

El entorno de trabajo UNIX

Enrique Blanco García

PID_00170741

Ninguna parte de esta publicación, incluido el diseño general y la cubierta, puede ser copiada, reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea éste eléctrico, químico, mecánico, óptico, grabación, fotocopia, o cualquier otro, sin la previa autorización escrita de los titulares del copyright.

Índice

Introducción	3
Objetivos	4
1. Introducción a los entornos de trabajo UNIX	5
1.1. Arquitectura de un ordenador	5
1.2. Funciones de un sistema operativo	6
1.3. La familia de sistemas operativos UNIX	7
1.4. Programas y procesos	9
1.5. El sistema de ficheros	11
1.6. Funcionamiento de las máquinas virtuales	14
1.7. Configurando una máquina virtual con Ubuntu	16
1.8. El terminal como herramienta de trabajo	20
1.9. Gestión básica de ficheros	24
1.10. Accediendo al contenido de los ficheros	28
1.11. Gestión básica de procesos	34
1.12. Buscar, ordenar y asociar ficheros	36
1.13. Combinación de comandos	39
1.14. El lenguaje de procesado de archivos GAWK	42
1.15. expresiones regulares: aplicación en el terminal	47
1.16. Definición de nuevos comandos	50
1.17. Diseño de protocolos automáticos en el terminal	51
1.18. Transferencia de ficheros desde el terminal	57
1.19. Ejemplo práctico 1: Analizando el genoma humano	58
Resumen	71
Actividades	72
Ejercicios de autoevaluación	73
Solucionario	74
Bibliografía	75
Índice alfabético	76

Introducción

La secuencia del genoma de múltiples especies está a disposición de cualquier persona que posea un ordenador con una simple 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. Sólo 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 aplicaciones computacionales utilizadas habitualmente por un bioinformático para procesar información genómica. La familia de sistemas operativos UNIX 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, focalizaremos nuestro interés en el manejo del terminal de UNIX, la herramienta de trabajo habitual para un bioinformático. Aprenderemos los comandos básicos, y cómo pueden combinarse éstos 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 establecido en este módulo, una vez finalizada la etapa de aprendizaje, debéis lograr la consecución de los siguientes objetivos:

1. Conocer las funciones del sistema operativo
2. Reconocer la familia de sistemas operativos UNIX
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

Nadie puede ignorar que, desde el nacimiento de las primeras computadoras, a mitad del siglo pasado, hasta la aparición de los microprocesadores de última generación (integrados en teléfonos móviles y electrodomésticos), el progreso tecnológico ha comportado una mejora sustancial en la calidad de vida del ser humano. Una de las claves de este vertiginoso avance es el constante acercamiento de las máquinas a las personas. En sus orígenes, sólo los técnicos formados específicamente para ello tenían acceso restringido a las primitivas computadoras. Resultaba indispensable, por tanto, la intervención directa de operadores humanos que actuaban sobre las máquinas para su puesta en funcionamiento. Paradójicamente, con la aparición de las aplicaciones (en inglés, *software*), que ocultan la complejidad técnica de los ordenadores (en inglés, *hardware*), cualquier persona puede acceder hoy en día a esta tecnología, convirtiendo poderosas máquinas de cálculo en herramientas de uso prácticamente cotidiano.

Los principales componentes de la arquitectura de un ordenador son:

La unidad central de procesamiento (en inglés, *Central Processing Unit* o CPU). El procesador ejecuta cada ciclo de reloj una instrucción del programa actual. Para realizar cualquier operación, los datos deben estar almacenados en sus registros internos.

La memoria central (en inglés, *Random Access Memory* o RAM). La memoria del ordenador es un dispositivo de almacén 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.

Los periféricos. Tanto los dispositivos de entrada y salida clásicos (pantalla, teclado, ratón o impresora) como los dispositivos de almacenamiento secundario (discos duros o lápices de memoria) o la conexión a la Red, cualquier dispositivo externo conectado a un ordenador se considera un periférico. La comunicación entre estos aparatos y la arquitectura básica se implementa mediante programas adaptadores (en inglés, *device drivers*).

El bus de comunicaciones. La transmisión de información entre cualquier componente del ordenador (procesador, memoria y periféricos) se lleva a cabo a través del bus de datos central del ordenador.

ENIAC, el primer ordenador

John Eckert y John Mauchly presentaron en Filadelfia el 15 de febrero de 1946 el Electronic Numerical Integrator And Calculator (ENIAC). Esta máquina de 30 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.

Lectura complementaria

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

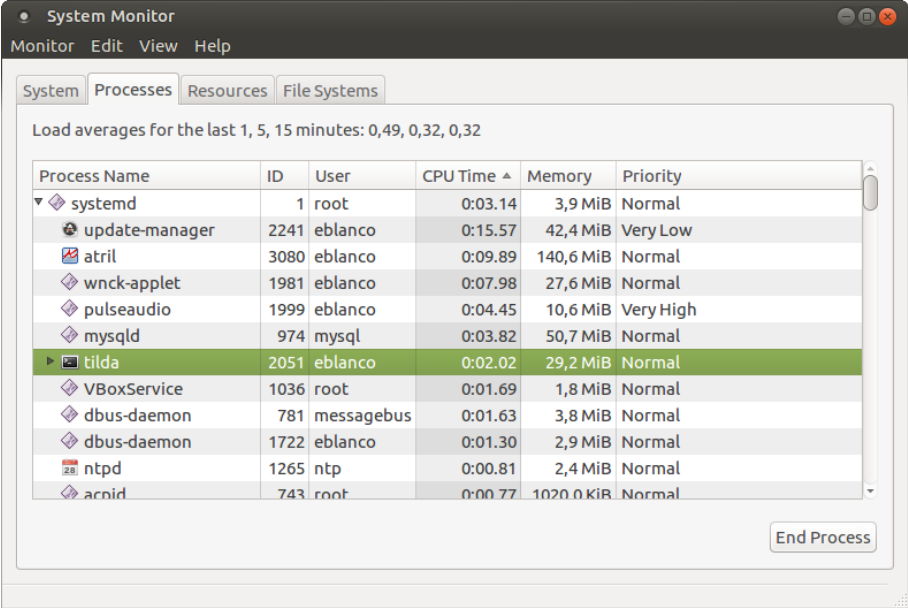
1.2. Funciones de un sistema operativo

Junto con la capacidad de miniaturización de la tecnología, el nacimiento de los sistemas operativos resulta fundamental para comprender la sorprendente penetración de la informática en nuestra sociedad. Los sistemas operativos disfrazan, a ojos del usuario, la máquina física como un entorno virtual donde los detalles técnicos de funcionamiento de cada dispositivo son ocultados bajo un interfaz uniforme que proporciona un manejo más cómodo. Para lograrlo, cada componente posee su propio programa gestor que enmascara su funcionamiento interno.

El Sistema Operativo (SO) es el programa que desempeña un rol de intermediario entre el usuario y la máquina, dotando a ésta de un 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.

Habitualmente se considera que el SO proporciona al usuario la ilusión de poseer un ordenador con capacidades ilimitadas. Para producir este espejismo deben gestionarse eficientemente todos los componentes disponibles en el ordenador. Dado que la frecuencia del reloj del procesador es superior a la frecuencia de percepción de los seres humanos, el ordenador puede responder a cientos de eventos en décimas de segundo, produciendo entonces la falsa sensación de poseer más capacidad y más recursos de los existentes. Por ejemplo, como muestra la Figura 1, mientras estamos realizando una actividad en nuestra propia estación de trabajo, hay cientos de programas ejecutándose simultáneamente (aunque en cada instante, sólo uno de ellos tiene acceso a la CPU).

Figura 1. Listado de programas en funcionamiento en un entorno Linux Ubuntu MATE



The screenshot shows the 'System Monitor' application window. The 'Processes' tab is selected, displaying a table of running processes. The table has columns for Process Name, ID, User, CPU Time, Memory, and Priority. The process 'tilda' is highlighted in green. Below the table, there is an 'End Process' button.

Process Name	ID	User	CPU Time	Memory	Priority
systemd	1	root	0:03.14	3,9 MiB	Normal
update-manager	2241	eblanco	0:15.57	42,4 MiB	Very Low
atril	3080	eblanco	0:09.89	140,6 MiB	Normal
wnck-applet	1981	eblanco	0:07.98	27,6 MiB	Normal
pulseaudio	1999	eblanco	0:04.45	10,6 MiB	Very High
mysqld	974	mysql	0:03.82	50,7 MiB	Normal
tilda	2051	eblanco	0:02.02	29,2 MiB	Normal
VBoxService	1036	root	0:01.69	1,8 MiB	Normal
dbus-daemon	781	messagebus	0:01.63	3,8 MiB	Normal
dbus-daemon	1722	eblanco	0:01.30	2,9 MiB	Normal
ntpd	1265	ntp	0:00.81	2,4 MiB	Normal
acpid	743	root	0:00.77	1020,0 KiB	Normal

Figura 1

El sistema operativo se encarga de gestionar la ejecución concurrente de numerosos programas del sistema junto con nuestras propias aplicaciones.

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

1. El procesador
2. La memoria y los dispositivos de almacen secundarios
3. Dispositivos de entrada y salida de datos
4. El sistema de archivos y directorios
5. La conexión a la red y con otras máquinas
6. La seguridad y privacidad de los datos
7. La información interna sobre el sistema

1.3. La familia de sistemas operativos UNIX

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. La leyenda cuenta, de hecho, que el término UNICS derivaba en realidad del vocablo *eunuco* (castrado), al considerarse este sistema, una versión limitada de MULTICS. Para evitar esta desagradable coincidencia, el nuevo SO fue bautizado finalmente como UNIX.

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.

Lecturas complementarias

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

Andrew S. Tanenbaum (1995). *Distributed operating systems*. Prentice-Hall. ISBN: 0132199084.

Lecturas complementarias

Brian W. Kernighan and Rob Pike (1984) *Unix Programming Environment*. Prentice Hall. ISBN: 013937681X.

Brian W. Kernighan and D.M. Ritchie (1988) *C Programming Language (2nd Edition)*. Prentice Hall. ISBN: 0131103628.

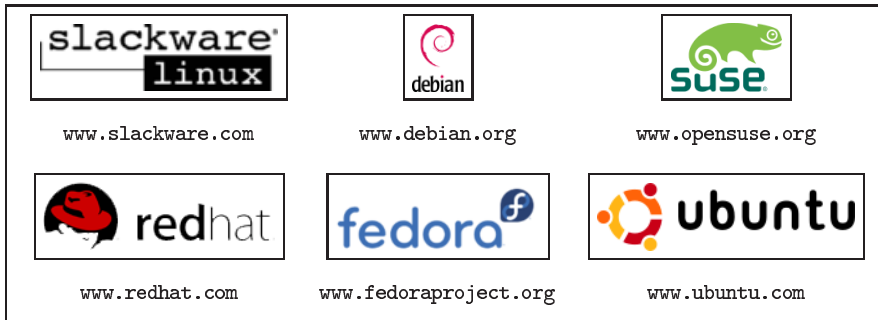
Otras versiones de UNIX

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.

Enlace de interés

Para visualizar el árbol completo de distintas versiones de UNIX, os recomendamos acceder a la siguiente página *web*: www.levenez.com/unix/

Figura 3. Distribuciones de Linux más populares



1.4. Programas y procesos

Como se ha mencionado anteriormente, los ordenadores ejecutan programas. Los programas son listados de instrucciones que indican cómo procesar un conjunto de datos. Para poder ejecutarse, un programa escrito en un lenguaje de alto nivel (p.e. C, PHP, Python o Java) debe ser antes procesado con el compilador adecuado, traduciéndose a lenguaje máquina (series de unos y ceros inteligibles para el ordenador), como muestra la Figura 4:

Figura 4. Generación de ficheros ejecutables

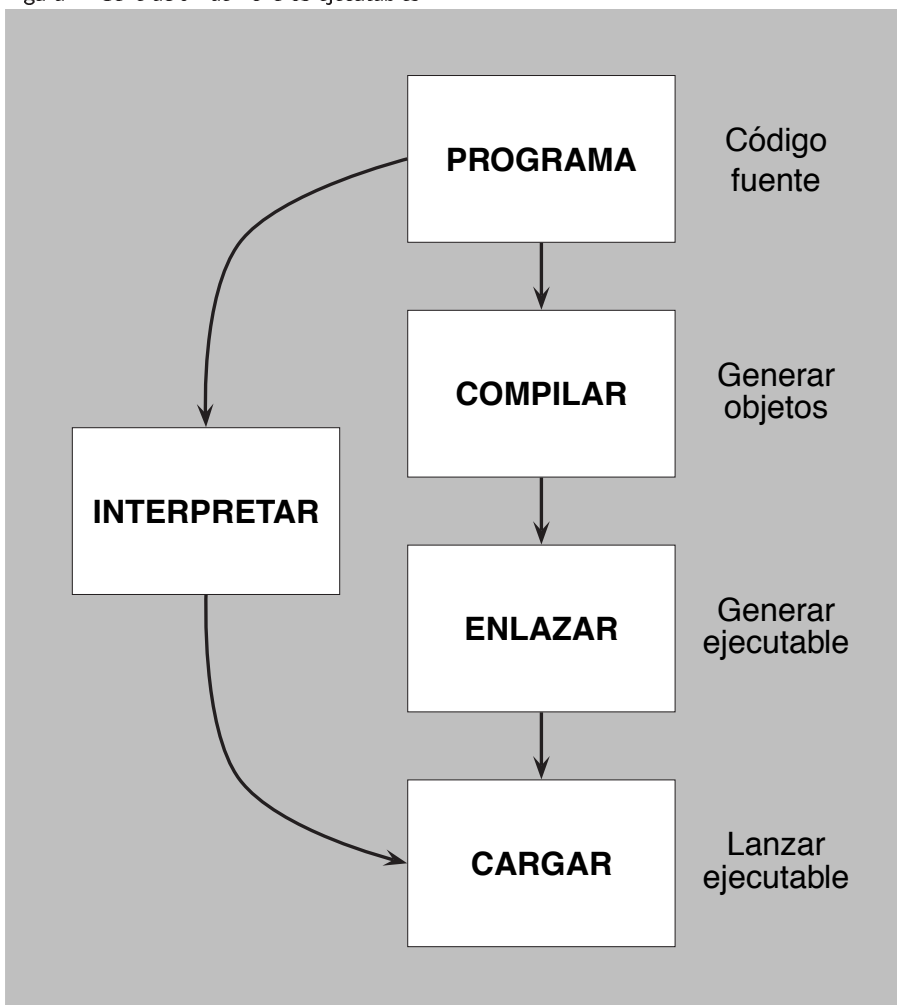


Figura 4

Los programas implementados en lenguajes de alto nivel deben ser compilados y enlazados con librerías del sistema, produciendo un fichero binario. En el caso de los lenguajes interpretados, cada línea es ejecutada individualmente sin generar ningún archivo adicional.

Los compiladores son programas encargados de realizar el análisis léxico, sintáctico y semántico del código fuente. 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 o binario.

El usuario puede ejecutar este archivo cuando sea preciso. La depuración de los programas escritos en alto nivel es costosa, siendo rentable sólo en los casos en que el rendimiento óptimo de éstos, en términos de tiempo de ejecución y espacio de memoria, es capital. Para realizar tareas más sencillas, es posible diseñar prototipos (en inglés, *scripts*) empleando lenguajes orientados a la producción rápida de programas, como Perl o Python. Estos lenguajes de *scripting* poseen un juego de instrucciones específicamente diseñado para facilitar la adquisición y tratamiento de ficheros de texto. Los intérpretes de éstos procesan los programas instrucción a instrucción, ahorrándose 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 fichero binario, 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).

Un proceso es un entorno de ejecución de una instancia de un programa concreto, gestionado por el SO para planificar su lanzamiento en el procesador.

Dado que sólo 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 paradójicamente 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 sólo existe un procesador. La Figura 5 muestra los diferentes estados de un proceso durante su funcionamiento. En un momento dado, éste puede estar en posesión del procesador, encontrarse bloqueado en espera de recibir datos de algún dispositivo o esperar en la cola de procesos debido a la ejecución de más programas.

Lectura complementaria

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

Ved también

Para profundizar en los conceptos básicos de la programación os recomendamos la asignatura *Fundamentos de la Programación*

Figura 5. Estados de un proceso

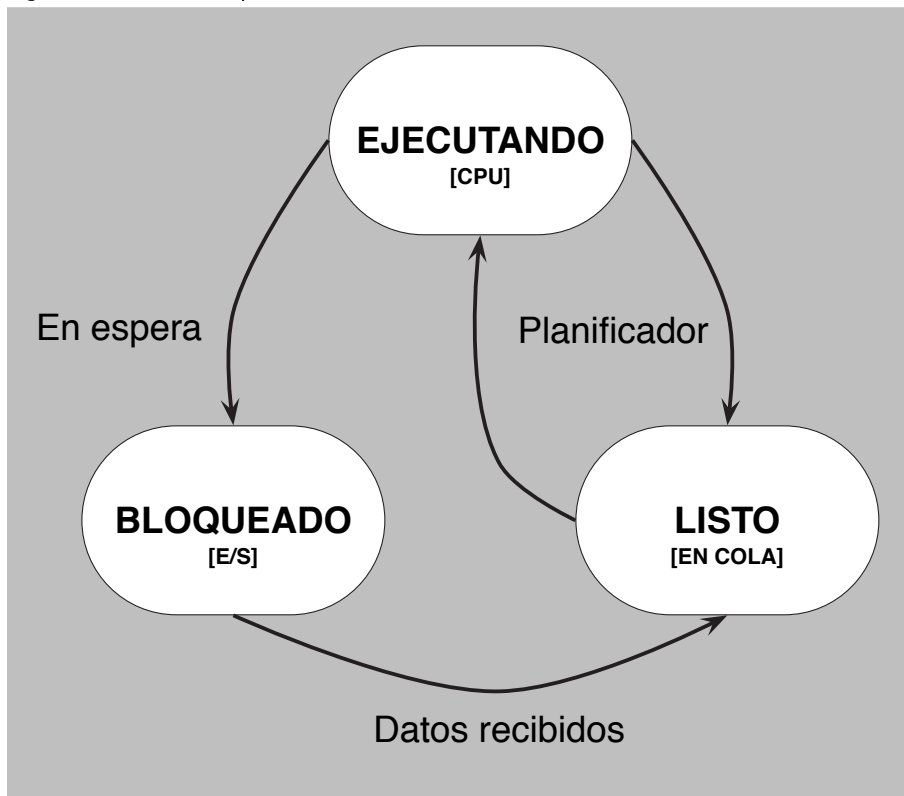


Figura 5

Un proceso bloqueado en espera de un recurso de entrada/salida ingresa en la cola de ejecución una vez recibe la información.

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 paralelizar sus programas, permitiendo que determinados fragmentos de éstos 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.

1.5. 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, CDs, DVDs, lápices de memoria o a través de la nube en la Red. Esta clase de memoria secundaria comparte un método de organización común denominado sistema de ficheros. 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.

Un fichero es una colección de *bytes* asociada a un nombre. Dicho identificador permite que esa información sea guardada en un dispositivo de almacenamiento para ser recuperada en el futuro.

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 éstos. 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 éste.

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.

El sistema de ficheros dota al usuario de mecanismos lógicos para localizar los ficheros a través de su nombre simbólico y una ruta de acceso (en inglés, *path*). Los ficheros son agrupados en carpetas o directorios, conformando una jerarquía que proporciona una organización coherente para el usuario. Para construir una ruta que pasa 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 éste (*path* absoluto), es decir, toda la serie de directorios desde la raíz hasta ese punto 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). En el ejemplo mostrado a continuación (Figura 6), el usuario puede utilizar la ruta absoluta `/users/eblanco/program.c` para acceder directamente al fichero `program.c`. En cambio, si nos encontramos dentro del propio directorio `/users`, podemos emplear la ruta relativa `eblanco/program.c` para acceder al mismo archivo desde ese punto.

En UNIX, el directorio inicial o raíz (en inglés, *root*) recibe por convenio el nombre lógico de "/". Para acceder a cualquier fichero o directorio debemos descender desde este punto a lo largo del sistema de ficheros.

Ved también

Podéis profundizar en los comandos del terminal para recorrer el árbol de directorios en la sección 1.9

Figura 6. Sistema jerárquico de ficheros

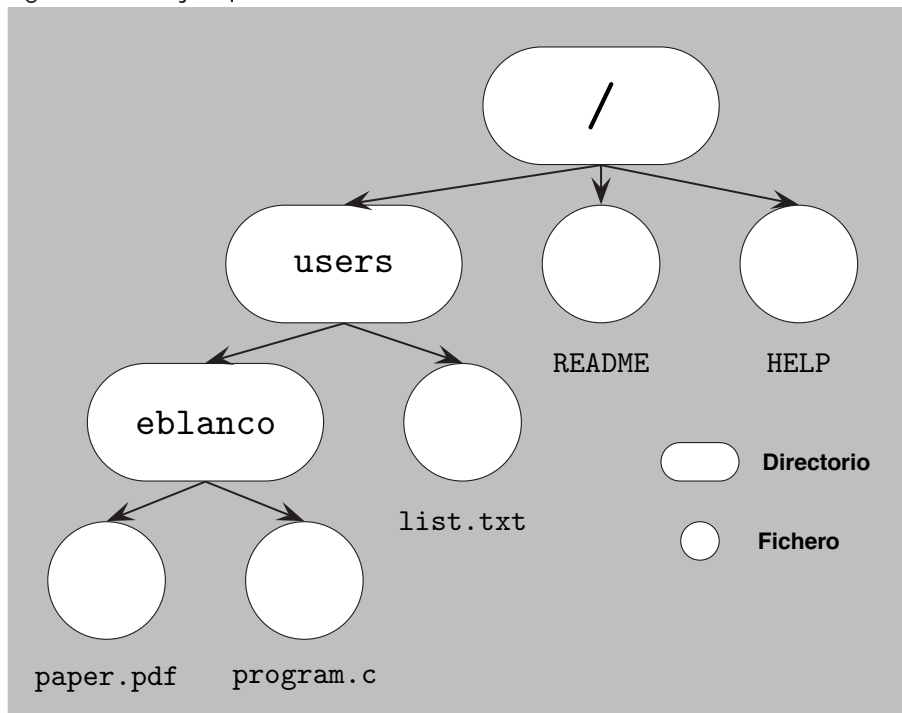


Figura 6

Ejemplo de un sistema de ficheros con tres directorios y cinco ficheros. En UNIX, una vez montados los dispositivos, no es necesario saber en qué unidad física se encuentra cada volumen.

En UNIX, el administrador de la máquina puede combinar los sistemas de archivos contenidos en diferentes medios mediante el montaje de éstos (en inglés, *mount*) dentro del árbol de ficheros general. Los discos duros externos y lápices de memoria son automáticamente 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. Una vez dentro del sistema, la independencia de dispositivos permite que veamos estos elementos como un directorio más dentro de la jerarquía de ficheros. Otra peculiaridad de los entornos UNIX es la creación de 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 pueden contener la ruta de acceso del fichero original (en inglés, *soft links*) o proporcionar un acceso físico compartido a éste con un nombre diferente (en inglés, *hard links*). Con este mecanismo, el usuario puede apuntar a un mismo fichero desde varios lugares del sistema, ahorrándose la existencia de múltiples copias de éste.

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: usuario (*user*), 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*), si es un fichero binario. 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 éste 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. Estos derechos se codifican normalmente con números binarios como se muestra en el siguiente ejemplo:

Figura 7. Permisos de acceso a un fichero

r	w	-	r	-	-	r	-	-
1	1	0	1	0	0	1	0	0
	6		4			4		
	r+w		r			r		
	u		g			o		

Ved también

Para llevar a la práctica la administración de los derechos de un archivo, os recomendamos en UNIX el uso del comando `chmod`, que se estudia en el subapartado 1.9 de este módulo.

Figura 7

Un fichero dotado de permiso de lectura/escritura para su propietario y sólo de lectura para el resto de dominios. Se muestra cada autorización en código binario, decimal y alfabético.

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 complicaban el acceso a esta tecnología. De hecho, hoy en día podemos probar fácilmente *in situ* la mayoría de 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

Los usuarios de ordenadores Apple Mac, pueden acceder a la mayoría de funciones de UNIX directamente cuando trabajan bajo el sistema Mac OS-X™.

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 ésta la emulación logra que el SO invitado crea que trabaja sobre una verdadera máquina física funcionando a su entera disposición. Por ejemplo, en la Figura 8 mostramos como un usuario de Mac OS-X™ puede trabajar a la vez con una MV funcionando bajo Windows™ y otra gobernada por Linux Ubuntu, ambas activas en su escritorio, sin interferir en su configuración original.

Figura 8. Máquinas virtuales gestionadas por Oracle VirtualBox™ en un entorno Mac OS-X™

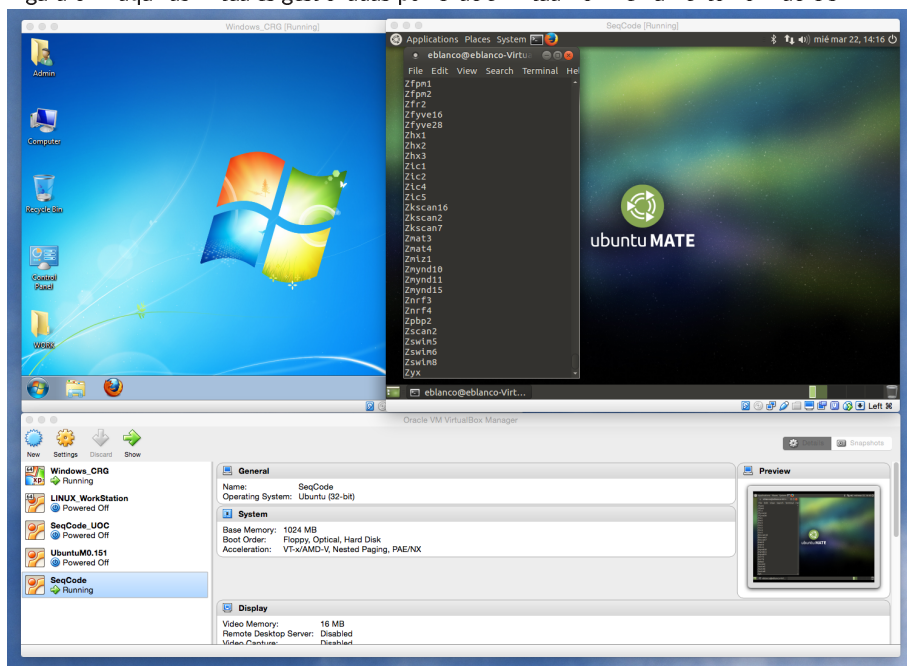


Figura 8

Las dos máquinas virtuales se muestran en dos ventanas separadas en la parte superior de la imagen y el panel de control del gestor de virtualización está ubicado en una ventana en la parte inferior de ésta.

El sistema de ficheros de la MV invitada se almacena físicamente en 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 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.

1.7. Configurando una máquina virtual con Ubuntu

A lo largo de esta sección vamos a explicar los pasos básicos para que dispongáis de una plataforma UNIX donde llevar a la práctica los ejemplos explicados a lo largo de este texto. En primer lugar, os recomendamos la aplicación Oracle VirtualBox™ como *software* de virtualización por su versatilidad para funcionar bajo múltiples plataformas. En segundo lugar, os proponemos emplear la distribución Ubuntu de Linux como SO invitado por tener vocación de aproximar el sistema Linux a los usuarios menos experimentados de estas plataformas. Dado que existen distintas versiones de Ubuntu, hemos decidido emplear una distribución ligera denominada MATE, porque posee una interfaz clásico muy eficiente. En tercer lugar, finalmente, realizaremos un inventario de los paquetes de programas de Linux que os resultarán necesarios para trabajar en entornos bioinformáticos.

El protocolo básico para disponer de una MV de VirtualBox funcionando bajo Linux Ubuntu MATE en nuestro ordenador consiste en los siguientes cuatro pasos:

1. Descargar e instalar la versión apropiada del gestor de virtualización VirtualBox para el SO de nuestro ordenador.
2. Descargar el CD de instalación de la distribución de Linux que deseamos instalar en nuestra MV.
3. Crear una nueva MV vacía y configurarla para que arranque virtualmente con el CD de instalación de Linux.
4. Descargar e instalar los paquetes de programas necesarios en la MV para trabajar con Linux en un entorno bioinformático.

Dada su versatilidad, a lo largo de este módulo os recomendamos trabajar con el software de virtualización Oracle VirtualBox™ para que dispongáis de una MV funcionando bajo Linux Ubuntu en vuestro ordenador. La aplicación VirtualBox es relativamente fácil de instalar en cualquier entorno y resulta notablemente eficiente en la emulación de otras plataformas. Lógicamente, nuestro ordenador debe cumplir con unos mínimos requisitos de capacidad de memoria y rendimiento del procesador para que la emulación resulte mucho más realista. Por esta razón, instalaremos la distribución Ubuntu MATE, que posee un nivel de requerimientos inferior a otras versiones más sofisticadas de Linux.

La mayoría de distribuidores de Linux suministran una versión en vivo de éste (en inglés, *LiveCD*) que puede descargarse en forma de un archivo en formato ISO. Esta clase de ficheros estaba originalmente concebido para escribirse en un CD virgen, convirtiéndolo en un disco de prueba e instalación. Al reiniciar nuestro ordenador con el CD previamente introducido, el sistema automáticamente iniciaba Linux, ejecutando todos los procesos desde el disco óptico. De este modo, era posible restaurar el funcionamiento habitual de nuestro SO original simplemente volviendo a reiniciar la máquina, si previamente habíamos extraído físicamente el CD de la unidad lectora.

Enlace de interés

El gestor de virtualización Oracle VirtualBox™ funciona en numerosas plataformas. Podéis obtener gratuitamente una copia desde la página web www.virtualbox.org

Este era un mecanismo habitual para permitir a un potencial usuario que probase una versión concreta de Linux antes de decidir si lo instalaba finalmente en su ordenador personal.

En una MV, sin embargo, podemos configurar su arranque directo desde una imagen ISO, sin necesidad de crear físicamente ningún CD, evitando involucrar a nuestro entorno de trabajo habitual. Dado que cualquier instalación se producirá sobre el disco duro virtual de nuestra MV, resulta más cómodo instalar Linux directamente sobre ésta una vez la arranquemos por primera vez. De este modo trabajaremos fácilmente con cualquier distribución de Linux en una nueva MV de nuestra VirtualBox. Desde la página *web* de cualquier distribución de Linux, como se muestra en la Figura 9 para la versión MATE de Ubuntu, podemos obtener una copia gratuita de la imagen del sistema.

Enlace de interés

Encontraréis la distribución Ubuntu MATE en la *web* <https://ubuntu-mate.org>

Figura 9. Descarga de la imagen de la distribución Ubuntu MATE para nuestra MV

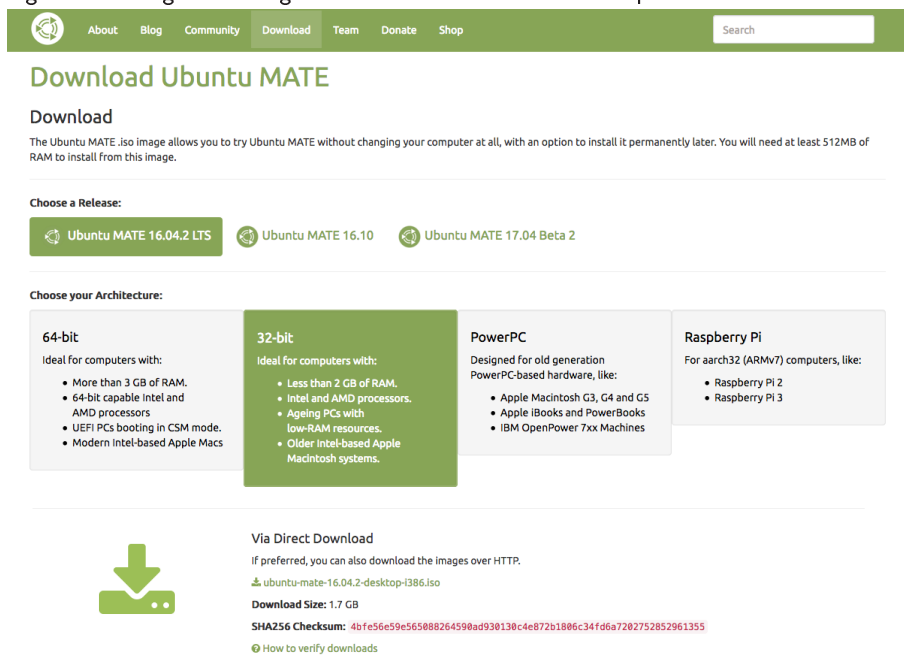
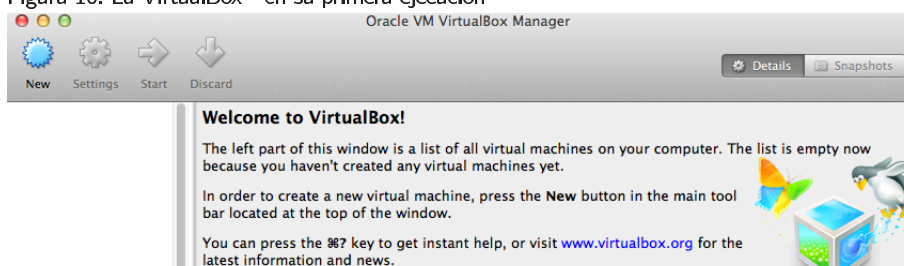


Figura 9

Para un determinado SO, cada dos o tres años suele aparecer una nueva versión. Habitualmente podremos obtener una copia de la última versión estable disponible desde la página de descargas. En la parte inferior observamos la imagen ISO que necesitamos descargar en nuestro ordenador.

Una vez ya hemos instalado el *software* de virtualización VirtualBox™ en nuestra plataforma y hemos descargado una copia de la imagen ISO de Ubuntu MATE en nuestro escritorio, estamos en condiciones de crear nuestra propia MV. Para empezar, procedemos a ejecutar el gestor de máquinas virtuales por primera vez. Observamos en la Figura 10 que el panel izquierdo de la aplicación se muestra vacío porque ninguna MV está instalada por el momento. Cuando finalicemos este protocolo, un acceso directo a nuestra MV aparecerá en este menú. Ahora, si presionamos el botón *New*, crearemos una nueva MV en nuestro inventario de máquinas. Recordad que inicialmente una MV corresponde a un ordenador ficticio que no posee ningún SO preinstalado en su disco duro virtual.

Figura 10. La VirtualBox™ en su primera ejecución



A continuación, como se muestra en la Figura 11, debemos especificar el tipo de SO invitado que instalaremos y la cantidad de recursos reales de nuestro ordenador (procesador, memoria, tamaño de disco duro virtual) que podrán ser utilizados en el futuro por esta nueva MV. Una vez finalizada la configuración inicial debemos presionar el botón *Settings* (en inglés, configuraciones) para acceder a los diferentes dispositivos virtuales asociados. En particular, debemos presionar la pestaña de *Storage* (en inglés, almacenamiento) para insertar la imagen ISO de Ubuntu MATE descargada previamente dentro de la unidad virtual de CD/DVD. El resultado de la operación se muestra en la Figura 11, donde hemos añadido una nueva lectora virtual con la imagen de Ubuntu.

Figura 11. Insertando el CD virtual de Ubuntu en la VirtualBox™

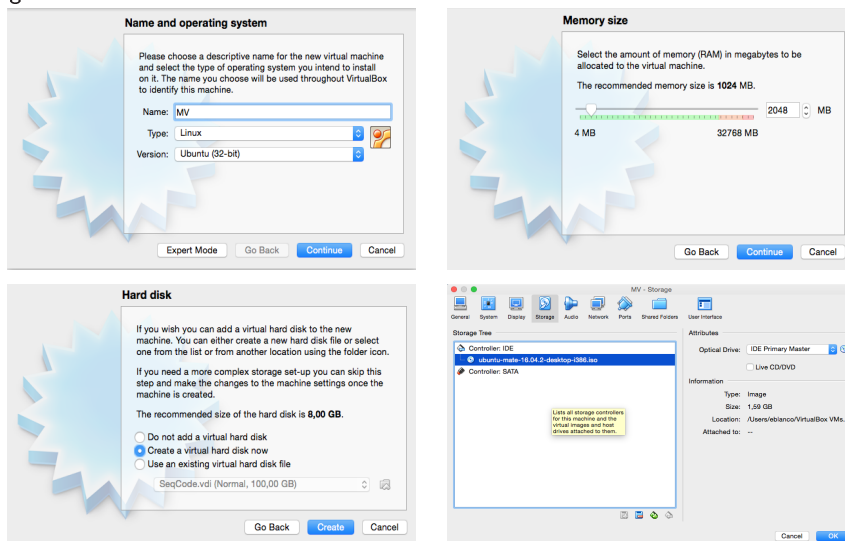


Figura 11

Protocolo de configuración de una MV para arrancar desde un CD virtual de Linux. Es posible que algunas de las pantallas aquí mostradas sean ligeramente distintas con el paso del tiempo.

Ya estamos en condiciones de arrancar nuestra MV por primera vez: presionamos el botón *Start*. VirtualBox™ abrirá una nueva ventana en vuestro escritorio emulando la pantalla de nuestra MV. Dado que estamos arrancando un ordenador con un disco de instalación insertado virtualmente en su lectora de CD/DVD, esto producirá que se inicie la instalación de Ubuntu MATE en nuestra MV desde el fichero ISO obtenido anteriormente. Una vez finalizada la instalación, que precisará de unos minutos, se nos pedirá confirmación para reiniciar la MV. En este punto extraeremos la imagen ISO de la unidad lectora virtual para que la MV trabaje definitivamente desde su disco duro virtual gobernado por Ubuntu.

Es importante recordar que este procedimiento no instalará Ubuntu en vuestro ordenador, que actúa de huésped, sino en el disco duro virtual de la MV que hemos creado a tal efecto.

Figura 12. Primera ejecución de la MV bajo Ubuntu MATE

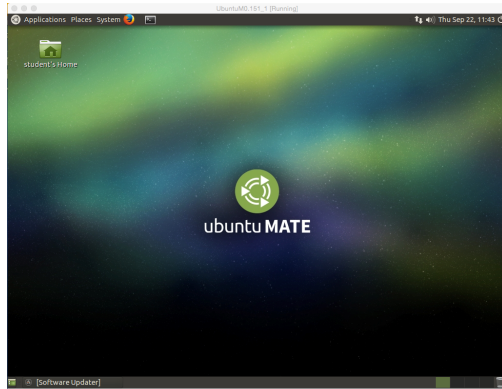
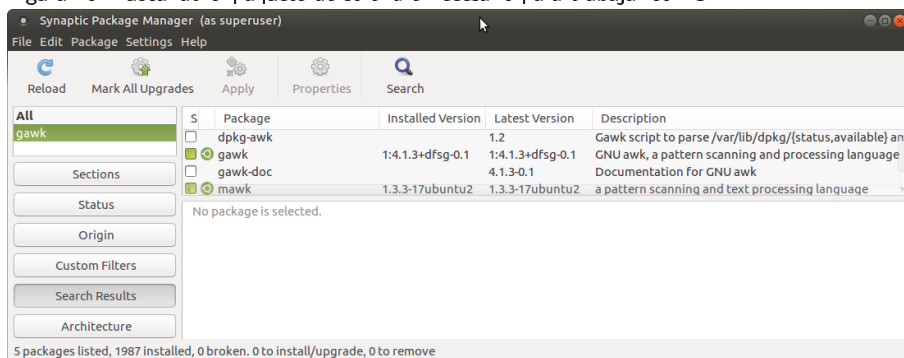


Tabla 1. Conjunto de paquetes recomendados para una instalación completamente funcional

Paquete	Descripción
APACHE	Gestor de servicios <i>web</i>
EMACS	Editor de texto muy extendido con numerosas funciones
GAWK	Lenguaje de reconocimiento de patrones de texto
L ^A T _E X	Lenguaje para la redacción profesional de documentos
MySQL	Gestor de bases de datos relacionales
PHP	Lenguaje para el diseño de servidores <i>web</i>
PHP-MySQL	Librerías de conexión entre PHP y MySQL
R	Lenguaje de análisis de datos estadísticos

Finalmente, dado que las distribuciones estándar de Ubuntu no poseen más que la funcionalidad elemental del sistema, debemos proceder a la instalación en nuestra MV del *software* necesario para llevar a cabo nuestro trabajo (ver Tabla 1 para un listado básico de aplicaciones). Es muy recomendable instalar inicialmente un programa de gestión de la instalación de estos paquetes de programas para realizar esta tarea de forma sistemática. Por ejemplo, la aplicación Synaptic Package Manager, mostrada en la Figura 13, os permitirá gestionar fácilmente la instalación de los programas necesarios para el tipo de análisis bioinformático que vamos a desarrollar en estos materiales. En cualquier momento, con esta herramienta tendremos acceso a la configuración actual de nuestra MV.

Figura 13. Buscando el paquete de *software* necesario para trabajar con GAWK

No siempre estaremos obligados a realizar la instalación completa de un SO invitado en una nueva máquina virtual. El *software* de virtualización VirtualBox™ permite realizar una copia de una MV que funciona con una distribución completa de un determinado sistema, para ser ejecutada directamente en cualquier otra plataforma. Este procedimiento de copia de una MV genera físicamente un único fichero (en inglés, denominado *appliance*), almacenable en cualquier dispositivo convencional o a través de la Red. Una vez obtenido este archivo, es suficiente con importarlo en la aplicación VirtualBox™ para disfrutar en unos minutos de una copia idéntica de la MV original. Esta función de copia resulta extremadamente útil porque permite distribuir fácilmente una configuración específica de una determinada plataforma con un conjunto de paquetes de programas ya preinstalados en su interior.

El formato OVA/OVF

La copia de la MV genera en nuestro ordenador un fichero en formato OVA (Open Virtualization Format Archive).

1.8. El terminal como herramienta de trabajo

El entorno de trabajo de Ubuntu, como cualquier sistema de la familia Linux, incluye un gestor gráfico de ventanas que proporciona al usuario un interfaz agradable para acceder a los recursos de su máquina. Los primitivos sistemas operativos de las primeras computadoras, sin embargo, no poseían entornos interactivos manejables mediante el ratón, ni pantallas con la resolución gráfica que conocemos hoy en día. En aquella época, el usuario se comunicaba con el ordenador utilizando el teclado y un interfaz de línea de comandos de texto (en inglés, *command line*). La infraestructura más habitual por aquellos tiempos consistía en un gran ordenador que servía peticiones introducidas a distancia desde los terminales manejados por los usuarios. Cada terminal estaba formado exclusivamente por un teclado y una pantalla.

A pesar del indudable atractivo y comodidad de los interfaces gráficos actuales, los terminales de línea de comandos poseen numerosas ventajas sobre estos:

- Permiten implementar fácilmente repeticiones de largas series de comandos para resolver un problema evitando errores.
- Poseen un registro interno de los comandos introducidos durante una sesión de trabajo reproducible en otro momento.
- Pueden ejecutar comandos de forma remota en otra plataforma de cálculo más potente ubicada en otro entorno de trabajo.
- El funcionamiento de los terminales de diferentes sistemas es similar, preservándose la mayoría de sus funciones.

En un entorno de investigación en Bioinformática, este tipo de trabajos constituye precisamente el núcleo de las actividades habituales. Por ello, la mayoría de los sistemas actuales mantienen todavía el terminal como una aplicación más. Esta herramienta, como veremos a lo largo de esta asignatura, es ideal para diseñar protocolos de trabajo que acceden en repetidas ocasiones a determinados conjuntos de datos con el objeto de realizar una serie de cálculos intensivos.

Los usuarios de Ubuntu pueden mezclar la utilización de ambas clases de interfaces en función de sus necesidades. De hecho, ya sea empleando ventanas o terminales, el sistema base de ficheros de nuestro ordenador es el mismo, aplicándose sobre los archivos las modificaciones realizadas indistintamente desde cualquiera de las dos aproximaciones. Por ejemplo, en la Figura 14 os mostramos el mismo directorio ya sea visualizado con el entorno gráfico o con el terminal de texto. El escritorio, en particular, es otro directorio más dentro del sistema de ficheros, siendo accesible también desde el terminal.

El terminal

Uno de los típicos iconos asociados a la aplicación del terminal. En UNIX también recibe el nombre de intérprete de comandos (en inglés, *shell*).



Lectura complementaria

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

Figura 14. El mismo directorio visualizado con un interfaz gráfico o con el terminal

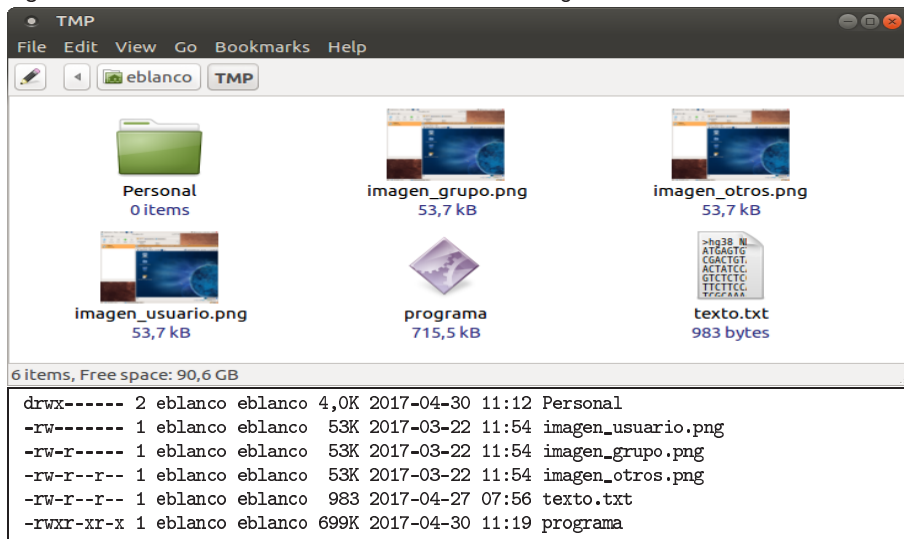


Figura 14

El directorio TMP visto desde un administrador de ventanas o desde el terminal.

El terminal siempre ejecuta el mismo programa, denominado intérprete de comandos. Dicho programa permanece en espera, mientras el usuario no introduzca una nueva orden. Una vez que el usuario ejecuta un comando, el intérprete crea un nuevo proceso para cumplir con esa orden. Existen dos modos de ejecución de un comando: en primer plano o síncrono (en inglés, *foreground*) y en segundo plano o asíncrono (en inglés *background*). Un proceso funcionando en primer plano bloquea el terminal durante su ejecución. Un proceso activado en segundo plano, en cambio, no interrumpe la actividad ordinaria del terminal, que lanza la tarea e inmediatamente está dispuesto para recibir nuevas órdenes. De este modo, por ejemplo, podemos editar en una ventana aparte un fichero de texto mientras ejecutamos otros comandos en el terminal. Una vez un comando asíncrono finaliza, el intérprete informa al usuario apropiadamente.

Ved también

La sección 1.11 explica en detalle los comandos para gestionar el estado de los procesos en ejecución.

El código de un intérprete de comandos genérico posee la siguiente estructura:

1. Esperar comandos del usuario desde el teclado
2. Recibir una petición del usuario
3. Crear un proceso (hijo) y asignarle recursos
4. (Síncrono): Esperar a que el proceso hijo termine. Volver al Paso 1
5. (Asíncrono): Volver al Paso 1 (proceso hijo ejecutándose)

Los comandos que el usuario de un terminal de UNIX puede introducir desde el teclado están compuestos habitualmente por un nombre, una serie de opciones y un conjunto de argumentos. La sintaxis genérica de estas instrucciones es la siguiente (el carácter “%” denota la entrada de comandos desde el terminal):

Figura 15. Estructura genérica de un comando UNIX

```
% COMANDO -OPCIONES ARGUMENTOS
```

Las opciones permiten al usuario configurar el comportamiento de un comando. Para activar éstas, debemos introducir un guión más una letra (-h) o dos guiones más una palabra completa (--help). La mayoría de los comandos necesitan un argumento (p.e. un valor numerico o un nombre de fichero) sobre el que realizar la acción. Algunos comandos, sin embargo, también funcionan perfectamente sin necesidad de argumentos. La Tabla 2 contiene los primeros comandos del terminal que vamos a explorar en estos materiales.

Os recomendamos que pongáis en práctica desde este momento los ejemplos incluidos a lo largo de esta asignatura para mostrar el funcionamiento del terminal.

Tabla 2. Primeros comandos del terminal

Comando	Descripción
cal	Mostrar el calendario
man	Mostrar el manual de ayuda
apropos	Buscar en el manual de ayuda
date	Mostrar la hora y la fecha
whoami	Mostrar el identificador del usuario
history	Listar los comandos ejecutados anteriormente

El comando `cal` (calendario) ejemplifica la flexibilidad de estas instrucciones (ver Figura 16). Cuando utilizamos el comando sin opciones ni argumentos, éste nos muestra el mes actual en el calendario de este año. Si añadimos la opción `-m` junto con un número entre 1 y 12 entonces veremos precisamente ese mes del año presente. Con la opción `-3`, el mismo comando ahora genera el mes anterior, el mes actual y el próximo. Por último, si usamos como argumento el valor 2017 mostramos el calendario completo del año actual.

Figura 16. Ejecutando el comando `cal`

```
% cal
    Mayo 2017
do lu ma mi ju vi sa
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

% cal -m 10
    Octubre 2017
do lu ma mi ju vi sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

% cal -3
    Abril          Mayo          Junio
do lu ma mi ju vi sa do lu ma mi ju vi sa do lu ma mi ju vi sa
          1          1  2  3  4  5  6          1  2  3
 2  3  4  5  6  7  8  7  8  9 10 11 12 13  4  5  6  7  8  9 10
 9 10 11 12 13 14 15 14 15 16 17 18 19 20 11 12 13 14 15 16 17
16 17 18 19 20 21 22 21 22 23 24 25 26 27 18 19 20 21 22 23 24
23 24 25 26 27 28 29 28 29 30 31          25 26 27 28 29 30
30

% cal 2017
    Enero          Febrero          Marzo
do lu ma mi ju vi sa do lu ma mi ju vi sa do lu ma mi ju vi sa
 1  2  3  4  5  6  7          1  2  3  4          1  2  3  4
 8  9 10 11 12 13 14  5  6  7  8  9 10 11  5  6  7  8  9 10 11
15 16 17 18 19 20 21 12 13 14 15 16 17 18 12 13 14 15 16 17 18
22 23 24 25 26 27 28 19 20 21 22 23 24 25 19 20 21 22 23 24 25
29 30 31          26 27 28          26 27 28 29 30 31
(...)
```

El caso del comando `cal` muestra cómo el usuario puede configurar un comando para adaptar su comportamiento del modo más conveniente. Para averiguar el inventario de las opciones disponibles, cualquier versión del terminal de UNIX contiene un manual del sistema. Para acceder a éste debemos hacer uso de la aplicación `man`. Dentro de la página del manual, encontraremos una breve definición del comando, la casuística de éste, su historia y varios ejemplos de uso:

Figura 17. Página del manual del sistema para el comando `cal`

```
% man cal

CAL(1)                BSD General Commands Manual                CAL(1)

NAME
  cal, ncal - displays a calendar and the date of Easter

SYNOPSIS
  cal [-3hjj] [-A number] [-B number] [[month] year]
  cal [-3hj] [-A number] [-B number] -m month [year]
  ncal [-3bhjpwysM] [-A number] [-B number] [-s country code] [[month] year]
  ncal [-3bhJeoSM] [-A number] [-B number] [year]
  ncal [-CW] [-H yyyy-mm-dd] [-d yyyy-mm]

DESCRIPTION
  The cal utility displays a simple calendar in traditional format and ncal offers an alternative layout, more options and the date of Easter. The new format is a little cramped but it makes a year fit on a 25x80 terminal. If arguments are not specified, the current month is displayed.

  The options are as follows:

  -h      Turns off highlighting of today.

  -J      Display Julian Calendar, if combined with the -o option, display date of Orthodox Easter according to the Julian Calendar.

  -e      Display date of Easter (for western churches).

  -j      Display Julian days (days one-based, numbered from January 1).

  -m month
          Display the specified month. If month is specified as a decimal number, appending 'f' or 'p' displays the same month of the following or previous year respectively.

  -o      Display date of Orthodox Easter (Greek and Russian Orthodox Churches).

  -p      Print the country codes and switching days from Julian to Gregorian Calendar as they are assumed by ncal. The country code as determined from the local environment is marked with an asterisk.

Manual page cal(1) line 1 (press h for help or q to quit)
```

Figura 17

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

El comando `apropos` permite realizar búsquedas más específicas dentro del mismo manual. Por ejemplo, podemos solicitar el conjunto de páginas del manual que contienen información sobre el término `calendar`. A continuación podemos acceder con el comando `man` a cada una de las entradas del manual:

Figura 18. Búsquedas en el interior del manual del sistema

```
% apropos calendar

cal (1)                - displays a calendar and the date of Easter
calendar (1)          - reminder service
ncal (1)              - displays a calendar and the date of Easter
```


El resto de comandos de la Tabla 2 funcionan de un modo similar. Como se muestra en la Figura 19, el comando `date` muestra por pantalla la fecha y hora del reloj del sistema en un determinado formato, actualizándose manualmente con la opción `-s`. La instrucción `whoami` muestra el nombre del usuario trabajando actualmente en el terminal.

Figura 19. Mostrando el reloj del sistema y el nombre de usuario

```
% date
lun may  1 09:24:48 CEST 2017

% whoami
eblanco
```

Finalmente, el comando `history` es especialmente útil para llevar un registro de la actividad diaria. El intérprete de comandos almacena toda la serie de instrucciones ejecutadas en el terminal. La recuperación de éstas permite volver a utilizarlas en otro momento. Dado que cada comando ejecutado anteriormente está asociado a un identificador numérico (mostrado en este listado), insertando el carácter `!` el usuario puede ejecutarlo nuevamente. Con las teclas `↑` y `↓` también es posible recorrer, paso a paso, el listado desde la propia línea de comandos para volver a ejecutar una instrucción anteriormente lanzada:

Figura 20. Historial de comandos

```
% history

371 cd
372 pwd
373 ls Desktop/
374 ls
375 ls Documents/
376 history

% !374

ls
Desktop Documents Downloads Music Pictures
Public Software Templates Videos
```

1.9. Gestión básica de ficheros

El sistema de ficheros permite organizar la información que debe ser almacenada de forma persistente. Existen varias formas de acceder a estos archivos (ver ejemplos en la Figura 14). En todo caso, el usuario, cuando accede por primera vez a un directorio, habitualmente desea obtener un listado de sus archivos. El terminal de UNIX posee un conjunto de comandos (mostrados en la Tabla 3) para gestionar tanto la circulación por el árbol de directorios del sistema como su eventual modificación.

Ved también

La organización jerárquica del sistema de ficheros de UNIX y el modo de acceso a los archivos se trata también en la sección 1.5.

Tabla 3. Comandos para gestionar archivos y directorios

Comando	Descripción
<code>pwd</code>	Mostrar el directorio de trabajo
<code>ls</code>	Listar los archivos de un directorio
<code>cd</code>	Cambiar al interior de un directorio
<code>pushd</code>	Saltar hacia un directorio
<code>popd</code>	Regresar de un directorio
<code>cp</code>	Copiar un archivo sobre otro
<code>rm</code>	Eliminar un archivo
<code>mkdir</code>	Crear un directorio
<code>rmdir</code>	Eliminar un directorio vacío
<code>chmod</code>	Cambiar los permisos de acceso

Cuando se abre un terminal el usuario se encuentra inicialmente en su directorio inicial (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 de la ruta de directorios nos encontramos, podemos usar el comando `pwd` (en inglés, *print working directory*):

Figura 21. Conocer la ruta absoluta del directorio actual

```
% pwd
/home/eblanco/Desktop
```

No hay que confundir el directorio raíz del sistema (denotado con el símbolo "/") con el directorio inicial de cada usuario (típicamente almacenado en `/home/`).

Una vez situado en un directorio, el usuario lógicamente desea ver el conjunto de archivos almacenados en su interior. Para ello, debe utilizarse el comando `ls` (en inglés, *list*). Como se muestra en la Figura 22, los elementos del directorio listado son volcados línea a línea con diferentes propiedades. Existen múltiples modos de dar formato a esta información, siendo todos ellos configurables mediante el uso de las opciones apropiadas. Las opciones de un mismo comando pueden combinarse fácilmente en una sola línea detrás del carácter "-".

Figura 22. Varios modos de listar los archivos del directorio actual

```
% ls (sin opciones, sin argumentos)
Directorio1 Directorio2 fichero1.txt imagen1.png

% ls -a (ver los elementos ocultos también)
. .. Directorio1 Directorio2 fichero1.txt imagen1.png

% ls -lah (ver atributos de forma legible)
drwxrwxr-x 4 eblanco eblanco 4,0K 2017-05-01 09:27 .
drwxrwxr-x 5 eblanco eblanco 4,0K 2017-05-01 09:27 ..
drwxrwxr-x 2 eblanco eblanco 4,0K 2017-05-01 09:27 Directorio1
drwxrwxr-x 2 eblanco eblanco 4,0K 2017-05-01 09:27 Directorio2
-rw-r--r-- 1 eblanco eblanco 983 2017-05-01 09:27 fichero1.txt
-rw-r--r-- 1 eblanco eblanco 53K 2017-05-01 09:27 imagen1.png
```

Figura 22

Para explorar el contenido del directorio actual podemos ejecutar el comando `ls` sin argumentos. Para ver el interior de un directorio en concreto debemos incluirlo como argumento. Es posible realizar la misma solicitud para múltiples directorios simultáneamente.

Para desplazarse a través del sistema jerárquico de ficheros mostrado en la Figura 6, debe utilizarse el comando `cd` (en inglés, *change directory*). Con dicho comando podemos ascender o descender por el árbol de directorios, mediante las rutas relativas o absolutas de los archivos. Existen varias combinaciones para cambiar de directorio (ver Figura 23). La combinación entre los comandos `cd`, `ls` y `pwd` permite al usuario un control completo de la navegación por su entorno de trabajo.

Figura 23. Usando el comando `cd`

<code>% cd</code>	Volver al directorio inicial del usuario
<code>% cd ~</code>	Volver al directorio inicial del usuario
<code>% cd ..</code>	Ascender al directorio anterior
<code>% cd .</code>	Permanecer en el directorio actual
<code>% cd Dir1</code>	Descender dentro del subdirectorio <code>Dir1</code>
<code>% cd Dir1/Dir2</code>	Descender dentro del subdirectorio <code>Dir2</code> en <code>Dir1</code>

Algunas versiones del intérprete de comandos poseen una instrucción que permite saltar dentro del árbol de directorios, con la particularidad de que el sistema memoriza el punto de salto original para regresar directamente con posterioridad. Las instrucciones `pushd` y `popd` (en inglés, apilar/desapilar un directorio) gestionan este modo de acceso, cuando el usuario debe realizar un acceso puntual a una región más distante del sistema de ficheros:

Figura 24. Saltos puntuales a directorios

```
% pwd
/home/eblanco

% pushd /usr/local/
/usr/local ~

% pwd
/usr/local

% ls
bin etc games include lib man sbin share src

% popd
~

% pwd
/home/eblanco
```

El intérprete de comandos proporciona al usuario varias herramientas adicionales para localizar los archivos rápidamente. Por ejemplo, si introducimos los caracteres iniciales de un nombre de fichero dentro de una línea de comandos y pulsamos la tecla Tabulador (TAB), el terminal completa el nombre de éste (el fichero debe existir en el directorio de trabajo actual). En caso de conflicto entre varios nombres posibles, nos mostrará un listado completo de alternativas.

El usuario puede emplear metacaracteres (en inglés, *wildcards*) para referirse a grupos de ficheros. El símbolo "*", por ejemplo, es un comodín que viene a significar cualquier patrón de varios caracteres. Si introducimos en un comando la expresión *.txt estaremos identificando los ficheros de texto de un directorio. El símbolo "?" es otro metacarácter que en este caso representa a un único carácter cualquiera. Por tanto, la expresión ?.txt será traducida como los ficheros de texto cuyo nombre posee un sólo carácter antes de esa extensión. Combinando ambos mecanismos con cualquier patrón de texto, agilizaremos las búsquedas en la línea de comandos (p.e. A*.txt para los ficheros cuyo nombre comienza por esa letra o A?.txt para casos como A1.txt, A2.txt, etc.).

El propio usuario es el principal responsable de gestionar su parte del sistema de ficheros. Para modificar esta organización dispone de los comandos de copia (cp), movimiento o cambio de nombre (mv) y borrado de ficheros o directorios (rm y rmdir). Su sintaxis es sencilla, introduciéndose dos argumentos cuando se realiza una copia (origen y destino) y uno sólo si se trata de una eliminación. El resultado de la operación tiene efecto siempre en el contexto del directorio actual.

Figura 25. Modificar la organización de ficheros

% cp Fichero1 Fichero2	Copiar un fichero
% mv Fichero1 Fichero2	Mover un fichero
% mv Directorio1 Directorio2	Mover un directorio
% rm Fichero	Eliminar un fichero
% mkdir Directorio	Crear un directorio
% rmdir Directorio	Eliminar un directorio (vacío)

Debemos realizar varias observaciones sobre estos comandos del terminal:

1. Cuando ejecutamos las operaciones de copia/eliminación sobre un directorio completo, existen ciertas peculiaridades que debemos tener en cuenta. Para referirnos a un directorio y a los subdirectorios y archivos que contiene en su interior, debemos emplear la opción -r (recursivo).

2. El comando rm (en inglés, *remove*) puede configurarse para solicitarnos confirmación antes de efectuar la eliminación de un archivo usando la opción -i, que activa el modo interactivo. Por el contrario, puede omitirse la pregunta con la opción -f, que significa forzar el borrado del fichero.

3. La operación de movimiento de directorios produce resultados diferentes en función del contexto. Si el segundo directorio ya existe, el primer directorio se convierte en un subdirectorio del segundo. En caso contrario, se renombra simplemente el primero de los directorios.

4. La instrucción rmdir (en inglés, *remove directory*) eliminara una carpeta, siempre que ésta no posea en su interior ningún contenido en ese momento. Para eliminar un directorio junto con su contenido, independientemente de si realmente almacena o no otra información en su interior emplearemos el comando rm -r, que activa el modo recursivo.

Ved también

En realidad, los metacaracteres pertenecen a una familia de métodos de representación con patrones denominados expresiones regulares. Para una información más completa sobre expresiones regulares, consultad la sección 1.15 más adelante en este mismo capítulo.

Hay dos combinaciones de opciones sobre las que os recomendamos especial atención: (1) El comando cp -rp Directorio1 Directorio2 realiza una copia idéntica de un directorio (preservando la fecha de modificación de los archivos). (2) La combinación rm -rf Directorio elimina directamente un directorio (junto con todo su contenido), sin efectuar ninguna pregunta de seguridad.

Como hemos explicado anteriormente es posible otorgar permisos de acceso a diferentes grupos de usuarios (ver Figura 7). El comando `chmod` (en inglés, *change file mode bits*) permite realizar estas modificaciones sobre un archivo o carpeta. Estos derechos pueden codificarse empleando numeración decimal (convertidos en código binario), o bien introducirse directamente mediante los mnemónicos `u/g/o` (dominios `user`, `group`, `others`) y `r/w/x` (modos `read`, `write`, `execute`):

Figura 26. Cambio de permisos de acceso

```
% ls -lh fichero.txt
-rw-rw-r-- 1 eblanco eblanco 2,0K 2017-05-01 10:53 fichero.txt

% chmod 600 fichero.txt

% ls -lh fichero.txt
-rw----- 1 eblanco eblanco 2,0K 2017-05-01 10:53 fichero.txt

% chmod g+r fichero.txt

% ls -lh fichero.txt
-rw-r----- 1 eblanco eblanco 2,0K 2017-05-01 10:53 fichero.txt

% chmod g+w fichero.txt

% ls -lh fichero.txt
-rw-rw---- 1 eblanco eblanco 2,0K 2017-05-01 10:53 fichero.txt

% chmod g-w fichero.txt

% ls -lh fichero.txt
-rw-r----- 1 eblanco eblanco 2,0K 2017-05-01 10:53 fichero.txt

% chmod 644 fichero.txt

% ls -lh
-rw-r--r-- 1 eblanco eblanco 2,0K 2017-05-01 10:53 fichero.txt
```

1.10. Accediendo al contenido de los ficheros

Los ficheros de texto son los contenedores elementales de información. Cada grupo de símbolos viene codificado, en particular, por una serie de *bytes* que pueden visualizarse directamente desde el terminal. Existen diferentes comandos para lograrlo, cada uno adaptado para un modo de análisis concreto. Posteriormente, en caso de necesitar introducirse nuevas modificaciones, los ficheros pueden ser manipulados con un editor de texto:

Tabla 4. Comandos para acceder a los ficheros

Comando	Descripción
<code>more</code>	Visualizar un fichero por pantalla
<code>head</code>	Extraer el inicio de un fichero
<code>tail</code>	Extraer el final de un fichero
<code>wc</code>	Contar caracteres, palabras y líneas
<code>od</code>	Visualizar caracteres ocultos en ficheros

Con frecuencia el usuario desea echar un vistazo rápido al interior de un fichero de texto. El comando que realiza esa acción se denomina `more`. Como se observa en la Figura 27, con este comando volcamos por pantalla una secuencia de ADN en formato FASTA. Para avanzar/retroceder por las páginas del documento, el usuario debe utilizar la barra espaciadora (página siguiente) y la tecla "b" (página anterior). Pulsando la tecla "q" terminamos la ejecución del visor:

El comando `less` amplía la funcionalidad del comando `more`.

Figura 27. Visualizando un fichero de texto

```
% more secuencia.fa
>secuencia
TTATTATTATTATTTTTTTTTTTTTTTTTTTTGTGACGGAGTCTCGCTCTGTC
ACCCAGGCTGGAGTGCAGTGGCACAATCTCGGCTCACTGCAAGCTCCACC
TCGCAGGTTACGCCATTCTCCTCCCTCAGCCTCCCGAGTAGCTGAGTAG
CTGGGACTACAGGCGCCCCCACTACGCCCTGGCTAATTTTTTCTATTTTT
AATAGAGACAGAGTTTCACTGCATTAGCGAGGATGGTCTCGATCTCCTGA
CCTCGCATCTGCCCGCTCAGCCTCCCAATGTGCTGGGATTACAGGCGTG
AGCCACCGCGCCCGGCTTATGTATTTATTTTTTGAGACAGAGTCTCGC
TGTGTCGTCAGGCTAGAGTGTGTGGCAGCATCTCGGCTCACTGCAACCT
CCAACTCCCTGGTTCAAAGGATTCTCCAGCCTCCACCTCCCGAGTAGCTG
GGATTACAGGCGTGACACCACACCCAGCTAATTTTTGTATTTTTAGTA
--More-- (6%)
```

Figura 27

El formato FASTA está compuesto por una cabecera, cuyo primer carácter es el símbolo ">", que informa del origen biológico de la secuencia, y el resto del contenido, agrupado en líneas con el mismo número de caracteres.

Podéis encontrar mas información en la asignatura *Genómica computacional*

En el contexto de protocolos más complejos (como los que veremos en próximas secciones), resulta interesante conocer cómo extraer un fragmento de un fichero para una exploración rápida, ya sea a partir del inicio o del final de éste. Mediante los comandos `head` y `tail` (en inglés, cabeza y cola) podemos realizar esta operación, indicando al terminal cuántas líneas deseamos ver:

Figura 28. Ver el inicio y el final de un fichero

```
% head -5 secuencia.fa
>secuencia
TTATTATTATTATTTTTTTTTTTTTTTTTTTTGTGACGGAGTCTCGCTCTGTC
ACCCAGGCTGGAGTGCAGTGGCACAATCTCGGCTCACTGCAAGCTCCACC
TCGCAGGTTACGCCATTCTCCTCCCTCAGCCTCCCGAGTAGCTGAGTAG
CTGGGACTACAGGCGCCCCCACTACGCCCTGGCTAATTTTTTCTATTTTT

% tail -5 secuencia.fa
CCGGACACCAAAGGTCACAACTGCATGACCCCATCTATATGCAATATCC
GCTACAGCCAAATCCATAGGGACCAAAGCGGATTAGTGGCTGCCGGGGC
CAGAGTTACTGTTAATGAGTACCGAGGTGGCGTTTGGGATGATGAAAAAG
TTCTGACCTAGATAGTGGTATGGCTGCATAACCTAAGTGTCTTAATA
TCACCAAATTTTATACCTGA
```

El comando `wc` (en inglés, *word count*) es extraordinariamente útil. Con esta instrucción es posible obtener fácilmente la longitud de un fichero de texto en términos de número de líneas, palabras y caracteres. En el análisis bioinformático, el número de líneas coincide con el número de anotaciones biológicas. En el caso de nuestra secuencia genómica tenemos que el fichero contiene en total 170 líneas, 170 palabras (cada línea en este caso es una palabra debido a la ausencia de espacios) y 8600 símbolos:

Figura 29. Número de caracteres, palabras y líneas de un fichero

```
% wc secuencia.fa
170 170 8600 secuencia.fa
```

Los sistemas UNIX codifican el cambio de línea en un fichero de texto con el carácter oculto "\n". Los sistemas derivados de Microsoft Windows™, sin embargo, representan este salto mediante el retorno de carro "\r" y el cambio de línea "\n". Para evitar problemas de interpretación debidos a la conversión entre plataformas, es recomendable verificar la codificación interna de un fichero de texto. El comando `od` (en inglés, *octal dump*) muestra el conjunto completo de los caracteres de los ficheros. De este modo podemos comprobar que los saltos de línea están adecuadamente codificados:

Figura 30. Ver todos los caracteres ocultos de un fichero

```
% od -c secuencia.fa
0000000 > s e c u e n c i a \n T T A T T
0000020 A T T A T T A T T T T T T T T
0000040 T T T T T T T T G T G A C G G A
0000060 G T C T C G C T C T G T C \n A C
0000100 C C A G G C T G G A G T G C A G
0000120 T G G C A C A A T C T C G G C T
0000140 C A C T G C A A G C T C C A C C
0000160 \n T C G C A G G T T C A C G C C
(...)
```

Una vez sabemos como realizar una exploración preliminar de los ficheros, podemos introducir otra serie de comandos para realizar operaciones básicas sobre éstos. Además de comparar ficheros, podemos concatenarlos, empaquetarlos o comprimirlos:

Tabla 5. Comandos para acceder a los ficheros

Comando	Descripción
<code>cat</code>	Concatenar ficheros
<code>diff</code>	Comparar ficheros de texto
<code>tar</code>	Empaquetar archivos
<code>gzip</code>	Comprimir archivos y paquetes
<code>zmore</code>	Descomprimir y visualizar un fichero
<code>zcat</code>	Descomprimir y volcar un fichero

Cuando trabajemos con listados de elementos biológicos, en numerosas ocasiones necesitaremos juntar varios ficheros de texto, cada uno a continuación del anterior. Con la instrucción `cat` podemos volcar fácilmente el contenido de un número variable de archivos. El comando `cat` utilizado sobre un único fichero, simplemente vuelca por pantalla su contenido. Sin embargo, si el número de ficheros que se introducen como argumento de éste es mayor, el resultado será la concatenación de los ficheros de entrada:

Figura 31. Concatenación de archivos

```
% cat a.txt
Prueba

% cat b.txt
Correcta

% cat a.txt b.txt

Prueba
Correcta

% cat a.txt b.txt a.txt

Prueba
Correcta
Prueba
```

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).

Figura 32. Comparación de archivos

```
% diff a.txt a.txt
% diff a.txt b.txt

1c1
< Prueba
---
> Correcta
```

La compresión de datos permite reducir el espacio que ocupan éstos en el disco duro, además de acelerar la transferencia de datos por la Red. En UNIX hay dos instrucciones específicas para realizar estas tareas: `gzip` y `tar`. La compresión/descompresión de un fichero se realiza con el comando `gzip`. El resultado es un fichero más compacto con una extensión `.gz`:

Figura 33. Compresión de ficheros

```
% ls -lh secuencia.fa
-rwx----- 1 eblanco eblanco 8,4K 2017-05-01 11:57 secuencia.fa

% gzip secuencia.fa

% ls -lh secuencia.fa.gz
-rwx----- 1 eblanco eblanco 2,5K 2017-05-01 11:57 secuencia.fa.gz

% gzip -d secuencia.fa.gz

% ls -lh secuencia.fa
-rwx----- 1 eblanco eblanco 8,4K 2017-05-01 11:57 secuencia.fa
```

El volumen de información almacenada en cualquier entorno bioinformático suele ocupar varios *Terabytes*. La secuencia del genoma humano, por ejemplo, está constituida por 3,000 millones de nucleótidos (aproximadamente, 3 *Gigabytes*).

Si es necesario comprimir directorios completos, la instrucción `tar` es capaz de generar a partir de un directorio un único fichero denominado paquete (en inglés, *tarfile*) que posteriormente puede ser comprimido con `gzip`. Ambos comandos pueden combinarse en una única instrucción simplemente añadiendo la opción `-z` en el comando `tar`. El nombre del fichero resultante debe especificarse con la opción `-f`. Finalmente, para ver por pantalla el listado de ficheros y carpetas empaquetados durante este proceso debemos activar la opción `-v`:

Tabla 6. Comandos para comprimir ficheros

Comando	Descripción
<code>gzip Fichero</code>	Comprimir el archivo Fichero
<code>gzip -d Fichero</code>	Descomprimir el archivo Fichero
<code>tar -vcf Paquete.tar Directorio</code>	Empaquetar la carpeta Directorio
<code>tar -vxf Paquete.tar</code>	Desempaquetar la carpeta Directorio
<code>tar -vzcf Paquete.tar.gz Directorio</code>	Empaquetar y comprimir un directorio
<code>tar -vzxf Paquete.tar.gz</code>	Descomprimir y desempaquetar un directorio

El siguiente ejemplo muestra como empaquetar, comprimir y descomprimir un directorio que contiene tres ficheros. De esta manera, la transferencia de ficheros a través de la Red o el correo electrónico se realiza más rápidamente dado que el archivo resultante resulta notablemente de menor peso:

Figura 34. Empaquetamiento y compresión de ficheros

```
% ls -lh directorio
-rw-r--r-- 1 eblanco eblanco 8,4K 2017-05-01 14:25 secuencia1.fa
-rw-r--r-- 1 eblanco eblanco 8,4K 2017-05-01 14:27 secuencia2.fa
-rw-r--r-- 1 eblanco eblanco 8,4K 2017-05-01 14:27 secuencia3.fa

% tar -vzcf directorio.tar.gz directorio/
directorio/
directorio/secuencia2.fa
directorio/secuencia3.fa
directorio/secuencia1.fa

% ls -lh directorio.tar.gz
-rw-rw-r-- 1 eblanco eblanco 3,0K 2017-05-01 14:29 directorio.tar.gz

% tar -vzxf directorio.tar.gz
directorio/
directorio/secuencia2.fa
directorio/secuencia3.fa
directorio/secuencia1.fa

% ls -lh directorio
-rw-r--r-- 1 eblanco eblanco 8,4K 2017-05-01 14:25 secuencia1.fa
-rw-r--r-- 1 eblanco eblanco 8,4K 2017-05-01 14:27 secuencia2.fa
-rw-r--r-- 1 eblanco eblanco 8,4K 2017-05-01 14:27 secuencia3.fa
```

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

Figura 35. Visualizar ficheros de texto comprimidos

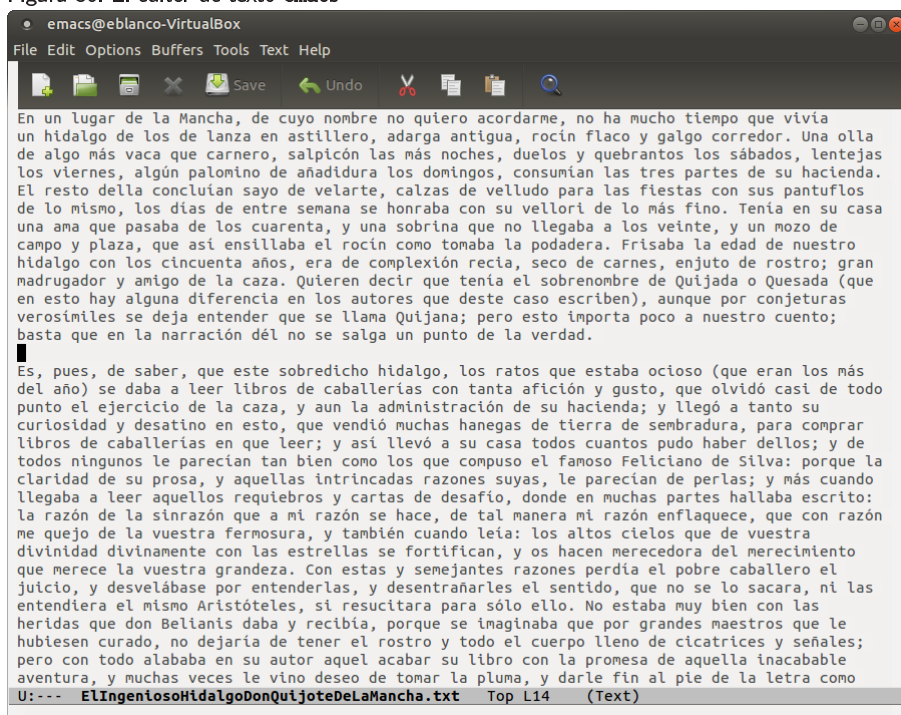
```
% gzip secuencia.fa

% zmore secuencia.fa.gz

>secuencia
TTATTATTATTATTTTTTTTTTTTTTTTTTTGTGACGGAGTCTCGCTCTGTC
ACCCAGGCTGGAGTGCAGTGGCACAATCTCGGCTCACTGCAAGCTCCACC
TCGCAGGTTACGCCATTCTCCTCCCTCAGCCTCCCGAGTAGCTGAGTAG
CTGGGACTACAGGGCCCCCACTACGCCTGGCTAAATTTTTTCTATTTTT
AATAGAGACAGAGTTTCACTGCATTAGCGAGGATGGTCTCGATCTCCTGA
CCTCGCATCTGCCCGCTCAGCCTCCCAATGTGCTGGGATTACAGGCGTG
AGCCACCGCGCCCGCCTTATGTATTTATTTTTTTGAGACAGAGTCTCGC
TGTGTCGTCAGGCTAGAGTGTGTGGCAGATCTCGGCTCACTGCAACCT
CCAACCTCCCTGGTTCAAAGGATTCTCCAGCCTCCACCTCCCGAGTAGCTG
--More--
```

Como en cualquier plataforma, existen numerosos editores de texto en Linux. Entre las aplicaciones más utilizadas por los usuarios encontramos `emacs` (Figura 36). También existen lógicamente múltiples herramientas de ofimática que poseen procesadores de texto más potentes. No obstante, recomendamos preferiblemente el uso de `emacs`, dado que no introduce codificación propia en el formato interno de los archivos. La visualización en el terminal de archivos de texto grabados bajo un determinado formato propietario podría resultar problemática posteriormente, generando resultados incorrectos.

Figura 36. El editor de texto `emacs`



Lectura complementaria

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

1.11. Gestión básica de procesos

El SO planifica cuidadosamente qué proceso puede ejecutarse en la CPU durante una fracción de tiempo limitada. Durante la vida de cada proceso, éste adopta diferentes estados (ver Figura 5). Pese a ello, una planificación óptima permite crear la ilusión de que todos los procesos activos están ejecutándose simultáneamente. El usuario, desde el terminal, puede controlar el funcionamiento de los programas en ejecución en cada instante mediante diferentes comandos:

Tabla 7. Comandos para gestionar procesos

Comando	Descripción
<code>sleep</code>	Suspender el terminal
<code>ps</code>	Lista de procesos
<code>fg</code>	Ejecutar en primer plano
<code>bg</code>	Ejecutar en segundo plano
<code>jobs</code>	Lista de trabajos del usuario
<code>top</code>	Lista de procesos en tiempo real
<code>kill</code>	Parar abruptamente un proceso
<code>nice</code>	Cambiar la prioridad de ejecución

Para observar el funcionamiento de estos operadores vamos a utilizar el comando `sleep` (en inglés, dormir). Dicho comando produce una pausa en la ejecución del terminal. Aunque de forma involuntaria, hemos ejecutado en primer plano todos los comandos introducidos hasta este momento. Ello significa que el terminal ha esperado la finalización del comando solicitado anteriormente para permitir al usuario ejecutar una nueva orden. Para lanzar un proceso y no suspender inevitablemente la ejecución del terminal debemos usar el operador de ejecución en segundo plano ("`&`"). De esta forma, el terminal continúa atendiendo peticiones mientras el comando actual realiza en segundo plano su tarea habitual:

Cada proceso en ejecución dentro de un sistema UNIX posee un identificador único denominado PID (Process Identifier).

Figura 37. Introducir una pausa en el intérprete de comandos

```
% sleep n Parar durante n segundos
(el terminal permanece inactivo durante ese intervalo de tiempo)

% sleep n & Parar durante n segundos en segundo plano
(el terminal permite ejecutar nuevos comandos)
```

Cuando ejecutamos el segundo comando, el terminal no suspende la ejecución de su programa convencional. Mediante un nuevo comando denominado `ps` (en inglés, *process status*), podemos listar los procesos en ejecución asociados a este terminal. En el siguiente ejemplo, el proceso `sleep` posee el identificador 8655. Junto con éste, tenemos otros dos procesos más ejecutándose: el propio comando `ps` y el intérprete de comandos (`bash`).

Con las opciones `aux` activadas, el comando `ps` proporciona abundante información sobre cada proceso: usuario, estado, comando, etc.

Figura 38. Listado de procesos en ejecución

```
% sleep 60 &
[1] 8655
% ps
  PID TTY          TIME CMD
 7905 pts/0    00:00:01 bash
 8655 pts/0    00:00:00 sleep
 8656 pts/0    00:00:00 ps
```

Cualquier proceso ejecutado en segundo plano puede sincronizarse con el terminal, utilizando el comando `fg` (en inglés, *foreground*). Igualmente, un comando ejecutándose en primer plano puede ser replanificado en segundo plano. Primero debe pulsarse la combinación de teclas Control + Z para suspender el proceso. Posteriormente, el proceso puede planificarse en segundo plano utilizando el comando `bg` (en inglés, *background*). Con la instrucción `jobs` (en inglés, trabajos) el usuario obtiene la lista numerada de sus procesos:

Figura 39. Suspender y reactivar un proceso

```
% sleep 60
~Z
[1]+  Stopped                  sleep 60

% bg
[1]+  sleep 60 &

% jobs
[1]+  Running                  sleep 60 &
```

El usuario también puede decidir en casos más problemáticos la suspensión definitiva de la ejecución de un proceso (p.e. cuando el gasto de CPU o de memoria es excesivo). El comando `kill` permite enviar señales a los procesos. Una de ellas es la terminación (`SIGKILL`), codificada también con el número 9.

Figura 40. Terminar definitivamente un proceso

```
% ps
  PID TTY          TIME CMD
 9732 pts/1    00:00:00 bash
 9945 pts/1    00:00:00 sleep
 9946 pts/1    00:00:00 ps

% kill -SIGKILL 9945
[1]+  Killed                  sleep 60

% ps
  PID TTY          TIME CMD
 9732 pts/1    00:00:00 bash
 9949 pts/1    00:00:00 ps
```

Por otro lado, el comando `top` proporciona un listado en tiempo real de procesos:

Figura 41. Listado de los procesos del sistema

```
% top
top - 18:01:01 up 1:38, 9 users, load average: 0.20, 0.25, 0.21
Tasks: 124 total, 1 running, 123 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.0%us, 0.7%sy, 0.0%ni, 97.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 505656k total, 491544k used, 14112k free, 26528k buffers
Swap: 126412k total, 24556k used, 101856k free, 165248k cached
  PID USER   PR  NI  VIRT  RES  SHR  S %CPU %MEM  TIME+  COMMAND
 7225 root    20   0  333m 20m 8568 S  2.0  4.2  4:10.61 Xorg
 7768 eblanco 20   0 23444 10m 8188 S  0.7  2.1  0:01.72 trashapplet
 7901 eblanco 20   0 98.7m 25m 12m S  0.7  5.1  0:25.62 gnome-terminal
10339 eblanco 20   0 2416 1148 876 R  0.7  0.2  0:00.04 top
     1 root    20   0 3056 1896 576 S  0.0  0.4  0:03.14 init
     2 root    15  -5   0    0    0 S  0.0  0.0  0:00.00 kthreadd
(...)
```

Figura 41

En el siguiente listado se observan los procesos de un usuario identificado como `eblanco`. A su vez, los procesos de sistema son ejecutados por el usuario `root`. Obsérvese cómo el proceso `init` (arranque del sistema) posee el PID 1.

La sobrecarga de un ordenador produce retrasos en la ejecución de los procesos y en el sistema en general. Además de suspender procesos (temporal o definitivamente), éstos pueden replanificarse. Cada proceso posee una prioridad de ejecución en función de distintos parámetros. Para equilibrar esta prioridad cuando sea necesario, podemos usar la instrucción `nice`.

1.12. Buscar, ordenar y asociar ficheros

Cuando trabajamos con datos bioinformáticos observamos que la mayoría de los ficheros de texto están organizados de forma tabular, es decir, con la información distribuida en una matriz de filas y columnas delimitadas por espacios o caracteres tabuladores. De este modo cada línea codifica la información relativa a un registro mientras que cada columna almacenaría, por tanto, los valores concretos de los atributos que lo caracterizan. Esta manera de definir una estructura de campos en el interior de un fichero favorece la posterior realización de determinados cálculos sistemáticos sobre su contenido.

Figura 42. Estructura genérica de Un fichero tabulado

```
registro1 atributo1 atributo2 ... atributoN
registro2 atributo1 atributo2 ... atributoN
...
registroM atributo1 atributo2 ... atributoN
```

Generalmente, realizaremos varios tipos de operaciones sobre los ficheros tabulados adoptando como unidad elemental de trabajo la línea o registro (ver Tabla 8). Bajo este paradigma, podremos buscar y apartar del fichero aquellas líneas que poseen un determinado patrón de texto, alterar el orden de las líneas en función de los valores de alguno de los atributos o filtrar registros que aparecen en más de una ocasión. En determinadas circunstancias, incluso, podremos identificar aquellos registros de dos ficheros de texto distintos que poseen el mismo valor para un determinado atributo.

Es altamente recomendable no incluir caracteres acentuados en los ficheros de texto para evitar problemas de visualización e interpretación en sistemas con alfabetos diferentes.

En el contexto del análisis bioinformático, estas operaciones aplicadas sobre los ficheros de anotaciones nos permitirán llevar a cabo con suma facilidad los contajes de diferentes características biológicas tales como los genes codificados en el interior de los genomas.

Tabla 8. Comandos para analizar ficheros de texto

Comando	Descripción
grep	Buscar los registros que poseen un patrón de texto
sort	Ordenar los registros en función de un atributo
uniq	Eliminar los registros duplicados
join	Combinar dos ficheros mediante un atributo común

Por ejemplo, mostramos un fragmento de un fichero de anotaciones génicas del genoma del ratón en la Figura 43. Cada fila de este fichero representa a un gen y cada columna representa el valor de un determinado atributo (nombre, cromosoma donde está ubicado, orientación y coordenadas dentro de éste). Disponemos, en consecuencia, de cuatro genes en nuestro fichero:

Figura 43. El fichero genes.txt de anotaciones genómicas estructurado por columnas

Klf4	chr4	-	55540008	55545347
Nanog	chr6	+	122657506	122664651
Pou5f1	chr17	+	35642976	35647722
Sox2	chr3	+	34548926	34551382

La función `grep` realiza la búsqueda de una palabra concreta, línea a línea, dentro de un fichero de texto. Es posible introducir una expresión regular en lugar de una palabra para caracterizar con más precisión el patrón de búsqueda. Como resultado, el comando vuelca por pantalla precisamente aquellas líneas que sí contienen dicho patrón. La opción `-v` realiza la operación complementaria, siendo muy útil para descartar líneas que contienen un patrón particular. Si deseamos tratar indistintamente mayúsculas y minúsculas debemos activar la opción `-i`. Para incluir en la salida el número de la línea donde se han encontrado las coincidencias podemos emplear la opción `-n`:

Ved también

Para una información más completa sobre expresiones regulares, consultad la sección 1.15 más adelante en este mismo capítulo.

Figura 44. Búsqueda de patrones en un fichero de texto

```
% grep Nanog genes.txt
Nanog chr6 + 122657506 122664651

% grep -v Nanog genes.txt
Klf4 chr4 - 55540008 55545347
Pou5f1 chr17 + 35642976 35647722
Sox2 chr3 + 34548926 34551382

% grep -i nanog genes.txt
Nanog chr6 + 122657506 122664651

% grep -n Nanog genes.txt
2:Nanog chr6 + 122657506 122664651
```

El comando `sort` (en inglés, ordenar) permite ordenar las líneas de un fichero utilizando como atributo de ordenación alguna de las columnas de éste. Por defecto, el orden de los resultados en las clasificaciones será ascendente. Para realizar ordenaciones descendentes debemos activar la opción `-r`. La opción `-n` indica al comando `sort` que debe interpretar numéricamente los valores del atributo especificado. Por defecto, la ordenación se realizará por la primera columna del fichero pero podemos especificar otra columna con la opción `-k`. También es posible realizar ordenaciones usando dos o más campos simultáneamente con esa misma opción:

La ordenación numérica no siempre se corresponde con la ordenación alfanumérica. Así, la serie ordenada de números (1,2,7,10), se organiza como (1,10,2,7) cuando se consideran los números como caracteres literales.

Figura 45. Ordenar un fichero de texto según varios criterios

```
% sort genes.txt
Klf4   chr4   -   55540008   55545347
Nanog  chr6   +   122657506  122664651
Pou5f1 chr17  +   35642976  35647722
Sox2   chr3   +   34548926  34551382

% sort -r genes.txt
Sox2   chr3   +   34548926  34551382
Pou5f1 chr17  +   35642976  35647722
Nanog  chr6   +   122657506  122664651
Klf4   chr4   -   55540008  55545347

% sort -k 3 genes.txt
Nanog  chr6   +   122657506  122664651
Sox2   chr3   +   34548926  34551382
Pou5f1 chr17  +   35642976  35647722
Klf4   chr4   -   55540008  55545347

% sort -k 4n genes.txt
Sox2   chr3   +   34548926  34551382
Pou5f1 chr17  +   35642976  35647722
Klf4   chr4   -   55540008  55545347
Nanog  chr6   +   122657506  122664651
```

Cuando trabajamos con dos ficheros, podemos identificar los registros comunes de ambos listados y completar la información contenida en el primer fichero con los nuevos atributos almacenados en el segundo. Supongamos que disponemos de un segundo archivo de anotaciones, que informa sobre el número de transcritos alternativos conocidos para algunos de los genes mostrados en la Figura 43. Para completar la primera anotación de los genes con el número de transcritos por gen registrado en la Figura 46 deberíamos comparar línea a línea ambos ficheros, seleccionando aquellos registros que poseen el mismo nombre de gen. El resultado debería contener, entonces, las líneas con el nombre del registro común junto con sus atributos rescatados de cada fichero comparado.

Figura 46. El fichero `transcritos.txt` de anotaciones génicas adicionales

```
Klf4      1
Nanog     4
Pax6      5
Pou5f1    2
```

El comando `join` (en inglés, unión) permite retener los registros comunes de dos ficheros de texto. Dado que en algunos casos será necesario obtener precisamente aquellos registros presentes exclusivamente en uno de los dos listados, podemos emplear las opciones `-v 1` y `-v 2` para separar estos casos correctamente. Por defecto, el comando emplea la primera columna de cada fichero para llevar a cabo esta operación, pero el usuario puede especificar con las opciones `-1` y `-2` qué columna de cada fichero debe ser utilizada como atributo compartido. Para una asociación correcta, los ficheros de texto deben ordenarse previamente con el comando `sort`, utilizando precisamente el atributo común como campo de ordenación.

Figura 47. Combinar dos ficheros de texto para generar un listado más completo

```
% join genes.txt transcritos.txt

Klf4  chr4  -  55540008  55545347  1
Nanog chr6  + 122657506 122664651  4
Pou5f1 chr17 + 35642976  35647722  2

% join -v 1 genes.txt transcritos.txt

Sox2 chr3 + 34548926 34551382

% join -v 2 genes.txt transcritos.txt

Pax6 5

% join -1 1 -2 1 genes.txt transcritos.txt

Klf4  chr4  -  55540008  55545347  1
Nanog chr6  + 122657506 122664651  4
Pou5f1 chr17 + 35642976  35647722  2
```

1.13. Combinación de comandos

Hasta ahora habéis visto numerosos ejemplos de comandos ejecutados individualmente en el terminal. Sin embargo, el gran potencial del intérprete de comandos permite implementar de un modo relativamente sencillo protocolos de trabajo más sofisticados. La canalización de los resultados de los comandos es un caso paradigmático. En general, la mayoría de comandos ejecutados en Linux generan más información de la que físicamente puede aparecer en pantalla. Para su posterior análisis, es preferible entonces almacenar los resultados dentro de un fichero que podemos visualizar después. Es posible que nos pueda interesar, por el contrario, convertir el flujo de datos generado por un proceso emisor de información en la entrada de otro, sin necesidad de generar ficheros intermedios. Para implementar todas estas operaciones relacionadas con la comunicación y el almacén de resultados, el intérprete de comandos proporciona una serie de funcionalidades mostradas en la Tabla 9.

Tabla 9. Comandos para combinar procesos

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

La operación de guardado de los datos generados por un proceso dentro un fichero recibe el nombre de redirección. Un proceso posee derechos de escritura sobre dos tipos de canales del terminal (salida y error). Los datos generados de forma convencional por el proceso son transmitidos por el canal de salida, mientras que los errores de ejecución son comunicados a través del canal de error. Por defecto, ambos canales se dirigen a la pantalla (terminal). Cuando el usuario desea desviar una de estas dos salidas debe indicarlo explícitamente al final del comando. El carácter ">" denota la redirección del canal de salida. Para redirigir el canal de error debemos emplear la construcción "2>". En ambos casos, los datos son depositados en un fichero, que puede ser consultado en cualquier momento sin necesidad de ejecutar nuevamente el comando.

El siguiente ejemplo ilustra cómo jugar con las redirecciones para almacenar los resultados y mensajes de error/información generados por cualquier programa. Observad cómo, al redireccionar los canales estándar, no volcamos información alguna directamente sobre el terminal:

Figura 48. Redirección de canales

```
% ls
Desktop Documents Music Pictures
Public Software Templates Videos

% ls > salida.txt
% cat salida.txt
Desktop
Documents
Music
Pictures
Public
salida.txt
Software
Templates
Videos

% ls noexiste.txt
ls: cannot access noexiste.txt: No such file or directory

% ls noexiste.txt 2> errores.txt
% cat errores.txt
ls: cannot access noexiste.txt: No such file or directory
```

La misma sintaxis empleada en la redirección de canales es útil también para crear archivos de texto al vuelo desde el teclado con el comando `cat`:

Figura 49. Creando un fichero al vuelo

```
% cat > fichero.txt

Ahora estoy tecleando el contenido
del nuevo fichero. Para acabar, debo
pulsar Control+D, que representa el
final del fichero.

% cat fichero.txt

Ahora estoy tecleando el contenido
del nuevo fichero. Para acabar, debo
pulsar Control+D, que representa el
final del fichero.
```

Siempre que se redirige la salida de un proceso sobre un fichero, el contenido de éste es sustituido por la nueva información. Utilizando los símbolos ">>", los datos se añaden al final del contenido existente (en inglés, *append*). Del mismo modo que el terminal recibe los comandos introducidos por el usuario desde el teclado, todos los procesos tienen abierto el acceso a un canal propio de entrada de datos. El usuario debe introducir el símbolo "<" para desviar explícitamente este canal de forma que el proceso pueda leer la información desde un fichero.

En ciertas situaciones creamos un fichero de texto con los datos producidos por un proceso únicamente para que éste sea leído por otro ejecutado después. Para gestionar este movimiento temporal de información que alimenta la comunicación entre procesos, UNIX implementa un sistema de transmisión de datos denominado tubería (en inglés, *pipe*). Denotadas por el usuario con el carácter "|", las tuberías permiten escribir combinaciones de instrucciones en una sola línea de comandos, evitando la creación de ficheros auxiliares en pasos intermedios. Para un hipotético productor de información denominado *proceso*, se muestran a continuación algunas de las estructuras más comúnmente utilizadas:

Pipelines

La encadenación de comandos mediante *pipes* está en el origen del término *pipelines* o protocolos basados en la aplicación secuencial de varios comandos.

Figura 50. Combinaciones típicas de dos comandos

```
% proceso | more
Ver pantalla a pantalla los resultados

% proceso | grep patron
Buscar un patrón en los resultados

% proceso | sort
Ordenar los resultados

% proceso | wc
Contar el número de resultados
```

Es importante subrayar que no hay límite en el número de comandos que pueden encadenarse. Lógicamente, podemos guardar el resultado de cualquier secuencia de comandos conectados mediante tuberías en un fichero si redireccionamos el canal de salida apropiadamente. A continuación mostramos varios patrones de combinaciones habitualmente empleados en la búsqueda, ordenación y asociación de dos o más ficheros de texto:

Figura 51. Combinaciones típicas de más de dos comandos

```
% proceso | grep patron | more
```

Buscar un patrón en los resultados, ver pantalla a pantalla

```
% proceso | grep patron | wc
```

Contar cuántas ocurrencias de un patron en los resultados

```
% proceso | grep patron | sort | uniq | more
```

Buscar ocurrencias, ordenar, filtrar duplicados y ver pantalla a pantalla

```
% proceso | sort | join - fichero2.txt
```

Asociar resultados con otro fichero

```
% proceso | sort | join - fichero2.txt | join - fichero3.txt
```

Asociar resultados con otros dos ficheros

Como se observa en los dos últimos casos, hay ciertos cambios en la sintaxis de algunos comandos cuando se utilizan las tuberías para transmitir la información. Dado que no existen ficheros auxiliares creados durante cada parte del *pipeline*, debemos 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 del siguiente comando `join` en la cadena de instrucciones.

1.14. El lenguaje de procesamiento de archivos GAWK

Hasta este momento hemos visto diferentes comandos para acceder a todo el contenido de los ficheros de texto organizados en filas y columnas de forma tabular. Sin embargo, en ocasiones es necesario realizar accesos individuales a determinados campos del fichero o introducir pequeños cálculos que se aplicaran únicamente sobre algunos de los registros. En otras palabras, con el terminal también necesitamos con frecuencia evaluar condiciones y calcular promedios sobre alguno de los atributos de los registros. GAWK (o AWK) es un lenguaje de prototipaje (en inglés, *scripting*) que permite implementar estas operaciones dado que posee un juego de instrucciones especialmente adaptado al reconocimiento de los campos de las líneas que componen los ficheros.

Lectura complementaria

Alfred V. Aho, Brian W. Kernighan and Peter J. Weinberger (1988). *The AWK Programming Language*. Addison Wesley. ISBN: 020107981X.

GAWK es una herramienta que posee la capacidad de ejecutar un bloque de instrucciones de código sobre un subconjunto de los atributos individuales de cada registro almacenado en las líneas de un fichero de texto. A partir del procesamiento de los datos de los registros, es posible generar una nueva línea de resultados, que será mostrada por pantalla o guardada posteriormente en un nuevo fichero de texto.

GAWK recorre el contenido de un fichero de texto línea a línea, separando automáticamente los distintos componentes de éstas. Para realizar los cálculos pertinentes, el usuario posee una serie de variables predefinidas que proporcionan un acceso selectivo a la información (ver Tabla 10). Podemos acceder individualmente a cada columna de una línea a través de su posición. De este modo, durante el análisis, el primer atributo de la línea en curso se guarda en la variable \$1, el segundo en \$2, etc.. Para referirse a la línea completa (incluyendo a todas las columnas de ésta) debe utilizarse la variable \$0. Mientras GAWK avanza por el fichero, la variable NR (en inglés, *Number of Records*) registra en todo momento el número de líneas procesadas. A su vez, la variable NF (en inglés, *Number of Fields*) informa del número de columnas reconocidas en la línea actual.

Para dividir una línea, GAWK utiliza un determinado carácter separador de campos (p.e. el espacio en blanco). El usuario puede cambiar este valor modificando la variable FS (en inglés, *Field Separator*). A la hora de imprimir en pantalla los resultados, el usuario puede emplear los mismos identificadores de columna para referirse a éstas (p.e. \$1, \$2, ...). Para redefinir el carácter de separación entre las columnas de la salida debemos modificar la variable OFS (en inglés, *Output Field Separator*). A nivel de registro, para distinguir las líneas durante la lectura podemos configurar el carácter separador de línea con la variable RS (en inglés, *Record Separator*) y para separar los registros en la salida a la hora de realizar la escritura de información, también podemos configurar la variable ORS (en inglés, *Output Record Separator*).

Tabla 10. Variables especiales en GAWK

Variable	Descripción
\$0	La línea actual completa
\$1	El primer valor de la línea actual
\$n	El enésimo valor de la línea actual
\$NF	El último valor de la línea actual
NF	Número de campos en la línea actual
NR	Número de líneas procesadas
FS	Carácter separador de campos (lectura)
RS	Carácter separador de líneas (lectura)
OFS	Carácter separador de campos (escritura)
ORS	Carácter separador de líneas (escritura)

AWK

El acrónimo está formado a partir de las iniciales de sus creadores: Aho, Weinberger, y Kernighan. La versión con licencia GNU se conoce como GAWK.

Por defecto, el separador de campos es el espacio en blanco y el separador de registros es el salto de línea.

Es habitual estructurar el código de un comando GAWK en tres bloques:

1. Instrucciones antes de la lectura del fichero. Se denota con la palabra clave `BEGIN`. Establece los caracteres que van a ser empleados como delimitadores e inicializa variables.
2. Instrucciones durante la lectura del fichero. Son el cuerpo del programa, ejecutado línea a línea sobre determinados atributos de cada registro del fichero.
3. Instrucciones después de la lectura del fichero. Se denota con la palabra clave `END`. Realiza cálculos con datos acumulados durante la lectura del proceso y muestra su resultado.

Formalmente, la sintaxis de una línea de comandos de GAWK es:

Figura 52. Estructura de un programa en GAWK

```
% gawk 'BEGIN{INSTRUCCIONES}
{INSTRUCCIONES}
END{INSTRUCCIONES}' fichero.txt
```

Las instrucciones deben encerrarse entre los símbolos "{" y "}" en todos los casos. Los comandos a realizar antes y después de leer el fichero son opcionales. La sintaxis más común de una ejecución en la línea de comandos es la siguiente:

Figura 53. Estructura compacta de un programa en GAWK

```
% gawk '{INSTRUCCIONES}' fichero.txt
```

Además de las variables predefinidas, cuyo valor nos proporciona automáticamente GAWK, el usuario dispone de sus propias variables para realizar cálculos. Empleando simples construcciones condicionales, podemos interrogar sobre el contenido de cada línea para mostrar después únicamente la información deseada. La Tabla 11 contiene un esquema de las instrucciones más elementales que podemos incluir en nuestros programas.

Tabla 11. Ejemplos de operaciones comunes en GAWK

Comando	Descripción
<code>i=0;</code>	Asignar un valor
<code>a[i]=0;</code>	Guardar un valor en una tabla
<code>i++;</code>	Incrementar un contador
<code>print i;</code>	Mostrar una variable por pantalla
<code>print NR;</code>	Mostrar el número de líneas procesadas
<code>print \$1;</code>	Mostrar la primera columna de la línea actual
<code>print \$1,\$2;</code>	Mostrar la primera y segunda columnas

GAWK, como cualquier otro lenguaje de prototipado (p.e. Perl o Python), posee un juego de instrucciones que permite implementar cualquier esquema algorítmico. No obstante, en el contexto de estos materiales hemos preferido aproximarnos únicamente a las instrucciones básicas para analizar ficheros de texto.

Ved también

Para profundizar en los conceptos básicos de la algorítmica os recomendamos la asignatura *Fundamentos de la Programación*

Para construir preguntas que deberán ser evaluadas sobre los campos de la línea en curso emplearemos la instrucción `if`. Según el tipo de comparación emplearemos una de las siguientes expresiones, mostradas a continuación sobre el primer atributo de las líneas:

Tabla 12. Ejemplos de instrucciones condicionales en GAWK

Comando	Descripción
<code>if (\$1 > 0) print \$0;</code>	Si la primera columna es positiva
<code>if (\$1 < 0) print \$0;</code>	Si la primera es negativa
<code>if (\$1 >= 0) print \$0;</code>	Si la primera es mayor o igual a cero
<code>if (\$1 <= 0) print \$0;</code>	Si la primera es menor o igual a cero
<code>if (\$1 == 0) print \$0;</code>	Si la primera es cero
<code>if (\$1 != 0) print \$0;</code>	Si la primera no es cero
<code>if (CONDICION1) && (CONDICION2)</code>	Si se cumplen las dos condiciones
<code>if (CONDICION1) (CONDICION2)</code>	Si se cumple una de las dos condiciones
<code>if !(CONDICION)</code>	Si no se cumple la condición

A modo de sencillo ejemplo, tomemos como modelo el siguiente archivo de texto estructurado en columnas, para mostrar la utilidad de los comandos de GAWK a la hora de acceder y procesar sus registros:

Figura 54. Censo de población

Ana	17	Estudiante	Mujer
Enrique	33	Investigador	Hombre
Francisco	40	Mecanico	Hombre
Maria	25	Estudiante	Mujer
Natalia	65	Jubilada	Mujer

Ahora veamos como manipular los diferentes campos de nuestro fichero. En primer lugar podemos implementar un simple programa que muestra por pantalla el contenido de éste. En el segundo caso observamos que para imprimir más de una variable es suficiente con concatenar sus nombres entre comas:

Figura 55. Mostrando un fichero con GAWK

```
% gawk '{print $0;}' fichero.txt

Ana      17  Estudiante  Mujer
Enrique  33  Investigador  Hombre
Francisco 40  Mecanico    Hombre
Maria    25  Estudiante  Mujer
Natalia  65  Jubilada    Mujer

% gawk '{print NR,$0;}' fichero.txt

1 Ana      17  Estudiante  Mujer
2 Enrique  33  Investigador  Hombre
3 Francisco 40  Mecanico    Hombre
4 Maria    25  Estudiante  Mujer
5 Natalia  65  Jubilada    Mujer
```

Podemos elegir fácilmente las columnas que deben ser mostradas, haciendo uso de las variables predefinidas. A continuación, únicamente volcamos por pantalla el nombre y la profesión. Mediante la variable OFS, los valores pueden presentarse de forma más legible empleando caracteres tabuladores para separar los campos:

Figura 56. Seleccionando campos de un fichero con GAWK

```
% gawk '{print $1,$3;}' fichero.txt

Ana Estudiante
Enrique Investigador
Francisco Mecanico
Maria Estudiante
Natalia Jubilada

% gawk 'BEGIN{OFS="\t"}{print $1,$3;}' fichero.txt

Ana      Estudiante
Enrique  Investigador
Francisco Mecanico
Maria    Estudiante
Natalia  Jubilada
```

Resulta sencillo calcular el promedio de los valores de una determinada columna. Por ejemplo, aquí obtenemos la media aritmética de la edad de las personas de nuestro censo. Primero, vamos sumando los valores de edad, persona a persona. Una vez recorrido todo el listado de personas, realizamos la división de la suma final de edades por el número de líneas del archivo.

Figura 57. Calculando el promedio de una columna con GAWK

```
% gawk '{edad=edad+$2;} END {print edad/NR;}' fichero.txt

34.4
```

GAWK proporciona un sistema de almacenado de valores agrupados en tablas o colecciones (en inglés, *arrays* asociativos). Para leer o escribir en su interior, debemos identificar cada elemento bien por un número o una palabra. A continuación, empleamos estas estructuras para contar el número de individuos agrupados por género y posteriormente, una vez visitadas todas las líneas del fichero, imprimimos por pantalla los totales:

Figura 58. Trabajando con colecciones de datos en GAWK

```
% gawk '{genero[$4]++;}
        END{print genero["Mujer"],"mujeres";
           print genero["Hombre"],"hombres;}' fichero.txt

3 mujeres
2 hombres
```

Para seleccionar aquellos registros que cumplen con ciertos requisitos, debemos aplicar el condicional únicamente sobre determinados atributos. Si éstos satisfacen ese requerimiento, los registros son mostrados por pantalla:

Ved también

Para profundizar en los conceptos básicos de la algorítmica os recomendamos la asignatura *Fundamentos de la Programación*

Figura 59. Seleccionando registros de un fichero con GAWK

```
% gawk '{if ($4 == "Mujer") print $0;}' fichero.txt
Ana      17  Estudiante  Mujer
Maria    25  Estudiante  Mujer
Natalia  65  Jubilada   Mujer

% gawk '{if ($2 >= 18) print $0;}' fichero.txt
Enrique  33  Investigador Hombre
Francisco 40  Mecanico    Hombre
Maria    25  Estudiante  Mujer
Natalia  65  Jubilada   Mujer
```

Figura 59

Además de preguntar específicamente sobre determinados valores, podemos introducir expresiones regulares en los condicionales. Para más información, consultad la sección 1.15 a continuación.

Estas instrucciones exhiben el mismo comportamiento que cualquier otro comando ejecutado en el intérprete del terminal. Por tanto, podemos integrar distintos comandos de GAWK en *pipelines* mediante tuberías e incluso redireccionar su salida hacia un fichero de texto, para diseñar protocolos más eficientes. A continuación, usamos los comandos `sort`, `uniq` y `wc` para regular la salida de varias instrucciones en GAWK:

Figura 60. Combinar procesos y comandos con GAWK

```
% gawk '{print $4,$2;}' fichero.txt | sort
Hombre 33
Hombre 40
Mujer 17
Mujer 25
Mujer 65

% gawk '{print $4;}' fichero.txt | sort | uniq
Hombre
Mujer

% gawk '{if ($4 == "Mujer") print $0;}' fichero.txt | wc
  3   12  108
```

1.15. expresiones regulares: aplicación en el terminal

Existen numerosas situaciones donde trabajaremos con un conjunto de varios términos relacionados (p.e. ficheros que poseen una parte de su nombre en común). En esta sección aprenderemos que no es necesario citar explícitamente todos y cada uno de estos términos para referirnos a ellos, sino que existen mecanismos para definir una única expresión genérica que permite agruparlos.

Una expresión regular es un método de representación basado en la definición de un patrón para describir un conjunto de términos.

Lectura complementaria

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

Para especificar una expresión aritmética necesitamos habitualmente emplear los operadores de suma, resta, multiplicación o división. De forma análoga, para construir una expresión regular será preciso disponer de un alfabeto de metacaracteres (en inglés, *wildcards*). Mediante el uso apropiado de estos símbolos especiales, podemos especificar el subconjunto de caracteres ordinarios que formarán el patrón de la expresión regular, la clase de combinaciones que necesitaremos realizar con ellos y el número de ocasiones en que cada uno deberá aparecer en nuestro texto de trabajo.

Lectura complementaria

Brian W. Kernighan and Rob Pike (1984). *Unix Programming Environment*. Prentice Hall. ISBN: 013937681X.

Tabla 13. Conjunto de metacaracteres genéricos en las expresiones regulares

Metacarácter	Descripción
[]	Cualquier símbolo de la lista
[!]	Cualquier símbolo que no está en la lista
[a-z]	Una letra del alfabeto
[0-9]	Un número entre 0 y 9
^	Símbolos al inicio
\$	Símbolos al final
	Disyuntiva de opciones
*	Cero o más ocurrencias de símbolos
+	Una o más ocurrencias de símbolos
{n}	Exactamente <i>n</i> ocurrencias de símbolos

Dada una expresión regular especificada por un patrón concreto y un texto formado por una serie de términos, podemos realizar dos tipos de operaciones:

- **Búsqueda:** Identificar los términos del texto que satisfacen el patrón definido por la expresión regular. Generalmente, se considera exclusivamente la primera instancia del patrón o, alternativamente, se trabaja con el conjunto completo de resultados positivos.
- **Sustitución:** Identificar los términos del texto que satisfacen el patrón definido por la expresión regular para reemplazarlos sistemáticamente con otro término diferente, posiblemente también especificado por otro patrón adicional.

Las expresiones regulares son herramientas extremadamente útiles en múltiples escenarios de análisis, dado que nos permiten ahorrar tiempo y espacio a la hora de referirnos a un conjunto, que puede ser amplio, de elementos en nuestro trabajo (p.e. textos o nombres de ficheros). Para mostrar su utilidad, introduciremos a continuación el uso de las expresiones regulares en tres tipos de comandos del terminal explicados con anterioridad: el comando `ls`, el comando `grep` y el lenguaje de procesamiento de textos `GAWK`.

El terminal posee un reducido juego de metacaracteres a la hora de construir expresiones para referirse a grupos de ficheros. El símbolo "*" significa literalmente cualquier cadena de caracteres. Combinado con un prefijo o un sufijo concretos permite identificar rápidamente los nombres de los ficheros que poseen esa serie de valores. El símbolo "?", en cambio, es útil para representar una instancia de un carácter cualquiera. Los símbolos "[" y "]" se emplean para especificar un carácter a elegir entre un rango de valores, mientras que el símbolo "!" expresa que es preciso encontrar aquello que es diferente de lo especificado a continuación.

Figura 61. Listar ficheros empleando metacaracteres

```
% ls
datos1.txt  datos2.txt  documento.pdf  imagen.png

% ls *.txt
datos1.txt  datos2.txt

% ls d*
datos1.txt  datos2.txt  documento.pdf

% ls datos?.txt
datos1.txt  datos2.txt

% ls [a-d]*
datos1.txt  datos2.txt  documento.pdf

% ls [a-d]*[0-9]*
datos1.txt  datos2.txt

% ls [!d]*
imagen.png
```

El comando `grep` permite realizar búsquedas de patrones en el interior de los ficheros de texto. En determinados casos resulta útil introducir una expresión regular para denotar un grupo de posibles resultados. El ejemplo más habitual suele ser la identificación de líneas que poseen en su inicio o bien en su final un determinado carácter o grupo de caracteres. A continuación mostramos varios ejemplos de búsquedas con el comando `grep` sobre el fichero `genes.txt`, previamente mostrado en la Figura 43.

Figura 64. Buscar patrones con expresiones regulares

```
% grep ^P genes.txt
Pou5f1 chr17 + 35642976 35647722

% grep 2$ genes.txt
Pou5f1 chr17 + 35642976 35647722
Sox2 chr3 + 34548926 34551382

% grep chr17.*35 genes.txt
Pou5f1 chr17 + 35642976 35647722
```

Lectura complementaria

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

Anteriormente, en la Figura 59, hemos explorado varias formas de utilizar GAWK para preguntar por el contenido de las líneas de un fichero de texto. Este lenguaje permite también la especificación con expresiones regulares de los patrones de búsqueda, con el objetivo de filtrar aquellas líneas que no sean interesantes para el análisis. En el interior de una composición condicional `if` debemos utilizar el símbolo `~` para indicarle a GAWK que vamos a preguntar con una expresión regular sobre el contenido de una variable. Posteriormente, para denotar la expresión regular debemos especificarlo explícitamente encerrándolo entre dos símbolos `/`. Mostramos a continuación, varios ejemplos de preguntas que podemos formular a través de este nuevo mecanismo:

Lectura complementaria

Alfred V. Aho, Brian W. Kernighan and Peter J. Weinberger (1988). *The AWK Programming Language*. Addison Wesley. ISBN: 020107981X.

Figura 64. Seleccionando registros empleando expresiones regulares con GAWK

```
% gawk '{if ($0 ~ /Estudiante/) print $0;}' fichero.txt
Ana      17  Estudiante  Mujer
Maria    25  Estudiante  Mujer

% gawk '{if ($0 !~ /Estudiante/) print $0;}' fichero.txt
Enrique  33  Investigador Hombre
Francisco 40  Mecanico    Hombre
Natalia  57  Jubilada    Mujer

% gawk '{if ($1 ~ /^[M|N]/) print $0;}' fichero.txt
Maria    25  Estudiante  Mujer
Natalia  57  Jubilada    Mujer

% gawk '{if ($3 ~ /e$/) print $0;}' fichero.txt
Ana      17  Estudiante  Mujer
Maria    25  Estudiante  Mujer
```

1.16. Definición de nuevos comandos

Como hemos visto a lo largo de este capítulo, los distintos comandos del terminal poseen un amplio número de opciones que permiten múltiples combinaciones. Lógicamente, cada usuario habitualmente aprovecha un pequeño conjunto de las posibilidades existentes de configuración. Por tanto, resulta lógico que el intérprete de comandos permita definir nuevos comandos en base a determinadas combinaciones de comandos y opciones que el usuario necesita frecuentemente. El comando `alias` implementa este mecanismo, asociando a un nuevo nombre de comando, una sucesión de comandos y opciones prefijados.

Con el comando `alias` el usuario puede redefinir el comportamiento de cualquier otro comando presentado anteriormente. Por ejemplo, es posible modificar el formato de la salida del comando `ls` para listar los ficheros incluyendo sus atributos. También es habitual modificar el funcionamiento de los comandos `cp` y `rm` para prevenir errores en forma de sobrescritura de ficheros o borrado accidental de éstos. Si tecleamos el comando `alias` sin argumentos, el terminal nos mostrará el listado de definiciones activo en ese momento (ver Figura 64).

Figura 64. Definición de nuevos comandos

```
% alias dir='ls -lah'

% dir imagen.png

-rw-rw-r-- 1 eblanco eblanco 0 may 13 16:04 imagen.png

% alias cp='cp -i'

% cp otraimagen.png imagen.png

cp: overwrite 'imagen.png'?

% alias rm='rm -i'

% rm imagen.png

rm: remove regular file 'imagen.png'?

% alias hgrep='history | grep'

% hgrep gawk

...
1430 gawk '{print $0}' a.txt
1431 hgrep gawk

% alias

alias cp='cp -i'
alias dir='ls -lah'
alias hgrep='history | grep'
alias rm='rm -i'
```

1.17. Diseño de protocolos automáticos en el terminal

Llegados a este punto, el estudiante habrá ejecutado aisladamente una serie de comandos del terminal que permiten realizar tareas elementales de gran utilidad sobre ficheros de texto con información relevante. Como acabamos de ver, existen mecanismos relativamente rudimentarios como el comando `alias` para renombrar o extender la funcionalidad de las instrucciones básicas.

Sin embargo, el gran potencial del interfaz de línea de comandos se hace más evidente, incluso, cuando encapsulamos un grupo de comandos cuya secuencia de ejecución resuelve un problema dentro de un protocolo (en inglés, *script*). A partir de ese mismo instante, podremos invocar repetidamente este *pipeline* sobre diferentes conjuntos de ficheros o directorios que contengan la misma clase de información. Genéricamente, la familia de elementos procesados internamente por el protocolo se denominan parámetros y variarán en cada ejecución, produciendo un resultado específico en cada caso. Lógicamente, en función del entorno de trabajo y de la clase de análisis a desarrollar, podemos diseñar nuestro propio juego de protocolos de comandos para ahorrarnos una cantidad sustancial de trabajo manual durante nuestros análisis bioinformáticos.

Lectura complementaria

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

Observaremos dos grandes ventajas trabajando bajo este paradigma:

1. Ya no será preciso introducir la misma secuencia de comandos en el terminal cada vez que necesitemos resolver la misma tarea. Será suficiente con invocar al protocolo desde el terminal como cualquier otro comando.
2. La automatización de tareas garantiza que se aplique consistentemente la misma secuencia de comandos en todos los casos, disminuyendo drásticamente los errores causados por la intervención manual del usuario.

Estos protocolos o *scripts* del terminal deben especificarse dentro de un fichero de texto con la extensión `.sh`. En su interior, incluiremos una serie de comandos para ser ejecutados secuencialmente. Para aumentar su flexibilidad, los protocolos permiten referirse internamente a sus parámetros de un modo genérico, funcionando por tanto sobre cualquier grupo de argumentos de la misma clase. Cuando invoquemos al protocolo desde la línea de comandos, los parámetros genéricos especificados dentro del *script* serán sustituidos por los valores concretos suministrados por el usuario junto con el nombre del comando.

Mediante el ejemplo de la Figura 65 vamos a explicar el funcionamiento de estos protocolos. Deseamos automatizar una secuencia de comandos para realizar una primera exploración del contenido de un fichero de texto. Nuestro *script* se denomina `analisisTexto.sh`: observad que la primera línea de cualquier protocolo debe especificar la ubicación exacta del intérprete de comandos, empleando para ello los símbolos `"#!"` (en inglés, *shebang*).

Los ejemplos presentados en esta asignatura se han ejecutado sobre el intérprete de comandos bash (GNU Bourne-Again SHell). En la mayoría de sistemas coexisten otros intérpretes como sh (Bourne) o csh (C shell).

Figura 65. Código del protocolo para realizar el análisis de un fichero de texto

```
#!/bin/bash

echo "1. Iniciando la ejecucion:";
echo "PID es $$";
echo "2. Numero de argumentos introducidos:";
echo "$#";
echo "3. El nombre del fichero que vamos a analizar es:";
echo "$1";
echo "4. Tamano del fichero:";
ls -sh $1 | gawk '{print $1}';
echo "5. Numero de lineas del fichero:";
wc -l $1 | gawk '{print $1}';
echo "6. Numero de caracteres del fichero:";
wc -c $1 | gawk '{print $1}';
echo "7. Numero de palabras del fichero:";
wc -w $1 | gawk '{print $1}';
echo "8. Primeras dos lineas del fichero:";
head -2 $1;
echo "9. Ultimas dos lineas del fichero:";
tail -2 $1;
```

Para mostrar por pantalla determinados mensajes de texto que informen del progreso del protocolo podemos emplear el comando `echo`. Intercalados entre las líneas de carácter informativo, encontraremos varios comandos del terminal para solucionar cada tarea dentro del protocolo. Podemos emplear cualquier comando o *pipeline* mostrado en secciones anteriores. Dado que este *script* va a ejecutarse sobre un fichero de texto cuyo nombre deberá ser introducido al invocar al protocolo `analisisTexto.sh` desde la línea de comandos, debemos referirnos dentro de éste de forma genérica empleando la nomenclatura `$1`. Si el protocolo trabaja con más ficheros, simplemente utilizaríamos los nombres `$2` o `$3` para ejecutar comandos sobre ellos. La palabra clave `$$` siempre albergará el identificador del proceso (PID) que está ejecutando físicamente este protocolo. Si desde el interior del *script* deseamos controlar el número de argumentos introducidos desde la línea de comandos, debemos usar la palabra clave `$#`.

Antes de ejecutar el protocolo debemos otorgarle permisos de ejecución con el comando `chmod`. Una vez asignados los derechos, podemos probar nuestro protocolo sobre cualquier fichero de texto. Mostramos a continuación los resultados sobre el fichero `secuencia.fa` que apareció anteriormente en la Figura 27.

Figura 66. Ejecutando el protocolo de análisis de ficheros de texto (1)

```
% chmod u+x analisisTexto.sh

% ./analisisTexto.sh secuencia.fa

1. Iniciando la ejecucion:
PID es 3604
2. Numero de argumentos introducidos:
1
3. El nombre del fichero que vamos a analizar es:
secuencia.fa
4. Tamano del fichero:
12K
5. Numero de lineas del fichero:
170
6. Numero de caracteres del fichero:
8600
7. Numero de palabras del fichero:
170
8. Primeras dos lineas del fichero:
>secuencia
TTATTATTATTATTTTTTTTTTTTTTTTTTTGTGACGGAGTCTCGCTCTGTC
9. Ultimas dos lineas del fichero:
TTCTGACCTAGATAGTGTGATGGCTGCATAACAATAAGTGTCTTAATA
TCACCAAATTTTATACCTGA
```

Una vez hemos verificado que el comportamiento del protocolo responde a nuestras necesidades y cualquier error de sintaxis ha sido depurado, podemos proceder a ejecutarlo sobre cualquier fichero de texto. Para ello, únicamente debemos cambiar el nombre del fichero de texto a la hora de invocar a nuestro *script* desde la línea de comandos. Mostramos a continuación los resultados sobre el fichero que apareció anteriormente en la Figura 36.

Lectura complementaria

Cameron Newham (2005).
Learning the bash Shell, Third Edition. O'Reilly Media.
ISBN: 0-596-00965-8.

Figura 67. Ejecutando el protocolo de análisis de ficheros de texto (2)

```

% ./analisisTexto.sh quijote.txt

1. Iniciando la ejecucion:
PID es 3642
2. Numero de argumentos introducidos:
1
3. El nombre del fichero que vamos a analizar es:
quijote.txt
4. Tamano del fichero:
8,0K
5. Numero de lineas del fichero:
49
6. Numero de caracteres del fichero:
4241
7. Numero de palabras del fichero:
740
8. Primeras dos lineas del fichero:
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivia
un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor. Una olla
9. Ultimas dos lineas del fichero:
verdad toda aquella maquina de aquellas sonadas invenciones que leia, que para el no habia otra
historia mas cierta en el mundo.

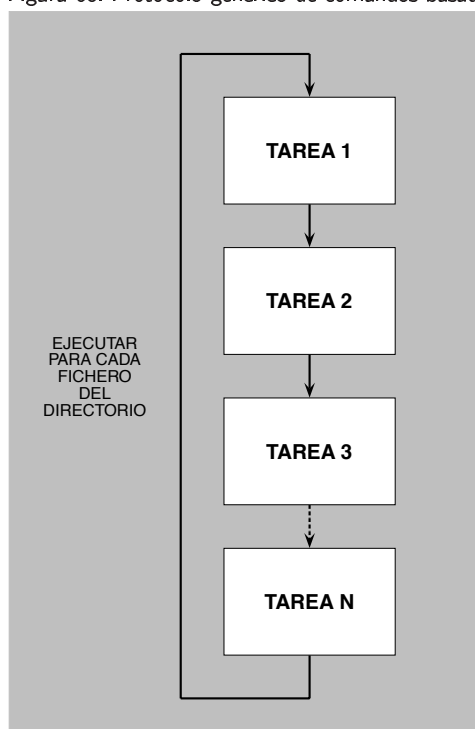
```

En función de la complejidad de nuestros análisis podemos introducir construcciones condicionales e iterativas convencionales en nuestros protocolos. Por ejemplo, sucede con bastante frecuencia que deseamos aplicar secuencialmente una misma serie de tareas o comandos a cada fichero guardado en un determinado directorio. Esquemáticamente, el flujo de datos aplicado a cada archivo de la carpeta suministrada por el usuario del terminal seguiría el orden representado en la Figura 68, donde cada tarea puede ser un comando del terminal o uno de nuestros propios *scripts*:

Ved también

Para profundizar en los conceptos básicos de la programación os recomendamos la asignatura *Fundamentos de la Programación*

Figura 68. Protocolo genérico de comandos basado en la iteración



Para ilustrar la enorme utilidad de esta aproximación, vamos a estudiar el protocolo `analisisDirectorio.sh` que analiza de forma análoga el conjunto de ficheros de texto de un directorio. Este *script* recibirá como parámetro desde la línea de comandos el nombre del directorio (Figura 69 y Figura 70). Mediante la construcción `while` y la variable `file`, que contiene en cada iteración el nombre de uno de los ficheros del directorio, podremos dirigir la ejecución del anterior *script* `analisisTexto.sh` sobre cada uno de ellos.

Figura 69. Código del protocolo para realizar el análisis de un directorio

```
#!/bin/bash

echo "1. Iniciando la ejecucion:";
echo "PID es $$";
echo "2. Numero de argumentos introducidos:";
echo "$#";
echo "3. El nombre del directorio que vamos a analizar es:";
echo "$1";
echo "4. Los ficheros que existen dentro de ese directorio son:";
ls $1;
echo "5. Trabajando iterativamente sobre cada fichero:";
ls $1 | while read file;
do
    echo "Lanzamos analisisTexto.sh $file";
    ./analisisTexto.sh $1/$file;
done
```

Observaréis que estamos ejecutando nuestros protocolos desde el directorio actual de trabajo (`./`), evitando invocar únicamente su nombre. Sin embargo, una vez estamos seguros de que nuestro *script* funciona correctamente, resulta más conveniente guardar toda nuestra colección de protocolos en un único directorio del sistema. De este modo, no deberemos mantener múltiples copias y podremos lanzarlos desde cualquier lugar de nuestro árbol de directorios.

Las plataformas UNIX poseen un conjunto de variables globales que contienen datos de carácter general. Aunque, el comando `set` muestra por pantalla los valores del inventario completo de variables, resulta más conveniente emplear el comando `echo` para extraer solamente una de ellas. En nuestro caso, la variable `PATH` contiene la serie de directorios donde el interprete de comandos busca cada comando que ejecutamos desde el terminal. El comando `which`, de hecho, explora los mismos directorios cuando solicitamos esa información explícitamente. Si deseamos añadir nuestro directorio de *scripts* a la lista de directorios almacenada en la variable `PATH` debemos emplear el comando `export`.

Ved también

Para convertir en permanentes estos cambios en la variable `PATH` debemos incluirlos en el fichero de configuración `.bash_profile`.

Tabla 14. Comandos para la ejecución de *scripts*

Comando	Descripción
<code>set</code>	Mostrar las variables del sistema
<code>echo</code>	Mostrar por pantalla
<code>which</code>	Localizar la ruta de un comando
<code>export</code>	Modificar una variable del sistema

Figura 70. Ejecutando el protocolo de análisis sobre un directorio

```
% chmod u+x analisisDirectorio.sh
```

```
% ./analisisDirectorio secuencias
```

```
1. Iniciando la ejecucion:
```

```
PID es 4514
```

```
2. Numero de argumentos introducidos:
```

```
1
```

```
3. El nombre del directorio que vamos a analizar es:
```

```
secuencias
```

```
4. Los ficheros que existen dentro de ese directorio son:
```

```
file1.fa
```

```
file2.fa
```

```
file3.fa
```

```
file4.fa
```

```
5. Trabajando iterativamente sobre cada fichero:
```

```
Lanzamos analisisTexto.sh file1.fa
```

```
1. Iniciando la ejecucion:
```

```
PID es 4519
```

```
2. Numero de argumentos introducidos:
```

```
1
```

```
3. El nombre del fichero que vamos a analizar es:
```

```
secuencias/file1.fa
```

```
4. Tamano del fichero:
```

```
4,0K
```

```
5. Numero de lineas del fichero:
```

```
20
```

```
6. Numero de caracteres del fichero:
```

```
980
```

```
7. Numero de palabras del fichero:
```

```
20
```

```
8. Primeras dos lineas del fichero:
```

```
>secuencia
```

```
TTATTATTATTATTTTTTTTTTTTTTTTTTTGTGACGGAGTCTCGCTCTGTC
```

```
9. Ultimas dos lineas del fichero:
```

```
GGGTGTCACAGTGAACCGACCTTAGGCCAGTGGGAGTCAGTCACA
```

```
CAAAGTGTGAGTCCATGACTTGGGGCTTAGCCAGCACCCACCCACG
```

```
Lanzamos analisisTexto.sh file2.fa
```

```
1. Iniciando la ejecucion:
```

```
PID es 4655
```

```
2. Numero de argumentos introducidos:
```

```
1
```

```
3. El nombre del fichero que vamos a analizar es:
```

```
secuencias/file2.fa
```

```
4. Tamano del fichero:
```

```
12K
```

```
5. Numero de lineas del fichero:
```

```
170
```

```
6. Numero de caracteres del fichero:
```

```
8600
```

```
7. Numero de palabras del fichero:
```

```
170
```

```
8. Primeras dos lineas del fichero:
```

```
>secuencia
```

```
TTATTATTATTATTTTTTTTTTTTTTTTTTTGTGACGGAGTCTCGCTCTGTC
```

```
...
```

```
Lanzamos analisisTexto.sh file3.fa
```

```
...
```

1.18. Transferencia de ficheros desde el terminal

La mayoría de entornos bioinformáticos de trabajo están basados en la filosofía de trabajo en línea de comandos. Debido a su versatilidad y eficiencia, estos sistemas permiten la instalación, gestión y mantenimiento de las herramientas necesarias para el análisis de información biológica. Por otro lado, mediante programas de transferencia y sincronización de envío de datos, estos entornos disponen de las versiones actualizadas de las anotaciones del genoma de numerosas especies. Estos datos son esenciales, no sólo para investigar un organismo en particular, sino para realizar también estudios comparativos. Una vez toda la información está almacenada localmente en nuestro ordenador de trabajo, no es preciso recurrir a posteriores conexiones a la Red que ralentizarían el análisis. Dado que los datos son accesibles desde nuestro terminal, podremos aplicar sobre ellos el abanico de distintos comandos detallados anteriormente para extraer nuevo conocimiento.

Desde nuestro terminal nos conectaremos a la Red para efectuar dos tareas:

- Conexión a una máquina remota para ejecutar trabajos en su interior.
- Descarga directa de ficheros de datos a través de las páginas *web*.

El terminal proporciona varios comandos para facilitar estas conexiones, ocultando la dificultad técnica inherente a estas actividades (ver Tabla 15). En general, para mayor seguridad en nuestras comunicaciones, es preferible emplear comandos que encripten la información cuando realizamos la transmisión de datos con otra máquina (p.e. *ssh* o *scp*).

Tabla 15. Comandos para acceder a la Red

Comando	Descripción
<i>ssh</i>	Conexión segura a una máquina remota para ejecutar comandos
<i>scp</i>	Copia segura de datos a una máquina remota
<i>sftp</i>	Copia segura de datos a una máquina remota por FTP
<i>wget</i>	Descarga de ficheros y páginas <i>web</i>

Es frecuente disponer en los entornos bioinformáticos de una infraestructura de ordenadores personales conectados en red con otro grupo de ordenadores de mayor potencia de cálculo (en inglés, *cluster*), que están dedicados a llevar a cabo tareas de análisis complejas. El comando *ssh* nos permite conectar con otra máquina desde nuestro terminal. 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*). Ambas informaciones son suministradas comúnmente por el administrador de la máquina. Los comandos *scp* y *sftp* resultan útiles para transferir datos entre nuestro ordenador y la máquina remota.

Para acceder a una página *web* con el objetivo de descargar su contenido a nuestro terminal podemos emplear el comando `wget`. Lógicamente, en este caso necesitaremos una dirección HTTP para localizar el fichero de datos en concreto que deseamos reproducir en nuestro ordenador. Por defecto, el comando `wget` es capaz de bajar directamente un archivo y copiarlo en el directorio de trabajo actual. No obstante, cuando es necesario descargar un grupo de páginas *web*, podemos emplear la opción `-r` (activar recursividad) para conservar la misma estructura organizativa en nuestro sistema de ficheros.

1.19. Ejemplo práctico 1: Analizando el genoma humano

El analista bioinformático trata de modelar estadísticamente distintos contextos de funcionamiento de las células. 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. El navegador genómico de UCSC pone gratuitamente a disposición de la comunidad científica mundial tanto las secuencias de los genomas de múltiples especies como sus respectivos catálogos de genes. Junto con estas informaciones, esta misma página *web* suministra distintos programas para analizar los datos.

El estudiante puede encontrar todos los detalles sobre el funcionamiento de los navegadores genómicos en la asignatura Genómica computacional.

Con el objetivo de aproximar al estudiante a un escenario de situación realista dentro de un entorno bioinformático, hemos estructurado este ejemplo práctico en tres tipos de actividades. Resulta fundamental para el aprovechamiento de estos materiales que intentéis reproducir cada etapa de este protocolo de análisis en vuestro propio terminal:

1. Descarga y exploración del genoma humano
2. Descarga y análisis del catálogo de genes humanos
3. Descarga de las herramientas del navegador genómico

Exploración del genoma humano

El genoma de un organismo está organizado en un conjunto de cromosomas. En este ejemplo, vamos a proceder a descargarnos los cromosomas del genoma humano, en su distribución hg38. Para ello, debemos en primer lugar acceder a la página de descargas (en inglés, *downloads*) del navegador genómico. Es posible obtener dicha página *web* ingresando primero en el portal principal del navegador para buscar la sección de descargas o saltando directamente (ver Tabla 16):

Tabla 16. Páginas *web* del navegador genómico de UCSC

Acceso	Dirección
Página principal	http://genome.ucsc.edu
Página de descargas	http://hgdownload.soe.ucsc.edu/downloads.html

El contenido de esta página nos muestra el listado de genomas disponibles organizado por especies (ver Figura 71, actualizado a 17 de Mayo de 2017):

Figura 71. Listado de genomas disponibles en UCSC

Sequence and Annotation Downloads		
VERTEBRATES - Complete annotation sets		
Human	Gorilla	Platypus
Alpaca	Green Monkey	Proboscis Monkey
American alligator	Guinea pig	Rabbit
Armadillo	Hedgehog	Rat
Atlantic cod	Horse	Rhesus
Baboon	Kangaroo rat	Rock hyrax
Bison	Lamprey	Sheep
Bonobo	Lizard	Shrew
Brown kiwi	Malayan flying lemur	Sloth
Budgerigar	Manatee	Squirrel
Bushbaby	Marmoset	Squirrel monkey
Cat	Medaka	Stickleback
Chicken	Medium ground finch	Tarsier
Chimpanzee	Megabat	Tasmanian devil
Chinese hamster	Microbat	Tenrec
Chinese pangolin	Minke whale	Tetraodon
Coelacanth	Mouse	Tibetan frog
Cow	Mouse lemur	Tree shrew
Crab-eating macaque	Naked mole-rat	Turkey
Dog	Nile tilapia	Wallaby
Dolphin	Opossum	White rhinoceros
Elephant	Orangutan	X. tropicalis
Elephant shark	Painted turtle	Zebra finch
Ferret	Panda	Zebrafish
Fugu	Pig	
Gibbon	Pika	
Golden Snub-Nosed Monkey		

Si utilizamos el enlace [Human](#) entraremos en la sección dedicada al genoma humano. La información relativa a cada genoma recibe actualizaciones con una determinada frecuencia. Es por esta razón que cada mejora sustancial posee un código de versión propio. Trabajaremos con la distribución denominada `hg38`, que es la más reciente en el momento de la redacción de estos materiales:

Figura 72. Versión hg38 del genoma humano

Human Genome
Dec. 2013 (GRCh38/hg38)
Full data set
Data set by chromosome
Annotation database
LiftOver files
Pairwise Alignments
Multiple Alignments
...

Ahora utilizaremos el enlace `Full data set` para llegar finalmente a la página *web* que contiene el listado de los ficheros en formato FASTA que constituyen el genoma humano. Todos los archivos están comprimidos y empaquetados para reducir el tiempo de transmisión.

Figura 73. Secuencias del genoma humano en la versión hg38

```
This directory contains the Dec. 2013 (GRCh38/hg38) assembly of the
human genome (hg38, GRCh38 Genome Reference Consortium Human Reference 38
(GCA_000001405.2)), as well as repeat annotations and GenBank sequences.
```

Name	Last modified	Size	Description	Parent Directory
analysisSet/	14-Apr-2014 09:51	-		
est.fa.gz	13-May-2017 18:04	1.5G		
est.fa.gz.md5	13-May-2017 18:04	44		
hg38.2bit	09-Jan-2014 14:14	797M		
hg38.agp.gz	15-Jan-2014 20:55	842K		
hg38.chrom.sizes	24-Dec-2013 21:06	11K		
hg38.chromFa.tar.gz	23-Jan-2014 17:18	938M		
hg38.chromFaMasked.tar.gz	23-Jan-2014 17:10	487M		
hg38.fa.align.gz	08-Jan-2014 23:43	2.4G		
hg38.fa.gz	15-Jan-2014 21:14	938M		
hg38.fa.masked.gz	15-Jan-2014 21:24	487M		
hg38.fa.out.gz	15-Jan-2014 20:56	172M		
hg38.trf.bed.gz	15-Jan-2014 20:56	7.9M		
md5sum.txt	03-Mar-2014 10:31	451		
mrna.fa.gz	13-May-2017 16:47	275M		
mrna.fa.gz.md5	13-May-2017 16:48	45		
refMrna.fa.gz	13-May-2017 18:06	64M		
refMrna.fa.gz.md5	13-May-2017 18:06	48		
xenoMrna.fa.gz	13-May-2017 17:13	5.8G		
xenoMrna.fa.gz.md5	13-May-2017 17:13	49		
xenoRefMrna.fa.gz	13-May-2017 18:05	234M		
xenoRefMrna.fa.gz.md5	13-May-2017 18:05	52		

Existen varios ficheros con el genoma humano almacenado en distintos formatos. El fichero `hg38.chromFa.tar.gz` contiene la secuencia original de los cromosomas separados en archivos independientes. Para descargar en nuestro directorio actual de trabajo los cromosomas humanos en dicho formato, podemos emplear el comando `wget`, que habíamos introducido anteriormente:

Figura 74. Descarga de los cromosomas humanos en el terminal

```
% wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.chromFa.tar.gz
--2017-05-17 10:24:19-- http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.chromFa.tar.gz
Resolving hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)... 128.114.119.163
Connecting to hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)|128.114.119.163|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 983726049 (938M) [application/x-gzip]
Saving to: hg38.chromFa.tar.gz

hg38.chromFa.tar.gz 100%[=====>]
938,15M 4,05MB/s in 3m 19s

2017-05-17 10:27:38 (4,72 MB/s) - hg38.chromFa.tar.gz saved [983726049/983726049]
```


Análisis de los genes humanos

El análisis del catálogo de genes de una especie representa un caso paradigmático de la enorme utilidad de todos los comandos del terminal vistos en esta sección. Un gen es un fragmento de ADN ubicado en el genoma que contiene la información precisa para sintetizar una determinada molécula biológica (una molécula de ARN o una proteína). 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 plausibles de exones, dando lugar a distintas formas alternativas de un mismo gen denominados transcritos alternativos.

Como se ha visto anteriormente, el genoma puede almacenarse en nuestro ordenador en forma de secuencias de letras, empleando un fichero por cromosoma. Para codificar la información relativa a la ubicación de los genes, en cambio, es habitual utilizar ficheros de texto tabulado. Cada línea de estos ficheros contiene el transcrito de un gen y cada columna, el valor de un atributo conocido sobre este gen. Básicamente, un transcrito de un gen posee una localización concreta, identificada por un cromosoma, una posición inicial/final y una dirección de lectura (hebra de ADN o *strand*, en inglés). Otras características que podemos recuperar sobre un transcrito son el nombre del gen al que pertenece, el número de exones o las coordenadas exactas de éstos.

En este ejercicio vamos a utilizar la anotación servida por el consorcio RefSeq para el genoma humano. Dicha información viene codificada, bajo el mismo formato para todas las especies, en un fichero denominado `refGene.txt`, incluido en cada distribución. De forma análoga a como descargamos la secuencia de los cromosomas que constituyen el genoma humano, procedemos a explorar la página *web* del navegador genómico de UCSC para obtener el fichero de genes. Partimos de la página relativa al genoma humano mostrada en la Figura 72. Si ahora seleccionamos el enlace `Annotation database` accederemos a las anotaciones del genoma, que habitualmente se representan gráficamente en forma de pista en el navegador genómico:

Figura 77. Extracto del listado de pistas del navegador para el genoma hg38

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_000001405.2)) .

Name	Last modified	Size	Description	Parent Directory
affyGnf1h.sql	11-May-2015 01:50	2.1K		
affyGnf1h.txt.gz	11-May-2015 01:50	596K		
...				
refGene.sql	08-May-2017 06:27	1.9K		
refGene.txt.gz	08-May-2017 06:28	5.3M		
...				
xenoRefSeqAli.sql	08-May-2017 06:49	2.1K		
xenoRefSeqAli.txt.gz	08-May-2017 06:49	16M		

Ved también

Para revisar los conceptos de genoma, cromosoma, gen y proteína os recomendamos la asignatura *Fundamentos de biología molecular*.

Ved también

El estudiante puede profundizar sobre el funcionamiento de los navegadores genómicos en la asignatura *Genómica computacional*.

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.

Ved también

El módulo Gestión de datos con MySQL contiene abundantes explicaciones sobre las bases de datos relacionales.

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. Para ello, nuevamente debemos utilizar el comando `wget` para transferir ambos ficheros a nuestro terminal.

Figura 78. Descarga de los genes humanos en el terminal

```
% wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.sql
--2017-05-15 08:41:36-- http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.sql
Resolving hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)... 128.114.119.163
Connecting to hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)|128.114.119.163|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1964 (1,9K) [text/plain]
Saving to: refGene.sql

refGene.sql      100%[=====]
1.964          --.-K/s   in 0s

2017-05-15 08:41:36 (215 MB/s) - refGene.sql saved [1964/1964]

% wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz
--2017-05-15 08:32:58-- http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz
Resolving hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)... 128.114.119.163
Connecting to hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)|128.114.119.163|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5540771 (5,3M) [application/x-gzip]
Saving to: refGene.txt.gz

refGene.txt.gz  100%[=====]
5.540.771    1,27MB/s  in 5,5s

2017-05-15 08:33:04 (978 KB/s) - refGene.txt.gz saved [5540771/5540771]
```

A continuación invitamos al estudiante a echar un vistazo al contenido del primer fichero: `refGene.sql`. En este capítulo no va a ser necesario utilizarlo, pero nos resultará útil para conocer que característica de los genes está anotada en cada columna del segundo fichero. Los atributos que consultaremos con mayor frecuencia serán (ver Figura 79): `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 puede pertenecer a un gen.

Figura 79. Atributos de los transcritos anotados por el consorcio RefSeq

```
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','cpl') NOT NULL,
  'cdsEndStat' enum('none','unk','incmpl','cpl') NOT NULL,
  'exonFrames' longblob NOT NULL,
  KEY 'chrom' ('chrom','bin'),
  KEY 'name' ('name'),
  KEY 'name2' ('name2')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Pasaremos ahora a visualizar el segundo fichero `refGene.txt`. Este archivo contiene los datos del catálogo completo de genes anotados en el genoma humano. En el contexto de este ejercicio, cada línea contiene información sobre un transcrito de un determinado gen. En el caso de que un gen posea varios transcritos, cada uno se codifica en líneas separadas (cada una con su propio código y sus correspondientes coordenadas). En todas las líneas, los atributos de cada transcrito están separados por el carácter tabulador o `"\t"`. En primer lugar, debemos descomprimir el fichero con el comando `gzip`.

Figura 80. El catálogo `refGene.txt` de genes humanos

```
% 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,

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, ONUBPL unk unk -1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,
```

Figura 80

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.

Debido a un exceso de información, estos ficheros tan voluminosos son poco manejables. A continuación, vamos a extraer únicamente aquellos campos que sean necesarios para nuestro ejercicio práctico con un comando GAWK. En particular, consideraremos interesantes los siguientes atributos: nombre del gen, identificador del transcrito, cromosoma, hebra, coordenadas y número de exones. Para evitar la ejecución del comando previo en cada ocasión que sean necesarios estos datos, podemos redireccionar la salida de éste hacia un fichero que denominaremos `catalogo.txt`. A continuación contaremos el número total de transcritos humanos a partir de este último fichero:

Figura 81. Número de transcritos en el catálogo de genes humanos

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

% head -5 catalogo.txt

DDX11L1    NR_046018    chr1    +    11873    14409    3
WASH7P    NR_024540    chr1    -    14361    29370    11
LINCO1204 NR_104645    chrX    +    45505387 45523644 3
ZNF107    NR_104148    chr7    +    64666082 64687830 4
NUBPL     NR_120408    chr14   +    31561384 31861223 10

% wc -l catalogo.txt

69853 catalogo.txt
```

Conforme avanzamos en el análisis, podemos introducir filtros en nuestras instrucciones para extraer solamente determinados registros. Por ejemplo, si utilizamos GAWK podemos solicitar el número de transcritos génicos distribuidos en la hebra positiva. De forma análoga, con el comando `grep` es posible realizar un conteo similar de todos los transcritos en el cromosoma 21 del genoma humano:

Figura 82. Filtros aplicados sobre el catálogo de genes humanos

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

35724

% grep chr21 catalogo.txt | wc -l

990
```

Resulta muy sencillo generar fácilmente innumerables tipos de listados. A continuación, ordenamos con el comando `sort` los transcritos humanos por el nombre del gen al que pertenecen para su posterior estudio:

Figura 83. Listados de genes humanos ordenados alfabéticamente

```
% sort catalogo.txt | head -5

A1BG-AS1  NR_015380    chr19   +    58351969 58355183 4
A1BG     NM_130786    chr19   -    58346805 58353499 8
A1CF     NM_001198818 chr10   -    50799408 50885675 14
A1CF     NM_001198819 chr10   -    50799408 50885675 15
A1CF     NM_001198820 chr10   -    50799408 50885675 14
```

También podemos obtener el listado de los transcritos ordenado por el número de exones. Para generar una clasificación descendente (de mayor a menor cantidad), emplearemos la opción `-r` del comando `sort`:

Figura 84. Listados de genes humanos ordenado por el número de exones

```
% sort -rnk 7 catalogo.txt | head -5
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
```

El fichero `refGene.txt` contiene el listado de los transcritos humanos. Como un gen puede dar lugar a varios transcritos alternativos, podemos contar cuantos genes tiene el genoma humano y averiguar cuales poseen un mayor número de transcritos. Para conseguirlo, ordenaremos los nombres de los genes e introduciremos diferentes variantes del comando `uniq` para agruparlos:

Figura 85. Trabajando con transcritos y genes humanos

```
% gawk '{print $1;}' catalogo.txt | sort | more
A1BG
A1BG-AS1
A1CF
A1CF
A1CF
A1CF
A1CF
A1CF
A1CF
...

% gawk '{print $1;}' catalogo.txt | sort | uniq | more
A1BG
A1BG-AS1
A1CF
...

% gawk '{print $1;}' catalogo.txt | sort | uniq | wc -l
27656

% gawk '{print $1;}' catalogo.txt | sort | uniq -c | more
1 A1BG
1 A1BG-AS1
6 A1CF
...

% gawk '{print $1;}' catalogo.txt | sort | uniq -c | sort -rn
185 KIR2DS2
148 LOC101928804
128 KIR2DS4
103 KIR3DS1
90 KIR2DL4
...
```

Figura 85

La mejor forma de estudiar el comportamiento de una línea de comandos es la deconstrucción. Eliminando las últimas instrucciones, podemos mostrar el resultado parcial de la ejecución en pantalla.

Otra operación muy frecuente es el cálculo de valores promedio. En el siguiente ejemplo, el estudiante calculará el promedio de exones por transcrito y la longitud media de éstos. El funcionamiento de GAWK es similar en los dos casos, trabajando con la variable `t`, que hace las veces de contador que acumula la suma total. La división por el número de líneas visitadas (`NR`), una vez finalizada la lectura del fichero, genera el valor promedio en cada caso:

Figura 86. Análisis bioinformático del catálogo de genes

```
% gawk 'BEGIN{t=0;}{t=t+$7;}END{print t/NR;}' catalogo.txt

9.41261

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

56983.3
```

Para ilustrar la utilidad de asociar dos ficheros de texto tabulado, combinaremos el catálogo de genes humanos introducido hasta ahora (*H. sapiens*, archivo `refGene.txt`) con el catálogo equivalente en el genoma del ratón (*M. musculus*, renombrado aquí como `refGene_mouse.txt` una vez descargado del servidor genómico de UCSC).

Figura 87. El catálogo `refGene_mouse.txt` de genes de ratón

```
% head -5 refGene_mouse.txt

614 NM_001037800 chr8 - 3917654 3926841 3918676 3926792 9 3917654,3919949,3921339,3921982,3923294,
3924209,3925354,3926551,3926746, 3918827,3920059,3921491,3922069,3923381,3924296,3925435,3926638,
3926841, 0 Cc209b cml cml 2,0,1,1,1,1,1,0,

186 NM_028506 chr4 - 118995497 119056439 118996602 119006187 11 118995497,118997145,119003727,
119004086,119006184,119012365,119022334,119025674,119029242,119032162,119056275, 118996798,
118997213,119003842,119004223,119006326,119012478,119022479,119025829,119029452,119032342,
119056439, 0 Ccdc30 cml cml 2,0,2,0,0,-1,-1,-1,-1,-1,-1,

911 NR_027891 chr1 + 42757180 42760021 42760021 42760021 1 42757180, 42760021, 0 2900092D14Rik
unk unk -1,

637 NR_027890 chr2 - 6843747 6849768 6849768 6849768 1 6843747, 6849768, 0 5031426D15Rik unk unk
-1,

815 NM_027904 chr16 - 30256464 30267618 30259323 30260967 2 30256464,30267542, 30260970,30267618,
0 Cpn2 cml cml 0,-1,
```

Supongamos que deseamos averiguar qué genes poseen en común estas dos especies, registrando el cromosoma donde están ubicados en cada especie. Una forma de lograrlo sería considerar aquellos genes que comparten el mismo nombre en los dos genomas, omitiendo las diferencias debidas al uso de mayúsculas o minúsculas. Para ello, primero es necesario extraer de ambos ficheros exclusivamente los dos atributos que nos interesan, guardándolos en dos archivos denominados `catalogo1.txt` y `catalogo2.txt`. Para descartar las combinaciones debidas a los transcritos alternativos del mismo gen en cada catálogo, filtramos únicamente la parte de información que nos interesa:

El estudiante encontrará en la asignatura Genómica computacional las técnicas más convenientes para identificar genes ortólogos entre dos especies.

Figura 88. Cruzando dos catálogos de genes (I)

```
% gawk '{print $13,$3}' refGene.txt | sort | uniq > catalogo1.txt

% more catalogo1.txt
...
NANOG chr12
...

% gawk '{print $13,$3}' refGene_mouse.txt | sort | uniq > catalogo2.txt

% more catalogo2.txt
...
Nanog chr6
...
```

Posteriormente, mediante el comando `join`, podemos asociar los genes que presentan el mismo nombre en las dos especies. Activaremos la opción `-i` para ignorar las diferencias de mayúsculas/minúsculas. Cada línea del resultado final contiene el nombre del gen y la localización de éste en ambos genomas:

Figura 89. Cruzando dos catálogos de genes (II)

```
% join -i catalogo1.txt catalogo2.txt | more
...
NANOG chr12 chr6
...
```

Figura 89

Genes comunes entre el hombre y el ratón. Para cada gen, mostramos su nombre y ubicación en cada especie.

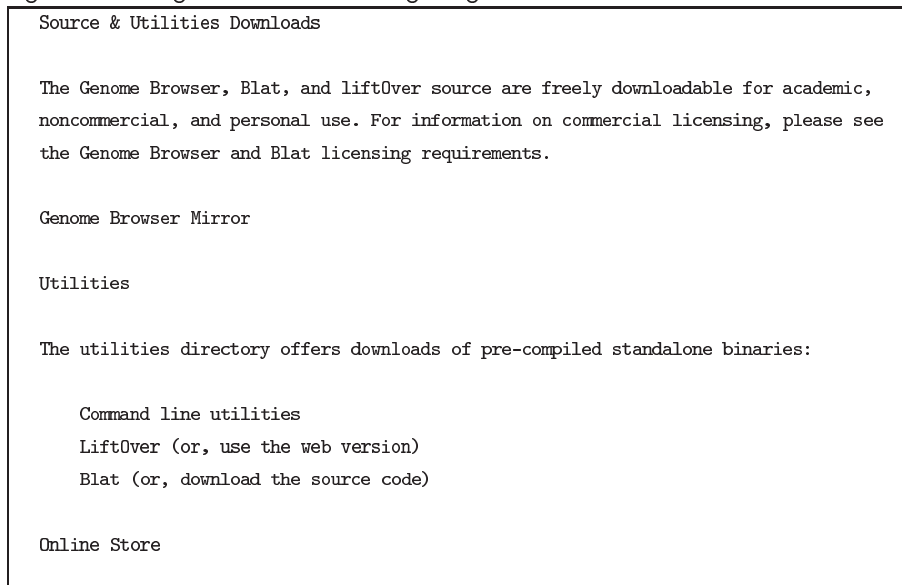
A partir de este análisis preliminar del catálogo de genes humanos, podéis poner en práctica múltiples variantes de los comandos aquí mostrados. Así, es posible añadir más atributos, más genomas o más ficheros con otras propiedades, para ampliar esta exploración. En consecuencia, animamos al estudiante a experimentar con cada bloque de comandos utilizado durante este ejercicio. Con esta aplicación práctica queda demostrada la validez del terminal de UNIX a la hora de analizar eficientemente datos biológicos. Más adelante, introduciremos sistemas más complejos para gestionar grandes conjuntos de datos, de modo que el usuario podrá reproducir el mismo caso práctico con dichas técnicas.

Descarga de las herramientas de UCSC

El navegador genómico de UCSC suministra gratuitamente 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 programa `wget` a la hora de descargar un directorio *web* completo, vamos a proceder a obtener una copia completa de las utilidades de UCSC.

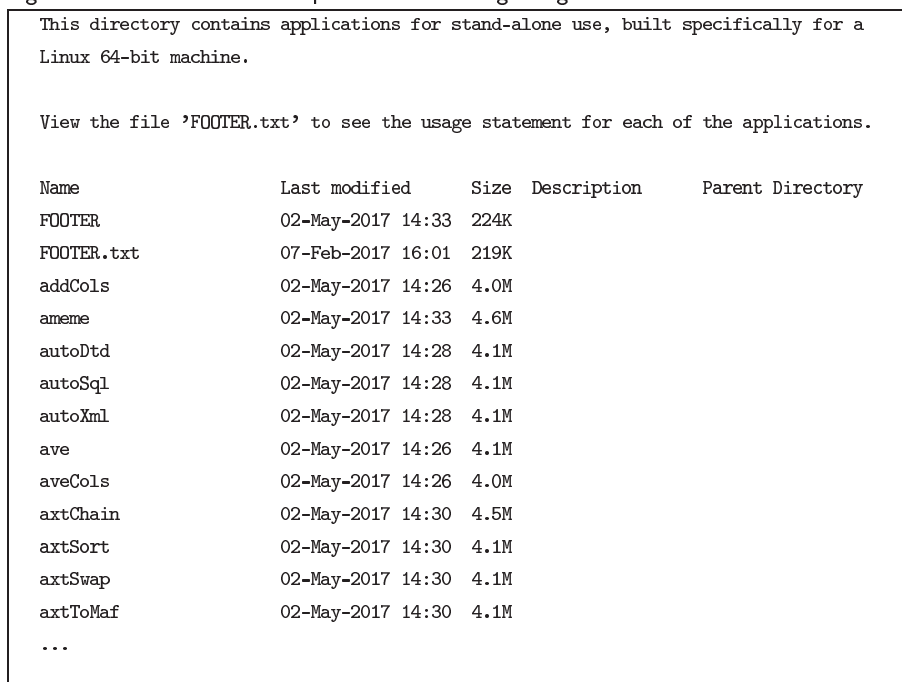
Debemos acceder nuevamente a la página de descargas del navegador, incluida en la Tabla 16. Una vez allí, buscaremos la sección **Source and Utilities Downloads** (en inglés, código fuente y descarga de utilidades).

Figura 91. Descarga de utilidades del navegador genómico UCSC



Ahora presionaremos sobre el enlace **The utilities directory** para acceder a la página de selección de plataforma de UNIX. Para instalar las herramientas de UCSC en Linux Ubuntu MATE debemos seleccionar el enlace `linux.x86_64`.

Figura 91. Listado de utilidades para Linux del navegador genómico UCSC



Para descargar una copia exacta del directorio de utilidades de UCSC para trabajar en Linux, debemos ejecutar el comando `wget` sobre esta página *web*. Añadiremos la opción `-r` para recuperar la carpeta completa.

Figura 92. Descarga de las utilidades de UCSC

```
% wget -e robots=off -r http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64

--2017-05-17 10:33:13-- http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64
Resolving hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)... 128.114.119.163
Connecting to hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)|128.114.119.163|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/ [following]

--2017-05-17 10:33:14-- http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/
Connecting to hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)|128.114.119.163|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64
  OK .....                               172M=0s
2017-05-17 10:33:14 (172 MB/s) - hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64 saved [21507]

...

--2017-05-17 10:33:20-- http://hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/addCols
Connecting to hgdownload.soe.ucsc.edu (hgdownload.soe.ucsc.edu)|128.114.119.163|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4221975 (4,0M) [text/plain]
Saving to: hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/addCols

  OK .....                               1% 130K 31s
 50K .....                               2% 259K 23s
100K .....                               3% 259K 20s
...
4000K .....                             96% 195M 0s
4050K .....                             99% 90,6M 0s
4100K .....                             100% 114M=3,2s

2017-05-17 10:33:23 (1,28 MB/s) - hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/addCols saved
[4221975/4221975]

...
```

Finalmente, podemos listar el nuevo directorio que se ha copiado en nuestro ordenador y comprobar que los programas están perfectamente instalados en el terminal de nuestro ordenador.

Figura 93. Verificando las herramientas descargadas

```
% ls -l hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64

addCols
ameme
autoDtd
autoSql
autoXml
...

% ./hgdownload.soe.ucsc.edu/admin/exe/linux.x86_64/addCols

addCols - Sum columns in a text file.
usage:
  addCols <fileName>
adds all columns (up to 16 columns) in the given file,
outputs the sum of each column. <fileName> can be the
name: stdin to accept input from stdin.
```

Resumen

A lo largo de esta unidad hemos explorado los métodos básicos para llevar a cabo el análisis bioinformático. Trabajando con máquinas virtuales que implementan plataformas funcionando bajo la filosofía UNIX, hemos introducido la terminología básica de los sistemas operativos, aprendiendo a emplear los comandos apropiados del terminal para operar con los ficheros de datos. Además, hemos introducido un conjunto de construcciones básicas para diseñar protocolos automáticos de análisis que reducen considerablemente el tiempo dedicado a estas actividades. De este modo, ahora sabemos manipular este tipo de información de forma sencilla pero eficiente. Junto con este entorno de trabajo, hemos aprendido que tenemos acceso libre a una gran cantidad de datos biológicos que podemos almacenar localmente en nuestra propia máquina para acelerar su procesamiento y reducir el tiempo de cálculo. En resumen, este núcleo de herramientas conforma, pues, una excelente aproximación para extraer nuevo conocimiento de la información biológica.

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 qué funciones realiza el comando `fold` del terminal. Analizad 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. Estudiad el comando `sed` del terminal. Para probar su eficacia, grabad una hoja de estilo de MicrosoftExcel™ en formato texto (seleccionad el tabulador como separador de campos). Una vez en 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).

4. Diseñad un *script* en el terminal que os permita realizar el análisis completo de una serie de ficheros de la clase `refGene.txt` almacenados en un directorio. Cada fichero debe contener en su propio nombre el organismo al que pertenece para evitar nombres de fichero duplicados (p.e. `refGene_human.txt`). Para cada genoma podemos realizar las mismas preguntas mostradas en el caso de estudio de los materiales teóricos.

Ejercicios de autoevaluación

1. Enumera los cuatro componentes básicos de una computadora.
2. Enumera las etapas principales de la generación de un fichero ejecutable.
3. Enumera las etapas en la vida de un proceso.
4. Enumera las cuatro operaciones básicas con ficheros.
5. Cita cómo se codifican el directorio raíz, el directorio anterior y el actual.
6. Enumera los tres dominios de usuarios.
7. Describe las clases de permisos que podemos asignar a un fichero.
8. ¿ Para qué es utilizado el comando `man` en el terminal ?
9. Enumera los comandos del terminal para navegar por el sistema de ficheros.
10. Enumera varios comandos del terminal para gestionar el sistema de ficheros.
11. Cita el comando para modificar los permisos sobre un fichero o un directorio.
12. Enumera algunos comandos para acceder al contenido de un fichero.
13. Explica cómo acceder a ficheros previamente comprimidos desde el terminal.
14. Explica cómo recuperar el listado de los comandos usados en una sesión.
15. Enumera los comandos adecuados para ordenar y combinar ficheros.
16. Describe el comando empleado habitualmente para buscar patrones de texto.
17. Cita la diferencia entre "|" o ">" en la comunicación entre procesos.
18. Discute la diferencia entre los bloques BEGIN y END en programas GAWK.
19. Explica las siguiente variables de GAWK: \$1, \$0, NR, NF, OFS y FS.
20. ¿ Por qué son enormemente útiles los protocolos de tareas automatizados ?

Solucionario

1. El procesador, la memoria, 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`, `pushd` y `popd`.
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 éste.
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 realizar un número (prácticamente) infinito de veces el mismo conjunto de tareas sobre múltiples grupos de datos sin asistencia del usuario.

Bibliografía

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

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 Rob Pike (1984). *Unix Programming Environment*. Prentice Hall. ISBN: 013937681X.

Brian W. Kernighan and D.M. Ritchie (1988) *C Programming Language (2nd Edition)*. Prentice Hall. ISBN: 0131103628.

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

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

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

Alfred V. Aho, Brian W. Kernighan and Peter J. Weinberger (1988). *The AWK Programming Language*. Addison Wesley. ISBN: 020107981X.

Cameron Newham (2005) *Learning the bash Shell, Third Edition*. O'Reilly Media. ISBN: 0-596-00965-8.

Índice alfabético

C	
canal estándar de entrada	40
canal estándar de error	40
canal estándar de salida	40
compiladores	10
compresión de ficheros	31
conexión remota	57
D	
descarga de datos	57
E	
empaquetar directorios	32
ENIAC	5
enlaces	13
expresiones regulares	27, 48
búsqueda	48
operaciones	48
sustitución	48
F	
FASTA	29, 58
formato	29, 58
fichero	12
añadir	12
abrir	12
cerrar	12
escribir	12
leer	12
operaciones	12
ficheros de texto	36
organización	36
ficheros de texto tabulados	36, 58
operaciones	36
G	
GAWK	42, 50
<i>arrays</i> asociativos	46
<i>pipes</i>	47
FS	43
NF	43
NR	43
OFS	43
ORS	43
RS	43
\$0	43
\$1,\$2,...	43
if	45
bloques de instrucciones	44
condicionales	45
estructura	44
expresiones regulares	50
tuberías	47
variables	43
genoma humano	59
catálogos de genes	62
descargar cromosomas	60
descargar genes	63
desempaquetar cromosomas	61
el fichero refGene.txt	62
FASTA	60
versiones	59
visualizar cromosomas	61
GNU	8
I	
interfaz	
gráfica	20
línea de comandos	20
ventajas de la línea de comandos	20
L	
Linux	8
LiveCD	16
gestión de paquetes	19
GNU	8
imagen ISO	16
listado de paquetes básicos	19
obtener una copia	16
Synaptic Package Manager	19
Ubuntu	17
Ubuntu MATE	17
variantes	8
M	
máquina virtual	14
<i>guest</i>	14
<i>host</i>	14
huésped	14
huesped	14
instalación de Ubuntu MATE	17
protocolo básico	16
VirtualBox	16
virtualización	14
metacaracteres	27, 48, 49
N	
navegador genómico	58, 62
descargas	58
utilidades	68
O	
ordenador	
<i>device driver</i>	5
<i>hardware</i>	5
<i>software</i>	5
arquitectura	5
bus de comunicaciones	5
CPU	5
memoria	5
PC	8
periféricos	5
procesador	5
RAM	5
P	
paralelización	11
proceso	10, 34
estados	10, 34
programa	9
compilación	9
compiladores	10
paralelización	11
programas	
<i>scripts</i>	10
ejecutables	10
R	
redirección de canales	40

RefSeq	62, 63	ssh	57
ruta de acceso a ficheros	12	wget	58, 63, 68, 69
S			
sistema de ficheros	11	acceso a la <i>web</i>	57
<i>path</i>	12	código	21
comandos	24	canal estándar de entrada	40
enlaces	13	canal estándar de error	40
independencia de dispositivos	13	canal estándar de salida	40
links	13	comandos	21, 22
montaje	13	combinación de comandos	39
permisos	13, 14	comunicación de comandos	39
ruta	12	conexión remota	57
ruta absoluta	12	directorio inicial	25
ruta relativa	12	directorio raíz	25
sistema operativo	6	expresiones regulares	48
fichero	12	ficheros de texto	36, 58
lista de procesos	6	ficheros de texto tabulados	36, 58
multiprocesadores	11	foreground	21, 34, 35
planificación de procesos	10	gestión de procesos	34
proceso	10, 34	metacaracteres	27, 48, 49
recursos	7	operaciones sobre ficheros de texto	36
sistema de ficheros	11	operaciones sobre ficheros de texto ta-	36
T			
terminal	20	bulados	36
<i>append</i>	41	primer plano	21, 34, 35
<i>background</i>	21, 34, 35	protocolos	51, 53–55
<i>pipes</i>	41, 42	redirección de canales	40
<i>scripts</i>	51, 53–55	segundo plano	21, 34, 35
<i>wildcards</i>	27, 48, 49	sintaxis	21, 22
alias	50	tabulador	26
apropos	23	tuberías	41, 42
cal	22	ventajas	20
cat	30, 41	transferencia de ficheros	57
cd	26, 27	U	
chmod	28	Ubuntu	
date	24	MATE	17
diff	31	UCSC genome browser	58, 62
echo	53	descargas	58
emacs	33	pista refGene	63–65
grep	37, 49	pista refGene (ratón)	67
gzip	31	utilidades	68
head	29	UNIX	7
history	24	desarrollo	7
join	39	genealogía	8
kill	35	Linux	8
less	29	permisos	13, 14
ls	25, 49	terminal	20
man	23	V	
more	29	VirtualBox	16
mv	27	copia de una MV	19
od	30	crear nueva MV	17, 18
popd	26	ficheros OVA	19
ps	34	gestión de paquetes	19
pushd	26	protocolo básico	16
pwd	25	virtualización	14
rmdir	27		
rm	27		
set	55		
sleep	34		
sort	38		
tail	29		
tar	31, 32		
top	36		
wc	29		
which	55		
whoami	24		
zcat	33		
zmore	33		
scp	57		
sftp	57		