ADMINISTRACIÓ DE SISTEMES OPERATIUS

1. Llenguatges de guions de shell

En l'àmbit de la informàtica, un programa és una seqüència d'instruccions que un ordinador ha d'executar per dur a terme una tasca determinada. El llenguatge de programació determina la forma amb què el programador ha d'escriure les operacions que l'ordinador ha de realitzar. De llenguatges de programació n'existeixen centenars, i cada any en sorgeixen de nous o versions millorades dels existents, que n'amplien les característiques per adaptar-se a les necessitats tecnològiques de cada moment.

Els intèrprets d'ordres o *shells* són programes que permeten la interacció dels usuaris amb el sistema operatiu i, més enllà d'aquesta funció, també incorporen llenguatges de programació que permeten crear programes que anomenem *guions*. Els guions de *shell* són molt útils per fer tasques d'administració del sistema i altres treballs repetitius que no requereixen un llenguatge de programació més sofisticat.

1.1. Conceptes previs

Abans d'abordar l'estudi dels llenguatges de guions de *shell*, fem un repàs d'alguns conceptes previs.

1.1.1. Llenguatges de programació

Un llenguatge de programació és un llenguatge informàtic usat per controlar el comportament d'una màquina, normalment un ordinador. Cada llenguatge té una sèrie de regles sintàctiques i semàntiques estrictes que cal seguir per escriure un programa informàtic, i que en descriuen l'estructura i el significat respectivament. Aquestes regles permeten especificar les dades amb què treballarà el programa i les accions que realitzarà.

El **llenguatge de màquina** o codi màquina és l'únic llenguatge de programació que poden entendre directament els circuits microprogramables de l'ordinador, com ara el microprocessador. El codi màquina està format exclusivament per codi binari, és a dir, per zeros (0) i uns (1). Un programa escrit en codi màquina consisteix en un seguit d'instruccions i dades codificats en binari. El llenguatge de màquina és específic de cada màquina, malgrat que el conjunt d'instruccions disponibles pugui ser similar.

Codi binari

Els circuits interns de l'ordinador treballen amb dos nivells de tensió que de manera abstracta representem amb el 0 i l'1. Per això el llenguatge màquina només utilitza aquests dos símbols.

Els llenguatges de programació se solen classificar principalment en **llenguatges de nivell baix**, que són molt propers al codi màquina utilitzat internament per un tipus d'ordinador determinat, i

ADMINISTRACIÓ DE SISTEMES OPERATIUS

Un llenguatge de programació d'alt nivell és un llenguatge que, en comparació amb els llenguatges de nivell baix, ofereix un nivell d'abstracció més alt, és més fàcil d'utilitzar i més portable entre plataformes de maquinari. L'objectiu d'aquests llenguatges de programació és alliberar als programadors de tasques complexes i augmentar la productivitat i l'eficiència en la generació de codi. Per posar un exemple, els llenguatges de programació de nivell alt no s'encarreguen directament de la gestió dels mapes de memòria. En canvi, en un programa escrit en un llenguatge de nivell baix, les dades utilitzades són referenciades per la seva posició en memòria, és a dir, és responsabilitat del programador controlar el mapa de memòria i l'assignació de memòria a cada dada.

Avui dia existeix una gran quantitat de llenguatges de programació de nivell alt. Es tracta d'un grup molt heterogeni amb una gran diversitat de característiques i objectius. Alguns exemples de llenguatges de nivell alt, més o menys especialitzats en diferents tasques, són Java, PHP, C, C++, Python, Visual Basic .NET, C#, Ruby, Cobol, Perl, JavaScript, etc.

Llenguatges de programació

La diferència entre llenguatges de nivell baix i alt es fa evident comparant dos programes que escriuen "Hola" a la pantalla, el primer usant llenguatge assemblador per màquines x86 (nivell baix) i el segon utilitzant el llenguatge de programació Python (nivell alt).

Programa en llenguatge assemblador

```
.MODEL SMALL
IDEAL
STACK 100H

DATASEG
HW DB 'Hola!$'

CODESEG
MOV AX, @data
MOV DS, AX
MOV DX, OFFSET HW
MOV AH, 09H
INT 21H
MOV AX, 4C00H
INT 21H
END
```

Programa en llenguatge d'alt nivell Python

```
print "Hola!"
```

1.1.2. Codi font

El **codi font** d'un programa es refereix a la sèrie d'instruccions escrites en algun llenguatge de programació llegible per l'home que conformen el programa. El codi font no és directament executable per l'ordinador, sinó que ha de ser traduït a llenguatge de màquina que sí podrà ser executat pel maquinari de l'ordinador. Per a aquesta traducció, depenent del tipus de llenguatge utilitzat, s'usen els anomenats compiladors i intèrprets.

Programari lliure i programari propietari

lo, modificar-lo o reutilitzar-lo lliurement. Quan es compleix aquest aspecte es parla de programari lliure, en contraposició al programari propietari (o privatiu), que, normalment, no va acompanyat del codi font ni de cap de les llibertats esmentades.

1.1.3. Llenguatges compilats i llenguatges interpretats

Els programes escrits en llenguatges de programació d'alt nivell no es poden executar directament a la màquina. És necessari executar un procés de traducció del llenguatge d'alt nivell a llenguatge màquina. Aquesta traducció pot ser una compilació (llenguatges compilats), una interpretació (llenguatges interpretats) o una combinació de les dues opcions anteriors (llenguatges híbrids).

La implementació d'un llenguatge de programació és la que proveeix una manera perquè s'executi un programa. Un **llenguatge compilat** es refereix a un llenguatge de programació que típicament s'implementa mitjançant un compilador.

Alguns exemples de llenguatges compilats són Fortran, C, C++, Ada, Cobol i Visual Basic.

Un **compilador** d'un determinat llenguatge de programació és un programa que llegeix un fitxer en codi font escrit en aquest llenguatge de programació i el converteix en una seqüència de codi màquina per a una plataforma determinada (Intel, Sparc, etc.). El codi traduït pot servir com a entrada d'un altre intèrpret o un altre compilador. Un compilador anomenat *de codi natiu* és el que tradueix el codi font d'un programa directament a codi màquina executable. Sovint, però, aquest procés se separa en més d'una fase, per exemple, en una generació de codi objecte i un enllaçat, tal com es veu a la figura 1.1.

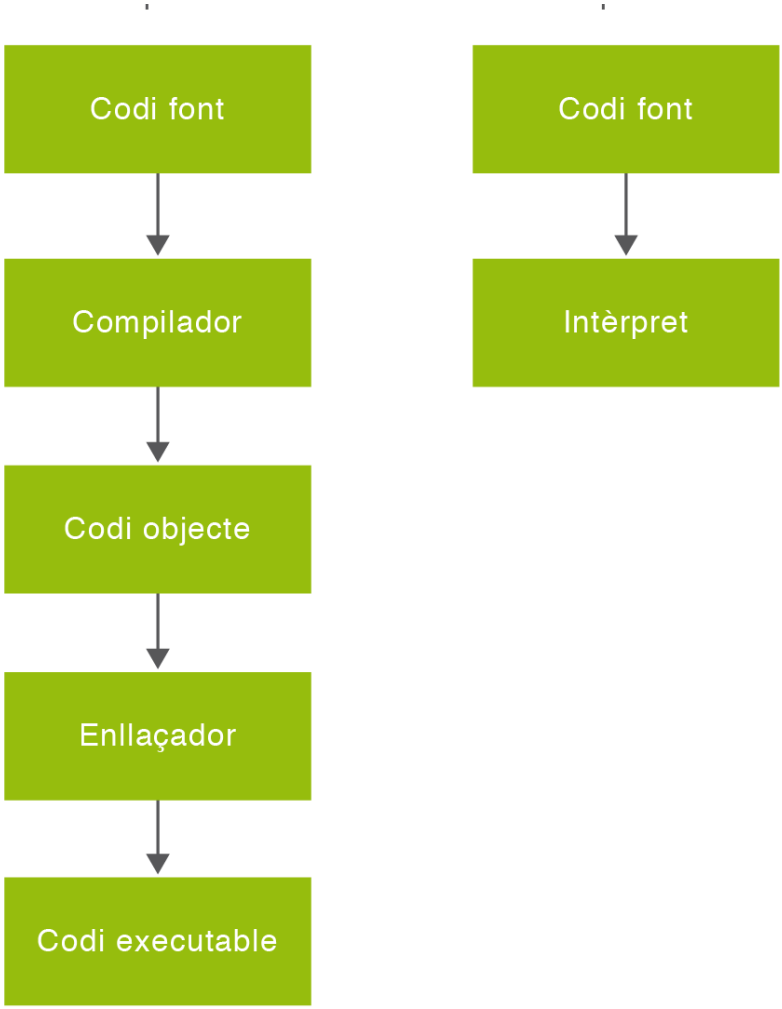
Alguns exemples de llenguatges interpretats són Perl, PHP, Python, Bash i JavaScript.

Un **llenguatge interpretat** és aquell en què les instruccions es tradueixen o interpreten una per una en temps d'execució a un llenguatge intermedi o llenguatge màquina.

Un **intèrpret** d'un llenguatge de programació és un programa que llegeix les ordres del codi font, en comprova la sintaxi i executa el codi relacionat amb aquestes ordres.

Els principals avantatges d'una implementació interpretada respecte d'una compilada són la seva independència respecte de la plataforma i que permet disminuir el cost de programació, en el sentit que permet desenvolupar i provar el programa més ràpidament. El desavantatge és que l'execució d'un programa interpretat normalment és més lenta que la d'un programa compilat i necessita més recursos (memòria, CPU, etc.).

Figura 1.1. Compilació i interpretació



1.1.4. Llenguatges de guions

Els **llenguatges de guions**, també coneguts amb el nom de **llenguatges d'*scripting***, són un tipus de llenguatges de programació d'alt nivell que poden ser de propòsit general o de propòsit específic i gairebé sempre són llenguatges interpretats. Els programes escrits amb aquests llenguatges s'anomenen **guions** o **scripts**.

Els llenguatges de guions tenen el seu origen en els llenguatges de control de tasques, concretament en el JCL (*job control language*), utilitzat als *mainframe*d'IBM per a la programació de treballs per lots.

Treball per lots

Es coneix com a treball per lots o batch job un tipus de programes que s'executen sense el control o supervisió directa de l'usuari.

El JCL és un llenguatge amb poques capacitats de programació i amb un propòsit molt específic. En canvi, molts dels llenguatges de guions actuals, com ara el Python o el Perl, són llenguatges de programació potents que permeten desenvolupar aplicacions de propòsit general.

web:

1. **Scripts de navegadors web**. S'utilitzen per ampliar les capacitats de l'HTML i per inserir accions en pàgines web. Permeten crear efectes especials i aporten interactivitat. Els scripts són interpretats i executats en la màquina client pel navegador web, el qual ha d'incorporar l'intèrpret del llenguatge. Un exemple de llenguatge d'aquest tipus molt utilitzat és JavaScript.
2. **Scripts de servidor**. Són programes que permeten donar funcionalitats a les pàgines web que no es poden resoldre només amb els scripts de navegador. Els scripts de servidor permeten dotar de certa "intel·ligència" els llocs web, la qual cosa fa que generin pàgines diferents segons les circumstàncies. Un dels llenguatges més utilitzats en aquest àmbit és PHP.

Llenguatge PHP

Alguns llenguatges de guions serveixen per a més d'un propòsit. Per exemple, encara que en el disseny del llenguatge PHP tot està orientat a facilitar la creació de llocs web, és possible crear aplicacions amb una interfície gràfica per a l'usuari, utilitzant l'extensió PHP-Qt o PHP-GTK. També pot ser usat des de la línia d'ordres amb el PHP-CLI (*command line interface*).

En l'àmbit dels sistemes operatius, els **intèrprets d'ordres** incorporen un tipus de llenguatges de guions que anomenem **llenguatges de guions de *shell***, que s'utilitzen amb diversos propòsits.

1.2. Intèrprets d'ordres o shells

Un intèrpret de línia d'ordres o simplement **intèrpret d'ordres**, és un programa informàtic que té la funció de llegir línies de text (ordres) escrites en un terminal o en un fitxer de text i interpretar-les.

Les ordres s'escriuen seguint la sintaxi incorporada per aquest intèrpret. En llegir l'ordre, l'intèrpret analitza la seqüència de caràcters i, si la sintaxi de l'ordre és correcta, passa l'ordre interpretada al sistema operatiu o al programa que representa (una sessió d'FTP, una sessió d'SSH, etc.) perquè l'executi.

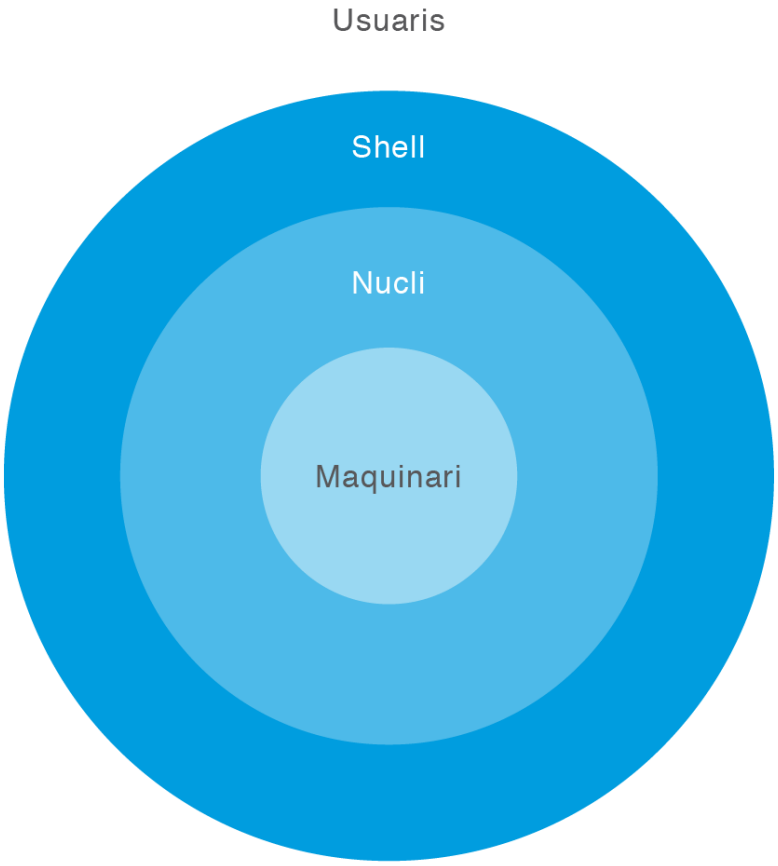
Dispositius de xarxa

Els dispositius de xarxa que poden ser configurats per programari normalment disposen d'una interfície de línia d'ordres. Per exemple, el sistema operatiu IOS de Cisco disposa d'una interfície de línia d'ordres que és utilitzada per executar ordres de configuració, supervisió i manteniment dels dispositius de Cisco, ja sigui amb una consola de l'encaminador (router), amb un terminal o a partir de mètodes d'accés remot.

Els intèrprets d'ordres dels sistemes operatius també es coneixen amb el nom de ***shell***. El *shell* és un programa encarregat de llegir les ordres que tecleja l'usuari i convertir-les en instruccions que el sistema operatiu pot executar.

Antigament la interacció dels usuaris amb el sistema operatiu es feia únicament mitjançant interfícies de línia d'ordres, atès que no existien ordinadors amb la capacitat de mostrar gràfics o imatges. El terme *shell*va aparèixer a la dècada dels 70 amb el sistema operatiu Unix, que va ser pioner en el concepte d'un entorn de línia d'ordres potent. La traducció de *shell* seria "embolcall o closca", perquè envolta la resta de capes del sistema a mode de closca. Tal com es pot apreciar a la figura 1.2, el *shell* és la capa més externa i actua com a interfície entre l'usuari i la resta del sistema.

Figura 1.2. Capes del sistema operatiu Unix

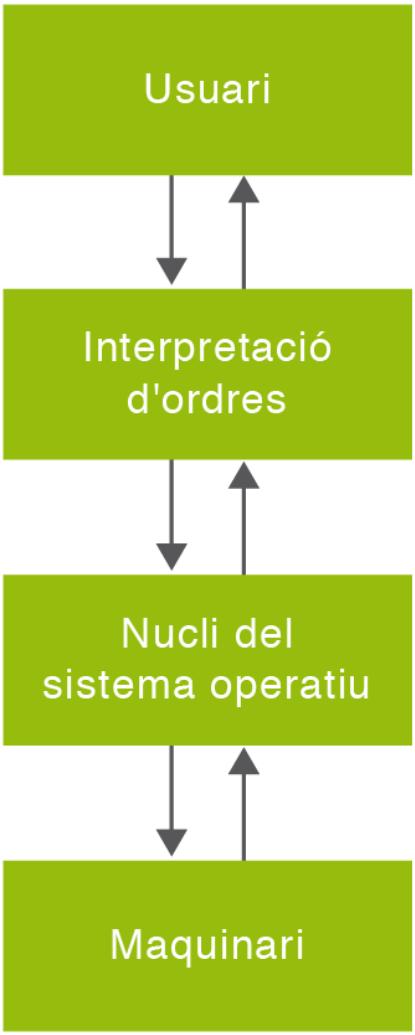


De vegades es confonen els termes d'intèrpret d'ordres i sistema operatiu i, de manera errònia, s'identifiquen l'un amb l'altre.

L'intèrpret d'ordres és el que els usuaris veuen del sistema i està escrit de la mateixa manera que un programa d'usuari. No està integrat en el nucli del sistema operatiu, sinó que s'executa com un programa més de l'usuari.

El sistema operatiu sempre està situat en un nivell inferior als programes, i l'intèrpret d'ordres o *shell* és un programa més que té la tasca de llegir les ordres que li dóna l'usuari, realitzar una serie

Figura 1.3. Interpretació d'ordres



En l'actualitat gairebé tots els sistemes operatius disposen d'interfícies gràfiques d'usuari (GUI, *graphical user interface*) que permeten interactuar amb el sistema d'una manera molt més senzilla que amb la línia d'ordres, cosa que facilita les tasques de l'usuari. Amb tot, les interfícies de línies d'ordres continuen sent una eina de treball molt utilitzada en l'àmbit de l'administració de sistemes i xarxes, especialment per a l'automatització de tasques.

1.2.1. Llenguatges de guions de shell

La majoria dels intèrprets d'ordres de tots els sistemes operatius compten amb un llenguatge de programació propi per escriure programes que anomenem de diverses maneres: *programes per lots*, *arxius d'ordres*, ***guions de shell*** o ***shell scripts***.

El terme script

i actrius (o, en el nostre cas, programes) seguint un ordre establert. En alguns textos es tradueix script com a guió i en d'altres s'utilitza l'expressió arxiu d'ordres.

Els **llenguatges de guions de *shell*** o de ***shell scripting*** són molt utilitzats en l'àmbit de l'administració de sistemes operatius, perquè permeten fer programes combinant crides a ordres del sistema, programes compilats, d'altres guions, etc. En principi són llenguatges pensats per a l'automatització de tasques en un sistema operatiu però realment la majoria permeten crear aplicacions de propòsit general.

Els intèrprets d'ordres interpreten guions escrits en el llenguatge que saben interpretar. Alguns intèrprets d'ordres també incorporen el motor intèrpret d'altres llenguatges a més del seu propi, la qual cosa permet l'execució d'scripts en aquests llenguatges directament al mateix intèrpret.

1.2.2. Automatització de tasques amb guions de shell

En general, no hi ha una manera única de fer les coses, però normalment algunes eines són més indicades per a determinats propòsits que d'altres.

> Els llenguatges de guions de *shell* ofereixen un entorn de programació ràpid per a l'automatització de tasques de manteniment i configuració del sistema.

Les principals raons que porten l'administrador del sistema a implementar l'automatització de tasques mitjançant la creació de *shell scripts* són les següents:

- Execució de tasques repetitives. Els administradors de sistemes sovint repeteixen seqüències d'ordres (com una alta massiva d'usuaris) o bé alguna línia d'ordres llarga i complexa. Fer un script proporciona una manera ràpida i segura d'executar aquestes seqüències o línies d'ordres.
- Planificació de tasques. Hi ha moltes tasques que han de dur-se a terme amb una base regular (fer còpies de seguretat, netejar fitxers de registre o fitxers temporals, etc.). Mitjançant la realització de guions de *shell* i el planificador de tasques es poden programar aquestes tasques perquè es duguin a terme de manera automàtica i periòdica.
    - Delegar tasques. Hi ha tasques de manteniment del sistema que es poden o que s'han de delegar a d'altres usuaris (operadors, administradors menys experimentats, etc.). El fet d'utilitzar *shell scripts* per implementar aquestes tasques proporciona una manera perquè els usuaris menys experts puguin aprendre, ja que en poden analitzar el codi i fins i tot mantenir-lo a mesura que el seu aprenentatge i habilitats progressen.
    - Personalització de l'inici de serveis. És habitual que en administrar un servidor haguem de dissenyar els nostres propis serveis per fer alguna tasca concreta. En la majoria de sistemes operatius (Unix, Windows, etc.), la realització de guions de *shell* ens permet l'automatització de l'inici i l'aturada de serveis.

> Vegeu la implementació de *shell scripts* per automatitzar tasques d'administració del sistema en l'apartat "Automatització de tasques del sistema" d'aquesta unitat.

ADMINISTRACIÓ DE SISTEMES OPERATIUS

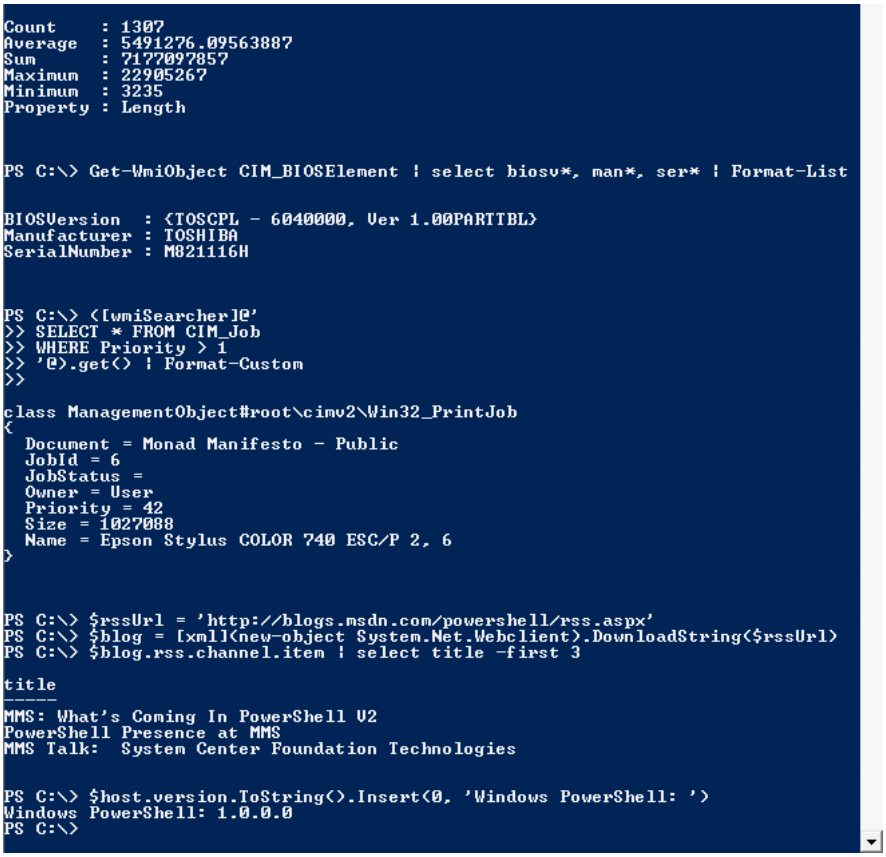
En els sistemes operatius Windows de Microsoft la interacció de l'usuari amb el sistema operatiu es fa generalment utilitzant la interfície gràfica integrada en el propi sistema operatiu, però també disposem d'intèrprets d'ordres que ens permeten interaccionar i crear guions per automatitzar tasques. Els intèrprets d'ordres existents per a aquests sistemes són els següents:

> Es pot accedir a l'intèrpret d'ordres cmd.exe dels sistemes Windows a *Inici* > *Executar* > *cmd*.

- **COMMAND.COM** és el nom de l'intèrpret d'ordres per a DOS i per a versions de Windows de 16/32 bits (95/98/98 SE/Me). Té dos modes d'execució, el mode interactiu, en el qual l'usuari escriu ordres per ser executades, i el mode per lots (*batch*), que executa una seqüència predefinida d'ordres guardada en un arxiu de text amb l'extensió *.bat* i que se sol anomenar *programa per lots*.
- L'executable **cmd.exe** és l'intèrpret d'ordres dels sistemes basats en Windows NT (incloent Windows 2000, XP, Server 2003, Vista i 7). És l'equivalent de COMMAND.COM a MS-DOS i sistemes de la família Windows 9x. En realitat, cmd.exe és un programa de Windows que actua com un intèrpret d'ordres de tipus DOS. En general és compatible, però proporciona extensions que eliminen algunes de les limitacions de COMMAND.COM.
- Windows **PowerShell** és una interfície de consola per a sistemes operatius Windows llançada el 2006 que té com a utilitat principal l'automatització de tasques administratives. Les funcions d'interpretació i de programació de guions estan molt millorades respecte a cmd.exe. Aquesta interfície no està instal·lada per defecte en el sistema i requereix de la instal·lació prèvia del *framework* (entorn de treball) .NET versió 2.0 per al seu funcionament. Permet interactuar amb el sistema operatiu i amb programes de Microsoft com SQL Server, Exchange o IIS. La característica distintiva d'aquest intèrpret d'ordres respecte als tradicionals és que està **orientat a objectes**. La figura 1.4 mostra una imatge d'una finestra de Windows PowerShell.

> Vegeu més informació sobre el llenguatge de guions Windows PowerShell a la pàgina oficial de Microsoft "Scripting with Windows PowerShell", que trobareu a la secció d'enllaços d'interès dels materials web del mòdul.

Figura 1.4. Finestra de Windows PowerShell



1.2.4. Shells del sistema operatiu Unix i derivats

Un sistema operatiu de tipus Unix o derivat és aquell que comparteix moltes de les característiques del sistema operatiu Unix, que va ser escrit al 1969 per, entre d'altres, Ken Thompson, Dennis Ritchie i Douglas McIlroy als Laboratoris Bell. En l'actualitat hi ha molts sistemes operatius de tipus Unix, com ara totes les distribucions GNU/Linux o el sistema operatiu Mac OS X.

Sistemes operatius de tipus Unix

Una distribució GNU/Linux o simplement distribució Linux és un sistema operatiu de tipus Unix que consisteix en el nucli de Linux, llibreries i utilitats del projecte GNU, i un conjunt de programari de tercers que permet afegir una sèrie d'aplicacions d'ús molt ampli, com ara escriptori gràfic, ofimàtica, navegadors, etc. Tant el nucli com la majoria de programari que formen les distribucions Linux són de codi lliure. Existeixen centenars de distribucions, en constant revisió i desenvolupament. Alguns dels noms més coneguts són Fedora (Red Hat), openSUSE (Novell), Ubuntu (Canonical Ltd), Mandriva Linux (Mandriva) i les distribucions Debian i Gentoo les quals tenen un model de desenvolupament independent d'empreses i estan creades pels seus propis usuaris.

Mac OS X és el sistema operatiu de codi propietari (amb part de codi obert) basat en Unix que utilitzen els ordinadors Macintosh de la companyia Apple Inc. A la X que acompanya el nom Mac OS se li atribueixen dos significats: el de numeral romà 10 (ja que les versions del Mac OS anomenat "Classic" acaben al 9) i la darrera lletra del mot Unix.

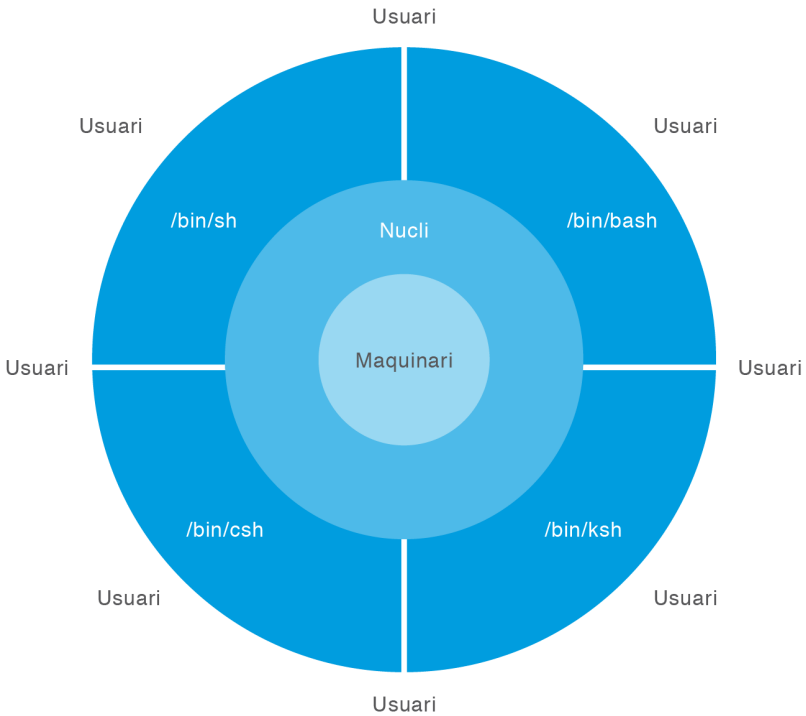
En els sistemes Unix i derivats, la interacció de l'usuari amb el sistema operatiu es pot fer amb mode gràfic o amb mode text. En aquests tipus de sistemes hi ha una gran varietat d'intèrprets

Taula: 1.1. Alguns dels shells disponibles als sistemes de tipus Unix

Nom del *shell*	**Pro-grama**	**Descripció i característiques**
Bourne	/bin/sh	És un dels primers *shells* que es van desenvolupar i és un dels intèrprets més coneguts. Va ser escrit per Steve Bourne als Laboratoris Bell i es va convertir en un estàndard de facto.
Almquist	/bin/ash	Va ser escrit com una substitució del *shell* Bourne però amb llicència BSD.
Bash	/bin/bash	Basat en el *shell* Bourne, va ser escrit com a part del projecte GNU. És el *shell* estàndard de la majoria de sistemes GNU/Linux i de Mac OS X.
Debian Almquist	/bin/dash	Una substitució del *shell* ash per a les distribucions Debian i basades en Debian, com ara Ubuntu.
C	/bin/csh	Es tracta del segon *shell* d'Unix. Fou dissenyat per facilitar l'ús interactiu afegint-hi noves funcions. La seva sintaxi és molt semblant al llenguatge C de programació i el seu ús és més comú en els sistemes Unix de Berkeley.
TENEX C	/bin/tcsh	Essencialment és el *shell* C amb alguna característica millorada de la línia d'ordres.
Korn	/bin/ksh	És el tercer *shell* d'Unix, fruit de l'evolució del *shell* Bourne amb característiques del *shell* C. És un *shell* estàndard del sistema Unix System V.
Z	/bin/zsh	És considerat el *shell* més complet, en el sentit que és el que té més funcionalitats. Engloba característiques d'sh, ash, bash, csh, ksh i tcsh.

Un dels aspectes que caracteritza un sistema operatiu de tipus Unix és el fet que és multiusuari, és a dir, permet l'ús del mateix ordinador a més d'un usuari al mateix temps (per exemple, a través d'SSH). Per a cada sessió d'usuari, el sistema genera un *shell* propi per a ell que actua com a intèrpret interactiu de les ordres que l'usuari executa, la qual cosa permet que els usuaris puguin triar un *shell* o un altre d'acord amb les seves necessitats. La figura 1.5 mostra gràficament l'existència de diferents *shells* en el diagrama de capes d'Unix.

Figura 1.5. Diagrama de capes d'Unix amb diferents shells



El fet que el *shell* estigui separat de la resta del sistema operatiu proporciona flexibilitat, perquè permet seleccionar la interfície més adequada a les necessitats de cada usuari.

El fitxer `/etc/shells` dóna informació sobre els *shells* vàlids per iniciar sessió en un sistema Unix determinat. Que siguin vàlids no significa que estiguin instal·lats, si es volen utilitzar cal que prèviament ens assegurem que estan instal·lats al sistema.

Exemple de contingut del fitxer /etc/shells d'un sistema Debian

```
# /etc/shells: valid login shells
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/esh
/bin/dash
/bin/bash
/bin/rbash
```

El *shell* que s'executa per defecte quan un usuari inicia sessió queda definit en el fitxer `/etc/passwd`, en el darrer camp de la línia corresponent a l'usuari.

Exemple de línia d'un fitxer /etc/passwd

La següent és una línia d'una usuària anomenada "mia" del fitxer `/etc/passwd`. En el darrer camp hi ha escrit `/bin/bash`, cosa que indica el shell per defecte de la usuària:

```
mia:x:1011:1000:Mia Maya:/home/mia:/bin/bash
```

Normalment, l'administrador del sistema és l'encarregat de fixar el *shell* d'inici de sessió dels usuaris. En alguns entorns, està habilitat l'ús d'ordres com `chsh` per permetre als usuaris canviar el seu *shell* per defecte d'inici de sessió.

Exemple d'utilització de l'ordre /usr/bin/chsh

Imaginem que la usuària "mia" està treballant en un entorn en què se li permet fer el canvi del seu shell d'inici i que el shell /bin/tcsh està instal·lat al sistema. La usuària té com a shell d'inici el bash i té la necessitat de canviar-lo pel tcsh. Per això executa l'ordre següent:

```
chsh -s /bin/tcsh
```

El resultat de l'ordre anterior és un canvi en la línia del fitxer `/etc/passwd` corresponent a la usuària "mia". En el darrer camp ara hi haurà escrit `/bin/tcsh` i a partir d'aquell

Si hem iniciat sessió com a usuaris amb un *shell* determinat, en la mateixa sessió podem arrencar un nou *shell* –al qual anomenem ***subshell*** o ***shell* fill**–, que pot ser del mateix tipus o d'un de diferent disponible en el sistema. Per iniciar un *subshell* senzillament hem d'escriure el nom del programa (sh, bash, csh, ksh, etc.) a la línia d'ordres i prémer la tecla de retorn. Si hem canviat de *shell*, en general, apareixerà un indicador de línia d'ordres diferent, ja que cada *shell* té una aparença diferent. Podem finalitzar l'execució del *subshell* i retornar al *shell* pare escrivint l'ordre *exit*.

1.3. El shell Bash

El *shell* **Bash** és l'intèrpret d'ordres predeterminat de gairebé totes les distribucions GNU/Linux, així com de Mac OS X, i pot executar-se en la majoria dels sistemes operatius tipus Unix. També s'ha portat a Microsoft Windows per al projecte Cygwin.

El nom Bash

És un acrònim de Bourne-Again Shell ("un altre *shell* Bourne"), fent un joc de paraules (*born-again* significa "renaixement") sobre el *shell* Bourne (sh), que va ser un dels primers intèrprets importants d'Unix.

El Bash és un intèrpret d'ordres compatible amb el *shell* Bourne (sh), que incorpora característiques útils del *shell* Korn (ksh) i del *shell* C (csh), amb l'objectiu de complir amb l'estàndard **POSIX**. Bash ofereix millores funcionals sobre sh tant per a la programació com per a l'ús interactiu i s'ha convertit en l'estàndard de facto per a la programació de guions de *shell*.

Com que el *shell* Bash (bash) és compatible amb el *shell* Bourne (sh), les ordres i els programes escrits per a sh poden ser executats amb bash sense cap modificació, però el contrari no sempre és cert.

POSIX

És l'acrònim de *portable operating system interface*. La X prové d'Unix, com a símbol d'identitat de l'API. Una traducció aproximada de l'acrònim podria ser "interfície de sistema operatiu portàtil basat en Unix".

Queda fora de l'abast d'aquesta unitat fer un estudi de totes les característiques del *shell* Bash i únicament ens centrarem en aquelles funcionalitats que són necessàries per poder abordar correctament la programació de *shell scripts* i l'automatització de tasques del sistema.

Quan arrenca, el sistema presenta als usuaris una interfície determinada que pot ser de text o gràfica, depenent dels sistemes operatius o de com estigui configurat el mode d'arrencada.

La interacció amb Bash es fa a través d'un emulador de terminal. Si hem iniciat sessió al sistema en mode consola (text), una vegada validats obtindrem l'accés directe a un *shell*. Si hem iniciat sessió en mode gràfic, haurem d'executar algun dels programes d'emulació de terminal disponibles. La majoria de sistemes Unix disposen d'una aplicació de terminal accessible per alguna de les opcions del menú principal, per exemple, a la distribució de Linux Debian accedim a un terminal des d'*Aplicacions* > *Accessoris* > *Terminal*.

> Els usuaris de Mac OS X podeu accedir a un *shell* Bash via *Finder* > *Aplicacions* > *Utilitats* > *Terminal*.

Un altre cas d'obtenció d'un intèrpret d'ordres interactiu és l'accés remot a la màquina, tant per qualsevol de les possibilitats de text (Telnet, rlogin, SSH) com per les gràfiques (per exemple, amb emuladors X Window).

Sigui quin sigui el mode d'accés, en iniciar sessió el sistema genera un *shell* per a nosaltres que farà d'intèrpret de les ordres que executem en aquella sessió. Per a això el *shell* fa el següent:

- Mostra per pantalla l'indicador de la línia d'ordres (el *prompt*) assenyalant que està llest per acceptar ordres.
- Quan l'usuari introdueix una ordre, el *shell* la interpreta (busca les ordres, fa l'expansió de noms de fitxers, substitueix els valors de variables per variables referenciades, etc.).
- Si troba algun error mostra a l'usuari un missatge d'error.
- Si l'ordre està ben escrita, aleshores localitza el programa que cal executar i demana al sistema operatiu que l'executi, passant-li a ell el control.
- Quan finalitza l'execució del programa, el control retorna al *shell*, que torna a mostrar el *prompt* esperant una nova ordre.

> Vegeu més informació sobre el *shell* per defecte a l'apartat "*Shells* del sistema operatiu Unix i derivats" d'aquesta unitat.

Després d'iniciar sessió, podem comprovar quin és el *shell* que tenim establert per defecte i amb el que estem treballant executant l'ordre següent:

```
echo $SHELL
```

1.3.2. Interpretació d'ordres

La funció principal del *shell* és interpretar ordres, ja sigui les ordres escrites de manera interactiva a la interfície de la línia d'ordres que proporciona, o les ordres escrites en un fitxer de text (guió de *shell*).

Les ordres que escrivim, ja sigui en l'indicador de la línia d'ordres del *shell* o en un programa de *shell*, tenen el format següent:

On:

- *nom_ordre*: és el nom de l'ordre que volem executar
- *opcions*: les ordres poden o no portar opcions. Normalment les opcions s'escriuen amb un guió davant.
- *arguments*: depenent de l'ordre, es poden posar arguments que moltes vegades representen una cadena de caràcters, el nom d'un fitxer o directori.

> El *shell* interpreta sempre l'espai en blanc com a separador d'ordres, opcions o arguments. Si no posem els espais correctament, obtindrem un missatge d'error.

Exemple d'execució d'ordres

És possible que una mateixa ordre accepti diferents modes d'execució, a soles, amb opcions, amb arguments. Per exemple l'ordre ls:

```
ls
ls -l
ls /etc/shells
ls -l /etc/shells
```

L'ordre següent dóna error perquè no hem posat l'espai requerit per separar l'opció i l'argument:

```
ls -l/etc/shells
```

1.3.3. Expansió de noms de fitxers

El *shell* ens proporciona una característica que s'anomena **generació de noms de fitxers** o **expansió de noms de fitxers**,que ens estalvia temps a l'hora de teclejar els noms dels fitxers amb els quals operen les ordres.

> La generació de noms de fitxers permet utilitzar uns caràcters especials per especificar grups de noms de fitxers.

Es poden trobar noms de fitxers o directoris que compleixen un patró determinat, per exemple, tots els noms que acaben en *.c*, o tots els que comencen per *test*, o tots els que tenen tres caràcters. Mitjançant l'ús de les **expressions regulars**, podem referir-nos als noms dels fitxers que compleixen el patró determinat. El *shell* expandeix el patró corresponent als noms de fitxers abans d'executar l'ordre. La taula 1.2 mostra les expressions regulars utilitzades per a la generació de noms de fitxers.

Expressions regulars

En informàtica, una expressió regular és una representació, segons unes regles sintàctiques d'un llenguatge formal, d'una porció de text genèric a buscar dins d'un altre text, com per exemple uns caràcters, paraules o patrons de text concrets. El text genèric de l'expressió regular pot representar patrons amb determinats caràcters que tenen un significat especial, com ara el caràcter interrogant, ?,per representar un caràcter qualsevol; el caràcter comodí, *, per representar un nombre qualsevol de caràcters, etc.

Taula: 1.2. Expressions regulars usades en la generació de noms de fitxers

Expressió	Descripció
?	Coincidència amb qualsevol caràcter simple, excepte el punt a l'inici del nom.
*	Coincidència amb zero o més caràcters (excepte el punt inicial).
[]	Coincidència amb qualsevol dels caràcters tancats.
[!]	Coincidència amb qualsevol dels caràcters no tancats.
[-]	Coincidència amb qualsevol dels caràcters del rang.

Les expressions regulars de generació de noms de fitxers no generen mai noms de fitxers que comencen per punt, el punt sempre s'ha d'indicar explícitament.

Exemple de generació de noms de fitxers amb expressions regulars

Situeu-vos en el directori /usr/bin i llisteu els noms de fitxers i directoris que comencin per c:

```
ls c*
```

Els que comencin per la lletra y o z:

```
ls [yz]*
```

Els que acabin en .sh:

```
ls *.sh
```

Els que comencin per r i acabin en e:

```
ls r*e
```

Els que comencin per alguna lletra compresa entre la a i la d:

```
ls [a-d]*
```

Tots aquells fitxers el nom dels quals tingui quatre lletres:

```
ls ????
```

---

una variable és:

```
nom_variable=valor
```

Fixeu-vos que no hi ha cap espai abans ni després del signe igual. Si se'n posa algun obtindrem un error.

Els següents són exemples vàlids de definició de variables:

```
NOM=Marc
COGNOMS="Ros Roig"
EDAT=31
```

El valor assignat a la variable COGNOMS l'hem tancat entre cometes dobles perquè el *shell* no interpreti l'espai que hi ha entre els dos cognoms. Per evitar la possibilitat de cometre errors, podem optar per posar el valor sempre entre cometes dobles:

```
nom_variable="valor"
```

El valor de la variable sempre el podem canviar assignant-li un altre valor. Per exemple:

```
NOM=Mia
```

El valor associat a una variable pot ser utilitzat mitjançant el nom de la variable precedit amb el símbol dòlar: **`$nom_variable`**. Aquest mecanisme es diu **substitució de variables**.

> El *shell* realitza la substitució de variables a qualsevol línia d'ordres que contingui un símbol `$` seguit d'un nom de variable vàlid.

Podem visualitzar el valor d'una variable definida amb l'ordre `echo` i utilitzant el mecanisme de substitució de variables:

```
echo $nom_variable
```

Per exemple:

```
echo $COGNOMS
```

Després d'executar l'ordre anterior, el valor associat a la variable anomenada COGNOMS es mostra per pantalla.

En la substitució de variables es pot fer ús de claus per delimitar el nom de la variable: **`${nom_variable}`**. La utilització de claus ens permet, per exemple, fer la substitució de la variable seguida d'un text. Per exemple:

```
PREFIX=extra
echo ${PREFIX}ordinari
```

```
echo “Mots que comencen per $PREFIX: ${PREFIX}ordinari, ${PREFIX}polar, ${PR
```

L’ordre anterior mostra per pantalla la frase “Mots que comencen per extra: extraordinari, extrapolar, extramurs, etc.”.

Per convenció, els noms de les variables s’escriuen en majúscules, però es poden posar en minúscules. Els noms de les variables **han de començar obligatòriament** per caràcter alfabètic (*a-z*, *A-Z*) i poden contenir caràcters alfabètics, numèrics i subratllats. No hi ha restriccions respecte al nombre de caràcters que pot tenir el nom d’una variable.

Les variables poden ser de dos tipus, variables locals o variables d’entorn:

- **Variables locals**: només són visibles pel *shell* en el qual estem treballant, no són visibles per cap *shell* fill.
- **Variables d’entorn**: són visibles tant pel *shell* pare com pels *shells* fills. En ser creat, el *shell* fill “hereta” les variables d’entorn del pare (en rep una còpia). El *shell* fill pot canviar les variables heretades del *shell* pare, però, com que són una còpia, els canvis fets en el fill no afecten el pare.

> Una variable definida en un procés fill, ja sigui local o d’entorn, no és visible pel seu procés pare.

La manera de fer una variable d’entorn és mitjançant l’ordre `export`:

```
export NOM_VAR
```

L’assignació i exportació es pot fer en un sol pas:

```
export NOM_VAR=valor
```

Exemple de manipulació de variables locals i d'entorn

Executeu de manera seqüencial les línies d’ordres següents:

```
x=11    # Definim una variable x amb valor 11
echo $x # Mostrem el valor de x
bash    # Obrim un shell fill
echo $x # El shell fill no coneix el valor de x
exit    # Sortim del shell fill i retornem al pare
export x# Exportem la variable x a l'entorn
bash    # Obrim un shell fill
echo $x # Ara el shell fill sí coneix el valor de x
x=12    # Modifiquem en el shell fill el valor de x
exit    # Tornem al shell pare
echo $x # El valor de x segueix sent 11
```

Hi ha diverses ordres relacionades amb les variables:

- Ordre `unset nom_variable`: elimina la variable i el valor associat a la variable.

En qualsevol sessió de *shell* Bash hi ha presents una sèrie de **variables d'entorn predefinides** pel sistema que ens poden resultar de molta utilitat en la programació de guions de *shell*, ja que guarden informació genèrica de l'entorn dels usuaris.

La taula 1.3 mostra el nom i la descripció d'algunes d'aquestes variables.

Localització

Localització es refereix al conjunt de convencions que es fan servir en una zona geogràfica o cultural determinada, com ara el format de la data i l'hora, el nom de la moneda, el separador per a desenes i centenes, etc. L'ordre locale mostra informació sobre el valor de les variables de localització del sistema.

Exemple d'ús de variables d'entorn predefinides

Fent ús de la substitució de variables i utilitzant variables d'entorn predefinides escriviu una línia d'ordres que en executar-se mostri per pantalla el text:

"Sóc nom_usuari i el meu directori de treball és directori".

Els valors de nom_usuari i directori s'han d'obtenir de les variables adequades i predefinides pel sistema.

```
echo "Sóc $USER i el meu directori de treball és $HOME"
```

Feu el mateix, però ara heu de mostrar:

"Treballo amb un sistema sistema_operatiu i un shellnom_shell"

```
echo "Treballo amb un sistema $OSTYPE i un shell $SHELL"
```

Taula: 1.3. Algunes de les variables predefinides del shell Bash

Nom variable	Descripció
BASH	Camí del programa Bash que s'està executant.
BASH_VERSION	Versió del Bash que utilitzem.
COLUMNS	Nombre de columnes a la pantalla.
HISTFILE	Fitxer on es guarda l'històric de les ordres executades per l'usuari.
HOME	Directori d'inici de l'usuari.
HOSTNAME	Nom de l'ordinador.

LC_ALL	Aquesta variable sobreescriu el valor de LANG i de qualsevol variable LC_ que especifiqui una categoria de localització.
LOGNAME	Nom de l'usuari connectat.
MACHTYPE	Arquitectura de l'ordinador.
OSTYPE	Tipus de sistema operatiu que estem utilitzant.
PATH	Llista de directoris on el *shell* ha de localitzar els programes quan els escrivim sense indicar el camí on es troben.
PPID	Identificador del procés pare.
PS1	Indicador de la línia d'ordres primari.
PS2	Indicador de la línia d'ordres secundari.
PWD	Camí del directori actual, és a dir, el directori on estem situats.
RANDOM	Número aleatori.
SECONDS	Temps en segons des que s'ha iniciat el *shell*.
UID	Identificador de l'usuari actual.

1.3.5. Substitució d'ordres

La substitució d'ordres s'utilitza per reemplaçar la sortida de l'execució d'una ordre dins de la mateixa línia d'ordres.

Es pot fer amb la sintaxi següent:

```
$(ordre)
```

O bé tancant l'ordre entre accents greus: `` `ordre` ``.

Les dues maneres són vàlides però si estem escrivint un *shell script* és millor triar-ne una i fer servir la mateixa manera al llarg de tot el programa.

En els exemples d'aquesta unitat utilitzarem `$(ordre)` i no `` `ordre` `` per fer la substitució d'ordres.

De la mateixa manera que en la substitució de variables, la substitució d'ordres es fa sempre abans d'executar la línia d'ordres. Quan el *shell* troba a la línia d'ordres un *$* seguit d'un parèntesi obert, executa tot el que troba fins a arribar al parèntesi tancat. El resultat el posa a la línia d'ordres en el lloc on hi havia l'ordre que substituïa.

Exemple de substitució d'ordres

Fent ús de la substitució d'ordres escriviu una línia d'ordres que en executar-la mostri per pantalla el text: "La data del sistema és: data". El valor de data s'ha d'obtenir del resultat d'executar l'ordre adequada.

També es podria haver fet així: `echo "La data del sistema és: `date`"`.

La substitució d'ordres és un mecanisme molt utilitzat en la programació de guions de *shell*. Moltes vegades es fa servir en l'assignació de valors a variables. Per exemple:

L'ordre `date` admet diversos modificadors de format de sortida que s'indiquen amb *+%*. Per exemple, `date +%x` indica la data del sistema amb format dd/mm/aa.

```
HORA=$(date +%H)
```

Si la sortida de l'ordre que substituïm conté salts de línia, el *shell* els substituirà per espais en blanc. Per exemple, executeu l'ordre següent:

```
seq 10
```

L'ordre `seq` és utilitzada per generar seqüències de números, vegeu la seva sintaxi amb man seq.

L'ordre anterior mostra per pantalla els números de l'1 al 10 separats per salts de línia. Ara poseu la mateixa ordre en una substitució d'ordres, per exemple:

```
echo $(seq 10)
```

El resultat de l'ordre anterior és la llista de números separats per espais.

Exemple d'ús de variables i substitució d'ordres

Assigneu a una variable la llista de nombres parells que hi ha entre el 0 i el 20 (inclosos) separats per espais:

```
PARELLS=$(seq 0 2 20)
```

1.3.6. Expansió aritmètica

El mecanisme d'expansió aritmètica del Bash permet l'avaluació d'una expressió aritmètica i la substitució del resultat.

```
$((expressió_aritmètica))
```

No confongueu el mecanisme de substitució d'ordres, `$(ordre)`, amb el d'expansió aritmètica, $((expressió)).

L'avaluació d'expressions aritmètiques només admet **nombres sencers**. Els operadors admesos són gairebé els mateixos que en el llenguatge de programació C. La taula 1.4 mostra la llista d'operadors existents (no incloem els que operen amb bits) **en ordre decreixent de precedència**. Podem utilitzar parèntesis per alterar l'ordre de precedència dels operadors.

Taula: 1.4. Operadors aritmètics

Operador	Descripció
VAR++, VAR–	Postincrement i postdecrement de la variable
++VAR, –VAR	Preincrement i predecrement
-, +	Menys i més unaris
!	Negació lògica
**	Potència
*, /, %	Multiplicació, divisió i residu
+, -	Suma i resta
⇐, >=, <, >	Operadors de comparació
==, !=	Igualtat i desigualtat
&&	AND lògic
||	OR lògic
expr ? expr : expr	Avaluació condicional
=, *=, /=, %=, +=, -=	Assignacions
,	Separador d'expressions

En les expressions s'admeten variables del *shell* com a operands i podem utilitzar-les fent l'expansió de la variable $NOM_VAR, o bé només amb el seu nom, NOM_VAR. Per exemple, donades dues variables amb els valors següents:

```
X=2
Y=3
```

L'expressió següent:

```
Z=$((X+Y))
```

També la podem escriure així:

```
Z=$(($X+$Y))
```

El mecanisme d'expansió aritmètica ens permet realitzar operacions i substituir el resultat en una altra ordre, per exemple:

En l'exemple anterior fem servir l'ordre *echo* i en l'argument utilitzem l'expansió aritmètica directament per visualitzar el resultat de l'operació X+Y.

1.3.7. Tractament dels caràcters especials

Els caràcters especials són aquells que tenen algun significat concret per al *shell*, per exemple:

- El caràcter espai indica separació d'ordres, opcions o arguments.
- El caràcter salt de línia indica final d'ordre.
- El caràcter *$* davant d'un nom indica referenciar el valor d'una variable amb aquest nom.
- El caràcter * és utilitzat com a expressió regular en la generació de noms de fitxers.

Quan no volem que el *shell* interpreti aquests caràcters de manera especial sinó com a caràcters normals, podem anul·lar el seu significat de les maneres següents:

- Precedint el caràcter del símbol\. Aquesta tècnica anul·la el significat especial del caràcter que va darrera.
- Amb cometes dobles, " ". Aquesta tècnica anul·la el significat de tots els caràcters especials que estiguin dins les cometes dobles excepte el caràcter especial dòlar, *$*, la contrabarra, \, l'accent greu, `, i les cometes dobles, " ".
- Amb cometes simples,' '. Aquesta tècnica anul·la el significat de tots els caràcters especials que estiguin dins les cometes simples.

Exemple d'anul·lació del significat dels caràcters especials

Escriviu una ordre que mostri una frase com la següent: Estic llegint el llibre "Córrer o morir". El títol "Córrer o morir" ha d'aparèixer entre cometes dobles.

```
echo Estic llegint el llibre \"Córrer o morir\".
```

Es pot aconseguir el mateix amb les cometes simples:

```
echo 'Estic llegint el llibre "Córrer o morir".'
```

Escriviu una ordre que mostri la frase següent: El contingut de *$USER és: nom_usuari*. S'ha de substituir *nom_usuari* pel valor de la variable USER.

```
echo El contingut de \$USER és $USER.
```

1.3.8. Redirecció de l'entrada i la sortida

A Unix els dispositius d'entrada i sortida (E/S) i els fitxers es tracten de la mateixa manera i el *shell* els tracta a tots com a fitxers. Tots els programes executats mitjançant un *shell* inclouen tres fitxers predefinits, especificats pels descriptors de fitxers (*file handles*) corresponents. Per defecte, aquests fitxers són els següents:

- Entrada estàndard (*standard input*). Normalment està assignada al teclat. Utilitza el descriptor número 0.

- Sortida estàndard d'errors (*standard error*). Normalment, està assignada a la pantalla. Utilitza el descriptor 2.

Això ens indica que, per defecte, qualsevol programa executat des del *shell* tindrà l'entrada associada al teclat, la seva sortida a la pantalla i, si es produeixen errors també els enviarà a la pantalla.

> Una característica que ens proporciona el *shell* és poder redirigir l'entrada o la sortida estàndards d'una ordre, és a dir, ens permet fer que una ordre rebi la seva entrada o enviï la seva sortida des de o cap a altres fitxers o dispositius.

De manera que:

- La **redirecció d'entrada** permet que les ordres agafin les dades d'un fitxer enlloc de des del teclat.
- La **redirecció de sortida** ens permet enviar la sortida d'una ordre a un fitxer enlloc de a la pantalla.
- La **redirecció de la sortida d'errors** ens permet enviar la sortida d'errors d'una ordre a un fitxer enlloc de a la pantalla.

La taula 1.5 mostra els caràcters utilitzats per redirigir l'entrada i la sortida de les ordres.

Taula: 1.5. Redireccionament d'entrada i sortida

Caràcter	Descripció	Exemple
<	Redirecció d'entrada.	write usuari < "Hola"
>	Redirecció de sortida. Si el fitxer no existeix el crea i si ja existeix en sobreescriu el contingut.	date > registre.log
»	Redirecció de sortida. Si el fitxer no existeix el crea i si ja existeix l'afegeix a continuació.	who » registre.log
2>	Redirecció de sortida d'errors. Si el fitxer no existeix el crea i si ja existeix en sobreescriu el contingut.	ls /tmp/kk 2> registre.log
2»	Redirecció de sortida d'errors. Si el fitxer no existeix el crea i si ja existeix l'afegeix a continuació.	ls /tmp/kk 2»registre.log
2>&1	Redirecció de la sortida d'errors a la sortida estàndard.	ls /tmp/kk 2>&1
1>&2	Redirecció de la sortida estàndard a la sortida d'errors.	ls /tmp/kk 1>&2
>&	Redirecció de sortida i de sortida d'errors a un fitxer.	ls /tmp/kk >& registre.log

Si volem que l'execució d'una ordre no generi activitat per pantalla (execució silenciosa), hem de redirigir totes les seves sortides a /dev/null. Per exemple:

```
ls /tmp/kk >& /dev/null
```

Si volem redirigir la sortida estàndard i la sortida d'errors a un fitxer amb la doble redirecció, per tal que si el fitxer no existeix el creï i que si ja existeix afegeixi les dues sortides, podem fer-ho així:

1.3.9. Canonades o 'pipes'

> El *shell* permet enllaçar la sortida d'una ordre com a entrada d'una altra ordre mitjançant el que anomenem **canonades** o ***pipes***.

La sintaxi és la següent:

```
ordre1 | ordre2
```

La sortida de l'ordre 1 s'utilitza com a entrada de l'ordre 2. El símbol | s'anomena *pipe* ("canonada" en anglès). Es poden posar espais entre la canonada i les ordres per fer més llegible la línia d'ordres, però no és obligatori, els espais són opcionals.

Per exemple, executem una ordre *echo* per mostrar per pantalla una línia de text:

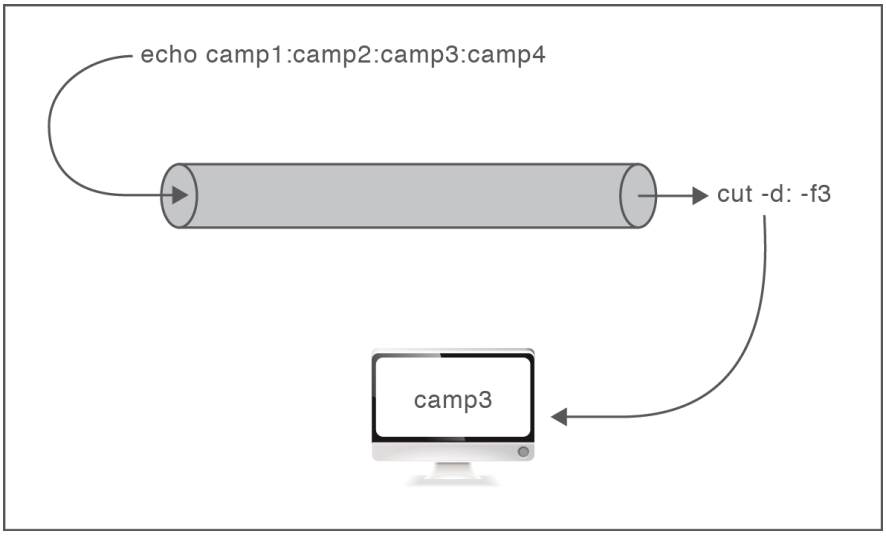
```
echo "Alba:Gomez:08004:BCN"
```

L'ordre anterior mostra per pantalla una serie de dades separades per dos punts així: "Nom:Cognoms:CP:Ciutat". Si d'aquesta sortida només volem prendre el tercer camp (el de codi postal), la podem redirigir amb una canonada cap a l'ordre *cut* perquè seleccioni únicament el camp que ens interessa de la manera següent:

```
echo "Alba:Gomez:08004:BCN" | cut -d: -f3
```

A la figura 1.6 podem veure aquest exemple de manera gràfica. La sortida de l'ordre *echo* passa per la canonada i la rep com a entrada l'ordre *cut*, la qual la processa i mostra per pantalla la sortida processada.

Figura 1.6. Funcionament de les canonades



Veiem un altre exemple senzill d'utilització de canonades:

```
cat /etc/passwd | more
```

s'omple la pantalla.

> Amb l'ordre *more* avancem de pàgina amb la tecla d'espai.

Una línia d'ordres escrita amb una pipe es coneix com una ***pipeline***. La *pipeline* no està limitada només a dues ordres. Es poden fer *pipelines* més llargues associant la sortida d'una ordre amb l'entrada de la següent i així successivament.

```
ordre1 | ordre2 | ordre3 | ordre4 ...
```

Per exemple:

```
cat /etc/passwd | sort | more
```

En l'exemple anterior hi ha un encadenament de tres ordres amb transmissió de les seves dades d'una a l'altra: la sortida de cat s'envia d'entrada a sort i la sortida de sort s'envia d'entrada a more. El resultat final consisteix en mostrar les línies del fitxer /etc/passwd, ordenades i fent una pausa cada vegada que s'omple la pantalla.

La manera com enllacem les diferents ordres no és aleatòria, sinó que s'ha de fer de manera adequada per obtenir els resultats esperats. En la construcció de *pipelines* llargues convé ser metòdics per evitar errades, és preferible fer la *pipeline* pas a pas i anar comprovant els resultats intermedis per corregir els possibles errors. El procés a seguir seria el següent:

- Comprovem les dues primeres ordres de la *pipeline*: `ordre1 | ordre2`
- Si el resultat és correcte, aleshores afegim l'ordre següent del procés i comprovem el resultat intermedi: `ordre1 | ordre2 | ordre3`
- Si el resultat és correcte, afegim la següent: `ordre1 | ordre2 | ordre3 | ordre4`
- I així successivament fins a obtenir la *pipeline* final.

Quan encara no s'està familiaritzat amb l'ús de les canonades, a vegades es tendeix a confondre el seu propòsit i no s'utilitzen bé. No totes les ordres es poden enllaçar de qualsevol manera amb canonades. Per fer una *pipeline* hem de tenir en compte les consideracions següents:

- Una ordre pot tenir a la dreta una canonada si admet sortida estàndard. Per exemple, les ordres *ls*, *who*, *date*, *cat*, etc.
- Una ordre pot tenir a l'esquerra una canonada si admet entrada estàndard. Per exemple, les ordres *wall*, *write*, etc.
- Una ordre pot tenir una canonada a la seva esquerra i a la seva dreta si admet tant entrada com sortida estàndards. Per exemple, les ordres *sort*, *cut*, *grep*, etc.

One liners

Entre els administradors de sistemes és comú el terme one liners. Es refereix a un conjunt d'ordres unides generalment per canonades que enllaçades adequadament produeixen un

```
ps -eo pcpu,user,pid,cmd | sort -r | head -6
```

Fixeu-vos que hem utilitzat l'opció o de l'ordre ps, que permet personalitzar la sortida, i hem indicat les columnes que volem que es mostrin. L'opció e de l'ordre indica extended, és a dir, els processos de tots els usuaris. Tota la sortida de ps és enviada a sort, que en fa un ordenament invers basant-se en la primera columna (pcpu). Per acabar, la sortida ordenada és enviada a l'ordre head, que mostrarà les sis primeres línies (la capçalera i els cinc processos amb més consum).

1.3.10. Filtres i 'pipelines'

A Unix hi ha unes ordres que es fan servir molt a les *pipelines* i que anomenem **filtres**. Els filtres són ordres que accepten dades de l'entrada estàndard, escriuen la sortida a la sortida estàndard i escriuen els errors a la sortida estàndard d'errors, i mai modifiquen les dades d'entrada. A causa del seu funcionament, és freqüent utilitzar filtres a les *pipelines*.

Per exemple, un filtre molt utilitat és l'ordre *sort*, que serveix per ordenar línies de text. L'ordre accepta dades des de l'entrada estàndard (el teclat). Per introduir-les només cal escriure l'ordre i prémer la tecla de retorn:

```
sort
```

El cursor se situa a sota esperant línies d'entrada. Escrivim les línies de text que volem que l'ordre ordeni separades per salts de línia i, en acabar, hem de prémer la combinació de tecles *Ctrl+D* per indicar la finalització de l'entrada de dades. A continuació ens apareixeran per pantalla les línies ordenades.

> Per comprendre les *pipelines* de l'exemple és recomanable que les executeu comprovant els resultats intermedis, seguint el procés descrit en l'apartat "Canonades o *pipes*" d'aquesta unitat.

Com que els filtres accepten les dades del teclat (entrada estàndard) i mostren el resultat per la pantalla (sortida estàndard), podem utilitzar-los en *pipelines* per processar mitjançant una canonada la sortida de qualsevol ordre que doni el resultat en línies de text. Per exemple:

```
who | sort
```

L'ordre *who* mostra una línia per a cada usuari connectat al sistema. En enviar aquesta sortida a l'entrada de *sort*, les línies es mostren ordenades per pantalla.

Vegem un altre exemple:

```
cat /etc/group | sort
```

L'ordre *cat* mostra les línies del fitxer /etc/group. En enviar aquesta sortida a l'ordre *sort*, les línies del fitxer es mostren ordenades per pantalla, però el fitxer no es modifica.

pot escriure així:

```
sort /etc/group
```

En general els filtres són molt útils per processar el contingut dels fitxers de text. Alguns dels més utilitzats són *wc*, *sort*, *grep* i *cut*:

- Comptar el nombre de línies d'un fitxer:
  `wc -l /etc/passwd`
- Ordenar alfabèticament les línies del fitxer:
  `sort /etc/passwd`
- Fer la recerca de línies que continguin un patró determinat:
  `grep root /etc/passwd`
- Extreure parts de les línies del fitxer:
  `cut -d: -f1 /etc/passwd`

Exemple de construcció de pipelines amb filtres

Compteu els fitxers del directori actual.

```
ls | wc -l
```

Mostreu els noms (només els noms) dels usuaris donats d'alta al sistema ordenats alfabèticament.

```
cat /etc/passwd | cut -f1 -d: | sort
```

Mostreu els noms dels usuaris donats d'alta en el sistema i el seu shell d'inici, ordenats alfabèticament i separant els dos camps per un tabulador.

```
cat /etc/passwd | cut -f1,7 -d: | sort | tr ":" "\t"
```

Mostreu l'identificador (PID) i el nom (CMD) de tots els processos que pertanyen a l'usuari root.

```
ps -ef | grep "^root " | tr -s " " | cut -f2,8 -d" "
```

Mostreu una llista amb el propietari, grup i nom de fitxer de tots els fitxers que hi ha a /etc. La llista ha d'estar ordenada pel nom del grup del fitxer.

```
ls -l /etc | tr -s " " | cut -f3,4,9 -d" " | sort -k2 -t" "
```

ADMINISTRACIÓ DE SISTEMES OPERATIUS