Eines informàtiques per a la bioinformàtica

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

Mòduls didàctics

Introducció

En els últims anys hem assistit a una revolució tecnològica sense parangó. Amb l'aparició de les primeres computadores, la incessant miniaturització dels seus components i la connexió global d'aquestes mitjançant una xarxa universal, els éssers humans hem construït una visió radicalment nova del món que ens envolta. Actualment és difícil concebre la recerca en qualsevol àrea de coneixement sense el concurs de la informàtica. La biologia molecular no n'és una excepció. Potser no som conscients de l'envergadura del salt conceptual experimentat en aquest camp. Arran del descobriment, a mitjan segle passat, de l'estructura de la molècula d'ADN i de l'obtenció de la seva seqüència a l'inici del segle actual, comencem a estar en condicions d'abordar sincerament el problema de la constitució de la vida mateixa. Ara gaudim de l'inigualable avantatge de conèixer, per a nombroses espècies, el catàleg de gens i proteïnes que governen la majoria de les respostes cel·lulars durant el seu desenvolupament o en resposta a les condicions canviants del nostre entorn.

Mentre la millora de la seqüenciació massiva permet reconstruir la cartografia dels genomes, la potenciació de les telecomunicacions ofereix un entorn ideal per posar aquestes dades a disposició de la comunitat científica. No obstant això, tota aquesta fantàstica amalgama d'informacions ha de ser gestionada de manera eficient. Evidentment, la bioinformàtica té un paper crucial en la realització rutinària d'aquest tipus d'anàlisi en qualsevol laboratori de genètica. Per tant, és fonamental formar nou personal investigador en les tècniques bioinformàtiques bàsiques amb l'objectiu que adquireixin el vocabulari específic d'aquest camp, s'agilitin les seves futures col·laboracions científiques amb especialistes d'aquesta disciplina i comprenguin l'abast d'aquesta tecnologia per integrar-la dins dels seus propis projectes de recerca.

Aquest mòdul se centra en les aplicacions computacionals més utilitzades pels bioinformàtics per processar informació genòmica. La família de sistemes operatius Gnu/Linux és la plataforma habitual en aquest tipus de laboratoris. En primer lloc, es presentaran conceptes bàsics relacionats amb els sistemes operatius. A continuació, s'abordarà el maneig del terminal de Gnu/Linux, una eina de treball fonamental per als bioinformàtics. S'aprendran els comandaments bàsics i com combinar-los per generar comandaments més potents, creant així protocols complets de treball. Finalment, s'explicarà com obtenir fàcilment còpies de grans conjunts de dades biològiques de referència, per poder analitzar-les localment als nostres ordinadors de manera senzilla. En resum, aquest mòdul us permetrà dominar els elements bàsics necessaris per integrar-se sense dificultats en qualsevol entorn de recerca bioinformàtica.

Objectius

Amb el programa de continguts d'aquests materials, una vegada finalitzada l'etapa d'aprenentatge, l'estudiant serà capaç de fer la major part de les activitats bàsiques habituals en un laboratori bioinformàtic de recerca:

1. Dominar el maneig elemental dels entorns de treball Linux.
2. Utilitzar el terminal de comandaments per analitzar dades biològiques.
3. Gestionar la informació dels genomes amb bases de dades i MySQL.
4. Publicar resultats a la xarxa mitjançant pàgines web.
5. Elaborar servidors de dades per a aplicacions bioinformàtiques a internet.

Bibliografia

Alfred V. Aho, Monica S. Lam, Ravi Sethi i Jeffrey D. Ullman (2006). *Compilers: Principles, Techniques, and Tools* (2nd Edition). Addison-Wesley. ISBN: 0321486811.

Andrew S. Tanenbaum (2007). *Modern operating systems* (3rd Edition). Prentice-Hall. ISBN: 0136006639.

Brian W. Kernighan 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.

Conrad Bessant, Ian Shadforth i Darren Oakley (2009). *Building Bioinformatics Solutions: with Perl, R and MySQL*. Oxford University Press. ISBN: 0199230234.

Debra Cameron, James Elliott, Marc Loy, Eric Raymond i Bill Rosenblatt (2004). *Learning GNU Emacs* (3rd Edition). O'Reilly Media. ISBN: 0-596-00648-9.

John L. Hennessy i David A. Patterson (2002). *Computer Architecture: A Quantitative Approach* (3rd Edition). Morgan Kaufmann. ISBN: 1558605967.

Kevin Tatroe, Peter MacIntyre i Rasmus Lerdorf (2014). *Programming PHP* (3rd Edition). O'Reilly Media. ISBN: 978-1-449-39277-2.

Paul DuBois (2008). *MySQL* (4th Edition). Addison-Wesley Professional. ISBN: 0672329387.

Steven Haddock i Casey Dunn (2011). *Practical computing for biologists*. Sinauer Associates. ISBN: 978-0-87893-391-4.

Vince Buffalo (2015). *Bioinformatics Data Skills*. O'Reilly Media. ISBN: 978-1-449-36737-4.

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.

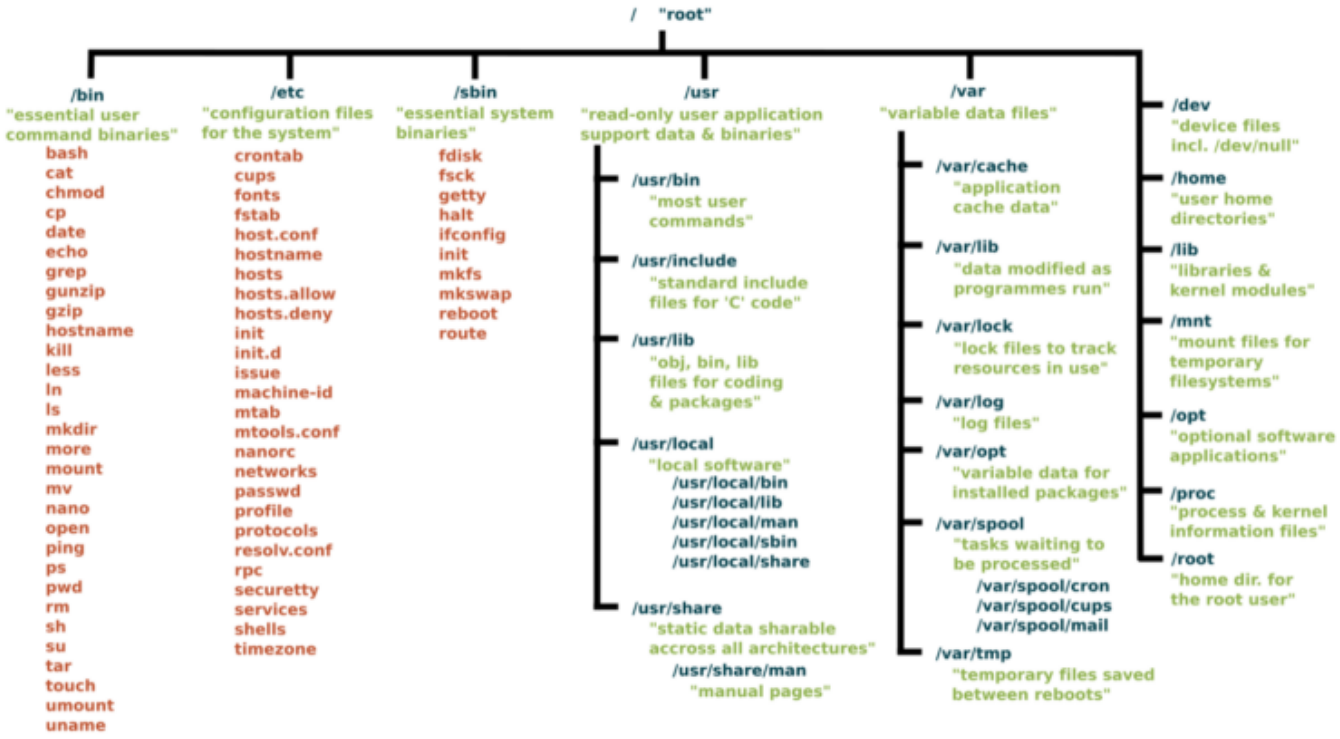


Figura 1. Estructura del *filesystem* de Linux.
Font: linuxfoundation.org

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
```

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:/
```

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/m
3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
```

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 mi*txt
ls: cannot access mi*txt': No such file or directory
$ rm fitxer.txt mi

# Amb el caràcter escape es genera un únic fitxer

$ touch mi\ fitxer.txt
$ ls mi*txt
'mi fitxer.txt'
```

```
$ rm mi\ 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

0    NM_001276352.2  chr1 -    67092164   67134970   67093579 67127240    9      67092164,6709

0    C1orf141   cmpl cmpl 2,1,0,1,2,0,0,-1,-1, 0     NM_001276351.2  chr1 -    67092164   6713
```

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).

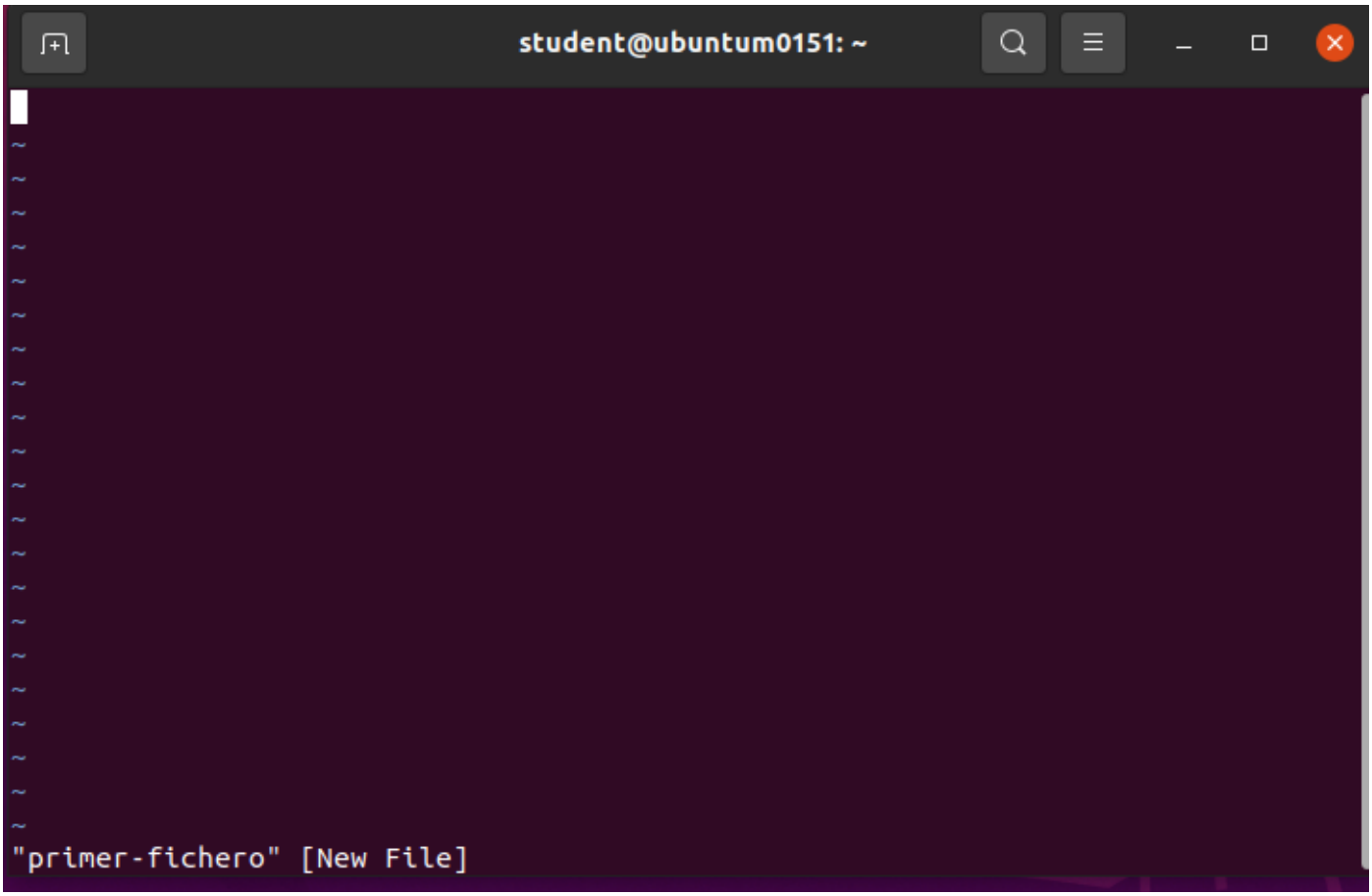


Figura 3. Imatge obtinguda en executar l'ordre *vi* primer-fitxer.

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.

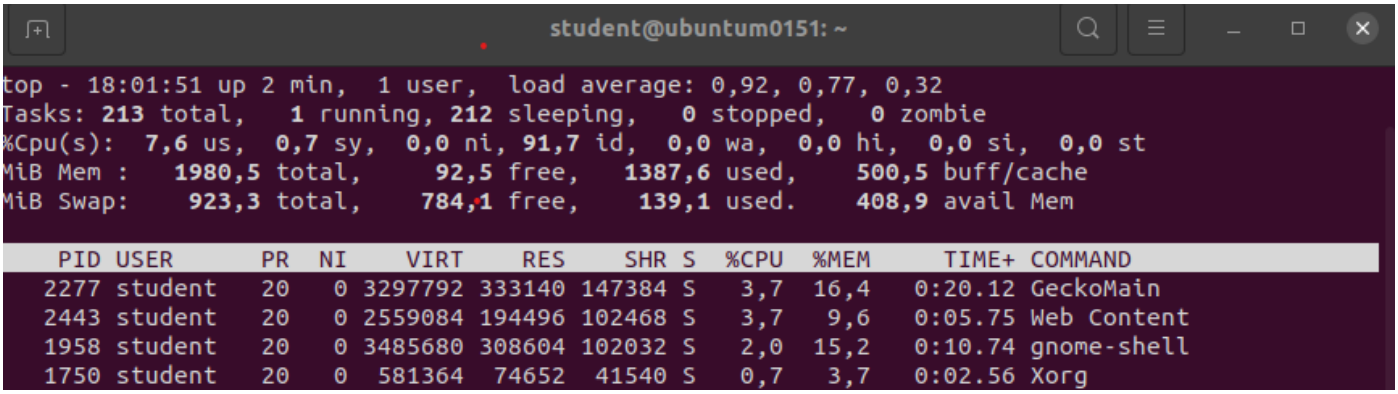


Figura 4. Imatge de la pantalla en executar l'ordre `top`.

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 fichero.txt
1.4G    genomica
316M    proteomica
746K    wdl.log
104K    gromacs.log
20K     sample.txt
```

Ordena tenint en compte la versió

```
$ sort -V tiempos.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
```

- **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 g*awk* 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_a
```

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.tx
```

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 201

  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-

  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

$ gawk -f 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); pr

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

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 inicïis 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))+' |



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 iniciïs 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
```

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/CosmicCodingMuts.vcf
```

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
```

- 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.

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...

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 2

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

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

…

…

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

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.

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...

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

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     14361,14969,15795,16606,16857,17232,1

585  NR_106918  chr1 -    17368 17436 17436 17436 1    17368,    17436,     0    MIR6859-1  un

585  NR_107062  chr1 -    17368 17436 17436 17436 1    17368,    17436,     0    MIR6859-2  un
```

```
585  NR_107063  chr1 -    17368 17436 17436 17436 1    17368,    17436,     0    MIR6859-3  un

585  NR_128720  chr1 -    17368 17436 17436 17436 1    17368,    17436,     0    MIR6859-4  un
```

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.

1.17.4. 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
```

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/

Entorns i contenidors

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

A mesura que la quantitat de dades en l'àrea de ciències de la vida i en altres àrees del coneixement augmenta exponencialment, sorgeix la necessitat d'estructurar i manejar sistemàticament la informació adquirida. Que les dades que manegem segueixin els principis FAIR (*) és fonamental per a la correcta manipulació de les dades i de la seva traçabilitat al llarg del cicle de vida de la dada. A més, hi ha dos processos fonamentals, els quals estudiarem en els propers temes, que són l'escalabilitat i la reproductibilitat.

Tradicionalment, per tal de minimitzar el nombre de passos manuals que es duen a terme en una anàlisi, els processos s'han unit programàticament en els anomenats *pipelines*. Aquesta concatenació de processos normalment s'estructura en arxius (*scripts*). Aquest tipus d'automatització normalment comporta una elevada dependència en les versions del programari instal·lades i en l'arquitectura local de l'ordinador en el qual es processa.

Com segurament ja heu vist, llenguatges de programació com Python, *bash*, R, etc. executen el que s'anomenen paquets. Els paquets poden contenir programes que poden ser utilitzats per poder processar les dades a analitzar. Un paquet per si sol normalment no té tot el codi necessari per realitzar les seves funcions i en necessita d'altres per tal d'executar-les. D'aquesta manera es reutilitza codi, minimitzant el temps de desenvolupament d'un paquet i creant eines més robustes. Per poder gestionar les dependències que un paquet té sobre d'altres van aparèixer els gestors de paquets (*apt*, *yum*, *Home Brew*, *pip*...).

En el procés d'utilització i actualització dels paquets ens podem trobar amb diverses dificultats. Durant el desenvolupament d'un paquet és possible que una funció es vegi modificada, alterant les dades d'entrada o sortida, o eliminada en una versió més recent del paquet. En el moment en què diferents paquets depenen d'aquesta funció es poden produir problemes de compatibilitat i no és suficient saber que un paquet depèn d'un altre sinó també si depèn d'una versió en específic. L'actualització d'un paquet pot representar que un altre paquet ja no funcioni. Això comportaria la necessitat d'instal·lar dues versions diferents del mateix *software*, fet que moltes vegades no és possible. Una altra problemàtica associada a les versions dels paquets és la reproductibilitat de resultats en sistemes diferents. Dos ordinadors podrien tenir instal·lats els mateixos paquets i les seves dependències, però amb versions diferents, produint resultats diferents. En projectes multicèntrics de vegades cal executar els mateixos processos en cadascuna de les institucions implicades, amb la seguretat que no hi haurà variabilitat en els resultats a causa del processament de les dades.

Així, hem de controlar:

- La utilització dels mateixos paquets i de les seves dependències.
- La implementació en altres entorns computacionals, evitant que la configuració local alteri el resultat final.

Per mantenir una configuració estable i aplicable a diferents infraestructures computacionals, fonamentalment hi ha dues estratègies a seguir: la creació d'entorns fixos i la creació de contenidors.

Objectius

1. Entendre els conceptes i la utilitat d'entorns i contenidors.
2. Aprendre a crear i a manejar entorns Conda.
3. Saber crear i manejar contenidors Docker

1. Conda

1.1. Introducció

Començarem per la creació d'entorns, i com a exemple utilitzarem Conda. Conda és un sistema de gestió de paquets i d'entorns *open source* compatible amb els sistemes operatius Linux, Mac OS i Windows. No requereix privilegis d'administrador per poder-se fer servir.

Tot i que inicialment Conda es va crear per a la gestió de paquets a Python, actualment es poden gestionar paquets creats en diferents llenguatges de programació, com R, Ruby, Scala, Java, JavaScript o C/C++.

Moltes vegades hi ha confusió entre Conda, MiniConda i Anaconda (figura 1). El nucli del gestor de paquets i entorns és Conda, mentre que MiniConda és l'instal·lador mínim de Conda, a més de combinar Python i uns paquets bàsics, i Anaconda engloba MiniConda i augmenta el nombre de paquets preinstal·lats.

Figura 1. Esquema de les diferències entre Conda, MiniConda i Anaconda.
Font: elaboració pròpia.

1. Conda

1.2. Què són paquets i canals a Conda?

Un paquet és un arxiu comprimit, sigui un tar (.tar.bz2) o un .conda, que conté:

- Llibreries de sistema.
- Python i altres mòduls.
- Executables i altres components.
- Metadates en el directori info/.
- Una col·lecció de fitxers que s'instal·len directament.

Un paquet no necessàriament ha de contenir tots aquests elements; per exemple, un paquet que únicament conté metadades es denomina *metapackage*. Podeu trobar un llistat dels paquets que gestiona Conda a https://anaconda.org/ per a la seva instal·lació.

Quan vulguem instal·lar un paquet en un entorn Conda haurem de saber on localitzar-lo, i aquesta és la funció dels canals. Els canals són URLs que ens dirigeixen on podem trobar els directoris que contenen les dades dels paquets. En instal·lar un paquet, l'ordre `conda` busca en un conjunt de canals que té per defecte. Si no s'especifica el contrari, Conda instal·larà els paquets continguts en aquests canals definits per defecte. A més dels canals per defecte hi ha un altre canal que ostenta un estatus diferencial. Aquest és el canal **conda-forge**. Aquest canal està administrat per la comunitat i pot tenir alguns avantatges respecte als canals per defecte gestionats per Anaconda Inc., com per exemple que els paquets usualment estan més actualitzats o que alguns paquets no són accessibles des dels canals per defecte. Finalment, cal destacar el canal **bioconda**, que se centra en paquets desenvolupats per a la bioinformàtica.

1. Conda

1.3. Entorns Conda

Un entorn Conda és un directori que conté una col·lecció específica de paquets Conda que has instal·lat. Si modifiques un entorn, els altres entorns no es veuen afectats i fàcilment pots activar i desactivar entorns.

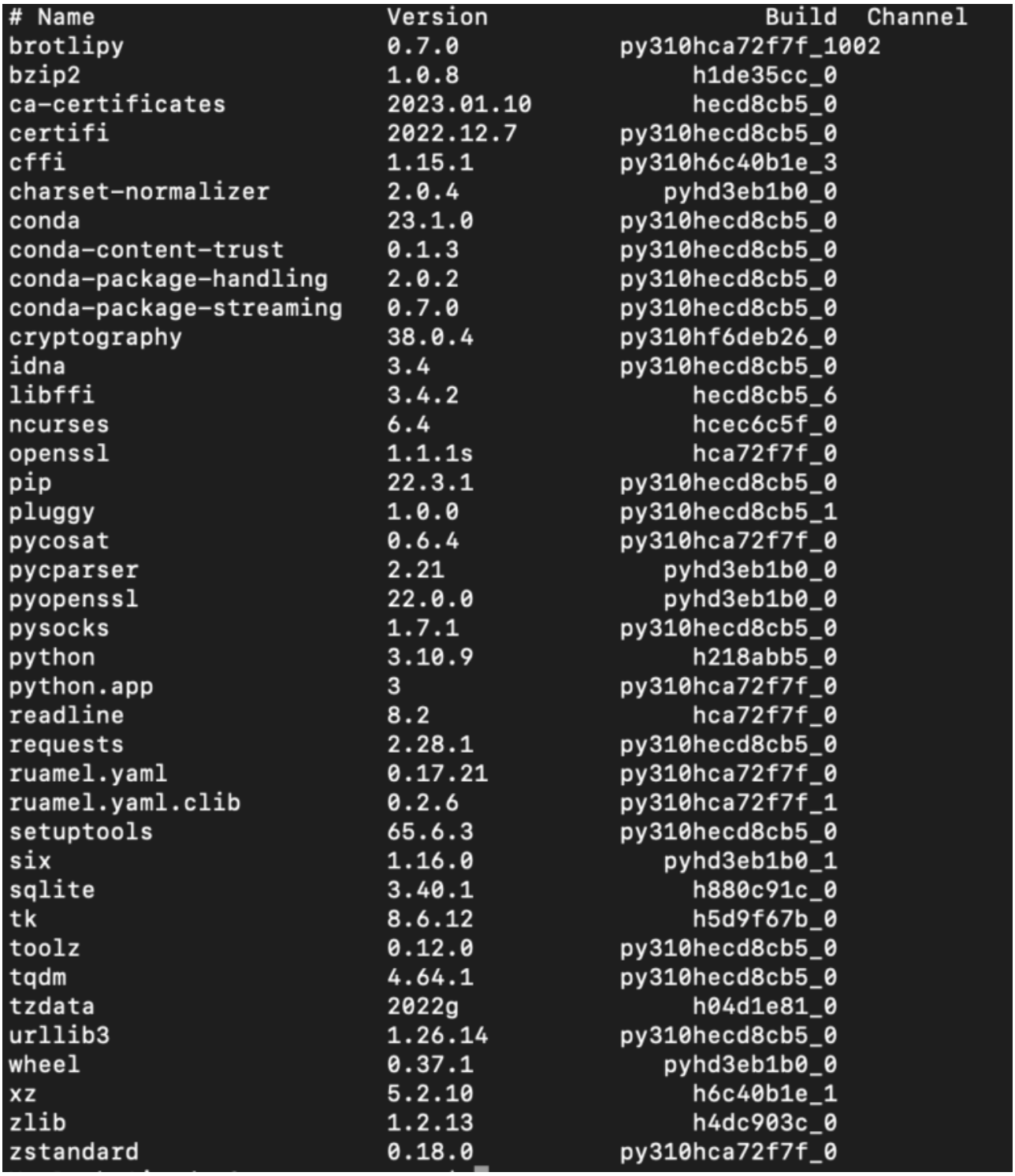
Per poder crear entorns Conda, en primer lloc, heu d'instal·lar MiniConda o Anaconda, depenent de les vostres preferències. Seguirem l'exemple utilitzant MiniConda i la instal·lació a Linux/Mac OS. En aquest enllaç podreu trobar les instruccions necessàries depenent del sistema operatiu que feu servir (https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html). Us heu de descarregar l'instal·lador depenent del vostre sistema operatiu i , en el cas de Linux o Mac OS, executar:

```
$ bash ~/miniconda.sh -b -p $HOME/miniconda
```

Per testar si la instal·lació ha finalitzat correctament podeu escriure al terminal

```
$ conda list
```

que retornarà el llistat de paquets instal·lats (figura 2). Recordeu que, si no es troba l'ordre conda, segurament és pel fet que no està en la ruta especificada d'ordres del vostre terminal, però el podreu trobar a la carpeta on heu instal·lat MiniConda o Anaconda, a la subcarpeta «$HOME/miniconda/bin».



```
# Name                    Version           Build  Channel
brotlipy                  0.7.0           py310hca72f7f_1002
bzip2                     1.0.8                h1de35cc_0
ca-certificates           2023.01.10           hecd8cb5_0
certifi                   2022.12.7       py310hecd8cb5_0
cffi                      1.15.1          py310h6c40b1e_3
charset-normalizer        2.0.4              pyhd3eb1b0_0
conda                     23.1.0          py310hecd8cb5_0
conda-content-trust       0.1.3           py310hecd8cb5_0
conda-package-handling    2.0.2           py310hecd8cb5_0
conda-package-streaming   0.7.0           py310hecd8cb5_0
cryptography              38.0.4          py310hf6deb26_0
idna                      3.4             py310hecd8cb5_0
libffi                    3.4.2                hecd8cb5_6
ncurses                   6.4                  hcec6c5f_0
openssl                   1.1.1s               hca72f7f_0
pip                       22.3.1          py310hecd8cb5_0
pluggy                    1.0.0           py310hecd8cb5_1
pycosat                   0.6.4           py310hca72f7f_0
pycparser                 2.21               pyhd3eb1b0_0
pyopenssl                 22.0.0             pyhd3eb1b0_0
pysocks                   1.7.1           py310hecd8cb5_0
python                    3.10.9               h218abb5_0
python.app                3               py310hca72f7f_0
readline                  8.2                  hca72f7f_0
requests                  2.28.1          py310hecd8cb5_0
ruamel.yaml               0.17.21         py310hca72f7f_0
ruamel.yaml.clib          0.2.6           py310hca72f7f_1
setuptools                65.6.3          py310hecd8cb5_0
six                       1.16.0             pyhd3eb1b0_1
sqlite                    3.40.1               h880c91c_0
tk                        8.6.12               h5d9f67b_0
toolz                     0.12.0          py310hecd8cb5_0
tqdm                      4.64.1          py310hecd8cb5_0
tzdata                    2022g                h04d1e81_0
urllib3                   1.26.14         py310hecd8cb5_0
wheel                     0.37.1             pyhd3eb1b0_0
xz                        5.2.10               h6c40b1e_1
zlib                      1.2.13               h4dc903c_0
zstandard                 0.18.0          py310hca72f7f_0
```

FFigura 2. Sortida de l'ordre *conda list*.
Font: elaboració pròpia.

Conda té un entorn per defecte anomenat *base*. No és recomanable instal·lar paquets en aquest entorn. Si es vol iniciar un nou projecte, s'ha de crear nous entorns Conda.

1. Conda

1.4. Creació d'entorns Conda

És important donar un nom descriptiu a l'entorn per poder reconèixer-ne el contingut. En un *pipeline* estàndard, podem utilitzar múltiples programes que poden ser reutilitzables en altres anàlisis. A causa d'aquesta redundància de processos, és recomanable crear un entorn per a cada eina i no crear entorns amb múltiples programes. D'aquesta manera, un entorn estarà definit pel *software* que compta mitjançant el nom, i la seva utilització serà més senzilla que si un mateix entorn conté múltiples paquets, ja que serà difícil determinar on es troba el programa que necessites en un moment determinat. De vegades també és recomanable no únicament especificar el programa en el nom de l'entorn, sinó també la seva versió.

En primer lloc, instal·larem un paquet molt usat a Python com és `numpy`. Per crear un entorn utilitzarem l'ordre `create` i n'especificarem la versió a instal·lar. Per saber quines versions són accessibles podem utilitzar l'ordre `search`:

```
$ conda search numpy
```

Això ens farà un llistat de totes les versions que estan disponibles. Si no especifiquem la versió del paquet a instal·lar, Conda intentarà instal·lar la versió més nova. Un cop seleccionem la versió que necessitem podem crear el nostre nou entorn:

```
$ conda create -n numpy1-23-5 numpy=1.23.5
```

Fent servir l'opció *-n* indiquem el nom que volem assignar al nou entorn.

Si voleu especificar des de quin canal volem instal·lar un paquet podem utilitzar l'opció `channel`:

```
$ conda create -n numpy1-23-5b numpy=1.23.5 --channel conda-forge
```

Conda no instal·la únicament el paquet especificat, sinó també les seves dependències. En el cas de *numpy*, per exemple, no hem especificat que instal·lés Python, però com que és una dependència automàticament s'instal·la en l'entorn especificat.

Si sempre utilitzem un set de paquets, podem instal·lar-los conjuntament:

```
$ conda create -n basic-analisis numpy=1.23.5 pandas=1.5.3 matplotlib=3.7.1
```

Un cop creats els entorns podem saber quins paquets s'han instal·lat utilitzant l'ordre `list`:

```
$ conda list -n numpy1-23-5
```

1. Conda

1.5. Activar entorns Conda

Un cop hem creat el nostre primer entorn Conda és hora d'utilitzar-lo. Per obtenir un llistat dels entorns que tenim en el nostre sistema utilitzarem l'ordre `env list`:

```
$ conda env list
```

Observareu que a més del nostre entorn `numpy1-23-5` i `basic-analisis` tenim l'entorn **base**, que com us havia comentat anteriorment està definit per defecte. És recomanable no instal·lar *software* en aquest entorn.

Per poder utilitzar un dels entorns l'hem d'activar:

```
$ conda activate numpy1-23-5
```

Sabrem que l'entorn s'ha activat, ja que apareixerà el seu nom entre parèntesis davant de la línia d'ordres.

1. Conda

1.6. Desactivar entorns Conda en ús

Un cop finalitzats els processos requerits a l'entorn hem de tancar l'entorn. Per a això utilitzarem l'ordre `deactivate`:

```
$ conda deactivate
```

D'aquesta manera tornarem a l'estat inicial, abans de l'activació de l'entorn.

1. Conda

1.7. Instal·lar paquets dins d'un entorn ja creat

Els entorns Conda poden ser modificats *a posteriori* de la seva creació. Simplement, haurem d'activar l'entorn i, un cop dins, instal·lar un nou paquet. L'ordre `install` per defecte instal·la un paquet a l'entorn actiu.

```
$ conda activate numpy1-23-5

( numpy1-23-5 ) $ conda install pandas=1.5.3
```

En aquest cas s'instal·laria el paquet *pandas* a l'entorn `numpy1-23-5`. Si el nou paquet no és compatible amb la configuració de l'entorn actiu, saltarà un error i no podrà ser instal·lat.

Si volem especificar el canal, podem fer-ho:

```
( numpy1-23-5 ) $ conda install conda-forge::pandas=1.5.3
```

1. Conda

1.8. Reanomenar entorns Conda

Si, quan instal·lem nous paquets, ho volem especificar en el nom de l'entorn, podem modificar el nom assignat:

```
$ conda rename -n numpy1-23-5 numpy1-23-5-pandas1-5-3
```

1. Conda

1.9. Eliminar entorns Conda

Si un entorn ja no s'utilitza, és convenient eliminar-lo i, per fer-ho, utilitzarem l'ordre `remove`:

```
$ conda remove -n numpy1-23-5-pandas1-5-3 —all
```

1. Conda

1.10. Compartir entorns

En projectes col·laboratius és freqüent la necessitat de reproduir tasques en els diferents centres. Per a això s'han de crear entorns agnòstics de sistema operatiu i plenament compatibles. Conda utilitza YAML (YAML *Ain't Markup Language*) com a arxius d'entorn que ens permetran importar i exportar entorns.

Per convenció, els arxius d'entorn de Conda s'anomenen *environment.yml.*

Si en el nostre directori de treball executem:

```
$ conda env create
```

automàticament Conda buscarà l'arxiu *environment.yml.* Si no el troba, saltarà un error. Si l'arxiu d'entorn té un altre nom, el podem especificar de la manera següent:

```
$ conda env create --file prova.yaml
```

Podeu observar l'estructura d'un arxiu d'entorn de la figura 3, on es poden observar tres apartats:

- `name`: nom de l'entorn que es crearà si no s'especifica el contrari.
- `Channels`: canals a usar.
- `dependencies`: paquets a instal·lar amb relació `canal:paquet:versió`

```yaml
name: nf-core-clipseq-1.0.0
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - conda-forge::python=3.7.3
  - conda-forge::markdown=3.1.1
  - conda-forge::pymdown-extensions=6.0
  - conda-forge::pygments=2.5.2
  - conda-forge::pigz=2.3.4
  - conda-forge::perl=5.26.2

  # bioconda packages
  - bioconda::fastqc=0.11.9
  - bioconda::multiqc=1.9
  - bioconda::cutadapt=3.0
  - bioconda::bowtie2=2.4.2
  - bioconda::star=2.6.1d # Needs to be 2.6 to work with iGenomes indices
  - bioconda::samtools=1.11
  - bioconda::umi_tools=1.1.1
  - bioconda::bedtools=2.29.2
  - bioconda::subread=2.0.1
  - bioconda::preseq=2.0.3
  - bioconda::rseqc=4.0.0

  # peak calling packages - may need to switch to pip for latest versions
  - bioconda::icount=2.0.0
  - bioconda::paraclu=9
  - bioconda::pureclip=1.3.1
  - bioconda::piranha=1.2.1

  # motif calling packages
  - bioconda::meme=5.1.1
```

Figura 3. Exemple d'arxiu d'entorn YAML.
Font: elaboració pròpia.

Per poder generar un arxiu YAML d'un entorn que nosaltres hem creat, executarem:

```
$ conda env export -n basic-analisis --file basic.yaml
```

Especifiquem el nom de l'arxiu amb l'opció `--file`

Per assegurar que l'entorn pot ser reproduïble independentment del sistema operatiu cal especificar l'opció `--from-history`:

```
$ conda env export -n basic-analisis —from-history --file basic.yaml
```

2. Docker

2.1. Introducció

Un cop hem vist els entorns Conda, ens introduirem en una altra metodologia per controlar els processos utilitzats: els contenidors (*).

Els contenidors són sistemes de virtualització que contenen totes les eines necessàries per executar un *software*. Molt sovint es comparen les màquines virtuals amb els contenidors. La diferència més important és que les màquines virtuals virtualitzen tota una màquina fins a les capes de maquinari, mentre que els contenidors únicament virtualitzen la capa de *software* damunt del sistema operatiu. Aquesta característica els fa més lleugers i fàcils de modificar.

Tot i que els contenidors no són una tecnologia nova, la seva aplicació de manera extensa va començar amb l'aparició de Docker el 2013 (consulta la caixa lateral). La popularització d'aquestes aplicacions va introduir la complexitat d'administrar centenars o milers de contenidors, i per això va aparèixer el que es coneix com la foscor de contenidors. Tot i que al llarg del temps han aparegut diferents plataformes d'orquestració, fins i tot una del mateix Docker, com és Docker Swarm, Google va crear el 2014 Kubernetes, de codi obert, que s'ha convertit en el *software* preferit de moltes empreses i s'ha consolidat com un estàndard. Les plataformes d'orquestració s'encarreguen de reiniciar les aplicacions si fallen, d'equilibrar la càrrega de treball, d'escalar automàticament, d'implementar sense temps d'inactivitat, etc.

Tot i que en sistemes HPC (High Performance Computing) s'utilitza més Singularity, en aquest apartat de contenidors ens centrarem en com utilitzar Docker.

2. Docker

2.2. Contenidors Docker

En primer lloc, necessitarem instal·lar Docker al nostre ordinador. La forma més senzilla és utilitzant Docker Desktop (https://www.docker.com/products/docker-desktop/).

Per saber que tens Docker instal·lat correctament al terminal pots interrogar la seva versió:

```
$ docker -v
```

L'output t'indicarà la versió i el *build* que tens instal·lats.

I també:

```
$ docker system info
```

Si saltés algun error, s'hauria de comprovar si el procés d'instal·lació ha finalitzat correctament i que Docker Desktop estigui obert.

Per crear un contenidor, necessites el que es denominen imatges. Les imatges són els motlles dels contenidors, són les receptes/instruccions per crear els contenidors.

2. Docker

2.3. Descarregar contenidors creats

En primer lloc, mirarem si tenim alguna imatge en el nostre sistema. En teoria, si és la primera vegada que utilitzem Docker, ens hauria d'aparèixer una llista en blanc quan utilitzem:

```
$ docker image ls
```

Començarem pel contenidor més senzill i el descarregarem directament:

```
$ docker image pull hello-world
```

Deixarem que s'executi el procés de descàrrega i, si tot ha funcionat correctament, quan repetim l'ordre de llistat d'imatges ens hauria d'aparèixer la imatge `hello-world`.

La imatge de «hello-world» procedeix de Docker Hub (*), un repositori d'imatges.

Seguidament, executarem el contenidor que es generi a partir de la imatge `hello-world` mitjançant:

```
$ docker container run hello-world
```

Un cop executat, rebreu un missatge de part de l'equip de Docker (figura 4).

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figura 4. Missatge de benvinguda de l'equip de Docker en executar «hello-world».
Font: elaboració pròpia.

Quan s'executa un contenidor succeeixen tres processos:

- S'inicialitza el contenidor a partir de la imatge.
- S'executa l'acció preestablerta del contenidor, si existeix.
- Un cop l'acció ha finalitzat, el contenidor s'atura.

En el cas de l'exemple anterior, l'acció preestablerta era imprimir el missatge de benvinguda, però l'acció realitzada pot ser molt més complexa.

A més de poder executar les ordres predeterminades pel contenidor, també podem passar ordres o entrar en mode interactiu. Per provar aquestes opcions executarem el contenidor Alpine que conté una distribució d'Ubuntu molt simple:

```
$ docker container run -it alpine sh
```

En aquesta ordre utilitzem l'opció `it` per ser interactiu i `sh` ens especifica que el terminal que volem utilitzar és *bash*.

Veurem que la línia d'ordres canvia a:

```
/ #
```

Podrem comprovar que ara ens trobem dins d'Alpine i quan executem:

```
/ #  cat /etc/os-release
```

ens mostrarà la versió d'Alpine que ens hem baixat.

D'aquesta manera, un cop executat el contenidor no s'ha finalitzat com havíem observat anteriorment, sinó que es manté actiu i respon a les ordres que hi introduïm.

Per poder sortir del contenidor i finalitzar la seva execució podem introduir-hi l'ordre `exit`.

2. Docker

2.4. Eliminar imatges i contenidors Docker

Sempre és recomanable mantenir una bona gestió d'imatges i contenidors, ja que, si no s'utilitzen, van ocupant espai de manera innecessària. Així, aprendrem a eliminar imatges i contenidors.

Per poder eliminar imatges, necessitarem saber la Image ID. Per a això, utilitzarem la següent ordre:

```
$ docker image ls
```

Ens apareixerà un *output* similar al següent:

Taula 1.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	9ed4aefc74f6	10 days ago	7.05MB
hello-world	latest	feb5d9fea6a5	18 months ago	13.3kB

Font: elaboració pròpia.

Així, per eliminar una imatge simplement s'ha d'especificar la Image ID:

```
$ docker image rm feb5d9fea6a5
```

O emprant el seu nom:

```
$ docker image rm hello-world
```

Moltes vegades ens podem trobar amb un missatge d'error:

```
Error response from daemon: conflict: unable to delete feb5d9fea6a5 (must be forced) — image i
```

Això ens està indicant que hi ha un contenidor que encara està actiu, que encara no ha estat netejat o que està o ha fet ús d'aquesta imatge. Per això, primer hem d'eliminar els contenidors que estan impedint l'eliminació d'aquesta imatge.

Primer llistem els contenidors actius:

```
$ docker container ls
```

O, de manera similar, podem utilitzar:

```
$ docker ps
```

És possible que no ens retorni cap contenidor actiu. En realitat, el missatge d'error previ ens indicava que el contenidor s'havia aturat; per tant, té sentit que no trobem el contenidor en aquest llistat. Així, necessitarem veure tant els contenidors actius com els que recentment s'han aturat:

```
$ docker container ls –all
```

Aquí sí que podem veure el contenidor anteriorment especificat com a enllaçat a «hello-world»:

Taula 2.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8a149fe84874	alpine	"sh"	About an hour ago	Exited		xenodochial_thompson
06dda9650983	hello-world	"/hello"	12 hours ago	Exited		upbeat_satoshi

Font: elaboració pròpia.

Per evitar que es vagin acumulant els contenidors, es pot afegir a l'ordre `run` una opció d'eliminar automàticament un cop finalitzat:

```
$ docker container run --rm hello-world
```

Si no hem especificat l'opció `rm`, l'hem d'eliminar manualment utilitzant com a referència el Container ID:

```
$ docker container rm 06dda9650983
```

Si s'elimina correctament, ens retornarà el Container ID al terminal. Es pot esborrar més d'un Container ID simultàniament.

Si tenim diversos contenidors associats a una imatge i volem esborrar-los tots utilitzant un patró podem:

```
$ docker container ps -a | grep "world" | awk '{print $1}' | xargs docker rm
```

En aquest cas, utilitzem l'ordre `xargs` per controlar una llista d'arguments per poder ser eliminats mitjançant `rm` de Docker.

Ara veurem que, quan llistem els contenidors, ja no apareix el contenidor i, si provem d'eliminar la imatge com prèviament havíem intentat, ho farà sense problemes.

```
$ docker image rm hello-world

$ docker image ls
```

Si es volen eliminar tots els contenidors existents, podem utilitzar:

```
$ docker container prune
```

D'aquesta manera tots els contenidors existents seran eliminats i, si volem tornar a reconnectar, haurem d'iniciar-los de nou.

2. Docker

2.5. Parar contenidors a Docker

Iniciar o aturar un contenidor no és el mateix que iniciar o aturar un procés. Per finalitzar un contenidor, Docker proporciona les ordres `stop` i `kill`. Encara que sembli que totes dues ordres facin el mateix, la seva execució és, internament, diferent. Per aturar un procés, podem utilitzar tant el ContainerId com el nom del contenidor.

L'ordre `stop` atura el contenidor d'una manera menys agressiva que `kill`. Això és degut al tipus de senyal que es mana al contenidor. L'ordre `stop` envia un senyal SIGTERM, que es pot bloquejar o parar, mentre que `kill` envia un senyal SIGKILL que no es pot gestionar. Si en un temps prudencial l'ordre `stop` no ha aturat el contenidor, Docker envia automàticament un senyal SIGKILL. Per defecte, aquest període són 10 segons, però si volem modificar-lo podem fer ús de l'opció *-t* expressada en segons. Així:

```
$ docker container stop -t 77 hello-world
```

Docker pararia el procés mitjançant una SIGKILL després de 77 segons.

Com hem vist en l'apartat anterior també podríem utilitzar l'ordre `rm` per finalitzar un contenidor. La diferència rau en el fet que l'ordre `rm` elimina el procés i no el podem visualitzar en la llista `docker ps -a`, mentre que, parant el contenidor, el podem mantenir i reutilitzar posteriorment.

Una altra opció interessant és aturar el contenidor. Si aturem un contenidor, alliberem els recursos de memòria i CPU; si el pausem, només alliberem CPU.

```
$ docker container pause hello-world
```

2. Docker

2.6. Buscar contenidors a Docker Hub

Un recurs que ja hem utilitzat per al contenidor «hello-world» és Docker Hub. En aquest repositori podem trobar molts contenidors ja creats. Molts d'ells ja han estat construïts i testats pels mateixos desenvolupadors del *software* que està buscant. Com a exemple anirem a la pàgina del *variant caller* GATK (https://hub.docker.com/r/broadinstitute/gatk). Aquí trobareu les instruccions per poder-vos-el descarregar. Si en voleu una versió determinada, en baixar la imatge necessiteu especificar-la. Per a això s'utilitzen els *tags* igual que com vam veure en l'apartat de Conda. A la pàgina de GATK teniu una pestanya on teniu les diferents versions indicades per *tags*.

Per indicar quina versió volem utilitzar ho indicarem amb els «`:`»:

```
$ docker image pull broadinstitute/gatk:4.4.0.0
```

Si no especifiquem la versió, es baixarà la imatge més recent denominada `latest`.

Com podeu apreciar en l'ordre `pull` que hem utilitzat per baixar GATK davant del nom de `gatk`, tenim el nom de la institució que l'ha creat i l'ha fet públic. En aquest cas, el Broad Institute. Si aquest nom previ no apareix, indica que els desenvolupadors són el mateix equip de Docker.

És important tenir en compte que qualsevol persona pot crear un compte a Docker Hub i, per tant, és important ser prudent quan descarreguem *software* de fonts no contrastades. Haureu de procurar baixar imatges directament dels desenvolupadors o de comunitats establertes. Docker manté una sèrie d'imatges referenciades com Docker Official Images, les quals han estat analitzades i proporcionen repositoris bàsics per a la comunitat, com ara Ubuntu o Centossucrades.

Una altra opció per buscar imatges és l'ordre `search`. En aquest cas, quan utilitzem

```
$ docker search GATK
```

obtenim un llistat d'imatges on trobaríem repositoris amb GATK (figura 5).

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
broadinstitute/gatk	Official release repository for GATK version…	91		
bitnami/keycloak-gatekeeper	Bitnami Keycloak Gatkeeper Docker Image	8		
broadinstitute/gatk3	Official release repository for GATK version…	4		
kfdrc/gatk	alpine based gatk	2		[OK]
gatksv/sv-pipeline		1		
broadinstitute/gatk-nightly	Official repository for nightly development …	1		[OK]
mgibio/gatk-cwl	Image containing gatk, for use in cwl workfl…	1		[OK]
gatksv/sv-pipeline-base		0		
gatksv/sv-pipeline-rdtest		0		
gatksv/gatk		0		
gatksv/cnmops		0		
gatksv/samtools-cloud		0		
gatksv/sv-base-mini		0		
gatksv/wham		0		
gatkworkflows/mtdnaserver		0		
gatksv/manta		0		
dukegcb/gatk-base	Dependencies (Java, R) for running GATK	0		[OK]
gatksv/sv-base		0		
gatksv/delly		0		
gatksv/sv-pipeline-qc		0		
biocontainers/gatk		0		
pegi3s/gatk-3	GATK 3 (https://gatk.broadinstitute.org/hc/e…	0		
pegi3s/gatk-4	GATK 4 (https://gatk.broadinstitute.org/hc/e…	0		
jsotobroad/gatk		0		
agrf/gatk	https://software.broadinstitute.org/gatk/	0		

Figura 5. *Output* de la recerca d'imatges que continguin GATK.
Font: elaboració pròpia.

2. Docker

2.7. Crear la teva pròpia imatge Docker

Si la nostra recerca a Docker Hub ha estat infructuosa o requerim una imatge específica per a les nostres necessitats, la solució és crear-la nosaltres mateixos.

En primer lloc, instal·larem *software* en una sessió interactiva. Amb aquesta finalitat, utilitzarem la imatge d'Alpine que prèviament havíem creat i intentarem instal·lar el paquet de *python numpy*. Alpine no és la distribució en la qual voldríem basar-nos per desenvolupar els nostres projectes; segurament Ubuntu o Debian siguin unes eleccions més apropiades. Abans de començar una imatge és important que tinguem ben clar què necessitarem instal·lar, quins són els nostres requisits i, finalment, escollir la distribució més apropiada. També és una bona pràctica no instal·lar massa programes en cada imatge perquè seran més difícils de crear i de mantenir. És la mateixa filosofia que aplicàvem als entorns Conda.

```
$ docker container run -it alpine sh
```

Seguidament comprovarem si tenim Python instal·lat:

```
/ # python --version
```

I podem observar que no està instal·lat.

Alpine té un gestor de paquets anomenat *Alpine Package Keeper* (apk) per instal·lar *software.* Depenent de la distribució de Linux que tinguem instal·lat, podríem utilitzar `apt-get, yum, zypper`...

En primer lloc, instal·larem Python i altres paquets necessaris. Posteriorment, afegirem el paquet *numpy*.

```
/ # apk --no-cache --update-cache add gcc gfortran python3 py3-pip python3-dev build-base wget


/ # pip install numpy
```

Si ara entrem a Python podem comprovar que s'ha instal·lat correctament el paquet *numpy*.

Un cop sortim d'aquest contenidor, els canvis no s'hauran guardat. Per crear la nostra pròpia imatge, el més recomanable és utilitzar un Dockerfile.

Un Dockerfile és un arxiu de text amb l'estructura mínima següent:

- FROM <Imatge preexistent>
- RUN <Ordres per instal·lar des de la línia d'ordres>
- CMD <Ordres que s'han de córrer per defecte>

Les instruccions que es llancen per defecte (CMD) tenen una estructura definida. Només hi pot haver una línia de CMD al Dockerfile; si n'hi hagués més d'una, s'executaria l'última.

Així, si volguéssim reproduir el contenidor anterior, hauríem de crear un arxiu anomenat *Dockerfile* de la següent manera:

```
FROM alpine
```

```
RUN apk --no-cache --update-cache add gcc gfortran python3 py3-pip python3-dev build-base wget

RUN pip install numpy

CMD python3 --version
```

Un cop tenim el Dockerfile definit crearem la imatge:

```
$ docker image build -t uoc/numpy .
```

L'opció `-t` ens indica el nom que li volem donar a la nostra imatge. Seguint la nomenclatura anteriorment esmentada indiquem l'autor i el paquet. El «`.`» ens indica que Dockerfile és a la mateixa carpeta on estem executant les ordres; hem d'especificar la ruta al directori del Dockerfile.

Ara podreu veure la nova imatge creada:

```
$ docker image ls
```

Taula 3.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
uoc/numpy	latest	af1700de32cb	25 minutes ago	588MB

Si executem el contenidor Docker, ens retornarà la versió de Python que tenim instal·lada a la imatge:

```
$ docker container run uoc/numpy
```

Si entréssim ordres en la línia d'execució del contenidor, aquestes són les que s'utilitzarien en detriment de les que estiguessin al Dockerfile. Així:

```
$ docker container run uoc/numpy which python3
```

Ens indicarà la localització de Python3 en el *filesystem* del contenidor.

2. Docker

2.8. Reanomenar imatges

En realitat, el que crearem és una còpia d'una imatge amb un nom nou. Per això, farem:

```
$ docker image tag uoc/numpy uoc/alpine-numpy
```

Si la imatge és susceptible d'evolucionar, és important mantenir una numeració en els seus *tags*, com hem vist a Docker Hub:

```
$ docker image tag uoc/alpine-numpy uoc/alpine-numpy:1.0.0
```

Si ara mireu les imatges que teniu creades, podreu veure que a la columna *tag* tindreu la versió que hem creat ara i la `latest`, que és per defecte, si no se n'especifica un altre.

Taula 4.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
uoc/numpy	1.0.0	af1700de32cb	25 minutes ago	588MB
uoc/numpy	latest	af1700de32cb	25 minutes ago	588MB

Font: elaboració pròpia.

2. Docker

2.9. Execució d'*scripts*

Com hem vist, la creació d'un contenidor és relativament senzill de fer, però, depenent de les nostres necessitats, és possible que els exemples anteriors es quedin curts. Per això hem d'ampliar els coneixements necessaris per crear imatges més complexes.

En primer lloc, utilitzarem el contenidor amb Alpine que hem creat anteriorment per executar un *script* a Python.

Si, directament, passem els paràmetres quan iniciem el contenidor, ens retornarà un error perquè l'arxiu no es troba en el seu sistema:

```
$ docker container run --rm uoc/alpine-numpy python3 num.py
```

Sent `num.py`:

```
import numpy as np

a = np.array([1,2,3,4,5,6,7])

print (a)
```

L'error és:

```
python3: can't open file '//num.py': [Errno 2] No such file or directory
```

Perquè aquesta ordre funcioni, necessitarem enllaçar els dos sistemes, el nostre propi i el del contenidor Docker. Per això, utilitzarem l'ordre `mount` especificant on és el nostre arxiu i posicionant-lo en el sistema del contenidor.

```
$ docker container run --rm --mount type=bind,source=${PWD},target=/temp uoc/alpine-numpy pyth
```

Existeixen diferents tipus de *mount*. En aquest cas, utilitzarem el mode *bind*, que és el que ens interessa per a aquest exemple. Amb el *source*, especifiquem el directori on s'ubica *l'script*, podem utilitzar la variable ${PWD} si llancem el contenidor des del mateix directori on hi ha l'*script* i també podem utilitzar *target*, que especifica on localitzarem el contenidor. És important que, quan executem Python3, especifiquem la carpeta on localitzem l'*script*, en aquest cas /temp.

D'aquesta manera ens retornarà:

```
[1 2 3 4 5 6 7]
```

2. Docker

2.10. Volums Docker

En l'exemple anterior hem utilitzat `mount` per unir els dos sistemes. `Mount` depèn del sistema en el qual s'executa, mentre que els **volums** són nadius de Docker. Els volums es poden compartir entre contenidors i persisteixen més que els contenidors.

En l'exemple anterior, podríem passar un volum al contenidor amb l'opció `-v`:

```
$ docker container run --rm -v $(PWD):/temp uoc/alpine-numpy python3 /temp/num.py
```

Definim la carpeta local $(PWD) i on es localitza al contenidor /temp.

Per crear un volum amb nom `data_set` i assignar una carpeta local:

```
$ docker volume create --driver local --opt device=/Path/al/directorio/local --opt type=none -
```

En aquest cas, podrem llançar:

```
$ docker container run --rm -v data_set:/temp uoc/alpine-numpy python3 /temp/num.py
```

Podem visualitzar els diferents volums:

```
$ docker volume ls
```

... inspeccionar-los:

```
$ docker volume inspect data_set
```

... i eliminar-los:

```
$ docker volume rm data_set
```

2. Docker

2.11. Integrar dades al contenidor Docker

Moltes vegades necessitarem utilitzar un *input* de manera sistemàtica en un contenidor. Per exemple, si volem fer una *variant call* utilitzant GATK, els genomes de referència sempre seran els mateixos i, tal vegada, és interessant guardar-los dins del contenidor per assegurar-nos la reproductibilitat dels resultats per tal que no depengui de la referència utilitzada. Per a un cas més simple introduirem el nostre *script num.py* al contenidor. Per això crearem una nova imatge modificant el Dockerfile.

Hi afegirem una nova línia:

```
COPY num.py /home
```

Estarem fent una còpia de l'*script* al *home* del contenidor. Fem una altra imatge:

```
$ docker image build -t uoc/alpine-num .
```

Si ara entrem de forma interactiva dins del contenidor i llistem els arxius dins el *home,* trobarem l'arxiu *num.py*:

```
$ docker container run --rm -it uoc/alpine-num sh
```

És important com ordenem les ordres al Dockerfile. És recomanable fer les ordres COPY després dels RUN, ja que, quan fem el *build,* Docker va per ordre i, si en algun moment volem afegir un altre *script* en lloc de *num.py*, si el COPY es troba al final del procés, Docker utilitzarà els RUN que té a la memòria no haurà de construir la imatge de zero, i el procés serà molt més ràpid. Docker va línia a línia i, si aquesta capa o conjunt de capes ja la té en memòria *caché,* agilitza el procés perquè no ha de reinstal·lar-les.

Si les dades que volem introduir en la nostra imatge són a internet, podem copiar-les directament fent servir RUN. Podríem afegir aquestes línies al Dockerfile com a exemple:

```
RUN wget https://ftp.ncbi.nlm.nih.gov/refseq/H_sapiens/annotation/GRCh38_latest/refseq_identif
```

Per defecte, l'arxiu es copia en el *root* del sistema del contenidor. Si el volguéssim moure a una altra localització, podríem especificar-la:

```
RUN mv GRCh38_latest_clinvar.vcf.gz /home
```

... i la copiaria a la nostra carpeta /home.

Ara que sabem com introduir l'*script* dins del contenidor, podem crear un nou contenidor que corri *l'script* automàticament. Hem de substituir CMD del Dockerfile a:

```
CMD ["python3","/home/num.py"]
```

D'aquesta manera creem una nova imatge:

```
$ docker image build -t uoc/alpine-numpy-ex .
```

... i executem directament:

```
$ docker container run --rm uoc/alpine-numpy-ex
```

... i ens retorna el resultat de l'arxiu *num.py*.

Si volem introduir un nou *script* de Python (*atcg.py*) on el seu resultat depengui d'un argument:

```
import sys

filename = sys.argv[1]

filenumb = len(filename)

print (filenumb)
```

... i el copiarem mitjançant Dockerfile:

```
COPY atcg.py /home
```

... i crearem una nova imatge:

```
$ docker image build -t uoc/alpine-atcg .
```

Si el fem córrer directament, tindrem el resultat del CMD (en el nostre cas, la versió de Python); mentre que, si afegim arguments a l'ordre `run`, se sobreescriu CMD i obtenim el resultat del nou *script*:

```
$ docker container run --rm uoc/alpine-atcg python3 /home/atcg.py ATG
```

Si volguéssim tenir aquest nou *script* com ordres per defecte, al Dockerfile hauríem de canviar la línia de CMD pels valors per defecte i crear una nova línia anomenada ENTRYPOINT per localitzar l'*script*:

```
ENTRYPOINT ["python3","atcg.py"]

CMD ["ACTG"]
```

També podem afegir abans de ENTRYPOINT i CMD una entrada per definir el directori de treball:

```
WORKDIR /home
```

D'aquesta manera, si creem una nova imatge `uoc/alpine-entry` i executem el contenidor directament:

```
$ docker container run  --rm uoc/alpine-entry
```

Ens retornarà «4», la longitud de la línia «ACTG» ubicada per defecte al Dockerfile. Si ara, al final de l'ordre *run,* hi afegim un altre *input*, aquest substituirà el de CMD:

```
$ docker container run  --rm uoc/alpine-entry Genetica
```

Retornarà «8».

Podem observar la relació entre ENTRYPOINT i CMD en la següent taula.

Taula 5.

	No ENTRYPOINT	**ENTRYPOINT** exec_entry p1_entry	**ENTRYPOINT** ["exec_entry", "p1_entry"]
No CMD	error, not allowed	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Font: elaboració pròpia.

Resum

En aquest apartat heu après com crear i utilitzar entorns Conda i contenidors Docker. Fer servir aquest tipus d'eines és molt important en la reproductibilitat de processos i resultats. Tant si hem de compartir el nostre codi amb altres persones com si hem de repetir el mateix procediment amb altres mostres en un futur, és crucial poder tenir un seguiment de les eines utilitzades. Aquests procediments milloren la qualitat de la recerca publicada i faciliten el procés de revisió per part d'investigadors externs. Actualment, pràcticament tots els programes depenen d'altres paquets de programes, que evolucionen independentment. La modificació d'una dependència pot derivar en diferents resultats d'un programa o, fins i tot, al seu mal funcionament. Per això, és rellevant mantenir els programes i les seves dependències tal com es van utilitzar per poder assegurar la correcta reproductibilitat. La utilització de petits entorns i contenidors facilita el manteniment i el replicat de processos. A més, tenir els programes aïllats del sistema general permet la seva eliminació i actualització d'una manera més senzilla i neta. Alguns programes poden necessitar certes dependències incompatibles amb altres des de la seva instal·lació en el sistema i, per això, la creació d'entorns o la utilització de contenidors facilita el fet de treballar en diferents configuracions dins el mateix sistema.

Activitats

Conda

1. Creeu un nou entorn de nom «UOC-bioinformatics» que contingui Python 2.7.

2. Instal·leu en aquest entorn ja creat R i el paquet Tidyverse en la seva última versió.

3. Enumereu els paquets instal·lats entorn de l'activitat anterior.

4. Instal·leu el paquet Scipy en un entorn nou a través del canal Bioconda.

5. Creeu un arxiu «environment.yml» d'aquest últim entorn.

Docker

1. Baixeu-vos la imatge d'Ubuntu de Docker Hub, entreu al contenidor de forma interactiva i determineu la versió d'Ubuntu descarregada.

2. Repetiu el procediment anterior amb la versió prèvia a la *latest*.

3. Creeu un Dockerfile on afegiu l'última versió de R al sistema operatiu Centos.

4. Creeu un volum Docker on pugueu passar al contenidor un *script* en *bash* que retorni «Hello World».

5. Repetiu el procediment anterior modificant el missatge a través del terminal en córrer el contenidor i que retorni «Hi, I'm back».

Bibliografia

Björn Grüning i altres (2018). Practical Computational Reproducibility in the Life Sciences. *Cell Systems*, *6*(6), p. 631-635. https://docs.anaconda.com/free/anacondaorg/user-guide/

Carole Goble i altres (2020). FAIR Computational Workflows. *Data Intelligence,* 2, p. 108-121.

Mark D. Wilkinson i altres (2016). The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, 3. 160018

Serghei Mangul i altres (2019). Challenges and Recommendations to Improve the Installability and Archival Stability of Omics Computational Tools. *PLoS Biology*, 17, e3000333.

Sean P. Kane, Karl Matthias (2023). *Docker: Up & Running* (3rd Edition). O'Reilly Media.

Victoria Stodden i altres (2018). An Empirical Analysis of Journal Policy Effectiveness for Computational Reproducibility. *Proceedings of the National Academy of* Sciences, 115, p. 2.584-2.589.

Wade L. Schulz i altres (2016). Use of Application Containers and Workflows for Genomic Data Analysis. *Journal of Pathology Informatics*, *7*(53). https://doi.org/10.4103/2153-3539.197197 (2016).

Yang-Min Kim i altres (2018). Experimenting with Reproducibility: A Case Study of Robustness in Bioinformatics. *Gigascience*, 7, giv077.

Yuxing Yan, James Yan (2018). *Hands-On Data Science with Anaconda*. Packt Publishing https://docs.docker.com/

Zachary D. Stephens i altres (2015). Big Data: Astronomical or Genomical? *PLoS Biol*ogy, *13*, e1002195.

Workflows

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

Els *pipelines* tradicionals estan molt lligats a les infraestructures de computació locals on s'executen. No tenen la capacitat de resumir un procés que s'hagi aturat, tenen poca documentació, no compten amb una traçabilitat dels paràmetres i versions de paquets utilitzats i requereixen instal·lació manual, la qual cosa impedeix una fàcil distribució d'aquest. Per poder solucionar aquests inconvenients s'han creat els Workflow Managers. Aquests permeten la utilització de *pipelines* d'anàlisis complexes en diferents entorns de computació assegurant la màxima reproductibilitat dels processos executats.

Diversos Workflow Managers s'han desenvolupat específicament per als camps de recerca i salut integrant entorns, contenidors i computació al núvol.

Hi ha cinc característiques que fan als Workflow Managers eines de gran utilitat:

1. Reproductibilitat. La utilització d'entorns i contenidors assegura una reproductibilitat apropiada dels processos executats.

2. Portabilitat. És un dels grans avantatges de la utilització de Workflow Managers, ja que crea els fluxos de treball necessaris per poder-se exportar a qualsevol entorn computacional. Molts d'ells permeten la fàcil migració a diferents entorns, inclosos els d'alta computació i serveis al núvol. Encara més, és possible la interacció directa amb orquestradors com Kubernetes o DockerSwarm.

3. Escalabilitat. Ser capaç de manejar i analitzar dades amb una complexitat creixent és cada vegada més comú. En aquest sentit, hi ha dos aspectes que s'han de tenir en compte: el maneig eficient dels recursos i ser capaç d'utilitzar dades més complexes i de major grandària. La majoria de Workflow Managers implementen la paral·lelització en diversos passos, sigui mitjançant gestor de cues o *scheduling* estàtic o adaptatiu. La paral·lelització es pot produir en l'àmbit de dades, processos o *pipelines*. Una assignació dinàmica dels recursos permet que els processos més intensius no es vegin afectats respecte dels que no en requereixen tants. Aquest balanceig minimitza colls d'ampolla i redueix els temps de computació. Els recursos es poden assignar específicament per a cada pas del flux de treball.

4. Robustesa. Molts *pipelines* requereixen processos complexos i de llarga durada. En el possible esdeveniment de la interrupció del *pipeline* en algun procés a causa d'un error, sigui programàtic o per l'absència d'un *input* requerit, els Workflow Managers són capaços de resumir el procés des del lloc on hi va haver l'últim pas correcte, resultant en l'estalvi en la utilització de recursos i temps. Aquest procés s'aconsegueix mitjançant la producció d'arxius i resultats intermedis, essent comparats amb els resultats esperats. Aquest procés genera un augment en les necessitats d'emmagatzematge, però comporta un avantatge substancial en el cas de tenir la necessitat d'una reentrada en el *pipeline*.

5. Modularitat. La compartimentació dels processos permet un gran dinamisme en l'actualització de certs passos del procés, així com de la introducció de punts de control per a cada etapa. La modularitat també permet la reutilització d'un procés en diversos *pipelines* simultàniament.

Finalment, cal indicar que alguns Workflow Managers també tenen recursos per augmentar la seguretat en l'execució dels processos, com la validació de l'origen de les dades o utilitzar autenticació d'usuaris.

Objectius

1. Oferir una visió general dels Workflow Managers.

2. Introduir els lectors a Nextflow.

1. Diferents tipus de *workflows*

La implementació d'un *pipeline* en un Workflow Manager requereix la definició precisa d'on s'extreuen les dades inicials (*input*), quin és el flux entre les diferents eines i quin és el resultat final (*output*). Per poder estructurar els diferents paràmetres s'utilitzen llenguatges de domini específic (DSL). La utilització de DSL augmenta la portabilitat i l'escalabilitat. Nextflow i Snakemake són dos dels exemples més populars de *workflows* que fan servir DSL propis en l'àmbit de la bioinformàtica. La diversitat de DSL amb la seva pròpia estructuració i nomenclatura propicia la reducció d'interoperabilitat entre els diferents *workflows*. Per mitigar aquesta disparitat s'han creat unes especificacions que afegeixen un nivell més gran d'abstracció, permetent un marc comú entre els diferents *workflows*. Exemples d'especificacions per a *workflows* són el Common Workflow Language (CWL) i Workflow Description Language (WDL). CWL prioritza la portabilitat i la reproductibilitat, mentre que WDL té com a principal objectiu reduir la corba d'aprenentatge mitjançant un llenguatge més comprensible. CWL defineix els *pipelines* usant fitxers YAML, mentre que WDL utilitza els seus propis fitxers descriptius. Alguns Workflow Managers, com *cwltool* o Cromwell, s'han creat basant-se en aquestes especificacions, mentre que d'altres com Snakemake implementen opcions d'exportació a aquestes especificacions.

La utilització de Workflow Managers per a la creació de *pipelines* bioinformàtics està creixent en popularitat. Diferents *workflows* estan disponibles per a la comunitat variant en la seva flexibilitat i facilitat d'ús (https://github.com/pditommaso/awesome-pipeline). Mentre que alguns *workflows* necessiten un coneixement avançat de programació, d'altres com Galaxy, KNIME o BioWorkflow implementen una interfície gràfica per facilitar la seva utilització. Tot i que el desenvolupament de sistemes de *workflow* es va iniciar a principis dels anys noranta, ha estat la capacitat de córrer *pipeline*s en clústers de computació o al núvol el que ha facilitat el seu ús més generalitzat.

2. Nextflow

2.1. Introducció

Com a exemple de Workflow Manager en aquesta secció utilitzarem un dels programes més estesos amb una comunitat creixent i amb més suport, com és Nextflow.

En primer lloc, hem d'instal·lar Nextflow seguint les instruccions directament de la seva pàgina web (https://www.nextflow.io/).

Nextflow combina tres elements: un entorn d'execució, un programa específic per llançar altres programes i un DSL específic. L'estratègia que implementa Nextflow en la creació de *pipelines* és la de crear mòduls per a cada pas en el *pipeline* i vehicular-los i interconnectar-los a través de canals. Una de les característiques més importants de Nextflow és la reutilització d'aquests mòduls per a diferents estadis d'un mateix *pipeline* o entre diferents *pipelines*. L'execució dels diferents mòduls no és lineal, sinó que depèn de la disponibilitat dels *inputs* per executar-se. Si dos mòduls independents requereixen els mateixos *inputs* o *inputs* independents es poden executar simultàniament per després poder convergir en un altre posteriorment si es requereix.

2. Nextflow

2.2. DSL2

Nextflow utilitza com a DSL una extensió del llenguatge de programació Groovy. A partir de la versió 20.07.1 de Nextflow es va actualitzar la sintaxi del DSL original creant DSL2. Al llarg d'aquesta secció farem ús d'aquesta nova implementació, DSL2. Aquesta actualització, entre moltes altres característiques, permet la utilització de més d'un *script* per definir el *workflow*, a diferència de DSL1, en el qual s'acumulava tot en un únic *script*. En el codi de Nextflow s'ha d'especificar la utilització de DSL2, ja que per defecte usa DSL1. Per a això s'indicarà a l'inici de l'*script*:

```
nextflow.enable.dsl=2
```

2. Nextflow

2.3. Estructura de *workflows*

Els *workflows* creats per a Nextflow contenen tres parts diferenciades: processos, canals i *workflows*. Un procés executa una tasca. Cada procés és independent de l'altre i pot tenir més d'un canal d'entrada i de sortida. Un canal és un sistema de cues asíncron que permet el flux d'informació entre processos (figura 1). Per ajuntar els diferents processos i el seu flux d'execució (canals) es resumeix en un apartat específic en l'*script* que es denomina *workflow*.

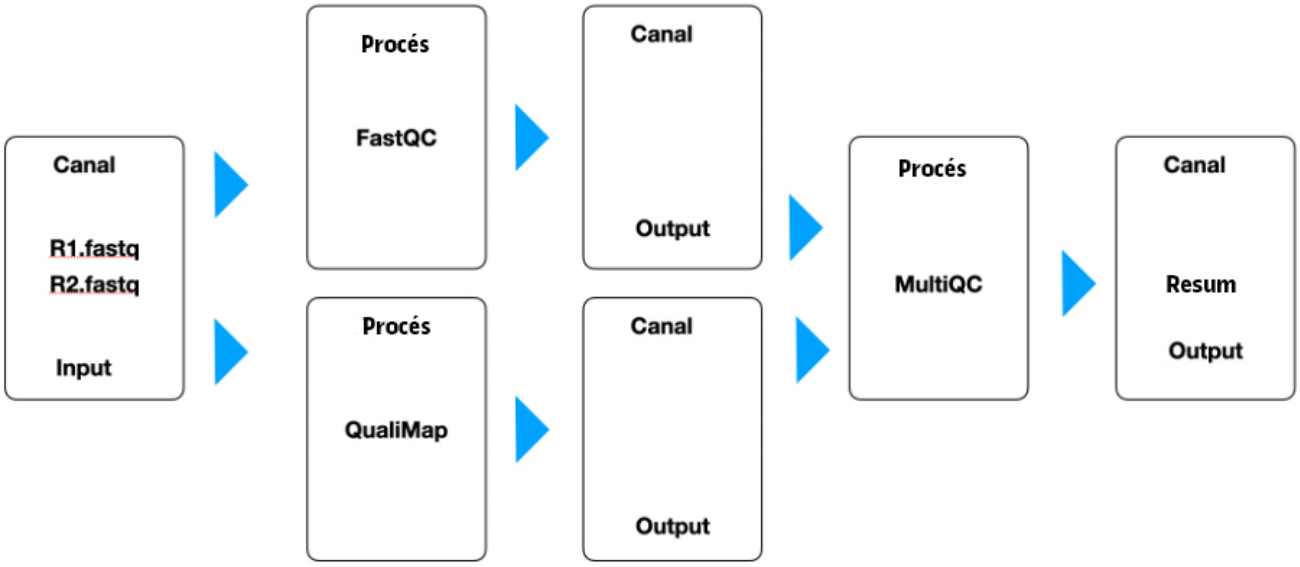


Figura 1. Exemple d'esquema de workflow a Nextflow.
Font: elaboració pròpia.

Nextflow diferencia les ordres que s'executaran dins d'un procés i qui serà l'encarregat d'executar-les. Això permet tenir un marc general que descrigui què es vol fer independentment de les eines que s'utilitzaran per executar-lo. Aquesta estructura permet llançar un procés en diferents entorns computacionals variant simplement un fitxer de configuració, el qual defineix els executors específics de l'entorn on un es trobi.

2. Nextflow

2.4. «Hello world»

Començarem per un *script* senzill (helloworld.nf) on imprimim una frase, per exemple «Hello world».

```
nextflow.enable.dsl=2


params.str = 'Hello world!'


process printStr {

 output:

    stdout


  """

echo '${params.str}'

  """

}



workflow {

 printStr | view ( )

}
```

- En primer lloc, a l'*script* `nf` li indicarem que utilitzem `DSL2`.
- Crearem un paràmetre que anomenarem «Hello world!». El nom del paràmetre està precedit per un punt i l'expressió `params`. En aquesta secció, com que es tracta d'un únic paràmetre, no cal crear un canal, ja que Nextflow assigna directament un canal *value*. Hi ha diferents tipus de canals, com veurem més endavant, i el canal *value* és el més simple.
- El següent segment defineix un procés al qual anomenarem printStr. La nomenclatura dels processos és *process* seguit del nom que li assignem. Els processos estan formats per tres parts: *input*, *output* i *script*. En aquest cas, li indiquem que l'*output* sigui l'estàndard, i en ser un valor simple tampoc cal que definim un canal específicament. Dins del procés definirem què volem fer. Imprimim el paràmetre str.
- Finalment, assignem l'última secció al flux del

Un cop creat l'*script* l'executarem:

```
$ nextflow run helloworld.nf
```

El resultat es mostra a la figura 2.

```
N E X T F L O W  ~  version 20.10.0
Launching `helloworld.nf` [boring_pare] - revision: 6452698d90
executor >  local (1)
[e7/6bcd02] process > printStr [100%] 1 of 1 ✔
Hello world!
```

Figura 2. Execució de l'*script* helloworld.nf.
Font: elaboració pròpia.

Com es pot observar, s'ha executat un procés `printStr` i se'n visualitza el resultat. Si voleu modificar un paràmetre, en aquest cas `str`, des de la línia d'ordres executarem:

```
$ nextflow run helloworld.nf --str 'Bye Bye World'
```

I obtindrem el nou resultat.

Ara li afegirem un segon procés: posar totes les lletres en majúscula. Per això afegirem el procés *allToUpper*.

```
nextflow.enable.dsl=2

params.str = 'Hello world!'

process printStr {

  output:

   path 'test.txt'


  """

echo '${params.str}' > test.txt

  """

}

process allToUpper {

 input:

   path x

  output:

    stdout

  """

cat $x | tr '[a-z]' '[A-Z]'
```

```
  """

}


workflow {

 printStr | allToUpper | view ( )

}
```

I el resultat el visualitzem a la figura 3.

```
N E X T F L O W  ~  version 20.10.0
Launching `helloworld.nf` [small_venter] - revision: 628d37e1e8
executor >  local (2)
[a1/5d17b2] process > printStr   [100%] 1 of 1 ✔
[a9/2c5e3e] process > allToUpper [100%] 1 of 1 ✔
HELLO WORLD!
```

Figura 3. *Output* de dos processos a Nextflow.
Font: elaboració pròpia.

Com podeu comprovar a la secció *workflow*, primer executem `printStr`, posteriorment `allToUpper` i finalment el visualitzem mitjançant `view( ). view` és un operador que veurem més endavant.

Seguidament substituïm l'ordre `allToUpper`:

```
cat $x | tr '[a-z]' '[A-Z]'
```

per:

```
rev $x
```

i tornem a executar l'*script*. En aquest cas, volem imprimir «Hello world!» al revés. Com que el primer pas ja l'havíem llançat anteriorment, podem resumir l'execució mitjançant l'opció *resumeix*:

```
$ nextflow run helloworld.nf -resume
```

i ens generarà *l'output* de la figura 4.

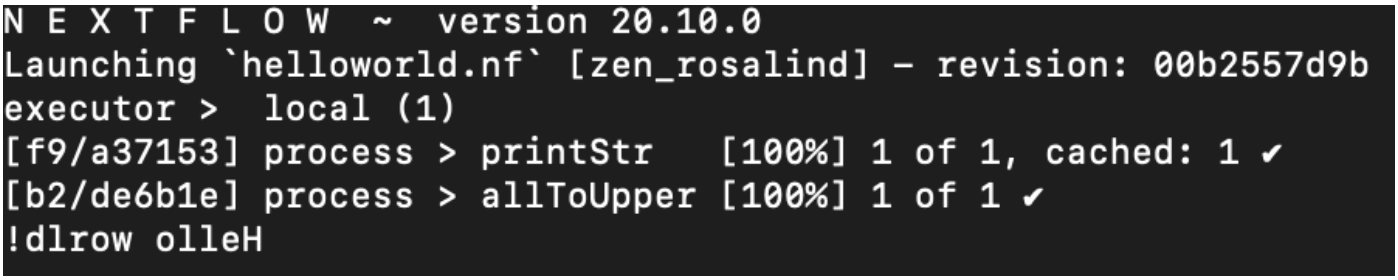


Figura 4. Resumir l'execució de l'*script* helloworld.nf.
Font: elaboració pròpia.

Com podeu veure, el primer pas de *l'script* no ha estat calculat de nou, sinó que s'ha utilitzat el procés ja generat anteriorment (*cached*). D'aquesta manera, si en algun pas del *pipeline* hi ha hagut algun error que ha aturat el procés, es pot resumir prèvia esmena del problema.

2. Nextflow

2.5. Canals a Nextflow

Com s'ha comentat anteriorment, els canals dirigeixen el flux d'informació a través dels diferents processos. Per crear explícitament un canal s'ha d'utilitzar un mètode de Channel Factory proporcionat per Nextflow.

A Nextflow hi ha dos tipus de canals:

- *Queue channels*: són canals asincrònics, unidireccionals i FIFO (*first-in-first-out*).

Alguns exemples són:

```
- of:

     ch = Channel.of(1, 3, 5, 7)


- fromPath:

     file_ch = Channel.fromPath('test/*.txt')


- fromFilePairs:

     fastq = Channel.fromFilePairs('/my/data/SRR*_{1,2}.fastq')
```

- *Value channels* (*singleton channel*): són molt similars als *queue channels*, però només admeten un valor.

```
        value:
pi = Channel.value('3.1416')
```

Per entendre una mica millor el funcionament dels canals crearem un *script* anomenat *orden.nf* amb un canal `value` i un altre `of` i imprimirem el resultat.

```
nextflow.enable.dsl=2
pi = Channel.value(3.1416)
queue_ch = Channel.of( 1, 3, 5, 7 )

process ordenE {
    input:
      val x
      val y
    output:
      stdout
"""
     echo $x $y
```

```
"""
}
workflow{
ordenE(pi,queue_ch) | view( )
}
```

Com podem observar, a l'apartat *workflow* estem especificant els *inputs* del procés `ordenE`. En el resultat d'aquest *script* (figura 5), l'ordre d'aparició dels valors del canal `of` no és el mateix que li hem indicat. Si repetim l'execució, segurament ens sortirà un resultat diferent. Es produeixen quatre processos de forma no correlativa.

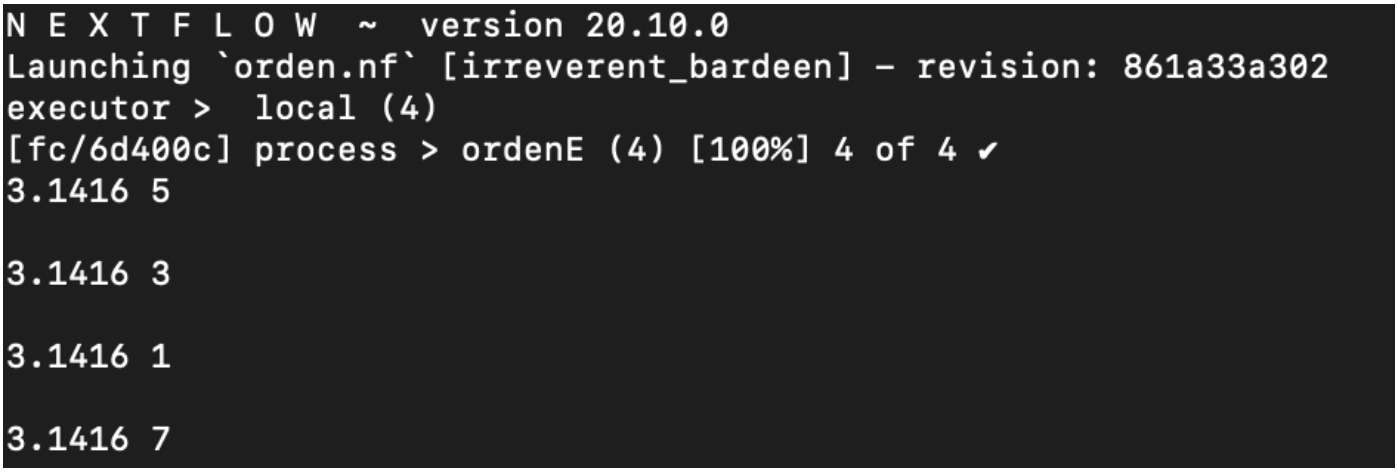


Figura 5. *Output* de l'*script* `orden.nf`.
Font: elaboració pròpia.

Nextflow també és capaç de generar mètriques i reports mitjançant opcions introduïdes via terminal.

Uns exemples serien:

- *with-report*: crea un informe d'execució.
- *with-trace*: generarà un arxiu on s'indiquin paràmetres de l'execució, com memòria i *cpus* utilitzades, inici d'execució...
- *with-timeline*: permet identificar els colls d'ampolla indicant el temps que consumeix cada procés.

```
$ nextflow run orden.nf -with-timeline
```

2. Nextflow

2.6. Operadors

A la secció anterior hem vist com moure canals per dirigir les dades entre els processos. Per poder modificar el contingut o el comportament d'un canal, Nexflow ha creat el que es denominen *operadors*. En els *scripts* anteriors hem vist l'operador *view*, però podem trobar operadors de filtratge, combinació o d'operacions matemàtiques entre molts altres. En aquesta secció en veurem alguns exemples.

Els operadors es poden introduir mitjançant un *pipe* (|), com hem vist anteriorment, o precedits per un punt. Així:

```
workflow{

ordenE(pi,queue_ch) | view( )

}
```

és anàleg a:

```
workflow{

ordenE(pi,queue_ch).view( )

}
```

A partir de l'*script* anterior eliminarem el canal *valor pi* i ens quedarem amb un exemple més senzill amb el canal *of queue_ch*. Com veureu a continuació hi afegim la notació *.view*, i dins d'aquest operador hi introduïm un prefix, *chr*, i un valor, *$it*, entre { }. Aquests parèntesis defineixen un bloc de codi que va al costat i utilitza la nomenclatura de *goovy* (*it*, d'*ítem*) per definir els paràmetres.

```
nextflow.enable.dsl=2

queue_ch = Channel.of( 1, 3, 5, 7 ).view({"chr$it"})


process ordenE {

    input:

      val x


    output:

      stdout


"""

    echo $x
```

```
"""

}

workflow{

ordenE(queue_ch)

}
```

En aquest cas l'*output* es visualitza a la figura 6.

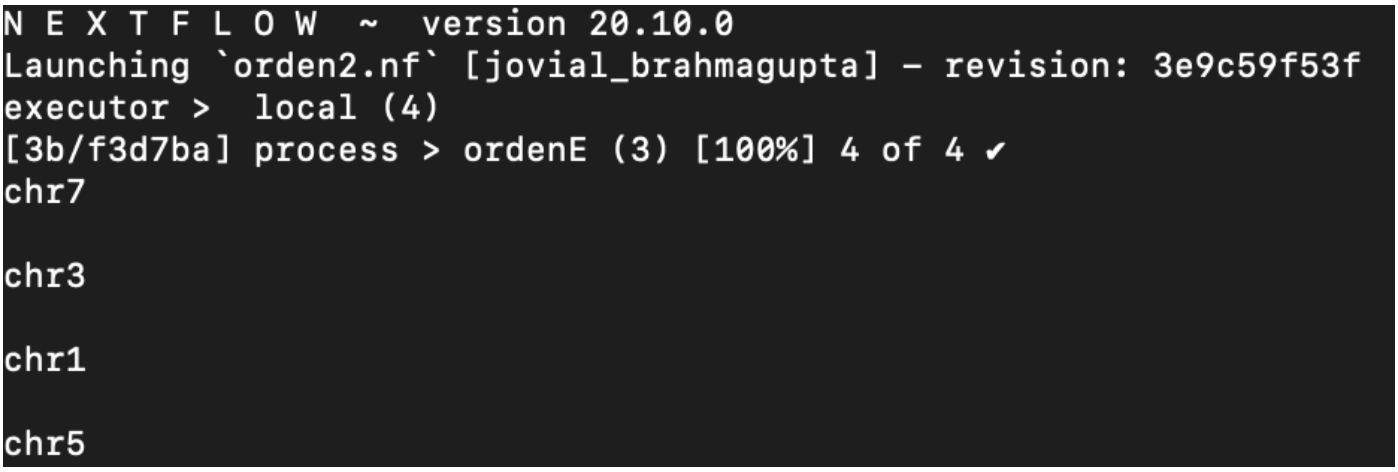


Figura 6. Resultat de l'operador *view* amb prefix.
Font: elaboració pròpia.

Podem introduir un filtre al canal per mostrar únicament els valors superiors a 4:

```
queue_ch = Channel.of( 1, 3, 5, 7 ).filter { it > 4 }.view({"chr$it"})
```

També podem combinar canals utilitzant l'operador *mix*:

```
ch1 = channel.of( 1..22 )

ch2 = channel.of( 'X','Y' )

ch3 = channel.of( 'MT' )


queue_ch  = ch1.mix(ch2,ch3).view({"chr$it"})
```

i fer operacions com comptar el nombre d'elements:

```
queue_ch  = ch1.mix(ch2,ch3).count().view()
```

Les possibilitats dels operadors proporcionen una versatilitat molt gran de poder manipular les dades que volem analitzar.

2. Nextflow

2.7. Configuració

Finalment tractarem els arxius de configuració de Nextflow. Aquests arxius són rellevants per poder migrar els *scripts* entre entorns de computació i per al control dels recursos a utilitzar.

Podem trobar diversos arxius de configuració i alguns poden entrar en conflicte entre ells. Per això Nextflow té una prioritització:

1. Paràmetres especificats a la línia d'ordres.
2. Paràmetres procedents de l'arxiu especificat mitjançant l'opció -params-file.
3. Arxiu de configuració especificat mitjançant l'opció -c my_config.
4. Arxiu de configuració nexflow.config en el directori de treball.
5. Arxiu de configuració nexflow.config en el directori del projecte de *workflow*.
6. Paràmetres definits en el mateix *script* de Nextflow.

Si un paràmetre es troba en més d'una d'aquestes fonts, Nextflow fa servir la primera font com a referència i no fa servir les subsegüents.

L'arxiu consta de parelles nom = valor

```
process.memory = '10G'
```

Un arxiu de configuració es pot incloure en un altre. Per exemple:

```
includeConfig 'path/foo.config'
```

La instrucció `includeConfig` busca l'arxiu de paràmetres i els inclou com a propis.

Els paràmetres de configuració es poden especificar en el que s'anomenen *scopes*. Són paràmetres que afecten específicament un tipus de configuració. Hi ha diversos *scopes* de configuració, els més habituals dels quals són:

- *aws*: Amazon S3.
- *conda*: entorns Conda.
- *docker*: contenidors Docker.
- *k8s*: clúster de Kubernetes.

Per habilitar la utilització d'una imatge Docker, per exemple, es podria introduir al nextflow.config:

```
docker.enabled = true
```

o es podria especificar directament a la línia d'ordres:

```
nextflow run <script> -with-docker [imagen Docker]
```

També és possible especificar una imatge Docker per a un procés determinat:

```
process uno {

  container 'nombre_imagen_1'


  '''

  execució

  '''

}


process dos {

  container 'nombre_imagen_2'


  '''

 execució

  '''

}
```

2. Nextflow

2.8. *nf-core*

Com altres iniciatives que ja hem tractat anteriorment, Nextflow també té un repositori de *workflows* per poder ser executades. Aquest conjunt de *pipelines* s'agrupa en el projecte *nf-core* (https://nf-co.re/). Per poder treballar directament des de el terminal s'ha generat el paquet *nf-core tools* per poder gestionar els diferents *workflows* accessibles en *nf-core*. *nf-core tools* està escrit a Python i es pot instal·lar des de la següent adreça: https://nf-co.re/tools.

Per poder veure els diferents paquets simplement es pot fer un llistat:

```
$ nf-core list
```

Moltes vegades necessitem ser més específics en la nostra recerca i, per això, podem filtrar:

```
$ nf-core list rna
```

O fins i tot ordenar per estrelles:

```
nf-core list rna --sort stars
```

Cada repositori té la seva pròpia pàgina amb instruccions i documentació. La utilització d'aquests repositoris públics permet un aprenentatge més ràpid de les eines de Nextflow en el context específic de la bioinformàtica. Igualment, *nf-core* també proporciona un canal directe de preguntes mitjançant un compte a Slack (https://nf-co.re/join).

Resum

En aquest mòdul hem donat una breu pinzellada als Workflow Managers i en especial a Nextflow. Per augmentar el control dels entorns en els quals es computen les anàlisis és imprescindible utilitzar Workflow Managers, i, a poder ser, conjuntament amb entorns tipus Conda o contenidors, com hem vist en capítols anteriors. La utilització d'aquest tipus d'eines permet la portabilitat a qualsevol entorn de computació i la implementació de *pipelines* generats per altres grups d'una manera àgil i senzilla. La modularització dels processos permet que la seva actualització pugui ser dinàmica i que la introducció de codi generat per altres programadors no interfereixi en altres parts del procés. La versatilitat d'opcions dels Workflow Managers permet un control precís de tot el procés, de vegades en detriment de la seva corba d'aprenentatge. La diversitat de Workflow Managers que actualment tenim permet poder escollir l'ecosistema en què ens trobem més confortables, variant des d'entorns molt visuals tipus Galaxy a altres enterament programàtics. És important arribar a un compromís entre l'aprenentatge necessari per entendre el funcionament d'un *workflow* específic i les eines que ens permet controlar. Finalment, a l'hora de triar el vostre Workflow Manager s'ha de tenir en compte la potencial interoperabilitat. En entorns col·laboratius és imprescindible la reproductibilitat de processos i la compartició de codi per poder seguir els principis FAIR de les dades.

Activitats

1. Creeu un *script* Nextflow que transformi la paraula «HOLA» (en majúscules) tota en minúscules.

2. Modifiqueu a través de la crida de l'*script* la paraula «HOLA» per «CIAO».

3. Introduïu un nou procés que substitueixi *l'A* per un 4.

4. Creeu un *script* amb dos canals *of* i un procés que sumi els valors de cada canal i imprimeixi l'operació realitzada.

5. Busqueu i baixeu un *workflow* a través de *nf-core*.

Bibliografia

Bjørn Fjukstad i Lars Ailo Bongo (2017). A Review of Scalable Bioinformatics Pipelines. Data Science and Engineering, 2, p. 245-251. https://doi.org/10.1007/s41019-017-0047-z

Ed H. B. M. Gronenschild i altres (2012). The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements. *PLoS ONE*, 7, e38234. Doi: 10.1371/journal.pone.0038234

Elise Larsonneur, Jonathan Mercier i Nicolas Wiart (2018). Evaluating Workflow Management Systems: A Bioinformatics Use Case. 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), p. 2.773-2.775. Doi: 10.1109/BIBM.2018.8621141

Jeffrey M. Perkel (2019). Workflow Systems Turn Raw Data into Scientific Knowledge. *Nature*, 573, p. 149-150. Doi: 10.1038/d41586-019-02619-z

Jeremy Leipzig (2017). A Review of Bioinformatic Pipeline Frameworks. *Briefings in Bioinformatics*, 18, p. 530-536. Doi: 10.1093/bib/bbw020

Paolo Di Tommaso i altres (2017). Nextflow Enables Reproducible Computational Workflows. *Nature Biotechnology*, 35, p. 316-319. https://doi.org/10.1038/nbt.3820

Paolo Di Tommaso i altres (2015). The Impact of Docker Containers on the Performance of Genomic Pipelines. PeerJ, 3, e1273. https://doi.org/10.7717/peerj.1273

Philip Ewels i altres (2020). The nf-core Framework for Community-curated Bioinformatics Pipelines. *Nature Biotechnology*, 38, p. 276-278. Doi: 10.1038/s41587-020-0439-x

Taylor Reiter i altres (2021). Streamlining Data-intensive Biology with Workflow Systems. *Gigascience*, 10, giaa140. https://doi.org/10.1093/gigascience/giaa140

Gestió de dades

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

Els fitxers són les unitats lògiques d'informació persistent dins d'un ordinador. Com que manquen d'estructura pròpia, és el programador qui estableix com organitzar la informació en el seu interior (per exemple, fitxers FASTA, XML o JSON).

Mitjançant les ordres apropiades del terminal o línia d'ordres, podem analitzar el contingut dels fitxers d'una manera relativament eficient i còmode. En el mòdul «Introducció als entorns de treball GNU/Linux» hem treballat amb ordres del terminal per gestionar la informació emmagatzemada en fitxers de text que contenen anotacions biològiques. Tanmateix, quan el volum d'informació excedeix certs límits, com és el cas de l'anotació completa d'un genoma, i s'incrementa el nombre de persones involucrades en un projecte de recerca, cal organitzar i estructurar la informació en una base de dades i gestionar-la utilitzant un programa especialitzat, també anomenat Sistema de Gestió de Bases de Dades (SGBD).

Un SGBD és l'eina idònia per administrar eficientment elevades quantitats de registres o dades. La responsabilitat sobre la gestió i els formats interns de les dades correspon al sistema, la qual cosa allibera el propi usuari d'aquestes tasques.

Amb un sistema de gestió de la informació tindrem a la nostra disposició un conjunt d'eines i instruccions que ens permetran extreure nou coneixement de tota aquesta informació.

En aquest mòdul veurem dos models de gestió de les dades, el model relacional basat en el llenguatge SQL, i utilitzarem el SGBD MySQL, i el model no relacional, també anomenat NoSQL, i utilitzarem el SGBD MongoDB basat en col·leccions de documents.

Objectius

1. Conèixer el model entitat-relació per dissenyar bases de dades.
2. Dominar la conversió d'entitats i relacions en taules amb atributs.
3. Crear i administrar una base de dades amb MySQL.
4. Entendre les característiques bàsiques del llenguatge SQL.
5. Crear i poblar amb registres reals les taules SQL.
6. Realitzar consultes SQL a una base de dades relacional.
7. Conèixer la utilitat dels subprogrames emmagatzemats, procediments, funcions i disparadors (Triggers).
8. Iniciar-se amb els fitxers JSON i el SGBD NoSQL MongoDB.
9. Administrar un sistema de gestió de base de dades.

1. Bases de dades relacionals

1.1. Introducció

El model relacional és un model de dades basat en la lògica de predicats i en la teoria de conjunts. La seva idea fonamental és l'ús de relacions. Aquestes relacions podrien considerar-se de forma lògica, com a conjunts de dades anomenades *tuples*. Pensem cada relació com si fos una taula que està composta per registres: cada fila de la taula seria un registre o *tupla*, i columnes, també anomenades *camps*.

Entre els paradigmes actuals de bases de dades, el model relacional està molt estès i s'adapta a la majoria d'entorns bioinformàtics per la seva eficiència i simplicitat.

Un altre avantatge d'aquest paradigma és que existeixen nombroses implementacions *open source* que proporcionen els serveis complets d'un sistema gestor de base de dades relacional amb diferents interfícies gràfiques d'usuari.

Formalment, el paradigma relacional està dividit en tres components bàsics:

- Les taules i les relacions entre aquestes estructuren les dades.
- L'àlgebra relacional opera sobre la informació.
- Un conjunt d'axiomes manté la integritat del sistema.

Una taula modela un element del món real, caracteritzant els seus atributs. Una relació entre dues taules emula les associacions lògiques existents entre dos elements de diferents classes en la realitat, permetent l'accés creuat d'informació.

Per a un univers de dades en particular, l'organització de les taules i les relacions que el conformen reben el nom d'esquema relacional. Una vegada definida aquesta estructura, s'ha de crear una base de dades per ser poblada amb les dades reals (conegudes com a instàncies o registres), sent administrada des d'aquell moment per un sistema de gestió de bases de dades.

Utilitzant l'àlgebra relacional, l'usuari pot realitzar consultes per extreure'n nova informació i actualitzar-la.

Per realitzar un disseny eficient de la base de dades hem de seguir aquestes regles:

- Estimular totes les classes d'informació que desitgem guardar.
- Estructurar de forma lògica la informació en diferents categories.
- Definir els atributs que caracteritzen cada categoria.
- Assignar identificadors suficientment descriptius als atributs.
- Decidir el tipus de dades associat a cada atribut.
- Descompondre cada peça d'informació en la unitat més elemental.
- Seleccionar els atributs que identifiquen de forma única cada categoria.
- Identificar les relacions entre categories.

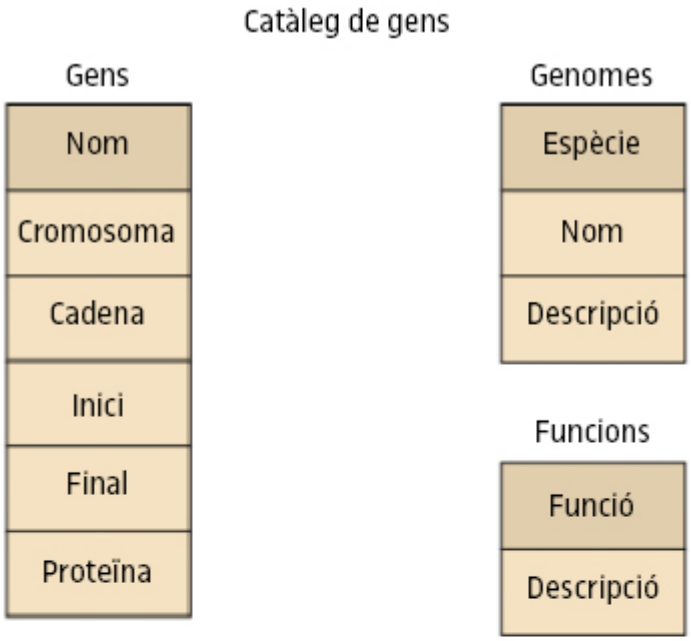
1. Bases de dades relacionals

1.2. El model entitat-relació

Podem estructurar qualsevol realitat en diferents entitats que poden interactuar entre elles seguint determinades regles. Per exemple, el genoma, en tant que part de la realitat biològica d'una cèl·lula, també es pot estructurar en diferents components. A partir d'aquesta organització artificialment construïda, podem modelar la totalitat dels elements que el conformen utilitzant entitats i relacions. Aquestes estructures, contenidors d'informació o dades poden ser digitalitzats en un ordinador per a la seva gestió i anàlisi; en el cas del genoma, anàlisis bioinformàtics.

El model entitat-relació ens ajuda a dissenyar la nostra pròpia base de dades. Una entitat representa una mena d'elements en l'entorn real que desitgem modelar. Una ocurrència és una instància o exemple particular d'una entitat. La connectivitat o participació entre entitats s'ha d'especificar explícitament mitjançant relacions. El nombre d'ocurrències d'una entitat que podran relacionar-se amb instàncies d'una altra entitat serà:

- Un a un (1:1): un element de la primera entitat pot relacionar-se amb un únic element de la segona.
- Un a diversos (1:N): un element de la primera entitat pot relacionar-se amb diversos elements de la segona (però no al contrari).
- Diversos amb diversos (M:N): un element de la primera entitat pot relacionar-se amb diversos elements de la segona (i viceversa).



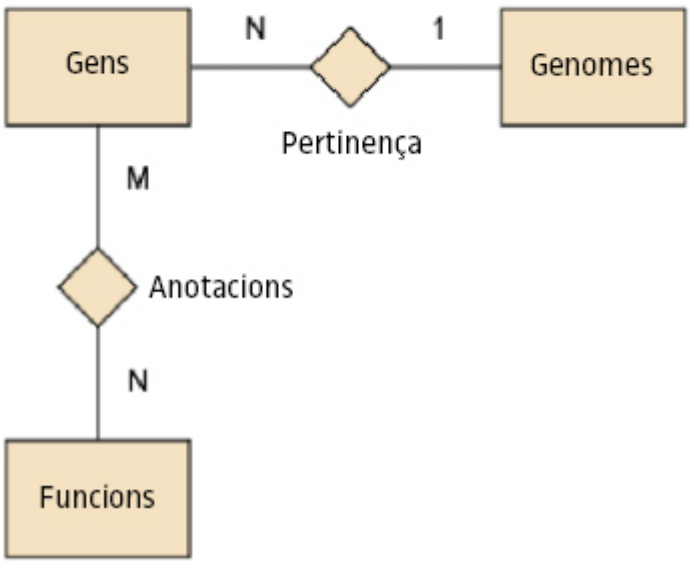


Figura 1. Model entitat-relació d'un catàleg de gens.
Font: elaboració pròpia.

El model conté tres entitats i dues relacions. Gràficament, les entitats es representen utilitzant rectangles, i les relacions, mitjançant línies rectes amb un rombe per indicar la connectivitat.

Aquest esquema basat en entitats i relacions resulta senzill d'emprar posteriorment per construir la base de dades definitiva. Per mostrar les diferents etapes de disseny, procedirem a modelar un escenari molt habitual en entorns de recerca bioinformàtics: la caracterització del catàleg de gens d'un genoma. Podeu observar les diferents entitats amb els seus atributs i les relacions entre entitats que han de formar el nostre model en la figura 1.

Els gens són fragments d'ADN ubicats en una localització precisa del genoma que codifiquen la seqüència d'una proteïna. Aquestes molècules, d'altra banda, exerceixen una funció biològica específica dins de l'organisme (*). Lògicament, cada genoma posseeix el seu propi catàleg de gens. Analitzant aquesta informació prèvia, decidim modelar aquest entorn utilitzant les entitats **gens**, **genomes** i **funcions**, amb els seus propis atributs (mostrats en la figura 2).

Lògicament, les entitats no són objectes aïllats del seu entorn. Per tant, hem de complint les especificacions del nostre problema, unir mitjançant relacions aquelles entitats que estan interconnectades en el món real. En aquest cas, els gens tenen la capacitat de pertànyer a un genoma per desenvolupar una funció concreta en l'organisme. Per satisfer ambdues propietats, definim les relacions binàries **pertinença** i **anotacions**, cadascuna amb una connectivitat diferent (figura 1). Les entitats i relacions que formen aquest model han de ser convertides en taules. Les entitats i els seus atributs passaran a ser taules de la nostra base de dades, però no totes les relacions seran taules, depèn del tipus de connectivitat. És bàsic establir per a cada taula un atribut especial (o una combinació d'atributs) que identifiqui cada instància de forma unívoca. Aquest atribut especial rep el nom de clau primària. És preferible usar un codi característic en lloc d'un nom per facilitar la identificació de qualsevol instància mitjançant la clau primària (per exemple, un codi numèric assignat en funció de l'ordre d'entrada a la taula).

```
GENES (nombre, cromosoma, hebra, inicio, final, proteina)
GENOMAS (especie, nombre, descripcion)
FUNCIONES (funcion, descripcion)
```

Figura 2. Entitats convertides en taules.
Hem subratllat la clau primària de cada taula.
Font: elaboració pròpia.

Les dues relacions existents en el nostre esquema s'han de modelar de diferent forma, atès que cadascuna presenta una combinació diferent de possibles ocurrències entre taules. L'associació entre les taules *gens* i *genomes* s'ha de representar amb una relació amb cardinalitat 1:N, ja que un gen només pertany a un genoma, però un genoma conté molts gens. Aquesta relació no necessita una taula nova, pot implementar-se referenciant simplement des d'una taula (gens), que és la part *n* de la relació *pertinença*, la clau primària de l'altra (genomes), que és la part 1 de la relació *pertinença*. Dins de la taula *gens*, aquest atribut rep la denominació de clau forana.

```
GENES (nombre, cromosoma, hebra, inicio, final, proteina, especie)
GENOMAS (especie, nombre, descripcion)
```

Figura 3. Relacions (1:N) convertides en claus foranes.
Indiquem amb un subratllat superior la clau forana de cada taula.
Font: elaboració pròpia.

La relació *anotacions* entre les taules *gens* i *funcions* té una connectivitat M:N, ja que un gen pot posseir diverses anotacions, però una anotació també pot ser compartida per diversos gens. Per a la seva correcta implementació, hi introduirem una nova taula anomenada *anotacions*. Aquesta posseirà, com a clau primària, la combinació de les claus primàries de cada taula original.

Atès que ambdues claus primàries per separat posseeixen totes les propietats necessàries, el dissenyador garanteix amb aquesta mesura que cada instància d'aquesta nova taula estarà dotada d'un identificador únic, que estarà format pel nom del gen juntament amb el nom de la funció en particular perquè sigui un identificador de la instància únic que no es pugui repetir.

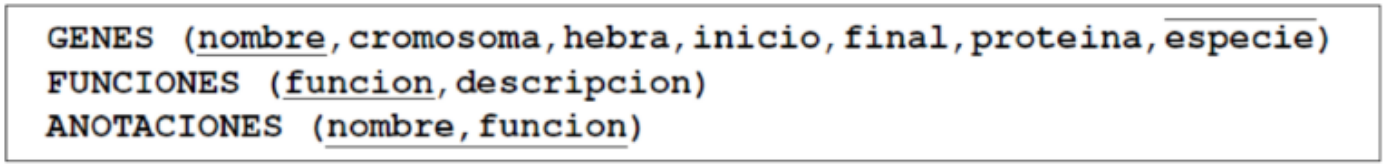


Figura 4. Relacions (M:N) convertides en taules.
Font: elaboració pròpia.

Les relacions amb cardinalitat M:N exportades des del model entitat-relació estan caracteritzades també pels seus propis atributs. Així, és possible afegir un camp per guardar l'origen de cada anotació (per exemple, computacional, experimental o font bibliogràfica). Podem extreure aquesta informació a partir de les dades recuperades del sistema d'anotació automàtica utilitzat per poblar d'exemples el nostre catàleg de gens:

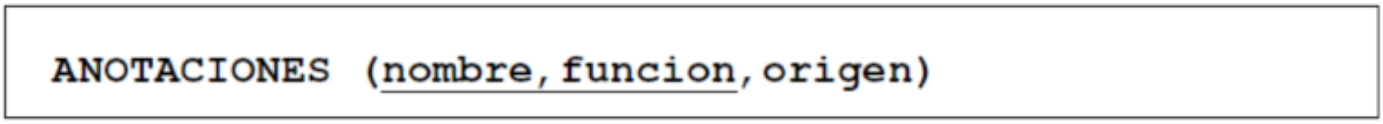


Figura 5. Atributs en relacions convertides en taules.
Font: elaboració pròpia.

La selecció de les claus idònies resulta essencial dins del disseny d'una base de dades relacional. De fet, la integritat d'un model relacional ha de complir dos requisits fonamentals relacionats amb la gestió d'aquestes:

1. La clau primària no ha de contenir un valor indefinit o nul i el valor ha de ser únic.

2. La clau forana ha de fer referència a una clau primària d'una altra taula.

1. Bases de dades relacionals

1.3. El llenguatge SQL i el SGBD MySQL

L'SQL (en anglès, *Structured Query Language*, 'llenguatge de consultes estructurades') és el llenguatge d'accés a les bases de dades relacionals més estès. Amb aquest sistema, el client especifica les instruccions per crear, dotar de contingut, modificar o eliminar les taules de la base de dades, i realitzar les consultes.

Per treballar amb el llenguatge SQL és necessari disposar d'un sistema gestor de bases de dades (SGBD), una aplicació informàtica normalment basada en el model client-servidor per administrar bases de dades relacionals.

Existeixen diferents SGBD, com Oracle, PostgreSQ... En aquest mòdul farem servir el SGBD MySQL. A Linux, el servidor MySQL funciona en segon pla, sense interferir en la planificació de processos del sistema. D'aquesta manera, quan l'usuari desitja utilitzar la base de dades, s'ha de connectar amb l'aplicació gestora mitjançant un programa client, a través d'un entorn gràfic o des del propi terminal amb l'intèrpret d'ordres de SQL, denominat **mysql**.

SQL va ser comercialitzat per IBM el 1981. MySQL és un gestor distribuït per Oracle sota llicència GNU o comercial.

1. Bases de dades relacionals

1.4. Començar a treballar amb MySQL

A la màquina virtual que us proporcionem hi ha instal·lat un SGBD MySQL. En iniciar la màquina virtual també s'inicia el servidor MySQL.

El sistema client-servidor permet accedir a un únic servidor des de diversos clients. Per defecte, el client del sistema és un terminal o línia d'ordres, però es pot accedir també al servidor des d'un client amb interfície gràfica d'usuari (GUI).

A la màquina virtual hi ha instal·lada la GUI **MySQL Workbench**, però és possible accedir al servidor MySQL des de molts clients amb diferents GUI.

Per començar a treballar amb MySQL utilitzarem el terminal de Linux executant l'ordre mysql.

Una vegada establerta la connexió al servidor, el programa client roman sempre a l'espera de la introducció d'una nova ordre SQL per part de l'usuari.

És important ser curós amb la sintaxi de les ordres, i finalitzar cada instrucció amb el símbol « **;** ».

Per il·lustrar el funcionament del joc d'instruccions de SQL mostrat a la taula 1 sobre el model relacional anterior (vegeu figura 4), implementarem una base de dades que denominarem **catalogo**.

Taula 1. Manual de referència d'ordres de MySQL.

Ordre	Descripció
`CREATE USER`	Donar d'alta un nou usuari
`DROP USER`	Donar de baixa un usuari existent
`ALTER USER`	Modificar el compte d'un usuari
`GRANT`	Autoritzar un usuari sobre una base de dades
`REVOKE`	Revocar les autoritzacions d'un usuari
`SHOW GRANTS`	Mostrar les autoritzacions d'un usuari
`CREATE DATABASE`	Crear una nova base de dades
`DROP DATABASE`	Eliminar una base de dades existent
`USE DATABASE`	Accedir a una base de dades existent
`SHOW DATABASES`	Mostrar la llista de les bases de dades
`CREATE TABLE`	Crear una nova taula
`DROP TABLE`	Eliminar una taula existent
`SHOW TABLES`	Mostrar la llista de les taules
`DESCRIBE`	Mostrar els atributs d'una taula
`LOAD DATA`	Poblar una taula amb un fitxer de registres
`INSERT`	Poblar una taula amb un registre
`UPDATE`	Actualitzar un registre de la taula
`DELETE`	Esborrar un registre de la taula
`SELECT`	Realitzar una consulta sobre una o més taules

`Help`	Mostrar ajuda sobre una ordre del sistema
`Pager`	Mostrar llistats pàgina a pàgina amb un altre programa
`Source`	Executar un *script* d'ordres de SQL
`Status`	Mostrar informació sobre l'estat del sistema
`System`	Executar una ordre del terminal
`Warnings`	Mostrar avisos del sistema
`Quit`	Sortir del gestor MySQL
`Exit`	Sortir del gestor MySQL
`Mysql`	Ordre del terminal per invocar el gestor MySQL
`Mysqldump`	Ordre del terminal per al *backup* d'una base de dades

Font: elaboració pròpia.

El SGBD MySQL permet gestionar múltiples bases de dades, implementant un sistema de seguretat basat en autoritzacions. Un cop instal·lat el servidor al nostre entorn Linux, es crea inicialment un usuari amb el rol d'administrador (*) (en anglès, *root*) amb tots els permisos. Per regla general, només l'usuari amb aquest perfil posseeix els permisos suficients per gestionar el conjunt d'usuaris i bases de dades en la seva totalitat. D'aquesta manera, l'administrador garantirà l'accés a una determinada base de dades exclusivament a un grup d'usuaris establert prèviament. Malgrat que es pot optar per treballar directament com a administrador, sempre és recomanable que configurem un nou compte al nostre nom com a usuari convencional per autoritzar-lo posteriorment a treballar amb una base de dades en concret.

En definitiva, el protocol que hem de seguir per dur a terme la nostra feina amb el gestor de bases de dades consisteix en dues fases:

1. Invoquem l'intèrpret de SQL com a administrador, realitzem les següents tasques i abandonem el programa:
    1. Creem un nou usuari al nostre nom.
    2. Creem una nova base de dades.
    3. Autoritzem el nou usuari a treballar amb aquesta base de dades.
2. Accedim a l'intèrpret de SQL amb el nostre propi usuari i procedim a interactuar amb la nova base de dades:
    1. Especifiquem que anem a treballar amb aquesta base de dades.
    2. Creem noves taules (buides) dins de la base de dades.
    3. Inserim nous registres en aquestes taules. També podem modificar-los i eliminar-los.
    4. Realitzem consultes sobre els registres de les taules per obtenir la informació desitjada.

Per començar a treballar primer hem d'executar el programa **mysql** des del nostre terminal de Linux emprant l'usuari *root*:

```
% mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 5
Server version: 5.7.17-0ubuntu0.16.04.1 (Ubuntu)
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All
rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or
its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or 'h' for help. Type 'c' to clear the current
input statement.

mysql>
```

Figura 6. Iniciar el gestor MySQL com a administrador.
Font: elaboració pròpia.

L'ordre CREATE USER permet a l'usuari ***root*** donar d'alta un nou usuari en el nostre sistema. És recomanable assignar una contrasenya a cada nou usuari del nostre sistema:

```
CREATE USER usuario

IDENTIFIED BY password;
```

Figura 7. Sintaxi de l'ordre CREATE USER.
Font: elaboració pròpia.

Ara, actuant com a administradors, crearem un nou usuari anomenat **eblanco**. Per indicar que aquest usuari treballarà localment des de la nostra màquina, que actua de servidor, emprarem el terme **local**.

En alguns exemples, perquè siguin més comprensibles, estructurarem les instruccions de SQL en diverses línies. Després d'introduir cada línia de l'ordre pressionant la tecla «Enter», l'intèrpret de MySQL inserirà els símbols -> per denotar que encara no hem acabat d'introduir l'ordre completa. MySQL no procedirà a executar l'ordre fins a reconèixer el caràcter «;».

Us animem a reproduir al vostre ordinador les ordres de SQL presentades en aquesta unitat.

```
mysql> CREATE USER 'eblanco'@'localhost'
    -> IDENTIFIED BY '123456';

Query OK, 0 rows affected (0,35 sec)
```

Figura 8. Crear un nou usuari amb l'administrador.
Font: elaboració pròpia.

Existeixen diverses ordres per gestionar el conjunt de bases de dades del sistema (consulteu taula 1). En aquest moment hem de procedir a crear la base de dades on treballarà el nou usuari amb l'ordre CREATE DATABASE:

```
CREATE DATABASE basededatos;
```

Figura 9. Sintaxi de l'ordre CREATE DATABASE.
Font: elaboració pròpia.

A continuació, creem la base de dades **catalogo** i posteriorment emprem l'ordre SHOW DATABASES per obtenir el llistat de bases de dades existents. Podem comprovar que la nova base de dades ha estat creada correctament:

```
mysql> CREATE DATABASE catalogo;

Query OK, 1 row affected (0,00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| catalogo           |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0,10 sec)
```

Figura 10. Creació de la base de dades *catalogo*.
En finalitzar l'execució d'una ordre, l'intèrpret mostra per pantalla el nombre d'elements dels resultats (en anglès, *rows*).
Font: elaboració pròpia.

L'administrador concedeix permisos sobre les operacions que un determinat grup d'usuaris pot realitzar sobre la base de dades. L'ordre GRANT permet autoritzar l'accés d'un usuari a una base de dades en particular:

```
GRANT operaciones ON basededatos

TO usuario IDENTIFIED BY password;
```

Figura 11. Sintaxi de l'ordre GRANT.
Font: elaboració pròpia..

Per tancar la feina de l'administrador, hem de garantir l'accés a la nova base de dades **catalogo** al nostre usuari **eblanco** emprant l'ordre GRANT. Amb la clàusula **ALL**, l'administrador concedeix el conjunt complet de permisos a aquest usuari, encara que exclusivament sobre aquesta base de dades. Si volem eliminar privilegis a un usuari hem d'utilitzar l'ordre REVOKE.

Finalment, per abandonar l'execució del programa **mysql** com a administradors, podem emprar les ordres `quit` o `exit`.

```
mysql> GRANT ALL ON catalogo.* TO 'eblanco'@'localhost';

Query OK, 0 rows affected (0,21 sec)

mysql> quit;
```

Figura 12. Autoritzar l'accés a un nou usuari.
Font: elaboració pròpia.

A partir d'aquest instant, a l'hora d'executar el programa **mysql** treballarem sobre la nostra base de dades amb el nou usuari **eblanco**. En primer lloc, hem d'accedir al gestor de MySQL emprant el nom d'usuari i la contrasenya que hem creat.

```
% mysql -u eblanco -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 8
Server version: 5.7.17-0ubuntu0.16.04.1 (Ubuntu)
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All
rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or
its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or 'h' for help. Type 'c' to clear the current
input statement.

mysql>
```

Figura 13. Iniciar el gestor MySQL com un usuari convencional.
Font: elaboració pròpia.

Com que és la primera vegada que hi accedim amb aquest usuari, obtindrem informació sobre el llistat de les bases de dades accessibles utilitzant l'ordre SHOW DATABASES.

Com mostrem a continuació, en la figura 14, el nostre nou usuari pot treballar amb dues bases de dades: **catalogo** i una segona base de dades (**information_schema**) que conté informació interna sobre la configuració del sistema.

Amb l'ordre SHOW GRANTS podem veure també les autoritzacions que l'administrador ha concedit a aquest usuari.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| catalogo           |
+--------------------+
2 rows in set (0,00 sec)

mysql> SHOW GRANTS;
+-----------------------------------------------------------------+
| Grants for eblanco@localhost                                    |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'eblanco'@'localhost'
|
| GRANT ALL PRIVILEGES ON `catalogo`.* TO 'eblanco'@'localhost'
|
+-----------------------------------------------------------------+
2 rows in set (0,00 sec)
```

Figura 14. Conèixer les bases de dades disponibles.
Font: elaboració pròpia.

Qualsevol usuari, una vegada dins del sistema, ha d'especificar el nom de la base de dades que ha d'utilitzar abans de començar a realitzar operacions sobre ella. Per indicar el nom de la base de dades que seleccionarem, farem servir l'ordre **USE**:

```
USE basededatos;
```

Figura 15. Sintaxi de l'ordre USE.
Font: elaboració pròpia..

Com que desitgem treballar amb la base de dades **catalogo**, procedim a declarar aquest fet en l'intèrpret de MySQL. Un cop aquesta instrucció ha estat executada amb èxit, ja estem en disposició de crear les taules que serveixen com a suport de les entitats i les relacions dissenyades amb anterioritat per poblar-les posteriorment amb nous registres.

```
mysql> USE catalogo;

Database changed
```

Figura 16. Seleccionar la base de dades per treballar.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.5. Creació de taules i restriccions

Una base de dades està formada per un conjunt de taules que ens permetran estructurar la informació que percebem en un escenari concret del món real. Cada taula emmagatzemarà en forma de registres la sèrie d'exemples de cada classe d'entitats o relacions entre entitats especificades prèviament. Per crear una taula de registres buida amb l'ordre CREATE TABLE cal primer declarar els seus atributs (consulteu la figura 17).

```
CREATE TABLE nombre
(campo1 tipo1 [NOT NULL, AUTO_INCREMENT],
 campo2 tipo2 [NOT NULL, AUTO_INCREMENT],
 ...
 campoN tipoN [NOT NULL, AUTO_INCREMENT],
 PRIMARY KEY(campox, campoy, ...),
 [FOREIGN KEY(campox, campoy, ...)
 REFERENCES tabla(campoa, campob, ...)]);
```

Figura 17. Sintaxi de l'ordre CREATE TABLE.
Els elements opcionals es mostren entre claudàtors.
Font: elaboració pròpia.

A l'hora de definir la classe d'informació que emmagatzemarem en cada atribut, MySQL proporciona una gran varietat de tipus numèrics i alfanumèrics bàsics (taula 2). L'espai de memòria requerit per emmagatzemar cada variable depèn de la precisió especificada en cada cas. Els tipus DATE i TIME resulten especialment útils per portar el registre de les nostres activitats en el temps. L'usuari pot declarar, a més, variables del tipus objecte (en anglès, *Binary Large Objects* o BLOB) per emmagatzemar fitxers de text, documents en format PDF o fins i tot imatges dins d'alguna taula de la base de dades.

Taula 2. Tipus de dades en MySQL.

Tipus genèric	Tipus MySQL
Sencer	TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
Decimal	DECIMAL, FLOAT, DOUBLE/REAL
Text	CHAR, VARCHAR, TINYTEXT, TEXT
Objectes	BLOB, MEDIUMBLOB, LONGBLOB
Temps	DATE, TIME

Font: elaboració pròpia.

Juntament amb la declaració dels atributs, per crear una taula hem d'especificar quin atribut o combinació d'atributs serà la clau primària, identificant de forma unívoca cada instància (figura 18). En cas d'existir, les claus foranes per referenciar els atributs d'altres taules també s'han d'indicar explícitament.

El dissenyador pot activar dos controls interns sobre el valor d'un atribut en el moment de registrar noves instàncies a la base de dades. En primer lloc, és possible rebutjar aquells registres que no posseeixin un valor definit per a un atribut concret. Aquesta circumstància s'especifica amb la construcció NOT NULL, just després de la declaració de tipus. NOT NULL seria una

restricció de camp obligatori, no accepta valors nuls. En cas contrari, el sistema assignarà per defecte el valor NULL a aquest camp i acceptarà valors nuls. Aquest requeriment s'ha de satisfer inexcusablement en aquells atributs que pertanyen a la clau primària. En segon lloc, per als identificadors numèrics associats a cada instància, el propi sistema pot encarregar-se de gestionar un comptador automàtic de valors mitjançant la construcció AUTO_INCREMENT.

Tornant novament a la nostra base de dades **catalogo** (figura 16), ens trobem ara en disposició de crear les taules del catàleg de gens especificades formalment a les figures 18, 19, 20 i 21. Hem d'escollir adequadament els tipus de dades per a cada atribut o camp, segons el seu contingut, definint clarament quines són les claus primàries i foranes. L'usuari pot començar creant les taules més elementals, és a dir, aquelles que no posseeixen claus foranes (genomes i funcions). Observeu com declarem la clau primària i ens assegurem que cap instància pot donar-se d'alta a la base de dades amb un valor nul per a les claus primàries, **espècie** i **funció**. Per verificar que el procés de creació ha funcionat correctament, l'usuari pot consultar la base de dades sobre les taules existents amb l'ordre SHOW TABLES.

```
mysql> SHOW TABLES;

Empty set (0,00 sec)

mysql> CREATE TABLE genomas
    -> (especie     VARCHAR(100) NOT NULL,
    ->  nombre      VARCHAR(100),
    ->  descripcion TEXT,
    ->  PRIMARY KEY (especie));

Query OK, 0 rows affected (0,28 sec)

mysql> CREATE TABLE funciones
    -> (funcion     VARCHAR(20) NOT NULL,
    ->  descripcion VARCHAR(100),
    ->  PRIMARY KEY (funcion));

Query OK, 0 rows affected (0,03 sec)

mysql> SHOW TABLES;

+--------------------+
| Tables_in_catalogo |
+--------------------+
| funciones          |
| genomas            |
+--------------------+
2 rows in set (0,00 sec)
```

Figura 18. Crear les taules *genomes* i *funcions* en la nostra base de dades *catalogo.*
Font: elaboració pròpia.

És possible revisar la definició d'una taula amb la instrucció DESCRIBE:

```
mysql> DESCRIBE genomas;

+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| especie    | varchar(100) | NO   | PRI | NULL    |       |
| nombre     | varchar(100) | YES  |     | NULL    |       |
| descripcion | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0,01 sec)
```

Figura 19. Mostra de la descripció d'una taula del nostre catàleg.
Font: elaboració pròpia.

A continuació, per crear la taula **genes**, a més de la clau primària, indiquem un camp o atribut que és la clau forana (que ha d'apuntar o fer referència a la clau primària de la taula *genomas*):

```
mysql> CREATE TABLE genes
    -> (nombre    VARCHAR(20) NOT NULL,
    ->  cromosoma VARCHAR(5),
    ->  hebra     VARCHAR(1),
    ->  inicio    INT,
    ->  final     INT,
    ->  proteina  VARCHAR(20),
    ->  especie   VARCHAR(100),
    ->  PRIMARY KEY (nombre),
    ->  FOREIGN KEY (especie)
    ->  REFERENCES genomas(especie));

Query OK, 0 rows affected (0,06 sec)

mysql> DESCRIBE genes;

+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| nombre   | varchar(20)  | NO   | PRI | NULL    |       |
| cromosoma | varchar(5)  | YES  |     | NULL    |       |
| hebra    | varchar(1)   | YES  |     | NULL    |       |
| inicio   | int(11)      | YES  |     | NULL    |       |
| final    | int(11)      | YES  |     | NULL    |       |
| proteina | varchar(20)  | YES  |     | NULL    |       |
| especie  | varchar(100) | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
7 rows in set (0,00 sec)
```

Figura 20. Creació de la taula *genes* a la nostra base de dades.
Font: elaboració pròpia.

Finalment, creem la taula **anotacions** per relacionar els gens amb les anotacions funcionals. Juntament amb els dos valors que identifiquen cada registre (**gen** i **funció**), afegirem un atribut per registrar l'origen de la informació:

```
mysql> CREATE TABLE anotaciones
    -> (nombre      VARCHAR(20) NOT NULL,
    ->  funcion     VARCHAR(20) NOT NULL,
    ->  origen      VARCHAR(20),
    ->  PRIMARY KEY (nombre,funcion),
    ->  FOREIGN KEY (nombre)
    ->  REFERENCES  genes(nombre),
    ->  FOREIGN KEY (funcion)
    ->  REFERENCES  funciones(funcion));

Query OK, 0 rows affected (0,11 sec)

mysql> DESCRIBE anotaciones;

+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| nombre | varchar(20) | NO   | PRI | NULL    |       |
| funcion | varchar(20) | NO  | PRI | NULL    |       |
| origen | varchar(20) | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
3 rows in set (0,01 sec)
```

Figura 21. Creació de la taula *anotaciones* a la nostra base de dades *catalogo.*
Els dos components de la clau primària de la taula *anotaciones* s'han de declarar com a claus foranes amb origen en les taules *genes* i *funciones*.
Font: elaboració pròpia.

És possible definir altres restriccions als camps de les taules amb l'ordre CHECK.

Per exemple, podem indicar que un camp numèric com el camp **inci** de la taula **genes** que és tipus numèric només accepti valors positius `CHECK (inici >0)`, o que un camp de tipus cadena de caràcters només accepti determinats valors; per exemple, perquè el camp *hebra* de la taula *genes* només accepti els caràcters «+» o «-», podem fer `hebra ENUM('+','-')`.

Un cop creada la taula podem modificar-la amb la instrucció **ALTER TABLE**.

Per exemple, si volem eliminar el camp origen de la taula **anotaciones** escrivim

```
ALTER TABLE anotacions DROP COLUMN origen;
```

i si volem tornar a afegir-hi el mateix camp escrivim

```
ALTER TABLE anotacions ADD origen VARCHAR(20);
```

Durant el temps de vida d'una base de dades és freqüent que n'haguem d'actualitzar el contingut. En determinats casos, això pot implicar l'eliminació completa d'usuaris, de taules o, fins i tot, de la pròpia base de dades. Per implementar aquests serveis, MySQL posseeix la família d'ordres DROP.

```
DROP DATABASE basededatos;

DROP USER usuario;

DROP TABLE tabla;
```

Figura 22. Sintaxi de l'ordre DROP.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.6. Inserir i manipular dades a les taules

Un cop l'esquema relacional d'entitats prèviament dissenyat està estructurat sobre MySQL mitjançant taules, és el moment de dotar de contingut cada taula.

MySQL disposa de dues formes d'inserir nous registres a les taules de la base de dades:

1. Càrrega simultània de múltiples registres des d'un fitxer de text.

2. Inserció individual de cada nou registre de forma manual.

Per importar una quantitat elevada de registres (*) és possible utilitzar l'ordre LOAD DATA. Per invocar aquesta ordre hem d'especificar el nom del fitxer de text que alberga la informació dels registres juntament amb el nom de la taula on han de ser donats d'alta. Cal disposar d'accés a un fitxer de text tabulat, on cada fila representa un nou registre i cada columna alberga el valor d'un atribut (especificat en el mateix ordre que a la taula).

```
LOAD DATA LOCAL INFILE fichero.txt INTO TABLE tabla;
```

Figura 23. Sintaxi de l'ordre LOAD DATA.
Font: elaboració pròpia.

> En determinats entorns d'UNIX cal activar específicament l'opció –local-infile a l'hora d'invocar el programa mysql. Aquesta opció, no obstant, està activada per defecte habitualment en la majoria de distribucions de Linux.

Podem procedir a introduir les primeres dades en el nostre catàleg de gens. És recomanable començar per les taules més elementals, aquelles que no posseeixen claus foranes. En el nostre cas, les taules **genomas** i **funciones** s'ajusten perfectament a aquesta descripció. Per exemple, per poblar la taula **genomas** editarem el següent fitxer, **genomas.txt**, des del nostre terminal d'UNIX:

```
D.melanogaster  Mosca de la fruta Tambien denominada del vinagre
H.sapiens       Hombre            Nuestra propia especie
M. musculus     Raton             Otro organismo modelo
```

Figura 24. El fitxer genomas.txt per poblar la taula *genomas*.
Font: elaboració pròpia.

Per procedir a la càrrega d'aquestes dades a la taula *genomas*, l'usuari ha d'introduir la següent ordre des de l'intèrpret de MySQL:

```
mysql> LOAD DATA LOCAL INFILE 'genomas.txt' INTO TABLE genomas;

Query OK, 3 rows affected (0,08 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 25. Carregar el fitxer genomas.txt a la taula *genomas*.
Font: elaboració pròpia.

Presentem el contingut del fitxer **funciones.txt,** que emprarem per poblar la taula *funciones* amb tres nous registres:

```
GO:0003700      Factor de transcripcion
GO:0006338      Remodelado de cromatina
GO:0007254      Via JNK
```

Figura 26. El fitxer funciones.txt per poblar la taula *funcions*.
Font: elaboració pròpia.

Ara emprem el fitxer **funciones.txt** per poblar la corresponent taula:

```
mysql> LOAD DATA LOCAL INFILE 'funciones.txt' INTO TABLE funciones;

Query OK, 3 rows affected (0,01 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 27. Carregar el fitxer funciones.txt a la taula *funciones*.
Font: elaboració pròpia.

Un cop hem poblat les taules **genomas** i **funciones** amb diverses instàncies d'espècies i funcions biològiques, respectivament, és el moment d'editar el fitxer **genes.txt** per donar d'alta nous gens a la taula *genes*:

```
MYC   chr8   + 128748314 128753678 NP_002458 H.sapiens
HNF1A chr12  + 121416548 121440312 NP_000536 H.sapiens
cbt   chr2L  -    476437    479046 NP_722636 D.melanogaster
ash2  chr3R  +  20477248  20479098 NP_733023 D.melanogaster
```

Figura 28. El fitxer genes.txt per poblar la taula *genes*.
Font: elaboració pròpia

I ara procedim a realitzar la càrrega amb l'ordre LOAD DATA:

```
mysql> LOAD DATA LOCAL INFILE 'genes.txt' INTO TABLE genes;

Query OK, 4 rows affected (0,02 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 29. Carregar el fitxer genes.txt a la taula *genes*.
Font: elaboració pròpia.

Abans de procedir a realitzar les primeres consultes, editarem el fitxer de text anotaciones.txt per assignar funcions als gens que hem registrat en les ordres prèvies.

```
MYC     GO:0003700  Experimental
MYC     GO:0006338  Literatura
HNF1A   GO:0003700  Experimental
cbt     GO:0003700  Experimental
cbt     GO:0007254  Experimental
ash2    GO:0006338  Experimental
ash2    GO:0003700  Computacional
```

Figura 30. El fitxer anotaciones.txt per poblar la taula *anotaciones*.
Font: elaboració pròpia.

Observeu que els gens poden posseir més d'una anotació funcional. D'altra banda, la mateixa funció biològica pot ser exercida per gens diferents. Però els dos valors junts formen una clau única.

Estem en condicions de poblar la nostra última taula, *anotaciones*:

```
mysql> LOAD DATA LOCAL INFILE 'anotaciones.txt'
    -> INTO TABLE anotaciones;

Query OK, 7 rows affected (0,08 sec)
Records: 7  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 31. Carregar el fitxer anotaciones.txt a la taula *anotaciones*.
Font: elaboració pròpia.

La càrrega de dades des d'un fitxer de text a les taules és extremadament útil. No obstant això, en determinats casos necessitem donar d'alta un nou registre de forma aïllada, però l'edició d'un fitxer de text únicament amb aquest objectiu és menys eficient.

En aquests casos, l'ordre INSERT és més adequada, atès que implementa aquesta funcionalitat en el gestor MySQL de bases de dades. L'usuari ha d'especificar en el mateix ordre tant la llista d'atributs del nou registre com els seus corresponents valors. La resta d'atributs no inclosos en la relació anterior prendran el valor NULL (excepte per a aquells on està expressament prohibida aquesta circumstància durant la creació de la taula, camps obligatoris):

```
INSERT INTO tabla (campo1, campo2, ..., campoN)
VALUES (valor1, valor2, ..., valorN);
```

Figura 32. Sintaxi de l'ordre INSERT.
Font: elaboració pròpia.

Per regla general, però, un registre conté tots els camps declarats per a una taula. Per tant, respectant l'ordre dels camps a la taula, podem ometre la relació completa dels atributs:

```
INSERT INTO tabla
VALUES (valor1, valor2, ..., valorN);
```

Figura 33. Sintaxi abreujada de l'ordre INSERT.
Font: elaboració pròpia.

A tall d'exemple, mostrem a continuació la seqüència d'ordres d'inserció equivalent a la càrrega simultània de les funcions executada en la figura 27.

> Las cadenes de text s'han d'introduir utilitzant sempre cometes simples, mentre que els valors numèrics no necessiten cap format addicional.

```
mysql> INSERT INTO funciones
    -> VALUES ('GO:0003700',
    ->         'Factor de transcripcion');

mysql> INSERT INTO funciones
    -> VALUES ('GO:0006338',
               'Remodelado de cromatina');

mysql> INSERT INTO funciones
    -> VALUES ('GO:0007254',
               'Via JNK');
```

Figura 34. Inserció de registres amb l'ordre INSERT en el nostre catàleg.
Font: elaboració pròpia..

> En el cas d'intentar donar d'alta un registre la clau primària del qual ja existeix, el sistema ens advertirà de l'error, avortant aquesta operació. Una taula no pot tenir una clau primària repetida

Un cop tenim les dades introduïdes a les taules podem modificar-les amb la instrucció UPDATE o eliminar registres amb la instrucció DELETE.

Per exemple, si volem modificar el valor «**Factor de transcipcion**», situat en el camp del registre o fila amb clau primària GO:0003700 de la taula, i volem que el nou valor sigui «**Transcription factor**», escriurem la instrucció següent:

```
UPDATE funcions
SET descripcion ='Transcription factor'
WHERE funcion = 'GO:0003700';
```

Per eliminar únicament alguns registres d'una determinada taula podem utilitzar l'ordre DELETE juntament amb la clàusula WHERE. D'aquesta manera, seleccionarem amb precisió els registres que han de ser donats de baixa.

Si l'usuari desitja eliminar tots els registres d'una taula, conservant l'estructura d'aquesta (per reutilitzar-la en el futur), n'hi ha prou amb ometre la condició:

```
DELETE FROM tabla WHERE condiciones;

DELETE FROM tabla;
```

Figura 35. Sintaxi de l'ordre DELETE.
Font: elaboració pròpia.

Si volem eliminar el registre de la taula **funciones** amb clau primària **GO:0007254** escriurem la instrucció següent:

```
DELETE FROM funcions WHERE funcion = 'GO:0007254';
```

Si intentem eliminar un registre que està referenciat per una clau forana d'una altra taula el sistema ho impedirà i saltarà un error, tret que a la clau forana li indiquem l'opció **ON DELETE CASCADE**, que permet eliminar en cascada el registre que desitgem eliminar i els registres de l'altra taula que estan referenciats.

1. Bases de dades relacionals

1.7. Consultes bàsiques a les taules

Les bases de dades són eines excepcionalment útils per consultar informació i extreure nou coneixement. En aquest sentit, els esquemes entitat-relació no són una excepció. Més aviat al contrari, utilitzant l'àlgebra relacional és possible consultar el contingut de les taules de múltiples formes. Bàsicament, les consultes SQL (en anglès, *queries*) consisteixen en filtres que delimiten el segment del conjunt complet dels registres d'una o més taules en el qual estem interessats, per mostrar després el valor dels seus atributs.

La instrucció **SELECT** implementa el procés de consulta sobre les taules, mostrant els atributs indicats per a aquells registres que compleixen una determinada condició.

> Podeu trobar informació sobre l'ús del terminal per dur a terme operacions similars sobre fitxers de text al mòdul «L'entorn de treball UNIX».

Anem a explorar en els pròxims anys com enriquir les nostres consultes, utilitzant per a això el nostre catàleg de gens, que està emmagatzemat en la base de dades **catalogo**. Abans de fer consultes més elaborades, mostrem a continuació la forma més senzilla de realitzar una consulta.

```
SELECT campo1, campo2, . . . , campon FROM tabla;
```

Figura 36. Sintaxi bàsica de l'ordre SELECT.
Font: elaboració pròpia..

La consulta més habitual consisteix a mostrar el contingut complet d'una taula. El caràcter *, precisament, indica que desitgem visualitzar el llistat íntegre dels valors de tots els atributs per al subconjunt de registres seleccionats. Per posar en pràctica aquesta ordre sobre el nostre catàleg, seleccionem tots els valors de cada instància guardada a la taula *genes*:

```
mysql> SELECT * FROM genes;

+--------+-----------+-------+-----------+-----------+-----------+----------------+
| nombre | cromosoma | hebra | inicio    | final     | proteina  | especie        |
+--------+-----------+-------+-----------+-----------+-----------+----------------+
| ash2   | chr3R     | +     |  20477248 |  20479098 | NP_733023 | D.melanogaster |
| cbt    | chr2L     | -     |    476437 |    479046 | NP_722636 | D.melanogaster |
| HNF1A  | chr12     | +     | 121416548 | 121440312 | NP_000536 | H.sapiens      |
| MYC    | chr8      | +     | 128748314 | 128753678 | NP_002458 | H.sapiens      |
+--------+-----------+-------+-----------+-----------+-----------+----------------+
4 rows in set (0,00 sec)
```

Figura 37. Mostrant el contingut íntegre de la taula *genes*.
Font: elaboració pròpia.

Per evitar l'excés d'informació, l'usuari pot seleccionar els atributs o camps dels registres d'una taula que desitja veure per pantalla. Simplement substituint en la pregunta el símbol * per un llistat d'atributs, separats per comes, podem delimitar la vista dels registres, en aquest cas gens que obtenim com a resultat:

```
mysql> SELECT nombre,especie FROM genes;

+--------+---------------+
| nombre | especie       |
+--------+---------------+
| ash2   | D.melanogaster |
| cbt    | D.melanogaster |
| HNF1A  | H.sapiens      |
| MYC    | H.sapiens      |
+--------+---------------+
4 rows in set (0,00 sec)
```

Figura 38. Mostrant els valors d'alguns atributs de la taula *genes*.
Font: elaboració pròpia.

El recompte del nombre de registres que compleix una condició concreta és una de les consultes més freqüents en SQL. En aquest cas és suficient amb afegir la funció COUNT a la consulta que estem realitzant per comptabilitzar el nombre de línies de la sortida:

```
mysql> SELECT COUNT(*) FROM genes;

+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0,00 sec)
```

Figura 39. Comptant tots els registres de la taula *genes*.
Font: elaboració pròpia.

La funció DISTINCT elimina els resultats duplicats. Per exemple, si desitgem comptar el nombre d'organismes a la nostra taula *genes*, podem combinar les funcions COUNT i DISTINCT sobre l'atribut *especie* de la manera següent:

```
mysql> SELECT especie FROM genes;

+----------------+
| especie        |
+----------------+
| D.melanogaster |
| D.melanogaster |
| H.sapiens      |
| H.sapiens      |
+----------------+
4 rows in set (0,00 sec)

mysql> SELECT DISTINCT especie FROM genes;

+----------------+
| especie        |
+----------------+
| D.melanogaster |
| H.sapiens      |
+----------------+
2 rows in set (0,07 sec)

mysql> SELECT COUNT(DISTINCT especie) FROM genes;

+------------------------+
| COUNT(DISTINCT especie) |
+------------------------+
|                      2 |
+------------------------+
1 row in set (0,07 sec)
```

Figura 40. Comptant registres únics d'una taula.
Font: elaboració pròpia.

L'ordre ORDER BY ordena la llista de resultats produïda per una ordre SELECT, de forma ascendent o descendent (segons si hi afegim la clàusula ASC o DESC, respectivament). En la propera figura ordenem els gens per la seva posició en cada cromosoma o pel seu nom, de diferents maneres:

```
mysql> SELECT nombre,cromosoma,inicio FROM genes ORDER BY inicio;

+--------+----------+-----------+
| nombre | cromosoma | inicio    |
+--------+----------+-----------+
| cbt    | chr2L     |    476437 |
| ash2   | chr3R     |  20477248 |
| HNF1A  | chr12     | 121416548 |
| MYC    | chr8      | 128748314 |
+--------+----------+-----------+
4 rows in set (0,00 sec)

mysql> SELECT nombre,cromosoma,inicio FROM genes ORDER BY nombre;

+--------+----------+-----------+
| nombre | cromosoma | inicio    |
+--------+----------+-----------+
| ash2   | chr3R     |  20477248 |
| cbt    | chr2L     |    476437 |
| HNF1A  | chr12     | 121416548 |
| MYC    | chr8      | 128748314 |
+--------+----------+-----------+
4 rows in set (0,00 sec)

mysql> SELECT nombre,cromosoma,inicio FROM genes
    -> ORDER BY nombre DESC;

+--------+----------+-----------+
| nombre | cromosoma | inicio    |
+--------+----------+-----------+
| MYC    | chr8      | 128748314 |
| HNF1A  | chr12     | 121416548 |
| cbt    | chr2L     |    476437 |
| ash2   | chr3R     |  20477248 |
+--------+----------+-----------+
4 rows in set (0,01 sec)
```

Figura 41. Ordenar els registres d'una taula.
Font: elaboració pròpia.

Quan es treballa amb taules que contenen milers d'elements, resulta convenient mostrar inicialment només els primers registres per comprovar el correcte funcionament de la consulta. La funció `LIMIT` permet mostrar exclusivament els primers *n* registres de la consulta en execució:

```
mysql> SELECT * FROM genes LIMIT 1;

| nombre | cromosoma | hebra | inicio   | final    | proteina  | especie        |
+--------+----------+-------+----------+----------+-----------+----------------+
| ash2   | chr3R     | +     | 20477248 | 20479098 | NP_733023 | D.melanogaster |
+--------+----------+-------+----------+----------+-----------+----------------+
1 row in set (0,00 sec)
```

Figura 42. Mostrar un fragment de la consulta.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.8. Consultes bàsiques: filtres i condicions

L'ordre SELECT permet també extreure de les taules únicament aquells registres que posseeixen certes propietats. Per especificar el filtre a realitzar sobre el contingut d'una taula, cal afegir-hi la clàusula WHERE:

```
SELECT campo1,campo2,...,campon FROM tabla WHERE condicion;
```

Figura 43. Sintaxi bàsica de l'ordre SELECT amb condicions.
Font: elaboració pròpia.

La condició pot ser simple o composta, avaluant-se sobre un o més atributs de diverses taules. Els operadors de comparació més habituals es mostren a la taula 3.

Taula 3. Operadors de comparació en consultes de MySQL.

Operador	Significat
=,<>	Igual/diferent
<,>	Menor/major
<=,>=	Menor/major o igual
LIKE	Recerca d'un patró de text
NOT	Negació d'una condició
AND/OR	Condicions combinades
REGEXP	Expressió regular

Font: elaboració pròpia.

Els operadors numèrics també resulten molt útils per buscar registres en un rang concret de dates del calendari.

Provarem aquests operadors per realitzar consultes més concretes sobre la nostra base de dades **catalogo**. En primer lloc, podem interrogar la base de dades sobre els gens ubicats en el fil positiu de la cadena d'ADN en qualsevol espècie, preguntar per aquells que no pertanyen a la nostra espècie o buscar els gens anotats abans del primer milió de bases en qualsevol cromosoma:

```
mysql> SELECT * FROM genes WHERE hebra LIKE '+';

+--------+----------+-------+----------+----------+----------+----------------+
| nombre | cromosoma | hebra | inicio   | final    | proteina | especie        |
+--------+----------+-------+----------+----------+----------+----------------+
| ash2   | chr3R    | +     |  20477248 |  20479098 | NP_733023 | D.melanogaster |
| HNF1A  | chr12    | +     | 121416548 | 121440312 | NP_000536 | H.sapiens      |
| MYC    | chr8     | +     | 128748314 | 128753678 | NP_002458 | H.sapiens      |
+--------+----------+-------+----------+----------+----------+----------------+
3 rows in set (0,00 sec)

mysql> SELECT * FROM genes WHERE especie NOT LIKE 'H.sapiens';

+--------+----------+-------+---------+---------+----------+----------------+
| nombre | cromosoma | hebra | inicio  | final   | proteina | especie        |
+--------+----------+-------+---------+---------+----------+----------------+
| ash2   | chr3R    | +     | 20477248 | 20479098 | NP_733023 | D.melanogaster |
| cbt    | chr2L    | -     |   476437 |   479046 | NP_722636 | D.melanogaster |
+--------+----------+-------+---------+---------+----------+----------------+
2 rows in set (0,00 sec)

mysql> SELECT * FROM genes WHERE inicio <= 1000000;

+--------+----------+-------+--------+--------+----------+----------------+
| nombre | cromosoma | hebra | inicio | final  | proteina | especie        |
+--------+----------+-------+--------+--------+----------+----------------+
| cbt    | chr2L    | -     | 476437 | 479046 | NP_722636 | D.melanogaster |
+--------+----------+-------+--------+--------+----------+----------------+
1 row in set (0,00 sec)
```

Figura 44. Consultes amb una condició.
Font: elaboració pròpia.

La clàusula es pot complementar amb el modificador %, que actua de comodí en les expressions alfanumèriques. A continuació, seleccionem només registres que pertanyen al genoma de la mosca:

```
mysql> SELECT * FROM genes WHERE especie LIKE '%melano%';

+--------+----------+-------+---------+---------+----------+----------------+
| nombre | cromosoma | hebra | inicio  | final   | proteina | especie        |
+--------+----------+-------+---------+---------+----------+----------------+
| ash2   | chr3R    | +     | 20477248 | 20479098 | NP_733023 | D.melanogaster |
| cbt    | chr2L    | -     |   476437 |   479046 | NP_722636 | D.melanogaster |
+--------+----------+-------+---------+---------+----------+----------------+
2 rows in set (0,00 sec)
```

Figura 45. Consultes sobre patrons de text.
Font: elaboració pròpia.

També podem combinar preguntes sobre valors de diferents tipus. Per exemple, si desitgem esbrinar quants gens de la mosca de la fruita estan anotats al fil positiu de la cadena d'ADN:

```
mysql> SELECT * FROM genes WHERE especie LIKE '%melano%'
    -> AND hebra LIKE '+';

+--------+----------+-------+---------+---------+----------+----------------+
| nombre | cromosoma | hebra | inicio  | final   | proteina | especie        |
+--------+----------+-------+---------+---------+----------+----------------+
| ash2   | chr3R    | +     | 20477248 | 20479098 | NP_733023 | D.melanogaster |
+--------+----------+-------+---------+---------+----------+----------------+
1 row in set (0,00 sec)
```

Figura 46. Consultes amb dues condicions.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.9. Consultes avançades: agrupacions

Mitjançant el desglossament de dades d'una taula, en funció d'algun camp concret, podem calcular estadístiques sobre cada categoria. L'ordre GROUP BY permet realitzar agrupacions de les dades de les taules segons els criteris que establim, i és possible combinar aquestes classificacions amb operadors d'agregació com COUNT, MAX, MIN, AVG o SUM.

```
SELECT campo1,campo2,...,campon FROM tabla GROUP BY atributo;
```

Figura 47. Sintaxi de l'ordre SELECT sobre grups.
Font: elaboració pròpia.

Per exemple, demanem les espècies presents a la nostra base de dades:

```
mysql> SELECT especie FROM genes GROUP BY especie;

+----------------+
| especie        |
+----------------+
| D.melanogaster |
| H.sapiens      |
+----------------+
2 rows in set (0,00 sec)
```

Figura 48. Dades agrupades per espècie.
Font: elaboració pròpia.

Posteriorment, podem comptar el nombre exacte d'exemples de cada espècie:

```
mysql> SELECT especie,COUNT(*) FROM genes GROUP BY especie;

+----------------+----------+
| especie        | COUNT(*) |
+----------------+----------+
| D.melanogaster |        2 |
| H.sapiens      |        2 |
+----------------+----------+
2 rows in set (0,00 sec)
```

Figura 49. Nombre d'espècies emmagatzemades a la taula *genes*.
Font: elaboració pròpia.

Ara obtenim les estadístiques bàsiques sobre la longitud dels gens:

```
mysql> SELECT especie,cromosoma,inicio,final,final-inicio
    -> FROM genes;

+---------------+-----------+-----------+-----------+--------------+
| especie       | cromosoma | inicio    | final     | final-inicio |
+---------------+-----------+-----------+-----------+--------------+
| D.melanogaster | chr3R     |  20477248 |  20479098 |         1850 |
| D.melanogaster | chr2L     |    476437 |    479046 |         2609 |
| H.sapiens     | chr12     | 121416548 | 121440312 |        23764 |
| H.sapiens     | chr8      | 128748314 | 128753678 |         5364 |
+---------------+-----------+-----------+-----------+--------------+
4 rows in set (0,00 sec)

mysql> SELECT especie,AVG(final-inicio),MIN(final-inicio),
    -> MAX(final-inicio) FROM genes GROUP BY especie;

+---------------+-------------------+-------------------+-------------------+
| especie       | AVG(final-inicio) | MIN(final-inicio) | MAX(final-inicio) |
+---------------+-------------------+-------------------+-------------------+
| D.melanogaster |         2229.5000 |              1850 |              2609 |
| H.sapiens     |        14564.0000 |              5364 |             23764 |
+---------------+-------------------+-------------------+-------------------+
2 rows in set (0,00 sec)
```

Figura 50. Càlcul de mitjanes a la base de dades *catalogo*.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.10. Consultes avançades: consultes multitaula

Per aprofitar al màxim el model relacional i generar nou coneixement de les dades existents resulta essencial combinar informació de diverses taules. Mitjançant un atribut que dues taules posseeixin en comú, aquesta operació resulta extremadament senzilla amb SQL. Mentre efectuem una consulta amb l'ordre SELECT, hem d'emprar la clàusula JOIN especificant l'atribut compartit per ambdues taules. Quan es referencien atributs de dues o més taules en la mateixa consulta, cal utilitzar la sintaxi `nombre_tabla.nom_ atribut` per especificar clarament l'origen de cada atribut. És possible afegir-hi condicions sobre altres atributs de les taules amb la clàusula WHERE.

Donades dues taules que denominarem **taula1** i **taula2**, que posseeixen un atribut comparable (**taula1.x** i **taula2.y**), la sintaxi de l'ordre **JOIN** per obtenir tant els valors en comú com els valors presents exclusivament en la primera o en la segona taula, respectivament, és la següent:

```
SELECT tabla1.x,tabla2.y

FROM tabla1 JOIN tabla2

ON tabla1.x=tabla2.y

WHERE condiciones;

-----------------------------------------

SELECT tabla1.x,tabla2.y

FROM tabla1 LEFT JOIN tabla2

ON tabla1.x=tabla2.y

WHERE condiciones;

-----------------------------------------

SELECT tabla1.x,tabla2.y

FROM tabla1 RIGHT JOIN tabla2

ON tabla1.x=tabla2.y

WHERE condiciones;
```

Figura 51. Sintaxi de la clàusula JOIN.
Font: elaboració pròpia.

Per treballar amb més de dues taules, podem generalitzar la mateixa sintaxi (per exemple, taula1 JOIN (taula2,33,44)).

Abans de procedir a executar consultes sobre les taules de la base de dades, proposem posar a prova el funcionament de la clàusula JOIN sobre un exemple més simple.

Generarem dues taules que denominarem **taula1** i **taula2**, amb un únic atribut. Posteriorment, poblarem ambdues taules amb una sèrie de valors tals que resultarà molt senzill mostrar el conjunt complet de combinacions a l'hora de comparar les dues taules.

Primer procedim a crear les dues taules:

```
mysql> CREATE TABLE tabla1
    -> (x VARCHAR(10));

Query OK, 0 rows affected (0,5 sec)

mysql> CREATE TABLE tabla2
    -> (y VARCHAR(10));

Query OK, 0 rows affected (0,5 sec)
```

Figura 52. Crear dues taules per combinar amb la clàusula JOIN.
Font: elaboració pròpia.

> Us animem a reproduir al vostre ordinador les ordres de SQL presentades en aquest apartat.

Ara crearem dos fitxers de text senzills amb tres valors cadascun: (1,2,3) i (3,4,5). D'aquesta manera, podrem estudiar detalladament la classe de consulta de MySQL que necessitem per recuperar els valors comuns entre ambdues taules (3) o, alternativament, els valors que únicament apareixen a la primera (1 i 2) o a la segona taula (4 i 5).

```
1
2
3

-----------------------------------------------

3
4
5
```

Figura 53. Les dades .txt i dades2.txt.
Font: elaboració pròpia.

Finalment, poblem les dues taules utilitzant ambdós fitxers:

```
mysql> LOAD DATA LOCAL INFILE 'datos1.txt' INTO TABLE tabla1;

Query OK, 3 rows affected (0,67 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA LOCAL INFILE 'datos2.txt' INTO TABLE tabla2;

Query OK, 3 rows affected (0,43 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 54. Poblem les dues taules per combinar amb la clàusula JOIN.
Font: elaboració pròpia.

Ja estem en disposició d'aprofundir en el funcionament de les consultes que inclouen la clàusula JOIN. Si no hi afegim cap opció, aquesta ordre genera totes les parelles possibles el primer element de les quals pertany a la primera taula i el segon element pertany a la segona:

```
mysql> SELECT tabla1.x,tabla2.y
    -> FROM tabla1 JOIN tabla2;

+------+------+
| x    | y    |
+------+------+
| 1    | 3    |
| 2    | 3    |
| 3    | 3    |
| 1    | 4    |
| 2    | 4    |
| 3    | 4    |
| 1    | 5    |
| 2    | 5    |
| 3    | 5    |
+------+------+
9 rows in set (0,00 sec)
```

Figura 54. Combinació amb JOIN de dues taules per obtenir totes les combinacions.
Font: elaboració pròpia.

El nostre primer objectiu en una comparació és localitzar els elements comuns. Per tal de fer-ho n'hi ha prou amb incorporar la clàusula **ON** a la consulta i especificar l'atribut compartit per les dues taules. Això filtrarà aquelles parelles del resultat anterior que no compleixin aquesta propietat, i es demostrarà allò que busquem:

```
mysql> SELECT tabla1.x,tabla2.y
    -> FROM tabla1 JOIN tabla2
    -> ON tabla1.x=tabla2.y;

+------+------+
| x    | y    |
+------+------+
| 3    | 3    |
+------+------+
1 row in set (0,00 sec)
```

Figura 55. Combinació amb JOIN de dues taules amb la clàusula ON.
Font: elaboració pròpia.

En determinats casos, en lloc de la llista dels valors comuns, estarem interessats també en aquells valors d'una o altra taula que no pertanyen a l'altra. Per això hem de modificar el comportament de l'ordre JOIN amb les clàusules LEFT o RIGHT. Si realitzem una unió per la part esquerra, recuperarem un llistat dels registres de la primera taula juntament amb el seu registre equivalent a la segona. En el cas que aquest valor anàleg no existís, MySQL ho indicarà amb el valor NULL. Si, per contra, realitzem la unió per la dreta, obtindrem un llistat dels valors de la segona taula sota les mateixes condicions.

```
mysql> SELECT tabla1.x,tabla2.y
    -> FROM tabla1 LEFT JOIN tabla2
    -> ON tabla1.x=tabla2.y;

+------+------+
| x    | y    |
+------+------+
| 3    | 3    |
| 1    | NULL |
| 2    | NULL |
+------+------+
3 rows in set (0,00 sec)

mysql> SELECT tabla1.x,tabla2.y
    -> FROM tabla1 RIGHT JOIN tabla2
    -> ON tabla1.x=tabla2.y;

+------+------+
| x    | y    |
+------+------+
| 3    | 3    |
| NULL | 4    |
| NULL | 5    |
+------+------+
3 rows in set (0,00 sec)
```

Figura 56. Combinació amb JOIN de dues taules amb les clàusules LEFT i RIGHT.
Font: elaboració pròpia.

Per acabar de refinar el resultat, hem de mantenir exclusivament els valors que només estan en una taula. Per aconseguir-ho, podem afegir al final de la consulta una condició WHERE que exigeixi que el valor no estigui present a l'altra taula. La clàusula IS realitza l'avaluació de l'expressió que es troba a continuació, per acabar responent amb un valor booleà (cert o fals).

```
mysql> SELECT tabla1.x,tabla2.y
    -> FROM tabla1 LEFT JOIN tabla2
    -> ON tabla1.x=tabla2.y
    -> WHERE tabla2.y IS NULL;

+------+------+
| x    | y    |
+------+------+
| 1    | NULL |
| 2    | NULL |
+------+------+
2 rows in set (0,00 sec)

mysql> SELECT tabla1.x,tabla2.y
    -> FROM tabla1 RIGHT JOIN tabla2
    -> ON tabla1.x=tabla2.y
    -> WHERE tabla1.x IS NULL;

+------+------+
| x    | y    |
+------+------+
| NULL | 4    |
| NULL | 5    |
+------+------+
2 rows in set (0,00 sec)
```

Figura 57. Combinació amb JOIN de dues taules emprant una condició.
Font: elaboració pròpia.

Ara ja estem en condicions de realitzar consultes sobre dues o més taules del nostre catàleg de gens. Per exemple, podem preguntar per aquells gens humans per als quals s'ha documentat una anotació funcional de caràcter experimental:

```
mysql> SELECT genes.nombre,genes.especie,
    -> anotaciones.funcion,anotaciones.origen
    -> FROM genes JOIN anotaciones
    -> ON genes.nombre=anotaciones.nombre
    -> WHERE anotaciones.origen='experimental'
    -> AND genes.especie='H.sapiens';

+--------+----------+-----------+-------------+
| nombre | especie  | funcion   | origen      |
+--------+----------+-----------+-------------+
| HNF1A  | H.sapiens | GO:0003700 | Experimental |
| MYC    | H.sapiens | GO:0003700 | Experimental |
+--------+----------+-----------+-------------+
2 rows in set (0,00 sec)
```

Figura 58. Consultes sobre el catàleg de gens utilitzant l'ordre JOIN.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.11. Consultes avançades: subconsultes

En ocasions desitgem realitzar un tipus de pregunta sobre les nostres dades, però no és factible perquè l'organització en taules escollida no ho permet. Per resoldre aquest problema, MySQL permet enniuar una consulta dins d'una altra, amb l'objectiu d'utilitzar la pregunta interior per donar-li la forma apropiada a les dades, que podran ser tractades posteriorment mitjançant la consulta exterior.

Sintaxi bàsica d'una subconsulta:

```
SELECT lista_columnas
FROM nombre_tabla
WHERE condición = (SELECT lista_columnas2
FROM nombre_tabla2
WHERE condiciones);
```

Sintàcticament, des del punt de vista de la consulta principal, la subconsulta interior exercirà el paper d'una taula convencional. Per aquesta raó, és possible assignar un nom tant a la consulta interior com als atributs dels resultats que se'n desprendran. Per a això emprarem la clàusula **AS**, que permet associar un nom a un grup d'operacions o atributs en SQL. El nom emprat per a aquests atributs resulta útil per referir-s'hi des de la consulta exterior.

```
SELECT subconsulta.valor_1, ..., subconsulta.valor_n FROM
   (SELECT atributo_1 AS valor_1, ..., atributo_n AS valor_n
    FROM tabla GROUP BY atributo_i) AS subconsulta;
```

Figura 59. Sintaxi de les subconsultes.
Font: elaboració pròpia.

Per exemplificar la classe d'escenari on les subconsultes resulten potencialment interessants, imaginem una taula genèrica anomenada *tabla* amb dos atributs, que denominarem *clase* i *subclsse*. Cada registre d'aquesta taula pertany a una classe general, i dins d'aquesta classe, a una subclasse més específica. Suposem que ens agradaria calcular la mitjana de subclasses diferents, classe per classe, que han estat utilitzades per etiquetar cada registre. Per obtenir la resposta, definirem una subconsulta que rebrà el nom de *contador*. Aquesta subpregunta agruparà les dades per classes per comptar el nombre total de subclasses assignat als registres de cada classe principal. Finalment, la consulta exterior simplement haurà de calcular la mitjana dels totals calculats per la subconsulta.

```
clase1 subclasex
clase1 subclasey
clase1 subclasez
clase2 subclasea
clase2 subclaseb
clase3 subclasen
...
-----------------------------------------------

SELECT AVG(contador.totales) FROM

(SELECT count(subclase) AS totales

 FROM tabla GROUP BY clase) AS contador;
```

Figura 60. Emprar una subconsulta dins d'una consulta principal.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.12. Altres utilitats de MySQL

Juntament amb l'inventari d'ordres que interactuen amb la nostra base de dades emprant el llenguatge SQL, el SGBD MySQL proporciona un conjunt d'aplicacions elementals per assistir-nos a l'hora de treballar amb el sistema.

Per exemple, l'ordre `help` mostra per pantalla un breu manual d'ajuda sobre cada instrucció de MySQL.

Per poder paginar, pantalla a pantalla, sobre les entrades del manual hem d'executar abans l'aplicació **pager**. Aquesta ordre ens permet vincular una aplicació de paginació del terminal de Linux amb l'intèrpret de MySQL (per exemple, el programa *more*).

```
mysql> pager more;

PAGER set to 'more'

mysql>  help DROP TABLE;

Name: 'DROP TABLE'
Description:
Syntax:
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]

DROP TABLE removes one or more tables. You must have the DROP privilege
for each table. All table data and the table definition are removed, so
be careful with this statement! If any of the tables named in the
argument list do not exist, MySQL returns an error indicating by name
which nonexisting tables it was unable to drop, but it also drops all
of the tables in the list that do exist.

--More--
```

Figura 61. Mostra del manual d'ajut de MySQL.
Font: elaboració pròpia.

L'ordre `source` permet executar fitxers de text que inclouen ordres MySQL amb la seqüència d'instruccions precises per realitzar una determinada tasca. Per exemple, podem editar un fitxer de text des del nostre terminal amb les primeres ordres que executem en entrar en el sistema:

```
USE catalogo;
SHOW TABLES;
DESCRIBE genes;
----------------------------------------
mysql> source comandos.sql;

Database changed

+--------------------+
| Tables_in_catalogo |
+--------------------+
| anotaciones        |
| funciones          |
| genes              |
| genomas            |
+--------------------+
4 rows in set (0,00 sec)

+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| nombre    | varchar(20)  | NO   | PRI | NULL    |       |
| cromosoma | varchar(5)   | YES  |     | NULL    |       |
| hebra     | varchar(1)   | YES  |     | NULL    |       |
| inicio    | int(11)      | YES  |     | NULL    |       |
| final     | int(11)      | YES  |     | NULL    |       |
| proteina  | varchar(20)  | YES  |     | NULL    |       |
| especie   | varchar(100) | YES  | MUL | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
7 rows in set (0,00 sec)
```

Figura 62. Execució del fitxer d'ordres comandos.sql de MySQL.
Font: elaboració pròpia.

L'ordre `status` permet veure la configuració del sistema:

```
mysql> status;

mysql  Ver 14.14 Distrib 5.7.17, for Linux (i686) using  EditLine wrapper
Connection id:          7
Current database:       catalogo
Current user:           eblanco@localhost
SSL:                    Not in use
Current pager:          more
Using outfile:          ''
Using delimiter:        ;
Server version:         5.7.17-0ubuntu0.16.04.1 (Ubuntu)
Protocol version:       10
Connection:             Localhost via UNIX socket
Server characterset:    latin1
Db      characterset:   latin1
Client characterset:    utf8
Conn.   characterset:   utf8
UNIX socket:            /var/run/mysqld/mysqld.sock
Uptime:                 4 hours 14 min 12 sec
Threads: 1  Questions: 101  Slow queries: 0  Opens: 129
Flush tables: 1  Open tables: 42  Queries per second avg: 0.006
```

Figura 63. Mostra de la configuració actual de MySQL.
Font: elaboració pròpia.

Si en algun instant necessitem accedir al terminal de Linux, podem emprar l'ordre **system** per invocar les seves ordres des de l'interior de MySQL:

```
mysql> system (ls /home/eblanco);

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

mysql> system (cat comandos.sql);

USE catalogo;

SHOW TABLES;

DESCRIBE genes;
```

Figura 64. Execució de l'intèrpret d'ordres de Linux a MySQL.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.13. Còpies de seguretat i restauració de BBDD

És altament recomanable dur a terme còpies de seguretat de les nostres dades amb certa periodicitat. En el cas de les bases de dades gestionades amb MySQL, el programa **mysqldump**, executat des del terminal de Linux, realitza un bolcat complet del seu contingut cap a un fitxer de text. Posteriorment, en cas de ser necessari, aquest fitxer d'ordres pot ser executat amb la instrucció **source** per regenerar la base de dades completa, creant automàticament les taules i inserint-hi els registres existents en aquell moment:

```
% mysqldump -vp catalogo > catalogo.sql

Enter password:
-- Connecting to localhost...
-- Retrieving table structure for table anotaciones...
-- Sending SELECT query...
-- Retrieving rows...
-- Retrieving table structure for table funciones...
-- Sending SELECT query...
-- Retrieving rows...
-- Retrieving table structure for table genes...
-- Sending SELECT query...
-- Retrieving rows...
-- Retrieving table structure for table genomas...
-- Sending SELECT query...
-- Retrieving rows...
-- Disconnecting from localhost...

% more catalogo.sql

-- MySQL dump 10.13 Distrib 5.7.17, for Linux (i686)
-- Host: localhost Database: catalogo
-- Server version 5.7.17-0ubuntu0.16.04.1
...
CREATE TABLE `anotaciones` (
...
INSERT INTO `anotaciones` VALUES ('ash2','GO:0003700','Computacional'),...
...
```

Figura 65. Realitzar una còpia de seguretat amb el programa mysqldump.
Font: elaboració pròpia.

També podem restaurar una base de dades MySQL des del terminal a partir d'un fitxer *backup* així:

```
mysql -u usuario -p basededatos < basededatos.sql
```

1. Bases de dades relacionals

1.14. Exemple pràctic: gestió del catàleg de gens humans

Per demostrar com extreure coneixement útil d'una base de dades relacional, us proposem analitzar amb MySQL el contingut d'un catàleg de gens humans. Un gen és un fragment d'ADN ubicat en el genoma que conté la informació precisa per sintetitzar una molècula d'ARN. En els organismes eucariotes, un gen està constituït per una successió de fragments útils denominats *exons*. En una proporció significativa dels gens humans hi ha diverses combinacions alternatives d'exons, donant lloc a diferents formes alternatives d'un mateix gen, denominades *transcrits alternatius*. Per codificar la informació relativa a la localització dels gens en el genoma és freqüent utilitzar fitxers de text tabulat. Cada línia d'aquests fitxers conté els valors dels atributs que caracteritzen un transcrit d'un determinat gen. Bàsicament, un transcrit d'un gen posseeix una localització concreta, identificada per un cromosoma, una posició inicial/final i una direcció de lectura. Altres característiques que podem recuperar sobre un transcrit són el seu codi, el nom del gen, el nombre d'exons o les seves coordenades exactes.

> **Vegeu també**
>
> Per revisar els conceptes de *genoma*, *cromosoma*, *gen* i *proteïna* us recomanem l'assignatura Fonaments de biologia molecular.

El navegador genòmic d'UCSC representa gràficament els diferents tipus d'anotacions existents sobre el genoma humà en forma de centenars de pistes. Per administrar eficientment aquest elevat volum d'informació, una còpia del SGBD MySQL està funcionant de forma transparent als milers d'usuaris que cada dia visiten aquest servidor. D'aquesta manera, en el cas que volguem reproduir una pista al nostre ordinador, disposem a la secció de descàrregues d'un fitxer SQL per ser executat amb l'ordre `source` i un fitxer de text amb el conjunt de dades que s'han d'importar amb la instrucció `LOAD DATA`. En aquest exercici utilitzarem l'anotació dels gens humans distribuïda pel consorci **RefSeq** per al genoma humà. Aquest format és comú a totes les espècies subministrades pel navegador.

> **Vegeu també**
>
> És possible aprofundir sobre el funcionament dels navegadors genòmics en l'assignatura Genòmica computacional.

Ara procedirem a descarregar-nos els dos fitxers associats a la pista **refGene**, que conté el catàleg de gens humans anotats pel consorci **RefSeq**, en la seva versió **hg38**.

Per a això, hem d'utilitzar l'ordre `wget` per transferir tots dos fitxers al nostre terminal.

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.sql
```

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz
```

Mostrem a continuació el contingut del fitxer **refGene.sql** que realitza la creació de la taula **refGene**. Els atributs que consultarem amb més freqüència seran (figura 66): **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 de **name** i **name2**: un gen pot tenir diversos transcrits, però un transcrit únicament pertany a un gen.

```
CREATE TABLE 'refGene' (
  'bin' smallint(5) unsigned NOT NULL,
  'name' varchar(255) NOT NULL,
  'chrom' varchar(255) NOT NULL,
  'strand' char(1) NOT NULL,
  'txStart' int(10) unsigned NOT NULL,
  'txEnd' int(10) unsigned NOT NULL,
  'cdsStart' int(10) unsigned NOT NULL,
  'cdsEnd' int(10) unsigned NOT NULL,
  'exonCount' int(10) unsigned NOT NULL,
  'exonStarts' longblob NOT NULL,
  'exonEnds' longblob NOT NULL,
  'score' int(11) DEFAULT NULL,
  'name2' varchar(255) NOT NULL,
  'cdsStartStat' enum('none','unk','incmpl','cmpl') NOT NULL,
  'cdsEndStat' enum('none','unk','incmpl','cmpl') NOT NULL,
  'exonFrames' longblob NOT NULL,
  KEY 'chrom' ('chrom','bin'),
  KEY 'name' ('name'),
  KEY 'name2' ('name2')
) ENGINE-MyISAM DEFAULT CHARSET-latin1;
```

Figura 66. Atributs dels transcrits anotats pel consorci RefSeq.
Font: elaboració pròpia.

Passarem ara a visualitzar amb el terminal el segon fitxer **refGene.txt**. Aquest arxiu conté les dades del catàleg complet de gens anotats en el genoma humà. Hem de carregar aquesta informació a la nostra base de dades un cop estigui creada la taula **refGene**. En el context d'aquest exercici, cada registre conté informació sobre el transcrit d'un determinat gen. En el cas que un gen posseeixi diversos transcrits, cadascun es codifica en registres separats (cadascú amb el seu propi codi i les seves corresponents coordenades).

En primer lloc, hem de descomprimir el fitxer amb l'ordre **gzip**.

```
% gzip -d refGene.txt.gz

% head -5 refGene.txt
585 NR_046018 chr1 + 11873 14409 14409 14409 3 11873,12612,13220, 12227,12721,14409, 0 DDX11L1
unk unk -1,-1,-1,

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,

932 NR_104645 chrX + 45505387 45523644 45523644 45523644 3 45505387,45510496,45521607, 45505465,
45510595,45523644, 0 LINC01204 unk unk -1,-1,-1,

1078 NR_104148 chr7 + 64666082 64687830 64687830 64687830 4 64666082,64669036,64679176,64684334,
64666285,64669178,64679336,64687830, 0 ZNF107 unk unk -1,-1,-1,-1,

103 NR_120408 chr14 + 31561384 31861223 31861223 31861223 10 31561384,31562067,31565013,31599288,
31673354,31673483,31826628,31846470,31850118,31859117, 31561547,31562215,31565048,31599379,
31673394,31673574,31826714,31846591,31850201,31861223, 0NUBPL unk unk -1,-1,-1,-1,-1,-1,-1,-1,
-1,
```

Figura 67. El catàleg refGene.txt de gens humans.
Font: elaboració pròpia.

> 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.

Un cop dins de l'intèrpret de MySQL, indicarem que treballarem dins de la nostra base **de dades**.

Executarem, posteriorment, el fitxer **refGene.sql** amb l'ordre `source` per crear la taula **refGene**.

Per verificar que la instrucció anterior ha funcionat correctament, podem veure el llistat d'atributs de la taula **refGene** amb l'ordre `DESCRIBE`:

```
mysql> USE catalogo;

Database changed

mysql> source refGene.sql;

Query OK, 0 rows affected (0,00 sec)

mysql> DESCRIBE refGene;

+--------------+-----------------------------------+------+-----+---------+-------+
| Field        | Type                              | Null | Key | Default | Extra |
+--------------+-----------------------------------+------+-----+---------+-------+
| bin          | smallint(5) unsigned              | NO   |     | NULL    |       |
| name         | varchar(255)                      | NO   | MUL | NULL    |       |
| chrom        | varchar(255)                      | NO   | MUL | NULL    |       |
| strand       | char(1)                           | NO   |     | NULL    |       |
| txStart      | int(10) unsigned                  | NO   |     | NULL    |       |
| txEnd        | int(10) unsigned                  | NO   |     | NULL    |       |
| cdsStart     | int(10) unsigned                  | NO   |     | NULL    |       |
| cdsEnd       | int(10) unsigned                  | NO   |     | NULL    |       |
| exonCount    | int(10) unsigned                  | NO   |     | NULL    |       |
| exonStarts   | longblob                          | NO   |     | NULL    |       |
| exonEnds     | longblob                          | NO   |     | NULL    |       |
| score        | int(11)                           | YES  |     | NULL    |       |
| name2        | varchar(255)                      | NO   | MUL | NULL    |       |
| cdsStartStat | enum('none','unk','incmpl','cmpl') | NO   |     | NULL    |       |
| cdsEndStat   | enum('none','unk','incmpl','cmpl') | NO   |     | NULL    |       |
| exonFrames   | longblob                          | NO   |     | NULL    |       |
+--------------+-----------------------------------+------+-----+---------+-------+
16 rows in set (0,14 sec)
```

Figura 68. Creació de la taula refGene.
Font: elaboració pròpia.

> Assumirem que ambdós fitxers (refGene.sql i refGene.txt) estan guardats en la mateixa carpeta de treball des de la qual hem invocat el programa MySQL, anteriorment.

El segon pas consisteix a poblar la taula amb les anotacions dels gens humans que hem descarregat dins del fitxer **refGene.txt**. Fent servir l'ordre **LOAD DATA** podem bolcar tot el contingut a la taula **refGene**:

```
mysql> LOAD DATA LOCAL INFILE 'refGene.txt' INTO TABLE refGene;

Query OK, 69853 rows affected (2,54 sec)
Records: 69853  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 69. Poblar la taula refGene.
Font: elaboració pròpia.

Ens trobem en condicions de començar a interrogar la base de dades. Recordem, novament, que cada registre de la taula **refGene** alberga la informació associada a l'espai d'un gen en particular. Igualment, és important tenir en compte que una elevada fracció dels gens humans posseeix dos o més transcrits alternatius. La nostra missió, a continuació, és mostrar l'enorme potencial de les consultes de SQL a l'hora d'extreure nou coneixement biològic de les dades emmagatzemades a les taules de la nostra base de dades.

Començarem mostrant els primers registres de la nostra taula, incloent-hi únicament diversos dels seus atributs per afavorir la llegibilitat dels valors dels registres per pantalla:

```
mysql> SELECT name2,name,chrom,strand,txStart,txEnd,exonCount
    -> FROM refGene ORDER BY name2 LIMIT 10;

+---------+-------------+-------+--------+----------+----------+-----------+
| name2   | name        | chrom | strand | txStart  | txEnd    | exonCount |
+---------+-------------+-------+--------+----------+----------+-----------+
| A1BG    | NM_130786   | chr19 | -      | 58346805 | 58353499 |         8 |
| A1BG-AS1| NR_015380   | chr19 | +      | 58351969 | 58355183 |         4 |
| A1CF    | NM_001198819| chr10 | -      | 50799408 | 50885675 |        15 |
| A1CF    | NM_014576   | chr10 | -      | 50799408 | 50885675 |        13 |
| A1CF    | NM_138932   | chr10 | -      | 50799408 | 50885675 |        13 |
| A1CF    | NM_001198820| chr10 | -      | 50799408 | 50885675 |        14 |
| A1CF    | NM_001198818| chr10 | -      | 50799408 | 50885675 |        14 |
| A1CF    | NM_138933   | chr10 | -      | 50799408 | 50885675 |        13 |
| A2M     | NM_001347423| chr12 | -      |  9067707 |  9116229 |        37 |
| A2M     | NM_000014   | chr12 | -      |  9067707 |  9116229 |        36 |
+---------+-------------+-------+--------+----------+----------+-----------+
10 rows in set (0,00 sec)
```

Figura 70. Mostra del contingut de la taula refGene.
Font: elaboració pròpia.

Atès que cada registre conté la informació d'un transcrit, el nombre de transcrits coneguts en el genoma humà coincidirà amb el nombre de registres emmagatzemats a la taula **refGene**. Aquest comptatge és senzill:

```
mysql> SELECT COUNT(*) FROM refGene;

+----------+
| COUNT(*) |
+----------+
|    69853 |
+----------+
1 row in set (0,00 sec)
```

Figura 71. Comptar els transcrits de la taula refGene.
Font: elaboració pròpia.

També podem comptar fàcilment el nombre total de gens codificats en el genoma humà. Si un gen posseeix diversos transcrits alternatius, llavors trobarem diversos registres en el nostre catàleg que posseeixen un valor diferent de l'atribut **name**, però que comparteixen el mateix valor per a l'atribut **name2**. Per tant, emprant la clàusula DISTINCT sobre aquest últim atribut, comptarem una única vegada cada gen de la nostra taula, encara que posseeixi diverses formes alternatives:

```
mysql> SELECT COUNT(DISTINCT name2) FROM refGene;

+----------------------+
| COUNT(DISTINCT name2) |
+----------------------+
|                27656 |
+----------------------+
1 row in set (0,07 sec)
```

Figura 72. Comptar els gens de la taula refGene.
Font: elaboració pròpia.

Si agrupem els registres de la taula per l'atribut **name2**, podem elaborar un inventari del nombre de transcrits alternatius anotats per a cada gen.

```
mysql> SELECT name2,COUNT(name2)
    -> FROM refGene GROUP BY name2 LIMIT 10;

+----------+--------------+
| name2    | COUNT(name2) |
+----------+--------------+
| A1BG     |            1 |
| A1BG-AS1 |            1 |
| A1CF     |            6 |
| A2M      |            4 |
| A2M-AS1  |            3 |
| A2ML1    |            2 |
| A2MP1    |            1 |
| A3GALT2  |            1 |
| A4GALT   |            3 |
| A4GNT    |            1 |
+----------+--------------+
10 rows in set (0,00 sec)
```

Figura 73. Comptar el nombre de transcrits de cada gen de la taula refGene.
Font: elaboració pròpia.

Podem obtenir resultats interessants aplicant la clàusula WHERE sobre els atributs de cada registre. Per exemple, imaginem que desitgem conèixer el nombre de transcrits ubicats en cada cadena de la molècula d'ADN:

```
mysql> SELECT COUNT(*) FROM refGene WHERE strand LIKE '+';

+----------+
| COUNT(*) |
+----------+
|    35724 |
+----------+
1 row in set (0,10 sec)

mysql> SELECT COUNT(*) FROM refGene WHERE strand LIKE '-';

+----------+
| COUNT(*) |
+----------+
|    34129 |
+----------+
1 row in set (0,09 sec)
```

Figura 74. Comptar en una cadena d'ADN.
Font: elaboració pròpia.

També podem comptar el nombre de transcrits localitzats en un cromosoma:

```
mysql> SELECT COUNT(*) FROM refGene WHERE chrom LIKE 'chr21';

+----------+
| COUNT(*) |
+----------+
|      961 |
+----------+
1 row in set (0,00 sec)
```

Figura 75. Comptar els transcrits d'un cromosoma.
Font: elaboració pròpia.

Novament, jugant amb l'atribut **name2** podem comptar el nombre de gens codificats en el mateix cromosoma:

```
mysql> SELECT COUNT(DISTINCT name2)
    -> FROM refGene WHERE chrom LIKE 'chr21';

+----------------------+
| COUNT(DISTINCT name2) |
+----------------------+
|                  408 |
+----------------------+
1 row in set (0,01 sec)
```

Figura 76. Comptar els gens d'un cromosoma.
Font: elaboració pròpia.

O identificar quins són els transcrits que posseeixen un major nombre d'exons:

```
mysql> SELECT name2,name,exonCount
    -> FROM refGene ORDER BY exonCount DESC LIMIT 10;

+-------+-------------+-----------+
| name2 | name        | exonCount |
+-------+-------------+-----------+
| TTN   | NM_001267550 |       363 |
| TTN   | NM_001256850 |       313 |
| TTN   | NM_133378    |       312 |
| TTN   | NM_133437    |       192 |
| TTN   | NM_133432    |       192 |
| TTN   | NM_003319    |       191 |
| NEB   | NM_001271208 |       183 |
| NEB   | NM_001164507 |       182 |
| NEB   | NM_001164508 |       182 |
| MUC19 | NM_173600    |       174 |
+-------+-------------+-----------+
10 rows in set (0,11 sec)
```

Figura 77. Recuperar els transcrits amb major nombre d'exons.
Font: elaboració pròpia.

També podem seleccionar aquells que posseeixen un únic exó:

```
mysql> SELECT name2,name,exonCount
     > FROM refGene WHERE exonCount=1
     > ORDER BY name2 LIMIT 10;

+-------------+-------------+-----------+
| name2       | name        | exonCount |
+-------------+-------------+-----------+
| AADACL2-AS1 | NR_110203   |         1 |
| ABALON      | NR_131907   |         1 |
| ABHD16B     | NM_080622   |         1 |
| ACKR1       | NM_001122951 |         1 |
| ACKR4       | NM_178445   |         1 |
| ACTBL2      | NM_001017992 |         1 |
| ACTG1P20    | NR_033926   |         1 |
| ACTG1P4     | NR_024438   |         1 |
| ACTL10      | NM_001024675 |         1 |
| ACTL7A      | NM_006687   |         1 |
+-------------+-------------+-----------+
10 rows in set (0,00 sec)
```

Figura 78. Recuperar els transcrits amb un únic exó.
Font: elaboració pròpia.

És possible calcular el nombre d'exons, de mitjana, per cada transcrit:

```
mysql> SELECT AVG(exonCount) FROM refGene;

+----------------+
| AVG(exonCount) |
+----------------+
|         9.4126 |
+----------------+
1 row in set (0,11 sec)
```

Figura 79. Calcular el nombre d'exons de mitjana per cada transcrit.
Font: elaboració pròpia.

I la longitud mitjana dels gens humans:

```
mysql> SELECT AVG(txEnd-txStart+1) FROM refGene;

+----------------------+
| AVG(txEnd-txStart+1) |
+----------------------+
|           56983.2770 |
+----------------------+
1 row in set (0,10 sec)
```

Figura 80. Calcular la longitud mitjana dels gens.
Font: elaboració pròpia.

Finalment, integrarem en aquesta anàlisi el genoma de ratolí domèstic. Descarreguem els fitxers **refGene.sql** i **refGene.txt** d'aquesta espècie en la seva versió mm9.

Per evitar sobreescriure les anotacions humanes, hem de gravar ambdós fitxers amb un nom diferent (per exemple, **refGene_mouse.sql** i **refGene_mouse.txt**). Posteriorment, cal editar el contingut del fitxer SQL per modificar el nom de la taula, per la mateixa raó (figura 81).

Després d'aquestes modificacions, ja estem en condicions de llançar la creació de la nova taula amb l'ordre **source** i la seva repoblació amb les dades relatives al genoma del ratolí amb l'ordre LOAD DATA.

```
DROP TABLE IF EXISTS 'refGene_mouse';

CREATE TABLE 'refGene_mouse' (
  'bin' smallint(5) unsigned NOT NULL,
  'name' varchar(255) NOT NULL,
  'chrom' varchar(255) NOT NULL,
  'strand' char(1) NOT NULL,
...
-----------------------------------------

mysql> source 'refGene_mouse.sql';

Query OK, 0 rows affected (0,00 sec)

mysql> LOAD DATA LOCAL INFILE 'refGene_mouse.txt'
    -> INTO TABLE refGene_mouse;

Query OK, 34904 rows affected (1,23 sec)
Records: 34904  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 81. Incorporar els gens de ratolí a la nostra base de dades.
Font: elaboració pròpia.

Comprovem que els registres emmagatzemats a la nova taula són correctes:

```
mysql> SELECT name2,name,chrom,strand,txStart,txEnd,exonCount
    -> FROM refGene_mouse ORDER BY name2 LIMIT 10;

+---------------+--------------+-------+--------+-----------+-----------+-----------+
| name2         | name         | chrom | strand | txStart   | txEnd     | exonCount |
+---------------+--------------+-------+--------+-----------+-----------+-----------+
| 0610005C13Rik | NR_038166    | chr7  | -      |  52823164 |  52830546 |         5 |
| 0610005C13Rik | NR_038165    | chr7  | -      |  52823164 |  52830546 |         4 |
| 0610007P14Rik | NM_021446    | chr12 | -      |  87156404 |  87165495 |         5 |
| 0610009B22Rik | NM_025319    | chr11 | -      |  51498886 |  51502136 |         2 |
| 0610009L18Rik | NR_038126    | chr11 | +      | 120209991 | 120212504 |         2 |
| 0610009O20Rik | NM_024179    | chr18 | +      |  38409902 |  38422283 |        13 |
| 0610010B08Rik | NM_001177543 | chr2  | -      | 175017505 | 175163713 |         6 |
| 0610010B08Rik | NM_001177543 | chr2  | -      | 174952492 | 175261278 |         6 |
| 0610010B08Rik | NM_001177543 | chr2  | +      | 175639522 | 175655901 |         5 |
| 0610010B08Rik | NM_001177543 | chr2  | +      | 175737073 | 175753460 |         5 |
+---------------+--------------+-------+--------+-----------+-----------+-----------+
10 rows in set (0,00 sec)
```

Figura 82. Visualitzar els primers transcrits del genoma del ratolí domèstic.
Font: elaboració pròpia.

Ara, si seleccionem aquells registres de les dues taules que pertanyen al mateix gen en ambdues espècies, podem construir un catàleg de gens homòlegs.

Podem dur a terme aquesta associació perquè SQL no distingeix entre majúscules o minúscules a l'hora de comparar la columna **name2**.

```
mysql> SELECT DISTINCT refGene.name2,refGene.chrom,refGene.strand,
    -> refGene.txStart,refGene.txEnd,refGene.exonCount,
    -> refGene_mouse.name2,refGene_mouse.chrom,
    -> refGene_mouse.strand,refGene_mouse.txStart,
    -> refGene_mouse.txEnd,refGene_mouse.exonCount
    -> FROM refGene JOIN refGene_mouse
    -> ON refGene.name2 = refGene_mouse.name2
    -> ORDER BY refGene.name2 ASC LIMIT 10;

+---------+-------+--------+----------+----------+-----------+---------+-------+--------+-----------+-----------+-----------+
| name2   | chrom | strand | txStart  | txEnd    | exonCount | name2   | chrom | strand | txStart   | txEnd     | exonCount |
+---------+-------+--------+----------+----------+-----------+---------+-------+--------+-----------+-----------+-----------+
| A1BG    | chr19 | -      | 58346805 | 58353499 |         8 | A1bg    | chr15 | -      |  60749143 |  60752825 |         7 |
| A1CF    | chr10 | -      | 50799408 | 50885675 |        13 | A1cf    | chr19 | +      |  31943250 |  32023896 |        12 |
| A1CF    | chr10 | -      | 50799408 | 50885675 |        14 | A1cf    | chr19 | +      |  31943250 |  32023896 |        12 |
| A1CF    | chr10 | -      | 50799408 | 50885675 |        15 | A1cf    | chr19 | +      |  31943250 |  32023896 |        12 |
| A2M     | chr12 | -      |  9067707 |  9116229 |        35 | A2m     | chr6  | +      | 121586190 | 121629256 |        36 |
| A2M     | chr12 | -      |  9067707 |  9116229 |        36 | A2m     | chr6  | +      | 121586190 | 121629256 |        36 |
| A2M     | chr12 | -      |  9067707 |  9116229 |        37 | A2m     | chr6  | +      | 121586190 | 121629256 |        36 |
| A3GALT2 | chr1  | -      | 33306765 | 33321098 |         5 | A3galt2 | chr4  | +      | 128436501 | 128446542 |         5 |
| A4GALT  | chr22 | -      | 42692111 | 42720910 |         3 | A4galt  | chr15 | -      |  83057151 |  83082161 |         3 |
| A4GALT  | chr22 | -      | 42692111 | 42720910 |         3 | A4galt  | chr15 | -      |  83057151 |  83082204 |         3 |
+---------+-------+--------+----------+----------+-----------+---------+-------+--------+-----------+-----------+-----------+
10 rows in set (5,14 sec)
```

Figura 83. Llistat de gens comuns entre el genoma humà i el genoma de ratolí.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.15. *Triggers*, procediments i funcions

La majoria de bases de dades relacionals ofereixen la possibilitat d'emmagatzemar subprogrames. Es denominen funcions, procediments i *triggers* o disparadors, i són molt útils per automatitzar tasques i guardar instruccions SQL que s'han d'utilitzar freqüentment. Són objectes que contenen codi SQL, com breus *scripts* de codi SQL, que poden acceptar paràmetres i declarar variables.

S'assigna un nom al subprograma i s'executa perquè quedi emmagatzemat a la base de dades. Després el podem invocar o *cridar* pel seu nom perquè s'executi el codi emmagatzemat en els subprogrames.

- **Procediment** emmagatzemat. És un objecte que es crea amb la sentència `CREATE PROCEDURE` i s'invoca amb la sentència `CALL`. Els procediments poden acceptar paràmetres i no fan cap retorn, és a dir, no retornen cap
- **Funció** emmagatzemada. És un objecte que es crea amb la sentència `CREATE FUNCTION` i s'invoca amb la sentència **SELECT** o dins d'una expressió. Les **funcions** poden acceptar paràmetres i retornen sempre un valor. Aquest valor retornat pot ser un valor nul i en aquest cas es comportaria com un procediment.
- ***Trigger***. És un objecte que es crea amb la sentència `CREATE TRIGGER` i ha d'estar associat a una taula. Un *trigger* s'activa, es dispara, quan ocorre un esdeveniment d'inserció, actualització o esborrat, sobre la taula a la qual està associat.

Vegem-ne alguns exemples:

Procediments

1. Creem un procediment a la base de dades **catalogo** per comptar el nombre de gens diferents de la taula **refGene**, que anomenem **numGenes**.

```
CREATE PROCEDURE numGenes()
SELECT COUNT(distinct name2) FROM refGene;
```

Aquest procediment no té paràmetres () i mostra per pantalla el nombre de gens diferents que trobem al camp **name2** de la taula **refGene**. En executar el codi, el procediment s'emmagatzema a la base de dades com un objecte més, com les taules, i no s'executa la consulta `SELECT` que conté fins que l'anomenem.

Per anomenar el procediment **numGenes** escrivim:

```
CALL numGenes();
```

Si volem veure els procediments emmagatzemats a la base de dades escrivim:

```
SHOW PROCEDURE STATUS WHERE db = 'catalogo';
```

Si volem eliminar un procediment creat escrivim:

```
DROP PROCEDURE IF EXISTS numGenes;
```

2. Ara utilizarem variables, el resultat del **SELECT** el guardarem en la variable local **genes**, que hem de declarar i de la qual hem de definir el tipus de dades.

Com que en el procediment hi ha sentències que acaben en «;», primer assignarem un nou delimitador. Escrivim:

```
DELIMITER //
```

Ara MySQL interpretarà que el final de les sentències SQL és el símbol //

Crearem un nou procediment anomenat **numGenes2**.

Per declarar la variable **genes** necessitem escriure BEGIN abans de DECLARE i acabar amb END.

En declarar les variables locals cal especificar quin tipus de dada guardaran, INT, CHAR, VARCHAR, etc. En aquest cas és un INT.

```
DECLARE gens INT;
```

Per guardar el resultat del SELECT en la variable local **genes** fem:

```
INTO gens.
```

Per mostrar el contingut de la variable **genes** fem:

```
SELECT gens;
```

Veiem tot el codi per crear el nou procediment usant variables:

```
CREATE PROCEDURE numGenes2()
BEGIN
DECLARE gens INT;
SELECT COUNT(distinct name2)
INTO gens
FROM refGene;
SELECT gens;
END //
```

Tornem a canviar el delimitador per poder fer servir «;» en finalitzar la sentència DELIMITER ;

Invoquem el procediment:

```
CALL numGenes2();
```

3. Ara passarem un paràmetre al procediment i l'utilitzarem en la condició WHERE de la consulta SELECT.

Creem un nou procediment anomenat **numTrans** que ens servirà per comptar el nombre de transcrits segons el tipus de transcrit que li passem com a paràmetre.

Igual que les variables locals, cal especificar als paràmetres quin tipus de dades esperen, INT, CHAR, VARCHAR... En aquest cas, `CHAR( 50)`.

El paràmetre el posem al costat del nom del procediment entre parèntesis `numTrans(transcritType CHAR(50))`.

Hi assignem una altra vegada un delimitador `//`:

```
DELIMITER //
```

Vegem el codi:

```
CREATE PROCEDURE numTrans ( transcritType CHAR(50))
BEGIN
DECLARE numT INT;
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
SELECT numT;
END //
```

Canviem una altra vegada el delimitador:

```
DELIMITER ;
```

Invoquem el procediment passant-li el paràmetre NR:

```
CALL numTrans('%NR%');
```

Ara invoquem el procediment passant-li el paràmetre NM:

```
CALL numTrans('%NM%');
```

En usar el comodí % ens assegurem que no perdem cap registre amb el contingut del paràmetre.

4. En comptes de declarar una variable local dins d'un procediment, també podem utilitzar un paràmetre com a variable de sortida. Indiquem que és un paràmetre de sortida amb la clàusula **OUT**. Per defecte, els paràmetres són només d'entrada, però també els podem indicar amb la clàusula **IN**.

Hi assignem un delimitador //:

```
DELIMITER //
```

Creem un nou procediment anomenat **numTrans2** que ens servirà també per comptar el nombre de transcrits segons el tipus de transcrit que li passem com a paràmetre, i amb un segon paràmetre que ens servirà per guardar el resultat de la consulta.

Vegem el codi:

```
CREATE PROCEDURE numTrans2(IN transcritType CHAR(50), OUT numT INT)
BEGIN
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
END //
```

Canviem el delimitador:

```
DELIMITER ;
```

Invoquem el procediment passant-li els dos paràmetres. Fem servir una variable definida per l'usuari amb @ trucada `@transcritos` per guardar el valor que retorna el **SELECT** del procediment:

```
CALL numTrans2('%NR%', @transcrits);
```

Fem un **SELECT** de la variable `@transcritos` que conté el nombre de transcrits:

```
SELECT @transcrits;
```

Funcions

Les funcions retornen un valor, així que, per anomenar una funció emmagatzemada, en comptes de fer `CALL` fem directament `SELECT nombre_de_la_función` i ens mostra el valor que retorna la funció.

Hi assignem un delimitador //:

```
DELIMITER //
```

Creem una nova funció anomenada **numTrans3** que ens servirà també per comptar el nombre de transcrits segons el tipus de transcrit que li passem com a paràmetre.

En les funcions hem d'indicar, després del nom i dels paràmetres, el tipus de dada que retorna la funció, en aquest cas un INT, i escrivim:

```
RETURNS INT
```

Al final de la funció li indiquem la variable que volem retornar, en aquest cas:

```
RETURN numT;
```

Vegem el codi íntegre de la funció:

```
CREATE FUNCTION numTrans3 (transcritType CHAR(50)) RETURNS INT
BEGIN
DECLARE numT INT;
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
RETURN numT;
END //
```

Canviem el delimitador:

```
DELIMITER ;
```

Invoquem la funció passant-li el paràmetre, en aquest cas els passem el paràmetre NR:

```
SELECT numTrans3('%NR%');
```

Si passem a la funció el paràmetre NM:

```
SELECT numTrans3('%NM%');
```

Si volem veure les funcions emmagatzemades a la base de dades escrivim la sentència:

```
SHOW FUNCTION STATUS WHERE db = 'catalogo';
```

Si volem eliminar una funció emmagatzemada, escrivim:

```
DROP FUNCTION IF EXISTS numTrans3;
```

Triggers

Un *trigger* és un objecte emmagatzemat a la base de dades que està associat amb una taula i que s'activa quan ocorre un esdeveniment sobre la taula.

Els esdeveniments que poden ocórrer sobre la taula són:

- INSERT. El *trigger* s'activa quan s'insereix una nova fila sobre la taula associada.
- UPDATE. El *trigger* s'activa quan s'actualitza una fila sobre la taula associada.
- DELETE. El *trigger* s'activa quan s'elimina una fila sobre la taula associada.

El *trigger* es pot activar o disparar abans (BEFORE) de l'esdeveniment o després (AFTER) de l'esdeveniment.

Com a exemples vam crear dos *triggers* associats a la taula *genes* de la nostra base de dades:

Un *trigger* amb el nom de **trig_check_genes_before_insert** que s'associa a la taula **genes**. S'activa abans d'una operació d'inserció. Si el nou valor del camp inici que es vol inserir és negatiu, es guarda com a 0. Si el nou valor del camp **final** que es vol inserir és menor que el valor del camp **inici**, es guarda el valor del camp **inici**.

Un t*rigger* amb el nom de **trig_check_genes_before_update** que s'associa a la taula **genes**. S'activa abans d'una operació de modificació. Si el nou valor del camp **inici** que es vol modificar és negatiu, es guarda com a 0. Si el valor del camp **final** del registre que es vol modificar és menor que el nou valor que volem actualitzar del camp **inici** es guarda com a 1.

Creem el *trigger* **trig_check_genes_before_insert**

```
DELIMITER //
CREATE TRIGGER trig_check_genes_before_insert
BEFORE INSERT
ON genes FOR EACH ROW
BEGIN IF NEW.inicio < 0 THEN SET NEW.inicio = 0;
ELSEIF NEW.inicio > NEW.final THEN SET NEW.final = NEW.inicio;
END IF;
END//
```

En executar aquest codi de creació del *trigger,* **trig_check_genes_before_insert** queda emmagatzemat a la nostra base de dades, i només actuarà, s'activarà quan l'usuari executi una sentència d'inserció, per exemple:

Canviem el delimitador:

```
DELIMITER ;
```

Realitzem operacions d'inserció a la taula *genes* perquè es dispari el *trigger* **trig_check_genes_before_insert**:

```
INSERT INTO genes (nom, cromosoma, filament, inici, final, proteïna, espècie) VALUES (

INSERT INTO genes (nom, cromosoma, filament, inici, final, proteïna, espècie) VALUES (
```

La variable composta **NEW** que utilitzem en el *trigger* emmagatzema tots els valors que inserim en cada operació **INSERT**. D'aquesta manera podem utilitzar-la en el *trigger* sense conèixer *a priori* quins valors s'hi inseriran.

En els nostres exemples la variable **NEW.inici** conté en el primer **INSERT** el valor -2527305 i en el segon **INSERT** la variable **NEW.inici** conté el valor 252730599 i la variable **NEW.final** conté el valor 2575270

En el primer **INSERT** es compleix `NEW.inici < 0`, aquesta condició dispara el *trigger* i en el camp **inici** es guarda el valor 0.

En el segon **INSERT** es compleix `NEW.inici > NEW.final`, aquesta condició dispara el *trigger* i en el camp **final** es guarda el valor `252730599`.

Ara creem el *trigger* **trig_check_genes_before_update**:

```
DELIMITER //
CREATE TRIGGER trig_check_genes_before_update
BEFORE UPDATE ON genes
FOR EACH ROW
BEGIN
IF NEW.inicio < 0 THEN SET NEW.inicio = 0;
ELSEIF NEW.inicio > OLD.final THEN SET NEW.inicio = 1;
END IF;
END //
```

En executar aquest codi de creació del *trigger*, **trig_check_genes_before_update** queda emmagatzemat a la nostra base de dades, i només actuarà, s'activarà quan l'usuari executi una sentència d'actualització, per exemple:

Canviem el delimitador:

```
DELIMITER ;
```

Realitzem operacions de modificació a la taula *genes* perquè es dispari el *trigger* **trig_check_genes_before_update**:

```
UPDATE genes SET inici = 228748314 WHERE nom = 'MYC';

UPDATE genes SET inici = -47643 WHERE nom = 'cbt';
```

En aquest cas, la variable composta **NEW** que utilitzem en el *trigger* emmagatzema el nou valor que volem actualitzar en la sentència **UPDATE**. En el primer **UPDATE** la variable `NEW.inici` conté el valor 228748314 i en el segon **UPDATE** la variable `NEW.inici` conté el valor `-47643`.

En canvi, la variable composta **OLD** emmagatzema tots els valors vells ja emmagatzemats a la taula del registre que es vol actualitzar. En el primer **UPDATE**, la variable `OLD.final` conté el valor emmagatzemat en el camp final del registre amb clau primària **MYC** en la taula **genes**, i en el segon **UPDATE**, la variable `OLD.final` conté el valor emmagatzemat en el camp final del registre amb clau primària **cbt**.

En el primer **UPDATE** es compleix `NEW.inici > OLD.final`, aquesta condició dispara el *trigger* i en el camp inici es guarda el valor 1.

En el segon **UPDATE** es compleix `NEW.inici < 0`, aquesta condició dispara el *trigger* i en el camp inici es guarda el valor 0.

En les operacions **DELETE** només s'utilitza la variable composta **OLD,** capaç d'emmagatzemar tots els valors del registre que es vol eliminar i accedir-hi, amb el format `OLD.nombre_campo`.

2. Bases de dades NoSQL

2.1. Introducció

Les bases de dades relacionals són molt eficients i mantenen la integritat de les dades, però tenen limitacions per gestionar amb rapidesa grans volums d'informació.

Les bases de dades relacionals escalen de forma vertical. Per créixer necessiten que els servidors tinguin més capacitat.

MySQL és un dels SGBD més utilitzats per a projectes web, però les noves aplicacions web es caracteritzen per haver de gestionar un immens volum d'informació i gran quantitat de dades.

Per afrontar aquest nou repte de gestió de les dades, van aparèixer les bases de dades no relacionals, conegudes també com NoSQL o Not Only SQL, i anomenades així perquè no depenen únicament del llenguatge estructurat SQL.

Les bases de dades no relacionals poden escalar de forma horitzontal, permeten la distribució dels processos de treball i conjunts de dades en múltiples servidors. D'aquesta manera és possible que l'escalabilitat d'aquestes bases de dades sigui pràcticament il·limitada.

Les bases de dades NoSQL es classifiquen com de clau/valor, orientades a documents, grafs, o de famílies de columnes.

En aquest mòdul ens centrarem en les bases de dades no relacionals orientades a documents, concretament a documents en format JSON.

2. Bases de dades NoSQL

2.2. Ficheros JSON

JavaScript Object Notation (JSON) és un format de dades basat en text estàndard per representar dades estructurades que segueix la sintaxi d'objecte de JavaScript. Tot i que és molt semblant a la sintaxi d'objecte literal de JavaScript, pot ser utilitzat independentment de JavaScript.

Els fitxers per emmagatzemar dades amb format JSON cada vegada són més usats en la informàtica i també en l'àmbit de la bioinformàtica.

Figura 84. Exemple d'un document amb format JSON.
Font: elaboració pròpia.

En aquest format la forma de guardar les dades és semblant al model **clau/valor**, on la clau seria el nom del camp o atribut i a continuació tenim el valor.

Els fitxers JSON en realitat emmagatzemen una **col·lecció** de documents amb aquest format i cada document és la representació o la instància d'una entitat.

Si fem el símil amb la informació que emmagatzemem en una taula relacional, cada fila de la taula correspon a un document, i totes les files de la taula serien una col·lecció de documents.

JSON requereix usar cometes dobles per a les cadenes i els noms de propietats. Les cometes simples no són vàlides.

Una coma o dos punts mal ubicats poden produir que un arxiu JSON no funcioni. S'ha de ser curós per validar qualsevol dada que es vulgui utilitzar. És possible validar JSON utilitzant una aplicació com JSONLint.

Vegem com s'emmagatzema la informació que tenim guardada a la taula *genomas* en un fitxer JSON.

```
{espècie:"D. Melanogaster", nom:"Mosca de la fruita", descripció:"També denominada del vinagre"} {espècie:"H. Sapiens", nom:"Huma",descripcio:"Nostra pròpia especie"}
{espècie:"M. Musculus", nom:"Ratolí", descripció:"Altre organisme model"}
```

En aquest cas tenim 3 documents que corresponen a les 3 files de la taula *genomas*. Cada document comença amb el símbol `{`, i finalitza amb el símbol `}`

En cada document es repeteix el nom del camp i a continuació el seu valor, separats pel símbol :

Cada combinació **camp/valor** se separa per comes.

No es descriuen els tipus de dades, si el valor és una cadena de caràcters s'utilitzen les cometes dobles ” “, i si el valor és numèric no cal escriure'l entre cometes.

Una altra forma molt habitual de guardar la col·lecció de documents és dins d'un **array** amb els símbols **[ ]** i separant els diferents documents per comes. Es considera tota la col·lecció de documents com un sol objecte, ja que estan tots en un **array**.

```
[{espècie:"D. Melanogaster", nom:"Mosca de la
fruita", descripció:"També denominada del vinagre" }, {espècie:"H.
Sapiens", nom:"Huma", descripció:"la nostra pròpia espècie" },
{espècie:"M. Musculus", nombre:"Ratolí", descripció:"Altre organisme model" }]
```

Un camp pot guardar un **array** de valors:

```
{dies:["dilluns", "dijous", "dissabte"]}
```

Un camp també pot guardar un altre **document JSON.** També s'anomena *document incrustat*.

```
{direcció: {carrer: "València", número: 334, codi: 08012}}
```

Un camp pot guardar un **array** de **documents JSON**.

```
{amics:

[{nom: "Pedro", edat: 34, telèfon: 666737211},

{nom: "Soraya", edat: 31, telèfon: 666737212},

{nom: "Arnau", edat: 29, telèfon: 666737213}

]}
```

A partir de la versió 8 de MySQL és possible treballar amb documents JSON a les taules SQL amb el tipus de dades JSON.

Podem emmagatzemar tot un document JSON en un camp de la taula i realitzar consultes amb l'ordre `JSON_EXTRACT`, actualitzacions amb `JSON_REPLACE` i eliminacions amb l'ordre `JSON_REMOVE`.

2. Bases de dades NoSQL

2.3. El SGDB MongoDB

Tot i que també és possible treballar amb les dades emmagatzemades en format JSON amb MySQL i altres SGBD relacionals com PostgreSQL, la millor forma de gestionar les dades emmagatzemades als fitxers JSON és utilitzant una base de dades NoSQL com MongoDB.

El SGBD MongoDB es va publicar l'any 2009 i permet gestionar bases de dades orientades a documents. Guarda els documents en BSON, que no és més que una implementació binària del format JSON.

MongoDB és la més popular de les bases de dades NoSQL. Bàsicament, retorna dades a JSON i incorpora els conceptes de col·leccions (en lloc de taules) i documents (en lloc de files), el seu API o llenguatge de consulta es coneix popularment com a MQL (MongoDB Query Language).

Perquè tinguem més clares les diferències entre el model relacional i MongoDB podem consultar la taula 4:

Taula 4. Comparativa entre el model relacional i MongoDB.

Model relacional	MongoDB
Database	Database
Table	Collection
Register	Document o BSON document
Columna	Field
Index	Index
Table joins	Embedded documents and linking
Primary key	Primary key
Specify any unique column or column combination as primary key	the primary key is automatically set to the_id field
Aggregation	Aggregation pipeline

Font: elaboració pròpia.

Bàsicament, la diferència més substancial és que mentre que en un SGBD relacional com MySQL tenim bases de dades, taules i columnes de les taules, a MongoDB i SGBD NoSQL basats en documents tenim també bases de dades, però en comptes de taules amb columnes hi tenim col·leccions de documents, i en cada document hi ha els noms dels camps en comptes de les columnes de les taules.

2. Bases de dades NoSQL

2.4. Començar a treballar amb el SGBD MongoDB

A la màquina virtual proporcionada per la UOC tenim instal·lat un SGBD MongoDB.

Per connectar-nos al servidor de MongoDB obrim un terminal, hi escrivim `mongo` i ens sortirà el cursor >

Figura 85. Exemple de connexió a MongoDB per terminal.
Font: elaboració pròpia.

Ja estem connectats al servidor amb el client de la línia d'ordres i ja podrem escriure les ordres i sentències per interactuar amb el servidor de MongoDB.

Per visualitzar les bases de dades creades:

```
show dbs
```

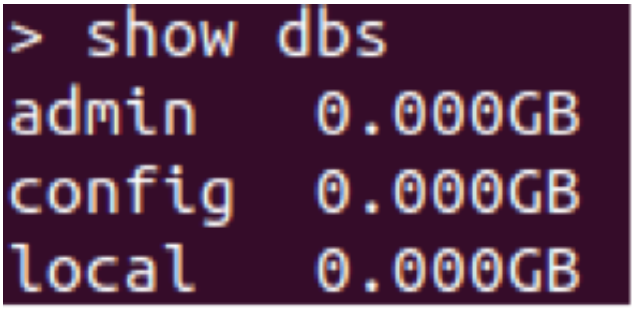


Figura 86. Mostrar les bases de dades creades.
Font: elaboració pròpia.

Com que encara no hem creat cap base de dades, només ens mostra les bases de dades del sistema.

Per crear una nova base de dades buida fem servir l'ordre `use`.

Vam crear la base de dades **uoc**.

```
use uoc
```

L'ordre *use* serveix tant per crear una base de dades nova com per connectar-se a una base de dades existent. L'ordre intenta connectar-se a una base de dades existent i si no existeix la crea.

Aquesta és una característica general de MongoDB, és molt flexible i dona pocs errors. En altres sistemes de gestió de bases de dades com MySQL, en intentar connectar amb una base de dades inexistent saltaria un error. A MongoDB no salta error i es crea la

nova base de dades. En realitat, no la crea, guarda espai per crear-hi col·leccions de documents. Si creem una nova base de dades amb l'ordre **use** i no creem col·leccions en ella, no queda emmagatzemada.

Veiem en aquesta seqüència d'ordres com consultem les bases de dades existents en el sistema amb `show dbs`, creem la base de dades **uoc** amb `use uoc`, i com quan tornem a consultar les bases de dades amb `show dbs` no apareix la base de dades **uoc**.

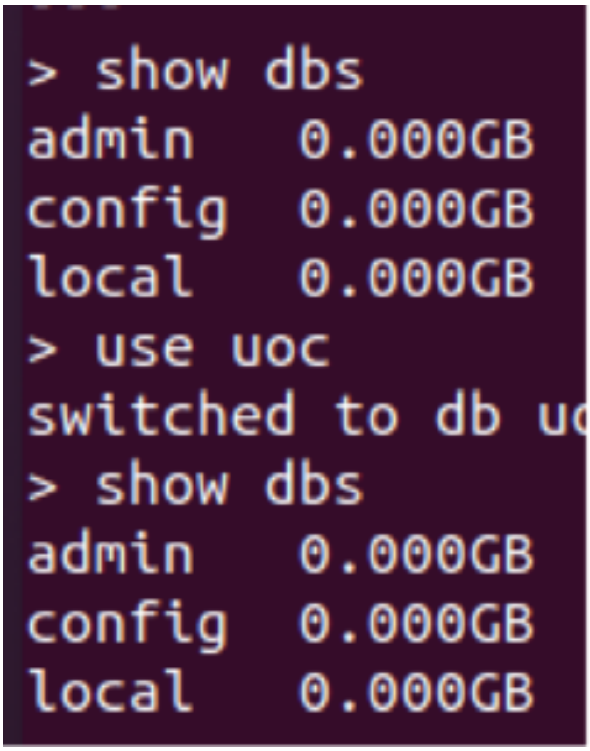


Figura 87. Utilitzar una base de dades.
Font: elaboració pròpia.

Perquè la nova base de dades quedi guardada al sistema cal que tingui col·leccions de documents.
Ara crearem una nova col·lecció buida a la base de dades **uoc** utilitzant la instrucció `createCollection()`.

```
> use uoc
switched to db uoc
> db.createCollection(chr1);
uncaught exception: ReferenceError: chr1 is not defined :
@(shell):1:1
> db.createCollection("chr1");
{ "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
uoc     0.000GB
>
```

Figura 88. Crear una col·lecció de documents.
Font: elaboració pròpia.

Tornem a connectar-nos a la base de dades **uoc** amb `use uoc` i creem la col·lecció de documents buida anomenada **chr1**.

Si us hi fixeu, primer executem la instrucció o funció `db.createCollection(chr1);` però dona error perquè el paràmetre que espera la funció no està entre cometes simples: **"chr1"**.

En escriure el paràmetre entre cometes simples `db.createCollection("chr1");` la nova col·lecció anomenada **chr1** es crea correctament a la base de dades **uoc**, que ja no està buida, i en executar `show dbs` ja ens mostra la base de dades **uoc**.

Analitzarem la instrucció `db.createCollection("chr1");` i ens servirà per entendre com funcionen les instriccions a MongoDB. Com que ja som a la base de dades **uoc**, amb la part de la instrucció `db.....` es refereix a la base de dades que estem fent servir, i a continuació la instrucció `createCollection()` que en realitat és una funció que pot fer servir paràmetres entre parèntesis.

Com ja veurem, la majoria d'instruccions són funcions que poden fer servir o no paràmetres. Si la instrucció no necessita paràmetres deixarem els parèntesis buits **()**.

Anem a veure algunes **ordres útils** de MongoDB:

- `db.help()` Mostra l'ayuda per als mètodes de la base de dades.
- `db.<collection>.help()` Mostra l'ajuda per als mètodes de la col·lecció. La <collection> pot ser el nom d'una col·lecció ja creada o encara no creada.
- `Show db` Mostra la llista de totes les bases de dades del sistema.
- `use db` Canvia la base de dades actual a la base de dades.
- `show collections` Mostra la llista de totes les col·leccions de la base de dades actual.
- `show users` Mostra la llista de tots els usuaris de la base de dades actual.
- `show databases` Nou a partir de la versió 2.4. Mostra la llista de totes les bases de dades disponibles.
- `db.<collection>.find()` Mostra tots els documents de la col·lecció <collection>.
- `db.<collection>.find().pretty()` Mostra tots els documents de la col·lecció <collection> en un format JSON més llegible.

Per veure les col·leccions creades a la base de dades fem servir la instrucció següent:

```
show collections
```

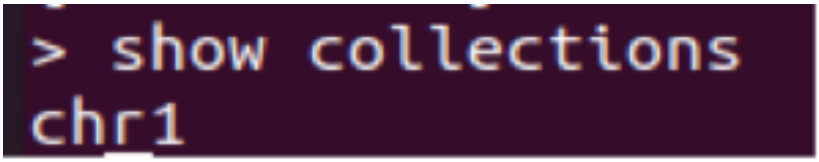


Figura 89. Mostrar les col·leccions de documents.
Font: elaboració pròpia.

2. Bases de dades NoSQL

2.5. Inserir documents

Com que tenim la col·lecció **chr1** buida hi inserirem documents:

Inserirem el següent document amb informació del gen **uc001aaa.3**, localitzat en el cromosoma 1:

```
{ "name": "uc001aaa.3", "chrom": "chr1", "strand": "+", "txStart":
11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12612,13220,", "exonEnds":
"12227,12721,14409,", "proteinID": "", "alignID": "uc001aaa.3"}
```

Per inserir aquest document a la col·lecció **chr1** utilitzarem la instrucció `db.chr1.insert();` on s'indica amb `db` la base de dades a la qual estem connectats, `db.chr1...`, la col·lecció on inserim el nou document, la col·lecció **chr1,** i la funció insert() que espera com a paràmetre el document que vam inserir:

```
db.chr1.insert({ "name": "uc001aaa.3", "chrom": "chr1", "strand": "+",
"txStart": 11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12612,13220,", "exonEnds":
"12227,12721,14409,", "proteinID": "", "alignID": "uc001aaa.3"});
```

Si la inserció s'ha realitzat correctament el sistema ens mostra el missatge següent:

```
WriteResult({ "nInserted" : 1 })
```

Ara hi inserirem dos documents més. Els documents JSON amb informació sobre els gens **uc010nxr.1** i **uc009vit.3**, també localitzats en el cromosoma 1.

```
db.chr1.insert({ "name": "uc010nxr.1", "chrom": "chr1", "strand": "+",
"txStart": 11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12645,13220,", "exonEnds":
"12227,12697,14409,", "proteinID": "", "alignID": "uc010nxr.1"});


db.chr1.insert({ "name": "uc009vit.3", "chrom": "chr1", "strand": "-",
"txStart": 14361, "txEnd": 19759, "cdsStart": 14361, "cdsEnd": 14361,
"exonCount": 9, "exonStarts": "14361,14969,15795,16606,16857,17232,17914,18267,18912,",
"exonEnds":
"14829,15038,15947,16765,17055,17742,18061,18366,19759,",
"proteinID": "", "alignID": "uc009vit.3"});
```

Ara eliminarem tota la col·lecció **chr1** de documents amb la instrucció `db.chr1.drop();`

I la tornem a crear:

```
db.createCollection("chr1");
```

per inserir els tres documents alhora amb la instrucció `insertMany();`

```
> db.chr1.drop();
true
> db.createCollection("chr1");
{ "ok" : 1 }
> show collections
chr1
```

Figura 90. Eliminar una col·lecció de documents.
Font: elaboració pròpia.

A la part de fitxers JSON hem vist que moltes vegades trobem un fitxer amb una col·lecció de documents que estan tots en un *array*.

Els tres documents amb la informació dels gens dels exemples anteriors els tenim ara dins d'un *array*:

```
[{ "name": "uc001aaa.3", "chrom": "chr1", "strand": "+", "txStart":
11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12612,13220,", "exonEnds":
"12227,12721,14409,", "proteinID": "", "alignID": "uc001aaa.3"},
{"name": "uc010nxr.1", "chrom": "chr1", "strand": "+", "txStart":
11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12645,13220,", "exonEnds":
"12227,12697,14409,", "proteinID": "", "alignID": "uc010nxr.1"},
{"name": "uc009vit.3", "chrom": "chr1", "strand": "-", "txStart":
14361, "txEnd": 19759, "cdsStart": 14361, "cdsEnd": 14361,
"exonCount": 9, "exonStarts":
"14361,14969,15795,16606,16857,17232,17914,18267,18912,",
"exonEnds": "14829,15038,15947,16765,17055,17742,18061,18366,19759,",
"proteinID": "", "alignID": "uc009vit.3"}]
```

I els inserirem en la col·lecció **chr1** amb la instrucció `insertMany();`

```
db.chr1.insertMany([{ "name": "uc001aaa.3", "chrom": "chr1",
"strand": "+", "txStart": 11873, "txEnd": 14409, "cdsStart": 11873,
"cdsEnd": 11873, "exonCount": 3, "exonStarts": "11873,12612,13220,",
"exonEnds": "12227,12721,14409,", "proteinID": "", "alignID":
"uc001aaa.3"}, {"name": "uc010nxr.1", "chrom": "chr1", "strand": "+",
"txStart": 11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12645,13220,", "exonEnds":
"12227,12697,14409,", "proteinID": "", "alignID": "uc010nxr.1"},
{"name": "uc009vit.3", "chrom": "chr1", "strand": "-", "txStart":
14361, "txEnd": 19759, "cdsStart": 14361, "cdsEnd": 14361,
"exonCount": 9, "exonStarts":
"14361,14969,15795,16606,16857,17232,17914,18267,18912,",
"exonEnds":
```

```
  "14829,15038,15947,16765,17055,17742,18061,18366,19759,",
  "proteinID": "", "alignID": "uc009vit.3"}]);
```

Ja tenim una altra vegada els tres documents a la col·lecció **chr1** de la base de dades **uoc,** i veurem algunes instruccions bàsiques per gestionar la informació amb MongoDB.

2. Bases de dades NoSQL

2.6. Buscar documents

Primer comptarem quants documents té la col·lecció chr1 amb la instrucció

```
db.chr1.find().count();
```

Aquesta instrucció conté dues funcions. La funció `find()` que és la funció que utilitzem per realitzar recerques en la col·lecció de documents. Seria la instrucció **SELECT** que hem vist a MySQL. Si no hi afegim paràmetres ens mostrarà tots els documents de la col·lecció, si hi afegim paràmetres, aquests seran els criteris de recerca o els filtres de la selecció. En aquest cas complementem la funció `find()` amb la funció `count()` per comptar els documents trobats. La funció `count()` no requereix paràmetres, però cal escriure els parèntesis igualment.

```
> db.chr1.find().count();
3
```

Figura 91. Comptar tots els documents d'una col·lecció.
Font: elaboració pròpia.

En executar la instrucció `db.chr1.find().count();` el sistema ens en retorna 3. La col·lecció **chr1** té 3 documents.

Utilitzarem la instrucció `db.chr1.find();` per veure tots els documents:

```
> db.chr1.find();
{ "_id" : ObjectId("64861afc32e0aa299185905c"), "name" : "uc001aaa
arts" : "11873,12612,13220,", "exonEnds" : "12227,12721,14409,", "
{ "_id" : ObjectId("64861df75d6ce7147d325b7e"), "name" : "uc010nxr
arts" : "11873,12645,13220,", "exonEnds" : "12227,12697,14409,", "
{ "_id" : ObjectId("64861e025d6ce7147d325b7f"), "name" : "uc009vit
arts" : "14361,14969,15795,16606,16857,17232,17914,18267,18912,",
>
```

Figura 92. Mostrar la informació de tots els documents d'una col·lecció.
Font: elaboració pròpia.

Per a cada document el sistema ens mostra el camp **_id** amb un valor generat aleatòriament per la funció del sistema `ObjectId();`

En el camp **_id** es guarda la clau primària del document. Aquesta clau primària és tan important per al sistema que si el document que inserim no té el camp **_id** el sistema el crea de forma automàtica. En els nostres exemples els tres documents inserits no tenen el camp **_id** i el sistema l'ha generat automàticament.

En un document hi pot haver molts camps i, tal com els mostra la funció `find();` , queden poc llegibles. Perquè el document es pugui llegir millor, la funció `find();` es pot complementar amb la funció `pretty();`

```
db.chr1.find().pretty();
```

Figura 93. Mostrar la informació de tots els documents d'una col·lecció amb un format més llegible.
Font: elaboració pròpia.

Com podem observar, la funció `pretty();` ens permet veure els valors de cada camp de cada document d'una forma molt més amigable.

Vegem ara alguns exemples d'utilització de la funció `find()` amb alguns paràmetres per filtrar els resultats.

Mostrarem només els gens de la nostra col·lecció que es troben en la cadena +:

```
db.chr1.find({"strand": "+"}).pretty();
```

Figura 94. Mostrar la informació dels gens de la cadena +.
Font: elaboració pròpia.

El paràmetre de la funció és la condició de selecció de la recerca `{"strand": "+"}`

Si no volem mostrar algun camp, com per exemple el camp `_id`, escrivim

```
db.chr1.find({"strand": "+"}, {"_id":0}).pretty();
```

```
> db.chr1.find({"strand": "+"}, {"_id":0}).pretty();
{
        "name" : "uc001aaa.3",
        "chrom" : "chr1",
        "strand" : "+",
        "txStart" : 11873,
        "txEnd" : 14409,
        "cdsStart" : 11873,
        "cdsEnd" : 11873,
        "exonCount" : 3,
        "exonStarts" : "11873,12612,13220,",
        "exonEnds" : "12227,12721,14409,",
        "proteinID" : "",
        "alignID" : "uc001aaa.3"
}
{
        "name" : "uc010nxr.1",
        "chrom" : "chr1",
        "strand" : "+",
        "txStart" : 11873,
        "txEnd" : 14409,
        "cdsStart" : 11873,
        "cdsEnd" : 11873,
        "exonCount" : 3,
        "exonStarts" : "11873,12645,13220,",
        "exonEnds" : "12227,12697,14409,",
        "proteinID" : "",
        "alignID" : "uc010nxr.1"
}
```

Figura 95. Mostrar la informació dels gens de la cadena + sense el camp `_id`.
Font: elaboració pròpia.

Si volem mostrar un sol camp dels documents seleccionats, com per exemple el camp **name**, escrivim

```
db.chr1.find({"strand": "+"},{"name":1, "_id":0 }).pretty();
```

```
> db.chr1.find({"strand": "+"},{"name":1, "_id":0 }).pretty();
{ "name" : "uc001aaa.3" }
{ "name" : "uc010nxr.1" }
```

Figura 96. Mostrar els gens de la cadena + sense mostrar el camp `_id` i mostrar el camp **name**.
Font: elaboració pròpia.

Les consultes amb la funció `find()` poden ser molt més elaborades, usant diversos camps com a condició de recerca, o també usant operadors especials com major que, menor que, etc.

Exemples d'operadors especials:

```
$gt, $gte, $lt, $lte, $ne, $in, $nin, $mod, $regex/$options,

$all, $size, $exists, $type, $not, $or, $nor, $elemMatch
```

Exemple de selecció de documents en què el camp **txStart** sigui major que 11873:

```
db.chr1.find({"txStart": {"$gt" : 11873 }}).pretty();
```

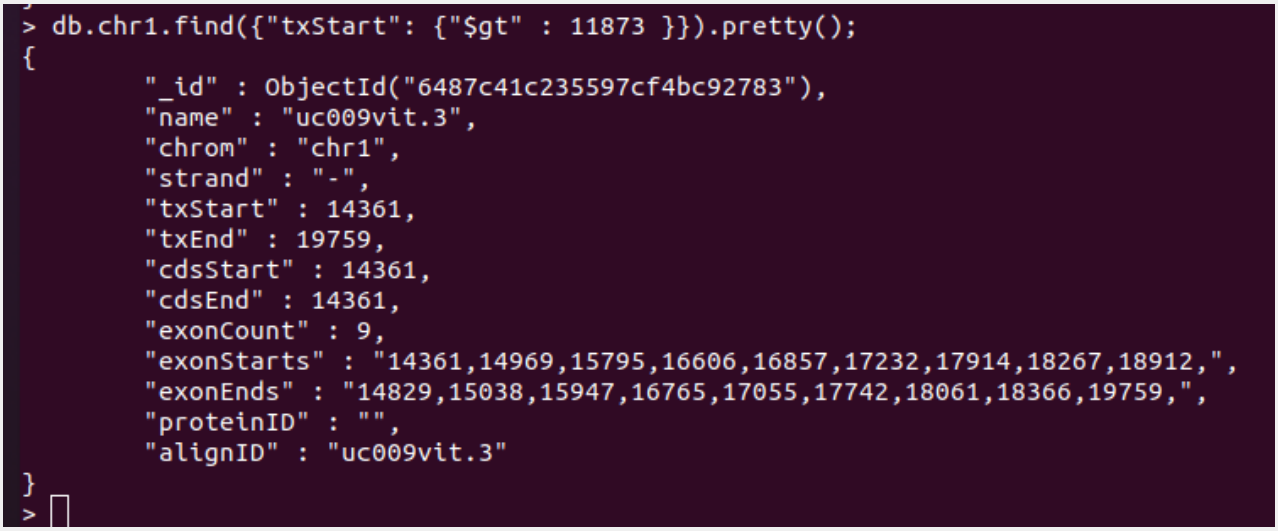


Figura 97. Mostrar els gens en què el camp txStar sigui més gran que 11873.
Font: elaboració pròpia.

O que compleixi dues condicions, que es trobin en la cadena + i que el camp **txStart** sigui major o igual a 11873.

```
db.chr1.find({"strand": "+", "txStart": {"$gte" : 11873 }}).pretty();
```

2. Bases de dades NoSQL

2.7. Modificar documents

Vegem ara com es pot actualitzar un document amb la funció `update()`.

Vam modificar la cadena del gen uc001aaa.3.

```
db.chr1.update({"name" : "uc001aaa.3"}, { "$set" : { "strand": "-" }});
```

I comprovem el canvi:

```
db.chr1.find({"name" : "uc001aaa.3"},{"name":1,"strand":1,"_id":0 }).pretty();
```

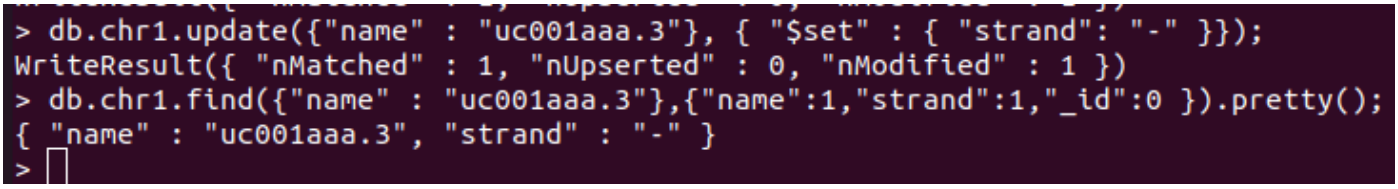


Figura 98. Modificar la cadena del gen uc001aaa.3.
Font: elaboració pròpia.

L'operador `$set` modifica el valor de camp si aquest existeix; si no existeix el camp, l'incorpora al document o documents que coincideixin amb la selecció.

L'operador `$unset` elimina el camp del document o documents que coincideixin amb la selecció.

També és possible afegir nous elements a un camp *array* amb l'operador `$push` i eliminar elements de l'*arrayt* amb els operadors `$pull`, `$pullAll`, `$pop`.

Els operadors de l'ordre update són els següents:

```
$set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $pop,

$addToSet, $rename, $bit, $ positional operator
```

2. Bases de dades NoSQL

2.8. Eliminar documents

Vegem ara com podem eliminar un document de la col·lecció amb la funció `remove()`.

Eliminem de la col·lecció el document amb el nom de gen **uc001aaa.3**

```
db.chr1.remove( { "name" : "uc001aaa.3"});
```

I comprovem si el document s'ha eliminat:

```
db.chr1.find().pretty();
```

Figura 99. Eliminar el gen uc001aaa.3.
Font: elaboració pròpia.

Ara eliminarem tota la col·lecció amb l'ordre `drop`

```
db.dropDatabase();
```

2. Bases de dades NoSQL

2.9. Importar fitxers JSON a MongoDB

Ara treballarem important un fitxer en format JSON que ens proporciona el navegador genòmic UCSC.

Descarreguem el fitxer JSON del següent enllaç:
https://api.genome.ucsc.edu/getData/track?genome=hg19;track=knownGene;chrom=chr1

Aquest fitxer conté la informació de tots els gens coneguts del cromosoma 1.

Baixem el fitxer i el guardem amb el nom hg19chr1.json

Per importar el fitxer a MongoDB obrim un nou terminal a la carpeta de la màquina virtual on hem guardat el fitxer i escrivim

```
mongoimport --db hg19 --collection chr1 --drop --file hg19chr1.json
```

Analitzarem aquesta instrucció:

L'ordre `mongoimport` s'executa **fora** del client (mongo) i té diverses opcions o paràmetres

- `--db` indica la base de dades. Si existeix la utilitza per crear la col·lecció de documents, si no existeix la crea.
- `--collection db` indica la col·lecció. Si existeix la utilitza per inserir-hi els documents del fitxer que importem, si no existeix la crea.
- `--drop elimina` els documents previs de la col·lecció, si existeix.
- `--file` indica el fitxer que importarem. Si els documents estiguessin dins d'un *array* hem d'afegir l'opció `--jsonArray`

Si ens connectem ara al servidor de MongoDB amb el client mongo i mirem les bases de dades amb `show dbs`, veiem com s'ha creat la base de dades **hg19.**

Si ens connectem a la base de dades **hg19** amb `use hg19` i veiem les col·leccions amb `show collections` podem veure la col·lecció **chr1** que hem creat i ja podem treballar amb ella.

2. Bases de dades NoSQL

2.10. Buscar en un *array* de documents

Si ens fixem en l'estructura del fitxer JSON importat, només conté un document JSON. I en un dels camps, l'anomenat **knownGene**, conté un *array* de documents JSON, i cada document de l'*array* conté informació dels gens del cromosoma 1: nom, cadena, cromosoma, etc.

Això ens obliga a conèixer com es treballa amb els continguts dels *arrays* si volem gestionar correctament aquesta informació.

Hem de fer servir el `dot.notation`.

Per exemple, mostrarem la informació del gen `uc009vis.3`.

Aquesta és la instrucció:

```
db.chr1.find({ "knownGene.name": "uc009vis.3"}, {"knownGene.$": 1}).pretty();
```

Figura 100. Mostrar la informació del gen uc009vis.3.
Font: elaboració pròpia.

Veiem com per referir-nos al nom del gen escrivim `"knownGene.name":` és a dir, el nom del camp del document que és un *array* de documents JSON, el `knownGene`, punt i a continuació el nom del camp dels documents que es troben *a* l'*array* `name`. Amb `knownGene.$": 1` indiquem que ens mostri tots els camps dels documents trobats a l'*array* `knownGene`.

El `dot.notation` també ens serveix per referir-nos a camps de documents embeguts.

2. Bases de dades NoSQL

2.11. Agregacions a MongoDB

Per treballar a fons amb els elements d'un *array* és millor utilitzar el `framework aggregation` que ens permet MongoDB, moltes opcions i realitzar tubs *pipelines* per gestionar la informació emmagatzemada.

Operacions d'agregacions

Les operacions d'agregació són eines de MQL que ens ajuden a processar documents i a retornar resultats calculats. Les operacions d'agregació s'utilitzen majoritàriament per:

- Agrupar valors de diversos documents.
- Processar i operar per al retorn de resultats.
- Analitzar canvis de dades al llarg del temps.

Tubs i transformacions

Per realitzar el processament de documents, MongoDB es basa en el patró de filtre de tubs, utilitzat comunament en arquitectures de programari. Aquest patró consta d'una o més etapes, on cada etapa realitza una operació amb les dades d'entrada, i la sortida o resultat l'entrega a la següent etapa per al seu processament.

Per aplicar aquest patró, MongoDB utilitza una sèrie d'operadors ja definits per poder processar documents.

Els operadors més utilitzats en tubs:

- Filtratge de documents amb criteris: `$match`.
- Ordre de documents: `$sort`.
- Selecció de camps en específic: `$project`.
- Agrupació de documents: `$group`.
- Treure els elements d'un *array*: `$unwind`.

Caldria un altre curs per aprofundir en totes les opcions que ens ofereix MongoDB.

Vegem un exemple d'una operació d'agregació. La funció `aggregate()`.

Entre moltes altres opcions, `aggregate` ens permet treure els elements d'un *array* amb l'operador `$unwind`, fer agrupacions amb `$group` i comptar amb `$sum`.

Anem a comptar quants gens conté cada fil de l'ADN del cromosoma 1.

```
db.getCollection('chr1').aggregate([{$unwind:"$knownGene"},{$group:{_id:"$knownGene.st
```

```
> db.getCollection('chr1').aggregate([{$unwind:"$knownGene"},{$group:{_id:"$knownGene.strand",cantidad:{$sum:1}}}]);
{ "_id" : "-", "cantidad" : 3894 }
{ "_id" : "+", "cantidad" : 4073 }
>
>
>
```

Figura 101. Mostrar quants gens conté cada fil de l'ADN del cromosoma 1.
Font: elaboració pròpia.

Resum

A nivell genòmic, habitualment treballem amb milers de registres durant una anàlisi bioinformàtica convencional. Malgrat que les eines d'anàlisi de fitxers de text basades en l'ús d'ordres del terminal de GNU/Linux ens permeten accedir fàcilment a les nostres dades, els sistemes de gestió de bases de dades com MySQL o MongoDB resulten el mitjà ideal per gestionar grans volums de dades de forma eficient i ràpida.

En aquest mòdul us hem mostrat un gran ventall d'ordres basades en el llenguatge SQL per consultar les taules de les nostres bases de dades relacionals i algunes ordres específiques de MongoDB per gestionar la informació d'una base de dades no relacional NoSQL orientada a documents.

Juntament amb aquest inventari d'instruccions, hem après també a modelar correctament les entitats del problema real que desitgem aproximar mitjançant estratègies bioinformàtiques.

Activitats

1. Portem a la pràctica sobre el teu propi SGBD MySQL els exemples mostrats durant aquest mòdul. Intentem enriquir cada base de dades amb noves taules que modelin entitats o relacions que no apareixien originalment en els casos estudiats. Ampliem el conjunt d'atributs de les taules per descriure amb més precisió les instàncies reals que es mostren al llarg del text.

2. Dissenyem una base de dades per emmagatzemar informació sobre un entorn complex natural que ens agradaria modelar: un ecosistema, el cos humà, la cèl·lula o l'univers. Reflexionem sobre les entitats, els seus atributs i les relacions entre aquestes, que són necessàries en cada cas per especificar aquest model. Repetim l'exercici amb un entorn generat per l'ésser humà (per exemple, un automòbil).

3. Implementem una base de dades a MySQL que permeti portar el registre de totes les activitats d'un servidor web genèric que rep peticions per executar una sèrie de serveis: pàgines més visitades, tipus de serveis sol·licitat, volum de dades, temps de resposta, usuaris, nombre de pàgines consultades per client, adreça IP i nom de la màquina, país de procedència, etc.

4. Ampliem el nostre catàleg de gens pensant en futures ampliacions. Podem afegir-hi una entitat PROTEÏNES per emmagatzemar informació estructural o sobre dominis funcionals. També seria interessant incorporar-hi una entitat CROMOSOMES, relacionada amb la taula GENOMAS, on guardar altres característiques, com el nombre de bases o el contingut en G + C.

5. Importem alguns fitxers JSON que ens proporciona UCSC Genome Browser https://genome.ucsc.edu/goldenPath/help/api.html#REST a MongoDB creant per a cadascun d'ells una base de dades nova i una nova col·lecció. Realitzem alguna consulta simple a les col·leccions i alguna consulta en un document situat en un camp *array* de documents.

Exercicis d'autoavaluació

1. Definiu el model entitat-relació en el disseny de bases de dades.
2. Descriviu les diferències entre bases de dades i gestors de base de dades.
3. Descriviu què és una taula en el model relacional.
4. Què són les claus primàries?
5. Què són les claus foranes?
6. Definiu què és una relació 1:N.
7. Recordeu quin símbol heu d'emprar sempre al final de cada ordre SQL.
8. Esmenteu cinc ordres essencials del llenguatge SQL.
9. Descriviu l'ordre de SQL per explorar el contingut de les taules.
10. Quin tipus d'operacions es realitzen sobre registres agrupats?
11. Diferencieu entre les ordres DISTINCT i LIMIT.
12. Enumereu els tres tipus d'unions a realitzar entre dues taules.
13. Definiu la utilitat d'una subconsulta.
14. Quines dues ordres són útils per esborrar taules? En què es diferencien?
15. Indiqueu la instrucció de SQL per executar un fitxer d'ordres.
16. Indiqueu la instrucció de SQL per carregar un fitxer de dades.
17. Quin format compleixen els fitxers de text per emprar-se d'aquesta manera?
18. Indiqueu el programa de GNU/Linux que copia una base de dades en un fitxer.
19. Com es pot iniciar el client de MySQL des de la línia d'ordres?
20. Com es concedeixen autoritzacions a un usuari sobre una base de dades MySQL?
21. Com es mostra un document amb un format més llegible?
22. Com es crea una nova col·lecció de documents a MongoDB?
23. Com es compten els documents d'una col·lecció a MongoDB?
24. Què és la sintaxi dot.notation a MongoDB?

Solucionari

1. El model entitat-relació es basa en l'ús de dos tipus d'elements per modelar un entorn real. Les entitats modelen cada classe d'elements de la realitat. Les relacions modelen les associacions entre les instàncies de cada entitat en aquest entorn.

2. Una base de dades és un conjunt d'informacions organitzades per fomentar un accés eficient a aquestes; un gestor de base de dades és precisament el programa que implementa el manteniment permanent de la base de dades, i que, a més, ofereix mecanismes per accedir-hi.

3. Una taula és una estructura que agrupa una col·lecció d'elements (instàncies) de la mateixa classe. Generalment, una taula modela una entitat juntament amb els seus atributs, tot i que pot ser necessària també per implementar algunes relacions entre entitats dins de la base de dades.

4. La clau primària d'una taula és l'atribut que identifica de forma unívoca cada instància o element en el seu interior. Per això, el valor d'aquest atribut no es pot repetir entre instàncies diferents.

5. La clau forana d'una taula és la clau primària d'una altra taula, assegurant la integritat del model, atès que cada instància a la primera taula haurà d'existir també a la segona.

6. Una relació 1:N entre dues entitats indica que cada instància de la primera entitat pot associar-se amb *N* instància a la segona taula. Pot implementar-se en la segona taula directament amb un atribut que prengui per valor algun dels valors del rang correcte per a la primera.

7. Qualsevol ordre que introduïm en l'intèrpret de MySQL ha d'acabar obligatòriament amb el símbol ;, per ser executada correctament.

8. Per exemple: `CREATE DATABASE`, `CREATE TABLE`, `SELECT`, `LOAD DATA` i `GRANT`.

9. L'ordre `SELECT... FROM... WHERE` permet realitzar una consulta sobre els registres de les taules que compleixen certes condicions.

10. Quan s'agrupen instàncies amb `GROUP BY`, és possible calcular mitjanes aritmètiques, mínims, màxims o comptes de totals (`AVG`, `MIN`, `MAX`, `COUNT`).

11. La clàusula DISTINCT serveix per eliminar valors repetits. La clàusula LIMIT és útil per mostrar només les primeres instàncies d'una taula.

12. A l'hora de comparar dues taules amb l'ordre `JOIN` emprant un atribut en comú, podem buscar les parelles de valors presents en ambdues taules o aquelles que només apareixen en una d'elles (`LEFT` o `RIGHT`).

13. Una subconsulta permet generar un grup de resultats en forma de taula temporal. Aquesta taula auxiliar podrà ser interrogada, al seu torn, per la consulta principal que alberga la subconsulta en el seu interior.

14. L'ordre `DROP TABLE` elimina la taula completament (definició i contingut). L'ordre `DELETE`, en canvi, elimina únicament determinats registres d'acord amb una condició.

15. La instrucció `SOURCE`.

16. La instrucció `LOAD DATA`.

17. El fitxer de text ha d'estar tabulat en columnes, que corresponen als atributs de les taules.

18. El programa **mysqldump** realitza el bolcat de la base de dades.

19. L'ordre seria: `mysql -o usuari -p`

20. Amb la instrucció `GRANT`

21. Amb l'ordre o funció `pretty()`

22. Amb la funció `db.createCollection("nomColecció");`

23. Amb la funció `count();`

24. És la forma d'identificar un camp situat en un document situat en camp *array* de documents de la forma
    `nom_camp_array.nom_ camp_ document`

25. És un document situat en un camp d'un document JSON.

Bibliografia

Conrad Bessant, Ian Shadforth i Darren Oakley (2009). *Building Bioinformatics Solutions: with Perl, R and MySQL*. Oxford University Press. ISBN: 0199230234.

Edgar F. Codd (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13, p. 377-387.

MySQL AB (2006). *Manual de referencia de MySQL 8.0*. http://dev.mysql.com

Paul DuBois (2008). *MySQL* (4th Edition). Addison-Wesley Professional. ISBN: 0672329387.

Peter Pin-Shan Chen (1976). The Entity-relationship Model–Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1, p. 9-36.

Steven Haddock i Casey Dunn (2011). *Practical Computing for Biologists*. Sinauer Associates. ISBN: 978-0-87893-391-4.

Vince Buffalo (2015). *Bioinformatics Data Skills.* O'Reilly Media. ISBN: 978-1-449-36737-4.

Webgrafia

Anaya Multimedia. Guía Práctica MySQL 5.1/Capítulo 11: Procedimientos almacenados. https://enreas.fandom.com/wiki/Gu%C3%ADa_Pr%C3%A1ctica_MySQL_5.1/Cap%C3%ADtulo_11:_Procedimientos_almacenados

COMANDOS MYSQL. Convertir consulta MySQL a JSON. https://thedevelopmentstages.com/convertir-consulta-mysql-a-json/

DelftStack. Cómo declarar y usar las variables en MySQL. https://www.delftstack.com/es/howto/mysql/mysql-declare-variable/

Guebs. MySQL 5.0 Reference Manual. Traducción. https://manuales.guebs.com/mysql-5.0/

Guebs. MySQL 5.0 Reference Manual. 19.2.1. CREATE PROCEDURE y CREATE FUNCTION. https://manuales.guebs.com/mysql-5.0/stored-procedures.html#create-procedure

JSONLint – The JSON Validator. https://jsonlint.com/

MongoDB. https://www.mongodb.com/

MongoDB. MongoDB CRUD Operations. https://www.mongodb.com/docs/manual/crud/

MongoDB Manual. Aggregation Operations. https://www.mongodb.com/docs/manual/aggregation/

MySQL. https://dev.mysql.com/

MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/

MySQL 8.0 Reference Manual. 13.1.20.5 FOREIGN KEY Constraints. https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html

MySQL 8.0 Reference Manual. 13.1.17 CREATE PROCEDURE and CREATE FUNCTION Statements. https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

MySQLTUTORIAL.MySQL Stored Procedures. https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

MySQL 8.0 Reference Manual. 25.3.1 Trigger Syntax and Examples. https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

MySQL 8.0 Reference Manual. 11.5 The JSON Data Type. https://dev.mysql.com/doc/refman/8.0/en/json.html

Universidad de Sevilla. Introducción a PL/SQL en MySQL. https://dam.org.es/introduccion-a-pl-sql-en-mysql/