Llenguatges de guions i automatització de tasques

Mª Del Mar Sánchez-Colomer Ruiz

Administració de sistemes operatius

Índex

Introducció

L'intèrpret d'ordres o *shell* és un programari que proporciona una interfície als usuaris en un sistema operatiu i els proveeix accés als serveis del nucli. La majoria d'intèrprets d'ordres incorporen un llenguatge de programació propi que permet crear programes que anomenem *guions de shell* o *shellscripts*. En l'administració de sistemes operatius, entendre i saber fer guions de *shell* és fonamental perquè és la manera principal d'automatitzar tasques de manteniment i configuració del sistema. Algunes d'aquestes tasques, com ara les còpies de seguretat, s'han de poder executar de manera periòdica i sense que hi intervingui l'usuari. Mitjançant la programació de *shellscripts* i un planificador de tasques aconseguim aquest propòsit.

Al llarg de la unitat "Llenguatges de guions i automatització de tasques" tractarem els continguts relacionats amb la matèria de manera genèrica per als sistemes operatius de servidor més representatius en l'actualitat i aprofundirem en el sistema operatiu Unix. En els sistemes de tipus Unix gairebé tots els fitxers de dades de configuració del sistema són fitxers de text i una bona part del codi d'inicialització del sistema pren la forma de guions de *shell*. Això fa d'Unix un sistema operatiu idoni per aprendre els conceptes i els procediments implicats en l'administració de sistemes en general i, en particular, dels que ens ocupen en aquesta unitat.

En l'apartat "Llenguatges de guions de *shell*" veurem alguns conceptes previs sobre llenguatges de programació i la diferència entre els llenguatges compilats i els llenguatges interpretats. A continuació introduirem què són els llenguatges de guions, els *shells*, els guions de *shell*, quins tipus de *shell* hi ha, i ens centrarem en el Bash, l'intèrpret d'ordres predeterminat de gairebé totes les distribucions GNU/Linux, així com del sistema operatiu Mac OS X. No farem un estudi de totes les característiques del *shell* Bash atès que queda fora de l'abast d'aquesta unitat, però veurem aquelles funcionalitats que és necessari conèixer per poder abordar correctament la programació de *shellscripts* i l'automatització de tasques del sistema.

En l'apartat "Programació del *shell* Bash" introduirem el llenguatge de programació de Bash i estudiarem els aspectes essencials per crear i executar guions de *shell*, interaccionar amb l'usuari, tractar paràmetres, especificar i avaluar condicions, utilitzar les estructures de control i fer funcions.

En l'apartat "Planificació i automatització de tasques" veurem què són els sistemes de planificació de tasques i estudiarem la configuració i el funcionament del planificador de tasques dels sistemes Unix i derivats, anomenat *cron*. Arribats a aquest punt estarem en disposició d'automatitzar i planificar tasques del sistema, de manera que acabarem analitzant alguns casos pràctics d'automatització amb les eines que hem estudiat. En concret, veurem exemples de procediments de còpies

de seguretat, manteniment de fitxers de registre, gestió massiva d'usuaris i inici automàtic de serveis.

Per treballar els continguts d'aquesta unitat, és necessari haver assimilat una sèrie de continguts bàsics relacionats amb el sistema operatiu Unix: el funcionament de la línia d'ordres, l'organització del sistema de fitxers i coneixements d'administració bàsica del sistema, tals com la gestió d'usuaris i de grups. Igualment, és convenient haver assolit els coneixements de fonaments de programació estructurada i haver fet programes en algun llenguatge d'alt nivell.

Aquesta unitat és eminentment pràctica, per això a mesura que aneu avançant en la lectura dels diferents apartats és molt recomanable que aneu provant els exemples que hi apareixen, i que aneu fent els exercicis i les activitats d'aprenentatge proposades a la web del mòdul.

Resultats de l'aprenentatge

1. Utilitza llenguatges de guions en sistemes operatius, descrivint la seva aplicació i administrant serveis del sistema operatiu.
    - Utilitza i combina les estructures del llenguatge per crear guions.
    - Utilitza eines per depurar errors sintàctics i d'execució.
    - Interpreta guions de configuració del sistema operatiu.
    - Realitza canvis i adaptacions de guions del sistema.
    - Crea i prova guions d'administració de serveis.
    - Crea i prova guions d'automatització de tasques.
    - Implanta guions en sistemes lliures i propietaris.
    - Consulta i utilitza llibreries de funcions.
    - Documenta els guions creats.
2. Gestiona l'automatització de tasques del sistema, aplicant criteris d'eficiència i utilitzant ordres i eines gràfiques.
    - Descriu els avantatges de l'automatització de les tasques repetitives en el sistema.
    - Utilitza les ordres del sistema per a la planificació de tasques.
    - Estableix restriccions de seguretat.
    - Realitza planificacions de tasques repetitives o puntuals relacionades amb l'administració del sistema.
    - Automatitza l'administració de comptes.
    - Instal·la i configura eines gràfiques per a la planificació de tasques.
    - Utilitza eines gràfiques per a la planificació de tasques.
    - Documenta els processos programats com a tasques automàtiques.

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 **llenguatges de nivell alt**, que són més propers al llenguatge humà (normalment l'idioma anglès) i més independents del tipus d'ordinador.

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

Un aspecte a tenir en compte quan es parla del codi font d'un programa informàtic és si la seva llicència permet que aquest codi font estigui disponible perquè qualsevol pugui estudiar-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).

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

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.

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.

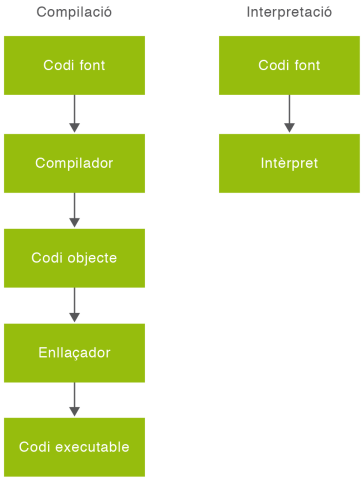
Alguns exemples de llenguatges interpretats són Perl, PHP, Python, Bash i JavaScript.

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**.

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.

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.

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.

Els llenguatges de guions sovint s'utilitzen per ampliar les prestacions que ofereix un altre llenguatge, entorn o aplicació. En aquest sentit són molt utilitzats en el desenvolupament d'aplicacions 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.

Dispositius de xarxa

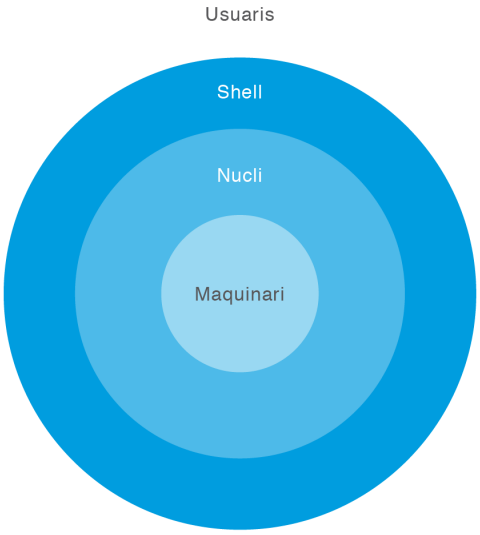
Els dispositius de xarxa que poden ser configurats per programari normalment disposen d'una interfície de línia d'ordres. Per exemple, el sistema operatiu IOS de Cisco disposa d'una interfície de línia d'ordres que és utilitzada per executar ordres de configuració, supervisió i manteniment dels dispositius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

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.

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

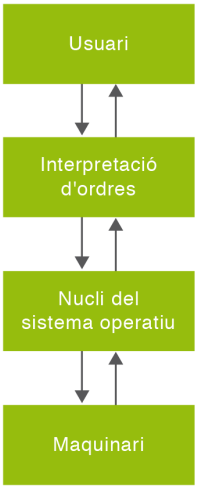


De vegades es confonen els termes d'intèrpret d'ordres i sistema operatiu i, de manera errònia, s'identifiquen l'un amb l'altre.

> 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 de funcions d'anàlisi i passar les ordres interpretades al nucli del sistema operatiu perquè les executi, com es veu a la figura 1.3.

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***.

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.

El terme script

El terme anglès script es va agafar del guió escrit de les arts escèniques, el qual és interpretat per una sèrie d'actors 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 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.

1.2.3 Shells del sistema operatiu Windows

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*.

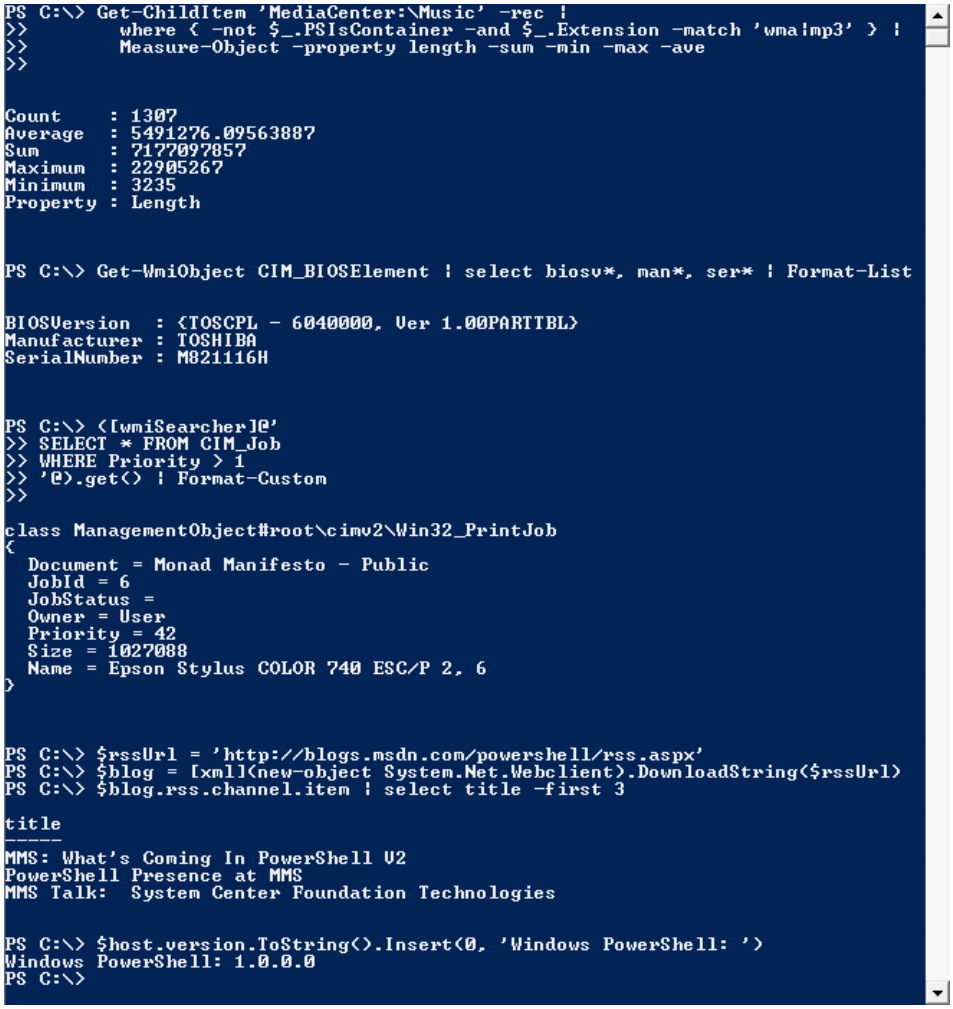
- L'executable **cmd.exe** és l'intèrpret d'ordres dels sistemes basats en Windows NT (incloent Windows 2000, XP, Server 2003, Vista i 7). És l'equivalent de COMMAND.COM a MS-DOS i sistemes de la família Windows 9x. En realitat, cmd.exe és un programa de Windows que actua com un intèrpret d'ordres de 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 d'ordres. A la taula 1.1 en podem veure el nom dels més rellevants amb una breu descripció i les seves característiques.

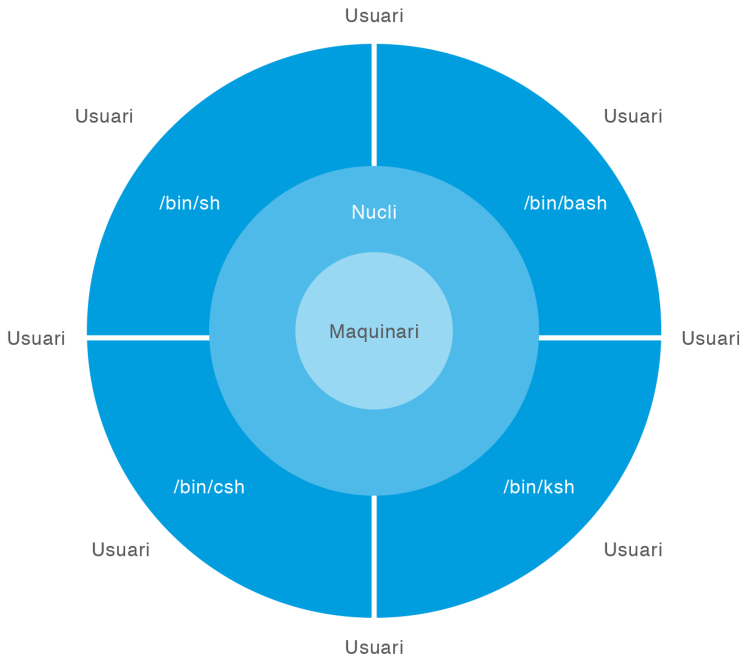
Taula 1.1. Alguns dels shells disponibles als sistemes de tipus Unix

Nom del *shell*	**Programa**	**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:

```
1 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 moment totes les sessions noves executaran aquest *shell* per defecte.

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

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.

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.

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".

1.3.1 Obrir una sessió amb Bash

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:

```
1 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:

```
1 nom_ordre [–opcions] [arguments] <return>
```

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:

```
1 ls
2 ls –l
3 ls /etc/shells
4 ls –l /etc/shells
```

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

```
1 ls –l/etc/shells
```

1.3.3 Expansió de noms de fitxers

El *shell* ens proporciona una característica que s'anomena **generació de noms defitxers** 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 ????
```

1.3.4 Variables del shell

Al *shell*, una variable és un nom que representa un valor. La sintaxi per definir i assignar un valor a 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
```

L'ordre anterior mostra per pantalla la paraula "extraordinari". Fent ús de la mateixa variable PREFIX podem escriure l'ordre següent:

```
echo "Mots que comencen per $PREFIX: ${PREFIX}ordinari, ${PREFIX}polar, ${
    PREFIX}murs, etc".
```

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 `set`: permet veure totes les variables (locals i d'entorn) definides en una sessió.
- Ordre `env`: permet veure les variables d'entorn definides en una sessió.
- 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.
LANG	Usada per determinar el valor de localització per defecte per qualsevol categoria no especificada amb la corresponent variable LC_ (LC_NUMERIC, LC_TIME, LC_MONETARY, etc.).
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:

```
1 $(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.

```
1 echo "La data del sistema és: $(date)"
```

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.

```
1 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:

```
1 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:

```
1 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:

```
1 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.

El format per fer l'expansió aritmètica és:

```
$((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:

```
echo “El resultat de sumar $X i $Y és: $((X+Y))”
```

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.

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

Es pot aconseguir el mateix amb les cometes simples:

```
1 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.

```
1 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 (*standard output*). Normalment està assignada a la pantalla. Utilitza el descriptor 1.
- 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:

```
1 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 ls /tmp/kk >> registre.log 2>> registre.log
```

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:

```
1 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:

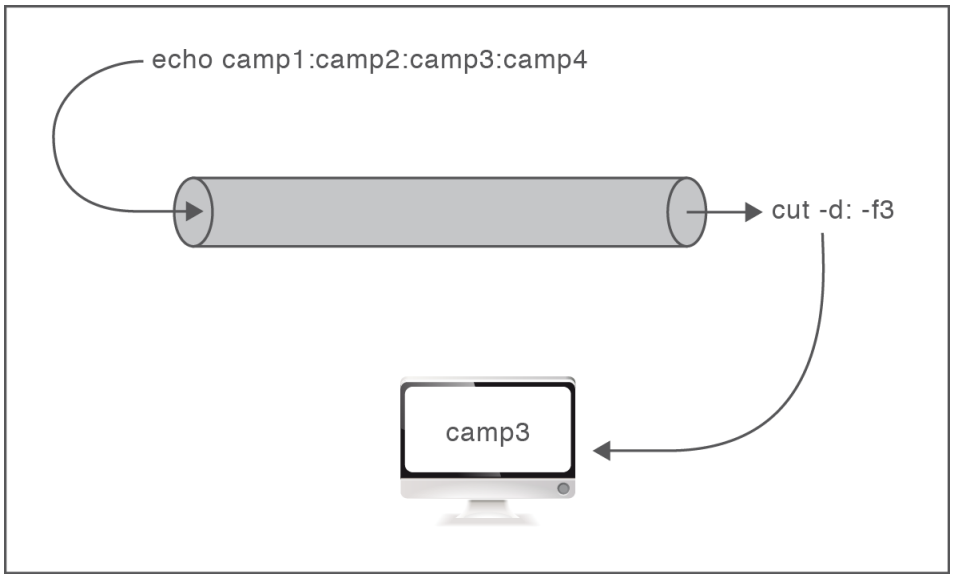
```
1 echo "Alba:Gomez:08004:BCN"
```

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:

```
1 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:

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

En l'exemple anterior la sortida de l'ordre cat /etc/passwd s'utilitza com a entrada de l'ordre more per veure el contingut del fitxer /etc/passwd per pàgines, és a dir, fent una pausa cada vegada que 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.

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

Per exemple:

```
1 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 resultat útil i pràctic. Per exemple, si volem saber quins són els cinc processos que consumeixen més CPU podem executar el següent:

```
1 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:

```
1 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.

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:

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.

```
1 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:

```
1 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.

Els filtres no només accepten dades des de l'entrada estàndard, sinó que també accepten dades des d'un o més fitxers posats com arguments en la línia d'ordres. Aleshores, l'ordre anterior també es pot escriure així:

```
1 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" "
```

2. Programació del shell Bash

El *shell* Bash (Bourne-Again *shell*) és un intèrpret d'ordres, un programa informàtic que té la funció d'interpretar ordres. El *shell* ens proporciona la possibilitat de programar l'execució d'un conjunt d'ordres amb el seu propi llenguatge i d'emmagatzemar-les en un fitxer, que executarem com qualsevol altra ordre del sistema. Aquest fitxer d'ordres s'anomena *guió de shell oshell script*. Es poden escriure guions de *shell* complexos, ja que el *shell* admet variables, paràmetres, entrada i sortida de dades interactiva, comparacions, bifurcacions, bucles, etc.

L'única manera d'aprendre *shell scripting* és fent guions de *shell*. Per això és molt recomanable que a mesura que aneu avançant en la lectura d'aquest apartat copieu els exemples i els proveu al vostre sistema, i que aneu fent les activitats d'aprenentatge proposades al web de la unitat.

2.1 Creació i execució d'un guió de shell

Un **guió de *shell*** o ***shell script*** és un conjunt d'ordres emmagatzemades en un fitxer de text pla per poder ser executades posteriorment amb el nom del fitxer. En el guió podem incloure la crida a qualsevol programa que sigui executable pel *shell* (ordres del sistema, altres guions de *shell*, etc.), així com crides a funcions, estructures de control del llenguatge del *shell*, comentaris, etc.

> Cada ordre que escrivim en un guió ha d'anar separada per un salt de línia o bé pel caràcter *;* (punt i coma) si està a la mateixa línia.

Les instruccions del guió s'executen seguides una darrere de l'altra en l'ordre en que estan escrites, com si les estiguéssim escrivint una a una a la línia d'ordres, i el salt de línia o el caràcter punt i coma s'interpreten com si preméssim *Retorn* després de cada ordre.

2.1.1 Creació i nom del fitxer

Per crear un *shell script* n'hi ha prou amb obrir un nou fitxer buit en un editor de text, escriure la seqüència d'ordres que volem que s'executin i desar el fitxer amb el nom que li volem donar al guió.

El fitxer del guió de *shell* ha d'estar compost únicament per text sense format i per això hem d'utilitzar un **editor de text pla**.

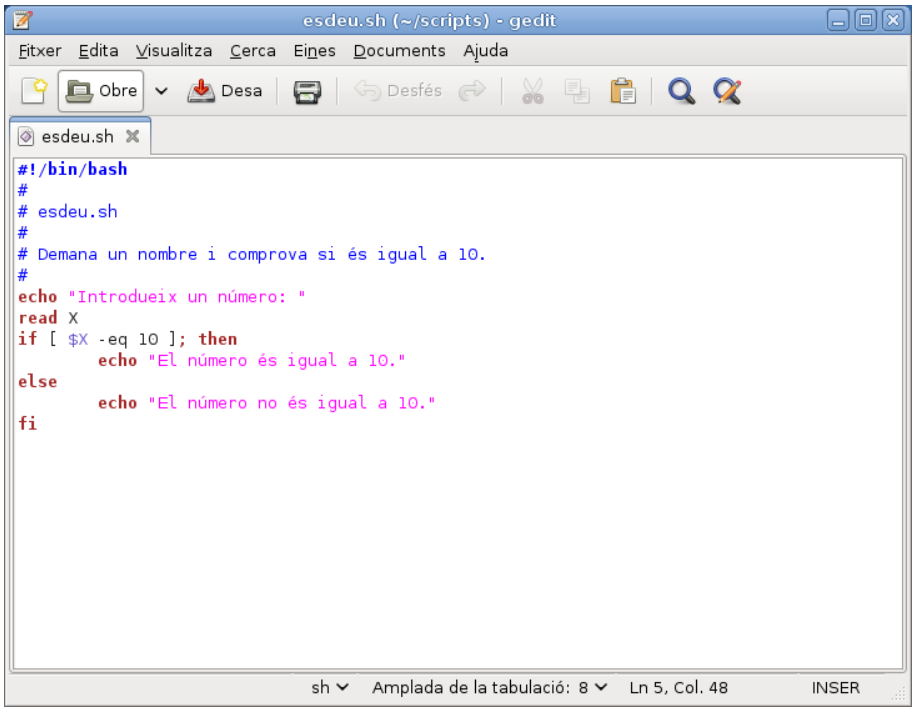
Editors de text avançats

El vi (o el vim a Linux) i l'emacs són editors de text molt potents que s'han utilitzat històricament en entorns Unix, però que poden resultar incòmodes per a usuaris novells. En teniu d'altres més fàcils d'utilitzar, com ara el gedit que es troba per defecte en l'entorn d'escriptori GNOME. I n'hi ha molts altres que no hi són per defecte però que us podeu instal·lar a part.

Els editors de texts plans es distingeixen dels processadors de textos en que es fan servir per escriure només text, sense format i sense imatges, és a dir sense diagramació. També podem optar per fer servir un editor de text avançat que reconegui el llenguatge del *shell* i utilitzi diferents colors per al ressaltat de la sintaxi, com el de la figura 2.1. Aquesta mena d'editors són especialment útils quan som principiants, ja que ens ajuden a prevenir errors de sintaxi, com ara oblidar de posar una clau de tancament, unes cometes dobles, etc.

El nom *vim* ve de *vi improved*. És compatible amb el vi però hi afegeix algunes funcionalitats.

FIGURA 2.1. Editor gedit de l'escriptori GNOME



El nom que donem al fitxer pot ser qualsevol nom vàlid d'acord amb les normes que hi ha per anomenar fitxers en el sistema en què estem treballant. Normalment triarem un nom que sigui representatiu del que fa el guió, per exemple, si fa una còpia de seguretat, podem anomenar-lo *copia*. En tot cas convé que ens assegurem que el nom no entra en conflicte amb d'altres programes o ordres existents al sistema. Per garantir això, els noms dels guions sovint acaben amb l'extensió *.sh*.

Podem anomenar els nostres guions amb l'extensió *.sh*. Ara bé, això només és una convenció i no és obligatori fer-ho.

Noms de fitxers

Les ordres which, whereis i locate ens serveixen per verificar si ja existeixen fitxers amb un nom determinat. Per exemple, si executem which date ens retorna /bin/date, que indica el camí complet on podem trobar aquest fitxer. Si no ens retorna res significa que no localitza cap fitxer amb aquest nom.

Per començar, obriu un editor de text i creeu un nou fitxer que contingui les quatre línies següents:

```
1 #!/bin/bash
2 # El meu primer guió de shell
3 echo "Hola, món"
4 echo "Sóc $USER"
```

Tot i que ara no volem analitzar a fons el significat de cada una de les línies anteriors, en fem una descripció breu per saber què fa el guió:

- La primera línia indica al sistema que el programa Bash ubicat al directori /bin ha d'executar la resta d'instruccions que hi ha a l'script.
- La línia següent és un comentari i, per tant, el shell la ignora.
- La tercera línia és una ordre *echo*, que permet mostrar text per pantalla, en aquest cas, la frase "Hola, món".
- La darrera línia és una altra ordre *echo* que mostra per pantalla el missatge ''Sóc $USER'' substituint el valor de la variable USER pel nom de l'usuari que executa el guió.

Una vegada hem escrit les línies sense errors, desem el fitxer i li donem un nom per poder executar-lo des de la línia d'ordres.

2.1.2 Execució del guió de shell

La manera més habitual d'executar un script és obrir una sessió de terminal i executar-lo com qualsevol altra ordre del sistema, és a dir, escrivint el nom del fitxer que conté l'script en la línia d'ordres seguit d'un salt de línia. Per poder-ho fer cal que el fitxer tingui permís d'execució per als usuaris que han d'executar-lo. En el cas més simple, donarem permís d'execució al propietari del fitxer amb l'ordre següent:

Vegeu com obrir una sessió amb Bash en l'apartat "Obrir una sessió amb Bash" d'aquesta unitat.

```
1 chmod u+x nom_fitxer
```

Després d'afegir el permís d'execució al fitxer, podem executar-lo amb el seu nom tenint en compte les consideracions següents:

- Si el directori que conté el fitxer del *shell script* és a PATH, podem posar només el nom del fitxer que conté el guió perquè s'executi:

  ```
  # Executar script pel nom

  nom_fitxer
  ```

- Si el directori que conté el fitxer del *shell script* no és a PATH, hem d'escriure el nom del fitxer indicant on es troba (amb camí relatiu o camí absolut):

  ```
  # Executar un guió de shell des del directori actual

  ./nom_fitxer

  \\# Executar un guió de shell ubicat a /home/usuari

  /home/usuari/nom_fitxer
  ```

Variable PATH

La variable PATH conté la llista de directoris on el shell s'ha d'adreçar per localitzar les ordres que executem. Això ens permet escriure les ordres escrivint només el seu nom i sense especificar el directori on estan situades, és a dir, sense preocupar-nos d'on es troben al disc. La recerca de l'ordre sol·licitada en els directoris que s'especifiquen en la variable PATH es fa d'esquerra a dreta. Si no troba l'ordre en cap dels directoris especificats en la variable PATH, el shell avisa amb un missatge d'error que no s'ha trobat l'ordre. Podeu visualitzar el valor de la variable PATH amb l'ordre: `echo $PATH`.

Exemple de creació i execució d'un guió de shell

Obriu una sessió de terminal, assegureu-vos que esteu situats al vostre directori d'inici (~) executant l'ordre següent:

```
cd
```

Amb un editor de text pla, creeu un fitxer que contingui les línies següents:

```
#!/bin/bash
# holamon.sh
echo "Hola, món"
echo "Avui és $(date +%x)"
echo "Sóc $USER"
echo "Treballo amb el sistema $(uname -sr)"
echo "Adéu!"
```

Deseu el fitxer al mateix directori d'inici i anomeneu-lo holamon.sh. Doneu permís d'execució al fitxer:

```
chmod u+x holamon.sh
```

Executeu el shell script, feu la crida indicant el nom d'on es troba amb camí relatiu:

```
./holamon.sh
```

Executeu una altra vegada el shell script. Ara feu la crida indicant el nom d'on es troba el fitxer amb camí absolut:

```
~/holamon.sh
```

Creeu un directori ~/scripts per guardar els vostres programes, moveu el fitxer holamon.sh al directori ~/scripts i afegiu el directori al contingut de la variable PATH amb les ordres següents:

```
mkdir ~/scripts
mv ~/holamon.sh ~/scripts
export PATH="$PATH:~/scripts"
```

Ara podeu executar holamon.sh cridant-lo només pel seu nom, sense indicar on es troba (ni amb camí relatiu ni amb camí absolut), perquè està en un directori contingut a la variable PATH.

```
holamon.sh
```

Podeu provar de situar-vos a qualsevol punt de la jerarquia de directoris del sistema i executar holamon.sh posant només el seu nom. Per exemple:

```
cd /tmp
holamon.sh
```

El canvi que heu fet a la variable PATH només afecta a la sessió de shell que teniu activa. Si voleu que la variable PATH tingui el directori ~/scripts en totes les sessions que obriu, cal que editeu el fitxer ~/.bashrc, afegiu al final la línia amb l'ordre export i deseu els canvis.

```
export PATH="$PATH:~/scripts"
```

Un guió de *shell* també es pot executar explícitament amb un *shell* determinat, posant el nom del *shell* i a continuació el nom del fitxer que conté el guió de *shell*:

```
bash holamon.sh
```

Aquest mètode generalment només el fem servir quan volem comprovar que el programa funciona amb un altre *shell* o quan volem depurar (*debug*) el guió de *shell*. Per exemple:

```
rbash nom_script   # Executar el guió amb rbash
sh nom_script      # Executar el guió amb sh
bash -x nom_script # Executar el guió amb mode debug
```

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, i en aquell moment el control torna al procés pare, el qual torna a posar l'indicador o *prompt*.

Els canvis en l'entorn d'un *shell* fill no afecten a l'entorn del *shell* pare.

Per tant, si en el guió hi ha ordres que modifiquen l'entorn, tals com canviar de directori actual, modificar el valor d'una variable d'entorn, crear una nova variable, etc., aquests canvis només tenen efecte a l'entorn del *shell* fill i desapareixen una vegada finalitza l'execució del guió.

Vegeu el significat, la definició i utilització de les variables d'entorn i locals en l'apartat "Variables del *shell*" d'aquesta unitat.

Si volem executar un guió en el *shell* actual en lloc d'amb un *shell* fill i, per tant, modificar l'entorn actual, hem d'executar el guió amb l'ordre `source`.

```
source nom_script # Executar en el shell actual
```

L'ordre del Bash anomenada `source` és un sinònim de l'ordre . (punt) del *shell* Bourne:

```
. nom_script
```

No confongueu l'ordre punt (.) del *shell* Bourne amb el directori punt, que indica el directori actual.

Exemple d'execució d'un guió de shell amb o sense modificació de l'entorn actual

Creeu un guió de shell en el vostre directori de treball que contingui les línies següents:

```
#!/bin/bash
cd /etc
echo El directori actual és:
pwd
```

Deseu el fitxer i anomeneu-lo canvi.sh. Doneu permís d'execució al fitxer:

```
chmod u+x canvi.sh
```

Executeu el guió des del directori actual mitjançant un shell fill:

```
./canvi.sh
```

La sortida us mostra que en el shell fill es canvia de directori. Ara bé, quan finalitza l'execució del guió, el directori on estem situats continua sent el nostre directori de treball, perquè el canvi en el shell fill no ens afecta. Ho comprovem executant l'ordre:

```
1 pwd
```

Executeu el guió en el shell actual (enlloc d'en un shell fill) mitjançant l'ordre source o l'ordre punt (.):

```
1 source canvi.sh
```

En aquest cas, en finalitzar l'execució ens trobem al directori /etc, ja que les ordres del guió s'han executat en el nostre shell. Ho comprovem executant l'ordre:

```
1 pwd
```

2.1.3 Definició del shell d'execució

En escriure guions de *shell* és recomanable que indiquem el *shell* que ha d'executar el guió. Per això els dos primers caràcters de la primera línia han de ser *#!*, seguits del nom del *shell* que ha d'interpretar les ordres que venen a continuació.

Per exemple:

```
1 #!/bin/bash
```

No comenceu el *shell script* amb una línia en blanc, perquè aquestes línies també són tingudes en consideració.

Si ometem la definició del *shell* d'execució, el *shell script* s'executa amb el *shell* que hi ha establert per defecte i, en el cas de no ser el *shell* per al qual s'ha escrit el guió, les ordres contingudes poden donar error en ser executades.

2.1.4 Comentaris al guió de shell

Els comentaris són útils per ajudar a entendre els scripts als lectors. Cal tenir en compte que probablement no serem els únics que llegirem el codi dels guions de *shell* que fem, o que, passat un temps, la memòria ens pot fallar i no entendre el programa que nosaltres mateixos hem escrit.

Scripts d'inici

Vegeu els scripts d'inici del sistema al directori /etc/init.d. Aquests guions estan molt ben comentats perquè siguin fàcilment llegibles pels administradors i els programadors del sistema.

Normalment, les primeres línies després de la línia que indica el *shell* d'execució són un comentari sobre la funcionalitat del programa. La resta del programa es comenta com calgui per a una major claredat i comprensió del codi.

Per afegir comentaris al programa només cal posar el símbol # seguit del text que es vulgui. Cada nova línia ha d'anar precedida del símbol #. Es pot posar el símbol

\ al final d'una línia per indicar que el text no ha acabat i continua a la línia següent. Per exemple, les primeres línies del guió de *shell* /etc/rc.init/kerneloops són:

```
#!/bin/bash
#
# kerneloops
#
# chkconfig: 345 90 88
# description: A tool that collects and submits \
# kernel crash signatures to the kerneloops.org \
# website for use by the Linux kernel developers.
# ...
```

2.1.5 Tabulació del codi

En el cas més simple, un guió no és més que una llista d'ordres del sistema escrites una darrere de l'altra que s'executen de manera seqüencial. Ara bé, en programar *shell scripts* sovint utilitzem estructures de control de flux (condicionals i iteratives) que ens permeten trencar el flux d'execució, repetir una part de les ordres, etc., i que fan que el guió de *shell* no sigui simplement una enumeració d'ordres.

Per tal de facilitar la lectura del codi del guió de *shell*, és molt important que el codi estigui ben tabulat. Al Bash no li cal que hi hagi tabuladors per poder interpretar el codi, però a les persones sí que ens va bé que el codi estigui organitzat amb tabuladors per tal de llegir-lo d'una manera còmoda.

> El codi d'un *shell script* ha d'estar ben tabulat per tal que el programa sigui llegible.

Per exemple, considerem el guió de *shell* següent, escrit sense tabular:

```
#!/bin/bash
echo -n "Escriu un número: "
read X
if [ $X -lt 10 ]; then
echo "X és més petit que 10"
else
if [ $X -gt 10 ]; then
echo "X és més gran que 10"
else
echo "X és igual a 10"
fi
fi
```

La lectura i la comprensió de les estructures condicionals imbricades del guió de *shell* anterior són difícils. El mateix codi, tabulat i comentat, queda així:

```
#!/bin/bash
#
# esdeu.sh
# Demana un número i comprova si és igual a 10.
#
echo -n "Escriu un número: "
read X
```

```
if [ $X -lt 10 ]; then
  echo "X és més petit que 10"
else
  if [ $X -gt 10 ]; then
    echo "X és més gran que 10"
  else
    echo "X és igual a 10"
  fi
fi
```

La tabulació ens permet llegir el codi i seguir la lògica del programa amb més comoditat, així com evitar errades de programació (oblidar una paraula clau de tancament d'una estructura de control, situar una paraula clau en algun lloc indegut, etc.).

No hi ha cap norma que fixi com s'ha de tabular el codi, hi ha qui utilitza tabuladors de dos espais i qui prefereix quatre espais. Tant és, el que importa és utilitzar un estil homogeni al llarg de tot el *shell script*.

Si usem un editor de text pla, hem de sagnar el codi a mà, o bé amb espais o bé amb tabuladors. Alguns editors avançats ens faciliten aquesta feina fent que a mida que escrivim el codi es vagi sagnant de manera automàtica sense que nosaltres ens haguem de preocupar de les tabulacions.

2.1.6 Depurar un guió de shell

De vegades el funcionament d'un guió de *shell* no és l'esperat i hem de determinar quina és la causa d'aquest funcionament incorrecte, és a dir, hem de depurar el *shell script*. Una tècnica consisteix a fer un seguiment pas a pas de les ordres que executa per tal de veure on es produeix l'error. Per fer aquest seguiment podem executar l'*script* amb *bash-x* seguit del nom del guió així:

```
bash -x nom_fitxer
```

O bé podem incloure *-x* a la primera línia:

```
#!/bin/bash -x
```

Si només volem depurar una part del programa, podem fer-ho de la manera següent:

```
...
# activar depuració des d'aquí
set -x

codi per depurar

# parar la depuració
set +x
...
```

L'opció *-x* de Bash indica que s'han d'imprimir per pantalla les ordres i els seus arguments mentre s'executen. Així podem veure a quin punt de l'execució s'ha

arribat quan es produeix un error.

Exemple d'execució pas a pas d'un guió de shell

Feu un guió de shell que es digui prova.sh amb les línies següents:

```
#!/bin/bash
# Shell script de prova
N=5
echo "El valor de N és: $N"
```

Executeu-lo de manera normal i veieu que la sortida és simplement el missatge següent:

```
El valor de N és:  5
```

Ara l'executem mostrant cada pas de l'execució de la manera següent:

```
bash –x prova.sh
```

El resultat és que ens apareixen les línies que es van executant amb un símbol + al davant:

```
+ N=5
+ echo 'El valor de N és:  5'
El valor de N és:  5
```

Podem afegir l'opció -v per mostrar totes les línies d'entrada al shell, incloent comentaris, a mesura que les va llegint:

```
bash –xv prova.sh
```

El resultat és:

```
#!/bin/bash
# Shell script de prova
N=5
+ N=5
echo ''El valor de N és:  $N''
+ echo 'El valor de N és:  5'
El valor de N és:  5
```

2.2 Interacció amb l'usuari

Un guió de *shell* pot requerir interacció amb l'usuari, és a dir, sol·licitar-li l'entrada de dades o mostrar-li dades de sortida. Per poder interaccionar amb el programa normalment utilitzem les ordres ***echo*** i ***read***, que ens permeten mostrar dades a la pantalla del terminal i llegir dades del teclat en mode text.

Si el que volem és fer una aplicació gràfica, aleshores Bash no és l'eina indicada. Tanmateix, podem tenir una interacció bàsica amb mode gràfic i utilitzar caixes de diàleg senzilles amb algun programari que ho permeti, com ara el Zenity.

2.2.1 Ordres echo i read

L'ordre ***echo*** mostra una cadena de text afegint un salt de línia per la sortida estàndard. La sintaxi de l'ordre és:

```
1 echo [opció] [cadena]
```

Executeu `man echo` per consultar la llista de caràcters d'escapament possibles del vostre sistema.

Les opcions que es poden utilitzar amb *echo* són:

- *-n* perquè no afegeixi un salt de línia després de mostrar la cadena.
- *-e* per habilitar la interpretació dels caràcters d'escapament (\n per afegir un salt de línia, \t per afegir un tabulador, etc.). Si utilitzem aquesta opció cal que posem el text de la cadena entre cometes dobles.

Per exemple:

```
1 echo –e "\n\nHola, món!!\n\"n
```

L'ordre ***read*** permet llegir dades de l'entrada estàndard. La utilització més habitual de l'ordre *read* és amb la sintaxi següent:

```
1 read nom_variable
```

L'ordre *read* llegeix el que l'usuari introdueix pel teclat fins que hi ha un salt de línia i assigna les dades a la variable *nom_variable*. Per exemple:

Executeu `man read` per veure la sintaxi completa d'aquesta ordre.

```
1 read N
```

Exemple de guió de shell que interacciona amb l'usuari amb les ordres echo i read.

Feu un guió de shell que s'anomeni salutacio.sh que demani un nom i a continuació mostri un missatge de salutació amb el nom que hem introduït.

```
1 #!/bin/bash
2 #
3 # salutacio.sh
4 #
5 # Exemple d'ús de les ordres echo i read
6 #
7 echo "Com et dius?"
8 read NOM
9 echo "Hola, $NOM"
```

Eines GTK

Les GTK o grup d'eines del GIMP (GIMP toolkit, en anglès) són unes llibreries pensades per al desenvolupament d'aplicacions gràfiques amb facilitat. Disposen de llicència GPL i hi ha nombrosos projectes que les utilitzen, com el GIMP (d'aquí li prové el nom, ja que inicialment es van crear per a aquest projecte), el GNOME, l'Eclipse o el Firefox, entre d'altres.

2.2.2 Interacció en mode gràfic

El *shell* Bash és un *shell* de línia d'ordres pensat per interactuar en mode text, però podem tenir una interacció bàsica en mode gràfic utilitzant altres programes, com

ara Zenity. Zenity permet utilitzar caixes de diàleg basades en GTK+ a la línia d'ordres i en els *shell scripts*.

Podem usar Zenity per crear diàlegs simples que interactuïn gràficament amb l'usuari, ja sigui per obtenir informació de l'usuari o per proporcionar-li informació. Per exemple, podem demanar a l'usuari que seleccioni una data d'un diàleg del calendari o que seleccioni un arxiu d'un diàleg de selecció d'arxiu. O es pot usar un diàleg de progrés per indicar l'estat actual d'una operació o usar un diàleg d'alerta per notificar a l'usuari algun error.

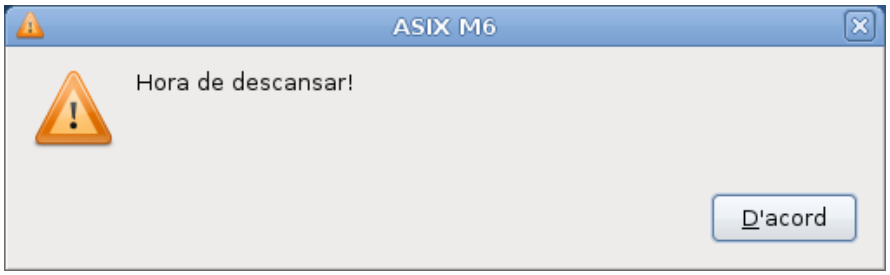
Per exemple, obriu una sessió de terminal a l'escriptori GNOME i executeu la línia d'ordres següent:

Vegeu la documentació oficial de Zenity a la secció del web "Adreces d'interès".

```
1 /usr/bin/zenity --warning --title="ASIX M6" --text="Hora de descansar\!" --
      width=500
```

El resultat és una caixa de diàleg com la de la figura 2.2.

FIGURA 2.2. Caixa de diàleg de Zenity



2.3 Paràmetres i variables especials

La utilització de **paràmetres** és un mètode per passar dades al programa de manera no interactiva. El llenguatge del *shell* disposa d'una serie de variables especials que permeten treballar amb aquests paràmetres.

2.3.1 Ús de paràmetres

Un **paràmetre** o **argument** de la línia d'ordres no és més que un valor que li passem al programa en el moment de la seva crida. Un programa pot tenir qualsevol nombre d'arguments a la línia d'ordres.

La majoria d'ordres d'Unix poden fer accions diferents en funció dels paràmetres que els donem en executar-les. Per exemple, escriviu l'ordre:

```
1 ls /etc/profile
```

`ls` és el nom de l'ordre que s'ha d'executar. Tot el que ve a continuació a la línia d'ordres es pren com a arguments per a aquesta ordre. Cada un dels arguments està separat per un o més espais. En aquest cas hi ha un únic argument, el nom del fitxer /etc/profile.

Considerem un exemple de crida d'una ordre amb dos arguments:

```
1 tail +10 /var/log/messages
```

`tail` és el nom de l'ordre, *+10* és el primer argument i el nom del fitxer /var/log/messages és el segon argument.

A la taula 2.1 podeu veure més exemples de crida d'ordres amb arguments.

TAULA 2.1. Exemples d'execució d'ordres i paràmetres

Ordre	Nom de l'ordre	Nombre d'arguments	Nom dels arguments
ls	ls	0	N/A
ls /etc/resolv.conf	ls	1	/etc/resolv.conf
cp /etc/resolv.conf /tmp/test.txt	cp	2	/etc/resolv.conf /tmp/test.txt
sort -r -n nom_fitxer	sort	3	-r -n nom_fitxer
date +"%d-%m-%Y"	date	1	+"%d-%m-%Y"

De la mateixa manera que ho fan la majoria de les ordres, els guions de *shell* poden acceptar paràmetres quan s'executen. Un paràmetre o argument és un valor que li donem al guió de *shell* en el moment de la seva crida. Els arguments d'un *shell script* es poden referenciar dins del programa mitjançant una sèrie de **variables especials**.

2.3.2 Variables especials

En executar un guió de *shell* amb paràmetres, hi ha un conjunt de variables especials del *shell* anomenat **paràmetres posicionals** que es fixen automàticament perquè coincideixin amb els paràmetres donats al programa. S'anomenen *paràmetres posicionals* perquè l'assignació de cada variable depèn de la posició d'un paràmetre en la línia d'ordres. Els noms d'aquestes variables es corresponen amb el valor numèric de la seva situació en la línia d'ordres: 0, 1, 2, 3... I així fins a l'últim paràmetre que es passa.

Els paràmetres posicionals es poden utilitzar dins del guió de *shell* com qualsevol altra variable del *shell*, és a dir, per saber el seu valor utilitzarem el símbol *$*. A partir del desè paràmetre, el nombre s'ha de tancar entre claus.

Els paràmetres dins del programa són accessibles utilitzant les variables: `$0`, `$1`, `$2`, `$3`... `${10}`, `${11}`, `${12}`...

Exemple d'ús de paràmetres en un shell script

Creeu un shell script que es digui args.sh amb el contingut següent:

```
#!/bin/bash
#
# args.sh
#
# Exemple d'ús de paràmetres
#
echo "Nom del guió de shell: $0"
echo "Valor del primer paràmetre del guió de shell: $1"
echo "Valor del segon paràmetre del guió de shell: $2"
echo "Valor del tercer paràmetre del guió de shell: $3"
```

Deseu el fitxer. Executeu-lo així:

```
chmod +x args.sh
./args.sh blau verd vermell
```

La sortida del programa és la següent:

```
Nom del guió de shell:  ./args.sh

Valor del primer paràmetre del guió de shell:  blau

Valor del segon paràmetre del guió de shell:  verd

Valor del tercer paràmetre del guió de shell:  vermell
```

Podeu provar diferents execucions del programa que acabeu de fer i comprovar-ne els resultats, per exemple:

```
./args.sh A B C
./args.sh Alba 32 Barcelona
```

A banda dels paràmetres posicionals, hi ha unes altres variables especials definides en qualsevol script que podem usar segons les nostres necessitats. La taula 2.2 mostra el nom i la descripció de les variables especials més utilitzades.

Taula 2.2. Variables especials

Variable	Descripció
`$0`	Nom del *shell script* que s'està executant
`$n`	Paràmetre passat al *shell script* en la posició *n*
`$*`	Cadena que conté tots els paràmetres rebuts, començant per *$1*
`$@`	Igual que *$**, excepte quan es posa entre cometes
`$#`	Nombre de paràmetres
`$$`	PID del procés del *shell* que s'està executant
`$!`	PID de l'últim procés executat
`$?`	Codi de sortida de la darrera ordre executada

Vegeu el significat de la variable especial *$?* a l'apartat "Codis de sortida" d'aquesta unitat.

Tingueu en compte les observacions següents:

- *$** i *$@* són el mateix quan no van entre cometes.
- "*$**" expandeix els paràmetres en una cadena: "par1 par2…". És una

sola cadena que comprèn tots els paràmetres units per espais en blanc. Per exemple '1 2' 3 esdevé "1 2 3".

- *"$@"* expandeix els paràmetres en cadenes diferenciades: "par1" "par2"... És a dir, la llista de cadenes resultant coincideix exactament amb allò que s'ha donat al guió de *shell*. Per exemple, *'1 2' 3* esdevé "1 2" "3".

El valor de les variables especials es pot guardar en altres variables, per exemple:

```
NOM=$1
```

Ara bé, l'assignació de valors a les variables especials no està permesa, per exemple:

```
# operació no permesa
$1=Alba
```

Exemple de visualització dels paràmetres i d'altres variables especials.

Modifiqueu el guió de shell args.sh i afegiu-li al final les línies que apareixen en negreta:

```
#!/bin/bash
#
# args.sh
#
# Exemple d'ús de paràmetres
#
echo "Nom del guió de shell: $0"
echo "Valor del primer paràmetre del guió de shell: $1"
echo "Valor del segon paràmetre del guió de shell: $2"
echo "Valor del tercer paràmetre del guió de shell: $3"
echo "Nombre de paràmetres passats al guió de shell: $#"
echo "Llista de tots els arguments rebuts: $*"
```

Executeu-lo:

```
./args.sh a b c
```

La sortida del programa és la següent:

```
Nom del guió de shell: ./cmdargs.sh
Valor del primer paràmetre del guió de shell: a
Valor del segon paràmetre del guió de shell: b
Valor del tercer paràmetre del guió de shell: c
Nombre de paràmetres passats al guió de shell: 3
Llista de tots els arguments rebuts: a b c
```

2.3.3 Control del nombre de paràmetres

La majoria d'ordres mostren un missatge d'error o informatiu quan els arguments requerits per l'ordre no s'han especificat en la seva crida. Per exemple, executeu l'ordre:

```
rm
```

La sortida que us dóna és:

```
rm: missing operand
Try 'rm --help' for more information.
```

Anàlogament, si un guió de *shell* espera rebre paràmetres, hem de verificar dins del programa que la crida al guió de *shell* s'ha fet amb el nombre de paràmetres esperat i, en cas contrari, mostrar un missatge d'error. Aquest control el podem dur a terme mitjançant l'estructura condicional if i la variable especial $#.

Vegeu el funcionament de l'estructura if en l'apartat "Estructures condicionals" d'aquesta unitat.

Exemple d'un shell script amb control de nombre de paràmetres

Creeu un shell script amb el codi següent i anomeneu-lo suma.sh. Aquest script rep dos números per paràmetre i mostra per pantalla el resultat de la suma dels dos números. A l'inici controla que el nombre d'arguments rebuts és correcte i dóna un error si no és així.

```
#!/bin/bash
#
# suma.sh
#
# Rep dos nombres per paràmetre i mostra la suma per
# la sortida estàndard.
#
# Control del nombre de paràmetres:
if [ $# -ne 2 ]; then
echo "Error: s'esperaven dos paràmetres."
echo "Ús del programa: $0 num1 num2"
exit 1
fi
# Rebuts dos paràmetres, fem la suma
(( SUMA = $1 + $2 ))
echo "La suma de $1 i $2 és: $SUMA"
```

Doneu permís d'execució al fitxer i executeu-lo passant-li dos números com a paràmetres:

```
chmod +x suma.sh
./suma.sh 10 5
```

La sortida del programa és la següent:

```
15
```

Executeu-lo passant-li un nombre erroni de paràmetres (diferent de 2), per exemple:

```
./suma.sh
```

La sortida del programa és la següent:

```
Error: s'esperaven dos paràmetres.
Ús del programa: suma.sh num1 num2
```

2.4 Codis de sortida

En finalitzar l'execució totes les ordres generen un **codi de sortida** (en anglès, *exit status* o *exit code*) que és un nombre sencer entre 0 i 255.

Per convenció, si l'ordre acaba bé, normalment retorna un zero (0) i si acaba malament retorna un valor diferent de zero (entre 1 i 255). Moltes vegades, el valor retornat per l'ordre representa l'error generat. Per exemple, els errors de sintaxi gairebé sempre fan que les ordres retornin el valor 1.

El *shell* ens proporciona una variable especial anomenada *?* (signe d'interrogació), que conté el codi de sortida de l'ordre executada anteriorment.

El codi de sortida pot ser utilitzat pel Bash, el podem mostrar o podem controlar el flux del guió amb ell. Per visualitzar el valor utilitzem l'ordre *echo*:

```
1 echo $?
```

En els guions de *shell*, la majoria de les decisions de programació es controlen analitzant el valor dels codis de sortida. Quan avaluem condicions, el codi de sortida ens indica si la condició és **vertadera** (retorna 0) o **falsa** (retorna un valor diferent de zero).

Exemple d'execució d'ordres i visualització dels codis de sortida

Obriu una sessió de terminal i executeu l'ordre ls:

```
1 ls
```

Visualitzeu el codi de sortida de l'ordre que acabeu d'executar:

```
1 echo $?
2 0
```

La sortida és un zero, fet que indica que no hi ha hagut cap error. Executeu l'ordre ls amb algun nom de fitxer inexistent per provocar una errada:

```
1 ls asdfg
2 ls: no s'ha pogut accedir a asdfg: El fitxer o directori no
      existeix
```

Visualitzeu el codi de sortida de l'ordre que acabeu d'executar:

```
1 echo $?
2 2
```

La sortida és un valor diferent de zero (un 2), fet que indica que hi ha hagut un error.

Executeu l'ordre cp sense arguments per provocar una errada:

```
cp
cp: manca un operand fitxer
Proveu «cp ——»help per a obtenir més informació.
```

Visualitzeu el codi de sortida de l'ordre que acabeu d'executar:

```
echo $?
1
```

La sortida és un 1, fet que indica error de sintaxi.

De la mateixa manera que ho fan les ordres, els *shell scripts* retornen un codi de sortida que és el de la darrera ordre executada. El codi de sortida d'un *shell script* també es pot consultar just després de la finalització del programa mitjançant la variable *$?*.

Exemple de visualització del codi de sortida d'un guió de shell

Creeu un guió de shell senzill anomenat hola.sh:

```
#!/bin/bash
echo "Hola, món"
```

Doneu permís d'execució al fitxer i executeu el guió:

```
chmod u+x hola.sh
./hola.sh
```

Consulteu el codi de sortida des de la línia d'ordres:

```
echo $?
0
```

El codi retornat és un zero, el de la darrera ordre executada en el guió de *shell*, és a dir, l'ordre `echo ‘‘Hola, món’’`.

Podeu modificar el guió i afegir al final una ordre que doni error per comprovar l'efecte que té sobre el codi de sortida.

2.4.1 Ordre exit

L'ordre **exit** provoca la finalització del *shell script* i de manera opcional permet fixar el codi de sortida amb un valor determinat. La seva sintaxi és:

```
exit [ n ]
```

essent *n* un nombre enter entre 0 i 255.

Per convenció el valor que es retorna és un zero si ha anat bé o un enter del rang entre 1 i 255 si hi ha hagut algun error. Normalment, el rang de 126 a 255 es reserva per ser utilitzat directament pel *shell* o per a finalitats especials, i els codis del rang de 0 a 125 es deixen per ser utilitzats pel programa.

En tot guió de *shell* convé proporcionar un codi de sortida significatiu, com a mínim un **0 (zero) si finalitza bé** i un **1 (en general, un valor diferent de zero) si hi ha algun error**. Això permetrà que el procés que ha fet la crida pugui verificar com ha anat l'execució del guió de *shell*.

Si no es passa cap paràmetre a *exit*, el codi retornat pel guió de *shell* és el de la darrera ordre executada just abans de l'*exit*. Per exemple:

```
#!/bin/bash
ordre_1
.. .
ordre_N
# Acaba amb el codi de sortida de l'ordre_N.
exit
```

L'exemple anterior, hauria estat exactament igual si s'hagués especificat el valor del codi de retorn de la manera següent:

```
#!/bin/bash
ordre_1
.. .
ordre_N
# Acaba amb el codi de sortida de l'ordre_N.
exit $?
```

Anàlogament, si un *shell script* finalitza sense especificar l'ordre *exit*, el codi de retorn és el de la darrera ordre executada al *shell script*.

```
#!/bin/bash
ordre_1
.. .
ordre_N
# Acaba amb el codi de sortida de l'ordre_N.
```

Exemple d'utilització de l'ordre exit i comprovació dels codis de sortida.

Feu un guió de shell que s'anomeni sortida.sh amb les línies següents:

```
#!/bin/bash

echo "Això és una prova."
# Retornem codi de sortida 0
exit 115
```

Deseu el fitxer i executeu-lo:

```
chmod +x sortida.sh
./sortida.sh
```

La sortida del programa és la següent:

```
Això és una prova.
```

Visualitzeu el codi de sortida del guió de *shell*:

```
echo $?
```

El valor de sortida és un 115, atès que l'hem forçat amb l'ordre *exit*.

```
115
```

En la programació de guions de *shell* és molt freqüent utilitzar l'ordre *exit* per finalitzar el programa en qualsevol punt sense seguir la norma de la programació estructurada que diu que un programa ha de tenir un únic punt de sortida. Per exemple, un *shell script* fet amb programació estructurada seria així:

```
if condicio_error1; then
  codi=1
else
  if condicio_error2; then
    codi=2
  else
    if condicio_error3; then
      codi=3
    else
      # No hi ha errors
      instruccions
      codi=0
    fi
  fi
fi
exit $codi
```

Però habitualment ens trobarem el guió de *shell* anterior escrit amb un estil similar al següent:

```
if condicio_error1; then
  exit 1
fi
if condicio_error2; then
  exit 2
fi
if condicio_error3; then
  exit 3
fi
# No hi ha errors
instruccions
exit 0
```

Tot i que els dos programes fan el mateix, el primer segueix les normes de la programació estructurada i té un únic punt de sortida en el darrer *exit*, mentre que el segon té quatre possibles punts de sortida, un per a cada ordre d'*exit*.

2.5 Avaluació aritmètica i lògica

En programar gairebé sempre apareix la necessitat d'operar amb nombres així com d'avaluar el resultat d'expressions aritmètiques o lògiques per prendre decisions.

A Bash disposem del mecanisme d'**expansió aritmètica** per operar i avaluar expressions amb nombres enters però també tenim altres mètodes, dels quals en veurem dos: **l'ordre** `let` **i la construcció doble parèntesi**. Per aquells casos que necessitem operar amb nombres amb decimals veurem com fer-ho utilitzant l'ordre `bc`.

Vegeu el mecanisme d'expansió aritmètica descrit a l'apartat "Expansió aritmètica" d'aquesta unitat.

Una altra ordre molt utilitzada en la programació de guions de *shell* per fer avaluacions lògiques és l'**ordre** `test`. Aquesta ordre ens permet avaluar expressions amb tres tipus d'elements: nombres enters, fitxers i cadenes de caràcters.

2.5.1 Ordre let

L'ordre `let` permet als programes de *shell* avaluar expressions aritmètiques amb els mateixos operadors que en el mecanisme d'expansió aritmètica i amb la sintaxi següent:

```
let expressió_aritmètica
```

L'ordre avalua d'esquerra a dreta l'expressió i torna un codi de sortida igual a 0 (vertader) o 1 (fals). Malgrat que no sempre cal tancar l'expressió entre cometes dobles, podem optar per posar-les per defecte per evitar errades.

L'expressió aritmètica pot constar de nombres enters, variables i operadors de *shell*. També podem utilitzar parèntesis per canviar l'ordre de preferència d'una expressió. Per exemple:

```
#!/bin/bash
# Definició de variables
x=12
y=2
# Utilització de l'ordre let
let "z = x / y + 1"
echo $z# z val 7, primer s'ha fet la divisió
# Alterar precedència amb parèntesis
let "z = x / (y + 1)"
echo $z# z val 4, primer s'ha fet la suma
```

Quan en l'expressió utilitzem operadors lògics o relacionals (*!*, <=, >=, <, >, ==, *!*=) avaluem el codi de sortida tornat pel *shell*. Per exemple:

```
let "11 < 10"
echo $?
```

L'ordre anterior retorna un 1 indicant que l'expressió s'ha avaluat com a falsa, ja que 11 no és més petit que 10.

Algunes consideracions a tenir en compte quan utilitzem l'ordre *let* són:

- L'ordre *let* no mostra cap resultat per pantalla. Un error comú és esperar que *let* retorni el resultat d'una operació per la sortida estàndard, per exemple, executar `let 4+2` i esperar un 6 per pantalla. Per operar amb *let* cal utilitzar variables i assignar el resultat a les variables, per exemple, `let x=4+2`.
- L'expressió de *let* s'escriu sense espais. Si voleu posar espais per separar els operands dels operadors perquè quedi més clar, heu de posar l'expressió entre cometes dobles. Per exemple, let ‘‘`x = x + 2`’’.
- Si en una expressió voleu utilitzar parèntesis per alterar l'ordre de precedència dels operadors, heu de posar doble cometa per anul·lar el significat especial dels parèntesis. Per exemple, `let ‘‘x=x/(y+1)’’`.
- Les variables que s'utilitzen dins d'una expressió poden anar referenciades o no. Per exemple, let x=x+2 és el mateix que `let x=$x+2`.

2.5.2 La construcció doble parèntesi

La construcció doble parèntesi, *(( ))*, permet avaluar expressions aritmètiques de manera equivalent a l'ordre *let* però amb la sintaxi següent:

```
(( expressió_aritmètica ))
```

Per exemple:

```
(( x = x + (y / 2) ))
```

És equivalent a:

```
let "x = x + (y / 2)"
```

Sovint trobem la construcció doble parèntesi formant part de bucles *while* amb l'estil del llenguatge de programació C. Per exemple:

Alguns autors recomanen utilitzar preferentment els dobles parèntesis enlloc de l'ordre *let*.

```
x=1
while (( x < 10 )); do
  ...
  ordres
  ...
  (( x++ ))
done
```

Igual que amb l'ordre *let*, les variables utilitzades com a operands en una expressió poden anar precedides del símbol *$* o no. Per exemple, les dues expressions següents són correctes:

Vegeu l'explicació dels bucles *while* utilitzant la construcció *(( ))* a l'apartat "Estructures repetitives" d'aquesta unitat.

```
(( x = x + 1 ))   # Correcte
(( x = $x + 1))   # Correcte
```

Però aneu amb compte i no cometeu errades com aquesta:

```
(( $x = x + 1 ))  # Incorrecte!!
```

2.5.3 Operacions amb nombres amb decimals

L'ordre *bc* és un programa molt potent que incorpora un llenguatge de programació propi i que permet fer càlculs amb precisió. Aquesta ordre ens pot ser de molta utilitat si necessitem fer operacions amb nombres amb decimals.

Si executem *bc* a la línia d'ordres, ens apareix una informació sobre la versió del programa similar a la següent:

Executeu *man bc* per veure'n el funcionament i la descripció de totes les opcions.

```
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc
    .
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
```

El programa queda esperant l'entrada de dades de manera interactiva des del teclat, de manera que podem introduir les operacions que volem fer, prémer *Retorn* i a continuació obtenim el resultat, per exemple:

```
6*3/2
9
```

Per sortir de la calculadora hem d'escriure la paraula *quit* seguida de *Retorn*.

Podem utilitzar *bc* dins d'un *shell script* mitjançant una canonada per redirigir les dades d'entrada al programa. Per exemple:

```
echo "(2 + 3) * 5" | bc
```

En l'exemple anterior utilitzem l'ordre *echo* per passar les dades d'entrada a l'ordre *bc*. El resultat de l'operació ens surt per pantalla, en aquest cas un 25. Per treballar amb nombres amb decimals especifiquem la precisió (quantitat de decimals) amb l'opció *scale*. Per exemple:

```
echo "scale=2; 7*5/3" | bc
```

Nombre π

En matemàtiques, π o pi és la constant d'Arquimedes, una constant que relaciona el diàmetre de la circumferència amb la longitud del seu perímetre. És un nombre irracional, és a dir, la seva part fraccionària té un nombre de xifres infinit i a més no té cap període. Per fer càlculs pràctics s'agafa un valor simplificat de pi, com per exemple 3,14159265.

L'ordre anterior ens mostra un 11,66 per pantalla, que és el resultat de l'operació que hem realitzat.

Veiem un darrer exemple en què mostrem el nombre pi (π) per pantalla utilitzant l'opció *-l* en la crida de *bc* per poder usar funcions matemàtiques, en aquest cas la funció *a(x)*, que ens retorna l'arc tangent d'*x* en radiants:

```
echo "scale=9; 4*a(1)" | bc -l
```

2.5.4 Ordre test

Vegeu el significat de la variable especial *?* a l'apartat "Codis de sortida" d'aquesta unitat.

L'ordre `test` avalua expressions lògiques i genera un codi de sortida que indica si l'expressió és **certa** (codi de sortida igual a zero) o **falsa** (codi de sortida diferent de zero). Aquesta ordre no escriu res a la sortida estàndard. Per determinar el resultat de l'ordre *test* cal avaluar el valor del codi de sortida amb la variable *?*.

La sintaxi de l'ordre test és la següent:

```
test expressió_test
```

O en forma abreujada:

```
[ expressió_test ]
```

Cal deixar un espai després del símbol *[* i abans del símbol *]*.

Quan utilitzeu la forma abreujada de *test* (l'expressió entre els claudàtors) cal que aneu amb molt de compte i no oblideu que hi ha espais entre l'expressió i els claudàtors. Si no poseu els espais es produirà un error de sintaxi.

A la taula 2.3 podeu veure les expressions que es poden avaluar amb l'odre *test*.

Taula 2.3. Avaluació d'expressions amb l'ordre test

Avaluació d'expressions amb nombres entersable	
[num1 -eq num2]	cert si num1 i num2 són iguals
[num1 -ne num2]	cert si num1 i num2 són diferents
[num1 -gt num2]	cert si num1 és més gran que num2
[num1 -ge num2]	cert si num1 és més gran o igual que num2
[num1 -lt num2]	cert si num1 és més petit que num2
[num1 -le num2]	cert si num1 és més petit o igual que num2
Avaluació d'expressions amb cadenes de caràcters	
[cad]	cert si és una cadena no buida
[-n cad]	cert si és una cadena no buida
[-z cad]	cert si és una cadena buida
[cad1 = cad2]	cert si cadena1 i cadena2 són iguals
[cad1 != cad2]	cert si cadena1 i cadena2 són diferents
Avaluació d'expressions amb fitxers	
[-e fit]	cert si el fitxer existeix
[-d fit]	cert si el fitxer existeix i és un directori
[-f fit]	cert si el fitxer existeix i és regular
[-L fit]	cert si el fitxer existeix i és un enllaç simbòlic
[-r fit]	cert si el fitxer existeix i té permís de lectura
[-w fit]	cert si el fitxer existeix i té permís d'escriptura
[-x fit]	cert si el fitxer existeix i té permís d'execució
[fit1 -nt fit2]	cert si fitxer1 és més nou que fitxer2
[fit1 -ot fit2]	cert si fitxer1 és més antic que fitxer2
Operadors lògics	
[exp1 -a exp2]	AND. Cert si l'expressió1 i l'expressió2 són certes.
[exp1 -o exp2]	OR. Cert si l'expressió1 o l'expressió2 són certes.
[! exp]	Negació. Cert si l'expressió és falsa.

Exemple d'avaluació de diferents tipus d'expressions amb test

Amb l'ordre test, comproveu si el nombre 11 és més gran que el nombre 15.

```
1 test 11 -gt 15
```

Avalueu el codi de sortida de l'ordre que acabeu d'executar per saber si l'expressió anterior és vertadera (0) o falsa (diferent de 0):

```
1 echo $?
```

```
2 1
```

La sortida és un 1, fet que indica que l'expressió "11 és més gran que 15" és falsa.

Amb la forma abreviada de l'ordre test, comproveu si el número 15 és igual que el número 15.

```
1 [ 15 -eq 15 ]
```

Avalueu el codi de sortida de l'ordre que acabeu d'executar per saber si l'expressió anterior és vertadera (0) o falsa (diferent de 0):

```
1 echo $?
2 0
```

La sortida és un 0, fet que indica que l'expressió "15 és igual que 15" és certa.

Amb la forma abreviada de *test*, comproveu si la cadena "hola" és igual a la cadena "HOLA":

```
1 [ "hola" = "HOLA" ]
```

Avalueu el codi de sortida per saber si l'expressió anterior és vertadera (cadenes iguals) o falsa (cadenes diferents):

```
1 echo $?
2 1
```

La sortida és un 1, fet que indica que l'expressió és falsa, ja que "hola" en minúscules no és igual a "HOLA" en majúscules.

Amb l'ordre test, comproveu si existeix un fitxer anomenat /etc/passwd:

```
1 test -f /etc/passwd
```

Avalueu el codi de sortida de l'ordre que acabeu d'executar per saber si l'expressió anterior és vertadera (el fitxer existeix) o falsa (el fitxer no existeix):

```
1 echo $?
2 0
```

La sortida és un zero, fet que indica que l'expressió és vertadera (el fitxer existeix).

Assigneu a una variable N el valor 15, a continuació amb l'ordre test comproveu si el valor d'N està entre 10 i 20:

```
1 N=15
2 [ $N -gt 10 -a $N -lt 20 ]
3 echo $?
4 0
```

El codi de sortida és un zero, fet que indica que l'expressió és vertadera.

Escriviu amb la forma abreviada de *test* una comprovació que doni cert si el directori /tmp *no* existeix:

```
1 [ ! -d /tmp ]
2 echo $?
3 1
```

El codi de sortida és un 1, fet que indica que l'expressió és falsa, atès que el directori /tmp sí que existeix.

Quan avaluem el valor d'una variable en una expressió, hem d'assegurar-nos que la variable sempre conté algun valor, perquè si no, la variable valdrà *null* i l'script ens donarà un error. Per exemple:

```
[ $X -eq 3 ]
bash: [: -eq: s'esperava un operador unari
```

La variable *X* no té cap valor, per tant el *shell* ha interpretat *[-eq 3]* i ha donat un error.

Si treballem amb cadenes de caràcters, podem evitar aquesta errada posant sempre les variables entre cometes dobles (“”). Així ens assegurem que la variable conté almenys el valor *null* i el shell interpretarà la cadena com a buida. Per exemple:

```
[ "$X" = 3 ]
```

En aquest cas, si *X* no té cap valor, el *shell* interpreta *[”” = 3]* i no dóna error.

> El *shell* interpreta els valors d'una expressió de *test* com a enters o com a cadenes de caràcters segons els operadors que utilitzem.

Les expressions que s'utilitzen amb l'ordre *test* poden ser connectades lògicament utilitzant els operadors propis de *test -a* i *-o*, però la manera més recomanable de fer-ho és utilitzar les ordres de *test* de manera independent i combinar-les amb els operadors lògics *&&* (AND lògic) i || (OR lògic). Per exemple, en lloc d'escriure

```
[ $N -gt 10 -a $N -lt 20 ]
```

és millor separar les expressions i connectar-les així:

```
[ $N -gt 10 ] && [ $N -lt 20 ]
```

Però aneu amb compte; els operadors lògics no poden anar dins dels claudàtors de *test*:

```
# Operació errònia
[ $N -gt 10 && $N -lt 20 ]
```

2.6 Estructures de control

Les estructures de control (*if*, *case*, *while*, *for*, etc.) permeten canviar el flux seqüencial d'un programa en funció de l'avaluació d'unes condicions i bifurcar-lo cap a un cantó o cap a un altre o repetir l'execució d'unes instruccions.

Sovint utilitzem l'ordre *test* abreujada amb claudàtors o la construcció doble parèntesi per avaluar expressions i prendre decisions. Però ni els claudàtors de l'ordre *test* ni els parèntesis de la construcció doble parèntesi **no formen part de la sintaxi de cap estructura de control**del *shell*. No ho confongueu amb altres llenguatges de programació en què l'especificació de condicions en les estructures de control s'ha de fer entre parèntesis perquè la sintaxi ho requereix.

2.6.1 Estructures alternatives

Les estructures alternatives són aquelles que ens permeten executar una part del programa en funció de si es compleix o no una condició.

Estructura if

L'estructura *if* proporciona un control de flux basat en el codi de retorn d'una ordre. La sintaxi és:

```
if condició; then
  ordre1
  ordre2
  ...
  ordreN
fi
```

El *shell* executa l'ordre que estableix la condició potserior a *if* i avalua el codi de retorn resultant:

- Si el valor del codi de retorn és igual a zero, aleshores s'executen les ordres que hi hagi entre les paraules clau *then* i *fi*.
- Si el valor del codi de retorn és diferent de zero, no s'executen les ordres que hi hagi entre les paraules clau *then* i *fi* i el programa segueix executant el que hi hagi darrera de la paraula clau *fi*.

Habitualment utilitzem les ordres *test* i *let* (o el doble parèntesi equivalent) per especificar les condicions. Per exemple, el codi següent comprova si existeix el fitxer /etc/profile:

```
if test -f /etc/passwd; then
  echo "El fitxer /etc/passwd existeix."
fi
```

O bé amb la forma abreviada de *test*:

```
if [ -f /etc/passwd ]; then
  echo "El fitxer /etc/passwd existeix."
fi
```

Però la sintaxi d'*if* accepta qualsevol ordre, perquè totes les ordres generen un codi de retorn. Per exemple:

```
if grep ^root /etc/passwd > /dev/null; then
  echo "L'usuari root existeix."
fi
```

Noteu que l'estructura *if* implica una execució de l'ordre que estableix la condició i una avaluació del codi de retorn de manera implícita. De manera equivalent, es pot executar l'ordre que estableix la condició abans de l'*if* i avaluar el codi de retorn de manera explícita, per exemple:

```
test -f /etc/passwd
if [ $? -eq 0 ]; then
  echo "El fitxer /etc/passwd existeix."
fi
```

I també:

```
grep root /etc/passwd > /dev/null
if [ $? -eq 0 ]; then
  echo "L'usuari root existeix."
fi
```

Ara bé, la manera més habitual d'utilitzar l'estructura *if* és amb l'avaluació del codi de sortida de manera implícita.

Exemple d'utilització de l'estructura de control alternativa if

Feu un guió de shell que s'anomeni esfitx.sh que indiqui si un nom donat com a paràmetre és un fitxer regular.

```
#!/bin/bash
#
# esfitx.sh
#
# Rep un paràmetre i comprova si és un fitxer.
#
if [ -f $1 ]; then
  echo "$1 és un fitxer."
fi
exit 0
```

L'script anterior es pot millorar fent una comprovació del nombre de paràmetres rebuts.

```
#!/bin/bash
#
# esfitx.sh
#
# Rep un paràmetre i comprova si és un fitxer.
#
#
# Control del nombre de paràmetres
if [ $# -ne 1 ]; then
  echo "Nombre d'arguments erroni."
  echo "Ús del programa: $0 nom"
  exit 1
fi
#
# Comprovar si el paràmetre és un fitxer
if [ -f $1 ]; then
  echo "$1 és un fitxer."
```

```
fi
exit 0
```

Estructura if-else

L'estructura *if-else* permet prendre un curs d'acció si el codi de retorn de l'ordre que controla la condició és zero (veritat) i un altre curs d'acció si el codi de retorn és diferent de zero (fals). La sintaxi és la següent:

```
if condició; then
  ordres1
else
  ordres2
fi
```

Per veure el funcionament fem un guió de *shell* senzill que demani l'entrada d'un número per teclat i que a continuació ens digui si el número llegit és igual a 10 o no:

```
#!/bin/bash
#
# esdeu.sh
#
# Demana un nombre i comprova si és igual a 10.
#
echo "Introdueix un número: "
read X
if [ "$X" = 10 ]; then
  echo "El número és igual a 10."
else
  echo "El número no és igual a 10."
fi
```

En l'exemple que acabem de veure hem utilitzat l'ordre *test* per especificar la condició de l'estructura condicional, però també es podria haver fet amb l'ordre *let* o amb la construcció doble parèntesi de la manera següent:

```
echo "Introdueix un número: "
read X
if (( "$X" == 10 )); then
  echo "El número és igual a 10."
else
  echo "El número no és igual a 10."
fi
```

Exemple d'utilització de l'estructura de control alternativa if-else.

Feu un guió de shell que s'anomeni esdir.sh que rebi un argument de manera que si és un directori mostri el missatge "El contingut del directori <nom_directori> és:" i llisti el seu contingut. Si l'argument no és cap directori ha de donar un missatge informatiu. Feu el control del nombre d'arguments.

```
#!/bin/bash
#
# esdir.sh
#
# Rep el nom d'un directori per paràmetre i en mostra
# el contingut.
#
# Control del nombre de paràmetres.
if [ $# -ne 1 ]; then
```

```
10    echo "Nombre d'arguments erroni."
11    echo "Ús del programa: $0 nom_directori"
12    exit 1
13  fi
14  #
15  # Comprovar si el paràmetre és un directori.
16  if [ -d $1 ]; then
17    echo "El contingut del directori $1 és:"
18    ls $1
19  else
20    echo "$1 no és un directori."
21  fi
22  exit 0
```

Estructura if-elif-else

L'estructura *if-elif-else* es pot utilitzar per construir una bifurcació amb múltiples direccions. La sintaxi és:

```
1  if condició1; then
2    ordres1
3  elif condició2; then
4    ordres2
5  elif condició3; then
6    ordres3
7  ...
8  else
9    ordresN
10 fi
```

Noteu que la manera recomanada de tabular l'estructura *if-elif-else* és diferent de l'habitual.

Amb aquesta estructura, coneguda com a escala *if-else-if*, el *shell* avalua les expressions condicionals començant per la primera i continuant per la següent de forma descendent fins a trobar una condició veritable (codi de retorn igual a zero), moment en què s'executa la llista d'ordres associada a aquesta condició i se salta la resta de l'escala. Si cap condició és veritable s'executaran les ordres associades a l'*else* final. Si totes les condicions són falses i no hi ha *else* final, no fa res.

En realitat, l'estructura *if-then-elif* no és més que una manera abreviada d'escriure el mateix codi amb estructures *if-else* imbricades, és a dir, és equivalent al següent:

```
if condició1; then
  ordres1
else
  if condició2; then
    ordres2
  else
    if condició3; then
      ordres3
    else
      if
        ...
      else
        ordresN
      fi
    fi
  fi
fi
```

Per exemple, el programa següent llegeix un nombre del teclat i mostra per pantalla si és més petit, més gran o igual que 10, utilitzant una estructura *if-elif-else*:

Fixeu-vos en la forma d'escala que adopta el codi quan s'implementa amb estructures *if-else* i es tabula de la manera habitual. Per això es coneix amb el nom d'escala *if-else*.

```
echo "Introdueix un número: "
read X
if [ $X -lt 10 ]; then
  echo "El número és més petit que 10."
elif [ $X -gt 10 ]; then
  echo "El número és més gran que 10."
else
  echo "El número és igual a 10."
fi
```

Es podria haver escrit el mateix codi utilitzant estructures *if* i *if-else* de la manera següent:

```
echo "Introdueix un número: "
read X
if [ $X -lt 10 ]; then
  echo "El número és més petit que 10."
else
  if [ $X -gt 10 ]
  then
    echo "El número és més gran que 10."
  else
    echo "El número és igual a 10."
  fi
fi
```

Si el nombre de condicions és elevat, l'estructura *if-elif-else* permet compactar el codi.

Exemple d'utilització d'una escala if-else

Feu un shell script anomenat nota.sh que demani el valor d'una nota (un nombre enter) i ens digui si la nota és una D (0, 1, 2), una C- (3, 4), una C+ (5, 6), una B (7, 8) o una A (9, 10).

```
#!/bin/bash
#
# nota.sh
#
```

```
# Demana el valor d'una nota (enter) i diu si és
# D (0, 1, 2), C- (3 o 4), C+ (5 o 6),
# B (7 o 8) o A (9 o 10).
#
echo "Quina nota tens (un enter de 1 a 10)?"
read NOTA
if [ $NOTA -lt 0 ] || [ $NOTA -gt 10 ]; then
  echo "Nota fora de rang."
elif [ $NOTA -lt 3 ]; then
  echo "Tens una D."
elif [ $NOTA -lt 5 ]; then
  echo "Tens una C-."
elif [ $NOTA -lt 7 ]; then
  echo "Tens una C+."
elif [ $NOTA -lt 9 ]; then
  echo "Tens una B."
else
  echo "Tens una A."
fi
```

Estructura case

L'estructura *case* és útil quan tenim múltiples bifurcacions **basades en avaluacions de cadenes de caràcters**. La seva sintaxi és:

```
case "$NOM_VAR" in
  patró1)
    ordre1
    ...
    ordreN
    ;;
  patró2)
    ordre1
    ...
    ordreN
    ;;

  ...

  patróN)
    ordre1
    ...
    ordreN
    ;;
esac
```

Es fa una comparació seqüencial de la cadena que hi ha darrera de la paraula clau *case* amb els patrons. Quan es troba la primera coincidència s'executa la corresponent llista d'ordres i cap altra. Si no es troba cap coincidència no s'executa res.

Veiem-ne un exemple:

Noteu l'ús del patró *, que es pot utilitzar per indicar patrons per defecte.

```
echo "Tria una opció de 1 a 3: "
read OPCIO
case "$OPCIO" in
  1)echo "Has triat l'opció 1." ;;
  2)echo "Has triat l'opció 2." ;;
  3)echo "Has triat l'opció 3." ;;
  *)echo "Opció incorrecta."
esac
```

Els patrons són cadenes de caràcters i podem utilitzar els mateixos caràcters amb significat especial que utilitzem en la generació de noms de fitxers:

- *, per indicar coincidència amb qualsevol cadena de caràcters, inclòs el caràcter *null*.
- *?*, per indicar coincidència amb qualsevol caràcter simple.
- *[ ]*, per indicar coincidència amb qualsevol dels caràcters que hi hagi entre els claudàtors. Els caràcters de la llista van seguits un darrere de l'altre , no van separats per comes ni per espais ni per cap altre caràcter delimitador. S'accepten rangs amb el símbol menys (-).

També es pot utilitzar el símbol | si es vol coincidència amb més d'un patró (s'interpreta com un *o* lògic):

```
case "$NOM_VAR" in
  patró1|patró2|patró3)
    ordre1
    ...
    ;;
  patró4|patró5)

  ...
esac
```

L'exemple següent mostra l'ús de patrons amb caràcters amb significat especial. Noteu que en alguns casos pot haver més d'una manera d'expressar un mateix patró, per exemple, el patró [0-2] també es podria haver escrit *[012]* o bé *0|1|2*.

Exemple d'utilització d'una estructura case i ús de patrons

Feu el mateix guió de shell notes.sh que heu implementat amb una escala if-else, però ara utilitzant una estructura case.

```
#!/bin/bash
#
# nota2.sh
#
# Demana el valor d'una nota (enter) i diu si és
# D (0, 1, 2), C- (3 o 4), C+ (5 o 6),
# B (7 o 8) o A (9 o 10).
#
echo "Quina nota tens (enter de 1 a 10)?"
read NOTA
case "$NOTA" in
  [0-2])
    echo "Tens una D."
    ;;
```

```
  [34])
    echo "Tens una C-."
    ;;
  [56])
    echo "Tens una C+."
    ;;
  [78])
    echo "Tens una B."
    ;;
  9|10)
    echo "Tens una A."
    ;;
  *)
  echo "Nota fora de rang."
esac
```

2.6.2 Operadors && i ||

Els operadors de control && (AND) i || (OR) permeten fer un control de flux bàsic.

Operador &&

L'**operador *&&*** s'utilitza per executar una ordre, i, si té èxit (codi de sortida igual a zero), executar la propera ordre de la llista. La sintaxi és:

```
ordre1 && ordre2
```

L'ordre2 s'executa si i només si l'ordre1 torna un estat de sortida de zero (vertader). En altres paraules, s'executa ordre1 si el codi de sortida és igual a zero, llavors s'executa ordre2. És a dir, és equivalent a la utilització de l'estructura *if* així:

```
if ordre1; then
  ordre2
fi
```

Per exemple:

```
rm /tmp/fitxer && echo "Fitxer esborrat."
```

L'ordre *echo* només s'executa si l'ordre *rm* s'executa amb èxit (amb un codi de sortida de zero). Si el fitxer s'elimina correctament, l'ordre *rm* dóna un codi de sortida igual a zero i a continuació es mostra el missatge "Fitxer esborrat". Per evitar els possibles missatges d'error de l'ordre *rm*, podem redirigir la sortida d'errors a /dev/null així:

```
rm /tmp/fitxer 2>/dev/null && echo "Fitxer esborrat."
```

Exemple d'utilització de l'operador &&

Escriviu les ordres necessàries per comprovar si hi ha un directori anomenat /tmp/foo i donar un missatge d'error si no existeix.

```
test ! -d /tmp/foo && echo "El directori no existeix."
```

O amb la forma abreviada de test:

```
[ ! -d /tmp/foo ] && echo "El directori no existeix."
```

Operador ||

L'operador `||` s'utilitza per executar una ordre, i, si no té èxit (codi de sortida diferent de zero), executar la propera ordre de la llista. La sintaxi és:

```
ordre1 || ordre2
```

L'ordre2 s'executa si i només si l'ordre1 retorna un estat de sortida diferent de zero. En altres paraules, s'executa ordre1, si el codi de sortida de ordre1 és diferent de zero, llavors s'executa ordre2. Per exemple:

```
cat /etc/shadow 2>/dev/null || echo "No s'ha pogut obrir el fitxer."
```

L'ordre cat intentarà llegir el fitxer /etc/shadow i si falla mostrarà el missatge "No s'ha pogut obrir el fitxer".

Exemple d'utilització de l'operador OR

Escriviu les ordres necessàries per cercar el nom d'un usuari anomenat "messi" al fitxer /etc/passwd i donar un missatge si no es troba.

```
grep "^messi" /etc/passwd || echo "No s'ha trobat messi a /etc/
    passwd."
```

Combinació dels operadors && i ||

La llista d'ordres unides per operadors `&&` i `||` pot ampliar-se. Sovint s'utilitza la combinació dels operadors de la manera següent:

```
ordre_condició && ordre_cert || ordre_fals
```

És a dir, s'executa l'*ordre_condició.* Si el codi de sortida és:

- cert (0), llavors s'executa ordre_cert.
- fals (diferent de 0), llavors s'executa ordre_fals.

És a dir, és equivalent a la utilització de l'estructura *if-else* així:

```
if ordre_condicio; then
  ordre_cert
else
  ordre_fals
fi
```

Exemple d'utilització dels operadors OR i AND combinats

Escriviu les ordres necessàries per cercar el nom d'un usuari anomenat "messi" al fitxer /etc/passwd i donar missatges diferents segons si es troba o no.

```
grep "^messi" /etc/passwd || echo "No s'ha trobat messi a /etc/
    passwd." && echo "Trobat messi a /etc/passwd."
```

Escriviu les ordres necessàries per assegurar-vos que el superusuari root és qui està executant el guió de shell i donar un missatge depenent de si ho és o no.

```
test $(id -u) -eq 0 && echo "S'executa l'script amb el
    superusuari root." || echo "Només l'usuari root pot executar
    aquest script."
```

Escriviu les ordres necessàries per comprovar si el fitxer /etc/resolv.conf existeix i donar un missatge si hi és i un altre missatge si no hi és.

```
[ -f /etc/resolv.conf ] && echo "El fitxer /etc/resolv.conf
    existeix." || echo "El fitxer /etc/resolv.conf no existeix."
```

2.6.3 Estructures iteratives

Les estructures iteratives són aquelles que ens permeten executar diversos cops una part de codi.

Estructura while

L'estructura *while* permet l'execució repetitiva d'unes sentències sempre que l'ordre de control del bucle *while* sigui certa (codi de sortida igual a zero). La sintaxi és:

```
while condició; do
  ordre1
  ordre2
  ...
  ordreN
done
```

Normalment, s'utilitzen les ordres *test* o *let* per controlar la condició del bucle, però podem utilitzar qualsevol ordre que retorni un valor.

Veiem-ne un exemple senzill, un bucle que mostra per pantalla els números de l'1 al 10:

```
X=1
while (( X <= 10 )); do
  echo X val $X
  (( X=X+1 ))
done
```

Exemple d'utilització de l'estructura de control while

Feu un guió de shell que s'anomeni endevina.sh que assigni un número aleatori entre 1 i 6 a una variable N i que a continuació ens demani que l'encertem. El programa finalitza quan encertem el número i ens diu quants intents hem necessitat per endevinar-lo.

```
#!/bin/bash
# endevina.sh
# Joc per endevinar un número entre 1 i 6
#
INTENTS=1
# Calculem un número aleatori entre 1 i 6
# Mòdul 6 dóna un número entre 0 i 5
(( N = $RANDOM % 6 + 1 ))
echo –n "Endevina un número entre 1 i 6: "
read TRIA
while (( "$TRIA" != $N )); do
  echo
  echo –n "El número no és $TRIA, torna–ho a intentar: "
  read TRIA
  (( INTENTS++ ))
done
echo
echo "L'has endevinat, era el $N!!"
echo "Has necessitat $INTENTS intents."
exit 0
```

Estructura until

L'estructura *until* és idèntica a *while*, excepte que la condició és negada. La llista d'ordres s'executa sempre que la condició retorni un codi de sortida diferent de zero.

```
until condició; do
  ordre1
  ordre2
  ...
  ordreN
done
```

Per exemple:

```
X=1
until (( X > 10 )); do
  echo X val $X
  (( X=X+1 ))
done
```

Estructura for

L'estructura *for* permet especificar una llista de valors i executar les sentències per a cada valor de la llista. La sintaxi d'aquest bucle és la següent:

```
for NOM_VAR in llista_de_valors; do
  ordre1
  ordre2
  ...
  ordreN
done
```

La llista de valors és una seqüència de valors separats per espai o tabulador que s'aniran assignant a la variable *NOM_VAR* en cada iteració del bucle *for*. Per tant, les sentències que hi ha entre les paraules reservades *do* i *done*, s'executaran tantes vegades com valors hi hagi a la llista de valors.

La construcció clàssica de bucle *for* del *shell* difereix significativament de la del seu homòleg C i d'altres llenguatges de programació.

Per exemple, el següent és un bucle *for* que simplement mostra el valor de cada un dels ítems de la llista de valors (mostra els números de l'1 al 5):

```
for X in 1 2 3 4 5; do
  echo $X
done
```

L'estructura *for* és un mecanisme de bucle molt flexible. Es pot construir un bucle amb qualsevol llista que es pugui generar. Podem generar llistes fàcilment, per exemple, mitjançant la llista de paràmetres passats en la crida al *shell script* o mitjançant la substitució d'ordres. Així, l'exemple que acabem de veure es podria haver expressat obtenint la llista de valors amb el resultat d'executar l'ordre *seq* de la manera següent:

```
for X in $(seq 1 5); do
  echo $X
done
```

I en el cas que el nostre guió s'executi amb paràmetres, per exemple així:

```
nom_guió 1 2 3 4 5
```

El codi següent tindrà el mateix efecte que els anteriors:

```
for X in $*; do
  echo $X
done
```

La variable especial *$** conté la llista de paràmetres passats en la crida al programa. Teniu més informació al respecte a l'apartat "Paràmetres i variables especials" d'aquesta unitat.

Exemple d'utilització de l'estructura de control for

Feu un guió de shell que calculi l'ocupació de disc de cada directori de /home. Podeu obtenir la llista d'aquests directoris executant l'ordre ls dins del directori /home. Per calcular l'ocupació del directori podeu utilitzar l'ordre du (disk usage):

```
#!/bin/bash
# ocupacio.sh
# Calcula l'ocupació dels directoris de /home
#
cd /home
for DIR in $(ls); do
```

```
  if [ -d $DIR ]; then
    # Calcular espai ocupat pel directori
    du -sh $DIR
  fi
done
```

Feu el mateix guió de shell, però rebent per paràmetre el nom dels directoris dels quals volem conèixer l'espai que ocupen al disc.

```
#!/bin/bash
# Nom: espai.sh
# Mostra ocupació dels directoris rebuts per paràmetre
# Crida: espai.sh DIR1 DIR2 DIR3 ...
#
for DIR in $*; do
  if [ -d $DIR ]; then
    # Calcular espai ocupat pel directori
    du -sh $DIR
  else
    echo "Error: $DIR no és cap directori."
  fi
done
```

El *shell* Bash també permet utilitzar la construcció *for* a l'estil del llenguatge de programació C amb una sintaxi com la següent:

La sintaxi de *for* a l'estil de C que permet Bash no funciona amb sh (*shell* Bourne).

```
for (( expr1 ; expr2 ; expr3 )) ; do
  ordre1
  ordre2
  ...
  ordreN
done
```

En aquest cas les expressions només poden ser expressions aritmètiques, no poden ser qualsevol ordre i s'utilitzen amb el propòsit següent:

- *expr1*: per inicialitzar el bucle.
- *expr2*: per comprovar si la llista d'ordres entre *do* i *done* s'ha d'executar.
- *expr3*: per canviar la condició després de cada iteració del bucle.

És equivalent a:

```
(( expr1 ))
while (( expr2 )); do
  ordre1
  ordre2
  ...
  ordreN
  (( expr3 ))
done
```

Per exemple, per mostrar els números de l'1 al 5:

```
for ((x=1; x<=5; x++))
{
  echo $x
}
```

Veiem un altre exemple en què utilitzem bucles *for*imbricats (un bucle *for* dins d'un altre bucle *for*):

```
#!/bin/bash
#
# estrelles.sh
# Exemples de bucles for a l'estil de C
#
N=10
for (( i=1; i<=$N; i++ ))
do
  for (( j=1; j<=i; j++ ))
  do
    echo -n " *"
  done
  echo
done
for (( i=$N; i>=1; i-- ))
do
  for (( j=1; j<=i; j++ ))
  do
    echo -n " *"
  done
  echo
done
```

El resultat del guió de *shell* és:

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

2.7 Funcions

El llenguatge del *shell* Bash permet definir funcions, encara que amb una implementació una mica limitada.

Una **funció** és un bloc de codi que implementa un conjunt d'operacions, una "caixa negra" que duu a terme una tasca específica. La utilització de funcions en un *shell script* és una manera d'agrupar una serie d'ordres perquè puguin ser executades posteriorment utilitzant un sol nom.

Considerarem l'ús de funcions en els casos següents:

- Quan el programa de *shell* és complex, és recomanable fer una programació modular, és a dir, agrupar codi en funcions o mòduls per estructurar el programa.
- Sempre que tinguem codi repetitiu, és a dir, una part de codi que s'ha d'utilitzar més d'una vegada, cal considerar l'ús d'una funció.

Fitxers de funcions

Hi ha un munt de scripts en el sistema que utilitzen les funcions com una manera estructurada d'agrupar una sèrie d'ordres. En alguns sistemes Linux, per exemple, es troba l'arxiu /etc/rc.d/init.d/functions, un arxiu de definició de funcions que s'inclou al principi de tots els scripts d'inici. Usant aquest mètode, les tasques comunes, com la comprovació de si un procés s'executa, iniciar o aturar un dimoni, etc., només cal escriure-les una vegada.

Com a administradors del sistema, podeu crear el vostre propi fitxer de funcions, per exemple /usr/local/bin/funcions, amb totes les funcions que utilitzareu amb freqüència en els vostres guions de shell. Després, en cada guió que hagi d'utilitzar les funcions, només cal que escriviu la línia següent:

```
. /usr/local/bin/funcions
```

No deixeu de posar el punt i l'espai (equivalent a l'ordre source) abans del nom del fitxer que conté les funcions perquè el shell actual reconegui les funcions.

La sintaxi per definir una funció és:

```
1 function nom_de_la_funcio {
2   codi de la funcio
3 }
```

O bé:

```
1 nom_de_la_funcio () {
2   codi de la funcio
3 }
```

Cal tenir en compte les consideracions següents:

- El nom de la funció ha de ser únic en el *shell* o script.
- El caràcter *{* d'obertura de la funció pot estar a la segona línia.
- La definició de la funció ha d'estar feta abans de la primera crida a la funció que hi hagi al programa.

> Es recomana fer totes les definicions de les funcions necessàries abans del codi principal del guió que s'executa, tret que hi hagi raons concretes per no fer-ho.

Això dóna una visió molt millor del programa i assegura que tots els noms de les funcions es coneixen abans de ser utilitzats. Generalment, l'estructura bàsica d'un guió de *shell* que implementa funcions és:

```
#!/bin/bash

VARIABLES_CONFIGURACIÓ

DEFINICIO_FUNCIONS

CODI_PRINCIPAL
```

Per cridar una funció des del programa de shell, simplement posem el nom de la funció, igual com fem amb qualsevol ordre. Quan es crida la funció, la llista d'ordres relacionades amb el nom de la funció s'executa.

```
#!/bin/bash
# funcs.sh
# Guió de shell per provar les funcions
#
# Definició de funcions
saludaUsuari() {
  echo "Execució de la funció $FUNCNAME..."
  echo Hola $USER!!
}
mostraData() {
  echo "Execució de la funció $FUNCNAME..."
  echo Avui és $(date)
}
### Programa principal ###
saludaUsuari
mostraData
```

2.7.1 Paràmetres a les funcions

Les funcions accepten paràmetres de la mateixa manera que les ordres o que els *shell scripts*. Per passar paràmetres a una funció només cal que en el moment de la crida a la funció especifiquem al costat del nom els valors dels paràmetres separats per espais. Per exemple:

```
nom_funcio param1 param2
```

La funció referencia els paràmetres rebuts per la posició que ocupen, igual que els paràmetres posicionals: `$1`, `$2`, etc. Tots els paràmetres es passen per valor, és a dir, quan retornem de la funció el valor dels paràmetres no haurà canviat.

Vegeu la descripció dels paràmetres posicionals i les variables especials a l'apartat "Paràmetres i variables especials" d'aquesta unitat.

Exemple de funció amb paràmetres

Feu una funció que faci la suma de dos números rebuts per paràmetre i en mostri el resultat. El programa principal ha de demanar els números per teclat.

```
#!/bin/bash
# suma.sh
# Demana dos números per teclat i en mostra la suma
#
# Definició de funcions
suma() {
  (( TOTAL = $1 + $2 ))
  echo "La suma de $1 i $2 és: $TOTAL"
}
### Programa principal ###
echo -n "Introdueix un número: "
read X
echo -n "Introdueix un altre número: "
read Y
# Crida a la funció suma() amb paràmetres
suma $X $Y
```

En executar una funció, els paràmetres passats a la funció esdevenen els paràmetres posicionals durant l'execució de la funció. Les variables especials *, @ i # també s'actualitzen per reflectir els canvis. La variable especial *0*, que té el nom del guió de *shell*, no canvia. La variable del Bash anomenada *FUNCNAME* s'estableix amb el nom de la funció mentre dura l'execució de la funció.

Quan es completa l'execució de la funció, els valors dels paràmetres posicionals i de les variables especials *, @ i # es restauren als valors que tenien abans de l'execució de la funció. Per tant, si necessitem accedir des d'una funció als valors dels paràmetres posicionals i de les variables especials *, @ i # del programa que crida a la funció, haurem de guardar-los prèviament a la crida de la funció.

Exemple de funció amb paràmetres

Un exemple típic de funció amb paràmetres és una funció per mostrar els missatges d'error d'un guió de shell. Imagineu que esteu fent un guió de shell i que teniu múltiples punts on el programa dóna missatges d'error:

```
if [ -z "$DIR" ] ; then
  echo "$0: especifiqueu el nom del directori."
  exit 1
fi
if [ ! -d $DIR ] ; then
  echo "$0: el directori no existeix."
  exit 1
fi
```

Quan això succeeix, és millor que creeu una funció per a aquest propòsit, per exemple:

```
mostraError() {
  echo "$0: $*"
  exit 1
}
...
if [ -z "$DIR" ] ; then
  mostraError "especifiqueu el nom del directori."
fi
if [ ! -d "$DIR" ] ; then
  mostraError "el directori no existeix."
fi
```

Noteu que la variable especial `$0` ha mantingut el valor dins de la funció i que la variable `$*` ha agafat el valor de la llista de paràmetres passats a la funció.

2.7.2 Codis de retorn

Les funcions sempre tornen un valor al *shell* que les crida anomenat *codi de retorn*, que és anàleg al codi de sortida de les ordres. El codi de retorn d'una funció pot ser especificat de manera explícita amb l'ordre return; altrament, el valor retornat per la funció és el valor del codi de sortida de l'última ordre executada. El codi de retorn de la funció pot utilitzar-se en el *shell script* mitjançant *$?*, de la mateixa manera que el codi de sortida de qualsevol altra ordre.

Ordre return

L'ordre **return** interromp l'execució d'una funció. La sintaxi de l'ordre és:

```
return [n]
```

De manera opcional accepta un nombre enter com a paràmetre que és retornat al guió de *shell* que fa la crida com el codi de retorn de la funció i és assignat a la variable *$?*. Si no indiquem paràmetre, el valor retornat és el valor del codi de retorn de l'última ordre executada abans del *return*. L'ordre return utilitzada fora del context d'una funció fa el mateix que l'ordre *exit*.

Exemple de funció amb codi de retorn

Modifiqueu la funció suma.sh perquè en lloc de mostrar el resultat de la suma dels dos números per pantalla, el retorni mitjançant l'ordre return. En el programa principal accediu al valor retornat per la funció amb la variable `$?`.

```
#!/bin/bash
# suma.sh
# Demana dos nombres per teclat i mostra la suma
#
# Definició de funcions
# Rep dos nombres i retorna la suma
suma() {
  (( TOTAL = $1 + $2 ))
  return $TOTAL
}
### Programa principal ###
echo –n "Introdueix un número: "
read X
echo –n "Introdueix un altre número: "
read Y
suma $X $Y
RESULTAT=$?
echo "La suma de $X i $Y és: $RESULTAT"
```

3. Planificació i automatització de tasques

En l'administració de sistemes sol ser necessària l'automatització de certes tasques a causa de la seva naturalesa repetitiva. Algunes d'aquestes tasques s'han de poder executar de manera no interactiva i s'han de poder planificar per fer-les sota algunes condicions, com ara, en horaris de menys ús de la màquina. Per dur a terme aquest tipus de feines com a serveis periòdics i programats hi ha diversos sistemes que ens permeten construir una mena d'agenda de tasques, és a dir, una planificació d'execució de les tasques.

3.1 Planificació de tasques

Un planificador de tasques és un programari que permet l'execució de tasques de manera desatesa (sense la intervenció de l'usuari) i d'acord amb les condicions descrites en funció d'un calendari o d'altres esdeveniments.

Les característiques que s'esperen d'un programari de planificació de tasques són:

- Permetre l'execució de tasques de manera automàtica.
- Tenir interfícies que ajudin a definir els fluxos de treballs o dependències entre tasques.
- Tenir interfícies que permetin la monitorització i el seguiment de les execucions de les tasques.
- Disposar d'algun mecanisme de prioritats o cues per controlar l'ordre d'execució dels treballs no relacionats.

Qualsevol programari que inclou totes o alguna d'aquestes característiques el podem considerar com un programari amb capacitats de planificació de tasques.

Des del punt de vista històric, podem distingir dues èpoques principals pel que fa als planificadors de tasques: l'era de l'ordinador central (*mainframe*) i l'era dels sistemes oberts i la informàtica distribuïda.

L'època de l'ordinador central es caracteritza pel desenvolupament de solucions de planificació sofisticades que inclouen característiques de planificació avançades i que formen part de les eines de gestió del propi sistema del *mainframe*. En l'ordinador central d'IBM, per exemple, el *job control language* (JCL) oferia des de bon principi la funcionalitat de gestionar dependències entre treballs.

Ordinador central

Un ordinador central o *mainframe* és un ordinador gran, potent i costós. En l'actualitat és utilitzat per grans companyies per a processaments de grans quantitats de dades; per exemple, per al processament de transaccions bancàries.

En l'època dels sistemes oberts, una gran diversitat de programari ofereix capacitats bàsiques de planificació de tasques:

- La majoria de sistemes operatius, com ara Windows o Unix i derivats, proporcionen eines de planificació que segueixen un model senzill per a l'execució de les tasques basat en els esdeveniments d'un calendari o en la càrrega del sistema en un moment determinat.
- Els serveis d'allotjament web ofereixen capacitats de planificació de treballs per mitjà d'un tauler de control o una solució de tipus webcron.
- D'altres aplicacions de diverses àrees, com ara programaris de còpies de seguretat, ERP, etc., incorporen facilitats per planificar les tasques pròpies de l'aplicació.

ERP

Els sistemes de planificació de recursos empresarials o ERP (de l'anglès enterprise resource planning) pretenen integrar totes les dades i processos d'una organització en un sistema unificat. Un sistema ERP típic utilitza múltiples components de programari i maquinari per aconseguir la integració de la producció, l'inventari, la distribució, els enviaments, les factures, la comptabilitat, les comandes, els lliuraments, els pagaments, etcètera.

En general, els sistemes operatius i aplicacions de l'era dels sistemes oberts no ofereixen la possibilitat de planificar l'execució de tasques més enllà d'una única instància de sistema operatiu o fora de l'àmbit del programa específic. A mesura que el nombre i la varietat de plataformes s'estén, les capacitats bàsiques de planificació de tasques es queden curtes i sorgeix la necessitat de sistemes de planificació avançats propers als planificadors estàndards dels ordinadors centrals i que a més proporcionin la possibilitat d'integrar diferents tipus de plataformes.

3.1.1 Sistemes distribuïts de planificació

Els sistemes distribuïts de planificació de treballs s'utilitzen en organitzacions amb un nombre de plataformes elevat i variat, per simplificar la gestió de la càrrega de treball de tota l'empresa i per tenir la capacitat de definir treballs en sistemes distribuïts, assegurant que s'executen en el temps i en la seqüència correcta.

En aquest tipus de sistemes se sol parlar de "gestió de càrrega de treball" en lloc de "planificació de tasques".

Normalment, les organitzacions mitjanes i grans disposen d'un nombre elevat de servidors amb diferents sistemes operatius i aplicacions, i un conjunt molt variat i molt ampli de treballs per executar, que poden tenir dependències complexes entre ells més enllà de la dimensió del temps o la càrrega del sistema. Per satisfer les demandes d'aquest tipus de centres, hi ha programaris específics de planificació de treballs –o de gestió de càrrega de treball– que són compatibles amb i que integren diverses plataformes i aplicacions i que disposen de característiques avançades com ara:

- Planificació de temps real basada en esdeveniments imprevisibles externs.
- Reinicialització automàtica de tasques i recuperació en cas de fallades.
- Alerta i notificació al personal d'operacions.

- Generació d'informes d'incidents.
- Registres d'auditoria per a propòsits de compliment de normatives.

En aquests sistemes hi ha diversos esquemes per decidir quin treball s'ha d'executar en un moment determinat. Per exemple, alguns paràmetres que poden ser considerats són:

- La prioritat de la feina.
- Els recursos de còmput existents.
- L'existència de la clau de llicència, si el treball està utilitzant programari amb llicència.
- El temps d'execució assignat a un usuari.
- El nombre de treballs permesos simultàniament a un mateix usuari.
- El temps estimat d'execució de la tasca.
- El temps transcorregut de la tasca.
- La disponibilitat de perifèrics.
- L'ocurrència d'esdeveniments requerits.

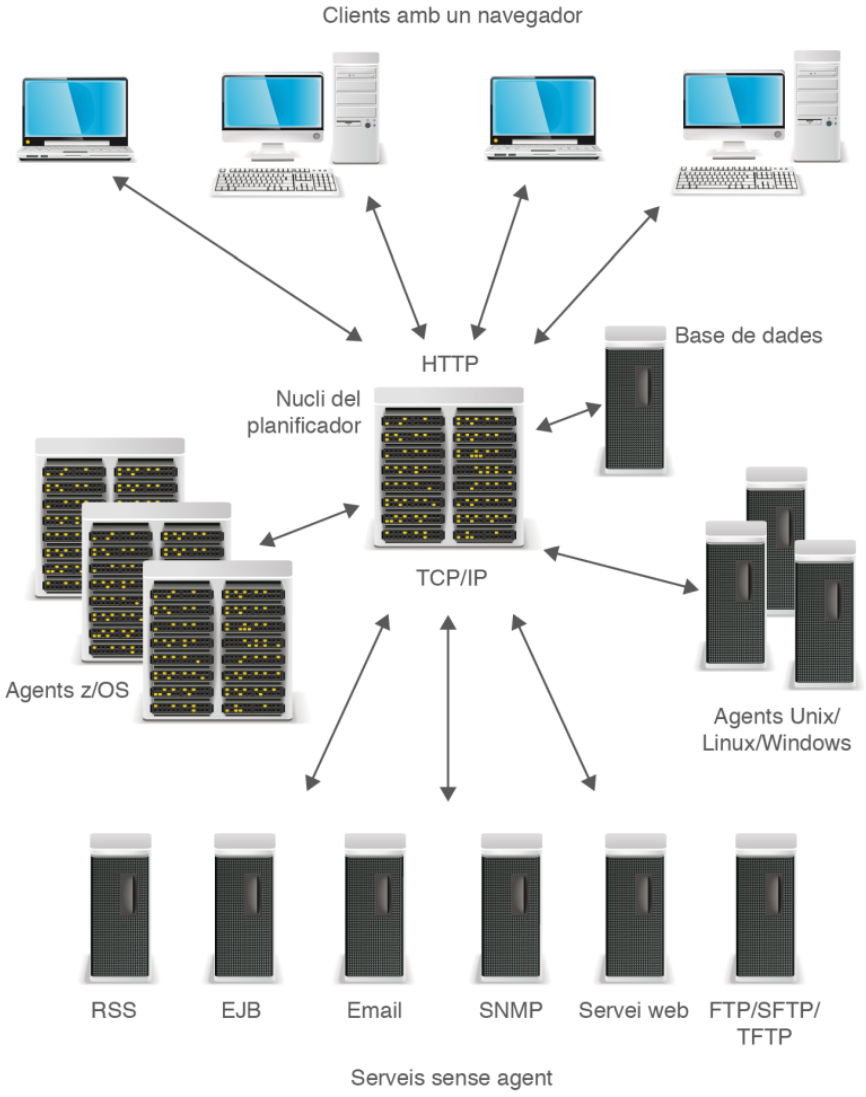
Aquests productes són complexos i solen oferir una interfície gràfica d'usuari amb un únic punt de control per a la definició i seguiment de les execucions de les tasques de tota la xarxa distribuïda d'ordinadors. La interfície d'usuari del programari sol ser de tipus web i normalment també incorpora una interfície de línia d'ordres.

La figura 3.1 mostra gràficament un entorn amb un sistema distribuït de planificació de treballs i una arquitectura típica implementada en aquests tipus de planificadors anomenada *arquitectura de mestre/agent*. El nucli del programari de planificació de treballs s'instal·la en una sola màquina (mestre), mentre que en les màquines de producció només s'instal·la un component molt petit (agent) que espera les ordres del mestre, les executa i li retorna el codi de sortida. Determinats serveis (FTP, SNMP, programes Java, etc.) no requereixen cap agent. Qualsevol màquina o dispositiu client amb un navegador permet el control i seguiment de les tasques.

Hi ha un ampli ventall de programari especialitzat en planificació de tasques per a sistemes distribuïts. L'elecció d'una eina sòlida continua sent essencial per a qualsevol empresa amb un nombre significatiu de servidors. Aquestes eines poden ser molt cares quan s'adquireixen de venedors tradicionals de programari de gestió de les TIC (tecnologies de la informació i la comunicació), però també hi ha eines de venedors independents a preus més competitius.

Vegeu una comparativa de programaris de planificació de tasques a la secció Adreces d'interès del web del mòdul.

FIGURA 3.1. Planificador de tasques distribuït



3.1.2 Planificadors del sistema operatiu

Les eines bàsiques de planificació proporcionades amb sistemes operatius com ara Unix i derivats o Windows s'utilitzen en entorns amb un nombre reduït de servidors que no requereixen planificació de tasques distribuïda. A continuació descriurem les més utilitzades en l'actualitat.

Sistema operatiu Windows

El planificador de tasques dels sistemes Windows de Microsoft s'anomena **task scheduler**, s'instal·la automàticament en el sistema operatiu com un servei i s'inicia cada vegada que el sistema és arrencat.

El task scheduler es pot utilitzar des de la línia d'ordres amb el programa `schtasks.exe`, però el més habitual és fer servir la interfície gràfica d'usuari

a la qual hi accedim a partir de *Panell de control* > *Sistema i seguretat* > *Eines administratives* > *Programador de tasques*.

La versió actual (2012) del task scheduler és la 2.0. Va ser introduïda amb Windows Vista i incorporada a Windows Server 2008. En aquesta versió la interfície d'usuari té un nou disseny basat en la consola de gestió de Microsoft (MMC, *Microsoft management console*) i les capacitats de planificació de tasques es milloren respecte la versió anterior, la 1.0.

La consola MMC

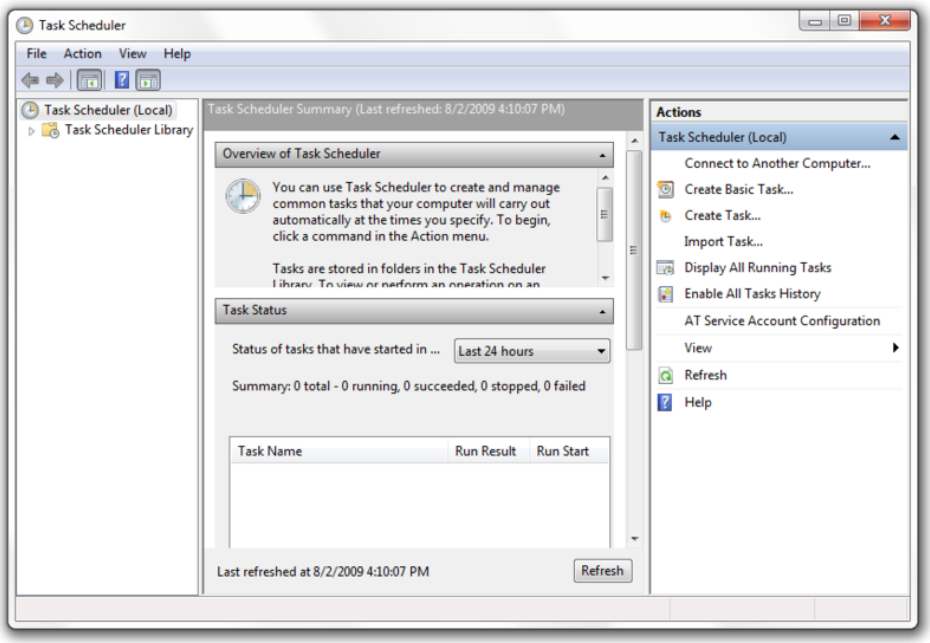
MMC (Microsoft management console) és un component de Windows 2000 i dels seus successors que proveeix als administradors del sistema una interfície per configurar i monitoritzar el sistema.

El task scheduler 2.0 permet seleccionar el tipus d'acció que volem planificar d'entre les següents:

- Executar una aplicació o guió de *shell*.
- Enviar un correu electrònic.
- Mostrar una caixa de diàleg amb un missatge.

La figura 3.2 mostra una finestra amb el task scheduler executant-se en un sistema operatiu Windows 7.

Figura 3.2. Planificador de tasques de Windows 7



Les accions es poden planificar perquè siguin executades quan es produeixi alguna de les condicions següents:

- D'acord amb una programació (a una hora específica una vegada, diàriament, setmanalment o mensualment).
- En obrir o tancar la sessió.
- En iniciar el sistema.
- Quan el sistema entra en estat inactiu.

- En produir-se un esdeveniment determinat del sistema.
- En bloquejar-se o desbloquejar-se l'estació de treball.

Sistemes de tipus Unix

En els sistemes de tipus Unix hi ha diverses eines relacionades amb la planificació de tasques:

- El programa cron, un planificador basat en el temps que assumeix que el sistema està sempre en funcionament.
- El programa anacron, que fa les tasques de cron però sense assumir que el sistema està sempre en funcionament. Aquest programa no substitueix a cron, és una eina complementària.
- L'ordre *at*, la qual es fa servir per fer execucions retardades i planificar tasques que es volen executar una sola vegada a una hora determinada en el futur.

Càrrega del sistema

La càrrega del sistema és un paràmetre que ens indica el grau d'activitat de l'ordinador. En sistemes de tipus Unix podem veure aquesta càrrega de manera interactiva amb l'ordre `top`.

Mentre que cron, anacron i *at* són eines per executar tasques quan arriba un determinat moment en el temps, hi ha d'altres eines que permeten executar tasques quan es dóna un esdeveniment, per exemple:

- L'ordre *batch* s'utilitza per executar tasques quan el nivell de càrrega del sistema ho permeti.
- El servei incron està dissenyat per executar tasques quan es produeix un esdeveniment en el sistema de fitxers. incron pot examinar un fitxer específic o un directori sencer i reaccionar a diferents esdeveniments com ara la creació, la modificació o l'esborrat de fitxers, etc.

Si bé tenim a l'abast aquestes i altres eines, el planificador per excel·lència és el cron i és en el que ens centrarem.

3.2 El planificador cron

Cron és el nom del programa que permet als usuaris de sistemes Unix i derivats planificar l'execució d'ordres o guions de *shell* de manera automàtica a una data i temps específics. És utilitzat sovint pels administradors de sistemes com a eina per automatitzar les tasques de manteniment, com ara les còpies de seguretat, però pot ser utilitzat per a qualsevol altre objectiu.

El nom de cron ve de cronògraf, l'aparell per enregistrar intervals de temps.

Cron ha estat reescrit diversos cops durant la seva història i en podem trobar implementacions, com ara la d'AT&T, que poden diferir lleugerament del **cron vixie**, que és el que utilitzarem i descriurem en els apartats següents.

Cron vixie

El seu autor, Paul Vixie, ho és també de Bind, el servidor de DNS lliure més utilitzat.

3.2.1 Iniciar el servei cron

Cron és un dimoni (servei) i generalment s'instal·la automàticament en fer la instal·lació del sistema operatiu i queda configurat per ser iniciat amb l'arrencada del sistema. Es pot comprovar l'estat del servei executant l'ordre:

```
1 /etc/rc.d/init.d/cron status
```

Executeu man cron per veure la pàgina de manual de l'ordre cron.

També podem veure si el dimoni està en execució mitjançant l'ordre *ps*:

```
1 ps –ef | grep cron
```

Si per alguna raó cron no està funcionant es pot iniciar el servei amb el superusuari *root* i amb el guió de *shell* d'inicialització corresponent:

```
1 /etc/rc.d/init.d/cron start
```

En altres sistemes potser anomenen al dimoni *crond* (*cron daemon*).

3.2.2 El fitxer de crontab

Anomenem genèricament el fitxer que utilitzem per configurar la planificació de les tasques que ha d'executar cron *fitxer de **crontab***.

Un fitxer de crontab conté instruccions per al servei cron en la forma general: "Executa aquesta ordre a aquesta hora en aquesta data".

La planificació de tasques amb cron es pot fer de dues maneres, a nivell d'un usuari particular i a nivell de tot el sistema i els fitxers de crontab en cada cas difereixen lleugerament.

Executeu `man 5 crontab` per veure la pàgina de manual sobre el format dels fitxers de crontab.

Quan el cron és utilitzat **a nivell d'usuari**, les línies d'un fitxer de crontab es formen amb sis camps separats per espais o tabuladors de la manera següent:

```
1 minut hora diaDelMes mes diaSetmana ordre
```

Els cinc primers camps indiquen el calendari d'execució i el sisè camp especifica la tasca que s'ha d'executar. Cada línia correspon a una tasca i no hi ha límit de tasques.

En el cas que el cron sigui utilitzat **a nivell de sistema**, afegim a les línies del fitxer de crontab un sisè camp per indicar el nom de l'usuari que ha d'executar l'ordre:

```
1 minut hora diaDelMes mes diaSetmana usuari ordre
```

En tots els casos, el significat de cada camp és el següent:

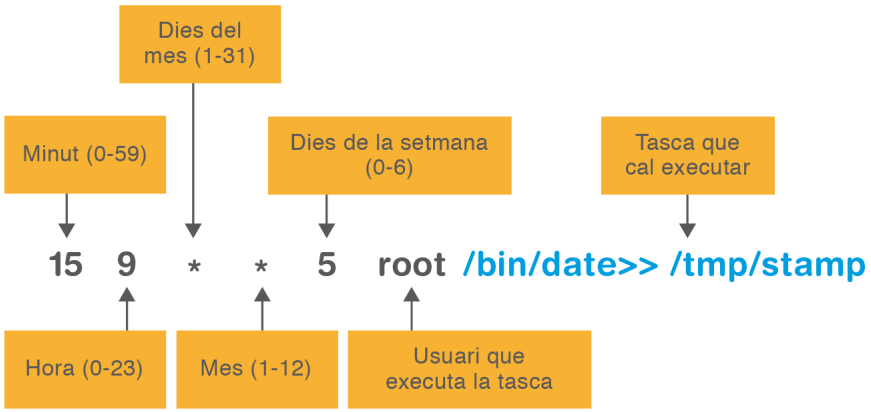
- *minut*: indica el minut de l'hora en què l'ordre serà executada. Ha de ser un valor entre 0 i 59.
- *hora*: indica l'hora en què l'ordre serà executada. Ha de ser un valor entre 0 i 23, éssent 0 la mitjanit.
- *diadelmes*: indica el dia del mes en què l'ordre serà executada. Ha de ser un valor entre 1 i 31.
- *mes*: Indica el mes en què l'ordre serà executada. Pot ser indicat numèricament (un valor entre 1 i 12) o amb les tres primeres lletres del nom del mes en anglès (*jan*, *feb*, *mar*, etc.).
- *diasetmana*: indica el dia de la setmana en què l'ordre serà executada. Pot ser indicat numèricament (un valor entre 0 i 7, sent tant el 0 com el 7 el diumenge) o amb les tres primeres lletres del nom del dia en anglès (*mon*, *tue*, etc.).
- *usuari*: indica l'usuari que executarà l'ordre (només a /etc/crontab).
- *ordre*: ordre, script o programa que es vol executar. Aquest camp ocupa fins al final de la línia; pot contenir múltiples paraules i espais.

Es poden utilitzar els símbols especials següents per indicar els valors dels camps:

- Un asterisc, *, indica tots els valors possibles.
- Llistes de valors separats per comes. Per exemple: *1,2,5,9*.
- Rangs de valors separats pel guió. Per exemple: *8-11* (indica 8, 9, 10 i 11).
- Intervals periòdics mitjançant **/valor*. Per exemple: */5 en el camp *minut* indica cada 5 minuts: 5, 10, 15, 20, 25...

La figura 3.3 mostra gràficament el format d'una línia d'un fitxer de crontab de sistema.

FIGURA 3.3. Format d'una línia de crontab de sistema



La taula 3.1 mostra exemples d'expressions temporals de cron.

Taula 3.1. Exemples d'expressions de cron

minut hora diaDelMes mes diaSetmana	Descripció
17 * * * *	En el minut 17 de cada hora de tots els dies.
25 6 * * *	A les 6.25 am cada dia.
47 6 * * 7	A les 6.47 am tots els diumenges.
52 6 1 * *	A les 6.52 am del primer de cada mes.
* 5 * * *	Cada minut de 5 a 5.59 am.
59 11 * 1-3 1,2,3,4,5	A les 11.59 am de dilluns a divendres, de gener a març.
10,30,50 * * * 1,3,5	En el minut 10, 30 i 50 de totes les hores dels dilluns, dimecres i divendres.
*/15 10-14 * * *	Cada quinze minuts de les 10.00 am a les 2.00 pm.
0 */5 1-10,15,20-23 * 3	Cada 5 hores dels dies 1 al 10, del dia 15 i del dia 20 al 23 de cada mes i que el dia sigui dimecres.

Exemple de crontab d'un usuari

El següent és un exemple del contingut d'un fitxer de crontab d'un usuari. Les línies precedides d'un símbol # són comentaris i són ignorades:

```
# Executar 5 minuts després de la mitjanit cada dia
5 0 * * * $HOME/bin/diari.sh
# Executar a les 2.15 pm el primer dia de cada mes
15 14 1 * * $HOME/bin/mensual.sh
```

Hi ha uns valors predefinits que es poden utilitzar per substituir tota l'expressió de cron. La taula 3.2 mostra quins són aquests valors.

Taula 3.2. Valors especials de cron

Entrada	Descripció	Equivalència
@yearly	S'executa un cop a l'any.	0 0 1 1 *
@annualy	Igual que @yearly.	0 0 1 1 *
@monthly	S'executa un cop al mes.	0 0 1 * *
@weekly	S'executa un cop a la setmana.	0 0 * * 0
@daily	S'executa un cop al dia.	0 0 * * *
@midnight	Igual que @daily.	0 0 * * *
@hourly	S'executa un cop cada hora.	0 * * * *
@reboot	S'executa cada vegada que el servei de cron es reinicia, normalment coincidirà amb la reinicialització del servidor.	Sense equivalència.

Hi ha diverses variables d'entorn que són establertes de manera predeterminada pel servei de cron. Per exemple, la variable *SHELL* s'estableix per defecte a /bin/sh per indicar que les tasques s'executin amb aquest *shell*, o la variable *PATH* s'estableix a /usr/bin:/bin.

Les ordres o programes que funcionen en un *shell* interactiu poden no funcionar correctament en ser executades amb cron ja que determinades variables d'entorn no tenen els valors adequats. Al fitxer de crontab es poden canviar els valors per defecte de les variables d'entorn afegint les línies de definició de variables amb la forma NOM_VAR=valor.

Exemple de crontab d'usuari amb definició de variables d'entorn

El següent és un exemple del contingut d'un fitxer de crontab d'un usuari que inclou definició de variables d'entorn:

```
SHELL=/bin/bash
PATH=~/bin:/usr/bin/:/bin
5 0 * * * $HOME/bin/diari.sh
15 14 1 * * $HOME/bin/mensual.sh
```

Planificació de tasques amb el crontab d'usuari

El planificador de tasques cron permet que cada usuari tingui el seu propi fitxer de planificació de tasques o crontab. Els crontab dels usuaris es guarden al directori `/var/spool/cron/crontabs` (aquest directori pot variar segons la versió d'Unix/Linux) amb el nom de l'usuari del sistema que ha generat el fitxer. Per exemple, hi podem trobar un fitxer anomenat root per al crontab del superusuari *root*, un fitxer anomenat alba per al crontab de la usuària *alba*, i així per a cada usuari del sistema.

Executeu `man crontab` per veure la pàgina de manual de l'ordre `crontab`.

En lloc que cada usuari pugui modificar al seu gust els fitxers que hi ha al directori `/var/spool/cron/crontabs`, existeix un programa denominat `crontab` que serveix per gestionar aquests fitxers, però d'una forma controlada.

Amb l'ordre `crontab` podem crear o modificar un fitxer de planificació de tasques, llistar-ne el contingut o esborrar-lo.

La sintaxi de l'ordre `crontab` és la següent:

```
crontab [ -u usuari ] fitxer
crontab [ -u usuari ] { -e | -l | -r }
```

L'opció -u només la pot utilitzar *root* i permet gestionar el crontab d'un altre usuari en lloc del propi.

La primera forma d'ús de crontab permet a l'usuari que executa l'ordre generar un fitxer de crontab a partir d'un fitxer creat prèviament. L'usuari crea un fitxer de text amb les línies corresponents a les tasques que vol planificar amb el format adequat, és a dir:

```
minut hora diaDelMes mes diaSetmana ordre
```

Un cop creat el fitxer de planificació de tasques, l'usuari executa l'ordre *crontab* posant com a argument el nom del fitxer:

```
crontab fitxer
```

Per exemple, la usuària *maria* crea un fitxer anomenat *tasques* que conté la línia següent:

```
0 22 * * * /home/maria/proces.sh
```

I a continuació executa l'ordre:

```
crontab tasques
```

En utilitzar l'ordre *crontab*, en gravar el fitxer es comprova automàticament que la sintaxi és correcta.

Si el fitxer de tasques no conté errades sintàctiques, l'ordre `crontab` generarà el fitxer `/var/spool/cron/crontabs/maria` amb la tasca planificada per la usuària *maria*. En cas contrari, l'ordre dóna un missatge informatiu indicant l'errada trobada i no instal·la el fitxer de crontab.

L'ordre `crontab` és pot utilitzar sense necessitat de rebre un fitxer com a argument. En aquest cas la sintaxi és:

```
crontab -e
```

L'ordre anterior obre l'editor preestablert de l'usuari i permet crear o modificar el fitxer de planificació de tasques directament. En sortir i desar el fitxer, si crontab no hi troba errades, crearà o modificarà el fitxer corresponent a `/var/spool/cron/crontabs` amb el nom de l'usuari.

Editor preestablert

Podeu canviar l'editor preestablert pel sistema amb la variable d'entorn EDITOR o la variable VISUAL. Per exemple:

```
export EDITOR=/usr/bin/vi
```

Amb l'opció *-l*, crontab permet llistar les tasques que té planificades l'usuari:

```
crontab -l
```

Amb l'opció *-r* s'eliminen totes les tasques de cron de l'usuari:

```
crontab -r
```

Exemple de gestió d'un fitxer crontab d'usuari amb l'ordre //crontab//

Genereu un crontab per al vostre usuari que guardi cada minut la data del sistema en un fitxer del vostre directori d'inici anomenat *provacron.log*.

Primerament, modifiqueu la variable d'entorn *EDITOR* per tal que s'obri l'editor gedit enlloc del preestablert pel sistema:

```
export EDITOR=/usr/bin/gedit
```

A continuació executeu l'ordre *crontab* amb l'opció *-e* per editar directament el fitxer de crontab:

```
crontab -e
```

S'obre el gedit amb un fitxer nou buit. Introduïu la línia desitjada:

```
* * * * * date >> /home/usuari/provacron.log
```

Deseu els canvis i sortiu. Si no hi ha cap errada sintàctica en els camps requerits, us sortirà un missatge informant-vos que s'ha instal·lat el vostre crontab.

Amb **l'usuari *root***podeu comprovar que s'ha creat un fitxer /var/spool/cron/crontabs/nom_usuari, que és un fitxer de text que conté el crontab que acabeu de generar.

Podeu verificar que el fitxer provacron.log va guardant a cada minut la data i hora del sistema obrint un nou terminal i executant:

```
tail –f /home/usuari/provacron.log
```

En una altra finestra de terminal, podeu llistar el contingut del vostre crontab amb l'ordre `crontab -l`.

Si hi voleu fer canvis, podeu tornar a editar el vostre fitxer de crontab amb `crontab -e`.

Finalment, després de fer totes les proves, podeu eliminar el vostre crontab amb l'ordre `crontab -r`.

Els fitxers de crontab ubicats al directori `/var/spool/cron/crontabs` són fitxers de text pla. No obstant això, quan treballem amb l'usuari *root* i volem modificar el seu crontab o el de qualsevol altre usuari, no hem d'editar directament aquests fitxers, sinó que hem d'utilitzar l'ordre `crontab` igual que fan la resta d'usuaris per assegurar-nos que no fem errades de sintaxi en els camps de planificació de les tasques.

Planificació de tasques amb el crontab del sistema

El fitxer de crontab del sistema es diu `/etc/crontab`. És un fitxer de text que només pot modificar l'usuari *root*. Té el mateix format que els crontab d'usuari, però s'afegeix un camp a les línies per especificar amb quin usuari s'executa cada tasca.

El format de les línies del fitxer és:

```
minut hora diaDelMes mes diaSetmana usuari ordre
```

Malgrat que tenen el mateix nom, no confongueu el fitxer de configuració `/etc/crontab` amb l'ordre de gestió del cron dels usuaris, `/usr/bin/crontab`.

Els canvis en el fitxer /etc/crontab prenen efecte pel sol fet d'editar-lo i modificar-lo, és a dir, a diferència del que fem amb els crontab d'usuari, no cal executar l'ordre *crontab* per instal·lar-lo.

El fitxer `/etc/crontab` es crea amb un contingut per defecte similar al següent:

```
# /etc/crontab: system–wide crontab
# Unlike any other crontab you don't have to run the 'crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow usercommand
17 * * * * root run–parts ––report /etc/cron.hourly
25 6 * * * root run–parts ––report /etc/cron.daily
47 6 * * 7 root run–parts ––report /etc/cron.weekly
```

```
14 52 6 1 * * root run-parts --report /etc/cron.monthly
```

Les primeres línies sense símbol de comentari són definicions de variables. La variable *SHELL* indica el *shell* amb el qual s'han d'executar les tasques, i la variable *PATH*, els camins dels directoris en els quals cron buscarà les tasques que s'han d'executar.

Després de les variables, hi ha les línies que executen les tasques planificades:

- La primera línia indica que s'executi la tasca en el minut 17 de cada hora de tots els dies.
- La segona línia indica que s'executi a les 6.25 am de cada dia.
- La tercera línia, a les 6.47 am tots els diumenges.
- La darrera, a les 6.52 am del primer de cada mes.

Programa run-parts

L'ordre `run-parts` serveix per executar tots els scripts i programes situats al directori especificat. Executeu `man run-parts` per veure la pàgina de manual de l'ordre.

L'usuari especificat per executar totes les tasques és *root* i l'ordre planificada sempre és `run-parts`, però posant com a argument en cada cas el nom d'un directori diferent:

- /etc/cron.hourly
- /etc/cron.daily
- /etc/cron.weekly
- /etc/cron.monthly

Com a administradors del sistema, podem desar un fitxer executable pel *shell* (guió de *shell*, programa compilat, etc.) dins de qualsevol d'aquests directoris perquè sigui executat una vegada cada hora, dia, setmana o mes, respectivament, a l'hora configurada en el fitxer /etc/crontab. Per exemple, si es deixa un *shell script* dins de `/etc/cron.daily` s'executarà a les 6.25 am de cada dia.

En el fitxer `/etc/crontab` es poden afegir línies addicionals per executar qualsevol altra ordre, però no és recomanable, ja que aquest fitxer pot ser sobreescrit en fer actualitzacions del sistema. És més recomanable posar les tasques en els directoris /etc/cron.hourly, /etc/cron.daily, etc. o bé utilitzar els crontab d'usuari.

Exemple de planificació de tasques amb el fitxer /etc/crontab

Tot i que no és recomanable, podem trobar sistemes en què s'han afegit línies per planificar tasques concretes al fitxer /etc/crontab del sistema. Per exemple:

```
1 0 22 * * * root /usr/local/diari.sh
2 0 8,20 * * * helena -s "Sistema viu" helena@ioc.cat
```

En aquest exemple, la primera tasca l'executa l'usuari *root* i la segona la usuària *helena*. La primera línia indica que s'executi l'script diari.sh tots els dies a les 10 pm. La segona línia indica que s'enviï un correu tots els dies a les vuit del matí i a les vuit del vespre amb l'assumpte "Sistema viu" a l'adreça helena@ioc.cat. És un mètode senzill d'estar assabentat de si un sistema remot està actiu a les hores indicades. Si la usuària no rep un correu en aquestes hores, significa que alguna cosa no va bé.

Crontabs de /etc/cron.d

En alguns sistemes, hi ha també un directori anomenat `/etc/cron.d` on es poden col·locar fitxers de crontab. Els fitxers d'aquest directori tenen el mateix format que el fitxer /etc/crontab, és a dir, les línies inclouen el camp de l'usuari:

```
minut hora diaDelMes mes diaSetmanausuari ordre
```

En general, l'administrador no hauria d'utilitzar el directori /etc/cron.d/. La finalitat d'aquest directori és permetre que les aplicacions que requereixen un control més fi de la seva planificació que el proporcionat pels directoris /etc/cron.daily, /etc/cron.weekly, etc, afegeixin un fitxer propi de crontab a /etc/cron.d. Aquests fitxers han de dur el nom del paquet de programari que els instal·la i sovint són enllaços a fitxers on, tant l'enllaç com el fitxer, tenen com a propietari l'usuari *root*.

Exemple de contingut del directori /etc/cron.d

El següent és un exemple dels fitxers trobats en un directori /etc/cron.d d'un sistema determinat. Llistem el contingut del directori amb l'ordre:

```
ls /etc/cron.d/
```

I obtenim aquesta sortida:

```
anacron mdadm php5
```

Veiem que hi ha tres programaris que han instal·lat un crontab. Examinem el contingut d'un dels fitxers anteriors, per exemple php5, per veure'n el format:

```
cat php5
```

Podem apreciar que les línies no comentades tenen el mateix format que les del fitxer /etc/crontab:

```
# /etc/cron.d/php5: crontab fragment for php5
# This purges session files older than X, where X is defined in
    seconds
# as the largest value of session.gc_maxlifetime from all your
    php.ini
# files, or 24 minutes if not defined. See /usr/lib/php5/
    maxlifetime

# Look for and purge old sessions every 30 minutes
09,39 * * * * root [ -x /usr/lib/php5/maxlifetime ] && [ -d /var/
    lib/php5 ] && find /var/lib/php5/ -type f -cmin +$(/usr/lib/
    php5/maxlifetime) -print0 | xargs -n 200 -r -0 rm
```

Un exemple d'aplicació que trobarem que instal·la un fitxer de crontab al directori /etc/cron.d és anacron. És un programari que no pretén substituir a cron, sinó que funciona amb cron. El sistema de cron implica que la màquina està sempre encesa, cosa que és certa en la majoria dels casos quan parlem de servidors, però per a aquells casos que això no és així, per exemple quan es tracta d'estacions de treball, és millor utilitzar anacron, que verifica si l'acció no es va fer quan l'hauria hagut de fer, i llavors l'executa.

Executeu *man anacron* per saber més sobre el funcionament d'aquest programari.

3.2.3 Sortida de les tasques de cron

El cron es desperta cada minut, examina tots els fitxers de crontab existents i executa les tasques que tenen els camps que es compleixen en aquell precís minut. Qualsevol sortida de la tasca s'envia per correu electrònic a l'usuari que l'executa o, si existeix, a l'usuari indicat en la variable d'entorn del crontab anomenada *MAILTO*.

En planificar tasques de manteniment del sistema, sovint volem guardar el rastre de les accions que es fan en fitxers de registre. Per redirigir la sortida d'una tasca que executem amb cron i enviar-la a un fitxer enlloc de al correu electrònic de l'usuari, utilitzem redireccions, ja sigui dins del guió de *shell* o en escriure l'ordre en la línia del crontab.

Tenim diverses opcions de redirecció. Les més utilitzades són:

- Ignorar la sortida redirigint a /dev/null: ordre > /dev/null
- Crear o afegir a un fitxer de registre: ordre >> nom_fitxer
- Redirigir també la sortida d'errors: ordre >> nom_fitxer 2>&1

Vegeu més opcions de redirecció d'entrada i sortida a l'apartat "Redirecció de l'entrada i la sortida" d'aquesta unitat.

Per exemple:

```
1 5 0 * * * $HOME/bin/diari.sh >> $HOME/tmp/diari.out 2>&1
```

La línia anterior especifica que s'executi cada dia a les 0.05 el guió de *shell* $HOME/bin/diari.sh. La sortida d'aquesta tasca està redirigida amb l'operador >> cap a un fitxer anomenat *$HOME/tmp/diari.out*, de manera que cada vegada que el guió de *shell* s'executa, la seva sortida es va afegint al final del fitxer. La sortida d'errors també està redirigida amb *2>&1* per tal que, si hi ha errors, s'afegeixin al mateix fitxer de sortida.

3.2.4 Notificacions del servei cron

El cron utilitza el servei anomenat ***syslog*** per enregistrar la seva activitat. Per defecte, està configurat per enregistrar un nivell de detall bàsic al fitxer `/var/log/syslog`.

Per exemple, podem trobar un rastre com el següent:

```
Feb 27 12:33:02 saturn crontab[2671]: (umart) BEGIN EDIT (umart)
Feb 27 12:33:23 saturn crontab[2671]: (umart) REPLACE (umart)
Feb 27 12:33:23 saturn crontab[2671]: (umart) END EDIT (umart)
Feb 27 12:34:01 saturn /usr/sbin/cron[1253]: (umart) RELOAD (crontabs/umart)
Feb 27 12:34:01 saturn /USR/SBIN/CRON[2677]: (umart) CMD (echo hola)
```

En les línies anteriors podem llegir que l'usuari *umart* està treballant en el servidor *saturn* i ha editat el seu crontab el 27 de febrer. Després el cron s'ha recarregat i s'ha executat una tasca del mateix usuari.

Podem configurar syslog perquè es creï un fitxer de registre exclusiu per a cron. Per fer-ho cal editar el fitxer de configuració `/etc/rsyslog.conf` i treure el símbol de comentari, #, de la línia següent:

```
#cron.*   /var/log/cron.log
```

Vegeu més informació sobre el servei syslog a l'apartat "Manteniment dels fitxers de registre" d'aquesta unitat.

Si volem activar un nivell superior de detall de registre de l'activitat de cron, hem d'editar el fitxer de configuració de cron anomenat `/etc/default/cron` i eliminar el símbol de comentari, #, de la línia EXTRA_OPTS="-L 2". El contingut del fitxer quedarà així:

```
# Cron configuration options
...
# Extra options for cron, see cron(8)
# Set a higher log level to audit cron's work
EXTRA_OPTS="-L 2"
```

3.2.5 Control d'accés a cron

És possible controlar quins usuaris poden o no utilitzar els serveis de cron d'una manera molt senzilla amb els fitxers `/etc/cron.allow` i `/etc/cron.deny`.

Si el fitxer /etc/cron.allow existeix, llavors l'usuari ha d'estar llistat (un usuari per línia) dins d'aquest fitxer per poder fer servir l'**ordre *crontab***. Si el fitxer /etc/cron.allow no existeix però existeix el fitxer /etc/cron.deny, llavors l'usuari no ha d'estar llistat dins de /etc/cron.deny per poder usar l'ordre *crontab*. Si es vol evitar que tots els usuaris utilitzin cron es pot escriure *ALL* dins del fitxer /etc/cron.deny.

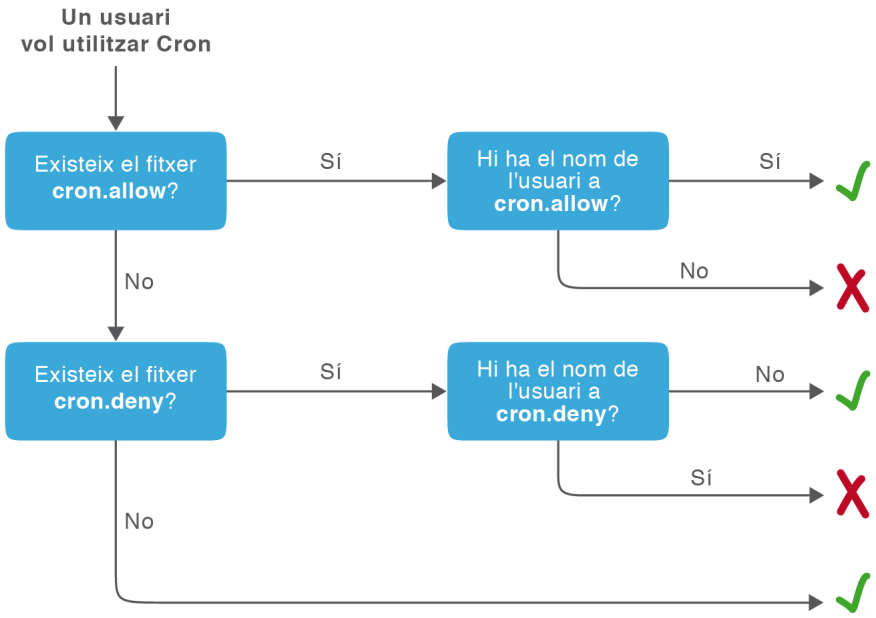
Vegeu el propòsit i el funcionament de l'ordre *crontab* en l'apartat "Crontabs d'usuari" d'aquesta unitat.

Cal tenir en compte les consideracions següents:

- Si cap dels dos fitxers existeix, per defecte tots els usuaris del sistema poden planificar treballs utilitzant l'ordre *crontab*.
- Si els dos fitxers existeixen, llavors el fitxer /etc/cron.allow té precedència, és a dir, el fitxer /etc/cron.deny s'ignora i l'usuari ha d'estar llistat dins de /etc/cron.allow per poder usar l'ordre crontab.

La figura 3.4 il·lustra gràficament el funcionament del control d'accés a cron.

Figura 3.4. Funcionament del control d'accés a cron



Independentment de l'existència d'aquests dos fitxers, l'usuari *root* sempre pot planificar tasques, ja sigui amb l'ordre *crontab* o amb el fitxer de sistema /etc/crontab i els directoris de cron situats a /etc.

3.2.6 Eines gràfiques

Hi ha diverses interfícies que es poden utilitzar per configurar cron de manera gràfica. Els entorns d'escriptori porten els seus paquets respectius per fer la configuració de cron gràficament. Per exemple:

- Amb GNOME podem instal·lar el paquet gnome-schedule. Una vegada instal·lat l'executem des d'*Aplicacions* > *Eines del sistema* > *Tasques programades*.
- Amb KDE tenim el paquet kde-config-cron, que és part del mòdul d'administració de KDE.

La figura 3.5 mostra l'aspecte del planificador de tasques de GNOME visualitzant el següent crontab d'un usuari:

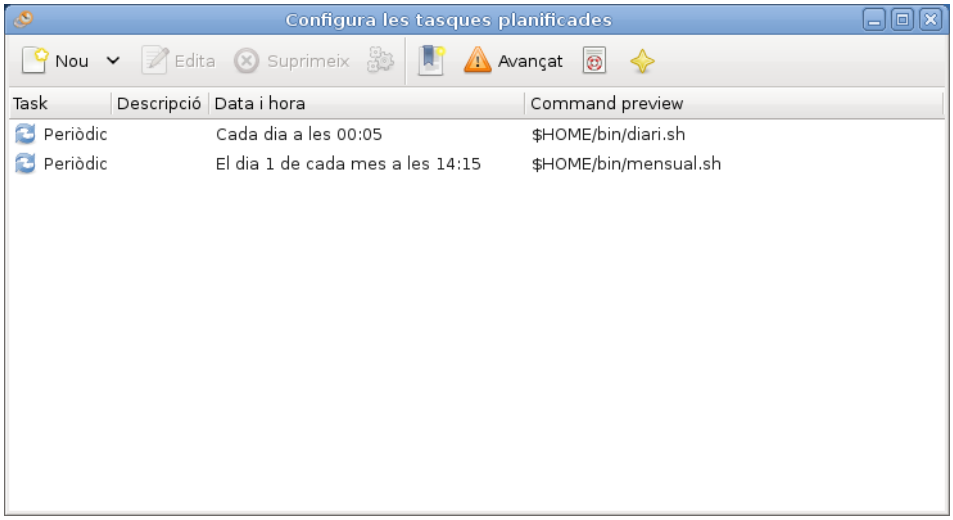
```
5 0 * * * $HOME/bin/diari.sh
15 14 1 * * $HOME/bin/mensual.sh
```

La majoria d'eines gràfiques d'administració del sistema també permeten configurar cron. Per exemple, l'eina d'administració Webmin incorpora un mòdul que permet configurar la planificació de tasques de cron de manera gràfica, en remot i des d'un navegador web.

Vegeu l'apartat "Gestió remota mitjançant Webmin" de la unitat "Administració de processos. Administració remota. Administració de serveis d'impressió".

Si sabem treballar amb cron i configurar-lo des de la línia d'ordres, l'ús de qualsevol d'aquestes eines gràfiques és immediat i explicar el seu funcionament és innecessari.

FIGURA 3.5. Planificador de tasques de GNOME



3.3 Automatització de tasques del sistema

Ubicació dels scripts d'automatització de tasques

En els sistemes de tipus Unix, després de provar i depurar un guió de *shell* d'administració, és habitual moure'l a /usr/local/bin amb el propietari, grup i permisos establerts de manera adequada. En el cas que el guió sigui una utilitat que voleu que estigui disponible per a tots els usuaris del sistema, cal que us assegureu que doneu permís d'execució a *others*.

La programació de *shell scripts* i la utilització d'un planificador de tasques ens facilita l'automatització de moltes de les tasques requerides per al manteniment i la configuració del sistema. A continuació veurem exemples de tasques que típicament requereixen ser automatitzades i utilitzarem el llenguatge de guions Bash per implementar-les.

En primer lloc veurem tasques que s'automatitzen i que normalment també es planifiquen per ser executades de manera periòdica, com són les còpies de seguretat i el manteniment dels fitxers de registre del sistema. Per fer la planificació de l'execució d'aquestes tasques utilitzarem cron.

Després veurem l'automatització de tasques del sistema que en general no requereixen d'un planificador de tasques per ser executades, com ara la gestió d'usuaris, que normalment executem de manera interactiva quan sorgeix la necessitat, i l'automatització de l'inici de serveis que es realitza amb l'arrencada del sistema.

3.3.1 Còpies de seguretat

En informàtica, una còpia de seguretat (en anglès *backup*) és la còpia d'informació que es realitza per tal de ser restaurada en cas de pèrdua de dades o en cas de ser requerida posteriorment.

Les còpies de seguretat poden ser del sistema o de les dades. Les còpies del sistema tenen com a objectiu poder arrencar un sistema després d'un incident de seguretat i per tant realitzen una còpia dels fitxers del sistema operatiu i del programari instal·lat. D'altra banda, les còpies de seguretat de dades pretenen recuperar informació i realitzen còpies de fitxers de dades o bases de dades.

Els administradors de sistemes han de proporcionar els mecanismes per realitzar còpies de seguretat del sistema i de les dades, així com per restaurar-les.

Normalment les dades es copien en un mitjà d'emmagatzemament diferent al de l'origen de les dades, com poden ser discos durs externs, CD-ROM, cintes magnètiques (DAT), etc. i, cada vegada més, s'utilitzen sistemes de còpia de seguretat remota que realitzen les còpies de manera automàtica a través de la xarxa o Internet.

Per tal de decidir quina tecnologia utilitzarem per dur a terme les còpies de seguretat hi ha tres factors clau a tenir en compte: el volum de dades a copiar, el cost econòmic del mitjans d'emmagatzemament i l'operativitat de la solució escollida tant pel que fa al temps de còpia com al de recuperació. Algunes decisions que s'han de prendre són:

- La periodicitat de les còpies. Com més alta és la freqüència més capacitat de recuperació es té.
- El nombre de còpies. Si es realitza més d'una còpia i aquestes es desen en ubicacions separades s'augmenta la seguretat.
- La retenció de les còpies. Com més còpies antigues es guarden es té més capacitat de recuperar informació de fa més temps.
- El model de còpia. Hi ha diversos models (còpia completa, diferencial o incremental) que analitzem més endavant i que són estratègics pel que fa al temps de còpia i recuperació, així com a la quantitat d'espai requerit per guardar les còpies.

Eines

Existeixen molts programaris específics de còpies de seguretat i recuperació. Ara bé, en els sistemes de tipus Unix aquestes tasques també poden fer-se fàcilment

mitjançant scripts planificats amb cron que utilitzen les eines bàsiques que ens proporciona el sistema, com *tar*, *cpio* o el parell *dump/restore*.

En els exemples que mostrarem utilitzarem l'eina *tar* per empaquetar fitxers i l'eina *gzip* per comprimir les dades. Empaquetar vol dir ajuntar dos o més fitxers en un de sol (paquet) i comprimir vol dir agafar el fitxer o paquet i comprimir-lo. A la taula 3.3 es mostra un resum del funcionament de les eines *tar* i *gzip* i l'extensió que se sol posar als fitxers que es generen.

TAULA 3.3. Eines per empaquetar i comprimir dades

Ordre *tar*: empaquetar dades		
Empaquetar	tar cf fitxer.tar /dades	Copia (c) tots els fitxers de /dades i els empaqueta al fitxer (f) fitxer.tar. L'extensió *.tar* li posem per convenció, no és obligatòria. Després de /dades pot haver-hi més noms de directoris o fitxers a empaquetar separats per espais.
Veure el contingut	tar tvf fitxer.tar	Llista (t) i mostra per pantalla (v) les dades empaquetades al fitxer (f) fitxer.tar.
Desempaquetar totes les dades	tar xf fitxer.tar	Extreu (x) totes les dades empaquetades al fitxer (f) fitxer.tar al directori actual.
Desempaquetar algunes dades	tar xvf fitxer.tar f1 f2 ...	Extreu i mostra per pantalla (v) només el fitxer o fitxers especificats (f1, f2...) al directori actual.
Desempaquetar en un directori concret	tar xf fit.tar -C dir tar xf fit.tar f1 f2 -C dir	Extreu les dades al directori especificat després de -C en lloc de al directori actual.
Ordre *gzip*: comprimir dades		
Comprimir	gzip -q fitxer	Comprimeix el fitxer i el reanomena com a fitxer.gz.
Descomprimir	gzip -d fitxer.gz	Descomprimeix fitxer.gz i el deixa com a fitxer.
Ordre *tar* amb *gzip*: empaquetar i comprimir dades. Funciona igual que *tar* però afegint una *z* a les opcions		
Empaquetar i comprimir	tar czf fitxer.tar.gz /dades	Copia (c) i comprimeix amb *gzip* (z) tots els fitxers de /dades i els empaqueta al fitxer (f) fitxer.tar. L'extensió *.tgz* li posem per convenció, no és obligatòria. També podem trobar l'extensió *.tar.gz* en aquests fitxers.
Veure el contingut	tar tzvf fitxer.tar.gz	Llista sense extreure (t) i mostra per pantalla (v) les dades comprimides (z) i empaquetades al fitxer (f) fitxer.tar.gz.
Desempaquetar i descomprimir	tar xzf fitxer.tar.gz	Extreu (x) i descomprimeix (z) les dades del fitxer (f) fitxer.tar.gz.

L'eina *tar* (*tape archiver*) va ser dissenyada originalment per transferir fitxers de disc a una cinta magnètica i viceversa, tot i que ara s'utilitza per empaquetar dades directament sobre qualsevol fitxer o dispositiu. Per exemple, si en el sistema hi ha muntada una unitat de cintes disponible a partir del nom de fitxer de dispositiu /dev/st0 i hi tenim posada una cinta, podem copiar-hi les dades del directori anomenat /dades així:

```
1 tar cf /dev/st0 /dades
```

Una vegada copiades les dades, podem utilitzar les diferents opcions de *tar* que ens permeten veure i extreure les dades.

Estratègies i implementació

L'estratègia més simple per realitzar una còpia de seguretat és copiar totes les dades diàriament. Aquest tipus de còpies són factibles per a sistemes amb un volum de dades petit, però quan la quantitat de dades és gran, cal combinar altres estratègies per dur a terme les còpies. Els tipus de còpies que es poden fer són els següents:

- Copia de seguretat completa. És aquella que inclou tots els fitxers d'un determinat conjunt de dades sense tenir en compte les còpies de seguretat prèvies. També les podem anomenar *còpies de nivell zero*.
- Còpia de seguretat diferencial. És una còpia de tots els fitxers d'un conjunt de dades determinat que ha estat afegit o modificat des de la darrera còpia de seguretat completa. Per restaurar un conjunt de dades, s'ha de restaurar el nivell zero i el diferencial més recent. Un diferencial inclou els mateixos fitxers que el diferencial anterior. Així, en el cas típic, la mida de cada diferencial augmentarà fins que s'executi el proper nivell zero.
- Còpia de seguretat incremental. Es tracta d'una còpia de seguretat de tots els fitxers que hi ha en un determinat conjunt de dades que han canviat des de la darrera còpia de seguretat (de qualsevol tipus). Així, donada una còpia de nivell zero el dilluns, si fem una incremental el dimarts només obtindríem els fitxers nous o canviats des del dilluns. Una incremental feta el dimecres inclouria els fitxers que es van afegir o modificar des del dimarts. Això contrasta amb la diferencial. El que convé recordar és l'ordre de restauració. Per restaurar completament el conjunt de dades, s'ha de recuperar en primer lloc la còpia de nivell zero i després tots els increments en la mateixa seqüència en què van ser creats.

Tenint en compte les definicions que acabem de donar, en funció dels requeriments de cada sistema, les estratègies que s'implementen amb més freqüència són tres:

- Còpia completa diària.
- Còpia completa setmanal i còpia diferencial diària
- Còpia completa setmanal i còpia incremental diària.

Còpia completa diària

La primera estratègia consisteix a fer còpies completes diàries, o, el que és el mateix, fer una còpia de nivell zero de dilluns a diumenge. En la seva forma més simple aquesta còpia és:

```
1 cd /
2 tar czpf /copies/home.tgz home
```

El primer que fem és situar-nos al directori / per evitar posar el símbol / a l'inici dels noms dels fitxers o directoris que volem copiar (l'opció *-C* de *tar* també fa

aquest servei). A continuació fem una còpia del directori /home empaquetant les dades i comprimint-les en un fitxer anomenat home.tgz i ubicat a /copies. L'opció *p* és per preservar els atributs de seguretat (propietari, grup i permisos) dels fitxers que copiem. Assumim que /copies és un sistema de fitxers creat en un disc extern.

Afegim aquestes dues línies en un guió de *shell* que podem ubicar al directori /usr/local/bin i l'anomenem ***completa.sh***. Després planifiquem l'execució d'aquesta tasca perquè s'executi diàriament i triem una hora en la qual no hi hagi activitat en el sistema per dues raons:

- Per garantir la integritat de les dades que es copien.
- Perquè el procés de còpia no afecti al rendiment global del sistema.

La línia del crontab podria quedar així:

En tots els exemples, *root* és el propietari dels guions de *shell* i l'usuari que els planifica i els executa amb cron.

```
0 2 * * */usr/local/bin/completa.sh
```

Fixeu-vos que si no fem res més, la retenció de les còpies amb aquest sistema és només d'un dia, ja que cada dia estem guardant la còpia completa amb el mateix nom, i, per tant, estem sobreescrivint el fitxer. Generalment voldrem mantenir les còpies de més d'un dia d'antiguitat.

En el nostre cas, podem fer una modificació molt senzilla per mantenir sempre les set darreres còpies realitzades. Consisteix a afegir un número al nom del fitxer que indiqui el dia de la setmana en què es fa la còpia, per exemple:

```
cd /
tar czpf /copies/home$(date +%u).tgz home
```

Amb aquesta modificació, el nom del fitxer generat conté el resultat de l'execució de l'ordre `date +%u` per indicar el dia de la setmana (home1.tgz els dilluns, home2.tgz els dimarts, etc.). Així, cada dia sobreescriurem la còpia de fa una setmana i mantindrem les dels sis dies anteriors.

Còpia completa setmanal i diferencial diària

La segona estratègia consisteix a fer una còpia setmanal completa (per exemple el diumenge) i una còpia de nivell 1 (diferencial) la resta de dies de la setmana. En aquesta estratègia, a la còpia de nivell 0 realitzada el diumenge que implementem com acabem de veure, li hem d'afegir la còpia diferencial diària, que en la seva forma més simple és:

```
find /home -type f -newer /copies/home0.tgz >/tmp/llista
tar czpTf /tmp/llista /copies/home$(date +%u).tgz
```

L'ordre *find* permet cercar fitxers que compleixen unes característiques determinades amb diversos criteris de selecció.

Veiem que ara estem utilitzant l'ordre *find* per generar una llista de tots aquells fitxers de /home que siguin més nous que el fitxer que conté la còpia completa. En el *tar* hem afegit l'opció *T* per llegir els fitxers a copiar des de la llista generada en el pas anterior.

Afegim les dues línies a un *shell script* que anomenem ***diferencial.sh*** i planifiquem la còpia completa perquè s'executi únicament els diumenges i la diferencial perquè s'executi la resta de dies de la setmana. Per exemple:

```
0 2 * * 0/usr/local/bin/completa.sh
0 2 * * 1-6/usr/local/bin/diferencial.sh
```

Còpia completa setmanal i incremental diària

La tercera estratègia consisteix a fer una còpia completa setmanal (per exemple el diumenge) i una incremental la resta de dies de la setmana (nivell 1 el dilluns, nivell 2 el dimarts, nivell 3 el dimecres, etc.). En aquesta estratègia, a la còpia realitzada el diumenge de nivell 0 que implementem com en els altres casos, li hem d'afegir la còpia incremental diària, que en la seva forma més simple és:

```
find /home -type f -mtime 1 > /tmp/llista
tar czpTf /tmp/llista /copies/home$(date +%u).tgz
```

En aquest cas estem utilitzant l'ordre *find* per generar una llista de tots aquells fitxers de /home que s'han modificat en les darreres 24 hores. El *tar* s'ha d'interpretar igual que en el cas anterior.

De manera anàloga al cas anterior, afegim les dues línies a un guió de *shell* que anomenem ***incremental.sh*** i planifiquem la còpia completa per als diumenges i la incremental per a la resta de dies. Per exemple:

```
0 2 * * 0/usr/local/bin/completa.sh
0 2 * * 1-6/usr/local/bin/incremental.sh
```

Millora dels guions de shell

Podem millorar els procediments per implementar les diferents estratègies de còpia (completa, diferencial i incremental) si en els *shell scripts* utilitzem variables, afegim control d'errors, etc. A continuació, i a tall d'exemple, veiem una versió millorada del *shell script* que hem anomenat *completa.sh*:

Vegeu exemples de guions de *shell* per copiar un servidor web i una base de dades local a la secció d'adreces d'interès del web del mòdul.

```
#/bin/bash
# completa.sh
# Còpia completa de dades
#
# Directoris a copiar, no incloem la / inicial
DIRS="etc home var"
# Directori de destinació de la còpia
BACKUPDIR="/copies"
# Nom del fitxer de la còpia
AVUI=$(date +"%Y-%m-%d")
FITXER="completa_$AVUI.tar.gz"
# Nombre de dies per guardar còpies antigues
ANTIGUES=14
#
# Missatge d'inici
echo "$(date +%X) Inici del procediment de còpia."
#
# Comprovació que l'eina tar hi és
TAR=$(which tar)
```

```
20 if [ -z "$TAR" ]; then
21   echo "Error: no s'ha trobat tar."
22   exit 1
23 fi
24 # Comprovació que el directori de còpies hi és
25 if [ ! -d $BACKUPDIR ] ; then
26   echo "Error: no s'ha trobat $BACKUPDIR"
27   exit 1
28 fi
29 # Fem la còpia
30 $TAR -zcPf $BACKUPDIR/$FITXER -C / $DIRS
31 if [ $? -ne 0 ]
32 then
33 # El tar ha fallat
34   echo "Error: hi ha hagut algun error en fer la còpia."
35   exit 1
36 fi
37 # Esborrem les còpies anteriors a $ANTIGUES
38 find $BACKUPDIR/ -name "*.gz" -type f -mtime +$ANTIGUES -delete
39 if [ $? -ne 0 ]
40 then
41   echo "Error: eliminant les còpies antigues."
42   exit 1
43 fi
44 # Missatge de finalització
45 echo "$(date +%X) Final correcte de la còpia."
46 exit 0
```

Aquest guió de *shell* permet guardar tantes copies com s'hagin definit a la variable *ANTIGUES*. El nom del fitxer de còpia que es genera és: completa_AAA-MM-DD.tar.gz. Per exemple, si fem una còpia el dia 11 de març de 2012 tindrem un fitxer que es dirà *completa_2012-03-11.tar.gz*.

Planifiquem l'execució diària del guió de *shell* a l'hora que convingui. Per exemple:

```
1 0 2 * * */usr/local/bin/completa.sh
```

Si volem guardar un registre de l'activitat d'aquest *shell script* hem de redirigir les sortides estàndard i d'errors a un fitxer. Per exemple:

```
1 0 2 * * * /usr/local/bin/completa.sh >>fitxer 2>>fitxer
```

El fitxer de registre usualment el crearem en el directori de registre del sistema, /var/log. En aquest cas podem crear un directori anomenat /var/log/local i desar allà els fitxers de registre dels procediments de còpia i d'altres procediments locals de manteniment del servidor. Per al guió de *shell* completa.sh, el nom del fitxer de sortida pot ser /var/log/local/completa.log.

3.3.2 Manteniment dels fitxers de registre

Els fitxers de registre o de *log* (de l'anglès *log files*) són fitxers de traça que generen el sistema o les aplicacions per enregistrar esdeveniments i deixar constància de les accions que es fan. Són molt útils per veure l'activitat d'un servei o aplicació i per diagnosticar problemes.

syslog

syslog és un estàndard de facto per a l'enviament de missatges de registre en una xarxa informàtica IP. Va ser desenvolupat el 1980 per Eric Allman com a part del projecte Sendmail. Posteriorment es va comprovar que era molt útil i d'altres aplicacions també van començar a usar syslog.

En els sistemes Unix i derivats hi ha un servei anomenat **syslog** que és l'encarregat d'enregistrar l'activitat del sistema operatiu i de les aplicacions que utilitzen aquest servei, com ara el servei cron (planificació de treballs) o lpr (subsistema d'impressió) entre d'altres.

Generalment, tots els fitxers de registre del sistema, siguin o no gestionats per syslog, se solen emmagatzemar al directori `/var/log/`. Encara que la majoria de fitxers de registre són de text i els podem veure amb qualsevol editor, en podem trobar algun d'especial que no desi les seves dades en aquest format, com ara els fitxers `/var/log/wtmp` i `/var/log/btmp`, que són els registres d'entrada d'usuaris en el sistema i d'entrades errònies respectivament. Per veure aquests dos fitxers, podem utilitzar les ordres `last` i `lastb`. Si tinguéssim configurats aquests registres en algun altre fitxer, també els podríem veure passant el paràmetre amb l'ordre `last -f nom_fitxer`.

La mida dels fitxers de registre sempre va creixent, ja que les dades enregistrades es van afegint al final del fitxer. Per evitar saturar el disc, cal automatitzar alguna tasca de manteniment d'aquests fitxers.

rsyslogd

Hi ha diverses implementacions de syslog. La distribució Linux Debian utilitza rsyslogd. Executeu `man rsyslogd` per veure el funcionament i la personalització d'aquest servei.

Normalment s'utilitza un sistema de rotació de registres, que consisteix a anar comprimint cada cert temps aquests fitxers i a desar-los fins que tinguin una antiguitat determinada. Per exemple, es poden comprimir cada setmana i desar només els d'un o dos mesos anteriors. Segons el servidor que estiguem administrant haurem de tenir en compte la legalitat vigent, que en alguns casos obliga a conservar els fitxers de registre durant un període de temps determinat.

Consola de logs

Podem configurar una consola del sistema per veure tots els registres que es van generant, afegint la línia *.* /dev/ttySX (X és la consola en què volem veure els registres) al fitxer /etc/rsyslog.conf i reiniciant el dimoni rsyslogd.

Hi ha programes que ens permeten fer ús del sistema de *log* que podem utilitzar en els nostres *shell scripts* per crear els nostres propis fitxers de registre o, si escau, manipular manualment els del sistema: amb `logger` podem escriure en el syslog del sistema i amb `savelog` podem desar i opcionalment comprimir els fitxers de registre.

Gestió dels fitxers de registre amb logrotate

El programa `logrotate` està dissenyat per facilitar les tasques de gestió de registres. Permet la rotació automàtica, compressió, eliminació i l'enviament per correu dels fitxers. Cada fitxer de registre pot ser tractat diàriament, setmanal, mensual o quan es fa massa gran.

Executeu `man logrotate` per veure la pàgina de manual del programa.

Normalment, logrotate s'executa com una tasca diària de cron. Al directori `/etc/cron.daily` hi ha un guió de *shell* anomenat `logrotate` que conté les línies següents:

```
#!/bin/sh
test -x /usr/sbin/logrotate || exit 0
/usr/sbin/logrotate /etc/logrotate.conf
```

El fitxer `/etc/logrotate.conf` és on es configura el mode d'operació de logrotate. Per defecte té un contingut similar al següent (els comentaris originals són en anglès):

```
# rotar els fitxers de registre setmanalment
weekly
# mantenir 4 setmanes de fitxers antics
rotate 4
# crear fitxers nous buits després de rotar
create
# comprimir els fitxers
#compress
# directori on desar fitxers de configuració específica de les aplicacions
include /etc/logrotate.d
# configuració per als serveis wtmp i btmp
/var/log/wtmp {
  missingok
  monthly
  create 0664 root utmp
  rotate 1
}
/var/log/btmp {
  missingok
  monthly
  create 0660 root utmp
  rotate 1
}
# altres logs específics del sistema es poden configurar aquí
```

Les aplicacions poden deixar els seus propis fitxers al directori /etc/logrotate.d per especificar les seves opcions. Per exemple, el servei CUPS instal·la per defecte aquest fitxer:

```
/var/log/cups/*log {
  daily
  missingok
  rotate 7
  sharedscripts
  postrotate
    if [ -e /var/run/cups/cupsd.pid ]; then
      invoke-rc.d --quiet cups force-reload > /dev/null
      sleep 10
    fi
  endscript
  compress
  notifempty
  create 640 root lpadmin
}
```

Gestió dels fitxers de registre amb guions de shell propis

El programa logrotate ens dóna la possibilitat d'automatitzar la gestió dels fitxers de registre del sistema i de les aplicacions, però la manipulació d'aquests fitxers també es pot fer amb guions de *shell* creats per nosaltres amb aquest propòsit.

A continuació veiem un exemple senzill per esborrar els registres del fitxer de *log* del sistema anomenat `/var/log/messages`. El programa esborra el fitxer i es queda amb 50 línies o les que indiqui l'usuari en la línia d'ordres.

```
#!/bin/bash
# borralog.sh
# Neteja fitxers de registre del sistema
#
# Definició de variables
LOG_DIR=/var/log
FITXER=messages
ROOT_UID=0
LINIES=50
# Definició de funcions
missatge () {
  DATA=$(date +"%b %x - %X")
  echo "$0: $DATA ---> $1"
}
############## Programa principal ##############
# Missatge d'inici
missatge "Inici esborrat de logs."
# Comprovem que som root.
if [ "$UID" -ne "$ROOT_UID" ]; then
  missatge "Error: heu de ser root."
  exit 1
fi
# Establir el nombre de línies que cal preservar
case "$1" in
  "")
    linies=$LINIES
    ;;
  *[!0-9]*)
    echo missatge "Error: $1 argument invàlid."
    exit 1
    ;;
  *)
    linies=$1
    ;;
esac
# Canviar al directori de logs
cd /var/log || {

  missatge "Error: no puc anar a $LOG_DIR." >&2
  exit 1;
}
# Guardar les línies que volem del fitxer de log
tail -n $linies $FITXER > $FITXER.tmp
mv $FITXER.tmp $FITXER
# Missatge de final
missatge "Final de l'esborrat de logs."
exit 0
```

El guió de *shell* anterior pot ser executat de manera interactiva o bé es pot planificar amb cron per ser executat periòdicament. Per exemple:

```
30 1 * * */usr/loc/bin/borralog.sh
```

Si volem guardar el registre de l'activitat d'aquest *shell script*, hem de redirigir les sortides estàndard i d'errors a un fitxer. Per exemple:

```
30 1 * * * /usr/local/bin/borralog.sh >>fitxer 2>>fitxer
```

El nom del fitxer de sortida podria ser: `/var/log/local/borralog.log`.

3.3.3 Gestió d'usuaris

Els usuaris dels sistemes de tipus Unix tenen assignat un compte o nom d'usuari que els identifica. Cada compte d'usuari també té una contrasenya que els permet accedir al sistema anomenada clau d'accés (*password*). A més a més, a Unix cada usuari pertany com a mínim a un grup d'usuaris, que anomenem el seu *grup primari* o *principal*. En cas de ser necessari, un usuari pot pertànyer a més d'un grup i en aquest cas es diu que són els seus grups secundaris o addicionals.

Alguns usuaris i grups es creen en el moment d'instal·lar el sistema operatiu, per exemple: *root*, *bin*, *sys*, *mail*, *uucp*, etc.

Una de les funcions de l'administrador del sistema és donar d'alta els grups i els usuaris al sistema. Hi ha un usuari especial anomenat superusuari o *root*, que es crea amb la instal·lació del sistema operatiu i disposa dels màxims drets al sistema.

> És convenient fer servir el superusuari només quan fem funcions d'administració de sistemes que requereixen els drets de *root*. En tots els altres casos és més adient emprar un altre compte. Per evitar l'ús de *root*, també podem configurar `sudo` per permetre que un compte d'usuari normal tingui drets per fer determinades tasques d'administració, com ara gestionar usuaris.

sudo

sudo és una ordre que permet que els usuaris executin ordres que inicialment només pot executar *root*. Al fitxer de configuració /etc/sudoers és on s'indica quins usuaris poden executar quines ordres.

Hi ha moltes eines que permeten fer la gestió d'usuaris i grups des d'una interfície gràfica, per exemple l'eina d'administració Webmin o les interfícies dels escriptoris de GNOME i KDE, entre d'altres. Alternativament, per treballar des de línia d'ordres, en algunes distribucions de Linux, entre d'elles a Debian, hi ha els programes useradd, userdel i usermod per a la gestió d'usuaris, i groupadd, groupdel i groupmod per a la gestió de grups.

La gestió d'usuaris pot donar molta feina tenint en compte que, per exemple, ens podem trobar que volem donar d'alta grups de més de 20 persones. En aquests casos el més habitual és fer guions de *shell* que permeten automatitzar aquesta tasca i donar d'alta usuaris nous de forma massiva a partir d'un fitxer de text amb un format especial.

A tall d'exemple mostrem un guió de *shell* que dóna d'alta tots els usuaris descrits en un fitxer de text amb format CSV. Els camps de les línies del fitxer d'usuaris són els següents:

```
1 nom_usuari,grup_primari,contrasenya_inicial
```

Fitxer CSV

Els fitxers CSV (de l'anglès comma-separated values) són un tipus de document que representa dades en forma de taula, en la qual les columnes se separen per comes (o punt i coma) i les files per salts de línia.

Per exemple, si volem donar d'alta la usuària *angela* i l'usuari *david* amb grup primari *alumnat* i contrasenya *123* i *456* respectivament, i l'usuari *marius* amb grup *professorat* i contrasenya *789*, el fitxer CSV ha de contenir les línies:

```
1 angela,alumnat,123
2 david,alumnat,456
3 marius,professorat,789
```

El codi del programa és el següent:

```
#/bin/bash
# altausu.sh
# Dóna d'alta usuaris a partir de la informació d'un
# fitxer CSV. El format de les línies del fitxer és:
# nom,grup,contrasenya
#
# Nom del fitxer CSV
FITXER=usuaris.csv
ROOT_UID=0
#
# Comprovacions
if [ "$UID" -ne "$ROOT_UID" ]; then
  echo "Error: heu de ser root."
  exit 1
fi
if [ ! -f "$FITXER" ]; then
  echo "Error: no s'ha trobat $FITXER."
  exit 1
fi
# Inici del procés
for linia in $(cat $FITXER); do
  echo $linia
  USUARI=$(echo $linia | cut -f1 -d",")
  GRUP=$(echo $linia | cut -f2 -d",")
  PASSWD=$(echo $linia | cut -f3 -d",")
  echo "Creant l'usuari $USUARI..."
  /usr/sbin/useradd -m -g $GRUP $USUARI
  echo "Assignant contrasenya a $USUARI..."
  echo "$USUARI:$PASSWD" | chpasswd
done
```

Aquest *shell script* automatitza i simplifica considerablement la tasca de creació d'usuaris. Podem millorar el codi del programa afegint, si cal, el tractament de més camps del fitxer d'informació d'usuaris (directori de treball, *shell* d'inici, etc.), fent control d'errors, guardant els missatges de sortida en un fitxer de registre, etc.

L'ordre *chpasswd* fa el mateix que *passwd*, però pot ser utilitzada dins d'un guió de *shell* de manera no interactiva.

3.3.4 Engegada i aturada automàtica de serveis

Durant el procés d'arrencada del sistema s'inicien els processos que anomenem *dimonis* (*daemons* en anglès). Generalment tots els dimonis tenen un guió de *shell* situat al directori `/etc/init.d/` que ens permet iniciar-los, aturar-los o veure el seu estat d'execució. La majoria d'aquests scripts utilitzen un programa anomenat `start-stop-daemon` que ens proporciona el sistema operatiu i que serveix per al tractament d'aquests processos.

En administrar un servidor es pot donar el cas que ens haguem de dissenyar els nostres propis dimonis per fer alguna tasca concreta. Al directori on se situen tots els guions de *shell* dels dimonis, se'n sol trobar un d'exemple anomenat `/etc/init.d/skeleton` perquè el puguem utilitzar quan en necessitem configurar un de nou. El codi del dimoni d'exemple és similar al següent:

Vegeu més informació sobre els processos que anomenem *dimonis* a l'apartat "Dimonis" de la unitat "Administració de processos. Administració remota. Administració de serveis d'impressió".

```
#!/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin: /usr/sbin:/usr/bin DAEMON=/usr/
    sbin/daemon
NAME=daemon
DESC="some daemon"
test -x $DAEMON || exit 0
set -e
case "$1" in
  start)
    echo -n "Starting $DESC: $NAME"
    start-stop-daemon --start --quiet --pidfile \
    /var/run/$NAME.pid --exec $DAEMON echo "."
    ;;
  stop)
    echo -n "Stopping $DESC: $NAME "
    start-stop-daemon --stop --quiet --pidfile \
    /var/run/$NAME.pid --exec $DAEMON echo "."
    ;;
  restart|force-reload)
    echo -n "Restarting $DESC: $NAME"
    start-stop-daemon --stop --quiet --pidfile \
    /var/run/$NAME.pid --exec $DAEMON
    sleep 1
    start-stop-daemon --start --quiet --pidfile \
    /var/run/$NAME.pid --exec $DAEMON echo "."
    ;;
  *)
    N=/etc/init.d/$NAME echo " Usage: $N {start|stop| \
    restart|force-reload}" >&2
    exit 1
    ;;
esac
exit 0
```

La primera línia indica quin és el *shell* d'execució. Gairebé en tots els scripts d'inici trobarem el shell /bin/sh. Com que Bash és compatible amb sh, les ordres i els programes escrits per a sh poden ser executats amb Bash sense cap modificació. Però al revés no és cert. Per tant, si necessitem escriure un *shell* d'inici amb característiques específiques del llenguatge Bash hem de canviar la primera línia i posar */bin/bash*.

En les variables declarades a l'inici del guió de *shell* especifiquem quin *PATH* és necessari per al procés del dimoni, el programa que executarem (*DAEMON*), el nom que li donem (NAME), que ha de ser igual que el nom del *shell script*, i la descripció (*DESC*).

En arrencar el dimoni la única cosa que fem és escriure al directori /var/run/ un fitxer amb el PID del procés. En aturar-lo, es buscarà aquest PID i s'enviarà el senyal d'acabament al procés corresponent.

Trobarem guions de *shell* preparats per a fer moltes més operacions amb el nostre dimoni, però, com a mínim, tots han de tenir aquesta estructura.