Herramientas informáticas para la bioinformática

Autores: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve
PID_00298303
Primera edición: septiembre 2023

Módulos didácticos

Introducción

En los últimos años hemos asistido a una revolución tecnológica sin parangón. Con la aparición de las primeras computadoras, la incesante miniaturización de sus componentes y la conexión global de estas mediante una red universal, los seres humanos hemos construido una visión radicalmente novedosa del mundo que nos rodea. Actualmente resulta difícil concebir la investigación en cualquier área de conocimiento sin el concurso de la informática. La biología molecular no es una excepción. Quizás no somos conscientes de la envergadura del salto conceptual experimentado en este campo. A raíz del descubrimiento a mitad del siglo pasado de la estructura de la molécula de ADN y de la obtención de su secuencia en los inicios del siglo actual, empezamos a estar en condiciones de abordar sinceramente el problema de la constitución de la vida misma. Ahora gozamos de la inigualable ventaja de conocer, para numerosas especies, el catálogo de genes y proteínas que gobiernan la mayoría de las respuestas celulares durante su desarrollo o en respuesta a las condiciones cambiantes de nuestro entorno.

Mientras la mejora de la secuenciación masiva permite reconstruir la cartografía de los genomas, la potenciación de las telecomunicaciones ofrece un entorno ideal para poner estos datos a disposición de la comunidad científica. No obstante, toda esta fantástica amalgama de informaciones debe ser gestionada de forma eficiente. Evidentemente, la bioinformática desempeña un papel crucial en la realización rutinaria de este tipo de análisis en cualquier laboratorio de genética. Por tanto, resulta fundamental formar a nuevo personal investigador en las técnicas bioinformáticas básicas, con el objetivo de que adquieran el vocabulario específico de este campo, se agilicen sus futuras colaboraciones científicas con especialistas de dicha disciplina y comprendan el alcance de esta tecnología para integrarla dentro de sus propios proyectos de investigación.

Este módulo se centra en las aplicaciones computacionales más utilizadas por los bioinformáticos para procesar información genómica. La familia de sistemas operativos Gnu/Linux es la plataforma habitual en este tipo de laboratorios. En primer lugar, se presentarán conceptos básicos relacionados con los sistemas operativos. A continuación, se abordará el manejo del terminal de Gnu/Linux, una herramienta de trabajo fundamental para los bioinformáticos. Se aprenderán los comandos básicos y cómo combinarlos para generar comandos más potentes, creando así protocolos completos de trabajo. Por último, se explicará cómo obtener fácilmente copias de grandes conjuntos de datos biológicos de referencia para poder analizarlos localmente en nuestros ordenadores de manera sencilla. En resumen, este módulo os permitirá dominar los elementos básicos necesarios para integraros sin dificultades en cualquier entorno de investigación bioinformática.

Objetivos

Con el programa de contenidos de estos materiales, una vez finalizada la etapa de aprendizaje, el estudiante será capaz de realizar la mayor parte de las actividades básicas habituales en un laboratorio bioinformático de investigación:

1. Dominar el manejo elemental de los entornos de trabajo Linux.
2. Utilizar el terminal de comandos para el análisis de datos biológicos.
3. Gestionar la información de los genomas con bases de datos y MySQL.
4. Publicar resultados en la red mediante páginas web.
5. Elaborar servidores de datos para aplicaciones bioinformáticas en Internet.

Bibliografía

Alfred V. Aho, Monica S. Lam, Ravi Sethi y 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 y 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 y Darren Oakley (2009). *Building Bioinformatics Solutions: with Perl, R and MySQL*. Oxford University Press. ISBN: 0199230234.

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

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

Kevin Tatroe, Peter MacIntyre y 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 y 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.

Introducción a los entornos de trabajo Gnu/Linux

Autores: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve
PID_00298303
Primera edición: septiembre 2023

Introducción

La secuencia del genoma de múltiples especies está a disposición de cualquier persona que posea un ordenador con conexión a Internet, gracias al enorme esfuerzo de la comunidad científica cohesionada en varios consorcios internacionales, públicos y privados. Los grandes proyectos de secuenciación han producido una voluminosa cantidad de información genómica que debe ser gestionada de forma extremadamente eficiente y precisa para su posterior análisis. Solo la secuencia de nucleótidos del genoma humano, que ocupa varios *gigabytes*, contiene decenas de miles de genes y reguladores transcripcionales. De hecho, la reciente aparición de nuevos métodos de secuenciación masiva para cartografiar la localización de distintos elementos funcionales a lo largo de las secuencias genómicas en cualquier contexto celular va a multiplicar exponencialmente los requisitos actuales de tiempo de cálculo y espacio de almacenamiento.

El análisis exhaustivo de toda esta información para extraer nuevo conocimiento no puede realizarse manualmente. La gestión informática resulta esencial, por tanto, para manipular con garantías este volumen de datos. La bioinformática proporciona, en este sentido, el entorno ideal de trabajo para el biólogo molecular. En un entorno bioinformático de investigación, los genomas y las aplicaciones están almacenados localmente, evitando problemas de conexión y tráfico de la red. Estas estaciones de trabajo son las herramientas esenciales del investigador para efectuar análisis bioinformáticos de cualquier tipo de secuencia biológica.

Este módulo profundiza sobre la mayoría de las aplicaciones computacionales utilizadas habitualmente por un bioinformático para procesar información genómica. La familia de sistemas operativos GNU/Linux es la plataforma habitual de trabajo en esta clase de laboratorios. En primer lugar, realizaremos una introducción general a los conceptos básicos relacionados con los sistemas operativos. Posteriormente, enfocaremos nuestro interés en el manejo del terminal de GNU/Linux, la herramienta de trabajo usual para un bioinformático. Aprenderemos los comandos básicos, y cómo pueden combinarse estos para generar comandos aún más potentes que conformen protocolos completos de trabajo. Finalmente, veremos que podemos obtener fácilmente una copia de los grandes conjuntos de datos biológicos de referencia para poder analizarlos localmente en nuestro ordenador con suma facilidad. En resumen, dominaréis los elementos básicos de trabajo para integraros fácilmente dentro de cualquier entorno de investigación bioinformático.

Objetivos

Con el programa de contenidos de este módulo, una vez finalizada la etapa de aprendizaje, debéis lograr los siguientes objetivos:

1. Conocer las funciones del sistema operativo.
2. Reconocer la familia de sistemas operativos GNU/Linux.
3. Distinguir entre procesos y programas.
4. Identificar la jerarquía del sistema de ficheros.
5. Trabajar con máquinas virtuales multiplataforma.
6. Utilizar el terminal para el análisis de datos bioinformáticos.
7. Combinar series de comandos para crear protocolos más complejos.
8. Obtener una copia de los entornos bioinformáticos reales.
9. Integrar los comandos del terminal con la información biológica.

1. Introducción a los entornos de trabajo UNIX

1.1. Arquitectura de un ordenador

Un ordenador es una máquina electrónica que es capaz de procesar y almacenar información. John Eckert y John Mauchly presentaron en Filadelfia el 15 de febrero de 1946 el Electronic Numerical Integrator And Calculator (ENIAC), el que se considera el primer ordenador de propósito general de la historia. Esta máquina de treinta toneladas, que ocupaba un piso completo de la Escuela Moore de Ingeniería Eléctrica (Universidad de Pensilvania), resolvía en una sola hora el mismo número de cálculos de trayectorias balísticas que doscientos especialistas en dos meses. Desde ese primer modelo hasta la aparición de los microprocesadores de última generación (integrados en teléfonos móviles y electrodomésticos), el progreso tecnológico ha sido increíblemente veloz.

Los ordenadores para poder manipular datos, hacer cálculos o ejecutar tareas preestablecidas requieren de unos componentes básicos listados a continuación:

- La unidad central de procesamiento (en inglés, Central Processing Unit o CPU). El «cerebro» del ordenador, el cual ejecuta los procesos y cálculos.

- La memoria central (en inglés, Random Access Memory o RAM). La memoria del ordenador es un dispositivo de almacenamiento temporal de información continuamente modificada por las operaciones realizadas en la CPU. Programas y datos deben previamente cargarse en la memoria, siendo entonces transferidos al procesador para su ejecución.

- Disco duro. Almacenamiento estable de la información. Actualmente, hay dos tipos de discos: *hard drive* (HDD) o *solid-state drive* (SSD).

- Dispositivos de entrada y salida (I/O). Los dispositivos de entrada permiten la introducción de información o comandos al ordenador (teclado, ratón, etc.), mientras que los dispositivos de salida proporcionan los resultados al usuario (pantalla, impresora, etc.).

- Los periféricos. Son dispositivos que mejoran la funcionalidad de los ordenadores (discos duros, lápices de memoria o tarjetas de red).

Para poder hacer que los ordenadores ejecuten las tareas que les encomendamos, necesitamos poder crear programas (software) con las instrucciones exactas a ejecutar. Los ordenadores no son capaces de procesar lenguaje natural como nosotros, su lenguaje es binario (Verdadero o Falso, Encendido o Apagado, 0 o 1). Esto es debido a la arquitectura de los ordenadores, que está formada por miles o millones de transistores que individualmente se pueden encender o apagar. Los transistores pueden estar en estado activo si una pequeña cantidad de corriente pasa a través suyo (1) o, si no hay corriente, el transistor está en estado 0. A partir de estos dos estados se puede generar un lenguaje completamente nuevo.

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

Para poder comunicar los comandos que queremos ejecutar en los ordenadores utilizaremos los lenguajes de programación, que transformarán nuestras instrucciones en código binario para poderlas procesar.

1. Introducción a los entornos de trabajo UNIX

1.2. Funciones de un sistema operativo

El sistema operativo (SO) es el programa que desempeña un rol de intermediario entre el usuario y la máquina, y que dota a esta de una interfaz de funciones elementales para su gestión. De este modo, el usuario de la máquina consigue extraer un rendimiento superior, despreocupándose de su complejidad técnica.

Las ventajas de trabajar con un sistema operativo son diversas:

- Proporciona un entorno de trabajo más amigable para su utilización.
- Utiliza eficientemente los recursos del ordenador.
- Permite un desarrollo y testeo de nuevas funciones sin interferir con los servicios ya existentes.
- Proporciona el mayor número de tareas por unidad de tiempo.

Entre los recursos que el SO gestiona de forma transparente encontramos:

- El procesador.
- La memoria y los dispositivos de almacén secundarios.
- Dispositivos de entrada y salida de datos.
- El sistema de archivos y directorios.
- La conexión a la red y con otras máquinas.
- La seguridad y privacidad de los datos.
- La información interna sobre el sistema.

Ejemplos de SO:

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

1. Introducción a los entornos de trabajo UNIX

1.3. La familia de sistemas operativos UNIX

1.3.1. Introducción

En 1969, Ken Thompson y Dennis Ritchie desarrollaron en lenguaje ensamblador un pequeño SO denominado *UNICS* (en inglés, UNiplexed Information and Computing System), suficiente para ser ejecutado en un miniordenador DEC PDP-7. Unos años antes, Ken Thompson había formado parte de un gran proyecto coordinado entre el Instituto Tecnológico de Massachusetts (MIT), los Laboratorios Bell de AT&T y General Electric para producir otro sistema que funcionaba sobre una gran computadora GE-645. Este último recibió el nombre de MULTICS (en inglés, MULTiplexed Information and Computing System). Pese a sus múltiples innovaciones, el proyecto fue abandonado por su pobre rendimiento. UNICS, en contraposición a MULTICS, fue concebido como un sistema ligero orientado a gobernar miniordenadores.

A medida que el proyecto inicial demostraba su gran potencial, surgieron más posibilidades de desarrollo. En 1970, con el apoyo económico de los Laboratorios Bell (AT&T), los autores finalizaron una primera versión estable, que incluía herramientas para editar texto, capaz de funcionar en una minicomputadora PDP-11/20. En 1972, los mismos autores reescribieron el código de UNIX en el lenguaje de alto nivel C, permitiendo la portabilidad de todo el sistema a cualquier plataforma. Al aumentar la comprensión del código, el propietario de UNIX distribuyó licencias de desarrollo a varias universidades y compañías. En particular, el departamento de Computación de la Universidad de California, con sede en Berkley, publicó su propia versión de UNIX, denominada *Berkeley Software Distribution* (BSD), todavía con amplia difusión actualmente. A finales de los años setenta, gracias a la distribución mediante licencia del código original, el número de variantes de UNIX comenzó a multiplicarse exponencialmente. La compañía AT&T, propietaria del sistema UNIX original, lanzó en 1983 la distribución UNIX System V, una versión estable que combinaba las mejoras contenidas en cada variante aparecida anteriormente.

1. Introducción a los entornos de trabajo UNIX

1.3. La familia de sistemas operativos UNIX

1.3.2. Otras versiones de UNIX

Diferentes compañías han desarrollado su propia distribución comercial de UNIX. Solaris fue producido por Sun, AIX por IBM, HP-UX por Hewlett-Packard o Mac OS-X por Apple. Incluso Microsoft trabajó en su propia distribución, denominada *Xenix*.

La década de los ochenta presenció el gran éxito de los ordenadores personales de IBM (en inglés, Personal Computer o PC), gestionados por el sistema operativo DOS (en inglés, Disk Operating System). Precisamente, un estudiante finlandés de informática llamado Linus Torvalds desarrolló en 1991 el núcleo de un sistema operativo basado en UNIX para gestionar microprocesadores Intel x86 (el corazón de los primeros PC de IBM). Esta variante, en conjunción con el software abierto creado por el proyecto GNU (por ejemplo, gcc o emacs), constituyó el embrión de la familia de sistemas operativos Linux. Aunque existen en funcionamiento múltiples proyectos de desarrollo con base en UNIX (especialmente para la versión BSD), Linux ha logrado una amplia difusión entre la comunidad informática debido a su libre distribución como software abierto. Gracias a las mejoras aportadas por miles de programadores, Linux es uno de los SO más extendidos actualmente.

1. Introducción a los entornos de trabajo UNIX

1.3. La familia de sistemas operativos UNIX

1.3.3. El proyecto GNU

GNU (en inglés, GNU's not Unix!) propugna un SO compuesto por piezas de software libre. Tenéis más información en la página web http://www.gnu.org.

Debido a su naturaleza abierta, no existe en realidad una única versión de Linux. Desde su nacimiento han surgido numerosas variantes (denominadas *distribuciones* o *distros*) que comparten el núcleo del sistema y un conjunto de bibliotecas y programas comunes surgidos a partir del proyecto GNU. Cada distribución se diferencia del resto por el tipo de aplicaciones que incorpora, orientándose a usuarios domésticos, empresas o grandes servidores. Existen distribuciones soportadas comercialmente por compañías que proporcionan asistencia durante la instalación y actualización del sistema. La tabla 1 muestra algunas de las variantes más conocidas que existen.

Tabla 1. Distribuciones Linux de carácter general más populares

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

Fuente: elaboración propia.

1. Introducción a los entornos de trabajo UNIX

1.4. Programas y procesos

Los ordenadores ejecutan programas que son listados de instrucciones que indican cómo procesar un conjunto de datos.

Podemos dividir los lenguajes de programación entre:

- Bajo nivel: ensambladores y de máquina. Cercanos al código binario.
- Alto nivel: fáciles de leer y entender (por ejemplo, C, PHP, Python o Java).

Así mismo podemos diferenciar los lenguajes de alto nivel compilados o interpretados.

Un programa escrito con un lenguaje de programación compilado, para poder ejecutarse, necesita ser antes procesado por un compilador adecuado, y traducido a lenguaje máquina (binario). Los compiladores son programas encargados de realizar el análisis léxico, sintáctico y semántico del código. Una vez superada esa etapa de verificación, el compilador genera un fichero objeto que debe ser enlazado con varias librerías de funciones del sistema para generar un fichero ejecutable binario. El usuario puede ejecutar este archivo cuando sea preciso. La depuración de los programas compilados es costosa, y es rentable solo en los casos en que el rendimiento óptimo de estos, en términos de tiempo de ejecución y espacio de memoria, es capital.

Para efectuar tareas más sencillas es posible diseñar prototipos (en inglés, *scripts*) empleando lenguajes orientados a la producción rápida de programas, como Perl o Python, lenguajes interpretados. Estos lenguajes de *scripting* poseen un juego de instrucciones específicamente diseñado para facilitar la adquisición y tratamiento de ficheros de texto. Sus intérpretes procesan los programas instrucción a instrucción, saltándose de ese modo la creación de un fichero binario. A cambio, su rendimiento, en comparación con los ficheros ejecutables, es menor.

Una vez el usuario decide ejecutar un programa, el SO debe crear una entidad lógica asociada a este código a la que dotar de recursos suficientes para desarrollar su actividad (procesador, memoria y acceso a dispositivos). Esta metodología permite ejecutar de forma concurrente varias instancias de la misma aplicación sin mayor inconveniente que los propios de la compartición de algunos recursos (fácilmente subsanables dentro del programa, utilizando nombres únicos para los ficheros y otros dispositivos).

Dado que solo un proceso puede estar simultáneamente en posesión de la CPU, debe realizarse una planificación óptima para decidir en cada momento a qué proceso le corresponde su uso. Pese a que la compartición del procesador parece una seria limitación, es en realidad una gran ventaja, pues un proceso gasta una fracción importante de su tiempo esperando para acceder a otros dispositivos más lentos. Por tanto, solapando el uso de la CPU con las esperas de los procesos, se consigue simular un trabajo en paralelo, cuando realmente solo existe un procesador.

Las estaciones de trabajo actuales están dotadas de multiprocesadores, esto es, nodos con dos, cuatro o más procesadores dentro de la misma máquina. Gracias a esta ampliación de los recursos disponibles, el SO, mediante las librerías de diseño de programas apropiadas, puede hacer trabajar sus programas en paralelo, permitiendo que determinados fragmentos de estos trabajen independientemente sobre distintos conjuntos de datos en diferentes procesadores. Cada unidad de código ejecutable en paralelo dentro del proceso recibe el nombre de *hilo de ejecución* (en inglés, *thread*). En un sistema con una única CPU, el SO también es capaz de planificar varios *threads* para simular paralelismo, siempre que la carga de trabajo de la máquina no sea excesiva.

1. Introducción a los entornos de trabajo UNIX

1.5. El sistema de ficheros

La memoria de un ordenador almacena temporalmente tanto los programas como sus propios datos durante su ejecución en el procesador. Para evitar la pérdida de información cuando cesa el suministro del fluido eléctrico en el momento del apagado de nuestro ordenador, mantenemos siempre una copia de toda la información en dispositivos diseñados para tal efecto, como discos duros, CD, DVD o lápices de memoria. Esta clase de memoria secundaria comparte un método de organización común denominado *sistema de ficheros* (*filesystem*). El tipo de estructuración de cada volumen puede establecerse en el momento de darle formato. Por regla general cada sistema operativo posee una predisposición hacia un determinado formato, tolerando no obstante la compatibilidad con otros sistemas de almacenamiento. En la tabla 2, encontraremos algunos ejemplos de tipos de *filesystem* por SO.

Tabla 2. Tipos de formatos de sistemas de ficheros.

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

Fuente: elaboración propia.

El *filesystem* indexa toda la información en un volumen de almacenamiento, incluyendo el tamaño del archivo, los atributos, la localización y la jerarquía en el directorio. El *filesystem* también especifica la ruta al archivo mediante la estructura de directorios.

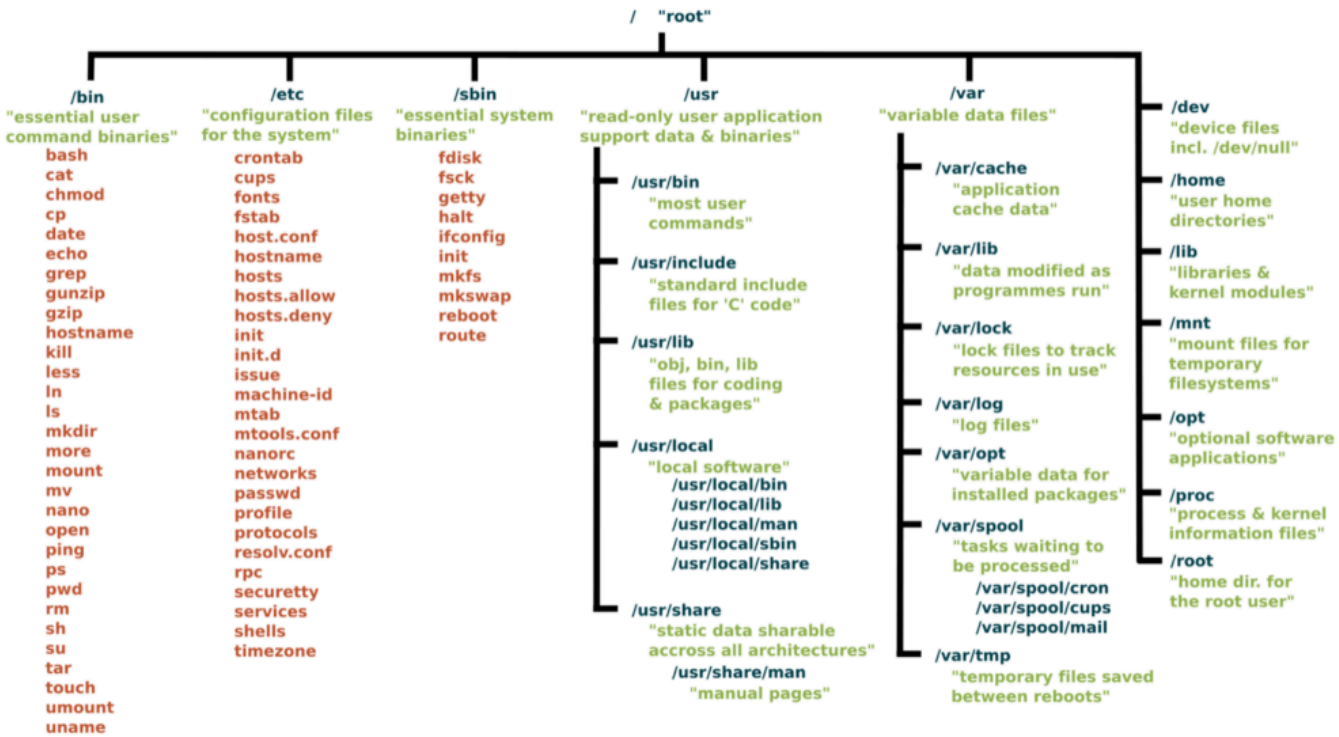


Figura 1. Estructura del filesystem de Linux.
Fuente: linuxfoundation.org

El sistema de ficheros dota al usuario de mecanismos lógicos para localizar los ficheros a través de su nombre y de una ruta de acceso (en inglés, *path*). En la figura 1 podemos ver qué estructura comparten los sistemas Linux del sistema de ficheros.

Los usuarios tienen su propia partición (espacio dedicado) dentro de «/home». Los discos duros externos y lápices de memoria son automáticamente localizados (montados) dentro del directorio «/media» en Linux o en el directorio «/Volumes» en Mac OS-X.

A diferencia de Microsoft Windows, los dispositivos no reciben una única letra como identificador, sino un nombre lógico más comprensible.

Los ficheros son agrupados en carpetas o directorios, conformando una jerarquía que proporciona una organización coherente para el usuario. Para construir una ruta que pase por dos directorios *A* y *B* debe introducirse el carácter separador /, formando la ruta A/B. En un instante concreto, el directorio en el que el usuario se encuentra recibe la denominación simbólica de «.», mientras que «..» representa al directorio inmediatamente anterior en el árbol de ficheros. Para acceder a un archivo especificaremos la ruta completa de este (*path* absoluto), es decir, toda la serie de directorios desde la raíz (*root o /*) hasta ese punto del sistema de ficheros. Para acceder a cualquier fichero o directorio podemos descender desde este punto a lo largo del sistema de ficheros. Opcionalmente, el usuario puede introducir sencillamente la parte de la ruta necesaria para completar el resto del camino a partir de la ubicación actual (*path* relativo). Si quisiéramos acceder al fichero /home/uoc/master/notas.txt (*path* absoluto) y estuviésemos en /home/uoc, simplemente necesitamos indicar ./master/notas.txt (*path* relativo). Como hemos mencionado anteriormente, el «.» indica el directorio actual que va seguido de la ruta hasta notas.txt.

En UNIX, se accede a los ficheros de una unidad como a cualquier otro dispositivo lógico, requiriendo ciertos permisos de seguridad para poder efectuar cualquier operación sobre estos.

Un programa puede realizar sobre un fichero las siguientes acciones:

- Abrir (en inglés, *open*). Para acceder a un fichero, el proceso debe abrirlo previamente y obtener un código identificador para referirse a este.
- Cerrar (en inglés, *close*). Una vez finalizado el acceso, el fichero debe cerrarse para que otro proceso pueda reutilizarlo posteriormente.
- Leer (en inglés, *read*). El proceso utiliza el identificador de un fichero para acceder secuencialmente a su contenido.
- Escribir (en inglés, *write*). Un proceso puede escribir nueva información en un fichero, sobrescribiendo el contenido existente anteriormente.
- Añadir (en inglés, *append*). Un proceso puede escribir nueva información a continuación del contenido registrado con anterioridad.

En UNIX todo se reduce a ficheros. Podemos encontrar tres tipos de ficheros:

- Archivos regulares
- Directorios
- Especiales:
  - *Block files*
  - *Character device files*
  - *Pipe files*
  - *Socket files*
  - *Symbolic link files*

De los tipos especiales nos centraremos en los *symbolic link files*. Además de los ficheros regulares y de los directorios que todos conocemos, se pueden crear enlaces a los ficheros desde otros puntos del sistema de ficheros (en inglés, *links*). De este modo, evitamos introducir la ruta completa de acceso en cada ocasión. Estos enlaces en UNIX pueden contener la ruta de acceso del fichero original (en inglés, *soft links* o *symbolic links*) o proporcionar un acceso físico compartido a este con un nombre diferente (en inglés, *hard links*). Los *soft links* equivaldrían a un acceso directo en Windows. Con este mecanismo, el usuario puede apuntar a un mismo fichero desde varios lugares del sistema, sin la existencia de múltiples copias de este.

UNIX posee un mecanismo característico de seguridad para proteger la integridad del sistema de ficheros. Únicamente los usuarios autorizados pueden realizar operaciones sobre un fichero o directorio. Según su procedencia, cada usuario del sistema pertenece a uno de estos dominios: el usuario (*user*), el grupo de trabajo (*group*) o el entorno exterior (*others*). Las operaciones permitidas sobre archivos son la lectura (*read*), la escritura/modificación/eliminación (*write*) y la ejecución (*execute*). En el caso

de los directorios, existen convenciones similares para restringir el acceso a su interior, denotado en este caso con el permiso de ejecución.

Por regla general, el autor de un fichero posee inicialmente todos los derechos, garantizando la lectura y la ejecución de este a los miembros de su grupo de trabajo. Según el grado de privacidad permitido por los tenedores de los derechos, un usuario del entorno exterior puede estar habilitado para ver esos ficheros o no. En cualquier contexto, el administrador de la máquina (en inglés, *root*) puede revocar los permisos de seguridad de un fichero. Para poder modificar los permisos de ficheros o directorios más adelante veremos la utilización del comando *chmod*.

1. Introducció als entorns de treball UNIX

1.6. Funcionamiento de las máquinas virtuales

Hasta hace relativamente poco tiempo, no era nada sencillo para los usuarios de ordenadores personales disponer de una máquina funcionando con Linux. Implicaba la desinstalación del SO previamente instalado, o cuando menos, la creación de particiones independientes con un gestor de arranque dual que permitiera la coexistencia de ambos sistemas. Las distribuciones de Linux, además, carecían de un protocolo simple de instalación, requiriendo de un profundo conocimiento a nivel técnico de la máquina. Afortunadamente, en el momento actual se han superado ampliamente muchas de las barreras que complican el acceso a esta tecnología. De hecho, hoy en día podemos probar fácilmente *in situ* la mayoría de las distribuciones Linux en nuestros ordenadores sin modificar su configuración, mediante el uso de las denominadas máquinas virtuales.

Una máquina virtual (MV) es un programa que imita el funcionamiento de un ordenador, trabajando como una aplicación convencional dentro de nuestra propia máquina. De este modo, mientras nuestra máquina está gobernada por un SO que recibe la denominación de huésped (en inglés, *host*), la MV funciona bajo el control de un segundo SO que actúa como invitado (en inglés, *guest*). Para dotar de esta funcionalidad a nuestro ordenador es preciso instalar en nuestra máquina un software de virtualización capaz de gestionar múltiples máquinas virtuales simultáneamente. Los programas gestores de máquinas virtuales pueden instalarse en múltiples plataformas para actuar como huésped, siendo capaces de ejecutar dentro de una ventana una máquina ficticia gestionada por otro SO diferente invitado. En términos prácticos, la MV para el ordenador huésped es una aplicación convencional, mientras que desde el interior de esta la emulación logra que el SO invitado crea que trabaja sobre una verdadera máquina física funcionando a su entera disposición.

El sistema de ficheros de la MV invitada se almacena físicamente en un único fichero dentro de nuestro ordenador, junto con las opciones de configuración establecidas durante la instalación. Lógicamente, la MV posee acceso a determinados periféricos de nuestra propia máquina, tales como el teclado, el ratón, la pantalla o el acceso a Internet. Para realizar el intercambio de información entre la MV y nuestra máquina podemos acceder a los dispositivos USB conectados físicamente en nuestro ordenador, crear una carpeta compartida entre el *guest* y el *host* o depositar nuestros ficheros en la red a través de distintos portales de la nube. No obstante, tanto la configuración real de nuestro ordenador como el núcleo de nuestro sistema de ficheros montado desde nuestros discos duros internos permanecerán ocultos para la MV. La gestión de diferentes tipos de datos dentro de la MV obviamente repercute en un tiempo de respuesta mayor que en el caso de trabajar de forma nativa con el mismo SO invitado, pero las últimas versiones de los programas de virtualización están claramente disminuyendo estas diferencias. En conclusión, esta aproximación resulta enormemente atractiva para probar cualquier nuevo sistema sin modificar nuestro entorno de trabajo habitual.

1. Introducción a los entornos de trabajo UNIX

1.7. El terminal como herramienta de trabajo

1.7.1. El terminal

El entorno de trabajo de Ubuntu (https://ubuntu.com/), al igual que cualquier sistema de la familia Gnu/Linux, cuenta con un gestor gráfico de X-Windows (en castellano, *ventanas*) que proporciona una interfaz amigable al usuario para acceder a los recursos de su máquina. Sin embargo, en los primeros sistemas operativos de las computadoras, no existían entornos interactivos controlados por el ratón ni pantallas con la resolución gráfica que conocemos hoy en día.

Uno de los típicos iconos asociados a la aplicación del terminal. En Gnu/Linux también recibe el nombre de *intérprete de comandos* (en inglés, *shell*). En la figura 2 se muestra el símbolo que representa el intérprete de comandos.

Figura 2. Símbolo de intérprete de comandos.

El trabajo desde terminales de líneas de comando (en inglés, *CLI*, *command line interface*) desde una *shell*, por ejemplo, tipo *bash* es importante en un entorno de bioinformática por varias razones:

- Flexibilidad. El uso del terminal permite al usuario una mayor flexibilidad y control sobre el procesamiento y análisis de los datos de bioinformática, ya que se puede utilizar una amplia variedad de herramientas de línea de comandos y se puede combinarlas de manera efectiva.

- Automatización. La automatización de tareas es mucho más fácil en un terminal tipo *bash*, puesto que se pueden crear *scripts* y programas para procesar grandes cantidades de datos sin la necesidad de realizar cada tarea manualmente.

- Reproductibilidad. El terminal tipo *bash* permite al usuario reproducir fácilmente un análisis o procesamiento de datos en cualquier momento, ya que todos los comandos y operaciones efectuados quedan registrados en el historial de comandos.

- Eficiencia. Trabajar desde un terminal tipo *bash* es a menudo más eficiente que trabajar en una interfaz gráfica de usuario, porque se pueden realizar muchas operaciones en una sola línea de comando.

En resumen, trabajar desde una *shell* tipo *bash* en un entorno de bioinformática es importante para permitir una mayor flexibilidad, automatización, reproducibilidad y eficiencia en el procesamiento y análisis de los datos de bioinformática.

En el entorno de investigación en bioinformática, el trabajo desde un terminal constituye el núcleo de las actividades habituales. Por esta razón, la mayoría de los sistemas actuales siguen manteniendo el terminal como una aplicación esencial. Como veremos en esta asignatura, esta herramienta resulta ideal para diseñar protocolos de trabajo que requieren el acceso repetido a determinados conjuntos de datos para efectuar cálculos intensivos.

La *shell* funciona mediante la ejecución del intérprete de comandos, la cual permanece en espera hasta que el usuario ingresa una nueva orden. Cuando se ejecuta un comando, el intérprete crea un nuevo proceso para llevar a cabo la tarea. Existen dos modos de ejecución de un comando: el modo en primer plano o síncrono, que bloquea el terminal durante la ejecución del proceso (en inglés, *foreground*), y el modo en segundo plano o asíncrono (en inglés, *background*), que no interrumpe la actividad ordinaria del terminal y permite lanzar nuevas órdenes mientras el proceso se ejecuta en segundo plano. Por ejemplo, es posible editar un archivo de texto en una ventana aparte mientras se ejecutan otros comandos en el terminal. Una vez que un comando asíncrono finaliza, el intérprete informa al usuario apropiadamente. En el subapartado 1.11 («Gestión básica de procesos») se amplían los conocimientos sobre los comandos para gestionar el estado de los procesos en ejecución.

En la mayoría de los sistemas Gnu/Linux, la *shell* predeterminada es la *shell bash*. Para saber cuál es la *shell* de inicio de sesión predeterminada, escribid en el terminal los comandos que os detallaremos a continuación. Desde este momento, es muy recomendable que pongáis en práctica los ejemplos incluidos a lo largo de esta asignatura para mostrar el funcionamiento del terminal. Por otra parte, el carácter `$` denota la entrada de comandos desde el terminal; tenéis que escribir a continuación de este símbolo lo siguiente:

```
$ whoami
```

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

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

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

El comando `who am i` muestra vuestro nombre de usuario, y el comando `grep` muestra la definición de vuestra cuenta de usuario en el archivo /etc/passwd. El último campo en esa entrada muestra que la *shell bash* (/bin/bash) es tu *shell* predeterminada (la que se inicia cuando iniciáis sesión o abrís una ventana del terminal).

> Vale la pena conocer el intérprete de comandos *bash*, no solo porque es el predeterminado en la mayoría de las instalaciones, sino porque es el más utilizado en las certificaciones profesionales de Gnu/Linux.

1. Introducción a los entornos de trabajo UNIX

1.7. El terminal como herramienta de trabajo

1.7.2. Ejecución de comandos

La forma más sencilla de ejecutar un comando es escribir el nombre del comando desde una *shell*. Desde vuestro escritorio *ubuntu*, abrid una ventana terminal y luego escribid el siguiente comando:

```
$ date
```

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

Escribir el comando `date`, sin opciones ni argumentos, muestra el día, mes, fecha, hora, zona horaria y año actuales, tal como se muestra arriba. Quizás en vuestro terminal no veáis lo mismo porque el formato puede cambiar dependiendo de la zona horaria. A continuación, algunos otros comandos que podéis probar:

```
$ pwd
```

```
/home/student
```

```
$ hostname
```

```
ubuntuM0151
```

```
$ ls
```

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

El comando `pwd` muestra vuestro directorio de trabajo actual. Al escribir `hostname` se muestra el nombre de `host` de vuestro ordenador. El comando `ls` lista los archivos y directorios en vuesro directorio actual.

Aunque muchos comandos se pueden ejecutar simplemente escribiendo los nombres de los comandos, es más común escribir algo más después del comando para modificar su comportamiento. Los caracteres y palabras que podéis escribir después de un comando se denominan *opciones* y *argumentos*.

1. Introducción a los entornos de trabajo UNIX

1.7. El terminal como herramienta de trabajo

1.7.3. Entendiendo la sintaxis de los comandos

La mayoría de los comandos tienen una o más opciones que podéis agregar para cambiar su comportamiento. Las opciones suelen consistir en una sola letra, precedida por un guion. Sin embargo, podéis agrupar opciones de una sola letra juntas o preceder cada una con un guion para usar más de una opción al mismo tiempo. Por ejemplo, los siguientes dos usos de opciones para el comando `ls` son equivalentes:

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

```
$ ls -lat
```

En ambos casos, se ejecuta el comando `ls` con las opciones `-l` (listado largo), `-a` (mostrar archivos ocultos con puntos), y `-t` (listar por tiempo). Algunos comandos incluyen opciones que están representadas por una palabra completa. Para indicarle a un comando que use una palabra completa como opción, generalmente debe ser precedida por dos guiones (`--`). Por ejemplo, para usar la opción de ayuda en muchos comandos, deberéis añadir `--help` en la línea de comandos. Sin los dos guiones, las letras *h*, *e*, *l* y *p* serían interpretadas como opciones separadas.

Muchos comandos también aceptan argumentos después de que ciertas opciones sean ingresadas o al final de toda la línea de comando. Un argumento es una pieza extra de información, como un nombre de archivo, directorio, nombre de usuario, dispositivo u otro elemento que indica al comando sobre qué actuar.

Por ejemplo, `cat /etc/passwd` muestra el contenido del archivo `/etc/passwd` en tu pantalla. En este caso, `/etc/passwd` es el argumento. Por lo general, podéis tener tantos argumentos como deseéis en la línea de comando, limitados solo por el número total de caracteres permitidos en una línea de comando. Aquí hay un ejemplo de una opción con tres letras que es seguida por un argumento:

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

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

En el ejemplo de `tar` mostrado anteriormente, las opciones dicen que se debe crear (`c`) un archivo (`f`) llamado *copiaseguridad.tar* que incluya todo el contenido del directorio `/home/student` y sus subdirectorios, y que muestre mensajes detallados mientras se crea la copia de seguridad (`v`). Debido a que *copiaseguridad.tar* es un argumento de la opción *f*, *copiaseguridad.tar* debe seguir inmediatamente a la opción.

Aquí hay algunos comandos que podéis probar. Observad cómo se comportan de manera diferente con diferentes opciones:

```
$ uname
```

```
Linux
```

```
$ uname -a
```

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

El comando `uname` muestra el tipo de sistema que estás ejecutando (Linux). Cuando agregáis `-a`, también podéis ver el nombre de *host* y la versión 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
```

El comando `date` tiene algunos tipos especiales de opciones. Por sí solo, `date` imprime simplemente el día, la fecha y la hora actuales, como se muestra en la primera orden. Pero el comando `date` admite la opción especial + de formato, que te permite mostrar la fecha en diferentes formatos. Escribid `date --help` para ver los diferentes indicadores de formato que podéis usar.

> Para avanzar por las páginas del manual utilizad las siguientes teclas: barra espaciadora (página siguiente), tecla *b* (página anterior), tecla *Enter* (avanzar línea a línea). Para localizar una palabra concreta, introducid el símbolo / y el patrón de búsqueda. Para salir del manual, pulsad la tecla *q* (*quit*).

1. Introducción a los entornos de trabajo UNIX

1.7. El terminal como herramienta de trabajo

1.7.4. Localizando comandos

Ahora que habéis escrito algunos comandos, puede que os preguntéis dónde están ubicados esos comandos y cómo el *shell* encuentra los comandos que habéis escrito. Para encontrar los comandos que escribís, el *shell* busca en lo que se conoce como *path* (en castellano, *ruta*). Para los comandos que no están en vuestro *path*, podéis escribir la identidad completa de la ubicación del comando.

Si conocéis el directorio que contiene el comando que deseas ejecutar, una forma de ejecutarlo es escribiendo la ruta completa, o absoluta, de ese comando. Por ejemplo, podéis ejecutar el comando `date` que se encuentra dentro del directorio /bin escribiendo:

```
$ /bin/date
```

Por supuesto, esto puede ser inconveniente, especialmente si el comando reside en un directorio con una ruta larga. La mejor manera es tener los comandos almacenados en directorios conocidos y luego agregar esos directorios a la variable de entorno PATH de tu *shell*. El *path* consiste en una lista de directorios que se verifican secuencialmente para los comandos que habéis escrito. Para ver vuestro *path* actual, escribid lo siguiente:

```
$ echo $PATH
```

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

Los resultados muestran un *path* predeterminado común para un usuario regular de Gnu/Linux. Los directorios en la lista del *path* están separados por dos puntos. La mayoría de los comandos de usuario que vienen con Gnu/Linux se almacenan en los directorios /bin, /usr/bin o /usr/local/bin. Los directorios /sbin y /usr/sbin contienen comandos administrativos (algunos sistemas Gnu/Linux no colocan esos directorios en los *paths* de los usuarios regulares). El primer directorio mostrado es el directorio *bin* en el directorio de inicio del usuario (/home/student/bin). Si deseáis agregar vuestros propios comandos o *scripts* de *shell*, colocadlos en el directorio *bin* en vuestro directorio de inicio (ie. /home/student/bin para un usuario llamado *student*). Este directorio se agrega automáticamente a vuestro *path* en algunos sistemas Gnu/Linux, aunque es posible que necesitéis crear ese directorio o agregarlo a vuestro PATH en otros sistemas Gnu/Linux. Entonces, siempre que agreguéis el comando a vuestro *bin* con permiso de ejecución, podéis comenzar a usarlo simplemente escribiendo el nombre del comando en el indicador de vuestro *shell*. Si se considera que el nuevo comando esté disponible para todos los usuarios, podéis agregarlo con el usuario *root* al directorio /usr/local/bin o /opt/nombre_paquete/bin.

A diferencia de algunos otros sistemas operativos, Gnu/Linux no verifica el directorio actual del ejecutable. Inmediatamente, comienza a buscar en el *path*, y los ejecutables en el directorio actual solo se ejecutan si están en la variable PATH o si dais su dirección absoluta (como /home/student/scriptx.sh) o relativa (por ejemplo, y fijaos en el punto antes de la contrabarra, ./scriptx.sh). Los directorios que pertenecen al *path* de ejecutables se pueden averiguar escribiendo $PATH en el *prompt* del terminal:

```
$ echo $PATH
```

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

El orden del directorio del *path* es importante. Los directorios se verifican de izquierda a derecha. Entonces, en este ejemplo, si hay un comando llamado `foo` ubicado en ambos directorios /bin y /usr/bin, se ejecuta el que está en `/bin`. Para que se ejecute el otro comando *foo*, debéis escribir el *path* completa del comando o cambiar vuestra variable PATH (un ejemplo práctico de cómo cambiar la variable PATH y agregar directorios se describe en los apartados 1.15 y 1.17).

1. Introducción a los entornos de trabajo UNIX

1.7. El terminal como herramienta de trabajo

1.7.5. Recuperación de comandos mediante el historial de comandos

Puede ser conveniente poder repetir un comando que ejecutasteis anteriormente en una sesión de *shell*. Recordar una línea de comandos larga y compleja que escribisteis de manera incorrecta os puede ahorrar problemas. Afortunadamente, algunas características del *shell* os permiten recordar líneas de comandos anteriores, editar esas líneas o completar una línea de comando parcialmente escrita.

El historial del *shell* es una lista de los comandos que habéis ingresado antes. Usando el comando `history` en un *shell bash*, podéis ver tus comandos anteriores. Luego, utilizando varias características del *shell*, podéis recuperar líneas de comando individuales de esa lista y cambiarlas como deseéis.

Para probar un poco de edición de línea de comandos, escribid lo siguiente:

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

Este comando muestra el contenido del directorio `/usr/bin`, ordena el contenido en orden alfabético (sin importar mayúsculas o minúsculas) y pasa la salida a `less`. El comando `less` muestra la primera página de salida, después de lo cual podéis navegar por el resto de la salida de una línea (presiona *enter*) o a páginas (presiona la barra espaciadora) a la vez. Simplemente, presionad la tecla *q* cuando hayáis terminado. Ahora, supongamos que queréis cambiar vuestra línea de comandos en el terminal de `/usr/bin` a `/bin` y no queréis escribir demasiado. Si os colocáis en la línea de comandos, y seguís los siguientes pasos, podréis cambiar parte de la línea de comandos:

- Presionad la flecha hacia arriba (↑). Esto muestra el comando más reciente de tu historial de *shell*. Prueba para presionar más de una vez.
- Presionad Ctrl+A. Esto mueve el cursor al principio de la línea de comandos.
- Presionad Ctrl+E. Esto mueve el cursor al final de la línea de comandos.
- Presionad Ctrl+F o la flecha derecha (→). Repetid este comando varias veces para colocar el cursor debajo de la primera barra (/).
- Presionad Ctrl+D. Escribid este comando cuatro veces para eliminar `/usr` de la línea.
- Presionad Enter. Ejecuta la línea de comandos una vez que hagáis decido qué ejecutar.

Mientras se edita una línea de comandos, en cualquier momento podéis escribir caracteres normales para añadirlos a la línea de comandos. Los caracteres aparecen en la posición del cursor de texto. Podéis utilizar la flecha derecha →, izquierda ← para mover el cursor de un extremo a otro de la línea de comandos. También podéis pulsar las teclas de flecha arriba ↑ y abajo ↓ para recorrer los anteriores comandos en la lista del historial para seleccionar una línea de comandos para editar.

Después de escribir una línea de comando, toda la línea se guarda en la lista de historial de vuestro *shell*. La lista se almacena en el *shell* actual hasta que salís del terminal. Además, cada orden que se escribe en la línea de comandos y se ejecuta, tenga sentido o no, se escribe en un archivo de historial, donde cualquier comando puede ser recordado para ser ejecutado nuevamente en otra sesión. Una vez que se recupera un comando, se puede modificar la línea de comando, como se ha descrito anteriormente.

Para ver vuestra lista de historial, utilizad el comando `history`. Escribid el comando sin opciones o seguido de un número para listar esa cantidad de los comandos más recientes. Por ejemplo,

```
$ 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 lugar de simplemente ejecutar una línea de comando del historial de inmediato, se puede recuperar una línea en particular y editarla. Se muestran varios ejemplos.

Ejemplos

`$ !n`: Ejecutar el número de comando. Reemplazad la *n* con el número de la línea de comando y esa línea se ejecutará. Por ejemplo, así es como puedéis repetir el comando de fecha mostrado como el número de comando *2012* en la lista de historial anterior:

```
$ !2012
```

```
date

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

`$ !!`: Ejecutar comando anterior. Ejecuta la línea de comando anterior. Aquí os mostramos cómo ejecutar inmediatamente ese mismo comando `date`:

```
$ !!
```

```
date

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

`$ !?string?`: Ejecuta el comando que contiene un `string` (cadena). Esto ejecuta el comando más reciente que contiene una cadena de caracteres específica. Por ejemplo, podéis ejecutar nuevamente el comando `date` buscando solo una parte de esa línea de comando de la siguiente manera:

```
$ !?at?
```

```
date

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

1. Introducción a los entornos de trabajo UNIX

1.8. Gestión básica de ficheros

1.8.1. Introducción

En esta sección, aprenderemos los conceptos básicos para movernos por el sistema. Muchas tareas dependen de poder llegar o referenciar la ubicación correcta en el sistema. Como tal, este conocimiento realmente forma la base para poder trabajar eficazmente en Gnu/Linux. Aseguraos de entenderlo bien. Si deseáis seguirlos, iniciad sesión y abrid un terminal. La tabla 3 muestra los comandos para crear y utilizar archivos y directorios.

Tabla 3. Comandos para gestionar archivos y directorios.

Comando	Resultado
`cd`	(*change directory*) Cambiar a otro directorio
`pwd`	(*print working directory*) Imprimir el nombre del actual directorio de trabajo
`mkdir`	(*make directory*) Crear un directorio
`rmdir`	(*remove directory*) Eliminar un directorio vacío
`rm -r`	(*remove*) Eliminar el contenido de un directorio no vacío
`rm`	(*remove*) Eliminar ficheros
`chmod`	(*change file mode*) Cambiar los permisos de un fichero o directorio
`ls`	(*list*) Listar el contenido de un directorio
`cp`	(*copy*) Copiar un archivo
`mv`	(*move*) Mover un fichero

Fuente: elaboración propia.

Al iniciar sesión en un sistema Gnu/Linux y abrir un terminal, directamente, os encontraréis en el directorio de inicio (abreviado también con el símbolo ~). Dependiendo de la instalación, dicho directorio suele almacenarse a su nombre en un subdirectorio de la carpeta /home/. Para certificar en cada momento en qué lugar del *path* de directorios nos encontramos, podemos usar el comando `pwd` (en inglés, *print working directory*).

Uno de los comandos más básicos que se emplea en el terminal es `cd`. El comando `cd` se puede utilizar sin opciones (para llevarlo a su directorio de inicio) o con *paths* absolutos o relativos. Considerad los siguientes comandos:

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

```
$ pwd
```

```
/usr/lib
```

```
$ cd gcc
```

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

```
$ cd
```

```
$ pwd
```

```
/home/student
```

La opción /usr/bin representa el *path* absoluto a un directorio en el sistema. Debido a que comienza con una barra diagonal (/), este *path* indica al terminal que comience en la raíz del sistema de archivos y lo lleve al directorio *lib* que se encuentra en el directorio *usr*. La opción *gcc* del comando `cd` indica que se debe buscar un directorio llamado *gcc* que es relativo al directorio actual. Por lo tanto, eso hizo que /usr/lib/gcc fuera su nuevo directorio. Después de eso, al escribir solo `cd`, se regresa al directorio de inicio. Si alguna vez os preguntáis dónde os encuentráis en el sistema de archivos, el comando `pwd` os lo mostrará.

Los siguientes pasos os guiarán a través del proceso de creación de directorios dentro de tu directorio de inicio y de moverte entre ellos, con una mención sobre cómo establecer los permisos apropiados de los archivos. Os recomiendo que ejecutéis todos los comandos que están escritos.

1. Id a vuestro directorio de inicio. Para hacer esto, simplemente escribid `cd` en un terminal y presionad *enter*.

2. Para aseguraros de que estáis en vuestro directorio de inicio, escribid `pwd`.

```
$ pwd
```

```
/home/student
```

3. Cread un nuevo directorio llamado `testHIB` en vuestro directorio de inicio,

```
$ mkdir testHIB
```

4. Verificad los permisos del directorio:

```
$ ls -ld testHIB
```

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

Esta lista muestra que `testHIB` es un directorio (*d*). La *d* va seguida de los permisos (`rwxr-xr-x`), los cuales se explican más adelante en la sección «Entendiendo los permisos y la propiedad de los archivos». El resto de la información indica el propietario (`student`), el grupo (`student`) y la fecha en que los archivos en el directorio fueron modificados por última vez.

5. Escribe `$ chmod 700 testHIB`

Este paso cambia los permisos del directorio para que tengáis acceso completo y nadie más tenga acceso (los nuevos permisos se deben leer `rwx------`).

6. Haced que el directorio de prueba sea su directorio actual:

```
$ cd testHIB
```

```
$ pwd
```

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

Podéis crear archivos y directorios en el directorio de prueba junto con las descripciones en el resto de este capítulo. Cuando necesitéis identificar vuestro directorio de inicio en una línea de comando de *shell*, podéis usar lo siguiente:

- `$HOME` Esta variable de entorno almacena el nombre de tu directorio de inicio.

- ~ Representa vuestro directorio de inicio en la línea de comando. También podéis usar la tilde para identificar el directorio de inicio de otra persona.

Otras formas especiales de identificar directorios en la *shell* se describen a continuación, con ejemplos.

- `.` Un solo punto (`.`) se refiere al directorio actual.
- `..` Dos puntos (`..`) se refieren a un directorio directamente encima del directorio actual.
- `$PWD` Esta variable de entorno se refiere al directorio de trabajo actual.

```
$ pwd
```

```
/home/student
```

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

```
$ cd ..
```

```
$ pwd
```

```
/home
```

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

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

```
$ cd ~
```

```
$ pwd
```

```
/home/student
```

Escribir *paths* puede volverse tedioso. La línea de comandos tiene un pequeño mecanismo que nos ayuda en este aspecto. Se llama *Autocompletado del tabulador*.

Cuando comenzáis a escribir un *path* (en cualquier lugar de la línea de comandos), podéis presionar la tecla *tab* en vuestro teclado en cualquier momento, lo que invocará una acción de autocompletado. Si no sucede nada, eso significa que hay varias posibilidades. Si presionáis *tab* nuevamente, os mostrará esas posibilidades. Luego podéis continuar escribiendo y presionar *tab* nuevamente, y volverá a intentar autocompletar para vosotros. Es un poco difícil de demostrar aquí, por lo que probablemente sea mejor si lo intentáis por vosotros mismos.

1. Introducción a los entornos de trabajo UNIX

1.8. Gestión básica de ficheros

1.8.2. Metacaracteres y operadores

Ya sea que estéis listando, moviendo, copiando, eliminando o realizando cualquier otra acción con archivos en tu sistema Gnu/Linux, existen ciertos caracteres especiales, denominados *metacaracteres* y *operadores*, que os ayudarán a trabajar con archivos de manera más eficiente. Los metacaracteres pueden ayudar a hacer coincidir uno o varios archivos sin tener que escribir completamente cada nombre de archivo. Los operadores te permiten dirigir información de un comando o archivo a otro comando o archivo.

Por otra parte, también hay que introducir ciertos mecanismos de citación. Con ejemplos, será más fácil de entender:

Carácter escape

Una barra invertida no citada «\» es el carácter de *escape* de *bash*: preserva el valor literal del siguiente carácter que sigue, con la excepción de la nueva línea.

```
$ echo variable; argumento
```

```
variable
argumento: command not found
```

Si se escribe «\;» te ayuda a usar «;» como un carácter normal

```
$ echo variable \; argumento
```

```
variable ; argumento
```

Un ejemplo más sutil. El objetivo es crear un solo fichero que se llame *mi fichero.txt*

```
$ touch mi fichero.txt
```

```
$ ls mi*txt
```

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

```
$ rm fichero.txt m
```

Con el carácter *escape* se genera un único fichero

```
$ touch mi\ fichero.txt
```

```
$ ls mi*txt
```

```
'mi fichero.txt'
```

```
$ rm mi\ fichero.txt
```

Comillas sencillas

Encerrar caracteres en comillas simples (') preserva el valor literal de cada carácter dentro de las comillas. Una comilla simple no puede darse entre comillas simples, incluso cuando esté precedida por una barra invertida. Y ningún carácter es especial dentro de las cadenas entre comillas simples. Ejemplos:

```
$ echo 'variable; argumento'
```

```
variable; argumento
```

Se pueden colocar cadenas representadas por diferentes mecanismos de citación una al lado de la otra para concatenarlas. Otro ejemplo:

```
# concatenación of 4 strings

# 1: '@comandos = '

# 2: \'

# 3: 'sort and grep'

# 4: \'
```

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

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

Comillas dobles

Encerrar caracteres en comillas dobles (") preserva el valor literal de todos los caracteres dentro de las comillas, con la excepción de $, `, , y, cuando la expansión de historia está habilitada, !. Aquí hay un ejemplo que muestra la interpolación de variables dentro de comillas dobles:

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

Ningún carácter es especial dentro de las comillas simples

```
$ echo 'Desde principio de año, he ejecutado $qty GWAS'
```

```
Desde principio de año, he ejecutado $qty GWAS
```

Un uso típico de las comillas dobles es habilitar la interpolación de variables

```
$ echo "Desde principio de año, he ejecutado $qty GWAS"
```

```
Desde principio de año, he ejecutado 5 GWAS
```

A menos que quieras específicamente que la *shell* interprete el contenido de una variable, siempre debes entrecomillar la variable para evitar problemas debido a la presencia de metacaracteres de la *Shell*

```
$ f='segundo fichero.txt'
```

Sería lo mismo que: echo 'pec informe' > segundo fichero.txt

```
$ echo 'pec informe' > $f
```

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

Sería lo mismo que: echo 'pec informe' > 'segundo fichero.txt'

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

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

```
pec informe
```

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

Ahora sí que nos introducimos más profundamente en la definición de metacaracteres y operadores.

Metacaracteres

Para ahorrar algunas pulsaciones de teclas y permitiros referencias fáciles a un grupo de archivos, la *shell bash* os permite usar metacaracteres. Aquí hay algunos metacaracteres útiles para hacer coincidir nombres de archivos. En la tabla 4 se describen los más habituales.

Tabla 4. Metacaracteres de fichero.

Comodín	Descripción
?	Coincidir un carácter, cualquier carácter
*	Coincidir cualquier cantidad de caracteres
[....]	Coincidir cualquiera de los caracteres entre los corchetes, que pueden incluir un rango de letras o números separados por un guion
[! ....]	No coincidir con ninguno de los caracteres entre los corchetes, que pueden incluir un rango de letras o números separados por un guion

Fuente: elaboración propia.

Probad algunos de estos metacaracteres de coincidencia de archivos yendo primero a un directorio vacío (como el directorio de prueba descrito en la sección anterior) y creando algunos archivos vacíos. El comando `touch` crea archivos vacíos. Las líneas de comandos que siguen le muestran cómo utilizar metacaracteres de *shell* con el comando `ls` para coincidir con nombres de archivo. Cualquier otro comando también funciona.

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

```
$ ls P*
```

```
Pharmacogenomics  Proteomics
```

Se imprime cualquier archivo que comience con P

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

```
Proteomics
```

Se imprime cualquier archivo que comience con P y contenga la t

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

```
Epigenomics  Metagenomics
```

Se imprime cualquier archivo que comience por E or M

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

```
Pharmacogenomics  Proteomics  Transcriptomics
```

Se imprimen todos los nombres de archivo que comienzan con una letra entre P y Z

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

```
Epigenomics
```

Se imprime cualquier archivo de 11 caracteres que termine con la cadena genomics

Metacaracteres de redireccionamiento de archivos

Los comandos reciben datos desde la entrada estándar y los envían a la salida estándar. Utilizando *pipes* (en castellano, tuberías) se puede dirigir la salida estándar de un comando a la entrada estándar de otro. Con archivos, se puede usar el signo menor que (<) y mayor que (>) para dirigir datos hacia y desde archivos. En la tabla 5 se recogen los caracteres de redireccionamiento de archivos:

Tabla 5. Metacaracteres de redireccionamiento.

Comodín	Descripción
<	Dirige el contenido de un archivo al comando. En la mayoría de los casos, esta es la acción predeterminada esperada por el comando y el uso del carácter es opcional; usar «`less seq.fa`» es lo mismo que «`less < seq.fa`»
>	Dirige la salida estándar de un comando a un archivo. Si el archivo existe, el contenido de ese archivo se sobrescribe
2>	Dirige el error estándar (mensajes de error) al archivo
&>	Dirige tanto la salida estándar como el error estándar al archivo
>>	Dirige la salida de un comando a un archivo, agregando la salida al final del archivo existente

Fuente: elaboración propia.

Las siguientes líneas son ejemplos realizados mediante líneas de comando donde la información se dirige hacia y desde archivos:

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

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

```
$ echo "Estoy practicando los ejercicios de HIB en $(date)" >>
~/testHIB/testimonio
```

En el primer ejemplo, el contenido del archivo *.bashrc* en el directorio de inicio se envía en un mensaje de correo al usuario *root* de la máquina Gnu/Linux. La segunda línea de comando formatea la página del manual de *chown* (utilizando el comando `man`), elimina los espacios en blanco adicionales (`col -b`) y envía la salida al archivo /tmp/chown (borrando el archivo /tmp/chown anterior, si existiera). El comando final genera un fichero de texto que se visualiza con el comando `cat`:

```
$cat testHIB/testimonio
```

```
Estoy practicando los ejercicios de HIB en Wed 19 Apr 19:08:04 CEST 2023
```

Otro tipo de redireccionamiento permite escribir texto que se puede emplear como entrada estándar para un comando. Los documentos implican, aquí, ingresar dos caracteres de menor (<<) después de un comando seguido de una palabra. Todo lo que escribáis después de esa palabra se tomará como entrada del usuario hasta que se repita la palabra en una línea aparte.

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

```
> Os recuerdo que hay que realizar todos los ejercicios
> propuestos para entender la asignatura.
>
> equipo HIB
> mensaje
$
```

```
$
```

Este ejemplo envía un mensaje de correo a los usuarios *student*, Josep, Joan, Guerau y Bego. El texto introducido entre <<mensaje>> y <<mensaje>> se convierte en el contenido del mensaje.

Caracteres de expansión de llaves

Al usar llaves ({}), podéis expandir un conjunto de caracteres en los nombres de archivo, nombres de directorios u otros argumentos que des a los comandos. Por ejemplo, si queréis crear el conjunto de archivos desde *sequence1* hasta *sequence7*, podéis hacerlo de la siguiente manera:

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

```
$ ls
```

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

```
$ rm sequence?
```

Los elementos que se expanden no tienen que ser números, ni siquiera dígitos únicos; podríais usar rangos de números o dígitos. También podríais utilizar cualquier cadena de caracteres, siempre y cuando se separen con comas. Ejemplo:

```
$ 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 de archivos y la propiedad

Después de trabajar con Gnu/Linux durante un tiempo, es casi seguro que obtendréis un mensaje de permiso denegado. Los permisos asociados a archivos y directorios en Gnu/Linux fueron diseñados para evitar que los usuarios accedan a los archivos privados de otros usuarios y para proteger los archivos importantes del sistema. Los nueve *bits* asignados a cada archivo para los permisos definen el acceso que vosotros y otros tienen a tu archivo. Los *bits* de permiso para un archivo regular aparecen como `-rwxrwxrwx`. Esos *bits* se usan para definir quién puede leer, escribir o ejecutar el archivo.

De los permisos de nueve *bits*, los primeros tres *bits* se aplican al permiso del propietario, los siguientes tres se aplican al grupo asignado al archivo y los últimos tres se aplican a todos los demás. La *r* significa lectura, la *w* significa escritura y la *x* significa permisos de ejecución. Si aparece un guion en lugar de la letra, significa que ese permiso está desactivado para ese *bit* asociado de lectura, escritura o ejecución.

Podéis ver los permisos de cualquier archivo o directorio escribiendo el comando `ls -ld`. El archivo o directorio nombrado aparece como se muestra en este ejemplo:

```
 $ ls -ld testHIB  testHIB/testimonio
```

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

La primera línea muestra que el archivo *testimonio* tiene permiso de lectura y escritura para el propietario y el grupo. Todos los demás usuarios tienen permiso de lectura, lo que significa que pueden ver el archivo, pero no pueden cambiar su contenido ni eliminarlo. La segunda línea muestra el directorio testHIB (indicado por la letra *d* antes de los *bits* de permiso). El propietario tiene permisos de lectura, escritura y ejecución, mientras que el grupo y otros usuarios solo tienen permisos de lectura y ejecución. Como resultado, el propietario puede agregar, cambiar o eliminar archivos en ese directorio, y todos los demás solo pueden leer el contenido, cambiar a ese directorio y listar el contenido del directorio.

Si sois propietarios de un archivo, podéis usar el comando `chmod` para cambiar los permisos como deseéis. En un método para hacer esto, a cada permiso (lectura, escritura y ejecución) se le asigna un número: *r* = 4, *w* = 2 y *x* = 1, y utiliza el número total de cada conjunto para establecer el permiso. Por ejemplo, para hacer que los permisos estén completamente abiertos para vosotros como propietarios, estableceríais el primer número en 7 (4 + 2 + 1), y luego daríais al grupo y a otros permisos de solo lectura, estableciendo ambos el segundo y el tercer número en 4 (4 + 0 + 0), de modo que el número final sea 744. Cualquier combinación de permisos puede resultar desde 0 (sin permiso) hasta 7 (permiso completo).

Aquí hay algunos ejemplos de cómo cambiar el permiso de un archivo (llamado *archivo*) y de qué permiso resultaría:

```
 # La ejecución del comando chmod da como resultado este permiso:
rwxrwxrwx
```

```
 $ chmod 777 archivo
```

```
 # La ejecución del comando chmod da como resultado este permiso: rwxr-xr-
x
```

```
 $ chmod 755 archivo
```

```
 # La ejecución del comando chmod da como resultado este permiso: rw-r--r-
-
```

```
 $ chmod 644 archivo rw-r--r-
```

```
# La ejecución del comando chmod da como resultado este permiso: --------
-
```

```
$ chmod 000 archivo
```

El comando chmod también se puede usar de manera recursiva. Por ejemplo, supongamos que deseáis dar una estructura de directorios completa con permiso 755 (rwxr-xr-x), comenzando en el directorio $HOME/testHIB. Para hacer eso, se podría usar la opción -R (recursiva), como sigue:

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

Todos los archivos y directorios debajo, e incluyendo el directorio Work en su directorio de inicio, tendrán permisos 755 establecidos. Debido a que el enfoque de los números para establecer cambios de permiso, con todos los *bits* de permiso a la vez, generan confusión, es más común usar letras para cambiar recursivamente los *bits* de permiso en un gran conjunto de archivos.

También podéis activar y desactivar los permisos de un archivo utilizando los signos más (+) y menos (-), respectivamente, junto con letras para indicar qué cambios y para quién. Empleando letras, para cada archivo, podéis cambiar los permisos para el usuario (*u*), el grupo (*g*), otros (*o*) y todos los usuarios (*a*). Lo que cambiaríais incluye los *bits* de lectura (*r*), escritura (*w*) y ejecución (*x*). Por ejemplo, comenzáis con un archivo que tenga todos los permisos abiertos (rwxrwxrwx). Ejecutáis los siguientes comandos chmod utilizando opciones con signo menos. Los permisos resultantes se muestran a la derecha de cada comando:

```
# La ejecución del comando chmod resulta en este permiso: r-xr-xr-x
```

```
$ chmod a-w archivo
```

```
# La ejecución del comando chmod resulta en este permiso: rwxrwxrw-
```

```
$ chmod o-x archivo
```

```
# La ejecución del comando chmod resulta en este permiso: rwx------
```

```
$ chmod go-rwx archivo
```

Asimismo, los siguientes ejemplos comienzan con todos los permisos cerrados (---------). El signo más se utiliza con chmod para activar los permisos:

```
# La ejecución del comando chmod resulta en este permiso: rw-------
```

```
$ chmod u+rw archivos
```

```
# La ejecución del comando chmod resulta en este permiso: --x--x--x
```

```
$ chmod a+x archivos
```

```
# La ejecución del comando chmod resulta en este permiso: r-xr-x---
```

```
$ chmod ug+rx archivos
```

El uso de letras para cambiar los permisos de manera recursiva con chmod generalmente funciona mejor que usar números, porque se pueden cambiar los *bits* selectivamente, en lugar de cambiar todos los *bits* de permiso a la vez.

1. Introducción a los entornos de trabajo UNIX

1.9. Acceder al contenido de los ficheros

1.9.1. Introducción

Un filtro, en el contexto de la línea de comandos de Gnu/Linux, es un programa que acepta datos textuales y los transforma de una manera particular. Los filtros son una forma de tomar datos en bruto, ya sea producidos por otro programa o almacenados en un archivo, y manipularlos para que se muestren de una manera más adecuada para encontrar lo que estamos buscando. Estos filtros a menudo tienen varias opciones de línea de comando que modificarán su comportamiento, por lo que siempre es bueno consultar la página del manual de un filtro para ver lo que está disponible.

En los ejemplos que se muestran a continuación, proporcionaremos entrada a estos comandos mediante un archivo, pero también veremos que podemos proporcionar entrada a través de otros medios que agregan mucha más potencia (tabla 6). Además, recuerda que el archivo se especifica como un *path* y, por lo tanto, puedes usar *paths* absolutos y relativos, y también comodines. Por otra parte, estas herramientas que estamos viendo solo sirven para trabajar con ficheros de texto, no binarios.

Tabla 6. Comandos para acceder al fichero.

Comando	Descripción
`cat`	Imprime un fichero en el terminal
`more`	Muestra el resultado de la ejecución de un comando en el terminal de una página a la vez
`head`	Es un comando que imprime las primeras diez líneas de su entrada, pero podemos modificar esto con un argumento de línea de comando
`tail`	Es un comando que imprime las últimas diez líneas de su entrada, pero podemos modificar esto con un argumento de línea de comando
`less`	Es un visor de ficheros de texto, con este comando no podremos editar el fichero, pero sí navegar por su contenido
`nl`	*`number line`* significa 'numerar líneas', y eso es exactamente lo que hace
`wc`	*`word count`* significa 'contar palabras' y hace precisamente eso (así como también cuenta caracteres y líneas). De forma predeterminada, dará un recuento de los tres, pero usando opciones de línea de comando podemos limitarlo a lo que necesitemos
`diff`	Compara línea a línea dos ficheros de texto
`paste`	Une ficheros tabulares línea por línea
`od`	*`octal dump`* convierte la entrada en múltiples formatos, con formato octal por defecto, y ayuda a comprender los datos complejos que no son legibles para los humanos
`sed`	*`stream editor`* `Editor` de flujo. Nos permite hacer una búsqueda y reemplazo, entre otras acciones, en nuestros datos

Fuente: elaboración propia.

En el campo de la bioinformática los archivos son muy grandes; incluso los editores en línea pueden tener problemas para abrirlos. Existen otras formas de acceder a los contenidos del fichero. Uno de ellos sería imprimir el fichero en el terminal utilizando el comando `cat`.

> Recuerda que con Crtl + C los programas se terminan inmediatamente y se vuelve a mostrar el *prom*pt (en castellano, la línea de comandos).

```
$cat hg38_RefSeq.txt
```

`cat` es, además, capaz de concatenar textos uno detrás de otro en el orden en que se los pasamos, y de mostrarlos en pantalla.

```
$ cat file1 file2 file3
```

O se puede generar un nuevo fichero.

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

Algunas opciones interesantes de `cat` son:

- `-A`: muestra también los caracteres de control, básicamente los tabuladores (como `^I`) y los retornos de carro (`$`).
- `-n`: numera todas las líneas.

Para obtener una visión general del contenido del archivo sin ocupar todo el terminal, se pueden imprimir solo las primeras líneas usando el comando `head`:

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

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

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

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

Existe el comando `tail`, y permite imprimir el final de los archivos.

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

Otro comportamiento de `tail` que resulta útil es que puede mostrar todas las líneas, excepto las k primeras líneas. Para ello hay que usar la opción `-n` y el número de líneas que queremos omitir precedido por un `+`. Si se quiere omitir las primeras veintidós líneas podemos escribir:

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

Cuando se necesita examinar un archivo de texto para familiarizarse con su contenido, es común abrirlo y navegar a través de él. Sin embargo, si el archivo es muy grande, puede haber problemas al intentar abrirlo con un editor de texto.

En estos casos, una herramienta útil es `less`, un visor de archivos de texto que puede manejar archivos inmensos sin problemas. Aunque `less` no permite la edición del archivo, sí nos permite navegar por su contenido de manera interactiva. Al ejecutar *less*, el programa se abrirá en el terminal y hará que el *prompt* desaparezca temporalmente. Podremos salir del programa en cualquier momento presionando la tecla *q*.

```
$ less hg38_RefSeq.txt
```

Dentro de `less` disponemos de varios comandos para movernos por el fichero:

- Barra de espacio: página siguiente.

- b: página anterior.
- 100g: va a la línea 100 (o a la que le indiquemos).
- -S: corta o no corta las líneas largas.
- /palabra: busca la cadena de texto que le indiquemos (acepta expresiones regulares).
- n: va a la siguiente palabra que coincide con la búsqueda.
- N: va la palabra anterior que coincide con la búsqueda.
- q: sale del fichero.
- h: ayuda.

El comando WC significa conteo de palabras, y eso es lo que hace (así como contar caracteres y líneas). Por defecto, nos dará un recuento de las tres posibilidades, pero usando opciones de línea de comando, se puede limitar lo que se busca. A veces solo queremos uno de estos valores. Por ejemplo, «`-l`» nos dará solo las líneas, «`-w`» nos dará las palabras y «`-m`» nos dará los caracteres.

```
$ wc hg38_RefSeq.txt
```

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

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

```
172767 hg38_RefSeq.txt
```

La segunda orden imprime por pantalla solo un recuento de líneas, pero la primera orden nos informa del número de líneas, palabras y caracteres que tiene el fichero.

El comando `diff` permite llevar a cabo la comparación línea a línea de dos ficheros de texto. Obviamente, hay formas más sofisticadas de comparar archivos. Sin embargo, esta función es extremadamente útil para confirmar cuando dos ficheros no son idénticos (una de las operaciones más comunes en bioinformática).

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

Por último, se comenta el comando `paste`. Supongamos que tenemos dos ficheros, uno con datos sobre la progresión de la enfermedad de una serie de enfermos y otro con el genotipado de los mismos:

```
$ cat pacientes.txt
```

```
id_paciente,nivel_glucosa

1,190

2,250

3,220

4,260

5,160
```

```
$ cat genotipado.txt
```

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

Se pueden fusionar los dos archivos usando el comando `paste` línea a línea:

```
$ paste -d',' pacientes.txt genotipado.txt
```

```
id_paciente,nivel_colesterol,id_paciente,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 cantidad de información almacenada en cualquier entorno bioinformático es considerable, y a menudo ocupa varios terabytes. Por ejemplo, la secuencia del genoma humano está compuesta por alrededor de tres mil millones de nucleótidos, lo que se traduce en aproximadamente tres gigabytes. Esto implica que a menudo es necesario comprimir directorios enteros, la instrucción `tar` puede crear un paquete único a partir del directorio, que posteriormente puede ser comprimido con `gzip`. La tabla 7 describe los comandos para comprimir/descomprimir más habituales:

Tabla 7. Comandos para acceder a los ficheros.

Comando	Descripción
`tar`	Empaquetar múltiples archivos y directorios
`gzip`	Comprimir y descomprimir archivos
`zmore`	Descomprimir y visualizar un archivo
`zcat`	Descomprimir y volcar un archivo

Fuente: elaboración propia.

Un ejemplo con `tar`:

```
$ tar -cvf backup.tar
```

En el ejemplo anterior de `tar`, las opciones indican que se crea (*c*) un archivo (*f*) llamado *backup.tar* que incluye todos los contenidos del directorio /home/student y sus subdirectorios, y que se muestren mensajes detallados mientras se crea la copia de seguridad (*v*). Debido a que backup.tar es un argumento de la opción *f*, backup.tar debe seguir inmediatamente a la opción. Otras opciones son:

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

La nueva opción (x) indica que se desempaquete el fichero backup.

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

La opción (z) indica que después de que se cree el fichero se comprima.

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

Esta combinación de opciones indica que el fichero se descomprima y se desempaquete.

Los comandos `more` y `cat` poseen una versión especial que integra el comando `gzip` como un filtro adicional. Como resultado, podemos visualizar directamente en el terminal un fichero comprimido:

```
$ 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. Introducción a los entornos de trabajo UNIX

1.9. Acceder al contenido de los ficheros

1.9.2. Edición de archivos con el editor *vim*

Vim, también conocido como *Vi Improved*, es un editor programable altamente potente y versátil que se encuentra presente en todos los sistemas GNU/Linux. Una de sus principales características es que dispone de diferentes modos (normal, visual, *insert*, *command-line*, *select* y *ex*), los cuales se alternan para llevar a cabo distintas operaciones. Esta particularidad lo diferencia de la mayoría de los editores comunes, que suelen tener un solo modo (*insert*), donde se introducen las órdenes mediante combinaciones de teclas o interfaces gráficas.

La totalidad del control de *vim* se realiza a través del teclado desde un terminal, lo que lo hace ideal para ser utilizado sin problemas a través de conexiones remotas, ya que no carga el sistema al no desplegar un entorno gráfico. Aprender a utilizar *vim* es altamente recomendable debido a su capacidad para aumentar la productividad y la eficiencia en la programación.

En este apartado se explican dos modos de *vim*. Modo de inserción (o entrada) y modo de edición. En el modo de inserción, se puede introducir o ingresar contenido en el archivo. En el modo de edición, puede moverse por el archivo, efectuar acciones como eliminar, copiar, buscar y reemplazar, guardar, etc. Un error común es comenzar a ingresar comandos sin volver primero al modo de edición o comenzar a escribir entrada sin entrar primero en el modo de inserción. Si haces cualquiera de estas cosas, generalmente es fácil recuperarse, así que no te preocupes, luego te explico.

¡Empecemos! Será difícil demostrar gran parte de esto, así que, en lugar vuestro, citaré lo que quiero que escribáis y tendréis que intentarlo, y vosotros, ver qué sucede.

Primero, generad un nuevo directorio porque se van a crear algunos archivos y esto los mantendrá fuera de tu material normal. Ahora vamos a editar nuestro primer archivo.

```
$ vi primer-fichero
```

Cuando ejecutáis este comando, se abre el archivo. Si el archivo no existe, lo creará por vosotros y luego lo abrirá (no es necesario ejecutar *touch* archivo antes de editarlo). Una vez que ingreséis la orden *vi* primer fichero, se verá algo como se muestra en la figura 3 (aunque dependiendo del sistema en el que te encuentres, puede verse ligeramente diferente).

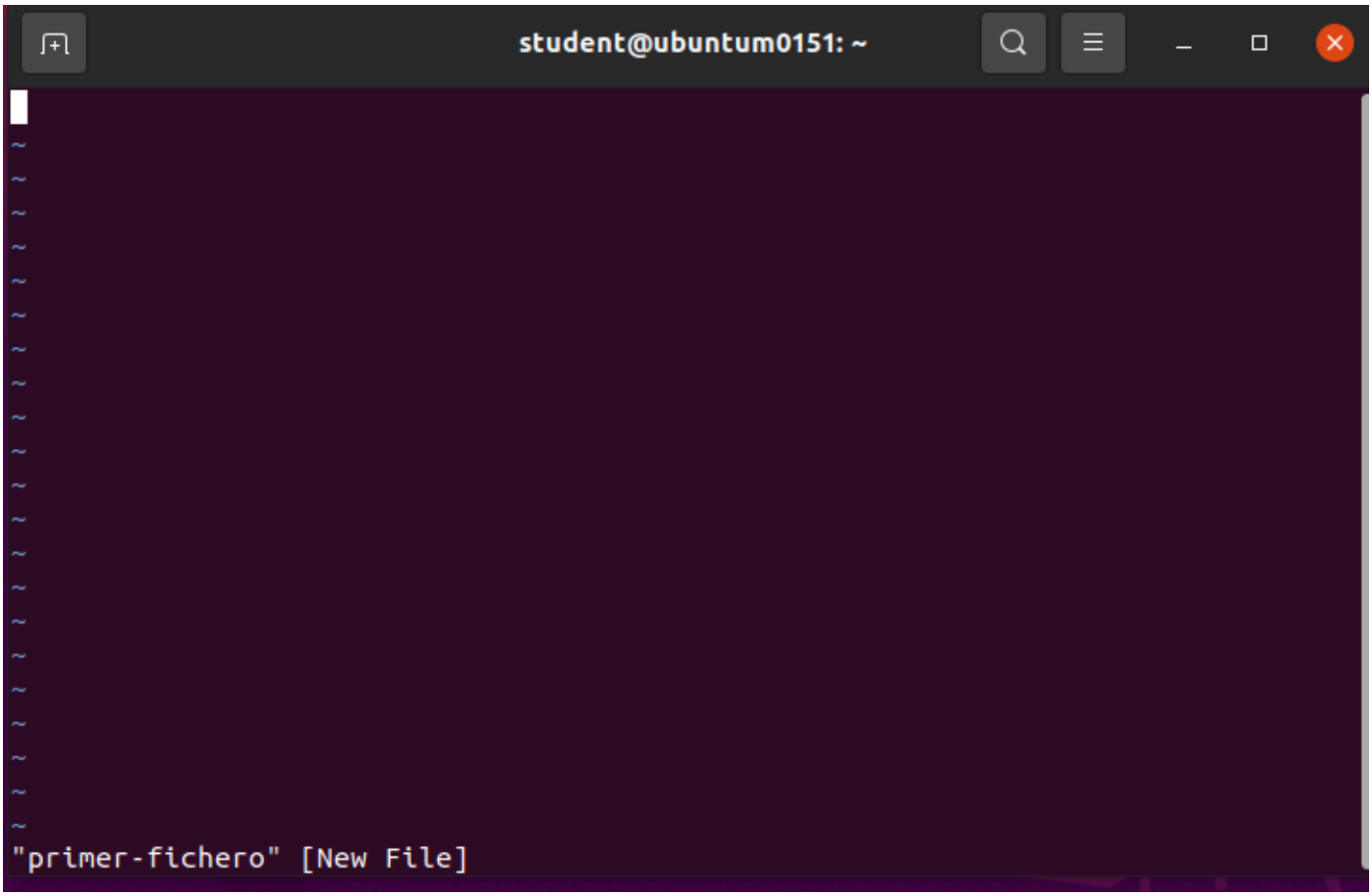


Figura 3. Imagen obtenida al ejecutar la orden *vi* primer-fichero.

Siempre comienzamos en modo de edición, por lo que lo primero que vamos a hacer es cambiar al modo de inserción presionando la letra *i*. Podéis saber cuándo estáis en modo de inserción porque estará indicado en la esquina inferior izquierda. Ahora escribid algunas líneas de texto y presionad *Esc*, lo que os llevará de vuelta al modo de edición.

Save* y *existing

Hay algunas formas de realizar esto. Todas ellas hacen esencialmente lo mismo, así que elegid la que prefiráis. Para todas estas, aseguraos primero de estar en modo de edición.

`:ZZ` (Nota: en mayúsculas). Guardar y salir.

`:q!` Descartar todos los cambios desde el último guardar y salir.

`:w` Guardar el archivo pero no salir.

`:wq` Nuevamente, guardar y salir.

La mayoría de los comandos dentro de *vi* se ejecutan tan pronto como se presiona una secuencia de teclas. Cualquier comando que comience con dos puntos (:) requiere que presionéis <enter> para completar el comando. Guardad y salvad el archivo que tenéis actualmente abierto.

El editor *vi* permite editar archivos. Si quisiéramos, también podríamos usarlo para ver archivos, pero hay otros más comandos que son un poco más convenientes para ese propósito. Probamos *cat*, que en realidad significa *concatenar* y su objetivo principal es unir archivos, pero en su forma más básica es útil para simplemente, como ya se ha visto, ver archivos.

Navegando en un fichero *vi*

Ahora volvamos al archivo que acabamos de crear e ingresemos más contenido. En el modo de inserción, podéis emplear las teclas de flecha para mover el cursor. Ingresad dos párrafos más de contenido y luego presionad *Esc* para volver al modo de edición.

A continuación, se muestran algunos de los muchos comandos que podéis ingresar para moveros por el archivo. Jugad con ellos y observad cómo funcionan.

- Teclas de flecha: mueve el cursor por el archivo.
- *j*, *k*, *h*, *l*: mueve el cursor hacia abajo, arriba, izquierda y derecha (similar a las teclas de flecha).
- ^ (acento circunflejo): mueve el cursor al principio de la línea actual.
- $: mueve el cursor al final de la línea actual.
- nG: mueve el cursor hasta la *n*-ésima línea (por ejemplo, *5G* se mueve a la quinta línea).
- G: mueve el cursor hasta la última línea.
- w: mueve el cursor al principio de la siguiente palabra.
- nw: mueve el cursor *n* palabras hacia delante (por ejemplo, *2w* mueve dos palabras hacia delante).
- b: mueve el cursor al principio de la palabra anterior.
- nb: mueve el cursor hacia atrás *n*
- { : mueve un párrafo hacia atrás, } : mueve un párrafo hacia delante.

Si escribís `:set nu` en el modo de edición dentro de *vi*, se habilitarán los números de línea. Descubriréis que trabajar con archivos es mucho más fácil con los números de línea habilitados.

Eliminando contenido

Acabamos de ver que si queremos movernos dentro de *vi*, hay muchas opciones disponibles. Varias de ellas también nos permiten procesarlas con un número para mover esa cantidad de veces. Eliminar funciona de manera similar al movimiento; de hecho, varios comandos de eliminación nos permiten incorporar un comando de movimiento para definir lo que se va a eliminar. A continuación, se muestran algunas de las muchas formas en que podemos eliminar contenido dentro de *vi*. Jugad con ellas ahora (consultad también la sección siguiente sobre cómo deshacer para poder deshacer vuestras eliminaciones).

- x: elimina un solo carácter.
- nx: elimina *n* caracteres (por ejemplo, *5x* elimina cinco caracteres).
- dd: elimina la línea actual.
- dn: *d* seguido de un comando de movimiento. Elimina hasta donde el comando de movimiento te llevaría (por ejemplo, *d5w* significa eliminar cinco palabras).

Deshacer

Deshacer cambios en *vi* es bastante fácil. Es el carácter *u*.

- u: deshace la última acción (puedes seguir presionando *u* para seguir deshaciendo).
- U (Nota: mayúscula): deshace todos los cambios en la línea actual.

Llevándolo más allá

Ahora podemos insertar contenido en un archivo, movernos por el archivo, eliminar contenido y deshacerlo, luego guardar y salir. Inmediatamente, podéis hacer ediciones básicas en *vi*. Sin embargo, esto es solo la introducción de lo que *vi* puede hacer. Hay más:

- copiar y pegar
- buscar y reemplazar
- *buffers*
- marcadores
- rangos
- configuraciones

Aunque no es posible dedicarle más tiempo, se recomienda un tutorial para aprender a utilizar *vim* o bien acceder directamente a su tutorial interactivo a través del comando *vimtutor* en el terminal.

1. Introducción a los entornos de trabajo UNIX

1.9. Acceder al contenido de los ficheros

1.9.3. Edición de archivos con el editor de flujo *sed* (*stream editor*)

El nombre del comando sed proviene de *stream editor* ('editor de flujo'). Aquí, *stream* se refiere a los datos que se pasan mediante tuberías de *shell*. Por lo tanto, la funcionalidad principal del comando es actuar como un editor de texto para los datos de entrada de la entrada estándar (*stdin*), con la salida estándar (*stdout*) como el destino de salida. También podéis editar la entrada de un archivo y guardar los cambios en el mismo archivo si es necesario.

La sintaxis básica de sed es `sed [options] {commands} {input-file}`

La manera de trabajar de sed es la siguiente. El comando *sed* lee la primera línea del {archivo-de-entrada} y ejecuta los {comandos} en la primera línea. Luego lee la segunda línea del {archivo-de-entrada} y ejecuta los {comandos} en la segunda línea. El comando sed repite este proceso hasta que llega al final del {archivo-de-entrada}.

Editad con *vi* el fichero *test.bed* y realizad cada una de las operaciones que se muestran a continuación para entender la potencia del comando *sed*.

```
$ cat test.bed
```

```
chr1 100 200

chr1 300 500

chr2 240 440

chr2 400 600

chr3 0 150
```

Sustitución

Sustituye todos los *strings* que coincidan con *chr1* por *chr2*

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

```
chr2 100 200

chr2 300 500

chr2 240 440

chr2 400 600

chr3 0 150
```

Sustituye todos los *strings* que coincidan con *chr1* por *chr2* solo si la línea contiene 300

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

```
chr1 100 200

chr2 300 500
```

```
chr2 240 440

chr2 400 600

chr3 0 150
```

Sustituye todos los *strings* que coincidan con *chr1* por *chr2* solo si la línea no contiene 300

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

```
chr2 100 200

chr1 300 500

chr2 240 440

chr2 400 600

chr3 0 150
```

Reemplaza todos los *strings* que coincidan con *chr* si son los primeros caracteres de la línea

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

```
1 100 200

1 300 500

2 240 440

2 400 600

3 0 150
```

Sustituye la primera ocurrencia en cada una de las líneas

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

```
chr1 155 200

chr1 355 500

chr2 240 440

chr2 455 600

chr3 0 150
```

Sustituye todas las ocurrencias que coincidan en el patrón

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

```
chr1 155 255

chr1 355 555

chr2 240 440

chr2 455 655

chr3 0 150
```

Imprime la línea donde coincida la sustitución

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

```
chr1 155  200

chr1 355  500

chr2 455  600
```

Se pueden realizar sustituciones con diferentes *flags* simultáneamente

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

```
chr1 155  255

chr1 355  555

chr2 455  655
```

Eliminar

Elimina la segunda línea del fichero

```
$ 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 desde que encuentres el *string chr1* a la línea 4

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

```
chr3 0 150
```

Elimina desde la primera línea que encuentres con *chr1* hasta la primera vez que encuentres *chr2*

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

```
chr2 400 600

chr3 0 150
```

Edición implícita

El fichero de entrada y de salida es el mismo. Hazlo cuando estés seguro del cambio.

```
$ 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
```

¡Porque el fichero original ha cambiado!

Sustituciones *regexp*

Las expresiones regulares son muy útiles, y vale la pena tomarse el tiempo para aprender los conceptos básicos. Se pueden ampliar conocimientos utilizando herramientas en línea para construir y probar expresiones regulares (por ejemplo, https://regex101.com/ A continuación se enumeran algunas de las importantes:

Anclas:

^ restringe la coincidencia al inicio de la cadena.

$ restringe la coincidencia al final de la cadena.

Metacaracteres y cuantificadores

- . coincide con cualquier carácter, incluyendo el carácter de nueva línea.
- ? coincide 0 o 1 vez.
- * coincide 0 o más veces.
- + coincide 1 o más veces.
- {m,n} coincide de *m* a *n* veces.

- `{m,}` coincide al menos *m* veces.
- `{,n}` coincide hasta *n* veces (incluyendo 0 veces).
- `{n}` coincide exactamente *n*.

Clases de caracteres:

- `[set123]` coincide con cualquiera de estos caracteres una vez.
- `[^set123]` coincide excepto con cualquiera de estos caracteres una vez.
- `[3-7AM-X]` rango de caracteres desde 3 hasta 7, A, otro rango desde M hasta X.
- `[[:digit:]]` similar a `[0-9]` `[[:alnum:]_]` similar a `\w`

Elimina la primera columna: . define cualquier carácter después de *chr*

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

```
       100  200

       300  500

       240  440

       400  600

       0    150
```

Elimina todos los ceros presentes en el fichero

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

```
chr1 1    2

chr1 3    5

chr2 24   44

chr2 4    6

chr3      15
```

Elimina todos los números entre el 1 y el 3

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

```
chr  00   00

chr  00   500

chr  40   440

chr  400  600
```

```
chr  0    50
```

Sustituye el *string ch* por nada

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

```
h1   100  200

h1   300  500

h2   240  440

h2   400  600

h3   0    150
```

Sustituye cualquier letra entre la *a* y la *z* por nada

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

```
1    100  200

1    300  500

2    240  440

2    400  600

3              0     150
```

sustituye por la cadena que coincide con el patrón

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

```
[chr1] 100 200

[chr1] 300 500

[chr2] 240 440

[chr2] 400 600

[chr3] 0 150
```

Sustituye todas las ocurrencias que encajan en el patrón y escribe al final de la línea '+'

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

```
chr1 155 255 +

chr1 355 555 +
```

```
chr2 240 440 +

chr2 455 655 +

chr3 0 150 +
```

Sustituye la ocurrencia que coincide exactamente con 2 ceros

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

```
chr1 1match    2match

chr1 3match    5match

chr2 240  440

chr2 4match    6match

chr3 0    150
```

Sustituye la ocurrencia que coincide con 1 o 2 ceros

```
$ 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 todas las líneas en blanco del fichero

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

1. Introducción a los entornos de trabajo UNIX

1.10. Gestión básica de procesos

1.10.1 Introducción

Una de las características distintivas de Gnu/Linux es su enfoque en otorgar al usuario un amplio control sobre los procesos en ejecución en el sistema. En este apartado, nos centraremos en explorar las órdenes de gestión de procesos que ofrece el *shell* de Gnu/Linux. Esto nos permitirá comprender cómo funciona el control de procesos en Gnu/Linux desde un terminal y cómo se pueden aprovechar estas herramientas para mejorar la eficiencia y productividad en la administración de sistemas. En la tabla 8 se enumeran los comandos más habituales.

Tabla 8. Comandos para el control de procesos.

Comando	Resultado
`sleep`	Suspende la ejecución actual de un comando por un intervalo de tiempo
`ps`	*process status*. Imprime el estado de los procesos
`fg`	*foreground* o primer plano
`bg`	*background* o segundo plano
`jobs`	Imprime los trabajos activos en la *shell*
`top`	*table of processes*. Imprime una vista en tiempo real de los procesos en ejecución en Linux y también muestra las tareas administradas por el *kernel*. Además, el comando proporciona un resumen de información del sistema que muestra la utilización de recursos, incluyendo el uso de CPU y memoria
`kill`	Detiene el proceso suministrado el PID
`nice`	Permite ejecutar un comando con una prioridad menor a la prioridad normal del comando
`renice`	modifica el valor «nice» de uno o más procesos en ejecución

Fuente: elaboración propia

Desde la línea de comandos, el comando `ps` es el más antiguo y común para listar los procesos que se están ejecutando en tu sistema. El comando `top` proporciona una forma más orientada a la pantalla de listar los procesos y también se puede usar para cambiar el estado de los procesos.

1. Introducción a los entornos de trabajo UNIX

1.10. Gestión básica de procesos

1.10.2. Listar procesos con «ps»

La utilidad más común para comprobar los procesos que se están ejecutando es el comando `ps`. Usadlo para ver qué programas se están ejecutando, los recursos que están utilizando y quién los está ejecutando. El siguiente es un ejemplo del comando `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 este ejemplo, la opción *u* solicita que se muestren los nombres de usuario, así como otra información como el tiempo de inicio del proceso y el uso de memoria y CPU para los procesos asociados con el usuario actual. Los procesos que se muestran están asociados con el terminal actual (*tty1*). El primer proceso muestra que el usuario llamado *student* abrió una *shell bash* después de iniciar sesión. El siguiente proceso muestra que *student* ha ejecutado el comando `ps -u`. El dispositivo terminal *tty1* se está empleando para la sesión de inicio de sesión. La columna STAT representa el estado del proceso, con *R* indicando un proceso que se está ejecutando actualmente y *S* representando un proceso que está durmiendo.

Para paginar a través de todos los procesos en ejecución en vuestro sistema Gnu/Linux para el usuario actual, agregad el símbolo *pipe* (|) y el comando `less` al comando `ps -ux`:

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

Para paginar a través de todos los procesos en ejecución para todos los usuarios en vuestro sistema, usad el comando `ps -aux` de la siguiente manera:

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

El símbolo *pipe* (|) os permite dirigir la salida de un comando para que sea la entrada del siguiente comando. En este ejemplo, la salida del comando `ps` (una lista de procesos) se dirige al comando `less`, que os permite paginar esa información. Usad la barra espaciadora para avanzar página por página y escribid *q* para terminar la lista. También podéis utilizar las teclas de flecha para avanzar una línea a la vez a través de la salida.

Consultad la página de manual de `ps` para obtener información sobre otras columnas de información que podéis mostrar y ordenar por ellas.

1. Introducción a los entornos de trabajo UNIX

1.10. Gestión básica de procesos

1.10.3. Listando y cambiando procesos con *top*

El comando `top` proporciona una forma orientada a la pantalla de mostrar los procesos que se están ejecutando en su sistema. Con `top`, por defecto, se muestran los procesos en función del tiempo de CPU que están consumiendo actualmente. Sin embargo, también se pueden ordenar por otras columnas. Por otra parte, si se identifica un proceso problemático, también puede usar `top` para matar (terminar completamente, en inglés *kill*) o `renice` (en castellano, *re-priorizar*) ese proceso. Si desea poder matar o `renice` procesos, debe ejecutar `top` como usuario *root*. Si solo desea mostrar procesos, y posiblemente matar o cambiar sus propios procesos, puede hacerlo como usuario regular.

La figura 4 muestra un ejemplo de la ventana `top`. La información general sobre su sistema aparece en la parte superior de la salida de `top`, seguida de información sobre cada proceso en ejecución. En la parte superior, puede ver cuánto tiempo ha estado activo el sistema, cuántos usuarios están actualmente conectados al sistema y cuánta demanda ha habido en el sistema en los últimos 1, 5 y 10 minutos. Otra información general incluye cuántos procesos (tareas) se están ejecutando actualmente, cuánta CPU se está utilizando y cuánta memoria RAM y *swap* están disponibles y se están utilizando. Después de la información general, hay listados de cada proceso, ordenados por el porcentaje de la CPU que está siendo utilizado por cada proceso. Toda esta información se vuelve a mostrar cada 5 segundos, este tiempo está definido de forma predeterminada.

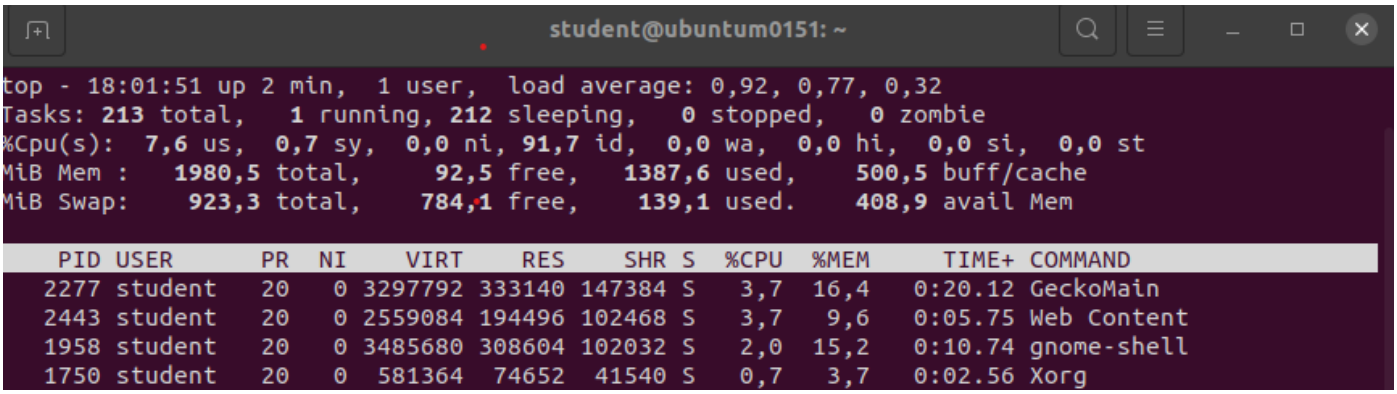


Figura 4. Imagen de la pantalla al ejecutar la orden `top`.

La siguiente lista incluye acciones que se pueden realizar cuando se está ejecutando `top` para mostrar información de diferentes formas y modificar procesos en ejecución:

- Presionad *h* para ver las opciones de ayuda, y luego presione cualquier tecla para volver a la pantalla `top`.
- *Presionad M* para ordenar por uso de memoria en lugar de CPU, y luego presione *P* para volver a ordenar por CPU.
- Presionad el número 1 para alternar entre mostrar el uso de la CPU de todas las CPU, si tiene más de una CPU en su sistema.
- *Presionad R* para ordenar su salida en orden inverso.
- *Presionad u* e ingrese un nombre de usuario para mostrar solo los procesos de un usuario en particular.

Una práctica común es usar `top` para encontrar procesos que estén consumiendo demasiada memoria o potencia de procesamiento y luego actuar sobre esos procesos de alguna manera. Un proceso que consume demasiada memoria puede ser matado, o un proceso que consume demasiada CPU puede ser `renice` para darle menos prioridad a los procesadores.

- Matar un proceso: tomad nota del ID de proceso del proceso que deseáis matar y presionad *k*. Escribid 15 para terminar limpiamente o 9 para matar el proceso directamente. Desde el terminal también se puede matar procesos.

```
$ kill 2277
```

```
$ kill -15 2277
```

```
$ kill -SIGKILL 2277
```

Cuando el *kernel* de Gnu/Linux intenta decidir qué procesos en ejecución tienen acceso a las CPU de su sistema, una de las cosas que tiene en cuenta es el valor *nice* establecido en el proceso. Cada proceso en ejecución en el sistema tiene un valor *nice* entre -20 y 19. De manera predeterminada, el valor *nice* se establece en 0. Aquí hay algunos datos sobre los valores *nice*:

- Cuanto más bajo sea el valor *nice*, más acceso a las CPU tendrá el proceso.
- El usuario *root* puede establecer el valor *nice* en cualquier proceso en cualquier valor válido, hacia arriba o hacia abajo.
- Un usuario *estándar* solo puede establecer el valor *nice* en los propios procesos del usuario, solo pueden ser positivos y el nuevo valor de *nice* siempre ha de ser mayor, no menor, al predeterminado.

Si volvemos a ejecutar el comando `top`,

- `Renice` un proceso: tomad nota del ID de proceso del proceso que deseáis `renice` y presionad *r*. Cuando aparezca el mensaje «PID to renice:», escribid el ID del proceso que desea *renicear*. Cuando se solicite «Renice PID to value:», escribid un número del 0 al 20. Desde el terminal se pueden *nice/renice* diferentes procesos (en este caso lo realiza *root*).

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

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

1. Introducción a los entornos de trabajo UNIX

1.10. Gestión básica de procesos

1.10.4. Gestión de procesos en segundo plano y primer plano

Si estáis trabajando con Gnu/Linux a través de una red o desde un terminal con una pantalla que solo permite entrada de texto sin soporte gráfico, es posible que solo tengáis acceso a la *shell*. Si estáis acostumbrados a trabajar en un entorno gráfico en el que puedes tener varios programas abiertos al mismo tiempo y cambiar entre ellos, la interfaz de la *shell* puede parecer limitada.

Sin embargo, aunque la *shell bash* no tiene una interfaz gráfica para ejecutar varios programas al mismo tiempo, sí permite mover los programas activos entre el fondo y el primer plano. Esto permite tener varios procesos en ejecución y seleccionar el que queremos manejar en ese momento.

Podemos poner un programa en segundo plano de varias maneras. Una forma es agregar el símbolo `&` al final de la línea de comando cuando lo ejecutamos por primera vez o podemos usar el comando `at` para ejecutar comandos de manera que no estén conectados a la *shell*.

Para detener un comando en ejecución y ponerlo en segundo plano, presionad Ctrl+Z. Después de detener el comando, podéis volver a ejecutarlo en primer plano con el comando `fg` o iniciarlo en segundo plano con el comando `bg`. Es importante tener en cuenta que cualquier comando en ejecución en segundo plano puede generar salida durante los comandos que ejecutéis posteriormente desde esa *shell*. Por ejemplo, si aparece salida de un comando en segundo plano durante una sesión de *vi*, simplemente presionad Ctrl+L para refrescar la pantalla y deshaceros de la salida.

Si tenéis programas que deseáis ejecutar mientras continuáis trabajando en la *shell*, podéis colocar los programas en segundo plano. Para colocar un programa en segundo plano en el momento en que se ejecuta el programa, escribid un *ampersand* (&) al final de la línea de comando, así:

```
$ find /usr > /tmp/fichero-usuarios &
```

```
[3] 15971
```

Este ejemplo de comando encuentra todos los archivos en el sistema Gnu/Linux (a partir de /usr), imprime esos nombres de archivo y los coloca en el archivo /tmp/fichero-usuarios. El *ampersand* (&) ejecuta esa línea de comando en segundo plano. Observad que se muestra el número de trabajo [3] y el número de identificación de proceso, 15971, cuando se lanza el comando. Para comprobar qué comandos tenéis en ejecución en segundo plano, usad el comando `jobs`, así:

```
$ jobs
```

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

[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
```

Se pueden traer cualquiera de los comandos de la lista de trabajos al primer plano. Para hacer referencia a un trabajo en segundo plano (para cancelarlo o llevarlo al primer plano), usad un signo de porcentaje (%) seguido del número de trabajo. Para editar el archivo *unficherocualquiera* nuevamente, escribid:

```
$ fg %1
```

Como resultado, el comando *vi* y el terminal que lo contiene se abre de nuevo. Todo el texto es como estaba cuando detuvisteis el trabajo de *vi*. Antes de poner un procesador de texto, un procesador de palabras u otro programa similar en segundo plano,

asegúraos de guardar vuestro archivo. Es fácil olvidar que tenemos un programa en segundo plano y perderíais vuestros datos si cerráis la sesión o reiniciáis el ordenador.

Si un comando se detiene, puedes hacer que se ejecute de nuevo en segundo plano utilizando el comando `bg`. Por ejemplo, tomad el trabajo número 5 de la lista de trabajos del ejemplo anterior. Escribid lo siguiente:

```
$ bg %5
```

Después de eso, el trabajo se ejecutará en segundo plano. Su entrada en la lista de trabajos aparecerá así:

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

1. Introducción a los entornos de trabajo UNIX

1.11. Buscar, ordenar y asociar ficheros

1.11.1. Introducción

Cuando se trabaja con datos bioinformáticos, es común encontrar ficheros de texto organizados en formato tabular, lo que implica que la información se distribuye en una matriz de filas y columnas delimitadas por espacios o caracteres tabuladores. Cada línea representa un registro, mientras que cada columna contiene valores específicos de los atributos que lo caracterizan. Esta estructura de campos facilita la realización de cálculos sistemáticos sobre el contenido del fichero en cuestión.

Generalmente, al trabajar con ficheros tabulados en el análisis bioinformático, se adopta como unidad elemental de trabajo la línea o registro. Bajo este paradigma, se pueden realizar varios tipos de operaciones, como buscar y separar del fichero las líneas que contienen un patrón de texto determinado, alterar el orden de las líneas según los valores de alguno de los atributos o filtrar registros duplicados. En ciertas circunstancias, incluso es posible identificar aquellos registros de dos ficheros de texto distintos que poseen el mismo valor para un determinado atributo.

Estas operaciones aplicadas sobre los ficheros de anotaciones son útiles en el análisis bioinformático para llevar a cabo fácilmente el recuento de diferentes características biológicas, como los genes codificados en el interior de los genomas. En la tabla 9 de se describen los comandos más útiles para buscar y ordenar ficheros, así como asociarse entre ellos.

Tabla 9. Descripción de comandos.

Comando	Descripción
`grep`	Busca líneas que coincidan con una expresión regular
`cut`	`cut` es un programa útil si el contenido está separado en campos (columnas) y solo se desea obtener ciertos campos
`sort`	`sort` ordenará su entrada, de manera simple y sencilla. Por defecto, ordenará alfabéticamente, pero existen muchas opciones disponibles para modificar el mecanismo de ordenamiento. Asegúrate de revisar la página de manual para ver todo lo que puede hacer
`uniq`	Elimina las líneas duplicadas sucesivas (usar `sort` antes de utilizar `uniq`)
`join`	Permite unir dos ficheros de texto en uno usando una columna como clave común

Fuente: elaboración propia.

1. Introducción a los entornos de trabajo UNIX

1.11. Buscar, ordenar y asociar ficheros

1.11.2. «*grep*»

En Wikipedia puedes leer que `grep` es una utilidad de línea de comando para buscar conjuntos de datos de texto plano en busca de líneas que coincidan con una expresión regular. Su nombre proviene del comando `ed g/re/p` (***g**lobally search a **r**egular **e**xpression and **p**rint*, en castellano *búsqueda global de una expresión regular e imprimir*), que tiene el mismo efecto.

El comando `grep` tiene muchas y variadas características, tanto es así que hay libros específicos al respecto. El uso más común es filtrar líneas de entrada usando expresiones regulares (en inglés, *regexp*).

Las opciones del comando `grep` comúnmente utilizadas se muestran a continuación. Los ejemplos se discutirán en secciones posteriores.

- `-i` ignora la distinción entre mayúsculas y minúsculas al hacer coincidencias.
- `-v` imprime solo las líneas que no coinciden.
- `-n` agrega un prefijo de números de línea a las líneas de salida.
- `-c` muestra solo el recuento de líneas de salida.
- `-l` imprime solo los nombres de archivo que coinciden con la expresión dada.
- `-L` imprime los nombres de archivo que no coinciden con el patrón.
- `-w` hace coincidir el patrón solamente con palabras completas.
- `-x` hace coincidir el patrón solamente con líneas completas.
- `-F` interpreta el patrón como una cadena fija (es decir, no como una expresión regular).
- `-o` imprime solamente las partes coincidentes.
- `-A N` imprime la línea coincidente y N líneas después de la línea coincidente.
- `-B N` imprime la línea coincidente y N líneas antes de la línea coincidente.
- `-C N` imprime la línea coincidente y N líneas antes y después de la línea coincidente.
- `-m N` imprime un máximo de N líneas coincidentes.
- `-q` sin salida estándar, sale inmediatamente si se encuentra una coincidencia, útil en *scripts*.
- `-s` suprime los mensajes de error, útil en *scripts*.
- `-r` busca recursivamente todos los archivos en las carpetas de entrada especificadas (por defecto busca en el directorio actual).
- `-R` como `-r`, pero también sigue los enlaces simbólicos.
- `--color=auto` resalta las porciones coincidentes, los nombres de archivo, los números de línea, etc. usando colores.

Búsqueda literal

Los siguientes ejemplos también serían adecuados con la opción `-F`, ya que no utilizan expresiones regulares. El comando `grep` está lo suficientemente bien diseñado como para hacer lo correcto en tales casos.

Imprime líneas que contengan el `string 'an'`. La opción \n genera una nueva línea

```
$ printf 'manzana\nbanana\nmango\nfigura\ntango\n' | grep 'an'
```

```
banana

mango

tango
```

Imprime coincidencias sin distinción entre mayúsculas y minúsculas

```
$ printf 'Buho\nlagartija\buHoNero\ntres buhos\n' | grep -i 'buho'
```

```
Buho

buHoNero

tres buhos
```

Imprime coincidencias de palabras completas

```
$ printf 'par basico\nparidad\n3-par' | grep -w 'par'
```

```
par basico

3-par
```

Imprime coincidencias de líneas vacías y las cuenta

```
$ printf 'cielo\n\ntierra\n\n\n\ninfierno\n' | grep -cx ''
```

```
4
```

Imprime coincidencias en una línea y las siguientes dos líneas (A, after)

```
$ printf 'red\ncactus\nsuculenta\npetunia\nesqueje' | grep -A2 'cactus'
```

```
cactus

suculenta

petunia
```

Expresiones regulares

De forma predeterminada, *grep* trata el patrón de búsqueda como una expresión regular básica (en inglés, *Basic Regular Expression*: BRE).

- `-G` se puede utilizar para especificar explícitamente que se necesita BRE.
- `-E` habilitará expresiones regulares extendidas (en inglés, *Extended Regular Expression*: ERE). **En GNU *grep*, BRE y ERE solo difieren en cómo se especifican los metacaracteres; no hay diferencia en las características.**

- `-F` hará que los patrones de búsqueda se traten literalmente.
- `-P` habilitará expresiones regulares compatibles con Perl, si está disponible (PCRE, por sus siglas en inglés).

Las siguientes líneas son las referencias cuando se quieren usar expresiones regulares extendidas. Existen de varias clases:

Anclas

- ^ restringe la coincidencia al inicio de la cadena.
- $ restringe la coincidencia al final de la cadena.
- < restringe la coincidencia al inicio de palabra.
- > restringe la coincidencia al final de palabra.
- \b restringe la coincidencia al inicio/final de palabras.
- \B coincide donde \b no coincide.

Metacaracteres y cuantificadores

- . coincide con cualquier carácter, incluyendo el carácter de nueva línea.
- ¿ coincide 0 o 1 vez.
- * coincide 0 o más veces.
- + coincide 1 o más veces.
- {m,n} coincide de *m* a *n*
- {m,} coincide al menos *m*
- {,n} coincide hasta *n* veces (incluyendo 0 veces).
- {n} coincide exactamente *n*

Clases de caracteres

- [set123] coincide con cualquiera de estos caracteres una vez.
- [^set123] coincide, excepto con cualquiera de estos caracteres, una vez.
- [3-7AM-X] rango de caracteres desde 3 hasta *7*, *A*, otro rango desde *M* hasta *X*.
- \w similar a [a-zA-Z0-9_] para coincidir con caracteres de palabras.
- \s similar a [\t\n\r\f\v] para coincidir con caracteres de espacio en blanco.
- \W coincide con caracteres que no son de palabras.
- \S coincide con caracteres que no son de espacio en blanco.
- [[:digit:]] similar a [0-9] [[:alnum:]_] similar a \w

Se recomienda ver el manual del comando `grep` para ser conscientes de la versatilidad de este comando.

Alternancia y agrupación

- Pat1|pat2|pat3 coincide con *pat1* o *pat2* o *pat3*.
- () agrupa patrones, a(b|c)d es lo mismo que abd|acd. También sirve como grupo de captura.

- \N referencia hacia atrás, proporciona la porción coincidente del N-ésimo grupo de captura.
- \1 referencia hacia atrás al primer grupo de captura.
- \2 referencia hacia atrás al segundo grupo de captura y así sucesivamente hasta \9.

En el manual de `grep` se citan las diferencias entre BRE y ERE. En las expresiones regulares básicas, los metacaracteres ?, +, {, |, (y) pierden su significado especial; en su lugar, se utilizan las versiones con barra invertida \?,\ +, \{,\ |, \(y \).

Imprime la coincidencia cuando el final de la línea termina con *–ar*

```
$ printf 'pedalear\ncorrer\nescribir\namar' | grep 'ar$'
```

```
pedalear

amar
```

Imprime las palabras que comienzan con `'par'`, terminan con `'t'`, y pueden tener `'en'` o `'ro'` en el medio, sin importar si están en mayúsculas o minúsculas.

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

```
part

PARROT

Parent
```

Imprime coincidencia cuando hay texto acotado

```
$ echo 'Disfruto practicando ejercicios de "bash" y "R"' | grep -oE '"
[^"]+"'
```

```
"bash"

"R"
```

Líneas de 8 caracteres que tienen las mismas 3 letras minúsculas al principio y al final

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

```
mesdames

respires

restores

testates
```

1. Introducción a los entornos de trabajo UNIX

1.11. Buscar, ordenar y asociar ficheros

1.11.3. «*cut*»

Si el archivo está organizado en campos, como en el caso de la tabla que estamos usando, podemos seleccionar un campo específico utilizando el comando `cut`. Para refinar nuestra búsqueda, podemos usar `cut` para extraer solo el nombre de transcritos, como en el siguiente ejemplo

```
$ 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
```

Con el parámetro *-f* le indicamos la lista de campos (*fields*) que queremos seleccionar.

Para indicar los campos que queremos seleccionar:

- `N`: el campo *N* (por ejemplo, cut -f 3 file1).
- `N-`: desde el campo *N* hasta el final (por ejemplo, cut -f 3- file1).
- `N-M`: desde el campo *N* al *M* (por ejemplo, cut -f 3-6 file1).
- `-M`: desde el primer campo al campo *M* (por ejemplo, cut -f -3 file1).
- `N,M`: los campos indicados (por ejemplo, cut -f 3,6,8 file1).

Así podríamos seleccionar los campos del 3 al 5 y del 8 al 10:

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

El comando `cut` asume que los campos en el fichero están divididos por tabuladores. Pero podríamos indicarle que los campos están divididos de otro modo, por ejemplo, por comas:

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

1. Introducción a los entornos de trabajo UNIX

1.11. Buscar, ordenar y asociar ficheros

1.11.4. «*sort*»

Como su nombre indica, en inglés, este comando se utiliza para ordenar el contenido de los archivos de entrada. ¿Orden alfabético y orden numérico? Posible. ¿Qué tal ordenar una columna específica? Posible. ¿Orden de clasificación múltiple prioritario? Posible. ¿Aleatorio? ¿Único? Muchas características son compatibles con este poderoso comando.

Se muestran a continuación las opciones de uso común. Los ejemplos se discutirán en secciones posteriores.

- `-n` ordenar numéricamente.
- `-g` ordenación numérica general.
- `-V` ordenar por versión (consciente de los números dentro del texto).
- `-h` ordenar números legibles para humanos (por ejemplo: 4K, 3M, 12G, etc.).
- `-k` ordenar mediante clave (ordenación de columna). Similar a -f del comando *cut*.
- `-t` separador de campo de un solo byte de carácter (el valor predeterminado es la transición de no espacio en blanco a espacio en blanco).
- `-u` ordenar de forma única.
- `-R` ordenar aleatoriamente.
- `-r` invertir la salida de ordenación.
- `-o` redirigir el resultado ordenado al archivo especificado.

De forma predeterminada, `sort` ordena la entrada lexicográficamente en orden ascendente. Puede utilizar la opción -r para invertir los resultados.

Orden por defecto

```
$ printf 'banana\ncereza\nfresa' | sort
```

```
banana

cerera

fresa
```

Ordena e imprime en orden inverso

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

```
residencia

hotel

esperanza
```

Ordena numéricamente e imprime

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

```
2

3

20
```

Ordena los números a la manera humana

```
$ sort -hr fichero.txt
```

```
1.4G    genomica

316M    proteomica

746K    wdl.log

104K    gromacs.log

20K     sample.txt
```

Ordena teniendo en cuenta la versión

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

```
3m20.058s

3m42.833s

4m3.083s

4m11.130s

5m35.363s
```

Genera el siguiente fichero

```
$ cat modelo.txt
```

```
GWAS    50

NGS     5

RNA-seq    2

ChIP-Seq    25

WES   10
```

Ordena teniendo en cuenta los números de la segunda columna

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

```
RNA-seq   2

NGS    5

WES  10

ChIP-Seq   25

GWAS   50
```

Por ejemplo, podemos ordenar los transcriptos de la siguiente manera:

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

En el ejemplo anterior, al utilizar el comando `sort`, notamos que la lista resultante contenía transcritos repetidos. Para eliminar líneas duplicadas consecutivas, podemos emplear el comando `uniq`. Es importante recordar que, para una eliminación completa de duplicados, es necesario ordenar el archivo con`sort` antes de utilizar `uniq`:

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

1. Introducción a los entornos de trabajo UNIX

1.11. Buscar, ordenar y asociar ficheros

1.11.5. «*uniq*»

Este comando te ayuda a identificar y eliminar duplicados. Se ha de utilizar con entradas ordenadas, ya que la comparación se realiza solo entre líneas adyacentes, lo que significa que primero hay que utilizar el comando `sort` antes que el comando `uniq`. Se muestran a continuación las opciones de uso común.

- `-u` muestra solo las entradas únicas.
- `-d` muestra solo las entradas duplicadas.
- `-D` muestra todas las copias de duplicados.
- `-c` prefijo de conteo.
- `-i` ignora mayúsculas y minúsculas al determinar duplicados.
- `-f` omite los primeros *N* campos la separación de campo; se basa solo en uno o más caracteres de espacio/tabulación.
- `-s` omite los primeros *N*
- `-w` restringe la comparación a los primeros *N*

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

```
$ cat sistemas.txt
```

```
GWAS

RNA-Seq

WES

GWAS

ChIP-Seq

RNA-Seq

NGS

RNA-Seq
```

Ordena y elimina los duplicados

```
$ sort sistemas.txt | uniq
```

```
ChIP-Seq

GWAS

NGS
```

```
RNA-Seq

WES
```

Ordena e imprime únicamente las ocurrencias no repetidas

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

```
ChIP-Seq

NGS

WES
```

Ordena e imprime únicamente las ocurrencias repetidas

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

```
GWAS

RNA-Seq
```

Ordena por número de ocurrencia

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

```
      3 RNA-Seq

      2 GWAS

      1 WES

      1 NGS

      1 ChIP-Seq
```

Si continuamos con el fichero hg38_RefSeq para eliminar líneas duplicadas consecutivas, se utiliza el comando `uniq`. Es importante recordar que, para una eliminación completa de duplicados, es necesario ordenar el archivo con `sort` antes de utilizar `uniq`:

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

1. Introducción a los entornos de trabajo UNIX

1.11. Buscar, ordenar y asociar ficheros

1.11.6. «*join*»

El comando `join` permite unir dos ficheros de texto en uno usando una columna como clave común. Por defecto, `join` asume que el separador de campos es el espacio. El comando `join` es parecido al comando `paste` donde la columna común, y que se utiliza como referencia entre ambas tablas, no queda duplicada y no requiere que un elemento esté en los dos archivos. Por otra parte, lo que sí requiere `join` es que ambos archivos estén ordenados por la columna que se quiere utilizar como clave. Imaginemos que tenemos los dos ficheros siguientes:

```
$ cat file1.txt
```

```
num id atributo

1 CDKL3 chr5

2 CLN8 chr8

5 SOCS2 chr4
```

```
$ cat file2.txt
```

```
num id atributo

1 AGRN +

3 CDKL3 +

5 CLN8 -

9 FCHO +
```

El comando `join` nos permite unir estas dos tablas en una sola utilizando el campo `num` (la primera columna de cada uno de los ficheros) como la clave de unión:

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

```
num id atributo id atributo

1 CDKL3 chr5 AGRN +

5 SOCS2 chr4 CLN8 -
```

Por defecto, `join` asume que la clave de unión es la primera columna, pero esto se puede modificar:

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

```
id num atributo num atributo

CDKL3 1 chr5 3 +
```

```
CLN8 2 chr8 5 -
```

1. Introducción a los entornos de trabajo UNIX

1.12. Combinación de comandos

Hasta ahora habéis visto numerosos ejemplos de comandos ejecutados individualmente en el terminal. Sin embargo, el intérprete de comandos tiene un gran potencial que permite implementar protocolos de trabajo más sofisticados de manera relativamente sencilla. Un caso paradigmático es la canalización de los resultados de los comandos. En general, la mayoría de los comandos ejecutados en Gnu/Linux generan más información de la que puede aparecer en pantalla físicamente. Por lo tanto, es preferible almacenar los resultados dentro de un archivo para su posterior análisis y visualización. También es posible que nos interese convertir el flujo de datos generado por un proceso emisor de información en la entrada de otro, sin necesidad de generar archivos intermedios. El intérprete de comandos proporciona una serie de funcionalidades para implementar todas estas operaciones relacionadas con la comunicación y el almacenamiento de resultados. En la tabla 10 se muestran las funcionalidades más utilizadas para combinar comandos.

Tabla 10. Combinación de comandos.

Comando	Descripción
proceso > fichero	Redirección de salida
proceso >> fichero	Redirección de salida sin sobrescribir
proceso 2> fichero	Redirección del canal error
proceso < fichero	Redirección de la entrada
proceso 1 | proceso 2	Comunicación de dos procesos

Fuente: elaboración propia.

El proceso de almacenamiento de datos generados por un proceso en un archivo se conoce como *redirección*. Un proceso tiene permiso de escritura en dos canales del terminal: salida y error. Los datos generados normalmente por el proceso se transmiten a través del canal de salida, mientras que los errores de ejecución se comunican mediante el canal de error. De manera predeterminada, ambos canales se redirigen a la pantalla (terminal). Si el usuario desea redirigir una de estas dos salidas, debe indicarlo explícitamente al final del comando. El símbolo > indica la redirección del canal de salida, mientras que para redirigir el canal de error se debe usar la construcción 2>. En ambos casos, los datos se almacenan en un archivo que se puede consultar en cualquier momento sin necesidad de volver a ejecutar el comando.

Cada comando tiene asociada una entrada estándar *stdin* (0) (por defecto, el teclado), una salida estándar *stdout* (1) (por defecto la consola) y una salida de errores estándar *stderr* (2) (por defecto también la consola). Por ejemplo, el comando `cat`, si no recibe argumentos, lee del teclado por la entrada estándar y lo pasa a la salida estándar.

```
$ cat
```

```
Primera secuencia por estudiar

Segunda secuencia por estudiar

^D
```

El final del *stream* se lo hemos asignado desde el teclado con la combinación de teclas Ctrl+D.

Se puede cambiar la entrada estándar de `cat` para que lea de un fichero. También serviría para la mayoría de los comandos: `cat, grep, cut, sed…`

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

```
Name of the system
```

```
Nucleotides

...
```

El operador de redirección de salida > permite cambiar la salida estándar de un comando

```
$ cal > calendario
```

Esta orden envía el calendario actual al fichero *calendario*. En este caso, el *stdout* del comando `cal` se redirige a un fichero llamado *calendario* que contendrá lo que se vería en la pantalla al ejecutar `cal`.

Para enviar la salida *stderr* de un programa a un fichero, se ejecuta

```
$ grep -ioE 'par(en|ro)?t' datos01 2> error.txt.
```

Para enviar la *stdout* a la *stderr*, escribimos en pantalla

```
$ grep -ioE 'par(en|ro)?t' datos01 1>&2
```

Y a la inversa, simplemente intercambiando el 1 por 2, se ejecuta

```
$ grep -ioE 'par(en|ro)?t' datos01 2>&1.
```

Si se quiere que la ejecución de un comando no genere actividad por pantalla, lo que se denomina *ejecución silenciosa*, solamente debemos redirigir todas sus salidas a /dev/null. Por ejemplo, pensando en utilizar el comando `find` (en castellano, *buscar*), que queremos utilizar para que borre todos los archivos terminados en *.mov* del sistema:

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

Pero se debe ir con cuidado y estar muy seguro, ya que no tendremos ninguna salida por pantalla.

Los *pipes* permiten utilizar en forma simple tanto la salida de una orden como la entrada de otra, por ejemplo,

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

donde se ejecuta el comando `ls` y su salida, en vez de imprimirse en la pantalla, se envía (por un tubo o *pipe*) al programa *sed*, que imprime su salida correspondiente. Otro ejemplo, para buscar en el fichero */etc/passwd* todas las líneas que acaben con la palabra *false* podríamos hacer

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

donde, primero, se ejecuta el comando `cat`, y su salida se pasa por `grep`. El símbolo $ significa 'final de la palabra', por lo que se está identificando con `grep` todas las líneas que termines con `false`.

A veces, hay ciertos cambios en la sintaxis de algunos comandos cuando se utilizan las tuberías al transmitir información. Dado que no existen ficheros auxiliares creados a lo largo del *pipeline*, se debe introducir el carácter «–» para denotar en este caso que la salida del proceso anterior, a través de la tubería, se convierte en el primer argumento de la siguiente ejecución. Para especial atención en la definición del primer archivo a emplear por el comando `join`.

```
$ cat fichero1 | sort -k2r | join - fichero2
```

1. Introducción a los entornos de trabajo UNIX

1.13. El lenguaje de procesado de archivos GAWK

1.13.1. Introducción

AWK es un lenguaje de programación usado para procesar datos de texto. El nombre AWK proviene de las iniciales de los apellidos de sus autores: Alfred Aho, Peter Weinberger y Brian Kernighan. Y el comando awk es el programa de Gnu/Linux que interpreta el lenguaje de programación AWK. AWK fue creado para reemplazar los algoritmos escritos en C para el análisis de texto. Ganó popularidad rápidamente en Gnu/Linux y se considera una de las utilidades necesarias del sistema operativo. El comando awk fue portado a GNU en 1986, y se llamó *gawk*. Actualmente, Arnold Robins es el principal mantenedor del código y la documentación de *gawk*. En sistemas modernos de Gnu/Linux, la llamada al intérprete de comandos awk está ligada a gawk.

Hasta ahora se han visto comandos para acceder al contenido de ficheros de texto organizados en filas y columnas. A veces, es necesario acceder a campos específicos o realizar cálculos en algunos registros. GAWK es un lenguaje de programación que permite realizar estas operaciones en ficheros de texto. GAWK procesa los registros y genera una nueva línea de resultados. Las variables predefinidas en GAWK permiten acceder selectivamente a la información.

1. Introducción a los entornos de trabajo UNIX

1.13. El lenguaje de procesado de archivos GAWK

1.13.2. Conceptos fundamentales

El comando gawk es una herramienta capaz de ejecutar un bloque de código sobre atributos individuales de cada registro almacenado en las líneas de un archivo de texto. Al procesar los datos, es posible generar una nueva línea de resultados, que se puede mostrar por pantalla o guardar en un nuevo archivo de texto.

El comando gawk recorre el contenido del archivo de texto línea por línea, separando automáticamente los distintos componentes de estas. El usuario tiene acceso a variables predefinidas que proporcionan información selectiva (ver tabla 11) y puede acceder a cada columna de una línea a través de su posición.

Tabla 11. Variables especiales de «awk».

Variable	Descripción
$0	Contiene el contenido del registro de entrada
$1	Primer campo
$2	Segundo campo, y así sucesivamente
NF	*Number of fields*. Número de campos (número de columnas)
FS	Separador de campo de entrada
OFS	Separador de campo de salida
NR	*Number of records*. Número de registros (número de líneas)
RS	Separador de registro de entrada
ORS	Separador de registro de salida
Filename	Nombre del archivo de entrada actualmente en procesamiento.

Fuente: elaboración propia.

Importante: por defecto, el separador de campos es el espacio en blanco, y el separador de registros es el salto de línea.

En general, al intérprete de comandos gawk se le suministran dos tipos de datos:

- un fichero de órdenes o programa,
- uno o más archivos de entrada.

Un fichero de órdenes (que puede ser un fichero como tal, o puede presentarse al invocar gawk desde la línea de comandos) contiene una serie de sentencias que le indican a gawk cómo procesar el fichero de entrada. Es decir, contiene el programa escrito en sintaxis **GAWK**.

```
$ gawk 'programa_GAWK' archivo1 archivo2 …
```

```
$ gawk -f 'archivo_con_código_GAWK' archivo1 archivo2
```

A continuación, tenemos varios ejemplos. Primero se genera un archivo *in situ* con secuencias en formato FASTA y con líneas en blanco entre ellas.

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

Visualizamos el fichero contando las líneas con la opción *-n*

```
$ cat -n secuencias.txt
```

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

Se utiliza *gawk* para imprimir solo las líneas que contengan al menos un campo **no** vacío

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

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

Imprime las primeras 7 líneas

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

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

```
5    CACCGA

6

7    >seq03
```

Imprime las dos primeras líneas y a partir de la sexta; además, elimina los registros vacíos. En este caso el uso de los paréntesis consigue que se cumplan las dos condiciones

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

```
>seq01

ACCTAT

>seq03

AAAACAGAG
```

En el campo de la bioinformática, el fichero de entrada está normalmente estructurado con un formato tabla, por defecto con campos separados por espacios o tabuladores (tablas). Se muestran ejemplos con el comando gawk de las posibilidades de acción de este comando.

Copia en tu terminal el fichero mostrado a continuación y llámalo *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
```

Imprime la primera, la segunda, la cuarta y la última columna del fichero

```
$ 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 se indica que el **separador de campo de la salida** OFS sea también un tabulador, la estructura final del archivo se parecerá al de 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
```

Se realiza una búsqueda literal en este fichero

```
$ 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
```

Se realiza una búsqueda literal de ausencia

```
$ 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 buscas dominar gawk, es crucial que aprendas a manejar correctamente las variables NR, FS y OFS. Los ejemplos anteriores te darán una comprensión profunda de su comportamiento por defecto, así como de cómo manipularlas para modelar adecuadamente la estructura de los registros de datos. Por lo tanto, te recomendamos que los estudies con detenimiento.

1. Introducción a los entornos de trabajo UNIX

1.13. El lenguaje de procesado de archivos GAWK

1.13.3. Síntesis condensada de GAWK

GAWK no se limita a ser una herramienta de filtrado de texto estructurado; es un lenguaje de programación completo que cuenta con estructuras de control de flujo, bucles, diversos operadores y funciones integradas para trabajar tanto con cadenas de caracteres como con números. Además, te brinda la posibilidad de controlar el formato de la salida impresa, utilizar estructuras de datos y escribir funciones *ad hoc*. La combinación inteligente de estos elementos te permitirá crear programas para realizar transformaciones complejas en archivos de texto, como podrás ver en muchos de los ejemplos que se presentan a continuación.

La siguiente lista presenta algunos de los elementos y estructuras sintácticas esenciales del lenguaje de programación gawk. Aunque no vamos a cubrir todos estos elementos en profundidad, esta lista te dará una idea del poder y la flexibilidad de gawk como lenguaje de programación.

- **condicionales**

```
if(condicion1){code1}else

if(condición2){code2}else

{code3}
```

- **bucles *for***

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


for (initialization;condition;

      increment|decrement)
```

- **bucles *while***

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

- **operadores aritméticos**

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

- **operadores boleanos**

```
||, &&
```

- **operadores relacionales**

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

- **funciones integradas**

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

- **funciones escritas por el usuario**

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

- **estructuras de datos**

```
(hashes o arreglos asociativos): array[string]=value
```

Evaluemos el potencial de gawk. Los conjuntos de datos genómicos a menudo se distribuyen con archivos separados para cada cromosoma, y generalmente necesitamos efectuar la misma operación en cada uno. Para realizar esta tarea, podemos usar la estructura de bucle *for* del *shell*. Si tuviéramos archivos llamados `data_chr[chromosome].txt`, donde [chromosome] varía de 1 a 22, y quisiéramos ejecutar ./command en ellos, se escribirían los siguientes comandos en el terminal:

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

El código anterior recorre todos los valores entre 1 y 22, estableciendo la variable de `shell` $i en cada valor, sucesivamente. La sintaxis para especificar los rangos de números es que `{A..B}` da un rango entre *A* y *B*, donde *A* y *B* son valores enteros.

También podemos utilizar `for` para recorrer las extensiones de archivo. Supongamos que teníamos algunos datos en formato PLINK llamados `data.bed`, `data.bim` y `data.fam`. Podríamos listar estos archivos individualmente ejecutando:

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

Aquí, la variable `$ext` (para extensión) se establece sucesivamente en *bed*, *bim* y *fam*. Este ejemplo en particular no es muy útil, ya que podríamos haber escrito `ls data.*` y visto que existen estos tres archivos. Lo útil es poder renombrar el conjunto de datos base con una sola línea de código en el *shell*. Si quisiera renombrar estos archivos `human_data.bed`, `human_data.bim` y `human_data.fam`, se podría escribir:

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

Otra construcción `for` útil es recorrer grupos de archivos. Podemos hacer lo siguiente para ejecutar `./command` en cada archivo con extensión .txt en nuestro directorio actual:

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

Ahora, supongamos que tenemos un archivo llamado *data.txt* y que queremos separar la información correspondiente a cada cromosoma en archivos separados. Podríamos usar un bucle *for* para recorrer cada número de cromosoma y luego una expresión booleana en gawk para extraer solo las líneas correspondientes a ese cromosoma. Comencemos nuestra tarea con un programa que no funciona del todo y luego lo arreglaremos.

Si la columna 2 de *data.txt* contiene los números de cromosoma y queremos archivos separados para cada cromosoma, podríamos pensar que lo siguiente funciona (ten en cuenta, nuevamente, el comportamiento predeterminado de gawk para imprimir las líneas que coinciden con la expresión booleana dada):

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

Esto es casi correcto, pero la expresión dada en gawk es `'$2 == $chr'` gawk no la entiende, porque no conoce ni sabe nada sobre la variable de *shell* `$chr` que se ha definido. En lugar de hacer referencia a una variable en la *shell*, podemos asignar explícitamente variables para que gawk las use con la opción -v:

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

¡No está mal! Podemos proporcionar a gawk tantas opciones -v como queramos para todas las variables que nos importen asignar:

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

    > data_chr$chr.txt; done
```

Esto separa cada cromosoma en un archivo individual con la condición de que los valores en la columna 4 sean mayores que 10.

¿Qué ocurre si tenemos un archivo que contiene dos columnas con un recuento de «éxitos» y «intentos» para algún proceso, cada uno recolectado de una fuente diferente, y queremos calcular la tasa promedio de éxito entre todas las fuentes? Podemos usar gawk para sumar cada columna y luego imprimir el promedio al final usando una sintaxis especial. Si el archivo *data.txt* tiene los éxitos y los intentos en las columnas 2 y 3, respectivamente, podríamos hacer esto:

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

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

END

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

El comando gawk ejecuta el código en la sección BEGIN antes de procesar cualquier línea en la entrada y el código en la sección END después de leer la entrada. La sección BEGIN inicializa dos variables a 0, la sección de código principal suma las columnas en cada línea, y la sección END imprime los totales y la tasa.

Supongamos ahora que queremos buscar mediante condicionales aquellos ensambles que contiene el organismo en *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 siguiente ejemplo, se quieren filtrar los datos de Xenopus tropicalis que tengan una fecha de lanzamiento inferior a 2015. El ejemplo utilizará un condicional en *bash* y se creará *script* que se pueda ejecutar. Salva las próximas órdenes en un fichero de tu terminal.

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

Leer la tabla y guardar las líneas que cumplen las condiciones en un nuevo archivo

```
awk -F'\t' 'NR==1 || ($2 == "Xenopus tropicalis" && $3 > "2015-01-01")' $1
> filtered_table.txt
```

Contar el número de líneas en el archivo filtrado

```
num_lines=$(wc -l < filtered_table.txt)
```

Si el número de líneas es mayor que 1 (es decir, si hay entradas que cumplen las condiciones),

imprimir un mensaje y el contenido del archivo filtrado

```
if [ $num_lines -gt 1 ]; then

  echo "Se encontraron las siguientes entradas para Xenopus tropicalis
publicadas después de 2015-01-01:"

  cat filtered_table.txt

# Si el número de líneas es igual a 1, imprimir un mensaje con el nombre
de la entrada

elif [ $num_lines -eq 1 ]; then

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

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

Si el número de líneas es 0, imprimir un mensaje de que no se encontraron entradas que cumplan las condiciones

```
else

  echo "No se encontraron entradas para Xenopus tropicalis publicadas
después de 2015-01-01."

fi
```

Salva el fichero y ejecuta las siguientes órdenes en el terminal:

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

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

Este *script* emplea gawk para leer la tabla y guardar las líneas que cumplen las condiciones en un nuevo archivo llamado *filtered_table.txt*. Luego, cuenta el número de líneas en ese archivo y utiliza dos condiciones *if* para imprimir mensajes diferentes dependiendo de si hay entradas que cumplan las condiciones o no. En el primer caso, cuando hay más de una entrada que cumple las condiciones, el *script* imprime un mensaje que indica que se encontraron entradas y muestra el contenido de la tabla filtrada. En el segundo caso, imprime un mensaje si solo hay una entrada que cumple la condición y la última acción ocurrirá si no hay ninguna entrada en el fichero de partida que cumpla las condiciones demandadas.

La tabla 12 es un listado de las operaciones más comunes y usadas con GAWK.

Tabla 12. Operaciones utilizadas en GAWK.

Variable	Descripción
`i=0`	Asignar un valor
`a[i]=0;`	Guardar un valor en una tabla
`i++;`	Incrementar un contador
`print i;`	Mostrar una variable por pantalla
`print NR`	Mostrar el número de líneas procesadas
`print $1,$4`	Mostrar la primera y cuarta columnas

Fuente: elaboración propia.

Y en la tabla 13 se muestran ejemplos de instrucciones condicionales con GAWK.

Tabla 13. Instrucciones condicionales con GAWK.

Variable	Descripción
`if ($1 > 0) print $0;`	Si la primera es positiva
`if ($1 < 0) print $0;`	Si la primera es negativa
`if ($1 >= 0) print $0;`	Si la primera columna es mayor o igual a cero
`if ($1 <= 0) print $0;`	Si la primera columna es menor o igual a cero
`if ($1 == 0) print $0;`	Si la primera columna es igual a cero
`if (CONDICION1) && (CONDICION2)`	Si se cumplen las dos condiciones
`if (CONDICION1) || (CONDICION2`	Si se cumple alguna de las dos condiciones
`if ¡(CONDICION1)`	Si no se cumple la condición

Fuente: elaboración propia.

1. Introducción a los entornos de trabajo UNIX

1.13. El lenguaje de procesado de archivos GAWK

1.13.4. Expresiones regulares (en inglés, *regexps*)

Dado que gawk es un lenguaje especializado en el procesamiento de archivos de texto basado en patrones, es esencial volver a presentar una sección sobre expresiones regulares. Una nueva presentación siempre ayuda a fijar los temas a estudiar.

Una expresión regular (*regex* o *regexp*) define uno o varios conjuntos de cadenas de caracteres utilizando una notación específica:

- Una cadena literal de caracteres es una *regex* que define una sola cadena: a sí misma.
- Una expresión regular más compleja:
  - Caracteres ordinarios usados en *regexe*: `_, A-Z, a-z, 0-9`
  - Metacaracteres usados en *regexe*: `.*[ ]^${ }+?|( )`

Las *regexes* se utilizan para encontrar patrones específicos en archivos de texto. Programas como *ed, vim, grep, sed, awk, perl,* entre otros muchos, hacen uso de *regexes* para buscar dichos patrones en archivos de texto.

Hay dos motores de búsqueda de patrones mediante expresiones regulares y hay que tenerlo muy en cuenta en función del comando que se quiere utilizar

- El motor básico de expresiones regulares (BRE), utilizado por ejemplo por *sed* y *grep*.
- El motor extendido de expresiones regulares (ERE), utilizado por ejemplo por gawk y *perl*.

Se añaden dos tablas con la notación de caracteres BRE/ERE empleados más frecuentemente con gawk (tablas 14 y 15).

Tabla 14. Notación de caracteres especiales. Motor BRE/ERE.

Notación	Significado
`\`	Escapa el significado del metacarácter, interpretación literal
`.`	Cualquier carácter sencillo salvo NULL
`*`	Cualquier cantidad de veces (o cero) el carácter precedente
`^`	La *regexp* coincide al inicio de la línea o cadena de caracteres
`$`	La *regexp* coincide al final de la línea o cadena de caracteres
`[123][A-Z]`	Cualquiera de los caracteres incluidos o rango indicado coinciden
`[[:alnum:]]`	Alfanuméricos [a-zA-Z0-9_]
`[[:alpha:]]`	Caracteres alfabéticos [a-zA-Z]
`[[:space:]]`	Espacios (`' '`, tabuladores, salto de línea)
`[[:blank:]]`	Coincide espacios y tabuladores
`[[:upper:]]`	Coincide [A-Z]
`[[:lower:]]`	Coincide [a-z]
`[[:digit:]]`	Coincide [0-9]

Fuente: elaboración propia.

Y otra tabla en particular, con la notación de caracteres ERE usados más frecuentemente:

Tabla 15. Clase de caracteres ERE más frecuentemente usados.

Notación	Significado
\w	Caracteres alfanuméricos [a-zA-Z0-9_]
\W	Caracteres no alfanuméricos [^[:alnum:]_]
\s	Coincide con espacios y tabuladores
\d	Coincide [0-9]
\<	Coincide con el inicio de una palabra
\>	Coincide con el final de una palabra
{n,m}	Expresión de intervalo: coinciden *n* instancias, o de *n* a *m* instancias
+	Coincide una o más instancias de la *regexp* precedente
¿	Coinciden cero o una instancia de la *regexp* precedente
|	Coincide la *regexp* especificada antes o después de | (esto|aquello)
( )	Busca un *match* al grupo de *regexes* incluidas: (esto|aquello)

Fuente: elaboración propia.

1. Introducción a los entornos de trabajo UNIX

1.13. El lenguaje de procesado de archivos GAWK

1.13.5. Manipulación de cadenas de caracteres

El comando gawk provee de diversas funciones para la manipulación de cadenas de caracteres. En este apartado solo se muestran las funciones muy sencillas de usar que se implementan con frecuencia en código *gawk*, y a continuación diferentes ejemplos prácticos con los que se puede 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 instancia más larga en la cadena diana de la *regexp*, sustituyéndola por la cadena reemplazo, mientras que gsub() realiza reemplazos globales.

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

```
GGCCCACACAAACGCGAA
```

```
$ gawk 'BEGIN { cad = "Es una cadena de las bases nitrogenadas: cadena";
print cad; sub(/cadena/, "secuencia", cad); print cad}'
```

```
 Es una cadena de las bases nitrogenadas: cadena
Es una secuencia de las bases nitrogenadas: cadena
```

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

```
Es una cadena de bases nitrogenadas: cadena
Es una secuencia de bases nitrogenadas: secuencia
```

2. index() imprime el índice en el que empieza la subcadena (AAA) en la cadena principal (*seq*, en este caso). index() y substr() con frecuencia se usan juntos.

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

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

```
10
```

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

10

GGCCCACACAAACGCGACGGCGAA

          AAACGCGACGGCGAA
```

3. Y un último ejemplo con `length` (en castellano, *longitud*).

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

```
El oligo GGCCCACACAAACGCGACGGCGAA tiene 24 nt de longitud.
```

1. Introducción a los entornos de trabajo UNIX

1.14. Definición de nuevos comandos

Durante este módulo, hemos observado que los comandos del terminal cuentan con un amplio número de opciones que permiten múltiples combinaciones y es comprensible que cada usuario habitualmente utilice solo un pequeño conjunto de estas posibilidades de configuración. Dado esta particularidad, el intérprete de comandos permite definir nuevos comandos basados en determinadas combinaciones de comandos y opciones que el usuario requiere con frecuencia. El comando `alias` implementa este mecanismo, asociando un nuevo nombre de comando a una secuencia de comandos y opciones prefijadas.

Para crear un alias, usa el comando `alias` con el nombre apropiado. Sin argumentos, este listará todos los alias definidos hasta ese momento. Mientras estás en la *shell*, puedes comprobar qué alias están establecidos escribiendo el comando `alias`. Para crear un alias, da un nombre, seguido de = y luego el comando a ser alias entre comillas simples. No debe haber espacios alrededor del operador =. Usa el comando `type` nombre para comprobar si ese nombre ya está siendo utilizado por otro comando. A continuación, algunos ejemplos:

```
$ type p
```

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

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

```
$ p
```

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

En este ejemplo se ha habilitado la letra *p* al comando `pwd`.

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

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

En el primer ejemplo, las letras `pl` se asignan para ejecutar el comando `pwd` y, a continuación, se ejecuta la orden `ls -CF` con lo que primero se imprime el directorio de trabajo actual y lista su contenido en forma de columna. El segundo ejemplo ejecuta el comando `rm` con la opción `-i` cada vez que escribes `rm` (este es un alias que a menudo se establece automáticamente para el usuario *root)*. En lugar de simplemente eliminar archivos, se te pedirá confirmación para cada eliminación individual de archivo. Esto evita que eliminen automáticamente todos los archivos de un directorio al escribir accidentalmente algo cómo `rm *`.

Si quieres eliminar un alias, escribe `unalias`, letras que definen el alias (recuerda que, si el alias está configurado en un archivo de configuración, se establecerá de nuevo cuando abras otra *shell*).

```
$ unalias p pl
```

```
$ . .bashrc
```

```
$ type p pl
```

```
bash: type: p: not found

bash: type: pl: not found
```

Si deseas guardar una lista personal de `alias` de forma permanente, puedes escribirlos en el archivo $HOME/.bash_aliases, que es uno de los archivos de configuración del entorno que los usuarios avanzados de Gnu/Linux guardan en su directorio $HOME (/home/student).

Cada vez que inicies tu máquina o establezcas una sesión remota, el archivo se leerá en memoria, lo que permitirá que los `alias` se integren en el entorno.

El archivo $HOME/.bash_aliases es utilizado por los archivos de configuración $HOME/.bashrc en la máquina local, y $HOME/.bash_profile cuando se establece una sesión remota a través de SSH. En la sección de administración básica de un sistema Gnu/Linux, profundizaremos en el uso de estos archivos (apartado 1.16).

Como ejercicio, te sugiero que busques y leas estos archivos. Te comunico que necesitas utilizar el comando *ls -a* para verlos, ya que el nombre de archivo está precedido por un punto para ocultarlos de una llamada de *ls* estándar.

1. Introducción a los entornos de trabajo UNIX

1.15. Diseño de protocolos automáticos en el terminal

En este apartado os vais a introducir en el diseño de protocolos automáticos en un terminal *bash*. Dentro del campo de la bioinformática aprender a diseñar protocolos es útil por varias razones. En primer lugar, permite la automatización de tareas repetitivas, lo que ahorra tiempo y reduce el riesgo de errores. En segundo lugar, permite la creación de flujos de trabajo reproducibles, que son esenciales para la investigación científica. En tercer lugar, *bash* es una herramienta poderosa y flexible que se puede utilizar para manipular conjuntos de datos grandes y realizar análisis complejos. Finalmente, el uso de protocolos (en inglés, *script*) *bash* facilita la colaboración y el intercambio de protocolos, lo que hace que sea más fácil para los investigadores reproducir el trabajo de otros y construir sobre él. En general, el diseño de protocolos automáticos en un terminal *bash* es una forma eficiente y efectiva de realizar análisis de bioinformática.

Al diseñar un protocolo en un terminal *bash*, hay varios aspectos técnicos y parámetros que deben tenerse en cuenta. Aquí se presentan los aspectos más importantes que hay que considerar.

- **Sintaxis y gramática**: el protocolo debe tener una sintaxis y una gramática bien definidas que sean fáciles de entender y seguir.
- **Manejo de errores**: el protocolo debe estar diseñado para manejar errores y excepciones de manera elegante. Esto incluye la definición de códigos de error y mensajes.
- **Seguridad**: el protocolo debe estar diseñado para garantizar la privacidad y seguridad de los datos. Esto incluye medidas como la encriptación, la autenticación y el control de acceso.
- **Compatibilidad**: el protocolo debe ser compatible con el *hardware* y el software del sistema en el que se utilizará.
- **Eficiencia**: el protocolo debe estar diseñado para ser eficiente y optimizar el uso de los recursos del sistema, como la CPU y la memoria.
- **Escalabilidad**: el protocolo debe estar diseñado para ser escalable, de modo que pueda manejar cantidades crecientes de datos y usuarios sin degradación del rendimiento.
- **Documentación**: el protocolo debe estar bien documentado, con instrucciones claras y ejemplos para su implementación y uso.
- **Pruebas**: el protocolo debe ser probado exhaustivamente para asegurar que su funcionalidad y rendimiento cumplan con los requisitos.

Teniendo en cuenta estos aspectos técnicos y parámetros, el protocolo puede ser diseñado para ser efectivo, seguro y eficiente en su funcionamiento. En este apartado se os muestra cómo diseñar protocolos sencillos, y no se tendrá en cuenta todo lo mencionado anteriormente, pero siempre se debe trabajar teniendo en cuenta todas las consideraciones mencionadas.

En este ejercicio se os va a mostrar cómo generar un protocolo para obtener información biológica relevante a partir de 3 secuencias FASTA. Te animo a que realices todos los pasos que se muestran a continuación. Abre un terminal y empieza.

1) Crea un directorio:

```
$ mkdir fasta_sequence
```

```
$ cd fasta_sequence
```

2) Descarga, en el anterior directorio, y desde la base de datos ENA (*European Nucleotide Archive*) las secuencias FASTA asociadas a los genes humanos de BRCA1, BRCA2 y 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) Cuando se quiere ejecutar una serie de comandos de forma secuencial en el terminal *bash*, se crea un *script* o protocolo y se guarda en un archivo de texto con extensión «**.** sh». Este *script* nos permite referirnos internamente a sus parámetros de manera genérica, lo que significa que puede funcionar en cualquier grupo de argumentos del mismo tipo. Además, para hacerlo más flexible, se deben especificar estos parámetros genéricos dentro del *script*, de manera que cuando se invoque el *script* desde la línea de comandos, podemos sustituir estos parámetros genéricos por valores específicos proporcionados por el usuario junto con el nombre del comando. Al utilizar este enfoque, se pueden automatizar tareas repetitivas o procesos complejos, lo que hace que el trabajo sea más eficiente y consistente.

```
$ vi analiza_fasta.sh
```

4) Antes de introducir el código para generar un protocolo, dos consideraciones a tener en cuenta:

- En la primera línea del protocolo se ha de indicar, siempre, el intérprete de comandos bash (GNU Bourne-Again SHell). En la mayoría de los sistemas coexisten otros intérpretes como sh (Bourne) o csh (C shell). El directorio donde se localiza el intérprete se escribe a continuación del conjunto de símbolos#!

- Para introducir comentarios en el protocolo se ha de introducir el símbolo# (en inglés, *shebang*) en la primera columna del fichero.

5) Escribe cada una de las siguientes líneas en el fichero *sh* que acabas de abrir. Debes escribir en modo *insert*, lo primero que tienes que hacer es teclear la letra *i*, y a continuación escribe las siguientes líneas:

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

Este protocolo analiza un fichero que contiene una única secuencia FASTA

Primero, se chequea si se suministra el nombre del fichero como argumento

```
if [ $# -eq 0 ]

then

    echo "Por favor, suministra un nombre de fichero como argumento"

    exit 1

fi
```

El símbolo $$ indica el PID del proceso

```
echo "El valor del PID del proceso es";

echo "PID es $$";
```

$1 representa el primer argumento a analizar. En este caso el nombre del fichero FASTA

```
echo "El tamaño del fichero FASTA es:";

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

echo "Número de líneas del fichero FASTA:";

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

```
echo "Extrae las primeras siete líneas del fichero";

head -7 $1;
```

Las siguientes dos líneas considera comentarlas, si la secuencia FASTA tiene muchas líneas

```
echo "Extrae la secuencia del fichero FASTA";

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

echo "$sequence"
```

Cálculo de la longitud de la secuencia

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

echo " Longitud de la secuencia: $length nucleotidos"
```

Cuenta el número de nucleótidos que tiene la secuencia

```
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)
```

Se han generado *in situ* 4 nuevas variables. Imprime cada número de nucleótidos

```
echo "Numero de nuclotido A: $num_A"

echo "Numero de nuclotido C: $num_C"

echo "Numero de nuclotido G: $num_G"

echo "Numero de nuclotido T: $num_T"
```

Cálculo del contenido GC de la secuencia

```
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%"
```

Identificación de los ORFs de la secuencia

```
echo "Identificando ORFs..."
```

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



 if [ -z "$ORFs" ]
 then
     echo "No se encuentran ORFs"
 else
     num_ORFs=$(echo -n "$ORFs" | awk '{print length}' | wc -l)
     echo "Numero de ORFs: $num_ORFs"
     echo "ORFs: $ORFs"
 fi
```

Salva el fichero que se ha creado escribiendo :wq! analiza_fasta.sh

6) Para ejecutar el *script*, salva el anterior fichero con la extensión ".sh" y hazlo ejecutable con el comando chmod +x. Ejecútalo con el nombre del fichero FASTA a analizar:

```
$ pwd
```

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

```
 $ chmod +x analiza_fasta.sh
```

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

7) Hasta ahora se ha generado un protocolo para analizar ficheros FASTA, se ha ejecutado y comprobado que funciona. El siguiente paso es ejecutar el anterior fichero de análisis en un directorio con diferentes secuencias. Para ello, se genera otro protocolo que incluya el anterior. El nuevo protocolo se llama analiza_fasta_dir.sh. Los pasos para seguir son:

```
$ pwd
```

```
/home/student
```

```
$ vi analiza_fasta_dir.sh
```

Las órdenes que hay que introducir en este fichero son:

```
#! /bin/bash

echo "Se inicia la ejecución con el PID $$";

echo "Número de argumentos a analizar $#";
```

```
 echo "El nombre del directorio a analizar $1";

 echo "Los nombres de los ficheros que se analizan son";

 ls $1;

 echo "Ejecución del protocolo de análisis FASTA para cada uno de los
ficheros en el directorio";

 ls $1 |    while read file;

      do

           echo "ejecución analiza_fasta.sh $file";

           ./analiza_fasta.sh $1/$file;

      done
```

Salva el fichero que se ha creado escribiendo :wq! analiza_fasta_dir.sh

8) Para ejecutar el *script*, salva el anterior fichero con la extensión .sh y hazlo ejecutable con el comando chmod +x . Ejecútalo con el nombre del directorio que contiene los ficheros FASTA a analizar:

```
$ pwd
```

```
/home/student
```

```
$ cp fasta_sequence/analiza_fasta.sh .
```

```
$ chmod +x analita_fasta_dir.sh
```

```
$ ./analiza_fasta_dir.sh fasta_sequence
```

Se puede observar que los protocolos que se ejecutan se encuentran en el directorio actual de trabajo (./) y se añade el anterior símbolo en lugar de solo invocar su nombre. Sin embargo, una vez que se está seguro de que el *script* funciona correctamente, es más conveniente guardar toda nuestra colección de protocolos en un solo directorio del sistema. De esta manera, no se han de mantener múltiples copias y se podrán ejecutar desde cualquier lugar en nuestro árbol de directorios.

Las plataformas Gnu/Linux tienen un conjunto de variables globales que contienen datos de carácter general. Se resumen en la siguiente tabla, la tabla 16.

Tabla 16. Comandos para la ejecución de *scripts*.

Comando	Descripción
set	Establece valores que serán usados por los programas, aplicación, *scripts*
echo	Imprime lo que se le encarga que haga
printenv	Lista la lista completa de variables de entorno de tu versión Gnu/Linux
which	Localiza los archivos ejecutables de una determinada aplicación
export	Crea/modifica una variable del sistema

Fuente: elaboración propia.

El comando `set` en Gnu/Linux muestra y establece variables de entorno para la sesión actual. Se utiliza para ver el valor de las variables de entorno del sistema y también para asignar valores a nuevas variables de entorno. El comando `echo`, por otro lado, se utiliza para mostrar mensajes de texto o el valor de las variables en la pantalla. También se puede utilizar para escribir texto en archivos o para concatenar varios mensajes de texto.

La ventaja de utilizar *echo* es que permite seleccionar y mostrar solo la información que se necesita, lo que hace que sea más fácil de leer y entender. Por otro lado, la ventaja de utilizar *set* es que permite establecer y modificar variables de entorno, lo que puede ser útil al automatizar tareas y *scripts* en Gnu/Linux.

Abre un terminal de Gnu/Linux. Puedes ver la lista completa de variables de entorno de tu versión de Gnu/Linux utilizando el comando `printenv`. Puedes tener una lista más manejable añadiendo diferentes comandos:

```
$printenv | less
```

Cada línea contiene el nombre de la variable de entorno Gnu/Linux seguido de = y del valor. Por ejemplo:

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

Esto quiere decir que HOME es una variable de entorno de Gnu/Linux que tiene el valor establecido como *directorio /home/student*.

Las variables de entorno suelen estar en mayúsculas, aunque también puedes crear variables de entorno en minúsculas. La salida de `printenv` muestra todas las variables de entorno en mayúsculas. Una cosa importante para tener en cuenta es que las variables de entorno de Gnu/Linux distinguen entre mayúsculas y minúsculas. Si deseas ver el valor de una variable de entorno específica, puedes hacerlo considerando el nombre de esa variable como argumento del comando `printenv`. La cadena de caracteres completa se vería así en la línea de comando:

```
$ printenv HOME
```

```
/home/student
```

```
$ echo $USER
```

```
student
```

La sintaxis básica para crear una variable de entorno en Gnu/Linux es la siguiente. Es fácil lograrlo, solo se necesita especificar un nombre y un valor. Seguiremos la convención de mantener todas las letras en mayúsculas para el nombre de la variable, y la estableceremos como una cadena simple.

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

```
$ export HIB_VAR
```

```
$ export HIB_VAR='BioInformatica y BioEstadistica!'    #en una única
línea
```

Se han usado comillas simples, ya que el valor de la variable contiene un espacio. Además, se han utilizado comillas simples porque el signo de exclamación es un carácter especial en la *shell bash* que normalmente se expande al historial de *bash* si no se escapa o se coloca entre comillas simples. Ahora tenemos una variable de *shell*. Esta variable está disponible en nuestra sesión actual, pero no se transmite a los procesos secundarios.

Se puede comprobar, buscando nuestra nueva variable dentro de la salida de *set*:

```
$ set | grep HIB_VAR
```

```
HIB_VAR='BioInformartica y BioEstadistica!'
```

Si la variable se ha definido correctamente, podrás utilizarla en cualquier protocolo que ejecutes en la misma sesión del terminal. Por ejemplo, si creas un archivo de *script* en *bash* que requiere utilizar la variable `HIB_VAR`, simplemente puedes referirte a ella utilizando el símbolo `$`. Por ejemplo, si tu archivo de *script* se llama *`myscript.sh`* y contiene el siguiente código:

```
#! /bin/bash

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

Si se ejecuta el archivo de *script* en la misma sesión del terminal utilizando el comando *bash*,

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

```
$ bash myscript.sh
```

```
BioInformatica y Bioestadistica!
```

el archivo de *script* debería imprimir el valor de la variable `HIB_VAR` que definiste previamente. Para revertir el valor de una variable se puede usar el comando `unset`.

```
$ echo $HIB_VAR
```

```
BioInformartica y BioEstadistica!
```

```
$ unset HIB_VAR
```

```
$ echo $HIB_VAR
```

Por otra parte, ten en cuenta que, si cierras la sesión del terminal y la vuelves a abrir, tendrás que volver a definir la variable local utilizando el comando `export`. Para evitar esto, puedes agregar la definición de la variable HIB_VAR a tu archivo de inicio de *bash* (por ejemplo, `~/.bashrc`), de modo que la variable esté disponible cada vez que inicies una nueva sesión del terminal.

El comando `which` es una herramienta que permite encontrar rápidamente los archivos ejecutables de una determinada aplicación y localiza los ficheros ejecutables mediante la variable de entorno PATH.

```
$ which nano docker gawk
```

```
/usr/bin/nano

/usr/bin/docker

/usr/bin/gawk
```

Finalmente, se define la variable PATH. El contenido de la variable PATH es una cadena que contiene *paths* de directorios separados por dos puntos, y estos son los directorios en los que el *shell* busca el comando que el usuario escribe desde el teclado.

```
$ echo $PATH
```

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

La búsqueda no se realiza en el orden en el que están los directorios en la variable PATH. Cuando se escribe un comando el *shell* buscará primero en `/usr/local/bin`, luego en `/usr/bin`, a continuación, en `/usr/games` y finalmente en `/snap/bin`. Desde el momento en que el *shell* encuentra el comando, detiene la búsqueda y ejecuta el comando encontrado. Podemos escribir un comando utilizando:

- Su nombre:
  El path absoluto (*/bin/cat /etc/passwd*).
  El path relativo (utilizando «.» o «..» en general para los programas o *scripts* que no se encuentran en PATH).
  Se puede añadir un directorio a la variable PATH, añadamos el directorio donde se encuentran todos los *scripts* que se generen.

- Únicamente para la sesión activa:
  Si deseas añadir, por ejemplo: `/home/student/HIB_scripts` a la variable `PATH`, escribe en el *shell* lo siguiente según el caso.
  # Para tener el directorio al final del PATH:

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

  # Para tener el directorio al inicio del PATH:

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

  Ahora puedes utilizar el programa escribiendo simplemente su nombre. Al desconectarse, PATH retomará a su valor por defecto, entonces /home/student/HIB_scripts no existirá más.

- De manera permanente:
  Si deseas configurar PATH de forma permanente debes editar el archivo de configuración de su *shell* de conexión. Como por lo general el *shell bash* es el más utilizado, debes editar el archivo: /home/user/.bashrc. El comando entonces sería:

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

  Después de esto, en cada conexión la variable PATH contendrá el directorio */home/student HIB_scripts.* Esta operación puede ser ejecutada por el usuario *student*, no se necesitan los permisos de *root*.

1. Introducción a los entornos de trabajo UNIX

1.16. Transferencia de ficheros desde el terminal

La transferencia de archivos desde un terminal es una tarea fundamental para los administradores de sistemas y desarrolladores que trabajan con servidores remotos. Aunque existen muchas herramientas gráficas para transferir archivos, las opciones de la línea de comandos son muy útiles para la automatización y el manejo de grandes cantidades de datos. Ejemplos de diferentes comandos se muestran en la tabla 17.

Tabla 17. Comandos de transferencia de ficheros.

Comandos	Significado
`scp`	*Secure copy*
`sftp`	*Secure File Transfer Protocol*
`rsync`	una herramienta rápida, versátil, remota (y local) de copia de ficheros
`wget`	Se utiliza para recuperar contenido y ficheros de varios servidores web
`curl`	La URL del cliente (cURL) os permite intercambiar datos entre el dispositivo y un servidor a través de una interfaz de línea de órdenes (CLI)

Fuente: elaboración propia.

Para establecer una sesión de trabajo es necesario conocer el nombre de la máquina remota o su dirección IP. También debemos disponer de un nombre de usuario (en inglés, *login name*) y de una contraseña de acceso (en inglés, *password*).

Uno de los comandos más comunes para transferir archivos es `scp` (*Secure Copy*). Este comando se utiliza para copiar archivos de un servidor a otro utilizando el protocolo SSH. La sintaxis básica es la siguiente:

```
$ scp [opciones] origen destino
```

Por ejemplo, si queremos copiar el archivo *archivo.txt* de la carpeta origen en el servidor remoto a nuestra carpeta actual en la máquina local, podemos utilizar el siguiente comando:

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

El punto final en la línea de comandos indica el directorio actual de la máquina local.

También existe el comando `sftp` (en inglés, *Secure File Transfer Protocol*), que es similar al cliente FTP pero con una capa de seguridad adicional a través de SSH. Con `sftp`, podemos conectarnos a un servidor remoto y transferir archivos de manera segura. La sintaxis básica es la siguiente:

```
$ sftp usuario@servidor-remoto
```

```
put secuencia.fa repositorio-secuencias/
```

Una vez conectados, podemos utilizar comandos como `put` para enviar archivos desde nuestra máquina local al servidor remoto, o `get` para descargar archivos del servidor remoto a nuestra máquina local. En el ejemplo anterior, se ha descrito como enviar el archivo *secuencia.fa* desde nuestra máquina local a la carpeta repositorio-secuencias en el servidor remoto.

Otro comando útil es `rsync`, que permite sincronizar directorios y archivos entre dos sistemas. Además, `rsync` tiene la *capacidad* de transferir solo los archivos modificados, lo que lo hace ideal para hacer copias de seguridad de manera eficiente. La sintaxis básica de `rsync` es la siguiente:

```
$ rsync [opciones] origen destino
```

Por ejemplo, si queremos sincronizar la carpeta *work-hib* de nuestro equipo local con la carpeta *work* en un servidor remoto, podemos utilizar el siguiente comando:

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

Otra herramienta útil es `wget`, que permite descargar archivos desde servidores web utilizando comandos de terminal. Este comando es muy útil para descargar datos de secuenciación de genomas completos o de grandes bases de datos biológicas, archivos desde internet y automatizar el proceso de descarga. Se muestra a continuación cómo descargar la base de datos COSMIC (*Catalogue Of Somatic Mutations In Cancer*) utilizando `wget`

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

Este comando descarga el archivo CosmicCodingMuts.vcf.gz de la página web del proyecto COSMIC. Este archivo contiene las mutaciones somáticas encontradas en la secuenciación de ADN de tumores de cáncer. Una vez descargado, lo podéis eliminar, ya que no se hará nada con este archivo.

Por último, otra herramienta útil es `curl`, que también permite descargar archivos desde servidores web y transferir archivos desde un terminal. La principal diferencia entre `wget` y `curl` es que `curl` es más versátil y permite transferir archivos mediante una gran variedad de protocolos, incluyendo FTP, FTPS, HTTP, HTTPS, SCP y SFTP.

En resumen, la transferencia de archivos desde un terminal es una tarea fundamental para los administradores de sistemas y desarrolladores que trabajan con servidores remotos. Los comandos `scp`, `rsync` y `sftp` son algunas de las herramientas más comunes y útiles para transferir archivos de manera segura y eficiente.

Existe un comando que ayuda a averiguar el espacio disponible de un disco, el comando `df` (*disk free*), y otro comando para averiguar el espacio ocupado por una carpeta, el comando `du` (*disk use*). Analicemos un par de ejemplos para ver su funcionamiento.

- `df`

Espacio disponible en disco. Esto nos devolverá las particiones montadas, el uso de espacio en cada una y lo que nos queda de resto, y todo de forma fácil para leer, sobre todo porque se añade la opción *-h,* que significa *human-readable* (en español, *humanamente legible*).

```
$ 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`

Tamaño total de una carpeta. Uso de disco. Muestra el espacio que está ocupado en disco. Este comando tiene varias opciones y las más utilizadas son las opciones *-b* (*-bytes*); *-s* (*sumarize*; en castellano, *en resumen*), *-h* (*-human-readable*; en castellano, *humanamente legible*), que imprime los tamaños de forma leíble, al agregar el tamaño de los archivos en kb, mb, gb; y, por último, el comando *-c*, que se emplea para que mostrar el total del espacio consumido, al final de la lista.

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

```
204M    fasta_sequences

45M     Escriptori

12G     hg38

12M     Work

4,0K    Videos
```

Con este ejemplo, se visualizan los 5 directorios más pesados en nuestro /home/student utilizando el comando *du*, así como *sort* para ordenar y *head* para visualizar únicamente algunos de los directorios de la carpeta.

Y finalmente se mencionan comandos con los que se obtiene información del sistema, como son los comandos uname, hostname y host.

- uname

El comando uname (*unix name*). Ofrece información sobre Kernel del sistema, información sobre el tipo de Linux en el que estamos. Teclea en el terminal

```
$  uname
```

```
Linux
```

```
$ uname -a
```

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

- hostname

El comando hostname se utiliza para identificar de forma única a un ordenador en una red y se utiliza para acceder a él y permitir que otros ordenadores de la red se comuniquen con él. El hostname puede ser un nombre descriptivo o un nombre único asignado por el sistema operativo del ordenador o servidor. Por ejemplo, el hostname de un servidor web podría ser *webserver.ejemplo.com*, donde *webserver* es el nombre del ordenador, *ejemplo* es el nombre del dominio y *com* es el dominio de nivel superior. Si tecleas *hostname* en tu terminal se obtiene

```
$ hostname -f
```

```
ubuntum0151
```

siendo ubuntum0151 el hostname de la máquina virtual que habéis instalado.

- host

El comando `host` determina la IP de HOST, y en el contexto de la informática, `host` se refiere a un servidor u ordenador que aloja y ofrece servicios o recursos a otros ordenadores conectados en una red. Si se escribe `host -a` se despliega toda la información 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. Introducción a los entornos de trabajo UNIX

1.17. Ejemplo práctico 1: Analizando el genoma humano

1.17.1. Introducción

El navegador genómico UCSC es una herramienta poderosa que permite a los bioinformáticos visualizar y analizar datos genómicos, incluida la expresión génica, las variaciones de secuencia y las modificaciones epigenéticas. Al dominar el uso de este navegador y otras herramientas de bioinformática, los bioinformáticos pueden contribuir al avance de la investigación genómica, incluidos el desarrollo de medicina personalizada y el descubrimiento de nuevos objetivos terapéuticos.

Es recomendable que un bioinformático sepa analizar el genoma humano, especialmente usando el navegador genómico UCSC. Para ello, es habitual emplear archivos que contienen las secuencias de ADN o de proteínas en formato FASTA, o ficheros de texto tabulado con la ubicación de los elementos codificados en su interior.

Con el objetivo de aproximar al estudiante a un escenario con una situación realista, dentro de un entorno bioinformático, se os propone un ejercicio de análisis de genoma humano y que se divide en tres tipos de actividades:

1. Descarga y exploración del genoma humano.

2. Descarga y análisis del catálogo de genes humanos.

3. Descarga de las herramientas del navegador genómico.

Fundamental e imprescindible para adquirir los conocimientos necesarios para entender un proyecto bioinformático: reproducir cada etapa de este protocolo de análisis en vuestro propio terminal.

1. Introducción a los entornos de trabajo UNIX

1.17. Ejemplo práctico 1: Analizando el genoma humano

1.17.2. Descarga y exploración del genoma humano

La organización del genoma de un organismo se da en un conjunto de cromosomas. En este ejemplo, se procede a descargar la anotación sobre los cromosomas del genoma humano en su distribución *hg38*. Se adjunta la tabla 18 con los accesos de descarga que se van a utilizar.

Tabla 18. Páginas web del navegador genómico UCSC.

Acceso	Dirección
Página principal servidor UCSC	http://genome.ucsc.edu/
Página descargas (genoma data)	https://hgdownload.soe.ucsc.edu/downloads.html
Página listado especies	https://hgdownload.soe.ucsc.edu/goldenPath/currentGenomes/
Página acceso especie *human*	https://hgdownload.soe.ucsc.edu/downloads.html#human
Página acceso especie *human*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/
Página *Sequence data by Chromosome*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes/
Página acceso *bigZips*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/

Fuente: elaboración propia.

Primeramente, se accede a la página de descargas (en inglés, *downloads*) del navegador genómico UCSC (https://hgdownload.soe.ucsc.edu/downloads.html).

El contenido de esta página nos muestra el listado de genomas disponibles organizado por especies. A fecha de 26 de abril de 2023 existen información sobre 108 especies.

Al hacer uso del enlace *Human*, entramos a la sección dedicada al genoma humano. Es importante destacar que la información correspondiente a cada genoma se actualiza con cierta frecuencia, por lo que cada mejora sustancial cuenta con un código de versión propio. En este caso, trabajaremos con la distribución conocida como *hg38*, la cual es la más reciente al momento de la redacción de estos materiales.

Si desde la página de acceso a la especie humana accedéis al enlace asociado a *Sequence data by chromosome* (https://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes) accederéis al listado de los ficheros comprimidos FASTA de cada uno de los cromosomas (chr*.fa.gz), a la secuencias *random*, que son secuencias no colocadas en los anteriores cromosomas de referencia (chr*_random), y a las secuencia *chrUn_**, que son secuencias no localizadas en las que el cromosoma de referencia no ha sido determinado. En la misma fecha mencionada anteriormente, hay 456 secuencias FASTA asociadas a diferentes cromosomas.

Si desde esta última localización se accede al *Parent Directory* (https://hgdownload.soe.ucsc.edu/goldenPath/hg38/), encontraréis el directorio llamado *bigZips* (https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/), que es otro repositorio de ficheros, con distintos formatos, asociado al genoma humano. Todos los archivos están comprimidos y empaquetados para reducir el tiempo de transmisión.

El fichero hg38.chromFa.tar.gz contiene la secuencia original de los cromosomas separados en archivos independientes. Hay que descargar este fichero y se hará con el comando *wget*, pero solo debes descargar el fichero si tienes más de 5 Gb disponibles en el disco duro. Si tienes menos de 5 Gb libres, descarga la secuencia FASTA del cromosoma 7 desde el directorio https://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes/

El comando `df` es el comando que se utiliza para averiguar el espacio en disco

```
$ 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 máquina en la que se está trabajando solo hay disponibles 2,9 G de espacio (columna *Available*), por lo que en este caso solo se descarga la secuencia FASTA del cromosoma 7. Si hubiera espacio en el disco duro para descargar el fichero con toda la información, el procedimiento sería el siguiente:

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

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

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

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

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

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

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

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

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

Una vez que el fichero está descargado en la máquina Gnu/Linux con la que se trabaje se debe desempaquetar y descomprimir el fichero con el objetivo de visualizar el contenido de este.

```
$ 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

…
```

Aunque la calidad de la secuencia del genoma humano es aceptable, todavía se encuentra en fase de mejora. Debido a esto, es común encontrar numerosos archivos que contienen fragmentos o variantes que aún están en discusión y que no necesariamente corresponden a un cromosoma completo. Es posible visualizar el primer cromosoma en el terminal; sin embargo, en algunas partes del cromosoma, como el inicio, la secuencia de nucleótidos es desconocida y se denota con el carácter *N*. Además, para indicar la presencia de elementos codificados en la secuencia, se puede utilizar una combinación de letras mayúsculas y minúsculas.

```
$ more chr7.fa
```

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

…

…

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

GAATTCTACATTAGAAAAATAAACCATAGCCTCATCACAGGCACTTAAAT

ACACTGAAGCTGCCAAAACAATCTATCGTTTTGCCTACGTACTTATCAAC

TTCCTCATAGCAAACTGGGAGAAAAAAGCAATGGAATGAATAAAATGATA

GCCACAAAAATCAAGGTGGGAGAAATACTTATTATATGTCCATAAAAAAT

TTTAATTAATGCAAAGTATTAACACCAATGATTGCAGTAATACAGATCTT

ACAAATGATAGTTTTAGTCTGAACAGGACTATCCAAAAGTTAATTTTCTA

TAGTAACAGTTTTTAAATAAAATATCAATTCCTGAAACACATAAAATGGT

CCATGAGTATACAACGAGTGAAAAAAAACAAATTCAGAGCAAAGATAAAT
```

```
TAAGAAGTATCTAATATTCAAACATAGTCAAAGAGAGGGAGATTTCTGGA

TAATCACTTAAGCCCATGGTTAAACATAAATGCAAATATGTTAATGTTTA

CTGAATAACTTATCTGTGCCAAGTGGTGTATTAATGATTCATTTTTATTT

TTCACTAAATCTTTTCTCTAAAGTTGGTGTAGCCTGCAACTAAATGCAAG

AAATCTGACCTAGGACCTGCACTTCTTACCATTTTGCTCATATTTATTCC

CTGTGCATTTTTGTAACATGTATATGTTATATATATAGAAAGAGAGAGAG

GCAGAGATGGAAAGTAATTTATGGAGTTTGATGTTATGTCAGGGTAATTA

CATGATTATATAATTAACAGGTTTCTTTTTAAATCAGCTATATCAATAGA

AAAATAAATGTAGGAATCAAGAGACTCATTCTGTCCATCTGTGATAGTTC

CATCATGATACTGCATTGTCAAGTCATTGCTCCAAAAATATGGTTTAGCT

CAACactgagtgactataggaaaccagaaaccaggctgggcgctaaagat

gcaaagatgaatgagacatcatctctgccgtccaaaagcttactgtctag

tgggagagttacacacgtaaggacagtaatctaataagagctaataagtg

aaaactaagataaattaataatacaagattacagggaaggtttccaaagt

caatgaggcctcaaatgaatcttgaaagtgtgcaaggattaaccaaatga
```

1. Introducción a los entornos de trabajo UNIX

1.17. Ejemplo práctico 1: Analizando el genoma humano

1.17.3. Análisis de los genes humanos

El análisis del catálogo de genes de una especie es un ejemplo perfecto de la utilidad de los comandos del terminal en este capítulo. Un gen es una secuencia de ADN que contiene información para crear una molécula biológica. Los genes de los organismos eucariotas se componen de exones. Muchos genes humanos tienen combinaciones plausibles de exones, lo que resulta en transcritos alternativos.

Para codificar la ubicación de los genes, se utilizan ficheros tabulados que contienen información como la localización y las características del transcrito del gen, como el número de exones y las coordenadas exactas.

En este ejercicio, se utiliza la anotación del consorcio *RefSeq* para obtener información sobre el genoma humano, en un primer momento, pero también se utilizará información del genoma de ratón (*mouse*). La información se encuentra en un fichero llamado *refGene.txt* y se puede obtener en la página web del navegador genómico de UCSC mediante el enlace *Annotation*. La dirección de descargas se muestra en la tabla 19.

Tabla 19. Páginas de descarga de los ficheros refGene.txt en función de la especie.

Acceso	Dirección
Página descarga *human*	https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/
Página descarga *mouse*	https://hgdownload.soe.ucsc.edu/goldenPath/mm39/database/

Fuente: elaboración propia.

Las anotaciones, en el servidor de UCSC, suelen representarse gráficamente en el navegador genómico en forma de pista. El navegador genómico está gestionado internamente por el administrador de bases de datos relacionales de MySQL. En consecuencia, en esta página web encontraremos dos tipos de fichero para cada pista de anotaciones de UCSC. El primer fichero, cuya extensión será del tipo *sql*, contiene una especificación genérica de los atributos de las anotaciones. Este archivo es necesario para crear una tabla vacía en una base de datos relacional. El segundo fichero, que estará comprimido y poseerá la extensión *txt*, contiene propiamente la información de cada anotación de forma tabulada. Al acceder a la página de descarga se visualiza la siguiente información:

```
 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
```

Ahora procedemos a descargar el fichero *txt* asociado a la pista *refGene*, que contiene el catálogo de genes humanos anotados por el consorcio *RefSeq*. Para ello, se utiliza el comando *wget* nuevamente para transferir el fichero al terminal.

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

```
 --2023-04-26 16:35:43--
https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz

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

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

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

 Mida: 8668756 (8,3M) [application/x-gzip]

 S'està desant a: «refGene.txt.gz»

 refGene.txt.gz                                          100%[================>]
8,27M  2,22MB/s    in 3,7s

 2023-04-26 16:35:49 (2,22 MB/s) - s'ha desat «refGene.txt.gz»
[8668756/8668756]
```

A continuación, te invitamos a que revises el contenido del primer fichero: refGene.sql. ya que es útil para conocer las características de los genes anotados en cada columna del fichero. Los atributos que se utilizan con mayor frecuencia son los siguientes: *name* (código del transcrito), *chrom* (cromosoma), *strand* (hebra), *txStart* y *txEnd* (coordenadas de inicio y final), *exonCount* (número de exones) y *name2* (nombre del gen). Es importante no confundir los campos *name* y *name2*: un gen puede tener varios transcritos, pero un transcrito solo puede pertenecer a un gen. Estos parámetros se encuentran en la cabecera del fichero descargado.

Vamos a visualizar el contenido del fichero refGene.txt. Este archivo contiene información sobre el catálogo completo de genes anotados en el genoma humano. Cada línea representa un transcrito de un gen determinado. En el caso de que un gen tenga varios transcritos, cada uno se codifica en líneas separadas, cada una con su propio código y sus coordenadas correspondientes. En todas las líneas, los atributos de cada transcrito están separados por el carácter tabulador o «\t». Antes de poder visualizar el contenido del archivo, debemos descomprimirlo utilizando el comando *gzip*.

Descompresión del fichero con *gzip*

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

Visualización de las primeras cinco líneas con el comando *head*

```
$ head -5 refGene.txt
```

```
585	NR_024540	chr1	-	14361	29370	29370	29370	11	14361,14969,15795,16606,16857,17232,17605,17914,18267,24737,29320,	14829,15038,15947,16765,17055,17368,17742,18061,18366,24891,29370,	0	WASH7P	unk	unk	-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,

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

585	NR_107062	chr1	-	17368	17436	17436	17436	1	17368,	17436,	0	MIR6859-2	unk	unk	-1,

585	NR_107063	chr1	-	17368	17436	17436	17436	1	17368,	17436,	0	MIR6859-3	unk	unk	-1,

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

Ten en cuenta que las anotaciones de un genoma suelen actualizarse frecuentemente. Por este motivo, los datos mostrados en este tutorial pueden variar ligeramente con el paso del tiempo si lo comparas con una nueva descarga del mismo *fichero*.

Debido a la gran cantidad de información que contiene este archivo son difíciles de manejar. A continuación, se decide trabajar con unos determinados campos y se utilizará el comando gawk para extraer únicamente los campos necesarios de este ejercicio. En particular, encontramos interesantes los siguientes atributos: nombre del gen, identificador del transcrito, cromosoma, hebra, coordenadas y número de exones. Para evitar tener que ejecutar el comando previo cada vez que se necesiten estos datos, se redirecciona la salida hacia un archivo refgene-select.txt. Una vez obtenido, se cuentan el número total de transcritos humanos.

Analizamos el fichero *refGene.sql* para determinar la posición de cada columna

```
$ gawk '{print $13,$2,$3,$4,$5,$6,$9;}' refGene.txt > refgene-select.txt
```

Se visualizan las últimas 5 líneas

```
$ 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 número de transcritos es el número de líneas del fichero

```
$ wc -l refgene-select.txt
```

```
88819 refgene-select.txt
```

Continuamos con diferentes análisis,

Se solicita el número de transcritos genéticos distribuidos en la hebra positiva

```
$ gawk '{if ($4 == "+") print $0;}' refgene-select.txt | wc -l
```

```
45121
```

Se solicita el número de transcritos del cromosoma 21

```
$ grep chr21 refgene-select.txt | wc -l
```

```
1121
```

Se muestran los primeros siete transcritos después de ordenar por nombre 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
```

Se muestran los primeros seis transcritos después de ordenar por el número de genes que contiene cada transcrito

```
$ 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 fichero *refgene-select.txt* contiene el listado de los transcritos humanos. Como un gen puede dar lugar a varios transcritos alternativos, podemos contar cuántos genes tiene el genoma humano y averiguar cuáles poseen un mayor número de transcritos. Para conseguirlo, ordenamos los nombres de los genes e introducimos diferentes variantes del comando `uniq` para agruparlos. Por otra parte, la mejor forma de estudiar el comportamiento de una línea de comandos es la deconstrucción: eliminando las últimas instrucciones se puede mostrar el resultado parcial de la ejecución en pantalla.

Ordena la columna por el nombre de los genes

```
$ gawk '{print $1;}' refgene-select.txt | sort | more
```

```
A1BG

A1BG-AS1

A1CF

A1CF

A1CF

A1CF

A1CF

…
```

Ordena la columna por el nombre de los genes y que sean valores únicos

```
$ gawk '{print $1;}' refgene-select.txt | sort | uniq | more
```

```
A1BG

A1BG-AS1

A1CF

A2M

A2M-AS1

A2ML1

…
```

Determina el número de genes únicos en el fichero

```
$ gawk '{print $1;}' refgene-select.txt | sort | uniq | wc -l
```

```
28307
```

Determina cuántos transcritos hay para cada uno de los genes

```
$ 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 cuántos transcritos hay para cada uno de los genes y ordenalos por el número de exones, de mayor 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

…
```

Otra operación muy frecuente consiste en el cálculo de valores promedio. En el siguiente ejemplo, el estudiante calculará el promedio de exones por transcrito y la longitud media de estos. El funcionamiento de gawk es similar en ambos casos, ya que trabaja con la variable *t* como contador que acumula la suma total. La división por el número de líneas visitadas (NR), una vez que se completa la lectura del archivo, genera el valor promedio en cada caso.

Cálculo de promedio de exones por transcrito

```
$ gawk 'BEGIN{t=0;}{t=t+$7;}END{print t/NR;}' refgene-select.txt
```

```
10.0961
```

Cálculo de la longitud media de los transcritos

```
$ gawk 'BEGIN{t=0;}{t=t+$6-$5+1;}END{print t/NR;}' refgene-select.txt
```

```
64883
```

Para ilustrar la utilidad de asociar dos ficheros de texto tabulado, se va a combinar el catálogo de genes humanos introducido hasta ahora (archivo *refGene.txt* para *H. sapiens se renombra a refGene-human.txt*) con el catálogo equivalente del genoma del ratón (archivo *refGene.txt* para *M. musculus* y renombrado como *refGene-mouse.txt* después de haberlo descargado del servidor genómico de UCSC).

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

```
$ mv refGene.txt refGene-human.txt
```

```
$ wget
https://hgdownload.soe.ucsc.edu/goldenPath/mm39/database/refGene.txt
```

Se renombre el fichero del genoma *M.musculus*

```
$ mv refGene.txt refGene-mouse.txt
```

Para cada uno de los ficheros, se seleccionan las columnas nombre del gen y cromosoma y posteriormente se visualizan los 5 primeros

```
$ 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
```

Para el fichero genoma de ratón

```
$ 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
```

Con esta transformación previa del fichero, se han preparado los ficheros para utilizar el comando `join`: a continuación, se muestra cómo asociar los genes que tienen el mismo nombre en ambas especies. Al comando `join` se añade la opción *-i*, para que se ignoran las diferencias de mayúsculas/minúsculas. Cada línea del resultado final contiene el nombre del gen y su ubicación en ambos genomas.

```
$ 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 de este análisis preliminar del catálogo de genes humanos, se pueden poner en práctica múltiples variantes de los comandos aquí mostrados. Por ejemplo, es posible añadir más atributos, más genomas o más ficheros con otras propiedades para ampliar esta exploración. Por lo tanto, os animamos a experimentar con cada bloque de comandos utilizado durante este ejercicio. Con esta aplicación práctica, se demuestra la validez del terminal de Gnu/Linux para analizar eficientemente datos biológicos. En futuros apartados, se introducirán sistemas más complejos para gestionar grandes conjuntos de datos, de modo que el usuario podrá reproducir el mismo caso práctico utilizando dichas técnicas.

1. Introducción a los entornos de trabajo UNIX

1.17. Ejemplo práctico 1: Analizando el genoma humano

1.17.4. Descarga de las herramientas de UCSC

El navegador genómico de UCSC ofrece de manera gratuita las secuencias de los genomas de múltiples especies y las pistas de anotaciones asociadas a cada versión. Además, este portal web también proporciona acceso libre a un conjunto de herramientas diseñadas específicamente para analizar los datos genómicos. Estos programas funcionan en línea de comandos y se distribuyen listos para funcionar en varias plataformas de la familia UNIX. Para mostrar el funcionamiento del comando `rsync` en la descarga de un directorio web completo, tendréis que obtener una copia completa de las utilidades de UCSC. En la tabla 20 se detallan los accesos a las direcciones de descarga.

Tabla 20. Acceso a las direcciones de descarga de utilidades de UCSC.

Acceso	Dirección
Página información	https://hgdownload.soe.ucsc.edu/downloads.html#utilities_downloads
Página descarga *utilities*	https://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/

Fuente: elaboración propia.

Para ello se accede nuevamente a la página de descargas del navegador. Una vez allí, busca la sección *Utility Tools and Software downloads* (en castellano, *Herramientas de utilidad y descargas de* software). En esta sección explican como descargarlo.

Para descargar todas las utilidades de línea de comandos en un directorio con los *bits* de permisos correctos, utiliza el siguiente comando:

```
$ pwd
```

```
/home/student
```

```
$ mkdir UCSC-utilities
```

```
$ cd UCSC-utilities
```

```
$ rsync -aP hgdownload.soe.ucsc.edu::genome/admin/exe/linux.x86_64/ ./
```

Después de más de 5 minutos se habrá descargado el 5 % de las utilidades. Este proceso es lento y nos damos cuenta de que va a ocupar mucho espacio, más espacio del que hay libre en el disco duro de nuestro ordenador. Mata el proceso apretando las teclas Ctrl + X y se parará la descarga. Es una prueba para ver que funciona el comando `rsync`. Borra todo lo que has generado.

```
$ cd ..
```

```
$ pwd
```

```
/home/student
```

```
$ rm -r UCSC-utilities
```

Otra opción es descargar únicamente las utilidades que se quieran utilizar, por ejemplo, se descarga la utilidad *liftOver*.

```
$ pwd
```

```
/home/student
```

```
$ wget https://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/liftOver
```

```
$ chmod +x /home/student/liftover/liftOver
```

```
# Ejecuta el programa sin argumentos para ver un mensaje de uso
```

```
$ cd liftOver
```

```
$ . /liftOver
```

Resumen

«Introducción a los entornos de trabajo GNU/Linux» es un capítulo que ofrece una guía detallada para trabajar en sistemas operativos basados en la filosofía GNU/Linux. El libro comienza con una introducción a la historia y la evolución de GNU/Linux, y luego se adentra en los conceptos básicos del sistema operativo. A lo largo del capítulo, se cubren temas como la estructura de archivos y directorios, los comandos y herramientas básicas de GNU/Linux, la gestión de procesos y recursos, la automatización de tareas y la programación de *scripts* en *shell*.

Además, el capítulo aborda temas avanzados, como la administración de usuarios y permisos. Los estudiantes aprenden a utilizar herramientas como *vi* y *sed* para la edición de archivos de texto, y a gestionar el sistema operativo a través de la línea de comandos.

Al finalizar el capítulo, el estudiante puede manipular muchos tipos de información de forma sencilla y eficiente. Junto con este entorno de trabajo, el estudiante ha descubierto que se tiene acceso libre a una gran cantidad de datos biológicos que, al almacenar localmente en la propia máquina, se acelera el procesamiento y se reduce el tiempo de cálculo. En resumen, este núcleo de herramientas conforma una excelente aproximación para extraer nuevo conocimiento a partir de la información biológica disponible.

Actividades

1. Realizad la instalación de la versión más reciente de Ubuntu MATE dentro de una máquina virtual de Oracle VirtualBox. Para ello, primero debéis obtener una imagen ISO de Ubuntu MATE. Posteriormente, es necesario crear una máquina virtual vacía, insertar la imagen ISO de Ubuntu y ejecutar la instalación. Podéis emplear un administrador de la gestión de paquetes de software para configurar el sistema final resultante.

2. Averiguad las funciones que realiza el comando `fold` del terminal. Analizar las opciones disponibles para este comando. Después, diseñad un pequeño *script* que combine *gawk* con el comando `fold` para calcular la frecuencia de aparición absoluta y relativa de cada clase de nucleótido en una secuencia genómica almacenada en un fichero de texto guardado en formato FASTA. Evaluad el funcionamiento de vuestro protocolo sobre varias secuencias de ADN.

3. Averiguad las funciones realizadas por *Bioawk*. Determinad cuál es el comando origen de esta extensión, los formatos de datos biológicos que soporta y si es posible trabajar con ficheros comprimidos con *gzip* o con otros comandos compresores.

4. Estudiad el comando `sedl`. Para probar su eficacia, grabad una hoja de estilo de *MicrosoftExcel* en formato texto (seleccionar el tabulador como separador de campos). Una vez en Gnu/Linux, verificad con el comando `od` que este formato propietario efectivamente introduce como salto de línea los caracteres `\r` y `\n`. Finalmente, emplead el comando *sed* para únicamente mantener el carácter `\n` (el terminal trabaja en este formato).

5. Diseñad un *script* en el terminal que os permita realizar el análisis completo de una serie de ficheros de la clase `refGene_human.txt` almacenados en un directorio. Cada fichero debe contener en su propio nombre el organismo al que pertenece para evitar nombres de ficheros duplicados (p.e. txt). Para cada genoma podemos realizar las mismas preguntas mostradas en el caso de estudio de los materiales teóricos.

6. A continuación, suministramos un archivo FASTA y debéis contestar a las siguientes preguntas. El separador entre las dos columnas es el `\t`

```
$ cat hib.fasta
```

```
HiB_C1    TGTTTGTTGTCACTGACTGATGTTGTGGTCTGG

HiB_C2    TATATATTACTT

HiB_C3    TATATATAACTTATA

HiB_C4    TATATATAACTTATA

HiB_C5    TATATATTACTT
```

- Imprimid la línea que coincide con el patrón `HiB_C4`.

- Imprimid la columna 1, un punto y coma, y la columna 2.

- Usad el concepto de un operador condicional en la declaración de impresión de la forma *print* **CONDITION ? PRINT_IF_TRUE_TEXT : PRINT_IF_FALSE_TEXT** para identificar las secuencias con longitudes > 14.

- Intentad realizar el siguiente ejercicio. ¿Qué es lo que ocurre?

Se puede usar e1 después del último bloque {} para imprimir todo (1 es una notación abreviada para {print $0} que se convierte en {print}, ya que, sin ningún argumento, *print* imprimirá $0 por defecto), y dentro de este bloque, podemos cambiar $0, por ejemplo, para asignar el primer campo a $0 para la segunda línea (NR==2).

- Usad el comando `getline` para cargar el contenido de otro archivo además del que estás leyendo. Intenta, con el bucle *while*, cargar cada línea del fichero *fasta* en una variable: la *b*, por ejemplo.

7. Contestad a las siguientes preguntas después de descargar el siguiente fichero: https://ftp.ncbi.nlm.nih.gov/genomes/ASSEMBLY_REPORTS/assembly_summary_refseq.txt

- Los valores únicos de la variable categórica *assembly_level* se encuentran indicados en la columna #12, la cual muestra el estado del *ensamble*. ¿Cuáles son?
- Determinad el número de genomas por especie, que se encuentra en la columna #8. Luego, debéis mostrar únicamente las 10 especies con la mayor cantidad de genomas secuenciados.
- ¿Cuántos genomas completos hay del género *Mycobacterium*?
- Contad los genomas de *Salmonella*, *Pseudomonas* y *Acinetobacter* (por género) y presentad la lista ordenada por número decreciente de genomas.
- Determinad la longitud de una cadena: ¿Por qué estos dos comandos dan diferente información?

```
$ echo 'atattttGAATTtattGAATCAGGACC' | wc -c
```

```
$ echo 'atattttGAATTtattGAATCAGGACC' | awk 'END{print "El
oligonucleótido", $0, "tiene" lenght($0), "nucleótidos de longitud"}'
```

8. Eligid varios ficheros que contengan secuencias FASTA.

- El número de secuencias para un fichero (*awk*, autoincremento).
- Con un bucle determina el número de secuencias para todos los ficheros (*for and*, *awk*, autoincremento).

Ejercicios de autoevaluación

1. Enumerad los cuatro componentes básicos de una computadora.
2. Enumerad las etapas principales de la generación de un fichero ejecutable.
3. Enumerad las etapas en la vida de un proceso.
4. Enumerad las cuatro operaciones básicas con ficheros.
5. Citad cómo se codifican el directorio raíz, el directorio anterior y el actual.
6. Enumerad los tres dominios de los usuarios.
7. Describid las clases de permisos que se pueden asignar a un fichero.
8. ¿Para qué es utilizado el comando `man` en el terminal?
9. Enumerad los comandos del terminal para navegar por el sistema de ficheros.
10. Enumerad varios comandos del terminal para gestionar el sistema de ficheros.
11. Citad el comando para modificar los permisos sobre un fichero o un directorio.
12. Enumerad algunos comandos para acceder al contenido de un fichero.
13. ¿Cómo debemos acceder a ficheros previamente comprimidos desde el terminal?
14. ¿Cómo debemos recuperar el listado de los comandos usados en una sesión?
15. Enumerad los comandos adecuados para ordenar y combinar ficheros.
16. Describid el comando empleado habitualmente para buscar patrones de texto.
17. Citad la diferencia entre | o > en la comunicación entre procesos.
18. Discutid la diferencia entre los bloques BEGIN y END en el lenguaje de programación *gawk*.
19. Explicad las siguientes variables de *gawk*: `$1`, `$0`, `NR`, `NF`, `OFS` y `FS`.
20. ¿Por qué son enormemente útiles los protocolos de tareas automatizadas?

Solucionario

1. El procesador, la secuencia, los periféricos y el bus de comunicaciones.
2. Programación, compilación, enlace y ejecución.
3. En ejecución, en espera para ejecutarse, bloqueado.
4. Abrir, cerrar, leer, escribir.
5. El directorio raíz es «/», el directorio anterior es «..» y el actual es « .».
6. Existen tres dominios: usuario, grupo de trabajo y el resto de usuarios.
7. Existen tres permisos: lectura, escritura y ejecución.
8. Proporciona acceso al manual del sistema.
9. Los comandos *pwd*, *ls*, *cd*.
10. Los comandos *cp*, *rm*, *mv* y *mkdir*.
11. Es el comando *chmod*.
12. Los comandos *more*, *head*, *cat* y *tail*.
13. Usando los comandos *gzip* y *tar*.
14. El comando *history*.
15. Los comandos *sort*, *uniq* y *join*.
16. El comando *grep* realiza la búsqueda de patrones de texto.
17. El operando | envía la información a otro proceso, mientras que el operando > envía la información a un fichero, necesariamente.
18. El bloque BEGIN ejecuta esas instrucciones antes de procesar el fichero, el bloque END ejecuta las instrucciones precisamente después de finalizar este.
19. La variable $1 se refiere al primer atributo de cada registro o línea. La variable $0 contiene la línea actual en curso. La variable NR almacena el número de registros leídos, mientras que la variable NF cuenta el número de columnas presente en cada línea. OFS representa el carácter que se empleará para separar campos en la salida, mientras que FS realiza la misma función en la adquisición de los datos de entrada.
20. Porque permiten ejecutar un número (prácticamente) infinito de veces el mismo conjunto de tareas sobre múltiples grupos de datos sin asistencia del usuario.

Bibliografía

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 y 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 y D. M. Ritchie (1988). *C Programming Language* (2nd Edition). Prentice Hall. ISBN: 0131103628.

Brian W. Kernighan and y Rob Pike (1984). *Unix Programming Environment*. Prentice Hall. ISBN: 013937681X.

Cameron Newham (2005). *Learning the bash Shell, Third Edition*. O'Reilly Media. ISBN: 0-596-00965-8.

Christopher Negus (2015). *Linux BIBLE. The comprehensive, tutorial resource* (9th. Ninth Edition). Wiley. ISBN: 1118999878

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

John L. Hennessy and y David A. Patterson (2002). *Computer Architecture: A Quantitative Approach* (, 3rdrd Edition). Morgan Kaufmann. ISBN: 1558605967.

Linux Bash Shell Scripting Tutorial. http://bash.cyberciti.biz/guide/Main_Page

Steven Haddock y 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/

Entornos y contenedores

Autores: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve
PID_00298303
Primera edición: septiembre 2023

Introducción

A medida que la cantidad de datos en el área de ciencias de la vida y en otras áreas del conocimiento aumenta exponencialmente, surge la necesidad de estructurar y manejar sistemáticamente la información adquirida. Que los datos que manejamos sigan los principios FAIR (*) es fundamental para la correcta manipulación de los datos y de su trazabilidad a lo largo del ciclo de vida del dato. Además, hay dos procesos fundamentales, los cuales estudiaremos en los próximos temas, que son la escalabilidad y la reproducibilidad.

Tradicionalmente, a fin de minimizar el número de pasos manuales que se llevan a cabo en un análisis, los procesos se han unido programáticamente en los denominados *pipelines*. Esta concatenación de procesos normalmente se estructura en archivos (*scripts*). Este tipo de automatización normalmente conlleva una elevada dependencia en las versiones del programario instaladas y en la arquitectura local del ordenador en el que se procesa.

Como seguramente ya os habréis dado cuenta, lenguajes de programación como Python, *bash*, R, etc. ejecutan lo que se denominan paquetes. Los paquetes pueden contener programas que pueden ser utilizados para poder procesar los datos a analizar. Un paquete por sí solo normalmente no tiene todo el código necesario para realizar sus funciones y necesita de otros a fin de ejecutarlas. De esta forma se reutiliza código, minimizando el tiempo de desarrollo de un paquete y creando herramientas más robustas. Para poder gestionar las dependencias que un paquete tiene sobre otros aparecieron los gestores de paquetes (*apt, yum, Home Brew, pip*...).

En el proceso de utilización y actualización de los paquetes nos podemos encontrar con diversas dificultades. Durante el desarrollo de un paquete es posible que una función se vea modificada, alterando los datos de entrada o salida, o eliminada en una versión más reciente del paquete. En el momento en que distintos paquetes dependen de esta función se pueden producir problemas de compatibilidad y no es suficiente saber que un paquete depende de otro sino también si depende de una versión en específico. La actualización de un paquete puede representar que otro paquete ya no funcione. Eso conllevaría la necesidad de instalar dos versiones distintas del mismo software, hecho que muchas veces no es posible. Otra problemática asociada a las versiones de los paquetes es la reproducibilidad de resultados en sistemas distintos. Dos ordenadores podrían tener instalados los mismos paquetes y sus dependencias, pero con versiones distintas, produciendo resultados diferentes. En proyectos multicéntricos a veces es necesario ejecutar los mismos procesos en cada una de las instituciones implicadas, con la seguridad de que no habrá variabilidad en los resultados debido al procesamiento de los datos.

Así, debemos controlar:

- La utilización de los mismos paquetes y de sus dependencias.
- La implementación en otros entornos computacionales evitando que la configuración local altere el resultado final.

Para mantener una configuración estable y aplicable a diferentes infraestructuras computacionales, fundamentalmente hay dos estrategias a seguir: la creación de entornos fijos y/o la creación de contenedores.

Objetivos

1. Entender los conceptos y la utilidad de entornos y contenedores.

2. Aprender a crear y a manejar entornos Conda.

3. Saber crear y manejar contenedores Docker.

1. Conda

1.1. Introducción

Empezaremos por la creación de entornos, y como ejemplo utilizaremos Conda. Conda es un sistema de gestión de paquetes y de entornos *open source* compatible con los sistemas operativos Linux, Mac OS y Windows. No requiere privilegios de administrador para poderse utilizar.

Aunque inicialmente Conda se creó para la gestión de paquetes en Python, actualmente se pueden gestionar paquetes creados en diferentes lenguajes de programación, como R, Ruby, Scala, Java, JavaScript o C/C++.

Muchas veces hay confusión entre Conda, MiniConda y Anaconda (figura 1). El núcleo del gestor de paquetes y entornos es Conda, mientras que MiniConda es el instalador mínimo de Conda, además de combinar Python y unos paquetes básicos, y Anaconda engloba MiniConda y aumenta el número de paquetes preinstalados.

Figura 1. Esquema de las diferencias entre Conda, MiniConda y Anaconda.
Fuente: elaboración propia.

1. Conda

1.2. ¿Qué son paquetes y canales en Conda?

Un paquete es un archivo comprimido, sea un tar (.tar.bz2) o un .conda, que contiene:

- Librerías de sistema.
- Python y otros módulos.
- Ejecutables y otros componentes.
- Metadatos en el directorio info/.
- Una colección de ficheros que se instalan directamente.

Un paquete no necesariamente ha de contener todos estos elementos; por ejemplo, un paquete que únicamente contiene metadatos se denomina *metapackage*. Podéis encontrar un listado de los paquetes que gestiona Conda en https://anaconda.org/ para su instalación.

Cuando queramos instalar un paquete en un entorno Conda deberemos saber dónde localizarlo, y esa es la función de los canales. Los canales son URLs que nos dirigen a donde podemos encontrar los directorios que contienen los datos de los paquetes. Al instalar un paquete, el comando `conda` busca en un conjunto de canales que tiene por defecto. Si no se especifica lo contrario, Conda instalará los paquetes contenidos en estos canales definidos por defecto. Además de los canales por defecto, hay otro canal que ostenta un estatus diferencial. Este es el canal **conda-forge**. Este canal está administrado por la comunidad y puede tener algunas ventajas respecto a los canales por defecto gestionados por Anaconda Inc., como por ejemplo que los paquetes usualmente están más actualizados o que algunos paquetes no son accesibles desde los canales por defecto. Finalmente, destacar el canal **bioconda**, que se centra en paquetes desarrollados específicamente para el área de la bioinformática.

1. Conda

1.3. Entornos Conda

Un entorno Conda es un directorio que contiene una colección específica de paquetes Conda que has instalado. Si modificas un entorno, los otros entornos no se ven afectados y fácilmente puedes activar y desactivar entornos.

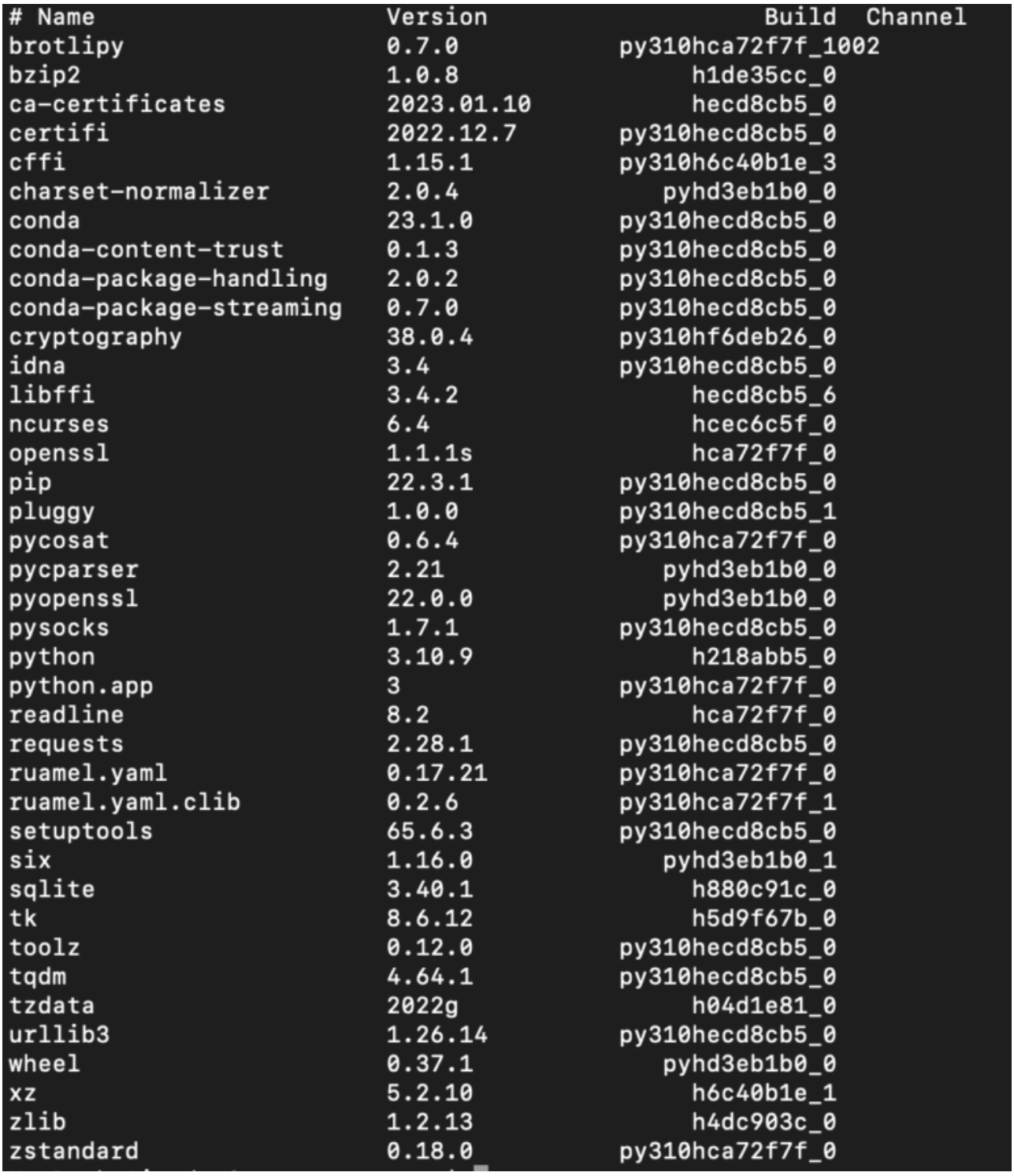
Para poder crear entornos Conda en primer lugar debéis instalar MiniConda o Anaconda, dependiendo de vuestras preferencias. Seguiremos el ejemplo utilizando MiniConda y la instalación en Linux/Mac OS. En este enlace podréis encontrar las instrucciones necesarias dependiendo del sistema operativo que utilicéis (https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html). Debéis descargaros el instalador dependiendo de vuestro sistema y, en el caso de Linux o Mac OS, ejecutar:

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

Para testar si la instalacion ha finalizado correctamente podéis escribir en el terminal

```
$ conda list
```

que devolverá el listado de paquetes instalados (figura 2). Recordad que si no se encuentra el comando conda seguramente es debido a que no está en la ruta especificada de comandos de vuestro terminal, pero lo podréis encontrar en la carpeta donde habéis instalado MiniConda o Anaconda, en 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
```

Figura 2. Salida del comando *conda list*.
Fuente: elaboración propia.

Conda tiene un entorno por defecto llamado *base*. No es recomendable instalar paquetes en este entorno. Si se desea iniciar un nuevo proyecto se deben crear nuevos entornos Conda.

1. Conda

1.4. Creación de entornos Conda

Es importante dar un nombre descriptivo al entorno para poder reconocer su contenido. En un *pipeline* estándar, podemos utilizar múltiples programas que pueden ser reutilizables en otros análisis. Debido a esta redundancia de procesos, es recomendable crear un entorno para cada herramienta y no crear entornos con múltiples programas. De este modo, un entorno estará definido por el software que contenga mediante el nombre, y su utilización será más sencilla que si un mismo entorno contiene múltiples paquetes, ya que será difícil determinar dónde se encuentra el programa que necesitas en un momento determinado. A veces también es recomendable no únicamente especificar el programa en el nombre del entorno, sino también la versión del mismo.

En primer lugar, instalaremos un paquete muy utilizado en Python como es `numpy`. Para crear un entorno utilizaremos el comando `create` y especificaremos la versión a instalar. Para saber qué versiones están accesibles podemos utilizar el comando `search`:

```
$ conda search numpy
```

Esto nos devolverá un listado de todas las versiones que están disponibles. Si no especificamos la versión del paquete a instalar, Conda intentará instalar la versión más nueva. Una vez seleccionamos la versión que necesitamos podemos crear nuestro nuevo entorno:

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

Utilizando la opción *-n* indicamos el nombre que queremos asignar al nuevo entorno.

Si quisiéramos especificar desde qué canal queremos instalar un paquete podemos utilizar la opción `channel`:

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

Conda no únicamente instala el paquete especificado, sino también sus dependencias. En el caso de *numpy*, por ejemplo, no hemos especificado que instalase Python, pero al ser una dependencia automáticamente se instala en el entorno especificado.

Si siempre utilizamos un set de paquetes podemos instalarlos conjuntamente:

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

Una vez creados los entornos podemos saber qué paquetes se han instalado utilizando el comando `list`:

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

1. Conda

1.5. Activar entornos Conda

Una vez hemos creado nuestro primer entorno Conda es hora de utilizarlo. Para obtener un listado de los entornos que tenemos en nuestro sistema utilizaremos el comando `env list`:

```
$ conda env list
```

Observaréis que además de nuestro entorno `numpy1-23-5` y `basic-analisis` tenemos el entorno **base**, que como os había comentado anteriormente está definido por defecto. Es recomendable no instalar software en este entorno.

Para poder utilizar uno de los entornos debemos activarlo:

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

Sabremos que el entorno se ha activado, ya que aparecerá su nombre entre paréntesis delante de la línea de comandos.

1. Conda

1.6. Desactivar entornos Conda en uso

Una vez finalizados los procesos requeridos en el entorno debemos cerrar el entorno. Para ello utilizaremos el comando `deactivate`:

```
$ conda deactivate
```

De esta forma volveremos al estado inicial, antes de la activación del entorno.

1. Conda

1.7. Instalar paquetes dentro de un entorno ya creado

Los entornos Conda pueden ser modificados *a posteriori* de su creación. Simplemente debéis activar el entorno y, una vez dentro, instalar un nuevo paquete. El comando `install` por defecto instala un paquete en el entorno activo.

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

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

En este caso se instalaría el paquete *pandas* en el entorno `numpy1-23-5`. Si el nuevo paquete no es compatible con la configuración del entorno activo, saltará un error y no podrá ser instalado.

Si queremos especificar el canal podemos hacerlo:

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

1. Conda

1.8. Renombrar entornos Conda

Si al instalar nuevos paquetes queremos especificarlo en el nombre del entorno podemos modificar el nombre asignado:

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

1. Conda

1.9. Eliminar entornos Conda

Si un entorno ya no se utiliza, es conveniente eliminarlo y, para ello, utilizaremos el comando `remove`:

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

1. Conda

1.10. Compartir entornos

En proyectos colaborativos es frecuente la necesidad de reproducir tareas en los distintos centros. Para ello se deben crear entornos agnósticos de sistema operativo y plenamente compatibles. Conda utiliza YAML (YAML *Ain't Markup Language*) como archivos de entorno que nos permitirán importar y exportar entornos.

Por convención, los archivos de entorno en Conda se llaman *environment.yml.*

Si en nuestro directorio de trabajo ejecutamos:

```
$ conda env create
```

automáticamente, Conda buscará el archivo *environment.yml.* Si no lo encuentra, saltará un error. Si el archivo de entorno tiene otro nombre, lo podemos especificar de la siguiente manera:

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

Podéis observar la estructura de un archivo de entorno en la figura 3, donde se pueden apreciar tres apartados:

- `name`: nombre del entorno que se creará si no se especifica lo contrario.
- `Channels`: canales a utilizar.
- `dependencies`: paquetes a instalar con relación `canal:paquete:versión`

```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. Ejemplo de archivo de entorno YAML.
Fuente: elaboración propia.

Para poder generar un archivo YAML de un entorno que nosotros hemos creado, ejecutaremos:

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

Especificamos el nombre del archivo con la opción `--file`

Para asegurar que el entorno puede ser reproducible independientemente del sistema operativo, hace falta especificar la opción `--from-history`:

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

2. Docker

2.1. Introducción

Una vez hemos visto los entornos Conda, nos introduciremos en otra metodología para controlar los procesos que utilizamos: los contenedores (*).

Los contenedores son sistemas de virtualización que contienen todas las herramientas necesarias para ejecutar un software. Muy a menudo se comparan las máquinas virtuales con los contenedores. La diferencia más importante es que las máquinas virtuales virtualizan toda una máquina hasta las capas de *hardware*, mientras que los contenedores únicamente virtualizan la capa de software encima del sistema operativo. Esta característica los hace más ligeros y fáciles de modificar.

Aunque los contenedores no son una tecnología nueva, su aplicación de manera extensa empezó con la aparición de Docker en 2013. La popularización de estas aplicaciones introdujo la complejidad de administrar cientos o miles de contenedores, y por ello apareció lo que se conoce como la orquestación de contenedores. Aunque a lo largo del tiempo han aparecido distintas plataformas de orquestación, inclusive una del mismo Docker, como es Docker Swarm, Google creo en 2014 Kubernetes, de código abierto, que se ha convertido en el software preferido de muchas empresas y se ha consolidado como un estándar. Las plataformas de orquestación se encargan de reiniciar las aplicaciones si fallan, de equilibrar la carga de trabajo, de escalar automáticamente, de implementar sin tiempo de inactividad, etc.

Aunque en sistemas HPC (High Performance Computing) se utiliza más Singularity, en este apartado de contenedores nos centraremos en cómo utilizar Docker.

2. Docker

2.2. Contenedores Docker

En primer lugar, necesitaremos instalar Docker en nuestro ordenador. La forma más sencilla es utilizando Docker Desktop (https://www.docker.com/products/docker-desktop/).

Para saber que tienes Docker instalado correctamente en el terminal puedes interrogar su versión:

```
$ docker -v
```

El *output* te indicará la versión y el *build* que tienes instalados.

Y también:

```
$ docker system info
```

Si saltase algún error se debería comprobar que el proceso de instalación estuviera finalizado correctamente y que Docker Desktop estuviera abierto.

Para crear un contenedor necesitas lo que se denominan imágenes. Las imágenes son los moldes de los contenedores, son las recetas/instrucciones para crear los contenedores.

2. Docker

2.3. Descargar contenedores creados

En primer lugar, miraremos si tenemos alguna imagen en nuestro sistema. En teoría, si es la primera vez que utilizas Docker, te debería aparecer una lista en blanco al utilizar:

```
$ docker image ls
```

Empezaremos por el contenedor más sencillo y lo bajaremos directamente:

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

Hay el proceso de descarga y si todo ha funcionado correctamente al repetir el comando de listado de imágenes os debería aparecer la imagen `hello-world`.

La imagen de «hello-world» procede de Docker Hub (*), un repositorio de imágenes.

Seguidamente, ejecutaremos el contenedor que se cree a partir de la imagen `hello-world` mediante:

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

Una vez ejecutado recibiréis un mensaje de parte del equipo 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. Mensaje de bienvenida del equipo de Docker al ejecutar «hello-world».
Fuente: elaboración propia.

Cuando se ejecuta un contenedor suceden tres procesos:

- Inicializa el contenedor a partir de la imagen.
- Se ejecuta la acción preestablecida del contenedor si esta existe.
- Una vez la acción ha finalizado el contenedor se para.

En el caso del ejemplo anterior la acción preestablecida era imprimir el mensaje de bienvenida, pero la acción realizada puede ser mucho más compleja.

Además de poder ejecutar los comandos predeterminados por el contenedor también le podemos pasar comandos o entrar en modo interactivo. Para probar estas opciones ejecutaremos el contenedor Alpine que contiene una distribución de Ubuntu muy simple.

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

En este comando utilizamos la opción `it` para ser interactivo y `sh` nos especifica que el terminal que queremos utilizar es *bash*.

Veréis que la línea de comandos cambia a:

```
/ #
```

Podréis comprobar que ahora estáis dentro de Alpine y al ejecutar

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

os mostrará la versión de Alpine que os habéis bajado.

De esta manera, una vez ejecutado el contenedor no se ha finalizado como habíamos observado anteriormente, sino que se mantiene activo y responde a los comandos que introduzcamos.

Para poder salir del contenedor y finalizar su ejecución podéis introducir el comando `exit`.

2. Docker

2.4. Eliminar imágenes y contenedores Docker

Siempre es recomendable mantener una buena gestión de imágenes y contenedores, ya que van ocupando espacio innecesariamente si no se utilizan. Así aprenderemos a eliminar imágenes y contenedores.

Para poder eliminar imágenes, necesitaremos saber la Image ID. Para ello utilizaremos el siguiente comando:

```
$ docker image ls
```

Nos aparecerá un *output* similar al siguiente:

Tabla 1.

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

Fuente: elaboración propia.

Así para eliminar una imagen simplemente se debe especificar el Image ID tal que así:

```
$ docker image rm feb5d9fea6a5
```

O utilizando su nombre:

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

Muchas veces nos podemos encontrar con un mensaje de error:

```
Error response from daemon: conflict: unable to delete feb5d9fea6a5 (must
be forced) - image is being used by stopped container 06dda9650983
```

Esto nos está indicando que hay un contenedor que aún está activo o que aún no ha sido limpiado que está o ha utilizado esta imagen. Para ello primero debemos eliminar los contenedores que están impidiendo la eliminación de esta imagen.

Primero listamos los contenedores activos:

```
$ docker container ls
```

O, de manera similar, podemos utilizar:

```
$ docker ps
```

Es posible que no nos devuelva ningún contenedor activo. En realidad, el mensaje de error previo nos indicaba que el contenedor había parado; por lo tanto, tiene sentido no encontrar el contenedor en este listado. Así necesitaremos ver los contenedores tanto activos como los que recientemente se han parado:

```
$ docker container ls -all
```

Aquí sí que podemos ver el contenedor anteriormente especificado como enlazado a «hello-world»:

Tabla 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

Fuente: elaboración propia.

Para evitar que se vayan acumulando los contenedores se puede añadir al comando `run` una opción de eliminar automáticamente una vez finalizado:

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

Si no hemos especificado la opción `rm`, debemos eliminarlo manualmente utilizando como referencia el Container ID:

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

Si se elimina correctamente nos devolverá el Container ID en el terminal. Se puede borrar más de un Container ID simultáneamente.

Si tienes varios contenedores asociados a una imagen y quieres borrarlos todos utilizando un patrón puedes:

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

En este caso utilizamos el comando `xargs` para controlar una lista de argumentos para poder ser eliminados mediante `rm` de Docker.

Ahora veréis que al listar los contenedores ya no aparece el contenedor, y si probamos de eliminar la imagen como previamente habíamos intentado lo hará sin problemas.

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

```
$ docker image ls
```

Si se quiere eliminar todos los contenedores existentes podemos utilizar:

```
$ docker container prune
```

De esta manera todos los contenedores existentes serán eliminados y si queremos volver a reconectarlos deberemos iniciarlos de nuevo.

2. Docker

2.5. Parar contenedores en Docker

Iniciar o parar un contenedor no es lo mismo que iniciar o parar un proceso. Para finalizar un contenedor, Docker proporciona los comandos `stop` y `kill`. Aunque parezca que ambos comandos hagan lo mismo, internamente su ejecución es distinta. Para parar un proceso, tanto podemos utilizar el ContainerId como el nombre del contenedor.

El comando `stop` para el contenedor de una manera menos agresiva que `kill`. Esto es debido al tipo de señal que se envía al contenedor. stop envía una señal SIGTERM, la cual se puede bloquear o parar, mientras que `kill` envía una señal SIGKILL que no se puede gestionar. Si en un tiempo prudencial el comando stop no ha parado el contenedor, Docker automáticamente envía una señal SIGKILL. Por defecto ese tiempo son 10 segundos, pero si queremos modificarlo podemos utilizar la opción *-t* expresada en segundos. Así:

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

Docker pararía el proceso mediante una SIGKILL pasador 77 segundos.

Como hemos visto en el apartado anterior también podríamos utilizar el comando `rm` para finalizar un contenedor. La diferencia reside en que el comando `rm` elimina el proceso y no lo podemos visualizar en la lista `docker ps -a`, mientras que parando el contenedor lo podemos mantener y reutilizar posteriormente.

Otra opción interesante es pausar el contenedor. Mientras que si paramos un contenedor liberamos los recursos de memoria y CPU, al pausarlo únicamente liberamos CPU.

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

2. Docker

2.6. Buscar contenedores en Docker Hub

Un recurso que ya hemos utilizado para el contenedor «hello-world» es Docker Hub. En este repositorio podemos encontrar muchos contenedores ya creados. Muchos de ellos ya han sido construidos y testeados por los mismos desarrolladores del software que estáis buscando. Como ejemplo iremos a la página del *variant caller* GATK (https://hub.docker.com/r/broadinstitute/gatk). Aquí encontraréis las instrucciones para podéroslo bajar. Si queréis una versión determinada al bajar la imagen necesitáis especificarla. Para ello se utilizan los *tags* igual que como vimos en el apartado de Conda. En la página de GATK tenéis una pestaña donde tenéis las distintas versiones indicadas por *tags*.

Para indicar qué versión queremos utilizar lo indicaremos con los «`:`»:

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

Si no especificamos la versión se bajará la imagen más reciente denominada `latest`.

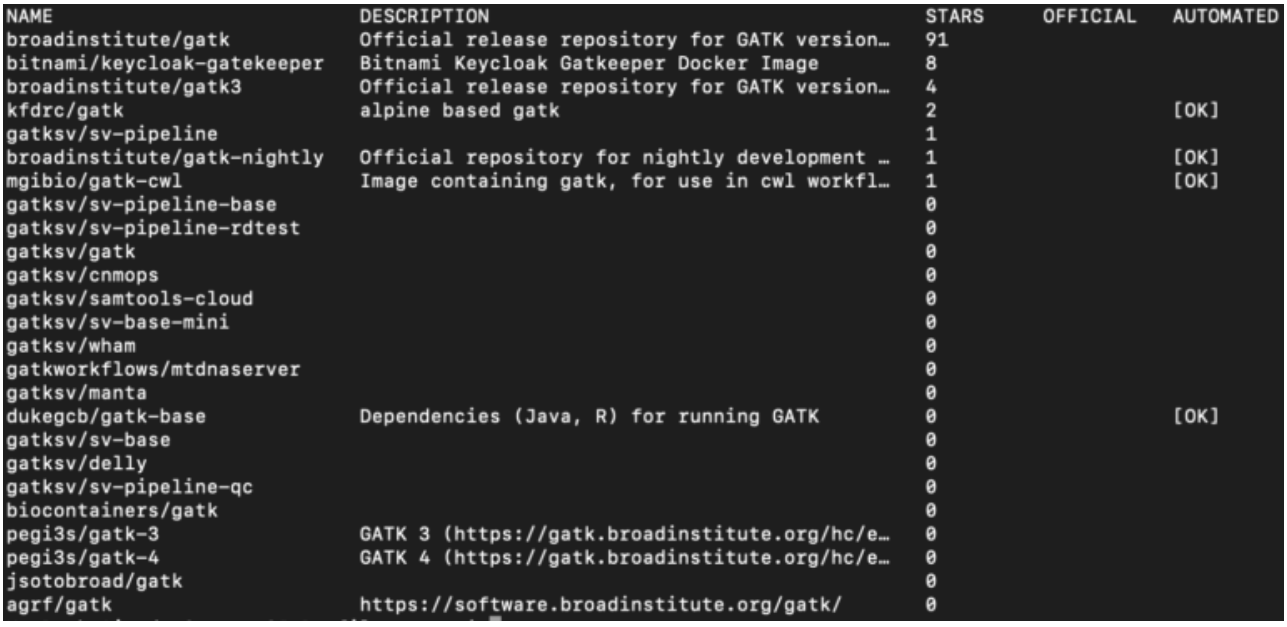
Como podéis apreciar en el comando `pull` que hemos utilizado para bajar GATK delante del nombre de `gatk` tenemos el nombre de la institución que lo ha creado y hecho público. En este caso el Broad Institute. Si este nombre previo no aparece, indica que los desarrolladores son el propio equipo de Docker.

Es importante tener en cuenta que cualquier persona puede crear una cuenta en Docker Hub, por lo tanto, es importante mantenerse cauteloso al bajar software de fuentes no contrastadas. Debéis procurar bajar imágenes directamente de los desarrolladores o de comunidades establecidas. Docker mantiene una serie de imágenes referenciadas como Docker Official Images, las cuales han sido analizadas y proporcionan repositorios básicos para la comunidad, como por ejemplo Ubuntu o Centos.

Otra opción para buscar imágenes es el comando `search`. En este caso, al utilizar

```
$ docker search GATK
```

obtendremos un listado de imágenes donde encontraríamos repositorios con 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 búsqueda de imágenes que contengan GATK.
Fuente: elaboración propia.

2. Docker

2.7. Crear tu propia imagen Docker

Si tu búsqueda en Docker Hub ha sido infructuosa o requieres una imagen específica para tus necesidades, la solución es crearla tú mismo.

En primer lugar, instalaremos software en una sesión interactiva. Para ello utilizaremos la imagen de Alpine que previamente habíamos creado e intentaremos instalar en paquete de *python numpy*. Alpine no es la distribución en la que querríais basaros para desarrollar vuestros proyectos; seguramente Ubuntu o Debian sean unas elecciones más apropiadas. Antes de empezar una imagen es importante que sepáis qué necesitaréis instalar, cuáles son vuestros requerimientos y escoger la distribución más apropiada. También es una buena práctica no instalar muchos programas en cada imagen, ya que serán más difíciles de crear y mantener. Es la misma filosofía que aplicábamos a los entornos Conda.

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

Seguidamente comprobaremos si tenemos Python instalado:

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

Y podemos observar que no está instalado.

Alpine tiene un gestor de paquetes llamado *Alpine Package Keeper* (apk) para instalar software. Dependiendo de la distribución de Linux que os hayáis instalado podríais utilizar `apt-get, yum, zypper`...

En primer lugar, instalaremos Python y otros paquetes necesarios, y posteriormente el paquete *numpy*.

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

/ # pip install numpy
```

Si ahora entramos en Python podemos comprobar que se ha instalado correctamente el paquete *numpy*.

Una vez salgas de este contenedor los cambios no se habrán guardado. Para crear tu propia imagen lo más recomendable es utilizar un Dockerfile.

Un Dockerfile es un archivo de texto con la siguiente estructura mínima:

- FROM <Imagen preexistente>
- RUN <Comandos para instalar desde la línea de comandos>
- CMD <Comandos que deben correrse por defecto>

Las instrucciones que se lanzan por defecto (CMD) tienen una estructura definida. Solo puede haber una línea de CMD en el Dockerfile; si hubiera más de una se ejecutaría la última.

Así si quisiéramos reproducir el contenedor anterior deberíamos crear un archivo nombrado *Dockerfile* de la siguiente manera:

```
FROM alpine

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

RUN pip install numpy

CMD python3 --version
```

Una vez tenemos el Dockerfile definido crearemos la imagen:

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

La opción `-t` nos indica el nombre que le queremos dar a nuestra imagen. Siguiendo la nomenclatura anteriormente mencionada indicamos el autor y el paquete. El «.» nos indica que Dockerfile está en la misma carpeta donde estamos ejecutando los comandos; debemos especificar la ruta al directorio del Dockerfile.

Ahora podréis ver la nueva imagen creada:

```
$ docker image ls
```

Tabla 3.

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

Si corremos el contenedor Docker nos devolverá la versión de Python que tenemos instalada en la imagen:

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

Si entrásemos comandos en la línea de ejecución del contenedor, estos son los que se utilizarían en detrimento de los que estuviesen en el Dockerfile. Así:

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

Te indicará la localización de Python3 en el *filesystem* del contenedor.

2. Docker

2.8. Renombrar imágenes

En realidad, lo que crearemos es una copia de una imagen con un nuevo nombre. Para ello haremos:

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

Si la imagen es susceptible a evolucionar es importante mantener una numeración por sus *tags*, como hemos visto en Docker Hub:

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

Si ahora miráis las imágenes que tenéis creadas podréis ver que en la columna *tag* tendréis la versión que hemos creado ahora y la *latest*, que es por defecto, si no se especifica uno.

Tabla 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

Fuente: elaboración propia.

2. Docker

2.9. Ejecución de *scripts*

Como hemos visto, la creación de un contenedor es relativamente fácil, pero dependiendo de vuestras necesidades es posible que los ejemplos anteriores se queden cortos. Por ello extenderemos los conocimientos necesarios para crear imágenes más complejas.

En primer lugar, utilizaremos el contenedor con Alpine que hemos creado anteriormente para ejecutar un *script* en Python.

Si directamente pasamos los parámetros al iniciar el contenedor, nos devolverá un error, ya que en su sistema el archivo no se encuentra:

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

Siendo `num.py`:

```
import numpy as np

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

print (a)
```

El error siendo:

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

Para que funcione este comando necesitaremos enlazar los dos sistemas, el nuestro propio y el del contenedor Docker. Para ello utilizaremos el comando mount especificando dónde está nuestro archivo y posicionándolo en el sistema del contenedor.

```
$ docker container run --rm --mount type=bind,source=${PWD},target=/temp
uoc/alpine-numpy python3 /temp/num.py
```

Hay distintos tipos de mount. En este caso utilizaremos el modo *bind*, que es el que nos interesa para este ejemplo. Con el *source* especificamos el directorio donde se ubica el *script*, podemos utilizar la variable ${PWD} si lanzamos el contenedor desde el mismo directorio donde está el *script*, y *target*, que especifica dónde lo localizaremos en el contenedor. Es importante que cuando ejecutamos Python3 especifiquemos la carpeta donde localizamos el *script*, en este caso /temp.

De esta manera nos retornará:

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

2. Docker

2.10 Volúmenes Docker

En el ejemplo anterior hemos utilizado `mount` para unir los dos sistemas. `Mount` depende del sistema en el cual se ejecuta, mientras que los **volúmenes** son nativos de Docker. Los volúmenes pueden compartirse entre contenedores y persisten más que los contenedores.

En el ejemplo anterior, podríamos pasar un volumen al contenedor con la opción `-v`:

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

Definimos la carpeta local $(PWD) y dónde se localiza en el contenedor /temp.

Para crear un volumen con nombre `data_set` y asignarle una carpeta local:

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

En este caso podremos lanzar:

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

Podemos visualizar los distintos volúmenes:

```
$ docker volume ls
```

... inspeccionarlos

```
$ docker volume inspect data_set
```

... y eliminarlos:

```
$ docker volume rm data_set
```

2. Docker

2.11. Integrar datos en el contenedor Docker

Muchas veces necesitaremos utilizar un *input* de manera sistemática en un contenedor. Por ejemplo, si queremos hacer un *variant call* utilizando GATK, los genomas de referencia siempre serán los mismos y tal vez es interesante guardarlos dentro del contenedor para asegurarnos la reproducibilidad de los resultados a fin de que no dependa de la referencia utilizada. Para un caso más simple introduciremos nuestro *script num.py* en el contenedor. Para ello crearemos una nueva imagen modificando el Dockerfile.

Añadiremos una nueva línea:

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

Estaremos haciendo una copia del *script* en el home del contenedor. Hacemos otra imagen:

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

Si ahora entramos de forma interactiva dentro del contenedor y listáis los archivos dentro de /home encontrareis el archivo *num.py*:

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

El orden de los comandos en el Dockerfile es importante. Es recomendable hacer los comandos COPY después de los RUN, ya que al hacer el *build*, Docker va por orden, y si en algún momento queremos añadir otro *script* en lugar de *num.py*, si el COPY está al final del proceso, Docker utiliza los RUNs que tiene en memoria y no ha de construir la imagen de cero, y el proceso es mucho más rápido. Docker va línea a línea, y si esa capa o conjunto de capas ya la tiene en memoria *caché* agiliza el proceso no reinstalándolas.

Si los datos que queremos introducir dentro de nuestra imagen están en internet, directamente podemos copiarlos utilizando RUN. Podríamos añadir estas líneas en el Dockerfile como ejemplo:

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

Por defecto, el archivo se copia en el *root* del sistema del contenedor. Si quisiéramos moverlo a otra localización podríamos especificarla:

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

y la copiaría en nuestra carpeta /home.

Ahora que sabemos cómo introducir el *script* dentro del contenedor, podemos crear un nuevo contenedor que corra el *script* automáticamente. Debemos substituir CMD del Dockerfile a:

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

De esta manera creamos una nueva imagen:

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

y ejecutamos directamente:

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

y nos devuelve el resultado del archivo *num.py*.

Si queremos introducir un nuevo *script* de Python (*atcg.py*) donde su resultado dependa de un argumento,

```
import sys

filename = sys.argv[1]

filenumb = len(filename)

print (filenumb)
```

lo copiaremos mediante Dockerfile:

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

y crearemos una nueva imagen:

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

Si lo corremos directamente, tendremos el resultado del CMD (en nuestro caso, la versión de Python); mientras que si añadimos argumentos al comando `run`, sobrescribe CMD y obtenemos el resultado del nuevo *script*:

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

Si quisiéramos tener este nuevo *script* como los comandos por defecto deberíamos cambiar en el Dockerfile la línea de CMD por los valores por defecto y crear una nueva línea nombrada ENTRYPOINT para localizar el *script*:

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

CMD ["ACTG"]
```

También podemos añadir antes que ENTRYPOINT y CMD una entrada para definir el directorio de trabajo:

```
WORKDIR /home
```

De esta manera, si creamos una nueva imagen `uoc/alpine-entry` y ejecutamos el contenedor directamente,

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

nos devolverá «4» la longitud de la línea «ACTG» ubicada por defecto en el Dockerfile. Si ahora al final del comando `run` añadimos otro *input*, este sustituirá al de CMD:

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

Devolverá «8».

La relación entre ENTRYPOINT y CMD la podemos observar en la siguiente tabla.

Tabla 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

Fuente: elaboración propia.

Resumen

En este apartado habéis aprendido cómo crear y utilizar entornos Conda y contenedores Docker. Utilizar este tipo de herramientas es muy importante en la reproducibilidad de procesos y resultados. Tanto si has de compartir tu código con otras personas como si debes repetir el mismo procedimiento con otras muestras en un futuro, es crucial poder tener un seguimiento de las herramientas utilizadas. Estos procedimientos mejoran la calidad de la investigación publicada y facilitan el proceso de revisión por parte de investigadores externos. Actualmente prácticamente todos los programas dependen de otros paquetes de programas, los cuales evolucionan independientemente. La modificación de una dependencia puede conllevar a diferentes resultados de un programa o inclusive al mal funcionamiento de este. Por ello es importante mantener los programas y sus dependencias tal y como se utilizaron para poder asegurar la correcta reproducibilidad. La utilización de pequeños entornos y contenedores facilita el mantenimiento y el replicado de procesos. Además, tener los programas aislados del sistema general permite la eliminación y actualización de estos de una manera más sencilla y limpia. Algunos programas pueden necesitar ciertas dependencias incompatibles con otras ya instaladas en el sistema, y por ello la creación de entornos o la utilización de contenedores facilita el trabajar en distintas configuraciones en el mismo sistema.

Actividades

Conda

1. Cread un nuevo entorno de nombre «UOC-bioinformatics» que contenga Python 2.7.

2. Instalad en este entorno ya creado R y el paquete Tidyverse en su última versión.

3. Enumerad los paquetes instalados en el entorno de la actividad anterior.

4. Instalad el paquete Scipy en un entorno nuevo a través del canal Bioconda.

5. Cread un archivo «environment.yml» de este último entorno.

Docker

1. Bajaos la imagen de Ubuntu de Docker Hub, entrad en el contenedor de forma interactiva y determinad la versión de Ubuntu descargada.

2. Repetid el procedimiento anterior con la versión previa a la *latest*.

3. Cread un Dockerfile donde añadáis la última versión de R al sistema operativo Centos.

4. Cread un volumen Docker donde le podáis pasar al contenedor un *script* en *bash* que retorne «Hello World».

5. Repetid el procedimiento anterior modificando el mensaje a través del terminal al correr el contenedor y que retorne «Hi, I'm back».

Bibliografía

Björn Grüning y otros (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 y otros (2020). FAIR Computational Workflows. *Data Intelligence,* 2, p. 108-121.

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

Serghei Mangul y otros (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 y otros (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 y otros (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 y otros (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 y otros (2015). Big Data: Astronomical or Genomical? *PLoS Biol*ogy, *13*, e1002195.

Workflows

Autores: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve
PID_00298303
Primera edición: septiembre 2023

Introducción

Los *pipelines* tradicionales están muy ligados a las infraestructuras de computación locales donde se ejecutan. Estos no tienen la capacidad de resumir un proceso que se haya parado, tienen poca documentación, no cuentan con una trazabilidad de los parámetros y versiones de paquetes utilizados y requieren de instalación manual, lo cual impide una fácil distribución de este. Para poder solucionar estos inconvenientes se han creado los Workflow Managers. Estos permiten la utilización de *pipelines* de análisis complejos en distintos entornos de computación asegurando la máxima reproducibilidad de los procesos ejecutados.

Varios Workflow Managers se han desarrollado específicamente para los campos de investigación y salud integrando entornos, contenedores y computación en la nube.

Hay cinco características que hacen a los Workflow Managers herramientas de gran utilidad:

1. Reproducibilidad. La utilización de entornos y contenedores asegura una apropiada reproducibilidad de los procesos ejecutados.

2. Portabilidad. Es una de las grandes ventajas de la utilización de Workflow Managers, ya que crea los flujos de trabajo necesarios para poderse exportar a cualquier entorno computacional. Muchos de ellos permiten la fácil migración a distintos entornos, inclusive de alta computación y servicios en la nube. Es más, es posible la interacción directa con orquestadores como Kubernetes o DockerSwarm.

3. Escalabilidad. Ser capaz de manejar y analizar datos con una complejidad creciente es cada vez más común. En este sentido hay dos aspectos que deben tenerse en cuenta: el manejo eficiente de los recursos y ser capaz de utilizar datos más complejos y de mayor tamaño. La mayoría de Workflow Managers implementan la paralelización en diversos pasos, sea mediante gestor de colas o *scheduling* estática o adaptativa. La paralelización puede producirse a nivel de datos, procesos o *pipelines*. Una asignación dinámica de los recursos permite que los procesos más intensivos no se vean afectados respecto a los que no requieren tantos. Este balanceo minimiza cuellos de botella y reduce los tiempos de computación. Los recursos pueden asignarse específicamente para cada paso del flujo de trabajo.

4. Robustez. Muchos *pipelines* requieren de procesos complejos y de larga duración. En el posible evento de la interrupción del *pipeline* en algún proceso debido a un error, sea programático o por la ausencia de un *input* requerido, los Workflow Managers son capaces de resumir el proceso desde el lugar donde hubo el último paso correcto, resultando en el ahorro en la utilización de recursos y tiempo. Este proceso se consigue mediante la producción de archivos y resultados intermedios, siendo comparados con los resultados esperados. Este proceso genera un aumento en las necesidades de almacenamiento, pero comporta una ventaja sustancial en el caso de tener la necesidad de una reentrada en el *pipeline*.

5. Modularidad. La compartimentación de los procesos permite un gran dinamismo en la actualización de ciertos pasos del proceso, así como de la introducción de puntos de control para cada etapa. La modularidad también permite la reutilización de un proceso en varios *pipelines* simultáneamente.

Finalmente, indicar que algunos Workflow Managers también tienen recursos para aumentar la seguridad en la ejecución de los procesos, como la validación del origen de los datos o utilizar autenticación de usuarios.

Objetivos

1. Ofrecer una visión general de los Workflow Managers.

2. Introducir a los lectores a Nextflow.

1. Diferentes tipos de *workflows*

La implementación de un *pipeline* en un Workflow Manager requiere de la definición precisa de dónde se extraen los datos iniciales (*input*), cuál es el flujo entre las distintas herramientas y cuál es el resultado final (*output*). Para poder estructurar los distintos parámetros se utilizan lenguajes de dominio específico (DSL). La utilización de DSLs aumenta la portabilidad y la escalabilidad. Nextflow y Snakemake son dos de los ejemplos más populares de *workflows* que utilizan DSLs propios en el ámbito de la bioinformática. La diversidad de DSLs con su propia estructuración y nomenclatura propicia la reducción de interoperabilidad entre los distintos *workflows*. Para mitigar esta disparidad se han creado unas especificaciones que añaden un nivel mayor de abstracción, permitiendo un marco común entre los distintos *workflows*. Ejemplos de especificaciones para *workflows* son el Common Workflow Language (CWL) y Workflow Description Language (WDL). CWL prioriza la portabilidad y la reproducibilidad, mientras que WDL tiene como principal objetivo reducir la curva de aprendizaje mediante un lenguaje más comprensible. CWL define los *pipelines* utilizando ficheros YAML, mientras que WDL utiliza sus propios ficheros descriptivos. Algunos Workflow Managers, como *cwltool* o Cromwell, se han creado basándose en estas especificaciones, mientras que otros como Snakemake implementan opciones de exportación a estas especificaciones.

La utilización de Workflow Managers para la creación de *pipelines* bioinformáticos está creciendo en popularidad. Distintos *workflows* están disponibles para la comunidad variando en su flexibilidad y facilidad de uso (https://github.com/pditommaso/awesome-pipeline). Mientras que algunos *workflows* necesitan de un conocimiento avanzado de programación, otros tal que Galaxy, KNIME o BioWorkflow implementan una interfaz gráfica para facilitar su utilización. Aunque el desarrollo de sistemas de *workflow* se inició a principios de los años noventa, ha sido la capacidad de correr *pipeline*s en clústeres de computación o en la nube lo que ha facilitado su uso más generalizado.

2. Nextflow

2.1. Introducción

Como ejemplo de Workflow Manager en esta sección utilizaremos uno de los programas más extendidos con una comunidad creciente y con mayor soporte como es Nextflow.

En primer lugar, debemos instalar Nextflow siguiendo las instrucciones directamente de su página web (https://www.nextflow.io/).

Nextflow combina tres elementos: un entorno de ejecución, un programa específico para lanzar otros programas y un DSL específico. La estrategia que implementa Nextflow en la creación de *pipelines* es la de crear módulos para cada paso en el *pipeline* y vehicularlos e interconectarlos a través de canales. Una de las características más importantes de Nextflow es la reutilización de estos módulos para distintos estadios de un mismo *pipeline* o entre distintos *pipelines*. La ejecución de los distintos módulos no es lineal, sino que depende de la disponibilidad de los *inputs* para ejecutarse. Si dos módulos independientes requieren los mismos *inputs* o *inputs* independientes se pueden ejecutar simultáneamente para después poder converger en otro posteriormente si se requiere.

2. Nextflow

2.2. DSL2

Nextflow utiliza como DSL una extensión del lenguaje de programación Groovy. A partir de la versión 20.07.1 de Nextflow se actualizó la sintaxis del DSL original creando DSL2. A lo largo de esta sección utilizaremos esta nueva implementación, DSL2. Esta actualización, entre otras muchas características, permite la utilización de más de un *script* para definir el *workflow*, a diferencia de DSL1, en el que se acumulaba todo en un único *script*. En el código de Nextflow se debe especificar la utilización de DSL2, ya que por defecto utiliza DSL1. Para ello se indicará al inicio del *script*:

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

2. Nextflow

2.3. Estructura de *workflows*

Los *workflows* creados para Nextflow contienen tres partes diferenciadas: procesos, canales y *workflows*. Un proceso ejecuta una tarea. Cada proceso es independiente del otro y puede tener más de un canal de entrada y de salida. Un canal es un sistema de colas asíncrono que permite el flujo de información entre procesos (figura 1). Para juntar los distintos procesos y su flujo de ejecución (canales) se resume en un apartado específico en el *script* que se denomina *workflow*.

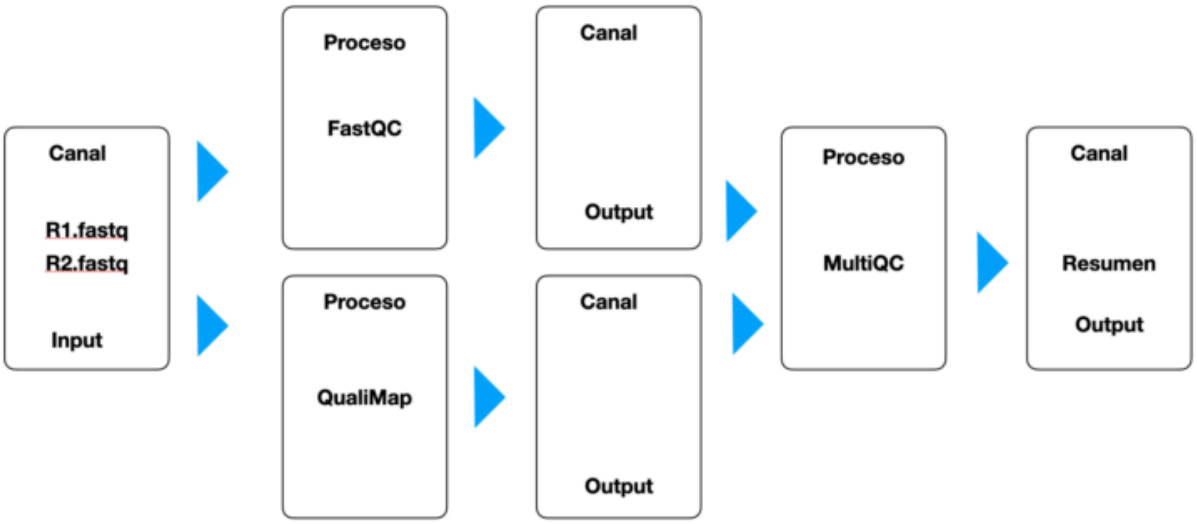


Figura 1. Ejemplo de esquema de *workflow* en Nextflow.
Fuente: elaboración propia.

Nextflow diferencia los comandos que se ejecutaran dentro de un proceso y quién será el encargado de ejecutarlos. Esto permite tener un marco general que describa qué se quiere hacer independientemente de las herramientas que se utilizaran para ejecutarlo. Esta estructura permite lanzar un proceso en distintos entornos computacionales variando simplemente un fichero de configuración, el cual define los ejecutores específicos del entorno donde uno se encuentre.

2. Nextflow

2.4. «Hello world»

Empezaremos por un *script* sencillo (helloworld.nf) donde imprimimos una frase, por ejemplo «Hello world».

```
nextflow.enable.dsl=2


params.str = 'Hello world!'


process printStr {

 output:

    stdout


  """
echo '${params.str}'
  """
}


workflow {

 printStr | view ( )

}
```

- En primer lugar, al *script* `nf` le indicaremos que utilizamos DSL2.
- Crearemos un parámetro al cual nombraremos str y le asignaremos la frase «Hello world!». El nombre del parámetro está precedido por un punto y la palabra `params`. En esta sección, al ser un único parámetro no hace falta crear un canal, ya que Nextflow le asigna directamente un canal *value*. Hay distintos tipos de canales, como veremos más adelante, y el canal *value* es el más simple.
- El siguiente segmento define un proceso al cual nombraremos printStr. La nomenclatura de los procesos es *process* seguido del nombre que le asignamos. Los procesos están formados de tres partes: *input*, *output* y *script*. En este caso, le indicamos que el *output* sea el estándar, y al ser un valor simple tampoco hace falta que definamos un canal específicamente. Dentro del proceso definiremos qué queremos hacer. Imprimimos el parámetro str.
- Finalmente asignamos la última sección al flujo del *workflow*.

Una vez creado el *script* lo ejecutaremos:

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

El resultado se muestra en 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. Ejecución del *script* helloworld.nf.
Fuente: elaboración propia.

Como se puede observar, se ha ejecutado un proceso `printStr` y se visualiza su resultado. Si quisiéramos modificar un parámetro, en este caso `str`, desde la línea de comandos ejecutaremos:

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

Y obtendremos el nuevo resultado.

Ahora le añadiremos un segundo proceso: poner todas las letras en mayúscula. Para ello añadiremos el proceso *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 ( )

}
```

Y el resultado lo visualizamos en 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 procesos en Nextflow.
Fuente: elaboración propia.

Como podéis comprobar en la sección *workflow*, primero ejecutamos `printStr`, posteriormente `allToUpper` y finalmente lo visualizamos mediante `view( ). view` es un operador que veremos más adelante.

Seguidamente sustituimos el comando de `allToUpper`:

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

por:

```
rev $x
```

y volvemos a ejecutar el *script*. En este caso, queremos imprimir «Hello world!» del revés. Como el primer paso ya lo habíamos lanzado anteriormente, podemos resumir la ejecución mediante la opción *resume*:

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

y nos generará el *output* de la figura 4.

```
N E X T F L O W  ~  version 20.10.0
Launching `helloworld.nf` [zen_rosalind] - revision: 00b2557d9b
executor >  local (1)
[f9/a37153] process > printStr   [100%] 1 of 1, cached: 1 ✔
[b2/de6b1e] process > allToUpper [100%] 1 of 1 ✔
!dlrow olleH
```

Figura 4. Resumir la ejecución del *script* helloworld.nf.
Fuente: elaboración propia.

Como podéis ver, el primer paso del *script* no ha sido calculado de nuevo, sino que se ha utilizado el proceso ya generado anteriormente (*cached*). De esta forma, si en algún paso del *pipeline* ha habido algún error que ha parado el proceso, se puede resumir previa subsanación del problema.

2. Nextflow

2.5. Canales en Nextflow

Como se ha comentado anteriormente, los canales dirigen el flujo de información a través de los distintos procesos. Para crear explícitamente un canal se debe utilizar un método de Channel Factory proporcionado por Nextflow.

En Nextflow hay dos tipos de canales:

- *Queue channels*: son canales asincrónicos, unidireccionales y FIFO (*first-in-first-out*).

Algunos ejemplos son:

```
- 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*): son muy similares a los *queue channels*, pero solo admiten un valor.

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

Para entender un poco mejor el funcionamiento de los canales crearemos un *script* llamado *orden.nf* con un canal `value` y otro `of` e imprimimos su resultado.

```
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( )
```

```
}
```

Como podéis observar, en el apartado *workflow* estamos especificando los *inputs* del proceso ordenE. En el resultado de este *script* (figura 5), el orden de aparición de los valores del canal of no es el mismo que le hemos indicado. Si repetís la ejecución seguramente os saldrá otro resultado distinto. Se producen cuatro procesos de forma no correlativa.

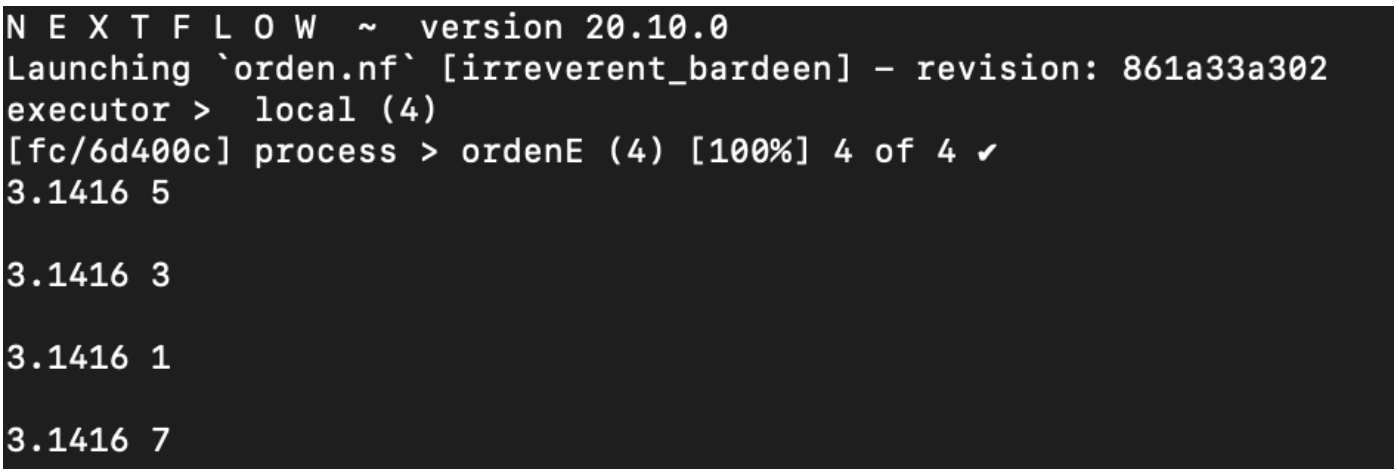


Figura 5. Output del *script* orden.nf.
Fuente: elaboración propia.

Nextflow también es capaz de generar métricas y reportes mediante opciones introducidas vía terminal.

Unos ejemplos serían:

- *with-report*: crea un informe de ejecución.
- *with-trace*: generará un archivo donde se indiquen parámetros de la ejecución, como memoria y *cpus* utilizadas, inicio de ejecución...
- *with-timeline*: permite identificar los cuellos de botella indicando el tiempo que consume cada proceso.

```
$ nextflow run orden.nf -with-timeline<p></p>
```

2. Nextflow

2.6. Operadores

En la sección anterior hemos visto cómo crear canales para dirigir los datos entre los procesos. Para poder modificar el contenido o el comportamiento de un canal, Nexflow ha creado lo que se denominan *operadores*. En los *scripts* anteriores hemos visto el operador *view*, pero podemos encontrar operadores de filtrado, combinación o de operaciones matemáticas entre muchos otros. En esta sección veremos algunos ejemplos.

Los operadores pueden introducirse mediante un *pipe* (|), como hemos visto anteriormente, o precedidos por un punto. Así:

```
workflow{

ordenE(pi,queue_ch) | view( )

}
```

es análogo a:

```
workflow{

ordenE(pi,queue_ch).view( )

}
```

A partir del script anterior eliminaremos el canal *valor pi* y nos quedaremos con un ejemplo más sencillo con el canal *of queue_ch*. Como veréis a continuación añadimos la notación *.view*, y dentro de este operador introducimos un prefijo, *chr*, y un valor *$it* entre {}. Estos paréntesis definen un bloque de código que va junto y utiliza la nomenclatura de *goovy* (*i*t, de *ítem*) para definir los parámetros.

```
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 este caso el *output* se visualiza en la figura 6.

```
N E X T F L O W  ~  version 20.10.0
Launching `orden2.nf` [jovial_brahmagupta] - revision: 3e9c59f53f
executor >  local (4)
[3b/f3d7ba] process > ordenE (3) [100%] 4 of 4 ✔
chr7

chr3

chr1

chr5
```

Figura 6. Resultado del operador *view* con prefijo.
Fuente: elaboración propia.

Podemos introducir un filtro en el canal para únicamente mostrar los valores superiores a 4:

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

También podemos combinar canales utilizando el 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"})
```

y hacer operaciones como contar el número de elementos:

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

Las posibilidades de los operadores proporcionan una versatilidad muy grande de poder manipular los datos a analizar.

2. Nextflow

2.7. Configuración

Finalmente trataremos los archivos de configuración de Netxflow. Estos archivos son relevantes para poder migrar los *scripts* entre entornos de computación y para el control de los recursos a utilizar.

Podemos encontrar diversos archivos de configuración y algunos pueden entrar en conflicto entre ellos. Por ello Nextflow tiene una priorización:

1. Parámetros especificados en la línea de comandos.
2. Parámetros procedentes del archivo especificado mediante la opción -params-file.
3. Archivo de configuración especificado mediante la opción -c my_config.
4. Archivo de configuración nexflow.config en el directorio de trabajo.
5. Archivo de configuración nexflow.config en el directorio del proyecto de *workflow*.
6. Parámetros definidos en el mismo *script* de Nextflow.

Si un parámetro está en más de una de estas fuentes, Nextflow utiliza la primera fuente como referencia y no utiliza las subsiguientes.

El archivo consta de parejas nombre = valor

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

Un archivo de configuración puede incluirse en otro. Por ejemplo:

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

La instrucción `includeConfig` busca el archivo de parámetros y los incluye como propios.

Los parámetros de configuración se pueden especificar en lo que se denominan *scopes*. Son parámetros que afectan específicamente a un tipo de configuración. Hay varios *scopes* de configuración, los más habituales de los cuales son:

- *aws*: Amazon S3.
- *conda*: entornos Conda.
- *docker*: contenedores Docker.
- *k8s*: clúster de Kubernetes.

Para habilitar la utilización de una imagen Docker, por ejemplo, se podría introducir en el nextflow.config:

```
docker.enabled = true
```

o se podría especificar directamente en la línea de comandos:

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

También es posible especificar una imagen Docker para un proceso determinado:

```
process uno {

  container 'nombre_imagen_1'

```

```
    '''
      ejecución
    '''
  }

  process dos {
    container 'nombre_imagen_2'

    '''
   ejecución
    '''
  }
```

2. Nextflow

2.8. *nf-core*

Como otras iniciativas que ya hemos tratado anteriormente, Nextflow también tiene un repositorio de *workflows* para poder ser ejecutados. Este conjunto de *pipelines* se agrupa en el proyecto *nf-core* (https://nf-co.re/). Para poder trabajar directamente desde el terminal se ha generado el paquete *nf-core tools* para poder gestionar los distintos *workflows* accesibles en *nf-core*. *nf-core tools* está escrito en Python y se puede instalar desde la siguiente dirección: https://nf-co.re/tools.

Para poder ver los distintos paquetes simplemente se puede hacer un listado:

```
$ nf-core list
```

Muchas veces necesitamos ser más específicos en nuestra búsqueda y por ello podemos filtrar:

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

O inclusive ordenar por estrellas:

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

Cada repositorio tiene su propia página con instrucciones y documentación. La utilización de estos repositorios públicos permite un aprendizaje más rápido de las herramientas de Nextflow en el contexto específico de la bioinformática. Igualmente, *nf-core* también proporciona un canal directo de preguntas mediante una cuenta en Slack (https://nf-co.re/join).

Resumen

En este módulo hemos dado una breve pincelada a los Workflow Managers y en especial a Nextflow. Para aumentar el control de los entornos en los cuales se computan los análisis es imprescindible utilizar Workflow Managers, y a poder ser, conjuntamente con entornos tipo Conda o contenedores, como hemos visto en capítulos anteriores. La utilización de este tipo de herramientas permite la portabilidad a cualquier entorno de computación y la implementación de *pipelines* generados por otros grupos de una manera ágil y sencilla. La modularización de los procesos permite que su actualización pueda ser dinámica y que la introducción de código generado por otros programadores no interfiera en otras partes del proceso. La versatilidad de opciones de los Workflow Managers permite un control preciso de todo el proceso, a veces en detrimento de su curva de aprendizaje. La diversidad de Workflow Managers que actualmente tenemos permite poder escoger el ecosistema en que nos encontremos más confortables, variando desde entornos muy visuales tipo Galaxy a otros enteramente programáticos. Es importante llegar a un compromiso entre el aprendizaje necesario para entender el funcionamiento de un *workflow* específico y las herramientas que este nos permite controlar. Finalmente, a la hora de escoger vuestro Workflow Manager se debe tener en cuenta la potencial interoperabilidad. En entornos colaborativos es imprescindible la reproducibilidad de procesos y la compartición de código para poder seguir los principios FAIR de los datos.

Actividades

1. Cread un *script* Nextflow que transforme la palabra «HOLA» (en mayúsculas) a toda en minúsculas.

2. Modificad a través de la llamada del *script* la palabra «HOLA» por «CIAO».

3. Introducid un nuevo proceso que sustituya la *A* por un 4.

4. Cread un *script* con dos canales *of* y un proceso que sume los valores de cada canal e imprima la operación realizada.

5. Buscad y bajaos un *workflow* a través de *nf-core*.

Bibliografía

Bjørn Fjukstad y 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 y otros (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 y 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 y otros (2017). Nextflow Enables Reproducible Computational Workflows. *Nature Biotechnology*, 35, p. 316-319. https://doi.org/10.1038/nbt.3820

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

Philip Ewels y otros (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 y otros (2021). Streamlining Data-intensive Biology with Workflow Systems. *Gigascience*, 10, giaa140. https://doi.org/10.1093/gigascience/giaa140

Gestión de datos

Autores: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve
PID_00298303
Primera edición: septiembre 2023

Introducción

Los ficheros son las unidades lógicas de información persistente dentro de un ordenador. Como carecen de estructura propia, es el programador quien establece cómo organizar la información en su interior (por ejemplo, ficheros FASTA, XML o JSON).

Mediante los comandos apropiados del terminal o línea de comandos, podemos analizar el contenido de los ficheros de un modo relativamente eficiente y cómodo. En el módulo «Introducción a los entornos de trabajo GNU/Linux» hemos trabajado con comandos del terminal para gestionar la información almacenada en ficheros de texto que contienen anotaciones biológicas. Sin embargo, cuando el volumen de información excede ciertos límites, como es el caso de la anotación completa de un genoma, y se incrementa el número de personas involucradas en un proyecto de investigación, es necesario organizar y estructurar la información en una base de datos y gestionarla utilizando un programa especializado, también llamado Sistema de Gestión de Bases de Datos (SGBD).

Un SGBD es la herramienta idónea para administrar eficientemente elevadas cantidades de registros o datos. La responsabilidad sobre la gestión y los formatos internos de los datos corresponde al sistema, liberando al propio usuario de estas tareas.

Con un sistema de la gestión de la información tendremos a nuestra disposición un conjunto de herramientas e instrucciones que nos permitirán extraer nuevo conocimiento de toda esta información.

En este módulo veremos dos modelos de gestión de los datos, el modelo relacional basado en el lenguaje SQL y utilizaremos el SGBD MySQL, y el modelo no relacional, también llamado NoSQL, y utilizaremos el SGBD MongoDB basado en colecciones de documentos.

Objetivos

1. Conocer el modelo entidad-relación para diseñar bases de datos.
2. Dominar la conversión de entidades y relaciones en tablas con atributos.
3. Crear y administrar una base de datos con MySQL.
4. Entender las características básicas del lenguaje SQL.
5. Crear y poblar con registros reales las tablas SQL.
6. Realizar consultas SQL a una base de datos relacional.
7. Conocer la utilidad de los subprogramas almacenados, procedimientos, funciones y disparadores (Triggers).
8. Iniciarse con los ficheros JSON y el SGBD NoSQL MongoDB.
9. Administrar un sistema de gestión de base de datos.

1. Bases de datos relacionales

1.1. Introducción

El modelo relacional es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica, como conjuntos de datos llamados *tuplas*. Pensamos cada relación como si fuese una tabla que está compuesta por registros: cada fila de la tabla sería un registro o *tupla*, y columnas, también llamadas *campos*.

Entre los paradigmas actuales de bases de datos, el modelo relacional está muy extendido y se adapta a la mayoría de los entornos bioinformáticos por su eficiencia y simplicidad.

Otra ventaja de este paradigma es que existen numerosas implementaciones *open source* que proporcionan los servicios completos de un sistema gestor de base de datos relacional con diferentes interfaces gráficas de usuario.

Formalmente, el paradigma relacional está dividido en tres componentes básicos:

- Las tablas y las relaciones entre estas estructuran los datos.
- El álgebra relacional opera sobre la información.
- Un conjunto de axiomas mantiene la integridad del sistema.

Una tabla modela un elemento del mundo real, caracterizando sus atributos. Una relación entre dos tablas emula las asociaciones lógicas existentes entre dos elementos de distintas clases en la realidad, permitiendo el acceso cruzado de información.

Para un universo de datos en particular, la organización de las tablas y las relaciones que lo conforman reciben el nombre de esquema relacional. Una vez definida esta estructura, debe crearse una base de datos para ser poblada con los datos reales (conocidos como instancias o registros), siendo administrada desde ese momento por un sistema de gestión de bases de datos.

Utilizando el álgebra relacional, el usuario puede realizar consultas para extraer nueva información y actualizarla.

Para realizar un diseño eficiente de la base de datos debemos seguir estas reglas:

- Reunir todas las clases de información que deseamos guardar.
- Estructurar de forma lógica la información en diferentes categorías.
- Definir los atributos que caracterizan cada categoría.
- Asignar identificadores suficientemente descriptivos a los atributos.
- Decidir el tipo de datos asociado a cada atributo.
- Descomponer cada pieza de información en la unidad más elemental.
- Seleccionar los atributos que identifican de forma única cada categoría.
- Identificar las relaciones entre categorías.

1. Bases de datos relacionales

1.2. El modelo entidad-relación

Podemos estructurar cualquier realidad en diferentes entidades que pueden interactuar entre ellas siguiendo determinadas reglas. Por ejemplo, el genoma, en tanto que parte de la realidad biológica de una célula, también puede estructurarse en distintos componentes. A partir de esta organización artificialmente construida, podemos modelar la totalidad de los elementos que lo conforman utilizando entidades y relaciones. Estas estructuras, contenedores de información o datos pueden ser digitalizados en un ordenador para su gestión y análisis; en el caso del genoma, análisis bioinformáticos.

El modelo entidad-relación nos ayuda a diseñar nuestra propia base de datos. Una entidad representa una clase de elementos en el entorno real que deseamos modelar. Una ocurrencia es una instancia o ejemplo particular de una entidad. La conectividad o participación entre entidades debe especificarse explícitamente mediante relaciones. El número de ocurrencias de una entidad que podrán relacionarse con instancias de otra entidad será:

- Uno a uno (1:1): un elemento de la primera entidad puede relacionarse con un único elemento de la segunda.
- Uno a varios (1:N): un elemento de la primera entidad puede relacionarse con varios elementos de la segunda (pero no al contrario).
- Varios con varios (M:N): un elemento de la primera entidad puede relacionarse con varios elementos de la segunda (y viceversa).

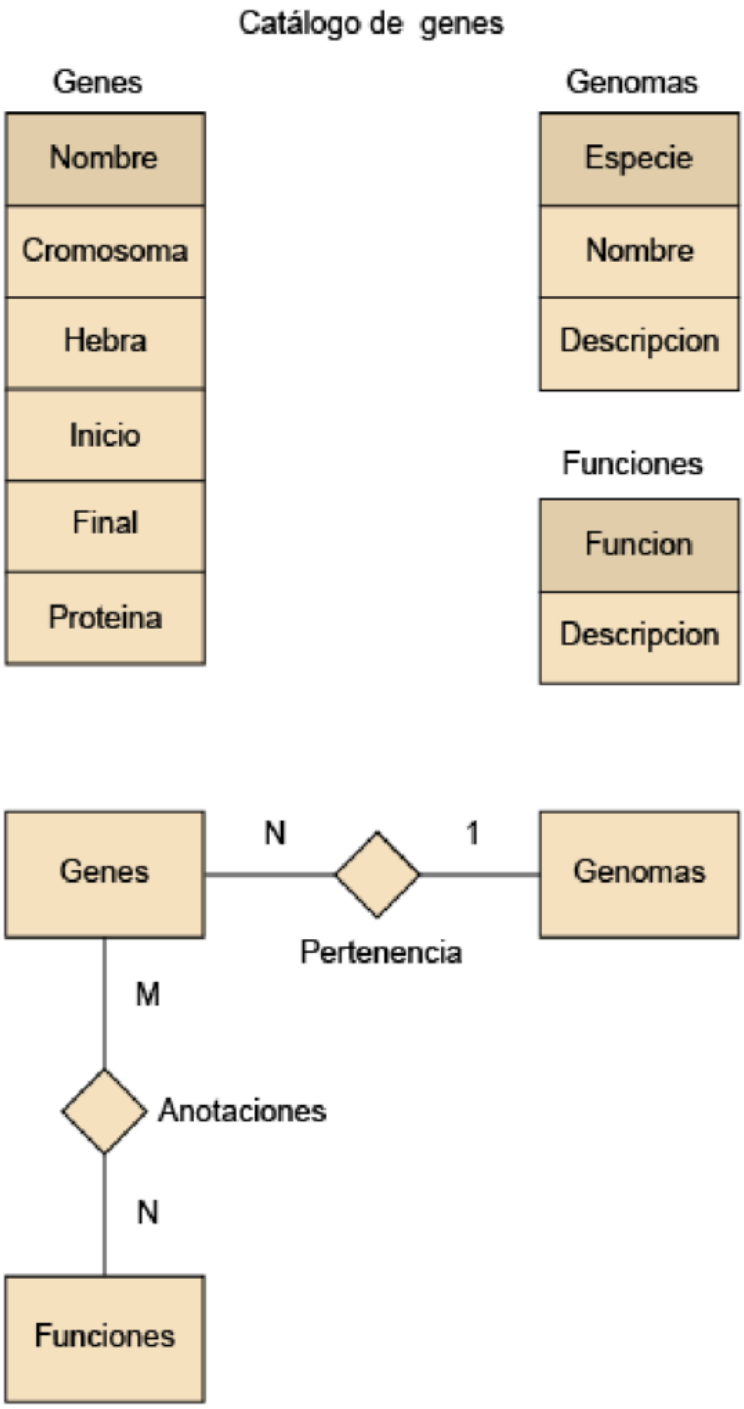


Figura 1. Modelo entidad-relación de un catálogo de genes.
Fuente: elaboración propia.

El modelo contiene tres entidades y dos relaciones. Gráficamente, las entidades se representan utilizando rectángulos, y las relaciones, mediante líneas rectas con un rombo para indicar la conectividad.

Este esquema basado en entidades y relaciones resulta sencillo de emplear posteriormente para construir la base de datos definitiva. Para mostrar las diferentes etapas de diseño, procederemos a modelar un escenario muy habitual en entornos de investigación bioinformáticos: la caracterización del catálogo de genes de un genoma. Podéis observar las diferentes entidades con sus atributos y las relaciones entre entidades que van a formar nuestro modelo en la figura 1.

Los genes son fragmentos de ADN ubicados en una localización precisa del genoma que codifican la secuencia de una proteína. Estas moléculas, por otro lado, desempeñan una función biológica específica dentro del organismo (*). Lógicamente, cada genoma posee su propio catálogo de genes. Analizando esta información previa, decidimos modelar este entorno utilizando las entidades **genes**, **genomas** y **funciones**, con sus propios atributos (mostrados en la figura 2).

Lógicamente, las entidades no son objetos aislados de su entorno. Debemos, por tanto, cumpliendo las especificaciones de nuestro problema, unir mediante relaciones aquellas entidades que están interconectadas en el mundo real. En este caso, los genes poseen la capacidad de pertenecer a un genoma para desarrollar una función concreta en el organismo. Para satisfacer ambas propiedades, definimos las relaciones binarias **pertenencia** y **anotaciones**, cada una con una conectividad diferente (figura

1). Las entidades y relaciones que forman este modelo deben ser convertidas en tablas. Las entidades y sus atributos pasarán a ser tablas de nuestra base de datos, pero no todas las relaciones van a ser tablas, va a depender del tipo de conectividad. Es básico establecer para cada tabla un atributo especial (o una combinación de ellos) que identifique cada instancia de forma unívoca. Este atributo especial recibe el nombre de clave primaria. Es preferible usar un código característico en lugar de un nombre para facilitar la identificación de cualquier instancia mediante la clave primaria (por ejemplo, un código numérico asignado en función del orden de entrada en la tabla).

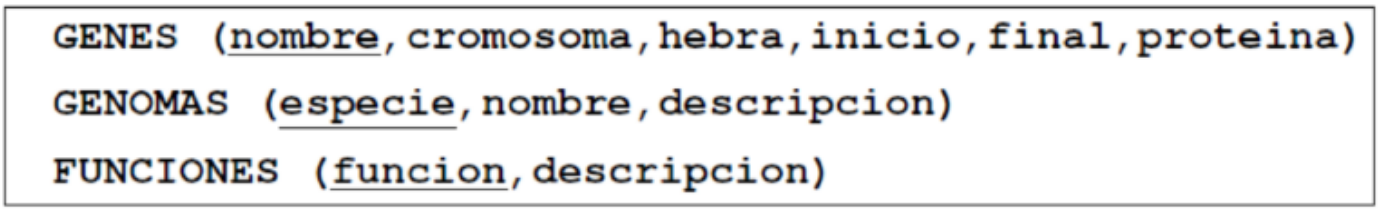


Figura 2. Entidades convertidas en tablas.
Hemos subrayado la clave primaria de cada tabla.
Fuente: elaboración propia.

Las dos relaciones existentes en nuestro esquema deben modelarse de distinta forma, debido a que cada una presenta una combinación diferente de posibles ocurrencias entre tablas. La asociación *pertenencia* entre las tablas *genes* y *genomas* debe representarse con una relación con cardinalidad 1:N, pues un gen solo pertenece a un genoma, pero un genoma contiene muchos genes. Esta relación no necesita una tabla nueva, puede implementarse referenciando simplemente desde una tabla (genes), que es la parte *n* de la relación *pertenencia*, la clave primaria de la otra (genomas) que es la parte 1 de la relación *pertenencia*. Dentro de la tabla *genes*, este atributo recibe la denominación de clave foránea.

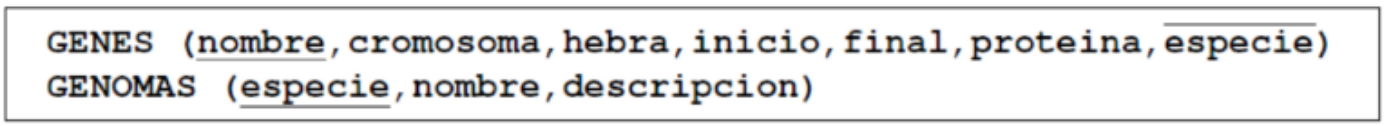


Figura 3. Relaciones (1:N) convertidas en claves foráneas.
Indicamos con un subrayado superior la clave foránea de cada tabla.
Fuente: elaboración propia.

La relación *anotaciones* entre las tablas *genes* y *funciones* tiene una conectividad M:N, pues un gen puede poseer varias anotaciones, pero una anotación también puede ser compartida por varios genes. Para su correcta implementación, introduciremos una nueva tabla llamada *anotaciones*. Esta poseerá, como clave primaria la combinación de las claves primarias de cada tabla original.

Dado que ambas claves primarias por separado poseen todas las propiedades necesarias, el diseñador garantiza con esta medida que cada instancia de esta nueva tabla estará dotada de un identificador único, que estará formado por el nombre del gen junto con el nombre de la función en particular para que sea un identificador de la instancia único que no se pueda repetir.

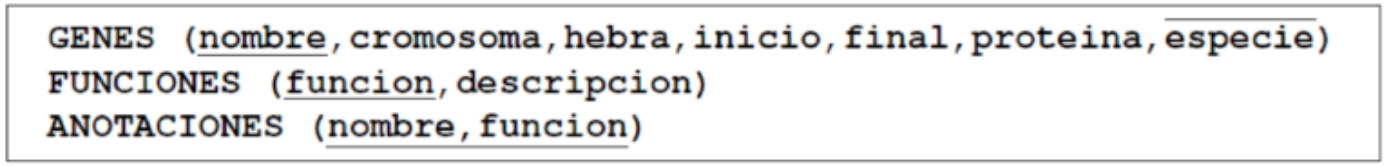


Figura 4. Relaciones (M:N) convertidas en tablas.
Fuente: elaboración propia.

Las relaciones con cardinalidad M:N exportadas desde el modelo entidad-relación están caracterizadas también por sus propios atributos. Así, es posible añadir un campo para guardar el origen de cada anotación (por ejemplo, computacional, experimental o fuente bibliográfica). Podemos extraer esta información a partir de los datos recuperados del sistema de anotación automática utilizado para poblar de ejemplos nuestro catálogo de genes:

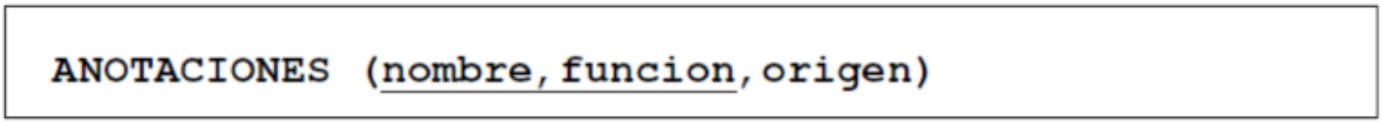


Figura 5. Atributos en relaciones convertidas en tablas.
Fuente: elaboración propia.

La selección de las claves idóneas resulta esencial dentro del diseño de una base de datos relacional. De hecho, la integridad de un modelo relacional debe cumplir dos requisitos fundamentales relacionados con la gestión de estas:

1. La clave primaria no debe contener un valor indefinido o nulo y el valor ha de ser único.

2. La clave foránea debe hacer referencia a una clave primaria de otra tabla.

1. Bases de datos relacionales

1.3. El lenguaje SQL y el SGBD MySQL

SQL (en inglés, *Structured Query Language*, 'lenguaje de consultas estructuradas') es el lenguaje de acceso a las bases de datos relacionales más extendido. Con este sistema, el cliente especifica las instrucciones para crear, dotar de contenido, modificar o eliminar las tablas de la base de datos, y realizar las consultas.

Para trabajar con el lenguaje SQL es necesario disponer de un sistema gestor de bases de datos (SGBD), una aplicación informática normalmente basada en el modelo cliente-servidor para administrar bases de datos relacionales.

Existen diferentes SGBD, como Oracle, PostgreSQ... En este módulo usaremos el SGBD MySQL. En Linux, el servidor MySQL funciona en segundo plano, sin interferir en la planificación de procesos del sistema. De este modo, cuando el usuario desea utilizar la base de datos, debe conectarse con la aplicación gestora mediante un programa cliente, a través de un entorno gráfico o desde el propio terminal con el intérprete de comandos de SQL, denominado **mysql**.

SQL fue comercializado por IBM en 1981. MySQL es un gestor distribuido por Oracle bajo licencia GNU o comercial.

1. Bases de datos relacionales

1.4. Empezar a trabajar con MySQL

En la máquina virtual que os proporcionamos hay instalado un SGBD MySQL. Al iniciar la máquina virtual también se inicia el servidor MySQL.

El sistema cliente-servidor permite acceder a un único servidor desde varios clientes. Por defecto, el cliente del sistema es un terminal o línea de comandos, pero se puede acceder también al servidor desde un cliente con interfaz gráfica de usuario (GUI).

En la máquina virtual está instalada la GUI **MySQL Workbench**, pero es posible acceder al servidor MySQL desde muchos clientes con diferentes GUI.

Para empezar a trabajar con MySQL utilizaremos el terminal de Linux ejecutando el comando mysql.

Una vez establecida la conexión al servidor, el programa cliente permanece siempre a la espera de la introducción de un nuevo comando SQL por parte del usuario.

Es importante ser cuidadoso con la sintaxis de los comandos, finalizando cada instrucción con el símbolo «;».

Para ilustrar el funcionamiento del juego de instrucciones de SQL mostrado en la tabla 1 sobre el modelo relacional anterior (ver figura 4), implementaremos una base de datos que denominaremos **catalogo**.

Tabla 1. Manual de referencia de comandos de MySQL

Comando	Descripción
`CREATE USER`	Dar de alta a un nuevo usuario
`DROP USER`	Dar de baja a un usuario existente
`ALTER USER`	Modificar la cuenta de un usuario
`GRANT`	Autorizar a un usuario sobre una base de datos
`REVOKE`	Revocar las autorizaciones de un usuario
`SHOW GRANTS`	Mostrar las autorizaciones de un usuario
`CREATE DATABASE`	Crear una nueva base de datos
`DROP DATABASE`	Eliminar una base de datos existente
`USE DATABASE`	Acceder a una base de datos existente
`SHOW DATABASES`	Mostrar la lista de las bases de datos
`CREATE TABLE`	Crear una nueva tabla
`DROP TABLE`	Eliminar una tabla existente
`SHOW TABLES`	Mostrar la lista de las tablas
`DESCRIBE`	Mostrar los atributos de una tabla
`LOAD DATA`	Poblar una tabla con un fichero de registros
`INSERT`	Poblar una tabla con un registro
`UPDATE`	Actualizar un registro de la tabla
`DELETE`	Borrar un registro de la tabla
`SELECT`	Realizar una consulta sobre una o más tablas

`Help`	Mostrar ayuda sobre un comando del sistema
`Pager`	Mostrar listados página a página con otro programa
`Source`	Ejecutar un *script* de comandos de SQL
`Status`	Mostrar información sobre el estado del sistema
`System`	Ejecutar un comando del terminal
`Warnings`	Mostrar avisos del sistema
`Quit`	Salir del gestor MySQL
`Exit`	Salir del gestor MySQL
`Mysql`	Comando del terminal para invocar al gestor MySQL
`Mysqldump`	Comando del terminal para el *backup* de una base de datos

Fuente: elaboración propia.

El SGBD MySQL permite gestionar múltiples bases de datos, implementando un sistema de seguridad basado en autorizaciones. Una vez instalado el servidor en nuestro entorno Linux, se crea inicialmente un usuario con el rol de administrador (*) (en inglés, *root*) con todos los permisos. Por regla general, solo el usuario con dicho perfil posee los permisos suficientes para gestionar el conjunto de usuarios y bases de datos en su totalidad. De este modo, el administrador garantizará el acceso a una determinada base de datos exclusivamente a un grupo de usuarios establecido previamente. Pese a que puede optarse por trabajar directamente como administrador, siempre es recomendable que configuremos una nueva cuenta a nuestro nombre como usuario convencional para autorizarle posteriormente a trabajar con una base de datos en concreto.

En definitiva, el protocolo que debemos seguir para llevar a cabo nuestro trabajo con el gestor de bases de datos consiste en dos fases:

1. Invocamos al intérprete de SQL como administrador, realizamos las siguientes tareas y abandonamos el programa:
    1. Creamos un nuevo usuario a nuestro nombre.
    2. Creamos una nueva base de datos.
    3. Autorizamos al nuevo usuario a trabajar con esa base de datos.
2. Accedemos al intérprete de SQL con nuestro propio usuario y procedemos a interactuar con la nueva base de datos:
    1. Especificamos que vamos a trabajar con dicha base de datos.
    2. Creamos nuevas tablas (vacías) dentro de la base de datos.
    3. Insertamos nuevos registros en dichas tablas. También podemos modificarlos y eliminarlos.
    4. Realizamos consultas sobre los registros de las tablas para obtener la información deseada.

Para empezar a trabajar primero debemos ejecutar el programa **mysql** desde nuestro terminal de Linux empleando el usuario *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 como administrador.
Fuente: elaboración propia.

El comando CREATE USER permite al usuario ***root*** dar de alta a un nuevo usuario en nuestro sistema. Es recomendable asignar una contraseña a cada nuevo usuario de nuestro sistema:

```
CREATE USER usuario

IDENTIFIED BY password;
```

Figura 7. Sintaxis del comando CREATE USER.
Fuente: elaboración propia.

Ahora, actuando como administradores, crearemos un nuevo usuario llamado **eblanco**. Para indicar que este usuario trabajará localmente desde nuestra máquina, que actúa de servidor, emplearemos el término **localhost**.

En algunos ejemplos, para que sean más compresibles, vamos a estructurar las instrucciones de SQL en varias líneas. Nótese que tras introducir cada línea del comando presionando la tecla «Enter», el intérprete de MySQL insertará los símbolos -> para denotar que aún no hemos terminado de introducir el comando completo. MySQL no procederá a ejecutar el comando hasta reconocer el carácter «;».

Os animamos a reproducir en vuestro ordenador los comandos de SQL presentados en esta unidad.

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

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

Figura 8. Crear un nuevo usuario con el administrador.
Fuente: elaboración propia.

Existen varios comandos para gestionar el conjunto de bases de datos del sistema (consultad tabla 1). En este momento debemos proceder a crear la base de datos donde trabajará el nuevo usuario con el comando CREATE DATABASE:

```
CREATE DATABASE basededatos;
```

Figura 9. Sintaxis del comando CREATE DATABASE.
Fuente: elaboración propia.

A continuación, creamos la base de datos **catalogo** y posteriormente empleamos el comando SHOW DATABASES para obtener el listado de bases de datos existentes. Podemos comprobar que la nueva base de datos ha sido creada correctamente:

```
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. Creación de la base de datos *catalogo*.
Al finalizar la ejecución de un comando, el intérprete muestra por pantalla el número de elementos de los resultados (en inglés, *rows*).
Fuente: elaboración propia.

El administrador concede permisos sobre las operaciones que un determinado grupo de usuarios puede realizar sobre la base de datos. El comando GRANT permite autorizar el acceso de un usuario a una base de datos en particular:

```
GRANT operaciones ON basededatos

TO usuario IDENTIFIED BY password;
```

Figura 11. Sintaxis del comando GRANT.
Fuente: elaboración propia.

Para cerrar el trabajo del administrador, debemos garantizar el acceso a la nueva base de datos **catalogo** a nuestro usuario **eblanco** empleando el comando GRANT. Con la cláusula **ALL**, el administrador concede el conjunto completo de permisos a este usuario, aunque exclusivamente sobre esta base de datos. Si queremos eliminar privilegios a un usuario debemos utilizar el comando REVOKE.

Finalmente, para abandonar la ejecución del programa **mysql** como administradores, podemos emplear los comandos `quit` o `exit`.

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

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

mysql> quit;
```

Figura 12. Autorizar el acceso a un nuevo usuario.
Fuente: elaboración propia.

A partir de este instante, a la hora de ejecutar el programa **mysql** vamos a trabajar sobre nuestra base de datos con el nuevo usuario **eblanco**. En primer lugar, debemos acceder al gestor de MySQL empleando el nombre de usuario y la contraseña que hemos creado.

```
% 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 como un usuario convencional.
Fuente: elaboración propia.

Como es la primera vez que accedemos con este usuario, vamos a obtener información sobre el listado de las bases de datos accesibles utilizando el comando SHOW DATABASES.

Como mostramos a continuación, en la figura 14, nuestro nuevo usuario puede trabajar con dos bases de datos: **catalogo** y una segunda base de datos (**information_schema**) que contiene información interna sobre la configuración del sistema.

Con el comando SHOW GRANTS podemos ver también las autorizaciones que el administrador ha concedido a este usuario.

```
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. Conocer las bases de datos disponibles.
Fuente: elaboración propia.

Cualquier usuario, una vez dentro del sistema, debe especificar el nombre de la base de datos que va a usar antes de empezar a realizar operaciones sobre ella. Para indicar el nombre de la base de datos que vamos a seleccionar, usaremos el comando **USE**:

```
USE basededatos;
```

Figura 15. Sintaxis del comando USE.
Fuente: elaboración propia.

Como deseamos trabajar con la base de datos **catalogo**, procedemos a declarar este hecho en el intérprete de MySQL. Una vez esta instrucción ha sido ejecutada con éxito, ya estamos en disposición de crear las tablas que sirven como soporte de las entidades y las relaciones diseñadas con anterioridad para poblarlas posteriormente con nuevos registros.

```
mysql> USE catalogo;

Database changed
```

Figura 16. Seleccionar la base de datos para trabajar.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.5. Creación de tablas y restricciones

Una base de datos está formada por un conjunto de tablas que nos permitirán estructurar la información que percibimos en un escenario concreto del mundo real. Cada tabla almacenará en forma de registros la serie de ejemplos de cada clase de entidades o relaciones entre entidades especificadas previamente. Para crear una tabla de registros vacía con el comando `CREATE TABLE` es necesario primero declarar sus atributos (consultad 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. Sintaxis del comando `CREATE TABLE`.
Los elementos opcionales se muestran entre corchetes.
Fuente: elaboración propia.

A la hora de definir la clase de información que almacenaremos en cada atributo, MySQL proporciona una gran variedad de tipos numéricos y alfanuméricos básicos (tabla 2). El espacio de memoria requerido para almacenar cada variable depende de la precisión especificada en cada caso. Los tipos `DATE` y `TIME` resultan especialmente útiles para llevar el registro de nuestras actividades en el tiempo. El usuario puede declarar, además, variables del tipo objeto (en inglés, *Binary Large Objects* o BLOB) para almacenar ficheros de texto, documentos en formato PDF o incluso imágenes dentro de alguna tabla de la base de datos.

Tabla 2. Tipos de datos en MySQL.

Tipo genérico	Tipos MySQL
Entero	`TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT`
Decimal	`DECIMAL, FLOAT, DOUBLE/REAL`
Texto	`CHAR, VARCHAR, TINYTEXT, TEXT`
Objetos	`BLOB, MEDIUMBLOB, LONGBLOB`
Tiempo	`DATE, TIME`

Fuente: elaboración propia.

Junto con la declaración de los atributos, para crear una tabla debemos especificar qué atributo o combinación de atributos será la clave primaria, identificando de forma unívoca cada instancia (figura 18). En caso de existir, las claves foráneas para referenciar a los atributos de otras tablas también deben indicarse explícitamente.

El diseñador puede activar dos controles internos sobre el valor de un atributo en el momento de registrar nuevas instancias en la base de datos. En primer lugar, es posible rechazar aquellos registros que no posean un valor definido para un atributo concreto. Esta circunstancia se especifica con la construcción `NOT NULL`, justo después de la declaración de tipos. `NOT NULL` sería una

restricción de campo obligatorio, no acepta valores nulos. En caso contrario, el sistema asignará por defecto el valor NULL a ese campo y aceptará valores nulos. Este requerimiento debe satisfacerse inexcusablemente en aquellos atributos que pertenecen a la clave primaria. En segundo lugar, para los identificadores numéricos asociados a cada instancia, el propio sistema puede encargarse de gestionar un contador automático de valores mediante la construcción AUTO_INCREMENT.

Volviendo nuevamente a nuestra base de datos **catalogo** (figura 16), nos encontramos ahora en disposición de crear las tablas del catálogo de genes especificadas formalmente en las figuras 18, 19, 20 y 21. Debemos escoger adecuadamente los tipos de datos para cada atributo o campo, según su contenido, definiendo claramente cuáles son las claves primarias y foráneas. El usuario puede empezar creando las tablas más elementales, es decir, aquellas que no poseen claves foráneas (genomas y funciones). Observad cómo declaramos la clave primaria y nos aseguramos de que ninguna instancia puede darse de alta en la base de datos con un valor nulo para las claves primarias, **especie** y **función**. Para verificar que el proceso de creación ha funcionado correctamente, el usuario puede consultar en la base de datos sobre las tablas existentes con el comando 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 las tablas *genomas* y *funciones* en nuestra base de datos *catalogo*.
Fuente: elaboración propia.

Es posible revisar la definición de una tabla con la instrucción 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. Muestra de la descripción de una tabla de nuestro catálogo.
Fuente: elaboración propia.

A continuación, para crear la tabla **genes**, además de la clave primaria, indicamos un campo o atributo que es la clave foránea (que debe apuntar o hacer referencia a la clave primaria de la tabla *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. Creación de la tabla genes en nuestra base de datos *catalogo*.
Fuente: elaboración propia.

Finalmente, creamos la tabla **anotaciones** para relacionar los genes con las anotaciones funcionales. Junto con los dos valores que identifican cada registro (**gen** y **funcion**), añadiremos un atributo para registrar el origen de la información:

```
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. Creación de la tabla *anotaciones* en nuestra base de datos *catalogo*.
Los dos componentes de la clave primaria de la tabla *anotaciones* deben declararse como claves foráneas con origen en las tablas *genes* y *funciones*.
Fuente: elaboración propia.

Es posible definir otras restricciones a los campos de las tablas con el comando CHECK.

Por ejemplo, podemos indicar que un campo numérico como el campo **incio** de la tabla **genes** que es tipo numérico solo acepte valores positivos `CHECK (inicio >0)`, o que un campo de tipo cadena de caracteres solo acepte determinados valores; por ejemplo, para que el campo *hebra* de la tabla *genes* solo acepte los caracteres «+» o «-», podemos hacer `hebra ENUM('+','-')`.

Una vez creada la tabla podemos modificarla con la instrucción **ALTER TABLE**.

Por ejemplo, si queremos eliminar el campo origen de la tabla **anotaciones** escribimos

```
ALTER TABLE anotaciones DROP COLUMN origen;
```

y si queremos volver a añadir el mismo campo escribimos

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

Durante el tiempo de vida de una base de datos es frecuente que debamos actualizar su contenido. En determinados casos, esto puede implicar la eliminación completa de usuarios, de tablas o incluso, de la propia base de datos. Para implementar estos servicios, MySQL posee la familia de comandos DROP.

```
DROP DATABASE basededatos;

DROP USER usuario;

DROP TABLE tabla;
```

Figura 22. Sintaxis del comando DROP.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.6. Insertar y manipular datos en las tablas

Una vez el esquema relacional de entidades previamente diseñado está estructurado sobre MySQL mediante tablas, es el momento de dotar de contenido cada tabla.

MySQL dispone de dos formas de insertar nuevos registros en las tablas de la base de datos:

1. Carga simultánea de múltiples registros desde un fichero de texto.

2. Inserción individual de cada nuevo registro de forma manual.

Para importar una cantidad elevada de registros (*) es posible utilizar el comando LOAD DATA. Para invocar este comando debemos especificar el nombre del fichero de texto que alberga la información de los registros junto con el nombre de la tabla donde deben ser dados de alta. Es necesario disponer de acceso a un fichero de texto tabulado, donde cada fila representa un nuevo registro y cada columna alberga el valor de un atributo (especificado en el mismo orden que en la tabla).

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

> En determinados entornos de UNIX es necesario activar específicamente la opción –local-infile a la hora de invocar al programa mysql. Esta opción, no obstante, está activada por defecto habitualmente en la mayoría de las distribuciones de Linux.

Podemos proceder a introducir los primeros datos en nuestro catálogo de genes. Es recomendable empezar por las tablas más elementales, aquellas que no poseen claves foráneas. En nuestro caso, las tablas **genoma** y **funciones** se ajustan perfectamente a esta descripción. Por ejemplo, para poblar la tabla **genomas** editaremos el siguiente fichero, **genomas.txt**, desde nuestro terminal de UNIX:

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

Para proceder a la carga de estos datos en la tabla *genomas*, el usuario debe introducir el siguiente comando desde el intérprete 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
```

Presentamos el contenido del fichero **funciones.txt,** que emplearemos para poblar la tabla *funciones* con tres nuevos registros:

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

Ahora empleamos el fichero **funciones.txt** para poblar la correspondiente tabla:

```
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
```

Una vez hemos poblado las tablas **genomas** y **funciones** con varias instancias de especies y funciones biológicas, respectivamente, es el momento de editar el fichero **genes.txt** para dar de alta nuevos genes en la tabla 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
```

Y ahora procedemos a realizar la carga con el comando **LOAD DATA**:

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

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

Antes de proceder a realizar las primeras consultas, editaremos el fichero de texto anotaciones.txt para asignar funciones a los genes que hemos registrado en los comandos previos.

```
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
```

> Observad que los genes pueden poseer más de una anotación funcional. Por otro lado, la misma función biológica puede ser desempeñada por genes distintos. Pero los dos valores juntos forman una clave única.

Estamos en condiciones de poblar nuestra última tabla *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
```

La carga de datos desde un fichero de texto en las tablas es extremadamente útil. No obstante, en determinados casos necesitamos dar de alta un nuevo registro de forma aislada, pero la edición de un fichero de texto únicamente con este objetivo es menos eficiente.

En estos casos, el comando **INSERT** es más adecuado, dado que implementa esta funcionalidad en el gestor MySQL de bases de datos. El usuario debe especificar en el mismo orden tanto la lista de atributos del nuevo registro como sus correspondientes valores. El resto de atributos no incluidos en la relación anterior tomarán el valor **NULL** (excepto para aquellos donde está expresamente prohibida esta circunstancia durante la creación de la tabla, campos obligatorios):

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

Por regla general, sin embargo, un registro contiene todos los campos declarados para una tabla. Por tanto, respetando el orden de los campos en la tabla, podemos omitir la relación completa de los atributos:

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

A modo de ejemplo, mostramos a continuación la secuencia de comandos de inserción equivalente a la carga simultánea de las funciones ejecutada anteriormente.

> Las cadenas de texto deben introducirse utilizando siempre comillas simples, mientras que los valores numéricos no necesitan ningún formato adicional.

```
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');
```

> En el caso de intentar dar de alta un registro cuya clave primaria ya existe, el sistema nos advertirá del error, abortando dicha operación. Una tabla no puede tener una clave primaria repetida.

Una vez tenemos los datos introducidos en las tablas podemos modificarlos con la instrucción UPDATE o eliminar registros con la instrucción DELETE.

Por ejemplo, si queremos modificar el valor **'Factor de transcripción'** situado en el campo **descripcion** del registro o fila con clave primaria 'GO:0003700' de la tabla funciones, y queremos que el nuevo valor sea **'Transcription factor'** escribiremos la instrucción siguiente:

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

Para eliminar únicamente algunos registros de una determinada tabla podemos utilizar el comando **DELETE** junto con la cláusula **WHERE**. De este modo, seleccionaremos con precisión los registros que deben ser dados de baja.

Si el usuario desea eliminar todos los registros de una tabla, conservando la estructura de esta (para reutilizarla en el futuro), basta con omitir la condición:

```
DELETE FROM tabla WHERE condiciones;
DELETE FROM tabla;
```

Si queremos eliminar el registro de la tabla **funciones** con clave primaria '**GO:0007254**' escribiremos la instrucción siguiente:

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

Si intentamos eliminar un registro que está referenciado por una clave foránea de otra tabla, el sistema lo impedirá y saltará un error, a no ser que a la clave foránea le indiquemos la opción **ON DELETE CASCADE**, que permite eliminar en cascada el registro que deseamos eliminar y los registros de la otra tabla que están referenciados.

1. Bases de datos relacionales

1.7. Consultas básicas en las tablas

Las bases de datos son herramientas excepcionalmente útiles para consultar información y extraer nuevo conocimiento. En este sentido, los esquemas entidad-relación no son una excepción. Más bien al contrario, utilizando el álgebra relacional es posible consultar el contenido de las tablas de múltiples formas. Básicamente, las consultas SQL (en inglés, *queries*) consisten en filtros que delimitan el segmento del conjunto completo de los registros de una o más tablas en el cual estamos interesados, para mostrar después el valor de sus atributos.

La instrucción **SELECT** implementa el proceso de consulta sobre las tablas, mostrando los atributos indicados para aquellos registros que cumplen una determinada condición.

> Podéis encontrar información sobre el uso del terminal para llevar a cabo operaciones similares sobre ficheros de texto en el módulo «El entorno de trabajo UNIX».

Vamos a explorar en los próximos subapartados cómo enriquecer nuestras consultas, empleando para ello nuestro catálogo de genes, que está almacenado en la base de datos **catalogo**. Antes de hacer consultas más elaboradas, mostramos a continuación la forma más sencilla de realizar una consulta.

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

Figura 23. Sintaxis básica del comando SELECT.
Fuente: elaboración propia.

La consulta más habitual consiste en mostrar el contenido completo de una tabla. El carácter *, precisamente, indica que deseamos visualizar el listado íntegro de los valores de todos los atributos para el subconjunto de registros seleccionados. Para poner en práctica este comando sobre nuestro catálogo, seleccionamos todos los valores de cada instancia guardada en la tabla *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 24. Mostrando el contenido íntegro de la tabla genes.
Fuente: elaboración propia.

Para evitar el exceso de información, el usuario puede seleccionar los atributos o campos de los registros de una tabla que desea ver por pantalla. Simplemente sustituyendo en la pregunta el símbolo * por un listado de atributos, separados por comas, podemos delimitar la vista de los registros, en este caso genes que obtenemos como resultado:

```
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 25. Mostrando los valores de algunos atributos de la tabla *genes*.
Fuente: elaboración propia.

El recuento del número de registros que cumple una condición concreta es una de las consultas más frecuentes en SQL. En este caso es suficiente con añadir la función `COUNT` a la consulta que estemos realizando para contabilizar el número de líneas de la salida:

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

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

Figura 26. Contando todos los registros de la tabla *genes*.
Fuente: elaboración propia.

La función `DISTINCT` elimina los resultados duplicados. Por ejemplo, si deseamos contar el número de organismos en nuestra tabla *genes*, podemos combinar las funciones `COUNT` y `DISTINCT` sobre el atributo *especies* del siguiente modo:

```
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 27. Contando registros únicos de una tabla.
Fuente: elaboración propia.

El comando **ORDER BY** ordena la lista de resultados producida por un comando **SELECT**, de forma ascendente o descendente (según si añadimos la cláusula **ASC** o **DESC**, respectivamente). En la próxima figura ordenamos los genes por su posición en cada cromosoma o por su nombre, de distintas maneras:

```
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 28. Ordenar los registros de una tabla.
Fuente: elaboración propia.

Cuando se trabaja con tablas que contienen miles de elementos, resulta conveniente mostrar inicialmente solo los primeros registros para comprobar el correcto funcionamiento de la consulta. La función `LIMIT` permite mostrar exclusivamente los primeros *n* registros de la consulta en ejecución:

```
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 29. Mostrar un fragmento de la consulta.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.8. Consultas básicas: filtros y condiciones

El comando SELECT permite también extraer de las tablas únicamente aquellos registros que poseen ciertas propiedades. Para especificar el filtro a realizar sobre el contenido de una tabla, debe añadirse la cláusula WHERE:

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

Figura 30. Sintaxis básica del comando SELECT con condiciones.
Fuente: elaboración propia.

La condición puede ser simple o compuesta, evaluándose sobre uno o más atributos de varias tablas. Los operadores de comparación más habituales se muestran en la tabla 3.

Tabla 3. Operadores de comparación en consultas de MySQL.

Operador	Significado
=,<>	Igual/diferente
<,>	Menor/mayor
<=,>=	Menor/mayor o igual
LIKE	Búsqueda de un patrón de texto
NOT	Negación de una condición
AND/OR	Condiciones combinadas
REGEXP	Expresión regular

Fuente: elaboración propia.

> Los operadores numéricos también resultan muy útiles para buscar registros en un rango concreto de fechas del calendario.

Vamos a probar estos operadores para realizar consultas más concretas sobre nuestra base de datos **catalogo**. En primer lugar, podemos interrogar a la base de datos sobre los genes ubicados en la hebra positiva de la cadena de ADN en cualquier especie, preguntar por aquellos que no pertenecen a nuestra especie o buscar los genes anotados antes del primer millón de bases en cualquier 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 31. Consultas con una condición.
Fuente: elaboración propia.

La cláusula puede complementarse con el modificador %, que actúa de comodín en las expresiones alfanuméricas. A continuación, seleccionamos solo aquellos registros que pertenecen 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 32. Consultas sobre patrones de texto.
Fuente: elaboración propia.

También podemos combinar preguntas sobre valores de distintos tipos. Por ejemplo, si deseamos averiguar cuántos genes de la mosca de la fruta están anotados en la hebra positiva de la cadena de 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 33. Consultas con dos condiciones.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.9. Consultas avanzadas: agrupaciones

Mediante el desglose de datos de una tabla, en función de algún campo concreto, podemos calcular estadísticas sobre cada categoría. El comando GROUP BY permite realizar agrupaciones de los datos de las tablas según los criterios que establezcamos, y es posible combinar estas clasificaciones con operadores de agregación como COUNT, MAX, MIN, AVG o SUM.

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

Figura 34. Sintaxis del comando SELECT sobre grupos.
Fuente: elaboración propia.

Por ejemplo, solicitamos las especies presentes en nuestra base de datos:

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

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

Figura 35. Datos agrupados por especie.
Fuente: elaboración propia.

Posteriormente, podemos contar el número exacto de ejemplos de cada especie:

```
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 36. Número de especies almacenadas en la tabla *genes*.
Fuente: elaboración propia.

Ahora obtenemos las estadísticas básicas sobre la longitud de los genes:

```
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 37. Cálculo de promedios en la base de datos *catalogo*.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.10. Consultas avanzadas: consultas multitabla

Para aprovechar al máximo el modelo relacional y generar nuevo conocimiento de los datos existentes resulta esencial combinar información de varias tablas. Mediante un atributo que dos tablas posean en común, esta operación resulta extremadamente sencilla con SQL. Mientras efectuamos una consulta con el comando `SELECT`, debemos emplear la cláusula `JOIN` especificando el atributo compartido por ambas tablas. Cuando se referencian atributos de dos o más tablas en la misma consulta, es necesario utilizar la sintaxis `nombre_tabla.nombre_atributo` para especificar claramente el origen de cada atributo. Es posible añadir condiciones sobre otros atributos de las tablas con la cláusula `WHERE`.

Dadas dos tablas que denominaremos **tabla1** y **tabla2**, que poseen un atributo comparable (**tabla1.x** y **tabla2.y**), la sintaxis del comando `JOIN` para obtener tanto los valores en común como los valores presentes exclusivamente en la primera o en la segunda tabla, respectivamente, es la siguiente:

```
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 38. Sintaxis de la cláusula `JOIN`.
Fuente: elaboración propia.

> Para trabajar con más de dos tablas, podemos generalizar la misma sintaxis (por ejemplo, `tabla1 JOIN (tabla2,3,4)`).

Antes de proceder a ejecutar consultas sobre las tablas de la base de datos **catalogo**, proponemos poner a prueba el funcionamiento de la cláusula `JOIN` sobre un ejemplo más simple.

Vamos a generar dos tablas que denominaremos **tabla1** y **tabla2**, con un único atributo. Posteriormente, poblaremos ambas tablas con una serie de valores tales que resultará muy sencillo mostrar el conjunto completo de combinaciones a la hora de comparar las dos tablas.

Primero procedemos a crear las dos tablas:

```
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 39. Crear dos tablas para combinar con la cláusula JOIN.
Fuente: elaboración propia.

> Os animamos a reproducir en vuestro ordenador los comandos de SQL presentados en este apartado.

Ahora vamos a crear dos sencillos ficheros de texto con tres valores cada uno: (1,2,3) y (3,4,5). De este modo, vamos a poder estudiar detalladamente la clase de consulta de MySQL que necesitamos para recuperar los valores comunes entre ambas tablas (3) o, alternativamente, los valores que únicamente aparecen en la primera (1 y 2) o en la segunda tabla (4 y 5).

```
1
2
3

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

3
4
5
```

Figura 40. Los ficheros datos1.txt y datos2.txt.
Fuente: elaboración propia.

Finalmente, poblamos las dos tablas utilizando ambos ficheros:

```
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 41. Poblamos las dos tablas para combinar con la cláusula JOIN.
Fuente: elaboración propia.

Ya estamos en disposición de profundizar en el funcionamiento de las consultas que incluyen la cláusula JOIN. Si no añadimos ninguna opción, este comando genera todas las parejas posibles cuyo primer elemento pertenece a la primera tabla y el segundo elemento pertenece a la segunda:

```
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 42. Combinación con JOIN de dos tablas para obtener todas las combinaciones.
Fuente: elaboración propia.

Nuestro primer objetivo en una comparación es localizar los elementos comunes. Para ello es suficiente con incorporar la cláusula ON a la consulta y especificar el atributo compartido por las dos tablas. Esto filtrará aquellas parejas del resultado anterior que no cumplan esta propiedad, mostrándose aquello que buscábamos:

```
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 43. Combinación con JOIN de dos tablas con la cláusula ON.
Fuente: elaboración propia.

En determinados casos, en lugar de la lista de los valores comunes, estaremos interesados también en aquellos valores de una u otra tabla que no pertenecen a la otra. Para ello debemos modificar el comportamiento del comando JOIN con las cláusulas LEFT o RIGHT. Si realizamos una unión por la parte izquierda, recuperaremos un listado de los registros de la primera tabla junto con su registro equivalente en la segunda. En el caso de que este valor análogo no existiera, MySQL lo indicará con el valor NULL. Si, por el contrario, realizamos la unión por la derecha, obtendremos un listado de los valores de la segunda tabla bajo las mismas condiciones.

```
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 44. Combinación con JOIN de dos tablas con las cláusulas LEFT y RIGHT.
Fuente: elaboración propia.

Para acabar de refinar el resultado, debemos mantener exclusivamente los valores que solo están en una tabla. Para lograrlo, podemos añadir al final de la consulta una condición WHERE que exija que el valor no esté presente en la otra tabla. La cláusula IS realiza la evaluación de la expresión que se encuentra a continuación, para acabar respondiendo con un valor booleano (cierto o falso).

```
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 45. Combinación con JOIN de dos tablas empleando una condición.
Fuente: elaboración propia.

Ahora ya estamos en condiciones de efectuar consultas sobre dos o más tablas de nuestro catálogo de genes. Por ejemplo, podemos preguntar por aquellos genes humanos para los cuales se ha documentado una anotación funcional de carácter 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 46. Consultas sobre el catálogo de genes utilizando el comando JOIN.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.11. Consultas avanzadas: subconsultas

En ocasiones deseamos realizar un tipo de pregunta sobre nuestros datos, pero esta no es factible porque la organización en tablas escogida no lo permite. Para solventar este problema, MySQL permite anidar una consulta dentro de otra, con el objetivo de utilizar la pregunta interior para darle la forma apropiada a los datos, que podrán ser tratados posteriormente mediante la consulta exterior.

Sintaxis básica de una subconsulta:

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

Sintácticamente, desde el punto de vista de la consulta principal, la subconsulta interior desempeñará el papel de una tabla convencional. Por esta razón, es posible asignar un nombre tanto a la consulta interior como a los atributos de los resultados que se desprenderán de esta. Para ello emplearemos la cláusula **AS**, que permite asociar un nombre a un grupo de operaciones o atributos en SQL. El nombre empleado para esos atributos resulta útil para referirse a ellos desde la consulta exterior.

```
SELECT subconsulta.valor1, ..., subconsulta.valorn FROM
   (SELECT atributo1 AS valor1, ..., atributon AS valorn
    FROM tabla GROUP BY atributoi) AS subconsulta;
```

Figura 47. Sintaxis de las subconsultas.
Fuente: elaboración propia.

Para ejemplificar la clase de escenario donde las subconsultas resultan potencialmente interesantes, imaginemos una tabla genérica llamada *tabla* con dos atributos, que denominaremos *clase* y *subclase*. Cada registro de esta tabla pertenece a una clase general, y dentro de esa clase, a una subclase más específica. Supongamos que nos gustaría calcular el promedio de subclases diferentes, clase por clase, que han sido utilizadas para etiquetar cada registro. Para obtener la respuesta, definiremos una subconsulta que recibirá el nombre de *contador*. Esta subpregunta agrupará los datos por clases para contar el número total de subclases asignado a los registros de cada clase principal. Finalmente, la consulta exterior simplemente deberá calcular el promedio de los totales calculados por 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 48. Emplear una subconsulta dentro de una consulta principal.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.12. Otras utilidades de MySQL

Junto con el inventario de comandos que interactúan con nuestra base de datos empleando el lenguaje SQL, el SGBD MySQL proporciona un conjunto de aplicaciones elementales para asistirnos a la hora de trabajar con el sistema.

Por ejemplo, el comando `help` muestra por pantalla un breve manual de ayuda sobre cada instrucción de MySQL.

Para poder paginar sobre las entradas del manual, pantalla a pantalla, debemos ejecutar antes la aplicación **pager**. Dicho comando nos permite vincular una aplicación de paginación del terminal de Linux con el intérprete de MySQL (por ejemplo, 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 49. Muestra del manual de ayuda de MySQL.
Fuente: elaboración propia.

El comando `source` permite ejecutar ficheros de texto que incluyen comandos MySQL con la secuencia de instrucciones precisas para realizar una determinada tarea. Por ejemplo, podemos editar un fichero de texto desde nuestro terminal con los primeros comandos que ejecutamos al 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 50. Ejecución del fichero de comandos comandos.sql de MySQL.
Fuente: elaboración propia.

El comando `status` permite ver la configuración 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 51. Muestra de la configuración actual de MySQL.
Fuente: elaboración propia.

Si en algún instante necesitamos acceder al terminal de Linux, podemos emplear el comando **system** para invocar sus comandos desde el 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 52. Ejecución del intérprete de comandos de Linux en MySQL.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.13. Copias de seguridad y restauración de BBDD

Es altamente recomendable llevar a cabo copias de seguridad de nuestros datos con cierta periodicidad. En el caso de las bases de datos gestionadas con MySQL, el programa **mysqldump**, ejecutado desde el terminal de Linux, realiza un volcado completo de su contenido hacia un fichero de texto. Posteriormente, en caso de ser necesario, este fichero de comandos puede ser ejecutado con la instrucción **source** para regenerar la base de datos completa, creando automáticamente las tablas e insertando los registros existentes en ese momento:

```
% 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 53. Realizar una copia de seguridad con el programa mysqldump.
Fuente: elaboración propia.

También podemos restaurar una base de datos MySQL desde el terminal a partir de un fichero *backup* así:

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

1. Bases de datos relacionales

1.14. Ejemplo práctico: gestión del catálogo de genes humanos

Para demostrar cómo extraer conocimiento útil de una base de datos relacional, os proponemos analizar con MySQL el contenido de un catálogo de genes humanos. Un gen es un fragmento de ADN ubicado en el genoma que contiene la información precisa para sintetizar una molécula de ARN. En los organismos eucariotas, un gen está constituido por una sucesión de fragmentos útiles denominados *exones*. En una proporción significativa de los genes humanos existen varias combinaciones alternativas de exones, dando lugar a distintas formas alternativas de un mismo gen, denominadas *transcritos alternativos*. Para codificar la información relativa a la localización de los genes en el genoma es frecuente utilizar ficheros de texto tabulado. Cada línea de estos ficheros contiene los valores de los atributos que caracterizan a un transcrito de un determinado gen. Básicamente, un transcrito de un gen posee una localización concreta, identificada por un cromosoma, una posición inicial/final y una dirección de lectura. Otras características que podemos recuperar sobre un transcrito son su código, el nombre del gen, el número de exones o sus coordenadas exactas.

> **Ved también**
>
> Para revisar los conceptos de *genoma*, *cromosoma*, *gen* y *proteína* os recomendamos la asignatura Fundamentos de biología molecular.

El navegador genómico de UCSC representa gráficamente los diferentes tipos de anotaciones existentes sobre el genoma humano en forma de cientos de pistas. Para administrar eficientemente este elevado volumen de información, una copia del SGBD MySQL está funcionando de forma transparente a los miles de usuarios que cada día visitan este servidor. De este modo, en el caso de que deseemos reproducir una pista en nuestro ordenador, disponemos en la sección de descargas de un fichero SQL para ser ejecutado con el comando `source` y un fichero de texto con el conjunto de datos que deben importarse con la instrucción `LOAD DATA`. En este ejercicio vamos a utilizar la anotación de los genes humanos distribuida por el consorcio **RefSeq** para el genoma humano. Este formato es común a todas las especies suministradas por el navegador.

> **Ved también**
>
> Es posible profundizar sobre el funcionamiento de los navegadores genómicos en la asignatura Genómica computacional.

Vamos a proceder ahora a descargar los dos ficheros asociados a la pista **refGene**, que contiene el catálogo de genes humanos anotados por el consorcio **RefSeq**, en su versión **hg38**.

Para ello, debemos utilizar el comando `wget` para transferir ambos ficheros a nuestro terminal.

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

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

Mostramos a continuación el contenido del fichero **refGene.sql** que realiza la creación de la tabla **refGene**. Los atributos que consultaremos con mayor frecuencia serán (figura 54): **name** (código del transcrito), **chrom** (cromosoma), **strand** (hebra), **txStart** y **txEnd** (coordenadas de inicio y final), **exonCount** (número de exones) y **name2** (nombre del gen).

Es importante no confundir los campos de **name** y **name2**: un gen puede tener varios transcritos, pero un transcrito únicamente pertenece 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 54. Atributos de los transcritos anotados por el consorcio RefSeq.
Fuente: elaboración propia.

Pasaremos ahora a visualizar con el terminal el segundo fichero **refGene.txt**. Este archivo contiene los datos del catálogo completo de genes anotados en el genoma humano. Debemos cargar esta información en nuestra base de datos una vez esté creada la tabla **refGene**. En el contexto de este ejercicio, cada registro contiene información sobre un transcrito de un determinado gen. En el caso de que un gen posea varios transcritos, cada uno se codifica en registros separados (cada uno con su propio código y sus correspondientes coordenadas).

En primer lugar, debemos descomprimir el fichero con el comando **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,
-1,
```

Figura 55. El catálogo refGene.txt de genes humanos.
Fuente: elaboración propia.

> Las anotaciones de un genoma suelen actualizarse frecuentemente. Por este motivo, los datos mostrados en este tutorial pueden variar ligeramente con el paso del tiempo.

Una vez dentro del intérprete de MySQL, indicaremos que vamos a trabajar dentro de nuestra base de datos **catalogo**.

Ejecutaremos, posteriormente, el fichero **refGene.sql** con el comando `source` para crear la tabla **refGene**.

Para verificar que la instrucción anterior ha funcionado correctamente, podemos ver el listado de atributos de la tabla **refGene** con el comando `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 56. Creación de la tabla refGene.
Fuente: elaboración propia.

> Vamos a asumir que ambos ficheros (refGene.sql y refGene.txt) están guardados en la misma carpeta de trabajo desde la cual hemos invocado al programa MySQL, anteriormente.

El segundo paso consiste en poblar la tabla con las anotaciones de los genes humanos que hemos descargado dentro del fichero **refGene.txt**. Usando el comando **LOAD DATA** podemos volcar todo el contenido en la tabla **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 57. Poblar la tabla refGene.
Fuente: elaboración propia.

Nos encontramos en condiciones de comenzar a interrogar a la base de datos. Recordemos, nuevamente, que cada registro de la tabla **refGene** alberga la información asociada al transcrito de un gen en particular. Igualmente, es importante tener en cuenta que una elevada fracción de los genes humanos posee dos o más transcritos alternativos. Nuestra misión, a continuación, es mostrar el enorme potencial de las consultas de SQL a la hora de extraer nuevo conocimiento biológico de los datos almacenados en las tablas de nuestra base de datos.

Comenzaremos mostrando los primeros registros de nuestra tabla, incluyendo únicamente varios de sus atributos para favorecer la legibilidad de los valores de los registros por 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 58. Muestra del contenido de la tabla refGene.
Fuente: elaboración propia.

Dado que cada registro contiene la información de un transcrito, el número de transcritos conocidos en el genoma humano coincidirá con el número de registros almacenados en la tabla **refGene**. Este contaje es sencillo:

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

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

Figura 59. Contar los transcritos de la tabla refGene.
Fuente: elaboración propia.

También podemos contar fácilmente el número total de genes codificados en el genoma humano. Si un gen posee varios transcritos alternativos, entonces encontraremos varios registros en nuestro catálogo que poseen un valor distinto del atributo **name**, pero que comparten el mismo valor para el atributo **name2**. Por tanto, empleando la cláusula `DISTINCT` sobre este último atributo, contaremos una única vez cada gen de nuestra tabla, aunque posea varias formas alternativas:

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

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

Figura 60. Contar los genes de la tabla refGene.
Fuente: elaboración propia.

Si agrupamos los registros de la tabla por el atributo **name2**, podemos elaborar un inventario del número de transcritos alternativos anotados para 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 61. Contar el número de transcritos de cada gen de la tabla refGene.
Fuente: elaboración propia.

Podemos obtener resultados interesantes aplicando la cláusula **WHERE** sobre los atributos de cada registro. Por ejemplo, imaginemos que deseamos conocer el número de transcritos ubicados en cada hebra de la molécula de 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 62. Contar transcritos en una hebra de ADN.
Fuente: elaboración propia.

También podemos contar el número de transcritos localizados en un cromosoma:

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

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

Figura 63. Contar los transcritos de un cromosoma.
Fuente: elaboración propia.

Nuevamente, jugando con el atributo **name2** podemos contar el número de genes codificados en el mismo cromosoma:

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

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

Figura 64. Contar los genes de un cromosoma.
Fuente: elaboración propia.

O identificar cuáles son los transcritos que poseen un mayor número de exones:

```
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 65. Recuperar los transcritos con mayor número de exones.
Fuente: elaboración propia.

También podemos seleccionar aquellos transcritos que poseen un único exón:

```
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 66. Recuperar los transcritos con un único exón.
Fuente: elaboración propia.

Es posible calcular el número de exones, en promedio, por cada transcrito:

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

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

Figura 67. Calcular el número de exones en promedio por cada transcrito.
Fuente: elaboración propia.

Y la longitud en promedio de los transcritos de los genes humanos:

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

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

Figura 68. Calcular la longitud en promedio de los genes.
Fuente: elaboración propia.

Finalmente, vamos a integrar en este análisis el genoma de ratón doméstico. Descargamos los ficheros **refGene.sql** y **refGene.txt** de esta especie en su versión mm9.

Para evitar sobrescribir las anotaciones humanas, debemos grabar ambos ficheros con un nombre diferente (por ejemplo, **refGene_mouse.sql** y **refGene_mouse.txt**). Posteriormente, es necesario editar el contenido del fichero SQL para modificar el nombre de la tabla, por la misma razón (figura 69).

Tras estas modificaciones, ya estamos en condiciones de lanzar la creación de la nueva tabla con el comando **source** y su repoblación con los datos relativos al genoma del ratón con el comando 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 69. Incorporar los genes de ratón en nuestra base de datos.
Fuente: elaboración propia.

Comprobamos que los registros almacenados en la nueva tabla son correctos:

```
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 70. Visualizar los primeros transcritos del genoma del ratón doméstico.
Fuente: elaboración propia.

Ahora, si seleccionamos aquellos registros de las dos tablas que pertenecen al mismo gen en ambas especies, podemos construir un catálogo de genes homólogos.

Podemos llevar a cabo esta asociación porque SQL no distingue entre mayúsculas o minúsculas a la 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 71. Listado de genes comunes entre el genoma humano y el genoma de ratón.
Fuente: elaboración propia.

1. Bases de datos relacionales

1.15. *Triggers*, procedimientos y funciones

La mayoría de bases de datos relacionales ofrecen la posibilidad de almacenar subprogramas. Se denominan funciones, procedimientos y *triggers* o disparadores, y son muy útiles para automatizar tareas y guardar instrucciones SQL que se tienen que utilizar frecuentemente. Son objetos que contienen código SQL, cómo breves *scripts* de código SQL, que pueden aceptar parámetros y declarar variables.

Se asigna un nombre al subprograma y se ejecuta para que quede almacenado en la base de datos. Después lo podemos invocar o *llamar* por su nombre porque se ejecute el código almacenado en los subprogramas.

- **Procedimiento** almacenado. Es un objeto que se crea con la sentencia `CREATE PROCEDURE` y se invoca con la sentencia `CALL`. Los procedimientos pueden aceptar parámetros y no hacen ningún retorno, es decir, no devuelven ningún valor.
- **Función** almacenada. Es un objeto que se crea con la sentencia `CREATE FUNCTION` y se invoca con la sentencia `SELECT` o dentro de una expresión. Las **funciones** pueden aceptar parámetros y devuelven siempre un valor. Este valor devuelto puede ser un valor nulo y en este caso se comportaría como un procedimiento.
- ***Trigger***. Es un objeto que se crea con la sentencia `CREATE TRIGGER` y tiene que estar asociado a una tabla. Un *trigger* se activa, se dispara, cuando ocurre un evento de inserción, actualización o borrado, sobre la tabla a la que está asociado.

Veamos algunos ejemplos:

Procedimientos

1. Creamos un procedimiento en la base de datos **catalogo** para contar el número de genes diferentes de la tabla **refGene**, que denominamos **numGenes**.

```
CREATE PROCEDURE numGenes()
SELECT COUNT(distinct name2) FROM refGene;
```

Este procedimiento no tiene parámetros () y muestra por pantalla el número de genes diferentes que encontramos en el campo **name2** de la tabla **refGene**. Al ejecutar el código, el procedimiento se almacena en la base de datos como un objeto más, como las tablas, y no se ejecuta la consulta `SELECT` que contiene hasta que lo llamemos.

Para llamar el procedimiento **numGenes** escribimos:

```
CALL numGenes();
```

Si queremos ver los procedimientos almacenados a la base de datos **catalogo** escribimos:

```
SHOW PROCEDURE STATUS WHERE db = 'catalogo';
```

Si queremos eliminar un procedimiento creado escribimos:

```
DROP PROCEDURE IF EXISTS numGenes;
```

2. Ahora vamos a utilizar variables, el resultado del **SELECT** lo guardaremos en la variable local **gens** que tenemos que declarar y de la cual debemos definir el tipo de datos.

Como en el procedimiento hay sentencias que acaban en «;» primero asignaremos un nuevo delimitador. Escribimos:

```
DELIMITER //
```

Ahora MySQL interpretará que el final de las sentencias SQL es el símbolo //

Crearemos un nuevo procedimiento llamado **numGenes2**.

Para declarar la variable **gens** necesitamos poner BEGIN antes de DECLARE y acabar con END.

Al declarar las variables locales es necesario especificar qué tipo de dato van a guardar, INT, CHAR, VARCHAR, etc. En este caso es un INT.

```
DECLARE gens INT;
```

Para guardar el resultado del SELECT en la variable local **gens** hacemos:

```
INTO gens.
```

Para mostrar el contenido de la variable **gens** hacemos:

```
SELECT gens;
```

Vemos todo el código para crear el nuevo procedimiento usando variables:

```
CREATE PROCEDURE numGenes2()
BEGIN
DECLARE gens INT;
SELECT COUNT(distinct name2)
INTO gens
FROM refGene;
SELECT gens;
END //
```

Volvemos a cambiar el delimitador para poder usar «;» al finalizar la sentencia DELIMITER ;

Invocamos el procedimiento:

```
 CALL numGenes2();
```

3. Ahora pasaremos un parámetro al procedimiento y lo utilizaremos en la condición WHERE de la consulta SELECT.

Creamos un nuevo procedimiento llamado **numTrans** que nos servirá para contar el número de transcritos según el tipo de transcrito que le pasamos como parámetro.

Al igual que las variables locales, es necesario especificar a los parámetros qué tipo de datos esperan, INT, CHAR, VARCHAR... En este caso, CHAR(50).

El parámetro lo ponemos junto al nombre del procedimiento entre paréntesis numTrans(transcritType CHAR(50)).

Asignamos otra vez un delimitador //:

```
DELIMITER //
```

Veamos el código:

```
CREATE PROCEDURE numTrans ( transcritType CHAR(50))
BEGIN
DECLARE numT INT;
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
```

```
SELECT numT;
END //
```

Cambiamos otra vez el delimitador:

```
DELIMITER ;
```

Invocamos el procedimiento pasándole el parámetro NR:

```
CALL numTrans('%NR%');
```

Ahora invocamos el procedimiento pasándole el parámetro NM:

```
CALL numTrans('%NM%');
```

Al usar el comodín % nos aseguramos que no perdemos ningún registro con el contenido del parámetro.

4. En vez de declarar una variable local dentro de un procedimiento, también podemos utilizar un parámetro como variable de salida. Indicamos que es un parámetro de salida con la cláusula **OUT**. Por defecto, los parámetros son solo de entrada, pero también los podemos indicar con la cláusula **IN**.

Asignamos un delimitador //:

```
DELIMITER //
```

Creamos un nuevo procedimiento llamado **numTrans2** que nos servirá también para contar el número de transcritos según el tipo de transcrito que le pasamos como parámetro, y con un segundo parámetro que nos servirá para guardar el resultado de la consulta.

Veamos el código:

```
CREATE PROCEDURE numTrans2(IN transcritType CHAR(50), OUT numT INT)
BEGIN
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
END //
```

Cambiamos el delimitador:

```
DELIMITER ;
```

Invocamos el procedimiento pasándole los dos parámetros. Usamos una variable definida por el usuario con @ llamada @transcritos para guardar el valor que devuelve el **SELECT** del procedimiento:

```
CALL numTrans2('%NR%', @transcrits);
```

Hacemos un **SELECT** de la variable @transcritos que contiene el número de transcritos:

```
SELECT @transcrits;
```

Funciones

Las funciones devuelven un valor, así que, para llamar una función almacenada, en vez de hacer `CALL` hacemos directamente `SELECT nombre_de_la_función` y nos muestra el valor que devuelve la función.

Asignamos un delimitador //:

```
DELIMITER //
```

Creamos una nueva función llamada **numTrans3** que nos servirá también para contar el número de transcritos según el tipo de transcrito que le pasamos como parámetro.

En las funciones tenemos que indicar, después del nombre y de los parámetros, el tipo de dato que devuelve la función, en este caso un INT, y escribimos:

```
RETURNS INT
```

Al final de la función le indicamos la variable que queremos devolver, en este caso:

```
RETURN numT;
```

Veamos el código íntegro de la función:

```
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 //
```

Cambiamos el delimitador:

```
DELIMITER ;
```

Invocamos la función pasándole el parámetro, en este caso les pasamos el parámetro NR:

```
SELECT numTrans3('%NR%');
```

Si pasamos a la función el parámetro NM:

```
SELECT numTrans3('%NM%');
```

Si queremos ver las funciones almacenadas a la base de datos **catalogo** escribimos la sentencia:

```
SHOW FUNCTION STATUS WHERE db = 'catalogo';
```

Si queremos eliminar una función almacenada, escribimos:

```
DROP FUNCTION IF EXISTS numTrans3;
```

Triggers

Un *trigger* es un objeto almacenado en la base de datos que está asociado con una tabla y que se activa cuando ocurre un evento sobre la tabla.

Los eventos que pueden ocurrir sobre la tabla son:

- **INSERT**. El *trigger* se activa cuando se inserta una nueva fila sobre la tabla asociada.
- **UPDATE**. El *trigger* se activa cuando se actualiza una fila sobre la tabla asociada.
- **DELETE**. El *trigger* se activa cuando se elimina una fila sobre la tabla asociada.

El *trigger* se puede activar o disparar antes (**BEFORE**) del evento o después (**AFTER**) del evento.

Como ejemplos vamos a crear dos *triggers* asociados a la tabla *genes* de nuestra base de datos **catalogo**:

Un *trigger* con el nombre de **trig_check_genes_before_insert** que se asocia a la tabla **genes**. Se activa antes de una operación de inserción. Si el nuevo valor del campo inicio que se quiere insertar es negativo, se guarda como 0. Si el nuevo valor del campo **final** que se quiere insertar es menor que el valor del campo **inicio**, se guarda el valor del campo **inicio**.

Un *trigger* con el nombre de **trig_check_genes_before_update** que se asocia a la tabla **genes**. Se activa antes de una operación de modificación. Si el nuevo valor del campo **inicio** que se quiere modificar es negativo, se guarda como 0. Si el valor del campo **final** del registro que se quiere modificar es menor que el nuevo valor que queremos actualizar del campo **inicio** se guarda como 1.

Creamos 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//
```

Al ejecutar este código de creación del *trigger,* **trig_check_genes_before_insert** queda almacenado en nuestra base de datos, y solo actuará, se activará cuando el usuario ejecute una sentencia de inserción, por ejemplo:

Cambiamos el delimitador:

```
DELIMITER ;
```

Realizamos operaciones de inserción en la tabla genes para que se dispare el *trigger* **trig_check_genes_before_insert**:

```
 INSERT INTO genes (nombre, cromosoma, hebra, inicio, final,
proteina, especie) VALUES ('WASH7P', 'chrX', '+', -2527305,
2575270,'NR_033380', 'H. Sapiens');

 INSERT INTO genes (nombre, cromosoma, hebra, inicio, final,
proteina, especie) VALUES ('WASH7P2', 'chrX', '+', 252730599,
2575270,'NR_033381', 'H. Sapiens');
```

La variable compuesta **NEW** que utilizamos en el *trigger* almacena todos los valores que insertamos en cada operación **INSERT**. De esta forma podemos usarla en el *trigger* sin conocer *a priori* qué valores se van a insertar.

En nuestros ejemplos la variable **NEW.inicio** contiene en el primer `INSERT` el valor -2527305 y en el segundo `INSERT` la variable **NEW.inicio** contiene el valor 252730599 y la variable **NEW.final** contiene el valor `2575270`

En el primer **INSERT** se cumple `NEW.inicio < 0`, esta condición dispara el *trigger* y en el campo **inicio** se guarda el valor 0.

En el segundo **INSERT** se cumple `NEW.inicio > NEW.final`, esta condición dispara el *trigger* y en el campo **final** se guarda el valor `252730599`.

Ahora creamos 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 //
```

Al ejecutar este código de creación del *trigger*, **trig_check_genes_before_update** queda almacenado en nuestra base de datos, y solo actuará, se activará cuando el usuario ejecute una sentencia de actualización, por ejemplo:

Cambiamos el delimitador:

```
DELIMITER ;
```

Realizamos operaciones de modificación en la tabla genes para que se dispare el *trigger* **trig_check_genes_before_update**:

```
UPDATE genes SET inicio = 228748314 WHERE nombre = 'MYC';

UPDATE genes SET inicio = -47643 WHERE nombre = 'cbt';
```

En este caso, la variable compuesta **NEW** que utilizamos en el *trigger* almacena el nuevo valor que queremos actualizar en la sentencia **UPDATE**. En el primer **UPDATE** la variable `NEW.inicio` contiene el valor `228748314` y en el segundo **UPDATE** la variable `NEW.inicio` contiene el valor –47643.

En cambio, la variable compuesta **OLD** almacena todos los valores viejos ya almacenados en la tabla del registro que se quiere actualizar. En el primer **UPDATE**, la variable `OLD.final` contiene el valor almacenado en el campo final del registro con clave primaria **'MYC'** en la tabla **genes**, y en el segundo **UPDATE**, la variable `OLD.final` contiene el valor almacenado en el campo final del registro con clave primaria **'cbt'**.

En el primer **UPDATE** se cumple `NEW.inicio > OLD.final`, esta condición dispara el *trigger* y en el campo inicio se guarda el valor 1.

En el segundo **UPDATE** se cumple `NEW.inicio < 0`, esta condición dispara el *trigger* y en el campo inicio se guarda el valor 0.

En las operaciones **DELETE** solo se utiliza la variable compuesta **OLD** capaz de almacenar todos los valores del registro que se quiere eliminar y acceder a ellos con el formato `OLD.nombre_campo`.

2. Bases de datos NoSQL

2.1. Introducción

Las bases de datos relacionales son muy eficientes y mantienen la integridad de los datos, pero tienen limitaciones para gestionar con rapidez grandes volúmenes de información.

Las bases de datos relacionales escalan de forma vertical. Para crecer necesitan que los servidores tengan más capacidad.

MySQL es uno de los SGBD más utilizados para proyectos web, pero las nuevas aplicaciones web se caracterizan por tener que gestionar un inmenso volumen de información y gran cantidad de datos.

Para afrontar este nuevo reto de gestión de los datos, aparecieron las bases de datos no relacionales, conocidas también como NoSQL o Not Only SQL, y llamadas así porque no dependen únicamente del lenguaje estructurado SQL.

Las bases de datos no relacionales pueden escalar de forma horizontal, permiten la distribución de los procesos de trabajo y conjuntos de datos en múltiples servidores. De esta forma es posible que la escalabilidad de estas bases de datos sea prácticamente ilimitada.

Las bases de datos NoSQL se clasifican como de clave/valor, orientadas a documentos, grafos, o de familias de columnas.

En este módulo nos centraremos en las bases de datos no relacionales orientadas a documentos, concretamente a documentos en formato JSON.

2. Bases de datos NoSQL

2.2. Ficheros JSON

JavaScript Object Notation (JSON) es un formato de datos basado en texto estándar para representar datos estructurados que sigue la sintaxis de objeto de JavaScript. Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript.

Los ficheros para almacenar datos con formato JSON cada vez son más usados en la informática y también en el ámbito de la bioinformática.

Figura 72. Ejemplo de un documento con formato JSON.
Fuente: elaboración propia.

En este formato la forma de guardar los datos es parecida al modelo **clave/valor**, donde la clave sería el nombre del campo o atributo y a continuación tenemos el valor.

Los ficheros JSON en realidad almacenan una **colección** de documentos con este formato y cada documento es la representación o la instancia de una entidad.

Si hacemos el símil con la información que almacenamos en una tabla relacional, cada fila de la tabla corresponde a un documento, y todas las filas de la tabla serían una colección de documentos.

JSON requiere utilizar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas.

Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione. Se debe ser cuidadoso para validar cualquier dato que se quiera usar. Es posible validar JSON empleando una aplicación como JSONLint.

Veamos cómo se almacena la información que tenemos guardada en la tabla *genomas* en un fichero JSON.

```
 {especie:"D. Melanogaster", nombre:"Mosca de la
fruta",descripcion:"Tambien denominada del
 vinagre"} {especie:"H. Sapiens", nombre:"Humano",descripcion:"Nuestra
propia especie"}
 {especie:"M. Musculus", nombre:"Raton",descripcion:"Otro organismo
modelo"}
```

En este caso tenemos 3 documentos que corresponden a las 3 filas de la tabla *genomas*. Cada documento empieza con el símbolo {, y finaliza con el símbolo }

En cada documento se repite el nombre del campo y a continuación su valor, separados por el símbolo :

Cada combinación **campo/valor** se separa por comas.

No se describen los tipos de datos, si el valor es una cadena de caracteres se usan las comillas dobles " ", si el valor es numérico no es necesario escribirlo entre comillas.

Otra forma muy habitual de guardar la colección de documentos es dentro de un **array** con los símbolos **[]** y separando los diferentes documentos por comas. Se considera toda la colección de documentos como un solo objeto, pues están todos en un **array**.

```
 [{especie:"D. Melanogaster", nombre:"Mosca de la
 fruta",descripcion:"Tambien denominada del vinagre" }, {especie:"H.
 Sapiens", nombre:"Humano",descripcion:"Nuestra propia especie" },
 {especie:"M. Musculus", nombre:"Raton",descripcion:"Otro organismo
modelo" }]
```

Un campo puede guardar un **array** de valores:

```
 {dias:["lunes", "jueves", "sábado"]}
```

Un campo también puede guardar otro **documento JSON.** También se llama *documento incrustado*.

```
 {direccion: {calle: "Valencia", número: 334, codigo: 08012}}
```

Un campo puede guardar un **array** de **documentos JSON**.

```
 {amigos:

 [{nombre: "Pedro", edat: 34, telefono: 666737211},

 {nombre: "Soraya", edat: 31, telefono: 666737212},

 {nombre: "Arnau", edat: 29, telefono: 666737213}

 ]}
```

A partir de la versión 8 de MySQL es posible trabajar con documentos JSON en las tablas SQL con el tipo de datos JSON.

Podemos almacenar todo un documento JSON en un campo de la tabla y realizar consultas con el comando JSON_EXTRACT, actualizaciones con JSON_REPLACE y eliminaciones con el comando JSON_REMOVE.

2. Bases de datos NoSQL

2.3. El SGDB MongoDB

Aunque también es posible trabajar con los datos almacenados en formato JSON con MySQL y otros SGBD relacionales como PostgreSQL, la mejor forma de gestionar los datos almacenados en los ficheros JSON es utilizando una base de datos NoSQL como MongoDB**.**

El SGBD MongoDB se publicó en el año 2009 y permite gestionar bases de datos orientadas a documentos. Guarda los documentos en BSON, que no es más que una implementación binaria del formato JSON.

MongoDB es la más popular de las bases de datos NoSQL. Básicamente, devuelve datos en JSON e incorpora los conceptos de colecciones (en lugar de tablas) y documentos (en lugar de filas), su API o lenguaje de consulta se conoce popularmente como MQL (MongoDB Query Language).

Para que tengamos más claro las diferencias entre el modelo relacional y MongoDB podemos consultar la tabla 4:

Tabla 4. Comparativa entre el modelo relacional y MongoDB.

Modelo 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

Fuente: elaboración propia.

Básicamente, la diferencia más sustancial es que mientras en un SGBD relacional como MySQL tenemos bases de datos, tablas y columnas de las tablas, en MongoDB y SGBD NoSQL basados en documentos tenemos también bases de datos, pero en vez de tablas con columnas tenemos colecciones de documentos, y en cada documento tenemos los nombres de los campos en vez de las columnas de las tablas.

2. Bases de datos NoSQL

2.4. Empezar a trabajar con el SGBD MongoDB

En la máquina virtual proporcionada por la UOC tenemos instalado un SGBD MongoDB.

Para conectarnos al servidor de MongoDB abrimos un terminal, escribimos `mongosh` y nos saldrá el cursor >

```
student@ubuntuM0151:~$ mongosh
Current Mongosh Log ID: 66f3c526cf082f27c4964032
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.
3.1
Using MongoDB:          7.0.14
Using Mongosh:          2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-09-25T10:08:11.986+02:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. Se
e http://dochub.mongodb.org/core/prodnotes-filesystem
   2024-09-25T10:08:17.269+02:00: Access control is not enabled for the database. Read and write access to data and confi
guration is unrestricted
------
```

Figura 73. Ejemplo de conexión a MongoDB por terminal.
Fuente: elaboración propia.

Ya estamos conectados al servidor con el cliente de la línea de comandos y ya podremos escribir las órdenes y sentencias para interactuar con el servidor de MongoDB.

Para visualizar las bases de datos creadas:

```
show dbs
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Figura 74. Mostrar las bases de datos creadas.
Fuente: elaboración propia.

Como aún no hemos creado ninguna base de datos, solo nos muestra las bases de datos del sistema.

Para crear una nueva base de datos vacía usamos la orden **use**.

Vamos a generar la base de datos **uoc**.

```
use uoc
```

La orden **use** sirve tanto para crear una base de datos nueva como para conectarse a una base de datos existente. La orden intenta conectarse a una base de datos existente y si no existe la crea.

Esta es una característica general de MongoDB, es muy flexible y da pocos errores. En otros sistemas de gestión de bases de datos como MySQL, al intentar conectar con una base de datos inexistente saltaría un error. En MongoDB no salta error y se crea la nueva base de datos. En realidad, no la crea, guarda espacio para crear en ella colecciones de documentos. Si generamos una nueva base de datos con el comando **use** y no creamos colecciones en ella, no queda almacenada.

Vemos en esta secuencia de comandos cómo consultamos las bases de datos existentes en el sistema con `show dbs`, generamos la base de datos **uoc** con `use uoc`, y cómo cuando volvemos a consultar las bases de datos con `show dbs` no

aparece la base de datos **uoc**.

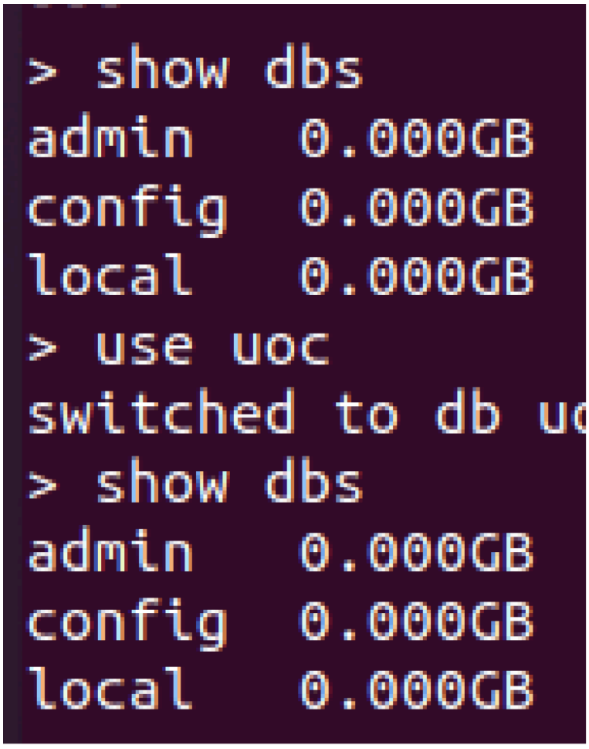


Figura 75. Utilizar una base de datos.
Fuente: elaboración propia.

Para que la nueva base de datos quede guardada en el sistema es necesario que tenga colecciones de documentos.

Vamos a crear ahora una nueva colección vacía en la base de datos **uoc** utilizando la instrucción `createCollection()`.

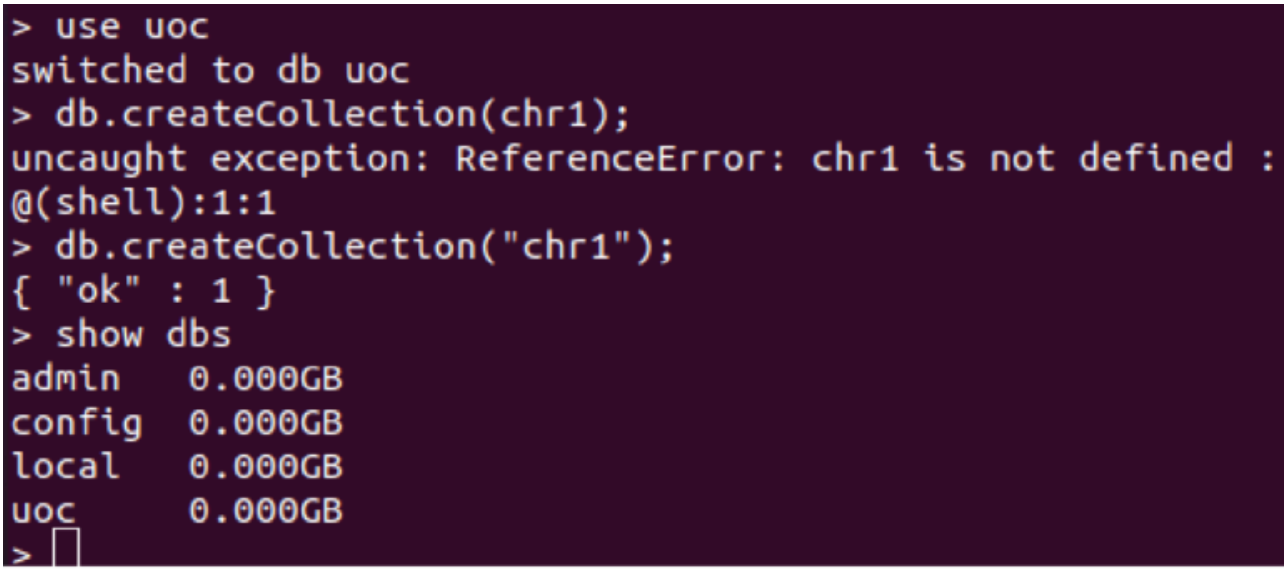


Figura 76. Crear una colección de documentos.
Fuente: elaboración propia.

Volvemos a conectarnos a la base de datos **uoc** con `use uoc` y creamos la colección de documentos vacía llamada **chr1**.

Si os fijáis, primero ejecutamos la instrucción o función `db.createCollection(chr1)`; pero da error porque el parámetro que espera la función no está entre comillas simples: **«chr1»**.

Al escribir el parámetro entre comillas simples `db.createCollection("chr1")`; la nueva colección llamada **chr1** se crea correctamente en la base de datos **uoc**, que ya no está vacía, y al ejecutar `show dbs` ya nos muestra la base de datos **uoc**.

Vamos a analizar la instrucción `db.createCollection("chr1")`; y nos servirá para entender cómo funcionan las instrucciones en MongoDB. Como ya estamos en la base de datos **uoc**, con la parte de la instrucción `db…..` se refiere a la base

de datos que estamos usando, y a continuación la instrucción `createCollection()` que en realidad es una función que puede usar parámetros entre paréntesis.

Como ya veremos, la mayoría de las instrucciones son funciones que pueden usar o no parámetros. Si la instrucción no necesita parámetros dejaremos los paréntesis vacíos **()**.

Vamos a ver algunos **comandos útiles** de MongoDB:

- `db.help()` Muestra la ayuda para los métodos de la base de datos.
- `db.<collection>.help()` Muestra la ayuda para los métodos de la colección. La <collection> puede ser el nombre de una colección ya creada o no creada.
- `Show db` Muestra la lista de todas las bases de datos del sistema.
- `use db` Cambia de la base de datos actual a la base de datos <db>.
- `show collections` Muestra la lista de todas las colecciones de la base de datos actual.
- `show users` Muestra la lista de todos los usuarios de la base de datos actual.
- `show databases` Nuevo a partir de la versión 2.4. Muestra la lista de todas las bases de datos disponibles.
- `db.<collection>.find()` Muestra todos los documentos de la colección <collection>.
- `db.<collection>.find().pretty()` Muestra todos los documentos de la colección <collection> en un formato JSON más leíble.

Para ver las colecciones creadas en la base de datos usamos la instrucción siguiente:

```
show collections
```

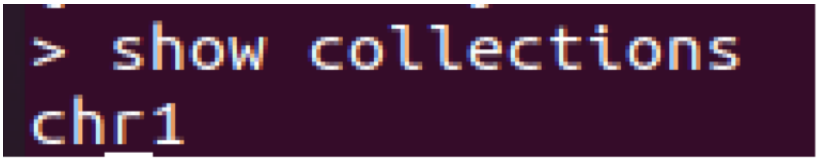


Figura 77. Mostrar las colecciones de documentos.
Fuente: elaboración propia.

2. Bases de datos NoSQL

2.5. Insertar documentos

Como tenemos la colección **chr1** vacía vamos a insertar documentos:

Vamos a insertar el siguiente documento con información del gen **uc001aaa.3**, localizado 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"}
```

Para insertar este documento en la colección **chr1** utilizaremos la instrucción `db.chr1.insert()`; donde se indica con db la base de datos a la que estamos conectados, `db.chr1...` , la colección donde insertamos el nuevo documento, la colección **chr1,** y la función `insert()` que espera como parámetro el documento que vamos a insertar:

```
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 inserción se ha realizado correctamente el sistema nos muestra el siguiente mensaje:

```
WriteResult({ "nInserted" : 1 })
```

Ahora vamos a insertar dos documentos más. Los documentos JSON con información sobre los genes **uc010nxr.1** y **uc009vit.3**, también localizados 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"});
```

Ahora vamos a eliminar toda la colección **chr1** de documentos con la instrucción `db.chr1.drop()`;

Y la volvemos a crear:

```
db.createCollection("chr1");
```

para insertar los tres documentos a la vez con la instrucción `insertMany();`

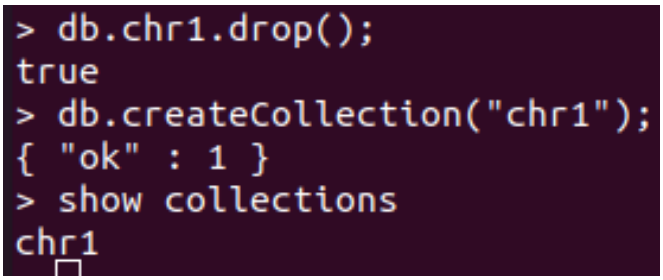


Figura 78. Eliminar una colección de documentos.
Fuente: elaboración propia.

En la parte de ficheros JSON vimos que muchas veces encontramos un fichero con una colección de documentos que están todos en un *array*.

Los tres documentos con la información de los genes de los ejemplos anteriores los tenemos ahora dentro de 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"}]
```

Y los vamos a inserir en la colección **chr1** con la instrucción `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"}]);
```

Ya tenemos otra vez los tres documentos en la colección **chr1** de la base de datos **uoc,** y vamos a ver algunas instrucciones básicas para gestionar la información con MongoDB.

2. Bases de datos NoSQL

2.6. Buscar documentos

Primero contaremos cuántos documentos tiene la colección **chr1** con la instrucción

```
db.chr1.find().count();
```

Esta instrucción contiene dos funciones. La función `find()` que es la función que utilizamos para realizar búsquedas en la colección de documentos. Sería la instrucción **SELECT** que hemos visto en MySQL. Si no añadimos parámetros nos mostrará todos los documentos de la colección, si añadimos parámetros, estos serán los criterios de búsqueda o los filtros de la selección. En este caso complementamos la función `find()` con la función `count()` para contar los documentos encontrados. La función `count()` no requiere parámetros, pero es necesario escribir los paréntesis igualmente.

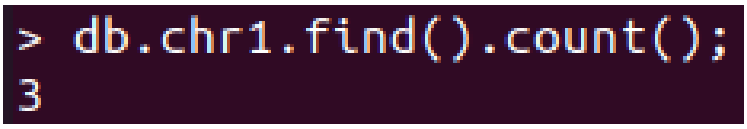


Figura 79. Contar todos los documentos de una colección.
Fuente: elaboración propia.

Al ejecutar la instrucción `db.chr1.find().count();` el sistema nos devuelve 3. La colección **chr1** tiene 3 documentos.

Vamos a utilizar la instrucción db.chr1.find(); para ver todos los documentos:

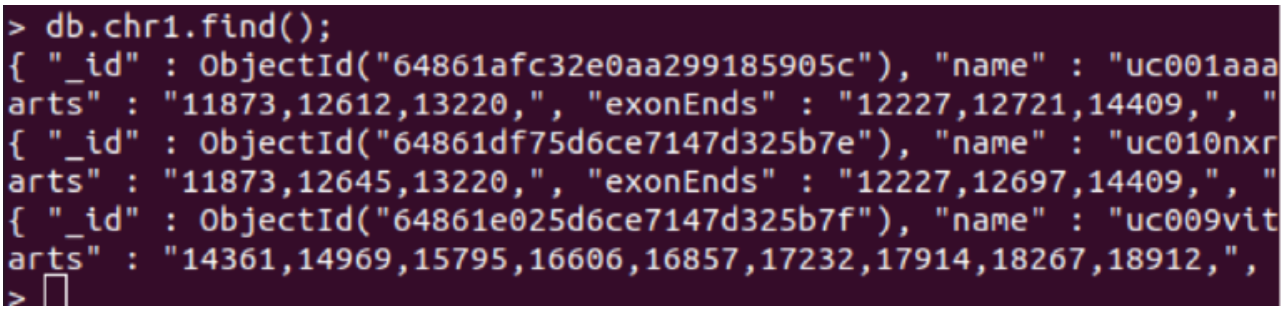


Figura 80. Mostrar la información de todos los documentos de una colección.
Fuente: elaboración propia.

Para cada documento, el sistema nos muestra el campo **_id** con un valor generado aleatoriamente por la función del sistema `ObjectId()`;

En el campo **_id** se guarda la clave primaria del documento. Esta clave primaria es tan importante para el sistema que si el documento que insertamos no tiene el campo **_id** el sistema lo crea de forma automática. En nuestros ejemplos los tres documentos inseridos no tienen el campo **_id** y el sistema lo ha generado automáticamente.

En un documento puede haber muchos campos y, tal como los muestra la función `find();`, quedan poco legibles. Para que el documento se pueda leer mejor, la función `find();` se puede complementar con la función `pretty();`

```
db.chr1.find().pretty();
```

Figura 81. Mostrar la información de todos los documentos de una colección con un formato más legible.
Fuente: elaboración propia.

Como podemos observar, la función pretty(); nos permite ver los valores de cada campo de cada documento de una forma mucho más amigable.

ahora algunos ejemplos de utilización de la función find() con algunos parámetros para filtrar los resultados.

Vamos a mostrar solo los genes de nuestra colección que se encuentra en la hebra +:

```
db.chr1.find({"strand": "+"}).pretty();
```

Figura 82. Mostrar la información de los genes de la hebra +.
Fuente: elaboración propia.

El parámetro de la función es la condición de selección de la búsqueda `{"strand": "+"}`

Si no queremos mostrar algún campo, como por ejemplo el campo `_id`, escribimos

```
db.chr1.find({"strand": "+"}, {"_id":0}).pretty();
```

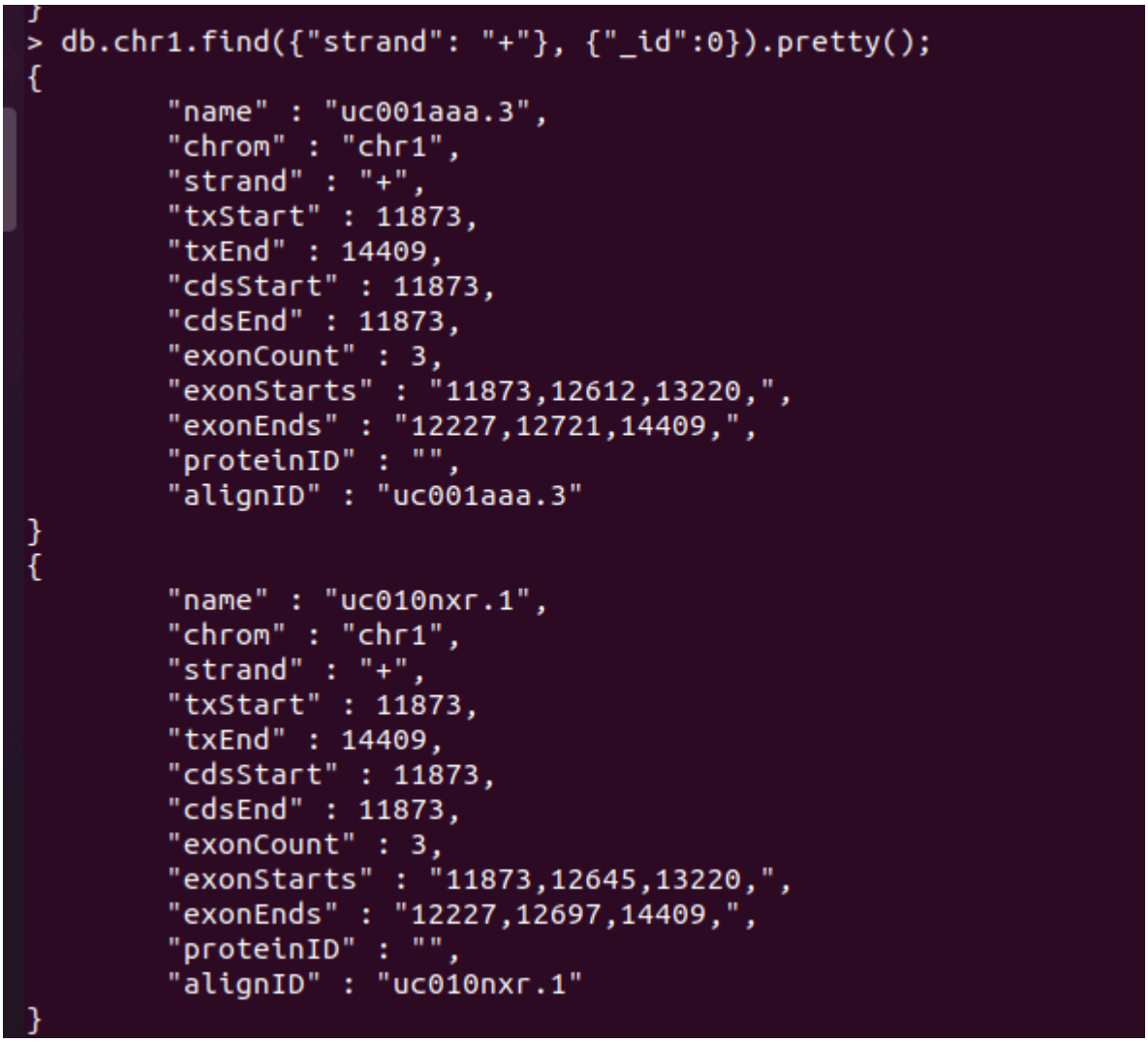


Figura 83. Mostrar la información de los genes de la hebra + sin el campo _id.
Fuente: elaboración propia.

Si queremos mostrar un solo campo de los documentos seleccionados, como por ejemplo el campo **name**, escribimos

```
db.chr1.find({"strand": "+"},{"name":1, "_id":0 }).pretty();
```

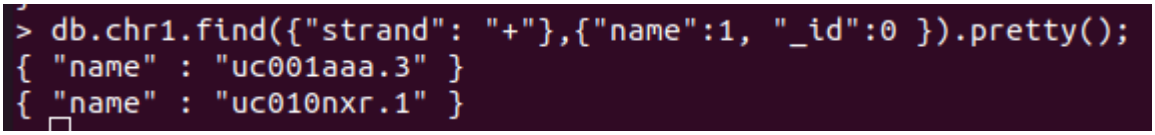


Figura 84. Mostrar los genes de la hebra + sin mostrar el campo _id y mostrar el campo name.
Fuente: elaboración propia.

Las consultas con la función `find()` pueden ser mucho más elaboradas, usando varios campos como condición de búsqueda, o también usando operadores especiales como mayor que, menor que, etc.

Ejemplos de operadores especiales:

```
$gt, $gte, $lt, $lte, $ne, $in, $nin, $mod, $regex/$options,
```

```
$all, $size, $exists, $type, $not, $or, $nor, $elemMatch
```

Ejemplo de selección de documentos en los que el campo **txStart** sea mayor que 11873:

```
db.chr1.find({"txStart": {"$gt" : 11873 }}).pretty();
```

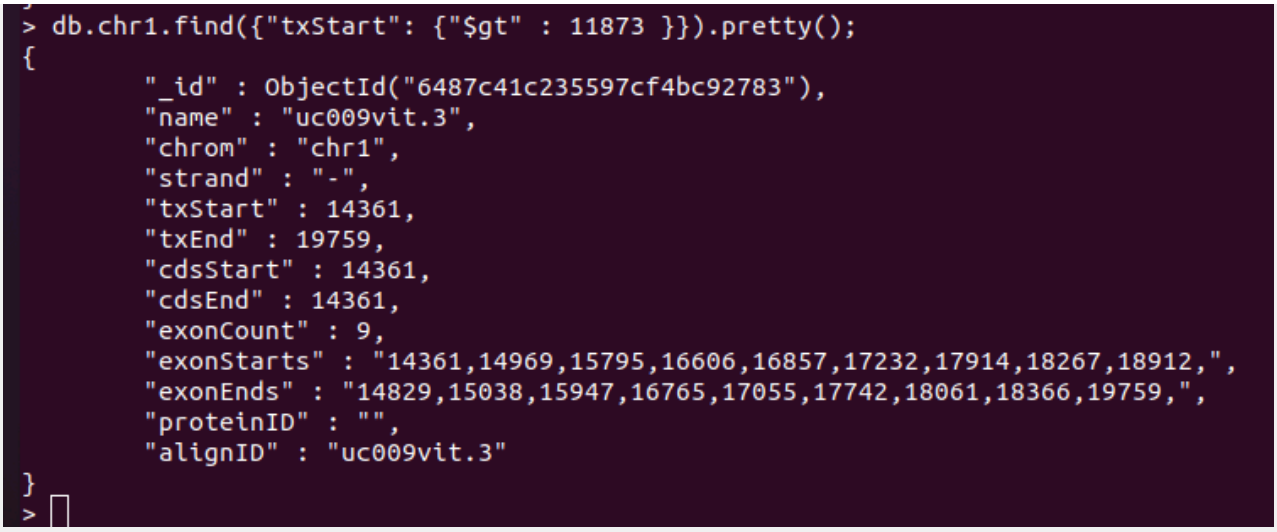


Figura 85. Mostrar los genes que el campo txStar sea mayor que 11873.
Fuente: elaboración propia.

O que cumpla dos condiciones, que se encuentren en la hebra + y que el campo **txStart** sea mayor o igual a 11873.

```
 db.chr1.find({"strand": "+", "txStart": {"$gte" : 11873
}}).pretty();
```

2. Bases de datos NoSQL

2.7. Modificar documentos

Veamos ahora cómo se puede actualizar un documento con la función `update()`.

Vamos a modificar la hebra del gen uc001aaa.3.

```
db.chr1.update({"name" : "uc001aaa.3"}, { "$set" : { "strand": "-" }});
```

Y comprobamos el cambio:

```
db.chr1.find({"name" : "uc001aaa.3"},{"name":1,"strand":1,"_id":0
}).pretty();
```

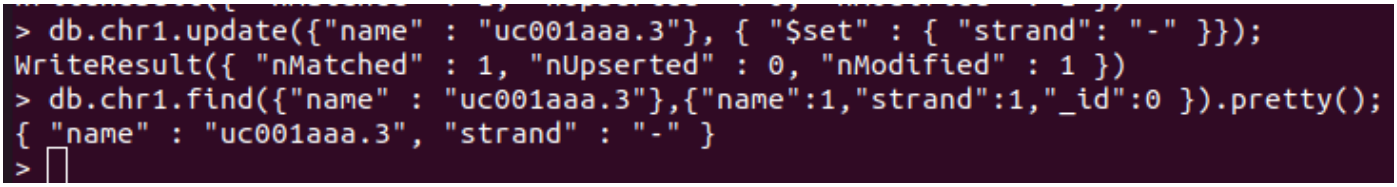


Figura 86. Modificar la hebra del gen uc001aaa.3.
Fuente: elaboración propia.

El operador `$set` modifica el valor de campo si este existe; si no existe el campo, lo incorpora al documento o documentos que coincidan con la selección.

El operador `$unset` elimina el campo del documento o documentos que coincidan con la selección.

También es posible añadir nuevos elementos a un campo *array* con el operador `$push` y eliminar elementos del *array* con los operadores `$pull`, `$pullAll`, `$pop`.

Los operadores del comando update son los siguientes:

```
$set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $pop,
```

```
$addToSet, $rename, $bit, $ positional operator
```

2. Bases de datos NoSQL

2.8. Eliminar documentos

Veamos ahora cómo podemos eliminar un documento de la colección con la función `remove()`.

Eliminamos de la colección el documento con el nombre de gen **uc001aaa.3**

```
db.chr1.remove( { "name" : "uc001aaa.3"});
```

Y comprobamos si el documento se ha eliminado:

```
db.chr1.find().pretty();
```

```
> db.chr1.remove( { "name" : "uc001aaa.3"});
WriteResult({ "nRemoved" : 1 })
> db.chr1.find().pretty();
{
        "_id" : ObjectId("6487c410235597cf4bc92782"),
        "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"
}
{
        "_id" : ObjectId("6487c41c235597cf4bc92783"),
        "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"
}
```

Figura 87. Eliminar el gen uc001aaa.3.
Fuente: elaboración propia.

Ahora vamos a eliminar toda la colección con el comando `drop`

```
db.dropDatabase();
```

2. Bases de datos NoSQL

2.9. Importar ficheros JSON a MongoDB

Vamos a trabajar ahora importando un fichero en formato JSON que nos proporciona el navegador genómico UCSC.

Descargamos el fichero JSON del siguiente enlace:
https://api.genome.ucsc.edu/getData/track?genome=hg19;track=knownGene;chrom=chr1

Este fichero contiene la información de todos los genes conocidos del cromosoma 1.

Bajamos el fichero y lo guardamos con el nombre `hg19chr1.json`

Para importar el fichero a MongoDB abrimos un nuevo terminal en la carpeta de la máquina virtual donde hemos guardado el fichero y escribimos

```
mongoimport --db hg19 --collection chr1 --drop --file hg19chr1.json
```

Vamos a analizar esta instrucción:

El comando `mongoimport` se ejecuta **fuera** del cliente (mongo) y tiene varias opciones o parámetros

- `--db` indica la base de datos. Si existe la utiliza para crear la colección de documentos, si no existe la crea.
- `--collection db` indica la colección. Si existe, la utiliza para insertar en ella los documentos del fichero que importamos, si no existe la crea.
- `--drop elimina` los documentos previos de la colección, si existe.
- `--file` indica el fichero que vamos a importar. Si los documentos estuvieran dentro de un *array* tenemos que añadir la opción `--jsonArray`

Si nos conectamos ahora al servidor de MongoDB con el cliente mongo y miramos las bases de datos con `show dbs`, vemos cómo se ha creado la base de datos **hg19.**

Si nos conectamos a la base de datos **hg19** con `use hg19` y vemos las colecciones con `show collections` podemos ver la colección **chr1** que hemos creado y ya podemos trabajar con ella.

2. Bases de datos NoSQL

2.10. Buscar en un *array* de documentos

Si nos fijamos en la estructura del fichero JSON importado, solo contiene un documento JSON. Y en uno de los campos, el llamado **knownGene** contiene un *array* de documentos JSON, y cada documento del *array* contiene información de los genes del cromosoma 1: nombre, hebra, cromosoma, etc.

Esto nos obliga a conocer cómo se trabaja con los contenidos de los *arrays* si queremos gestionar correctamente esta información.

Tenemos que usar el `dot.notation`.

Por ejemplo, vamos a mostrar la información del gen `uc009vis.3`.

Esta es la instrucción:

```
db.chr1.find({ "knownGene.name": "uc009vis.3"}, {"knownGene.$": 1}).pretty();
```

Figura 88. Mostrar la información del gen uc009vis.3.
Fuente: elaboración propia.

Vemos como, para referirnos al nombre del gen, escribimos `"knownGene.name":` o sea, el nombre del campo del documento que es un *array* de documentos JSON, el `knownGene`, punto y a continuación el nombre del campo de los documentos que se encuentran en el *array* `name`. Con `knownGene.$": 1` indicamos que nos muestre todos los campos de los documentos encontrados en el *array* `knownGene`.

El `dot.notation` también nos sirve para referirnos a campos de documentos embebidos.

2. Bases de datos NoSQL

2.11. Agregaciones en MongoDB

Para trabajar a fondo con los elementos de un *array* es mejor utilizar el `framework aggregation` que nos permite MongoDB muchas opciones y realizar tuberías *pipelines* para gestionar la información almacenada.

Operaciones de agregaciones

Las operaciones de agregación son herramientas de MQL que nos ayudan a procesar documentos y a retornar resultados calculados. Las operaciones de agregación se utilizan mayoritariamente para:

- Agrupar valores de varios documentos.
- Procesamiento y operaciones para el retorno de resultados.
- Analizar cambios de datos a lo largo del tiempo.

Tuberías y transformaciones

Para realizar el procesamiento de documentos, MongoDB se basa en el patrón de filtro de tubería, utilizado comúnmente en arquitecturas de software. Este patrón consta de una o más etapas, en donde cada etapa efectúa una operación con los datos de entrada, y la salida o resultado se la entrega a la siguiente etapa para su procesamiento.

Para aplicar este patrón, MongoDB utiliza una serie de operadores ya definidos para poder procesar documentos.

Los operadores más utilizados en tuberías:

- Filtrado de documentos con criterios: `$match`.
- Orden de documentos: `$sort`.
- Selección de campos en específico: `$project`.
- Agrupación de documentos: `$group`.
- Sacar los elementos de un *array*: `$unwind`.

Sería necesario otro curso para profundizar en todas las opciones que nos ofrece MongoDB.

Veamos un ejemplo de una operación de agregación. La función `aggregate()`.

Entre muchas otras opciones, `aggregate` nos permite sacar los elementos de un *array* con el operador `$unwind`, hacer agrupaciones con `$group` y contar con `$sum`.

Vamos a contar cuántos genes contiene cada hebra del ADN del cromosoma 1.

```
db.getCollection('chr1').aggregate([{$unwind:"$knownGene"},{$group:{_id:"$knownGene.strand",cantidad:{$sum:1}}}])
```

```
> db.getCollection('chr1').aggregate([{$unwind:"$knownGene"},{$group:{_id:"$knownGene.strand",cantidad:{$sum:1}}}]);
{ "_id" : "-", "cantidad" : 3894 }
{ "_id" : "+", "cantidad" : 4073 }
>
>
>
```

Figura 89. Mostrar cuántos genes contiene cada hebra del ADN del cromosoma 1.
Fuente: elaboración propia.

Resumen

A nivel genómico, habitualmente trabajamos con miles de registros durante un análisis bioinformático convencional. Pese a que las herramientas de análisis de ficheros de texto basadas en el uso de comandos del terminal de GNU/Linux nos permiten acceder fácilmente a nuestros datos, los sistemas de gestión de bases de datos como MySQL o MongoDB resultan el medio ideal para gestionar grandes volúmenes de datos de forma eficiente y rápida.

En este módulo os hemos mostrado un gran abanico de comandos basados en el lenguaje SQL para consultar las tablas de nuestras bases de datos relacionales y algunos comandos específicos de MongoDB para gestionar la información de una base de datos no relacional NoSQL orientada a documentos.

Junto con este inventario de instrucciones, hemos aprendido también a modelar correctamente las entidades del problema real que deseamos aproximar mediante estrategias bioinformáticas.

Actividades

1. Llevamos a la práctica sobre tu propio SGBD MySQL los ejemplos mostrados durante este módulo. Intentamos enriquecer cada base de datos con nuevas tablas que modelen entidades o relaciones que no aparecían originalmente en los casos estudiados. Ampliamos el conjunto de atributos de las tablas para describir con más precisión las instancias reales que se muestran a lo largo del texto.

2. Diseñamos una base de datos para almacenar información sobre un entorno complejo natural que te gustaría modelar: un ecosistema, el cuerpo humano, la célula o el universo. Reflexionamos sobre las entidades, sus atributos y las relaciones entre estas, que son necesarias en cada caso para especificar dicho modelo. Repetimos el ejercicio con un entorno generado por el ser humano (por ejemplo, un automóvil).

3. Implementamos una base de datos en MySQL que permita llevar el registro de todas las actividades de un servidor web genérico que recibe peticiones para ejecutar una serie de servicios: páginas más visitadas, tipo de servicios solicitado, volumen de datos, tiempo de respuesta, usuarios, número de páginas consultadas por cliente, dirección IP y nombre de la máquina, país de procedencia, etc.

4. Ampliamos nuestro catálogo de genes pensando en futuras ampliaciones. Podemos añadir una entidad PROTEINAS para almacenar información estructural o sobre dominios funcionales. También sería interesante incorporar una entidad CROMOSOMAS, relacionada con la tabla GENOMAS, donde guardar otras características, como el número de bases o el contenido en G+C.

5. Importamos algunos ficheros JSON que nos proporciona UCSC Genome Browser https://genome.ucsc.edu/goldenPath/help/api.html#REST a MongoDB creando para cada uno de ellos una base de datos nueva y una nueva colección. Realizamos alguna consulta simple a las colecciones y alguna consulta en un documento situado en un campo *array* de documentos.

Ejercicios de autoevaluación

1. Definid el modelo entidad-relación en el diseño de bases de datos.
2. Describid las diferencias entre bases de datos y gestores de base de datos.
3. Describid qué es una tabla en el modelo relacional.
4. ¿Qué son las claves primarias?
5. ¿Qué son las claves foráneas?
6. Definid qué es una relación 1:N.
7. Recordad qué símbolo debéis emplear siempre al final de cada comando SQL.
8. Mencionad cinco comandos esenciales del lenguaje SQL.
9. Describid el comando de SQL para explorar el contenido de las tablas.
10. ¿Qué tipo de operaciones se realizan sobre registros agrupados?
11. Diferenciad entre los comandos DISTINCT y LIMIT.
12. Enumerad los tres tipos de uniones a realizar entre dos tablas.
13. Definid la utilidad de una subconsulta.
14. ¿Qué dos comandos son útiles para borrar tablas, en qué se diferencian?
15. Indicad la instrucción de SQL para ejecutar un fichero de comandos.
16. Indicad la instrucción de SQL para cargar un fichero de datos.
17. ¿Qué formato cumplen los ficheros de texto para ser empleados de ese modo?
18. Indicad el programa de GNU/Linux que copia una base de datos en un fichero.
19. ¿Cómo puede iniciarse el cliente de MySQL desde la línea de comandos?
20. ¿Cómo se conceden autorizaciones a un usuario sobre una base de datos MySQL?
21. ¿Cómo se muestra un documento con un formato más legible?
22. ¿Cómo se crea una nueva colección de documentos en MongoDB?
23. ¿Cómo se cuentan los documentos de una colección en MongoDB?
24. ¿Qué es la sintaxis dot.notation en MongoDB?
25. ¿Qué es un documento embebido en un fichero JSON?

Solucionario

1. El modelo entidad-relación se basa en el uso de dos tipos de elementos para modelar un entorno real. Las entidades modelan cada clase de elementos de la realidad. Las relaciones modelan las asociaciones entre las instancias de cada entidad en dicho entorno.

2. Una base de datos es un conjunto de informaciones organizadas para fomentar un acceso eficiente a estas; un gestor de base de datos es precisamente el programa que implementa el mantenimiento permanente de la base de datos, ofreciendo además mecanismos para acceder a estos.

3. Una tabla es una estructura que agrupa una colección de elementos (instancias) de la misma clase. Generalmente, una tabla modela una entidad junto con sus atributos, aunque puede ser necesaria también para implementar algunas relaciones entre entidades dentro de la base de datos.

4. La clave primaria de una tabla es el atributo que identifica de forma unívoca a cada instancia o elemento en su interior. Por ello, el valor de este atributo no puede repetirse entre instancias diferentes.

5. La clave foránea de una tabla es la clave primaria de otra tabla, asegurando la integridad del modelo, dado que cada instancia en la primera tabla deberá existir también en la segunda.

6. Una relación 1:N entre dos entidades indica que cada instancia de la primera entidad puede asociarse con *N* instancias en la segunda tabla. Puede implementarse en la segunda tabla directamente con un atributo que tome por valor alguno en el rango correcto para la primera.

7. Cualquier comando que introducimos en el intérprete de MySQL debe terminar obligatoriamente con el símbolo **;**, para ser ejecutado correctamente.

8. Por ejemplo: `CREATE DATABASE`, `CREATE TABLE`, `SELECT`, `LOAD DATA` y `GRANT`.

9. El comando `SELECT...FROM...WHERE` permite realizar una consulta sobre los registros de las tablas que cumplen ciertas condiciones.

10. Cuando se agrupan instancias con **`GROUP BY`**, es posible calcular medias aritméticas, mínimos, máximos o cuentas de totales (**`AVG`**, **`MIN`**, **`MAX`**, **`COUNT`**).

11. La cláusula DISTINCT sirve para eliminar valores repetidos. La cláusula LIMIT es útil para mostrar solo las primeras instancias de una tabla.

12. A la hora de comparar dos tablas con el comando `JOIN` empleando un atributo en común, podemos buscar las parejas de valores presentes en ambas tablas o aquellas que solo aparecen en una de ellas (**`LEFT`** o **`RIGHT`**).

13. Una subconsulta permite generar un grupo de resultados en forma de tabla temporal. Dicha tabla auxiliar podrá ser interrogada, a su vez, por la consulta principal que alberga la subconsulta en su interior.

14. El comando `DROP TABLE` elimina la tabla completamente (definición y contenido). El comando `DELETE`, en cambio, elimina únicamente determinados registros de acuerdo a una condición.

15. La instrucción `SOURCE`.

16. La instrucción `LOAD DATA`.

17. El fichero de texto debe estar tabulado en columnas, que corresponden a los atributos de las tablas.

18. El programa **mysqldump** realiza el volcado de la base de datos.

19. El comando sería: `mysql -u usuario -p`

20. Con la instrucción `GRANT`

21. Con el comando o función `pretty()`

22. Con la función `db.createCollection("nombreColeccion");`

23. Con la función `count();`

24. Es la forma de identificar un campo situado en un documento situado en campo *array* de documentos de la forma `nombre_campo_array.nombre_campo_documento`

25. Es un documento situado en un campo de un documento JSON.

Bibliografía

Conrad Bessant, Ian Shadforth y 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 y 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.

Webgrafía

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/