Introducció als entorns de treball Gnu/Linux

Autors: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Josep Jorba Esteve
PID_00298304
Primera edició: setembre 2023

Introducció

La seqüència del genoma de múltiples espècies està a disposició de qualsevol persona que posseeixi un ordinador amb connexió a Internet, gràcies a l'enorme esforç de la comunitat científica, cohesionada en diversos consorcis internacionals, públics i privats. Els grans projectes de seqüenciació han produït una voluminosa quantitat d'informació genòmica que ha de ser gestionada de forma extremadament eficient i precisa per a la seva posterior anàlisi. Només la seqüència de nucleòtids del genoma humà, que ocupa diversos *gigabytes*, conté desenes de milers de gens i reguladors transcripcionals. De fet, la recent aparició de nous mètodes de seqüenciació massiva per cartografiar la localització de diferents elements funcionals al llarg de les seqüències genòmiques en qualsevol context cel·lular multiplicarà exponencialment els requisits actuals de temps de càlcul i espai d'emmagatzematge.

L'anàlisi exhaustiva de tota aquesta informació per extreure'n nou coneixement no es pot realitzar manualment. La gestió informàtica resulta essencial, per tant, per manipular amb garanties aquest volum de dades. La bioinformàtica proporciona, en aquest sentit, l'entorn ideal de treball per al biòleg molecular. En un entorn bioinformàtic de recerca, els genomes i les aplicacions estan emmagatzemats localment, evitant problemes de connexió i trànsit de la xarxa. Aquestes estacions de treball són les eines essencials de l'investigador per efectuar anàlisis bioinformàtiques de qualsevol mena de seqüència biològica.

Aquest mòdul aprofundeix sobre la majoria d'aplicacions computacionals utilitzades habitualment per un bioinformàtic per processar informació genòmica. La família de sistemes operatius GNU/Linux és la plataforma habitual de treball en aquesta classe de laboratoris. En primer lloc, farem una introducció general als conceptes bàsics relacionats amb els sistemes operatius. Posteriorment, enfocarem el nostre interès en el maneig del terminal de GNU/Linux, l'eina de treball usual per a un bioinformàtic. Aprendrem les ordres bàsiques, i com poden combinar-se per generar ordres encara més potents que conformin protocols complets de treball. Finalment, veurem que podem obtenir fàcilment una còpia dels grans conjunts de dades biològiques de referència per poder analitzar-les localment al nostre ordinador amb summa facilitat. En resum, dominareu els elements bàsics de treball per integrar-vos fàcilment dins de qualsevol entorn de recerca bioinformàtic.

Objectius

Amb el programa de continguts d'aquest mòdul, un cop finalitzada l'etapa d'aprenentatge, haureu d'assolir els objectius següents:

1. Conèixer les funcions del sistema operatiu.
2. Reconèixer la família de sistemes operatius GNU/Linux.
3. Distingir entre processos i programes.
4. Identificar la jerarquia del sistema de fitxers.
5. Treballar amb màquines virtuals multiplataforma.
6. Utilitzar el terminal per a l'anàlisi de dades bioinformàtiques.
7. Combinar sèries d'ordres per crear protocols més complexos.
8. Obtenir una còpia dels entorns bioinformàtics reals.
9. Integrar les ordres del terminal amb la informació biològica.

1. Introducció als entorns de treball UNIX

1.1. Arquitectura d'un ordinador

Un ordinador és una màquina electrònica que és capaç de processar i emmagatzemar informació. John Eckert i John Mauchly van presentar a Filadèlfia el 15 de febrer de 1946 l'Electronic Numerical Integrator And Calculator (ENIAC), el que es considera el primer ordinador de propòsit general de la història. Aquesta màquina de trenta tones, que ocupava un pis complet de l'Escola Moore d'Enginyeria Elèctrica (Universitat de Pensilvània), resolia en una sola hora el mateix nombre de càlculs de trajectòries balístiques que dos-cents especialistes en dos mesos. Des d'aquest primer model fins a l'aparició dels microprocessadors d'última generació (integrats en telèfons mòbils i electrodomèstics), el progrés tecnològic ha estat increïblement veloç.

Els ordinadors, per poder manipular dades, fer càlculs o executar tasques preestablertes, requereixen d'uns components bàsics llistats a continuació:

- La unitat central de processament (en anglès, Central Processing Unit o CPU). El «cervell» de l'ordinador, el qual executa els processos i càlculs.

- La memòria central (en anglès, Random Access Memory o RAM). La memòria de l'ordinador és un dispositiu d'emmagatzemament temporal d'informació contínuament modificada per les operacions realitzades a la CPU. Programes i dades s'han de carregar prèviament en la memòria, i llavors són transferits al processador per a la seva execució.

- Disc dur. Emmagatzematge estable de la informació. Actualment, hi ha dos tipus de discos: *hard drive* (HDD) o *solid-state drive* (SSD).

- Dispositius d'entrada i sortida (I/O). Els dispositius d'entrada permeten la introducció d'informació o ordres a l'ordinador (teclat, ratolí, etc.), mentre que els dispositius de sortida proporcionen els resultats a l'usuari (pantalla, impressora, etc.).

- Els perifèrics. Són dispositius que milloren la funcionalitat dels ordinadors (discos durs, llapis de memòria o targetes de xarxa).

Per poder fer que els ordinadors duguin a terme les tasques que els encomanem, necessitem poder crear programes (*software*) amb les instruccions exactes que volem executar. Els ordinadors no són capaços de processar llenguatge natural com nosaltres, el seu llenguatge és binari (Veritable o Fals, Obert o Apagat, 0 o 1). Això és a causa de l'arquitectura dels ordinadors, que està formada per milers o milions de transistors que individualment es poden encendre o apagar. Els transistors poden estar en estat actiu si una petita quantitat de corrent passa a través seu (1) o, si no hi ha corrent, el transistor està en estat 0. A partir d'aquests dos estats es pot generar un llenguatge completament nou.

```
A -> 01000001
B -> 01000010
C -> 01000011
```

Per poder comunicar les ordres que volem executar als ordinadors utilitzarem els llenguatges de programació, que transformaran les nostres instruccions en codi binari per poder-les processar.

1. Introducció als entorns de treball UNIX

1.2. Funcions d'un sistema operatiu

El sistema operatiu (SO) és el programa que exerceix un rol d'intermediari entre l'usuari i la màquina, i que dota una interfície de funcions elementals per a la seva gestió. D'aquesta manera, l'usuari de la màquina aconsegueix extreure'n un rendiment superior, despreocupant-se de la seva complexitat tècnica.

Els avantatges de treballar amb un sistema operatiu són diversos:

- Proporciona un entorn de treball més amigable per a la seva utilització.
- Utilitza eficientment els recursos de l'ordinador.
- Permet un desenvolupament i testatge de noves funcions sense interferir amb els serveis ja existents.
- Proporciona el nombre més gran de tasques per unitat de temps.

Entre els recursos que el SO gestiona de forma transparent hi trobem:

- El processador.
- La memòria i els dispositius d'emmagatzematge secundaris.
- Dispositius d'entrada i sortida de dades.
- El sistema d'arxius i directoris.
- La connexió a la xarxa i amb altres màquines.
- La seguretat i privacitat de les dades.
- La informació interna sobre el sistema.

Exemples de SO:

- Windows
- GNU/Linux
- macOS
- Android
- Solaris

1. Introducció als entorns de treball UNIX

1.3. La família de sistemes operatius UNIX

1.3.1. Introducció

El 1969, Ken Thompson i Dennis Ritchie van desenvolupar en llenguatge assemblador un petit SO anomenat *UNICS* (en anglès, UNiplexed Information and Computing System), suficient per ser executat en un miniordinador DEC PDP-7. Uns anys abans, Ken Thompson havia format part d'un gran projecte coordinat entre l'Institut Tecnològic de Massachusetts (MIT), els Laboratoris Bell d'AT&T i General Electric per produir un altre sistema que funcionava sobre una gran computadora GE-645. Aquest sistema va rebre el nom de MULTICS (en anglès, MULTiplexed Information and Computing System) i, malgrat les seves múltiples innovacions, el projecte va ser abandonat pel seu pobre rendiment. UNICS, en contraposició a MULTICS, va ser concebut com un sistema lleuger orientat a governar miniordinadors.

A mesura que el projecte inicial demostrava el seu gran potencial, van sorgir més possibilitats de desenvolupament. El 1970, amb el suport econòmic dels Laboratoris Bell (AT&T), els autors van finalitzar una primera versió estable, que incloïa eines per editar text, i era capaç de funcionar en una minicomputadora PDP-11/20. El 1972, els mateixos autors van reescriure el codi d'UNIX en el llenguatge d'alt nivell C, permetent la portabilitat de tot el sistema a qualsevol plataforma. En augmentar la comprensió del codi, el propietari d'UNIX va distribuir llicències de desenvolupament a diverses universitats i companyies. En particular, el departament de Computació de la Universitat de Califòrnia, amb seu a Berkeley, va publicar la seva pròpia versió d'UNIX, anomenada *Berkeley Software Distribution* (BSD), encara d'àmplia difusió actualment. A finals dels anys setanta, gràcies a la distribució mitjançant llicència del codi original, el nombre de variants d'UNIX va començar a multiplicar-se exponencialment. La companyia AT&T, propietària del sistema UNIX original, va llançar el 1983 la distribució UNIX System V, una versió estable que combinava les millores contingudes en cada variant apareguda anteriorment.

1. Introducció als entorns de treball UNIX

1.3. La família de sistemes operatius UNIX

1.3.2. Altres versions d'UNIX

Diferents companyies han desenvolupat la seva pròpia distribució comercial d'UNIX. Solaris va ser produït per Sun, AIX per IBM, HP-UX per Hewlett-Packard o Mac OS-X per Apple. Fins i tot Microsoft va treballar en la seva pròpia distribució, anomenada *Xenix*.

La dècada dels vuitanta va presidir el gran èxit dels ordinadors personals d'IBM (en anglès, Personal Computer o PC), gestionats pel sistema operatiu DOS (en anglès, Disk Operating System). Precisament, un estudiant finlandès d'informàtica anomenat Linus Torvalds va desenvolupar el 1991 el nucli d'un sistema operatiu basat en UNIX per gestionar microprocessadors Intel x86 (el cor dels primers PCs d'IBM). Aquesta variant, en conjunció amb el *software* obert creat pel projecte GNU (per exemple, gcc o emacs), va constituir l'embrió de la família de sistemes operatius Linux. Tot i que e troben en funcionament múltiples projectes de desenvolupament sobre la base d'UNIX (especialment per a la versió BSD), Linux ha aconseguit una àmplia difusió entre la comunitat informàtica a causa de la seva lliure distribució com a *software* obert. Gràcies a les millores aportades per milers de programadors, Linux és un dels SO més estesos actualment.

1.3. La família de sistemes operatius UNIX

1. Introducció als entorns de treball UNIX

1.3.3. El projecte GNU

GNU (en anglès, GNU's not Unix!) propugna un SO compost per peces de *software* lliure. Teniu més informació a la pàgina web http://www.gnu.org.

A causa del seu caràcter obert, no hi ha en realitat una única versió de Linux. Des del seu naixement n'han sorgit nombroses variants (denominades *distribucions*) que comparteixen el nucli del sistema i un conjunt de biblioteques i programes comuns sorgits a partir del projecte GNU. Cada distribució es diferencia de la resta pel tipus d'aplicacions que incorpora, orientant-se a usuaris domèstics, empreses o grans servidors. Existeixen distribucions suportades comercialment per companyies que proporcionen assistència durant la instal·lació i actualització del sistema. La taula 1 en mostra algunes de les variants més conegudes que existeixen.

Taula 1. Distribucions Linux de caràcter general més populars

Distro	Web
Ubuntu	ubuntu.com
CentOS	centos.org
Debian	debian.org
Red Hat Enterprise	redhat.com
Slackware	slackware.com
Fedora	fedoraproject.org
openSUSE	opensuse.org
Arch	archlinux.org
Linux Mint	linuxmint.com

Font: elaboració pròpia.

1. Introducció als entorns de treball UNIX

1.4. Programes i processos

Els ordinadors executen programes que són llistats d'instruccions que indiquen com processar un conjunt de dades.

Podem dividir els llenguatges de programació entre:

- Baix nivell: assembladors i de màquina. Propers al codi binari.
- Alt nivell: fàcils de llegir i entendre (per exemple, C, PHP, Python o Java).

Així mateix, podem diferenciar els llenguatges d'alt nivell compilats o interpretats.

Un programa escrit amb un llenguatge de programació compilat, per poder executar-se, necessita ser abans processat per un compilador adequat, i traduït a llenguatge màquina (binari). Els compiladors són programes encarregats de dur a terme l'anàlisi lèxica, sintàctica i semàntica del codi. Un cop superada aquesta etapa de verificació, el compilador genera un fitxer objecte que ha de ser enllaçat amb diverses llibreries de funcions del sistema per generar un fitxer executable binari. L'usuari pot executar aquest arxiu quan calgui. La depuració dels programes compilats és costosa, i és rendible només en els casos en què el rendiment òptim d'aquests, en termes de temps d'execució i espai de memòria, és capital.

Per executar tasques més senzilles és possible dissenyar prototips (en anglès, *scripts*) emprant llenguatges orientats a la producció ràpida de programes, com Perl o Python, llenguatges interpretats. Aquests llenguatges de *scripting* posseeixen un joc d'instruccions específicament dissenyat per facilitar l'adquisició i tractament de fitxers de text. Els seus intèrprets processen els programes a instrucció, saltant-se d'aquesta manera la creació d'un fitxer binari. A canvi, el seu rendiment, en comparació amb els fitxers executables, és menor.

Un cop l'usuari decideix executar un programa, el SO ha de crear una entitat lògica associada a aquest codi a la qual dotar de recursos suficients per desenvolupar la seva activitat (processador, memòria i accés a dispositius). Aquesta metodologia permet executar de forma concurrent diverses instàncies de la mateixa aplicació sense més inconvenient que els propis de la compartició d'alguns recursos (fàcilment esmenables dins del programa, utilitzant noms únics per als fitxers i altres dispositius).

Atès que només un procés pot estar simultàniament en possessió de la CPU, s'ha de realitzar una planificació òptima per decidir en cada moment a quin procés li correspon el seu ús. Malgrat que la compartició del processador sembla una seriosa limitació, és en realitat un gran avantatge, ja que un procés gasta una fracció important del seu temps esperant per accedir a altres dispositius més lents. Per tant, superposant l'ús de la CPU amb les esperes dels processos s'aconsegueix simular un treball en paral·lel, quan realment només existeix un processador.

Les estacions de treball actuals estan dotades de multiprocessadors, això és, nodes amb dos, quatre o més processadors dins de la mateixa màquina. Gràcies a aquesta ampliació dels recursos disponibles, el SO, mitjançant les llibreries de disseny de programes apropiades, pot fer treballar els seus programes en paral·lel, permetent que determinats fragments d'aquests treballin independentment sobre diferents conjunts de dades en diferents processadors. Cada unitat de codi executable en paral·lel dins del procés rep el nom de *fil d'execució* (en anglès, *thread*). En un sistema amb una única CPU, el SO també és capaç de planificar diversos *threads* per simular paral·lelisme, sempre que la càrrega de treball de la màquina no sigui excessiva.

1. Introducció als entorns de treball UNIX

1.5. El sistema de fitxers

La memòria d'un ordinador emmagatzema temporalment tant els programes com les seves pròpies dades durant la seva execució en el processador. Per evitar la pèrdua d'informació quan cessa el subministrament del fluid elèctric en el moment de l'apagat del nostre ordinador, mantenim sempre una còpia de tota la informació en dispositius dissenyats per a tal efecte, com discos durs, CD, DVD o llapis de memòria. Aquesta classe de memòria secundària comparteix un mètode d'organització comuna denominat *sistema de fitxers* (*filesystem*). El tipus d'estructuració de cada volum es pot establir en el moment de donar-li format. Per regla general cada sistema operatiu té una predisposició cap a un determinat format, tolerant no obstant la compatibilitat amb altres sistemes d'emmagatzematge. A la taula 2, hi trobarem alguns exemples de tipus de *filesystem* per SO.

Taula 2. Tipus de formats de sistemes de fitxers.

SO	Format
Linux	EXT2/3/4, XFS, JFS, Btrfs
Windows	FAT, NTFS, exFAT
macOS	HFS, APFS, HFS+

Font: elaboració pròpia.

El *filesystem* indexa tota la informació en un volum d'emmagatzematge, incloent-hi la mida de l'arxiu, els atributs, la localització i la jerarquia en el directori. El *filesystem* també especifica la ruta a l'arxiu mitjançant l'estructura de directoris.

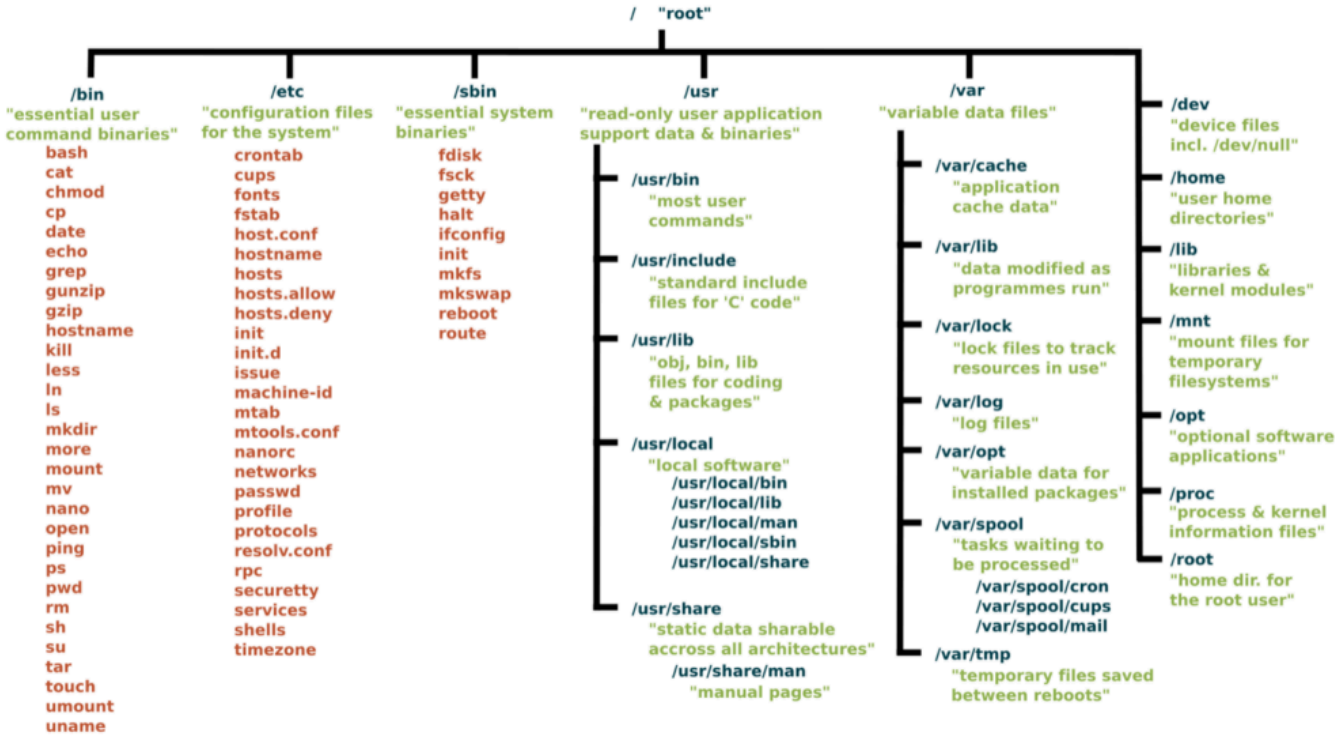

Figura 1. Estructura del *filesystem* de Linux.
Font: linuxfoundation.org

El sistema de fitxers proporciona a l'usuari mecanismes lògics per localitzar els fitxers a través del seu nom i d'una ruta d'accés (en anglès, *path*). A la figura 1 podem veure-hi quina estructura comparteixen els sistemes Linux del sistema de fitxers.

Els usuaris tenen la seva pròpia partició (espai dedicat) dins de «/home». Els discos durs externs i llapis de memòria es localitzen automàticament (es munten) dins del directori «/media» a Linux o al directori «/Volumes» a Mac OS-X. A diferència de Microsoft Windows, els dispositius no reben una única lletra com a identificador, sinó un nom lògic més comprensible.

Els fitxers s'agrupen en carpetes o directoris, conformant una jerarquia que proporciona una organització coherent per a l'usuari. Per construir una ruta que passi per dos directoris *A* i *B* s'ha d'introduir el caràcter separador /, formant la ruta A/B. En un instant concret, el directori en el qual l'usuari es troba rep la denominació simbòlica de «.», mentre que «..» representa el directori immediatament anterior en l'arbre de fitxers. Per accedir a un arxiu especificarem la seva ruta completa (*path* absolut), és a dir, tota la sèrie de directoris des de l'arrel (*root o /*) fins a aquest punt concret del sistema de fitxers. Per accedir a qualsevol fitxer o directori podem descendir des d'aquest punt al llarg del sistema de fitxers. Opcionalment, l'usuari pot senzillament introduir la part de la ruta necessària per completar la resta del camí a partir de la ubicació actual (*path* relatiu). Si volguéssim accedir al fitxer /home/uoc/master/notes.txt (*path* absolut) i estiguéssim a /home/uoc, simplement necessitem indicar ./master/notes.txt (*path* relatiu). Com hem esmentat anteriorment, el «.» indica el directori actual que va seguit de la ruta fins a notes.txt.

A UNIX, s'accedeix als fitxers d'una unitat com a qualsevol altre dispositiu lògic, requerint certs permisos de seguretat per poder efectuar qualsevol operació sobre aquests.

Un programa pot realitzar sobre un fitxer les següents accions:

- Obrir (en anglès, *open*). Per accedir a un fitxer, el procés l'ha d'obrir prèviament i obtenir un codi identificador per referir-s'hi.
- Tancar (en anglès, *close*). Un cop finalitzat l'accés, el fitxer s'ha de tancar perquè un altre procés pugui reutilitzar-lo posteriorment.
- Llegir (en anglès, *read*). El procés utilitza l'identificador d'un fitxer per accedir seqüencialment al seu contingut.
- Escriure (en anglès, *write*). Un procés pot escriure nova informació en un fitxer i sobreescriure el contingut existent anteriorment.
- Afegir (en anglès, *append*). Un procés pot escriure nova informació a continuació del contingut registrat amb anterioritat.

A UNIX tot es redueix a fitxers. Podem trobar tres tipus de fitxers:

- Arxius regulars
- Directoris
- Especials:
  - *Block files*
  - *Character device files*
  - *Pipe files*
  - *Socket files*
  - *Symbolic link files*

Dels tipus especials ens centrarem en els *symbolic link files*. A més dels fitxers regulars i dels directoris que tots coneixem, es poden crear enllaços als fitxers des d'altres punts del sistema de fitxers (en anglès, *links*). D'aquesta manera, evitem introduir la ruta completa d'accés en cada ocasió. Aquests enllaços a UNIX poden contenir la ruta d'accés del fitxer original (en anglès, *soft links* o *symbolic links*) o proporcionar un accés físic compartit a aquest amb un nom diferent (en anglès, *hard links*). Els *soft links* equivaldrien a un accés directe a Windows. Amb aquest mecanisme, l'usuari pot apuntar a un mateix fitxer des de diversos llocs del sistema i sense l'existència de múltiples còpies.

UNIX té un característic mecanisme de seguretat per protegir la integritat del sistema de fitxers. Únicament els usuaris autoritzats poden efectuar operacions sobre un fitxer o directori. Segons la seva procedència, cada usuari del sistema pertany a un d'aquests dominis: l'usuari (*user*), el grup de treball (*group*) o l'entorn exterior (*others*). Les operacions permeses sobre arxius són la lectura (*read*), l'escriptura/modificació/eliminació (*write*) i l'execució (*execute*). En el cas dels directoris, hi ha convencions similars per restringir l'accés al seu interior, denotat en aquest cas amb el permís d'execució.

Per regla general, l'autor d'un fitxer en posseeix inicialment tots els drets, i en garanteix la lectura i l'execució als membres del seu grup de treball. Segons el grau de privacitat permès pels tenidors dels drets, un usuari de l'entorn exterior pot estar habilitat per veure aquests fitxers o no. En qualsevol context, l'administrador de la màquina (en anglès, *root*) pot revocar els permisos de seguretat d'un fitxer. Per poder modificar els permisos de fitxers o directoris més endavant veurem la utilització de l'ordre *chmod*.

1. Introducció als entorns de treball UNIX

1.6. Funcionament de les màquines virtuals

Fins fa relativament poc temps, no era gens senzill per als usuaris d'ordinadors personals disposar d'una màquina funcionant amb Linux. Implicava la desinstal·lació del SO prèviament instal·lat o, si més no, la creació de particions independents amb un gestor d'arrencada dual que permetés la coexistència d'ambdós sistemes. Les distribucions de Linux, a més, mancaven d'un protocol simple d'instal·lació, per la qual cosa requerien un profund coneixement en l'àmbit tècnic de la màquina. Afortunadament, en el moment actual s'han superat àmpliament moltes de les barreres que compliquen l'accés a aquesta tecnologia. De fet, avui dia podem provar fàcilment *in situ* la majoria de les distribucions Linux als nostres ordinadors sense modificar la seva configuració, mitjançant l'ús de les anomenades màquines virtuals.

Una màquina virtual (MV) és un programa que imita el funcionament d'un ordinador, treballant com una aplicació convencional dins de la nostra pròpia màquina. D'aquesta manera, mentre la nostra màquina està governada per un SO que rep la denominació d'hoste (en anglès, *host*), la MV funciona sota el control d'un segon SO que actua com a convidat (en anglès, *guest*). Per proporcionar aquesta funcionalitat al nostre ordinador, cal instal·lar un *software* de virtualització capaç de gestionar múltiples màquines virtuals simultàniament. Els programes gestors de màquines virtuals poden instal·lar-se en múltiples plataformes per actuar com a hoste i, dins d'una finestra, són capaços d'executar una màquina fictícia gestionada per un altre SO diferent com a convidat. En termes pràctics, la MV per a l'ordinador hoste és una aplicació convencional, mentre que des de l'interior de l'emulació agraeix que el SO convidat cregui que treballa sobre una veritable màquina física funcionant a la seva sencera disposició.

El sistema de fitxers de la MV convidada s'emmagatzema físicament en un únic fitxer dins del nostre ordinador, juntament amb les opcions de configuració establertes durant la instal·lació. Lògicament, la MV té accés a determinats perifèrics de la nostra pròpia màquina, com ara el teclat, el ratolí, la pantalla o l'accés a Internet. Per realitzar l'intercanvi d'informació entre la MV i la nostra màquina podem accedir als dispositius USB connectats físicament al nostre ordinador, crear una carpeta compartida entre el *guest* i el *host* o dipositar els nostres fitxers a la xarxa a través de diferents portals del núvol. No obstant això, romandran ocults per a la MV tant la configuració real del nostre ordinador com el nucli del nostre sistema de fitxers muntat des dels nostres discos durs interns. La gestió de diferents tipus de dades dins de la MV repercuteix, òbviament, en un temps de resposta més gran que en el cas de treballar de forma nativa amb el mateix SO convidat, però les últimes versions dels programes de virtualització estan clarament disminuint aquestes diferències. En conclusió, aquesta aproximació resulta enormement atractiva per provar qualsevol nou sistema sense modificar el nostre entorn de treball habitual.

1. Introducció als entorns de treball UNIX

1.7. El terminal com a eina de treball

1.7.1. El terminal

L'entorn de treball d'Ubuntu (https://ubuntu.com/), igual que qualsevol sistema de la família Gnu/Linux, compta amb un gestor gràfic de X-Windows (en català, *finestres*) que proporciona a l'usuari una amigable interfície per accedir als recursos de la seva màquina. No obstant això, en els primers sistemes operatius de les computadores no hi havia entorns interactius controlats pel ratolí ni pantalles amb la resolució gràfica que coneixem avui dia.

Una de les típiques icones associades a l'aplicació del terminal. A Gnu/Linux també rep el nom d'*intèrpret d'ordres* (en anglès, *shell*). En la figura 2 es mostra el símbol que representa l'intèrpret d'ordres.

Figura 2. Símbol d'intèrpret d'ordres.

El treball des de terminals de línies d'ordres (en anglès, *CLI*, *command line interface*) des d'una *shell*, per exemple, tipus *bash* és important en un entorn de bioinformàtica per diverses raons:

- Flexibilitat. L'ús del terminal permet a l'usuari una major flexibilitat i control sobre el processament i anàlisi de les dades de bioinformàtica, ja que es pot utilitzar una àmplia varietat d'eines de línia d'ordres i es poden combinar de manera efectiva.
- Automatització. L'automatització de tasques és molt més fàcil en un terminal tipus *bash*, perquè es poden crear *scripts* i programes per processar grans quantitats de dades sense la necessitat d'executar cada tasca manualment.
- Reproductibilitat. El terminal tipus *bash* permet a l'usuari reproduir fàcilment una anàlisi o processament de dades en qualsevol moment, ja que totes les ordres i operacions realitzades queden registrats en l'historial d'ordres.
- Eficiència. Treballar des d'un terminal tipus *bash* és sovint més eficient que treballar en una interfície gràfica d'usuari, perquè es poden fer moltes operacions en una sola línia d'ordres.

En resum, treballar des d'una *shell* tipus *bash* en un entorn de bioinformàtica és important per permetre una major flexibilitat, automatització, reproductibilitat i eficiència en el processament i anàlisi de les dades de bioinformàtica.

En l'entorn de recerca en bioinformàtica, la feina des d'un terminal constitueix el nucli de les activitats habituals. Per aquesta raó, la majoria dels sistemes actuals mentenen encara el terminal com una aplicació essencial. Com veurem durant l'assignatura, aquesta eina resulta ideal per dissenyar protocols de treball que requereixen l'accés repetit a determinats conjunts de dades per dur a terme càlculs intensius.

La *shell* funciona mitjançant l'execució de l'intèrpret d'ordres, que roman en espera fins que l'usuari hi agrega una nova ordre. Quan s'executa una ordre, l'intèrpret crea un nou procés per dur a terme la tasca. Existeixen dos modes d'execució d'una ordre: el mode en primer pla o síncron, que bloqueja el terminal durant l'execució del procés (en anglès, *foreground*), i el mode en segon pla o asíncron (en anglès, *background*), que no interromp l'activitat ordinària del terminal i permet llançar noves ordres mentre el procés s'executa en segon pla. Per exemple, és possible editar un arxiu de text en una finestra a part mentre s'executen altres ordres al terminal. Un cop una ordre asíncrona finalitza, l'intèrpret informa l'usuari apropiadament. En el subapartat 1.11 («Gestió bàsica de processos») s'amplien els coneixements sobre les ordes per gestionar l'estat dels processos en execució.

En la majoria dels sistemes Gnu/Linux, la *Shell* és la *shell bash*. Per saber quina és la *shell* d'inici de sessió predeterminada, escriviu les ordres al terminal que us traslladem a continuació. Des d'aquest moment, és molt recomanable que poseu en pràctica els exemples inclosos en aquesta assignatura per mostrar el funcionament del terminal. D'altra banda, el caràcter $ denota l'entrada d'ordres des del terminal; haureu d'escriure el següent a continuació d'aquest símbol:

```
$ whoami
```

```
student  pts/1        2023-04-19 15:12 (:0)
```

```
$ grep student /etc/passwd
```

```
student:x:1000:1000:student,,,:/home/student:/bin/bash
```

L'ordre `who am i` mostra el vostre nom d'usuari, i l'ordre `grep` mostra la definició del vostre compte d'usuari a l'arxiu /etc/passwd. L'últim camp en aquesta entrada mostra que la *shell bash* (/bin/bash) és el vostre *shell* (la que s'inicia quan obrim sessió o una finestra del terminal).

Val la pena conèixer l'intèrpret d'ordres *bash*, no només perquè és el predeterminat en la majoria de les instal·lacions, sinó perquè és el més utilitzat en les certificacions professionals de Gnu/Linux.

1. Introducció als entorns de treball UNIX

1.7. El terminal com a eina de treball

1.7.2. Execució d'ordres

La forma més senzilla d'executar una ordre és escriure el nom de l'ordre des d'una *shell*. Des del teu escriptori *ubuntu*, obriu una finestra terminal i després escriviu la següent ordre:

```
$ date
```

```
Wed 19 Apr 15:26:05 CEST 2023
```

Escriure l'ordre `date`, sense opcions ni arguments, mostra el dia, mes, data, hora, zona horària i any actuals, tal com es mostra a dalt. Potser en el vostre terminal no veieu el mateix perquè el format pot canviar depenent de la zona horària. A continuació, algunes altres ordres que podeu provar:

```
$ pwd
```

```
/home/student
```

```
$ hostname
```

```
ubuntuM0151
```

```
$ ls
```

```
2021   Desktop     Downloads    PAC1   Scripts   Work
d2     Documents   index.html   PAC2   var
```

L'ordre `pwd` mostra el vostre directori de treball actual. Escriure `hostname` mostra el nom de `host` del vostre ordinador. L'ordre `ls` llista els arxius i directoris en el vostre directori actual.

Tot i que moltes ordres es poden executar simplement escrivint els noms de les ordres, és més comú escriure alguna cosa més després de l'ordre per modificar el seu comportament. Els caràcters i paraules que podeu escriure després d'una ordre es denominen *opcions* i *arguments*.

1. Introducció als entorns de treball UNIX

1.7. El terminal com a eina de treball

1.7.3. Entenent la sintaxi de les ordres

La majoria de les ordres tenen una o més opcions que podeu agregar per canviar el seu comportament. Les opcions solen consistir en una sola lletra, precedida per un guió. Tanmateix, podeu agrupar opcions d'una sola lletra juntes o precedir cadascuna d'elles amb un guió per usar més d'una opció alhora. Per exemple, els següents dos usos d'opcions per a l'ordre `ls` són equivalents:

```
$ ls -l -a -t
```

```
$ ls -lat
```

En tots dos casos, s'executa l'ordre `ls` amb les opcions `-l` (llistat llarg), `-a` (mostrar arxius ocults amb punts), i `-t` (llistar per temps). Algunes ordres inclouen opcions que estan representades per una paraula completa. Per indicar-li a una ordre que faci servir una paraula completa com a opció, generalment cal precedir-la amb dos guions (`--`). Per exemple, per utilitzar l'opció d'ajuda, en moltes ordres afegeixes `--help` en la línia d'ordres. Sense els dos guions, les lletres *h*, *e*, *l* i *p* serien interpretades com a opcions separades.

Moltes ordres també accepten arguments després que certes opcions siguin ingressades o al final de tota la línia d'ordres. Un argument és una peça extra d'informació, com un nom d'arxiu, directori, nom d'usuari, dispositiu o un altre element que indica a l'ordre sobre què actuar.

Per exemple, `cat /etc/passwd` mostra el contingut de l'arxiu `/etc/passwd` a la teva pantalla. En aquest cas, `/etc/passwd` és l'argument. En general, podeu tenir tants arguments com desitgeu en la línia d'ordre, limitats només pel nombre total de caràcters permesos en una línia d'ordre. Aquí hi ha l'exemple d'una opció amb tres lletres que és seguida per un argument:

```
$ cat /etc/passwd
```

```
$ tar -cvf copiaseguridad.tar /home/student
```

En l'exemple de `tar` mostrat anteriorment, les opcions diuen que s'ha de crear (`c`) un arxiu (`f`) anomenat *copieseguretat.tar* que inclogui tot el contingut del directori `/home/student` i els seus subdirectoris, i que mostri missatges detallats mentre es crea la còpia de seguretat (`v`). Atès que *copiaseguretat.tar* és un argument de l'opció *f*, *copiesseguretat.tar* ha de seguir immediatament l'opció.

Aquí hi ha algunes ordres que podeu provar. Observeu com es comporten de manera diferent amb diferents opcions:

```
$ uname
```

```
Linux
```

```
$ uname -a
```

```
Linux ubuntuM0151 4.4.0-138-generic #164-Ubuntu SMP Wed Oct 3 15:02:00 UTC
2018 i686 i686 i686 GNU/Linux
```

L'ordre `uname` mostra el tipus de sistema que estàs executant (Linux). Quan agregueu `-a`, també podeu veure el nom de *host* i la versió del *kernel*.

```
$ date
```

```
Wed 19 Apr 15:44:23 CEST 2023
```

```
$ date +'%d/%m/%y'
```

```
19/04/23
```

```
$ date +'%A, %B %d, %Y'
```

```
Wednesday, April 19, 2023
```

L'ordre `date` té alguns tipus especials d'opcions. Per ell mateix, `date` imprimeix simplement el dia, la data i l'hora actuals com es mostra en la primera ordre. Però l'ordre `date` admet l'opció especial + de format, que et permet mostrar la data en diferents formats. Escriviu `date --help` per veure els diferents indicadors de format que podeu fer servir.

> Per avançar per les pàgines del manual fa servir les següents tecles: barra espaiadora (pàgina següent), tecla *b* (pàgina anterior), tecla *Enter* (avançar línia a línia). Per localitzar una paraula concreta, introdueixo el símbol / i el patró de recerca. Per sortir del manual, premeu la tecla *q* (*quit*).

1. Introducció als entorns de treball UNIX

1.7. El terminal com a eina de treball

1.7.4. Localitzant ordres

Ara que heu escrit algunes ordres, pot ser que us pregunteu on estan ubicades aquestes ordres i com el *shell* troba les ordres que escriviu. Per trobar les ordres, el *shell* busca en el que es coneix com *a path* (en castellà, ruta). Per a les ordres que no estan en el vostre *path*, podeu escriure la identitat completa de la ubicació de l'ordre.

Si coneixeu el directori que conté l'ordre que desitgeu executar, podeu escriure la ruta completa, o absoluta, d'aquesta ordre. Per exemple, podeu executar l'ordre `date` que es troba dins del directori /bin escrivint:

```
$ /bin/date
```

Per descomptat, això pot ser inconvenient, especialment si la comanda resideix en un directori amb una ruta llarga. La millor manera és tenir les ordres emmagatzemades en directoris coneguts i després afegir aquests directoris a la variable d'entorn PATH del vostre *shell*. El *path* consisteix en una llista de directoris que es verifiquen seqüencialment per a les ordres que escriviu. Per veure el vostre *path* actual, escriviu el següent:

```
$ echo $PATH
```

```
/home/student/bin:/home/student/.local/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
```

Els resultats mostren un *path* predeterminat comú per a un usuari regular de Gnu/Linux. Els directoris a la llista del *path* estan separats per dos punts. La majoria de les ordres d'usuari que venen amb Gnu/Linux s'emmagatzemen en els directoris /bin, /usr/bin o /usr/local/bin. Els directoris /sbin i /usr/sbin contenen ordres administratives (alguns sistemes Gnu/Linux no col·loquen aquests directoris en els *paths* dels usuaris regulars). El primer directori mostrat és el directori *bin* en el directori d'inici de l'usuari (/home/student/bin). Si desitges agregar les teves pròpies ordres o *scripts* de *shell*, col·loques el directori *bin* al teu directori d'inici (ie. /home/student/bin per a un usuari anomenat *student*). Aquest directori s'afegeix automàticament al teu *path* en alguns sistemes Gnu/Linux, tot i que és possible que necessitis crear aquest directori o agregar-lo al teu PATH en altres sistemes Gnu/Linux. Llavors, sempre que agregues l'ordre al teu *bin* amb permís d'execució, pots començar a usar-lo simplement escrivint el nom de l'ordre en l'indicador del teu *shell*. Si es considera que la nova ordre estigui disponible per a tots els usuaris, amb l'usuari *root* agrega'l al directori /usr/local/bin o /opt/nom_paquet/bin.

A diferència d'alguns altres sistemes operatius, Gnu/Linux no verifica el directori actual de l'executable. Immediatament, comença a buscar en el *path*, i els executables en el directori actual només s'executen si estan en la variable PATH o si doneu la seva direcció absoluta (com /home/student/scriptx.sh) o relativa (per exemple, i fixeu-vos en el punt abans de la contrabarra, ./scriptx.sh). Els directoris que pertanyen al *path* d'executables es poden esbrinar escrivint $PATH en el *prompt* del terminal:

```
$ echo $PATH
```

```
/home/student/.sdkman/candidates/java/current/bin:
/home/student/miniconda3/bin:
/home/student/miniconda3/condabin:
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games:/snap/bin
```

L'ordre del directori del *path* és important. Els directoris es verifiquen d'esquerra a dreta. Llavors, en aquest exemple, si hi ha una ordre anomenada `foo` ubicada en ambdós directoris /bin i /usr/bin, s'executa el que està al `/bin`. Perquè s'executi l'altra ordre *foo*, heu d'escriure el *path* complet de l'ordre o canviar la teva variable PATH (un exemple pràctic de com canviar la variable PATH i agregar directoris es descriu en els apartats 1.15 i 1.17).

1. Introducció als entorns de treball UNIX

1.7. El terminal com a eina de treball

1.7.5. Recuperació d'ordres mitjançant l'historial d'ordres

Pot ser convenient tenir l'opció de repetir una ordre que executada anteriorment en una sessió de *shell* . Recordar una línia d'ordres llarga i complexa que vàreu escriure de manera incorrecta us pot estalviar problemes. Afortunadament, algunes característiques del *shell* us permeten recordar línies d'ordres anteriors, editar aquestes línies o completar una línia d'ordres parcialment escrita.

L'historial del *shell* és una llista de les ordres que heu ingressat abans. Fent servir l'ordre `history` en un *shell bash*, podeu veure els vostres ordres anteriors. Després, utilitzant diverses característiques del *shell*, podeu recuperar línies d'ordres individuals d'aquesta llista i canviar-les com desitgeu.

Per provar una mica d'edició de línia d'ordres, escriviu el següent:

```
$ ls /usr/bin | sort -f | less
```

Aquesta ordre mostra el contingut del directori `/usr/bin`, ordena el contingut en ordre alfabètic (sense importar majúscules o minúscules) i passa la sortida a `less`. L'ordre `less` mostra la primera pàgina de sortida, després de la qual pots navegar per la resta de la sortida d'una línia (pressiona *enter*) o a pàgines (pressiona la barra espaiadora) alhora. Simplement pressioneu la tecla *q* quan hàgiu acabat. Ara, suposem que voleu canviar la vostra línia d'ordres al terminal d'`usr/bin a /bin` i no voleu escriure massa. Si us col·loqueu en la línia d'ordres, i seguiu els següents passos, podreu canviar part de la línia d'ordres:

- Pressioneu la fletxa cap amunt (↑). Això mostra l'ordre més recent del teu historial de *shell*. Prova de pressionar-la més d'una vegada.
- Pressioneu Ctrl + A. Això mou el cursor al principi de la línia d'ordres.
- Pressioneu Ctrl + E. Això mou el cursor al final de la línia d'ordres.
- Pressioneu Ctrl+F o la fletxa dreta (→). Repeteix aquesta ordre diverses vegades per col·locar el cursor sota la primera barra (/).
- Pressioneu Ctrl+D. Escriu aquesta ordre quatre vegades per eliminar `/usr` de la línia.
- Pressioneu Enter. Executa la línia d'ordres un cop has decidit què executar.

Mentre s'edita una línia d'ordres, en qualsevol moment podeu escriure caràcters normals per afegir-los a la línia d'ordres. Els caràcters apareixen en la posició del cursor de text. Podeu utilitzar la fletxa dreta →, esquerra ← per moure el cursor d'un extrem a un altre de la línia d'ordres. També podeu prémer les tecles de fletxa amunt ↑ i avall ↓ per recórrer les anteriors ordres a la llista de l'historial per seleccionar una línia d'ordres per editar.

Després d'escriure una línia d'ordres, tota la línia es guarda a la llista d'historial del teu *shell*. La llista s'emmagatzema en el *shell* actual fins que sortiu del terminal. A més, cada ordre que s'escriu en la línia d'ordres i s'executa, tingui sentit o no, s'escriu en un arxiu d'historial, on qualsevol ordre pot ser recordada per ser executada novament en una altra sessió. Un cop es recupera una ordre, es pot modificar la línia d'ordres, com s'ha descrit anteriorment.

Per veure la vostra llista d'historial, utilitzeu l'ordre `history`. Escriviu l'ordre sense opcions o seguida d'un número per llistar aquesta quantitat de les ordres més recents. Per exemple,

```
$ history 9
```

```
2012  date
2013  date +'%d/%m/%y'
2014  date +'%A, %B %d, %Y'
2015  ls
2016  ls Scripts/
```

```
2017  ls
2018  echo $PATH
2019  ls /usr/bin | sort -f | less
2020  history 9
```

En lloc de simplement executar una línia d'ordre de l'historial de manera immediata, es pot recuperar una línia en particular i editar-la. Se'n mostren diversos exemples.

Exemples

`$ !n`: Executar el número d'ordre. Reemplaceu la *n* amb el número de la línia d'ordre i aquesta línia s'executarà. Per exemple, així és com podeu repetir l'ordre de data que es mostra com el número d'ordre *2012* en la llista d'historial anterior:

```
$ !2012
```

```
date

Wed 19 Apr 16:26:58 CEST 2023
```

`$ !!`: Executar ordre anterior. Executa la línia d'ordre anterior. Aquí us mostrem com executar immediatament aquesta mateixa comanda `date`:

```
$ !!
```

```
date

Wed 19 Apr 16:29:06 CEST 2023
```

`$ !?string?`: Executeu l'ordre que conté un `string` (cadena). Això executa l'ordre més recent que conté una cadena de caràcters específica. Per exemple, podeu executar novament l'ordre `date` buscant només una part d'aquesta línia d'ordre de la següent manera:

```
$ !?at?
```

```
date

Wed 19 Apr 16:32:04 CEST 2023
```

1. Introducció als entorns de treball UNIX

1.8. Gestió bàsica de fitxers

1.8.1. Introducció

En aquesta secció, aprendrem els conceptes bàsics per moure'ns pel sistema. Moltes tasques depenen de poder arribar a referenciar la ubicació correcta en el sistema. Com a tal, aquest coneixement forma la base per poder treballar eficaçment a Gnu/Linux. Assegureu-vos d'entendre-ho bé. Si desitgeu seguir-los, inicieu sessió i obriu un terminal. La taula 3 mostra les ordres per crear i utilitzar arxius i directoris.

Taula 3. Ordres per gestionar arxius i directoris.

Ordre	Resultat
`cd`	(*change directory*) Canviar un altre directori
`pwd`	(*print working directory*) Imprimir el nom de l'actual directori de treball
`mkdir`	(*make directory*) Crear un directori
`rmdir`	(*remove directory*) Eliminar un directori buit
`rm -r`	(*remove*) Eliminar el contingut d'un directori no buit
`rm`	(*remove*) Eliminar fitxers
`chmod`	(*change file mode*) Canviar els permisos d'un fitxer o directori
`ls`	(*list*) Llistar el contingut d'un directori
`cp`	(*copy*) Copiar un arxiu
`mv`	(*move*) Moure un fitxer

Font: elaboració pròpia.

Quan inicieu sessió en un sistema Gnu/Linux i obriu un terminal, directament, us trobareu al directori d'inici (abreviat també amb el símbol ~). Depenent de la instal·lació, aquest directori acostuma a emmagatzemar-se al seu nom en un subdirectori de la carpeta /home/. Per certificar en cada moment en quin lloc del *path* de directoris ens trobem, podem fer servir l'ordre `pwd` (en anglès, *print working directory*).

Una de les ordres més bàsiques que s'utilitza al terminal és `cd`. L'ordre `cd` es pot fer servir sense opcions (per portar-lo al seu directori d'inici) o amb *paths* absoluts o relatius. Considereu les següents ordres:

```
$ cd /usr/lib/
```

```
$ pwd
```

```
/usr/lib
```

```
$ cd gcc
```

```
/usr/lib/gcc
```

```
$ cd
```

```
$ pwd
```

```
/home/student
```

L'opció /usr/bin representa el *path* absolut a un directori en el sistema. Atès que comença amb una barra diagonal (/), aquest *path* indica al terminal que comenci a l'arrel del sistema d'arxius i el porti al directori *lib* que es troba al directori *usr*. L'opció *gcc* de l'ordre `cd` indica que s'ha de buscar un directori anomenat *gcc* que és relatiu al directori actual. Per tant, això fa que /usr/lib/gcc sigui el seu nou directori. Després d'això, es torna al directori d'inici només d'escriure `cd`. Si alguna vegada us pregunteu on us trobeu en el sistema d'arxius, l'ordre `pwd` us ho mostrarà.

Els següents passos us guiaran a través del procés de creació de directoris dins del vostre directori d'inici i com us podeu moure entre ells, amb una menció sobre com establir els permisos apropiats dels arxius. Us recomano que executeu totes les ordres que estan escrites.

1. Aneu al vostre directori d'inici. Per fer això, simplement escriviu `cd` en un terminal i pressioneu *enter*.

2. Per assegurar-vos que esteu en el vostre directori d'inici, escriviu `pwd`.

```
$ pwd
```

```
/home/student
```

3. Creeu un nou directori anomenat `testHIB` en el vostre directori d'inici,

```
$ mkdir testHIB
```

4. Verifiqueu els permisos del directori:

```
$ ls -ld testHIB
```

```
drwxrwxr-x 2 student student 4096 Apr 19 18:06 testHIB/
```

Aquesta llista mostra que `testHIB` és un directori (*d*). La *d* va seguida dels permisos (`rwxr-xr-x`), els quals s'expliquen més endavant a la secció «Entenent els permisos i la propietat dels arxius». La resta de la informació indica el propietari (`student`), el grup (`student`) i la data en què els arxius en el directori van ser modificats per última vegada.

5. Escriu `$ chmod 700 testHIB`

Aquest pas canvia els permisos del directori perquè hi tingueu accés complet i ningú més hi tingui accés (els nous permisos s'han de llegir `rwx------`).

6. Feu que el directori de prova sigui el seu directori actual:

```
$ cd testHIB
```

```
$ pwd
```

```
/home/student/testHIB
```

Podeu crear arxius i directoris en el directori de prova juntament amb les descripcions en la resta d'aquest capítol. Quan necessiteu identificar el vostre directori d'inici en una línia d'ordre de *shell*, podeu fer servir el següent:

- `$HOME` Aquesta variable d'entorn emmagatzema el nom del teu directori d'inici.
- ~ Representa el teu directori d'inici en la línia d'ordre. També podeu fer servir l'accent per identificar el directori d'inici d'una altra persona.

Altres formes especials d'identificar directoris a la *shell* es descriuen a continuació, amb exemples.

- `.` Un sol punt (`.`) es refereix al directori actual.
- `..` Dos punts (`..`) es refereixen a un directori directament damunt del directori actual.
- `$PWD` Aquesta variable d'entorn es refereix al directori de treball actual.

```
$ pwd
```

```
/home/student
```

```
$ cp file01 ../file02
```

```
$ cd ..
```

```
$ pwd
```

```
/home
```

```
$ mv file02 ./student
```

```
$ cp file01 /home/student
```

```
$ cd ~
```

```
$ pwd
```

```
/home/student
```

Escriure *paths* es pot tornar tediós. La línia d'ordres té un petit mecanisme que ens ajuda en aquest aspecte. Es diu *Autocompletat del tabulador*.

Quan comenceu a escriure un *path* (en qualsevol lloc de la línia d'ordres), podeu pressionar la tecla *tab* al teu teclat en qualsevol moment, cosa que invocarà una acció d'autocompletat. Si no succeeix res, això significa que hi ha diverses possibilitats. Si pressioneu *tab*, us mostrarà aquestes possibilitats. Després podeu continuar escrivint, pressionar novament *tab*, i tornarà a intentar autocompletar per a vosaltres. És una mica difícil de demostrar aquí, per la qual cosa probablement serà millor si ho intenteu vosaltres mateixos.

1. Introducció als entorns de treball UNIX

1.8. Gestió bàsica de fitxers

1.8.2. Metacaràcters i operadors

Si esteu llistant, movent, copiant, eliminant o realitzant qualsevol altra acció amb arxius en el vostre sistema Gnu/Linux, existeixen certs caràcters especials, denominats *metacaràcters* i *operadors*, que us ajudaran a treballar amb arxius de manera més eficient. Els metacaràcters poden ajudar a fer coincidir un o diversos arxius sense haver d'escriure completament cada nom d'arxiu. Els operadors us permeten adreçar informació d'una ordre o arxiu a una altra ordre o arxiu.

D'altra banda, també cal introduir certs mecanismes de citació. Amb exemples, serà més fàcil d'entendre:

Caràcter escapament

Una barra invertida no citada «\» és el caràcter *d'escapament* de *bash*: preserva el valor literal del següent caràcter que segueix, amb l'excepció de la nova línia.

```
$ echo variable; argument
```

```
variable

argument: command not found
```

Si s'escriu «\;» t'ajuda a utilitzar «;» com un caràcter normal

```
$ echo variable \; argument
```

```
variable ; argument
```

Un exemple més subtil. L'objectiu és crear un sol fitxer que s'anomeni *el meu fitxer.txt*

```
$ touch el meu fitxer.txt
```

```
$ ls meu*txt
```

```
ls: cannot access meu*txt': No such file or directory
```

```
$ rm fitxer.txt meu
```

Amb el caràcter *escape* es genera un únic fitxer

```
$ touch meu\ fitxer.txt
```

```
$ ls meu*txt
```

```
'meu fitxer.txt'
```

```
$ rm meu\ fitxer.txt
```

Cometes simples

Tancar caràcters amb cometes simples (') preserva el valor literal de cada caràcter dins de les cometes. No hi pot haver una cometa simple entre cometes simples, fins i tot quan està precedida per una barra invertida. I cap caràcter és especial dins de les cadenes entre cometes simples. Exemples:

```
$ echo 'variable; argument'
```

```
variable; argument
```

Es poden col·locar cadenes representades per diferents mecanismes de citació una al costat de l'altra per concatenar-les. Un altre exemple:

```
# concatenació of 4 strings

# 1: '@ordre = '

# 2: \'

# 3: 'sort and grep'

# 4: \'
```

```
$ echo '@ordre = '\'' sort and grep'\'
```

```
@ordre = 'sort and grep\
```

Cometes dobles

Tancar caràcters en cometes dobles (") preserva el valor literal de tots els caràcters dins de les cometes, amb l'excepció de $,' , , i quan l'expansió d'història està habilitada, ! . Aquí hi ha un exemple que mostra la interpolació de variables dins de cometes dobles:

```
$ qty='5'
```

Cap caràcter és especial dins de les cometes simples

```
$ echo 'Des de principi d'any, he executat $qty GWAS'
```

```
Des de principi d'any, he executat $qty GWAS
```

Un ús típic de les cometes dobles és habilitar la interpolació de variables

```
$ echo "Des de principi d'any, he executat $qty GWAS"
```

```
Des de principi d'any, he executat 5 GWAS
```

Llevat que vulguis específicament que la *shell* interpreti el contingut d'una variable, sempre has d'escriure entre cometes la variable per evitar problemes a causa de la presència de metacaràcters de la *shell*

```
$ f='segon fitxer.txt'
```

Seria el mateix que: `echo 'pac informe' > segon fitxer.txt`

```
$ echo 'pac informe' > $f
```

```
bash: $f: ambiguous redirect
```

Seria el mateix que: `echo 'pac informe' > 'segon fitxer.txt'`

```
$ echo 'pac informe' > "$f"
```

```
$ cat "$f"
```

```
pac informe
```

```
$ rm "$f"
```

Ara sí que ens introduïm més profundament en la definició de metacaràcters i operadors.

Metacaràcters

Per estalviar-vos haver de prémer algunes tecles i per poder-vos referir fàcilment a un grup d'arxius, la *shell bash* us permet l'ús de metacaràcters. Aquí hi ha alguns metacaràcters útils per fer coincidir noms d'arxius. A la taula 4 es descriuen els més habituals.

Taula 4. Metacaràcters de fitxer.

Comodí	Descripció
?	Coincidir un caràcter, qualsevol caràcter
*	Coincidir qualsevol quantitat de caràcters
[....]	Coincidir qualsevol dels caràcters entre claudàtors, que poden incloure un rang de lletres o números separats per un guió
[! ....]	No coincidir amb cap dels caràcters entre els claudàtors, que poden incloure un rang de lletres o números separats per un guió

Font: elaboració pròpia.

Proveu alguns d'aquests metacaràcters de coincidència d'arxius anant primer a un directori buit (com el directori de prova descrit a la secció anterior) i creant alguns arxius buits. L'ordre `touch` crea arxius buits. Les línies d'ordres que segueixen mostren com usar metacaràcters de *shell* amb l'ordre `ls` per coincidir amb noms d'arxiu. Qualsevol altra ordre també funciona.

```
$ touch Transcriptomics Proteomics Epigenomics Metagenomics
Pharmacogenomics
```

```
$ ls P*
```

```
Pharmacogenomics  Proteomics
```

S'imprimeix qualsevol arxiu que comenci amb P

```
$ ls P*t*
```

```
Proteomics
```

S'imprimeix qualsevol arxiu que comenci amb P i contingui la t

```
$ ls [EM]*
```

```
Epigenomics  Metagenomics
```

S'imprimeix qualsevol arxiu que comenci per E or M

```
$ ls [P-Z]*
```

```
Pharmacogenomics  Proteomics  Transcriptomics
```

#S'imprimeixen tots els noms d'arxiu que comencen amb una lletra entre P i Z

```
$ ls ???genomics
```

```
Epigenomics
```

S'imprimeix qualsevol arxiu d'11 caràcters que acabi amb la cadena genomics

Metacaràcters de redireccionament d'arxius

Les ordres reben dades des de l'entrada estàndard i les envien a la sortida estàndard. Fent servir *pipes* (en català, tubs) es pot dirigir la sortida estàndard d'una ordre a l'entrada estàndard d'una altra. Amb arxius, es pot fer servir el signe menor que (<) i més gran que (>) per dirigir dades cap a i des d'arxius. A la taula 5 es recullen els caràcters de redirecció d'arxius:

Taula 5. Metacaràcters de redirecció.

Comodí	Descripció
<	Dirigeix el contingut d'un arxiu a l'ordre. En la majoria dels casos, aquesta és l'acció predeterminada esperada per l'ordre i l'ús del caràcter és opcional; usar «`less seq.fa`» és el mateix que «`less < seq.fa`»
>	Dirigeix la sortida estàndard d'una ordre a un arxiu. Si l'arxiu existeix, el contingut d'aquest arxiu se sobreescriu
2>	Dirigeix l'error estàndard (missatges d'error) a l'arxiu
&>	Dirigeix tant la sortida estàndard com l'error estàndard a l'arxiu
>>	Dirigeix la sortida d'una ordre a un arxiu, agregant la sortida al final de l'arxiu existent

Font: elaboració pròpia.

Les següents línies són exemples realitzats mitjançant línies d'ordre on la informació s'adreça cap a i des d'arxius:

```
$ mail root < ~/.bashrc
```

```
$ man chown | col -b > /tmp/chown
```

```
$ echo "Estic practicant els exercicis d'HIB a $(date)" >>
~/testHIB/testimoni
```

En el primer exemple, el contingut de l'arxiu . *bashrc* en el directori d'inici s'envia en un missatge de correu a l'usuari *root* de la màquina Gnu/Linux. La segona línia d'ordre de la pàgina del manual de *chown* (usant l'ordre `man`) elimina els espais en blanc addicionals (`col -b`) i envia la sortida a l'arxiu /tmp/chown (esborrant l'arxiu /tmp/chown anterior, si existís). L'ordre final genera un fitxer de text que es visualitza amb l'ordre `cat`:

```
$cat testHIB/testimoni
```

```
Estic practicant els exercicis d'HIB a Wed 19 Apr 19:08:04 CEST 2023
```

Un altre tipus de redirecció us permet escriure text que es pot fer servir com a entrada estàndard per a una ordre. Aquí, els documents impliquen ingressar dos caràcters de menor (<<) després d'una ordre seguida d'una paraula. Tot el que escrigui després d'aquesta paraula s'interpretarà com a entrada de l'usuari fins que es repeteixi la paraula en una línia a part.

```
$ mail student Josep Joan Guerau Bego << missatge
```

```
> Us recordo que cal realitzar tots els exercicis
> proposats per entendre l'assignatura.
> > equip HIB
> missatge
```

```
$
```

Aquest exemple envia un missatge de correu als usuaris *student*, Josep, Joan, Guerau i Bego. El text introduït entre <<missatge>> i <<missatge>> esdevé el contingut del missatge.

Caràcters d'expansió de claus

Quan useu claus ({}), podeu expandir un conjunt de caràcters en els noms d'arxiu, noms de directoris o altres arguments que doneu a les ordres. Per exemple, si voleu crear el conjunt d'arxius des de *sequence1* fins a *sequence7*, podeu fer-ho de la següent manera:

```
$ touch sequence{1,2,3,4,5,6,7}
```

```
$ ls
```

```
sequence1 sequence2 sequence3 sequence4 sequence5 sequence6 sequence7
```

```
$ rm sequence?
```

Els elements que s'expandeixen no han de ser números, ni tan sols dígits únics; podríeu fer servir rangs de números o dígits. També podríeu fer servir qualsevol cadena de caràcters, sempre que se separin amb comes. Exemple:

```
$ touch sequence{1..4}-{human,drosophila}
```

```
$ ls
```

```
 sequence1-drosophila  sequence2-drosophila  sequence3-drosophila
sequence4-drosophila
 sequence1-human       sequence2-human       sequence3-human
sequence4-human
```

```
$ rm sequence*
```

Permisos d'arxius i la propietat

Després de treballar amb Gnu/Linux durant un temps, és gairebé segur que obtindreu un missatge de permís denegat. Els permisos associats a arxius i directoris a Gnu/Linux van ser dissenyats per evitar que els usuaris accedeixin als arxius privats d'altres usuaris i per protegir els arxius importants del sistema. Els nou *bits* assignats a cada arxiu per als permisos defineixen l'accés que vosaltres i d'altres usuaris tenen en el vostre arxiu. Els *bits* de permís per a un arxiu regular apareixen com -rwxrwxrwx. Aquests *bits* es fan servir per definir qui pot llegir, escriure o executar l'arxiu.

Dels permisos de nou *bits*, els primers tres *bits* s'apliquen al permís del propietari, els següents tres s'apliquen al grup assignat a l'arxiu i els darrers tres s'apliquen a tots els altres. La *r* significa lectura, la w significa escriptura i la *x* significa permisos d'execució. Si apareix un guió en lloc de la lletra, significa que aquest permís està desactivat per a aquest *bit* associat de lectura, escriptura o execució.

Podeu veure els permisos de qualsevol arxiu o directori escrivint l'ordre ls -ld. L'arxiu o directori anomenat apareix com es mostra en aquest exemple:

```
$ ls -ld testHIB  testHIB/testimoni
```

```
-rw-rw-r--  1 student student     73 Apr 19 19:08 testHIB/testimoni
drwxr-xr-x 2 student student 4096 Apr 19 19:27 testHIB
```

La primera línia mostra que l'arxiu *testimoni* té permís de lectura i escriptura per al propietari i el grup. Tots els altres usuaris tenen permís de lectura, cosa que significa que poden veure l'arxiu, però que no poden canviar el seu contingut ni eliminar-lo. La segona línia mostra el directori testHIB (indicat per la lletra *d* abans dels *bits* de permís). El propietari té permisos de lectura, escriptura i execució, mentre que el grup i altres usuaris només tenen permisos de lectura i execució. Com a resultat, el propietari pot agregar, canviar o eliminar arxius en aquest directori, i tots els altres només poden llegir-ne el contingut, canviar aquest directori i llistar el contingut del directori.

Si sou propietari d'un arxiu, podeu fer servir l'ordre chmod per canviar els permisos com desitges. En un mètode per fer-ho, a cada permís (lectura, escriptura i execució) se li assigna un número: *r* = 4, *w* = 2 i *x* = 1, i s'utilitza el nombre total de cada conjunt per establir el permís. Per exemple, per fer que els permisos estiguin completament oberts per a vosaltres com a propietaris, establiríeu el primer número en 7 (4 + 2 + 1), i després donaríeu al grup i a altres persones permisos només de lectura establint tant el segon com el tercer número en 4 (4 + 0 + 0), de manera que el número final sigui 744. Qualsevol combinació de permisos pot resultar des de 0 (sense permís) fins a 7 (permís complet).

Aquí hi ha alguns exemples de com canviar el permís d'un arxiu (anomenat *arxiu*) i de quin permís en resultaria:

```
# L'execució de l'ordre chmod dona com a resultat aquest permís:
rwxrwxrwx
```

```
$ chmod 777 arxiu
```

```
# L'execució de l'ordre chmod dona com a resultat aquest permís: rwxr-xr-
x
```

```
$chmod 755 arxiu
```

```
# L'execució de l'ordre chmod dona com a resultat aquest permís: rw-r--r-
-
```

```
$ chmod 644 arxiu rw-r--r-
```

```
# L'execució de l'ordre chmod dona com a resultat aquest permís: --------
-
```

```
$ chmod 000 arxiu
```

L'ordre chmod també es pot fer servir de manera recursiva. Per exemple, suposem que es desitja donar una estructura de directoris completa amb permís 755 (rwxr-xr-x), començant al directori $HOME/testHIB. Per fer això, es podria fer servir l'opció -R (recursiva):

```
$ chmod -R 755 $HOME/testHIB
```

Tots els arxius i directoris de sota, incloent-hi el directori Work en el seu directori d'inici, tindran els permisos 755 establerts. Atès que, amb tots els *bits* de permís alhora, l'enfocament dels números per establir canvis de permís generen confusió, és més comú usar lletres per canviar els *bits* de permís en un gran conjunt d'arxius.

També podeu activar i desactivar els permisos d'un arxiu utilitzant els signes més (+) i menys (-), respectivament, juntament amb lletres per indicar quins canvis i per a qui. Fent servir lletres per a cada arxiu podeu canviar els permisos per a l'usuari (*u*), el grup (*g*), altres (*o*) i tots els usuaris (*a*). El que canviaríeu inclou els *bits* de lectura (*r*), escriptura (*w*) i execució (*x*). Per exemple, comenceu amb un arxiu que tingui tots els permisos oberts (rwxrwxrwx). Executeu les següents ordres chmod utilitzant opcions amb signe menys. Els permisos resultants es mostren a la dreta de cada ordre:

```
# L'execució de l'ordre chmod resulta en aquest permís: r-xr-xr-x
```

```
$ chmod a-w arxiu
```

```
# L'execució de l'ordre chmod resulta en aquest permís: rwxrwxrw-
```

```
$ chmod o-x arxiu
```

```
# L'execució de l'ordre chmod resulta en aquest permís: rwx------
```

```
$ chmod go-rwx arxiu
```

Així mateix, els següents exemples comencen amb tots els permisos tancats (---------). El signe més s'utilitza amb chmod per activar els permisos:

```
# L'execució de l'ordre chmod resulta en aquest permís: rw-------
```

```
$ chmod u+rw arxiu
```

```
# L'execució de l'ordre chmod resulta en aquest permís: --x--x--x--x
```

```
$ chmod a+x arxiu
```

```
# L'execució de l'ordre chmod resulta en aquest permís: r-xr-x---
```

```
$ chmod ug+rx arxiu
```

L'ús de lletres per canviar els permisos de manera recursiva amb chmod funciona generalment millor que l'ús de números, perquè es poden canviar els *bits* selectivament, en lloc de canviar tots els *bits* de permís alhora.

1. Introducció als entorns de treball UNIX

1.9. Accedir al contingut dels fitxers

1.9.1. Introducció

Un filtre, en el context de la línia d'ordres de Gnu/Linux, és un programa que accepta dades textuals i les transforma d'una manera particular. Els filtres són una forma de prendre dades en brut, ja hagin estat produïdes per un altre programa o emmagatzemats en un arxiu, i manipular-les perquè es mostrin d'una manera més adequada per trobar el que estem buscant. Aquests filtres sovint tenen diverses opcions de línia d'ordres que modificaran el seu comportament, per la qual cosa sempre és bo consultar la pàgina del manual d'un filtre per veure quin està disponible.

En els exemples que es mostren a continuació, proporcionarem entrada en aquestes ordres mitjançant un arxiu, però també veurem que podem proporcionar entrada a través d'altres mitjans que agreguen molta més potència (taula 6). A més, recordeu que l'arxiu s'especifica com un *path* i, per tant, podeu fer servir *paths* absoluts i relatius, i també comodins. D'altra banda, aquestes eines que estem veient només serveixen per treballar amb fitxers de text, no binaris.

Taula 6. Ordres per accedir al fitxer.

Ordre	Descripció
`cat`	Imprimeix un fitxer al terminal
`more`	Mostra el resultat de l'execució d'una ordre al terminal d'una pàgina alhora
`head`	És una ordre que imprimeix les primeres deu línies de la seva entrada, però podem modificar això amb un argument de línia d'ordre
`tail`	És una ordre que imprimeix les últimes deu línies de la seva entrada, però podem modificar això amb un argument de línia d'ordre
`less`	És un visor de fitxers de text. Amb aquest programa no podrem editar el fitxer, però sí navegar pel seu contingut
`nl`	*`number line`* significa 'numerar línies', i això és exactament el que fa l'ordre
`wc`	*`word count`* significa 'comptar paraules' i fa concretament això (i també compta caràcters i línies). De forma predeterminada, donarà un recompte de les tres possibilitats, però, usant opcions de línia d'ordre, podem limitar-lo al que necessitem
`diff`	Compara línia a línia dos fitxers de text
`paste`	Uneix fitxers tabulars línia per línia
`od`	*`octal dump`* converteix l'entrada en múltiples formats, amb format octal per defecte, i ajuda a comprendre les dades complexes que no són llegibles per als humans
`sed`	`stream editor Editor` de flux. Us permet fer una recerca i reemplaçament, entre altres accions, en les vostres dades

Font: elaboració pròpia.

En el camp de la bioinformàtica els arxius són molt grans; fins i tot els editors en línia poden generar problemes per obrir-los. Existeixen altres formes d'accedir als continguts del fitxer. Un d'ells seria imprimir el fitxer al terminal utilitzant l'ordre `cat`.

> Recorda que amb Crtl + C els programes s'acaben immediatament i es torna a mostrar el *prom*pt (en català, la línia d'ordres).

```
$cat hg38_RefSeq.txt
```

`cat` és, a més, capaç de concatenar textos un darrere l'altre en l'ordre en què els passem, i de mostrar-los en pantalla..

```
$ cat file1 file2 file3
```

O es pot generar un nou fitxer.

```
$ cat seq1 seq2 >> set1-2.txt
```

Algunes opcions interessants de `cat` són:

- `-A`: mostra també els caràcters de control, bàsicament els tabuladors (com `^I`) i els retorns de carro (`$`).
- `-n`: numera totes les línies.

Per obtenir una visió general del contingut de l'arxiu sense ocupar tot el terminal, es poden imprimir només les primeres línies usant l'ordre `head`:

```
$ head -3 hg38_RefSeq.txt
```

```
 #bin name chrom strand    txStart    txEnd cdsStart   cdsEnd    exonCount
exonStarts exonEnds   score name2     cdsStartStat    cdsEndStat exonFrames

 0    NM_001276352.2  chr1 -    67092164   67134970   67093579 67127240    9
67092164,67096251,67103237,67111576,67115351,67125751,67127165,67131141,6713
67093604,67096321,67103382,67111644,67115464,67125909,67127257,67131227,6713

 0    C1orf141   cmpl cmpl 2,1,0,1,2,0,0,-1,-1, 0     NM_001276351.2  chr1 -
67092164   67134970     67093004   67127240   8
67092164,67095234,67096251,67115351,67125751,67127165,67131141,67134929,
67093604,67095421,67096321,67115464,67125909,67127257,67131227,67134970,   (
C1orf141   cmpl cmpl     0,2,1,2,0,0,-1,-1,
```

Hi ha l'ordre `tail`, i permet imprimir el final dels arxius.

```
$ tail -2 hg38_RefSeq.txt
```

Un altre comportament de `tail` que resulta útil és que pot mostrar totes les línies excepte les k primeres línies. Per això cal fer servir l'opció `-n` i el nombre de línies que volem ometre precedit per un `+`. Si es volen ometre les primeres vint-i-dues línies, podeu escriure:

```
$ tail -n +22 hg38_RefSeq.txt
```

Quan es necessita examinar un arxiu de text per familiaritzar-se amb el seu contingut, és comú obrir-lo i navegar-hi. Tanmateix, si l'arxiu és molt gran, hi poden haver problemes quan intenteu obrir-lo amb un editor de text.

En aquests casos, una eina útil és `less`, un visor d'arxius de text que pot operar arxius immensos sense problemes. Tot i que `less` no permet l'edició de l'arxiu, sí que ens permet navegar pel seu contingut de manera interactiva. Quan executeu *less*, el programa s'obrirà en el terminal i farà que el *prompt* desaparegui temporalment. Podrem sortir del programa en qualsevol moment pressionant la tecla *q*.

```
$ less hg38_RefSeq.txt
```

Dins de `less` disposem de diverses ordres per moure'ns pel fitxer:

- Barra d'espai: pàgina següent.

- b: pàgina anterior.
- 100g: va a la línia 100 (o a la que li indiquem).
- -S: talla o no talla les línies llargues.
- /paraula: busca la cadena de text que li indiquem (accepta expressions regulars).
- n: va a la següent paraula que coincideix amb la recerca.
- N: va a la paraula anterior que coincideix amb la recerca.
- q: surt del fitxer.
- h: ajuda.

L'ordre `wc` significa 'comptatge de paraules', i això és el que fa (així com comptar caràcters i línies). Per defecte, ens donarà un recompte de totes tres coses, però fent servir opcions de línia d'ordre, es pot limitar allò que es busca. De vegades només volem un d'aquests valors. Per exemple, «`-l`» ens donarà només les línies, «`-w`» ens donarà les paraules i «`-m`» ens donarà els caràcters.

```
$ wc hg38_RefSeq.txt
```

```
172767  2764272 56256545 hg38_RefSeq.txt
```

```
$ wc -l hg38_RefSeq.txt
```

```
172767 hg38_RefSeq.txt
```

La segona ordre imprimeix per pantalla només un recompte de línies, però la primera ordre ens informa del nombre de línies, paraules i caràcters que té el fitxer.

L'ordre `diff` permet dur a terme la comparació línia a línia de dos fitxers de text. Òbviament, hi ha formes més sofisticades de comparar arxius. Tanmateix, aquesta funció és extremadament útil per confirmar quan dos fitxers no són idèntics (una de les operacions més comunes en bioinformàtica).

```
$ diff file1.txt file2.txt
```

Finalment, es comenta la comanda `paste`. Suposem que tenim dos fitxers, un amb dades sobre la progressió de la malaltia d'una sèrie de malalts i un altre amb el seu genotipat:

```
$ cat pacients.txt
```

```
id_pacient,nivell_glucosa

1,190

2,250

3,220

4,260

5,160
```

```
$ cat genotipat.txt
```

```
id_pacient,SNP_a,SNP_b
1,AA,CC
2,AC,GG
3,AA,CG
4,AT,GG
5,AA,CC
```

Es poden fusionar els dos arxius usant l'ordre `paste` línia a línia:

```
$ paste -d',' pacients.txt genotipat.txt
```

```
id_pacient,nivell_colesterol,id_pacient,SNP_a,SNP_b

1,190,1,AA,CC

2,250,2,AC,GG

3,220,3,AA,CG

4,260,4,AT,GG

5,160,5,AA,CC
```

La quantitat d'informació emmagatzemada en qualsevol entorn bioinformàtic és considerable, i sovint ocupa diversos terabytes. Per exemple, la seqüència del genoma humà està composta per al voltant de tres mil milions de nucleòtids, cosa que es tradueix en aproximadament tres gigabytes. Això implica que sovint cal comprimir directoris sencers. La instrucció `tar` pot crear un paquet únic a partir del directori, que posteriorment pot ser comprimit amb `gzip`. La taula 7 descriu les ordres per comprimir/descomprimir més habituals:

Taula 7. Ordres per accedir als fitxers.

Ordre	Descripció
`tar`	Empaquetar múltiples arxius i directoris
`gzip`	Comprimir i descomprimir arxius
`zmore`	Descomprimir i visualitzar un arxiu
`zcat`	Descomprimir i bolcar un arxiu

Font: elaboració pròpia.

Un exemple amb `tar`:

```
$ tar -cvf backup.tar /home/student
```

En l'exemple anterior de `tar`, les opcions indiquen que es crea (*c*) un arxiu (*f*) anomenat *backup.tar* que inclou tots els continguts del directori /home/student i els seus subdirectoris, i que es mostrin missatges detallats mentre es crea la còpia de seguretat (*v*). Atès que backup.tar és un argument de l'opció *f*, backup.tar ha de seguir immediatament a l'opció. Altres opcions són:

```
$ tar -xvf backup.tar
```

La nova opció (x) indica que es desempaqueta el fitxer backup.

```
$ tar -czvf backup.tar.gz /home/student
```

L'opció (z) indica que després que es creï el fitxer es comprimeixi.

```
$ tar -xzvf backup.tar.gz
```

Aquesta combinació d'opcions indica que el fitxer es descomprimeix i es desempaqueta.

Les ordres `more` i `cat` posseeixen una versió especial que integra l'ordre `gzip` com un filtre addicional. Com a resultat, podem visualitzar directament al terminal un fitxer comprimit:

```
$ gzip NANOGgene.fa
```

```
$ zmore NANOGgene.fa.gz | head -5
```

```
>hg19_refGene_NM_024865 range=chr12:7941992-7948657
TTCATTATAAATCTAGAGACTCCAGGATTTTAACGTTCTGCTGGACTGAG
CTGGTTGCCTCATGTTATTATGCAGGCAACTCACTTTATCCCAATTTCTT
GATACTTTTCCTTCTGGAGGTCCTATTTCTCTAACATCTTCCAGAAAAGT
CTTAAAGCTGCCTTAACCTTTTTTCCAGTCCACCTCTTAAATTTTTTCCT
```

```
$ zcat NANOGgene.fa.gz | tail -3
```

```
GTTGGTTTAAGTTCAAATGAATGAAACAACTATTTTTCCTTTAGTTGATT
TTACCCTGATTTCACCGAGTGTTTCAATGAGTAAATATACAGCTTAAACA
TAA
```

```
$ gzip -d NANOGgene.fa.gz
```

1. Introducció als entorns de treball UNIX

1.9. Accedir al contingut dels fitxers

1.9.2. Edició d'arxius amb l'editor *vim*

Vim, també conegut com *a Vi Improved*, és un editor programable altament potent i versàtil que es troba present en tots els sistemes GNU/Linux. Una de les seves principals característiques és que disposa de diferents modes (normal, visual, *insert*, *command-line, select* i *ex*), que s'alternen per dur a terme diferents operacions. Aquesta particularitat el diferencia de la majoria dels editors comuns, que solen tenir una única opció (*insert*) on s'introdueixen les ordres mitjançant combinacions de tecles o interfícies gràfiques.

La totalitat del control de *vim* es realitza a través del teclat des d'un terminal, cosa que el fa ideal per ser utilitzat sense problemes a través de connexions remotes, ja que no desplegar un entorn gràfic perquè no carrega el sistema. Aprendre a utilitzar *vim* és altament recomanable per la seva capacitat per augmentar la productivitat i l'eficiència en la programació.

En aquest apartat us expliquem dues modalitats de *vim*: mode d'inserció (o entrada) i mode d'edició. En el mode d'inserció, es pot introduir o ingressar contingut a l'arxiu. En el mode d'edició, ens podem moure per l'arxiu, realitzar accions com eliminar, copiar, buscar i reemplaçar, guardar, etc. Un error comú és començar a ingressar ordres sense tornar primer al mode d'edició o començar a escriure una entrada sense entrar primer en el mode d'inserció. Si feu qualsevol d'aquestes coses, generalment serà fàcil recuperar, així que no us preocupeu, després us ho explicarem.

Comencem! Serà difícil demostrar-vos tot el procés en la seva major part, així que, en lloc vostre, citarem el que cal escriure i haureu d'intentar-ho, a veure com us va.

Primer, genereu un nou directori, perquè es crearan alguns arxius i això els mantindrà fora del vostre material normal. Ara editarem el nostre primer arxiu.

```
$ vi primer-fitxer
```

Quan executes aquesta ordre, s'obre l'arxiu. Si l'arxiu no existeix, el crearà per a nosaltres i després l'obrirà (no cal que executeu *touch arxiu* abans d'editar-lo). Un cop ingressem a *vim*, es veurà alguna cosa similar a la figura 3 (tot i que, depenent del sistema en què us trobeu, es pot veure lleugerament diferent).

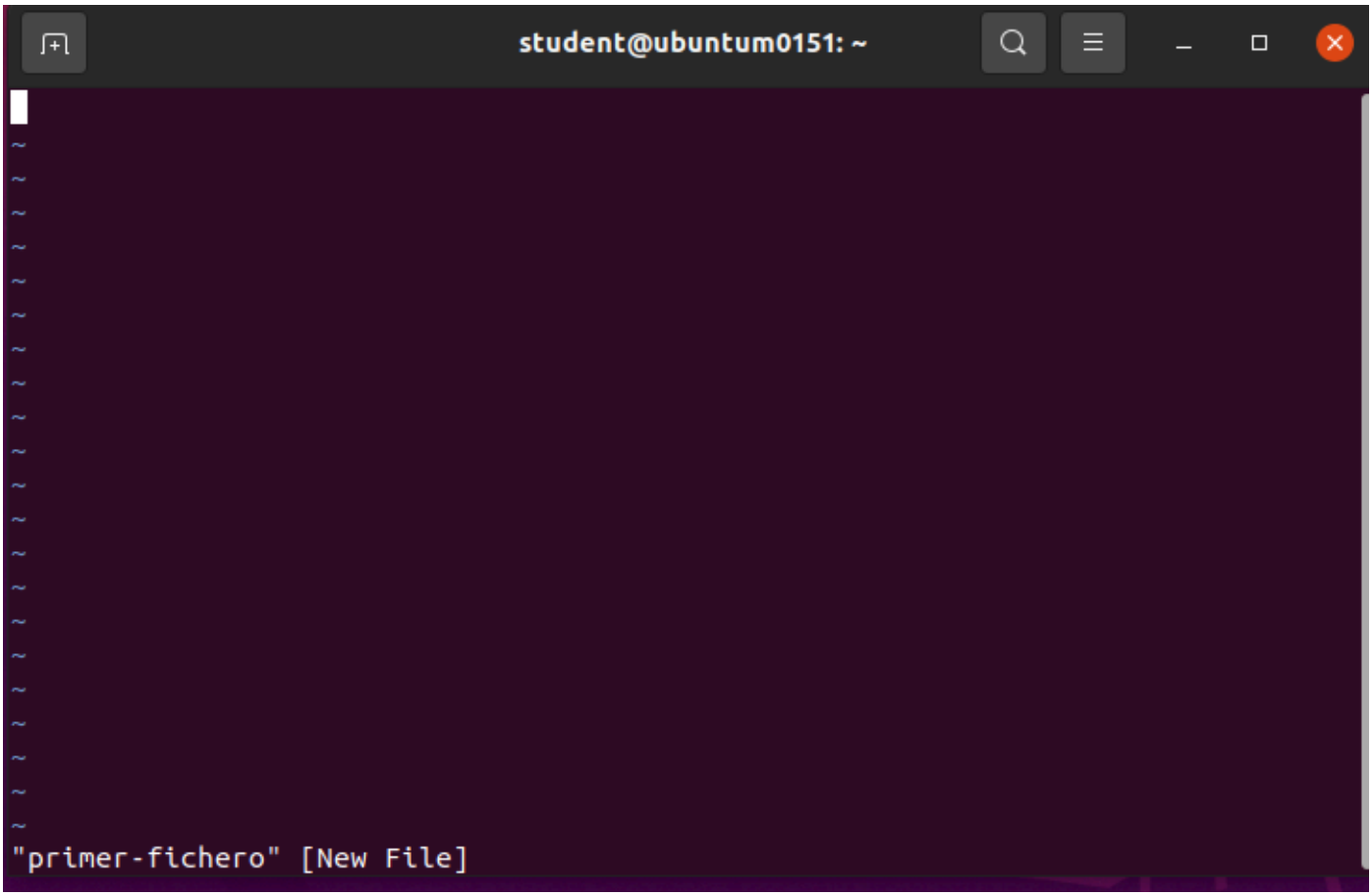

Figura 3. Imatge obtinguda en executar l'ordre *vi* primer-fitxer.

Sempre començarem en mode d'edició, de manera que el primer que farem és passar al mode d'inserció pressionant la lletra *i*. Podeu saber quan esteu en mode d'inserció, perquè ho veureu indicat a la cantonada inferior esquerra. Ara escriviu algunes línies de text i pressioneu *Esc*, cosa que et portarà de tornada al mode d'edició.

Save* i *existing

Hi ha algunes formes de fer-ho. Totes fan essencialment el mateix, així que tria la que prefereixis. En qualsevol cas, assegura't d'estar primer en mode d'edició.

`:ZZ` (Nota: en majúscules). Guardar i sortir.

`:q!` Descartar tots els canvis des de l'últim guardar i sortir.

`:w` Guardar l'arxiu però no sortir.

`:wq` Novament, guardar i sortir.

La majoria de les ordres dins de *vi* s'executen tan bon punt pressionem una seqüència de tecles. Qualsevol ordre que comenci amb dos punts (:) requereix que pressionem <enter> per completar-la. Guardeu i salveu l'arxiu que teniu actualment obert.

L'editor *vi* permet editar arxius. Si volguéssim, també podríem fer-lo servir per veure arxius, però hi ha altres ordres que són una mica més convenients per a aquest propòsit. Provem *cat*, que en realitat significa *concatenar* i té com a objectiu principal unir arxius, però que en la seva forma més bàsica és útil per, simplement, veure arxius, com ja s'ha vist.

Navegant en un fitxer *vi*

Ara tornem a l'arxiu que acabem de crear i afegim-hi més contingut. En el mode d'inserció, podem fer servir les tecles de fletxa per moure el cursor. Afegiu dos paràgrafs més de contingut i després pressioneu *Esc* per tornar al mode d'edició.

A continuació, es mostren algunes de les moltes ordres que podem utilitzar per moure'ns per l'arxiu. Jugueu-hi i observeu com funcionen.

- Tecles de fletxa: mou el cursor per l'arxiu.
- *j*, *k*, *h*, *l*: mou el cursor cap avall, amunt, esquerra i dreta (similar a les tecles de fletxa).
- ^ (accent circumflex): mou el cursor al principi de la línia actual.
- $: mou el cursor al final de la línia actual.
- nG: mou el cursor fins a la *n*-ena línia (per exemple, *5G* es mou a la cinquena línia).
- G: mou el cursor fins a l'última línia.
- w: mou el cursor al principi de la següent paraula.
- nw: mou el cursor *n* paraules cap endavant (per exemple, *2w* el mou dues paraules cap endavant).
- b: mou el cursor al principi de la paraula anterior.
- nb: mou el cursor cap enrere *n*
- { : mou un paràgraf cap enrere, }: mou un paràgraf cap endavant.

Si escriviu `:set nu` en el mode d'edició dins de *vi*, s'habilitaran els números de línia. Descobrireu que és una manera molt més senzilla de treballar amb arxius.

Eliminant contingut

Acabem de veure que, si volem moure'ns dins de *vi*, hi ha moltes opcions disponibles. Diverses d'elles també ens permeten precedir-les amb un número per moure'ns la mateixa quantitat de vegades. Eliminar i desplaçar-se serveixen, de fet, de manera similar: diverses ordres d'eliminació ens permeten incorporar una ordre de moviment per definir el que s'eliminarà. A continuació, es mostren algunes de les moltes formes com podem eliminar contingut dins de *vi*. Jugueu amb elles ara (consulteu també la secció següent sobre com desfer per poder tirar enrere les vostres eliminacions).

- x: elimina un sol caràcter.
- nx: elimina *n* caràcters (per exemple, *5x* elimina cinc caràcters).
- dd: elimina la línia actual.
- dn: *d* seguit d'una ordre de moviment. Elimina fins on l'ordre de moviment et portaria (per exemple, *d5w* significa eliminar cinc paraules).

Desfer

Desfer canvis en *vi* és força fàcil. És el caràcter *u*.

- u: desfà l'última acció (pots continuar pressionant *u* per continuar desfent).
- U (Nota: majúscula): desfà tots els canvis en la línia actual.

Portant-ho més enllà

Ara podem inserir contingut en un arxiu, moure'ns per l'arxiu, eliminar contingut i desfer-lo, després guardar i sortir. Ara podeu fer edicions bàsiques en *vi*. Tanmateix, això és només la introducció del que *vi* pot fer. Hi ha molt més:

- copiar i enganxar
- buscar i reemplaçar
- *buffers*
- marcadors
- rangs
- configuracions

Tot i que no és possible dedicar-hi més temps, es recomana un tutorial per aprendre a utilitzar *vim* o bé accedir directament al seu tutorial interactiu a través de l'ordre *vimtutor* al terminal.

1. Introducció als entorns de treball UNIX

1.9. Accedir al contingut dels fitxers

1.9.3. Edició d'arxius amb l'editor de flux *sed* (*stream editor*)

El nom de l'ordre `sed` prové de *stream editor* ('editor de flux'). Aquí, *stream* es refereix a les dades que es passen mitjançant tubs de *shell*. Per tant, la funcionalitat principal de l'ordre és actuar com un editor de text per a les dades d'entrada de l'entrada estàndard (*stdin*), amb la sortida estàndard (*stdout*) com el destí de sortida. També podem editar l'entrada d'un arxiu i guardar els canvis en el mateix arxiu si és necessari.

La sintaxi bàsica de `sed` és `sed [options] {commands} {input-file}`

La manera de treballar de `sed` és la següent: l'ordre *sed* llegeix la primera línia de l'{arxiu-de-entrada} i executa les {ordres} a la primera línia. Després llegeix la segona línia de l'{arxiu-de-entrada} i executa les {ordres} a la segona línia. L'ordre `sed` repeteix aquest procés fins que arriba al final de l'{arxiu-de-entrada}.

Editeu amb *vi* el fitxer *test.bed* i realitzeu cadascuna de les operacions que es mostren a continuació per entendre la potència de l'ordre *sed:*

```
$ cat test.bed
```

```
chr1 100 200

chr1 300 500

chr2 240 440

chr2 400 600

chr3 0 150
```

Substitució

Substitueix tots els *strings* que coincideixen amb *chr1* per *chr2*

```
$ sed 's/chr1/chr2/' test.bed
```

```
chr2 100 200

chr2 300 500

chr2 240 440

chr2 400 600

chr3 0 150
```

Substitueix tots els *strings* que coincideixen amb *chr1* per *chr2* només si la línia conté 300

```
$sed '/300/s/chr1/chr2/' test.bed
```

```
chr1 100 200

chr2 300 500
```

```
chr2 240 440

chr2 400 600

chr3 0 150
```

Substitueix tots els *strings* que coincideixen amb *chr1* per *chr2* només si la línia no conté 300

```
$sed '/300/! s/chr1/chr2/' test.bed
```

```
chr2 100 200

chr1 300 500

chr2 240 440

chr2 400 600

chr3 0 150
```

Reemplaça tots els *strings* que coincideixen amb *chr* si són els primers caràcters de la línia

```
$ sed 's/^chr//' test.bed
```

```
1 100 200

1 300 500

2 240 440

2 400 600

3 0 150
```

Substitueix la primera ocurrència en cadascuna de les línies

```
$ sed 's/00/55/' test.bed
```

```
chr1 155 200

chr1 355 500

chr2 240 440

chr2 455 600

chr3 0 150
```

Substitueix totes les ocurrències que coincideixin en el patró

```
$ sed 's/00/55/g' test.bed
```

```
chr1 155 255

chr1 355 555

chr2 240 440

chr2 455 655

chr3 0 150
```

Imprimeix la línia on coincideix la substitució

```
$ sed -n 's/00/55/p' test.bed
```

```
chr1 155  200

chr1 355  500

chr2 455  600
```

Es poden realitzar substitucions amb diferents *flags* simultàniament

```
$ sed -n 's/00/55/pg' test.bed
```

```
chr1 155  255

chr1 355  555

chr2 455  655
```

Eliminar

Elimina la segona línia del fitxer

```
$ sed '2 d' test.bed
```

```
chr1 100 200

chr2 240 440

chr2 400 600

chr3 0 150
```

Elimina de la línea 2 a la 4

```
$ sed '2,4 d' test.bed
```

```
chr1 100 200

chr3 0 150
```

Elimina des de l'*string chr1* a la línia 4

```
$ sed '/chr1/, 4 d' test.bed
```

```
chr3 0 150
```

Elimina des de la primera línia que troba amb *chr1* fins a la primera vegada que troba *chr2*

```
$ sed '/chr1/, /chr2/ d' test.bed
```

```
chr2 400 600

chr3 0 150
```

Edició implícita

El fitxer d'entrada i de sortida és el mateix. Feu-lo quan estiguis segur del canvi.

```
$ sed -i 's/00/55/g' test.bed
```

```
$ cat test.bed
```

```
chr1 155 255

chr1 355 555

chr2 240 440

chr2 455 655

chr3 0 150
```

Perquè el fitxer original ha canviat!

Substitucions *regexp*

Les expressions regulars són molt útils, i val la pena dedicar el temps per aprendre els conceptes bàsics. Es poden ampliar coneixements utilitzant eines en línia per construir i provar expressions regulars (per exemple, https://regex101.com/). A continuació enumerem algunes de les importants:

Àncores:

^ restringeix la coincidència a l'inici de la cadena.

$ restringeix la coincidència al final de la cadena.

Metacaràcters i quantificadors

- . coincideix amb qualsevol caràcter, incloent-hi el caràcter de nova línia.
- ? coincideix 0 o 1 vegada.
- * coincideix 0 o més vegades.
- +coincideix 1 o més vegades.
- {m,n} coincideix de *m* a *n* vegades.

- `{m,}` coincideix almenys *m* vegades.
- `{,n}` coincideix fins a *n* (incloent-hi 0 vegades).
- `{n}` coincideix exactament *n*.

Classes de caràcters:

- `[set123]` coincideix amb qualsevol d'aquests caràcters una vegada.
- `[^set123]` coincideix excepte amb qualsevol d'aquests caràcters una vegada.
- `[3-7AM-X]` rang de caràcters des de 3 fins a 7, A, un altre rang des de M fins a X.
- `[[:digit:]]` similar a `[0-9]` `[[:alnum:]_]` semblant a `\w`

Elimina la primera columna: . defineix qualsevol caràcter després de *chr*

```
$ sed 's/^chr.//' test.bed
```

```
        100   200

        300   500

        240   440

        400   600

        0     150
```

Elimina tots els zeros presents al fitxer

```
$ sed 's/0*//g' test.bed
```

```
chr1 1     2

chr1 3     5

chr2 24    44

chr2 4     6

chr3       15
```

Elimina tots els números entre l'1 i el 3

```
$sed 's/[1-3]//g' test.bed
```

```
chr   00    00

chr   00    500

chr   40    440

chr   400   600
```

```
chr  0    50
```

Substitueix l'*string ch* per res

```
$sed 's/[cr]//g' test.bed
```

```
h1   100  200

h1   300  500

h2   240  440

h2   400  600

h3   0    150
```

S'utilitza qualsevol lletra entre la *a* i la *z* per res

```
$sed 's/[a-z]//g' test.bed
```

```
1    100  200

1    300  500

2    240  440

2    400  600

3              0     150
```

Substitueix per la cadena que coincideix amb el patró

```
$ sed 's/^chr[0-9]/[&]/' test.bed
```

```
[chr1] 100 200

[chr1] 300 500

[chr2] 240 440

[chr2] 400 600

[chr3] 0 150
```

Substitueix totes les ocurrències que encaixen en el patró i escriu al final de la línia '+'

```
$ sed 's/00/55/g ; s/$/\+/' test.bed
```

```
chr1 155 255 +

chr1 355 555 +
```

```
chr2 240 440 +

chr2 455 655 +

chr3 0 150 +
```

Substitueix l'ocurrència que coincideix exactament amb 2 zeros

```
$ sed 's/0\{2\}/match/g' test.bed
```

```
chr1 1match    2match

chr1 3match    5match

chr2 240  440

chr2 4match    6match

chr3 0    150
```

Substitueix l'ocurrència que coincideix amb 1 o 2 zeros

```
$ sed 's/0\{1,2\}/match/g' test.bed
```

```
chr1 1match    2match

chr1 3match    5match

chr2 24match   44match

chr2 4match    6match

chr3 match 15match
```

Elimina totes les línies en blanc del fitxer

```
sed '/^$/ d' text.bed or sed '/^#/ d' test.bed
```

1. Introducció als entorns de treball UNIX

1.10. Gestió bàsica de processos

1.10.1 Introducció

Una de les característiques distintives de Gnu/Linux és el seu enfocament a atorgar a l'usuari un ampli control sobre els processos en execució en el sistema. En aquest apartat, ens centrarem a explorar les ordres de gestió de processos que ofereix la *shell* de Gnu/Linux. Això ens permetrà comprendre com funciona el control de processos a Gnu/Linux des d'un terminal i com es poden aprofitar aquestes eines per millorar l'eficiència i productivitat en l'administració de sistemes. A la taula 8 s'enumeren les ordres més habituals.

Taula 8. Ordres per al control de processos.

Ordre	Resultat
`sleep`	Suspèn l'execució actual d'una ordre per un interval de temps
`ps`	*process status*. Imprimeix l'estat dels processos
`fg`	*foreground* o primer pla
`bg`	*background* o segon pla
`jobs`	Imprimeix els treballs actius a la *shell*
`top`	*table of processes*. Imprimeix una vista en temps real dels processos en execució a Linux i també mostra les tasques administrades pel *kernel*. A més, l'ordre proporciona un resum d'informació del sistema que mostra la utilització de recursos, incloent-hi l'ús de CPU i memòria
`kill`	Atura el procés subministrat el PID
`nice`	Permet executar una ordre amb una prioritat menor a la prioritat normal de l'ordre
`renice`	Modifica el valor «nice» d'un o més processos en execució

Font: elaboració pròpia

Des de la línia d'ordres, l'ordre `ps` és la més antiga i comuna per llistar els processos que s'estan executant en el teu sistema. L'ordre `top` proporciona una forma més orientada a la pantalla de llistar els processos i també es pot fer servir per canviar l'estat dels processos.

1. Introducció als entorns de treball UNIX

1.10. Gestió bàsica de processos

1.10.2. Llistar processos amb «ps»

La utilitat més comuna per comprovar els processos que s'estan executant és l'ordre `ps`. Utilitzeu-lo per veure quins programes s'estan executant, els recursos que estan utilitzant i qui els està executant. El següent és un exemple de l'ordre `ps`:

```
$ ps -u
```

```
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

student 2147 0.0 0.7 1836 1020 tty1 S+ 14:50 0:00 -bash

student 2310 0.0 0.7 2592   912 tty1 R+ 18:22 0:00 ps u
```

En aquest exemple, l'opció *u* sol·licita que es mostrin els noms d'usuari i altra informació, com el temps d'inici del procés i l'ús de memòria i CPU per als processos associats amb l'usuari actual. Els processos que es mostren estan associats amb el terminal actual (*tty1*). El primer procés mostra que l'usuari anomenat *student* va obrir una *shell bash* després d'iniciar sessió. El següent procés mostra que *student* ha executat l'ordre `ps -u`. El dispositiu terminal *tty1* s'està utilitzant per a la sessió d'inici de sessió. La columna STAT representa l'estat del procés, amb *R* indicant un procés que s'està executant actualment i *S* representant un procés que està dormint.

Per paginar a través de tots els processos en execució en el vostre sistema Gnu/Linux per a l'usuari actual, agregueu el símbol *pipe* (|) i l'ordre less a l'ordre `ps -ux`:

```
$ ps -ux | less
```

Per paginar a través de tots els processos en execució per a tots els usuaris en el vostre sistema, feu servir l'ordre `ps -aux` de la següent manera:

```
$ ps -aux | less
```

El símbol *pipe* (|) us permet dirigir la sortida d'una ordre perquè sigui l'entrada de l'ordre següent. En aquest exemple, la sortida de l'ordre `ps` (una llista de processos) es dirigeix a l'ordre `less`, que us permet paginar aquesta informació. Feu servir la barra espaiadora per avançar pàgina per pàgina i escriviu *q* per acabar la llista. També podeu fer servir les tecles de fletxa per avançar una línia alhora a través de la sortida.

Consulteu la pàgina de manual de `ps` per obtenir informació sobre altres columnes d'informació que podeu mostrar i ordenar-hi.

1. Introducció als entorns de treball UNIX

1.10. Gestió bàsica de processos

1.10.3. Llistant i canviant processos amb *top*

L'ordre `top` proporciona una forma orientada a la pantalla de mostrar els processos que s'estan executant en el seu sistema. Amb `top`, per defecte, es mostren els processos en funció del temps de CPU que estan consumint actualment. Tanmateix, també es poden ordenar per altres columnes. D'altra banda, si s'identifica un procés problemàtic, també es pot fer servir `top` per matar (acabar completament, en anglès *kill*) o `renice` (en català, *re-prioritzar*) aquest procés. Si desitja poder matar o `renice` processos, s'ha d'executar `top` com a usuari *root*. Si només desitja mostrar processos, i possiblement matar o canviar els seus propis processos, es pot fer com a usuari regular.

La figura 4 mostra un exemple de la finestra `top`. La informació general sobre el sistema apareix a la part superior de la sortida de `top`, seguida d'informació sobre cada procés en execució. A la part superior, es pot veure quant temps ha estat actiu el sistema, quants usuaris estan actualment connectats al sistema i quanta demanda hi ha hagut en el sistema en els últims 1, 5 i 10 minuts. Una altra informació general inclou quants processos (tasques) s'estan executant actualment, quanta CPU s'està utilitzant i quanta memòria RAM i *swap* estan disponibles i s'estan utilitzant. Després de la informació general, hi ha llistats de cada procés, ordenats pel percentatge de la CPU que s'està utilitzant en cada procés. Tota aquesta informació es torna a mostrar cada 5 segons, aquest temps està definit de forma predeterminada.

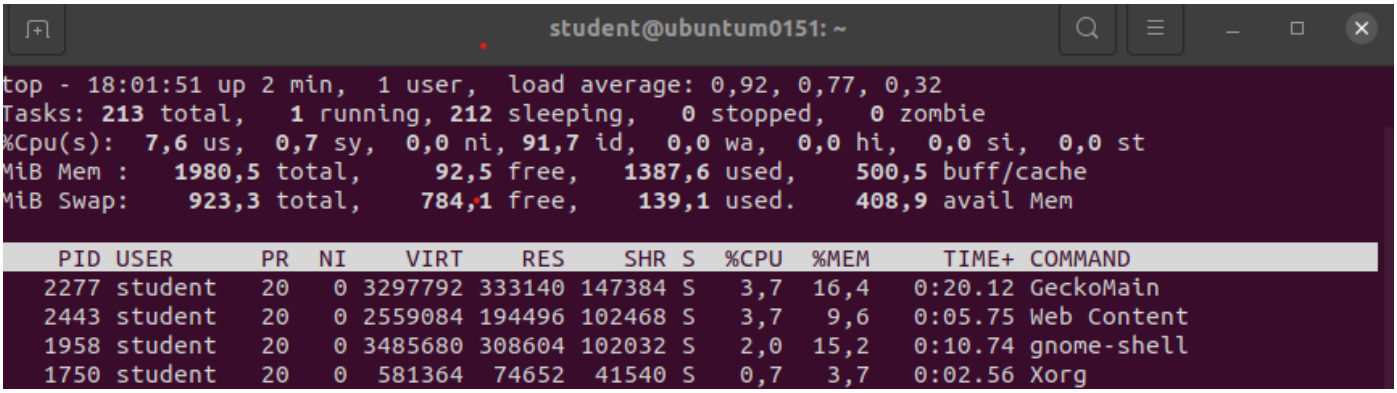

Figura 4. Imatge de la pantalla en executar l'ordre `top`.

La següent llista inclou accions que es poden realitzar quan s'està executant *top* per mostrar informació de diferents formes i modificar processos en execució:

- Pressioneu *h* per veure les opcions d'ajuda, i després pressioni qualsevol tecla per tornar a la pantalla `top`.
- *Pressioneu M* per ordenar per ús de memòria en lloc de CPU, i després pressioneu *P* per tornar a ordenar per CPU.
- Pressioneu el número 1 per alternar entre mostrar l'ús de la CPU de totes les CPU, si té més d'una CPU en el seu sistema.
- *Pressioneu R* per ordenar la seva sortida en ordre invers.
- *Pressioneu u* i afegiu un nom d'usuari per mostrar només els processos d'un usuari en particular.

Una pràctica comuna és utilitzar `top` per trobar processos que estiguin consumint massa memòria o potència de processament i després actuar sobre aquests processos d'alguna manera. Un procés que consumeix massa memòria pot ser matat, o un procés que consumeix massa CPU pot ser `renice` per donar-li menys prioritat als processadors.

- Matar un procés: preneu nota de l'ID de procés del procés que es desitja matar i pressioneu *k*. Escriviu 15 per acabar de manera neta o 9 per matar el procés directament. Des del terminal també es poden matar processos.

```
$ kill 2277
```

```
$ kill -15 2277
```

```
$ kill -SIGKILL 2277
```

Quan el *kernel* de Gnu/Linux intenta decidir quins processos en execució tenen accés a les CPUs del sistema, una de les coses que té en compte és el valor *nice* establert en el procés. Cada procés en execució en el sistema té un valor *entre* -20 i 19. De manera predeterminada, el valor *nice* s'estableix en 0. Aquí hi ha algunes dades sobre els valors *nice*:

- Com més baix sigui el valor *nice*, més accés a les CPUs tindrà el procés.
- L'usuari *root* pot establir el valor *nice* en qualsevol procés en qualsevol valor vàlid, cap amunt o cap avall.
- Un usuari *estàndard* només pot establir el valor *nice* en els propis processos de l'usuari, només poden ser positius i el nou valor de *nice* sempre ha de ser major, no menor, al predeterminat.

Si tornem a executar l'ordre `top`,

- `Renice` un procés: preneu nota de l'ID de procés del procés que es desitja `renice` i pressioneu *r*. Quan aparegui el missatge «PID to renice:», escriviu l'ID del procés que es desitja *renicejar*. Quan es demani «Renice PID to value:», escriviu un número del 0 al 20. Des del terminal es poden *nice/renice* diferents processos (en aquest cas el realitza *root*).

```
# $ nice +5 GekoMain &
```

```
# $ renice -n -5 2243
```

1. Introducció als entorns de treball UNIX

1.10. Gestió bàsica de processos

1.10.4. Gestió de processos en segon pla i primer pla

Si esteu treballant amb Gnu/Linux a través d'una xarxa o des d'un terminal amb una pantalla que només permet entrada de text sense suport gràfic, és possible que només tingueu accés a la *shell*. Si esteu acostumats a treballar en un entorn gràfic en el qual pots tenir diversos programes oberts alhora i canviar entre ells, la interfície de la *shell* pot semblar limitada.

No obstant això, tot i que la *shell bash* no té una interfície gràfica per executar diversos programes alhora, sí que permet moure els programes actius entre el fons i el primer pla. Això et permet tenir diversos processos en execució i seleccionar el que volem utilitzar en aquell moment.

Podeu posar un programa en segon pla de diverses maneres. Una forma és agregar el símbol & al final de la línia d'ordre quan l'executeu per primera vegada o podeu fer servir l'ordre `at` per executar ordres de manera que no estiguin connectades a la *shell*.

Per aturar una ordre en execució i posar-la en segon pla, pressioneu Ctrl+Z. Després d'aturar l'ordre, podeu tornar a executar-la en primer pla amb l'ordre `fg` o iniciar-la en segon pla amb l'ordre `bg`. És important tenir en compte que qualsevol ordre en execució en segon pla pot generar sortida durant les ordres que executem posteriorment des d'aquesta *shell*. Per exemple, si apareix sortida d'una ordre en segon pla durant una sessió de *vi*, simplement pressioneu Ctrl+L per refrescar la pantalla i desfer-vos de la sortida.

Si teniu programes que desitgeu executar mentre treballeu a la *shell*, podeu col·locar els programes en segon pla. Per col·locar un programa en segon pla en el moment en què s'executa el programa, cal escriure un *ampersand* (&) al final de la línia d'ordre, així:

```
$ find /usr > /tmp/fitxer-usuaris &
```

```
[3] 15971
```

Aquest exemple d'ordre troba tots els arxius en el teu sistema Gnu/Linux (a partir d'`usr`), imprimeix aquests noms d'arxiu i els col·loca a l'arxiu /tmp/fitxer-usuaris. L'*ampersand* (&) executa aquesta línia d'ordre en segon pla. Observeu que es mostra el número de treball, [3], i el número d'identificació de procés, 15971, quan es llança l'ordre. Per comprovar quines ordres teniu en execució en segon pla, fa servir l'ordre `jobs`, així:

```
$ jobs
```

```
[1] Stopped (tty output) vi /tmp/unficheroqualsevol

[2] Running find /usr -print > /tmp/allusrfiles &

[3] Running nroff -man /usr/man2/* >/tmp/man2 &

[4]- Running nroff -man /usr/man3/* >/tmp/man3 &

[5]+ Stopped nroff -man /usr/man4/* >/tmp/man4
```

Es poden portar qualsevol de les ordres de la llista de treballs al primer pla. Per fer referència a una tasca en segon pla (per cancel·lar-lo o portar-lo al primer pla), es fa servir un signe de percentatge (%) seguit del número de tasca. Per editar l'arxiu *unfitxerqualsevol* novament, escriviu:

```
$ fg %1
```

Com a resultat, l'ordre *vi* i el terminal que el conté s'obren de nou. Tot el text és com estava quan vas aturar la feina de *vi*. Abans de posar un processador de text, un processador de paraules o un altre programa similar en segon pla, assegureu-vos de guardar l'arxiu. És fàcil oblidar que teniu un programa en segon pla i perdreu les vostres dades si tanqueu la sessió o reinicieu l'ordinador.

Si una ordre s'atura, podem fer que s'executi de nou en segon pla fent servir l'ordre *bg*. Per exemple, preneu la tasca número 5 de la llista de tasques de l'exemple anterior. Escriviu el següent:

```
$ bg %5
```

Després d'això, la tasca s'executarà en segon pla. La seva entrada a la llista de tasques apareixerà així:

```
[5] Running nroff -man man4/* >/tmp/man4 &
```

1. Introducció als entorns de treball UNIX

1.11. Buscar, ordenar i associar fitxers

1.11.1. Introducció

Quan es treballa amb dades bioinformàtiques, és comú trobar fitxers de text organitzats en format tabular, la qual cosa implica que la informació es distribueix en una matriu de files i columnes delimitades per espais o caràcters tabuladors. Cada línia representa un registre, mentre que cada columna conté valors específics dels atributs que el caracteritzen. Aquesta estructura de camps facilita la realització de càlculs sistemàtics sobre el contingut del fitxer en qüestió.

Generalment, en treballar amb fitxers tabulats en l'anàlisi bioinformàtica, s'adopta com a unitat elemental de treball la línia o registre. Sota aquest paradigma, es poden realitzar diversos tipus d'operacions, com buscar i separar del fitxer les línies que contenen un patró de text determinat, alterar l'ordre de les línies segons els valors d'algun dels atributs o filtrar registres duplicats. En certes circumstàncies, fins i tot és possible identificar aquells registres de dos fitxers de text diferents que posseeixen el mateix valor per a un determinat atribut.

Aquestes operacions aplicades sobre els fitxers d'anotacions són útils en l'anàlisi bioinformàtica per dur a terme fàcilment el recompte de diferents característiques biològiques, com els gens codificats a l'interior dels genomes. A la taula 9 es descriuen les ordres més útils per buscar i ordenar fitxers, així com associar-se entre ells.

Taula 9. Descripció d'ordres.

Ordre	Descripció
`grep`	Busca línies que coincideixin amb una expressió regular
`cut`	`cut` és un programa útil si el contingut està separat en camps (columnes) i només es volen obtenir certs camps
`sort`	`sort` ordenarà la seva entrada, de manera simple i senzilla. Per defecte, ordenarà alfabèticament, però hi ha moltes opcions disponibles per modificar el mecanisme d'ordenació. Assegura't de revisar la pàgina de manual per veure tot el que pot fer
`uniq`	Elimina les línies duplicades successives (usar `sort` abans d'utilitzar `uniq`)
`join`	Permet unir dos fitxers de text en un usant una columna com a clau comuna

Font: elaboració pròpia.

1. Introducció als entorns de treball UNIX

1.11. Buscar, ordenar i associar fitxers

1.11.2. «*grep*»

A la Wikipedia pots llegir que `grep` és una utilitat de línia d'ordre per buscar conjunts de dades de text pla a la recerca de línies que coincideixin amb una expressió regular. El seu nom prové de l'ordre `ed g/re/p` (***g**lobally search a **r**egular **e**xpression and **p**rint*, en català *recerca global d'una expressió regular i imprimir*), que té el mateix efecte.

L'ordre `grep` té moltes i variades característiques, tant és així que hi ha llibres específics al respecte. L'ús més comú és filtrar línies d'entrada fent servir expressions regulars (en anglès, *regexp*).

Les opcions de l'ordre `grep` comunament utilitzades es mostren a continuació. Els exemples es discutiran en seccions posteriors.

- `-i` ignora la distinció entre majúscules i minúscules en fer coincidències.
- `-v` imprimeix només les línies que no coincideixen.
- `-n` agrega un prefix de números de línia a les línies de sortida.
- `-c` mostra només el recompte de línies de sortida.
- `-l` imprimeix només els noms d'arxiu que coincideixen amb l'expressió donada.
- `-L` imprimeix els noms d'arxiu que no coincideixen amb el patró.
- `-w` fa coincidir el patró només amb paraules completes.
- `-x` fa coincidir el patró només amb línies completes.
- `-F` interpreta el patró com una cadena fixa (és a dir, no com una expressió regular).
- `-o` imprimeix només les parts coincidents.
- `-A N` imprimeix la línia coincident i N línies després de la línia coincident.
- `-B N` imprimeix la línia coincident i N línies abans de la línia coincident
- `-C N` imprimeix la línia coincident i N línies abans i després de la línia coincident.
- `-m N` imprimeix un màxim de N línies coincidents.
- `-q` sense sortida estàndard, surt immediatament si es troba una coincidència, útil en *scripts*.
- `-s` suprimeix els missatges d'error, útil en *scripts*.
- `-r` busca tots els arxius a les carpetes d'entrada especificades (per defecte els busca en el directori actual).
- `-R` com `-r`, però també segueix els enllaços simbòlics.
- `--color=auto` ressalta les porcions coincidents, els noms d'arxiu, els números de línia, etc. usant colors.

Recerca literal

Els següents exemples també serien adequats amb l'opció `-F` ja que no utilitzen expressions regulars. L'ordre `grep` està prou ben dissenyada com per fer el que és correcte en aquests casos.

Imprimeix línies que continguin l'`string 'an'`. L'opció \n genera una nova línia

```
$ printf 'poma\nplatan\nmango\nfigura\ntango\n' | grep 'an'
```

```
platan

mango

tango
```

Imprimeix coincidències sense distinció entre majúscules i minúscules

```
$ printf 'mussol\nsargantana\muSsoLina\ntres mussols\n' | grep -i
'mussol'
```

```
Mussol

muSsoLina

tres mussols
```

Imprimeix coincidències de paraules completes

```
$ printf 'parell basic\nparitat\n3-parell' | grep -w 'parell'
```

```
parell basic

3-parell
```

Imprimeix coincidències de línies buides i les compta

```
$ printf 'cel\n\nterra\n\n\n\ninfern\n' | grep -cx ''
```

```
4
```

Imprimeix coincidències en una línia i les següents dues línies (A, after)

```
$ printf 'xarxa\ncactus\nsuculenta\npetunia\nesqueix' | grep -A2 'cactus'
```

```
cactus

suculenta

petunia
```

Expressions regulars

De forma predeterminada, *grep* tracta el patró de recerca com una expressió regular bàsica (en anglès, *Basic Regular Expression*: BRE).

- `-G` es pot utilitzar per especificar explícitament que es necessita BRE.
- `-E` habilitarà expressions regulares esteses (en anglès, *Extended Regular Expression*: ERO). **A GNU *grep*, BRE i ERO només difereixen en com s'especifiquen els metacaràcters; no hi ha diferència en les característiques.**
- `-F` farà que els patrons de recerca es tractin literalment.

- `-P` habilitarà expressions regulares compatibles amb Perl, si està disponible (PCRE, per les seves sigles en anglès).

Les següents línies són les referències quan es volen utilitzar expressions regulars esteses. N'hi ha de diverses classes:

Àncores

- ^ restringeix la coincidència a l'inici de la cadena.
- $ restringeix la coincidència al final de la cadena.
- < restringeix la coincidència a l'inici de paraula.
- > restringeix la coincidència al final de paraula.
- \b restringeix la coincidència a l'inici/final de paraules.
- \B coincideix on \b no coincideix.

Metacaràcters i quantificadors

- . coincideix amb qualsevol caràcter, incloent-hi el caràcter de nova línia.
- ¿ coincideix 0 o 1 vegada.
- * coincideix 0 o més vegades.
- + coincideix 1 o més vegades.
- {m,n} coincideix de *m* a *n*
- {m,} coincideix almenys *m*
- {,n} coincideix fins *a n* (incloent-hi 0 vegades).
- {n} coincideix exactament *n*

Classes de caràcters

- [set123] coincideix amb qualsevol d'aquests caràcters una vegada.
- [^set123] coincideix, excepte amb qualsevol d'aquests caràcters, una vegada.
- [3-7AM-X] rang de caràcters des de 3 fins a *7*, *A*, un altre rang des de *M* fins a *X*.
- \w similar a [a-zA-Z0-9_] per coincidir amb caràcters de paraules.
- \s similar a [\t\n\r\f\v] per coincidir amb caràcters d'espai en blanc.
- \W coincideix amb caràcters que no són de paraules.
- \S coincideix amb caràcters que no són d'espai en blanc.
- [[:d igit:]] similar a [0-9] [[:alnum:]_] similar a \w

Es recomana veure el manual de l'ordre `grep` per ser conscient de la versatilitat d'aquesta ordre.

Alternança i agrupació

- Pat1|pat2|pat3 coincideix amb *pat1* o *pat2* o *pat3*.
- () agrupa patrons, a(b|c)d és el mateix que abd|acd. També serveix com a grup de captura.
- \N referència cap enrere, proporciona la porció coincident de l'N-è grup de captura.

- \1 referència cap enrere al primer grup de captura.
- \2 referència cap enrere al segon grup de captura i així successivament fins a \9.

En el manual de `grep` se citen les diferències entre BRE i ERE. En les expressions regulars bàsiques els metacaràcters ?, +, {, |, (i) perden el seu significat especial; en el seu lloc, s'utilitzen les versions amb barra invertida \?,\ +, \{,\ |, \(*i* \).

Imprimeix la coincidència quan el final de la línia acaba amb *–ar*

```
$ printf 'pedalejar\ncorrer\nescriure\namar' | grep 'ar$'
```

```
pedalejar

amar
```

Imprimeix les paraules que comencin amb `'par'`, acaben en `'t'`, i poden tenir `'en'` o `'ro'` al mig, sense importar si estan en majúscules o minúscules.

```
$ echo 'par apartment PARROT parent' | grep -ioE 'par(en|ro)?t'
```

```
part

PARROT

Parent
```

Imprimeix coincidència quan hi ha text acotat

```
$ echo 'Disfruto practicant exercicis de "bash" i "R"' | grep -oE '"
[^"]+"'
```

```
"bash"

"R"
```

Línies de 8 caràcters que tenen les mateixes 3 lletres minúscules al principi i al final

```
$ grep -xE '([a-z]{3})..\1' /usr/share/dict/words
```

```
mesdames

respires

restores

testates
```

1. Introducció als entorns de treball UNIX

1.11. Buscar, ordenar i associar fitxers

1.11.3. «*cut*»

Si l'arxiu està organitzat en camps, com en el cas de la taula que estem usant, podem seleccionar un camp específic utilitzant l'ordre `cut`. Per refinar la nostra recerca, podem fer servir `cut` per extreure només el nom de transcrits, com en el següent exemple

```
$ cut -f 2 hg38_RefSeq.txt | head -7
```

```
name

NM_001276352.2

NM_001276351.2

NR_075077.2

XM_011541469.1

XM_011541467.1

XM_017001276.1
```

Amb el paràmetre *-f* li indiquem la llista de camps (*fields*) que volem seleccionar.

Per indicar els camps que volem seleccionar:

- `N`: el camp *N* (per exemple, cut -f 3 file1).
- `N-`: des del camp *N* fins al final (per exemple, cut -f 3- file1).
- `N-M`: des del camp *N* al *M* (per exemple, cut -f 3-6 file1).
- `-M`: des del primer al M (per exemple, cut -f -3 file1).
- `N,M`: els camps indicats (per exemple, cut -f 3,6,8 file1).

Així podríem seleccionar els camps del 3 al 5 i del 8 al 10:

```
$ cut -f 3-5,8-10 hg38_RefSeq.txt
```

L'ordre `cut` assumeix que els camps al fitxer estan dividits per tabuladors. Però podríem indicar-li que els camps estan dividits d'una altra manera, per exemple, per comes:

```
$ cut -d ',' fitxer_separat_per_comes.txt
```

1. Introducció als entorns de treball UNIX

1.11. Buscar, ordenar i associar fitxers

1.11.4. «*sort*»

Com el seu nom indica, en anglès, aquesta ordre s'utilitza per ordenar el contingut dels arxius d'entrada. Ordre alfabètic i ordre numèric? Possible. Què tal ordenar una columna específica? Possible. Ordre de classificació múltiple prioritari? Possible. ¿Aleatori? ¿Únic? Moltes característiques són compatibles amb aquesta ordre tan poderosa.

Es mostren a continuació les opcions d'ús comú. Els exemples es discutiran en seccions posteriors.

- `-n` ordenar numèricament.
- `-g` ordenació numèrica general.
- `-V` ordenar per versió (conscient dels números dins del text).
- `-h` ordenar números llegibles per a humans (per exemplo: 4K, 3M, 12G, etc.).
- `-k` ordenar mitjançant clau (ordenació de columna). Similar a -f de l'ordre *cut*.
- `-t` separador de camp d'un sol byte de caràcter (el valor predeterminat és la transició de no espai en blanc a espai en blanc).
- `-u` ordenar de forma única.
- `-R` ordenar aleatòriament.
- `-r` invertir la sortida d'ordenació.
- `-o` redirigir el resultat ordenat a l'arxiu especificat.

De forma predeterminada, `sort` ordena l'entrada lexicogràficament en ordre ascendent. Pot utilitzar l'opció -r per invertir els resultats.

Ordre per defecte

```
$ printf 'banana\ncirera\nmaduixa' | sort
```

```
banana

cirera

maduixa
```

Ordena i imprimeix en ordre invers

```
$ printf 'hotel\nresidencia\nesperanza' | sort -r
```

```
residencia

hotel

esperanza
```

Ordena numèricament i imprimeix

```
$ printf '20\n2\n3' | sort -n
```

```
2

3

20
```

Ordena els números a la manera humana

```
$ sort -hr fitxer.txt
```

```
1.4G    genomica

316M    proteomica

746K    wdl.log

104K    gromacs.log

20K     sample.txt
```

Ordena tenint en compte la versió

```
$ sort -V temps.txt
```

```
3m20.058s

3m42.833s

4m3.083s

4m11.130s

5m35.363s
```

Genera el següent fitxer

```
$ cat model.txt
```

```
GWAS    50

NGS     5

RNA-seq    2

ChIP-Seq    25

WES   10
```

Ordena tenint en compte els números de la segona columna

```
$ sort -k2,2n model.txt
```

```
RNA-seq   2

NGS    5

WES  10

ChIP-Seq   25

GWAS   50
```

Per exemple, podem ordenar els transcriptors de la manera següent:

```
$ cat hg38_RefSeq | cut -f 2,4| sort
```

En l'exemple anterior, en utilitzar l'ordre `sort` notem que la llista resultant contenia transcrits repetits. Per eliminar línies duplicades consecutives, podem utilitzar l'ordre `uniq`. És important recordar que, per a una eliminació completa de duplicats, cal ordenar l'arxiu amb `sort` abans d'utilitzar `uniq`:

```
$ cat hg38_RefSeq | cut -f 2,4| sort -2rn | uniq
```

1. Introducció als entorns de treball UNIX

1.11. Buscar, ordenar i associar fitxers

1.11.5. «*uniq*»

Aquesta ordre t'ajuda a identificar i eliminar duplicats. S'ha d'utilitzar amb entrades ordenades, ja que la comparació es realitza només entre línies adjacents, la qual cosa significa que primer cal utilitzar l'ordre `sort` abans que l'ordre `uniq`. Es mostren a continuació les opcions d'ús comú.

- `-u` mostra només les entrades úniques.
- `-d` només les entrades duplicades.
- `-D` mostra totes les còpies de duplicats.
- `-c` prefix de comptatge.
- `-i` ignora majúscules i minúscules en determinar duplicats.
- `-f` omet els primers *N* camps la separació de camp; es basa només en un o més caràcters d'espai/tabulació.
- `-s` omet els primers *N* caràcters.
- `-w` restringeix la comparació als primers *N* caràcters.

De forma predeterminada, `uniq` retiene sólo una copia de las líneas duplicadas.

```
$ cat sistemes.txt
```

```
GWAS

RNA-Seq

WES

GWAS

ChIP-Seq

RNA-Seq

NGS

RNA-Seq
```

Ordena i elimina els duplicats

```
$ sort sistemes.txt | uniq
```

```
ChIP-Seq

GWAS

NGS
```

```
RNA-Seq

WES
```

Ordena i imprimeix únicament les ocurrències no repetides

```
$ sort sistemes.txt | uniq -u
```

```
ChIP-Seq

NGS

WES
```

Ordena i imprimeix únicament les ocurrències repetides

```
$ sort sistemes.txt | uniq -d
```

```
GWAS

RNA-Seq
```

Ordena per número d'ocurrència

```
$ sort sistemes.txt | uniq -c | sort -nr
```

```
      3 RNA-Seq

      2 GWAS

      1 WES

      1 NGS

      1 ChIP-Seq
```

Si continuem amb el fitxer hg38_RefSeq per eliminar línies duplicades consecutives, s'utilitza l'ordre `uniq`. És important recordar que, per a una eliminació completa de duplicats, cal ordenar l'arxiu amb `sort` abans d'utilitzar `uniq`:

```
$ cat hg38_RefSeq | cut -f 2,4| sort -2rn | uniq
```

1. Introducció als entorns de treball UNIX

1.11. Buscar, ordenar i associar fitxers

1.11.6. «*join*»

L'ordre `join` permet unir dos fitxers de text en un fent servir una columna com a clau comuna. Per defecte, `join` assumeix que el separador de camps és l'espai. L'ordre `join` és semblant a l'ordre `paste` en què la columna comuna que serveix com a enllaç entre ambdues taules no queda duplicada i no requereix que un element estigui als dos arxius. D'altra banda, el que sí que requereix `join` és que ambdós arxius estiguin ordenats per la columna que es vol usar com a clau. Imaginem que tenim els dos fitxers següents:

```
$ cat file1.txt
```

```
num id atribut

1 CDKL3 chr5

2 CLN8 chr8

5 SOCS2 chr4
```

```
$ cat file2.txt
```

```
num id atribut

1 AGRN +

3 CDKL3 +

5 CLN8 -

9 FCHO +
```

L'ordre `join` ens permet unir aquestes dues taules en una sola utilitzant el camp `num` (la primera columna de cadascun dels fitxers) com la clau d'unió:

```
$ join file1.txt file2.txt
```

```
num id atributo id atributo

1 CDKL3 chr5 AGRN +

5 SOCS2 chr4 CLN8 -
```

Per defecte, `join` assumeix que la clau d'unió és la primera columna, però això es pot modificar:

```
$ join -1 2 -2 2 file1.txt file2.txt
```

```
id num atributo num atributo

CDKL3 1 chr5 3 +
```

```
CLN8 2 chr8 5 -
```

1. Introducció als entorns de treball UNIX

1.12. Combinació d'ordres

Fins ara heu vist nombrosos exemples d'ordres executades individualment al terminal. No obstant això, l'intèrpret d'ordres té un gran potencial que permet implementar protocols de treball més sofisticats de manera relativament senzilla. Un cas paradigmàtic és la canalització dels resultats de les ordres. En general, la majoria de les ordres executades a Gnu/Linux generen més informació de la que pot aparèixer en pantalla físicament. Per tant, és preferible emmagatzemar els resultats dins d'un arxiu per a la seva posterior anàlisi i visualització. També és possible que ens interessi convertir el flux de dades generat per un procés emissor d'informació a l'entrada d'un altre, sense necessitat de generar arxius intermedis. L'intèrpret d'ordres proporciona una sèrie de funcionalitats per implementar totes aquestes operacions relacionades amb la comunicació i l'emmagatzematge de resultats. A la taula 10 es mostren les funcionalitats més utilitzades per combinar ordres.

Taula 10. Combinació d'ordres.

Ordre	Descripció
procés > fitxer	Redirecció de sortida
procés >> fitxer	Redirecció de sortida sense sobresecriure
procés 2> fitxer	Redirecció del canal error
procés < fitxer	Redirecció de l'entrada
procés 1 | procés 2	Comunicació de dos processos

Font: elaboració pròpia.

El procés d'emmagatzematge de dades generades per un procés en un arxiu es coneix com *a redirecció*. Un procés té permís d'escriptura en dos canals del terminal: sortida i error. Les dades generades normalment pel procés es transmeten a través del canal de sortida, mentre que els errors d'execució es comuniquen mitjançant el canal d'error. De manera predeterminada, ambdós canals es redirigeixen a la pantalla (terminal). Si l'usuari vol redirigir una d'aquestes dues sortides, ho ha d'indicar explícitament al final de l'ordre. El símbol > indica la redirecció del canal de sortida, mentre que per redirigir el canal d'error s'ha de fer servir la construcció 2>. En ambdós casos, les dades s'emmagatzemen en un arxiu que es pot consultar en qualsevol moment sense necessitat de tornar a executar l'ordre.

Cada ordre té associada una entrada estàndard *stdin* (0) (per defecte, el teclat), una sortida estàndard *stdout* (1) (per defecte, la consola) i una sortida d'errors estàndard *stderr* (2) (per defecte, també la consola). Per exemple, l'ordre `cat`, si no rep arguments, llegeix del teclat per l'entrada estàndard i el passa a la sortida estàndard.

```
$ cat
```

```
Primera seqüència per estudiar

Segona seqüència per estudiar

^D
```

El final de l'*stream* l'hem assignat des del teclat amb la combinació de tecles Ctrl+D.

Es pot canviar l'entrada estàndard de cat perquè llegeixi d'un fitxer. També serviria per a la majoria de les ordres: `cat, grep, cut, sed`...

```
$ cat < fasta.fa
```

```
Name of the system

```

```
Nucleotides

...
```

L'operador de redirecció de sortida > permet canviar la sortida estàndard d'una ordre

```
$ cal > calendari
```

Aquesta ordre envia el calendari actual al fitxer *calendari*. En aquest cas, el *stdout* de l'ordre `cal` es redirigeix a un fitxer anomenat *calendari* que contindrà el que es veuria a la pantalla en executar `cal`.

Per enviar la sortida *stderr* d'un programa a un fitxer, s'executa

```
$ grep -ioE 'par(en|ro)?t' dades01 2> error.txt.
```

Per enviar la *stdout* a la *stderr*, escrivim en pantalla

```
$ grep -ioE 'par(en|ro)?t' dades01 1>&2
```

I a la inversa, simplement intercanviant l'1 per 2, s'executa

```
$ grep -ioE 'par(en|ro)?t' dades01 2>&1.
```

Si es vol que l'execució d'una ordre no generi activitat per pantalla, el que s'anomena *execució silenciosa*, només hem de redirigir totes les seves sortides a /dev/null. Per exemple, pensant a utilitzar l'ordre `find` (en català, *buscar*), que volem utilitzar perquè esborri tots els arxius acabats en *.mov* del sistema:

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

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

Els *pipes* permeten utilitzar de forma simple tant la sortida d'una ordre com l'entrada d'una altra, per exemple,

```
$ ls -l | sed -e "s/[aeio]/u/g"
```

on s'executa l'ordre `ls` i la seva sortida, en comptes d'imprimir-se a la pantalla, s'envia (per un tub o *pipe*) al programa *sed*, que imprimeix la seva sortida corresponent. Un altre exemple: per buscar en el fitxer */etc/passwd* totes les línies que acabin amb la paraula *false* podríem fer

```
$ cat /etc/passwd | grep false$
```

on, primer, s'executa l'ordre `cat`, i la seva sortida es passa per `grep`. El símbol `$` significa 'final de la paraula', per la qual cosa s'estan identificant amb `grep` totes les línies que acabis amb `false`.

De vegades, hi ha certs canvis en la sintaxi d'algunes ordres quan s'utilitzen els tubs en transmetre informació. Atès que no existeixen fitxers auxiliars creats al llarg del *pipeline*, s'ha d'introduir el caràcter «–» per denotar en aquest cas que la sortida del procés anterior, a través del tub, esdevé el primer argument de la següent execució. Per a especial atenció en la definició del primer arxiu a utilitzar per l'ordre `join`.

```
$ cat fitxer1 | sort -k2r | join - fitxer2
```

1. Introducció als entorns de treball UNIX

1.13. El llenguatge de processament d'arxius GAWK

1.13.1. Introducció

AWK és un llenguatge de programació usat per processar dades de text. El nom AWK prové de les inicials dels cognoms dels seus autors: Alfred Aho, Peter Weinberger i Brian Kernighan. I l'ordre awk és el programa de Gnu/Linux que interpreta el llenguatge de programació AWK. AWK va ser creat per reemplaçar els algoritmes escrits en C per a l'anàlisi de text. Va guanyar popularitat ràpidament a Gnu/Linux i es considera una de les utilitats necessàries del sistema operatiu. L'ordre awk va ser portada a GNU el 1986, i es va anomenar *gawk*. Actualment, Arnold Robins és el principal mantenidor del codi i la documentació de *gawk*. En sistemes moderns de Gnu/Linux, la crida a l'intèrpret d'ordres awk està lligada a gawk.

Fins ara s'han vist ordres per accedir al contingut de fitxers de text organitzats en files i columnes. De vegades, cal accedir a camps específics o realitzar càlculs en alguns registres. GAWK és un llenguatge de programació que permet realitzar aquestes operacions en fitxers de text. GAWK processa els registres i genera una nova línia de resultats. Les variables predefinides a GAWK permeten accedir selectivament a la informació.

1. Introducció als entorns de treball UNIX

1.13. El llenguatge de processament d'arxius GAWK

1.13.2. Conceptes fonamentals

L'ordre gawk és una eina capaç d'executar un bloc de codi sobre atributs individuals de cada registre emmagatzemat en les línies d'un arxiu de text. En processar les dades, és possible generar una nova línia de resultats, que es pot mostrar per pantalla o guardar en un nou arxiu de text.

L'ordre gawk recorre el contingut de l'arxiu de text línia per línia, separant automàticament els diferents components d'aquestes. L'usuari té accés a variables predefinides que proporcionen informació selectiva (vegeu la taula 11) i pot accedir a cada columna d'una línia a través de la seva posició.

Taula 11. Variables especials d'«awk».

Variable	Descripció
$0	Conté el contingut del registre d'entrada
$1	Primer camp
$2	Segon camp, i així successivament
NF	*Number of fields*. Nombre de camps (nombre de columnes)
FS	Separador de camp d'entrada
OFS	Separador de camp de sortida
NR	*Number of records*. Nombre de registres (nombre de línies)
RS	Separador de registre d'entrada
ORS	Separador de registre de sortida
Filename	Nom de l'arxiu d'entrada actualment en processament.

Font: elaboració pròpia.

Important: per defecte, el separador de camps és l'espai en blanc, i el separador de registres és el salt de línia.

En general, a l'intèrpret d'ordres gawk se li subministren dos tipus de dades:

- un fitxer d'ordres o programa,
- un o més arxius d'entrada.

Un fitxer d'ordres (que pot ser un fitxer com a tal, o pot presentar-se en invocar gawk des de la línia d'ordres) conté una sèrie de sentències que li indiquen a gawk com processar el fitxer d'entrada. És a dir, conté el programa escrit en sintaxi **GAWK**.

```
$ gawk 'programa_GAWK' arxiu1 arxiu2 …
```

```
$ gawk -f 'arxiu_amb_codi_GAWK' arxiu1 arxiu2
```

A continuació, en tenim diversos exemples. Primer es genera un arxiu *in situ* amb seqüències en format FASTA i amb línies en blanc entre elles.

```
$ echo -e  ">seq01\nACCTAT\n\n>seq02\nCACCGA\n\n>seq03\nAAAACAGAG\n\n" >
seqüencia.txt
```

Visualitzem el fitxer comptant les línies amb l'opció *-n*

```
$ cat -n seqüencia.txt
```

```
1     >seq01
2     ACCTAT
3
4     >seq02
5     CACCGA
6
7     >seq03
8     AAAACAGAG
9
10
```

S'utilitza gawk per imprimir només les línies que continguin almenys un camp **no** buit

```
$ gawk 'NF > 0' seqüencia.txt | nl
```

```
1    >seq01
2    ACCTAT
3    >seq02
4    CACCGA
5    >seq03
6    AAAACAGAG
```

Imprimeix les primeres 7 línies

```
$ gawk 'NR <= 7' seqüencia.txt | cat -n
```

```
1    >seq01
2    ACCTAT
3
4    >seq02
5    CACCGA
```

```
6

7    >seq03
```

Imprimeix les dues primeres línies i a partir de la sisena; a més, elimina els registres buits. En aquest cas l'ús dels parèntesis aconsegueix que es compleixin les dues condicions

```
$ gawk 'NF > 0 && (NR <= 2 || NR >= 6)' seqüencia.txt
```

```
>seq01

ACCTAT

>seq03

AAAACAGAG
```

En el camp de la bioinformàtica, el fitxer d'entrada està normalment estructurat amb un format taula, per defecte amb camps separats per espais o tabuladors (taules). Es mostren exemples amb l'ordre gawk de les possibilitats d'acció d'aquesta ordre.

Còpia en el teu terminal el fitxer mostrat a continuació i anomena'l *ensamble.txt*

```
#assembly_accession  organism_name   seq_rel_date    asm_name   submitter

GCA_000004195.4 Drosophila melanogaster    2021-07-01 dm6  NCBI

GCA_011586765.1 Arabidopsis thaliana 2022-04-11 Araport11  NCBI

GCA_009859395.1 Mus musculus    2022-02-08 GRCm39    NCBI

GCA_004115215.2 Caenorhabditis elegans     2022-03-16 WBcel235   NCBI

GCA_016590495.1 Rattus norvegicus    2022-02-25 Rnor_6.0   NCBI

GCA_009722195.1 Xenopus tropicalis   2022-01-24 Xenopus_tropicalis_v9.1
NCBI

GCA_017527675.1 Phaeodactylum tricornutum  2022-03-24 Phatr3.0   NCBI

GCA_011586775.1 Xenopus tropicalis   2022-04-11 Xenbase_v9.2    NCBI
```

```
$ head -3 ensamble.txt
```

```
GCA_000004195.4 Drosophila melanogaster    2021-07-01 dm6  NCBI
GCA_011586765.1 Arabidopsis thaliana 2022-04-11 Araport11  NCBI
GCA_009859395.1 Mus musculus    2022-02-08 GRCm39    NCBI
```

Imprimeix la primera, la segona, la quarta i l'última columna del fitxer

```
$ gawk 'BEGIN{FS="\t"} {print $1, $2, $4, $NF}' ensamble.txt
```

```
GCA_000004195.4 Drosophila melanogaster dm6 NCBI
```

```
GCA_011586765.1 Arabidopsis thaliana Araport11  NCBI

GCA_009859395.1 Mus musculus GRCm39 NCBI

GCA_004115215.2 Caenorhabditis elegans     WBcel235 NCBI

GCA_016590495.1 Rattus norvegicus Rnor_6.0 NCBI

GCA_009722195.1 Xenopus tropicalis Xenopus_tropicalis_v9.1 NCBI

GCA_017527675.1 Phaeodactylum tricornutum Phatr3.0 NCBI

GCA_011586775.1 Xenopus tropicalis Xenbase_v9.2 NCBI
```

Si s'indica que el **separador de camp de la sortida** OFS sigui també un tabulador, l'estructura final de l'arxiu s'assemblarà al d'entrada

```
$ gawk 'BEGIN{FS="\t"; OFS=FS} {print $1, $2, $4, $NF}' ensamble.txt
```

```
GCA_000004195.4 Drosophila melanogaster    dm6  NCBI

GCA_011586765.1 Arabidopsis thaliana Araport11  NCBI

GCA_009859395.1 Mus musculus          GRCm39     NCBI

GCA_004115215.2 Caenorhabditis elegans     WBcel235   NCBI

GCA_016590495.1 Rattus norvegicus    Rnor_6.0   NCBI

GCA_009722195.1 Xenopus tropicalis   Xenopus_tropicalis_v9.1    NCBI

GCA_017527675.1 Phaeodactylum tricornutum  Phatr3.0   NCBI

GCA_011586775.1 Xenopus tropicalis   Xenbase_v9.2    NCBI
```

Es realitza una recerca literal en aquest fitxer

```
$ gawk 'BEGIN{FS="\t"; OFS=FS} /tropicalis/ {print $1, $2, $4, $NF}'
ensamble.txt
```

```
GCA_009722195.1 Xenopus tropicalis   Xenopus_tropicalis_v9.1    NCBI

GCA_011586775.1 Xenopus tropicalis   Xenbase_v9.2    NCBI
```

Es realitza una recerca literal d'absència

```
$ gawk 'BEGIN{FS="\t"; OFS=FS} '\!/e/' {print $1, $2, $4, $NF}'
ensamble.txt
```

```
GCA_011586765.1 Arabidopsis thaliana Araport11  NCBI
```

```
GCA_009859395.1 Mus musculus          GRCm39      NCBI
```

Si busques dominar gawk, és crucial que aprenguis a manejar correctament les variables NR, FS i OFS. Els exemples anteriors et donaran una comprensió profunda del seu comportament per defecte, així com de com manipular-les per modelar adequadament l'estructura dels registres de dades. Per tant, et recomanem que els estudiïs amb deteniment.

1. Introducció als entorns de treball UNIX

1.13. El llenguatge de processament d'arxius GAWK

1.13.3. Síntesis condensada de GAWK

GAWK no es limita a ser una eina de filtratge de text estructurat; és un llenguatge de programació complet que compta amb estructures de control de flux, bucles, diversos operadors i funcions integrades per treballar tant amb cadenes de caràcters com amb números. A més, et brinda la possibilitat de controlar el format de la sortida impresa, utilitzar estructures de dades i escriure funcions *ad hoc*. La combinació intel·ligent d'aquests elements et permetrà crear programes per realitzar transformacions complexes en arxius de text, com podràs veure en molts dels exemples que es presenten a continuació.

La següent llista presenta alguns dels elements i estructures sintàctiques essencials del llenguatge de programació gawk. Tot i que pretenem cobrir tots aquests elements en profunditat, aquesta llista et donarà una idea del poder i la flexibilitat de gawk com a llenguatge de programació.

- **condicionals**

```
if(condició1){code1}else

if(condició2){code2}else

{code3}
```

- **bucles *for***

```
for (i in array) {code};


for (initialization;condition;

      increment|decrement)
```

- **bucles *while***

```
while(true){code}
```

- **operadors aritmètics**

```
+, -, *, /, %, =, ++, -, +=, -=, ...)
```

- **operadors boleans**

```
||, &&
```

- **operadors relacionals**

```
<, <=, ==, !=,>=, >
```

- **funcions integrades**

```
 length(str); int(num); index (str1, str2); split(str,arr,del);
substr(str,pos,len); printf(fmt,args); tolower(str); toupper(str);
gsub(regexp, replacement [, target])
```

- **funcions escrites per l'usuario**

```
function FUNNAME (arg1, arg1) {code}
```

- **estructures de dades**

```
(hashes o arranjaments associatius): array[string]=value
```

Avaluem el potencial de gawk. Els conjunts de dades genòmiques sovint es distribueixen amb arxius separats per a cada cromosoma, i generalment necessitem realitzar la mateixa operació en cadascun. Per realitzar aquesta tasca, podem fer servir l'estructura de bucle *for* de la *shell*. Si tinguéssim arxius anomenats `data_chr[chromosome].txt`, on [chromosome] varia d'1 a 22, i volguéssim executar ./command en ells, s'escriurien les següents ordres al terminal:

```
$ for i in {1..22}; do ./command data_chr$i.txt; done
```

El codi anterior recorre tots els valors entre 1 i 22, establint la variable de la `shell` $i en cada valor, successivament. La sintaxi per especificar els rangs de números és que `{A..B}` dona un rang entre *A* i *B*, on *A* i *B* són valors sencers.

També podem fer servir `for` per recórrer les extensions d'arxiu. Suposem que teníem algunes dades en format PLINK anomenades `data.bed`, `data.bim` i `data.fam`. Podríem llistar aquests arxius individualment executant:

```
$ for ext in bed bim fam; do ls data.$ext; done
```

Aquí, la variable `$ext` (per a extensió) s'estableix successivament en *bed*, *bim* i *fam*. Aquest exemple en particular no és gaire útil, ja que podríem haver escrit `ls data.*` i vist que existeixen aquests tres arxius. El que és útil és poder reanomenar el conjunt de dades base amb una sola línia de codi a la *shell*. Si es volguessin reanomenar aquests arxius `human_data.bed`, `human_data.bim` i `human_data.fam`, es podria escriure:

```
$ for ext in bed bim fam; do mv -i data.$ext human_data.$ext; done
```

Una altra construcció `for` útil és recórrer grups d'arxius. Podem fer el següent per executar `./command` a cada arxiu amb extensió .txt en el nostre directori actual:

```
$ for file in *.txt; do ./command $file; done
```

Ara, suposem que tenim un arxiu anomenat *data.txt* i que volem separar la informació corresponent a cada cromosoma en arxius separats. Podríem fer servir un bucle *for* per recórrer cada número de cromosoma i després una expressió booleana en gawk per extreure només les línies corresponents a aquest cromosoma. Comencem la nostra tasca amb un programa que no funciona del tot i després ho arreglarem.

Si la columna 2 de *data.txt* conté els números de cromosoma i volem arxius separats per a cada cromosoma, podríem pensar que el següent funciona (tingues en compte, novament, el comportament predeterminat de gawk per imprimir les línies que coincideixen amb l'expressió booleana donada):

```
$ for chr in {1..22}; do awk '$2 == $chr' data.txt > data_chr$chr.txt;
done
```

Això és gairebé correcte, però l'expressió donada en *gawk* és `'$2 == $chr'` gawk no l'entén, perquè no coneix ni sap res sobre la variable de la *shell* `$chr` que s'ha definit. En lloc de fer referència a una variable a la *shell*, podem assignar explícitament variables perquè gawk les usi amb l'opció -v:

```
$ for chr in {1..22}; do awk -v chr=$chr '$2 == chr' data.txt >
data_chr$chr.txt; done
```

No està malament! Podem proporcionar a gawk tantes opcions -v com volem per a totes les variables que haguem d'assignar:

```
$ for chr in {1..22};
    do awk -v chr=$chr -v threshold=10 '$2 == chr && $4 > threshold'
data.txt

    > data_chr$chr.txt; done
```

Això separa cada cromosoma en un arxiu individual amb la condició que els valors a la columna 4 siguin majors que 10.

Què passa si tenim un arxiu que conté dues columnes amb un recompte d'«èxits» i «intents» per a algun procés, cadascun recol·lectat d'una font diferent, i volem calcular la taxa mitjana d'èxit entre totes les fonts? Podem fer servir gawk per sumar cada columna i després imprimir la mitjana al final fent servir una sintaxi especial. Si l'arxiu *data.txt* té els èxits i els intents a les columnes 2 i 3, respectivament, podríem fer això:

```
$ gawk 'BEGIN
 {total_success = total_attempts = 0;}

 {total_success += $2; total_attempts += $3}

END

 {print "Éxitos:", total_success, "Intentos:", total_attempts, "Tasa:",
total_success / total_attempts}' data.txt
```

L'ordre gawk executa el codi a la secció BEGIN abans de processar qualsevol línia a l'entrada i el codi a la secció END després de llegir l'entrada. La secció BEGIN inicialitza dues variables a 0, la secció de codi principal suma les columnes a cada línia, i la secció END imprimeix els totals i la taxa.

Suposem ara que volem buscar mitjançant condicionals aquells assemblatges que conté l'organisme a *Xenopus tropicalis*:

```
$ gawk ' BEGIN {
    FS="\t";

    print
"assembly_accession\torganism_name\tseq_rel_date\tasm_name\tsubmitter"

    }



{

    if ($2 == "Xenopus tropicalis") {

        print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5

    }

}

' ensamble.txt
```

```
GCA_017527675.1 Phaeodactylum tricornutum  2022-03-24 Phatr3.0   NCBI
```

```
GCA_011586775.1 Xenopus tropicalis   2022-04-11 Xenbase_v9.2    NCBI
```

En el següent exemple, es volen filtrar les dades de Xenopus tropicalis que tinguin una data de llançament inferior al 2015. L'exemple utilitzarà un condicional en *bash* i es crearà *script* que es pugui executar. Salva les pròximes ordres en un fitxer del teu terminal.

```
#!/bin/bash

# Llegir la taula i guardar les línies que compleixen les condicions en un nou arxiu
awk -F'\t' 'NR==1 || ($2 == "Xenopus tropicalis" && $3 > "2015-01-01")' $1
> filtered_table.txt

# Comptar el nombre de línies a l'arxiu filtrat
num_lines=$(wc -l < filtered_table.txt)

# Si el nombre de línies és més gran que 1 (és a dir, si hi ha entrades que compleixen les condicions),
# imprimir un missatge i el contingut de l'arxiu filtrat
if [ $num_lines -gt 1 ]; then

  echo "Es van trobar les següents entrades per a Xenopus tropicalis
publicades després de 2015-01-01:"

  cat filtered_table.txt

# Si el nombre de línies és igual a 1, imprimir un missatge amb el nom de
l'entrada
elif [ $num_lines -eq 1 ]; then

  echo "Se encontró la siguiente entrada para Xenopus tropicalis publicada
después de 2015-01-01:"

  awk -F'\t' '{print $4}' filtered_table.txt

# Si el nombre de línies és 0, imprimir un missatge que digui que no es van trobar entrades que compleixin les condicions
else

  echo "No es van trobar entrades per a Xenopus tropicalis publicades
després de 2015-01-01."

fi
```

Salva el fitxer i executa les següents ordres en el terminal:

```
$ chmod +x script.sh
```

```
$ ./script.sh ensamble.txt
```

Aquest *script* utilitza gawk per llegir la taula i guardar les línies que compleixen les condicions en un nou arxiu anomenat *filtered_table.txt*. Després, compta el nombre de línies en aquest arxiu i utilitza dues condicions *if* per imprimir missatges

diferents depenent de si hi ha entrades que compleixin les condicions o no. En el primer cas, quan hi ha més d'una entrada que compleix les condicions, l'*script* imprimeix un missatge que indica que es van trobar entrades i mostra el contingut de la taula filtrada. En el segon cas, imprimeix un missatge si només hi ha una entrada que compleix la condició i l'última acció ocorrerà si no hi ha cap entrada en el fitxer de partida que compleixi les condicions demandades.

La taula 12 és un llistat de les operacions més comunes i utilitzades amb GAWK.

Taula 12. Operacions utilitzades a GAWK.

Variable	Descripció
`i=0`	Assignar un valor
`a[i]=0;`	Guardar un valor en una taula
`i++;`	Incrementar un comptador
`print i;`	Mostrar una variable per pantalla
`print NR`	Mostrar el nombre de línies processades
`print $1,$4`	Mostrar la primera i quarta columnes

Font: elaboració pròpia.

I a la taula 13 es mostren exemples d'instruccions condicionals amb GAWK.

Taula 13. Instruccions condicionals amb GAWK.

Variable	Descripció
`if ($1 > 0) print $0;`	Si la primera és positiva
`if ($1 < 0) print $0;`	Si la primera és negativa
`if ($1 >= 0) print $0;`	Si la primera columna és major o igual a zero
`if ($1 <= 0) print $0;`	Si la primera columna és menor o igual a zero
`if ($1 == 0) print $0;`	Si la primera columna és igual a zero
`if (CONDICIÓ1) && (CONDICIÓ2)`	Si es compleixen les dues condicions
`if (CONDICIÓ1) || (CONDICIÓ2`	Si es compleix alguna de les dues condicions
`if ¡(CONDICIÓ1)`	Si no es compleix la condició

Font: elaboració pròpia.

1. Introducció als entorns de treball UNIX

1.13. El llenguatge de processament d'arxius GAWK

1.13.4. Expressions regulars (en anglès, regexps)

Atès que gawk és un llenguatge especialitzat en el processament d'arxius de text basat en patrons, és essencial tornar a presentar una secció sobre expressions regulars. Una nova presentació sempre ajuda a fixar els temes a estudiar.

Una expressió regular (*regex* o *regexp*) defineix un o diversos conjunts de cadenes de caràcters utilitzant una notació específica:

- Una cadena literal de caràcters és una *regex* que defineix una sola cadena: a si mateixa.
- Una expressió regular més complexa:
  - Caràcters ordinaris utilitzats en *regexe*: _, `A-Z, a-z, 0-9`
  - Metacaràcters utilitzats en *regexe*: `.*[ ]^${ }+?|( )`

Les *regexes* s'utilitzen per trobar patrons específics en arxius de text. Programes com *ed, vim, grep, sed, awk, perl,* entre molts altres, fan ús de *regexes* per buscar aquests patrons en arxius de text.

Hi ha dos motors de recerca de patrons mitjançant expressions regulars i cal tenir-ho molt en compte en funció de l'ordre que es vol utilitzar

- El motor bàsic d'expressions regulars (BRE), usat per exemple per *sed* i *grep*.
- El motor estès d'expressions regulars (ERE), usat per exemple per gawk i *perl*.

S'afegeixen dues taules amb la notació de caràcters BRE/ERE usats més freqüentment amb gawk (taules 14 i 15).

Taula 14. Notació de caràcters especials. Motor BRE/ERE.

Notació	Significat
\	Escapa el significat del metacaràcter, interpretació literal
.	Qualsevol caràcter senzill excepte NULL
*	Qualsevol quantitat de vegades (o zero) el caràcter precedent
^	La *regexp* coincideix a l'inici de la línia o cadena de caràcters
$	La *regexp* coincideix al final de la línia o cadena de caràcters
[123][A-Z]	Qualsevol dels caràcters inclosos o rang indicat coincideixen
[[:alnum:]]	Alfanumèrics [a-zA-Z0-9_]
[[:alpha:]]	Caracters alfabètics [a-zA-Z]
[[:space:]]	Espais (' ', tabuladors, salt de línia)
[[:blank:]]	Coincideixen espais i tabuladors
[[:upper:]]	Coincideix [A-Z]
[[:lower:]]	Coincideix [a-z]
[[:digit:]]	Coincideix [0-9]

Font: elaboració pròpia.

I una altra taula en particular, amb la notació de caràcters ERE usats més freqüentment:

Taula 15. Classe de caràcters ERE més freqüentment usats.

Notació	Significat
\w	Caràcters alfanumèrics [a-zA-Z0-9_]
\W	Caràcters no alfanumèrics [^[:alnum:]_]
\s	Coincideix amb espais i tabuladors
\d	Coincideix [0-9]
\<	Coincideix amb l'inici d'una paraula
\>	Coincideix amb el final d'una paraula
{n,m}	Expressió d'interval: coincideixen instàncies, o de *n* a *m* instàncies
+	Coincideixen una o més instàncies de la *regexp* precedent
¿	Coincideixen zero o una instància de la *regexp* precedent
|	Coincideix la *regexp* especificada abans o després de | (això|allò)
( )	Busca un *match* al grup de *regexes* incloses: (això|allò)

Font: elaboració pròpia.

1. Introducció als entorns de treball UNIX

1.13. El llenguatge de processament d'arxius GAWK

1.13.5. Manipulación de cadenas de caracteres

L'ordre gawk proveeix de diverses funcions per a la manipulació de cadenes de caràcters. En aquest apartat només es mostren les funcions molt senzilles d'usar que s'implementen amb freqüència en codi *gawk*, i a continuació diferents exemples pràctics amb els quals es pot practicar.

- **match()** match(cadena, regexp [,array])
- **index()** index(in_str1, find_str2)
- **sub()** /regexp/, "reemplazo", [, diana]
- **gsub()** /regexp/, "reemplazo", [,diana]
- **substr()** cadena, inicio, [,longitud]
- **split()** split(string, array [, fieldsep [,seps]])
- **length ()** ([string])
- **tolower()** (string)
- **toupper()** (string)

1. sub() busca la instància més llarga en la cadena diana de la *regexp*, substituint-la per la cadena reemplaçament, mentre que gsub() realitza reemplaçaments globals.

```
$ echo 'GGCCCACACAAACGCGACGGCGAA' | gawk 'sub(/GACGGC/, "")'
```

```
GGCCCACACAAACGCGAA
```

```
$ gawk 'BEGIN { cad = "És una cadena de les bases nitrogenades: cadena";
print cad; sub(/cadena/, "seqüència", cad); print cad}'
```

```
És una cadena de les bases nitrogenades: cadena
És una seqüència de les bases nitrogenades: cadena
```

```
$ gawk 'BEGIN { cad = "És una cadena de bases nitrogenades: cadena";
print cad; gsub(/cadena/, "secuencia", cad); print cad}'
```

```
És una cadena de bases nitrogenades: cadena
És una seqüència de bases nitrogenades: seqüència
```

2. index() imprimeix l'índex en què comença la subcadena (AAA) a la cadena principal (*seq*, en aquest cas).. index() i substr() amb freqüència s'usen junts.

```
$ gawk 'BEGIN { seq = "GGCCCACACAAACGCGACGGCGAA";
```

```
 idx = index(seq, "AAA"; print idx}'
```

```
 10
```

```
 $ echo 'GGCCCACACAAACGCGACGGCGAA' | gawk '{ idx = index($0, "AAA"); gene =
substr($0, idx); print idx, "\n"$0, "\n          " gene }'

 10

 GGCCCACACAAACGCGACGGCGAA

           AAACGCGACGGCGAA
```

3. I un últim exemple amb `length` (en català, *longitud*).

```
 $ echo 'GGCCCACACAAACGCGACGGCGAA' | gawk 'END {print "El oligo",  $0,
"té", length($0), "nt de longitud."}'
```

```
 L'oligo GGCCCACACAAACGCGACGGCGAA té 24 nt de longitud.
```

1. Introducció als entorns de treball UNIX

1.14. Definició de noves ordres

Durant aquest mòdul, hem observat que les ordres del terminal compten amb un ampli nombre d'opcions que permeten múltiples combinacions i és comprensible que cada usuari habitualment utilitzi només un petit conjunt d'aquestes possibilitats de configuració. Donat aquesta particularitat, l'intèrpret d'ordres permet definir noves ordres basades en determinades combinacions d'ordres i opcions que l'usuari requereix amb freqüència. L'ordre `alias` implementa aquest mecanisme, associant un nou nom d'ordre a una seqüència d'ordres i opcions prefixades.

Per crear un àlies, fa servir l'ordre `alias` amb el nom apropiat. Sense arguments, aquest llistarà tots els àlies definits fins aquell moment. Mentre estàs a la *shell*, pots comprovar quins àlies estan establerts escrivint l'ordre `alias`. Per crear un àlies, dona un nom, seguit de = i després l'ordre a ser àlies entre cometes simples. No hi ha d'haver espais al voltant de l'operador =. Fes servir l'ordre `type` nom per comprovar si aquest nom ja està sent utilitzat per una altra ordre. A continuació, alguns exemples:

```
$ type p
```

```
bash: type: p: not found
```

```
 $ alias p='pwd'
```

```
$ p
```

```
/home/student/HIB
```

En aquest exemple s'ha habilitat la lletra *p* a l'ordre `pwd`.

```
$ alias pl='pwd ; ls -CF'
```

```
$ alias rm='rm -i'
```

En el primer exemple, les lletres `pl` s'assignen per executar l'ordre `pwd` i, a continuació, s'executa l'ordre `ls -CF` amb el que primer s'imprimeix el directori de treball actual i llista el seu contingut en forma de columna. El segon exemple executa l'ordre `rm` amb l'opció `-i` cada vegada que escrius `rm` (aquest és un àlies que sovint s'estableix automàticament per a l'usuari *root*. En lloc de simplement eliminar arxius, se't demanarà confirmació per a cada eliminació individual d'arxiu. Això evita que eliminin automàticament tots els arxius d'un directori en escriure accidentalment alguna cosa com `rm *`.)

Si vols eliminar un àlies, escriu `unalias`, lletres que defineixen l'àlies (recorda que, si l'àlies està configurat en un arxiu de configuració, s'establirà de nou quan obris una altra *shell*).

```
$ unalias p pl
```

```
$ . .bashrc
```

```
$ type p pl
```

```
bash: type: p: not found

bash: type: pl: not found
```

Si desitges guardar una llista personal d'àlies de forma permanent, pots escriure'ls a l'arxiu $HOME/.bash_aliases, que és un dels arxius de configuració de l'entorn que els usuaris avançats de Gnu/Linux guarden en el seu directori $HOME (/home/student).

Cada vegada que iniciïs la teva màquina o estableixis una sessió remota, l'arxiu es llegirà en memòria, cosa que permetrà que els àlies s'integrin a l'entorn.

L'arxiu $HOME/.bash_aliases és utilitzat pels arxius de configuració $HOME/.bashrc a la màquina local, i $HOME/.bash_profile quan s'estableix una sessió remota a través de SSH. En la secció d'administració bàsica d'un sistema Gnu/Linux, aprofundirem en l'ús d'aquests arxius (apartat 1.16).

Com a exercici, et suggereixo que busquis i llegeixis aquests arxius. Et comunico que necessites utilitzar l'ordre *ls -a* per veure'ls, ja que el nom d'arxiu està precedit per un punt per ocultar-los d'una crida de *ls* estàndard.

1. Introducció als entorns de treball UNIX

1.15. Disseny de protocols automàtics al terminal

En aquest apartat us introduireu en el disseny de protocols automàtics en un terminal *bash*. Dins del camp de la bioinformàtica aprendre a dissenyar protocols és útil per diverses raons. En primer lloc, permet l'automatització de tasques repetitives, la qual cosa estalvia temps i redueix el risc d'errors. En segon lloc, permet la creació de fluxos de treball reproduïbles, que són essencials per a la recerca científica. En tercer lloc, *bash* és una eina poderosa i flexible que es pot utilitzar per manipular conjunts de dades grans i realitzar anàlisis complexes. Finalment, l'ús de protocols (en anglès, *script*) *bash* facilita la col·laboració i l'intercanvi de protocols, cosa que fa que sigui més fàcil per als investigadors reproduir la feina d'altres i construir-hi. En general, el disseny de protocols automàtics en un terminal *bash* és una forma eficient i efectiva de realitzar anàlisis de bioinformàtica.

En dissenyar un protocol en un terminal *bash*, hi ha diversos aspectes tècnics i paràmetres que s'han de tenir en compte. Aquí es presenten els aspectes més importants que cal considerar.

- **Sintaxi i gramàtica**: el protocol ha de tenir una sintaxi i una gramàtica ben definides que siguin fàcils d'entendre i seguir.
- **Maneig d'errors**: el protocol ha d'estar dissenyat per manejar errors i excepcions de manera elegant. Això inclou la definició de codis d'error i missatges.
- **Seguretat**: el protocol ha d'estar dissenyat per garantir la privacitat i seguretat de les dades. Això inclou mesures com l'encriptació, l'autenticació i el control d'accés.
- **Compatibilitat**: el protocol ha de ser compatible amb el *hardware* i el *software* del sistema en el qual s'utilitzarà.
- **Eficiència**: el protocol ha d'estar dissenyat per ser eficient i optimitzar l'ús dels recursos del sistema, com la CPU i la memòria.
- **Escalabilitat**: el protocol ha d'estar dissenyat per ser escalable, de manera que pugui manejar quantitats creixents de dades i usuaris sense degradació del rendiment.
- **Documentació:** el protocol ha d'estar ben documentat, amb instruccions clares i exemples per a la seva implementació i ús.
- **Proves**: el protocol ha de ser provat exhaustivament per assegurar que la seva funcionalitat i rendiment compleixin amb els requisits.

Tenint en compte aquests aspectes tècnics i paràmetres, el protocol pot ser dissenyat per ser efectiu, segur i eficient en el seu funcionament. En aquest apartat se us mostra com dissenyar protocols senzills, i no es tindrà en compte tot el que s'ha esmentat anteriorment, però sempre s'ha de treballar tenint en compte totes les consideracions esmentades.

En aquest exercici se us mostrarà com generar un protocol per obtenir informació biològica rellevant a partir de 3 seqüències FASTA. T'animo a realitzar tots els passos que es mostren a continuació. Obre un terminal i comença.

1) Crea un directori:

```
$ mkdir fasta_sequence
```

```
$ cd fasta_sequence
```

2) Descarrega, en l'anterior directori i des de la base de dades ENA (*European Nucleotide Archive*), les seqüències FASTA associades als gens humans de BRCA1, BRCA2 i HOXB13:

- Uniprot ID: P38398, ENA accesion number: AC060780
  https://www.ebi.ac.uk/ena/browser/view/AC060780
- Uniprot ID: P51587, ENA accesion number: AL137247
  https://www.ebi.ac.uk/ena/browser/view/AL137247
- Uniprot ID: Q92826, ENA accesion number: BC007092
  https://www.ebi.ac.uk/ena/browser/view/BC007092

3) Quan es volen executar una sèrie d'ordres de forma seqüencial al terminal *bash*, es crea un *script* o protocol i es guarda en un arxiu de text amb extensió «`.sh`». Aquest *script* ens permet referir-nos internament als seus paràmetres de manera genèrica, la qual cosa significa que pot funcionar en qualsevol grup d'arguments del mateix tipus. A més, per fer-lo més flexible s'han d'especificar aquests paràmetres genèrics dins de *l'script*, de manera que, quan s'invoqui *l'script* des de la línia d'ordres, podem substituir aquests paràmetres genèrics per valors específics proporcionats per l'usuari juntament amb el nom de l'ordre. En utilitzar aquest enfocament, es poden automatitzar tasques repetitives o processos complexos, la qual cosa fa que la feina sigui més eficient i consistent.

```
$ vi analitza_fasta.sh
```

4) Abans d'introduir el codi per generar un protocol, dues consideracions a tenir en compte:

- En la primera línia del protocol s'ha d'indicar, sempre, l'intèrpret d'ordres `bash` (GNU Bourne-Again SHell). En la majoria dels sistemes coexisteixen altres intèrprets com `sh` (Bourne) o `csh` (C shell). El directori on es localitza l'intèrpret s'escriu a continuació del conjunt de símbols `#!`
- Per introduir comentaris en el protocol s'ha d'introduir el símbol `#` (en anglès, *shebang*) a la primera columna del fitxer.

5) Escriu cadascuna de les següents línies en el fitxer *sh* que acabes d'obrir. Has d'escriure en mode *insert*, el primer que has de fer és teclejar la lletra `i`, i a continuació escriu les línies següents:

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

Aquest protocol analitza un fitxer que conté una única seqüència FASTA

Primer, se subministra el nom del fitxer com a argument

```
if [ $# -eq 0 ]

then

    echo "Per favor, subministra un nom de fitxer com a argument"

    exit 1

fi
```

El símbol $$ indica el PID del procés

```
echo "El valor del PID del procés és";

echo "PID és $$";
```

$1 representa el primer argument a analitzar. En aquest cas el nom del fitxer FASTA

```
echo "La mida del fitxer FASTA és:";

ls -sh $1 | gawk '{print $1}';

echo "Nombre de línies del fitxer FASTA:";

wc -l $1 | gawk '{ print $1 }';

echo "Extreu les primeres set línies del fitxer";
```

```
head -7 $1;
```

Les següents dues línies considera comentar-les, si la seqüència FASTA té moltes línies

```
echo "Extreu la seqüència del fitxer FASTA";

sequence=$(awk '/^>/ {next} {printf "%s", $0} END {print ""}' "$1")

echo "$sequence"
```

Càlcul de la longitud de la seqüència

```
length=$(echo -n "$sequence" | wc -c)

echo " Longitud de la seqüència: $length nucleotids"
```

Compte el número de nucleòtids que té la seqüència

```
num_A=$(grep -o 'A' <<< "$sequence" | wc -l)

num_C=$(grep -o 'C' <<< "$sequence" | wc -l)

num_G=$(grep -o 'G' <<< "$sequence" | wc -l)

num_T=$(grep -o 'T' <<< "$sequence" | wc -l)
```

S'han generat *in situ* 4 noves variables. Imprimeix cada nombre de nucleòtids

```
echo "Nombre de nucleòtid A: $num_A"

echo "Nombre de nucleòtid C: $num_C"

echo "Nombre de nucleòtid G: $num_G"

echo "Nombre de nucleòtid T: $num_T"
```

Càlcul del contingut GC de la seqüència

```
num_GC=$((num_C + num_G))

total=$((num_A + num_C + num_G + num_T))

gc_content=$(bc -l <<< "scale=2; $num_GC / $total * 100")

echo "GC contenido: $gc_content%"
```

Identificació dels ORFs de la seqüència

```
echo "Identificant ORFs..."

ORFs=$(echo -n "$sequence" | tr 'ATCG' 'tacg' | grep -Eo '(atg([acgt]
{3})*?(taa|tag|tga))+' | tr 'tacg' 'ATCG')
```

```
if [ -z "$ORFs" ]
then
    echo "No es troben ORFs"
else
    num_ORFs=$(echo -n "$ORFs" | awk '{print length}' | wc -l)
    echo "Nombre d'ORFs: $num_ORFs"
    echo "ORFs: $ORFs"
fi
```

Salva el fitxer que s'ha creat escrivint: `wq!  analitza_fasta.sh`

6) Per executar l'*script*, salva l'anterior fitxer amb l'extensió ".sh" i fes-lo executable amb l'ordre `chmod +x`. Executa'l amb el nom del fitxer FASTA a analitzar:

```
$ pwd
```

```
/home/student/fasta_sequence
```

```
 $ chmod +x analitza_fasta.sh
```

```
$ ./analitza_fasta.sh AC060780.18.fasta
```

7)Fins ara s'ha generat un protocol per analitzar fitxers FASTA, s'ha executat i comprovat que funciona. El següent pas és executar l'anterior fitxer d'anàlisi en un directori amb diferents seqüències. Per a això, es genera un altre protocol que inclogui l'anterior. El nou protocol es diu `analitza_fasta_dir.sh`. Els passos per seguir són:

```
$ pwd
```

```
/home/student
```

```
$ vi analitza_fasta_dir.sh
```

Les ordres que cal introduir en aquest fitxer són:

```
#! /bin/bash

echo "S'inicia l'execució amb el  PID $$";

echo "Nombre d'arguments a analitzar  $#";

echo "El nom del directori a analitzar  $1";

echo "Els noms dels fitxers que s'analitzen són";
```

```
 ls $1;

 echo "Execució del protocol d'anàlisi FASTA per a cadascun dels fitxers en
el directori";

 ls $1 |    while read file;

      do

           echo "execució analitza_fasta.sh $file";

           ./analitza_fasta.sh $1/$file;

      done
```

Salva el fitxer que s'ha creat escrivint :wq! analitza_fasta_dir.sh

8) Per executar l'*script*, salva l'anterior fitxer amb l'extensió .sh i fes-lo executable amb l'ordre chmod +x. Executa-ho amb el nom del directori que conté els fitxers FASTA a analitzar:

```
$ pwd
```

```
/home/student
```

```
$ cp fasta_sequence/analitza_fasta.sh .
```

```
$ chmod +x analitza_fasta_dir.sh
```

```
$ ./analitza_fasta_dir.sh fasta_sequence
```

Es pot observar que els protocols que s'executen es troben en el directori actual de treball (./) en lloc de només invocar el seu nom. No obstant això, un cop s'està segur que *l'script* funciona correctament, és més convenient guardar tota la nostra col·lecció de protocols en un sol directori del sistema. D'aquesta manera, no s'han de mantenir múltiples còpies i es podran executar des de qualsevol lloc en el nostre arbre de directoris.

Les plataformes Gnu/Linux tenen un conjunt de variables globals que contenen dades de caràcter general. Es resumeixen a la taula següent, la taula 16.

Taula 16. Ordres per a l'execució de *scripts*.

Ordre	Descripció
set	Estableix valors que seran usats pels programes, aplicació, *scripts*
echo	Imprimeix el que se li encarrega que faci
printenv	Llista la llista completa de variables d'entorn de la teva versió Gnu/Linux
which	Localitza els arxius executables d'una determinada aplicació
export	Crea/modifica una variable del sistema

Font: elaboració pròpia.

L'ordre `set` a Gnu/Linux mostra i estableix variables d'entorn per a la sessió actual. S'utilitza per veure el valor de les variables d'entorn del sistema i també per assignar valors a noves variables d'entorn. L'ordre `echo`, d'altra banda, s'utilitza per mostrar missatges de text o el valor de les variables a la pantalla. També es pot utilitzar per escriure text en arxius o per concatenar diversos missatges de text.

L'avantatge d'utilitzar *echo* és que permet seleccionar i mostrar només la informació que es necessita, el que fa que sigui més fàcil de llegir i entendre. D'altra banda, l'avantatge d'utilitzar *set* és que permet establir i modificar variables d'entorn, la qual cosa pot ser útil en automatitzar tasques i *scripts* a Gnu/Linux.

Obre un terminal de Gnu/Linux. Pots veure la llista completa de variables d'entorn de la teva versió de Gnu/Linux utilitzant l'ordre `printenv`. Pots tenir una llista més manejable afegint-hi diferents ordres:

```
$printenv | less
```

Cada línia conté el nom de la variable d'entorn Gnu/Linux seguit de = i del valor. Per exemple:

```
HOME=/home/student
```

Això vol dir que HOME és una variable d'entorn de Gnu/Linux que té el valor establert com *a directori /home/student*.

Les variables d'entorn solen estar en majúscules, tot i que també pots crear variables d'entorn en minúscules. La sortida de `printenv` mostra totes les variables d'entorn en majúscules. Una cosa important per tenir en compte és que les variables d'entorn de Gnu/Linux distingeixen entre majúscules i minúscules. Si desitges veure el valor d'una variable d'entorn específica, pots fer-ho considerant el nom d'aquesta variable com a argument de l'ordre `printenv`. La cadena de caràcters completa es veuria així en la línia d'ordre:

```
$ printenv HOME
```

```
/home/student
```

```
$ echo $USER
```

```
student
```

La sintaxi bàsica per crear una variable d'entorn a Gnu/Linux és la següent. És fàcil aconseguir-ho, només es necessita especificar un nom i un valor. Seguirem la convenció de mantenir totes les lletres en majúscules per al nom de la variable, i l'establirem com una cadena simple.

```
$ HIB_VAR='BioInformatica i BioEstadistica!'
```

```
$ export HIB_VAR
```

```
$ export HIB_VAR='BioInformatica i BioEstadistica!'    #en una única línia
```

S'han fet servir cometes simples, ja que el valor de la variable conté un espai. A més, s'han utilitzat cometes simples perquè el signe d'exclamació és un caràcter especial en la *shell bash* que normalment s'expandeix a l'historial de *bash* si no s'escapa o es col·loca entre cometes simples. Ara tenim una variable de la *shell*. Aquesta variable està disponible en la nostra sessió actual, però no es transmet als processos secundaris.

Es pot comprovar, buscant la nostra nova variable dins de la sortida de *set*:

```
$ set | grep HIB_VAR
```

```
HIB_VAR='BioInformartica i BioEstadistica!'
```

Si la variable s'ha definit correctament, podràs utilitzar-la en qualsevol protocol que executis en la mateixa sessió del terminal. Per exemple, si crees un arxiu de *script* en *bash* que requereix utilitzar la variable HIB_VAR, simplement pots referir-t'hi utilitzant el símbol $. Per exemple, si el teu arxiu de *script* s'anomena *myscript.sh* i conté el següent codi:

```
#! /bin/bash

echo " El valor de HIB_VAR es: $HIB_VAR"
```

Si s'executa l'arxiu de *script* en la mateixa sessió del terminal utilitzant l'ordre *bash*,

```
$ chmod +x myscript.sh
```

```
$ bash myscript.sh
```

```
BioInformatica i Bioestadistica!
```

l'arxiu de *script* hauria d'imprimir el valor de la variable HIB_VAR que vas definir prèviament. Per revertir el valor d'una variable es pot fer servir l'ordre unset.

```
$ echo $HIB_VAR
```

```
BioInformartica i BioEstadistica!
```

```
$ unset HIB_VAR
```

```
$ echo $HIB_VAR
```

D'altra banda, tens en compte que, si tanques la sessió del terminal i la tornes a obrir, hauràs de tornar a definir la variable local utilitzant l'ordre export. Per evitar això, pots agregar la definició de la variable HIB_VAR al teu arxiu d'inici de *bash* (per exemple, ~/.bashrc), de manera que la variable estigui disponible cada vegada que inicïis una nova sessió del terminal.

L'ordre which és una eina que permet trobar ràpidament els arxius executables d'una determinada aplicació i localitza els fitxers executables mitjançant la variable d'entorn PATH.

```
$ which nano docker gawk
```

```
/usr/bin/nano

/usr/bin/docker

/usr/bin/gawk
```

Finalment, es defineix la variable PATH. El contingut de la variable PATH és una cadena que conté paths de directoris separats per dos punts, i aquests són els directoris en què la *shell* busca l'ordre que l'usuari escriu des del teclat.

```
$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games:/snap/bin
```

La recerca no es realitza en l'ordre en el qual estan els directoris en la variable PATH. Quan s'escriu una ordre a la *shell* buscarà primer a /usr/local/bin, després a /usr/bin, a continuació a /usr/games i finalment a /snap/bin. Des del moment en què la *shell* troba l'ordre, deté la recerca i executa l'ordre trobada. Podem escriure una ordre utilitzant:

- El seu nom:
  El path absolut (*/bin/cat /etc/passwd*).
  El path relatiu (utilitzant «.» o «....» en general per als programes o *scripts* que no es troben al PATH).
  Es pot afegir un directori a la variable PATH, afegim el directori on es troben tots els *scripts* que es generin.

- Únicament per a la sessió activa:
  Si desitges afegir, per exemple: `/home/student/HIB_scripts` a la variable `PATH`, escriu a la *shell* el següent segons el cas.

Per tenir el directori al final del PATH:

```
$ export PATH=$PATH:/home/student/HIB_scripts
```

Per tenir el directori a l'inici del PATH:

```
$ export PATH=/home/student/HIB_scripts/:$PATH
```

Ara pots utilitzar el programa escrivint simplement el seu nom. En desconnectar-se, PATH reprendrà al seu valor per defecte, llavors /home/student/HIB_scripts no existirà més.

- De manera permanent:
  Si desitges configurar PATH de forma permanent has d'editar l'arxiu de configuració de la seva *shell* de connexió. Com que en general la *shell bash* és el més utilitzat, has d'editar l'arxiu: /home/user/.bashrc. L'ordre llavors seria:

  ```
  $ echo 'export PATH=$PATH:/home/student/HIB_scripts' >>
  /home/user/.bashrc
  ```

  Després d'això, en cada connexió la variable PATH contindrà el directori */home/student HIB_scripts.* Aquesta operació pot ser executada per l'usuari *student*, no es necessiten els permisos de *root*.

1. Introducció als entorns de treball UNIX

1.16. Transferència de fitxers des del terminal

La transferència d'arxius des d'un terminal és una tasca fonamental per als administradors de sistemes i desenvolupadors que treballen amb servidors remots. Tot i que hi ha moltes eines gràfiques per transferir arxius, les opcions de la línia d'ordres són molt útils per a l'automatització i el maneig de grans quantitats de dades. Exemples de diferents ordres es mostren a la taula 17.

Taula 17. Ordres de transferència de fitxers.

Ordre	Significat
`scp`	*Secure copy*
`sftp`	*Secure File Transfer Protocol*
`rsync`	una eina ràpida, versàtil, remota (i local) de còpia de fitxers
`wget`	Podeu utilitzar-lo per recuperar contingut i fitxers de diversos servidors web
`curl`	curl L'URL del client (cURL) us permet intercanviar dades entre el dispositiu i un servidor a través d'una interfície de línia d'ordres (CLI)

Font: elaboració pròpia.

Per establir una sessió de treball cal conèixer el nom de la màquina remota o la seva adreça IP. També hem de disposar d'un nom d'usuari (en anglès, *login name*) i d'una contrasenya d'accés (en anglès, *password*).

Una de les ordres més comunes per transferir arxius és `scp` (*Secure Copy*). Aquesta ordre s'utilitza per copiar arxius d'un servidor a un altre utilitzant el protocol SSH. La sintaxi bàsica és la següent:

```
$ scp [opcions] origen destí
```

Per exemple, si volem copiar l'arxiu *arxiu.txt* de la carpeta origen del servidor remot a la nostra carpeta actual a la màquina local, podem utilitzar l'ordre següent:

```
$ scp usuario@servidor-remoto:/path/al/origen/arxiu.txt .
```

El punt final en la línia d'ordre indica el directori actual de la màquina local.

També existeix l'ordre `sftp` (en anglès, *Secure File Transfer Protocol*), que és similar al client FTP però amb una capa de seguretat addicional a través de SSH. Amb *`sftp`*, podem connectar-nos a un servidor remot i transferir arxius de manera segura. La sintaxi bàsica és la següent:

```
$ sftp usuario@servidor-remot
```

```
put seqüencia.fa repositori-seqüencies/
```

Un cop connectats, podem utilitzar ordres com `put` per enviar arxius des de la nostra màquina local al servidor remot, o `get` per descarregar arxius del servidor remot a la nostra màquina local. En l'exemple anterior, hem descrit com podem enviar l'arxiu *seqüència.fa* des de la nostra màquina local a la carpeta repositori-seqüències en el servidor remot.

Una altra ordre útil és `rsync`, que permet sincronitzar directoris i arxius entre dos sistemes. A més, `rsync` té la *capacitat* de transferir només els arxius modificats, cosa que el fa ideal per fer còpies de seguretat de manera eficient. La sintaxi bàsica de `rsync` és la següent:

```
$ rsync [opciones] origen destí
```

Per exemple, si volem sincronitzar la carpeta *work-hib* del nostre equip local amb la carpeta *work* en un servidor remot, podem utilitzar l'ordre següent:

```
$ rsync -avz /path/a/work-hib usuario@servidor-remoto:/path/a/work
```

Una altra eina útil és `wget`, que permet descarregar arxius des de servidors web fent ús d'ordres de terminal. Aquesta ordre és molt útil per descarregar dades de seqüenciació de genomes complets o de grans bases de dades biològiques, arxius des d'internet i automatitzar el procés de descàrrega. Es mostra a continuació com descarregar la base de dades COSMIC (*Catalogue Of Somatic Mutations In Cancer*) utilitzant `wget`

```
$ wget
https://cancer.sanger.ac.uk/cosmic/file_download/GRCh38/cosmic/v94/CosmicCoc
```

Aquesta ordre descarrega l'arxiu *CosmicCodingMuts.vcf.gz* de la pàgina web del projecte COSMIC. Aquest arxiu conté les mutacions somàtiques trobades en la seqüenciació d'ADN de tumors de càncer. Un cop descarregat, el podeu eliminar, ja que no en farem res d'aquest arxiu.

Finalment, una altra eina útil és `curl`, que també permet descarregar arxius des de servidors web i transferir arxius des d'un terminal. La principal diferència entre `wget` i `curl` és que `curl` és més versàtil i permet transferir arxius mitjançant una gran varietat de protocols, incloent-hi FTP, FTPS, HTTP, HTTPS, SCP i SFTP.

En resum, la transferència d'arxius des d'un terminal és una tasca fonamental per als administradors de sistemes i desenvolupadors que treballen amb servidors remots. Les ordres `scp`, `rsync` i `sftp` són algunes de les eines més comunes i útils per transferir arxius de manera segura i eficient.

Hi ha una ordre que ajuda a esbrinar l'espai disponible d'un disc, l'ordre df (*disk free*), i una altra per esbrinar l'espai ocupat per una carpeta, l'ordre du (*disk use*). Analitzem un parell d'exemples per veure el seu funcionament.

- `df`

Espai disponible en disc. Això ens retornarà les particions muntades, l'ús d'espai en cadascuna i el que ens queda de resta, i tot de forma fàcil per llegir, sobretot perquè s'hi afegeix l'opció *-h,* que significa *human-readable* (en català, *humanament llegible*).

```
$ df -h
```

```
Filesystem      Size  Used Available Use% Mounted on

udev            959M     0       959M   0% /dev

tmpfs           199M   1,4M      197M   1% /run

/dev/sda5        20G    16G       2,9G  85% /

tmpfs           991M     0       991M   0% /dev/shm

tmpfs           5,0M   4,0K      5,0M   1% /run/lock

tmpfs           991M     0       991M   0% /sys/fs/cgroup

/dev/loop1       64M    64M          0 100%
/snap/core20/1852
```

- `du`

Mida total d'una carpeta. Ús de disc. Mostra l'espai que està ocupat en disc. Aquesta ordre té diverses opcions i les més utilitzades són les opcions *-b* (*-bytes*); *-s* (*sumarize*; en català, *en resum*), *-h* (*-human-readable*; en català, *humanament llegible*), que

imprimeix les mides de forma llegible, quan agreguem la mida dels arxius en kb, mb, gb; i, finalment, l'ordre *-c*, que emprarem per mostrar el total de l'espai consumit, al final de la llista.

```
$ du -hs * | sort -nr | head -5
```

```
204M    fasta_sequences

45M     Escriptori

12G     hg38

12M     Work

4,0K    Videos
```

Amb aquest exemple, es visualitzen els 5 directoris més pesants en el nostre /home/student utilitzant l'ordre *du*, així com *sort* per ordenar i *head* per visualitzar únicament alguns dels directoris de la carpeta.

I finalment s'esmenten ordres amb les quals s'obté informació del sistema, com són uname, hostname y host.

- uname

L'ordre uname (*unix name*). Ofereix informació sobre Kernel del sistema, informació sobre el tipus de Linux en el qual estem. Teclegeu en el terminal:

```
$  uname
```

```
 Linux
```

```
$ uname -a
```

```
Linux ubuntum0151 5.15.0-71-generic #78~20.04.1-Ubuntu SMP Wed Apr 19
11:26:48 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
```

- hostname

L'ordre hostname s'utilitza per identificar de forma única un ordinador en una xarxa i s'utilitza per accedir-hi i permetre que altres ordinadors de la xarxa s'hi comuniquin. El hostname pot ser un nom descriptiu o un nom únic assignat pel sistema operatiu de l'ordinador o servidor. Per exemple, el hostname d'un servidor web podria ser *webserver.exemple.com*, on *webserver* és el nom de l'ordinador, *exemple* és el nom del domini i *com* és el domini de nivell superior. Si teclegeu *hostname* al vostre terminal s'obté:

```
$ hostname -f
```

```
ubuntum0151
```

sent ubuntum0151 el hostname de la màquina virtual que heu instal·lat.

- host

L'ordre host determina l'IP d'HOST, i en el context de la informàtica, host es refereix a un servidor o ordinador que allotja i ofereix serveis o recursos a altres ordinadors connectats en una xarxa. Si s'escriu host -a es desplega tota la informació de DNS.

```
$ host ubuntum0151
```

```
ubuntum0151 has address 10.0.2.15
ubuntum0151 has address 172.17.0.1
ubuntum0151 has IPv6 address fe80::1cc6:9af7:249f:6c5e
```

```
 $ host -a ubuntum0151
```

```
host -a ubuntum0151

Trying "ubuntum0151"

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27128

;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0


;; QUESTION SECTION:

;ubuntum0151.                  IN    ANY


Received 29 bytes from 127.0.0.53#53 in 188 ms
```

1. Introducció als entorns de treball UNIX

1.17. Exemple pràctic 1: Analitzant el genoma humà

1.17.1. Introducció

El navegador genòmic UCSC és una eina poderosa que permet als bioinformàtics visualitzar i analitzar dades genòmiques, inclosa l'expressió gènica, les variacions de seqüència i les modificacions epigenètiques. En dominar l'ús d'aquest navegador i altres eines de bioinformàtica, els bioinformàtics poden contribuir a l'avenç de la recerca genòmica, inclòs el desenvolupament de medicina personalitzada i el descobriment de nous objectius terapèutics.

És recomanable que un bioinformàtic sàpiga analitzar el genoma humà, especialment fent servir el navegador genòmic UCSC. Per a això, és habitual emprar arxius que contenen les seqüències d'ADN o de proteïnes en format FASTA, o fitxers de text tabulat amb la ubicació dels elements codificats en el seu interior.

Amb l'objectiu d'aproximar l'estudiant a un escenari amb una situació realista, dins d'un entorn bioinformàtic, se us proposa un exercici d'anàlisi de genoma humà que es divideix en tres tipus d'activitats:

1. Descàrrega i exploració del genoma humà.

2. Descàrrega i anàlisi del catàleg de gens humans.

3. Descàrrega de les eines del navegador genòmic.

Fonamental i imprescindible per adquirir els coneixements necessaris per entendre un projecte bioinformàtic: reproduir cada etapa d'aquest protocol d'anàlisi en el vostre propi terminal.

1. Introducció als entorns de treball UNIX

1.17. Exemple pràctic 1: Analitzant el genoma humà

1.17.2. Descàrrega i exploració del genoma humà

L'organització del genoma d'un organisme es dona en un conjunt de cromosomes. En aquest exemple, es procedeix a descarregar l'anotació sobre els cromosomes del genoma humà en la seva distribució *hg38*. S'adjunta la taula 18 amb els accessos de descàrrega que s'utilitzaran.

Taula 18. Pàgines web del navegador genòmic UCSC.

Accés	Direcció
Pàgina principal servidor UCSC	http://genome.ucsc.edu/
Pàgina descàrregues (genoma data)	https://hgdownload.soe.ucsc.edu/downloads.html
Pàgina llistat espècies	https://hgdownload.soe.ucsc.edu/goldenPath/currentGenomes/
Pàgina accés espècie *human*	https://hgdownload.soe.ucsc.edu/downloads.html#human
Pàgina accés espècie *human*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/
Pàgina *Sequence data by Chromosome*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes/
Pàgina accés *bigZips*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/

Font: elaboració pròpia.

Primerament, s'accedeix a la pàgina de descàrregues (en anglès, *downloads*) del navegador genòmic UCSC (https://hgdownload.soe.ucsc.edu/downloads.html).

El contingut d'aquesta pàgina ens mostra el llistat de genomes disponibles organitzat per espècies. A data de 26 d'abril de 2023 hi ha informació sobre 108 espècies.

En fer ús de l'enllaç *Human*, entrem a la secció dedicada al genoma humà. És important destacar que la informació corresponent a cada genoma s'actualitza amb certa freqüència, per la qual cosa cada millora substancial compta amb un codi de versió propi. En aquest cas, treballarem amb la distribució coneguda com *a hg38*, la qual és la més recent al moment de la redacció d'aquests materials.

Si des de la pàgina d'accés a l'espècie humana accediu a l'enllaç associat a *Sequence data by chromosome* (https://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes) accedireu al llistat dels fitxers comprimits FASTA de cadascun dels cromosomes (chr*.fa.gz), a la seqüències *random*, que són seqüències no col·locades en els anteriors cromosomes de referència (chr*_random), i a les seqüències *chrUn_**, que són seqüències no localitzades en les quals el cromosoma de referència no ha estat determinat. En la mateixa data esmentada anteriorment, hi ha 456 seqüències FASTA associades a diferents cromosomes.

Si des d'aquesta última localització s'accedeix al *Parent Directory* (https://hgdownload.soe.ucsc.edu/goldenPath/hg38/), trobareu el directori anomenat *bigZips* (https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/), que és un altre repositori de fitxers, amb diferents formats, associat al genoma humà. Tots els arxius estan comprimits i empaquetats per reduir el temps de transmissió.

El fitxer hg38.chromFa.tar.gz conté la seqüència original dels cromosomes separats en arxius independents. Cal descarregar aquest fitxer i es farà amb l'ordre *wget*, però només has de descarregar el fitxer si tens més de 5 Gb disponibles al disc dur. Si tens menys de 5 Gb lliures, descarrega la seqüència FASTA del cromosoma 7 des del directori https://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes/

L'ordre `df` és l'ordre que s'utilitza per esbrinar l'espai en disc

```
$ df -h
```

```
Filesystem      Size      Used     Available    Use%     Mounted on

udev            959M      0        959M         0%       /dev

tmpfs           199M      1,4M     197M         1%       /run

/dev/sda5       20G       16G      2,9G         85%      /

tmpfs           991M      0        991M         0%       /dev/shm

tmpfs           5,0M      4,0K     5,0M         1%       /run/lock

tmpfs           991M      0        991M         0%       /sys/fs/cgroup

/dev/loop1      64M       64M      0            100%     /snap/core20/1852
```

En la màquina en la qual s'està treballant només hi ha disponibles 2,9 G d'espai (columna *Available*), per la qual cosa en aquest cas només es descarrega la seqüència FASTA del cromosoma 7. Si hi hagués espai en el disc dur per descarregar el fitxer amb tota la informació, el procediment seria el següent:

```
$ wget
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.chromFa.tar.gz
```

```
--2023-04-26 13:22:55--
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.chromFa.tar.gz

S'està resolent hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)...
128.114.119.163

S'està connectant a hgdownload.soe.ucsc.edu
(hgdownload.soe.ucsc.edu)|128.114.119.163|:443... conectat.

HTTP: s'ha enviat la petició, s'està esperant una resposta... 200 OK

Mida: 983726049 (938M) [application/x-gzip]

S'està desant a: «hg38.chromFa.tar.gz»

hg38.chromFa.tar.gz    100%[===================================>] 938,15M
7,82MB/s    in 2m 25s

2023-04-26 13:25:22 (6,47 MB/s) - s'ha desat «hg38.chromFa.tar.gz»
[983726049/983726049]
```

Un cop el fitxer està descarregat a la màquina Gnu/Linux amb la qual es treballi s'ha de desempaquetar i descomprimir el fitxer amb l'objectiu de visualitzar-ne el contingut.

```
$ ls -alh hg38.chromFa.tar.gz
```

```
-rw-rw-r-- 1 student student 939M de gen.  24  2014 hg38.chromFa.tar.gz
```

```
$ tar -vzxf hg38.chromFa.tar.gz
```

```
./chroms/

./chroms/chr1.fa

./chroms/chr10.fa

./chroms/chr11.fa

./chroms/chr11_KI270721v1_random.fa

./chroms/chr12.fa

./chroms/chr13.fa

…
```

Tot i que la qualitat de la seqüència del genoma humà és acceptable, encara es troba en fase de millora. A causa d'això, és comú trobar nombrosos arxius que contenen fragments o variants que encara estan en discussió i que no necessàriament corresponen a un cromosoma complet. És possible visualitzar el primer cromosoma al terminal; no obstant això, en algunes parts del cromosoma, com l'inici, la seqüència de nucleòtids és desconeguda i es denota amb el caràcter *N*. A més, per indicar la presència d'elements codificats en la seqüència, es pot utilitzar una combinació de lletres majúscules i minúscules.

```
$ more chr7.fa
```

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

…

…

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

GAATTCTACATTAGAAAAATAAACCATAGCCTCATCACAGGCACTTAAAT

ACACTGAAGCTGCCAAAACAATCTATCGTTTTGCCTACGTACTTATCAAC

TTCCTCATAGCAAACTGGGAGAAAAAAGCAATGGAATGAATAAAATGATA

GCCACAAAAATCAAGGTGGGAGAAATACTTATTATATGTCCATAAAAAAT

TTTAATTAATGCAAAGTATTAACACCAATGATTGCAGTAATACAGATCTT

ACAAATGATAGTTTTAGTCTGAACAGGACTATCCAAAAGTTAATTTTCTA

TAGTAACAGTTTTTAAATAAAATATCAATTCCTGAAACACATAAAATGGT

CCATGAGTATACAACGAGTGAAAAAAAACAAATTCAGAGCAAAGATAAAT
```

TAAGAAGTATCTAATATTCAAACATAGTCAAAGAGAGGGAGATTTCTGGA

TAATCACTTAAGCCCATGGTTAAACATAAATGCAAATATGTTAATGTTTA

CTGAATAACTTATCTGTGCCAAGTGGTGTATTAATGATTCATTTTTATTT

TTCACTAAATCTTTTCTCTAAAGTTGGTGTAGCCTGCAACTAAATGCAAG

AAATCTGACCTAGGACCTGCACTTCTTACCATTTTGCTCATATTTATTCC

CTGTGCATTTTTGTAACATGTATATGTTATATATATAGAAAGAGAGAGAG

GCAGAGATGGAAAGTAATTTATGGAGTTTGATGTTATGTCAGGGTAATTA

CATGATTATATAATTAACAGGTTTCTTTTTAAATCAGCTATATCAATAGA

AAAATAAATGTAGGAATCAAGAGACTCATTCTGTCCATCTGTGATAGTTC

CATCATGATACTGCATTGTCAAGTCATTGCTCCAAAAATATGGTTTAGCT

CAACactgagtgactataggaaaccagaaaccaggctgggcgctaaagat

gcaaagatgaatgagacatcatctctgccgtccaaaagcttactgtctag

tgggagagttacacacgtaaggacagtaatctaataagagctaataagtg

aaaactaagataaattaataatacaagattacagggaaggtttccaaagt

caatgaggcctcaaatgaatcttgaaagtgtgcaaggattaaccaaatga

1. Introducció als entorns de treball UNIX

1.17. Exemple pràctic 1: Analitzant el genoma humà

1.17.3. Anàlisi dels gens humans

L'anàlisi del catàleg de gens d'una espècie és un exemple perfecte de la utilitat de les ordres del terminal en aquest capítol. Un gen és una seqüència d'ADN que conté informació per crear una molècula biològica. Els gens dels organismes eucariotes es componen d'exons. Molts gens humans tenen combinacions plausibles d'exons, el que resulta en transcrits alternatius.

Per codificar la ubicació dels gens, s'utilitzen fitxers tabulats que contenen informació com la localització i les característiques del transcrit del gen, com el nombre d'exons i les coordenades exactes.

En aquest exercici, s'utilitza l'anotació del consorci *RefSeq* per obtenir informació sobre el genoma humà, en un primer moment, però també s'utilitzarà informació del genoma de ratolí (*mouse*). La informació es troba en un fitxer anomenat *refGene.txt* i es pot obtenir a la pàgina web del navegador genòmic d'UCSC mitjançant l'enllaç *Annotation*. La direcció de descàrregues es mostra a la taula 19.

Taula 19. Pàgines de descàrrega dels fitxers refGene.txt en funció de l'espècie.

Accés	Direcció
Pàgina descàrrega *human*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/
Pàgina descàrrega *mouse*	https://hgdownload.soe.ucsc.edu/goldenPath/mm39/database/

Font: elaboració pròpia.

Les anotacions, en el servidor d'UCSC, solen representar-se gràficament en el navegador genòmic en forma de pista. El navegador genòmic està gestionat internament per l'administrador de bases de dades relacionals de MySQL. En conseqüència, en aquesta pàgina web trobarem dos tipus de fitxer per a cada pista d'anotacions d'UCSC. El primer fitxer, l'extensió del qual serà del tipus *sql*, conté una especificació genèrica dels atributs de les anotacions. Aquest arxiu és necessari per crear una taula buida en una base de dades relacional. El segon fitxer, que estarà comprimit i posseirà l'extensió *txt*, conté pròpiament la informació de cada anotació de forma tabulada. En accedir a la pàgina de descàrrega es visualitza la informació següent:

```
 This directory contains a dump of the UCSC genome annotation database for
the

     Dec. 2013 (GRCh38/hg38) assembly of the human genome

     (hg38, GRCh38 Genome Reference Consortium Human Reference 38
(GCA_000001405.15))




       affyGnf1h.sql                                    2015-05-11 01:50  2.1K

       affyGnf1h.txt.gz                                 2015-05-11 01:50  596K

       ...

       refGene.sql                                       2020-08-17 18:56
1.9K

       refGene.txt.gz                                   2020-08-17 18:56  8.3M
```

```
        …

        xenoRefSeqAli.sql                          2020-08-17 19:17  2.1K

        xenoRefSeqAli.txt.gz                       2020-08-17 19:17   17M
```

Ara procedim a descarregar el fitxer *txt* associat a la pista *refGene*, que conté el catàleg de gens humans anotats pel consorci *RefSeq*. Per a això, s'utilitza l'ordre *wget* novament per transferir el fitxer al terminal.

```
 $ wget
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz
```

```
 --2023-04-26 16:35:43--
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz

 S'està resolent hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)...
128.114.119.163

 S'està connectant a hgdownload.soe.ucsc.edu
(hgdownload.soe.ucsc.edu)|128.114.119.163|:443... conectat.

 HTTP: s'ha enviat la petició, s'està esperant una resposta... 200 OK

 Mida: 8668756 (8,3M) [application/x-gzip]

 S'està desant a: «refGene.txt.gz»

 refGene.txt.gz                                      100%[================>]
8,27M  2,22MB/s    in 3,7s

 2023-04-26 16:35:49 (2,22 MB/s) - s'ha desat «refGene.txt.gz»
[8668756/8668756]
```

A continuació, et convidem a revisar el contingut del primer fitxer: refGene.sql. ja que és útil per conèixer les característiques dels gens anotats a cada columna del fitxer. Els atributs que s'utilitzen amb més freqüència són els següents: *name* (codi del transcrit), *chrom* (cromosoma), *strand* (cadena), *txStart* i *txEnd* (coordenades d'inici i final), *exonCount* (nombre d'exons) i *name2* (nom del gen). És important no confondre els camps *name* i *name2*: un gen pot tenir diversos transcrits, però un transcrit sols pot pertànyer a un gen. Aquests paràmetres es troben a la capçalera del fitxer descarregat.

Visualitzarem el contingut del fitxer refGene.txt. Aquest arxiu conté informació sobre el catàleg complet de gens anotats en el genoma humà. Cada línia representa un transcrit d'un gen determinat. En el cas que un gen tingui diversos transcrits, cadascun es codifica en línies separades, cadascuna amb el seu propi codi i les seves coordenades corresponents. En totes les línies, els atributs de cada dia estan separats pel caràcter tabulador o «\t». Abans de poder visualitzar el contingut de l'arxiu, l'hem de descomprimir utilitzant l'ordre *gzip*.

Descompressió del fitxer amb *gzip*

```
 $ gzip -d refGene.txt.gz
```

Visualització de les primeres cinc línies amb l'ordre head

```
 $ head -5 refGene.txt
```

```
 585  NR_024540  chr1 -    14361 29370 29370 29370 11
14361,14969,15795,16606,16857,17232,17605,17914,18267,24737,29320,
14829,15038,15947,16765,17055,17368,17742,18061,18366,24891,29370,    0
WASH7P    unk  unk  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,

 585  NR_106918  chr1 -    17368 17436 17436 17436 1    17368,    17436,
0    MIR6859-1  unk  unk  -1,

 585  NR_107062  chr1 -    17368 17436 17436 17436 1    17368,    17436,
0    MIR6859-2  unk  unk  -1,

 585  NR_107063  chr1 -    17368 17436 17436 17436 1    17368,    17436,
0    MIR6859-3  unk  unk  -1,

 585  NR_128720  chr1 -    17368 17436 17436 17436 1    17368,    17436,
0    MIR6859-4  unk  unk  -1,
```

Tingueu en compte que les anotacions d'un genoma solen actualitzar-se freqüentment. Per aquest motiu, les dades mostrades en aquest tutorial poden variar lleugerament amb el pas del temps en comparació amb una nova descàrrega del mateix fitxer.

A causa de la gran quantitat d'informació que conté aquest arxiu són difícils de manejar. A continuació, es decideix treballar amb uns determinats camps i s'utilitzarà l'ordre *gawk* per extreure únicament els camps necessaris d'aquest exercici. En particular, trobem interessants els següents atributs: nom del gen, identificador del transcrit, cromosoma, cadena, coordenades i nombre d'exons. Per evitar haver d'executar l'ordre prèvia cada vegada que es necessitin aquestes dades, es redirecciona la sortida cap a un arxiu refgene-select.txt. Una vegada obtingut, es compten el nombre total de transcrits humans.

Analitzem el fitxer *refGene.sql* per determinar la posició de cada columna

```
$ gawk '{print $13,$2,$3,$4,$5,$6,$9;}' refGene.txt > refgene-select.txt
```

Es visualitzen les últimes 5 línies

```
$ tail -5 refgene-select.txt
```

```
KIAA0825   NM_001385728    chr5 -   94519215   94618604   6

KIAA0825   NM_001385729    chr5 -   94519215   94618604   6

KIAA0825   NR_169752      chr5 -    94519215   94618604   4

KIAA0825   NR_169753       chr5 -    94519215   94618604   6

KIAA0825   NR_169754       chr5 -   94519215   94618604   6
```

El nombre de transcrits és el nombre de línies del fitxer

```
$ wc -l refgene-select.txt
```

```
88819 refgene-select.txt
```

Continuem amb diferents anàlisis,

Es demana el nombre de transcrits genètics distribuïts en la cadena positiva

```
$ gawk '{if ($4 == "+") print $0;}' refgene-select.txt | wc -l
```

```
45121
```

Es demana el nombre de transcrits del cromosoma 21

```
$ grep chr21 refgene-select.txt | wc -l
```

```
1121
```

Es mostren els primers set transcrits després d'ordenar per nom del gen

```
$ sort refgene-select.txt | head -7
```

```
A1BG NM_130786      chr19     -    58345182   58353492   8

A1BG-AS1 NR_015380   chr19     +    58351969   58355183   4

A1CF NM_001198818    chr10     -    50799408   50885675   14

A1CF NM_001198819    chr10     -    50799408   50885675   15

A1CF NM_001198820    chr10     -    50799408   50885675   14

A1CF NM_001370130    chr10     -    50799408   50885627   12

A1CF NM_001370131    chr10     -    50799408   50885627   12
```

Es mostren els primers sis transcrits després d'ordenar pel nombre de gens que conté cada transcrits

```
$ sort -rnk 7 refgene-select.txt | head -6
```

```
TTN NM_001267550     chr2 -   178525990 178807423 363

TTN NM_001256850     chr2 -    178525990 178807423 313

TTN NM_133378   chr2 -    178525990 178807423 312

TTN NM_133437   chr2 -    178525990 178807423 192

TTN NM_133432   chr2 -    178525990 178807423 192

TTN NM_003319   chr2 -    178525990 178807423 191
```

El fitxer *refgene-select.txt* conté el llistat dels transcrits humans. Com que un gen pot donar lloc a diversos transcrits alternatius, podem comptar quants gens té el genoma humà i esbrinar quants d'ells posseeixen un major nombre de transcrits. Per aconseguir-ho, ordenem els noms dels gens i introduïm diferents variants de l'ordre `uniq` per agrupar-los. D'altra banda, la millor forma d'estudiar el comportament d'una línia d'ordres és la desconstrucció: eliminant les últimes instruccions es pot mostrar el resultat parcial de l'execució en pantalla.

Ordena la columna pel nom dels gens

```
$ gawk '{print $1;}' refgene-select.txt | sort | more
```

```
A1BG

A1BG-AS1

A1CF

A1CF

A1CF

A1CF

A1CF

…
```

Ordena la columna pel nom dels gens i que siguin valors únics

```
$ gawk '{print $1;}' refgene-select.txt | sort | uniq | more
```

```
A1BG

A1BG-AS1

A1CF

A2M

A2M-AS1

A2ML1

…
```

Determina el nombre de gens únics en el fitxer

```
$ gawk '{print $1;}' refgene-select.txt | sort | uniq | wc -l
```

```
28307
```

Determina quants transcrits hi ha per a cadascun dels gens

```
$ gawk '{print $1;}' refgene-select.txt | sort | uniq -c | more
```

```
1 A1BG

1 A1BG-AS1

8 A1CF

4 A2M
```

```
  3 A2M-AS1

  2 A2ML1

  1 A2MP1

  …
```

Determina quants transcrits hi ha per a cadascun dels gens i ordena'ls pel nombre d'exons, de major a menor

```
$ gawk '{print $1;}' refgene-select.txt | sort | uniq -c | sort -rn
```

```
260 KIR2DS2

215 LOC101928804

177 KIR2DS4

144 MAP4

144 KIR3DS1

129 KIR2DL

…
```

Una altra operació molt freqüent consisteix en el càlcul de valors mitjà. En el següent exemple, l'estudiant calcularà la mitjana d'exons per transcrit i la longitud mitjana d'aquests. El funcionament de gawk és similar en tots dos casos, ja que treballa amb la variable *t* com a comptador que acumula la suma total. La divisió pel nombre de línies visitades (NR), un cop es completa la lectura de l'arxiu, genera el valor mitjà en cada cas.

Càlcul de mitjana d'exons per transcrit

```
$ gawk 'BEGIN{t=0;}{t=t+$7;}END{print t/NR;}' refgene-select.txt
```

```
10.0961
```

Càlcul de la longitud mitjana dels transcrits

```
$ gawk 'BEGIN{t=0;}{t=t+$6-$5+1;}END{print t/NR;}' refgene-select.txt
```

```
64883
```

Per il·lustrar la utilitat d'associar dos fitxers de text tabulat, es combinarà el catàleg de gens humans introduït fins ara (arxiu *refGene.txt* per *a H. sapiens es reanomena com a refGene-human.txt*) amb el catàleg equivalent del genoma del ratolí (arxiu *refGene.txt* per a *M. musculus* i reanomenat com a *refGene-mouse.txt* després d'haver-lo descarregat del servidor genòmic d'UCSC).

```
$ gzip -d refGene.txt.gz
```

```
 $ mv refGene.txt refGene-human.txt
```

```
$ wget
https://hgdownload.soe.ucsc.edu/goldenPath/mm39/database/refGene.txt
```

Es reanomena el fitxer del genoma *M.musculus*

```
$ mv refGene.txt refGene-mouse.txt
```

Per a cadascun dels fitxers, se seleccionen les columnes nom del gen i cromosoma i posteriorment se'n visualitzen els 5 primers

```
$ gawk '{print $13, $3;}' refGene-human.txt | sort | uniq > refgene-
select-human.txt
```

```
$ head -5 refgene-select-human.txt
```

```
A1BG chr19

A1BG-AS1 chr19

A1CF chr10

A2M chr12

A2M-AS1 chr12
```

Per al fitxer genoma de ratolí

```
$ gawk '{print $13, $3;}' refGene-mouse.txt | sort | uniq > refgene-
select-mouse.txt
```

```
$ head -5 refgene-select-mouse.txt
```

```
0610005C13Rik chr7

0610009B22Rik chr11

0610009E02Rik chr2

0610009L18Rik chr11

0610010F05Rik chr11
```

Amb aquesta transformació prèvia del fitxer, s'han preparat els fitxers per utilitzar l'ordre `join`: a continuació, es mostra com associar els gens que tenen el mateix nom en ambdues espècies. A l'ordre `join` s'hi afegeix l'opció *-i*, perquè s'ignoren les diferències de majúscules/minúscules. Cada línia del resultat final conté el nom del gen i la seva ubicació en ambdós genomes.

```
$ join -i refgene-select-human.txt refgene-select-human.txt > refgene-
comun.txt
```

```
$ head -5 refgene-comun.txt
```

```
A1BG chr19 chr19
```

```
A1BG-AS1 chr19 chr19

A1CF chr10 chr10

A2M chr12 chr12

A2M-AS1 chr12 chr12
```

A partir d'aquesta anàlisi preliminar del catàleg de gens humans, es poden portar a la pràctica múltiples variants de les ordres que s'han mostrat aquí. Per exemple, és possible afegir més atributs, més genomes o més fitxers amb altres propietats per ampliar aquesta exploració. Per tant, us animem a experimentar amb cada bloc d'ordres utilitzat durant aquest exercici. Amb aquesta aplicació pràctica, es demostra la validesa del terminal de Gnu/Linux per analitzar eficientment dades biològiques. En futurs apartats, s'introduiran sistemes més complexos per gestionar grans conjunts de dades, de manera que l'usuari podrà reproduir el mateix cas pràctic utilitzant aquestes tècniques.

1. Introducció als entorns de treball UNIX

1.17. Exemple pràctic 1: Analitzant el genoma humà

1.17.4. Descàrrega de les eines d'UCSC

El navegador genòmic d'UCSC ofereix de manera gratuïta les seqüències dels genomes de múltiples espècies i les pistes d'anotacions associades a cada versió. A més, aquest portal web també proporciona accés lliure a un conjunt d'eines dissenyades específicament per analitzar les dades genòmiques. Aquests programes funcionen en línia d'ordres i es distribueixen llestos per funcionar en diverses plataformes de la família UNIX. Per mostrar el funcionament de l'ordre `rsync` en la descàrrega d'un directori web complet, haureu d'obtenir una còpia completa de les utilitats d'UCSC. A la taula 20 es detallen els accessos a les adreces de descàrrega.

Taula 20. Accés a les adreces de descàrrega d'utilitats d'UCSC.

Accés	Direcció
Pàgina informació	https://hgdownload.soe.ucsc.edu/downloads.html#utilities_downloads
Pàgina descarrega *utilities*	https://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/

Font: elaboració pròpia.

Per a això s'accedeix novament a la pàgina de descàrregues del navegador. Un cop allà, busca la secció *Utility Tools and Software downloads* (en català, *Eines d'utilitat i descàrregues de* software). En aquesta secció expliquen com descarregar-lo.

Per descarregar totes les utilitats de línia d'ordres en un directori amb els *bits* de permisos correctes, utilitza l'ordre següent:

```
$ pwd
```

```
/home/student
```

```
 $ mkdir UCSC-utilities
```

```
 $ cd UCSC-utilities
```

```
$ rsync -aP hgdownload.soe.ucsc.edu::genome/admin/exe/linux.x86_64/ ./
```

Després de més de 5 minuts s'haurà descarregat el 5 % de les utilitats. Aquest procés és lent i ens adonem que ha d'ocupar molt espai, més espai del que hi ha lliure en el disc dur del nostre ordinador. Mata el procés prement les tecles Ctrl + X i es pararà la descàrrega. És una prova per veure que funciona l'ordre `rsync`. Esborra tot el que has generat.

```
$ cd ..
```

```
$ pwd
```

```
/home/student
```

```
$ rm -r UCSC-utilities
```

Una altra opció és descarregar únicament les utilitats que es vulguin utilitzar, per exemple, es descarrega la utilitat *liftOver*.

```
$ pwd
```

```
/home/student
```

```
$ wget https://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/liftOver
```

```
 $ chmod +x /home/student/liftover/liftOver
```

```
# Executa el programa sense arguments per veure un missatge d'ús
```

```
$ cd liftOver
```

```
$ . /liftOver
```

Resum

«Introducció als entorns de treball GNU/Linux» és un capítol que ofereix una guia detallada per treballar en sistemes operatius basats en la filosofia GNU/Linux. El llibre comença amb una introducció a la història i l'evolució de GNU/Linux, i després s'endinsa en els conceptes bàsics del sistema operatiu. Al llarg del capítol, es cobreixen temes com l'estructura d'arxius i directoris, les ordres i eines bàsiques de GNU/Linux, la gestió de processos i recursos, l'automatització de tasques i la programació de *scripts* en la *shell*.

A més, el capítol aborda temes avançats, com l'administració d'usuaris i permisos. Els estudiants aprenen a utilitzar eines com *vi* i *sed* per a l'edició d'arxius de text, i a gestionar el sistema operatiu a través de la línia d'ordres.

En finalitzar el capítol, l'estudiant pot manipular molts tipus d'informació de forma senzilla i eficient. Juntament amb aquest entorn de treball, l'estudiant ha descobert que es té accés lliure a una gran quantitat de dades biològiques que, en emmagatzemar localment en la pròpia màquina, s'accelera el processament i es redueix el temps de càlcul. En resum, aquest nucli d'eines conforma una excel·lent aproximació per extreure nou coneixement a partir de la informació biològica disponible.

Activitats

1. Realitzeu la instal·lació de la versió més recent d'Ubuntu MATE dins d'una màquina virtual d'Oracle VirtualBox. Per a això, primer hauríeu d'obtenir una imatge ISO d'Ubuntu MATE. Posteriorment, cal crear una màquina virtual buida, inserir la imatge ISO d'Ubuntu i executar la instal·lació. Podeu emprar un administrador de la gestió de paquets de *software* per configurar el sistema final resultant.

2. Esbrineu les funcions que realitza l'ordre `fold` del terminal. Analitzeu les opcions disponibles per a aquesta ordre. Després, dissenyeu un petit *script* que combini *gawk* amb l'ordre `fold` per calcular la freqüència d'aparició absoluta i relativa de cada classe de nucleòtid en una seqüència genòmica emmagatzemada en un fitxer de text guardat en format FASTA. Avalueu el funcionament del vostre protocol sobre diverses seqüències d'ADN.

3. Esbrineu les funcions realitzades per *Bioawk*. Determineu quina és l'ordre origen d'aquesta extensió, els formats de dades biològiques que suporta i si és possible treballar amb fitxers comprimits amb *gzip* o amb altres ordres compressores.

4. Estudieu l'ordre `sedl`. Per provar la seva eficàcia, graveu un full d'estil de *MicrosoftExcel* en format text (seleccioneu el tabulador com a separador de camps). Un cop a Gnu/Linux, verifiqueu amb l'ordre `od` que aquest format propietari efectivament introdueix com a salt de línia els caràcters `\r` i `\n`. Finalment, empreu l'ordre per únicament mantenir el caràcter `\n` (el terminal treballa en aquest format).

5. Dissenyeu un *script* en el terminal que us permeti realitzar l'anàlisi completa d'una sèrie de fitxers de la classe refGene.txt emmagatzemats en un directori. Cada fitxer ha de contenir en el seu propi nom l'organisme al qual pertany per evitar noms de fitxers duplicats (p.e. `refGene_human.txt`). Per a cada genoma podeu realitzar les mateixes preguntes mostrades en el cas d'estudi dels materials teòrics.

6. A continuació, us subministrem un arxiu FASTA i heu de contestar a les següents preguntes. El separador entre les dues columnes és el `\t`

```
$ cat hib.fasta
```

```
HiB_C1    TGTTTGTTGTCACTGACTGATGTTGTGGTCTGG

HiB_C2    TATATATTACTT

HiB_C3    TATATATAACTTATA

HiB_C4    TATATATAACTTATA

HiB_C5    TATATATTACTT
```

- Imprimiu la línia que coincideix amb el patró `HiB_C4`.
- Imprimiu la columna 1, un punt i coma, i la columna 2.
- Utilitzeu el concepte d'un operador condicional en la declaració d'impressió de la forma *print* **CONDITION ? PRINT_IF_TRUE_TEXT : PRINT_IF_FALSE_TEXT** per identificar les seqüències amb longituds > 14.
- Intenteu realitzar el següent exercici. Què és el que passa?

Es pot fer servir e1 després de l'últim bloc {} per imprimir-ho tot (1 és una notació abreujada per a {print $0} que es converteix en {print}, ja que, sense cap argument, *print* imprimirà $0 per defecte), i dins d'aquest bloc, podem canviar $0, per exemple, per assignar el primer camp a $0 per a la segona línia (NR==2).

- Utilitzeu l'ordre `getline` per carregar el contingut d'un altre arxiu a més del que esteu llegint. Intenteu, amb el bucle *while*, carregar cada línia del fitxer *fasta* en una variable: la *b*, per exemple.

7. Contesteu les següents preguntes després de descarregar-vos el següent fitxer: https://ftp.ncbi.nlm.nih.gov/genomes/ASSEMBLY_REPORTS/assembly_summary_refseq.txt

- Els valors únics de la variable categòrica *assembly_level* es troben indicats a la columna #12, la qual mostra l'estat de l'*ensamble*. Quins són?
- Determineu el nombre de genomes per espècie, que es troba a la columna #8. Després, mostreu únicament les 10 espècies amb la major quantitat de genomes seqüenciats.
- Quants genomes complets hi ha del gènere *Mycobacterium*?
- Compteu els genomes de *Salmonella*, *Pseudomonas* i *Acinetobacter* (per gènere) i presenteu la llista ordenada per nombre decreixent de genomes.
- Determineu la longitud d'una cadena. Per què aquestes dues ordres donen diferent informació?

```
$ echo 'atattttGAATTtattGAATCAGGACC' | wc -c
```

```
$ echo 'atattttGAATTtattGAATCAGGACC' | awk 'END{print "El
oligonucleótido", $0, "tiene" lenght($0), "nucleótidos de longitud"}'
```

8. Trieu diversos fitxers que continguin seqüències FASTA.

- El nombre de seqüències per a un fitxer (*awk*, autoincrement).
- Amb un bucle determineu el nombre de seqüències per a tots els fitxers (*for and*, *awk*, autoincrement).

Exercicis d'autoavaluació

1. Enumereu els quatre components bàsics d'una computadora.
2. Enumereu les etapes principals de la generació d'un fitxer executable.
3. Enumereu les etapes en la vida d'un procés.
4. Enumereu les quatre operacions bàsiques amb fitxers.
5. Citeu com es codifiquen el directori arrel, el directori anterior i l'actual.
6. Enumereu els tres dominis dels usuaris.
7. Descriviu les classes de permisos que es poden assignar a un fitxer.
8. Per què és utilitzada l'ordre `man` al terminal?
9. Enumereu les ordres del terminal per navegar pel sistema de fitxers.
10. Enumereu diverses ordres del terminal per gestionar el sistema de fitxers.
11. Citeu l'ordre per modificar els permisos sobre un fitxer o un directori.
12. Enumereu algunes ordres per accedir al contingut d'un fitxer.
13. Expliqueu com accedir a fitxers prèviament comprimits des del terminal.
14. Expliqueu com recuperar el llistat de les ordres usades en una sessió.
15. Enumereu les ordres adequades per ordenar i combinar fitxers.
16. Descriviu l'ordre emprada habitualment per buscar patrons de text.
17. Citeu la diferència entre `|` o `>` en la comunicació entre processos.
18. Discutiu la diferència entre els blocs BEGIN i END en el llenguatge de programació *gawk*.
19. Expliqueu les següents variables de *gawk*: `$1`, `$0`, `NR`, `NF`, `OFS` i `FS`.
20. Per què són enormement útils els protocols de tasques automatitzades?

Solucionari

1. El processador, la seqüència, els perifèrics i el bus de comunicacions.
2. Programació, compilació, enllaç i execució.
3. En execució, en espera per executar-se, bloquejat.
4. Obrir, tancar, llegir, escriure.
5. El directori és «`/`», el directori anterior és «`..`» i l'actual és « .».
6. Existeixen tres dominis: usuari, grup de treball i la resta d'usuaris.
7. Existeixen tres permisos: lectura, escriptura i execució.
8. Proporciona accés al manual del sistema.
9. Les ordres *pwd*, *ls*, *cd*.
10. Les ordres *cp*, *rm*, *mv* i *mkdir*.
11. És l'ordre *chmod*.
12. Les ordres *more*, *head*, *cat* i *tail*.
13. Fent servir les ordres *gzip* i *tar*.
14. L'ordre *history*.
15. Les ordres *sort*, *uniq* i *join*.
16. L'ordre *grep* realitza la recerca de patrons de text.
17. L'operant | envia la informació a un altre procés, mentre que l'operant > envia la informació a un fitxer, necessàriament.
18. El bloc BEGIN executa aquestes instruccions abans de processar el fitxer, el bloc END executa les instruccions precisament després de finalitzar aquest.
19. La variable `$1` es refereix al primer atribut de cada registre o línia. La variable `$0` conté la línia actual en curs. La variable `NR` emmagatzema el nombre de registres llegits, mentre que la variable `NF` compta el nombre de columnes present en cada línia. `OFS` representa el caràcter que s'emprarà per separar camps a la sortida, mentre que `FS` realitza la mateixa funció en l'adquisició de les dades d'entrada.
20. Perquè permeten dur a terme un número (pràcticament) infinit de vegades el mateix conjunt de tasques sobre múltiples grups de dades sense assistència de l'usuari.

Bibliografia

Advanced Bash-Scripting Guide – LDP, Mendel Cooper. https://tldp.org/LDP/abs/abs-guide.pdf

Advanced Bash-Scripting, Michael F. Herbst, Uni Heidelberg. https://michael-herbst.com/teaching/advanced-bash-scripting-2017/advanced-bash-scripting-2017/notes.pdf

Alfred V. Aho, Brian W. Kernighan and i Peter J. Weinberger (1988). *The AWK Programming Language.* Addison Wesley. ISBN: 020107981X.

Andrew S. Tanenbaum (2007). *Modern operating systems* (3rd Edition). Prentice-Hall. ISBN: 0136006639. Andrew S. Tanenbaum (1995). Distributed operating systems. Prentice-Hall. ISBN: 0132199084.

Brian W. Kernighan and i D. M. Ritchie (1988). *C Programming Language* (2nd Edition). Prentice Hall. ISBN: 0131103628.

Brian W. Kernighan and i Rob Pike (1984). *Unix Programming Environment*. Prentice Hall. ISBN: 013937681X.

Cameron Newham (2005). *Learning the bash Shell* (3rd Edition). O'Reilly Media. ISBN: 0-596-00965-8.

Christopher Negus (2015). *Linux BIBLE. The comprehensive, tutorial resource* (9th Edition). Wiley. ISBN: 1118999878

Debra Cameron, James Elliott, Marc Loy, Eric Raymond and i Bill Rosenblatt (2004). *Learning GNU Emacs* (3rd Edition). O'Reilly Media. ISBN: 0-596-00648-9

John L. Hennessy and i David A. Patterson (2002). *Computer Architecture: A Quantitative Approach* (3rd Edition). Morgan Kaufmann. ISBN: 1558605967.

Linux Bash Shell Scripting Tutorial. http://bash.cyberciti.biz/guide/Main_Page

Steven Haddock i Casey Dunn (2011). *Practical computing for biologists. Sinauer Associates*. ISBN: 978-0-87893-391-4.

The GNU Awk User's Guide. https://www.gnu.org/software/gawk/manual/gawk.html

The GNU Bash Reference Manual. https://www.gnu.org/software/bash/manual/bash.html

The GNU/Linux Documentation Project – LDP. https://tldp.org/