
- **/usr:** segona estructura jeràrquica, utilitzada per a emmagatzemar/configurar tot el programari instal·lat en el sistema.
- **/var:** directori per als gestors de cues o *spoolers* d'impressió, arxius de registre (*logs*), etc.

No s'han d'esborrar aquests directoris, malgrat que sembli que no s'utilitzen, per al bon funcionament del sistema (moltes aplicacions poden no instal·larse o donar errors si els directoris estàndard no estan definits).

Per a moure'ns per l'estructura de directoris hem d'utilitzar les ordres per a fer una llista dels continguts i canviar de carpeta. Quan entrem en el sistema, és usual que la connexió ens situï en el nostre directori personal, que generalment se sol indicar amb el caràcter "~". Si volem veure el que hi ha en el directori en el qual estem situats, podem fer una llista dels continguts utilitzant l'ordre `ls` o, en la seva versió més completa, `ls -la`, que implica tots els fitxers (incloent-hi els que comencen per un ".", que no es mostren nor-

Exemple

Per exemple, si fem **ls .** mostrarà la llista del directori actual i si fem **ls ..** mostrarà la llista del directori immediatament superior.

malment) amb *-a*, i en format llarg amb *-l*. En tots els directoris hi ha dues entrades indicades amb "." i ".."; la primera fa referència al directori actual i la segona al directori superior.

Per a canviar de directori podem utilitzar l'ordre `cd`, la qual, si no li passem cap paràmetre, ens situarà en el directori *home* de l'usuari que l'ha executada. Totes les ordres accepten adreces relatives; per exemple, `ls ./grub` si estem en el directori */boot* ens mostrarà el contingut del directori */boot/grub* (forma relativa) o l'ordre `ls /boot/grub` també ens mostrarà el contingut del directori */boot/grub*, però des de qualsevol lloc on estiguem (forma absoluta). Una altra ordre útil per a saber on estem aturats és `pwd`, que ens indicarà en quin directori som.

2.4. Enllaços

Un element molt utilitzat en els arxius són els enllaços o vincles. Un enllaç és un pont a un arxiu o directori i representa una referència que podem posar en qualsevol lloc que ens interessi, i actua com un accés directe a qualsevol altre.

Aquest mecanisme ens permet accedir a carpetes o fitxers de manera segura i còmoda, sense haver-nos de desplaçar per la jerarquia de directoris.

Exemple

Si necessitem accedir freqüentment a l'arxiu */etc/network/if-up/mountnfs*, per exemple, podem utilitzar un enllaç en el nostre directori amb l'ordre

`ln -s /etc/network/if-up/mountnfs nfs-conf`,

i fent un `cat nfs-conf` tindrem el mateix resultat que fent

`cat /etc/network/if-up/mountnfs`,

i així evitem haver d'introduir cada vegada tota la ruta completa.

En l'exemple anterior hem creat un enllaç simbòlic paràmetre (-s) és a dir, que si esborrem l'arxiu l'enllaç quedarà apuntant a res i donarà un error quan executem l'ordre `cat nfs-conf`, però aquest tipus d'enllaç es pot fer en qualsevol recurs i en qualsevol de les particions del disc. L'altra possibilitat és fer un enllaç fort (*hard link*), permès només per a recursos en la mateixa partició; en aquest cas, si esborrem l'arxiu, l'enllaç queda actiu fins que no hi hagi més enllaços apuntant a aquest arxiu (moment en el qual s'esborrarà l'arxiu de destinació). Aquest recurs s'ha d'utilitzar amb compte (només el *root* pot fer enllaços forts a directoris), ja que permet ocultar a quin arxiu està apuntant, mentre que amb un enllaç simbòlic es pot veure l'arxiu de destinació (per exemple, amb l'ordre `ls -al`).

2.5. Permisos

Ja que els sistemes *nix són multiusuari, necessitem que els arxius emmagatzemats tinguin una sèrie de propietats que permetin llegir, escriure i executar (paràmetres *r, w, x –read, write, execute*). Per a això GNU/Linux pot treballar amb un mètode simplificat (anomenat *access control list reduïdes*) en què per a cada element en el sistema d'arxius es consideren tres bits (que representen els atributs per a *rwx*) per al propietari de l'element (*owner*), tres per al grup i tres per a la resta d'usuaris. Llavors, per a cada element (arxiu, directori, dispositiu, enllaç, etc.) hi ha 9 bits a més de la identificació de qui és el propietari de l'element (uid) i a quin grup pertany (gid). Quan fem `ls -l` tindrem per a cada element una sortida com la següent:

```
-rw-r--r-- 1 root root 2470 May 26 17:10 /etc/passwd
```

Els primers deu caràcters (començant per l'esquerra) ens indiquen els permisos del fitxer de la manera següent:

- Caràcter 1: indica el tipus d'arxiu; els més comuns són "-" per a un arxiu, *d* per a un directori, i *l* per a un enllaç simbòlic.

- Caràcters 2, 3, 4: ens indiquen, respectivament, els permisos de lectura, escriptura i execució per al propietari del fitxer. En cas de no tenir el permís corresponent activat, hi ha el caràcter "-" i si no, *r, w* o *x*. En el tercer caràcter, a més, ens podem trobar una *s*, que ens indica si l'arxiu és de tipus *SetUserId*, que significa que en executar-lo obtindrà els permisos del propietari del fitxer. Si només té el permís *x*, quan el programa s'executa ho fa amb els permisos de qui l'hagi llançat.

- Caràcters 5, 6, 7: aquests caràcters tenen exactament el mateix significat que els anteriors, però fan referència als permisos concedits als usuaris del grup a què pertany l'arxiu. Aquí també podem trobar una *s* en el tercer caràcter que indica el bit de *SetGid* activat, la qual cosa significa que qualsevol que executi aquest arxiu obtindrà els permisos del grup del propietari del fitxer.

- Caràcters 8, 9, 10: igual que en el cas anterior, però per als altres usuaris del sistema.

La xifra següent (1, en aquest cas) ens indica el nombre d'enllaços forts que té l'arxiu. Per als directoris, aquest nombre indica quantes carpetes hi ha en l'interior, a més dels enllaços forts que té. A continuació es troba el propietari i el grup de l'arxiu, seguit de la mida (en bytes) que ocupa i la data de l'última modificació. En tots els arxius es desa la data de creació, de l'últim accés i de l'última modificació, que podem manipular amb l'ordre `touch`. Al final es troba el nom del fitxer, en el qual es diferencien minúscules de majúscules, i aquí podem tenir tot tipus de caràcters sense cap problema (encara que després

serà més complicat executar ordres amb aquests caràcters, ja que s'haurà de posar entre cometes ("") perquè no s'interpretin). Es recomana utilitzar: a-z A-Z . - _ 0-9.

El mecanisme de *SetUserId* és molt útil quan un programa necessita tenir els permisos del seu propietari per a accedir a certs arxius o fer algun tipus d'operació en el sistema. S'ha de vigilar aquest tipus d'arxius perquè poden generar problemes de seguretat en el sistema si són mal utilitzats. Per a canviar els permisos d'un arxiu determinat podem utilitzar l'ordre `chmod`, i aquesta acció només la pot fer el propietari de l'arxiu. Hi ha dues maneres comunes d'utilitzar `chmod`. Una és: `chmod XXX nom_arxiu`, en què cada X està compresa entre 0 i 7, que corresponen al valor en octal dels permisos *rwx* per al propietari (primer número), el grup (segon) i públic (tercer). Per a treure el número hem de considerar que 1 vol dir amb permís concedit, i 0, sense; per exemple, *r-x* es tradueix com 101, que en octal (o binari) és 5. Per això, r-x--xr-- es tradueix com 101001100, que queda com 514.

Proteccions

Per a canviar les proteccions de file a *rwxr--r-*:

```
chmod 744 file
```

Per a canviar de propietari i grup:

```
chown user file
chgrp group file
```

L'altra manera d'utilitzar l'ordre és indicar de manera explícita quin permís volem donar o eliminar en l'arxiu, indicant amb les lletres *u, g, o* l'usuari, el grup o la resta, respectivament, un + per a agregar el permís i un – per a treure'l, i *r, w, x* o *s* (aquest últim per al *SetUserId*) per al permís en concret. Per exemple, `chmod go +r mountfs` concediria el permís de lectura al grup i als altres usuaris per a l'arxiu mountnfs. Perquè canviï el propietari d'un fitxer, s'utilitza l'ordre `chown`[1], i per a canviar el grup d'un arxiu es pot utilitzar l'ordre `chgrp`. Els permisos per defecte per als arxius es poden definir amb l'ordre *umask*, amb la mateixa notació que per a la primera forma del `chmod` però complementat (és a dir, si volem *rw-r- -r--* el valor hauria de ser 133).

(1)Aquesta ordre només la pot utilitzar el *root*, ja que un usuari podria fer una acció maliciosa i després canviar el propietari de l'arxiu i responsabilitzar un altre usuari de l'acció feta.

Hi ha un modificador anomenat *sticky bit* (representat per la lletra *t*) que és un permís que només és útil en directoris. És utilitzat especialment en directoris temporals per als quals tots els usuaris tenen permisos d'escriptura (com */tmp/*) i que permet que solament el propietari o el propietari del directori pare puguin eliminar d'arxius per a evitar que qualsevol un altre usuari pugui eliminar arxius que no li pertanyen del directori compartit.

2.6. Manipulació, patrons, cerques i continguts

L'ordre `rm` permet eliminar els arxius, i per a eliminar un directori podem utilitzar l'ordre `rmdir`, encara que només l'esborrarà quan sigui buit (si volguéssim esborrar completament un directori i tot el seu contingut, podem utilitzar `rm -r`, que funciona de manera recursiva). Per a copiar arxius d'un lloc a un altre tenim l'ordre `cp`, indicant el fitxer o directori origen i el lloc o nom de destinació, encara que sigui en el directori actual (si volem moure un arxiu o directori, es pot utilitzar l'ordre `mv`).

Caràcters modificadors

*: qualsevol cosa,
?: un caràcter
[Yy]: un o l'altre
[A-Z]: un rang
[:classe:]: una classe

Podem utilitzar modificadors en els noms dels arxius o directoris, com per exemple "*", per a referir-nos a qualsevol cadena, i "?" per a indicar un caràcter qualsevol. Per exemple, si volem mostrar tots els arxius que comencin per *a* i acabin per *x* caldrà executar `ls a*x`, però si volem mostrar tots els arxius de tres lletres que comencin per *a* i acabin per *x*, serà `ls a?x`. Es pot utilitzar "[]" per a una selecció de caràcters; per exemple `ls [Yy]*` indicaria tots els arxius que comencin per *Y* o per *y*, i després qualsevol cosa. Podem agregar també "!", que significa la negació del que indiquem, com en *[!Yy]*, és a dir, que no comencin per *Y* o *y*. Finalment, per a facilitar certes cerques, dins de "[]" podem especificar classes de caràcters com [:classe:], que pot ser una de les següents:

- *alnum:* [A-Za-z0-9]
- *alpha:* [A-Za-z]
- *blank:* [\]
- *cntrl:* caràcters de control
- *digit:* [0-9A-Fae-f]
- *graph:* caràcters imprimibles (sense espais)
- *lower:* [a-z]
- *print:* caràcters imprimibles (amb espais)
- *punct:* [.,¡!¿?:;]
- *space:* []
- *upper:* [A-Z]
- *xdigit:* [0-9A-Fa-f]

Un altre tipus d'operació molt útil és la cerca d'arxius. Hi ha diverses ordres per a fer cerques de diferents tipus: `find` és l'ordre més versàtil per a buscar informació sobre els arxius o directoris (per nom, mida, data, proteccions, etc.), `locate` permet utilitzar una base de dades del sistema per a fer cerques més ràpides que el `find`, però s'ha de tenir en compte que pot donar informació no actualitzada i que es pot actualitzar amb `updatedb`, i `whereis` (també `wich`) indica on es troba l'arxiu especificat.

Com els arxius poden ser de molts tipus (executables, text, dades, música, etc.) en els sistemes *nix no s'utilitza l'extensió per a identificar el tipus d'arxiu sinó un nombre intern a la capçalera de l'arxiu anomenat *magic number*, que determina el tipus d'arxiu segons les seves dades (es pot utilitzar l'ordre `file` per a llegir i determinar el tipus d'arxiu). Si necessitem veure el contingut d'un arxiu, una de les ordres bàsiques és `cat`, o `more` si l'arxiu és ASCII; llavors el mostrarà paginat. Si fem un `cat` d'un arxiu executable alguns dels caràcters poden desconfigurar el terminal i es pot reinicialitzar amb l'ordre `reset`, i `clear` per a esborrar la pantalla. L'ordre `less` ens permet moure'ns de manera més eficient (endavant i endarrere per l'arxiu). Si l'arxiu és binari i volem veure què conté, podem utilitzar les ordres `hexdump` per a veure el contingut de forma hexadecimal o `strings` per a buscar les cadenes de caràcters. A l'ordre `grep` li podem passar com a segon paràmetre el nom de l'arxiu, i com a primer, el patró que vulguem buscar (segons la sintaxi que hem vist anteriorment, estesa a altres opcions). A més, l'ordre ens permet múltiples accions més, com comptar el

nombre de línies en les quals apareix el patró (paràmetre *-c*), etc. Amb `cut` podem separar en camps el contingut d'una línia del fitxer especificant quin caràcter és el separador, molt útil en tasques d'administració. També podem visualitzar un determinat nombre de línies del començament o del final d'un arxiu amb les ordres `head` i `tail`, respectivament, i amb `wc` podem comptar el nombre de línies o paraules, la màxima longitud de línia d'un fitxer, etc. Finalment, per a comparar diferents arxius hi ha diverses ordres per a fer-ho: `diff`, `cmp` i `comm` fan comparacions de diferents maneres i mètodes en els fitxers que indiquem; `sdiff` a més permet barrejar les dades d'acord amb els paràmetres indicats.

2.7. Processos

Com hem dit en la introducció, els *nix són sistemes operatius multitasca que executen processos i fils (*threads*) mitjançant una tècnica anomenada *multiprogramació*, que permet executar més d'un procés o fil alhora de manera concurrent i més eficient que si l'executéssim seqüencialment, ja que es pot encavalcar l'execució d'entrada o sortida d'un procés amb l'execució en la CPU d'un altre procés. Per a identificar de manera inequívoca cada procés, el nucli del sistema els assigna un número denominat *PID* (*process identification*), necessari per a administrar i referenciar el procés.

Per a saber quins processos s'estan executant, podem utilitzar l'ordre `ps`. Una altra ordre interessant per a mirar l'execució interactiva dels processos és `top`, que mostra la càrrega i l'estat de sistema de manera dinàmica (cal prémer "q" –*quit*– per a sortir de la instrucció).

Ordre *ps*

Per exemple, *ps -edaf* mostra un conjunt d'informació sobre tots els processos en execució i en diferents estats.

A més d'això, podem enviar senyals als processos a fi d'informar-los d'algun esdeveniment, els podem treure de la cua d'execució, eliminar-los, donar-los més prioritat, etc. Saber manipular correctament tots aquests aspectes també és molt important, ja que ens permetrà utilitzar el nostre ordinador de manera més eficient. L'ordre `kill` ens permet enviar senyals als processos que ens interessin.

En general, tots els programes es dissenyen perquè puguin rebre aquest tipus de senyals (fins i tot els scripts poden capturar els senyals). D'aquesta manera, segons el tipus de senyal rebut saben que han de fer unes operacions o d'altres (per exemple, suspendre l'execució quan un usuari fa un `Ctrl-d`). Per a veure els tipus de senyals, es pot consultar *man* `kill`. `killall` és una ordre per a fer el mateix, però utilitza el nom del procés en lloc del PID, i `skill` és similar, però amb una sintaxi diferent: per exemple, per a detenir totes les execucions d'un usuari determinat, podríem utilitzar l'ordre `skill -STOP -u login`, amb la qual cosa s'acabarà l'execució dels processos d'aquest usuari. A més de `Ctrl-d` o `Ctrl-c` per a finalitzar un procés (la primera és amb espera fins que el procés acabi el seu E/S i la segona és en el moment que la rep), amb la combinació `Ctrl-z` podem interrompre un programa i reviure'l amb `fg`.

Nota

Ordres per a processos:

- *kill -9 <pid>*
- *skill -STOP -u <login>*
- *ps -edaf*
- *top*
- *pstree*
- *nice [-n increment] <ordre>*

Kill

Per exemple, amb el senyal TERM –que és 15– si fem `kill -15 PID`, que és equivalent a fer **Ctrl-c** en un procés interactiu sobre un terminal, indiquem al procés que volem que acabi, de manera que en rebre el senyal haurà de desar tot el necessari i acabar l'execució; si el programa no està preparat per a rebre aquest tipus de senyal, podem utilitzar el *-9* (que obeeixen tots els processos) i forçar que acabi independentment del que estiguin fent amb `kill -9 PID`.

L'ordre `pstree` permet veure aquesta jerarquia de manera gràfica, i veurem que el pare de tots els processos és un anomenat *init* (PID=1; en algunes distribucions es visualitzarà `systemd` ja que aquest procés és el que substitueix al tradicional *init* del sistema *nix), ja que és el procés inicial per a posar en marxa els processos restants del sistema, i a partir d'aquest neixen tots els altres, que al seu torn poden tenir més fills. Aquesta estructura és molt útil per a identificar d'on vénen els processos (qui els ha posat en marxa) i per a eliminar-ne un conjunt, ja que, en eliminar un procés pare, també s'eliminen tots els seus fills.

Un paràmetre important dels processos és un valor anomenat *prioritat*, que està relacionat amb la CPU que rebrà aquest procés (com més prioritat més temps de CPU). El rang de prioritats va des del -20 fins al 19 (les negatives només les pot utilitzar el *root*), de major a menor. Per a llançar un procés amb una prioritat determinada, podem utilitzar l'ordre `nice` i `renice`, si volem donar una prioritat diferent d'un procés que ja estigui en execució. Per defecte, la prioritat amb què s'executen els programes és la 0. L'ordre `time` permet calcular el temps que utilitza un procés per a executar-se (generalment amb finalitats comptables, per exemple si hem de facturar pel temps de CPU que gasta un procés).

2.8. Ordres complementàries

Totes les ordres disposen en GNU/Linux (i en tots els *nix) d'un manual complet que indica tots els paràmetres i opcions (`man ordre`), i s'utilitza l'ordre `less` per a visualitzar-les, i per això podem anar endavant i enrere amb les tecles d'avançar i retrocedir pàgina, buscar una paraula amb el caràcter "/" seguit de la paraula (*n* ens serveix per a buscar les aparicions següents i *N*, per a les anteriors), *q* per a sortir, etc. Els manuals del sistema es divideixen en diferents seccions segons la seva naturalesa:

1) Programes executables (aplicacions, ordres, etc.).
2) Crides al sistema proporcionades per l'intèrpret d'ordres.
3) Crides a biblioteques del sistema.
4) Arxius especials (generalment els de dispositiu).
5) Format dels arxius de configuració.
6) Jocs.
7) Paquets de macros.
8) Ordres d'administració del sistema (generalment les que només el *root* pot utilitzar).
9) Rutines del nucli.

Nota

Ordres complementàries:

a) Manuals:

- man <ordre>
- man -k <paraula>

b) Comprimir:

- tar cvf <destinació> <origen>
- tar zcvf <destinació> <origen>
- gzip <file>
- bzip <file>

c) Espai de disc:

- df -k
- du -k <directori/file>

d) Paràmetres de sistema d'arxius:

- dumpe2fs <partició>

e) Sincronitzar sistema d'arxius:

- sync

Si hi ha més d'un manual disponible per a una mateixa paraula, el podem especificar indicant el número corresponent de la secció que ens interessa abans de la paraula; per exemple, `man 3 printf` (`man -k paraula` buscarà entre les pàgines del manual les que tinguin la "paraula" passada com a argument, equivalent a `apropos`, i `mandb` permetrà actualitzar la base de dades dels manuals). Si el manual no ens proporciona tota la informació que necessitem, podem utilitzar la instrucció `info`, que és el mateix que el manual però amb informació estesa.

Una altra ordre útil és la utilitzada per a comprimir un arxiu, agrupar-ne diversos en un de sol o veure què conté un arxiu comprimit. Si bé hi ha desenes de programes diferents en tots els sistemes GNU/Linux trobarem l'ordre `tar`. Aquest programa ens permet manipular de qualsevol manera un o diversos arxius per a comprimir-los, agrupar-los, etc. La seva sintaxi és `tar opcions arxiu-destinació arxius-origen`, en què si l'arxiu origen és una carpeta hi treballarà de manera recursiva i desarà a l'arxiu de destinació el contingut de tota la carpeta. Els paràmetres comuns són *c* per a crear i *f* si ho volem desar en un arxiu (`tar cf arxiu.tar o*` empaquetarà tots els arxius del directori actual que comencin per *o*). Si a més volguéssim comprimir, podríem utilitzar *czf*, i llavors s'utilitzaria el programa `gzip` després d'empaquetar-los. Per a desempaquetar un arxiu determinat, el paràmetre necessari és *x*, de manera que hauríem d'escriure `tar xf` per a indicar l'arxiu empaquetat. Si estigués comprimit, hauríem de passar *xzf*.

El programa `gzip` (utilitzat pel `tar` per a comprimir) usa un format de compressió propi i diferent del popular *zip* o del *compress* (estàndard en els *nix però obsolet) i que es pot utilitzar per a comprimir un arxiu de manera independent (`gunzip` per a fer el procés invers o `gzip -u`). Una altra aplicació de compressió bastant utilitzada i que proporciona molt bons resultats és el `bzip2`.

La gestió i manipulació dels discos durs de l'ordinador és un altre aspecte fonamental en les tasques d'administració del sistema. Més endavant es veurà tot un apartat per a discos, però ara tractarem les ordres útils per a obtenir informació dels discos. El disc dur es divideix en particions, a les quals podem accedir com si es tractés d'un dispositiu independent, i les denominarem *unitat*. Això és molt útil perquè ens permet separar de manera adequada la informació que tinguem en el sistema, tenir més d'un sistema operatiu instal·lat al mateix disc, etc. L'ordre `df` ens mostrarà, de cada unitat muntada en el sistema, l'espai que s'ha utilitzat i el que queda lliure, i l'ordre `du` ens mostra realment el que ens ocupa un fitxer en disc o un directori (ens ho mostrarà en blocs de discos però amb el paràmetre *-k* en kilobytes).

Un disc és organitzat en pistes i dins de les pistes en sectors (zones on es desarà la informació) i com aquest últim valor és configurable, quan es crea el sistema d'arxius amb finalitats d'optimitzar les prestacions de disc o espai utilitzat ens

pot interessar veure aquests paràmetres (per a sistemes ext2/3/4), per la qual cosa podem utilitzar l'ordre `dumpe2fs partició` i esbrinar els paràmetres amb els quals ha estat creat el disc.

Un altre concepte molt estès és la desfragmentació d'un disc, que no és més que la reorganització dels blocs dels fitxers perquè quedin en llocs consecutius i l'accés sigui més ràpid. En els sistemes de fitxers que utilitzem amb GNU/Linux no és necessari desfragmentar els discos (encara que hi ha programes amb aquesta finalitat), ja que el sistema s'encarrega automàticament de tenir el disc sempre desfragmentat, i a més, en el procés d'arrencada sempre es comproven els errors i es fa una desfragmentació total si és necessària.

L'optimització del sistema d'arxius de GNU/Linux prové del fet que totes les funcions del nucli que s'encarreguen de la gestió de fitxers utilitzen uns mètodes específics per a agilitar els processos de lectura i escriptura. Un és la utilització d'una memòria cau de disc per a evitar estar llegint constantment i escrivint al disc físic (procés lent i costós). Això pot representar problemes si tenim un tall d'alimentació, ja que les últimes operacions de lectura/escriptura no s'hauran desat perquè són en memòria. El programa `fsck` comprova i arregla un sistema de fitxers que hagi quedat en aquest estat. Encara que el podem executar quan vulguem (sempre que la partició no estigui muntada en un directori), el sistema operatiu mateix l'executa quan en el procés d'arrencada detecta que el sistema no es va tancar adequadament. Per això, per a apagar l'ordinador correctament hem d'executar l'ordre `shutdown`, que s'encarrega de llançar tots els processos necessaris perquè els programes acabin, es desmunti el sistema de fitxers, etc. En aquest sentit, el sistema de fitxers *ext3/4*, és més eficaç que l'*ext2*, ja que el *journaling* li permet recuperar més informació dels arxius perduts i de manera més eficient (la integritat física d'una partició es pot comprovar amb l'ordre `badblocks`). Si bé no és aconsellable, es pot desactivar la memòria cau de disc i si volem en algun moment bolcar de cau a disc, per a evitar problemes podem executar l'ordre `sync`.

S'ha de tenir en compte que la majoria de les ordres esmentades s'han d'executar com a *root* (o alguns dels usuaris que formin part del grup de *root*).

3. Instal·lació i arrencada de GNU/Linux (conceptes bàsics)

En aquest apartat veurem els passos essencials que se segueixen en la majoria dels processos d'instal·lació de GNU/Linux, i que seran complementats posteriorment amb els tallers d'instal·lació. Si bé cada distribució té el seu entorn d'instal·lació propi, en totes hi ha uns passos bàsics per a instal·lar el sistema operatiu, i que es descriuran de manera resumida. És important notar que avui dia qualsevol de les distribucions té una instal·lació molt optimitzada que necessita molt poca atenció de l'usuari, ja que obté informació del maquinari subjacent, per la qual cosa, generalment, per a un usuari novell no és necessari prendre decisions importants (distribucions com Debian/Ubuntu o Fedora, per exemple, són instal·lacions pràcticament automàtiques).

També hem de tenir en compte que un usuari novell pot iniciar el seu camí en el món Linux amb un altre tipus d'execucions de GNU/Linux que no modifiquen l'ordinador i permeten treballar en el sistema operatiu sense haver-ne d'instal·lar un altre. És altament recomanable iniciar els primers passos sobre un **GNU/Linux live:** l'usuari s'ha de baixar la imatge del sistema operatiu, crear-hi un CD o DVD i arrencar des d'aquest dispositiu sense tocar el disc dur de la màquina. Aquest tipus de distribucions (és recomanable utilitzar Knoppix http://www.knoppix.net/, per la seva eficiència i versatilitat en les seves versions per a CD, DVD o USB), o Ubuntu (en l'opció "prova de l'SO"). Aquestes distribucions tenen "l'inconvenient" que per a desar el treball de l'usuari s'ha de fer sobre un dispositiu de disc USB o sobre un dispositiu en la xarxa (com ara Dropbox), ja que si es desa sobre el sistema d'arxiu, com que és a la RAM, es perdran les dades.

VirtualBox

VirtualBox que permet arrencar una imatge o instal·lar un SO sobre un altre SO amfitrió (*host*) tant en 32 bits com en 64. Si no volem fer una instal·lació des de 0 es pot trobar una gran quantitat de distribucions amb les seves imatges ja creades com per exemple en http://virtualboxes.org/images/.

Una altra opció interessant és utilitzar algun servei en el núvol (*cloud*) que permetrà executar una màquina virtual Linux des d'un navegador i per tant des de qualsevol dispositiu sense tenir res instal·lat. Exemples d'aquests serveis són els de TutorialPoint/CodingGround, orientat a l'aprenentatge de diferents entorns/programes, o Cloud9, orientat al desenvolupament d'aplicacions.

Altra opció totalment recomanable com a introducció (o com a forma de treball habitual) sense haver de tocar el sistema operatiu d'una màquina és treballar amb màquines virtualitzades. Per a això, es recomana utilitzar **VirtualBox** http://www.virtualbox.org/, que permet arrencar una imatge o instal·larne una sobre un sistema operatiu amfitrió (*host*); tant en 32 bits com en 64 bits, és altament configurable i si no volem fer una instal·lació es poden trobar gran quantitat de distribucions amb les seves imatges ja fetes, com per exemple a

http://virtualboxes.org/images/, que té aproximadament 30 distribucions de Linux i 15 distribucions d'altres sistemes *nix o fins i tot d'Android o sistemes no *nix.

És important que abans d'instal·lar un nou sistema coneguem els components maquinari que tenim instal·lats en el nostre ordinador per a poder configurar-lo adequadament, encara que la distribució que utilitzem incorpori detecció de maquinari. També és possible que en un sol disc dur tinguem instal·lats dos sistemes operatius (*dualboot*) o més totalment independents, i si bé el procés d'instal·lació d'un altre sistema operatiu al mateix disc no hauria d'interferir amb les particions dels altres, és aconsellable fer còpies de seguretat de tots els documents importants. En gairebé totes les distribucions de Linux actuals la instal·lació de Linux sobre un disc que contingui Windows, per exemple, és totalment gairebé automàtica i lliure de problemes; no obstant això, es recomana fer còpies de seguretat dels arxius importants de Windows.

És necessari, abans de començar, tenir informació de la marca i el model de la targeta gràfica, la de so i la de xarxa; la marca, el tipus i les característiques del monitor, i també qualsevol altre maquinari especial que tinguem (la resta del maquinari serà detectat pel sistema: placa base, la CPU i la memòria RAM).

Generalment, totes les distribucions de GNU/Linux proporcionen algun tipus de mitjà per a l'arrencada del procés d'instal·lació, i el més comú és un CD o DVD d'arrencada, per la qual cosa és necessari configurar la BIOS perquè pugui arrencar (*boot*) des de CD/DVD. Les instal·lacions són autoguiades i és important parar atenció a la selecció de l'idioma i del teclat per a evitar problemes des de l'inici (si bé es podrà configurar posteriorment).

Per a usuaris ja avançats, hi ha alguna altra manera d'instal·lar GNU/Linux que permet fer la instal·lació des de qualsevol mitjà: FTP, HTTP, disc dur, NFS, USB, però és recomanable no utilitzar-les com a primera experiència.

La partició del disc dur és una de les parts més crítiques de tot el procés, ja que implica dividir el disc dur en diverses seccions que seran considerades independents. Si ja tenim un sistema operatiu instal·lat al nostre ordinador, el disc estarà particionat en una o diverses particions, però si el disc és nou, tindrà una única partició. Per a instal·lar GNU/Linux hem de disposar, almenys, d'una partició per a ús propi (si bé és possible instal·lar-lo sobre altres sistemes d'arxius, no és recomanable aquesta opció per qüestions de rendiment i fiabilitat) i una altra de més petita per a una extensió de la memòria RAM de l'ordinador, anomenada *partició de swap* (generalment del doble de la memòria RAM instal·lada).

El procediment més comú (i més conservador) per a reduir, crear o canviar la mida de les particions Windows és utilitzar eines com les disponibles en el propi sistema operatiu Windows (administració de discos o l'aplicació `fips` amb llicència GPL i per a sistemes FAT). També pot utilitzar-se qualsevol Linux *live* que, normalment, fan servir l'aplicació `gparted` per a modificar la mida d'una partició ja creada sense perdre'n el contingut (si bé es recomana fer còpies de seguretat dels arxius més importants). El procediment recomanable és engegar amb una Linux *live* i utilitzar la instrucció `gparted`, que és molt eficient i segur. A Ubuntu, per exemple, s'executarà automàticament quan detecti que no hi ha prou espai durant el procés d'instal·lació [How to Resize Windows Partitions].

MBR

La informació de particions i el programa de càrrega d'un sistema operatiu es desen en una zona de dades reservada anomenada *MBR* (*master boot record*) sobre el primer disc.

Com ja s'ha comentat, és recomanable que GNU/Linux utilitzi dues particions al disc dur (una per al sistema de fitxers i l'altra per a la *swap*). Si bé totes les distribucions tenen un particionament guiat, es pot fer de manera manual recorrent a diferents utilitats (`fdisk`, `cfdisk`, `diskDruid`, etc.). La manera com el sistema operatiu GNU/Linux identifica els discos és amb */dev/hdX* per als discos IDE i */dev/sdX* per als SCSI i Serial ATA, en els quals X és una lletra, corresponent al disc al qual ens vulguem referir: */dev/hda* és el mestre del primer canal IDE, */dev/hdb* el segon i així successivament (o */dev/sda* el primer disc SCSI o SATA i */dev/sdb* el segon, etc.). L'aplicació d'instal·lació ens farà una llista dels discos i haurem d'escollir sobre quin volem fer la instal·lació.

En Debian (des de la versió 6 Squeeze) es va unificar aquest esquema de noms en el nucli Linux, i tots els discos durs (IDE/PATA, SATA, SCSI, USB, IEEE 1394) es representen amb */dev/sd**. En aquest cas, per a tots els discos s'han de representar les particions pel seu número en el disc en el qual existeix: per exemple, */dev/sda1* és la primera partició del primer disc, i */dev/sdb3* és la tercera partició del segon disc.

Un segon aspecte a tenir en compte és el tipus de particions que es poden crear i que denominarem *primàries*, *esteses* i *lògiques*. A causa que l'arquitectura de PC (i386) està limitada a **quatre particions primàries** per disc i poder superar aquesta limitació una d'elles ha de ser creada com una partició **estesa** i després aquesta podrà contenir diverses particions **secundàries o lògiques** addicionals. Aquestes particions secundàries/lògiques s'han de numerar a partir del 5. Per tant, la primera partició secundària seria */dev/sda5* seguida a continuació de */dev/sda6*, etc. Hi pot haver un màxim de seixanta-quatre particions secundàries, però només setze particions per cada partició estesa.

No és fàcil recordar quin disc està connectat a quin controlador (sobretot si són discos removibles), però el gestor de dispositius (anomenat *udev*) crea, a més de */dev/sd**, enllaços simbòlics amb noms fixos que pot utilitzar per a identificar un disc dur de manera unívoca. Aquests enllaços simbòlics s'emmagatzemen a */dev/disk/by-id*. Per exemple, en la llista següent d'aquest directori:

```
lrwxrwxrwx 1 root root 9 Jun 3 16:26 ata-HD_VB09c4cef5-d95686b9 ->../../sdd
```

```
lrwxrwxrwx 1 root root 10 Jun 3 16:26 ata-HD_VB09c4cef5-d95686b9-part1 ->../../sdd1
...
lrwxrwxrwx 1 root root 9 Jun 3 16:26 scsi-SATA_HD_VB621f947e-e8e45580->../../sda
lrwxrwxrwx 1 root root 10 Jun 3 16:26 scsi-SATA_HD_VB621f947e-e8e45580-part1-> ../../sda1
```

En què mostra dos discos (un SATA i un altre ATA) amb informació dels models, el número de sèrie i on s'han assignat.

Si no necessitem més de 4 particions, podem elegir qualsevol dels dos tipus. Si en necessitem més, haurem de tenir en compte que les lògiques se situen dins d'una de primària (fins a un màxim de 16 per a cada una), de manera que no podem tenir 4 particions primàries creades i després afegir-ne de lògiques. En aquest cas, n'hauríem de crear 3 de primàries i fins a 16 lògiques en la quarta partició primària.

Partició activa

De totes les particions d'un disc dur en podem elegir una perquè sigui l'activa. Aquest indicador (*flag*) serveix per a indicar a la BIOS o a l'EFI del sistema (sistema d'inicialització i càrrega del sistema operatiu) quina és la partició que ha d'iniciar si en l'MBR no troba cap programa d'arrencada.

Quan es crea una partició s'ha d'indicar quin sistema de fitxers utilitzarà (Linux ext3, Linux ext4, Linux *swap* o un altre) i una vegada fetes les particions, desarem la configuració i hem d'indicar al procés d'instal·lació on volem situar l'arrel del sistema de fitxers (*root filesystem*) i la *swap* del sistema, i a partir d'aquest moment es podrà continuar amb la instal·lació.

Una part important de la instal·lació són els mòduls del nucli, que són parts de programari especialitzades que treballen amb alguna part del maquinari o del sistema. En les distribucions actuals simplement s'ha de seleccionar quin dispositiu tenim (monitor, xarxa, so, gràfics) i la instal·lació carregarà pràcticament tots els mòduls necessaris, encara que en la majoria hi ha processos d'autodetecció, per la qual cosa no serà necessari seleccionar pràcticament res.

Si algun mòdul no s'inclou durant la instal·lació, és possible fer-ho després amb ordres com `insmod` o `modprobe` (per a afegir un nou mòdul), `lsmod` (per a fer una llista dels instal·lats), `rmmod` (per a eliminar-ne algun) i també `modprobe` (per a provar-ne algun i, si funciona correctament, incloure'l en el nucli). Tots aquests mòduls són fitxers binaris que solem trobar en el directori */lib/modules/versió-del-sistema-operatiu/*.

Després de configurar els dispositius, es configurarà la xarxa (si tenim la targeta necessària, tant si és per cable com sense cable). Encara que en aquest document no entrarem en detall sobre xarxes, descriurem els conceptes necessaris per a poder fer aquest pas de manera bàsica. La primera dada que sol·licitarà la instal·lació és el nom del sistema (per a referir-nos-hi de manera amigable i que pot ser un nom amb lletres i xifres, com ara *NteumSys*) i a continuació demanarà si a la nostra xarxa utilitzem un mecanisme anomenat DHCP (un servidor especial que s'encarrega d'assignar automàticament les IP als ordinadors que engeguen; molts dels *routers* domèstics d'ADSL incorporen aquest mecanisme per a assignar els paràmetres de xarxa). Si utilitzem aquest mecanisme, ho hem d'indicar, i si no, ens preguntarà la IP (quatre nombres entre 0 i 255 separats per punts) i la màscara del nostre ordinador (quatre nombres entre

0 i 255, i és comú utilitzar 255.255.255.0). Si no coneixem aquestes dades, ens hem de dirigir a l'administrador de la nostra xarxa. Seguidament haurem d'introduir l'IP de la passarel·la de la nostra xarxa (dispositiu o ordinador que actua de pont entre la nostra xarxa local i Internet; si no tenim cap dispositiu d'aquest tipus, podem deixar en blanc aquest camp). Si tenim un servidor de dhcp, no cal que ens amoïnem per cap d'aquests paràmetres perquè aquest servei ja els configura.

A continuació, hem d'especificar el servidor (o servidors) de noms que utilitzem, anomenat DNS, que és una màquina que ens proporciona l'equivalència entre un nom i una adreça IP (és a dir, ens permetrà conèixer per exemple la IP de www.uoc.es de manera transparent). Si no sabem quins són, haurem de recórrer a l'administrador de la xarxa (o seleccionar-ne alguns de domini public, com ara els de Google, que són 8.8.8.8 i 8.8.4.4).

Adreça IP

Una adreça IP és la identificació d'un ordinador dins d'una xarxa quan utilitzem el protocol TCP/IP (protocol utilitzat en Internet).

Si som en una xarxa local podem consultar l'administrador perquè ens proporcioni tota la informació necessària o, si tenim un altre sistema operatiu instal·lat a l'ordinador, en podrem obtenir aquesta informació, però en cap cas no hem d'inventar aquests valors, ja que si l'ordinador està connectat a una xarxa local pot generar problemes a altres ordinadors.

Una vegada configurats aquests aspectes, haurem de seleccionar si volem instal·lar un petit programa al disc dur perquè en el procés d'arrencada de l'ordinador puguem elegir quin sistema operatiu dels que tenim instal·lats volem arrencar (fins i tot si només hem instal·lat GNU/Linux). Les aplicacions més usuals són el `lilo` (*Linux loader*) o el `grub` (recomanat) (*GNU, grand unified bootloader*), que tenen per objectiu iniciar el procés de càrrega i execució del nucli del sistema operatiu que li indiquem interactivament o per defecte després d'un temps d'espera. Totes les distribucions (si no hi ha problemes) detecten si tenim algun altre sistema operatiu instal·lat en el disc dur i configuren automàticament el sistema d'arrencada. Aquest programa generalment s'instal·la en l'MBR (*Master Boot Record*) del disc mestre del primer canal IDE o SCSI, que és el primer lloc que la BIOS o EFI de l'ordinador inspecciona buscant un programa d'aquestes característiques.

L'últim pas de la instal·lació és la selecció de paquets per instal·lar a més dels estrictament necessaris per al funcionament bàsic del sistema operatiu. La majoria dels processos d'instal·lació inclouen dues maneres de seleccionar els programes del sistema: bàsic o expert. Amb el procés de selecció bàsic, s'agrupen els paquets disponibles per a grans grups de programes: administració, desenvolupament de programari, ofimàtica, matemàtiques, etc.; és una opció recomanable per a fer els primers passos. Si no seleccionem un paquet després es podrà fer una instal·lació posterior amb l'eina de la qual disposen totes les distribucions per a instal·lar o desinstal·lar paquets. Debian GNU/Linux

va ser una de les primeres a incloure aplicacions per a gestionar els paquets; s'anomena `apt` i és molt útil per a fer el manteniment i actualització de tots els paquets instal·lats fins i tot en el sistema operatiu mateix.

Si la instal·lació no ha funcionat correctament, pot passar que no es pugui arrencar cap dels sistemes operatius instal·lats (ni el nou ni l'anterior), però totes les distribucions tenen un mode de rescat en l'arrencada (*rescue mode*), que ens permetrà arrencar el sistema GNU/Linux des del CD/DVD, accedir al disc dur i arreglar aquelles coses que no han funcionat o recuperar el sistema operatiu inicial, si bé per a alguns casos són necessaris una sèrie de coneixements avançats en funció de quina hagi estat la causa d'error. Si tenim *dual-boot* i l'altre sistema operatiu és Windows, sempre és possible utilitzar el Windows-Rescue-BootDisk i executar des d'un terminal l'ordre `bootrec /fixmbr` (que escriu un MBR compatible amb Windows en la partició del sistema i serveix per a reiniciar l'antic MBR de Windows) i `bootrec /fixboot` (que escriu un nou sector d'arrencada en la partició del sistema utilitzant-ne un de compatible amb Windows).

4. Configuracions bàsiques

4.1. El sistema d'entrada

Tant en mode gràfic com en mode text, el procediment d'identificació i entrada es denomina connexió (*login*). Generalment, el sistema arrencarà en mode gràfic, però podem passar a mode text seleccionant el tipus de sessió a la pantalla d'entrada o amb Ctrl-Alt-F1... F6 per a passar a un dels 6 terminals en mode text. Ctrl-Alt-F7 retorna al mode gràfic.

En mode text es llancen 6 terminals independents, als quals es pot accedir mitjançant Ctrl-Alt-F1... F6, que permetran treballar simultàniament amb diferents comptes alhora. El procés de connexió posa en pantalla, primer, un missatge que es pot modificar fàcilment des de l'arxiu */etc/issue* i que admet diferents variables (*d*, data actual, *s*, nom del SO, *t*, hora actual, etc.) i en segon lloc el missatge del dia des de */etc/motd* (creant un fitxer buit anomenat *.hushlogin* en el nostre directori d'usuari s'anul·la aquest missatge). A continuació el procés de connexió llança l'intèrpret per defecte per a l'usuari (indicat en l'últim camp a */etc/passwd*).

L'intèrpret d'ordres executa l'arxiu *.profile* del directori de l'usuari per a les opcions per defecte d'aquest mateix usuari, que es complementa amb */etc/profile*, que configura opcions per defecte per a tots els usuaris. Cada intèrpret a més té arxius de configuració propis, com per exemple el *shell Bash*, que executa, a més, dos fitxers més anomenats *.bashprofile* (que s'executa en cada connexió) i *.bashrc* (que s'executa cada vegada que s'obre un nou terminal). Veurem algunes de les instruccions que podem trobar en aquests arxius:

Recordeu

Totes les línies que comencen amb # són comentaris.

```
# ~/.profile: executat per l'intèrpret a l'entrada
# No serà llegit per bash si hi ha ~/.bash_profile o ~/.bash_login
# Vegeu-ne exemples a /usr/share/doc/adduser/examples/adduser.local.conf.examples/skel
# El valor per defecte d'umask

umask 022

# Si està executant bash executa .bashrc si hi és
if [ -n "$BASH_VERSION" ]; then

  if [ -f "$HOME/.bashrc" ]; then

    . "$HOME/.bashrc"
```

```
    fi
  fi

# Inclou en PATH un directori bin de l'usuari

if [ -d "$HOME/bin" ] ; then PATH="$HOME/bin:$PATH"
 fi

 # Canvia l'indicador (prompt)
export PS1='\h:\w\$ '

_____________________________________________________________

# # ~/.bashrc: executat per bash(1) per a terminals no d'entrada.
# Si no s'executa interactivament no fa res.
[ -z "$PS1" ] && return
# Habilita el suport de color per a l'ordre ls
if [ -x /usr/bin/dircolors ]; then
   eval "`dircolors -b`"
   alias ls='ls --color=auto' alias dir='dir --color=auto'
   fi

# altres àlies
alias ll='ls -l'

# habilita programmable completion features
if [ -f /etc/bash_completion ]; then
 . /etc/bash_completion
fi
```

Com es pot observar en aquests fitxers (que són un exemple de *shell script* i que veurem més endavant), s'inclouen diferents definicions de variables (PATH, per exemple, que és la variable on es buscaran els programes per a executar-los sense haver d'incloure tot el camí; PS1, que és la variable que emmagatzema el *prompt*, que és el caràcter que surt a l'esquerra de la línia d'ordres acabat en $ o % per a l'usuari normal i en # per al *root*). També tenim àlies d'ordres o execució condicional d'altres arxius (en el primer si s'està executant el *bash* executa el *$HOME/.baslırc*). Si es volen executar programes del directori des del qual estem situats sense necessitat de posar ". /" al principi, podríem afegir aquesta entrada en la declaració del PATH, però generalment no s'inclou, ja que pot ocasionar problemes de seguretat. Per al `prompt` hem utilitzat l'ordre `export` per a definir les anomenades *variables d'entorn*, que es mantenen durant tota la sessió i es poden consultar amb la mateixa ordre.

Amb `set` i `unset` també podem inicialitzar o treure altres variables o atributs representats per defecte (amb `echo $variable` podrem consultar el valor de cada variable). Algunes de Bash són:

- PWD: directori actual.
- LOGNAME: nom de l'usuari connectat.
- BASH_VERSION: versió del Bash que utilitzem.
- SHELL: intèrpret d'ordres utilitzat.
- RANDOM: genera un nombre aleatori diferent cada vegada que en mostrem el contingut.
- SECONDS: nombre de segons que han passat des que hem obert l'intèrpret d'ordres.
- HOSTNAME: nom del sistema.
- OSTYPE: tipus de sistema operatiu que estem utilitzant.
- MACHTYPE: arquitectura de l'ordinador.
- HOME: directori personal de l'usuari.
- HISTFILESIZE: mida de l'arxiu d'història (nombre d'ordres que es desen).
- HISTCMD: número d'ordre actual en la història.
- HISTFILE: fitxer en el qual es desa la història d'ordres.

La configuració addicional de la connexió gràfica es farà per mitjà de l'entorn gràfic i dependrà del tipus d'entorn.

4.2. L'intèrpret d'ordres (*shell*)

Com ja hem avançat, el terme genèric *shell* s'utilitza per a denominar un programa que serveix d'interfície entre l'usuari i el nucli del sistema GNU/Linux. En aquest apartat veurem algunes característiques bàsiques dels intèrprets interactius de text, que és el programa que veurà l'usuari (i l'atendrà) una vegada fet el procediment de connexió.

L'intèrpret és el que els usuaris veuen del sistema, ja que la resta del sistema operatiu roman ocult als seus ulls. L'intèrpret està escrit de la mateixa manera que un procés (programa) d'usuari; no està integrat en el nucli i s'executa com un programa més de l'usuari.

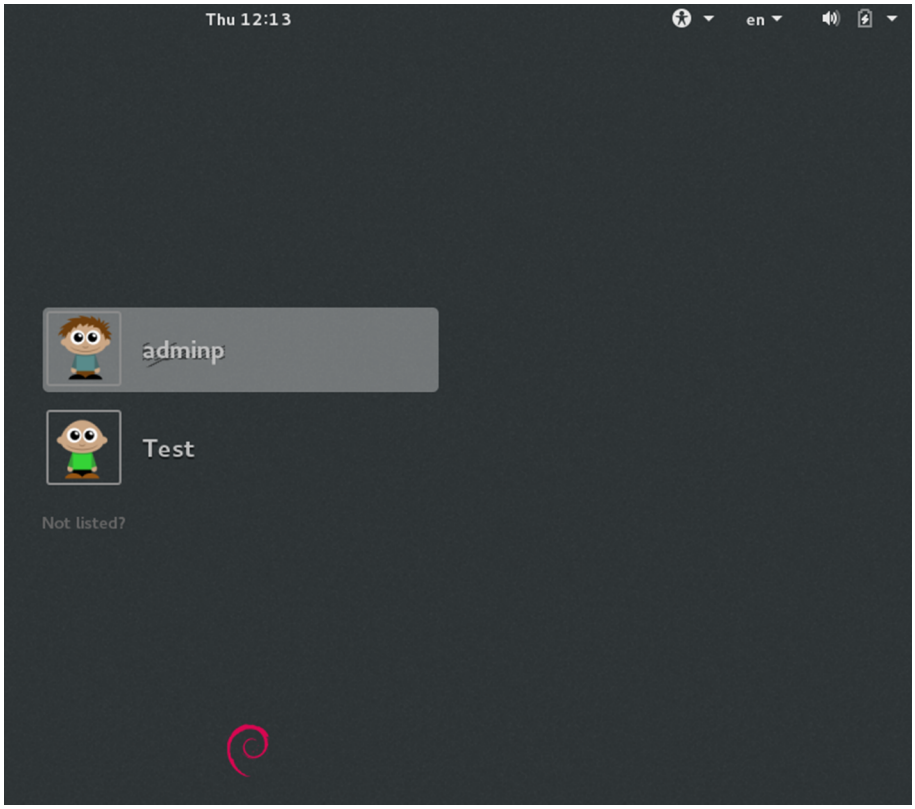
Quan el sistema GNU/Linux arrenca, sol presentar als usuaris una interfície d'entrada que pot ser gràfica o en mode text.

En els sistemes operatius actuals, el procediment d'arrencada original (anomenat *SysV* o *init)* ha canviat per un nou procediment anomenat *systemd*, que permet una considerable quantitat de millores i que ha estat adoptat per la majoria de distribucions (si bé algunes d'elles encara són compatibles amb el sistema anterior). La seqüència d'arrencada s'inicia a la BIOS de la màquina, detecta el disc, carrega el MBR i executa el programa de càrrega del sistema operatiu *(bootloader)*. Aquest ubica el sistema operatiu, el carrega i l'executa. A continuació, el sistema operatiu inicialitza les variables, els mòduls, la partició

de disc *(root filesystem)* i, finalment, executa el primer programa anomenat *init*. Es troben generalment a la memòria RAM —d'aquí el seu nom *initramfs*—, i és el segon pas en l'arrencada.

Quan ja està muntada la partició principal del disc i els dispositius necessaris inicialitzats per a l'arrencada, el control passa a l'*init* "real" anomenat *systemd*, que inicialitzarà un conjunt de processos (a través de *scripts* de configuració que es troben a */etc/systemd* per inicialitzar teclat, controladors, sistemes d'arxiu, xarxa, serveis,...), entre els quals es troba aquell que permetrà realitzar el *login* de l'usuari (configurat a */lib/systemd/system/systemd-logind.service*, i que es diu *systemd-logind)*. El resultat de la seva execució serà crear una consola per permetre a l'usuari iniciar una sessió o desencadenar els passos per activar una sessió en l'entorn gràfic.

En el mode d'arrencada gràfica, la interfície està composta per algun gestor d'accés que administra el procés de connexió de l'usuari des d'una pantalla (caràtula) gràfica, en la qual se sol·licita la informació d'entrada corresponent: el seu identificador com a usuari i la seva contrasenya. En GNU/Linux solen ser habituals els gestors d'accés: **xdm** (propi d'X Window), **gdm** (Gnome) i **kdm** (KDE), i també algun altre associat a diferents gestors de finestres (*window managers*). La la figura següent mostra l'accés donat per gdm3 en Debian:



Una vegada validat l'accés, l'usuari trobarà una interfície gràfica amb algun gestor d'escriptoris, com Gnome o KDE (vegeu l'últim punt del capítol). Des del mode gràfic també és possible "interactuar" amb el sistema per mitjà d'un entorn de treball en mode text simplement obrint un terminal (per exemple,

Xterm) des dels menús de la interfície gràfica (en Debian-Gnome3 → Activities i veurem el terminal en la barra lateral o, si no, en Gnome2 Aplicacions → Accessoris → Terminal). És important notar que moltes de les versions actuals del Linux no tenen el mateix gestor d'escriptori i les opcions dels menús/ubicació dels programes poden variar.

Si l'accés és en mode text (anomenat també *mode consola*), una vegada identificats obtindrem l'accés a l'intèrpret de manera interactiva (es pot passar del mode gràfic al text amb Crtl+Alt+F1 a F6, que permetrà tenir 6 consoles diferents i tornar-hi amb Crtl+Alt+F7). Un altre mode de treball amb un intèrpret interactiu és per mitjà d'una connexió remota des d'una altra màquina connectada en xarxa i amb aplicacions com ara Telnet o `rlogin` (poc utilitzades per insegures), `ssh`, o gràfiques com els emuladors X Window. Per exemple, des de sistemes Windows és possible connectar-se en mode text a sistemes Linux utilitzant l'aplicació `putty` o `mobaXterm`. Aquesta última inclou un servidor d'Xwindows, per la qual cosa es pot connectar al sistema Linux en mode text però executar aplicacions gràfiques sobre Linux visualitzant-les sobre Windows. Si es vol fer el mateix amb `putty`, és necessari instal·lar sobre Windows un servidor d'Xwindows com per exemple **Xming**.

Una vegada iniciat l'intèrpret interactiu [Qui01], es mostra un indicador d'ordres (símbol o seqüència de caràcters com $, %, # –utilitzat generalment per a identificar l'usuari *root*– o també quelcom configurable per l'usuari com *MySys*) que indica a l'usuari que pot introduir una línia d'ordres.

Després de la introducció, l'intèrpret assumeix la responsabilitat de validar la sintaxi i posar els processos necessaris en execució, mitjançant una sèrie de seqüències o fases:

1) Llegir i interpretar la línia d'ordres.

2) Avaluar els caràcters comodí com $ * ? o d'altres.

3) Gestionar les redireccions E/S necessàries, les *pipes* i els processos en segon pla (*background*) necessaris (&).

4) Manejar senyals.

5) Preparar l'execució dels programes.

Normalment, les línies d'ordres podran ser execucions d'ordres del sistema, ordres pròpies de l'intèrpret interactiu, l'engegada d'aplicacions o *shell scripts* (seqüències d'ordres, variables, sentències de control, etc., que generalment es troben en un arxiu ASCII).

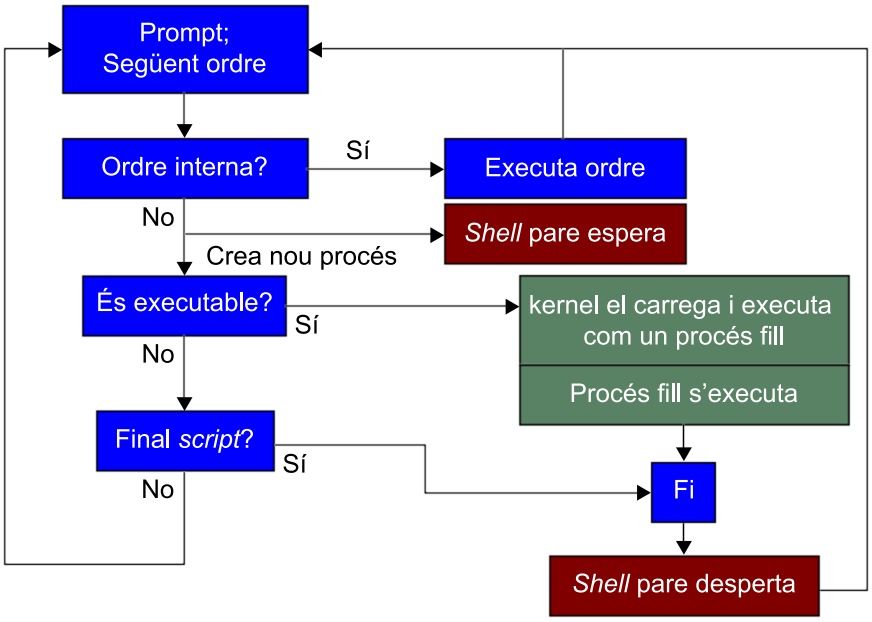
Els fitxers de *script* són directament executables pel sistema sota el nom que s'hagi donat al fitxer. Per a executar-los, s'haurà d'invocar l'intèrpret juntament amb el nom del fitxer, o bé es donaran permisos d'execució al *shell script* (`chmod 755 script`). Atès que generalment (com ja s'ha esmentat) el directori actual no forma part de la variable PATH, per executar un ordre/*script* que el directori actual no estigui en la variable PATH haurem de fer `./nom_script`, la qual cosa significa directori actual i *script* a executar (això no passa amb les ordres internes o les que estan en el */bin* o */usr/bin* per exemple, ja que aquests directoris formen part de la variable PATH que podem veure amb l'ordre `echo $PATH`).

En certa manera, podem veure un *shell script* com un codi d'un llenguatge interpretat que s'executa sobre l'intèrpret interactiu corresponent. Per a l'administrador, els *shell scripts* són molt importants bàsicament per dues raons:

1) La configuració del sistema i de la majoria dels serveis proporcionats es fan mitjançant eines en forma de *shell scripts*.

2) La manera principal d'automatitzar processos d'administració és mitjançant la creació de *shell scripts* per part de l'administrador.

La figura següent mostra el flux de control bàsic d'un intèrpret d'ordres:



Tots els programes invocats mitjançant un intèrpret posseeixen tres fitxers predefinits, especificats pels descriptors de fitxers corresponents (*file handles*). Per defecte, aquests fitxers són:

1) ***standard input*** (entrada estàndard): normalment assignada al teclat del terminal (consola); usa el descriptor número 0 (en UNIX els fitxers utilitzen descriptors sencers).

2) ***standard output*** (sortida estàndard): normalment assignada a la pantalla del terminal; usa el descriptor 1.

3) ***standard error*** (sortida estàndard d'errors): normalment assignada a la pantalla del terminal; utilitza el descriptor 2.

Això ens indica que qualsevol programa executat des de l'intèrpret tindrà per defecte l'entrada associada al teclat del terminal, la seva sortida cap a la pantalla i, en el cas de produir-se errors, també els envia a la pantalla.

A més, els intèrprets poden proporcionar tres mecanismes següents de gestió de l'E/S:

1) Redirecció: atès que els sistemes UNIX tracten de la mateixa manera els dispositius d'E/S i els fitxers, l'intèrpret els tracta a tots simplement com a fitxers. Des del punt de vista de l'usuari, es poden reassignar els descriptors dels fitxers perquè els fluxos de dades d'un descriptor vagin a qualsevol altre descriptor; això s'anomena *redirecció*. Per exemple, ens referirem a la redirecció dels descriptors 0 o 1 com a la redirecció de l'E/S estàndard. Per a això s'utilitzen els símbols > >> < <<; per exemple `ls > dir.txt` redirecciona la sortida de la instrucció *ls* a un fitxer anomenat *dir.txt*, que si existeix s'esborra i desa la sortida de l'ordre, i si no, es crea i es desa la sortida. Si fos `ls >> dir.txt` la sortida s'afegeix al final si el fitxer existeix.

2) Tubs (*pipes*): la sortida estàndard d'un programa es pot usar com a entrada estàndard d'un altre per mitjà de *pipes*. Diversos programes poden ser connectats entre si mitjançant *pipes* per a formar el que es denomina un *pipeline*. Per exemple `ls | sort | more`, en què la sortida de la instrucció `ls` s'ordena (`sort`) i la sortida d'aquest es mostra per pantalla en forma paginada.

3) Concurrència de programes d'usuari: els usuaris poden executar diversos programes simultàniament, tot indicant que l'execució es produirà en segon terme (*background*), oposat al primer terme (o *foreground*), en què es té un control exclusiu de pantalla. Un altre ús consisteix a permetre tasques llargues en segon terme quan interactuem l'intèrpret amb altres programes en primer terme. Per exemple, `ls > dir.txt &` s'executarà en segon terme i permetrà a l'usuari continuar interactuant amb l'intèrpret mentre l'ordre es va executant.

L'intèrpret d'ordres per defecte és el que s'especifica en l'últim camp del fitxer */etc/passwd* per a l'usuari, i es pot esbrinar des de la línia d'ordres pel valor de la variable d'entorn amb `echo $SHELL`

Algunes consideracions comunes a tots els intèrprets d'ordres:

- Tots permeten l'escriptura de *shell scripts*, que posteriorment són interpretats executant-los o bé mitjançant el nom (si el fitxer té permís d'execució) o bé passant-lo com a paràmetre a la instrucció de l'intèrpret (per exemple `bash -xv mishell`, que executarà en mode descriptiu mostrant les variables i la seva execució de les sentències que existeixin en el *shell-script* mishell).

- Els usuaris del sistema tenen un intèrpret associat per defecte. Aquesta informació es proporciona en crear els comptes dels usuaris. L'administrador assigna un intèrpret a cada usuari, o si no s'assigna l'intèrpret per defecte (Bash en GNU/Linux). Aquesta informació es desa en el fitxer */etc/passwd*, i es pot canviar amb l'ordre `chsh`; aquesta mateixa instrucció amb l'opció `-l` ens fa una llista els intèrprets disponibles en el sistema (o vegeu també */etc/shells*).

- Cada intèrpret és en realitat una instrucció executable, normalment present en els directoris */bin* en GNU/Linux (o */usr/bin*). Per la qual cosa, per a canviar d'un *shell* a un altre solament s'ha d'executar el desitjat (per exemple, si estic en Bash Shell i vull canviar-me a Korn, simplement he de fer `ksh` i Crtl+D per a tornar a Bash).

- Es poden escriure *shell scripts* en qualsevol intèrpret, però ajustant-se a la sintaxi de cadascun, que normalment és diferent (de vegades hi ha només petites diferències). La sintaxi de les construccions, i també les ordres internes, estan documentats a la pàgina `man` de cada intèrpret (`man bash`, per exemple).

- Cada intèrpret té alguns fitxers d'arrencada associats (fitxers d'inicialització), i cada usuari els pot adaptar a les seves necessitats, incloent-hi codi, variables, rutes (*path*)...

- La potència en la programació es troba a combinar la sintaxi de cada intèrpret (de les seves construccions), amb les ordres internes de cada intèrpret, i una sèrie d'ordres UNIX molt utilitzades en els *scripts*, com per exemple `cut`, `sort`, `cat`, `more`, `echo`, `grep`, `wc`, `awk`, `sed`, `mv`, `ls`, `cp`...

- Si com a usuaris estem utilitzant un intèrpret determinat, res no ens impedeix arrencar una còpia nova de l'intèrpret (anomenat *subshell*), tant si és el mateix com un altre de diferent. Senzillament, l'invoquem pel nom de l'executable, ja sigui `sh`, `bash`, `csh` o `ksh`. També quan executem un *shell script* es llança un *subshell* amb l'intèrpret que correspongui per a executar l'*script* demanat.

Atesa la importància dels *shell scripts* en les tasques d'administració d'un sistema UNIX, en el mòdul següent es veuran alguns detalls més sobre l'intèrpret i exemples de sintaxi i de programació aplicats a `bash`.

4.3. El sistema d'arrencada

Ja hem vist en el punt anterior la manera de configurar durant la instal·lació de l'arrencada del sistema per mitjà d'un gestor com `lilo` o `grub` (recomanable). Com ja sabem, a partir de la BIOS o EFI, l'ordinador llegeix l'MBR del primer disc mestre i executa el gestor d'arrencada (si no es troba aquest programa, s'inspecciona el sector d'arrencada de la partició activa del disc) encara que recomanem instal·lar `grub` a l'MBR, que és el primer lloc que s'inspecciona.

El `grub` ens permet múltiples configuracions com ara, tenir un petit intèrpret d'ordres en arrencar l'ordinador, accedir als arxius de les particions del disc sense carregar cap sistema operatiu, etc. En aquest subapartat només veurem algunes configuracions bàsiques, però si es necessiten opcions avançades es pot consultar la documentació existent a `info grub`.

Com ja hem dit, GRUB és un gestor d'arrencada i és el primer programari que s'executa quan s'inicia un equip. És responsable de la càrrega i la transferència de control al nucli (*kernel*) d'un sistema operatiu, el qual al seu torn inicialitza la resta del sistema i pot carregar una gran varietat de sistemes operatius tant lliures com propietaris.

Una de les característiques importants de GRUB és la flexibilitat, i permet carregar un sistema operatiu en la forma que vulgui i sense necessitat de tenir una posició física predeterminada (per exemple, es pot carregar el *kernel* simplement especificant-ne el nom d'arxiu i la unitat i partició on aquest resideix).

En arrencar amb GRUB es pot utilitzar una interfície de línia d'ordres o una interfície de menú. En la primera s'han d'escriure les ordres, mentre que en la segona només s'ha d'escollir l'opció que es vulgui o editar algunes de les opcions abans de carregar-la.

La història de GRUB està marcada per un canvi important a partir del GRUB 2 en el qual és reescrit i canvia la seva estructura/flux/configuració per a adaptar-se a noves necessitats (el GRUB anterior contínua en algunes –poques– distribucions i es denomina *GRUB Legacy*).

La sintaxi de dispositiu utilitzat en GRUB és diferent de com es pot considerar dins del sistema operatiu, així per exemple la sintaxi és (hd0, msdos2), en què hd significa que és una unitat de disc dur, i el primer número, 0, indica el número de la unitat, és a dir, el primer disc dur; la cadena msdos indica l'esquema de partició, mentre que el número 2 indica el número de partició

(els números de particions es compten des d'1 i no des de 0 com és el cas en les versions anteriors de GRUB). GRUB no distingeix un dispositiu IDE d'un SCSI, simplement compta el nombre d'unitats de 0, independentment del seu tipus.

Per a identificar un arxiu, simplement hem de tenir, per exemple, (hd0, msdos1)/vmlinuz, en què s'especifica l'arxiu anomenat *vmlinuz*, que es troba en la primera partició de la primera unitat de disc dur.

La instal·lació de paquet GRUB (generalment ja estarà instal·lat en el sistema operatiu GNU/Linux) utilitza per defecte *boot images* que estaran en el directori */usr/lib/grub/<cpu>-<platform>* anomenat *image directory* i el directori on el GRUB necessita trobar-les (generalment */boot*) serà anomenat *boot directory*.

Per a instal·lar GRUB simplement s'ha d'executar, per exemple, la instrucció `grub-install /dev/hda` com a *root*. Això assumeix que posarà les imatges en el directori */boot*, però si es vol canviar s'ha d'especificar amb el modificador `boot-directory`, i si bé `grub-install` és un *shell script* i la tasca és realitzada les ordres `grub-mkimage` i `grub-setup`, és molt recomanable utilitzar l'ordre `grub-install` per a evitar errors que poden resultar fatals i desconfigurar tot el sistema d'arrencada. Per a arrencar un sistema Linux és summament fàcil i està basat en tres passos:

1. Carregar el *kernel*: `GRUB> linux /vmlinuz root=/dev/sda1`

És possible utilitzar opcions, per exemple, acpi=off'

2. A continuació, executar l'ordre *initrd* per a carregar els mòduls:

```
grub> initrd /initrd
```

3. Finalment executar l'ordre `boot`

Si bé aquest és el procediment a seguir, és necessari escriure aquesta configuració per a no repetir-la en una arrencada automàtica. GRUB es configura mitjançant l'arxiu */boot/grub/grub.cfg* i no és necessari configurar-lo manualment.

L'ordre `grub-mkconfig` genera en la majoria dels casos el *grub.cfg* adequat i s'ha d'executar quan actualitzem una versió del sistema operatiu o volem canviar a un nou *kernel*. Aquesta ordre té algunes limitacions, ja que per exemple es poden agregar noves entrades al menú de *boot* editant */etc/grub.d/40_custom* o creant un arxiu específic a */boot/grub/custom.cfg*, però canviar l'ordre de les entrades, per exemple, pot significar canvis importants i complexos en els *scripts* que són a */etc/grub.d/*.

L'arxiu */etc/default/grub* controla l'operació de `grub-mkconfig` mitjançant seqüències de l'estil *KEY=valor* per exemple si tenim una entrada com:

```
menuentry 'Ejemplo GNU/Linux' --class gnu-linux {
   ...
}
```

Per indicar a l'ordre `grub-mkconfig` que serà l'entrada per defecte haurem de posar `GRUB_DEFAULT='Ejemplo GNU/Linux'`. Vegeu la documentació (`info grub`) per a totes les KEY possibles i la seva configuració, i la sintaxi per a opcions avançades, temes, aparença, etc.

Per a fer petits canvis es poden editar els *scripts* a */etc/grub.d* directament, en què per exemple */etc/grub.d/40_custom* és útil per a afegir entrades al menú de *boot* o adequar-les a les necessitats que es tinguin copiant les anteriors i modificant-les al final de l'arxiu (com a precaució sempre s'ha de fer una còpia del fitxer, per exemple, `cp 40_custom 40_custom.org`, i deixar les dues primeres entrades del menú sense tocar per a poder arrencar sempre amb la configuració correcta).

En cas que tinguem errors irreparables sempre podrem arrencar en mode *rescue* des d'una unitat de DVD, muntar el sistema d'arxius del disc dur i reemplaçar *40_custom* per *40_custom.org* per a deixar-ho tot com estava abans de canviar la configuració del *boot*.

4.4. Accés a particions i dispositius

Els sistemes tipus Unix tracten tots els dispositius de l'ordinador com si fossin arxius. Això permet total flexibilitat, ja que es poden aprofitar tots els mecanismes i les funcions que s'utilitzen amb fitxers per als dispositius. En el directori */dev* es tenen tots els dispositius reconeguts pel sistema. Si el sistema no reconeix adequadament un dispositiu o volem crear-ne un d'especial, es pot utilitzar l'ordre `mknod`, però s'ha d'utilitzar amb compte, ja que l'ús incorrecte podria danyar parts del sistema.

Per a les unitats d'emmagatzemament, el sistema proveeix instruccions com `mount` i `umount`, que situen (munten) o desmunten tot el sistema d'arxius d'un determinat dispositiu o unitat en un directori existent del sistema. La manera bàsica d'utilitzar l'ordre és `mount dispositiu directori`, en què el *dispositiu* pot ser qualsevol del canal IDE o SCSI (*/dev/hdXX*, */dev/sdXX*), la disquetera (/dev/FDX), memòries USB, etc., i *directori* és la ubicació en la qual muntarem l'estructura de fitxers del dispositiu, i si el sistema d'arxius no és el mateix amb el qual estem treballant, s'haurà d'agregar el paràmetre *-t filesystem*, en què *filesystem* és una sigla que es pot consultar a la pàgina del manual de `mount`. És recomanable que el directori en el qual muntem aquests dispositius sigui buit, ja que quan s'utilitza com a punt de muntatge no s'hi pot accedir. Per a desmuntar un d'aquests dispositius, podem utilitzar `umount directori`, en què el directori ha de ser el punt de muntatge utilitzat. Si muntem dispositius mòbils com CD o USB, és important no treure el dispositiu del suport, ja que abans hem d'avisar al sistema perquè actualitzi la memòria cau del sistema de

fitxers del dispositiu (o actualitzar les taules internes en el cas que el dispositiu solament sigui de lectura). Igualment, tampoc no podem desmuntar el dispositiu si algun usuari o aplicació està utilitzant algun dels arxius o directoris (en intentar-ho, el sistema donaria un missatge d'error).

Per defecte, per a poder muntar i desmuntar sistemes d'arxiu, es necessiten privilegis de superusuari, però es poden cedir aquests permisos a usuaris "administradors" afegint una entrada en el fitxer */etc/sudoers*, de manera que amb l'ordre `sudo mount ...` l'usuari habilitat podrà fer tasques permeses només al *root* (consulteu el manual de **mount** per a altres opcions en el treball amb dispositius i particions, com per exemple *rw, suid, dev, exec, auto, nouser, async*, que són les considerades per defecte).

Tot el muntatge de particions i dispositius es pot fer durant la inicialització del sistema operatiu per mitjà d'entrades en l'arxiu */etc/fstab*. Un exemple típic d'aquest arxiu és:

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>  <type> <options><dump> <pass>
  proc          /proc          proc         defaults             0      0
  /dev/hda1     /              ext3         errors=remount-ro    0      1
  /dev/hda5     none           swap         sw                   0      0
  /dev/hdc      /media/cdrom0  udf,iso9660  user,noauto          0      0
```

Cada línia és un `mount` i els paràmetres són (en ordre): dispositiu per muntar, on es munta, tipus de sistema d'arxius (*auto* perquè ho detecti automàticament), opcions de muntatge, especificació de si volem fer còpies de seguretat (*dump*), i l'últim camp serveix per a indicar l'ordre de muntatge (0, indica que l'ordre no és important). L'arrel del sistema d'arxius és el primer que s'ha de muntar, i per això en aquest camp hi hauria d'haver un 1.

Quant a les opcions, n'hi ha moltes (consulteu la pàgina del manual de `mount`) i depèn del sistema d'arxius. Les més comunes són:

- **rw/ro** que signifiquen que es muntarà el dispositiu amb permisos de lectura i escriptura o només lectura, respectivament.

- **noauto** inhibeix el muntatge automàtic durant l'arrencada.

- **user** permet que tots els usuaris puguin muntar aquest sistema d'arxius (si no s'indica solament ho podrà fer el *root*).

- **defaults** és equivalent a indicar les opcions predeterminades: *rw*, *suid*, *dev*, *exec*, *auto*, *nouser* i *async*, cadascuna de les quals es pot desactivar després

de *defaults* agregant *nosuid*, *nodev*, etc. Agregar l'opció *user* ho reactiva, ja que *defaults* inclou *nouser*.

Una entrada que sempre veurem en aquest fitxer i que ens pot sorprendre és el directori */proc*, que té un significat especial. Realment, el que hi ha en aquest directori no són fitxers, sinó el valor de moltes de les variables que utilitza el nucli del sistema, que permetrà ajustar paràmetres del nucli com si es tractessin d'un arxiu.

És interessant consultar l'ordre `autofs`, que permet muntar automàticament un sistema d'arxiu quan es detecta la inserció d'un dispositiu en el sistema.

En la primera entrada hi pot haver diferents formats per a indicar la unitat/partició a muntar a més de l'esmentada indicant el dispositiu de blocs (per ex., */dev/sda1*). Per a muntar un disc remot per NFS (*network file system*) haurem d'indicar <host>:<dir>; per ex. *nteum.uoc.edu:/home*, i també es pot fer pel *label* del disc (`e2label`) o per l'identificador físic del disc (UUID) (per a obtenir-los s'ha d'executar `blkid` o `lsblk -o NAME,UUID`). Un exemple d'UUID seria:

```
UUID=68d3272f-928a-473c-93f9-60decfb29530 / ext4 - 0 1
```

La qual cosa indica que el disc amb aquest UUID es muntarà com a *root file system*.

4.5. Instal·lació de paquets

La instal·lació de paquets addicionals o actualització dels paquets és particular de cada branca de distribucions (branca Debian-Ubuntu, branca Fedora-RH-Centos, etc.), però l'essència és similar i el tema es veurà en capítols següents. A tall d'exemple, es mostrarà la gestió i actualització de paquets en Debian que usa el gestor de paquets *apt* però també, a diferents nivells, `dpkg`, `aptitude`, `synaptic`, etc.

Per a configurar `apt` s'ha de modificar el fitxer */etc/apt/sources.list* i agregar els repositoris corresponents, per exemple (# significa 'comentari' i el repositori no està actiu):

```
#
deb http://httpredir.debian.org/debian jessie main contrib non-free
# deb-src http://httpredir.debian.org/debian jessie main

deb http://httpredir.debian.org/debian jessie-updates main contrib non-free
# deb-src http://httpredir.debian.org/debian jessie-updates main

deb http://security.debian.org/ jessie/updates main contrib non-free
# deb-src http://security.debian.org/ jessie/updates main
```

En aquest arxiu, *deb* significa 'paquets binaris', i *deb-src* 'fonts', l'URL on es troben, *jessie*, és la distribució amb la qual treballem (anar amb extrema cura de no barrejar paquets de diferents distribucions), i l'últim apartat és la secció de la distribució (*main*, *contrib*, *non-free*, que es poden combinar en la mateixa línia si coincideix el repositori).

Després s'ha d'executar `apt-get update` per a actualitzar la llista dels paquets, i amb `apt-get upgrade` podrem actualitzar tots els paquets actualitzats en *debian-security*. Amb `apt-get clean` podrem treure els paquets actualitzats/trets i recuperar l'espai en el disc.

Per a instal·lar un paquet s'ha de fer `apt-get install paquet` i per desinstal·lar-lo farem `apt-get remove [--purge] paquet`, en què l'opció *--purge* serveix per a eliminar tots els arxius de configuració a més del paquets en qüestió. Si no se sap el nom del paquet es pot fer una recerca amb l'ordre `apt-cache search keywords`, i la informació d'un paquet es pot obtenir amb `apt-cache show paquet`.

4.6. Configuració de dispositius

És important, abans d'intentar configurar algun dispositiu, buscar-ne informació, i fins i tot abans de comprar-ne un, assegurar-se que disposa de controladors compatibles amb la versió amb què pretenem treballar.

1) Teclat

La configuració del teclat s'ha simplificat de manera notable en gairebé totes les distribucions i, si bé hi ha diferències, el procediment i les ordres emprades són similars. En aquest cas, es descriurà el procediment en Debian (però és similar en altres distribucions, per exemple, en CentOS, tenint en compte que la ubicació dels arxius canvia). Debian ofereix una única forma de configuració que funciona tant per a la definició del teclat, a manera de consola, com per a la manera gràfica (si bé dins dels ajustos hi ha una utilitat gràfica que permet canviar i configurar el teclat). El procés de configuració es realitza a través del paquet de *keyboard-configuration*, amb la qual cosa només s'ha d'executar `dpkg-reconfigure keyboard-configuration` i respondre les preguntes sobre els diferents aspectes del teclat: tipus de teclat, mapa, qüestions en relació amb les tecles Alt/AltGr, caràcters composts (per als teclats en particular vegeu */usr/share/X11/locale/compose.dir*), etc.

En alguns sistemes, pot passar que s'hagin creat arxius i directoris amb un altre conjunt de caràcters i que, en visualitzar-los, no acceptin els caràcters configurats (*locale*) en el present sistema. Per això és força útil la instrucció `convmv`, que permet canviar d'un joc de caràcters a un altre els arxius i directoris que es visualitzen incorrectament. Així doncs, per exemple, si tenim un entorn en UTF-8 i en el directori */tmp* hi ha arxius codificats en ISO-8859-15, es veurien com "Ic?nes ?l?ments graphiques Textes". Per solucionar-ho, executem

`convmv -r --notest -f iso- 8859-15 -t utf-8 /tmp/`, i quedarà com "Éléments graphiques Icônes Textes". Per al contingut dels arxius es pot utilitzar l'ordre `recode`.

També es pot fer la modificació del teclat, de manera manual, editant l'arxiu */etc/default/keyboard* (vegeu `man keyboard`), per a les opcions i valors del teclat. Es pot utilitzar l'ordre `loadkeys` per carregar un teclat en particular o reinicialitzar-lo en els valors per defecte; l'ordre `dumpkeys` ens permet veure les assignacions de tecles i el codi del teclat en ús `showkeys` ens permet visualitzar interactivament el codi generat per una tecla o per la seva combinació. Un aspecte addicional relacionat amb el teclat són els accents i dièresis que es poden configurar a partir del fitxer */etc/inputrc*.[2] A continuació, es mostra el contingut d'*/ etc/default/keyboard* per a un teclat en castellà (ES).

(2)Consulteu totes les directives possibles d'aquest fitxer especificades en el manual de readline.

```
# KEYBOARD CONFIGURATION FILE
# Consult the keyboard (5) manual page.
XKBMODEL="pc105"
XKBLAYOUT="es"
XKBVARIANT=""
XKBOPTIONS="terminate:ctrl_alt_bksp"
BACKSPACE="guess"
```

Finalment, una altra configuració important (indirectament relacionada amb el teclat) és la de *locales*, on es pot configurar la zona geogràfica en què ens trobem, per poder utilitzar tecles especials del teclat, veure les dates en el format correcte, etc. Aquesta configuració és utilitzada per moltes de les llibreries del sistema, de manera que, en moltes ordres i aplicacions pròpies, utilitzarà la seva configuració per adaptar algunes funcions de l'entorn local. La seva configuració es troba a */etc/locale.gen* i es poden utilitzar les ordres `locale` i `locale-gen` per visualitzar o actualitzar respectivament la configuració.

2) Targeta de xarxa (Ethernet)

Per a configurar una nova targeta de xarxa (de tipus Ethernet), en primer lloc, cal afegir el mòdul necessari perquè es reconegui adequadament. Si bé no és necessari per a algunes targetes, ens hem d'assegurar (abans de comprar l'ordinador o la targeta) que tenim el controlador o mòdul necessari.

Amb l'ordre `discover` podem saber quin tipus de maquinari tenim i trobar el mòdul corresponent.

Una configuració pas a pas seria:

- Executar l'ordre `ping google.com`
  - Si el resultat és similar a:

```
PING google.com (91.213.30.166) 56(84) bytes of data.
```

```
64 bytes from 91.213.30.166: icmp_req=1 ttl=58 time=4.85 ms
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 10037ms.
```

Indica que la xarxa està configurada i que no s'ha de fer cap acció.

– Si el resultat indica `100% packet loss`, la xarxa (o el servidor de noms DNS) no està configurada i haurem de configurar-la. Per a això, si estem en mode gràfic haurem d'obrir un terminal (`xterm` o `gnome-terminal` per exemple).

- Verificar que tenim el dispositiu de xarxa executant com a l'usuari *root* l'ordre `ifconfig`.
  – El resultat serà una cosa així:

```
eth0 Link encap:Ethernet HWaddr 08:00:27:b4:51:d9
inet addr: Bcast: Mask:
...
RX bytes: TX bytes:
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
...
RX bytes:654302082 (623.9 MiB) TX bytes:654302082 (623.9 MiB)
```

En què *loopback* (lo) és una interfície especial de connexió local (per a verificar el servei del propi ordinador) a la qual s'assigna la IP 127.0.0.1 i el nom *localhost* (vegeu */etc/hosts*).
Si no apareix l'estat d'una interfície Ethernet (eth0) o similar, el dispositiu de xarxa no està instal·lat en el nucli (o bé no es va carregar o no està compilat dins del nucli).

– La instrucció `lspci` permetrà mostrar si existeix el dispositiu connectat al bus PCI (la majoria dels ordinadors actuals). Per exemple, executar: `lspci | grep Ethernet`. I el resultat serà una cosa com:

```
 00:03.0 Ethernet controller: Intel Corporation 82540EM
        Gigabit Ethernet Controller (rev 02)
```

– Per a carregar el mòdul podem utilitzar `modprobe e1000`, que indica que s'ha de carregar el mòdul per a una targeta Intel PRO/1000 gigabit Ethernet. Si no tenim errors significa que s'ha reconegut l'HW i podem passar a la configuració (es pot fer un `lsmod` per a veure els mòduls carregats). Si hi ha errors haurem de buscar el mòdul adequat, per exemple 3c59x per a 3Com-3c590/3c900, 8139too per a RealTek 81xx, b44 per a Broadcom 4xx, eepro100 per a EtherExpressPro/100, tg3 per a Broadcom Tigon3, e1000e per a Intel PRO/1000 PCI-Express, etc.

  - Amb el mòdul carregat fent `systemctl restart networking` (també funciona per compatibilitat */etc/init.d/networking restart* però es troba obsolet). Per veure si el servei funciona correctament haurem de fer `systemctl status networking`.

- Per a configurar manualment la targeta de xarxa haurem de modificar l'arxiu */etc/network/interfaces* i la configuració serà similar a:

```
# Interfaz de loopback auto lo
iface lo inet loopback

# NIC
auto eth0

iface eth0 inet static
   address 192.168.0.10
   netmask 255.255.255.0
   network 192.168.0.0
# opcional broadcast 192.168.0.255
   gateway 192.168.0.1
```

  En què la interfície *lo* no s'ha de tocar i eth0 serà el nostre dispositiu a configurar, amb *static* indiquem que la IP serà la que està a *address*, i amb *gateway* indiquem el nostre punt de connexió amb la xarxa (podria ser l'encaminador o una altra màquina que ens permeti la connexió d'Internet). Si tenim un servei DHCP de configuració automàtica, aquí **només** hauria de posar *iface eth0 inet dhcp*. A netmask indicarem que aquesta màquina pertany a una classe C (es veurà més endavant) i per això hem de posar 255.255.255.0. Compte en indicar aquests valors, ja que si posem la IP d'una altra màquina o una màscara equivocada podem causar problemes a altres màquines o no tenir connexió de xarxa respectivament. La IP indicada pertany a una xarxa interna i no a una xarxa pública, per la qual cosa si volem tenir connexió a Internet, per exemple, la passarel·la (*gateway*) serà fonamental per a redirigir els paquets cap a la xarxa externa.

- Després haurem de revisar la configuració del DNS en */etc/resolv.conf*, que indicarà els nostres servidors (fixem-nos que en l'Ubuntu la configuració del DNS és lleugerament diferent, ja que té instal·lat el paquet *resolvconf* per defecte). En aquest arxiu haurem de tenir una cosa com:

```
domain xxx.xxx
search xxx.xxxx
nameserver 8.8.8.8
nameserver 8.8.4.4
```

En aquest cas, les línies importants són la tercera i la quarta, i per a aquest exemple hem posat els DNS públics de Google.

- Si hem configurat amb DHCP i amb el `ifconfig` veiem que no té IP assignada podem instal·lar `apt-get install dhcp3-client`.

Ordres addicionals: `ifdown` i `ifup` (per a apagar o encendre la interfície), `ip` (per a configurar en línia d'ordres tots els paràmetres de la xarxa) i `route` (que ens mostra la taula d'encaminament).

3) Targeta sense fil (Wi-Fi)

La xarxa sense fil (*wireless* LAN/WLAN/Wi-Fi) es basa en l'estàndard IEEE 802.11, i les definicions més habituals són *b*, *a*, *g*, que indiquen velocitat, freqüència i altres paràmetres representatius del protocol (per exemple *b-g* utilitzen la banda de 2,4 GHz, mentre que la *a* la de 5 GHz). Com que les freqüències es regulen a cada país, el programari de les targetes de Wi-Fi no es pot distribuir com a *open source*, per la qual cosa la part del dispositiu que defineix les freqüències es distribueix com a microprogramari (*firmware*) i s'ha de descarregar posteriorment. Per a la configuració haurem de saber quin és el joc de xips (*chipset*) de la targeta i on en descarregarem el microprogramari, i després bàsicament haurem repetir els passos fets per a la targeta Ethernet (amb algunes particularitats). Per passos hauríem de fer:

- Executar l'ordre `lspci` per a determinar si s'està reconeixent l'HW (buscar paraules com *Wifi*, *WLAN*, *Wireless* o similars).

- Buscar el mòdul oportú (p. ex., amb `apt-cache search firmware`) i instal·lar-lo, per exemple en el cas dels més comuns ipw2200 per a Intel PRO/Wireless 2xxx, atheros per a Atheros (3Com, Belkin, D-Link i Linksys), etc. Per a carregar el microprogramari haurem d'instal·lar el paquet corresponent (o baixar-lo del fabricant i compilar-lo si no hi és). El Debian té molts microprogramaris com paquets en el repositori non-free, per la qual cosa s'haurà d'incloure el repositori i després instal·lar el paquet. Per exemple, per a un dispositiu ipw2200 sobre Jessie haurem de fer:
  - Agregar el repositori a l'arxiu */etc/apt/source.list* amb la següent línia: *deb http://httpredir.debian.org/debian jessie main contrib non-free*.

  - Executar:

```
apt-get update
apt-get install firmware-ipw2x00
```

  - Acceptar la llicència i instal·lar el *driver* amb `modprobe -r ipw2200`; `modprobe ipw2200`

– Mirar a `dmesg` si hi ha errors del mòdul carregat.

- Fer la instal·lació del paquet per a gestionar les *wireless* (si no està instal·lat) `apt-get install wireless-tools` i executar `iwconfig` per a mirar els dispositius sense fils. El resultat serà una cosa com:

```
lo        no wireless extensions.
eth0      no wireless extensions.
wnl0 unassociated  ESSID:off/any
            Mode:Managed  Channel=0  Access Point: Not-Associated
            Bit Rate=54 Mb/s   Tx-Power=10 dBm   Sensitivity=5/0
```

  On el dispositiu Wi-Fi serà wnl0

- Executar l'ordre `iwlist wnl0 scanning` per a veure les xarxes disponibles que identificarem per l'ESSID, per exemple ESSID:"uoc.wifi". És important la informació del paràmetre *Ecryption*, ja que ens dirà si la xarxa és oberta (pública) o encriptada (que podrà ser WEP –habitual però insegura–, WPA/WPA2 –no la suporten tots els dispositius però és més segura).

- Per a connectar-se a un **xarxa oberta** solament hem d'agregar el dispositiu a */etc/network/interfaces* i dues línies com:

```
auto wnl0
iface wnl0 inet dhcp
```

  Després configurar el */etc/resolv.conf* i reiniciar la xarxa (es pot instal·lar el paquet *resolvconf* també per a la configuració automàtica del DNS). Si s'utilitza static en lloc de *dhcp* s'han d'indicar els valors similars a la configuració d'Ethernet.

- Si la xarxa té encriptació WEP, el */etc/network/interfaces* haurà d'incloure a sota *iface*:

```
wireless_essid uoc-wifi
wireless_channel 6
wireless_mode managed
wireless_keymode open
wireless_key1 mykeyHEX
wireless_key2 s:mykeyASCII
wireless_defaultkey 1
```

  Continuar amb els passos similars al punt anterior per al *resolvconf* i es reinicia la xarxa.

- Si la xarxa té encriptació WPA s'ha d'instal·lar `apt-get install wpasupplicant`

i modificar les configuracions corresponents en */etc/network/interfaces* agregant:

```
wpa-driver wext
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

El *driver* pot ser diferent (*nl80211* per al Linux 802.11, *wext* per al Linux *wireless extensions* o *wired* per a *wired Ethernet driver*) essent *wext* el genèric (usat habitualment). Teniu més informació a */usr/share/doc/wpasupplicant*. L'arxiu */etc/wpa_supplicant/wpa_supplicant.conf* haurà de contenir alguna cosa com:

```
ctrl_interface=/var/run/wpa_supplicant
network={
        ssid="uoc-wifi"
        scan_ssid=1
        proto=WPA
        key_mgmt=WPA-PSK
        psk=mykey
}
```

En què s'ha de modificar l'SSID i psk per a posar la clau d'encriptació. Perquè la clau no sigui visible es pot usar `wpa_passphase uoc-wifi mykey` i el resultat de l'execució s'haurà de tallar i enganxar com a valor en psk. S'ha de continuar amb els passos explicats en el cas de la xarxa oberta per al `resolvconf` i reinici de la xarxa.

Una alternativa per a configurar la xarxa (mitjançant una interfície gràfica) és utilitzar un paquet anomenat *NetworkManager* (NM), que inclou moltes distribucions. Veurem més detalls sobre la seva configuració en l'apartat de xarxa, però per a una primera configuració i, per exemple, per a ordinadors portàtils pot ser adequada (sempre que tinguem interfície gràfica).

Si el paquet no està instal·lat (verifiqueu la icona de xarxes en la barra superior a la dreta a Gnome3, per exemple)

`apt-get install network-manager wpasupplicant resolvconf`

L'NM intentarà que la xarxa funcioni buscant tota la informació necessària i configurant-ho tot perquè la xarxa simplement funcioni, per la qual cosa de vegades no s'ajusta a necessitats especials o configuracions específiques i no és adequat per a algunes configuracions.

Primer, l'NM no gestionarà interfícies definides a */etc/network/interfaces* i que a */etc/NetworkManager/NetworkManager.conf* contingui:

```
[main]
plugins= ifupdown,keyfile
[ifupdown]
managed=false
```

S'haurà de canviar managed=true si es vol que NM gestioni les interfícies definides a */ec/network/interfaces*. Podeu trobar més informació a https://wiki.debian.org/NetworkManager.

4) Targeta de so

Igual com en els casos anteriors, la targeta de so també necessita el mòdul del nucli per a poder funcionar correctament. Amb l'aplicació `discover`, podem descobrir quin mòdul és el que es correspon amb la nostra targeta o amb l'ordre `lspci | grep audio`. Per a instal·lar el mòdul, procedirem com amb la targeta de xarxa amb `insmod` o `modprobe`, per a configurar-lo permanentment a */etc/modules*. Si bé amb el mòdul corresponent ja podrem utilitzar la targeta de so adequadament, generalment també se sol instal·lar la infraestructura de so ALSA (*Advanced Linux Sound Architecture*) inclosa per defecte en la majoria de les distribucions.

Si el paquet ALSA no està instal·lat, es pot fer `apt-get install alsa-base` i iniciar el servei amb `/etc/init.d/alsa-utils start`, la qual cosa ens mostrarà si hi ha errors o no. Per a verificar que els mòduls estan carregats podem executar `lsmod` i mirar si tenim dispositius com *snd_xxx, snd_ac97_codec* i *soundcore*. Perquè un usuari pugui utilitzar els dispositius de so, haurà de formar part del grup "àudio", que es pot verificar fent `grep audio /etc/group`, que ens donarà alguna cosa com *audio:x:29:pulse,adminp*, i si l'usuari que volem no hi és podem executar `addgroup usuari audio` (s'haurà de reiniciar la sessió X de l'usuari agregat per a actualitzar els canvis i a Gnome s'hauran d'instal·lar els paquets *esound* i *gnome-audio* per a gestionar la interfície als dispositius d'àudio).

5) Impressora

En GNU/Linux la configuració d'impressores es pot fer de diferents maneres, si bé `lpd` (*line printer daemon*) va ser un dels primers programes de gestió d'impressió, actualment n'hi ha d'altres més fàcils de configurar i gestionar. Els més bàsics són:

- `lpd`: un dels primers dimonis d'impressió dels sistemes tipus Unix. La configuració s'ha de fer manualment.

- `lpr`: la versió de BSD de l'*lpd*. És molt recomanable utilitzar algun tipus de filtre automàtic com *magicfilter* o *apsfilter* per a configurar-lo.

- `LPRng`: aplicacions basades en *lpr*, amb l'avantatge que incorporen una eina de configuració denominada `lprngtool`, que permet configurar-la de manera gràfica i senzilla.

- `gnulpr`: la versió de GNU del sistema d'impressió `lpr`. També incorpora eines gràfiques de configuració, gestió dels serveis, etc.

- **CUPS (recomanat):** de *Common UNIX Printing System*, aquest conjunt d'aplicacions és compatible amb les ordres d'`lpr` i també serveix per a xarxes Windows.

Normalment, totes aquestes ordres tenen els seus mètodes de configuració propis, però utilitzen el fitxer */etc/printcap* per a desar-la i utilitzen un dimoni (procés que s'executa indefinidament) perquè el sistema d'impressió sigui operatiu i fins i tot es pugui imprimir des d'altres ordinadors.

Com a exemple mostrarem la instal·lació de CUPS a Debian i la configuració d'una impressora PDF (és a dir que imprimirem sobre un document).

Com a *root* cal executar `apt-get install cups` (i `cups-bsd` si es volen utilitzar ordres com `lpr`, `lpq`, `lpc`, `lprm` però no seran necessaris), a més es poden instal·lar els paquets PPD (*PostScript Printer Description* i la seva extensió per a impressores no PostScript) executant

```
apt-get install openprinting-ppds foomatic-filters-ppds
```

i també `apt-get install system-config-printer` per a configurar la impressora en forma gràfica des de Gnome. Es pot visitar la pàgina web de la Fundació Linux (http://www.linuxfoundation.org/collaborate/workgroups/openprinting/database/databaseintro) per a buscar el fabricant/model de la nostra impressora i buscar el dispositiu que es recomana.

A més a més, de l'opció de configurar la impressora des de Gnome, el més habitual és configurar-la des de la interfície web de CUPS, en concret des del navegador http://localhost:631/, la qual ens permetrà definir, configurar i administrar la impressora.

Per configurar una impressora PDF instal·lem `apt-get install cups-pdf`

A *System -> Administration -> Printing* es veurà una llista d'impressores i la icona d'Add *Printer* on podrem seleccionar:

Local Printer i *Detected printer --> PDF printer.*
Manufacturer --> Generic,
Model --> postscript color printer rev4,
Driver --> Standard
Apply

També es podrà fer des de la interfície gràfica de CPUS a http://localhost:631/.
Podeu trobar més informació a https://wiki.debian.org/SystemPrinting

6) Rellotge

Els ordinadors disposen d'un rellotge CMOS també anomenat *rellotge maquinari* que, depenent de la seva implementació, pot acumular desfasaments amb el pas del temps (per exemple, 10 segons/dia) i depèn de factors externs, per exemple, la càrrega de la bateria de la CMOS. El rellotge del sistema permet fer una copia dels valors d'aquest rellotge maquinari, per la qual cosa pot traslladar valors inexactes. Per a modificar el rellotge maquinari s'ha d'executar l'ordre `hwclock --set --date="mm/dd/yyyy hh:mm:ss"` i reemplaçar el que calgui amb els valors adequats. Aquesta ordre no modifica l'hora del sistema, per la qual cosa haurem de fer `hwclock -s` (o `-hctosys`) que significa *Hardware Clock -> System Time* (verifiqueu-ho amb l'ordre `date`). Si modifiquem la data/hora amb l'ordre `date` solament modificarem els valors del sistema, que es perdran quan apaguem la màquina. Una opció per a tenir el rellotge sincronitzat és utilitzar un servei anomenat NPT (*network time protocol*), que donen rellotges atòmics (molt precisos) distribuïts pel món.

Primer haurem de configurar adequadament la zona/regió on som amb l'ordre `dpkg-reconfigure tzdata` i després instal·lar `apt-get install ntp` (o `ntpdate` si solament volem instal·lar el client). Si instal·lem el paquet `ntp` la configuració dels servidors es troba a */etc/npt.conf*, on ja inclou una sèrie de servidors per defecte. Per a consultar la sincronització i l'estat podem executar `ntpq -p`, que ens donarà els valors d'ajust i la progressió.

Amb l'ordre `ntpdate` (instal·lada pel segon paquet) es permet sincronitzar ràpidament l'hora fent, per exemple, `ntpdate 0.debian.pool.ntp.org`.

Per a sincronitzar el rellotge de manera automàtica des dels servidors distribuïts pel món és recomanable la primera opció (ntp). Com a ordres addicionals a l'`ntpq` podem utilitzar `ntptrace`. Recordem que si es modifica */etc/ntp.conf*, llavors és necessari reiniciar el servidor amb `systemctl restart ntp`. I, finalment, per a copiar l'hora del sistema al rellotge maquinari podem fer servir l'ordre `/etc/init.d/hwclock.sh restart`. Si volem que el nostre `ntpd` funcioni com a servidor, s'ha de modificar la línia de *restrict* traient el *noquery* i reiniciar el *daemon* amb `systemctl restart ntp`.

4.7. Elements addicionals d'una instal·lació

- **Rotació d'arxius de registre.** Els arxius d'errors i funcionament/registre de les aplicacions/serveis poden créixer molt ràpidament, i és necessari arxivar-los d'acord amb una política predeterminada. L'esquema més comú és un arxivament rotatiu: l'arxiu de registre s'emmagatzema regularment i només es mantenen els últims *X* arxius. `logrotate`, el programa utilitzat en la majoria de les distribucions que du a terme aquestes rotacions i que té les polítiques generals predefinides a */etc/logrotate.conf* i específicament per a les aplicacions a */etc/logrotate.d/* que podran ser adaptades i/o modificades de la política per defecte que inclou la distribució. L'ordre té

un conjunt d'opcions (vegeu el full del manual per a totes les seves possibilitats) per a modificar la quantitat d'arxius mantinguts en la rotació o moure els arxius de registres a un directori específic dedicat a arxivar-los en lloc d'eliminar-los, enviar-los per correu electrònic per a arxivar-los a un altre banda, etc. Aquest programa s'executa diàriament per mitjà d'un servei d'execució programada anomenat `cron`, que permet fer tasques de manera automàtica i desatesa per part de l'administrador.

- **Compartició de permisos d'administració.** En determinades ocasions, amb els servidors compartits els administradors treballen en la mateixa xarxa, i compartir contrasenyes de *root* no és gaire elegant i obre la porta a l'abús per l'anonimat que es genera. Per a evitar aquestes situacions hi ha l'ordre `sudo` (única manera de treballar com a *root* en algunes distribucions com Ubuntu), que permet els usuaris executar certes ordres amb permisos especials, és a dir, en el seu ús més comú permet a un usuari normal executar qualsevol ordre com a *root*. Per a fer-ho, l'usuari simplement executa `sudo programa` i proveeix la seva contrasenya personal com a autenticació. Per a delegar els permisos, l'administrador ha d'utilitzar el programa `visudo`, que li permetrà modificar l'arxiu de configuració */etc/sudoers* (consulteu la pàgina del manual de `sudoers` per a més detalls). S'ha d'anar amb compte en la seva configuració, ja que agregar una línia amb *usuari ALL=(ALL) ALL* permetrà a aquest usuari executar qualsevol programa com a *root* i generalment s'utilitza perquè els usuaris puguin executar ordres específiques.

- **Qüestions relacionades amb l'arrencada (*boot*) del SO**. Si bé es veurà amb detall en capítols posteriors, farem una breu descripció del que ocorre durant l'arrencada de l'ordinador. Quan s'inicia (o després d'un *reset*) el BIOS pren el control de l'ordinador, detecta els discos, carrega el registre mestre d'arrencada (MBR) i executa el gestor d'arrencada. Aquest pren el control, busca el nucli en el disc, el carrega i l'executa, i aquest al seu torn s'inicialitza i cerca/munta la partició que conté el sistema d'arxius arrel. A partir d'aquest moment executa el primer programa anomenat `init`. Generalment, aquesta "partició arrel" i l'`init` estan situats en un arxiu virtual del sistema que només existeix en RAM, que es diu *initramfs* (*initialization RAM file system*) i que després reemplaçarà pel sistema d'arxiu arrel autèntic. Aquest pas previ es fa perquè pot ser necessari carregar abans controladors de disc o altres dispositius (LVM, RAID, xarxa, etc.) perquè el sistema pugui arrencar i accedir allà on es troba realment la informació. Després d'això, el sistema passa el control a l'"init real", que es pot dur a terme de diferents maneres en funció del mètode de què disposi la distribució (tant en Debian –en la última versió– com en altres distribucions l'*init* ha estat substituit per `systemd` com procés d'arrencada).

L'*initrd* passa el control a `systemd` i aquest carrega els controladors, munta el sistema d'arxius i inicialitza tots els serveis configurats, i activa totes les unitats que tenen dependències (*default.target*) i vincles (*multi-user.target* o *graphi-*

cal-user.target), segons estiguin configurats. Es pot veure l'arbre de dependències (i quins estan o no actius) de *default.target* fent l'execució de la instrucció `systemctl list-dependencies default.target`.

`systemd` treballa amb *units* i *targets*. Les entitats anomenades *units* encapsulen diferents objectes necessaris per a l'arrencada i manteniment del sistema, i en la seva majoria estan configurades en arxius de configuració propis[3]. Es pot analitzar algun exemple mirant els arxius de */etc/systemd/system*.

[3]També es poden crear automàticament, a partir d'una altra configuració, o responent a un estat del sistema, o a partir d'una altra unitat.

Els diversos nivells d'execució (antics *levels* definits en */etc/inittab*, en versions anteriors amb *SysV* i, ara, obsolets amb *systemd*) estan definits en unitats que permeten agrupar els nivells de dependència (per exemple, apagar, iniciar, reiniciar, monousuari, multiusuari, multiusuari de manera gràfica). Cada unitat iniciarà els serveis associats a aquest nivell d'execució. Aquestes agrupacions es defineixen com *.target*. Existeixen diferents *target* —si bé alguns inclouen serveis similars— l'objectiu dels quals és iniciar els processos i serveis necessaris per a aquest "estat" i que es troben definits a les *units*. Un exemple d'aquesta *unit* es pot veure a l'arxiu */etc/systemd/system/sshd.service*.

```
[Unit]
Description=OpenBSD Secure Shell server
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure

[Install]
WantedBy=multi-user.target
Alias=sshd.service
```

Un *target* pot heretar els processos de l'anterior i sumar-li'n els propis. Per exemple, si llistem en Debian l'arbre de dependències de *default.target* (utilitzant l'ordre abans indicada), veurem que *multi-user.target* va heretar-lo de *default.target* i, al seu torn, *basic.target* d'aquest i, així, per a la resta dels *target*. Es poden conèixer tots els *target* disponibles executant

`systemctl list-units --type=target –all`.

`systemctl list-units --type=target` mostrarà els *target* en ús. Emprarem `systemctl list-unit-files` per conèixer totes les unitats instal·lades i el seu estat.

Per a aquells que coneixen els *levels* de *SysV*, es pot fer una equivalència aproximada. Por exemple, *level 3* = *multi-user.target* (multiusuari sense entorn gràfic) o *level 5* = *graphical.target* (que inclou tots els de 3 més l'entorn gràfic). Per altra banda, per canviar a un altre *target* es pot executar, per exemple, la instrucció `systemctl isolate multi.user.target` i, per visualitzar les dependències, es pot executar `systemctl xou -p "Wants" graphical.target`, que ens indicarà de què depèn aquest *target* (el resultat serà *Wants=display-manager.service systemd-update-utmp-runlevel.service nagios-nrpe-s*).

Les que segueixen són altres ordres útils, vinculades a la càrrega i gestió dels processos:

- `systemd-analyze`: mostra el temps d'inicialització i càrrega (per a majors detalls, afegirem `blame` com a paràmetre `plot > plot.svg` per veure un gràfic sobre els temps emprats).

- `systemd-cgls`: mostra tots els processos i l'arbre d'execució per identificar qui ho va engegar.

- `systemctl list-units`: llista les unitats instal·lades.

- `systemctl list-unit-files`: llista les unitats disponibles.

- `systemctl list-units --type=service`: llista les unitats per tipus d'unitat.

- `systemctl -failed`: les que presenten alguna fallada.

- `systemctl start/stop/restart sshd.service`: engega/para/reinicia el servei.

- `systemctl is-enabled sshd.service`: mostra si el servei està habilitat o no.

- `systemctl enable/disable sshd.service`: habilita/deshabilita el servei.

- `systemctl halt/poweroff/reboot/suspend/hibernate`: fa una apagada d'emergència, apagada, reinici, suspensió o hibernació del sistema.

- `journalctl`: aquest és el sistema de *log* de *systemd* i, si executem aquesta ordre, mostrarà el registre guardat.

- `journalctl -b`: mostra tan sols les línies de *boot* de l'execució en curs.

- `journalctl -b -p err`: mostra només aquells que presenten error.

5. L'entorn gràfic

Els sistemes *nix utilitzen una arquitectura d'entorn gràfic anomenada *X-Window*, dissenyada en la dècada dels vuitanta, que és independent de la plataforma i serveix per a qualsevol tipus de *nix. X.Org és una implementació de codi obert del sistema X-Window (que sorgeix com a bifurcació de projecte XFree86) i funciona en mode client/servidor, de manera que no podem connectar gràficament un servidor remot o executar sobre un ordinador local aplicacions gràfiques d'un ordinador remot. Avui dia es treballa juntament amb una infraestructura de DRI (*Direct Rendering Infrastructure*), que permet aprofitar els xips de processament de les targetes per a estalviar treball de visualització al client X-Window.

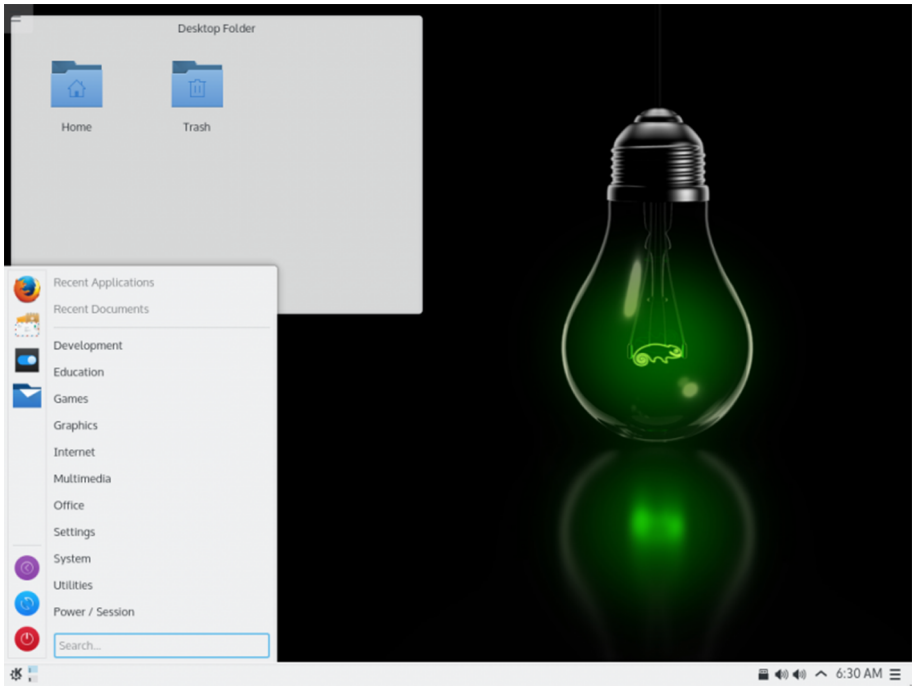
El sistema X-Window (basat en una biblioteca anomenada *xlibs*) proporciona els mètodes per a crear les interfícies gràfiques d'usuaris (GUI), però no n'implementa cap sinó que aquestes són proporcionades per jocs d'eines (*toolkits*), que són biblioteques generalment implementades amb *xlibs* i que proporcionen un GUI particular. El gestor de finestres és un servidor especial de X-Window, que s'encarrega de gestionar totes les finestres, els escriptoris, les pantalles virtuals, etc. Òbviament, totes les aplicacions poden funcionar amb qualsevol gestor de finestres, ja que aquest només s'encarrega de gestionar la finestra on està ubicat el programa, i n'hi ha desenes (FVWM, MWM, AfterStep, Enlightenment, IceWM, Sawfish, Blackbox, etc. Podeu consultar l'adreça següent per a obtenir-ne més informació: http://xwinman.org), de manera que l'usuari pot elegir el que més li agradi.

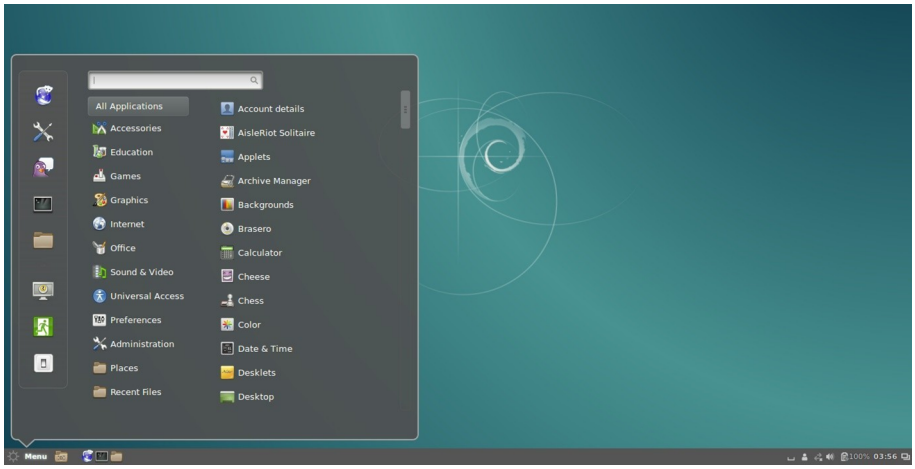
A més dels *Windows Managers*, els desenvolupaments més recents han creat una nova generació d'entorns d'escriptori que tenen com a objectiu proporcionar una interfície més completa per al sistema operatiu, i una gamma d'utilitats pròpies i aplicacions integrades. Aquesta comoditat i facilitat d'ús els fan particularment atractius per a instal·lacions noves i com una manera de tenir-ho tot integrat i funcional des del primer moment.

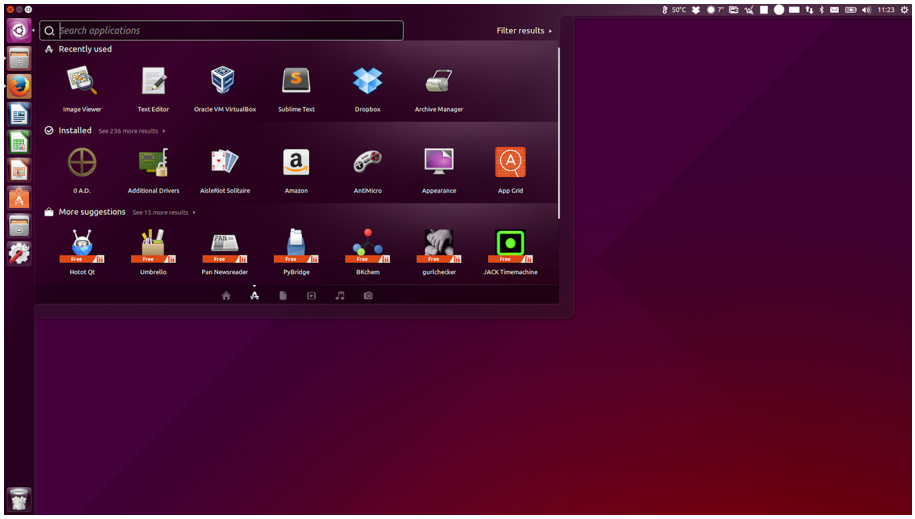
Els més populars en l'actualitat són KDE (*the K desktop environment*) i GNOME (*GNU*, *object model environment*), però hi ha una sèrie d'entorns (promocionats per diferents distribucions) que estan irrompent amb força com per exemple Cinnamon, LXDE, MATE, Unity, Xfce entre d'altres.

Tots ells proporcionen un joc d'eines particular, un entorn d'escriptori amb moltes funcionalitats i configuracions diferents i una llista d'aplicacions integrades, que com més va creix més, i són els més usats en totes les distribucions

GNU/Linux. A les figures següents podem veure, en primer lloc, l'aspecte de KDE Plasma5 (utilitzat, per exemple, a OpenSuse); després, GNOME3 i CINNAMON (utilitzat a Debian 7), i, finalment, Unity (utilitzat a Ubuntu):







Actualment, la majoria de les targetes gràfiques del mercat estan suportades i molts fabricants ja donen suport per a GNU/Linux i proporcionen els seus controladors propis (*drivers*).

Per a instal·lar X.Org al nostre ordinador, és necessari baixar els paquets que contenen les eines bàsiques i el programari per al client i el servidor. Generalment, aquests paquets se solen denominar `xorg`, `xserver-xorg`, etc., i porten implícites diverses dependències de fonts i algunes utilitats bàsiques per al maneig d'X-Window (es pot utilitzar en Debian `apt-cache search Xorg` per a veure els servidors disponibles i `apt-get install <Xorg.server>` per a instal·lar-ne un en particular).

Després s'haurà d'instal·lar el paquet *desktop* (recomanable) per a instal·lar totes les opcions i aplicacions sobre el servidor X-Windows seleccionat (si s'opta per instal·lar el *desktop-manager*, aquest també instal·larà el servidor que consideri més adequat per a la nostra targeta).

És necessari indicar que en alguns dels *desktop-managers* actuals (per exemple, KDE4, Gnome3 o Unity) es necessita suport d'acceleració maquinari, per la qual cosa en targetes de vídeo antigues funcionaran sense totes les seves possi-

bilitats en un mode anomenat *de compatibilitat* (sobre màquines virtualitzades s'hauran d'instal·lar els paquets addicionals –per exemple, *GuestAdditions* per a VirtualBox–, que compilaran els *drivers* i extensions DRI per a adequar-los al maquinari subjacent o també és possible instal·lar els paquets vbox* que ja venen compilats en algunes distribucions –per exemple a Debian dins el repositori *non-free*–).

Per exemple, a partir de la distribució Debian es fa una configuració automàtica de Xorg i generalment no s'ha de tocar gran cosa en aspectes de configuració. Els passos bàsics per a instal·lar en entorn gràfic seran executar com a *root* `apt-get install xorg`, la qual cosa instal·la un conjunt de paquets bàsics i el servidor més adequat a la nostra targeta. Després instal·lem el servidor d'escriptori i el *display manager* (GNOME en el nostre cas, però és similar per a KDE i fins i tot es poden instal·lar tots dos, tenint en compte que cadascun ocupa més de 600 MB) amb `apt-get install gnome gdm3` i arrenquem el servidor amb `systemctl start gdm3` (o reiniciar la màquina).

Si hi ha errors es pot recórrer a l'arxiu */var/log/Xorg.0.log* per a analitzar on són les fonts de l'error i corregir-les prestant essencialment atenció a les línies que comencen per (EE) que indiquen un error.

Si no es disposa d'espai de disc suficient, és possible instal·lar altres entorns que ocupen menys espai, com per exemple `lxde` *–lightweight X11 desktop environment–* o `xfce4` *–xfce lightweight desktop environment–*, que no superen els 70 MB.

Si trobem errors de configuració mirant */var/log/Xorg.0.log* podem ajustar la configuració mitjançant l'arxiu */etc/X11/xorg.conf*. Com a partir de la versió Debian 6 Squeeze aquest arxiu no existeix per defecte, n'haurem de crear un. Per a fer-ho primer cal parar el servidor (si està funcionant) executant l'ordre `systemctl stop gdm3` i després `Xorg -configure`, la qual cosa crearà l'arxiu *xorg.conf.new* en el directori local basant-se en la detecció automàtica del maquinari; després hem de moure aquest arxiu al directori corresponent, per exemple, `mv xorg.conf.new /etc/X11/xorg.conf`.

Aquest arxiu (*/etc/X11/xorg.conf*) conté la configuració d'X.org i està dividit en seccions (*Files, Module, InputDevice, Device, Monitor, Screen, DRI, ServerLayout*). Cada secció està marcada per Section i EndSection amb ordres específiques per a cadascuna. Per exemple, la secció per a configurar la targeta gràfica és:

```
Section "Device"
    Identifier  "ATI Video Device"
    Driver      "ati"
    BusID       "PCI:00:02:00"
EndSection
```

En què l'entrada Driver indica el *driver* de vídeo utilitzat i que es pot adequar al dispositiu que tinguem (si no hi ha aquesta entrada Xorg, la configura automàticament segons el seu criteri). Les dades es poden obtenir de l'execució de `lspci` buscant VGA, vídeo, etc., i el nom del *driver* generalment és la marca que identifica el dispositiu (per exemple, ATI=ati, Intel i740=i740, etc.). A http://www.calel.org/pci-devices/xorg-device-list.html es poden buscar les targetes i el *driver* corresponent.

Cada secció té valors i paràmetres d'ajust que es poden adaptar a les configuracions desitjades. Podeu consultar https://wiki.debian.org/Xorg per a obtenir més informació. Recordeu que, després de modificar la configuració, sempre s'ha de fer `systemctl restart gdm3` (o `/etc/init.d/gdm3 restart`, tot i que aquesta ordre està obsoleta).

Nota

Aquest capítol no pretén ser un llibre d'administració i instal·lació de Debian, sinó recollir els aspectes més interessants del GNU/Linux i executar alguns exemples de configuració sobre el Debian (en la seva darrera versió). Podeu obtenir informació detallada sobre els processos d'instal·lació de Debian en *El llibre de l'administrador de Debian* (disponible com a llibre electrònic: http://debian-handbook.info/browse/es-ES/stable)

6. Programació d'ordres combinades (*shell scripts*)

En aquest apartat veurem la importància fonamental que té l'intèrpret d'ordres o instruccions (*shell*) i sobretot analitzarem amb cert detall les seves possibilitats per a executar fitxers d'ordres seqüencials i escrits en text pla (ASCII), que seran interpretats pel *shell*. Aquests fitxers, anomenats *shell scripts*, són l'eina fonamental de qualsevol usuari avançat, i imprescindibles per a un administrador de sistemes *nix.

Un coneixement pràctic de *shell scripting* serà necessari, ja que el mateix GNU/Linux es basa en aquest tipus de recurs per a iniciar el sistema (per exemple, executant els arxius d'inicialització del sistema a */etc/systemd/* –o a */etc/init.d/*–), per la qual cosa els administradors han de tenir els coneixements necessaris per a treballar-hi, entendre el funcionament del sistema, modificar-los i adequar-los a les necessitats específiques de l'entorn en el qual treballin.

Molts autors consideren que fer *scripting* és un art, però, segons el nostre parer, no és difícil d'aprendre, ja que es poden aplicar tècniques de treballar per etapes (*divide & conquer*), ja que s'executarà de manera seqüencial i el conjunt d'operadors/opcions no és tan extens perquè generi dubtes sobre quin recurs s'ha d'utilitzar. Sí que la sintaxi serà dependent del *shell* utilitzat, però en ser un llenguatge interpretat amb encadenament de seqüències d'ordres serà molt fàcil de depurar i posar en funcionament.

Avui dia l'*scripting* ha arribat a molts àmbits amb la potencialitat presentada per alguns llenguatges com PHP per a desenvolupar codi del costat del servidor (originalment dissenyat per al desenvolupament web de contingut dinàmic); Perl, que és un llenguatge de programació (1987) que pren característiques de C, de *Bourne shell*, AWK, `sed`, Lisp entre d'altres; Phyton (1991), la filosofia del qual posa l'accent en una sintaxi que afavoreixi un codi llegible, essent un llenguatge de programació (interpretat) que suporta orientació a objectes, programació imperativa i, en menys mesura, programació funcional; i amb tipus dinàmics o com Ruby (1993-1995), que és un llenguatge de programació (també interpretat), reflexiu i orientat a objectes, que combina una sintaxi inspirada en Python i Perl amb característiques de programació orientada a objectes similars a Smalltalk.

També, un *shell script* és un mètode *quick-and-dirty* (que es podria traduir com a ràpid i en brut) de prototipatge d'una aplicació complexa per a aconseguir fins i tot un subconjunt limitat de la funcionalitat útil en una primera etapa del desenvolupament d'un projecte. D'aquesta manera, l'estructura de l'aplicació i

les grans línies es poden provar i es pot determinar quines seran les dificultats principals abans de procedir al desenvolupament de codi en C, C++, Java, o un altre llenguatge estructurat i/o interpretat més complex.

D'acord amb M. Cooper hi ha una sèrie de situacions en les quals ell categòricament indica que "no s'han d'utilitzar *shell scripts*", si bé al nostre entendre diríem que s'haurà de ser prudent i analitzar en profunditat el codi desenvolupat sobretot des del punt de vista de la seguretat per a evitar que es puguin dur a terme accions més enllà d'aquelles per a les quals es van dissenyar (per exemple, en *scripting* per la banda del servidor en aplicacions web és una de les causes principals d'intrusió o execució no autoritzades). Entre les causes podem enumerar: ús intensiu de recursos, operacions matemàtiques d'alt rendiment, aplicacions complexes en què es necessiti estructuració (per exemple llistes, arbres) i tipus de variables, situacions en les quals la seguretat sigui un factor determinant, accés a arxius extensos, matrius multidimensionals, treballar amb gràfics, accés directe al maquinari/comunicacions, No obstant això, per exemple, amb la utilització de Perl, Phyton o Rugbi, moltes d'aquestes recomanacions deixen de tenir sentit i en alguns àmbits científics (per exemple, en genètica, bioinformàtica, química, etc.) l'*scripting* és la forma habitual de treball.

Aquest capítol està orientat a recollir les característiques principals de Bash (*bourne-again shell*), que és el *shell script* estàndard en GNU/Linux, però molts dels principis explicats es poden aplicar a Korn Shell o C Shell.

6.1. Introducció: l'intèrpret d'ordres

L'intèrpret (*shell*) és una peça de programari que proporciona una interfície per als usuaris en un sistema operatiu i que proveeix accés als serveis del nucli. El seu nom anglès prové de l'embolcall extern d'alguns mol·luscs, ja que és la "part externa que protegeix el nucli".

Els intèrprets (o *shells*) es divideixen en dues categories: línia d'ordres i de gràfics, depenent de si la interacció es fa mitjançant una línia d'instruccions (CLI, *command line interface*) o en mode gràfic per mitjà d'una GUI (*graphical user interface*). En qualsevol categoria l'objectiu principal de l'intèrpret és invocar o "llançar" un altre programa; tanmateix, solen tenir capacitats addicionals, com ara veure el contingut dels directoris, interpretar ordres condicionals, treballar amb variables internes, gestionar interrupcions, redirigir entrada/sortida, etc.

Si bé un intèrpret gràfic és agradable per a treballar i permet a usuaris sense gaires coneixements exercir-se amb certa facilitat, els usuaris avançats prefereixen els de mode text, ja que permeten una manera més ràpida i eficient de treballar. Tot és relatiu, ja que en un servidor és probable que un usuari administrador no utilitzi ni tan sols interfície gràfica, mentre que un usuari d'edició

de vídeo mai no treballarà en mode text. El principal atractiu del sistema *nix és que també en mode gràfic es pot obrir un terminal i treballar en mode text com si estigués en un intèrpret de text, o canviar interactivament del mode gràfic i treballar en mode text (fins amb 6 terminals diferents amb Ctrl-Alt-F1... F6, generalment) i després tornar al mode gràfic amb una simple combinació de tecles (Ctrl-Alt-F7). En aquest apartat ens dedicarem a l'intèrpret en mode text i a les característiques avançades en la programació de *shell scripts*.

Entre els intèrprets més populars (o històrics) en els sistemes *nix tenim:

- Bourne *shell* (*sh*).
- Almquist *shell* (*ash*) o la seva versió de Debian (*dash*).
- Bourne-Again *shell* (*bash*).
- Korn *shell* (*ksh*).
- Z *shell* (*zsh*).
- C *shell* (*csh*) o la versió Tenex C *shell* (*tcsh*).

El Bourne *shell* ha estat l'estàndard *de facto* en els sistemes *nix, ja que va ser distribuït per Unix Version 7 el 1977, i Bourne-Again (Bash) és una versió millorada del primer escrita pel projecte GNU sota llicència GPL, la qual s'ha transformat en l'estàndard dels sistemes GNU/Linux. Com ja hem comentat, Bash serà l'intèrpret que analitzarem, ja que, a més de ser l'estàndard, és molt potent, té característiques avançades, recull les innovacions plantejades en altres intèrprets i permet executar sense modificacions (generalment) scripts fets per a qualsevol intèrpret amb sintaxi compatible amb Bourne *shell*. L'ordre `cat /etc/shells` ens proporciona els intèrprets coneguts pel sistema Linux, independentment de si estan instal·lats o no. L'intèrpret per defecte per a un usuari s'obtindrà de l'últim camp de la línia corresponent a l'usuari del fitxer */etc/passwd* i canviar d'intèrpret simplement significa executar el nom de l'intèrpret sobre un terminal actiu, per exemple:

```
RS@DebSyS:~$ echo $SHELL
/bin/bash
RS@DebSyS:~$ tcsh
DebSyS:~>
```

Una part important és el que es refereix als modes d'execució d'un intèrpret. S'anomena *login* interactiu o *entrada interactiva* quan prové de l'execució d'una entrada, fet que implicarà que s'executaran els arxius */etc/profile*, *~/.bash_profile*, *~/.bash_login* o *~/.profile* (la primera de les dues que existeixi i es pugui llegir) i *~/.bash_logout* quan acabem la sessió. Quan és de no-connexió interactiu (s'executa un terminal nou) s'executarà *~/.bashrc*, ja que un *bash* interactiu té un conjunt d'opcions habilitades respecte a si no ho és (consulteu el manual). Per a saber si l'intèrpret és interactiu, executeu `echo $-` i ens haurà de respondre **himBH**, en què ***i*** indica que sí que ho és.

Hi ha un conjunt d'ordres internes[4] en l'intèrpret, és a dir, integrats amb el codi d'intèrpret, que per a Bourne és:

(4)Consulteu el manual per a una descripció completa.

`:, ., break, cd, continue, eval, exec, exit, export, getopts, hash, pwd, readonly, return, set, shift, test, [, times, trap, umask, unset.`

I a més el Bash inclou:

`alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit, unalias.`

Quan el Bash executa un *shell script*, crea un procés fill que executa un altre Bash, el qual llegeix les línies de l'arxiu (una línia per vegada), les interpreta i executa com si vinguessin del teclat. El procés Bash pare espera mentre el Bash fill executa l'script fins al final, en què el control torna al procés pare, el qual torna a posar l'indicador o *prompt* novament. Una ordre d'intèrpret és una cosa tan simple com `touch arxiu1 arxiu2 arxiu3`; consisteix en l'ordre seguida d'arguments, separats per espais, *arxiu1* és el primer argument i així successivament.

6.1.1. Redireccions i *pipes*

Hi ha tres descriptors de fitxers: *stdin, stdout* i *stderr* (l'abreviatura *std* significa 'estàndard'), i en la majoria dels intèrprets (fins i tot en Bash) es pot redirigir *stdout* i *stderr* (junts o separats) a un fitxer, *stdout* a *stderr* i viceversa. Totes es representen per un número; 0 representa *stdin*, 1, *stdout*, i 2, *stderr*.

Per exemple, enviar *stdout* de l'ordre *ls* a un fitxer serà `ls -l > dir.txt`, en què es crearà un fitxer anomenat *dir.txt*, que contindrà el que es veuria a la pantalla si s'executés `ls -l`

Per a enviar la sortida *stderr* d'un programa a un fitxer, escriurem la instrucció `grep xx yy 2> error.txt`. Per a enviar l'*stdout* a l'*stderr*, farem servir l'ordre `grep xx yy 1>&2` i a la inversa simplement intercanviant l'1 per 2, és a dir, `grep xx yy 2>&1`.

Si volem que l'execució d'una ordre no generi activitat per pantalla, la qual cosa es denomina *execució silenciosa*, només hem de redirigir totes les seves sortides a */dev/null*; per exemple, pensant en una ordre del `cron` que volem que esborri tots els arxius acabats en .mov del sistema:

```
rm -f $(find / -name "*.mov") &> /dev/null.
```

Però s'ha d'anar amb compte i estar molt segur, ja que no tindrem cap sortida per pantalla.

Els *pipes* permeten utilitzar de manera simple tant la sortida d'una ordre com l'entrada d'una altra; per exemple, `ls -l | sed -i "s/[aeio]/u/g"`, en què s'executa l'ordre `ls`, i la seva sortida, en comptes d'imprimir-se a la pantalla, s'envia (per un tub o *pipe*) al programa `sed`, que imprimeix la seva sortida corresponent. Per exemple, per a buscar en el fitxer */etc/passwd* totes les línies que acabin amb *false* podríem fer `cat /etc/passwd | grep false$`, en què s'executa el `cat`, i la seva sortida es passa al `grep` (el símbol $ al final de la paraula indica al `grep` que és final de línia).

6.1.2. Aspectes generales

Si l'entrada no és un comentari, és a dir (deixant de banda els espais en blanc i tabulador), la cadena **no** comença per **#**, l'intèrpret llegeix i el divideix en paraules i operadors, que es converteixen en ordres, operadors i altres construccions. A partir d'aquest moment, es fan les expansions i substitucions, les redireccions, i finalment, l'execució de les ordres.

En el Bash es podran tenir funcions, que són una agrupació d'ordres que es poden invocar posteriorment. I quan s'invoca el nom de la funció d'intèrpret (s'usa com un nom d'ordre simple), la llista d'ordres relacionades amb el nom de la funció s'executarà tenint en compte que les funcions s'executen en el context de l'intèrpret en curs, és a dir, no es crea cap procés nou per a això.

Un **paràmetre** és una entitat que emmagatzema valors, que poden ser un nom, un nombre o un valor especial. Una **variable** és un paràmetre que emmagatzema un nom i té atributs (1 o més o cap) i es creen amb la sentència `declare` i es treuen amb `unset;` i si no els donem dóna valor, tenen assignada la cadena nul·la.

Finalment, hi ha un conjunt d'expansions que fa l'intèrpret que es du a terme en cada línia i que poden ser enumerades com: d'accent/cometes, paràmetres i variables, substitució d'ordres, d'aritmètica, de separació de paraules i de noms d'arxius.

6.2. Elements bàsics d'un *shell script*

6.2.1. Què és un *shell script*?

Un *shell script* és simplement un arxiu (mysys.sh per a nosaltres) que té aquest contingut (hem numerat les línies per a referir-nos-hi com el `cat -n` però no formen part de l'arxiu):

```
RS@debian:~$ cat -n mysys.sh

1 #!/bin/bash
2 clear; echo "Informació donada pel shell script mysys.sh."
3 echo "Hola, $USER"
4 echo "La data és `date`, i aquesta setmana `date +"%V"`."
5 echo -n "Usuaris connectats:"
6 w | cut -d " " -f 1 - | grep -v USER | sort -u
7 echo "El sistema és `uname -s` i el processador és `uname -m`."
8 echo "El sistema està engegat des de fa:"
9 uptime
10 echo
11 echo "Això és tot!"
```

Per a executar-lo podem fer `bash mysys.sh` o bé canviar-li els atributs per a fer-lo executable i executar-lo com una ordre (al final es mostra la sortida després de l'execució):

```
RS@debian:~$ chmod 744 mysys.sh

RS@debian:~$ ./mysys.sh

Informació donada pel shell script mysys.sh.

Hola, RS

La data és Wen Jun 8 10:47:33 EDT 2016, i aquesta és la setmana 23.

Usuaris connectats: RS

El sistema és Linux i el processador és x86_64.

El sistema està engegat des de fa:
10:53:07 up 1 day, 11:35, 1 user, load average: 0.12, 0.125, 0.33

Això és tot!
```

L'script comença amb "#!", que és una línia especial per a indicar amb quin intèrpret s'ha de llegir aquest script. La línia 2 és un exemple de dues ordres (una esborra la pantalla i l'altra imprimeix el que hi ha a la dreta) en la mateixa línia, i per això han d'estar separades per ";". El missatge s'imprimeix amb la sentència `echo` (ordre interna) però també es podria haver utilitzat la sentència `printf` (també ordre interna) per a una sortida amb format. Quant als aspectes més interessants de la resta, la línia 3 mostra el valor d'una variable, la 4 forma una cadena de caràcters (*string*), la 5 forma una cadena amb la sortida d'una ordre (reemplaçament de valors), la 6 executa la seqüència de 4 ordres amb paràmetres encadenats per *pipes* "|" i la 7 (similar a la 4) també reemplaça valors d'ordres per a formar una cadena de caràcters abans d'imprimir-se per pantalla.

#!/bin/bash

Així evitem que si l'usuari té un altre intèrpret, l'execució doni errors de sintaxi, és a dir, garantim que aquest script sempre s'executarà amb *bash*.

Per a depurar un *shell script* podem executar l'script amb `bash -x mysys.sh` o incloure en la primera línia *-x:* `#!/bin/bash -x`.

També es pot depurar una secció de codi incloent-hi:

```
set -x # activa depuració des d'aquí
codi per depurar
set +x # per a la depuració
```

6.2.2. Variables i matrius (*arrays*)

Es poden usar variables però no hi ha tipus de dades. Una variable de Bash pot contenir un nombre, un caràcter o una cadena de caràcters; no es necessita declarar una variable, ja que es crearà només assignar-li un valor. També es pot declarar amb `declare` i després assignar-li un valor:

```
#!/bin/bash
a="Hola UOC"
echo $a
declare b
echo $b
b="Com esteu?"
echo $b
```

Com es pot veure, es crea una variable *a* i se li assigna un valor (per a delimitar-lo s'ha de posar entre " ") i es recupera el VALOR d'aquesta variable posant-li un $ al principi, i si no se li posa $, només imprimirà el nom de la variable, no el valor. Per exemple, per a fer un script que faci una còpia (*backup*) d'un directori, incloent-hi la data i hora, en el moment de fer-ho, en el nom de l'arxiu (intenteu fer això tan fàcil amb ordres en un sistema W i acabareu frustrats):

```
#!/bin/bash
OF=/home/$USER-$(date +%d%m%Y).tgz
tar -cZf $OF /home/$USER
```

Aquest script introdueix una cosa nova, ja que estem creant una variable i assignem un valor (resultat de l'execució d'una ordre en el moment de l'execució). Fixeu-vos en l'expressió *$(date +%d%m%Y)*, que es posa entre () per a capturar-ne el valor. USER és una variable d'entorn i només es reemplaçarà pel seu valor. Si volguéssim agregar (concatenar) a la variable USER, per exemple, una cadena més, hauríem de delimitar la variable amb {}. Per exemple, tenint en compte que USER=RS:

```
RS@debian:~$OF=/home/${USER}uppi-$(date +%d%m%Y).tgz
RS@debian:~$ echo $OF
/home/RSuppi-17042010.tgz
```

Així, el nom del fitxer serà diferent cada dia. És interessant veure el reemplaçament d'ordres que fa el Bash amb els parèntesis; per exemple, `echo $(ls)`.

Les variable locals es poden declarar anteposant *local*, i això en delimita l'àmbit.

```
#!/bin/bash
HOLA=Hola
function uni {
local HOLA=UOC
echo -n $HOLA
}
echo -n $HOLA
uni
echo $HOLA
```

La sortida serà *HolaUOCHola*, en què hem definit una funció amb una variable local que recupera el seu valor quan s'acaba d'executar la funció.

Les variables les podem declarar com a **ARRAY[INDEXNR]=valor**, en què *INDEXNR* és un nombre positiu (començant des de 0) i també podem declarar una matriu com `declare -a ARRAYNAME`, i es poden fer assignacions múltiples amb: **ARRAY=(value1 value2... valueN)**

```
RS@debian:~$ array=( 1 2 3)
RS@debian:~$ echo $array
1
RS@debian:~$ echo ${array[*]}
1 2 3
RS@debian:~$ echo ${array[2]}
3
RS@debian:~$ array[3]=4
RS@debian:~$ echo ${array[*]}
1 2 3 4
RS@debian:~$ echo ${array[3]}
```

```
4
RS@debian:~$ unset array[1]
RS@debian:~$ echo ${array[*]}
1 3 4
```

6.2.3. Estructures condicionals

Les estructures condicionals ens permeten decidir si es fa una acció o no; aquesta decisió es pren avaluant una expressió.

- La més bàsica és: **if expressió; then sentència; fi**, en què *sentència* només s'executa si *expressió* s'avalua com a verdadera. 2 < 1 és una expressió que s'avalua falsa, mentre que 2 > 1 s'avalua verdadera.

- Els condicionals tenen altres formes, com ara: **if expressió; then sentència1; else sentència fi**. Aquí *sentència1* s'executa si *expressió* és verdadera. D'altra manera, s'executa *sentència2*.

- Una altra forma més de condicional és: **if expressió1; then sentència1; elif expressió2; then sentència2; else sentència3**. En aquesta forma només s'afegeix *else if expressió2 then sentència2*, que fa que *sentència2* s'executi si *expressió2* s'avalua verdadera. La sintaxi general és:

```
if [expressió];
   then
   codi si 'expressió' és verdadera.
   fi
```

Un exemple en el qual el codi que s'executarà si l'expressió entre claudàtors és verdadera es troba entre la paraula *then* i la paraula *fi*, que indica el final del codi executat condicionalment:

```
#!/bin/bash
if [ "pirulo" = "pirulo" ]; then
echo expressió avaluada com a verdadera
fi
```

Un altre exemple amb variables i comparació per igualtat i per diferència (= o !=):

```
#!/bin/bash
T1="pirulo"
T2="pirulon"
if [ "$T1" = "$T2" ]; then
echo expressió avaluada com a verdadera
else
echo expressió avaluada com a falsa
```

```
fi
if [ "$T1" != "$T2" ]; then
echo expressió avaluada com a verdadera
else
echo expressió avaluada com a falsa
fi
```

Un exemple d'expressió de comprovació si hi ha un fitxer (cal vigilar amb els espais després de [i abans de]):

```
FILE=~/.bashrc
if [ -f $FILE ]; then
echo el fitxer $FILE existeix
else
echo fitxer no trobat
fi
if [ 'test -f $FILE' ]; then
echo Una altra manera d'expressió per a saber si existeix o no
fi
```

Hi ha versions curtes d'**if**, com per exemple:

```
[ -z "${COLUMNS:-}" ] && COLUMNS=80
```

És la versió curta de:

```
if [ -z "${COLUMNS:-}" ]; then
COLUMNS=80
fi
```

Un altre exemple d'ús de la sentència **if** i variables:

```
#!/bin/bash
FILE=/var/log/syslog
MYMAIL="adminp@master.hpc.local"
SUBJECT="Error - $(hostname)"
BODY="Existeixen ERRORS en $(hostname) @ $(date). Veure syslog."
OK="OK: No es detecten errors en Syslog."
WARN="Possibles Errors. Analitzar"
# Verificar que $FILE existeix
if test ! -f "$FILE"
then
   echo "Error: $FILE no existeix. Analitzar quin és el problema."
   exit 1
fi
# Busquem errors
errors=$(grep -c -i "rror" $FILE)
```

```
# Si?
if [ $errors ]
then     # Si ...
            echo "$BODY" | mail -s "$SUBJECT" $MYMAIL
            echo "Mail enviat ..."
else      # No
            echo "$OK"
fi
```

La sentència `if then else` es pot resumir com:

```
if list; then list; [ elif list; then list; ] ... [ else
list; ] fi
```

S'ha de parar esment als ";" i a les paraules clau, i no és necessari posar tota la sentència en una sola línia.

6.2.4. Els bucles

El bucle **for** té dues formes, una d'elles és diferent a la d'altres llenguatges de programació, ja que permet iterar sobre una sèrie de "paraules" contingudes dins d'una cadena. La segona forma de **for** segueix els patrons dels llenguatges de programació habituals. El bucle **while** executa una secció de codi si l'expressió de control és verdadera, i només s'atura quan és falsa (o es troba una interrupció explícita dins del codi en execució; per exemple, un **break**). El bucle **until** és gairebé idèntic al bucle **while**, tret que el codi s'executa mentre l'expressió de control s'avalua com a falsa. Vegem-ne uns exemples i observem-ne la sintaxi:

```
#!/bin/bash
for i in $( ls ); do
echo element: $i
done
```

En la segona línia declarem *i* com la variable que rebrà els diferents valors continguts a `$( ls )`, que seran els elements del directori obtingut en el moment de l'execució. El bucle es repetirà (entre *do* i *done*) tantes vegades com elements tingui la variable i en cada iteració la variable *i* adquirirà un valor diferent de la llista.

Una altra sintaxi acceptada podrà ser:

```
#!/bin/bash
for i in `seq 1 10`;
do
echo $i
```

```
done
```

En la segona forma del `for`, la sintaxi és

```
for (( expr1 ; expr2 ; expr3 )) ; do list ; done
```

on primer s'avalua l'expressió `expr1;` després s'avalua l'expressió repetidament fins que és 0. Cada vegada que la `expr2` és diferent de 0, s'executa `list` i l'expressió aritmètica `expr3` és avaluada. Si alguna de les expressions s'omet, aquesta sempre s'avaluarà com a 1. Per exemple:

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
   echo "Repetició Núm. $c "
done
```

Un bucle infinit podria ser:

```
#!/bin/bash
for (( ; ; ))
do
   echo "Bucle infinit [ introduir CTRL+C per a finalitzar]"
done
```

La sintaxi de **while** és:

```
#!/bin/bash
comptador=0
while [ $comptador -lt 10 ]; do
echo El comptador és $comptador
let comptador=comptador+1
done
```

Com podem veure, a més de **while** hem introduït una comparació en l'expressió *per menor* "-lt" i una operació numèrica amb *let comptador=comptador +1*. La sintaxi d'**until** és equivalent:

```
#!/bin/bash
comptador=20
until [ $comptador -lt 10 ]; do
echo Comptador-Until $comptador
let comptador-=1
done
```

En aquest cas veiem l'equivalència de la sentència i a més una altra manera de fer operacions sobre les mateixes variables amb "-=".

Les sentències repetitives es poden resumir com:

```
for name [ [ in [ word ... ] ] ; ] do list ; done
for (( expr1 ; expr2 ; expr3 )) ; do list ; done
while list-1; do list-2; done
until list-1; do list-2; done
```

Com en la sentència de `if` s'han de tenir en compte els ";" i les paraules clau de cada instrucció.

6.2.5. Funcions, *select*, *case*, arguments i altres qüestions

Les funcions són la manera més fàcil d'agrupar seccions de codi, que es podran tornar a utilitzar. La sintaxi és *function nom { codi }*, i per a cridar-les només és necessari que siguin dins del mateix arxiu i escriure el seu nom.

```
#!/bin/bash
function sortir {
exit
}
function hola {
echo Hola UOC!
}
hola
sortir
echo Mai no arribareu a aquesta línia
```

Com es pot veure tenim dues funcions, *sortir* i *hola*, que després s'executaran (no és necessari declarar-les en un ordre específic) i com la funció *sortir* executa un **exit** en l'intèrpret mai no arribarà a executar l'*echo* final. També podem passar arguments a les funcions, que seran llegits per ordre (de la mateixa manera com es llegeixen els arguments d'un script): *$1* el primer, *$2* el segon i així successivament:

```
#!/bin/bash
function sortir {
exit
}
function hola-amb-parametres {
echo $1
}
hola-amb-parametres Hola
hola-amb-parametres UOC
```

```
sortir
echo Mai no arribareu a aquesta línia
```

El `select` és per fer a opcions i llegir des de teclat:

```
#!/bin/bash
OPCIONS="Sortir Opció1"
select opt in $OPCIONS; do
if [ "$opt" = "Sortir" ]; then
echo Sortir. Adéu.
exit
elif [ "$opt" = "Opció1" ]; then
echo Hola UOC
else
clear
echo opció no permesa
fi
done
```

El `select` permet fer menús en mode text i és equivalent a la sentència `for`, només que itera sobre el valor de la variable indicada en el `select`. Un altre aspecte important és la lectura d'arguments, com per exemple:

```
#!/bin/bash
if [ -z "$1" ]; then
echo ús: $0 directori
exit
fi
A=$1
B="/home/$USER"
C=backup-home-$(date +%d%m%Y).tgz
tar -cZf $A$B $C
```

Si el nombre d'arguments és 0, escriu el nom de l'ordre ($0) amb un missatge d'ús. La resta és similar al que hem vist anteriorment excepte per al primer argument ($1). Fixeu-vos amb atenció en l'ús de " i la substitució de variables.

El `case` és una altra forma de selecció; mirem aquest exemple que ens alerta en funció de l'espai de disc que tenim disponible:

```
#!/bin/bash
space=`df -h | awk '{print $5}' | grep % \
| grep -v Use | sort -n | tail -1 | cut -d "%" -f1 -`

   case $space in
   [1-6]*)
```

```
   Message="tot bé."
   ;;
   [7-8]*)
   Message="Podríeu començar a esborrar alguna cosa en $space %"
   ;;
   9[1-8])
   Message="Uff. Millor un disc nou. Partició $space % fins a dalt."
   ;;
   99)
   Message="Pànic! No teniu espai en $space %!"
   ;;
   *)
   Message="No teniu res..."
   ;;
   esac
   echo $Message
```

A més de parar atenció a la sintaxi del **case**, podem mirar com s'escriu una ordre en dues línies amb "\", el filtratge de seqüència amb diversos `grep` i | i una ordre interessant (filtre) com `awk` (la millor ordre de tots els temps).

Com a resum de les sentències abans vistes tenim:

```
select name [ in word ] ; do list ; done
case word in [ [(] pattern [ | pattern ] ... ) list ;; ] ... esac
```

Un exemple de la sentència `break` podria ser el següent, que busca un arxiu en un directori fins que el troba:

```
#!/bin/bash
for file in /etc/*
do
if [ "${file}" == "/etc/passwd" ]
then
   nroentradas=`wc -l /etc/passwd | cut -d" " -f 1`
   echo "Existeixen ${nroentrades} entrades definides en ${file}"
   break
fi
done
```

L'exemple següent mostra com s'utilitza el `continue` per a continuar amb el bucle següent del llaç:

```
#!/bin/bash
for f in `ls`
do
```

```
    # if l'arxiu .org existeix llegeixo la següent
    f=`echo $f | cut -d"." -f 1`
    if [ -f ${f}.org ]
    then
       echo "Següent $f arxiu..."
       continue  # llegeixo la següent i salto l'ordre cp
    fi
    # no tenim l'arxiu .org llavors el copio
    cp $f $f.org
done
```

Moltes vegades, podeu voler sol·licitar a l'usuari alguna informació, i hi ha diverses maneres per a fer-ho:

```
#!/bin/bash
echo Si us plau, escriviu el nom
read nom
echo "Hola, $nombre!"
```

Com a variant, es poden obtenir múltiples valors amb *read:*

```
#!/bin/bash
echo Si us plau, escriviu el nom i el primer cognom
read NO AP
echo "Hola $AP, $NO!"
```

Si fem `echo 10 + 10` i esperàveu veure un 20 quedareu desil·lusionats; la forma d'avaluació directa és amb `echo $((10+10))` o també `echo $[10+10]` i si necessiteu operacions com fraccions o d'altres podeu utilitzar `bc`, ja que l'anterior només serveix per a nombres enters. Per exemple, `echo $[3/4]` donarà 0, però `echo 3/4|bc -l` sí que funcionarà. També recordem l'ús del `let;` per exemple, `let a=75*2` assignarà 150 a la variable *a*.

Un exemple més:

```
RS@debian:~$ echo $date
20042011
RS@debian:~$ echo $((date++))
20042011
RS@debian:~$ echo $date
20042012
```

És a dir, com a operadors podem utilitzar:

- VAR++ VAR-- variable postincrement i postdecrement.
- ++VAR --VAR com l'anterior però abans.
- + - operadors unaris.

- ! ~ negació lògica i negació en *string*.
- ** exponent.
- / + - % operacions.
- < < > > desplaçament a esquerra i dreta.
- < = >= < > comparació.
- == != igualtat i diferència.
- & ^ | operacions AND, OR exclusiu i OR.
- && || operacions AND i OR lògics.
- *expr* ? *expr* : *expr* avaluació condicional.
- = *= /= %= += -= < < = >>= &= ^= |= operacions implícites.
- , separador entre expressions.

Per a capturar la sortida d'una ordre, és a dir, el resultat de l'execució, podem utilitzar el nom de l'ordre entre accents oberts ` `.

```
#!/bin/bash
A=`ls`
for b in $A ;
do
file $b
done
```

Per a la comparació tenim les opcions següents:

- s1 = s2 verdader si s1 coincideix amb s2.
- s1 != s2 verdader si s1 no coincideix amb s2.
- s1 < s2 verdader si s1 és alfabèticament anterior a s2.
- s1 > s2 verdader si s1 és alfabèticament posterior a s2.
- -n s1 s1 no és nul (conté un o més caràcters).
- -z s1 s1 és nul.

Per exemple, cal anar amb compte perquè si S1 o S2 són buits, ens donarà un error d'interpretació (*parse*):

```
#!/bin/bash
S1='cadena'
S2='Cadena'
if [ $S1!=$S2 ];
then
echo "S1('$S1') no és igual que S2('$S2')"
fi
if [ $S1=$S1 ];
then
echo "S1('$S1') és igual que S1('$S1')"
fi
```

Quant als operadors aritmètics i relacionals, podem utilitzar:

a) Operadors aritmètics:

- + (addició).
- - (sostracció).
- * (producte).
- / (divisió).
- % (mòdul).

b) Operadors relacionals:

- -lt (<).
- -gt (>).
- -li (< =).
- -ge (>=).
- -eq (==).
- -ne (!=).

Fem un exemple de script que ens permetrà rebatejar fitxers S1 amb una sèrie de paràmetres (prefix o sufixos) d'acord amb els arguments. La manera serà *p* [*prefix*] *fitxers*, o si no, *s* [*sufix*] *fitxers*, o també *r* [*patró-antic*] [*patró-nou*] *fitxers* (i com sempre diem, els scripts són millorables):

```
#!/bin/bash -x
# renom: reanomena múltiples arxius d'acord amb certes regles
# Basat en un original de F. Hudson. Gener de 2000
# comprova si vull reanomenar amb prefix i deixo només els noms
# d'arxius
if [ $1 = p ]; then
      prefix=$2 ; shift ; shift

      # si no hi ha entrades d'arxius acabo.
      if [ "$1" = '' ]; then
         echo "no s'han especificat arxius"
         exit 0
      fi
      # Interacció per a reanomenar
      for arxiu in $*
         do
         mv ${arxiu} $prefix$arxiu
         done
      exit 0
fi
# comprova si vull reanomenar amb prefix i deixo només els noms
# d'arxius
if [ $1 = s ]; then
```

```
      sufix=$2 ; shift ; shift
      if [ "$1" = '' ]; then
         echo "no s'han especificat arxius"
         exit 0
      fi
      for arxiu in $*
         do
         mv ${arxiu} $arxiu$sufix
         done
      exit 0
fi
# comprova si és una substitució de patrons
if [ $1 = r ]; then
shift
# s'ha inclòs això com a mesura de seguretat
      if [ $# -lt 3 ] ; then
         echo "ús: renom r [expressió] [substitut] arxius... "
         exit 0
      fi
      VELL=$1 ; NOU=$2 ; shift ; shift
      for arxiu in $*
         do
         nou=`echo ${arxiu} | sed s/${VELL}/${NOU}/g`
         mv ${arxiu} $nou
         done
      exit 0
fi
# si no mostro l'ajuda
echo "ús:"
echo " renom p [prefix] arxius..."
echo " renom s [sufix] arxius..."
echo " renom r [patró-antic] [patró-nou] arxius..."
exit 0
```

Cal parar una atenció especial a "", '', `` {}, etc, per exemple:

```
RS@debian:~$ date=20042010
RS@debian:~$ echo $date
20042010
RS@debian:~$ echo \$date
$date
RS@debian:~$ echo '$date'
$date
RS@debian:~$ echo "$date"
20142005
RS@debian:~$ echo "`date`"
Mon Jun 9 00:33:42 CEST 2014
```

```
RS@debian:~$ echo "el que se m'ocorre és: \"Estic cansat\""
el que se m'ocorre és: "Estic cansat"
RS@debian:~$ echo "\"
>
(espera més entrades: feu Ctrl-C)
remo@debian:~$ echo "\\"
\
RS@debian:~$ echo grand{et,ot,às}
grandet grandot grandàs
```

Una combinació d'entrada de teclat i operacions/expressions per a calcular un any de traspàs:

```
#!/bin/bash
clear;
echo "Entreu l'any que cal verificar (4 dígits), i després [ENTER]:"
read year
if (( ("$year" % 400) == "0" )) || (( ("$year" % 4 == "0") \
&& ("$year" % 100 !="0") )); then
      echo "$year és de traspàs."
else
      echo "$year l'any no és de traspàs."
fi
```

Una ordre interessant que combina `read` amb delimitador en una expressió (està posat tot en una línia i per això la sintaxi està separada per ;). El resultat serà "interessant" (ens mostrarà totes les cadenes donades pel `find` en una línia i sense /:

```
find "$PWD" -name "m*" | while read -d "/" file; do echo $file; done
```

Un aspecte interessant per a treballar amb string (vegeu el manual de Bash: `man bash`) és ${*VAR:*OFFSET:*LENGTH*}.

```
RS@debian:~$ export str="unstring molt llarg"
RS@debian:~$ echo $str
unstring molt llarg
RS@debian:~$ echo ${str:4}
ring molt llarg
RS@debian:~$ echo $str
unstring molt llarg
RS@debian:~$ echo ${str:9:3}
mol
RS@debian:~$ echo ${str%llarg}
unstring molt
```

La instrucció `trap` ens permetrà capturar un senyal (per exemple, de teclat) dins d'un *script*.

```
#!/bin/bash
# Per a acabar feu des d'un altre terminal kill -9 pid

trap "echo ' Ja!'" SIGINT SIGTERM
echo "El pid és $$"

while : # això és el mateix que "while true".
      do
      sleep 10 # L'script no fa res.
done
```

6.2.6. Filtres: *grep*

El *grep* és un filtre de patrons molt útil i versàtil; a continuació en podem veure alguns exemples:

```
RS@debian:~$ grep root /etc/passwd    busca patró root
root:x:0:0:root:/root:/bin/bash

RS@debian:~$ grep -n root /etc/passwd    a més numera les línies
1:root:x:0:0:root:/root:/bin/bash

RS@debian:~$ grep -v bash /etc/passwd | grep -v nologin
     mostra els que no tenen el patró i bash ni el patró nologin
daemon:x:1:1:daemon:/usr/sbin: /bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
...

RS@debian:~$ grep -c false /etc/passwd    compta els que tenen false
7
RS@debian:~$ grep -i ps ~/.bash* | grep -v history
     busca el patró ps en tots els arxius que comencen per .bash en el directori /home
     excloent-hi els que tinguin history en l'arxiu
/home/RS/.bashrc:[ -z "$PS1" ] && return
/home/RS/.bashrc:export HISTCONTROL=$HISTCONTROL${HISTCONTROL+,}ignoredups
/home/RS/.bashrc:# ... or force ignoredups and ignorespace
...

RS@debian:~$ grep ^root /etc/passwd     busca l'inici de línia
root:x:0:0:root:/root:/bin/bash

RS@debian:~$ grep false$ /etc/passwd    busca el final de línia
```

```
Debian-exim:x:101:105::/var/spool/exim4:/bin/false
statd:x:102:65534::/var/lib/nfs:/bin/false

RS@debian:~$ grep -w / /etc/fstab    busca /
/dev/hda1 / ext3 errors=remount-ro 0 1

RS@debian:~$ grep [yf] /etc/group    busca y o Y
sys:x:3:
tty:x:5:
....

RS@debian:~$ grep '\<c...h\>' /usr/share/dict/words    busca començant per c i acabant per h amb
                                                                                    un màxim de tres.
catch
cinch
cinch's
....
RS@debian:~$ grep '\<c.*h\>' /usr/share/dict/words     busca començant per c i acabant per h totes.
caddish
calabash
...
```

6.2.7. Filtres: Awk

Awk, o també la implementació de GNU *gawk*, és una ordre que accepta un llenguatge de programació dissenyat per a processar dades basades en text, ja siguin fitxers o fluxos de dades, i ha estat la inspiració de Larry Wall per a escriure Perl. La seva sintaxi més comuna és `awk 'programa' arxius...` i programa pot ser: **patró {acció} patró {acció}...** *awk* llegeix l'entrada d'arxius una línia alhora. Cada línia es compara amb cada patró en ordre; per a cada patró que concordi amb la línia s'efectua l'acció corresponent. Un exemple pregunta pel primer camp si és *root* de cada línia de l'arxiu */etc/passwd* i la imprimeix considerant com a separador de camps el ":" amb -F:; en el segon exemple reemplaça el primer camp per *root* i imprimeix el resultat canviat (a la primera ordre utilitzem '=='; a la segona, només un '=').

```
RS@debian:~$ awk -F: '$1=="root" {print}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
RS@debian:~$ awk -F: '$1="root" {print}' /etc/passwd
root x 0 0 root /root /bin/bash
root x 1 1 daemon /usr/sbin /bin/sh
root x 2 2 bin /bin /bin/sh
root x 3 3 sys /dev /bin/sh
root x 4 65534 sync /bin /bin/sync
root x 5 60 games /usr/games /bin/sh
root x 6 12 man /var/cache/man /bin/sh
```

L'*awk* divideix automàticament l'entrada de línies en camps, és a dir, una cadena de caràcters que no siguin blancs separats per blancs o tabuladors; per exemple, el `who` té 5 camps i `awk` ens permetrà filtrar cada un d'aquests camps.

```
RS@debian:~$ who
RS tty7 2010-04-17 05:49 (:0)
RS pts/0 2010-04-17 05:50 (:0.0)
RS pts/1 2010-04-17 16:46 (:0.0)
remo@debian:~$ who | awk '{print $4}'
05:49
05:50
16:46
```

L'*awk* anomena aquests camps $1 $2... $NF, en què NF és una variable igual que el nombre de camps (en aquest cas NF = 5). Per exemple:

```
ls -al | awk '{print NR, $0}    Agrega nombre d'entrades (la variable NR compta el nombre de línies,
                                          $0 és la línia sencera.
ls -al | awk '{printf "%d %s\n", NR, $9}'   %d significa un nombre decimal (NR) i %s un
                                                   string ($9) i  un new line
awk -F: '$2 == "" ' /etc/passwd      donarà els usuaris que en l'arxiu passwd no tinguin posada
                                                   la contrasenya
```

El patró es pot escriure de diverses maneres:

```
$2 == ""      si el segon camp és buit
$2 ~ /^$/     si el segon camp coincideix amb la cadena buida
$2 !~ /./      si el segon camp no concorda amb cap caràcter (! és negació)
length($2) ==     si la longitud del segon camp és 0, length és una funció interna de l'awk
~               indica coincidència amb una expressió
!~              significa el contrari (sense coincidència)
NF % 2 != 0        mostra la ratlla només si hi ha un nombre parell de camps
awk 'lenght ($0) 32 {print "Línia", NR, "llarga", substr($0,1,30)} ' /etc/passwd       avalua las
                     línies de /etc/passwd i genera un substring</i>
```

Hi ha dos patrons especials, BEGIN i END. Les accions BEGIN es fan abans que la primera línia s'hagi llegit; es pot usar per a inicialitzar les variables, imprimir encapçalaments o posicionar separadors d'un camp assignant-los a la variable FS, per exemple:

```
awk 'BEGIN {FS = ":"} $2 == "" ' /etc/passwd         igual que l'exemple d'abans
awk 'END { print NR } ' /etc/passwd         imprimeix el nombre de línies processades al final de la
                                                        lectura de l'última ratlla.
```

L'`awk` permet operacions numèriques en forma de columnes, per exemple, per a sumar tots els nombres de la primera columna:

```
{ s=s + 1} END { print s}          i per a la suma i la mitjana END { print s, s/NR}
```

Les variables s'inicialitzen a zero en declarar-se, i es declaren en utilitzar-se, per la qual cosa resulta molt simple (els operadors són els mateixos que a C):

```
{ s+=1} END {print s}                  Per exemple, per a comptar línies, paraules i caràcters:
{ nc += length($0) +1 nw +=NF } END {print NR, nw, nc}
```

Hi ha variables predefinides en l'*awk:*

- FILENAME nom de l'arxiu actual.
- FS caràcter delimitador del camp.
- NF nombre de camps del registre d'entrada.
- NR número del registre de l'entrada.
- OFMT format de sortida per a nombres (%g per defecte).
- OFS cadena separadora de camp a la sortida (blanc per defecte).
- ORS cadena separadora de registre de sortida (*new line* per defecte).
- RS cadena separadora de registre d'entrada (*new line* per defecte).

Operadors (ídem C):

= += -= *= /= %= || && ! >> >>= << <<= == != ~ !~
+ - * / %
++ --

Funcions predefinides a *awk:*

cos(), exp(), getline(), index(), int(), length(), log(), sin(), split(), sprintf(), substr().

A més suporta dins d'acció sentències de control amb la sintaxi similar a C:

```
if (condició)
proposició1
else
proposició2

for (exp1;condició;exp2)

while (condició) {
proposició
expr2
}
continue     continua, avalua la condició novament
break          trenca la condició
next            llegeix la línia d'entrada següent
```

```
exit             salta a END
```

També l'`awk` maneja arranjaments, per exemple, per a fer un `head` de */etc/passwd*:

```
awk '{ line[NR] = $0} \
END { for (i=NR; i>2; i--) print line[i]} ' /etc/passwd
```

Un altre exemple:

```
awk 'BEGIN { print "Usuari UID Shell\n------- --- -----" } $3 >= 500 { print $1, $3, $7 | \
"sort -r"}' FS=":" /etc/passwd
Usuari UID Shell
------- --- -----
RS 1001 /bin/bash
nobody 65534 /bin/sh
debian 1000 /bin/bash

RS@debian:~$ ls -al | awk '
BEGIN { print "File\t\t\tOwner" }
{ print $8, "\t\t\t", \
$3}
END { print "done"}
'
File Owner
. RS
.. root
.bash_history RS
.bash_logout RS
.bashrc RS
.config RS
...
```

6.2.8. Exemples complementaris

1) Un exemple complet per a esborrar els registres (esborra */var/log/wtmp* i es queda amb cinquanta línies –o aquelles que passi l'usuari en línia d'ordres– de */var/log/messages*)

```
#!/bin/bash
#Variables
LOG_DIR=/var/log
ROOT_UID=0      # només el pot executar el root
LINES=50            # Línies por defecte.
E_XCD=86           # No em puc canviar de directori
E_NOTROOT=87 # Sortida de no-root error.
```

```
if [ "$UID" -ne "$ROOT_UID" ]   # Sóc root?
      then
      echo "Heu de ser root. Em sap greu."
      exit $E_NOTROOT
      fi

if [ -n "$1" ]            # Nombre de línies per preservar?
      then
      lines=$1
      else
      lines=$LINES   # valor per defecte.
      fi

cd $LOG_DIR

if [ `pwd` != "$LOG_DIR" ]
# també pot ser if [ "$PWD" != "$LOG_DIR" ]
      then
      echo "No puc anar a $LOG_DIR."
      exit $E_XCD
      fi #

# Una altra manera de fer-ho seria:
# cd /var/log || {
# echo "No puc !" >&2
# exit $E_XCD;
# }

tail -n $lines messages > mesg.temp     # Deso temporalment
mv mesg.temp messages                      # Moc.

cat /dev/null > wtmp                              #Esborro wtmp.
echo "Registres esborrats."
exit 0
# Un zero indica que tot ha anat bé.
```

2) Utilització de l'*expr*.

```
#!/bin/bash
echo "Aritmètics"
echo
a=`expr 5 + 3`
echo "5 + 3 = $a"
a=`expr $a + 1`
echo
echo "a + 1 = $a"
echo "(increment)"
```

```
a=`expr 5 % 3`
# mòdul
echo
echo "5 mod 3 = $a"
# Lògics
# 1 si true, 0 si false,
#+ oposat a normal Bash convention.

echo "Lògics"
x=24
y=25
b=`expr $x = $y` # per igual.
echo "b = $b" # 0 ( $x -ne $y )
a=3
b=`expr $a \> 10`
echo 'b=`expr $a \> 10`, per la qual cosa...'
echo "If a > 10, b = 0 (false)"
echo "b = $b" # 0 ( 3 ! -gt 10 )
b=`expr $a \< 10`
echo "If a < 10, b = 1 (true)"
echo "b = $b" # 1 ( 3 -lt 10 )
echo
# Compte amb els operadors d'escapada \.
b=`expr $a \<= 3`
echo "If a <= 3, b = 1 (true)"
echo "b = $b" # 1 ( 3 -le 3 )
# S'utilitza un operador "\>=" operator (greater than or equal to).

echo "String"
echo
a=1234zipper43231
echo "String base \"$a\"."
b=`expr length $a`
echo "Long. de \"$a\" es $b."
# index: posició del primer caràcter que satisfà en la cadena
#
b=`expr index $a 23`
echo "Posició numèrica del \"2\" en \"$a\" és \"$b\"."
# substr: extract substring, starting position & length specified
b=`expr substr $a 2 6`
echo "Substring de \"$a\", començant a 2,\
i de 6 chars és \"$b\"."
# Using Regular Expressions ...
b=`expr match "$a" '[0-9]*'` # comptador numèric.
echo Nombre de dígits de \"$a\" és $b.
b=`expr match "$a" '\([0-9]*\)'` # compte amb els caràcters d'escapada
echo "Els dígits de \"$a\" són \"$b\"."
```

```
exit 0
```

3) Un exemple que mostra diferents variables (amb lectura des del teclat), sentències i execució d'ordres:

```
#!/bin/bash
echo -n "Introduir el % de CPU del procés que més consumeix a supervisar, l'interval[s],
         Nre. repeticions [0=sempre]: "
read lim t in
while true
do
           A=`ps -e -o pcpu,pid,comm | sort -k1 -n -r | head -1`
           load=`echo $A | awk '{ print $1 }'`
           load=${load%.*}
           pid=`echo $A | awk '{print $2 }'`

           if [ $load -gt $lim ]
           then
              # verifico cada t segons la CPU load
              sleep $t
              B=`ps -e -o pcpu,pid,comm | sort -k1 -n -r | head -1`
              load2=`echo $B | awk '{ print $1 }'`
              load2=${load2%.*}
              pid2=`echo $B | awk '{print $2 }'`
              name2=`echo $B | awk '{print $3 }'`
              [ $load2 -gt $lim ] && [ $pid = $pid2 ] && echo $load2, $pid2, $name2
           let in--
              if [ $in == 0 ]; then  echo "Fi"; exit 0; fi
fi
done
```

4) Moltes vegades en els *shell scripts* és necessari fer un menú perquè l'usuari esculli una opció. En el primer codi s'ha de fer la instal·lació del paquet `dialog` `(apt-get install dialog)`.

a. Menú amb l'ordre `dialog`:

```
#!/bin/bash
# original http://serverfault.com/questions/144939/multi-select-menu-in-bash-script

cmd=(dialog --separate-output --checklist "Select options:" 22 76 16)
options=(1 "Opció 1" off    # Si se'n desitja qualsevol
                            # es pot posar en on
         2 "Opció 2" off
         3 "Opció 3" off
         4 "Opció 4" off)
choices=$("${cmd[@]}" "${options[@]}" 2>&1 >/dev/tty)
```

```
clear
for choice in $choices
do
    case $choice in
        1)
            echo "Primera Opció"
            ;;
        2)
            echo "Segona Opció"
            ;;
        3)
            echo "Tercera Opció"
            ;;
        4)
            echo "Quarta Opció"
            ;;
    esac
done
```

b. Menú només amb sentències:

```
#!/bin/bash
# Basat en original de D. Williamson
# http://serverfault.com/questions/144939/multi-select-menu-in-bash-script

toogle () {
    local choice=$1
    if [[ ${opts[choice]} ]]
    then
        opts[choice]=
    else
        opts[choice]="<-"
    fi
}

PS3='Seleccionar opció: '
while :
do
    clear
    options=(Opció A ${opts[1]}" "Opció B ${opts[2]}" "Opció C ${opts[3]}" "Sortir")
    select opt in "${options[@]}"
    do
        case $opt in
            "Opció A ${opts[1]}")
                toogle 1
                break
                ;;
```

```
            "Opció B ${opts[2]}")
                 toogle 2
                 break
                 ;;
            "Opció C ${opts[3]}")
                 toogle 3
                 break
                 ;;
             "Sortir")
                 break 2
                 ;;
             *) printf '%s\n' 'Opció no vàlida';;
        esac
    done
done

printf '%s\n' 'Opcions seleccionades:'
for opt in "${!opts[@]}"
do
    if [[ ${opts[opt]} ]]
    then
        printf '%s\n' "Opció $opt"
    fi
done
```

5) De vegades és necessari filtrar una sèrie d'arguments passats en la línia d'ordres i `getop` o `getopts` poden ajudar. Les ordres `getopt` i `getopts` s'utilitzen per a processar i validar arguments d'un *shell script* i són similars però no idèntics. A més, la funcionalitat pot variar en funció de la seva implementació per la qual cosa és necessari llegir les pàgines del manual amb atenció. En el primer exemple utilitzarem `getopts` i filtrarà `nombre_cmd -a valor`:

```
#!/bin/bash
# Més opcions a
while getopts ":a:" opt; do
   case $opt in
   a)
      echo "-a va ser introduït, Paràmetre: $OPTARG" >&2
      ;;
   \?)
      echo "opció invàlida: -$OPTARG" >&2
      exit 1
      ;;
   :)
      echo "Opció -$OPTARG requereix un argument." >&2
      exit 1
      ;;
```

```
   esac
done
```

I amb `getopt`:

```
#!/bin/bash
args=`getopt abc: $*`
if test $? != 0
   then
      echo 'Ús: -a -b -c valor'
      exit 1
fi
set -- $args
for i
do
   case "$i" in
        -c) shift;echo "Opció c com a argument amb $1";shift;;
         -a) shift;echo "Opció a com a argument";;
         -b) shift;echo "Opció b com a argument";;
   esac
done
```

Activitats

1. Analitzeu les ordres següents i els seus paràmetres principals:

`7z`, `awk`, `apt-cache`, `at`, `apt-get`, `bg`, `cd`, `chmod`, `chown`, `chgrp`, `cal`, `cut`, `cat`, `cp`, `crontab`, `curl`, `crontab`, `chmod`, `chown`, `chgrp`, `compress`, `clear`, `date`, `df`, `du`, `dpkg`, `dc`, `dig`, `diff`, `env`, `expr`, `exit`, `fold`, `fg`, `ftp`, `free`, `file`, `find`, `gzip`, `grep`, `head`, `id`, `info`, `kill`, `less`, `last`, `ln`, `lp`, `ls`, `mesg`, `man`, `mail`, `mkdir`, `more`, `mv`, `mount`, `nice`, `nohup`, `ps`, `passwd`, `pwd`, `page`, `pstree`, `ping`, `quota`, `sed`, `ssh`, `sleep`, `sort`, `scp`, `su`, `sudo`, `rm`, `rmdir`, `tail`, `tar`, `tee`, `telnet`, `test`, `tail`, `tr`, `touch`, `top`, `users`, `uptime`, `uname`, `vi`, `whereis`, `which`, `whois`, `wget`, `who`, `w`, `wc`, `write`, `zip`.

2. Instal·leu VirtualVox sobre un sistema operatiu hoste del qual disposeu i carregueu una imatge ja configurada (per exemple, des de http://virtualboxes.org/images/), tot verificant que el sistema funcioni (incloent-hi sistema gràfic, xarxa, accés a disc de l'hoste, etc.).

3. Repetiu el punt anterior instal·lant el sistema des dels CD o DVD enviats per la UOC.

4. Creeu un script interactiu que calculi el nombre de dies entre dues dates introduïdes per l'usuari segons el format dia/mes/any. També haureu de calcular el nombre d'hores si l'usuari introdueix hora d'inici i hora de final, en el format hh:mm. Si introdueix només una hora, es calcularà fins a les 23:59 del dia donat com a final del període per calcular.

5. Creeu un script que es faci una còpia (recursiva) dels arxius a /etc, de manera que un administrador del sistema pugui modificar els arxius d'inici sense temor.

6. Escriviu un script anomenat *homebackup* que automatitzi el **tar** amb les opcions correctes i en el directori */var/backups* per a fer una còpia de seguretat del directori principal de l'usuari amb les condicions següents:

a) Fer un test del nombre d'arguments. L'script s'ha d'executar sense arguments, i si n'hi ha, ha d'imprimir un missatge d'ús.
b) Determinar si el directori de còpies de seguretat té suficient espai lliure per a emmagatzemar la còpia de seguretat.
c) Preguntar a l'usuari si vol una còpia completa (tot) o una còpia incremental (només els arxius que hagin canviat). Si l'usuari no té una còpia de seguretat completa, s'haurà de fer, i en cas d'una còpia de seguretat incremental, només es pot fer si la còpia de seguretat completa no té més d'una setmana.
d) Comprimir la còpia de seguretat utilitzant qualsevol eina de compressió.
e) Informar a l'usuari que es farà en cada moment, ja que això pot trigar algun temps i així s'evitarà que l'usuari es posi nerviós si la sortida no apareix a la pantalla.
f) Imprimir un missatge que informi l'usuari sobre la mida de la còpia de seguretat sense comprimir i comprimida, el nombre d'arxius desats o actualitzats i el nombre de directoris desats o actualitzats.

7. Escriviu un script que executi un navegador web simple (en mode text), utilitzant `wget` i `links -dump` per a mostrar les pàgines HTML en un terminal.

L'usuari té 3 opcions: introduir una adreça URL, entrar **b** per a retrocedir (*back*) i **q** per a sortir. Les 10 últimes URL introduïdes per l'usuari s'emmagatzemen en una matriu, des d'on l'usuari pot restaurar l'URL utilitzant la funcionalitat **b** (*back*).

Bibliografia

URL visitades en 2016

"ALSA" http://www.alsa-project.org/main/index.php/Main_Page.

Barnett, Bruce. "AWK" http://www.grymoire.com/Unix/Awk.html. © General Electric Company.

"Bash Reference Manual" http://www.gnu.org/software/bash/manual/bashref.html.

Comer, Douglas (2001). *TCP/IP Principios básicos, protocolos y arquitectura*. Prentice Hall.

"¿Cómo elegir un passwd seguro?" http://en.wikipedia.org/wiki/Password_strength.

Comunidad Debian. "Distribución Debian". http://www.debian.org

Configuración Xorg en Debian https://wiki.debian.org/Xorg.

Cooper, M. (2006). "Advanced bash Scripting Guide". *The Linux Documentation Project* (guías). http://tldp.org/LDP/abs/html/index.html

"CUPS" http://www.cups.org/.

Drivers de targetas gráficas http://www.calel.org/pci-devices/xorg-device-list.html.

"El libro del administrador de Debian" http://debian-handbook.info/browse/es-ES/stable/.

The Fedora Project. http://fedoraproject.org.

FHS. "Filesystem Hierarchy Standard (FHS)" https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard; http://www.pathname.com/fhs.

"Firmware ipw2x00" https://wiki.debian.org/ipw2200.

Garrels, Machtelt. "Bash Guide for Beginners" http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html.

GNU. "GNU Operating System (Software)" http://www.gnu.org/software.

"The GNU awk programming language" http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_06.html.

"Guía de instalación Debian Squeeze" http://man-es.debianchile.org/index.html. Gran part de l'explicació també serveix per a Wheezy.

HispaLinux. "Comunidad Hispana de Linux" http://www.hispalinux.es.

"How to Resize Windows Partitions" https://help.ubuntu.com/community/HowtoResizeWindowsPartitions.

Koehntopp, K. "Linux Partition HOWTO". *The Linux Documentation Project.*

Mike, G. "Programación en BASH - COMO de introducción" http://es.tldp.org/COMO-INSFLUG/COMOs/Bash-Prog-Intro-COMO/.

MIPS i3-370M http://www.notebookcheck.net/Intel-Core-i3-370M-Notebook-Processor.32767.0.html.

[Mou01] Mourani, Gerhard (2001). *Securing and Optimizing Linux: The Ultimate Solution. Open Network Architecture, Inc*. http://www.tldp.org/LDP/solrhe/Securing-Optimizing-Linux-The-Ultimate-Solution-v2.0.pdf.

"Network Manager en Debian 7 Wheezy" https://wiki.debian.org/NetworkManager.

"NPT" http://support.ntp.org/bin/view/Support/GettingStarted; https://wiki.debian.org/es/NTP.

Pritchard, S. "Linux Hardware HOWTO". *The Linux Documentation Project.*

Quigley, E. (2001). *Linux shells by Example*. Prentice Hall.

Robbins, Arnold. "UNIX in a Nutshell" (3.ª ed.) http://shop.oreilly.com/product/9781565924277.do (cap. 11)

"The Shell Scripting Tutorial" http://bash.cyberciti.biz/guide/Main_Page.

"System Printing" https://wiki.debian.org/SystemPrinting.

"TLDP guides" http://www.tldp.org/guides.html.

Ubuntu. Distribución Ubuntu http://www.ubuntu.com.

van Vugt, Sander. "Beginning Ubuntu Server Administration: From Novice to Professional" (Disponible a Safari Books).

Wells, M. i altres. *Running Linux*. O'Reilly.

"Xming X Server" http://www.straightrunning.com/XmingNotes.