UNIVERSIDAD AUTÓNOMA DE SINALOA
Facultad de Informática Culiacán
MAESTRÍA EN INFORMÁTICA APLICADA

Desarrollo de un Protocolo Informático para el Análisis Genómico de Organismos Procariotas.

Que como requisito parcial para obtener el grado de
MAESTRO EN INFORMÁTICA APLICADA
presenta

Rogelio Prieto Alvarado

DIRECTOR DE TESIS:
Dr. Inés Fernando Vega López
Dr. Cristóbal Chaidez Quiroz

Culiacán, Sinaloa a Marzo de 2023

Agradecimientos

Es necesario *agradecer profundamente* a quienes también están presentes en este trabajo.

A mis padres, Mercedes y Manuel, por su amor permanente.

A mi esposa, Xóchitl por su paciencia y apoyo para concretar esta meta.

A mi alma máter, la Universidad Autónoma de Sinaloa por abrir sus puertas para mi formación académica. Al Parque de Innovación Tecnológica por el apoyo brindado en infraestructura. A mi querida Facultad de Informática Culiacán por recibirme otra vez como alumno.

Al Dr. Inés F. Vega López, mi asesor, por su confianza y por ser guía indispensable para realizar este trabajo. Su congruencia académica es ejemplar.

Al Dr. Luis Fernando Lozano Aguirre, del Centro de Ciencias Genómicas UNAM (CCG-UNAM), por brindarme la capacitación para incursionar en bioinformática.

A los miembros de Laboratorio Nacional para Investigación en Inocuidad Alimentaria (LANIIA, CIAD-UAS) por permitirme contribuir y ser un equipo.

A la Dra. María Elena Báez y al Dr. Ricardo Parra (FCQB-UAS) por invitarme a participar en su equipo de bioinformática.

Índice general

Índice de figuras

Índice de tablas

Resumen

La bioinformática es una subdisciplina científica que se apoya en las ciencias de la computación para almacenar, analizar, visualizar y anotar información biológica como secuencias de ADN y proteínas. Este trabajo propone un protocolo informático para la realización de análisis biológicos que se enmarcan dentro de la genómica comparativa, específicamente para organismos procariotas. Se presentan apartados de computación para entender y usar instrucciones del sistema operativo GNU/Linux y formatos de archivo que se recomienda conocer para quienes deseen incursionar en la bioinformática. También se describen conceptos biológicos fundamentales para entender el análisis de este tipo de información: estructura del ADN, secuenciación; y un flujo de trabajo de genómica comparativa para realizar evaluación de los ensambles, anotación de genes, establecer relaciones filogenéticas y su interpretación. La generación de un árbol filogenético está incluida en la parte final.

Además, este trabajo provee información actualizada y brinda un enfoque holístico de la bioinformática, aborda el protocolo propuesto desde la perspectiva biológica y computacional. De esta manera, la audiencia puede ser multidisciplinaria: a) personas con formación académica del área biológica que requieren fortalecer sus conocimientos en computación para implementar el protocolo propuesto y b) personas con formación en ciencias de la computación que requieren reforzar sus conocimientos biológicos para entender el resultado del análisis.

Palabras clave: bioinformática, genómica comparativa, árboles filogéneticos, Linux

Capítulo 1
Introducción

El comercio internacional ha crecido de manera significativa en los últimos años [71]. En el comercio internacional, los países se agrupan, según el producto a considerar, en países productores y países consumidores. México es la novena mayor economía mundial por sus exportaciones [68]. En el caso de productos agropecuarios, México es un país productor principalmente de aguacate, tomate, pimiento, fresas, pepino, melón, sandía, papaya, café, ganado vacuno, miel y camarón [18]. Los países consumidores tienen establecidos criterios elevados para evaluar la calidad y la inocuidad de los productos alimenticios que adquieren. La inocuidad se refiere a que los productos estén libres de químicos que puedan ser dañinos para los consumidores o a que los productos no contengan contaminantes biológicos.

La Organización Mundial de la Salud (OMS) declara que las enfermedades transmitidas por los alimentos suponen una importante carga para la salud, pues anualmente se reportan millones de personas que enferman (y miles de ellas mueren) por consumir alimentos insalubres, por tanto, la OMS contempla que la inocuidad de los alimentos debe ser incorporada en los programas de seguridad alimentaria. Los esfuerzos a nivel mundial se deben enfocar en generar políticas y actividades que persigan la seguridad alimentaria. Estos esfuerzos deben estar presentes desde la producción hasta el consumo, a lo largo de toda la cadena alimentaria [96].

La OMS señala que los alimentos transmiten más de doscientas enfermedades y pueden causar problemas de salud a largo plazo, siendo las personas vulnerables las más expuestas. La contaminación de los alimentos puede darse en cualquier momento de la compleja cadena de suministro. Este factor, en conjunto con la globalización, el incremento de bacterias dañinas que se han vuelto resistentes a fármacos, así como el

carácter multisectorial y multidisciplinario de la inocuidad alimentaria hacen que la inocuidad alimentaria sea aún más compleja y esencial [96].

Cuando un país consumidor recibe un producto contaminado, aplica la normatividad vigente [13] [14] [15]. La normatividad incluye detectar la causa de la contaminación y deslindar responsabilidades. Para detectar la causa de la contaminación es necesario mantener la trazabilidad del producto con el propósito de encontrar el origen de la contaminación y prevenir futuros problemas de salud pública. Si el productor es el responsable, recibe sanciones que pueden ser de tipo económicas o de prohibición temporal para comercializar sus productos.

Para detectar el origen de la contaminación biológica ocasionada por la presencia de microorganismos patógenos, de manera tradicional, se han empleado técnicas microbiológicas, serológicas y de biología molecular. Sin embargo, estas técnicas han demostrado limitantes: son lentas y tienen alta sensibilidad. Actualmente, se están implementando técnicas novedosas incluidas en la bioinformática. Específicamente, aplicando técnicas de genómica comparativa que permiten identificar similitudes biológicas.

La genómica tiene como meta determinar y analizar la secuencia completa del ácido desoxirribonucleico (ADN) de un organismo, es decir, su genoma [74]. La genómica comparativa es la encargada de estudiar la relación entre la estructura y la función del genoma a través de distintas especies o cepas, también entre individuos de la misma especie [66]. Se utilizan los árboles filogenéticos como herramienta de la genómica comparativa [74] [66] para identificar similitudes de genomas entre organismos y entre grupos de organismos. Un árbol filogenético es la representación gráfica de las relaciones evolutivas entre un conjunto de secuencias de ADN, especies u otro nivel de taxonomía definido. El proceso de búsqueda de similitudes de genomas entre organismos tiene como consecuencia la generación de una gran cantidad de datos biológicos que requieren ser analizados mediante técnicas eficientes y aplicar inferencia estadística. Es necesario el uso de computadoras para diseñar y desarrollar

herramientas de software que permitan mejorar y hacer más eficiente el análisis de datos biológicos. Por esta necesidad surgió la bioinformática.

El surgimiento y desarrollo de la bioinformática se ha dado gracias a las aportaciones de las matemáticas, la informática, la química y la biología; con conocimiento específico de ingeniería de software, inteligencia artificial, estadística, conocimientos sobre procesos biológicos y genéticos, entre otros. La bioinformática realiza análisis de datos biológicos mediante la utilización de herramientas de software que permiten automatizar el proceso de análisis [58]. La bioinformática es una subdisciplina científica que se apoya en las ciencias de la computación para almacenar, analizar, visualizar y anotar información biológica como secuencias de ADN y aminoácidos [67]. La bioinformática permite aportar conocimiento integral de la función biológica sobre el funcionamiento de un gen o parte de un sistema biológico [74]. El desarrollo de la bioinformática impacta en las relaciones comerciales entre países al aportar conocimiento científico para garantizar la inocuidad y la seguridad alimentarias. El desarrollo de la bioinformática también impacta en distintas áreas: medicina, agricultura, derecho, ciencias forenses, entre otras [44]. En la medicina, la contribución de la bioinformática permite diseñar fármacos y vacunas en menor tiempo, fortalecer la medicina preventiva al identificar los patrones y las causas de las enfermedades que afectan a individuos o poblaciones específicas. Un ejemplo reciente es la contribución de la bioinformática para contar con la capacidad científica para analizar el genoma del COVID-19 y desarrollar vacunas seguras y efectivas en menos de un año [92]. En agricultura, la bioinformática permite entender mejor el funcionamiento de las plantas y contribuye en la seguridad alimentaria al mejorar la calidad de las plantas para consumo humano. En derecho y ciencias forenses, permite aportar elementos de prueba como resultado de la comparación de genomas.

Para la utilización de las herramientas de software desarrolladas por las ciencias de la computación para ser utilizadas en la bioinformática, además de infraestructura tecnológica especializada y de vanguardia, se requiere formar recursos humanos que

sean capaces de utilizar las herramientas e interpretar los resultados obtenidos. Esto exige a los profesionales que se incorporan a la bioinformática contar con conocimiento multidisciplinario tanto de biología como de ciencias de la computación.

La secuenciación permite leer el ADN de un organismo. Leer el ADN significa tener la capacidad de leer el orden de nucleótidos (`A`, `C`, `G` y `T`) que constituye el ADN de un organismo. Adicionalmente, en el contexto actual, cada vez se tiene más fácil acceso a obtener datos de secuenciación de organismos [10], los costos de secuenciación han disminuido a la par que ha aumentado la disponibilidad de cepas en bases de datos o repositorios públicos. La Figura 1.1 muestra un seguimiento de costos del *Short Read Archive* (SRA, por sus siglas en inglés). El *Sequence Read Archive* (SRA) es un repositorio que almacena datos de secuencias crudas de tecnologías de secuenciación de «próxima generación», como Illumina, 454, Ion Torrent, Complete Genomics, PacBio y Oxford Nanopore. Además de los datos de secuencias crudos, el SRA almacena ahora información de alineamiento en forma de colocación de lecturas en una secuencia de referencia. La Figura 1.1 tiene definido el eje y con escala logarítmica para denotar la disminución exponencial de los costos de secuenciación a lo largo del tiempo, principalmente a partir de 2007. Por su parte, la Figura 1.2 muestra el crecimiento del repositorio SRA, el eje y tiene escala logarítmica para mostrar el crecimiento exponencial del tamaño del repositorio. Los datos de este repositorio de datos están disponibles para el público en acceso libre.

Las grandes cantidades de datos biológicos almacenados en este y otros repositorios públicos requieren ser analizados masivamente con técnicas de bioinformática y específicamente mediante la genómica comparativa para detectar similitudes biológicas. Para diseñar y ejecutar el análisis de los datos biológicos residentes en los repositorios públicos, el capital humano requiere contar con conocimientos especializados en biología y computación, en resumen, con capacitación de alto nivel en bioinformática.

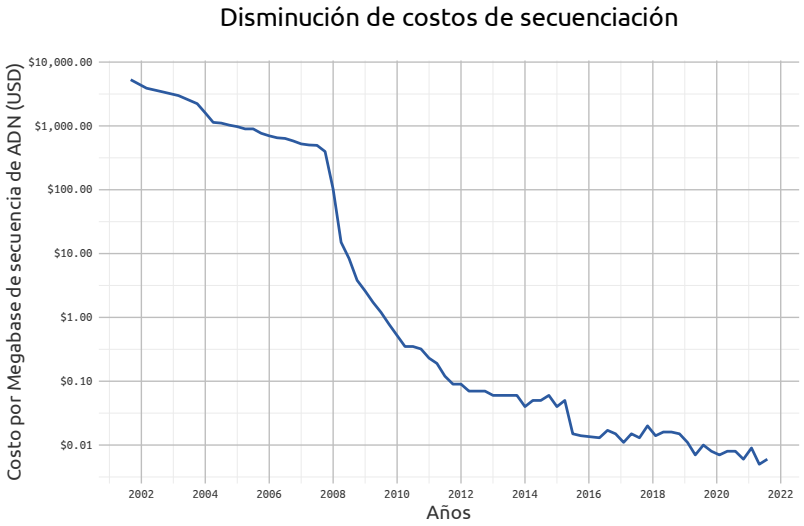

Figura 1.1: Disminución de los costos de secuenciación en el repositorio *Short Read Archive*[1].

Atendiendo este contexto, el objetivo de este documento es proponer un protocolo informático para la realización de análisis biológicos que se enmarcan dentro de la genómica comparativa, específicamente para organismos procariotas. El documento provee información actualizada y brinda un enfoque holístico de la bioinformática, aborda el protocolo propuesto desde la perspectiva biológica y computacional. De esta manera la audiencia puede ser multidisciplinaria: a) personas con formación académica del área biológica que requieren fortalecer sus conocimientos en computación para implementar el protocolo propuesto y b) personas con formación en ciencias de la computación que requieren reforzar sus conocimientos biológicos para entender el resultado del análisis.

El documento consiste de seis capítulos. El actual, que brinda un primer acercamiento y una breve descripción del contenido del documento. El Capítulo 2 y 5 contienen conocimientos del área biológica, mientras que los Capítulos 3 al 5 se refieren al conocimiento referente a ciencias de la computación. El Capítulo 2 aborda

[1]Figura generada con datos descargados del National Institute of Health (NIH), disponibles en: https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost. Datos actualizados en abril de 2022.

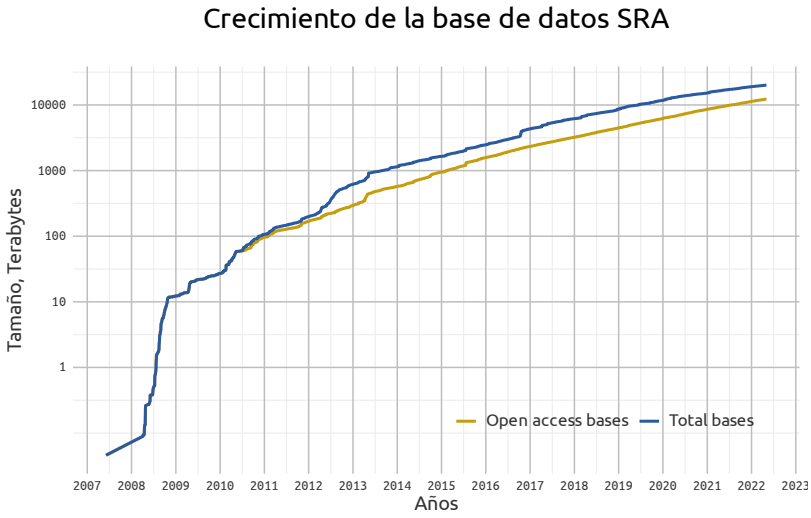

Figura 1.2: Crecimiento del repositorio *Short Read Archive*[2].

contenido del área biológica que se considera fundamental para incursionar en bioinformática. En dicho capítulo se aborda la estructura del ADN, la definición de análisis genómico y las técnicas de secuenciación del ADN.

Por su parte, el Capítulo 3 trata sobre los conocimientos de GNU/Linux requeridos para realizar tareas específicas de bioinformática. También este capítulo incluye la descarga de datos biológicos de bases de datos públicas y la instalación de herramientas bioinformáticas para realizar análisis.

En el Capítulo 4 se abordan los formatos de archivo utilizados en bioinformática para realizar análisis biológicos. Conocer estas estructuras permitirá entender los criterios específicos que permiten el almacenamiento y recuperación de información.

El Capítulo 5 presenta un análisis mediante genómica comparativa de genomas bacterianos que muestra como resultado la generación de un árbol filogenético, una herramienta gráfica para establecer las relaciones filogenéticas. El capítulo incluye las etapas generales: la evaluación inicial de la calidad de las lecturas de secuenciación,

[2]Figura generada con datos descargados del National Institute of Health (NIH), disponibles en: https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost. Datos actualizados en abril de 2022.

ensamblado de lecturas, establecimiento de relaciones filogenéticas. En este capítulo, se describen las etapas considerando su enfoque, herramienta de análisis, descripción e interpretación de resultados. El Capítulo 6 presenta las conclusiones del presente trabajo.

Finalmente, en el presente documento se incluyen dos anexos. El Anexo I que aporta conocimiento sobre el uso de GNU/Linux utilizando la Línea de Comandos, la ejecución de los comandos más relevantes para realizar tareas básicas y la ejecución de los comandos para la operación con archivos y directorios.

Capítulo 2
Antecedentes

Este capítulo presenta contenido del área biológica que se considera fundamental para incursionar en bioinformática. El capítulo aborda la estructura del ADN, la definición de análisis genómico y las técnicas de secuenciación del ADN.

2.1. Estructura del ADN

El ADN, es la molécula que contiene la información genética y es el componente del la cual están hechos los genes [48]. La Figura 2.1 muestra la formas `A-`, `B-` y `Z` de la doble hélice del ADN. El ADN es un polímero lineal formado por monómeros llamados nucleótidos. Los nucleótidos poseen un esqueleto fijo, constituido por unidades repetitivas de azúcar-fosfato. Específicamente un nucleótido consiste de azúcar más fosfato más base nitrogenada. Los azúcares son moléculas de desoxirribosa que dan nombre al ADN. Cada moleécula de azúcar está unido a dos grupos por fosfato mediante dos diferentes enlaces. A cada desoxirribosa se puede unir una de estas cuatro bases: Adenina (`A`), Citosina (`C`), Guanina (`G`), o Timina (`T`) [9].

Un gen, es la unidad fundamental de la información genética que constituye una secuencia del genoma. Un gen es un fragmento de ADN que contiene toda la información necesaria para realizar la síntesis de un polipéptido, y ello lo hace a través del mRNA (ARN mensajero), el cual funge como intermediario. La función de un gen es dar origen a un producto (ya sea proteína o RNA).

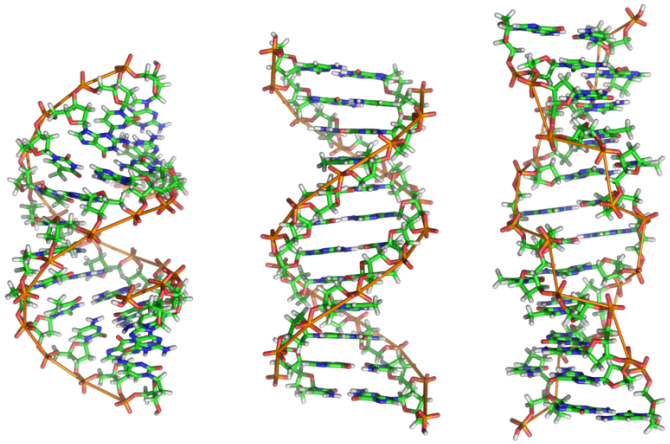

Figura 2.1: ADN. Las formas A-, B-, y Z de la doble hélice del ADN. Imagen de Richard Wheeler bajo licencia GFDL versión 1.2.

2.2. Análisis genómico

El análisis genómico estudia el genoma de un organismo. El genoma está conformado por una secuencia de nucleótidos que en bioinformática se representan por sus iniciales `A, C, G, T`. Cada organismo puede identificarse a través de su genoma. El análisis genómico analiza los datos generados por la secuenciación que son almacenados en archivos con una estructura específica: **`FASTA`**, **`FASTQ`**, **`SAM`**, **`GFF`**, entre otros. Se recomienda leer el Capítulo 4 que aborda los formatos de archivo utilizados en bioinformática.

Los productos principales del análisis genómico son la identificación de los genes (anotación de genes), predecir y estudiar homólogos, identificar el pangenoma (genoma accesorio y *core*) y el *core* genoma, determinar las interacciones con otros organismos y su entorno, diseño de fármacos, entre otros. El término pangenoma se refiere a la conformación de un genoma «central», que contiene los genes presentes en, al menos, alguna de las cepas de una especie. El *core* genoma es el conjunto de genes presentes en todas las cepas de una especie. En cambio, el genoma accesorio, se refiere al genoma «prescindible», está compuesto por genes que se presentan en algunas cepas pero no en todas las cepas de la especie analizada.

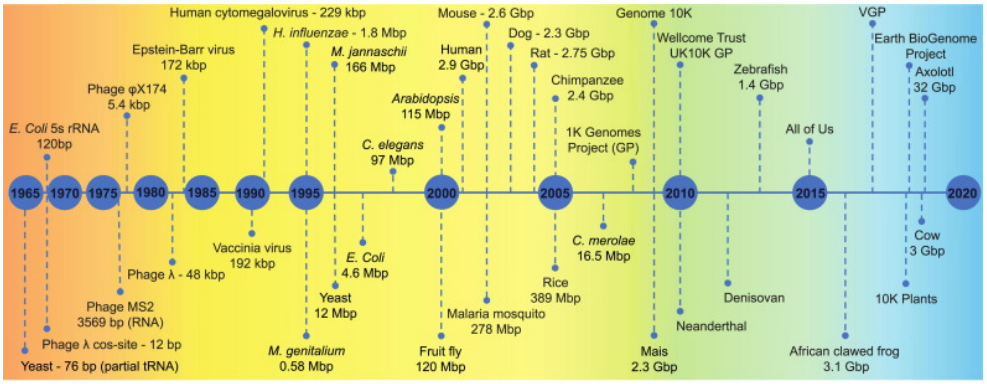

Figura 2.2: Hitos en el ensamblaje del genoma. Cada genoma o proyecto de genoma (GP) tiene un color de acuerdo al enfoque de secuenciación utilizado. Rojo: métodos de secuenciación temprana, Amarillo: secuenciación aleatoria basada en Sanger, Verde: NGS, Azul claro: TGS. Imagen tomada de: «Long walk to genomics» Alice Maria Giani *et. al.* con licencia Creative Commons CC-BY-NC-ND.

2.3. Secuenciación

Tener la capacidad de leer el ADN de un humano o de un organismo es posible gracias a la secuenciación genómica. Leer el ADN significa tener la capacidad de obtener el orden de `A, C, G` y `T` que constituye el ADN de interés [42].

La tecnología de secuenciación pionera fue propuesta por Walter Gilbert and Frederick Sanger en 1977. La tecnología Sanger se utilizó a partir de la dećada de 1980 en máquinas que automatizaban el proceso, por ejemplo en la máquina «Model 373» (Applied Biosystems y Hitachi) y en la máquina «ALF» (Pharmacia y Molecular Dynamics) [85].

La tecnología de Sanger marco un hito, con la salida al mercado de los primeros dispositivos automatizados, permitió que se iniciaran prometedores proyectos genómicos (ver Figura 2.2, franja en color amarillo). La primera generación de dispositivos secuenciadores producían lecturas de poco menos de 10 kilobases [31].

Un limitación de los dispositivos Sanger era que precisamente que producían longitudes de lectura cortas. Esto ocasionaba dificultades a los científicos para resolver

problemas biológicos como el ensamblado y la determinación de regiones genómicas complejas [77].

Actualmente se utilizan las tecnologías de Secuenciación de Siguiente Generación (NGS, por sus siglas en inglés, *Next Generation Sequencing*), que debido al procesamiento masivo y en paralelo que realizan de las muestras biológicas, han logrado reducir notablemente los costos y el tiempo para obtener la secuencia genómica. Una ventaja de las tecnologías NGS es que obtienen lecturas más largas que facilitan el ensamblado.

Roche 454 (2004) fue el primer sistema comercial de secuenciación que utilizó la tecnología NGS. Roche 454 debe su nombre a la implementación de la tecnología de piro-secuenciación bajo la plataforma 454 que fue realizado por la empresa Roche. Posteriormente, se desarrollaron las plataformas Solexa-Illumina (2006) basada en secuenciación por síntesis, la plataforma SOLID (2007) que se basó en secuenciación por ligación, y la plataforma Ion Torrent (2010) basada en detección de pH. Las plataformas antes mencionadas requieren que el ADN sea amplificado previo a la secuenciación. Posteriormente se desarrollaron plataformas que no requieren la amplificación del ADN, Helicos (2008) y SMRT Pacific Biosciences (PacBio, 2010). Ambas plataformas realizan la secuenciación directamente de una molécula de ADN.

En esta sección se revisarán las plataformas de secuenciación que son tendencia actual, específicamente, las plataformas PacBio, Ion Torrent y Solexa-Illumina.

2.3.1. Pacific Bioscience (PacBio)

La tecnología PacBio [77] realiza la secuenciación durante el proceso de replicación de la molécula de ADN objetivo utilizando la tecnología SMRT (*Single Molecule, Real-Time*). SRMT utiliza una plantilla en forma de matriz llamada SMRTbell, donde cada elemento de la matriz contiene una porción de ADN circular de cadena sencilla, la cual es creada uniendo los extremos con adaptadores de horquilla del ADN bicate-

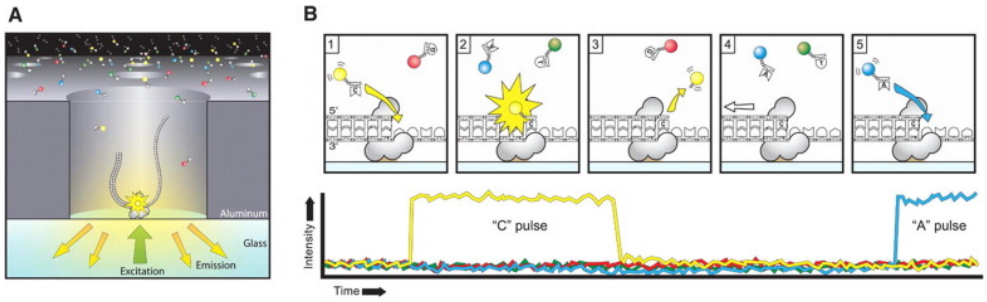

Figura 2.3: PacBio. Secuenciación a través de pulsos de luz. En la parte izquierda, SMTbell se difunde en un ZMW y el adaptador se une a una polimerasa que queda inmovilizada en la parte inferior. En la parte derecha, cada uno de los cuatro nucleótidos es marcado con un tinte fluorescente: G (rojo), C (amarillo), T (azul), A(azul). Imagen tomada de «PacBio Sequencing and Its Applications» de Anthony Rhoads *et. al.* con licencia Creative Commons CC-BY-NC-ND.

nario a secuenciar. La plantilla SMRTbell es cargada en un chip llamado SMRTcell. Toda la plantilla es alineada e infundida en una matriz base compuesta de guías de onda de modo-cero (ZMW, *Zero-Mode Waveguide*). Las ZMW consisten en ranuras o pozos microscópicos, dentro de los cuales se fija en la parte inferior una única molécula de polimerasa. A continuación, se añaden las cuatro bases nucleotídicas (`ATCG`) marcadas fluorescentemente a la SMRTcell. En cada ZMW la polimerasa se une a uno de los extremos de horquilla de la muestra de ADN en la SMRTbell e inicia la replicación, los nucleótidos emiten diferentes pulsos de luz característicos al ser sintetizados por la polimerasa durante la replicación. El pulso de luz emitidos por la reacción de la polimerasa con las bases es capturado mediante una "película", los pulsos correspondientes para cada ZMW son interpretados como una secuencia de bases llamada CLR (*Continuos Long read*). Ver Figura 2.3

2.3.2. Ion Torrent

La tecnología chip de Ion Torrent se basa en un concepto básico: ejecutar la secuenciación por síntesis con detección electroquímica de la misma [62]. Cada una de estas reacciones acoplada a su propio sensor, que a su vez está organizado en una matriz de sensores masivamente paralelos en un chip de semiconductores complemen-

tarios de óxidos metálicos (CMOS, *Complementary Metal Oxide Semiconductor*). Los subproductos químicos fundamentales de síntesis, en este caso, la incorporación de polimerasa son la liberación de pirofosfato escindido del nucleótido incorporado, que es bien conocido y es la base para la detección de piro-secuenciación, y también la liberación de un ion hidrógeno (H+) del 3' OH en la hebra de crecimiento.

La tecnología Ion Torrent también adopta un sistema de detección electroquímica llamado *ion-sensitive field-effect transistors* (ISFET) que puede detectar iones a medida que son liberados por ADN polimerasa durante la secuenciación por la síntesis de ADN. Todos estos elementos electrónicos se centran en detectar y analizar la liberación de un ion de hidrógeno (H+) que se produce cada vez que se agrega un nucleótido trifosfato. La liberación de iones provoca un ligero cambio de pH que es detectado por un sensor CMOS, este dispositivo de transistores detectores de pH se conoce como pHFET (*pH-sensitive Field Effect Transistor*).

Si es lo primero, entonces debe indicar de cual chip está hablando. Cada chip tiene al menos 1.2 millones de sensores. Los chips se componen de un orificio que contiene desoxirribonucleótidos trifosfatos (dNTP) y un cordón de acrilamida con una plantilla de ADN. Justo debajo del orificio se encuentra la capa de detección de óxido metálico, que se encuentra sobre una placa censadora y una "puerta" metálica flotante que transmite información electrónica (los cambios de pH) al semiconductor. El cordón portador contiene una población amplificada de plantillas de secuenciación de ADN clonadas, producida por un proceso de PCR (*Polymerase Chain Reaction*) en emulsión. Este cordón lleva típicamente cientos de miles de copias del mismo fragmento corto de ADN, con el fin de amplificar la señal química producida por la reacción de secuenciación. El orificio actúa para retener el cordón en su sitio, y para ayudar a localizar y retener el H+ liberado después de la incorporación.

Ion Torrent genera lecturas de aproximadamente una longitud de 200 bp (pares de bases), estas lecturas son más cortas que las lecturas de ROCHE-454 y PacBio. Sin embargo, Ion Torrent se considera una opción razonable debido a su bajo costo con

respecto a las tecnologías mencionadas. Las lecturas generadas por Ion Torrent pueden ser utilizadas para llenar los espacios (*gaps*) generados en los ensambles realizados con otras tecnologías [36].

2.3.3. Illumina

Illumina [39] es una plataforma de secuenciación que utiliza la tecnología de secuenciación por síntesis, utilizando terminadores reversibles [5]. La tecnología Illumina aprovecha la amplificación de los fragmentos de ADN y la tecnología patentada de terminación reversible para una secuenciación rápida y precisa a gran escala. La amplificación de los fragmentos de ADN es realizada, mediante el método PCR en puente, para la generación del clúster. El clúster se define como grupos del mismo fragmento.

Las plantillas de secuenciación se inmovilizan en una superficie de celda de flujo (*Cell-Flow*) diseñada para presentar el ADN de una manera que facilita el acceso a las enzimas, al tiempo que se asegura una alta estabilidad de plantilla ligada a la superficie y una baja unión no específica de nucleótidos marcados fluorescentemente.

Primero se genera un clúster, la amplificación en fase sólida crea copias idénticas de cada molécula molde único en proximidad más cercana (diámetro de un micrón o menos). Debido a que este proceso no implica fotolitografía o colocación de fragmentos en pozos, se consiguen densidades del orden de diez millones de racimos de molécula única por centímetro cuadrado.

Para realizar la síntesis de la polimerasa se utilizan cuatro nucleótidos marcados fluorescentemente para secuenciar las decenas de millones de racimos en la superficie de la celda de flujo en paralelo. Durante cada ciclo de secuenciación, un único nucleótido- trifosfato marcado (dNTP) se añade a la cadena de ácido nucleico. El marcador de nucleótidos sirve como un terminador para la polimerización, de modo que después de cada incorporación de dNTP, el colorante fluorescente se visualiza para

identificar la base y luego se escinde enzimáticamente para permitir la incorporación del siguiente nucleótido.

Las lecturas de las bases nitrogenadas (*Base Calls*) se realizan directamente a partir de mediciones de intensidad de fluorescencia durante cada ciclo, lo que reduce en gran medida las tasas de errores brutos en comparación con otras tecnologías. El resultado final es una secuenciación base a base muy precisa que elimina los errores específicos del contexto de secuencia, permitiendo *base calls* confiables a lo largo del genoma, incluyendo regiones de secuencia repetitivas y dentro de homopolímeros.

Capítulo 3
GNU/Linux para bioinformática

En este capítulo se abordarán los conocimientos de GNU/Linux requeridos para realizar tareas de bioinformática específicamente para ejecutar el flujo de trabajo propuesto en este documento. Este capítulo incluye la descarga de datos biológicos de bases de datos públicas y la instalación de herramientas bioinformáticas para realizar análisis biológicos. Existen otros conocimientos de GNU/Linux que son recomendables para quien desee incursionar en bioinformática, debido al espacio limitado y propósito del presente documento, están contenidos en el Anexo I.

Aunque el *prompt* tiene una estructura definida, tal como se muestra en la Figura AI.3, en los siguientes ejemplos de este documento el *prompt* estará conformado únicamente por el signo de pesos `$` para ahorrar espacio y mejorar la legibilidad de las instrucciones mostradas en el *shell*.

3.1. Administración de procesos

En un sistema operativo, un programa es una secuencia de instrucciones almacenadas para indicar a la computadora que realice tareas específicas. Un proceso es un programa en ejecución. Un proceso para su ejecución requiere utilizar recursos de la computadora: tiempo de cómputo del procesador, memoria, archivos, entre otros [89]. Es importante conocer cómo administrar los procesos en GNU/Linux. La administración de procesos permite identificar, monitorear y manipular procesos para detener o reanudar la ejecución. La administración de procesos también permite asignar prioridades a los procesos para que se asigne un mayor o menor tiempo de cómputo del procesador durante la ejecución de un proceso o grupo de procesos determinados.

3.1.1. Monitoreo de procesos

Es posible monitorear los procesos para conocer el tiempo de ejecución así como los recursos que utiliza (tiempo de uso del procesador, memoria RAM, entre otros). En GNU/Linux cada proceso tiene un identificador único, se denota por `PID` (*Process Identifier*). El `PID` es un número entero asignado por el sistema operativo cuando el proceso es creado. El sistema operativo es el responsable de realizar la administración de procesos para alternar el uso del procesador para los procesos. Los procesos pueden estar en diferentes estados:

- En ejecución (*Running*)
- Esperando (*Waiting*)
- Detenido (*Stopped*)
- Zombie

Identificadores de procesos

Durante su ejecución es posible que un proceso invoque o ejecute a otros procesos. Por tanto, el proceso que ejecuta a otro es considerado `proceso padre`. GNU/Linux también mantiene un registro del usuario y del grupo que han invocado a un proceso determinado. Los identificadores son:

- de proceso (`pid`)
- del padre (`ppid`)
- de usuario (`uid`)
- de grupo (`gid`)

Estos identificadores nos permitirán referirnos a cada elemento cuando se utilice un comando relacionado a procesos.

3.1.2. Visualizar identificadores de usuarios y grupos. `id`

Comando `id`. El comando **`id`** nos permite visualizar el id del usuario (`uid`) y grupos (`gid`) a los que pertenece. Si se omite el usuario entonces muestra la información del usuario actual.

Sintaxis:

```
id [opciones] [usuario]
```

Opciones:

`-u` Muestra únicamente la identificación del usuario actual (`uid`) en número.

`-g` Muestra únicamente la identidad del grupo actual (`gid`) como un número

Ejemplo 3.1: Obtener el identificador (uid) de un usuario y los identificadores de los grupos (gid) a los que pertenece.

```
1 $ id
2 uid=1001(rogelio) gid=1002(rogelio) grupos=1002(rogelio)
3 $ id rogelio
4 uid=1001(rogelio) gid=1002(rogelio) grupos=1002(rogelio)
```

En el Ejemplo 3.1, en la línea 1 se solicita la información del usuario actual. En la línea 3, se solicita la información del usuario `rogelio`.

Ejemplo 3.2: Obtener el uid del usuario actual. Obtener los gid de los grupos a los que pertenece el usuario actual.

```
1 $ id -u
2 1001
3 $ id -g
4 1002
```

El Ejemplo 3.2 muestra el uso del comando **`id`** con cada unos de los parámetros. La línea 1 permite obtener únicamente el identificador (`uid`) del usuario actual. La

línea 3 obtiene los identificadores de los grupos (`gid`) a los que pertenece el usuario actual. Las líneas 2 y 4 muestran el resultado de la ejecución de cada comando.

Ejemplo 3.3: Obtener únicamente el uid de un usuario en específico. Obtener el gid de los grupos a que pertenece un usuario.

```
1 $ id -u rogelio
2 1001
3 $ id -g rogelio
4 1002
```

El Ejemplo 3.3, línea 1 muestra cómo obtener el identificador (`uid`) del usuario rogelio. En la línea 2 se obtiene el `gid` de los grupos a los que pertenece el usuario rogelio.

3.1.3. Listado de procesos. `ps`, `pstree`, `top`

Los comandos **`ps`**, **`pstree`**, **`top`** nos permiten visualizar la lista de procesos en ejecución, así como información relevante de los mismos.

Comando `ps`. El comando **`ps`** muestra la lista de procesos activos del sistema y algunas de sus características: hora de inicio, uso de memoria, estado de ejecución, propietario y otros detalles. De manera predeterminada, sin opciones, el comando **`ps`** muestra lista los procesos creados por el usuario actual y asociados a la terminal de usuario.

Sintaxis:

```
ps [opciones]
```

Las **`opciones`** más frecuentes son:

`-a` Muestra los procesos creados por cualquier usuario y asociados a una Terminal.

`-e` Muestra todos los procesos.

`-l` Formato detallado (largo). Muestra la prioridad, el `PID` del proceso padre, entre

otros.

`-u` Formato de usuario. Incluye el usuario propietario del proceso y la hora de inicio.

`-U nombreusuario` Lista los procesos creados por el usuario `nombreusuario`.

`-U uid` Lista los procesos creados por el usuario con identificador `UID`.

`-x` Mostrar los procesos que no están asociados a ningún terminal del usuario. Útil para ver los «demonios» (programas residentes) no iniciados desde la Terminal.

`f` *forest*. Muestra en pantalla un árbol de procesos que permite identificar procesos padres e hijos.

Ejemplo 3.4: Muestra la lista de los procesos creados por el usuario actual y asociados a una terminal de usuario.

```
1 $ ps
2     PID TTY          TIME CMD
3    1930 pts/0    00:00:00 bash
4    1937 pts/0    00:00:00 ps
```

El Ejemplo 3.4 muestra la ejecución del comando **ps** sin parámetros. La línea 1 permite listar los procesos creados por el usuario actual y asociados a la terminal de usuario. La líneas 2-5 muestran la salida obtenida. La información se muestra en columnas. La Tabla 3.1 muestra la descripción de cada columna.

Tabla 3.1: Columnas mostradas por el comando **ps**.

Columna	Descripción
`PID`	Identificador único del proceso. Este datos nos permite identificar y controlar al proceso por medio del número de proceso (`PID`).
`TTY`	El nombre de la Terminal donde el proceso está ejecutándose. Esta columna es útil porque nos permitirá distinguir entre diferentes procesos que se ejecutan con el mismo nombre.
`TIME`	El tiempo total de procesador utilizado por el proceso. Con frecuencia esta columna es información útil para los administradores del sistema.
`CMD`	El comando que inició el proceso.

Ejemplo 3.5: Muestra la lista de los procesos creados por cualquier usuario y asociados a una terminal.

```
$ ps -a
    PID TTY          TIME CMD
   1393 tty2     00:00:00 Xorg
   1432 tty2     00:00:00 gnome-session-b
   1938 pts/0    00:00:00 ps
```

El Ejemplo 3.5 utiliza el parámetro -a para obtener un listado de los procesos creados por cualquier usuario y asociados a una terminal.

Ejemplo 3.6: Muestra la lista de todos los procesos activos en formato detallado.

```
$ ps -el
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0       1       0  0  80   0 - 41897 -      ?        00:00:00 systemd
1 S     0       2       0  0  80   0 -     0 -      ?        00:00:00 kthreadd
1 I     0       3       2  0  60 -20 -     0 -      ?        00:00:00 rcu_gp
...
```

El Ejemplo 3.6 utiliza dos parámetros: el parámetro `e` para listar todos los procesos activos y el parámetro `l` para indicar que sea un listado detallado.

En el Ejemplo 3.7 se obtiene una lista en forma de árbol (parámetro `f`) de los procesos asociados al usuario `rogelio`. Se utiliza el parámetro `-u` para especificar el usuario.

Comando `pstree`. Este comando muestra la jerarquía de los procesos mediante una estructura de árbol. Si se especifica el `PID` de un proceso, el árbol empezará desde ese proceso, de lo contrario el árbol empezará por el proceso **`init`** (`PID=1`) y mostrará todos los procesos del sistema. Si se especifica un usuario válido se mostrará la jerarquía de todos los procesos de ese usuario.

Sintaxis:

```
pstree [opciones] [PID | usuario]
```

Ejemplo 3.7: Muestra la lista, en forma de árbol, de todos los procesos activos asociados al usuario rogelio.

```
1 $ ps -u rogelio f
2     PID TTY      STAT   TIME COMMAND
3    1391 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env
     GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=
     ubuntu
4    1393 tty2     Sl+    0:01  \_ /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/
     user/1001/gdm/Xauthority -background none -noreset -keeptty -verbose 3
5    1432 tty2     Sl+    0:00  \_ /usr/libexec/gnome-session-binary --systemd --
     systemd --session=ubuntu
6 ...
```

Opciones:

-a Incluye en el árbol de procesos la línea de comandos que se usó para iniciar el proceso.

-p Incluye el `PID` de los procesos en el árbol.

Ejemplo 3.8: Uso del comando pstree.

```
1 $ pstree
2 systemd-+-ModemManager---2*[{ModemManager}]
3         ├─NetworkManager---2*[{NetworkManager}]
4         ├─ accounts-daemon---2*[{accounts-daemon}]
5         ├─ acpid
6         ...
```

El Ejemplo 3.8 ejecuta el comando **pstree** para obtener una lista de los procesos activos en una estructura de árbol. A partir de la línea 2 se muestra el resultado obtenido.

Comando top. El comando **top** ofrece una lista de los procesos, similar al comando **ps**, pero la salida se actualiza de manera predeterminada cada tres (3) segundos. El comando **top** es especialmente útil cuando es necesario observar el estado de uno o más procesos o comprobar los recursos (CPU, memoria, etc.) que consumen los

procesos. El comando **top** también muestra una información actualizada del estado de la computadora: cantidad de procesadores, memoria RAM (total, disponible y utilizada), cantidad de procesos en ejecución, entre otros.

Sintaxis:

```
top [opciones]
```

Opciones:

-i Ignora los procesos inactivos, listando únicamente los que utilizan recursos del sistema.

-d número Especifica la frecuencia de actualización de la pantalla en segundos. El valor **número** es un entero que establece la cantidad de segundos.

El comando **top** se ejecuta en modo interactivo lo que le permite recibir órdenes del usuario. La Tabla 3.2 muestra las órdenes interactivas más frecuentes, el usuario puede indicar estas órdenes durante la ejecución del comando **top**.

Tabla 3.2: Órdenes interactivas del comando **top**.

Tecla	Orden
h	*help*. Muestra una pantalla de ayuda.
q	*quit*. Salir
k	*kill*. Permite detener un proceso o enviarle una señal a un proceso.
s	*seconds*. Especifica el tiempo de actualización, en segundos, de la pantalla.
L	*Locate*. Busca una cadena en la lista de procesos. Se utiliza la tecla & (ampersand) para buscar la siguiente coincidencia.

El Ejemplo 3.9 muestra la información de salida obtenida al ejecutar el comando **top**. La información se divide en dos secciones: el encabezado y la lista de procesos. El encabezado (líneas 2-6) contiene información de rendimiento del sistema: cantidad de tareas y sus estados, uso de CPU, uso de memoria RAM y uso de memoria de intercambio. Las líneas 8 en adelante, muestran la información de los procesos organizadas en columnas.

Ejemplo 3.9: Información de salida obtenida al ejecutar el comando top.

```
$ top
top - 16:56:52 up 50 min,  1 user,  load average: 0.08, 0.22, 0.21
Tareas: 195 total,   1 ejecutar,  194 hibernar,    0 detener,    0 zombie
%Cpu(s):  2.2 rogelio,  0.0 sist,  0.0 adecuado, 97.8 inact,  0.0 en espera,
    0.
MiB Mem :   3923.8 total,   2098.7 libre,    901.2 usado,    923.9 búfer/caché
MiB Intercambio:   2048.0 total,   2048.0 libre,      0.0 usado.   2766.7 dispon

    PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU  %MEM     HORA+ ORDEN
   1721 rogelio  20   0  847748  53852  40576 S   6.2   1.3   0:05.88 gnome-t+
      1 root      20   0  167688  11308   8116 S   0.0   0.3   0:01.70 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.02 kthreadd
```

3.1.4. Envío de señales. Comandos: `kill`, `killall`

Los procesos están alertas para recibir señales enviadas por el sistema operativo o por los usuarios. Los comandos **`kill`** y **`killall`** se utilizan para enviar señales a un proceso. Cada señal tiene asociada un nombre y un número entero predefinidos conocidos por los procesos. Para enviar una señal a un proceso, es posible utilizar de manera indistinta el nombre o el número entero asociado. La Tabla 3.3 muestra el listado de las señales más frecuentes. Las señales marcadas en negritas, `SIGTERM` y `SIGKILL`, se utilizan para matar procesos. Existen más de 30 diferentes señales definidas en `GNU/Linux` [46], para ver la lista completa de señales usted puede ejecutar el comando **`kill -l`**.

Comando `kill`

El comando **`kill`** permite enviar una señal a un proceso especificando un único `PID`, o a varios procesos especificando una lista de `PID`.

Sintaxis:

```
kill [señal] [pids]
```

Tabla 3.3: Señales más frecuentes que pueden ser enviadas a los procesos.

Nombre	Entero	Descripción
`SIGHUP`	1	Colgar. Esta señal es enviada automáticamente cuando un módem se desconecta. También se usa por muchos demonios para forzar la relectura del archivo de configuración. Por ejemplo, en los procesos **`init`** y `inetd`.
`SIGINT`	2	Detener y desaparecer el proceso. Esta señal se envía con la secuencia de teclas Ctrl+C.
`SIGKILL`	**9**	Kill. Mata un proceso incondicionalmente e inmediatamente. Enviar esta señal es un método drástico de terminar el proceso, ya que no se puede ignorar.
`SIGTERM`	**15**	Terminar. Matar un proceso de forma controlada.
`SIGCONT`	18	Continuar. Cuando un proceso detenido recibe esta señal continúa su ejecución.
`SIGSTOP`	19	Parar, pero preparado para continuar. Esta señal se envía con la secuencia de teclas Ctrl+Z.

El parámetro `señal` es el valor entero de la señal o el nombre de la señal que enviaremos al proceso. La Tabla 3.3 muestra el listado de las señales más frecuentes. El nombre de la señal se puede especificar en minúsculas o mayúsculas; normalmente se especifica en mayúsculas. Si se omite el parámetro `señal` en el comando **`kill`**, de manera predeterminada se asigna el valor `SIGTERM`, es decir, la señal 15 que mata un proceso de forma controlada. El parámetro `PID` es el identificador del proceso al que se desea enviar la señal. Es posible obtener el valor del `PID` de un proceso determinado, utilizando el comando **`ps`**.

Ejemplo 3.10: Matar de manera controlada el proceso con PID 337.

```
1 $ kill -15 337
2 $ kill -SIGTERM 337
3 $ kill -s sigterm 337
```

En el Ejemplo 3.10, las instrucciones de las líneas 1-3 son equivalentes. En la línea 1 se envía un número entero como parámetro para enviar la señal de terminación. En

la línea 2 se envía el nombre de la señal de terminación, en mayúsculas. En la línea 3 se utiliza el parámetro `-s` (*signal*) y se especifica el nombre de la señal de terminación, en minúsculas.

Ejemplo 3.11: Matar incondicionalmente el proceso con PID 337.

```
1 $ kill -9 337
2 $ kill -SIGKILL 337
```

En el Ejemplo 3.11, las dos instrucciones son equivalentes. En la línea 1 la instrucción envía la señal `SIGKILL` utilizando el número entero asociado. En la línea 2 se envía la señal `SIGKILL` utilizando explícitamente el nombre de la señal.

Comando `killall`

Sintaxis:

```
killall [señal] nombre_proceso
```

Este comando es ligeramente diferente al comando `kill` por dos motivos; en primer lugar, utiliza el nombre de proceso en lugar del `PID`, y además le envía la señal a todos los procesos que tengan el mismo nombre. Por lo demás, su comportamiento es idéntico.

Ejemplo 3.12: Matar de manera controlada a todos los procesos llamados firefox

```
1 $ killall -SIGTERM firefox
2 $ killall -15 firefox
```

El Ejemplo 3.12 muestra el uso del comando `killall` para enviar la señal `SIGTERM` para matar de manera controlada a todos los procesos que tengan el mismo nombre. Las líneas 1 y 2 son equivalentes. En la línea 1 se utiliza el nombre de la señal. En la línea 2 se utiliza el número entero asociado a la señal `SIGTERM`.

En el Ejemplo 3.13 se envía la señal de matar incondicionalmente a todos los procesos llamados `firefox`. La línea 1 y 2 son equivalentes. En la línea 1 se utiliza

Ejemplo 3.13: Matar incondicionalmente a todos los procesos llamados firefox

```
1 $ killall -SIGKILL firefox
2 $ killall -9 firefox
```

el nombre de la señal. En la línea 2 se utiliza el número entero asociado a la señal `SIGKILL`.

3.1.5. Procesos en primer y segundo plano

Un proceso en primer plano (*foreground*) es un proceso con el cual el usuario puede interactuar. Si el usuario trabaja desde una Terminal, es el proceso en primer plano recibe las entradas de teclado. Cuando se ejecuta un comando en la Terminal, de manera predeterminada se ejecuta en primer plano. Para el usuario, trabajando en una Terminal, solo puede haber un proceso en primer plano.

Cuando un comando se ejecuta en primer plano, el *shell* ejecuta el comando y espera a que finalice su ejecución. Una vez que el comando ha finalizado, el *shell* escribe el *shell prompt* (indicador del *shell*) para indicar que está listo para recibir un nuevo comando de parte del usuario.

Cuando un comando se ejecuta en segundo plano (*background*), el usuario no tiene que esperar a que el comando finalice su ejecución para ejecutar otro comando desde la Terminal. El proceso ejecutado en segundo plano, se ejecuta sin necesidad de interacción con el usuario, no recibe ninguna entrada desde el teclado.

3.1.6. Enviar procesos a segundo plano. Uso de `&` y de `nohup`

Para ejecutar un comando o programa en segundo plano es necesario agregar **`&`** (carácter et o *ampersand*) al final de la instrucción. De esta manera, el proceso se ejecuta en segundo plano aunque sigue ligado a la Terminal (por ser el proceso padre);

si la Terminal se cierra, el proceso será interrumpido aunque no haya terminado de ejecutarse.

Sintaxis:

```
comando [opciones] &
```

Donde:

- **comando** puede ser cualquier comando válido para el *shell*.
- **opciones** son las opciones válidas para el comando incluido.

Ejemplo 3.14: Ejecutar un comando en segundo plano.

```
1 $ find / -type f -name "*.fasta" &
2 [1] 7200
3 $
```

El Ejemplo 3.14 muestra la ejecución en segundo plano del comando **find** para buscar archivos cuyo nombre terminen con `.fasta`. La línea 3 muestra que mientras el comando se ejecuta, se libera la Terminal y el usuario, en cualquier momento, puede ejecutar otro comando. Los mensajes de salida generados por el comando **find** se mostrarán en pantalla. La línea 2 muestra datos acerca del proceso: el número dentro de los corchetes se refiere al número de tarea, en este ejemplo, es la tarea `1`; el número `7200` es el **id** del proceso ejecutado en segundo plano.

Comando nohup

Una solución para desligar el proceso de la Terminal es utilizar el comando **nohup** (*no hang up*) para ejecutar el comando o programa. Utilizar el comando **nohup** permitirá que cuando cierre la Terminal o la sesión, el proceso continúe en ejecución.

Sintaxis:

```
$ nohup comando [opciones] [&]
```

Donde:

- **comando** puede ser cualquier comando válido para el *shell*.
- **opciones** son las opciones válidas para el comando incluido.

Ejemplo 3.15: Ejecutar una búsqueda de directorios que inicien con letra D. Se ejecutará el proceso en primer plano y desligado de la terminal.

```
1 $ nohup find / -type d -name "D*"
2 nohup: se descarta la entrada y se añade la salida a 'nohup.out'
```

En el Ejemplo 3.15, al utilizar el comando **nohup**, el comando **find** se ejecuta pero la entrada por teclado es omitida y la salida es redireccionada (enviada) al archivo **nohup** `.out`. Si la Terminal es cerrada durante la ejecución del proceso, el proceso continúa en ejecución al no estar ligado a la Terminal.

Ejemplo 3.16: Ejecutar una búsqueda de archivos. Se ejecutará el proceso en segundo plano y desligado de la Terminal. Obtener un listado únicamente de los directorios cuyo nombre inicie con la letra D.

```
1 $ nohup find / -type d -name "D*" &
2 [1] 2806
```

Observe que en el Ejemplo 3.16, el carácter `&` (et o *ampersand*) se agregó al final. El carácter `&` especifica la ejecución en segundo plano. Al utilizar el comando **nohup** ~, el comando **find** se ejecuta pero la entrada por teclado es omitida y la salida es redireccionada (enviada) al archivo **nohup**`.out`. Si la Terminal es cerrada durante la ejecución del proceso, el proceso continúa en ejecución al no estar ligado a la Terminal. La línea 2 indica el número de tarea y el número de proceso.

3.1.7. Enviar procesos a primer o segundo plano. Comandos: `bg`, `fg` y `jobs`

El control de tareas (*control job*) es una característica del sistema operativo GNU/-Linux que le permite al usuario trabajar con varios procesos. El usuario puede suspender la ejecución de procesos y continuar su ejecución posteriormente.

Comando `jobs`. El comando `jobs` se utiliza para listar los procesos asociados a la Terminal (de la sesión actual) que se están ejecutando en primer o segundo plano.

Sintaxis:

```
jobs [-l]
```

El parámetro `-l` agrega al listado el número de proceso.

Comando `bg`. El comando `bg` se utiliza para enviar una tarea a segundo plano.

Sintaxis:

```
bg [%numero-tarea]
```

Comando `fg`. El comando `fg` se utiliza para mover un tarea a primer plano.

Sintaxis:

```
fg [%numero-tarea]
```

El parámetro `numero-tarea` es un número entero que identifica a la tarea. Si se omite el parámetro `numero-tarea` entonces se ejecutará la más reciente tarea ejecutada.

El Ejemplo 3.17 muestra la ejecución de un comando, se suspende la ejecución y posteriormente se continúa con la ejecución en primer plano. La línea 1 ejecuta el editor nano en primer plano. Al ejecutar un proceso en primer plano, usted puede presionar las teclas Ctrl+Z (señal de `SIGSTOP`) para enviar el proceso a segundo plano. En ese momento el proceso se detiene, la Terminal muestra un mensaje de notificación (líneas 3 y 4). Si se desea que el proceso siga en ejecución en segundo

Ejemplo 3.17: Ejecutar nano, suspender su ejecución. Continuar con la ejecución de nano en primer plano.

```
1 $ nano
2 (presionar Ctrl+Z)
3 Use "fg" to return to nano.
4 [1]+  Stopped                 nano
5 $ fg %1
```

plano se puede utilizar el comando **bg**; si se desea que el proceso se ejecute en primer plano se puede utilizar el comando **fg**. Puede observar que en la línea 5 se reanuda la ejecución del proceso en primer plano.

Ejemplo 3.18: Ejecutar un proceso en segundo plano, mover su ejecución a primer plano, detener su ejecución y posteriormente continuar su ejecución en segundo plano.

```
1 $ ./prog.sh &
2 $ fg %1
3 (presione Ctrl+Z, para suspender la ejecución)
4 $ jobs
5 [1]+  Stopped                 ./prog.sh &
6 $ bg %1
```

En el Ejemplo 3.18, en la línea 1 se ejecuta un comando en segundo plano. En la línea 2 se mueve la ejecución del proceso a primer plano. Para continuar con el ejemplo puede suspender la ejecución del proceso al presionar Ctrl+Z. En la línea 4 se muestra el listado de tareas. Las instrucciones **bg** y **bg %+** son equivalentes a la instrucción de la línea 6, podría ejecutarse cualquiera de estas tres instrucciones. Finalmente, en la línea 6 se reanuda la ejecución del proceso, en segundo plano.

En el Ejemplo 3.19, la línea 1 ejecuta el comando **nano** en primer plano. Se suspende la ejecución del proceso presionando las teclas Ctrl+Z.

En línea 3, el segundo comando se ejecuta en primer plano, realiza una búsqueda de archivos con extensión `fasta`. Se suspende la ejecución del proceso presionando Ctrl+Z.

Ejemplo 3.19: Ejecutar dos comandos. Suspender la ejecución de los dos procesos para posteriormente elegir cuál de ellos se ejecuta en primer o segundo plano.

```
1  $ nano datos.txt
2  (presione Ctrl+Z, para suspender la ejecución)
3  $ find / -name *.fasta
4  (presionar Ctrl+Z)
5  $ jobs
6  [1]-  Stopped                 nano datos.txt
7  [2]+  Stopped                 find / -name *.fasta
8  $ fg %1
9  (presione Ctrl+X, para salir de nano)
10 $ jobs
11 [2]+  Stopped                 find / -name *.fasta
12 bg %2
```

En la línea 5, se muestra la lista de tareas. Observe que aparecen las dos tareas numeradas. El signo `+` se refiere a la más reciente tarea ejecutada y el signo `-` a la segunda más reciente tarea ejecutada. Es posible referirse a estas tareas con los signos en lugar de su correspodiente `numero-tarea`.

En la línea 8, se trae a primer plano la primera tarea. Las instrucciones `fg %-` y `%1` son equivalentes a la instrucción de la línea 8.

En la línea 10, se puede observar que solamente queda una tarea en el listado de tareas. Para continuar su ejecución, en segundo plano, se utiliza el comando `bg %2` en la línea 12. Las instrucciones `bg` y `bg %+` son equivalentes, podría ejecutarse cualquiera de estas tres instrucciones.

3.2. Manejo de scripts

El `bash` contiene un lenguaje de programación que permite realizar automatización de tareas repetitivas o complejas mediante un archivo que pueden incluir una secuencia de comandos que es ejecutada cada vez que se requiera.

Un **shell script** es un archivo que contiene uno más comandos del *shell*. Los comandos se ejecutan secuencialmente. El archivo *shell script* puede incluir cualquier comando que se interpreta y ejecuta correctamente desde el *shell*. Los archivos *shell script* se utilizan con el objetivo de automatizar tareas. Cuando se crea un archivo *shell script* se recomienda agregar la extensión `.sh`. Aunque GNU/Linux no requiere la extensión, es útil incluir la extensión `.sh` para que los humanos identifiquen con mayor facilidad el tipo de archivo al leer el nombre del archivo. Se recomienda que en la primera línea del archivo *shell script* se especifique en cuál programa se ejecutarán las instrucciones contenidas en el archivo. Una vez que se ha creado el archivo *shell script* con comandos en su interior es necesario asignarle permisos de ejecución.

3.2.1. Crear un archivo script (*shell script*)

Puede crear un nuevo archivo utilizando el editor de texto `nano`. El comando `nano script01.sh` le permitirá abrir el editor de texto `nano` para crear un nuevo archivo con el nombre `script01.sh`.

Ejemplo 3.20: script01.sh

```
1 #!/bin/bash
2 whoami
3 echo "¡bienvenido!"
```

En el Ejemplo 3.20, la línea 1 contiene la ruta al programa que interpretará o ejecutará el resto de las instrucciones, en este caso, es `bash`. La línea debe iniciar con `#!`, a esta línea se le conoce como *shebang* o *hashbang*. A partir de la línea 2, se incluyen las instrucciones que serán ejecutadas: `whoami` muestra el nombre del usuario actual, en la línea 3 el comando `echo` imprime una cadena.

En el Ejemplo 3.21, la línea 1 ejecuta el script. El bash muestra el error `command not found` (no se encontró la orden) porque no busca en el directorio actual; busca en

Ejemplo 3.21: Ejecución de un script en bash.

```
1 $ script01.sh
2 -bash: script01.sh: command not found
3 $ ./script01.sh
4 -bash: ./script01.sh: Permission denied
5 $ chmod u+x script01.sh
6 $ ./script01.sh
7 rogelio
8 ¡bienvenido!
9 $ /home/rogelio/script01.sh
10 rogelio
11 ¡bienvenido!
```

las rutas preconfiguradas un archivo llamado `script01.sh`, al no encontrarlo muestra el error (línea 2). En la línea 3, se ejecuta el script utilizando el punto-diagonal `./` para especificar que el script se ubica en el directorio actual. El bash muestra el error `Permission denied` (permiso denegado). Se requiere que el archivo cuente con permiso de ejecución. La línea 5 le otorga al script permisos de ejecución únicamente para el usuario. También se podría utilizar el comando **`chmod`** `+x script01.sh` para otorgar permisos al usuario propietario, al grupo y a otros usuarios. La línea 6 ejecuta nuevamente el script. El script se ejecuta correctamente y muestra el resultado en pantalla (líneas 7 y 8). La línea 9 es similar a la línea 6, solamente que utiliza la ruta absoluta (completa) del script. Las líneas 10 y 11 muestran el resultado en pantalla.

3.2.2. Configurar variables de entorno

Una variable es un espacio de memoria que permite almacenar uno o más valores. Las variables de entorno son un tipo especial de variable que definen el comportamiento del ambiente sobre cual el usuario está trabajando. La Tabla 3.4 muestra un listado de algunas variables de entorno. Para ver un listado completo de las variables de entorno puede usar el comando **`env`**.

Tabla 3.4: Listado de algunas variables de entorno importantes.

Variable	Descripción
`PATH`	Rutas de búsqueda de programas
`HOME`	Directorio inicial del usuario
`USER`	Nombre del usuario actual
`SHELL`	Ruta del *Shell* predeterminado
`PWD`	Directorio actual de trabajo

Al ejecutar la `Terminal` se lee el archivo de configuración `.bashrc` ubicado en la carpeta `home` del usuario. El archivo `.bashrc` contiene variables de entorno que controlan el comportamiento de la Terminal con respecto a la interacción del usuario.

Ejemplo 3.22: script02.sh. Crear un script que muestre el contenido de variables de entorno.

```
1 #!/bin/bash
2 echo "Usted ingresó con el usuario:" $USER
3 echo "Su nombre es:" $MINOMBRE
4 echo "Sus rutas de búsqueda son:" $PATH
5 echo "Su directorio inicial es:" $HOME
6 echo "Su shell es :" $SHELL
```

El Ejemplo 3.22 realiza la impresión en pantalla de las variables de entorno incluidas en la Tabla 3.4.

Crear variables de entorno. `export`

Es posible crear variables de entorno, además de las variables de entorno ya existentes. También es posible actualizar su valor. El comando **`export`** permite crear variables de entorno. **`export`** recibe como parámetros el nombre de la variable y el valor que le será asignado a la variable.

Sintaxis:

```
export NOMBREDEVARIABLE=valor
```

El NOMBREDEVARIABLE puede ser una cadena que contenga letras mayúsculas, dígitos o guión-bajo (_). El NOMBREVARIABLE debe iniciar con una letra o guión-bajo. Se recomienda elegir nombres de variables significativos o que sean fáciles de recordar. Ejemplos de nombres de variables: MINOMBRE, NOMBRESECUENCIA.

Ejemplo 3.23: Crear variables de entorno y mostrar su contenido.

```
1 $ export MINOMBRE=Rogelio
2 $ export NOMBREGEN=MIR17
3 $ echo $MINOMBRE
4 Rogelio
5 $ echo $NOMBREGEN
6 MIR17
```

En el Ejemplo 3.23, líneas 1 y 2 se definen variables, para asignar valores a las variables se utiliza el operador =. En las líneas 3 y 5 se muestra en pantalla el contenido de cada variable, observe que se utiliza el signo $ para obtener el valor de la variable. Estas variables son válidas en la instancia de la Terminal en la que fueron creadas. Si se desea que sean permanentes, entonces deben ser agregadas al archivo de configuración ~/.bashrc.

Ejemplo 3.24: Crear una variable de entorno permanente.

```
1 $ echo "export MINOMBRE=Rogelio" >> ~/.bashrc
2 $ source ~/.bashrc
```

En el Ejemplo 3.24, línea 1 se imprime una cadena que contiene el comando **export** para crear la variable MINOMBRE. Se agrega esa cadena al final del archivo de configuración .bashrc. El comando **export** también se podría agregar utilizando **nano** ~/.bashrc para agregar manualmente esa línea al final del archivo. En la línea 2 se lee y ejecuta el contenido del archivo .bashrc. Esta lectura y ejecución del archivo .bashrc sucede también cada vez que se abre una instancia de la Terminal.

3.2.3. Uso de parámetros

Un *shell script* puede recibir parámetros: información que puede utilizar para procesar y obtener el resultado esperado. Un *shell script* puede recibir hasta nueve (9) parámetros.

Ejemplo 3.25: script04.sh. El script recibe dos parámetros.

```
1 #!/bin/bash
2 echo "Hola, bienvenido" $1
3 echo "Hola, bienvenido $2"
```

El Ejemplo 3.25 muestra un *shell script* que puede recibir parámetros. El *shell script* recibe cada parámetro en las variables `$1, $2, $3 ... $9`. En las líneas 2 y 3 usa el comando **`echo`** para mostrar el contenido de los dos parámetros que recibió el `script04.sh`.

Ejemplo 3.26: Ejecución del script04.sh

```
1 $ ./script04.sh Rogelio sofia
2 Hola, bienvenido Rogelio
3 Hola, bienvenido Sofia
```

El Ejemplo 3.26 muestra la ejecución del `script04.sh` y el resultado obtenido. En la línea 1 se ejecuta el archivo de script. En las líneas 2 y 3 se muestra el resultado.

El Ejemplo 3.27 es un script que recibe recibe como parámetro la ruta a un archivo y obtiene como salida la cantidad de líneas, palabras y caracteres del archivo proporcionado. El script utiliza tres variables para almacenar cada uno de los valores: `CANTIDADLINEAS`, `CANTIDADPALABRAS` y `CANTIDADCARACTERES`.

El Ejemplo 3.28, línea 1 ejecuta el `script05.sh` con un parámetro. Las líneas 2 a la 5 muestran el resultado obtenido para el archivo `miarchivo.txt`

Ejemplo 3.27: script05.sh. El script recibe como parámetro la ruta a un archivo. El script obtiene la cantidad de líneas, palabras y caracteres del archivo proporcionado.

```
1 #!/bin/bash
2 CANTIDADLINEAS=$(wc -l $1)
3 CANTIDADPALABRAS=$( wc -w $1)
4 CANTIDADCARACTERES=$(wc -c $1)
5 echo "Características del archivo" $1
6 echo "Número de líneas: " $CANTIDADLINEAS
7 echo "Número de palabras: " $CANTIDADPALABRAS
8 echo "Número de caracteres: " $CANTIDADCARACTERES
```

Ejemplo 3.28: Ejecución del script05.sh.

```
1 $ ./script05.sh miarchivo.txt
2 Características del archivo miarchivo.txt
3 Número de líneas:  3 miarchivo.txt
4 Número de palabras:  10 miarchivo.txt
5 Número de caracteres:  40 miarchivo.txt
```

3.2.4. Operadores

El lenguaje de programación incluido en **bash** permite realizar operaciones aritméticas, lógicas y relacionales.

Operadores aritméticos

La Tabla 3.5 muestra los operadores aritméticos utilizados en **bash** para ejecutar operaciones aritméticas. El resultado de una operación aritmética es un valor numérico.

El Ejemplo 3.29, líneas 1-3 realiza una operación de suma de dos valores enteros predefinidos. Observe que en la línea 4 se utiliza el operador aritmético de suma. El Ejemplo 3.30 muestra la ejecución del `script06.sh` con dos números enteros como parámetros.

Ejemplo 3.29: script06.sh.

```
1 #!/bin/bash
2 num1=10
3 num2=20
4 misuma=$(($num1+$num2))
5 echo "La suma es: "$misuma
```

Ejemplo 3.30: Ejecución del script06.sh.

```
1 $ ./script06.sh
2 La suma es: 30
```

Tabla 3.5: Listado de operadores aritméticos

Operador	Descripción
`+`	Suma
`-`	Resta
`*` (asterisco)	Multiplicación
`/`	División
`%`	Resto o residuo
`**` (doble asterisco)	Potenciación

Ejemplo 3.31: script07.sh

```
1 #!/bin/bash
2 num1=$1
3 num2=$2
4 misuma=$(($num1+$num2))
5 echo "La suma es: "$misuma
```

El Ejemplo 3.31 realiza una operación de suma de dos valores enteros cualquiera que recibe como parámetros. La línea 2 y 3 asigna el valor de los parámetros recibidos a las variables `num1` y `num2` respectivamente. Observe que en la línea 4 se utiliza el operador aritmético de suma.

El Ejemplo 3.32 muestra la ejecución del `script07.sh` con dos números enteros como parámetros.

Ejemplo 3.32: Ejecución del script07.sh.

```
$ ./script07.sh 10 20
La suma es: 30
```

Operadores relacionales para valores numéricos

Los operadores relacionales permiten evaluar la relación que existe entre valores. Los operadores relacionales son binarios, es decir, evalúan dos operandos (valores). El resultado de la evaluación es `true` (verdadero) o `false` (falso). La Tabla 3.6 muestra el listado de los operadores relacionales para valores numéricos.

Tabla 3.6: Listado de operadores relacionales para valores numéricos

Expresión	Resultado `true` si
`entero1 -eq entero2`	los enteros son iguales. `-eq` significa *equal to*.
`entero1 -ne entero2`	los enteros son distintos. `-ne` significa *not equal to*.
`entero1 -gt entero2`	`entero1` mayor que `entero2`. `-gt` signfica *greater than*.
`entero1 -ge entero2`	`entero1` mayor o igual que `entero2`. `-ge` significa *greater than or equal to*.
`entero1 -lt entero2`	`entero1` menor que entero2. `-lt` significa *less than*.
`entero1 -le entero2`	`entero1` menor o igual que `entero2`. `-le` significa *less than or equal to*.

Operadores relacionales para cadenas

La Tabla 3.7 muestra el listado de operadores relacionales para cadenas. Observe que los operadores `=` y `!=` requieren espacio antes y después del operador.

Tabla 3.7: Listado de operadores relacionales para cadenas.

Expresión	Resultado `true` si
`cadena`	`cadena` contiene al menos un carácter.
`-z cadena`	la longitud de la `cadena` es 0.
`-n cadena`	la longitud de la `cadena` no es 0.
`cadena1 = cadena2`	las cadenas son iguales.
`cadena1 != cadena2`	las cadenas son distintas.

3.2.5. Estructuras de selección (condiciones)

En un script las instrucciones se ejecutan en orden que fueron escritas, una después de otra, es decir, en un script se realiza una ejecución secuencial. La estructura de *selección simple* permite decidir si se debe ejecutar un bloque de comandos alternativo u opcional. La estructura de selección simple permite evaluar una expresión, únicamente si el resultado de la evaluación es `verdadero` entonces se ejecuta un bloque de comandos alternativo.

Ejemplo 3.33: script08.sh

```
1 #!/bin/bash
2 edad=$1
3 if [ $edad -gt 18 ]; then
4   echo "Eres un adulto"
5 fi
```

Ejemplo 3.34: Ejecución del script08.sh

```
1 $ ./script08.sh 10
2 $ ./script08.sh 20
3 Eres un adulto
```

El Ejemplo 3.33 muestra un script que utiliza una estructura de selección simple. Recibe un parámetro que lo asigna a la variable `edad`. Evalúa el valor de `edad`, únicamente si es mayor que `18` entonces imprime el mensaje "Eres un adulto". El Ejemplo 3.34 ejecuta el script en dos ocasiones. En la línea 1 lo ejecuta con el parámetro `10` al no cumplir con la expresión relacional no se muestra el mensaje. En cambio en la línea 2 se ejecuta el script con el parámetro 20 y se imprime el mensaje "Eres un adulto".

El Ejemplo 3.35 muestra la utilización de la estructura de selección simple y operadores relacionales. El Ejemplo 3.36 ejecuta el script, usted puede analizar el

Ejemplo 3.35: script09.sh

```
1  #!/bin/bash
2  metadato01="Clostridium"
3  metadato02="clostridium"
4  echo $metadato01
5  if [ $metadato01 ]
6  then
7    echo $metadato01": no está vacía."
8  fi
9  if [ -z $metadato01 ]
10 then
11   echo $metadato01": la longitud de la cadena es 0."
12 fi
13 if [ -n $metadato01 ]
14 then
15   echo $metadato01": la longitud de la cadena NO es 0."
16 fi
17 if [ $metadato01 = "Clostridium" ]
18 then
19   echo "$metadato01 y Clostridium: las cadenas son iguales."
20 fi
21 if [ $metadato01 != $metadato02 ]
22 then
23   echo "$metadato01 y ${metadato02}: las cadenas son diferentes."
24 fi
```

resultado. En el Ejemplo 3.36, la evaluación de las líneas 5, 13, 17 y 21 da como resultado `verdadero` y ejecuta el comando **`echo`** que está en el interior de la estructura de selección respectiva. Al evaluar la línea 9, se obtiene un resultado `falso` y no se ejecuta el comando **`echo`** que está dentro de la estructura de selección.

La estructura de *selección doble* permite elegir entre dos opciones. La estructura de selección doble permite ejecutar una determinada secuencia de comandos si el criterio evaluado es `verdadero` y otra secuencia de comandos, si el criterio es `falso`.

Observe el Ejemplo 3.37, la estructura de selección aparece en las líneas 3, 5 y 7. Si se cumple la condición que la variable `edad` es menor que 18, se obtiene un resultado

Ejemplo 3.36: Ejecución del script09.sh

```
$ ./script09.sh
Clostridium
Clostridium: no está vacía.
Clostridium: la longitud de la cadena NO es 0.
Clostridium y Clostridium: las cadenas son iguales.
Clostridium y clostridium: las cadenas son diferentes.
```

Ejemplo 3.37: script10.sh

```
#!/bin/bash
edad=$1
if [ $edad -lt 18 ]; then
  echo "Eres un niño"
else
  echo "Eres un adulto"
fi
echo "¡Hasta pronto!"
```

`verdadero` y se ejecuta la línea 4; de no cumplirse, se obtiene un resultado `falso` y se ejecuta la línea 6. La línea 7 finaliza la estructura de selección.

Ejemplo 3.38: Ejecución del script10.sh

```
$ ./script10.sh 20
Eres un adulto
¡Hasta pronto!
$ ./script10.sh 10
Eres un niño
¡Hasta pronto!
```

El Ejemplo 3.38 ejecuta dos veces el `script10.sh`. En la línea 1, envía al script el parámetro 20. En la línea 4, ejecuta nuevamente el script con el parámetro 10.

Es posible que algunos problemas presenten la situación de elegir una de tres o más opciones. Para ello, se puede insertar una estructura de selección dentro de otra, a esto se le conoce como estructura de *selección doble anidada*.

Ejemplo 3.39: script11.sh

```
1  #!/bin/bash
2  edad=$1
3  if [ $edad -lt 18 ]; then
4    echo "Eres un niño"
5  elif [ $edad -lt 65 ]; then
6    echo "Eres un adulto"
7  else
8    echo "Eres un adulto mayor"
9  fi
10 echo "¡Hasta pronto!"
```

Ejemplo 3.40: Ejecución del script11.sh

```
1 $ ./script11.sh 10
2 Eres un niño
3 ¡Hasta pronto!
4 $ ./script11.sh 20
5 Eres un adulto
6 ¡Hasta pronto!
7 $ ./script11.sh 70
8 Eres un adulto mayor
9 ¡Hasta pronto!
```

El Ejemplo 3.39 muestra un script que utiliza estructura de selección doble anidada para elegir una de las tres opciones. En la línea 2 se asigna el primer parámetro a la variable `edad`. En la línea 3 evalúa si el valor de `edad` es menor a `18`, de obtener un resultado `verdadero` ejecuta la línea 4 y continúa en la línea 9, de lo contrario continúa en la línea 5. Evalúa la línea 5, es decir si edad es menor que `65`. Si el resultado es `verdadero`, entonces continúa en la línea 6, de lo contrario continúa en la línea 7. En la línea 7 no hay expresión a evaluar, por tanto se ejecuta la línea 8. Continúa la ejecución en las líneas 9 y 10.

3.2.6. Estructuras repetitivas (bucles o iteraciones)

Las estructuras repetitivas permiten ejecutar un bloque de instrucciones repetidamente mientras una condición permanezca con resultado verdadero. En este documento se incluyen las estructuras repetitivas `while` y `for`.

Estructura repetitiva `while`. Sintaxis 01:

```
while [ condition ]
do
  instrucción1
  instrucción2
  ...
  instrucciónN
done
```

Ejemplo 3.41: script12.sh

```
1 #!/bin/bash
2 i=1
3 while [ $i -le 1000 ]
4 do
5   echo "Número:" $i
6   i=$(( $i+1 ))
7 done
```

La estructura repetitiva `while` ejecuta un bloque de código mientras la condición sea `verdadera`. El Ejemplo 3.41 imprime los números del 1 al 1000. En la línea 3, se evalúa la condición, de obtener resultado `verdadero` entonces se ejecuta el bloque de código de las líneas 4 a la 7. Al ejecutar la línea 7, se regresa a la línea 3, se evalúa nuevamente la condición, de obtener resultado `verdadero`, se ejecuta nuevamente el bloque de código de las líneas 4 a la 7. Continuará la ejeucución del script repetidamente mientras la evaluación de la condición en la línea 2 se mantenga con resultado `verdadero`. La estructura repetitiva termina cuando la variable `i` tiene un valor de `1001`.

Ejemplo 3.42: Ejecución del script12.sh

```
$ ./script12.sh
Número: 1
Número: 2
Número: 3
...
Número: 999
Número: 1000
```

El Ejemplo 3.42 muestra el resultado obtenido al ejecutar el `script12.sh`. Se muestran los números del 1 al 1000.

Estructura repetitiva `while`. Sintaxis 02:

```
while IFS= read -r linea
  do
    instrucción1 para $linea
    instrucción2 para $linea
    ...
    instrucciónN
  done < "/path/to/filename"
```

Ejemplo 3.43: script13.sh

```
#!/bin/bash
archivo=/etc/resolv.conf
while IFS= read -r linea
do
  # mostrar la línea que está almacenada en la variable $linea
  echo $linea
done < "$archivo"
```

En el Ejemplo 3.43 se lee el archivo de entrada `/etc/resolv.conf` línea por línea hasta alcanzar el final del archivo. En esta estructura repetitiva `while` se realizan iteraciones mientras sea posible leer una línea del archivo de entrada. En la línea 3, el comando `read` lee una línea, si es posible leer la línea regresa un valor `verdadero`; de lo contrario, regresa un valor `falso`.

Estructura repetitiva `for`

La estructura repetitiva **`for`** permite ejecutar un bloque de código un número definido de iteraciones. La cantidad de interaciones se conoce de antemano.

Ejemplo 3.44: script14.sh

```
1 #!/bin/bash
2 for ((i=1; i<=1000; i++));
3 do
4   echo "Número:" $i
5 done
```

En el Ejemplo 3.44 observe que la primera vez que se ejecuta la línea 2, la variable `i` toma el valor de 1; se evalúa la condición `i<=1000`, de ser `verdadera` se procede a ejecutar el bloque de código de las líneas 3 a la 5. Al ejecutarse la línea 5, se regresa a ejecutar nuevamente la línea 2. En esta ocasión, en la línea 2, primero se incrementa la variable `i` y luego se evalúa la condición, al obtener resultado `verdadero` se ejecuta nuevamente el bloque de código de las líneas 3 a la 5. Continuará ejecutándose repetidamente mientras la evaluación de la condición en la línea 2 se mantenga con resultado `verdadero`. La estructura repetitiva termina cuando la variable `i` tiene un valor de `1001`. El Ejemplo 3.45 muestra el resultado obtenido al ejecutar el `script14.sh`.

Ejemplo 3.45: Ejecución del script14.sh

```
1 $ ./script14.sh
2 Número: 1
3 Número: 2
4 Número: 3
5 ...
6 Número: 999
7 Número: 1000
```

En la estructura repetitiva utilizada en el Ejemplo 3.46, en la línea 1, primero se obtiene un listado de números del 1 al 1000 al ejecutar la instrucción `{1...1000}`.

Ejemplo 3.46: script15.sh

```
1 #!/bin/bash
2 for i in {1..1000}
3 do
4   echo "Numero:" $i
5 done
```

También en la línea 1, a la variable `i` se le asigna el valor del primer elemento de la lista. Se ejecuta el bloque de código de las líneas 3 a la 5. Al ejecutarse la línea 5, se regresa a ejecutar nuevamente la línea 2. Si aún quedan elementos en la lista, se asigna a la variable `i` el siguiente elemento de la lista. Continuará ejecutándose repetidamente mientras la lista contenga elementos, el resultado se puede observar en el Ejemplo 3.47.

Ejemplo 3.47: Ejecución del script15.sh

```
1 $ ./script15.sh
2 Número: 1
3 Número: 2
4 Número: 3
5 ...
6 Número: 999
7 Número: 1000
```

3.3. Extracción de Información - Filtros

Un comando que recibe datos de la entrada estándard, modifica o procesa estos datos de entrada y envía el resultado a la salida estándard se considera un *filtro*. Un uso común de los filtros es reestructurar los datos para mostrarlos en la salida estándard.

En este documento utilizaremos los filtros para visualizar información de interés, es decir, para extraer la información relevante de conjuntos de datos (*dataset*).

Figura 3.1: Ciclo vital de la metodología y el modelo de proceso CRISP-DM según la Guía IBM SPSS Modeler CRISP-DM Guide. Créditos de imagen para Kenneth Jensen, CC BY-SA 3.0, vía Wikimedia Commons

3.3.1. Preparación de datos. Comandos: `file` , `iconv`, `tr`

En bioinformática, es posible obtener una gran cantidad de datos derivados de la secuenciación genómica, proteómica, transcriptómica, metabolómica, metagenómica, epigenómica; de estudios clínicos y epidemiológicos, entre otros.

Para los proyectos dedicados a manipular y procesar grandes cantidades de datos para extraer valor, como los proyectos bioinformáticos, se recomienda apegarse a la metodología y modelo de proceso CRISP-DM (*Cross-Industry Standard Process for Data Mining*), propuesto por la industria. CRISP-DM es un estándar *de facto* [59] para proyectos de ciencia de datos. La Figura 3.1 muestra el ciclo de vida de un proyecto según CRISP-DM.

De las seis (6) fases contempladas en CRISP-DM, las primeras dos fases, la fase de comprensión de los datos y de preparación de los datos, pueden realizarse con comandos filtros de GNU/Linux. Los filtros están incluidos en esta sección del documento.

Codificación UTF-8. Comando `file`, `iconv`

Para almacenar información en archivos se requiere de una codificación para que los caracteres (símbolos) sean representados con un valor numérico.

`ASCII` fue el primer estándar de codificación de caracteres. ASCII definió 128 caracteres diferentes que podrían usarse en Internet: números (0-9), letras inglesas (A-Z) y algunos caracteres especiales como `! $ + - ( ) @ < >`. [95]. En la Tabla 3.8 muestra algunas de las codificaciones actuales más comunes.

Actualmente, algunas de las codificaciones que permiten almacenar caracteres en idiomas extranjeros (diferentes al inglés) son `ISO Latin 1 (iso-8859-1)`, `Shift JIS` y `UTF-8`. Estas codificaciones utilizan un byte o múltiples bytes para almacenar cada carácter (símbolo).

Tabla 3.8: Codificaciones de caracteres y la cantidad de caracteres que pueden representar [95].

Codificación	Cantidad de caracteres
`ASCII`	128 códigos de carácter.
`ISO-8859-1`	256 códigos de caracteres.
`ANSI (Windows-1252)`	288 códigos de carácter. Los mismos que la codificación ISO-8859-1 más 32 códigos de carácter extras.
`JIS X 0213`	es la versión más reciente de `Shift JIS`, puede representar 11,233 caracteres.
`UTF-8`	Incluye códigos para casi todos los carácteres y símbolos del mundo, puede representar 1,112,064 caracteres.

Una primera acción recomendada durante la «Fase 1 Preparación de datos» de `CRISP-DM` es verificar la codificación de los archivos del conjunto de datos a utilizar. Se recomienda utilizar la codificación `UTF-8` porque es la codificación más frecuente para contenidos en idiomas diferentes al inglés publicados en Internet [32]. La codificación `UTF-8` puede representar 1,112,064 caracteres; casi todos los caracteres y símbolos utilizados en el mundo.

Si usted ha obtenido el conjunto de datos de Internet y estos están codificados en `ASCII` o en `ISO Latin 1 (iso-8859-1)` es posible que sea necesario modificar la codifica-

ción, de cada uno de los archivos del conjunto de datos, a `UTF-8`. La codificación `UTF-8` es necesaria, ya sea que usted desee agregar algunos caracteres o símbolos del español en el conjunto de datos; o que el software que utilizará para el análisis tenga como requerimiento que el conjunto de datos utilice la codificación `UTF-8` específicamente.

Comando `file`

El comando **`file`** obtiene una descripción de tipo del archivo especificado como parámetro. La descripción incluye el tipo de archivo y la codificación utilizada.

Sintaxis:

```
file [opciones] archivo
```

Opciones:

`-i` Muestra el tipo de codificación.

Ejemplo 3.48: Obtener la descripción de tipo del archivo ecolli.fasta.

```
1 $file GCF_ecolli.fasta
2 GCF_ecolli.fasta: ASCII text
```

En el Ejemplo 3.48, la línea 1 ejecuta el comando **`file`** para obtener la descripción de tipo del archivo `GCF_ecolli.fasta`. En la línea 2, se muestra que el archivo `GCF_ecolli.fasta` es de tipo `ASCII text`, esto que indica que el archivo contiene texto sin formato.

Ejemplo 3.49: Obtener el nombre de la codificación que utiliza el archivo ecolli.fasta.

```
1 $file -i GCF_ecolli.fasta
2 GCF_ecolli.fasta: text/plain; charset=us-ascii
```

En el Ejemplo 3.49, la línea 1 ejecuta el comando **`file`** con el parámetro `-i` para obtener la codificación del archivo `GCF_ecolli.fasta`. La línea 4 indica que la codificación es `us-ascii`.

Comando `iconv`. El comando **`iconv`** convierte texto de una codificación a otra.

Sintaxis:

```
iconv [opciones] [-f codificación] [-t codificación] [archivo ...]
```

Opciones:

`-f codificación` es la codificación actual (*from encoding*) del archivo.

`-t codificación` es la codificación a la que se desea convertir (*to encoding*) el archivo.

`-c` Se descartan silenciosamente los caracteres que no pueden ser convertidos, en lugar de generar un error de conversión.

`-s` Se omiten los mensajes de error acerca de caracteres inválidos o incovertibles, pero el texto de entrada permanece sin modificaciones.

`-l` Listar las codificaciones disponibles.

Ejemplo 3.50: Convertir la codificación de un archivo a UTF-8, mostrar el resultado en pantalla.

```
1 $ file -i GCF_ecolli.fasta
2 GCF_ecolli.fasta: text/plain; charset=iso-8859-1
3 $ iconv -f iso-8859-1 -t utf-8 GCF_ecolli.fasta
4 >NC_000913.3 Escherichia coli str. K-12 substr. MG1655. Creación: NCBI
5 AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTTCTGAACTG
6 GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGCGCACAGAC
7 ...
```

Observe que en el Ejemplo 3.50, la línea 1 ejecuta el comando **`file`** para obtener la codificación del archivo `GCF_ecolli.fasta`. En la línea 2 el resultado indica que la codificación (*charset*) es `iso-8859-1`. En la línea 3, se realiza la conversión de codificación `iso-8859-1` a `utf-8`. De la línea 4 a la 6, el resultado se muestra en pantalla.

En el Ejemplo 3.51, en la línea 3 ejecuta el comando **`iconv`**. Se utiliza el direccionamiento `>` para almacenar el resultado en el archivo `GCF_ecolli-v02.fasta`.

Ejemplo 3.51: Convertir la codificación de un archivo iso-8859-1 a UTF-8 y almacenar el resultado en otro archivo.

```
1 $ file -i clostridium.csv
2 clostridium.csv: application/csv; charset=iso-8859-1
3 $ iconv -f iso-8859-1 -t utf-8 clostridium.csv > clostridium02.csv
4 $ file -i clostridium02.csv
5 clostridium02.csv: application/csv; charset=utf-8
```

3.3.2. Visualización de información. Comandos: `tr`, `nl`, `cat`, `column`

Comando `tr`. El comando `tr` convierte los caracteres de la entrada. Realiza la sustitución carácter por carácter.

Sintaxis:

```
tr [opciones] cadena1 [cadena2] [< archivo]
```

Opciones:

`-d` elimina los caracteres indicados en la `cadena1`.

`-s` por cada carácter indicado en `cadena1` busca cadenas de caracteres idénticos y las reemplaza por una única aparición del carácter.

Ejemplo 3.52: Convierte el carácter a por A.

```
1 $ tr a A
2 $ tr a A < GCF_ecolli.fasta
```

En el Ejemplo 3.52, línea 1 se ejecuta el comando `tr` realiza la conversión recibiendo el texto que le es proporcionado del teclado. En la línea 2 el comando `tr` recibe como entrada el contenido del archivo `GCF_ecolli.fasta`, convierte `a` por `A` y lo muestra en pantalla.

El Ejemplo 3.53 utiliza el comando `tr` para eliminar las vocales del archivo de entrada `prueba.txt`.

Ejemplo 3.53: Borra todas las vocales del archivo.

```
$ tr -d aeiou < prueba.txt
```

Ejemplo 3.54: Borra el código de nucleótido N.

```
$ tr -d N < GCF_ecolli.fasta
```

El Ejemplo 3.54 utiliza el comando **tr** para eliminar el carácter `N` del archivo de entrada `GCF_ecolli.fasta`

Ejemplo 3.55: Cambia múltiples espacios por uno solo.

```
$ tr -s " " < cat prueba.txt
```

El Ejemplo 3.55 utiliza el comando **tr** para buscar cadenas con múltiples espacios y las reemplaza por una único espacio.

Comando nl. El comando **nl** numera las líneas que recibe de un archivo o del teclado.

Sintaxis:

```
nl [opciones] [archivo]
```

Opciones:

`-v` Asigna el número inicial de línea. A partir de ahí se incrementa de uno en uno a menos que se indique un incremento diferente con el parámetro `-i`.

`-i` Asigna el número de incremento en cada línea.

En el Ejemplo 3.56 se numeran las líneas del archivo `GCF_ecolli.fasta`. De manera predeterminada el comando **nl** inicia la numeración en 1, con incrementos de 1.

El Ejemplo 3.57 permite numerar las líneas del archivo `GCF_ecolli.fasta` iniciando la numeración en diez (10), con incrementos de uno (1).

Ejemplo 3.56: Numerar las líneas del archivo GCF_ecolli.fasta.

```
$ nl GCF_ecolli.fasta
     1 >NC_000913.3 Escherichia coli str. K-12 substr. MG1655, complete genome
     2 AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTT
     3 GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAG
     ...
```

Ejemplo 3.57: Numerar las líneas del archivo GCF_ecolli.fasta, iniciando con la numeración en diez (10).

```
$ nl -v10 GCF_ecolli.fasta
    10 >NC_000913.3 Escherichia coli str. K-12 substr. MG1655, complete genome
    11 AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTT
    12 GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGC
    ...
```

El Ejemplo 3.58 permite numerar las líneas del archivo `GCF_ecolli.fasta` iniciando la numeración en cinco (5), con incrementos de dos (2).

Comando `cat`. El comando **`cat`** hace referencia al término en inglés *concatenate*. **`cat`** nos permite visualizar en la salida estándar (en pantalla) el contenido de uno o más archivos que almacenan texto sin formato.

Sintaxis:

```
cat [opciones] [archivo]
```

Opciones:

`-u` Muestra únicamente la identificación del usuario actual (`uid`) en número.

`-g` Muestra únicamente la identidad del grupo actual (`gid`) como un número

El Ejemplo 3.59 muestra el contenido del archivo `GCF_ecolli.fasta`, pantalla por pantalla.

El Ejemplo 3.60, línea 1 muestra pantalla por pantalla, secuencialmente y según el orden especificado, el contenido de los archivos especificados: `GCF_ecolli.fasta` y `prueba.txt`

Ejemplo 3.58: Numerar las líneas del archivo GCF_ecolli.fasta iniciando con la numeración en diez (10) e incrementando la numeración de cada línea en dos (2).

```
$ nl -v 5 -i 2 GCF_ecolli.fasta
    5 >NC_000913.3 Escherichia coli str. K-12 substr. MG1655, complete genome
    7 AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTT
    9 GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGC
    ...
```

Ejemplo 3.59: Mostrar el contenido del archivo, pantalla por pantalla.

```
$ cat GCF_ecolli.fasta
>NC_000913.3 Escherichia coli str. K-12 substr. MG1655, complete genome
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTTCTGAACTG
GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGCGCACAGAC
...
```

En el Ejemplo 3.61, la línea 1 muestra pantalla por pantalla, secuencialmente y según el orden especificado, el contenido de los archivos. El operador de direccionamiento `>` permite que el contenido de los archivos `GCF_ecolli.fasta` y `prueba.txt` sea almacenado en el archivo `concatenar.txt`.

El Ejemplo 3.62 muestra pantalla por pantalla, secuencialmente y según el orden especificado, el contenido de los archivos `GCF_ecolli.fasta` y `prueba.txt` que es agregado al final del archivo `concatenar.txt`.

Si el archivo `concantenar.txt` no existe, lo crea y agrega el contenido indicado. En cambio, si el archivo `concatenar.txt` ya existe entonces agrega el contenido al final del archivo.

Comando `column`. El comando `column` permite formatear la entrada en múltiples columnas. El separador de campos de la entrada es el carácter espacio a menos que se especifique otro.

Sintaxis:

```
column [opciones] [archivo]
```

Ejemplo 3.60: Mostrar el contenido de dos archivos.

```
1 $ cat GCF_ecolli.fasta prueba.txt
```

Ejemplo 3.61: Almacenar el contenido de dos archivos de entrada en un archivo.

```
1 $ cat GCF_ecolli.fasta prueba.txt > concatenar.txt
```

Opciones:

`-s` especifica el o los caracteres que separan las columnas.

`-t` determina la cantidad de columnas de la entrada para mostrar una tabla.

El Ejemplo 3.63, la línea 1 ejecuta el comando **`column`** para mostrar los datos en forma de tabla especificando que la coma (`,`) es el separador. En la línea 2 se obtiene un error debido a la codificación. Se debe convertir el archivo a la codificación `utf-8`. En la línea 3 se ejecuta el comando **`column`** pero esta vez se especifica un archivo que cuenta con codificación `utf-8`. En la línea 4 se muestra el texto en forma de tabla.

3.3.3. Extracción de información. Comandos: `sort`, `uniq`, `cut`, `grep`

En esta sección se incluyen comandos de tipo filtro que permiten extraer información de uno o más archivos.

Comando `sort`. Permite ordenar los registros o líneas de uno o más archivos.

La ordenación se puede hacer por línea, por el primer campo de la línea o por un campo distinto al primero en el caso de archivos estructurados.

De manera predeterminada, el comando **`sort`**: a) realiza un ordenamiento alfabético, b) realiza un ordenamiento ascendente y c) asume que los campos están separados por espacio en blanco.

Sintaxis:

```
sort [opciones] [archivo]
```

Ejemplo 3.62: Añadir al final de un archivo, el contenido de otros dos archivos.

```
$ cat GCF_ecolli.fasta prueba.txt >> concatenar.txt
```

Ejemplo 3.63: Obtener el identificador (uid) de un usuario y el identificador del grupo (gid) o grupos a los que pertenece.

```
$ column -s, -t clostridium.csv
column: Invalid or incomplete multibyte or wide character
$ column -s, -t clostridium02.csv
grupo_organismo          aislamiento     fecha_creación
Clostridium botulinum    PDT001099917.1  2021-07-30
Clostridium perfringens  PDT001116478.1  2021-08-24
Clostridium perfringens  PDT000797115.1  2020-07-27
...
$ column -s, -t clostridium02.csv
```

Opciones:

`-r` Ordenar en orden inverso (*reverse*), en orden descendente.

`-k 9` Especifica el campo sobre el cual se realizará el ordenamiento. El `9` se refiere al número de campo.

`-t` Especificar el carácter utilizado como separador de campos (o columnas).

`-n` Ordena considerando un orden numérico.

Ejemplo 3.64: Ordena alfabéticamente por línea ascendente.

```
$ sort clostridium02.csv
```

El Ejemplo 3.64 muestra la ejecución del comando **`sort`**. De manera predeterminada **`sort`** ordena por línea, alfabética y ascendentemente.

El Ejemplo 3.65 realiza un ordenamiento por línea, alfabético y descendente.

El Ejemplo 3.66 utiliza el comando **`sort`** para ordenar por el tercer campo de la línea, especificamente que los campos están separados por coma (,). El ordenamiento se realiza alfabética y ascedentemente.

Ejemplo 3.65: Ordena alfabéticamente por línea, descendentemente.

```
$ sort -r clostridium02.csv
```

Ejemplo 3.66: Ordena por el tercer campo de la línea donde cada campo está separado por coma.

```
$ sort -k 3 -t , clostridium02.csv
```

El Ejemplo 3.67 realiza un ordenamiento por el segundo campo de la línea, considerando que los valores son numéricos. Los campos están separados por coma (`,`).

3.4. Acceso y descarga de datos biológicos en bases de datos

En esta sección se mostrará la manera de acceder y descargar datos biológicos, específicamente de ensambles, de las bases de datos públicas.

Se utilizarán las bases de datos públicas NCBI Reference Sequence Database (Refseq) y GenBank de NCBI. GenBank es una base de datos de seuencias, almacena una colección de datos biológicos anotados de todas las secuencias de ADN disponibles públicamente.

Para descargar datos biológicos de ensambles de las bases de datos públicas realice los pasos:

1. Acceda a una de las plataformas mencionadas:
    a. Para Refseq: https://www.ncbi.nlm.nih.gov/refseq/
    b. Para Genbank: https://www.ncbi.nlm.nih.gov/genbank/
2. Realice una búsqueda de la especie deseada. Por ejemplo, teclee `Escherichia coli` para obtener un listado de los datos biológicos de esta especie.
3. Del resultado obtenido puede seleccionar las cepas de interés y dar clic en Descargar.

Ejemplo 3.67: Ordena por el segundo campo de la línea donde cada campo esta separado por coma (,) y los valores son numéricos.

```
$ sort -n -k 2 -t , clostridium-muestras.csv
```

3.5. Instalación y configuración de herramientas bioinformáticas en Linux

La instalación de software en GNU/Linux se puede realizar manualmente o automatizada. La instalación de software manualmente es necesaria cuando el autor del software ha publicado el código fuente que necesita ser configurado y ejecutado en nuestra computadora o servidor. La instalación de software automatizada, es más cómoda para el usuario, utiliza un software de administrador de paquetes que se encarga de realizar la instalación del software así como de los requerimientos del software.

Un paquete es un software preparado para ser instalado en la distribución GNU/-Linux para la que fue creado. Los paquetes se almacenan en un respositorio, un servidor que almacena y mantiene actualizados los paquetes.

3.5.1. Instalación de paquetes

Las distribuciones de GNU/Linux utilizan diferentes software de administración de paquetes. Ubuntu utiliza el software de administrador de paquetes *Advance Package Tool* (APT).

Buscar paquetes y comprobar versiones disponibles

Sintaxis:

```
apt-cache [opciones] nombrepaquete
```

Opciones:

`search` realiza búsqueda en el repositorio por el nombre del paquete.

`policy` muestra la versión instalada y la versión disponible en el repositorio, de un paquete.

Ejemplo 3.68: Buscar el paquete fastqc en el repositorio.

```
1 $ apt-cache search fastqc
2 atropos - NGS read trimming tool that is specific, sensitive, and speedy
3 fastqc - quality control for high throughput sequence data
4 qcumber - quality control of genomic sequences
5 trim-galore - automate quality and adapter trimming for DNA sequencing
```

En el Ejemplo 3.68, línea 1 se busca por el paquete `fastqc`. A partir de la línea 2 muestran los nombres y la descripción de los paquetes obtenidos como resultado.

Ejemplo 3.69: Comprobar las versiones instalada y disponible del paquete fastqc.

```
1 $ apt-cache policy fastqc
```

El Ejemplo 3.69 muestra cómo usar el comando `apt-cache` con el parámetro `policy` para comprobar las versiones instalada y disponible de un paquete en específico. Esto es útil para verificar si la versión instalada es la más reciente, por ejemplo.

Instalar y eliminar paquetes

Sintaxis:

```
apt-get [opciones] nombrepaquete
```

Opciones:

`install` Permite instalar el paquete.

`remove` Permite eliminar el paquete. Permanecerán la configuración y los datos de usuario generados durante el uso del paquete.

`purge` Permite eliminar el paquete incluyendo la configuración y los datos de usuario generados durante el uso del paquete.

Para instalar o eliminar un paquete es necesario tener permisos de administrador. El comando **sudo** se utiliza para ejecutar el comando **apt-get** con permisos de administrador.

Los Ejemplos 3.70, 3.71, y 3.72 muestran cómo instalar y eliminar el paquete **fastqc**. Adicionalmente, el Ejemplo 3.72 elimina también la configuración y los datos de usuario generados por el paquete.

Ejemplo 3.70: Instalar el paquete fastqc

```
1 $ sudo apt-get install fastqc
```

Ejemplo 3.71: Eliminar el paquete fastqc

```
1 $ sudo apt-get remove fastqc
```

Ejemplo 3.72: Eliminar el paquete fastqc incluyendo la configuración y los datos de usuario.

```
1 $ sudo apt-get purge fastqc
```

Configuración y uso de repositorios

Un escenario que se presenta ocasionalmente es que requerimos instalar software que no está disponible en los repositorios de Ubuntu. El desarrollador del software lo ha publicado en un repositorio externo. Para instalar el software adicional, será necesario dos pasos: a) registrar el nuevo repositorio en Ubuntu con el comando `add-apt-repository` y b) instalar el software con el comando **apt-get**.

Agregar un nuevo repositorio puede representar un riesgo de seguridad, usted debe verificar que el desarrollador del software es confiable.

Sintaxis:

```
1 $ sudo add-apt-repository  nombre-repositorio
```

3.5.2. Anaconda

`Anaconda` es una opción muy útil y práctica para instalar paquetes en Ubuntu. Tiene la ventaja que la instalación de paquetes es sencilla y tiene la posibilidad de instalar paquetes en entornos aislados para garantizar la compatibilidad entre prerequisitos de los paquetes.

`Anaconda` distribución libre, descargable, gratuita, de código abierto, de alto rendimiento y optimizada de Python y R.

`Anaconda` incluye los paquetes conda, conda-build, Python y más de 250 paquetes adicionales de uso científico y de código abierto instalados automáticamente.

Existe `Miniconda`, una versión reducida de `Anaconda`. `Miniconda` solamente incluye los paquetes conda, Python y los paquetes de los que dependen; así como una pequeña cantidad de otros paquetes útiles: pip, zlib y algunos otros.

Instalación y configuración de Miniconda para la gestión de paquetes, dependencias y entornos

Para descargar `Miniconda`, se puede obtener la versión más reciente en su sitio web oficial. Hasta la fecha de publicación de este documento, la versión 3 es la más reciente.

El Ejemplo 3.73 ejecuta dos instrucciones que permiten descargar el archivo de instalación de `Miniconda` y verificar la integridad del archivo descargado.

Ejemplo 3.73: Descarga de Miniconda.

```
#for Linux 64-bit
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
#verify integrity
$ sha256sum Miniconda3-latest-Linux-x86_64.sh
78f39f9bae971ec1ae7969f0516017f2413f17796670f7040725dd83fcff5689
```

En el Ejemplo 3.73, línea 1 se utiliza el comando `wget` para descargar el archivo de instalación de `Miniconda`. Usted debe confirmar que al ejecutar la línea 4, obtiene la cadena mostrada en la línea 5.

El siguiente paso será instalar y configurar `Miniconda`. Para ello deberá ejecutar el código mostrado en el Ejemplo 3.74.

Ejemplo 3.74: Instalación de Miniconda.

```
1 $ chmod +x Miniconda3-py37_4.12.0-Linux-x86_64.sh
2 $ ./Miniconda3-py37_4.12.0-Linux-x86_64.sh
3 ...
4 Thank you for installing Miniconda3!
5 $ export PATH=$PATH:/home/rogelio/miniconda3/bin
6 $ echo "export PATH=$PATH:/home/rogelio/bin" >> ~/.bashrc
7 $ conda --version
```

En el Ejemplo 3.74, línea 1 se agregan permisos de ejecución al archivo de instalación. En la línea 2 se ejecuta el archivo de instalación. Al finalizar la ejecución, debe aparecer el mensaje de la línea 4.

En la línea 5 se agrega la ruta a la variable `PATH`. La variable `PATH` se agrega al final del archivo de configuración `.bashrc`. Esta configuración permitirá ejecutar `Miniconda` desde cualquier directorio.

Creación y administración de entornos

`Miniconda` permite la creación de entornos. Un entorno puede tener versiones instaladas de paquetes y de Python diferentes a otro entorno. Esto nos permite instalar herramientas bioinformáticas que no podrían estar instaladas en la misma computadora si los paquetes requeridos son incompatibles.

Para crear un entorno, la sintaxis es:

```
conda create --name nombredelentorno [python=version]
```

Opciones:

`nombredelentorno` es el nombre que identificará a ese entorno para activarlo o desactivarlo.

`python=version` especifica el número de versión de python que se instalará en el entorno. La versión predeterminada de Python es 3.9. Las versiones disponibles de Python son 3.7, 3.8, 3.9 y 3.10.

Una vez creado el entorno, para utilizarlo es necesario indicarle a Conda que active el entorno. Conda tiene opciones para activar o desactivar un entorno.

Sintaxis:

```
conda [activate] [deactivate] nombredelentorno
```

Instalación de herramientas bioinformáticas

Para instalar herramientas de software se puede utilizar el respositorio de Ubuntu o Conda. Si desea instalar la versión más reciente de cada paquete, puede comparar la versión disponible en Ubuntu y Conda y elegir el más reciente.

Conda permite instalar paquetes de manera sencilla, provee el sitio web https://anaconda.org/ que cuenta con un buscador de paquetes. Al localizar un paquete, el sitio web le brinda el comando, que deberá ejecutar en la Terminal, para realizar la instalación.

En el código contenido en el Ejemplo 3.75 se muestra la instalación de las herramientas que se utilizarán en el Sección 5.4. Por ejemplo, el paquete **`fastqc`** aparece con la versión 0.11.9 en ambos repositorios; por tanto, se instalará desde el repositorio de Ubuntu para que esté disponible para todos los usuarios y para todos los entornos de Conda.

Ejemplo 3.75: Instalación de herramientas de software utilizando Conda y el repositorio de Ubuntu.

```
#Paso 1. instalar fastqc utilizando el repositorio de Ubuntu
$ sudo apt-get install fastqc

#Paso 2. crear y activar un ambiente en Conda
$ conda create --name bioinformatica
$ conda activate bioinformatica

#Paso 3. Instalar herramientas de software en el ambiente bioinformatica
$ conda install -c "bioconda/label/cf201901" a5-miseq

#Paso 3. crear y activar un ambiente en Conda
$ conda deactivate
$ conda create --name anotacion
$ conda activate anotacion

#Paso 4. Instalar herramientas de software en el ambiente anotacion
$ conda install -c "bioconda/label/cf201901" prokka
$ conda install -c "bioconda/label/cf201901" parsnp
$ conda install -c "bioconda/label/cf201901" harvesttools
$ conda install -c "bioconda/label/cf201901" raxml
```

3.6. Manipulación de cadenas

La manipulación de cadenas permite realizar la recuperación, transformación, reducción y validación de datos de entrada o de archivos de texto sin formato. En bioinformática, aprender sobre la manipulación de cadenas nos permitirá renombrar encabezados (*headers*), reemplazar secuencias, modificar salidas para ser utilizadas como entradas en un flujo de trabajo, o automatizar tareas tediosas y repetitivas que pueden propiciar errores humanos. En esta sección se incluyen dos comandos que permiten manipular cadenas. El comando **sed**, un editor de cadenas en línea de comandos, y **awk** un lenguaje de programación que permite manipular cadenas mediante la construcción de programas que usualmente contienen muy pocas líneas de código.

3.6.1. awk

awk es un lenguaje de programación interpretativo. Un programa escrito en **awk** es una secuencia instrucciones que contienen patrones y acciones. Los patrones permiten especificar lo que se busca en los datos de entrada. Las acciones se realizarán en cada línea de entrada cuando se encuentren los patrones. [53] [2].

awk está incluido de manera predeterminada en la mayoría de las distribuciones actuales de GNU/Linux. **awk** permite manipular datos particulamente cuando los datos de entrada o archivos están estructurados. Se considera que los datos de entrada están estructurados cuando están organizados como registros (líneas) y campos (columnas).

Funcionamiento de awk

La función básica de **awk** es buscar, en archivos, por líneas que contengan ciertos patrones.

En el contexto de **awk** un registro es una cadena separada por un *separador de registro*. De manera predeterminada, el separador de registro es el carácter nueva línea. Por tanto, cada línea en el archivo de entrada es considerado un registro. En esta sección usaremos de manera indistinta línea o registro a menos que se especifique otro *separador de registro*.

Una vez que **awk** lee un registro, obtiene los campos utilizando el *separador de campos*. De manera predeterminada, **awk** establece que los campos están separados por espacios en blanco (espacios en blanco o tabulaciones).

La Figura 3.2 muestra el flujo de trabajo que realiza **awk**. Inicialmente **awk** ejecuta el código contenido en el bloque **BEGIN**, posteriormente realiza los pasos:

1. Buscar, en los archivos, por líneas que contengan los patrones especificados.
2. Cuando una línea se encuentra uno de los patrones, **awk** ejecuta en esa línea las acciones especificadas.

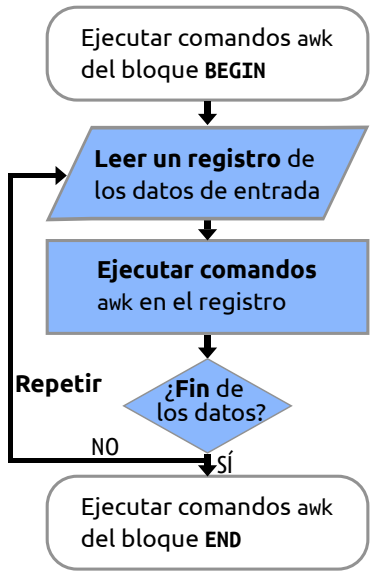

FIGURA 3.2: Flujo de trabajo de **awk**.

3. **awk** continúa el procesamiento de las líneas de entrada de esta manera hasta que alcanza el final de los datos de entrada o del archivo de entrada.

Finalmente, ejecuta por única ocasión el código contenido en el bloque **END**.

Sintaxis y ejecución

Es posible ejecutar **awk** de dos maneras.

1. Sintaxis para ejecutar explícitamente el programa:

```
awk [opciones] '[BEGIN { action; }] /search/ [{ action; }] [END { action; }]
    ' archivo_entrada1 [archivo entrada2 ...]
```

Opciones:

`-F` Especifica el separador de campos.

La sintaxis consiste de dos partes: a. las instrucciones que ejecutará **awk** y b. uno o más archivos de datos de entrada.

2. Sintaxis para ejecutar el programa desde un archivo:

```
awk [opciones] -f archivo_programa archivo_entrada1 [archivo_entrada2 ...]
```

Esta sintaxis permite que **awk** reciba: a. un archivo con las instrucciones que serán ejecutadas por **awk** y b. uno o más archivos de datos de entrada. La ventaja de utilizar esta sintaxis es que las instrucciones, por estar almacenadas en un archivo, se pueden reutilizar.

Ejemplo 3.76: Listar las líneas que contengan AA.

```
1 $ awk '/AA/' GCF_ecolli.fasta
2 $ awk '/AA/ {print $0}' GCF_ecolli.fasta
3 $ awk '$0 ~ /AA/ {print $0}' GCF_ecolli.fasta
```

En el Ejemplo 3.76, las líneas 1, 2 y 3 obtienen el mismo resultado. En la línea 1 se utiliza **awk** para buscar el patrón compuesto por la cadena `AA`, en caso de encontrar una línea que contenga el patrón, de manera predeterminada se imprime la línea donde fue encontrado el patrón. La línea 2 imprime de manera explícita la variable `$0` que se refiere al contenido de la línea leída. La línea 3 indica explícitamente que el patrón debe ser buscado en `$0` (en toda la línea), cuando encuentra el patrón, imprime de manera explícita la línea. El operador ~ indica coincidencia.

Ejemplo 3.77: Listar las líneas que no contengan AA.

```
1 $ awk '$0 !~ /AA/ {print $0}' GCF_ecolli.fasta
```

awk cada vez que lee una línea guarda su contenido en la variable `$0`. En el Ejemplo 3.77 línea 1, **awk** busca que la línea no contenga `AA` en cada una de las líneas del archivo `GCF_ecolli.fasta`. El operador `!~` indica la no coincidencia.

Uso de variables de columnas

awk cada vez que lee una línea, separa los valores de los campos utilizando el espacio como separador de campos, o el separador de campos indicado por el usuario; almacena cada campo en una variable: `$1,$2,$3,...$n`. Es posible utilizar estas variables para imprimir sus valores o para buscar coincidencias de patrones en una columna en específico.

Ejemplo 3.78: Listar las líneas que contengan 2021 en la tercera columna.

```
1 $ awk '$3 ~ /2021/' clostridium02.txt
```

awk cada vez que lee una línea, separa los valores de los campos y almacena cada campo en una variable: `$1,$2,$3,`. En el Ejemplo 3.78, `$3` se refiere al campo 3 de la línea leída, es decir, busca el patrón `2021` en la tercera columna.

Expresiones regulares en awk

Una expresión regular es una cadena que describe a un conjunto de cadenas. *Definición matemática*: una expresión regular `r` hace coincidencia con una cadena `s` si `s` está en el conjunto de cadenas descrito por `r` [26].

awk permite el uso de expresiones regulares para buscar coincidencias en las líneas de los datos de entrada. Para escribir expresiones regulares es posible utilizar caracteres especiales. A estos caracteres especiales también se les conoce como *comodines* (en inglés, *wildcards*). La Tabla 3.9 muestra los comodines que se utilizan en **awk**.

Ejemplo 3.79: Listar las líneas que contengan AA al inicio.

```
1 $ awk '/^AA/' GCF_ecolli.fasta
```

En el Ejemplo 3.79, el carácter ^ representa el inicio de la cadena. El patrón de búsqueda AA debe aparecer al inicio de la línea. De manera predeterminada **awk** imprimirá cada línea que contenga el patrón.

Tabla 3.9: Comodines utilizados en **awk** para escribir expresiones regulares.

Símbolo	Uso/significado
\	Suprime el significado especial del caracter al buscar coincidencias.
^	Encuentra el inicio de una cadena.
$	Similar a ^, pero encuentra solo al final de la cadena.
.	Coincide con cualquier carácter individual, incluido el carácter de nueva línea.
[...]	Lista de caracteres. Coincide con cualquiera de los caracteres que se incluyen entre corchetes. Ej. [0-9]
[^...]	Lista de caracteres complemento. Ej. [^0-9].
|	Se utiliza para especificar alternativas.
*	La regexp que lo precede es repetida tantas veces como sea necesario. Por ejemplo: ph*.
+	Similar a *, pero la regexp precedente debe aparecer una vez.
$	Similar a ^, pero encuentra solo al final de la cadena.
?	Es similar a *, pero la regexp precedente puede ser encontrada una o ninguna vez.

Ejemplo 3.80: Listar las líneas que contengan 2021 al inicio de la tercera columna.

```
1 $ awk '$3 ~ /^2021/' clostridium02.txt
```

En el Ejemplo 3.80 el carácter ^ representa el inicio de la cadena. El patrón de búsqueda 2021 debe aparecer al inicio de la tercera columna.

Ejemplo 3.81: Listar las líneas que no contengan 2021 al inicio de la tercera columna.

```
1 $ awk '$3 !~ /^2021/' clostridium02.txt
```

El operador !~ indica no coincidencia. En el Ejemplo 3.81 se buscan líneas que en la tercer columna no coincidan con el patrón de búsqueda.

Ejemplo 3.82: Elegir las líneas no que contengan 2021 al inicio de la tercera columna. Imprimir solo el primer y segundo campo de las líneas que coincidan con la búsqueda.

```
$ awk '$3 !~ /^2021/ {print $1,$2;}' clostridium02.txt
```

El Ejemplo 3.82 se basa en el Ejemplo 3.81, la diferencia es que agrega el bloque de acción para imprimir solo los campos 1 y 2.

Ejemplo 3.83: Encontrar las líneas de un archivo fasta que terminan con la secuencia de nucleótidos TTAAA.

```
$ awk '/TTAAAA$/' GCF_ecolli.fasta
awk '$0 ~ /TTAAAA$/' GCF_ecolli.fasta
```

En el Ejemplo 3.83, las líneas 1 y 2 obtienen el mismo resultado. Se buscan líneas que contengan `TTAAAA` e inmediatamente después se encuentre el final de la línea.

Ejemplo 3.84: Encontrar las líneas de un archivo cuyo tercer campo termina en 24.

```
$ awk '$3 ~ /24$/' clostridium02.txt
```

En el Ejemplo 3.84, el patrón de búsqueda debe cumplir que la cadena `24` debe aparecer en la tercera columna y que inmediatamente después se encuentre el final de la cadena.

En el Ejemplo 3.85, el patrón de búsqueda debe cumplir que antes del final de la tercera columna no se encuentre la cadena `24`.

Variables internas

awk cuenta con variables internas. Las variables internas son variables definidas a las cuales **awk** les asigna valor inicial y las mantiene con valores actualizados. La Tabla 3.10 muestra un listado de las variables internas. Usted puede utilizar y asignar valores a estas variables internas en un programa **awk**.

Ejemplo 3.85: Encontrar las líneas de un archivo cuyo tercer campo no termina en 24.

```
$ awk '$3 !~ /24$/' clostridium02.txt
```

Existen variables internas que son actualizadas por **awk** antes de iniciar la lectura de las líneas del archivo de entrada. Por ejemplo, a la variable `FILENAME` se asigna el valor del nombre del archivo de entrada; Las variables `FS` y `OFS` son inicializadas con un carácter de espacio. También existen variables internas que son actualizadas al realizar la lectura de cada línea. Por ejemplo, la variable `NF` se actualiza con la cantidad de campos que tiene el registro actual considerando el valor del separador de campos `FS`; la variable `NR` se incrementa para contar el número de línea o registro actual.

Ejemplo 3.86: Numerar e imprimir cada línea del archivo.

```
$ awk '{print NR $0}' GCF_ecolli.fasta
```

La variable `NR` se refiere al número de registro o de línea. La variable `$0` hace referencia al contenido de la línea leída. Como **awk** lee línea por línea entonces por cada línea se imprime el número de registro y el contenido de la línea. El Ejemplo 3.86 obtiene un resultado similar al comando **nl**.

Ejemplo 3.87: Mostrar el nombre del archivo y numerar las líneas del registro actual y el número de línea general.

```
$ awk  '{print FILENAME, FNR, NR}' seq.list mini_fasta.fst
```

El Ejemplo 3.87 muestra la diferencia entre las variables `FNR` y `NR`. `FNR` muestra el número de línea del archivo actual y cuando se lee la primera línea del siguiente archivo entonces el valor se reinicia en 1. En cambio, `NR` muestra el número de línea general.

Tabla 3.10: Variables internas de awk.

Variable	Nombre en inglés	¿Qué almacena?
FILENAME	*Filename*	El nombre del archivo de entrada actual.
FNR	*Number of Record of current File*	El número de registro actual en relación con el archivo de entrada actual. Si se tienen dos archivos de entrada, esto le indicaría el número de registro de cada archivo en lugar de un número de registro general.
FS	*Field Separator*	El separador de campo usado para denotar cada campo en un registro. De manera predeterminada se utiliza el espacio.
NF	*Number of Fields*	El número de campos en el registro actual.
NR	*Number of Record*	El número de registro actual. Si se tienen dos archivos de entrada, esto le indicaría el número de registro general.
OFS	*Output Field Separator*	El separador de campos para los datos de salida. De manera predeterminada es el espacio.
RS	*Record Separator*	El separador de registros usado para distinguir registros separados en el archivo de entrada. De manera predeterminada, es el carácter de nueva línea.

Los patrones especiales `BEGIN` *y* `END`

El bloque de código `BEGIN` se ejecuta por única ocasión, antes de iniciar la lectura de los datos de entrada o del archivo de datos. En el bloque `BEGIN` es posible asignar valores iniciales a variables.

En cambio, el bloque de código `END` se ejecuta por única ocasión, una vez que se ha alcanzado el final de los datos de entrada o del archivo de entrada. En el bloque `END` es posible imprimir el valor obtenido por las variables, una vez que se ha concluido la lectura de los datos de entrada.

En el Ejemplo 3.88 las tres (3) instrucciones son equivalentes, generan el mismo resultado. En la línea 1 se imprime al inicio y por única ocasión la cadena indicada. Después `awk` lee cada línea del archivo `GCF_ecolli.fasta` y si la línea leída contiene el

Ejemplo 3.88: Listar las líneas que contengan AA en un archivo. Imprimir un mensaje previo a la impresión de las líneas.

```
1 $ awk 'BEGIN{print "Líneas que contienen AA:"} /AA/' GCF_ecolli.fasta
2 $ awk 'BEGIN{print "Líneas que contienen AA:"} /AA/ {print $0}' GCF_ecolli.fasta
3 $ awk 'BEGIN{print "Líneas que contienen AA:"} $0 ~ /AA/ {print $0}' GCF_ecolli.
    fasta
```

patrón `AA` imprime la línea completa. Si no se especifica la instrucción `print` entonces **awk** imprime la línea completa. En la línea 2, se muestra que también se puede especificar de manera explícita la impresión de la línea. En la línea 3, hace lo mismo que la línea 2 pero agrega explícitamente que se busque el patrón `AA` en el registro. Considere que la variable `$0` se refiere al registro (línea completa).

Ejemplo 3.89: Numerar e imprimir cada línea del archivo. Mostrar los campos separados por guión bajo.

```
1 $ awk 'BEGIN{OFS="_"}{print NR,$0}' GCF_ecolli.fasta
```

El Ejemplo 3.89 asigna el guión bajo como separador de campos de salida. Por cada línea leída se ejecuta el comando `print`. El comando `print` mostrará como salida el contenido de las dos variables utilizando como separador el guión bajo. `NR` se refiere al número de registro (línea) y `$0` hace referencia a la línea leída.

Ejemplo 3.90: Contar las líneas que contengan AA en un archivo.

```
1 $ awk '/AA/ {contador=contador+1} END{print contador, "líneas contienen AA"}'
    GCF_ecolli.fasta
2 $ awk '/AA/ {contador=++contador} END{print contador, "líneas contienen AA"}'
    GCF_ecolli.fasta
```

En el Ejemplo 3.90, la línea 1 y 2 obtienen el mismo resultado. Por cada línea que contenga el patrón `AA` la variable `contador` se incrementa. Al concluir la lectura de todas las líneas de datos, se ejecuta el bloque **END** para mostrar el valor final de

la variable `contador`. La línea 2 utiliza la expresión `++contador` que permite primero incrementar la variable y luego utilizar su valor.

Ejemplo 3.91: Listar las líneas que contengan pi en un archivo cuyos campos están separados por comas (csv).

```
1 $ awk 'BEGIN{FS=",";} /AA/' GCF_ecolli.fasta
2 $ awk -F, '/AA/' GCF_ecolli.fasta
3 $ awk -F, '$0 ~ /AA/ {print $0}' GCF_ecolli.fasta
4 $ awk 'BEGIN{FS=",";} $0 ~ /AA/ {print $0}' GCF_ecolli.fasta
```

En el Ejemplo 3.91 todas las líneas obtienen el mismo resultado. Si los datos de entrada están separados por un carácter diferente al espacio, se debe especificar el carácter separador en la variable `FS`. En este ejemplo, en las líneas 1 y 4 se utiliza el bloque **`BEGIN`** para asignar a la variable `FS` la coma (`,`) como separador de campos. En la línea 2 y 3 se utiliza el parámetro `-F` para indicar que el separador de campos será la coma (`,`).

Ejemplo 3.92: Elegir las líneas no que contengan 2021 al inicio de la tercera columna. Imprimir solo el primer y segundo campo de las líneas que coincidan con la búsqueda. Considere que el archivo de entrada está separado por comas (csv).

```
1 $ awk 'BEGIN{FS=",";} $3 !~ /^2021/ {print $1,$2;}' clostridium02.txt
2 $ awk -F, '$3 !~ /^2021/ {print $1,$2;}' clostridium02.txt
```

En el Ejemplo 3.92 las instrucciones de la línea 1 y 2 son equivalentes, obtienen el mismo resultado. En el bloque de acción, se imprimen las columnas 1 y 2.

Ejemplo 3.93: Cantidad de registros (líneas) de un archivo.

```
1 $ awk 'END{print NR}' comida-favorita.txt
```

En el Ejemplo 3.93 se utiliza el bloque **`END`** para imprimir el valor de la variable `NR` (*Number of Record*) que contiene el número de registros. **`awk`** por cada línea leída

(registro leído) va sumando uno a la variable `NR`. Al imprimir la variable en el bloque END, se imprime el valor final, una vez que se han leído todos las líneas del archivo de datos de entrada.

Estructuras condicionales

awk permite condicionar la ejecución de acciones. **awk** permite escribir expresiones relacionales que expresan la relación para cadenas o enteros. La Tabla 3.11 muestra los operadores relacionales permitidos en **awk**. Si el resultado de evaluar una expresión relacional es **true** (verdadero) entonces se ejecutan las acciones asociadas.

Ejemplo 3.94: Eliminar las líneas en blanco de un archivo.

```
1 $ awk 'NF > 0' comida-favorita.txt
```

awk lee el contenido del archivo línea por línea. Cuando lee una línea **awk** cuenta la cantidad de campos y almacena el valor en la variable `NF`. En el Ejemplo 3.94 por cada línea se evalúa `NF > 0` para determinar si la línea tiene al menos un campo. Si se cumple que la línea no está vacía se imprime la línea leída.

Tabla 3.11: Operadores relacionales permitidos en **awk**.

Expresión	resultado **true** si
`x ~ y`	la cadena `x` coincide con la expresión regular denotada por `y`
`x !~ y`	la cadena `x` no coincide con la expresión regular denotada por `y`
`x < y`	`x` es menor que `y`
`x <= y`	`x` es menor o igual que `y`
`x > y`	`x` es mayor que `y`
`x >= y`	`x` es mayor o igual que `y`
`x == y`	`x` es igual a `y`
`x != y`	`x` no es igual a `y`
`subíndice in arreglo`	el arreglo `arreglo` tiene un elemento con el subíndice `subíndice`

Operadores lógicos

awk permite utilizar los operadores lógicos AND (&&), OR (||) y NOT (!). La Tabla 3.12 muestra cómo representar los operadores lógicos en **awk**.

Tabla 3.12: Operadores lógicos permitidos en **awk**.

Expresión	resultado **true** si
`expresión1 && expresion2`	ambas expresiones obtienen un resultado **true**
`expresión1 || expresion2`	cualquier expresión obtiene un resultado **true**
`!expresión1`	la `expresión1` tiene un valor **false**, la negación obtiene un valor **true**.

Ejemplo 3.95: Listar las líneas que contengan 24 en la tercera columna. Solo las primeras 5 líneas.

```
1 $ awk '$3 ~ /24/ && NR<=5' clostridium02.txt
```

El Ejemplo 3.95 utiliza el operador lógico de conjunción **AND** (&&) para obtener un listado de las líneas que cumplan con las dos condiciones. La primera condición es que la tercera línea contenga la cadena `24`, la segunda condición es que el número de registro sea menor a `5`.

Ejemplo 3.96: Listar las líneas que cumplan con alguno de los siguientes criterios: a) que contengan 24 en la tercera columna o b) que el número de línea sea menor a 5.

```
1 $ awk '$3 ~ /24/ || NR<5' clostridium02.txt
```

El Ejemplo 3.96 utiliza el operador lógico de disyunción **OR** (&&) para obtener un listado de las líneas que cumplan con alguna de dos condiciones. La primera condición es que la tercera línea contenga la cadena `24`, la segunda condición es que el número de registro sea menor a `5`.

El Ejemplo 3.97 utiliza el operador lógico de disyunción **OR** (||) para obtener un listado de las líneas que cumplan con cualquiera de las dos condiciones.

Ejemplo 3.97: Listar las líneas que cumplan con cualquiera de los siguientes criterios: a) que contengan 2022 en la tercera columna o b) que contenga perfringens en la primera columna.

```
$ awk '$3 ~ /2022/ || $1 ~ /perfringens/' clostridium02.txt
```

3.6.2. sed

sed un editor de cadenas en línea de comandos. **sed** al igual que **awk** procesa las líneas de los datos de entrada, una por una; les aplica la transformación indicada y muestra en la salida estándar (en pantalla) las líneas modificadas. **sed** permite realizar transformaciones de texto básicas: buscar y reemplazar, eliminar e insertar líneas y convertir a mayúsculas y minúsculas.

Sintaxis:

```
sed 'programa' archivo_entrada1 [archivo entrada2 ...]
```

Un programa escrito en **sed** puede estar conformado por uno o más comandos de **sed**.

Operaciones básicas: sustitución, eliminación, inserción

La operación **sustitución** se indica con una letra `s`, la cadena o patrón a buscar y la cadena que reemplaza.

Sintaxis:

```
sed '[9,9] s/cadenabuscar/cadenareemplazo/[n][g]' archivo_entrada1 [archivo
    entrada2 ...]
```

`[9,9]` es el rango de líneas en las cuales ejecutará la acción.

`[n]` número de coincidencia para realizar el reemplazo.

`[g]` sustitución global, se realizará el reemplazo en todas las coincidencias de cada línea.

Ejemplo 3.98: Sustituir, en cada línea, la primera aparición de Clostridium_ por clostridium- en el archivo clostridium02.txt.

```
$ sed 's/Clostridium_/clostridium-/' clostridium02.txt
```

El Ejemplo 3.98 realiza la operación sustitución, en cada línea, de la primera aparición de `Clostridium_` por `clostridium-` en el archivo `clostridium02.txt`. Si se omite la `g` (global), **`sed`** realizará la sustitución solamente de la primera coincidencia de cada línea.

Ejemplo 3.99: Sustituir, en cada línea, todas las apariciones de - por _ en el archivo clostridium02.txt.

```
$ sed 's/-/_/g' clostridium02.txt
```

El Ejemplo 3.99 realiza la operación sustitución, en cada línea, de todas las apariciones de la primera cadena. Es decir, realiza la sustitución *global* en todo el archivo.

Ejemplo 3.100: Sustituir en cada línea todas apariciones de TTT por AAA en el archivo GCF_ecolli.fasta.

```
$ sed 's/TTT/AAA/g;' GCF_ecolli.fasta
```

El Ejemplo 3.100 muestra cómo sustituir de manera global la cadena `TTT` por `AAA` en el archivo `GCF_ecolli.fasta`, es decir, todas las apariciones de `TTT` son sustituidas.

Si se desea eliminar una cadena o patrón en específico se puede ejecutar una operación de sustitución utilizando una cadena vacía (cadena sin ningún carácter) de reemplazo. En el Ejemplo 3.101 se eliminan las tripletas `TTT`.

El Ejemplo 3.102 utiliza la sintaxis para especificar que solamente se sustituya la segunda aparición de `TTT` por `AAA`. En la instrucción, el número 2 se refiere al número de coincidencia para realizar el reemplazo.

Ejemplo 3.101: Eliminar en cada línea todas apariciones de TTT en el archivo GCF_ecolli.fasta.

```
$ sed 's/TTT//g;' GCF_ecolli.fasta
```

Ejemplo 3.102: Sustituir en cada línea la segunda aparición de TTT por AAA en el archivo GCF_ecolli.fasta.

```
$ sed "s/TTT/AAA/2" GCF_ecolli.fasta
```

El Ejemplo 3.103 utiliza el `2g` para indicar que se reemplazará la cadena de búsqueda a partir de la segunda aparición en cada línea.

El Ejemplo 3.104 utiliza una sintaxis para indicar que el reemplazo se realice únicamente en el rango de líneas especificado, en las líneas 1 a 3. El rango de líneas antecede a la operación sustitución.

Expresiones regulares en `sed`

`sed` permite el uso de expresiones regulares para buscar coincidencias en las líneas de los datos de entrada. Para escribir expresiones regulares es posible utilizar caracteres especiales. A estos caracteres especiales también se les conoce como *comodines* (en inglés, *wildcards*). La Tabla 3.13 muestra los comodines que se utilizan en `sed`.

Tabla 3.13: Comodines utilizados en `sed` para escribir expresiones regulares.

Comodín	Significado
`$`	encuentra el final de la línea
`^`	encuentra el inicio de la línea
`*` (*asterisco*)	encuentra *cero o más* ocurrencias del carácter que lo antecede
`[ ]`	encuentra cualquiera de los caracteres contenidos en los corchetes

El Ejemplo 3.105 utiliza los corchetes para buscar cadenas que contengan `AAA` y el cuarto carácter sea cualquiera de los contenidos al interior de los corchetes. Se buscan cadenas que contengan `AAAC` o `AAAG` y son reemplazadas por `AAA-`.

Ejemplo 3.103: Sustituir en cada línea, a partir de la segunda aparición, todas las apariciones de TTT por AAA en el archivo GCF_ecolli.fasta.

```
$ sed "s/TTT/AAA/2g" GCF_ecolli.fasta
```

Ejemplo 3.104: Sustituir apariciones de - por _ en las líneas 1 a 3.

```
$ sed '1,3 s/-/_/g' clostridium02.txt
```

El Ejemplo 3.106 busca cadenas que contengan al menos un dígito y las reemplaza por cadena vacía, es decir, las elimina. El patrón de búsqueda consiste de `[0-9]` que hace referencia a cualquier dígito, seguido de `[0-9]*`. El asterisco en esta expresión significa que el carácter anterior puede aparecer cero o más veces, en este caso, puede hacer coincidencia con cadenas que contengan números enteros de cualquier longitud.

El Ejemplo 3.107 utiliza el carácter especial `^` para referirse a cadenas ubicadas al inicio de la línea. En el ejemplo reemplaza `AAA` ubicada al inicio de la línea por por su complemento `TTT`.

El comodín `$` se refiere al final de la línea o campo. En el Ejemplo 3.108 se reemplaza la cadena `AAA`, ubicada al final de la línea, por su complemento `TTT`.

En **sed** es posible realizar **eliminación** (`d`). La operación eliminación descarta cada línea de los datos de entrada que contenga el patrón de búsqueda. La operación eliminación también permite descartar un rango de líneas en específico. La operación `d` se escribe después del patrón, a diferencia de la operación `s` que va al inicio del patrón.

Sintaxis:

```
sed '[9,9] [/cadenabuscar/cadenareemplazo/]d' archivo1 [archivo2 ...]
```

El Ejemplo 3.109 busca y elimina las líneas que contengan la cadena `AAA`.

El Ejemplo 3.110 busca y elimina líneas que inmediatamente después del inicio de línea esté el fin de línea, esto hace coincidencia con líneas en blanco.

Ejemplo 3.105: Sustituir apariciones de la cadena AAAC o AAAG por AAA- en el archivo GCF_ecolli.fasta.

```
$ sed 's/AAA[CG]/AAA-/g' GCF_ecolli.fasta
```

Ejemplo 3.106: Sustituir dígitos en el archivo clostridium02.txt.

```
$ sed "s/[0-9][0-9]*//g" clostridium02.txt
```

El Ejemplo 3.111 elimina el rango de líneas especificado, de la línea 2 a la línea 7, del archivo `clostridium02.txt`.

Ejecución de múltiples comandos

sed tiene la capacidad de ejecutar múltiples comandos en el mismo programa. Para ejecutar múltiples comandos, cada comando debe finalizar con un punto y coma (`;`).

En el Ejemplo 3.112 **sed** ejecuta dos instrucciones. Las instrucciones se escriben separadas por punto y coma (`;`). Se eliminan las líneas 2-7 y la línea 9 del archivo de entrada.

En el Ejemplo 3.113 se utiliza la expresión `!d`, es el operador lógico negación junto con la operación `d`, que indica mostrar ("no eliminar") las líneas que cumple con el criterio de búsqueda.

En el Ejemplo 3.114 importa el orden. Por ejemplo, la línea `TGAAACCAGACGTTTCGCCCCTA` formaría parte del resultado.

En **sed** es posible realizar la operación **inserción** de diversas maneras. En el Ejemplo 3.115 se utiliza la operación sustituir para reemplazar el inicio de línea (carácter no visible) por 3 espacios. Esto genera que se inserten 3 espacios al inicio de cada línea.

También es posible insertar antes de una línea utilizando el comando `i` (*insert*) o insertar después de una línea utilizando el comando `a` (*append*). El Ejemplo 3.116

Ejemplo 3.107: Sustituir una cadena ubicada al inicio de línea. En las líneas que contengan AAA al inicio, reemplazar AAA por su complemento TTT.

```
1 $ sed "s/^AAA/TTT/" GCF_ecolli.fasta
```

Ejemplo 3.108: Sustituir una cadena ubicada al final de línea. En las líneas que contengan AAA al final, reemplazar AAA por su complemento TTT.

```
1 $ sed "s/AAA$/TTT/" GCF_ecolli.fasta
```

realiza inserción antes de una línea en específico, la línea 2, utilizando la orden `2i`. También realiza una insersión después de la línea 2, al utilizar la orden `2a`.

En el Ejemplo 3.117, línea 1 se ejecuta la operación `i` (*insert*). la operación `i` inserta la cadena `Prueba`, el `$` hace referencia a la última línea. El resultado es que se inserta la cadena `Prueba` antes de la última línea.

En el Ejemplo 3.117, línea 2 se ejecuta la operacion a (*append*). La operación `a` inserta la cadena `Prueba`, el `$` hace referencia a la última línea. El resultado es que se inserta la cadena `Prueba` después de la última línea.

Ejecutar sed para modificar el archivo de datos

sed puede ejecutar los comandos especificados y guardar el resultado en el archivo de datos proporcionado. Para ello se utiliza el parámetro `-i` (*in place*) que además de guardar las modificaciones, también puede realizar una copia de seguridad si se proporciona el `sufijo`.

Sintaxis:

```
1 sed -i[sufijo] 'programa' archivo_entrada1 [archivo entrada2 ...]
```

Antes de ejecutar las modificaciones, **sed** realiza una copia del archivo de datos. El archivo copiado tendrá el nombre del archivo de datos original y al final se agrega el sufijo proporcionado. En el Ejemplo 3.118 se creó el archivo de respaldo con el nombre

Ejemplo 3.109: Eliminar las líneas que contengan la cadena AAA de un archivo fasta.

```
$ sed "/AAA/d" GCF_ecolli.fasta
```

Ejemplo 3.110: Eliminar las líneas en blanco de un archivo fasta.

```
$ sed "/^$/d" GCF_ecolli.fasta
```

`GCF_ecolli.fastamyrespaldo`. Posteriormente **`sed`** ejecuta y guarda las modificaciones en el archivo de datos.

En el Ejemplo 3.119, al omitirse el sufijo, **`sed`** ejecuta y guarda las modificaciones en el archivo de datos sin realizar una copia de respaldo. Debe tener precaución al utilizar esta instrucción porque estará modificando definitivamente el archivo de datos.

`sed` tiene la capacidad de convertir cadenas a **mayúsculas y minúsculas**. Para realizar convertir a mayúsculas utiliza el código `\U`. Para convertir a minúsculas utiliza el código `\L`.

En el Ejemplo 3.120 se utiliza el punto (`.`) para buscar y reemplazar, todas las apariciones de, cualquier carácter (`.`) por su correspondiente mayúscula. El carácter ampersand (`&`) hace referencia a la cadena encontrada.

En el Ejemplo 3.121 se utiliza el punto (`.`) para buscar y reemplazar, todas las apariciones de, cualquier carácter (`.`) por su correspondiente minúscula. El carácter ampersand (`&`) hace referencia a la cadena encontrada.

El Ejemplo 3.122 busca y sustituye la cadena `PDT` por su correspondiente minúscula en todo el archivo.

Ejercicios aplicados a bioinformática

Considere trabajar con el archivo FASTA que fue descargado previamente: `GCF_ecolli.fasta`

Ejemplo 3.111: Eliminar las líneas 2 a la 7 del archivo clostridium02.txt

```
$ sed "2,7 d" clostridium02.txt
```

Ejemplo 3.112: Eliminar las líneas 2 a la 7 y la línea 9 del archivo clostridium02.txt

```
$ sed "2,7d; 9d" clostridium02.txt
```

1. ¿Cuántas secuencias genómicas contiene el archivo `GCF_ecolli.fasta`?

```
grep -c '^>' GCF_ecolli.fasta
awk '/^>/' `GCF_ecolli.fasta` | wc -l
awk '/^>/{contador=contador+1} END{print contador}' GCF_ecolli.fasta
```

Cualquiera de las 3 líneas obtiene el mismo resultado.

2. Contabilice el número de géneros y especies que contiene el archivo `GCF_ecolli.fasta`?

```
grep '^>' GCF_ecolli.fasta | cut -d' ' -f2,3 | sort | uniq -c
```

3. Obtenga una lista ordenada de manera ascendente (de mayor a menor), del número de especies que contiene el archivo FASTA `GCF_ecolli.fasta`.

```
grep '^>' GCF_ecolli.fasta | cut -d ' ' -f 2,3 | sort | uniq -c | sort -n -r -k1
awk '/^>/ {print $2,$3}' GCF_ecolli.fasta | sort | uniq -c | sort -n -r -k1
```

4. En el archivo `GCF_ecolli.fasta`, modifique los encabezados de cada secuencia del archivo FASTA para eliminar los caracteres `( , ; : )` que impiden la generación o visualización del árbol filogenético por ser estos caracteres reservados. Además, se desea que permanezca el número de accesión; que permanezca entre corchetes el género, especie y cepa.

Ejemplo 3.113: Buscar las cadenas AAA y TTT y CCC, en cualquier orden, en la misma línea.

```
$ sed '/AAA/!d; /TTT/!d; /CCC/!d' GCF_ecolli.fasta
```

Ejemplo 3.114: Buscar las líneas que contienen las cadena AAA, seguido de cualquier texto, TTT, seguido de cualquier texto, y CCC.

```
$ sed '/AAA.*TTT.*CCC/!d' GCF_ecolli.fasta
```

```
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb
    .*/]/; s/[[:space:]]/_/g;' GCF_ecolli.fasta | grep '>' |Â head -n 10
```

5. Guardar el resultado del Ejercicio 4 en un nuevo archivo: `NUEVO NOMBRE`.

```
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb
    .*/]/; s/[[:space:]]/_/g;' GCF_ecolli.fasta > NUEVO NOMBRE.
```

Ejemplo 3.115: Insertar 3 espacios en blanco al principio de cada línea.

```
$ sed 's/^/   /' clostridium02.txt
```

Ejemplo 3.116: Insertar la cadena Prueba una línea antes o después de la línea 2 del archivo.

```
$ sed '2i Prueba' clostridium02.txt
$ sed '2a Prueba' clostridium02.txt
```

Ejemplo 3.117: Insertar una línea antes o después del final del archivo.

```
$ sed '$i Prueba' clostridium02.txt
$ sed '$a Prueba' clostridium02.txt
```

Ejemplo 3.118: Sustituir en cada línea todas apariciones de AAA por TTT en el archivo GCF_ecolli.fasta.

```
$ sed -imyrespaldo "s/AAA/TTT/" GCF_ecolli.fasta
```

Ejemplo 3.119: Sustituir en cada línea todas apariciones de AAA por TTT en el archivo GCF_ecolli.fasta.

```
$ sed -i "s/AAA/TTT/" GCF_ecolli.fasta
```

Ejemplo 3.120: Sustituir todos los caracteres a mayúsculas en el archivo GCF_ecolli.fasta.

```
$ sed "s/./\U&/g" GCF_ecolli.fasta
```

Ejemplo 3.121: Convertir todos los caracteres a minúsculas en el archivo GCF_ecolli.fasta.

```
$ sed "s/./\L&/g" GCF_ecolli.fasta
```

Ejemplo 3.122: Convertir la cadena PDT a minúsculas en el archivo clostridium02.txt.

```
$ sed "s/PDT/\L&/g" clostridium02.txt
```

Capítulo 4
Formatos de archivo para bioinformática

Los datos obtenidos a partir del proceso de secuenciación son almacenados en grandes bases de datos de secuencias moleculares: el *National Center for Biotechnology Information* (NCBI [25]), la base de datos de secuencias genéticas de los Institutos Nacionales de Salud (NIH) de EE.UU. [30] (Genbank), el *European Molecular Biology Laboratory* (EMBL [23]), el *DNA Data Bank of Japan* (DDJB [20]) y la *Universal Protein Resource Knowledge Base* (UniProtKB [93], antes Swissprot). Tan solo GenBank, que es parte de la Colaboración Internacional de Bases de Datos de Secuencias de Nucleótidos, almacena 240,539,282 secuencias hasta octubre 2022 [30].

Los datos requieren ser almacenados, recuperados y procesados para realizar análisis biológicos. Por ello, es necesario utilizar formatos de archivo específicos. Un formato de archivo permite almacenar la información con una organización que facilita el almacenamiento y recuperación de información priorizando la eficiencia basada en la naturaleza de los datos y las operaciones requeridas para su procesamiento [54] [61]. En este capítulo se abordan los tipos de archivo utilizados en bioinformática para realizar análisis biológicos.

4.1. Formato `FASTA`

Este formato de archivo de secuencia de datos fue especificado en la documentación del programa `FASTA` (FAST-All), un programa de alineamiento rápido propuesto por W. R. Pearson [72] [24]. Estos archivos se utilizan para almacenar la codificación de secuencias de ADN que son anotadas empleando el estándar de la IUPAC (Inter-

national Union of Pure and Applied Chemistry [43]. La especificación del formato **FASTA** consiste de dos partes: a) la línea de descripción seguida de b) las líneas de la secuencia. La línea de descripción se diferencian de las líneas de secuencias porque inicia con el carácter mayor-que `>` al inicio. Las líneas de descripción contiene información general acerca de la secuencia del organismo que se esté tratando, así como su longitud (*length*). Las líneas de secuencia contiene una secuencia de nucleótidos utilizando sus iniciales (`A,C,G,T`). No están permitidas las líneas en blanco en las líneas de secuencia. En el archivo **FASTA** se recomienda que todas las líneas de texto tengan una longitud menor a 80 caracteres.

El Ejemplo 4.1 muestra una secuencia de ADN en formato **FASTA**.

Ejemplo 4.1: Secuencia de ADN en formato FASTA.

```
>NC_000913.3 Escherichia coli str. K-12 substr. MG1655. Creación: NCBI
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTTCTGAACTG
GTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGCGCACAGAC
AGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATTACCACAGGT
AACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGG
TAACGAGGTAACAACCATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCG
```

El código IUPAC (Tabla 4.1) para aminoácidos (proteínas) y ácidos nucleicos (ADN), se aplica en el formato **FASTA** con las siguientes excepciones; letras en minúscula están permitidas y son mapeadas a mayúsculas; se puede usar el símbolo guión '-' para representar un espacio en blanco o *gap* dentro de la secuencia de longitud indeterminada; y en secuencias de aminoácidos, los caracteres `U` y `\*` son aceptados. Cualquier dígito en la secuencia deberá ser eliminado o reemplazado por carácter un aceptado (ejemplo, usar `N` para residuos de Ácidos Nucleicos desconocidos o `X` para residuos de Aminoácidos desconocidos). La Tabla 4.2 muestra el código IUPAC para aminoácidos. El código IUPAC para aminoácidos es utilizado por programas que solicitan secuencias de aminoácidos.

Tabla 4.1: Código IUPAC para ácidos nucleicos.

Código	Significado	Código	Significado
A	adenosina	W	A/T (débil)
T	timidina	D	G/A/T
K	G/T(keto)	\-	*gap* de longitud indeterminada
M	A/C(amino)	G	guanina
B	G/T/C	U	uridina
V	G/C/A	Y	T/C (pirimidina)
C	citidina	R	G/A (purina)
N	A/G/C/T (cualquiera)	H	A/C/T
S	G/C (fuerte)		

Tabla 4.2: Código IUPAC para aminoácidos.

Código	Significado	Código	Significado
A	alanina	P	prolina
B	aspartato/asparagina	Q	glutamina
C	cistina	R	arginina
D	aspartato	S	serina
E	glutamato	T	treonina
F	fenilalanina	U	selenocisteina
G	glicina	V	valina
H	histidina	W	triptófano
I	isoleucina	Y	tirosina
K	lisina	Z	glutamato/glutamina
L	leucina	X	cualquiera
M	metionina	*	parada de la traducción
N	asparagina	-	*gap* de longitud indeterminada

El formato **FASTA** es también utilizado para almacenar o vincular una serie no contigua de secuencias genómicas en un *scaffold*, que consiste en secuencias separadas por espacios de longitud conocida. Las secuencias que están vinculadas son típicamente secuencias contiguas correspondientes a las superposiciones de lectura.

4.2. Formato FASTQ

El formato de archivo **FASTQ**, en el área de secuenciación de ADN, es utilizado como un formato común para el intercambio de datos entre herramientas. El formato **FASTQ** fue creado en el *Wellcome Trust Sanger Institute* por Jim Mullikin, gradualmente su uso cobro más relevancia, pero nunca fue documentado formalmente [16]. El formato **FASTQ**, al igual que el formato **FASTA**, son formatos de archivo que contienen secuencias de ADN. La información contenida en un archivo de este formato se codifica en ASCII. **FASTQ** provee una extensión al formato **FASTA**: la habilidad de almacenar calidades numéricas asociadas a cada nucleótido en una secuencia.

Ejemplo 4.2: Archivo cuya estructura sigue el formato FASTQ.

```
1 @071112_SLXA-EAS1_s_7:5:1:817:345
2 GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
3 +
4 IIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9IC
```

El Ejemplo 4.2 muestra el contenido de un archivo en formato **FASTQ**, descargado del respositorio **SRA** (*Sequence Read Archive*) de NCBI. Existen cuatro (4) tipos de líneas en el formato **FASTQ**. La primera, la línea del título, inicia con el carácter `@`, con frecuencia contiene solo un identificador. Este es un campo de formato libre sin restricción de longitud y permite incluir arbitrariamente anotaciones y comentarios, puede ser como el ejemplo de archivo **FASTA** que incluye un ID alternativo y el tamaño de la secuencia. El segundo tipo del líneas, son las líneas de secuencia, las cuales como en el formato **FASTA** pueden estar agrupadas en una sola línea. También al igual que el formato **FASTA**, no hay una limitación explícita en los caracteres esperados, pero la restricción del código IUPAC de caracteres para ADN o ARN ambiguos está vigente, y se usan letras en mayúsculas de manera convencional. En algunos contextos, el uso de letras minúsculas en combinación con mayúsculas o la inclusión del carácter *gap* pudiera tener sentido. El tercer tipo de líneas, es una línea que sirve como separador,

contiene únicamente al carácter +. El uso de esta línea es para indicar el final de las líneas de secuencia.

Finalmente, se encuentran las líneas de calidad, donde cada puntuación de calidad coincide con cada base en la secuencia. La calidad de las bases se define como la probabilidad de que la base correspondiente sea incorrecta. Se han utilizado dos ecuaciones diferentes para calcular la calidad Q de las bases. La más reciente, actualmente en uso, es la variante propuesta por Sanger para evaluar la confiabilidad de una base, también conocida como puntuación de calidad `Phred+33`:

$$Q_{Sanger} = -10\ log_{10}\ P$$

El pipeline de Solexa (una implementación de Illumina *Genome Analyzer*) usó anteriormente una ecuación diferente, actualmente en desuso, codificando las probabilidades como cuota estadística $P/(1-P)$:

$$Q_{Illumina} = -10\ log_{10}\ P/(1-P)$$

En el siguiente ejemplo se muestra como calcular la calidad Q de una base a partir de una probabilidad P dada por el secuenciador, haciendo uso de la fórmula propuesta por Sanger, para una probabilidad $P = 0.11$ tenemos que:

$$Q_{Sanger} = (-10)\ log_{10}(0.11)$$
$$Q_{Sanger} = (-10)(-0.9586)$$
$$Q_{Sanger} = 9.586$$

La calidad Q solo acepta valores decimales enteros, por lo tanto, con $Q_{Sanger} = 9.586$ se redondea al entero más cercano $Q_{Sanger} = 10$. Una calidad $Q_{Sanger} = 10$ significa que la probabilidad de error en las bases es de $\frac{1}{10}$ con una precisión aproximada de 90 %. Una calidad $Q_{Sanger} = 20$ significa que la probabilidad de error en las bases secuenciadas es de $\frac{1}{100}$ con una precisión aproximada de 99 %. De cada 100 bases secuenciadas se tiene la probabilidad de encontrar una base errónea. La precisión de las

bases se utiliza como métrica de precisión para las plataformas de secuenciación. La Tabla 4.12 muestra los valores para la calidad $\boldsymbol{Q}_{Sanger}$ obtenidos a partir de diferentes valores de $\boldsymbol{P}$.

La codificación del puntaje de la calidad *Phred* se realiza en ASCII y es dependiente de la tecnología de secuenciación utilizada, ya que estas entregan calidades diferentes, y por tanto, se posicionan de forma diferente en la escala. Al almacenar puntajes *Phred* como símbolos individuales (o *bytes*), se obtiene una codificación simple y eficiente en espacio.

Con el fin de que el formato sea legible y fácilmente editable, se restringe la elección a los símbolos ASCII imprimibles, de 32 a 126 en decimal, y debido a que el símbolo ASCII 32 codifica el símbolo "espacio en blanco", se usan los símbolos ASCII de 33 a 126 para codificar la calidad *Phred* para Sanger de 0 a 73. Por ejemplo, para codificar un puntaje de calidad Sanger igual a 0 se utiliza un símbolo ASCII 33).

El formato Solexa/Illumina (`Solexa+64`) puede codificar un puntaje de calidad -5 a 62, lo que corresponde a un ASCII de 59 a 126, (aunque en datos de lectura sin procesar se esperan puntuaciones de calidad Solexa/Illumina de -5 a 40 solamente). `Illumina 13.+` (`Phred+64`), una variante del formato `Solexa+64`, codifica el puntaje *Phred* con un símbolo ASCII, y puede contener puntajes de 0 a 62 (ASCII 64 a 126), pero a diferencia de `Solexa+64` este tiene un rango esperado de 0 a 40 solamente. En la Figura 4.1 se muestra un ejemplo de cómo las distintas tecnologías de secuenciación hacen uso de la escala *Phred*.

El formato **FASTQ** Sanger original no es perfecto. Los caracteres `@` y `+` tienen un uso doble como marcadores de línea y pueden aparecer en cualquier parte dentro de la cadena de calidad. No es posible realizar una simple indexación buscando las líneas que empiezan con `@`. Debido a esta complicación, la mayoría de las herramientas dan salida a archivos de formato **FASTQ** sin envolver la secuencia y la cadena de calidad con caracteres marcadores. Esta lectura media se compone de exactamente cuatro líneas (a veces líneas muy largas), ideal para que un analizador muy simple lidie con ellas.

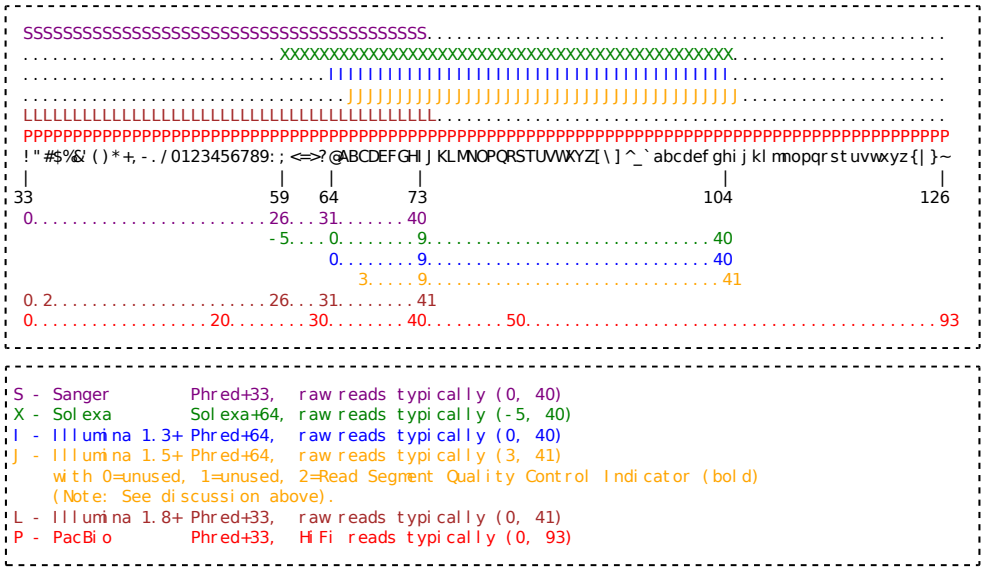

Figura 4.1: Escala *Phred*. Ejemplo de cómo las distintas tecnologías de secuenciación hacen uso de la escala *Phred*. Basada en Bioinformatics Workbook.

Debido a que los archivos **FASTQ** están codificados en formato ASCII, los caracteres de fin de línea normalmente siguen la convención del sistema operativo utilizado. El carácter de fin de linea estilo Unix (salto de línea, ASCII 10) es diferente al carácter de fin de línea utilizado por Windows (salto de línea y retorno, ASCII 13 y luego ASCII 10), por ello se recomienda que cualquier analizador debe soportar ambos saltos de línea.

4.3. Formato SAM

Un archivo en formato **SAM** (*Sequence Alignment/Map* [55]), por sus siglas en inglés, almacena los *reads* resultantes del proceso de secuenciación una vez que estos han sido alineados. Al igual que en los formatos **FASTA** y **FASTQ**, la información contenida en este archivo se codifica en ASCII [50].

En un archivo en formato **SAM** cada línea tiene un significado. En cada línea, la información está separada por un símbolo de tabulación; cada línea termina con un símbolo de fin de línea estilo Unix (salto de línea, ASCII 10). Un archivo **SAM** contiene

una sección de encabezado (opcional) y una sección de alineamiento. La sección de encabezado (si está presente) debe aparecer al inicio del archivo. Esta sección puede contener cero o más líneas que empiezan con el símbolo `@`, seguido de uno los códigos de tipo de registro de encabezado, los cuales se muestran en la Tabla 4.3.

Tabla 4.3: Formato **SAM**. Tipos de registro que pueden ser utilizados en la sección de encabezado.

Campo de tipo de registro	Descripción
`@HD`	Describe la versión **SAM** y coordenadas del genoma. Siempre debe estar presente.
`@SQ`	Diccionario de secuencias de referencia. El orden de las líneas.
`@RG`	Grupo de *read*. Se permiten múltiples líneas `@RG` no ordenadas.
`@PG`	Program.
`@CO`	Comentario de texto de una línea. Se permiten múltiples líneas

En la sección de encabezado, los campos en cada línea están delimitados por el carácter de tabulación. Exceptuando a las líneas que inician con `@CO`, cada campo sigue el formato `ETIQUETA:VALOR` donde `ETIQUETA` es una cadena de dos caracteres que define el formato y el contenido de `VALOR`.

Ejemplo 4.3: Sección de encabezado de un archivo en formato SAM.

```
1 @HD VN:1.4SO:coordinate
2 @SQ SN:scaffold1.1LN:663604
3 @SQ SN:scaffold2.1LN:584242
4 @SQ SN:scaffold3.1LN:541984
5 @RG ID:4PL:illumina PU:unit1LB:lib1SM:20
6 @PG ID:bwaPN:bwaVN:0.7.15-r1140 CL:/home/bioinformatica/
7 software/bwa-0.7.15/bwa mem /home/
8 @CO secuencias de un genoma de salmonella entérica
```

En el Ejemplo 4.3, la sección de encabezado de un archivo en formato **SAM**, la línea 1 muestra el campo `@HD`, las líneas 2, 3 y 4 son entradas del diccionario de secuencias de referencia, el grupo de lectura aparece en la línea 5, el (los) programa(s) utilizado(s) aparece(n) en la línea 6 y finalmente, en la línea 7 aparece el campo `@CO` con comentarios en formato libre.

En la sección de alineamiento, cada línea típicamente representa el alineamiento de un segmento. En esta sección, cada registro (línea) tiene 11 campos obligatorios para almacenar la información esencial del alineamiento tal como la posición del mapeo; y un número variable de campos opcionales `@PG` para almacenar información específica del alineador o información relevante para el usuario. En la Tabla 4.4 se muestran los campos obligatorios y opcionales de cada línea de alineamiento.

Ejemplo 4.4: Ejemplo de la sección de alineamiento de un archivo en formato SAM.

```
1 r001 99 Ref 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
2 r002 0 Ref 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *
3 r003 0 Ref 9 30 5S6M * 0 0 GCCTAAGCTAA *
4 r004 0 Ref 16 30 6M14N5M * 0 ATAGCTTCAGC *
5 r003 2064 Ref 29 17 6H5M * 0 TAGGC *
6 r001 147 Ref 37 30 9M = 7 -39 CAGCGGCAT *
```

En el Ejemplo 4.4 se muestra la sección de alineamiento de un archivo en formato **SAM**. Las líneas 1 y 6 muestran los campos del *read* `001`. Los campos del *read* `002` son mostrados en la línea 2. En las líneas 3 y 5 se muestran los campos del *read* `003`. A continuación se explicará el contenido de la línea 1. El campo **QNAME** es `r001`. El valor `99` del campo **FLAG**, es la suma de los valores `1+32+64` explicados a detalle en la Tabla 4.9, significa que el **read** está pareado de forma inversa (valor 1), que el **SEQ** del siguiente segmento está complementado de forma inversa (valor 32) y que es el primer segmento de la plantilla (valor 64). El campo **RNAME** almacena el nombre de la secuencia de referencia, en este caso, el valor es `ref`. La posición de mapeo más a la izquierda, campo **POS**, tiene valor de 7. El campo **MAPQ** indica que la calidad del mapeo tiene un valor de 30 en escala *Phred*, esto significa un 99.99 % de precisión. El campo **CIGAR** tiene valor de `8M2I4M1D3M`, este es un reporte de alineamiento que especifica los tipos de operaciones que se necesitaron para alinear la lectura con la ubicación específica del genoma. Las operaciones de alineamiento que se especifican son: `M` indica coincidencia con la referencia en el alineamiento, `I` para inserción, `D` para eliminación. El siguiente campo es **RNEXT**, tiene el valor `=` significa que es el mismo

que el campo **RNAME**. El siguiente campo es **PNEXT** que tiene valor de 37. El 39 es el valor del campo **TLEN**. El campo **SEQ** tiene como valor la secuencia TTAGATAAAGGATACTG. Finalmente, el campo **QUAL** se refiere a la calidad medida utilizando Sanger del formato **FASTQ**, en este ejemplo, el * significa que no tiene asignado valor.

Tabla 4.4: Formato **SAM**. Campos contenidos en cada línea del alineamiento.

No.	Nombre	Descripción
1	**QNAME**	Nombre del segmento de secuencia o plantilla.
2	**FLAG**	Bandera de bits.
3	**RNAME**	Nombre de la secuencia de referencia
4	**POS**	Posición/coordenada de mapeo más a la izquierda, inicia en 1.
5	**MAPQ**	Calidad del mapeo en escala *Phred*.
6	**CIGAR**	*Compact Idiosyncratic Gapped Alignment Report*. Es una secuencia que define el número de las coincidencias, no coincidencias y eliminaciones de un alineamiento.
7	**RNEXT**	Nombre de referencia del *read-mate* o del *next-read* ('=' si es la misma que RNAME)
8	**PNEXT**	Posición del *read-mate* o *next-read*, inicia en 1.
9	**TLEN**	Longitud (LENgth) de la platilla observada (tamaño de inserción)
10	**SEQ**	Segmento de secuencia o la secuencia de consulta es la misma cadena que la referencia.
11	**QUAL**	Medida de la calidad de una base dentro de una secuencia.

A continuación, se realizará la descripción detallada de las secciones de encabezado y de alineamiento de un archivo en formato SAM. El contenido de esta sección permitirá profundizar en el conocimiento del formato SAM y de cada una de sus secciones con mayor profundidad y precisión.

1. **QNAME**: Nombre de la plantilla de consulta. Las lecturas/segmentos que tienen idéntico **QNAME** se considera que proceden de la misma plantilla del fragmento de ADN. Un * en este campo significa que la información no está disponible. En el archivo **SAM**, una lectura puede ocupar múltiples líneas de alineamiento, cuando su alineamiento es quimérico o cuando son proporcionados múltiples mapeos. Un alineamiento quimérico se representa como un conjunto de alineamientos

que no tiene sobrelapamientos grandes. Se considera una alineamiento quimérico cuando el alineamiento de una lectura no puede representarse como alineamiento lineal.

2. **FLAG**: Combinación de banderas representadas a nivel de *bits* (*bitwise flags*). Este campo se expresa mediante una codificación consistente de 12 dígitos binarios (*bits*), que permite hacer más eficiente su almacenamiento y facilita su decodificación. Cada dígito funciona como una bandera que se enciende asignando un `1` o se apaga asignando un `0` y tiene un significado como se muestra en la Tabla 4.9. Cuando se encienden los *bits* requeridos para describir el **SEQ**, el número binario resultante es convertido a sistema decimal y se almacena en el archivo con formato **SAM**.

Por ejemplo, si se desea calcular el valor del campo **FLAG** para un alineamiento descrito mediante los siguientes valores:

- El *read* está pareado en la secuencia (000000000001_2) bit encendido en la posición 1.
- Cada segmento está alineado apropiadamente de acuerdo al alineador (000000000010_2) bit encendido en la posición 2.
- El *read* complementado de forma reversa (000000010000_2) bit encendido en la posición 5.
- Es el primer segmento en la plantilla (000001000000_2) bit encendido en la posición 7.

El número en binario resultante al combinar los bits activados (aplicar la operación XOR a los valores binarios), basados en la Tabla 4.9, es 000001010011_2 que al convertirlo a sistema decimal se obtiene un valor 83_{10}. Este valor es almacenado en el campo **FLAG** del archivo en formato **SAM**.

En cambio, si se lee un archivo en formato **SAM**, se requiere decodificar el valor del campo **FLAG** para identificar las operaciones realizadas sobre ese *read*. Esto se realiza de la siguiente manera, se ubica el valor del campo **FLAG** (segundo campo del alineamiento), se convierte a base 2, cada dígito (*bit*) de este número obtenido es una bandera. Los bits que están encendidos nos permiten identificar las operaciones realizadas en base a la Tabla 4.9.

Por ejemplo, si en un alineamiento, el valor correspondiente al segundo campo, es decir al campo **FLAG**, tiene un valor de 83_{10}, al convertir el número 83_{10} a base 2, se obtiene el número 000001010011_2. Los bits que están encendidos nos permiten identificar las operaciones realizadas con base en la Tabla 4.9. Significa que de manera combinada están activadas las operaciones mostradas en la Tabla 4.10.

Al aplicar la operación `XOR` a dichos números se obtiene el número 000001010011_2, que al convertir a base 10 se obtiene el valor 83_{10}. Existen herramientas disponibles en línea, de acceso libre que explican las banderas a nivel de bits utilizadas en el campo **FLAG**.

A continuación, se enumeran consideraciones respecto al campo **FLAG**:

- Por cada *contig* en un archivo en formato **SAM**, se requiere que una y solo una línea asociada con la lectura satisfaga `FLAG & 100100000000 == 0`. Esta línea es llamada la línea primaria de la lectura.

- Si el *bit* en la posición 1 (000000000000_2) está apagado, no se pueden realizar no se pueden realizar suposiciones sobre los *bits* en la posiciones 2 (000000000010_2), 4 (000000001000_2), 6 (000000100000_2), 7 (000001000000_2) y 8 (000010000000_2).

- El *bit* en la posición 3 (000000000100_2) es el único lugar confiable para decir si la lectura no está mapeada. Si el *bit* en la posición 3 (000000000100_2) está encendido, no se pueden realizar suposiciones acerca de **RNAME**, **POS**, **CIGAR**, **MAPQ**, y los *bits* en la posición 2 (000000000010_2), 9 (000100000000_2) y 12 (100000000000_2).

- El *bit* en la posición 5 (000000010000_2) indica si el **SEQ** ha sido reverso complementado y **QUAL** invertido. Cuando el *bit* en la posición 3 (000000000100_2) está desactivado, esto corresponde a la hebra de la cual el segmento ha sido mapeado. Cuando el *bit* en la posición 3 (000000000100_2) está activado, esto indica que si la lectura no mapeada está almacenada en su orientación original tal como salió del secuenciador.

- Los *bits* en la posición 6 (000001000000_2) y 8 (000010000000_2) reflejan el orden de lectura dentro de cada molde inherente a la tecnología de secuenciación utilizada. Si ambos *bits*, en la posición 2 /(000001000000_2) y en la posición 8 (000010000000_2) están encendidos, el *read* es parte de un molde linear, pero este no es ni el primero ni el último *read*. Si ambos están apagados, se desconoce el índice del *read* en el molde. Esto puede suceder para moldes no lineales o cuando esta información se ha perdido durante el procesamiento de datos.

- El *bit* en la posición 9 (000100000000_2) indica que el alineamiento no será usado en ciertos análisis cuando las herramientas en uso tienen conocimiento de este bit. Típicamente se usa para indicar/marcar mapeos alternativos cuando se presentan múltiples mapeos en un **SAM**.

- El *bit* en la posición 12 (100000000000_2) indica que la línea del alineamiento correspondiente es parte de un alineamiento quimérico. Una línea marcada con el bit en la posición 12 (100000000000_2) es llamada línea suplementaria.

- Los *bits* que no son listados en la Tabla 4.9 están reservados para uso futuro. Ellos no deberían ser usados o asignarles valor cuando se realiza la escritura y, en la lectura, deben ser ignorados por el software actual.

3. **RNAME**: Nombre de la secuencia de referencia de alineamiento. Si las líneas `@SQ` del encabezado están presentes, **RNAME** debe estar presente en una de las etiquetas `SQ-SN`. Un segmento no mapeado sin coordenadas tiene un `*` en este campo.

Sin embargo, un segmento no mapeado puede tener también una coordenada ordinaria tal que puede ser colocada en una posición deseada después del ordenamiento. Si RNAME es *, no se pueden realizar conjeturas acerca de **POS** y **CIGAR**.

4. **POS**: Posición de mapeo más a la izquierda basada en la primera base coincidente. La primera base en una secuencia de referencia tiene una coordenada 1. **POS** se establece como `0` para una lectura no asignada sin coordenadas. Si **POS** es `0`, no se pueden hacer suposiciones sobre **RNAME** y **CIGAR**.

5. **MAPQ**: Mapeo de calidad (*Mapping Quality*). Equivale a la calidad *Phred* redondeada al entero más cercano. Un valor de `255` en este campo indica que el mapeo de calidad no está disponible.

6. **CIGAR**: *Compact Idiosyncratic Gapped Alignment Report*, por sus siglas en inglés. La cadena **CIGAR** se refiere a un sistema de notación para variantes. La cadena **CIGAR** se forma de una secuencia estructurada de la siguiente manera: `[valor]` `operación`, donde `[valor]` es un número entero de bases seguido de la operación realizada que se representa mediante un símbolo, tal como se muestra en la Tabla 4.11. En el Ejemplo 4.5 se muestra un ejemplo de cálculo del campo **CIGAR** en formato **SAM**.

En el estándar **CIGAR** las operaciones son `M` (*Match/mismatch*), I (*Insertion*) y `D` (*Deletion*). Mientras que en el estándar **CIGAR** extendido además se incluyen las opciones `N` (bases 'saltadas' en la referencia), `S` (*soft clipping*), `H` (*hard clipping*), y `P` (*Padding*).

Cuando una base no se encuentra disponible en un alineamiento, se representada mediante un * y se utiliza la operación `I` o `P` para indicar si es en el *read* o en la referencia donde hace falta.

Las operaciones **CIGAR** se muestran en la Tabla 4.11 (se asigna * si no está disponible):

- `H` solo puede estar presente como la primera o la última operación.
- `S` podría solo tener operaciones H entre ellas y los finales de la cadena **CIGAR**.
- Para alineamiento de un genoma nRNA, una operación N representa un *intron*. Para otros tipos de alineamientos, la interpretación de `N` no está definida.
- La suma de las longitudes de las operaciones `M/I/S/=/X` deberá ser igual a la longitud de **SEQ**.

La sección de encabezado, si está presente, debe aparecer al inicio del archivo. Esta sección puede contener cero o más líneas que empiezan con el símbolo `@`, seguida de uno los códigos de tipo de registro de encabezado, los cuales se muestran en la Tabla 4.5. En la sección de encabezado, en cada línea, la información está separada por un símbolo de tabulación; cada línea termina con un símbolo de fin de línea (salto de línea y retorno, ASCII 13 y luego ASCII 10). Un registro es un conjunto de campos, en un archivo en formato **SAM** los campos que forman cada registro se ubican en la misma línea, esto también aplica para la sección encabezado. Exceptuando a las líneas que inician con `@CO`, cada campo de la sección encabezado sigue el formato `ETIQUETA:VALOR` donde `ETIQUETA` es una cadena de dos caracteres que define el formato y el contenido del campo `VALOR`. En las Tabla 4.6, y 4.7 se describen los tipos de registro del encabezado que pueden ser usados y sus etiquetas predefinidas. Las etiquetas marcadas con * son obligatorias; por ejemplo, cada línea de encabezado con `@SQ` debe tener los campos `SN` y `LN`.

Al igual que los campos opcionales del alineamiento mostrados en la Tabla 4.4, es posible agregar libremente nuevas etiquetas para campos de datos adicionales.

Las etiquetas que contienen letras en minúsculas son reservadas para experimentación con nuevas etiquetas y no se prevé que estas nuevas etiquetas sean definidas

formalmente en ninguna versión futura de la especificación del formato **SAM**. En computación, una expresión regular es un patrón o plantilla utilizado en una operación de búsqueda o comparación de textos (cadenas). Su nombre viene de la teoría matemática en la que se basa, con frecuencia el término es abreviado como *regex* o *regexp* [27]. Las líneas de la sección de encabezado pueden contener una cadena formada por tres partes: a) la primera parte formada por el símbolo '@' seguido de dos letras mayúsculas. b) la segunda parte se conforma de una o más cadenas cada una de esas cadenas formada por un tabulador (se teclea presionando la tecla tab) seguido de una letra (mayúscula o minúscula), seguida de una letra (mayúscula o minúscula) o un dígito, seguido del símbolo ':' (dos puntos), seguido de al menos uno de estos símbolos '-' o '~'. c) La última parte, contiene al carácter '/'.

Tabla 4.5: Formato **SAM**. Tipo de registro `@HD`.

Etiqueta	Descripción
`VN`*	Versión de formato. Es un número que puede contener decimales. Se utiliza el punto para separar la parte entera de la parte decimal.
`SO`	Forma en que se ordenaron los alineamientos. Valores válidos: `unknown` (predeterminado), `unsorted`, `queryname` y `coordinate`. Para el ordenamiento `coordinate`, primero se considera como primer criterio el campo **RNAME** basado en el orden de las líneas `@SQ` del encabezado. También se considera el campo **POS** como el segundo criterio de ordenamiento. El campo **POS** es la clave menor de ordenamiento. Para alineamientos con igual **RNAME** y **POS**, el orden es arbitrario. Todos los alineamientos con * en el campo **RNAME** siguen a alineamientos con algún otro valor o de lo contrario están en orden arbitrario.
`GO`	Agrupamiento de alineamientos, indica que registros similares del alineamiento son agrupados juntos pero el archivo no necesariamente está ordenado completamente. Valores válidos: `none` (predeterminado), `query` (alineamientos están agrupados por **QNAME**), y `reference` (alineamientos están agrupados por **RNAME/POS**).

En el formato **SAM**, cada línea de alineamiento típicamente representa el alineamiento de un segmento. Cada línea tiene 11 campos obligatorios. Estos campos siempre

aparecen en el mismo orden y deben estar presentes, pero sus valores pueden ser `0` o `*` (dependiendo del campo) si la información correspondiente no está disponible. La Tabla 4.8 da una descripción general de los campos obligatorios en el formato SAM.

Tabla 4.6: Formato **SAM**. Etiquetas que pueden ser usadas después del tipo de registro del encabezado `@RG`.

Etiqueta	Descripción
`ID*`	Identificador del grupo de lectura. Cada línea `@RG` debe tener un único `ID`. El valor de `ID` es usado en las etiquetas `RG` de los registros de alineamiento. Debe ser único a lo largo de todos los grupos de lectura en la sección de encabezado. Los `IDs` de los grupos de lectura pueden ser modificados cuando se combinan archivos **SAM** con el propósito de manejar las colisiones.
`CN`	Nombre del centro de la secuenciación que produce la lectura.
`DS`	Descripción.
`DT`	Fecha en que se realizó la ejecución. Valor expresado.
`FO`	Orden de flujo. La matriz de las bases de nucleótidos que corresponde a los nucleótidos usados para cada flujo de cada lectura. Flujos Multi-base son codificados en formato IUPAC, y los no-nucleótidos fluyen por la expresión: `/\*|[ACMGRSVTWYHKDBN]+/`. Esta expresión regular permite cadenas que son un `*` o cadenas que tienen al menos una de estos caracteres: `ACMGRSVTWYHKDBN`.
`KS`	La matriz de las bases de nucleótido que corresponden a la secuencia clave de cada lectura.
`LB`	Biblioteca.
`PG`	Programas usados para el procesamiento del grupode lectura.
`PI`	Tamaño promedio de la inserción predicha.
`PL`	Plataforma de secuenciación utilizada para producir los *reads*. Valores válidos: `CAPILLARY, LS454, ILLUMINA, SOLID, HELICOS, IONTORRENT, ONT,` y `PACBIO`.
`PM`	Modelo de plataforma. Texto en formato libre para proveer más detalles de la plataforma o tecnología utilizada.
`PU`	Unidad de plataforma (*p. ej.* Celda de flujo paraIllumina o *Slide* para SOLiD). Identificador único.
`SM`	Muestreo. Utilice el nombre del grupo donde se está secuenciando una agrupación.

7. **RNEXT**: Nombre de la secuencia de referencia del alineamiento primario de la siguiente lectura en la plantilla. Para la última lectura, la siguiente lectura es la primera lectura en la plantilla. Si las líneas `@SQ` del encabezado están presentes, **RNEXT** (si no tiene valor de * o =) debe estar presente en una de las etiquetas `SQ-SN`. Este campo se establece como * cuando la información no está disponible, y se establece como = si RNEXT es idéntico a **RNAME**. Si no tiene asignado el valor =, y el siguiente *read* en la plantilla tiene un mapeo primario (ver también el bit `100000000` en **FLAG**), este campo es idéntico a **RNAME** en la línea primaria del siguiente *read*. Si **RNEXT** tiene asignado el valor *, no se pueden hacer suposiciones sobre **PNEXT** y el *bit* `100000`.

8. **PNEXT**: Posición del alineamiento primario del siguiente en la plantilla. Establecer como `0` cuando la información no esté disponible. Este campo equivale a **POS** en la línea primaria de la siguiente lectura. Si **PNEXT** es 0, no se pueden hacer suposiciones sobre **RNEXT** y el bit `100000`.

9. **TLEN**: Longitud del plantilla observada (*observed Template LENgth*). Si todos los segmentos son mapeados para la misma referencia, la longitud de plantilla observada no firmada es igual al número de bases desde la base mapeada más a la izquierda hasta la base mapeada más a la derecha. El segmento más a la izquierda tiene un signo más (+) y el de más a la derecha un signo menos (-). El signo de segmentos en el medio es indefinido. Este es establecido como `0` para una plantilla de un solo segmento o cuando la información no está disponible.

10. **SEQ**: Es la secuencia del segmento. Este campo puede ser un * cuando la secuencia no está almacenada. Si no un *, la longitud de la secuencia debe igualar la suma de las longitudes de las operaciones `M/I/S/=/X` en el campo **CIGAR**. Un = denota que la base es idéntica a la base de referencia. No se realizan suposiciones en el caso de bases en mayúsculas o minúsculas.

11. **QUAL**: La etiqueta **QUAL** se refiere a la medida de la calidad de una base dentro de una secuencia, a cada base en la secuencia se le asigna una calidad `Q`, la cual es un mapeo entero de `P` (la probabilidad de que la base correspondiente sea incorrecta). Consulte la Sección 4.2 formato **FASTQ**.

Para el campo **QUAL** se asigna un símbolo del código ASCII correspondiente al valor de la calidad de la base Q más un valor fijo de `33` para datos de secuenciación *Sanger* o al valor de la calidad Q más un valor fijo de `64` para Illumina. Este campo puede tener asignado un valor $*$ cuando la calidad no está disponible. En cambio, cuando la calidad sí está presente, **SEQ** no debe tener asignado un $*$ y la longitud de la cadena de calidad debería ser igual a la longitud de **SEQ**.

Todos los campos opcionales cumplen con el formato `ETIQUETA:TIPO:VALOR`, donde `ETIQUETA` es una cadena de dos caracteres: el primero es una letra (minúscula o mayúscula) seguido de otra letra (minúscula o mayúscula) o un dígito. Cada `ETIQUETA` puede aparecer solo una vez en la línea de alineamiento. Las `ETIQUETAS` que contienen letras minúsculas se utilizan para experimentación con nuevas etiquetas y no se prevé que estas nuevas etiquetas sean definidas formalmente en ninguna versión futura de la especificación del formato **SAM**. En un campo opcional, `TIPO-VALORG` es una sola letra sensible a mayúsculas la cual define el formato del `VALOR`, la Tabla 4.13 muestra las letras que se utilizan en el campo `TIPOVALOR`.

Para un entero o un arreglo numérico (tipo `B`) [11], la primera letra indica el tipo de datos numéricos en el siguiente arreglo separado por comas. La letra puede ser una de estas `cCsSiIf`, correspondiente a `int8_t` (numero entero de 8 *bits* con signo), `uint8_t` (numero entero de 8 *bits* sin signo), `int16_t`, `uint16_t`, `int32_t`, `uint32_t` y *float*, respectivamente. Durante la importación/exportación, el tipo de elemento puede ser cambiado si el nuevo tipo es también compatible con el arreglo.

Las etiquetas predefinidas se describen en el documento técnico «Especificación de los campos opcionales del *Sequence Alignment/Map*» [80]. También en ese documen-

to se pueden consultar más detalles de los campos de etiquetas estándar existentes y las convenciones acerca de la creación de nuevas etiquetas que puedan ser de interés general. Las etiquetas iniciarán con X, Y o Z y las etiquetas que contengan letras minúsculas en cualquier posición están reservadas para uso local y no serán formalmente definidas en futuras versiones de estas especificaciones.

Para el campo **QUAL** se asigna un símbolo del código ASCII correspondiente al valor de la calidad de la base Q más un valor fijo de 33 para datos de secuenciación *Sanger* o al valor de la calidad Q más un valor fijo de 64 para Illumina. Este campo puede tener asignado un valor $*$ cuando la calidad no está disponible. En cambio, cuando la calidad sí está presente, SEQ no debe tener asignado un $*$ y la longitud de la cadena de calidad debería ser igual a la longitud de **SEQ**.

Tabla 4.7: Formato **SAM**. Etiquetas que pueden ser utilizadas después del tipo de registro del encabezado @PG.

Etiqueta	Descripción
ID*	Identificador del registro del programa. Cada @PG debe tener un único ID. El valor de ID es usado en la etiqueta PG de alineamiento y en las etiquetas PP de otras líneas @PG. Los IDs de PG pueden ser modificados cuando se combinan archivos **SAM** con la finalidad de manejar colisiones.
PN	Nombre del programa.
CL	Línea de comando.
PP	Previos @PG-ID. Debe coincidir con la etiqueta ID de PP otro encabezado @PG. Los registros @PG deben estar concatenados usando la etiqueta PP, con el último registro en la cadena que no tenga la etiqueta PP. Esta cadena define el orden de los programas que se han aplicado al alineamiento. Los valores PP pueden ser modificados cuando se combinan los archivos para manejar colisiones de PG IDs. El primer registro PG en una cadena (p. ej. el único referido por la etiqueta PG en un registro **SAM**) describe el programa más reciente que ha operado sobre el registro **SAM**. El siguiente registro PG en la cadena describe el más reciente programa que ha operado sobre el registro **SAM**. El PG ID sobre el registro **SAM** no es requerido para referirse al más reciente registro en la cadena. Puede referirse a cualquier registro PG en una cadena, lo que implica que el registro **SAM** ha sido operado por el programa sobre ese registro PG, y el(los) programa(s) referido(s) a través de la etiqueta PP.

Etiqueta	Descripción
DS	Descripción.
VN	Versión del programa.

Ejemplo 4.5: Formato SAM. Ejemplo del cálculo del campo CIGAR.

```
Coor 12345678901234 5678901234567890123456789012345 Ref AGCATGTTAGATAA**
    GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT
r001/1 TTAGATAAAGGATA*CTG
r002 aaaAGATAA*GGATA
r003 gcctaAGCTAA
r004 ATAGCT..............TCAGC
r003 ttagctTAGGC
r001/2 CAGCGGCAT
@SQ SN:ref LN:45
```

TODO ESTA TABLAAAA

4.4. Formato GFF/GFF3

GFF (*Generic Feature Format*) es utilizado para representar características genómicas, genes en un cromosoma, pero no cubre todos los casos que se necesitaban, eventualmente diferentes grupos escogieron extender este formato con su propia implementación, entonces nuevos formatos derivados aparecieron volviéndose comunes y creando confusión.

El formato **GFF3** (*Generic Feature Format* v3) es la tercera versión de **GFF**, fue creado en intento de estandarizar y unificar las extensiones más comunes de **GFF** también mantiene compatibilidad con versiones anteriores del formato.

La estructura básica del formato **GFF3** consiste de información genómica estructurada en nueve columnas o campos que se muestran en la Tabla 4.14. Cada columna es explicada a continuación.

Columna 1: `SEQid.ID` del punto de referencia utilizado para establecer el sistema de coordenadas de la función actual. Los ID pueden contener cualquier carácter, pero

Tabla 4.8: Descripción general de los campos obligatorios en el formato **SAM**.

Col	Campo	Tipo	Regex/Rango	Descripción breve
1	**QNAME**	*String*	`[!-?A-~]{1,254}`	Nombre de la plantilla de consulta.
2	**FLAG**	*Int*	$[0, 2^{16} - 1]$	Combinación de banderas representadas a nivel de *bits*.
3	**RNAME**	*String*	`\*|[:rname: ^ *=][:rname:]*`	Nombre de la secuencia de referencia.
4	**POS**	*Int*	$[0, 2^{31} - 1]$	
5	**MAPQ**	*Int*	$[0,\ 2^{8} - 1]$	Calidad del mapeo.
6	**CIGAR**	*String*	`\*|([0-9]+[MIDNSHPX=])`	La cadena **CIGAR** define las coincidencias/no coincidencias y eliminaciones (o espacios) del alineamiento con respecto a una secuencia de referencia.
7	**RNEXT**	*String*	`\*|=|[:rname: ^ *=][:rname:]*`	Nombre de referencia del *read* mate o del *read* siguiente.
8	**PNEXT**	*Int*	$[0,\ 2^{31} - 1]$	Posición del read mate o siguiente.
9	**TLEN**	*Int*	$[-2^{31} + 1,\ 2^{31} - 1]$	Longitud de la platilla observada.
10	**SEQ**	*String*	`\*|[A-Za-z=.]+`	Secuencia del segmento.
11	**QUAL**	*String*	`[!-~]+`	Medida de la calidad de una base dentro de una secuencia.

deben escapar de cualquier carácter que no esté en el conjunto `[a-zA-Z0-9.:^*$@!+_?-|]`. En particular, los identificadores no pueden contener espacio en blanco sin desencadenarse y no deben comenzar con `>`.

Columna 2: **source**. La fuente u origen es un calificador de texto libre destinado a describir el algoritmo o procedimiento operativo que generó esta característica. Normalmente, este es el nombre un programa de software, como *Genescan* o un nombre de base de datos, como *Genbank*. La fuente se utiliza para extender la ontología de características añadiendo un calificador al tipo que crea un nuevo tipo compuesto que es una subclase del tipo en la columna de tipo.

Tabla 4.9: Formato **SAM**. Campo **FLAG**, definición de cada bit.

Valor decimal	Valor binario	Valor hexadecimal	Descripción
1	000000000001_2	$0x1_{16}$	El *read* está pareado en la secuencia.
2	000000000010_2	$0x2_{16}$	Cada segmento está alineado apropiadamente de acuerdo al alineador.
4	000000000100_2	$0x4_{16}$	El *read* no está mapeado.
8	000000001000_2	$0x8_{16}$	El siguiente segmento en el molde no está mapeado.
16	000000010000_2	$0x10_{16}$	El *read* es complementado de forma inversa.
32	000000100000_2	$0x20_{16}$	**SEQ** del siguiente segmento en el molde es complementado de forma inversa.
64	000001000000_2	$0x40_{16}$	El *read* es el primer segmento en la plantilla.
128	000010000000_2	$0x80_{16}$	El *read* es el último segmento en la plantilla.
256	000100000000_2	$0x100_{16}$	Es un alineamiento secundario.
512	001000000000_2	$0x200_{16}$	No pasa los filtros, tal como los controles de calidad la plataforma.
1024	010000000000_2	$0x400_{16}$	PCR duplicado óptico.
2048	100000000000_2	$0x800_{16}$	Alineamiento complementario.

Columna 3: **type**. El tipo de la entidad (anteriormente denominado método). Esto es un término de la Ontología de Secuencia o un número de acceso. La última alternativa se distingue usando la sintaxis SO:000000.

Columnas 4 y 5: `start` y `end`. Coordenadas de secuencia de inicio (`start`) y de fin (`end`) dentro del genoma. El inicio es siempre menor o igual al final. Para las características que cruzan el origen de un rasgo circular (por ejemplo, la mayoría de

Tabla 4.10: Formato SAM. Campo FLAG, ejemplo del uso de banderas a nivel de bits.

Valor decimal	Valor binario	Valor hexadecimal	Descripción
1	000000000001_2	$0x1_{16}$	El *read* está pareado en la secuencia.
2	000000000010_2	$0x2_{16}$	Cada segmento está alineado apropiadamente de acuerdo al alineador.
16	000000010000_2	$0x10_{16}$	El *read* es complementado de forma inversa.
64	000001000000_2	$0x40_{16}$	El *read* es el primer segmento en la plantilla.
83	00001010011_2	$0x53_{16}$	Valor encontrado en el campo FLAG.

Tabla 4.11: Campo CIGAR, representación de las operaciones permitidas.

Operación	BAM	Descripción
M	0	Coincidencia en el alineamiento (puede ser que una secuencia haya tenido o no coincidencia)
I	1	Inserción a la referencia debido a que la secuencia de búsqueda no coincide.
D	2	Eliminación de la referencia debido a que la secuencia de referencia no coincide.
N	3	Región 'saltada' de la referencia.
S	4	Soft clipping (Secuencias añadidas presentes en la SEQ). Bases de 5' a 3' que no forman parte del alineamiento.
H	5	Hard clipping (secuencias añadidas NO presentes en la SEQ). Bases de 5' a 3' que no forman parte del alineamiento, estas bases son removidas del *read* al transformarlo en un formato BAM.
P	6	Relleno (eliminación silenciosa de la referencia amortiguada)
=	7	Coincidencia en la secuencia.
X	8	No coincidencia en la secuencia.

los genomas bacterianos, plásmidos y algunos genomas virales), el requisito de que el comienzo sea menor o igual al extremo se satisface haciendo que:

Tabla 4.12: Valores de calidad Q.

Probabilidad P	Calidad Q_{Sanger}	Precisión
0.1	10	90 %
0.01	20	99 %
0.001	30	99.9 %
0.0001	40	99.99 %

Tabla 4.13: Formato **SAM**. `TIPOVALOR` que se utilizan en los campos opcionales del alineamiento.

TIPOVALOR	Regex coincidente con valor	Descripción
`A`	[!-~]	Símbolos ASCII imprimibles.
`i`	[-+]?[0-9]+	Entero con signo
`f`	[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?	Número real.
`Z`	[!-~]*	Cadena imprimible, incluyendo espacio.
`HA`	([0-9A-F][0-9A-F])*	Arreglo de caracteres que se interpreta como un número en formato hexadecimal
`B`	[cCsSiIf](,[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?)+	Cadena de caracteres que se interpreta como un entero o arreglo de enteros.

Tabla 4.14: `GGF3`. Estructura básica.

1	2	3	4	5	6	7	8	9
Genome	.	gene	301	2169	.	+	.	ID=SPAC1F7.08;Name=iron %20 transport %20multi..

$$end = P + L$$

Donde: P es la posición del extremo y L es la longitud de la característica de punto de referencia.

Para características de longitud cero, como sitios de inserción, comienzan iguales y el sitio implícito está a la derecha de la base indicada en la dirección del punto de referencia.

Columna 6: `score`. La puntuación de la característica, un número real. En versiones anteriores del formato, la semántica de la partitura está mal definida. Se recomienda que los valores E se utilicen para características de similitud de secuencia, y que los valores P se utilicen para las características de predicción de genes *ab initio*.

Columna 7: `strand`. El filamento de la característica. Carácter `+` para la hebra positiva (relativa al hito), `-` para la hebra negativa, y `.` para las características que no están fijas. En adición, `?` se puede utilizar para las características cuyo *strandedness* es relevante, pero desconocido.

Columna 8: `fase`. Para las características del tipo `CDS`, la fase indica dónde empieza la función con referencia al marco de lectura. La fase es uno de los enteros `0, 1` o `2`, indicando el número de bases que deben ser eliminadas desde el comienzo de esta característica para alcanzar la primera base del siguiente codón. Una fase de `0` indica que el siguiente codón comienza en la primera base de la región descrita por la línea actual. Una fase de `1` indica que el siguiente codón comienza en la segunda base de esta región. Una fase de `2` indica que el codón comienza en la tercera base de esta región. Esto no se debe confundir con el marco, que es simplemente iniciar el módulo 3. Para las características de la cadena delantera, la fase se cuenta desde el campo de inicio. Para las características de la cadena inversa, la fase se cuenta desde el campo final. La fase es requerida para todas las funciones del `CDS`.

Columna 9: atributtes. La lista de atributos de entidades en el formato `etiqueta =valor`. Varios pares `etiqueta=valor` están separados por punto y coma. Las reglas de escape de `URL` se utilizan para etiquetas o valores que contienen los siguientes caracteres: `,=;`. Los espacios están permitidos en este campo, pero los carácteres de tabulación deben ser reemplazados por el carácter de codificación URL: `%09`. Los

valores de atributo no necesitan ni deben estar entrecomillados. Los analizadores sintácticos deben incluir las comillas como parte del valor y no eliminarlas.

Los archivos **GFF3** también pueden incluir secuencias en formato **FASTA** al final del archivo. Las secuencias son precedidas por una línea `##FASTA`. Esta sección es opcional, si está presente, la sección de secuencia puede definir secuencia para cualquier punto de referencia utilizado en la columna 1 (el marco de referencia).

Capítulo 5
Genómica Comparativa

Se le conoce como genómica comparativa al proceso por el cual se establecen similitudes y diferencias estructurales y funcionales en una colección de genomas [9]. Actualmente, la genómica comparativa está siendo ampliamente utilizada por la comunidad científica debido a la facilidad de acceso a genomas y la reducción del costo de secuenciación tal como muestran las Figura 1.2, y 1.1; [30]. Existen repositorios públicos como NCBI [25], GenBank [30], EMBL [23], DDJB [20] UniProtKB [93] donde se almacenan datos biológicos, los cuales pueden ser descargados y utilizados por el público en general, los científicos o expertos libremente. Las principales aplicaciones de la genómica comparativa incluyen la identificación de genes, establecimiento de relaciones de parentesco, identificación de mutaciones y mecanismos evolutivos [9]. Las etapas generales para llevar a cabo un análisis mediante genómica comparativa de genomas bacterianos, conlleva la evaluación inicial de la calidad de las lecturas de secuenciación, ensamblado de lecturas, establecimiento de relaciones filogenéticas y la identificación funcional y estructural de los genes. A continuación, se describen las etapas considerando su enfoque, herramienta de análisis, descripción e interpretación de resultados.

5.1. Evaluación inicial de la calidad

El proceso de secuenciación trae como resultado la generación de segmentos cortos de ADN conocidos como lecturas o *reads*, que varían de tamaño de acuerdo a la plataforma de secuenciación. Para poder hacer uso de dichas lecturas, resulta crucial la evaluación de su calidad para verificar la confiabilidad y eliminar información no útil que pudiera haber quedado del mismo proceso de secuenciación. En esta sección

se describirá cómo utilizar el software **fastqc** para evaluar la calidad [6]. **fastqc** realiza comprobaciones de control de calidad en los datos de secuencia crudos (sin procesar). **fastqc** utiliza archivos con formato **FASTQ** que albergan la representación de la secuencia nucleotídica asociada a un valor en código ASCII para denotar la calidad (ver Sección 4.2). **fastqc** realiza una serie de análisis llamados módulos, que producen una serie de gráficos sobre los aspectos básicos de las lecturas; por ejemplo, calidad de la secuencia por base, contenido de guanina y citosina, niveles de secuencias duplicadas, distribución del tamaño de secuencia, entre otros. En la Figura 5.1, se aprecia la interfaz gráfica que es visualizada mediante un navegador web. En la sección izquierda, aparece el listado de aspectos evaluados; mientras que, en la parte derecha, la gráfica asociada para cada aspecto. Adicionalmente, cada gráfica viene acompañada de tres caracteres: uno en verde, naranja y rojo. Estos, indican si las lecturas analizadas cumplen con el parámetro de calidad asociado a cada criterio.

Para ejecutar **fastqc**, se utiliza la sintaxis:

```
$ fastqc [ruta/de/fastq_1 ruta/de/fastq_2] [--outdir=/otro/dir/] [--extract]
    [--help]
```

Donde:

fastqc invoca al programa **fastqc** para la realizar la evaluación de la calidad.

`ruta/de/fastq_1` es la ruta del primer archivo con formato `fastq` que contiene las lecturas.

`ruta/de/fastq_2` es la ruta del segundo archivo con formato `fastq` que contiene las lecturas.

`--outdir=/otro/dir/` indica el directorio destino donde se guardarán los archivos resultantes.

`--extract` se especifica para descomprimir de manera automática el archivo `zip` generado como salida que contiene los gráficos e información general asociadas a la

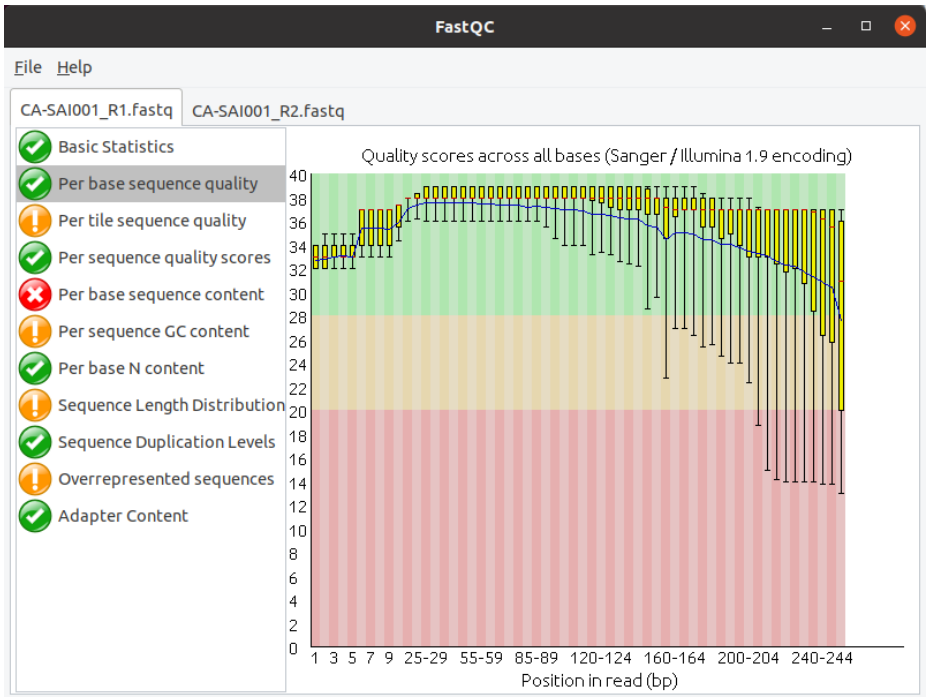

Figura 5.1: `fastqc`. Módulo *per base sequence quality*. Se muestra la interfaz gráfica con los módulos de calidad evaluados.

evaluación de la calidad.

`--help` muestra la ayuda y muestra las opciones disponibles.

Ejemplo 5.1: Ejecución de fastqc.

```
1 fastqc CA-SAI001_R1.fastq.gz CA-SAI001_R2.fastq
```

En Ejemplo 5.1, para ejecutar `fastqc` únicamente se especificaron los dos archivos `.fastq` requeridos; se omitieron los parámetros `outdir` y `extract`.

Dentro de los módulos más utilizados en la evaluación de la calidad inicial de las lecturas, se encuentra *per base sequence quality*, el cual se muestra en la Figura 5.1. La Figura 5.1 se compone de la posición de la lectura en pares de bases (eje X) y la calidad asociada a cada lectura (eje Y). Adicionalmente, en la Figura 5.1 se utilizan franjas de colores (verde, amarillo y rojo) para señalar la calidad. Por ejemplo, la

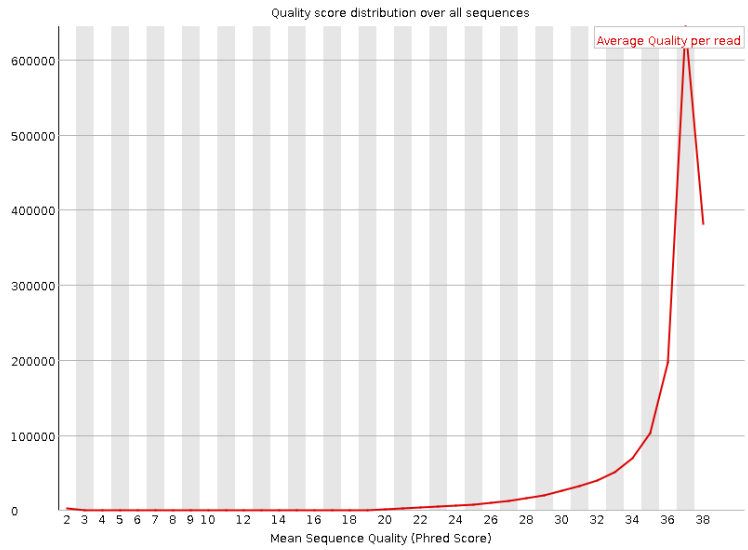

Figura 5.2: `fastqc`. Total de lecturas frente a la puntuación de calidad promedio de cada lectura.

zona verde (calidad deseable) indica una calidad de 28 o superior en escala *Phred*, la amarilla (calidad media) una calidad de 20 a 27, mientras que la roja (mala calidad) de 0 a 19. Se espera que calidad de las lecturas de secuenciación se sitúen dentro del rango verde, sin embargo, en la primera revisión generalmente hay secuencias que se ubican por debajo de este nivel, por lo que el siguiente paso contempla la remoción de lecturas de baja calidad.

Posteriormente, `fastqc` presenta un gráfico del total de lecturas frente a la puntuación de calidad promedio en la duración de cada lectura. El valor deseable de distribución debe contemplar que la calidad de la lectura promedio sea ajustada en el rango superior de la gráfica (Figura 5.2).

El tercer módulo (Figura 5.3), el «análisis de calidad» nos entrega el contenido de secuencias refiriéndose al porcentaje de bases solicitadas para cada uno de los cuatro nucleótidos (`A, T, G` y `C`) en cada posición de las lecturas del archivo.

Para obtener una buena secuenciación, la proporción de las cuatro bases debe permanecer constante a lo largo de la lectura considerando `%A=%T` y `%G=C%`, por complementariedad de bases. En caso contrario, si la composición no es uniforme el pará-

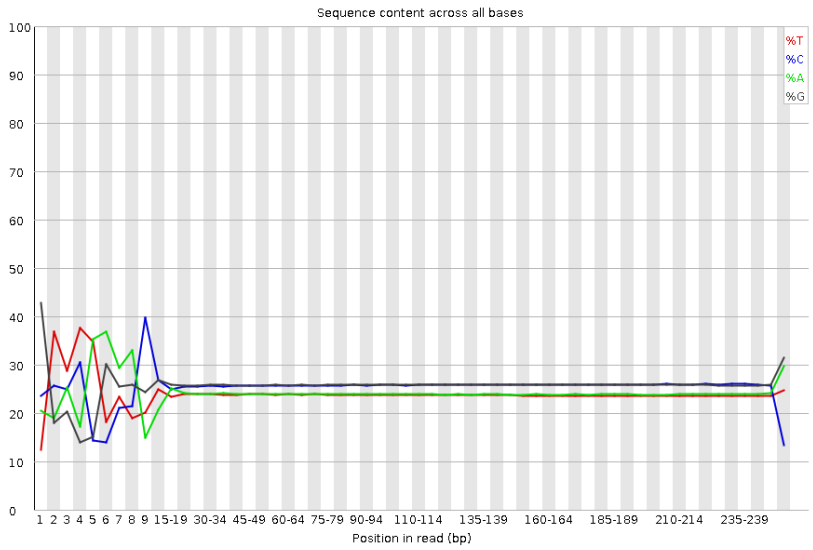

Figura 5.3: `fastqc`. Contenido de la secuencias en todas las bases nucleótidicas (`A`, `T`, `G` y `C`).

metro se calificará como no aprobado. La proporción de las cuatro bases se representa en la Figura 5.3 con colores diferentes para cada base.

El módulo «niveles de duplicación en las secuencias» también es relevante. El módulo «niveles de duplicación en las secuencias» representa el porcentaje de lecturas de una secuencia dada en el archivo que están presentes determinado un número de veces en el archivo (línea azul en la Figura 5.4). Generalmente, se espera que el 100 % de las lecturas para un genoma sean únicas (que aparezcan solo una vez en los datos de secuencia); lo que indica diversidad y garantiza que no se secuenció en exceso. De lo contrario, la línea azul tendrá picos a lo largo de su aparición por la gráfica y se tendrán que eliminar estas lecturas en el proceso de filtrado.

Otro aspecto importante al considerar la evaluación inicial de la calidad consiste en la «identificación de adaptadores de secuenciación». Los adaptadores de secuenciación son secuencias nucleotídicas cortas que se unen a la cadena de ADN para favorecer el proceso de secuenciación. Idealmente, los datos no deben presentar algún tipo de secuencia adaptadora. Sin embargo, en lecturas muy largas, es posible que aparezcan algunas inserciones secuencias adaptadoras en las lecturas. La Figura 5.5 muestra en

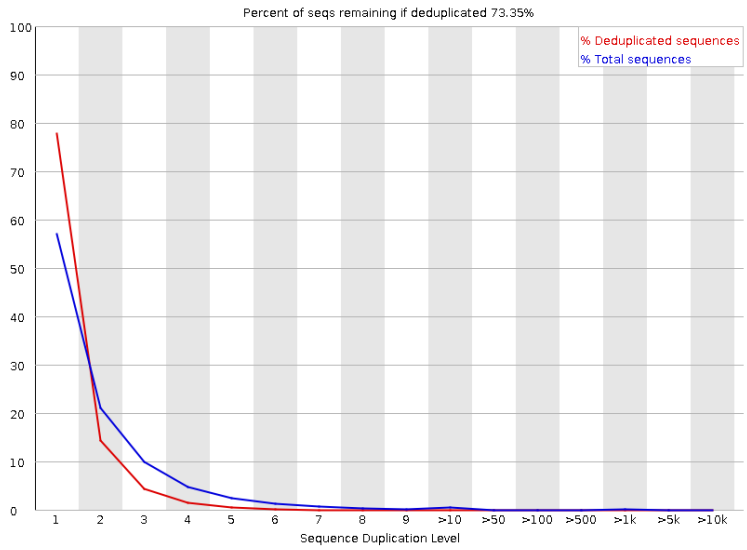

Figura 5.4: **fastqc**. Niveles de duplicación en las secuencias.

línea sólida azul la identificación de adaptadores correspondientes a la plataforma Illumina para las lecturas. El resultado adecuado debe ser una línea azul que sea asintótica para el porcentaje del 0 %.

La evaluación inicial de la calidad permite obtener una valoración de las secuencias obtenidas. Esta información resulta útil para determinar si es posible el uso de las lecturas para los procesos de análisis subsecuentes. En general, una buena calidad estará representada en su mayoría mediante símbolos verdes mostrados en el panel izquierdo de la interfaz del programa **fastqc** (Figura 5.1) sobre todo, en los módulos que hemos incluido en esta sección.

5.2. Ensamblado de lecturas

Los archivos **FASTQ** generados por el proceso de secuenciación tienen un tamaño grande debido a la asociación existente entre cada nucleótido y su calidad (Ver sección Sección 4.2). Por otra parte, estos archivos almancenados en formato**FASTQ** representan segmentos de lecturas cortos que son poco utilizables para genómica comparativa. Para superar estas limitaciones, se realiza un proceso de ensamblado de lecturas, cuyo

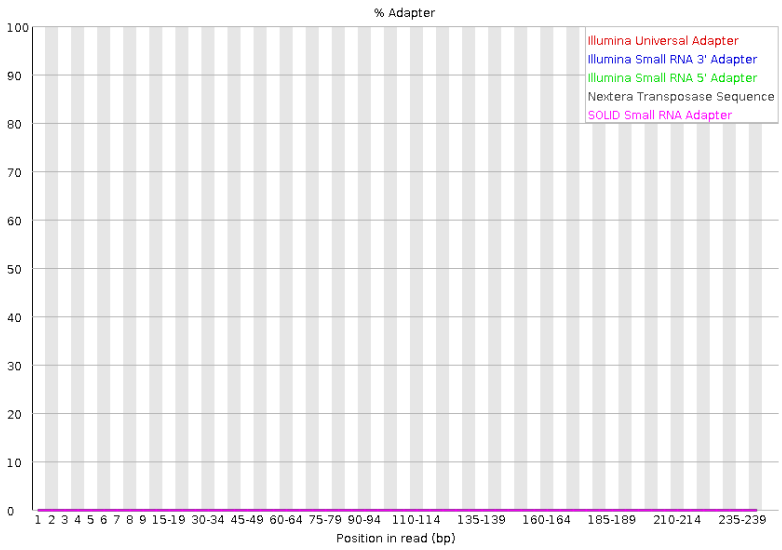

Figura 5.5: `fastqc`. Presencia de secuencias adaptadoras.

objetivo es reconstruir el genoma bacteriano. El proceso de ensamblado de lecturas se puede lograr mediante dos aproximaciones:

- Ensamblado *de novo*: no se tiene una secuencia de referencia que sirva como molde para la reconstrucción. Se basa en el proceso de solapar lecturas que coincidan en cierto número de nucleótidos (*kmers*) en la parte inicial y final de dos lecturas. De esta manera se va creando una secuencia contigua de mayor longitud.
- Ensamblado con referencia: se cuenta con un genoma completo que sirve como molde para alinear cada una de las lecturas en una posición determinada de acuerdo su similitud con la referencia.

Existen principalmente tres (3) niveles de organización de acuerdo a la complejidad de las lecturas (Figura 5.6). El primero de ellos es el *contig*, expresado como un conjunto reducido de secuencias (2 a 10) que se solapan formando una secuencia de mayor tamaño. El segundo nivel de organización se denomina *scaffold* que resulta de la combinación de varios *contigs* unidos por espacios vacíos (*gaps*) de los cuales no

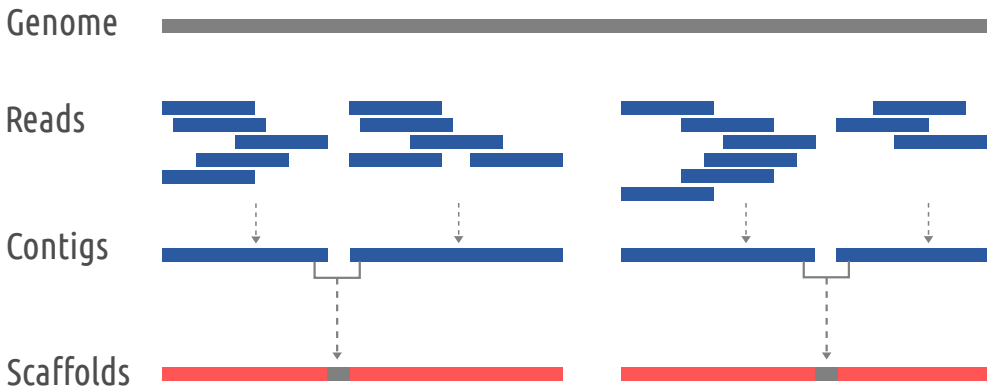

Figura 5.6: Ensamblado de lecturas. Niveles de organización de acuerdo a la complejidad de las lecturas.

se tiene información. Por último, en el tercer nivel de organización, se encuentra el *genoma* que agrupa los *scaffolds* generados.

Para encontrar las terminaciones de coincidencia entre las lecturas se han utilizado tres algoritmos informáticos:

1. Algoritmos avaros (*Greeddy algorithms*). Siendo el más sencillo basado en conectar las lecturas que mejor se solapan de manera iterativa mientras no vayan en contra del ensamble final. Se centra en utilizar información local para generar las coincidencias, razón por la cual está en desuso, al no resolver de manera eficiente las regiones largas repetidas [75].

2. *Overlap-Layout-Consensus* (OLC). Construye un gráfico que representa nodos unidos por líneas que simbolizan el orden de las lecturas. Esta aproximación considera la información global de alineamiento, por lo que eligen caminos donde el solapamiento no interfiera o se repita en dos o más secuencias. Evitando de esta manera tener secuencias repetidas dentro del genoma [63].

3. Grafos de *de Bruijn*. Es un método ampliamente utilizado y muy similar a OLC donde se construyen grafos basados en nodos y conectores. Su diferencia radica en el solapamiento de un número determinado de nucleótidos en cada lectura.

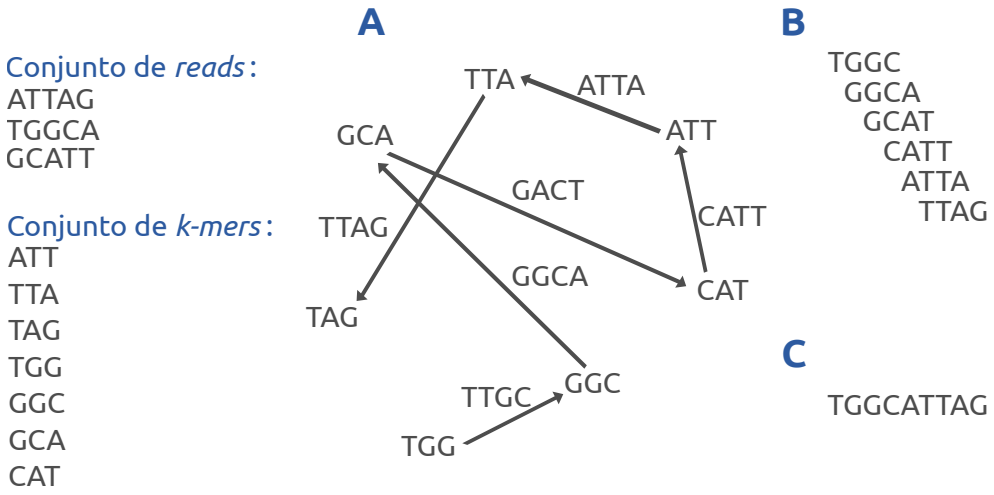

Figura 5.7: Método *de Bruijn* para la construcción de gráficos. Imagen basada en Wojcieszek et. al. [97]

El algoritmo de *de Bruijn* también utiliza información global de las lecturas para resolver repeticiones y eliminar patrones no consistentes, con la ventaja de incluir métodos para corregir los errores y mejorar la calidad de los ensambles (ver Figura 5.7) [75].

`A5-miseq` es un software utilizado para el ensamblado de lecturas de secuenciación, especialmente de aquellas generadas por la plataforma Illumina. `A5-miseq` implementa una serie de etapas automáticas que van desde la evaluación de calidad hasta, filtrado y remoción de adaptadores [17]. `A5-miseq` obtiene ensambles con buenos criterios de calidad, para el ensamblado de *contigs* utiliza el algoritmo IDBA-UD que implementa grafos de *de Bruijn*. `A5-miseq` toma como base dos criterios relevantes para la generación de ensambles de buen calidad: el tamaño de genoma y número de fragmentos.

La sintaxis para ejecutar `A5-miseq`:

```
a5_pipeline.pl [--end=5] read_1.fastq read_2.fastq nombre_output
```

Donde:

`a5_pipeline`: se usa para ejecutar el algoritmo.

`--end=5`: El 5 puede ser un valor entero del 1 al 5 que indica la última actividad que se realizará en el análisis.

`read_1.fastq`: secuencia correspondiente a la lectura 1 del proceso de secuenciación.

`read_2.fastq`: secuencia correspondiente a la lectura 2 del proceso de secuenciación.

`nombre_output`: es el nombre que se asignará a los archivos resultantes.

Ejemplo 5.2: Ejecución de A5-miseq.

```
1 $ a5_pipeline.pl read_1.fastq read_2.fastq nombre_output
```

El Ejemplo 5.2 muestra cómo ejecutar `A5-miseq`. En el Ejemplo 5.2 se especificaron únicamente los dos archivos de las secuencias y nombre del archivo de salida. Adicionalmente, se puede ejecutar `A5-miseq` con el parámetro `--end=5` para indicar que se realice el *pipeline* completo desde evaluación de calidad hasta reordenamiento de lecturas para obtener un mejor ensamble.

Una vez finalizada la ejecución `A5-miseq` genera archivos de salida con la información asociada al resultado de análisis, la Tabla 5.1 muestra el listado de los archivos de salida obtenidos al ejecutar `A5-miseq`. Los archivos de salida tienen como nombre base el asignado en el parámetro `nombre_output`. Si se omite el parámetro `nombre_output`, le asigna el nombre base `assembly.out`.

Tabla 5.1: Listado de los archivos de salida obtenidos al ejecutar `A5-miseq`.

Archivo	Contenido
`assembly.out.ec.fastq.gz`	Lecturas (*reads*) corregidas de errores
`assembly.out.contigs.fasta`	Contigs
`assembly.out.crude.scaffolds.fasta`	*Scaffolds* crudos: No se han comprobado los errores de ensamblado
`assembly.out.broken.scaffolds.fasta`	*Scaffolds* rotos: Se comprueba si hay ensamblados erróneos, pero no se generan nuevamente los *scaffolds*
`assembly.out.final.scaffolds.fasta`	*Scaffolds* finales: Comprobados en busca de ensamblados erróneos y *re-scaffolds*

Archivo	Contenido
`assembly.out.final.scaffolds.fastq`	*Scaffolds* finales en formato `fastq` con calidades asociadas (en puntuación de calidad Phred)
`assembly.out.final.scaffolds.qvl`	Valores de calidad de los *scaffolds* finales en formato QVL para su envío al NCBI
`assembly.out.final.scaffolds.agp`	Archivo AGP que describe el proceso de generación *scaffolds* para su envío al NCBI
`assembly.out.assembly_stats.csv`	Archivo separado por comas con las estadísticas de resumen del ensamblado

Para evaluar el ensamble resultante, se deben tomar varias consideraciones. La primera consideración es el *tamaño del genoma generado*. El tamaño del genoma generado, es un parámetro importante, se debe conocer antes de realizar la ejecución de `A5-miseq`. Generalmente, esta información para el organismo de interés se puede encontrar en NCBI o en su defecto, un anuncio de genoma (*genome announcement*).

Otro factor importante es el *número de fragmentos generados*. Las bacterias de manera general poseen un cromosoma, por lo que se espera un ensamble que se acerque a ese número. Sin embargo, es raro que se generen ensambles de un cromosoma debido a la fragmentación del genoma durante el proceso de secuenciación. Por lo que se considera un ensamble adecuado cuando el número de fragmentos es bajo.

En el Ejemplo 5.3 se muestra el contenido del archivo `assembly _stats.csv` que contiene la información estadística del ensamble generado por `A5-miseq`. Para este caso en particular, el genoma tuvo un tamaño de `503058 Mpb`, información que coincide con lo reportado para `S. enterica`, cumpliendo con este parámetro. En cuanto al número de fragmentos, se obtuvieron `740`, siendo un número bajo cumpliendo con el rango de aceptación. En este ejemplo en particular, al revisar los resultados, se puede concluir que el ensamble evaluado puede ser utilizado para los análisis siguientes.

Ejemplo 5.3: Contenido del archivo assembly _stats.csv generado por A5-miseq.

```
$ column -s $'\t' -t CA-SAI001.assembly_stats.csv
File Name  Contigs  Scaffolds  Genome Size  Longest Scaffold  N50  Raw reads
CA-SAI001  740      740        503058       4159              652  3281192
EC Reads   % reads passing EC  Raw nt     EC nt     % nt passing EC  Raw cov
207443    6.32                601852954  33853329  5.62             1196.39
EC cov  Median cov  10th percentile cov  bases >= Q40  % GC  assembler version
67.30   10          5                    450879        51.3  A5-miseq 20160825
```

5.3. Anotación

El proceso por el cual se identifica y se le asigna función a cada gen se denomina anotación estructural y funcional, respectivamente. La anotación permite asignar un significado biológico a un conjunto de nucleótidos que codifican para alguna proteína en particular. Para realizar la anotación, utilizaremos el software `Prokka` por ser confiable y uno de los más utilizados [82]. `Prokka` además de identificar los genes, también logra predecir los rRNA (ribosomales) y tRNA (de transferencia). En la Tabla 5.2 se muestra una serie de dependencias que contiene `Prokka` cumpliendo una función adicional a la identificación de genes.

Para llevar a cabo el análisis se requiere de un genoma en formato **FASTA** (ver Sección 4.1), siendo el único archivo de entrada necesario. Los genes son anotados en dos etapas. En la primera etapa, `Prodigal` identifica la localización de los posibles genes sin asignarle nombre a la proteína. Esto se realiza mediante la comparación con una base de datos de proteínas conocidas, para posteriormente anotar o rotular la coincidencia más significativa. De forma predeterminada `Prokka` utiliza un umbral `e-value` de 10^{-6} (describe el número de aciertos que uno puede ver que ocurran por azar en una determinada base de datos).

Tabla 5.2: Herramientas de predicción de características utilizadas por Prokka [81].

Herramienta (referencia)	Características esperadas
Prodigal [37]	Secuencia codificante (CDS)
RNAmmer [49]	Genes de ARN ribosómico (rRNA)
Aragorn [51]	Genes de ARN de transferencia
SignalP [73]	Péptidos líderes de señal
Infernal [47]	ARN no codificante

La sintaxis de Prokka es la siguiente:

```
$ prokka [--genus] [--kingdom] [--quiet] [--usegenus] --outdir anotacion
      --prefix genoma1 contigs.fa
```

Las *opciones* son:

`--outdir`: nombre del directorio donde se guardarán los archivos de salida.

`--prefix`: prefijo que se utilizará nombrar a los archivos de salida.

`contigs.fa`: archivo en formato FASTA que contiene el genoma a anotar.

`--genus`: utilizada para indicar el género de la especie bacteriana a analizar.

`--kingdom`: utilizada para indicar el reino del organismo a analizar.

`--quiet`: para que no aparezca en consola la ejecución del programa.

`--usegenus`: para utilizar una base de datos específica para la identificación de proteínas y anotación.

Para anotar un genoma de *Salmonella enterica* especificando el reino y especies, se procedería a ejecutar Prokka tal como lo muestra el Ejemplo 5.4.

Ejemplo 5.4: Ejecución de Prokka especificando reino y especies.

```
1 $ prokka --kingdom bacteria --genus salmonella --prefix genoma1 contigs.fa
```

En el Ejemplo 5.4, al ejecutar la línea 1, Prokka busca la base de datos específica para *Salmonella* y empezar a buscar similitudes en las secuencias para identificar los

genes implicados. La extensión de los archivos de salida generados por esta herramienta de análisis así como una breve descripción son mostrados en la Tabla 5.3.

Tabla 5.3: Prokka. Descripción de los archivos de salida de Prokka [81].

Sufijo	Descripción del contenido
`.fna`	Archivo fasta de contigs originales de entrada (nucleótidos)
`.faa`	Archivo fasta de los genes codificantes traducidos (proteína)
`.ffn`	Archivo fasta de todas las características genómicas (nucleótidos)
`.fsa`	Secuencias de contig para enviar (nucleótidos)
`.tbl`	Tabla de características para someter a envío
`.sqn`	Archivo editable de secuencias para someter a envío
`.gbk`	Archivo Genbank que contiene secuencias y anotaciones
`.gff`	Archivo GFF v3 que contiene secuencias y anotaciones
`.log`	Archivo de bitáctora de los resultados del procesamiento de Prokka
`.txt`	Estadísticas de resumen de anotaciones

El archivo de salida con extensión `.gff` permite visualizar el producto de proteína (gen) para cada secuencia. El contenido del archivo está organizado en columnas. En la primera columna se encuentra el nombre o código de la cepa, seguido por el programa utilizado para la identificación de las secuencias codificantes, la posición de inicio y final de dicha secuencia, cadena en la que se encuentra el gen (*forward* o *reverse*) y por último el nombre del gen asociado a su función. En el Ejemplo 5.5 se muestra un listado de las primeras líneas del archivo `.gff` obtenido.

Para una mejor visualización, se recomienda utilizar los softwares Artemis [12] o Integrative Genomics Viewer (IGV) [78] [91].

5.4. Relaciones filogenéticas

En esta sección se definirá el concepto de filogenia, trataremos los métodos de contrucción de árboles filogenéticos y finalmente, se generará un árbol filogenético.

Ejemplo 5.5: Primeras líneas de un archivo .gff.

```
1 Nombre        Software     id  Inicio Final        Nnombre del gen asociado
2 NC_009792.1  Prodigal:2.6 CDS 102546 104192 . + 0 ID=EJNPAANP_00093;eL_number=
    2.2.1.6;Name=i1vG;gene=i1vG;inference=ab initio prediction:Prodigal:2.6,
    similar t
3 NC_009792.1  Prodigal:2.6 CDS 104189 104452 . + 0 ID=EJNPAANP_00094;eL_number=
    2.2.1.6;Name=i1vM;gene=i1vM;inference=ab initio prediction:Prodigal:2.6,
    similar t
4 NC_009792.1  Prodigal:2.6 CDS 104470 105399 . + 0 ID=EJNPAANP_00095;eL_number=
    2.6.1.42;Name=ilvE;gene=ilvE;inference=ab initio prediction:Prodigal:2.6,
    similar
5 NC_009792.1  Prodigal:2.6 CDS 105466 107316 . + 0 ID=EJNPAANP_00096;eL_number=
    4.2.1.9;Name=i1v0;gene=ilvD;inference=ab initio prediction:Prodigal:2.6,
    similar t
6 NC_009792.1  Prodigal:2.6 CDS 107319 108863 . + 0 ID=EJNPAANP_00097;eL_number=
    4.3.1.19;Name=i1vA;gene=ilvA;inference=ab initio prediction:Prodigal:2.6,
    similar
7 NC_009792.1  Prodigal:2.6 CDS 108956 109846 . - 0 ID=EJNPAANP_00098;Name=oxyR_1;
    gene=oxyR_1;inference=ab initio prediction:Prodigal:2.6,similar to AA
    sequence:
```

5.4.1. Filogenia

Filogenómica derivó en «Filogenia», proviene de tres palabras «phylo» que significa grupo o tribu en griego, la raíz «gen» que significa producir y el sufijo «ia» (cualidad de). En la actualidad «Filogenia» se expresa como un gráfico para representar grupos de organismos. Estos gráficos permiten identificar similitudes entre organismos y entre grupos de organismos.

La filogenia se apoya en la biología molecular, bioquímica, sistemática, bioinformática y genética de poblaciones. El objetivo de la filogenia es anotar, sistematizar y archivar grandes cantidades de datos biológicos obtenidos a partir de secuenciación de nueva generación (NGS). Los datos obtenidos requieren aplicaciones bioinformáticas para encontrar respuestas biológicas y administrar los datos biológicos.

Los productos que genera bioinformática en colaboración con la filogenia son las matrices filogenéticas, la construcción de árboles evolutivos, expresión genica, ensamblado de genomas, y predecir el pliegue filogenéticos de proteínas.

5.4.2. Métodos de construcción

Los árboles filogenéticos permiten identificar similitudes entre organismos y entre grupos de organismos. Un arbol filogenético es la representación gráfica de las relaciones evolutivas entre un conjunto de secuencias de ADN, especies u otro nivel de taxonomía definido. Cuando el investigador y su equipo de colaboradores deciden iniciar un trabajo de filogenia, es relevante decir la Unidad Taxómica Operativa. La Unidad Taxonómica Operativa (OTU, por sus siglas en inglés) se refiere a una unidad de clasificación que le permite al investigador individualizar a los objetos de su estudio, ya sea una especie u otro taxón de cualquier categoría.

Los métodos de construcción filogenética se clasifican de acuerdo a:

- tipo de dato empleado (caracteres discretos vs distancias)
- método que utilizan para encontrar la topología óptima bajo el criterio seleccionado.
    - método algorítmico.
    - método de búsqueda basado en un criterio de optimización.

5.4.3. Significado biológico

Un árbol filogenético tiene como objetivo determinar las relaciones de dos o más OTUs (usualmente individuos o grupos de organismos). Un árbol filogenético se puede considerar una reconstrucción hipotética de la historia evolutiva.

La Figura 5.8 muestra diferentes representaciones de árboles filogenéticos; todas igualmente sirven para comparar individuos (de manera general OTUs). En la Figura 5.8 se muestran 4 diferentes árboles filogenéticos. La figura contiene cuatro (4)

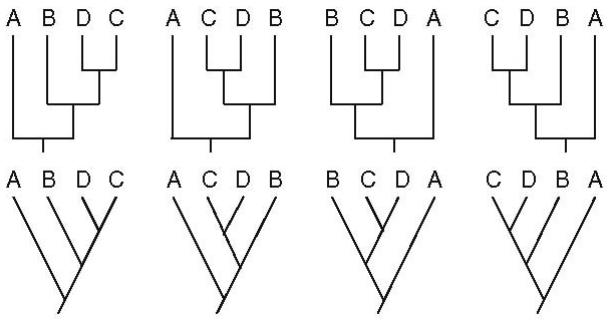

Figura 5.8: Árbol filogenético. Imagen tomada de Taxonomía y filogenia: Figura 3 de Robert Bear et al., CC BY 4.0

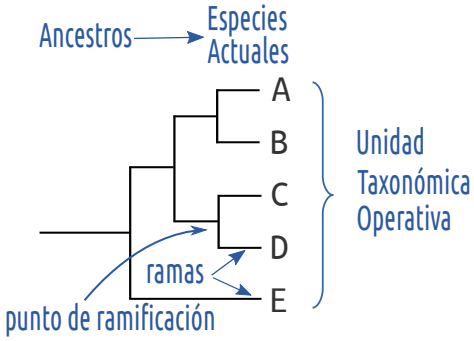

Figura 5.9: Árbol filogenético. Significado biológico.

columnas, cada columna se muestra dos diferentes representaciones de un árbol filogenético. Los dos árboles que aparecen en cada columna son equivalentes. En los árboles mostrados en esta figura, se establecen y permanecen las mismas relaciones entre los organismos `A`, `B`, `C`, y `D`.

La Figura 5.9 muestra el significado de cada parte del árbol filogenético. Con flechas está indicada cada parte del árbol filogenético. El elemento más a la izquierda es el ancestro, el elemento más a la derecha son los OTUs (en algunos análisis el OTU de nivel más bajo es la especie). Los agrupamientos intermedios que aparecen en la imagen se refieren a agrupamientos que expresan que similitudes entre grupos de elementos.

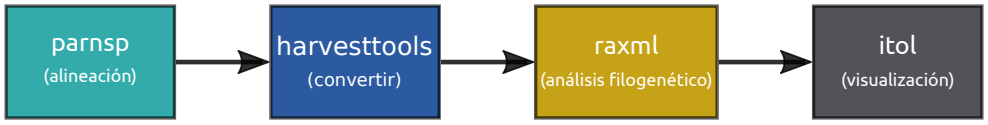

Figura 5.10: Flujo de trabajo propuesto para generar un árbol filogenético.

5.4.4. Ejecución

El conjunto de datos que se utilizará en esta sección está disponible en: https://bit.ly/genomas-tesis-bioinformatica-RPA

La Figura 5.10 muestra el flujo de trabajo para generar un árbol filogenético de genomas bacterianos propuesto en esta sección. El flujo de trabajo contempla el uso de Harvest Suite [69], un software que incluye a las herramientas `parsnp`, `gingr` y `HarvestTool`. También se incluye la visualización del árbol filogenético mediante la herramienta `itol`.

Los pasos de ejecución que contempla este flujo de trabajo se describen a continuación:

1. El primer paso ejecuta la herramienta `Parsnp`. `Parsnp` permite la multialineación rápida del genoma central. Parsnp tiene la capacidad de alinear cientos de miles de genomas bacterianos en poco tiempo minutos u horas en dependencia del volumen del conjunto de datos. Parsnp recibe como entrada borradores de ensambles o genomas completos y genera como salida llamadas variantes SNP (*variant calls* SNP), filogenia del genoma central y alineamientos múltiples.

2. El segundo paso consiste en la ejecución de `harvestools`. `harvestools` es un herramienta para crear y convertir con archivos en formato `.ggr`. El formato `ggres` eficiente, permite almacenar alineaciones múltiples comprimidas por referencia, árboles filogenéticos, variantes filtradas y anotaciones. En el flujo de trabajo, se utiliza para convertir del formato `.ggr` a formato `.fasta`.

3. El tercer paso consiste en ejecutar `raxml`. `raxml` [86] realiza el análisis filogenético de grandes conjuntos de datos mediante la estimación de máxima verosimilitud. En el flujo de trabajo permite generar el árbol filogenético. `raxml` brinda como salida archivos con extension `.out`. Aunque `raxml` genera como resultado varios versiones de árboles filogenéticos, a uno de ellos lo evalúa como el mejor y así lo indica en el nombre el archivo resultante: `RAxML_bestTree.$MYOUT.out1`.

4. El cuarto paso ejecuta `itol`. `itol` [52] es una herramienta web que permite la visualización de árboles filogenéticos. Su interfaz permite la carga de archivos generados por `raxml` con los datos de árboles filogenéticos. Una vez que muestra el árbol filogenético, itol permite personalizar la visualización. En este flujo de trabajo se carga el archivo `RAxML_bestTree.$MYOUT.out1` a `itol`.

El Ejemplo 5.6 contiene las instrucciones que requieren ser ejecutadas para generar el árbol filogenético a partir de los genomas proporcionados, es decir, muestra las instrucciones requeridas para ejecutar el flujo de trabajo completo.

En el Paso 1, se descomprime el archivo que contiene los genomas. El Paso 2 permite mostrar el listado de los archivos que serán analizados. En el Paso 3, `parsnp` recibe el directorio que contiene los archivos `.fasta`. `parsnp` da como salida un archivo `.ggr`. El archivo `.ggr` se almacena en un directorio con el nombre `P_AAAA_MM_DD_HHMMSS` conformado por la fecha y hora actual. En el Paso 4, `harvesttools` recibe como entrada el archivo `.ggr` generado en el paso anterior y lo convierte a `.fasta`. En el Paso 5, `raxml` genera el árbol filogenético. `raxml` recibe el archivo `.fasta` para obtener como salida tres archivos; uno de ellos es el que `raxml` considera el mejor. Finalmente, el Paso 6 abre la url de la herramienta `itol`. El objetivo es usar la interfaz web de itol para subir el archivo `RAxML_bestTree.$MYOUT.out1` para generar la visualización del árbol filogenético.

La Figura 5.11 muestra el árbol filogenético obtenido. La visualización fue obtenida utilizando la herramienta web `itol`. El árbol filogenético obtenido puede interpretarse

Ejemplo 5.6: Flujo de trabajo para generar un árbol filogenético.

```
#Paso 1.
tar -xvzf genomas.tar.gz
#Paso 2.
$ ls genomas/FASTA
CA-SAI080010.fasta  CA-SAI08003.fasta  CA-SAI08006.fasta  CA-SAI08009.fasta
CA-SAI08001.fasta   CA-SAI08004.fasta  CA-SAI08007.fasta
CA-SAI08002.fasta   CA-SAI08005.fasta  CA-SAI08008.fasta
#Paso 3.
$ parsnp -r ! -d genomas/FASTA -c
#Paso 4.
$ harvesttools -i P_2022_12_08_105510866504/parsnp.ggr -M multifasta.fasta
#Paso 5.
$ MYINPUT=P_2022_12_08_105510866504/multifasta.fasta
$ MYOUT=resultado_2022_12_08
$ raxmlHPC  -m GTRGAMMA -f a -p 01234 -x 01234  -# 100 -s $MYINPUT -n $MYOUT.
    out1
$ raxmlHPC -m GTRGAMMA -f b -t RAxML_bestTree.$MYOUT.out1 -z RAxML_bootstrap.$
    MYOUT.out1 -n $MYOUT.out2
#Paso 6.
$ xdg-open https://itol.embl.de/upload.cgi
```

considerando que debe ser leído de izquierda a derecha, donde a la izquierda está el ancestro y a la derecha las especies actuales. Por ejemplo, en el árbol filogenético se puede observar que las cepas CA-SAI08003 y CA-SAI08003 forman un grupo. Las dos cepas mencionadas comparten ancestro con la cepa CA-SAI08005. También se observa que las cepas CA-SAI08009 y la CA-SAI08010 forman un grupo. Todas las cinco (5) cepas mencionadas a su vez comparten ancestro en común y forman un grupo.

En el flujo de trabajo propuesto en este trabajo, gracias a la utilización de las herramientas bioinformáticas y a la asignación de un significado biológico a los genomas, se logró identificar a los genes que componen el género *Salmonella*. Los genes fueron organizados en diferentes categorías dependiendo de su funcionalidad y adicionalmente se establecieron las relaciones de parentesco. La información obtenida es de

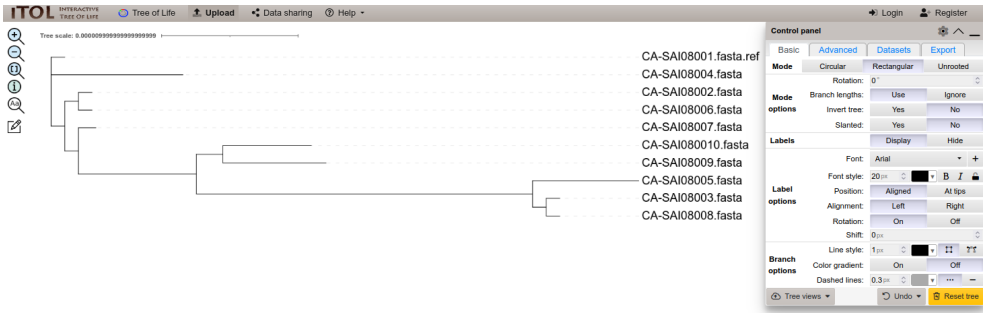

Figura 5.11: Árbol filogenético. Visualización mediante la herramienta web `itol` del árbol filogenético generado.

suma utilidad debido a que permite encontrar patrones o similitudes epidemiológicas, realizar procesos de trazabilidad y conocer más acerca de la biología básica de este patógeno; lo cual contribuye a establecer medidas que permitan evitar la diseminación del patógeno.

Capítulo 6
Conclusiones

Este trabajo propone un flujo de trabajo que incluye los pasos y herramientas de software requeridos en genómica comparativa para realizar el análisis de datos biológicos y la generación de productos como un árbol filogenético. El flujo de trabajo tiene potencial para ser escalado en futuros análisis bioinformáticos que requieran de una gran cantidad de datos. El sistema operativo GNU/Linux y las herramientas de software utilizadas tienen capacidad para trabajar con volúmenes grandes datos si operan en infraestructura adecuada.

Así también, el impacto de este trabajo radica en su aportación para reducir la brecha de conocimiento, comunicación y entendimiento que se presenta en los equipos de trabajo (alumnos, profesores e investigadores) que incursionan en bioinformática. El trabajo presentado brinda información actualizada con un enfoque desde dos áreas del conocimiento: la biología y la computación. Por una parte, incluye los conocimientos de computación requeridos para un biólogo que se enfrenta a las necesidades de utilizar GNU/Linux y ejecutar las herramientas de software que le permiten analizar datos biológicos. Por otra parte, también incluye los conocimientos del área biológica que un colaborador del área de la computación requiere para entender los conceptos biológicos y los tipos de archivo que se utilizan en la bioinformática.

Anexo I.
Uso de GNU/Linux

En este capítulo se incluye información complementaria sobre la utilización del sistema operativo GNU/Linux con la finalidad de facilitar la comprensión del documento. Se incluyen una guía para abrir la aplicación Terminal y su utilización. También están incluidos comandos útiles para obtener información y manipular archivos y directorios. Se muestran los usos más típicos de cada comando.

AI.1. Cómo abrir la aplicación *Terminal*

Usted tiene estas dos opciones para abrir la aplicación *Terminal* en `Ubuntu 20.04`:

1. Usar atajo de teclado.

   a) Use esta combinación de teclas: Ctrl+Alt+T. El símbolo '+' significa que debe oprimir la tecla y sin soltar, presionar la siguiente tecla.

   b) Se abrirá la aplicación *Terminal*, tal como se muestra en la Figura AI.3

2. Usar el tablero o *Dash*. Para ello:

   a) Presione la *super key*, si estás usando teclado de windows, es la tecla que tiene el logo de windows: Win. Se activará el Dash tal como muestra la Figura AI.1.

   b) Escriba *terminal*. Se realizará la búsqueda en el Dash, tal como se muestra en la Figura AI.2.

   c) Presiona Enter. Se abrirá la aplicación *Terminal*, tal como se muestra en la Figura AI.3.

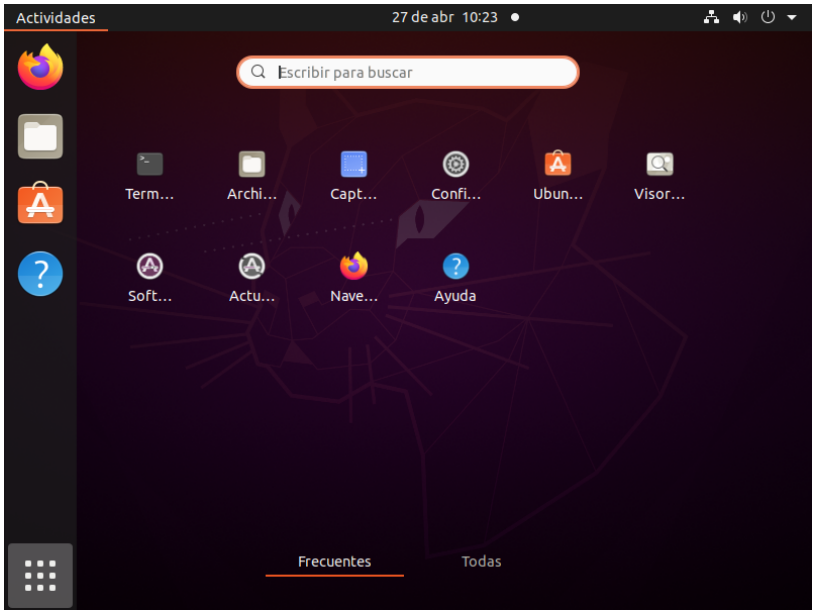

Figura AI.1: Ubuntu. Tablero (o *Dash*) activado.

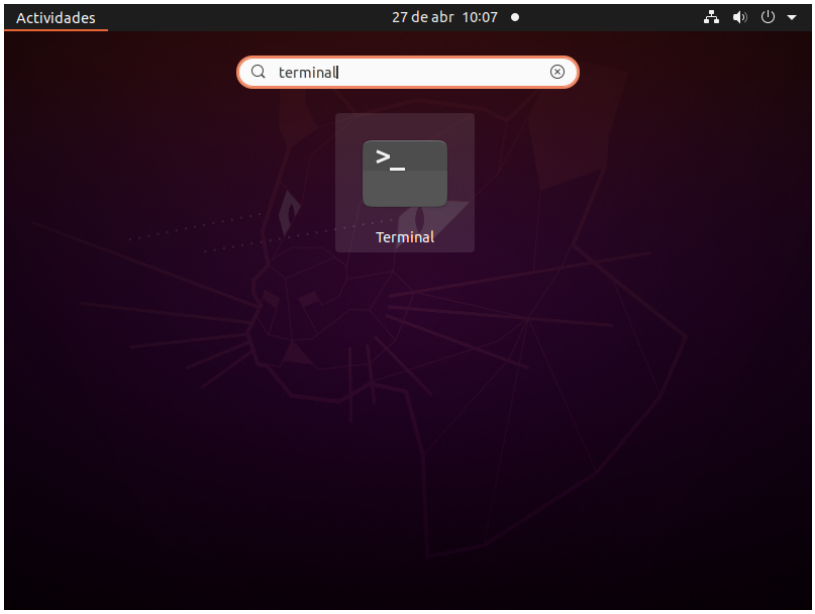

Figura AI.2: Ubuntu. Búsqueda de *terminal* en el Dash.

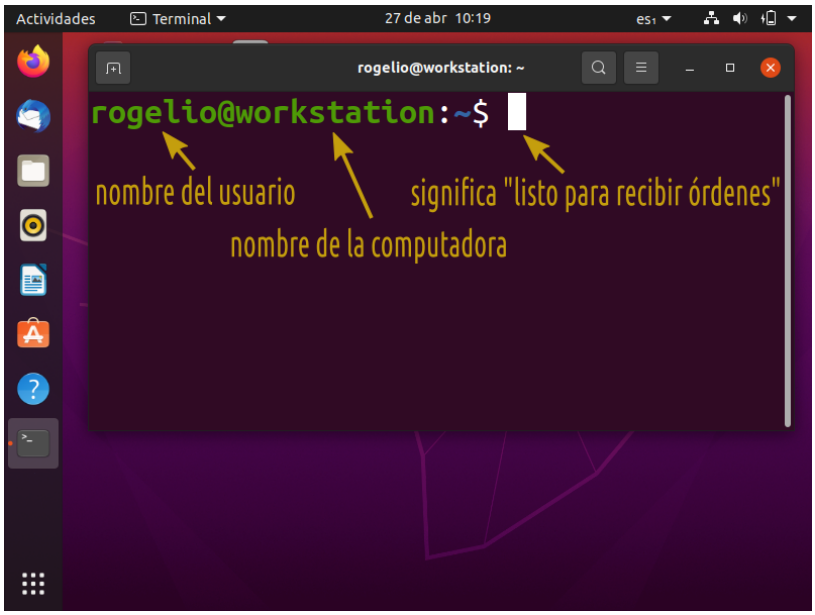

Figura AI.3: Ubuntu. Ventana de la aplicación *Terminal*. Se muestra el *Shell prompt* o indicador del *shell*

AI.1.1. Sobre el *shell prompt*

Cuando se abre la aplicación *Terminal*, muestra un saludo que le indica a usted (al usuario) que el *shell* está listo para recibir instrucciones. Este saludo se llama *prompt* o *shell prompt* (indicador del *shell*).

El *shell prompt* está compuesto por el nombre del usuario seguido de un arroba, el nombre de la computadora, el directorio actual y un signo de pesos. La Figura AI.3 muestra un ejemplo de un *shell prompt*. En este ejemplo el *shell prompt* indica que el usuario `rogelio` está utilizando la computadora `workstation`, que el directorio de trabajo es ~. El caŕacter `@` es un separador entre el nombre del usuario y el nombre de la computadora. Finalmente, el `$` junto con el cursor (el ▉ destellante) indica que el *shell* está listo para recibir una orden o comando del usuario.

AI.2. Linux: Acceso a Línea de Comandos

En GNU/Linux con frecuencia para indicar instrucciones a la computadora se utiliza un intérprete de comandos, también llamado línea de comandos (CLI: *Command Line Interface*). Consiste en una ventana que le permite al usuario escribir órdenes a la computadora, formalmente le llamamos instrucciones. Estas instrucciones se conforman por una acción o comando y/o parámetros asociados. Un intérprete de comandos (*shell*) es un programa que permite ejecutar en el sistema operativo las instrucciones indicadas por el usuario utilizando el teclado. Está diseñado para facilitar la forma en que se ejecutan los programas y recursos en la computadora. El *shell* es una capa intermedia entre el usuario y el sistema operativo.

El *shell* interpreta las instrucciones de usuario, que son escritas directamente por el usuario o ejecutadas desde un archivo llamado *shell script* o *shell program*. El *shell* lee los comandos del script línea por línea, busca y ejecuta cada uno de estos comandos en el sistema, a diferencia de un compilador que convierte un programa en una forma legible para la computadora, un archivo ejecutable que puede ser usado en un *shell script* [28]. El *shell* también cuenta con un lenguaje de programación de alto nivel que puede utilizarse en combinación con otros programas de utilería para crear aplicaciones completas. Además de enviar comandos al sistema operativo, la tarea principal de un *shell* es proporcionar un entorno de usuario, que se puede configurar individualmente mediante archivos de configuración de recursos. El entorno de usuario almacena variables y valores que permiten definir los elementos del entorno.

AI.2.1. Principios básicos del shell

La aplicación *shell* que se utiliza en `Ubuntu 20.04` es la aplicación *Terminal*. Cuando el usuario escribe el nombre de un comando en la *Terminal* y presiona la tecla `Enter` ↵ para indicar su ejecución, el *shell* interpreta el comando, invoca y ejecuta el programa deseado.

En `Ubuntu 20.04` están disponibles los *shells*: **`sh`**, **`bash`**, **`rbash`** y **`dash`**. **`sh`** es un shell liberado en 1979, fue el *shell* predeterminado en la séptima versión de Unix. **`sh`** fue escrito por Stephen Bourne [8]. **`bash`** es un *shell* cuyo nombre es un acrónimo de **Bourne-Again SHell** (se lee *«born again SHell»*) haciendo un juego de palabras sobre el **Bourne shell (sh)** y considera una re-implementación del *Bourne shell* realizada por el proyecto GNU[1]. Además, **`bash`** ofrece mejoras funcionales con respecto a **`sh`** entre las que destacan la edición de línea de comandos, el historial ilimitado de comandos y el control de procesos, entre otros [34]. Actualmente, **`bash`** es de los *shell* más utilizados, es el intérprete de comandos disponible en la mayoría de las distribuciones de GNU/Linux. Por su parte, **`rbash`**, también incluido en `Ubuntu 20.04`, es un *shell* restringido con el objetivo de brindar una capa de seguridad y proporciona un entorno de trabajo más controlado que el *shell* estándar (**`bash`**) [21]. **`dash`** es un *shell* cuyo nombre se forma de las siglas: «*Debian Almquist Shell*», es una moderna implementación de **`sh`**. **`dash`** no es compatible con **`bash`**; significa que algunos comandos escritos en **`dash`** no se podrán ejecutar en **`bash`**. **`dash`** es el *shell* predeterminado en las distribuciones de Linux Debian [19].

El estándar POSIX 1003.2 [1] [38] en sus diferentes versiones es específico para definir las características y funcionalidades de la *shell*; fue impulsado por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, por sus siglas en inglés). **`bash`** y **`dash`** cumplen con este estándar, adicionalmente cuentan con extensiones que les permiten mayor funcionalidad.

En `Ubuntu 20.04`, el *shell* que se utiliza de manera predeterminada es **`bash`**. Este *shell* es el que usaremos en este documento.

[1] El proyecto GNU se considera el fundador del movimiento del software libre. Puede leer su filosofía en https://www.gnu.org/gnu/thegnuproject.es.html

AI.2.2. Nociones básicas para el uso de comandos en el *shell*

Para ejecutar instrucciones en el *shell* se deben de tomar en cuenta las nociones básicas:

- Los comandos, nombres de archivos y directorios se deben teclear exactamente. Las letras mayúsculas y minúsculas se consideran como diferentes (*case sensitive*).
- En su forma más común, el sistema operativo utiliza un signo de pesos (`$`) en el *prompt*[2] para indicar que está preparado para aceptar comandos.
- Los comandos se deben ejecutar cumpliendo con la sintaxis. La sintaxis nos indica la manera y el orden correctos de escribir las instrucciones o comandos. En la `Terminal`, la sintaxis para escribir comandos es:

```
comando [parametro1] [parametro2]...
```

Donde:

- **`comando`** es el nombre del programa o instrucción que se desea ejecutar.
- Después del **`comando`**, se escribe una lista de parámetros separados por un espacio.
- Cuando se indican dentro de paréntesis cuadrados, los parámetros son *opcionales*, es decir, podrían omitirse por completo. Esto dependerá del comando en específico; puede suceder que un comando requiera parámetros obligatorios. En tal caso, en la sintaxis el parámetro no estará contenido dentro de paréntesis cuadrados.

Si se desea ejecutar el comando con uno de los parámetros opcionales, se incluye el parámetro pero sin incluir los paréntesis cuadrados. Ejemplo:

```
comando parametro1
```

[2]Se recomienda leer la Sección AI.1.1 «Sobre el *shell prompt*».

Ejemplo con el comando calendario (**cal**). La sintaxis es: **cal** [año]. Esto indica que el parámetro es opcional. Por tanto, se podría ejecutar el comando **cal** de cualquiera de las dos maneras mostradas en los Ejemplos AI.1, AI.2.

Ejemplo AI.1: Terminal. Ejecución de un comando sin parámetros.

```
1 rogelio@workstation:~$ cal
2    Diciembre 2022
3 do lu ma mi ju vi sá
4                1  2  3
5  4  5  6  7  8  9 10
6 11 12 13 14 15 16 17
7 18 19 20 21 22 23 24
8 25 26 27 28 29 30 31
```

Ejemplo AI.2: Terminal. Ejecución de un comando utilizando un parámetro.

```
1  rogelio@workstation:~$ cal 2022
2                                2022
3        Enero                 Febrero                 Marzo
4  \d\o lu ma mi ju vi sá  do lu ma mi ju vi sá  do lu ma mi ju vi sá
5                      1           1  2  3  4  5           1  2  3  4  5
6   2  3  4  5  6  7  8    6  7  8  9 10 11 12    6  7  8  9 10 11 12
7   9 10 11 12 13 14 15   13 14 15 16 17 18 19   13 14 15 16 17 18 19
8  16 17 18 19 20 21 22   20 21 22 23 24 25 26   20 21 22 23 24 25 26
9  23 24 25 26 27 28 29   27 28                  27 28 29 30 31
10 30 31
11 ...
12 ...
```

En la Tabla AI.1 se muestran algunos comandos básicos de GNU/Linux. Para ejecutar estos comandos, puede abrir la aplicación `Terminal`[3], escribir cada comando y presionar la tecla `Enter` ↵ para ejecutar. En el Ejemplo AI.3 se muestra la ejecución de dos comandos en el *shell*. En el Ejemplo AI.3, la línea 1 ejecuta el comando **date**. En la línea 2 se muestra la fecha actual como resultado. La línea 3 permite ejecutar el

[3]Se recomienda leer la Sección AI.1 «Cómo abrir la aplicación `Terminal`».

comando **whoami**, la línea 4 como resultado el nombre del usuario que está trabajando en la sesión actual. Observe que solamente las líneas 1 y 3 muestran el *prompt* por ser precisamente esas líneas las que ejecutan comandos.

Tabla AI.1: Comandos básicos del *shell* en GNU/Linux.

Comando	Descripción
date	Muestra por pantalla el día y la hora.
cal 1873	Muestra el calendario del año 1873.
cal 05 1873	Muestra el calendario de mayo de 1873.
who	Indica cuáles usuarios están conectados en la computadora en ese momento, en qué `Terminal` está y desde cuándo se conectó cada uno.
whoami	Indica cuál es el nombre de usuario que está trabajando en la sesión actual.
man comando	Muestra la información de los manuales de referencia del comando especificado.
clear	Este comando limpia la pantalla de la consola o terminal.

Ejemplo AI.3: Ejecución de comandos básicos en el shell.

```
1 rogelio@workstation:~$ date
2 jue 08 abr 2021 17:38:45 MDT
3 rogelio@workstation:~$ whoami
4 rogelio
```

AI.2.3. Sistema de archivos

La información que se procesa en una computadora con frecuencia requiere ser conservada a lo largo del tiempo, es decir, requiere ser almacenada de manera persistente. Para ello, la información se almacena en archivos. Los archivos se almacenan en dispositivos de almacenamiento secundario, por ejemplo, en discos duros, discos de estado sólido o memorias USB.

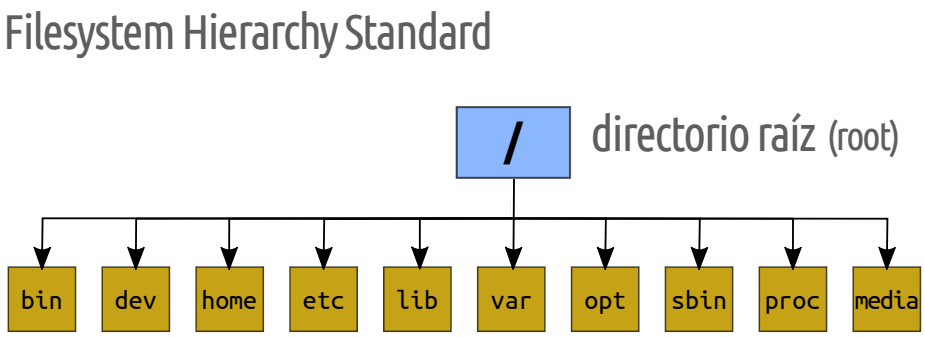

Figura AI.4: Directorios principales contemplados en el *Filesystem Hierarchy Standard* (FHS) [79] [35].

La administración de los archivos es tarea del sistema operativo, específicamente, del sistema de archivos. El sistema de archivos define las reglas para el almacenamiento y recuperación de datos en el sistema operativo. El sistema de archivos define la estructura, nombre, directrices de acceso, de seguridad y de protección de los archivos en el sistema operativo [89].

La estructura del sistema de archivos es la de un árbol. Dentro de ese árbol, los directorios se ordenan de forma jerárquica. Todos los archivos y directorios aparecen bajo el directorio raíz `/`, aunque se encuentre en distintos dispositivos físicos; ver la Figura AI.4.

Desde 1994, con actualizaciones en 2004 y en 2015, se generó el estándar de la Jerarquía del Sistema de Archivos (en inglés: *Filesystem Hierarchy Standard* o *FHS*). El FHS es el estándar del sistema de archivos, brinda requerimientos y directrices para la estructura de archivos y directorios en los sistemas operativos basados en Unix [79] [35]. Ubuntu y varias distribuciones Linux se apegan al estándar FHS [41] [87] [90] [22].

El FHS define lineamientos específicos pero a la vez es extensible. El FHS contempla cómo colocar los directorios principales y sus contenidos en el sistema operativo. En la Tabla AI.2 se mencionan los principales directorios contemplados en el estándar FHS 3.0.

Para cada usuario registrado en el sistema operativo se creará un directorio propio ubicado en el directorio `/home/nombreusuario`.

Por seguridad, el directorio `/home/nombreusuario` de manera predeterminada únicamente puede ser modificado por el usuario propietario, en este caso: `nombreusuario`. Ejemplo, si se registraron los usuarios `sofia` y `rogelio`, se habrá creado un directorio para cada uno tal como muestra la Figura AI.5.

Tabla AI.2: Principales directorios bajo el directorio raíz contemplados en el *Filesystem Hierarchy Standard* (FHS) [79] [35].

Directorio	Descripción
`/bin/`	Órdenes esenciales, archivos binarios (ejecutables) para todos los usuarios (comandos **`date`**, **`cal`**,. . .).
`/dev/`	Ubicación de archivos especiales o de dispositivos.
`/home/`	Directorios de inicio (de datos) de los usuarios.
`/etc/`	Archivos de configuración del sistema.
`/lib/`	Bibliotecas esenciales para los binarios de `/bin` y `/sbin`.
`/var/`	Archivos variables utilizados por programas instalados, como bitácoras (*logs*) y archivos temporales.
`/opt/`	Paquetes de programas de aplicaciones estáticos.
`/sbin/`	Archivos binarios de superusuario esenciales (**`init`**, **`route`**, **`ifup`**..).
`/proc/`	Sistema de archivos que documenta el estado del núcleo (kernel).
`/media/`	contiene subdirectorios que muestran el contenido de dispositivos de almacenamiento extraíbles (USB, DVD,. . .).

El sistema de archivos de Ubuntu, la distribución de GNU/Linux que utilizaremos a lo largo de este documento, se apega al estándar FHS. Por tanto, lo que hemos descrito en esta sección aplica para Ubuntu.

AI.3. Linux: Comandos básicos

En esta sección están incluidos los comandos útiles para manipular archivos y directorios. Se muestran los usos más típicos de cada comando. Si desea conocer la lista completa de parámetros de cada comando puede utilizar la instrucción: **`man`** `nombre-del-`**`comando`**. Ejemplo: **`man ls`**.

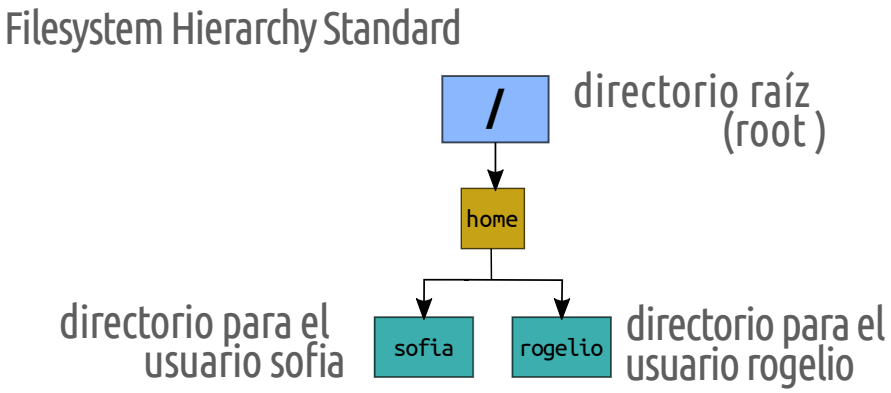

Figura AI.5: Ejemplo. Se han creado dos usuarios: `sofia` y `rogelio`. La figura muestra la ubicación del directorio para cada usuario cumpliendo con los lineamientos del *Filesystem Hierarchy Standard* (FHS) [79] [35].

AI.3.1. Listar el contenido de un directorio

El comando **`ls`** lista información de archivos. En GNU/Linux un directorio es un tipo de archivo que tiene como función agrupar otros archivos y sus nombres. El comando **`ls`**, cuando se trata de un directorio, también brinda información de él y de su contenido.

Sintaxis:

```
ls [opciones] [nombres]
```

Si se omite el parámetro *nombres*, lista el contenido del directorio actual. Si se incluye el parámetro *nombres*, el parámetro puede contener el nombre de uno o más directorios de los cuales se desea visualizar su contenido.

Las **`opciones`** más frecuentes son:

`-l` Muestra la información detallada (*long listing*) de archivos y directorios.

`-a` Incluye en el listado los archivos y directorios ocultos. El nombre de los archivos y directorio ocultos comienza con un punto.

`-d` Muestra solo el nombre de los directorios y sin el contenido.

`-h` Normalmente el comando **`ls`** muestra el tamaño de los archivos en bytes. Con el

parámetro `-h` (*human readable*), **`ls`** muestra el tamaño del archivo convertido a otras unidades de almacenamiento (kilobyte, megabyte, gigagytes, terabytes, petabytes, etc.) para que sean más «legibles para los humanos» (*human-readable units*). Este parámetro se debe utilizar, en combinación con el parámetro `-l`, de cualquiera de estas dos maneras: a) de manera abreviada: `-lh` b) por separado: `-l -h`.

El Ejemplo AI.4, en las líneas 1, 4, 12 y 14, muestra la ejecución del comando **`ls`** con los parámetros más frecuentes. El Ejemplo AI.5, utiliza los parámetros `-lh` para mostrar un listado detallado con formato «más legible para los humanos».

Ejemplo AI.4: Uso del comando ls con los parámetros más frecuentes.

```
1  rogelio@workstation:~$ ls
2  Descargas   Escritorio  Música      Público
3  Documentos  Imágenes    Plantillas  Vídeos
4  rogelio@workstation:~$ ls -a
5  .             Documentos  .profile
6  ..            Escritorio  Público
7  .bash_logout  .gnupg      Vídeos
8  .bashrc       Imágenes
9  .cache        .local
10 .config       Música
11 Descargas     Plantillas
12 rogelio@workstation:~$ ls -d
13 .
14 rogelio@workstation:~$ ls -l
15 total 32
16 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Descargas
17 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Documentos
18 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Escritorio
19 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Imágenes
20 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Música
21 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Plantillas
22 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Público
23 drwxr-xr-x 2 rogelio rogelio 4096 abr 10 12:21 Vídeos
```

Cuando usted ejecuta **`ls`** con el parámetro **`-l`** obtendrá una salida conformado por nueve (9) columnas. La Figura AI.6 muestra un ejemplo del resultado obtenido.

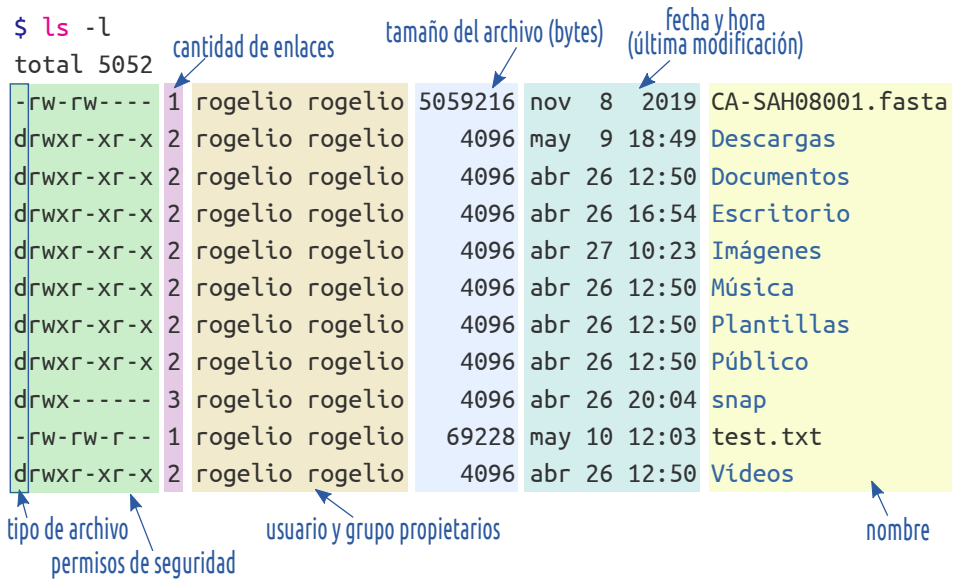

Figura AI.6: Resultado obtenido al ejecutar el comando `ls -l`. Se visualizan nueve (9) columnas con información detallada de los archivos almacenados en el directorio.

- La primera columna muestra el tipo de archivo. Muestra una letra `d` si el elemento es un directorio, un `-` si es un archivo *regular*. En la Tabla AI.3 puede ver la lista completa de tipos de archivo. También en esa columna aparecen nueve (9) caracteres (letras o guiones) referente a los permisos de seguridad del elemento (para más detalles puede revisar la Sección AI.4.7, página 174).
- La segunda columna es el número de enlaces al archivo.
- La tercera y cuarta columna son el usuario propietario del archivo y el grupo de usuarios Linux (Unix) al que este archivo pertenece. Esto es relevante si usted está compartiendo el trabajo con otros usuarios y editan archivos de manera conjunta.
- La quinta columna muestra el tamaño del archivo en bytes.
- La columnas sexta, séptima y octava muestran la fecha y hora de la última modificación del archivo. Si el elemento es un directorio, entonces muestra la fecha y hora más recientes en que un archivo fue creado o eliminado en ese directorio.

- La novena columna contiene el nombre del archivo.

Tabla AI.3: Tipos de archivo. La tabla muestra los caracteres que pueden aparecer en la primera columna del resultado obtenido al ejecutar el comando `ls -l`. Los tipos más utilizados son `-`, `d` y `l`.

Carácter	Descripción
`-`	archivo *regular*
`d`	directorio
`c`	archivos de dispositivo de caracteres
`b`	archivo de dispositivo de bloque
`s`	archivo de socket local
`p`	*pipe* (tubería) con nombre
`l`	enlace simbólico

Ejemplo AI.5: Uso del comando ls para mostrar el listado de archivos en formato detallado y la columna tamaño en formato «legible por humanos».

```
1  rogelio@workstation:~$ ls -lh
2  total 32K
3  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Descargas
4  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Documentos
5  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Escritorio
6  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Imágenes
7  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Música
8  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Plantillas
9  drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Público
10 drwxr-xr-x 2 rogelio rogelio 4.0K abr 10 12:21 Vídeos
```

La Figura AI.7 muestra el resultado que obtendrá: una salida conformada por nueve (9) columnas. La quinta columna mostrará el tamaño del archivo convertido a otras unidades de almacenamiento (`K` para kilobytes, `M` para megabytes, `G` para gigagytes, `T` para terabytes, `P` para petabytes, etc.).

También es posible mostrar el contenido de un directorio en específico, no del directorio actual. Las instrucciones contenidas en las líneas 1, 6, 12, 14 y 21 del

Figura AI.7: Resultado obtenido al ejecutar el comando **`ls -lh`**. Se visualizan nueve columnas con información detallada de los archivos almacenados en el directorio. La quinta columna muestra el tamaño del archivo convertido a otras unidades de almacenamiento.

Ejemplo AI.6 utilizan ruta absoluta para mostrar el contenido del directorio `/bin/` al ejecutar el comando **`ls`** con diferentes parámetros.

AI.3.2. Directorio Actual

El comando **`pwd`** (*print working directory*) visualiza o imprime la ruta del directorio de trabajo actual.

Sintaxis:

```
pwd
```

Cuando se abre la aplicación `Terminal` de manera predeterminada el directorio de trabajo actual es `/home/nombre-de-usuario`. Ese directorio también es posible representarlo con el signo de virgulilla (~, la raya que tiene encima la letra eñe). En el Ejemplo AI.7, se muestra la salida que obtendrá al abrir la `Terminal` y ejecutar el comando **`pwd`**, si ha iniciado sesión con el usuario `rogelio`.

Ejemplo AI.6: Mostrar el contenido del directorio /bin/, utilizando ruta absoluta.

```
rogelio@workstation:~$ ls /bin/
'['                                  nl
 411toppm                            nm
 aa-enabled                          nm-applet
...
rogelio@workstation:~$ ls -a /bin/
  .                                   nisdomainname
 ..                                  nl
'['                                  nm
 411toppm                            nm-applet
...
rogelio@workstation:~$ ls -d /bin/
/bin/
rogelio@workstation:~$ ls -l /bin/
total 217516
-rwxr-xr-x 1 root root       59736 sep  5  2019 '['
-rwxr-xr-x 1 root root       10104 abr 23  2016  411toppm
-rwxr-xr-x 1 root root       31248 may 19  2020  aa-enabled
-rwxr-xr-x 1 root root       35344 may 19  2020  aa-exec
...
rogelio@workstation:~$ ls -lh /bin/
total 213M
-rwxr-xr-x 1 root root      59K sep  5  2019 '['
-rwxr-xr-x 1 root root     9.9K abr 23  2016  411toppm
-rwxr-xr-x 1 root root      31K may 19  2020  aa-enabled
-rwxr-xr-x 1 root root      35K may 19  2020  aa-exec
...
```

AI.3.3. Rutas para localizar archivos

La ruta (*path*) de un archivo o directorio es la secuencia de directorios que se deben recorrer para acceder a ese archivo o directorio. La Figura AI.8 muestra dos ejemplos de rutas. El directorio raíz (o inicial) se representa con `/` (diagonal o *slash* en inglés). A partir de ese directorio, se va escribiendo la secuencia de directorios que se necesita recorrer para acceder al archivo o directorio deseado. Se utiliza la `/` como separador.

Ejemplo AI.7: Comando pwd obtiene el directorio de trabajo actual.

```
rogelio@workstation:~$ pwd
/home/rogelio
```

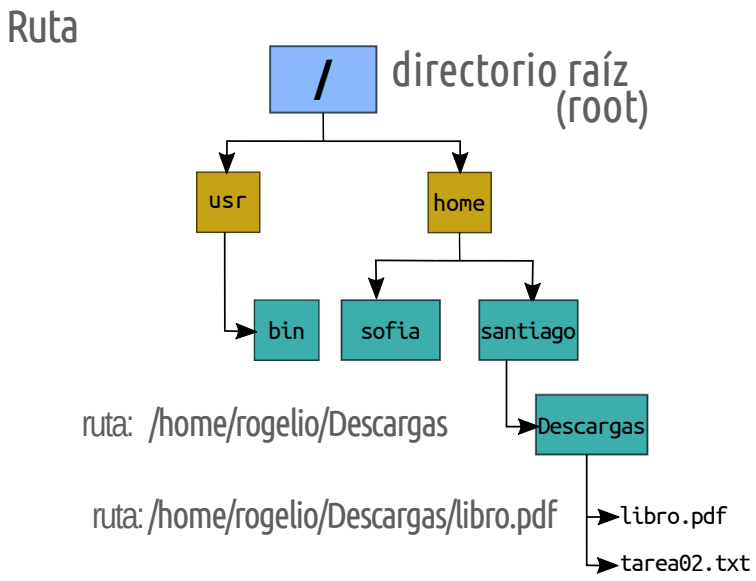

Figura AI.8: Dos ejemplos de rutas absolutas: a) para acceder a un directorio y b) para acceder a un archivo.

Existen dos tipos de rutas que se pueden utilizar para localizar archivos y directorios: rutas absolutas y rutas relativas.

- **Rutas Absolutas**. Son aquellas que comienzan con `/` (diagonal o *slash* en inglés). Una ruta absoluta especifica una ubicación desde la raíz `/`. Las rutas absolutas incluye la ubicación completa al archivo o directorio sin importar cuál sea el directorio de trabajo actual. El Ejemplo AI.8 muestra el uso de rutas absolutas utilizadas en el comando **`ls`**. En el Ejemplo AI.8 podemos notar que se utiliza la `/` (diagonal o *slash* en inglés) como separador para escribir la secuencia de directorios.

- **Rutas Relativas**. Estas rutas no comienzan con `/`. Las rutas relativas consideran como «punto de partida» el directorio de trabajo actual; a partir de esta

Ejemplo AI.8: Comando ls. Uso de rutas absolutas.

```
rogelio@workstation:~$ ls /usr/bin
'['                                 nl
 411toppm                           nm
 aa-enabled                         nm-applet
 aa-exec                            nmcli
...
rogelio@workstation:~$ ls /home/rogelio/Descargas
```

ruta es posible especificar la secuencia de directorios que faltaría recorrer para acceder al archivo o directorio deseado.

Ejemplo AI.9: Comando ls. Uso de rutas relativas.

```
1 rogelio@workstation:~$ pwd
2 /home/rogelio
3 rogelio@workstation:~$ ls Descargas
4 rogelio@workstation:~$ ls /home/rogelio/Descargas
```

En el Ejemplo AI.9 se desea mostrar el contenido del directorio `/home/rogelio/Descargas`. En el primera línea se ejecuta el comando **`pwd`** para visualizar el directorio de trabajo actual. Tal como se aprecia en la línea 2, el directorio de trabajo actual es: `/home/rogelio`. Por tanto, en la línea 3, se utiliza la ruta relativa `Descargas` que hace referencia a la ruta `/home/rogelio/Descargas`. En este ejemplo, la línea 3 y la línea 4 ejecutan la misma acción; la diferencia radica en que la línea 3 utiliza ruta relativa y la línea 4 utiliza ruta absoluta.

Existen dos nombres que se pueden utilizar para escribir una ruta; **un sólo punto (.)** se refiere al directorio actual, y un **par de puntos (..)** al directorio inmediatamente superior. Estas rutas se pueden utilizar en cualquier comando de GNU/Linux que reciba una ruta como parámetro.

El Ejemplo AI.10 muestra el contenido de directorios utilizando el comando **`ls`** y rutas relativas utilizando un punto (**.**) y un par de puntos (**..**). En la línea 1 se ejecuta

Ejemplo AI.10: Comando ls. Uso de . y de ..

```
1 rogelio@workstation:~$ pwd
2 /home/rogelio
3 rogelio@workstation:~$ ls ./Documentos
4 rogelio@workstation:~$ ls ./Descargas
5 rogelio@workstation:~$ ls ..
6 rogelio  sofia
```

el comando **pwd** para visualizar el directorio de trabajo actual. En la línea 2 se aprecia que el directorio de trabajo actual es: `/home/rogelio`. En la línea 3 se hace referencia al directorio `Documentos` que está ubicado en dentro del directorio de trabajo actual; el punto (**.**) hace referencia explícita al directorio de trabajo actual. La línea 4, también utiliza un punto (**.**) para referirse al directorio de trabajo actual. Finalmente, la línea 5 ejecuta el comando **ls** utilizando un par de puntos (**..**) para referirse al directorio padre, o directorio inmediatamente superior, del directorio de trabajo actual; en este caso se refiere al directorio `/home`. En la línea 6 se observa el resultado obtenido al ejecutar la instrucción contenida en la línea 5, muestra los elementos del directorio `/home`.

AI.4. Linux: Operaciones con archivos y directorios

En esta sección se incluyen comandos para realizar operaciones comunes en archivos y directorios: navegar, crear y eliminar directorios; utilizar expresiones regulares para describir cadenas; asignar permisos de acceso; desplegar contenido y realizar búsquedas de archivos.

AI.4.1. Navegar en directorios. `cd`

El comando **cd** (*change directory*) permite posicionar un directorio específico como directorio de trabajo actual.

Sintaxis:

```
cd [directorio]
```

El parámetro **directorio** es una ruta absoluta o relativa al directorio específico en el que desea ubicar el directorio de trabajo actual. En el Ejemplo AI.11, en la línea 1 se ejecuta el comando **cd** /bin, con ello se establece el directorio /bin como directorio de trabajo actual. En la línea 2, se observa el *prompt* que muestra en su parte final /bin para mostrar que ese es el directorio de trabajo actual.

Ejemplo AI.11: Especificar el directorio /bin como directorio de trabajo actual.

```
1 rogelio@workstation:~$ cd /bin
2 rogelio@workstation:/bin$
```

Ejemplo AI.12: Especificar un directorio inmediatamente superior (arriba del directorio de trabajo actual) como directorio de trabajo actual.

```
1 rogelio@workstation:/bin$ cd ..
2 rogelio@workstation:/$ cd /
3 rogelio@workstation:/$
```

Al ejecutar la línea 1 del Ejemplo AI.12 se ubicará en el directorio / . También se podría indicar la ruta absoluta. En este ejemplo, las dos instrucciones (línea 1 y 2) son equivalentes.

Ejemplo AI.13: En caso de omitirse el parámetro directorio, entonces el comando cd cambia al directorio del usuario (/home/nombre-del-usuario).

```
1 rogelio@workstation:~$ cd /
2 rogelio@workstation:/$ cd
3 rogelio@workstation:~$
4 rogelio@workstation:~$ pwd
5 /home/rogelio
```

En el Ejemplo AI.13 se ejecuta, en la línea 2, el cambio del directorio de trabajo sin especificar ningún parámetro. Esta instrucción establece como directorio actual el directorio de datos del usuario: ~ o `/home/rogelio`; lo cual se puede observar en las líneas 3, 4 y 5.

Ejemplo AI.14: Utilizar el parámetro - (guión medio) que representa el directorio de trabajo inmediato anterior utilizado.

```
1 rogelio@workstation:~$ cd /
2 rogelio@workstation:/$ cd -
3 /home/rogelio
4 rogelio@workstation:~$ cd -
5 /
6 rogelio@workstation:/$
```

En el Ejemplo AI.14, en la línea 1 se tiene como directorio de trabajo al directorio ~ o `/home/rogelio`, ejecuta el comando **`cd /`** para establecer el directorio `/` como directorio de trabajo actual. La línea 2 muestra como directorio de trabajo actual al directorio `/`, ejecuta **`cd -`** para referirse al directorio de trabajo utilizado previamente, en este caso se refiere al directorio `/home/rogelio`. El *shell* además de establecer el nuevo directorio de trabajo, también imprime la nueva ruta. La línea 4, ejecuta nuevamente **`cd -`**, en esta ocasión se refiere al directorio `/`.

AI.4.2. Crear directorios. `mkdir`

El comando **`mkdir`** permite crear uno o más directorios.

Sintaxis:

```
mkdir [-p] nombres
```

El parámetro `-p` se utiliza para crear un árbol de directorios.

El parámetro **`nombres`** puede ser una ruta absoluta o relativa por cada directorio que desee crear.

Ejemplo AI.15: Crear el directorio cursolinux.

```
1 rogelio@workstation:~$ mkdir cursolinux
2 rogelio@workstation:~$ mkdir /home/rogelio/cursolinux
3 mkdir: no se puede crear el directorio «/home/rogelio/cursolinux»: El archivo ya
      existe
```

En el Ejemplo AI.15 ambas instrucciones crean el directorio `cursolinux` dentro del directorio `/home/rogelio/`. La línea 3 muestra el error obtenido al intentar crear por segunda vez el mismo directorio.

Es posible crear más de un directorio con una única instrucción, para ello, se pueden incluir los nombres de cada directorio separados por espacio. El Ejemplo AI.16 muestra cómo utilizar un único comando **`mkdir`** para crear múltiples directorios. En el Ejemplo AI.16, la línea 1 crea los directorios `cursoR` y `cursopython` dentro del directorio de trabajo actual, en el directorio: `/home/rogelio`. Al ejecutar la línea 1 el *shell* no muestra ningún mensaje, significa que la instrucción fue ejecutada con éxito. La línea 2 es equivalente a la línea 1, solo que utiliza rutas absolutas (rutas completas) que están separadas entre ellas por un espacio. Las líneas 5 y 7, en conjunto, son equivalentes a las líneas 1 y 2. Las líneas 9 y 11, en conjunto son equivalentes a las líneas 1 y 2. Observe que las líneas 3, 4, 6, 8, 10 y 12 muestran mensajes de error indicando que el directorio que se desea crear ya existe.

El comando **`mkdir`** puede usarse con el parámetro `-p` para crear un árbol de directorios. Esto se refiere a que si no existen los directorios incluidos en la ruta del nuevo directorio, los crea también.

En el Ejemplo AI.17, si el directorio `misprogramas` no existiera, lo crea; posteriormente dentro de éste crea el directorio `linux`. La instrucción mostrada en el Ejemplo AI.17 es equivalente a las instrucciones del Ejemplo AI.18.

Ejemplo AI.16: Crear dos directorios dentro del directorio actual.

```
1  rogelio@workstation:~$ mkdir cursoR cursopython
2  rogelio@workstation:~$ mkdir /home/rogelio/cursoR /home/rogelio/cursopython
3  mkdir: no se puede crear el directorio «/home/rogelio/cursoR»: El archivo ya
       existe
4  mkdir: no se puede crear el directorio «/home/rogelio/cursopython/»: El archivo
       ya existe
5  rogelio@workstation:~$ mkdir cursoR
6  mkdir: no se puede crear el directorio «cursoR»: El archivo ya existe
7  rogelio@workstation:~$ mkdir cursopython
8  mkdir: no se puede crear el directorio «cursopython»: El archivo ya existe
9  rogelio@workstation:~$ mkdir /home/rogelio/cursoR
10 mkdir: no se puede crear el directorio «/home/rogelio/cursoR»: El archivo ya
       existe
11 rogelio@workstation:~$ mkdir /home/rogelio/cursopython
12 mkdir: no se puede crear el directorio «/home/rogelio/cursopython»: El archivo
       ya existe
```

Ejemplo AI.17: Crear el directorio linux dentro del directorio misprogramas.

```
1  rogelio@workstation:~$ mkdir -p misprogramas/linux
```

AI.4.3. Eliminar directorios. `rmdir`

El comando **`rmdir`** permite eliminar uno o más directorios. El o los directorios que se desean eliminar deben estar están vacíos, de lo contrario, es posible utilizar el comando **`rm`** `-r`.

Sintaxis:

```
rmdir [-p] nombre
```

El parámetro `-p` se utiliza para eliminar un árbol de directorios.

El parámetro `nombre` puede ser una ruta absoluta o relativa.

En el Ejemplo AI.19 las líneas 1 y 2 son equivalentes, ambas eliminan el directorio `cursolinux`. La línea 1 utiliza ruta relativa. La línea 2 utiliza ruta absoluta. La línea

Ejemplo AI.18: Creación de múltiples directorios utilizando un comando para cada directorio creado.

```
1 rogelio@workstation:~$ mkdir misprogramas
2 rogelio@workstation:~$ cd misprogramas
3 rogelio@Linux:~/misprogramas$ mkdir linux
```

Ejemplo AI.19: Eliminar el directorio cursolinux en el directorio de trabajo actual.

```
1 rogelio@workstation:~$ rmdir cursolinux
2 rogelio@workstation:~$ rmdir /home/rogelio/cursolinux
3 rmdir: fallo al borrar '/home/rogelio/cursolinux': No existe el archivo o el
      directorio
```

3 muestra un mensaje de error por tratar de eliminar el directorio por segunda vez, el directorio ya no existe.

Es posible eliminar más de un directorio con una única instrucción, para ello, se pueden incluir los nombres de cada directorio separados por espacio. En el Ejemplo AI.20, las líneas 1 y 2 son equivalentes. La línea 1 utiliza ruta relativa. La línea 2 utiliza ruta absoluta. Las líneas 3 y 4 muestran mensajes de error al intentar eliminar los directorios por segunda vez, puesto que los directorios ya no existen.

Ejemplo AI.20: Eliminar los directorios cursoR y cursopython ambos ubicados en el directorio de trabajo actual.

```
1 rogelio@workstation:~$ rmdir cursoR cursopython
2 rogelio@workstation:~$ rmdir /home/rogelio/cursoR /home/rogelio/cursopython
3 rmdir: fallo al borrar '/home/rogelio/cursoR': No existe el archivo o el
      directorio
4 rmdir: fallo al borrar '/home/rogelio/cursopython': No existe el archivo o el
      directorio
```

En el Ejemplo AI.21, la línea 1 elimina el directorio `linux`, en caso de que el directorio padre `misprogramas` haya quedado vacío (sin ningún elemento dentro de él) también se eliminará. La línea 2 también elimina el directorio `linux` pero utiliza

Ejemplo AI.21: Eliminar el árbol de directorios misprogramas/linux. Se desea eliminar el último directorio: linux, en caso que como resultado de esta eliminación el directorio padre queda vacío, entonces también será eliminado.

```
1 rogelio@workstation:~$ rmdir -p misprogramas/linux
2 rogelio@workstation:~$ rmdir -p /home/rogelio/misprogramas/linux
3 rmdir: fallo al borrar '/home/rogelio/misprogramas/linux': No existe el archivo
    o el directorio
```

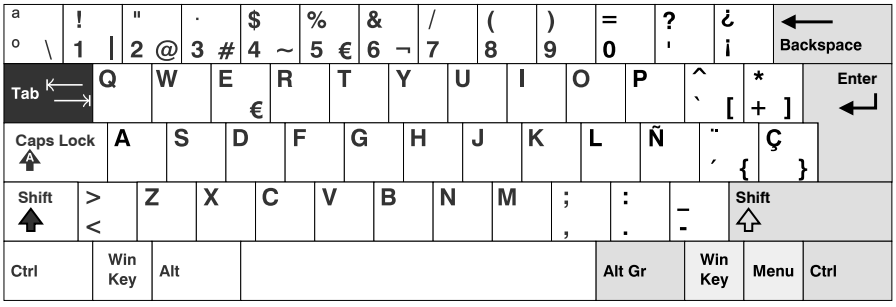

Figura AI.9: Ubicación de la tecla ⇥ tabulador o `tab` en el teclado. La imagen fue obtenida de wikimedia.

ruta absoluta, en este caso intentará eliminar cada uno de los directorios padres que aparecen en la ruta, en caso de hayan quedado vacíos. La línea 3 muestra un menssaje de error porque el directorio `linux` ya no existe, fue borrado previamente por la instrucción de la línea 1.

AI.4.4. Completar comandos

La `Terminal` permite el autocompletado de comandos, nombres de archivos o directorios usando la tecla ⇥ (tabulador o `tab`). El autocompletadoes una de las acciones más utilizadas porque permite ahorrar tiempo, escribir menos y evitar errores de escritura. La tecla ⇥ (tabulador o `tab`) se ubica a la izquierda de la letra `Q`, ver Figura AI.9.

La función de autocompletado funciona así:

1. El usuario escribe uno o más caracteres y presiona la tecla ⇥. Si con estos caracteres es suficiente para identificar de manera única a un comando, nombre de archivo o directorio, entonces se autocompleta.
   De lo contrario, la `Terminal` emite un sonido de error (bip) para indicar que no fue posible autocompletar.

   a. Si el usuario presiona nuevamente la tecla ⇥, entonces la `Terminal`, a manera de guía, le muestra los elementos que coinciden con esos caracteres. El usuario puede continuar escribiendo caracteres para identificar de manera única a algún elemento, como se describe en el Paso 1.

Ejemplo AI.22: Auto-completado. Si escribimos us, y presionamos la tecla (tab), se emite el sonido de error para indicar que no fue posible identificar a ningún elemento, si usted presiona nuevamente la tecla (tab), se mostrarán los comandos que tienen como primer y segundo carácter: us.

```
1 rogelio@workstation:~$ us
2 usb-creator-gtk            usb_printerid
3 usb-devices                usbreset
4 usbhid-dump                useradd
5 usb_modeswitch             userdel
6 usb_modeswitch_dispatcher  usermod
7 usbmuxd                    users
8 rogelio@workstation:~$ us
```

Finalmente, como se observa en la línea 8 del Ejemplo AI.22, el *prompt* queda listo para que usted continúe tecleando el resto de la ruta o instrucción.

También es aplicable al auto-completado de una ruta de directorios o bien en completar el nombre de un archivo. En el Ejemplo AI.23 se muestra que para escribir una ruta se puede iniciar con el nombre del directorio, acto seguido, presionar la tecla ⇥ (`tab`) para utilizar la función autocompletado. La función autocompletado listará los archivos y los subdirectorios del directorio **`/var/`**. El Ejemplo AI.24 muestra que si de los elementos listados, usted desea referirse al directorio `metrics/`, puede teclear solamente `me` seguido de la tecla ⇥ (`tab`); con esos dos caracteres es posible identificar de manera única a ese elemento, por tanto, funciona el auto-completado.

El Ejemplo AI.24, en la línea 4, muestra el directorio `metrics` como resultado del auto-completado.

Ejemplo AI.23: Auto-completado. Si escribimos /var/ y la tecla tab, la aplicación Terminal nos mostrará los archivos y los subdirectorios del directorio /var/.

```
1 rogelio@workstation:~$ /var/
2 backups/ crash/   local/   log/     metrics/ run/     spool/
3 cache/   lib/     lock/    mail/    opt/     snap/    tmp/
4 rogelio@workstation:~$ /var/
```

Ejemplo AI.24: Autocompletado. Teclear me y la tecla tab para completar el nombre del directorio metrics.

```
1 rogelio@workstation:~$ /var/
2 backups/ crash/   local/   log/     metrics/ run/     spool/
3 cache/   lib/     lock/    mail/    opt/     snap/    tmp/
4 rogelio@workstation:~$ /var/metrics
```

De manera general, para completar un nombre de archivo o directorio, simplemente escribimos los primeros caracteres que componen su nombre, seguido de la tecla ⇤ (`tab`).

El Ejemplo AI.25 muestra el resultado de utilizar la función de autocompletado para ubicar el directorio `/home/rogelio/Documentos` como directorio de trabajo. Para esta tarea, se utiliza el comando **`cd`**. Para utilizar la función autocompletar, usted puede teclear la secuencia **`cd`** `/h`⇤`sa`⇤`Do`⇤ como se describe a continuación:

1. Escriba el comando **`cd`** seguido de un espacio
2. Escriba `/h` y presione la tecla ⇤ (`tab`)
3. Escriba `sa` y presione la tecla ⇤ (`tab`)
4. Finalmente escriba `Do` y presione la tecla ⇤ (`tab`)

Ejemplo AI.25: Ejemplo de escritura de una ruta utilizando la función autocompletar.

```
cd /home/rogelio/Documentos
```

El Ejemplo AI.26 muestra el contenido del directorio `/home/rogelio` obtenido al presionar la tecla ⇥. Para continuar, usted puede realizar los siguientes ejercicios de autocompletado. Si usted escribe lo siguiente y presiona la tecla ⇥ (`tab`):

a) `De` ⇥, se completará `Descargas`.

b) `Do` ⇥, se completará `Documentos`.

c) `P` ⇥, no mostrará nada, generará un sonido de error (*bip*) puesto que no se logra identificar a un elemento de manera única.

d) `P` ⇥ ⇥, mostrará `Plantillas` y `Público`, son los elementos que coinciden en iniciar con esos caracteres.

e) `P` ⇥ ⇥ `l` ⇥, completará `Plantillas`.

Ejemplo AI.26: Autocompletado. Visualizar el listado del directorio de usuario /home/rogelio al presionar la tecla tab.

```
rogelio@workstation:~$ ls /home/rogelio/
Descargas   Escritorio  Música      Público  Vídeos
Documentos  Imágenes    Plantillas  snap
```

AI.4.5. Expresiones regulares. Comodines: * y ?

Una cadena es una secuencia de caracteres que se trata como una sola unidad. Una cadena puede incluir letras, dígitos y varios caracteres especiales (tales como `*`, `?`, `[`, `]`). La información que se considera como cadena puede ser de distinta naturaleza, tal como lo muestra la Tabla AI.4, es posible que las cadenas contengan nombres propios, direcciones, números telefónicos, códigos postales o incluso secuencias genómicas.

Tabla AI.4: Ejemplos de cadenas que almacenan información de distinta naturaleza.

Cadena	Naturaleza de la información
`Rogelio Prieto`	Nombre
`Josefa Ortiz s/n`	Dirección
`Culiacán, Sinaloa`	Ciudad y estado
`(667) 7581424`	Número teléfonico
`80013`	Código postal
`CAGTAATTCC`	Secuencia genómica

Una característica de *bash* es que permite utilizar expresiones regulares (patrones). Una expresión regular es una cadena que describe a un conjunto de cadenas.

Definición matemática: una expresión regular `r` hace coincidencia con una cadena `s` si `s` está en el conjunto de cadenas descrito por `r`. [26]

Para escribir expresiones regulares es posible utilizar caracteres especiales. A estos caracteres especiales también se les conoce como *comodines* (en inglés, *wildcards*). Los comodines tienen un significado adicional a lo que representan literalmente. La Tabla AI.5 muestra los comodines que se utilizan en **bash**.

Tabla AI.5: Comodines utilizados en **bash** para escribir expresiones regulares.

Comodín	Uso/significado
`*`	Cualquier cadena (secuencia de caracteres), también puede hacer coincidencia con cadena nula (cadena con ningún carácter).
`?`	Cualquier carácter.
`[caracteres]`	Cualquiera de los caracteres que estén incluidos dentro de los corchetes (paréntesis cuadrados). Se puede utilizar un guión para indicar un rango (ejemplo: `[a-z]`, `[0-9]`)
`[!caracteres]`	Cualquier carácter que *no* esté dentro de los corchetes.

La Tabla AI.6 muestra ejemplos de expresiones regulares (patrones) escritas utilizando los comodines `*` y `?`. La columna 2 contiene una explicación sobre las coincidencias que tendría cada expresión regular.

Tabla AI.6: Ejemplos de expresiones regulares (patrones) que utilizan los comodines * y ?.

Expresión regular	Coincide con
`n*`	Cadenas que empiezan con el carácter `n`
`*s`	Cadenas que terminan con el carácter `s`
`*c*`	Cadenas que contengan un carácter `c`
`?s`	Cadenas cuyo segundo carácter es `s`
`???e`	Cadenas de cuatro caracteres, cuyo cuarto carácter es `e`
`a?s`	Cadenas de tres caracteres, el primer carácter es `a`, el segundo carácter es un carácter cualquiera y el tercer carácter es una `s`
`a?s*`	Cadenas de tres o más caracteres, cuyo primer carácter es `a`, el segundo es un carácter cualquiera, el tercero es una `s` y el resto de los caracteres pueden ser cualquier cadena e incluso ningún carácter
`?i*s`	Cadenas donde el primer carácter es un carácter cualquiera, el segundo carácter es `i` y terminan con `s`

Es posible utilizar expresiones regulares para describir cadenas que permitan seleccionar un grupo de nombres de archivos. En *bash* las expresiones regulares se pueden utilizar en cualquier comando que requiera un nombre de archivo o ruta como parámetro.

Para ilustrar este uso, en el Ejemplo AI.27 se utilizan expresiones regulares para indicar al comando **`ls`** que muestre solamente el listado de archivos en los cuales el nombre coincide con una expresión regular.

En el Ejemplo AI.27, la línea 1 lista todos los archivos cuyo nombre inicia con la letra `c`. La línea 2 lista todos los archivos cuyo nombre termina con la letra `c`. La línea 3 lista todos los archivos cuyo nombre contiene la letra `c`. La línea 4 lista todos los archivos cuyo nombre inicia con la cadena `ca`.

En el Ejemplo AI.28, la línea 1 lista todos los archivos cuyo nombre contiene la letra `e` en la cuarta posición y el nombre tiene cuatro (4) caracteres. La línea 2 lista todos los archivos cuyo nombre contiene la letra `t` en la tercera posición y después puede contener cero o más caracteres. La línea 3 lista todos los archivos cuyo

Ejemplo AI.27: Listar archivos o directorios ubicados en /bin/ cuyos nombres coincidan con una expresión regular.

```
1  rogelio@workstation:~$ ls /bin/c*
2  /bin/c++                /bin/cmp
3  /bin/c89                /bin/cmuwmtopbm
4  /bin/c89-gcc            /bin/codepage
5  ...
6  rogelio@workstation:~$ ls /bin/*c
7  /bin/aa-exec            /bin/perli11ndoc
8  /bin/allec              /bin/pfb2t1c
9  /bin/bc                 /bin/pic
10 ...
11 rogelio@workstation:~$ ls /bin/*c*
12 /bin/aa-exec            /bin/lz4c
13 /bin/aconnect           /bin/lz4cat
14 /bin/acpi_listen        /bin/lzcat
15 ...
16 rogelio@workstation:~$ ls /bin/ca*
17 bin/cachepic            /bin/captoinfo
18 /bin/cal                /bin/cat
19 /bin/calendar           /bin/catchsegv
20 ...
```

nombre contiene la letra `t` en la tercera posición y después puede contener cero o más caracteres, al final tiene una letra `y`. La línea 4 lista todos los archivos cuyo nombre contiene la letra `t` en la tercera posición, después puede contener cero o más caracteres, al final tiene un carácter cualquiera.

Es posible escribir expresiones regulares para indicar que en una posición específica de la cadena puede aparecer cualquier elemento de una lista o conjunto de caracteres. Para escribir estas listas se utilizan los corchetes (`[]` o paréntesis cuadrados). La Tabla AI.5 muestra la sintaxis para el uso de los corchetes. La Tabla AI.7 muestra ejemplos de expresiones regulares utilizando corchetes.

Ejemplo AI.28: Listar archivos o directorios ubicados en /bin/ cuyos nombres coincidan con una expresión regular.

```
rogelio@workstation:~$ ls /bin/???e
/bin/date  /bin/free  /bin/more  /bin/node  /bin/size  /bin/true
/bin/file  /bin/make  /bin/nice  /bin/rake  /bin/time
rogelio@workstation:~$ ls /bin/??t*
/bin/antigenic                  /bin/extractseq              /bin/
    ontogetobsolete
/bin/apt                        /bin/fmt                     /bin/ontogetroot
...
rogelio@workstation:~$ ls /bin/??t*y
/bin/apt-add-repository  /bin/apt-key  /bin/dotty  /bin/stty
rogelio@workstation:~$ ls /bin/??t*?
/bin/antigenic                  /bin/geteltorito             /bin/ontogetsibs
/bin/apt-add-repository         /bin/getent                  /bin/ontogetup
```

Tabla AI.7: Ejemplos de expresiones regulares (patrones) que utilizan los corchetes `[]` para indicar una lista de caracteres.

Expresión regular	Coincide con
`ca[eln]`	cadenas de tres caracteres, que inician con `ca` y el tercer carácter puede ser `l`, `n` o `e`. Es decir, las coincidencias son : `cae`, **`cal`** y `can`.
`[abc]*`	cadenas que empiezan con cualquiera de los caracteres `a`, `b` o `c`. El resto de los caracteres pueden ser cualquier cadena e incluso ningún carácter.
`genoma[0-9][0-9][0-9]`	cadenas que inician con `genoma` seguido de tres dígitos. El `[0-9]` se refiere al rango del número cero al número nueve. Utilizar rangos permite enumerar caracteres de una manera breve.
`[AT][CG]`	cadena de dos caracteres, el primero puede ser `A` o `T`, el segundo carácter puede ser `C` o `G`.
`*[AT][CG]*`	cadena que contenga un carácter `A` o `T` seguido de un carácter `C` o `G`. Ejemplo de cadenas que coinciden: `AC`, `AG`, `TC`, `TG`, `AAAACN`, `NAGNTT`
`AAAT?T*`	cadenas donde el primer, segundo y tercer carácter es `A`, el cuarto y sexto carácter es `T`, el cuarto es un carácter cualquiera, el resto de los caracteres pueden ser cualquier cadena e incluso ningún carácter.

El Ejemplo AI.29 contiene instrucciones que utilizan expresiones regulares para indicar al comando **ls** que muestre solamente el listado de archivos en los cuales el nombre coincide con una expresión regular que usa corchetes (`[]` o paréntesis cuadrados) para indicar una lista de caracteres.

Ejemplo AI.29: Listar archivos o directorios ubicados en /bin/ cuyos nombres coincidan con una expresión regular.

```
1  rogelio@workstation:~$ ls /bin/e[vx]*
2  /bin/evince              /bin/ex        /bin/expiry
3  /bin/evince-previewer    /bin/exceltex  /bin/expr
4  /bin/evince-thumbnailer  /bin/expand    /bin/extractbb
5  ...
6  rogelio@workstation:~$ ls /bin/*[jq]
7  bin/dvilj  /bin/gslj     /bin/pnmhisteq  /bin/seq
8  /bin/gsbj   /bin/lpq      /bin/ppmtolj    /bin/ucfq
9  /bin/gsdj   /bin/pbmtolj  /bin/ppmtopj    /bin/uniq
10 ...
11 rogelio@workstation:~$ ls /bin/*.fast[aq]
12 ls: no se puede acceder a '/bin/*.fast[aq]': No existe el archivo o el
       directorio
```

En el Ejemplo AI.29, la línea 1 lista todos los archivos cuyo nombre inicia la letra `e`, el segundo carácter puede ser `v` o `x`, después puede contener cero o más caracteres. La línea 2 lista todos los archivos cuyo nombre termina con cualquiera de las letras `j` o `q`. La línea 3 lista todos los archivos cuya extensión (la parte después del punto) puede ser `fasta` o `fastq`.

AI.4.6. Edición de texto en línea de comandos. nano

En Ubuntu se incluye el editor de textos **nano** (*Nano's ANOther editor*). **nano** es un sencillo editor de textos para la `Terminal`, permite insertar y editar texto para almacenarlo en archivos.

El editor **nano** también permite realizar las operaciones: de búsqueda, abrir múltiples archivos, deshacer/rehacer, desplazamiento por línea, identificación y resaltado

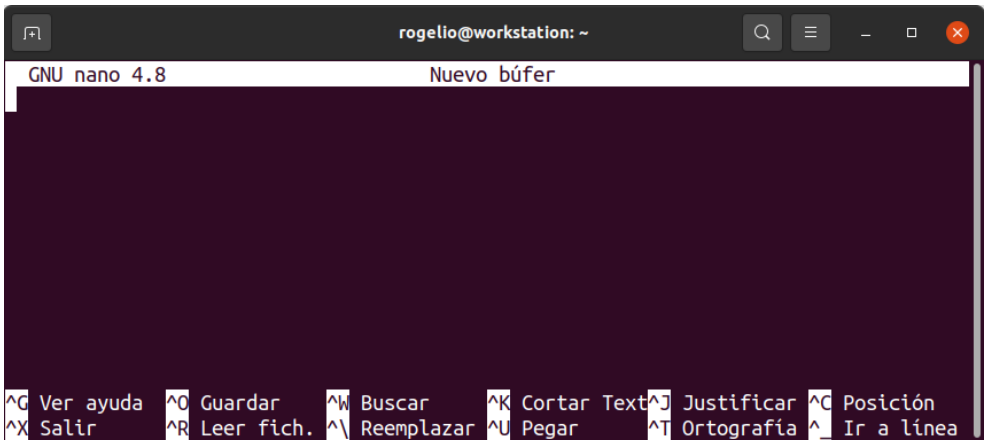

Figura AI.10: Ventana principal del editor de texto **nano**. El título «Nuevo búfer» se refiere a un nuevo archivo; un búfer es el espacio de memoria donde se almacenan datos temporales. En la parte inferior se muestra el menú de funciones básicas. El símbolo ^ se refiere a la tecla Control. Para ejecutar una función básica usted debe mantener presionadas la tecla Ctrl y la tecla indicada para la opción deseada.

de sintaxis mediante colores, numeración de líneas y visualización de líneas demasiado largas en múltiples líneas (*soft wrap*).

Sintaxis:

```
nano [opciones] [[+línea[,columna]] archivos]
```

Si se omite el parámetro **archivos** se crea un nuevo archivo. El parámetro **archivos** puede ser una ruta absoluta o relativa por cada archivo que desee abrir, si uno de estos archivos no existe se crea uno nuevo con el nombre especificado.

Las **opciones** más frecuentes son:

`-l` Muestra los números de línea a la izquierda del texto.

`-c` Muestra constantemente la posición (línea y columna) actual del cursor. En **nano**, la columna se refiere al número de carácter de la línea actual.

`-$` Muestra todo el contenido de cualquier línea, incluso si es más larga que el ancho de la pantalla, divide el contenido en múltiples líneas de pantalla. *Nota:* Debe especificar esta opción en último lugar cuando se use con otras opciones. Ejemplo: **nano** `-lc$`.

Al ejecutar **nano** se muestra el contenido del archivo e información de identificación y ayuda en la pantalla. En la parte superior muestra la versión y el nombre del archivo; si es archivo nuevo mostrará «Nuevo búfer». La Figura AI.10 muestra la ventana principal de **nano**, en la parte inferior muestra dos líneas de ayuda. Estas líneas de ayuda nos indican las funciones básicas que pueden ser ejecutadas oprimiendo la tecla Ctrl (^ se refiere a la tecla Control) o las teclas Función (F1 a F11) o la tecla Alt (M se refiere a la tecla Alt).

Funciones básicas

Las funciones básicas de **nano** pueden ser ejecutadas con atajos de teclado. La Tabla AI.8 muestra el listado de los atajos de teclado y las funciones básicas. Por ejemplo para salir de **nano**, escriba Ctrl + x.

Tabla AI.8: Funciones básicas del editor de texto **nano**.

Tecla	Función
Ctrl + g o F1	Muestra la ayuda
Ctrl + x o F2	Salir sin guardar
Ctrl + o o F3	Guarda el archivo actual
Ctrl + w o F6	Busca una cadena de texto o expresión regular
Ctrl + k o F9	Corta la línea actual
Ctrl + u o F10	Pega la línea cortada
Alt + U	Deshacer la última acción
Alt + E	Rehacer la última acción
Alt + R	Reemplazar texto
Ctrl + _	Ubicar el cursor en una línea en específico

Ejemplo AI.30: Crear un nuevo archivo en nano, insertar datos y guardar el archivo.

```
1 $ nano misdatos.txt
```

El Ejemplo AI.30 permite abrir **nano** para crear un nuevo archivo con el nombre `misdatos.txt`. Para continuar, inserte el texto deseado. En la línea 1 teclee su nombre completo y en la línea 2 el nombre de la universidad en la que estudia o desea estudiar.

En cualquier momento usted puede guardar los cambios presionando Ctrl + o. Al finalizar puede salir de **nano**, presione Ctrl + x.

Ejemplo AI.31: Abrir el archivo misdatos.txt. Copiar y pegar una línea de texto.

```
1 rogelio@workstation:~$ nano misdatos.txt
```

El Ejemplo AI.31 permite abrir el archivo `misdatos.txt` en **nano**. Para continuar este ejemplo, ubique el cursor en la línea que desea copiar. Presione Ctrl + k Ctrl + u para cortar y pegar, esto copiará la línea en memoria y dejará la línea intacta en el archivo. Desplace el cursor, utilizando las teclas de flecha (←,→,↑,↓) a la posición donde insertará la copia; finalmente, para pegar el texto presione Ctrl + u.

Ejemplo AI.32: Abrir el archivo misdatos.txt. Copiar y pegar una cadena.

```
1 rogelio@workstation:~$ nano misdatos.txt
```

En el Ejemplo AI.32 se muestra cómo copiar y pegar una cadena en nano. Para concluir el ejemplo, ubique el cursor al inicio del texto que desea copiar. Presione Ctrl + 6. Utilizando las teclas de flecha marque el texto que desea copiar. Copie el texto presionando Alt + 6. Finalmente, para pegar, desplace el cursor en la posición deseada y presione Ctrl + u.

Ejemplo AI.33: Abrir el archivo misdatos.txt y posicionar el cursor en la línea 10, columna 5.

```
1 rogelio@workstation:~$ nano +2,5 misdatos.txt
```

Es posible indicar a **nano** que, al abrir un archivo, se posicione en una línea y columna. El Ejemplo AI.33 abre un archivo y se posiciona en la línea 2, columna 5.

Opcional. Presione Ctrl + c si desea que **nano**, por única ocasión, le indique la posición (línea y columna) actual del cursor. Presione Alt + c para indicar a **nano** que siempre muestre la posición actual del cursor.

AI.4.7. Permisos estándar

GNU/Linux es un sistema operativo multiusuario. La característica de ser multiusuario se refiere a que GNU/Linux es capaz de ser utilizado por más de un usuario de manera simúltanea. Es decir, es posible que dos o más usuarios utilicen la misma computadora que opera con GNU/Linux.

La característica multiusuario de GNU/Linux está acompañada con criterios de seguridad que determinan los permisos de archivos y directorios. Estos permisos son directrices que establecen quién puede leer, escribir (modificar) o ejecutar un archivo o directorio determinado. Los permisos de archivos y directorios también permiten establecer que el elemento pertenece a un usuario; también establecen que los archivos y directorios sean compartidos entre usuarios y grupos de usuarios.

Existen tres tipos de permisos, se representan por su letra inicial en inglés:

- Permiso de lectura (**r**)
- Permiso de escritura (**w**)
- Permiso de ejecución (**x**)

El comando `ls -l` permite visualizar los permisos para cada archivo o directorio. La Figura AI.6 muestra el resultado obtenido al ejecutar el comando `ls -l`. En la columna marcada en color verde se pueden observar los permisos de seguridad asignados a cada elemento.

En GNU/Linux, los permisos para un archivo o directorio están compuestos de tres bloques:

1. El primer bloque consiste de tres posiciones en este orden: `rwx` que aplican para el *propietario* del elemento.
2. El segundo bloque consiste también de tres posiciones en este orden: `rwx` que aplican para el *grupo* de usuarios especificado. En GNU/Linux, todos los usuarios pertenecen al menos a un grupo.

Figura AI.11: Resultado obtenido al ejecutar el comando `ls -l` con la información detallada de los archivos almacenados en el directorio. La primera columna muestra los permisos asignados a cada archivo o directorio. Los primeros tres bloques marcados en color, especifican los permisos para el *propietario*, el *grupo* y los *otros* usuarios.

3. El tercer bloque consiste de tres posiciones en este orden: `rwx` que aplican para los *otros* usuarios, es decir, los usuarios que no son el propietario ni pertenecen al grupo.

Un ejemplo puede ser visualizado en la Figura AI.11 que muestra un listado detallado de archivos. Los primeros tres bloques marcados en color, especifican los permisos para el *propietario*, el *grupo* y los *otros* usuarios. Por ejemplo, la carpeta `secuenciacion` tiene asignados permisos de `rwx` (lectura, escritura y ejecución) para el usuario *propietario* `rogelio`, de `rwx` (lectura, escritura y ejecución) para el *grupo* `genomica` y de `r-x` (lectura y ejecución) para los *otros* usuarios.

Para la asignación de permisos se permite utilizar notación numérica. La Figura AI.12 muestra todas las combinaciones para los valores de lectura (**r**: reading), escritura (**w**: write) y ejecución (**x**: execute).

Valor en binario			Valor en decimal
r	w	x	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
4	2	1	Valor Posicional

Figura AI.12: Tabla de permisos. Muestra todas las posibles combinaciones para activar los permisos de lectura (`r`: reading), escritura (`w`: writing) y ejecución (`x`: execute) y su respectiva conversión en sistema decimal. En la última fila de la tabla se especifica el valor por posición, cada dígito tiene un valor en sistema decimal: activar la Lectura (**r**) tiene valor de `4`, activar la Escritura (**w**) tiene valor de `2` y activar la Ejecución (**x**) tiene un valor de `1`.

Imagine que usted tuviera, para cada permiso, un interruptor que tiene dos estados: encendido (`1`) o apagado (`0`). Si se desea «encender» (activar), por ejemplo, los tres permisos (**rwx**), obtendría el número: `111` en sistema binario. Al convertir ese número a su correspondiente valor en sistema decimal se obtiene el valor `7`.

Por su posición, cada dígito del número `111` tiene un valor diferente en sistema decimal: activar la Lectura (**r**) tiene valor de 4 ($1 \times 2^2 = 4$) , activar la Escritura (**w**) tiene valor de 2 ($1 \times 2^1 = 2$) y activar la Ejecución (**x**) tiene un valor de 1 ($1 \times 2^0 = 1$); al sumar estos valores se obtiene el 7.

Si deseara asignar el permiso **rwx** al bloque del propietario, grupo y otros, entonces se podría asignar el permiso: `777`.

Comando `chmod`. El comando **`chmod`** (*change mode*) nos permite asignar permisos de un archivo o directorio. La sintaxis es:

```
chmod [quien] [operador] [permiso] archivos
```

- **quien** Indica a quien afecta el permiso que se desea cambiar. Es una combinación de cualquiera de las letras:
  u para el propietario o usuario
  g para el grupo
  o para los otros usuarios
  a para todos los anteriores.

Si no indica el **quien**, el *shell* asigna **a** de manera predeterminada.

- **operador** Indica la operación que se desea hacer con el permiso.
  = para asignar permisos (y omitir los existentes).
  + para añadir un permiso a los ya existentes.
  - para quitar un permiso.

- **permiso** Indica el permiso que se quiere otorgar o quitar. Será una combinación cualquiera de las letras:
  r para lectura
  w para escritura
  x para ejecución

- **archivos** Nombre del archivo o de los archivos cuyos modos de acceso (permisos) se quieren cambiar.

El comando **chmod** (*change mode*) también permite utilizar la notación numérica en su sintaxis. Permite especificar tres números (cada uno dentro del rango del 1 al 7) uno para propietario, grupo y otros respectivamente:

```
1 chmod 777 archivos
```

Las tres instrucciones incluidas en el Ejemplo AI.34 son equivalentes. La tercera instrucción usa una notación más breve. La Figura AI.12 puede ser utilizada como

Ejemplo AI.34: Asignar permisos de lectura, escritura y ejecución al propietario, al grupo y a otros.

```
1 rogelio@workstation:~$ chmod u=rwx,g=rwx,o=rwx misdatos.txt
2 rogelio@workstation:~$ chmod ugo=rwx misdatos.txt
3 rogelio@workstation:~$ chmod 777 misdatos.txt
```

referencia para obtener el número `7` que significa activar los permisos de lectura (`r`), escritura (`w`) y ejecución (`x`).

Ejemplo AI.35: Asignar para asignar los permisos de lectura y escritura solamente al propietario de un archivo.

```
1 rogelio@workstation:~$ chmod u=rw,g=,o= misdatos.txt
2 rogelio@workstation:~$ chmod 600 misdatos.txt
```

En el Ejemplo AI.35, las dos instrucciones son equivalentes. En la línea 2 el número `6`, si se toma como referencia la Figura AI.12, activa los permisos de lectura y escritura. El `0` se refiere a no activar ninguno de los permisos (lectura, escritura y ejecución) del grupo y de otros.

Ejemplo AI.36: Asignar permiso de lectura y ejecución al grupo al que pertenece el archivo.

```
1 rogelio@workstation:~$ chmod g=rx misdatos.txt
2 rogelio@workstation:~$ chmod 050 misdatos.txt
```

Las dos instrucciones del Ejemplo AI.36 son equivalentes. En la línea 2 el número `5`, si se toma como referencia la Figura AI.12, activa los permisos de lectura (`r`) y ejecución (`x`).

En el Ejemplo AI.37, las dos instrucciones son equivalentes. Al usar el operador `+` se mantienen los permisos que actualmente tiene el archivo. En este ejemplo se agrega el permiso de ejecución (`x`) para todos: propietario, grupo y otros.

Ejemplo AI.37: Agregar permiso de ejecución a todos: al propietario, al grupo y a otros.

```
rogelio@workstation:~$ chmod a=+x misdatos.txt
rogelio@workstation:~$ chmod +x misdatos.txt
```

De manera predeterminada, Ubuntu al crear un archivo le asigna permisos de `rw-rw-r--` (`664`). El propietario y el grupo pueden leer y escribir, los otros usuarios pueden leer el archivo.

Ubuntu al crear un directorio, de manera predeterminada, le asigna los permisos `rwxrwxr-x` (`775`). Para acceder a un directorio es necesario el permiso de ejecución (`x`) por este motivo este permiso está activado para todos los bloques. El propietario y el grupo pueden leer, escribir y ejecutar. Los otros tienen permiso de lectura y ejecución. Es importante señalar que estos permisos aplican al directorio pero los elementos (archivos y directorios) contenidos dentro de él tienen sus propios permisos de manera individual.

En la Tabla AI.9 se muestra una lista de permisos y sus significado. Algunos usos típicos son los permisos `600`, `644`, `700` y `755`. No recomienda utilizar el permiso `777` por motivos de seguridad, puesto que cualquier usuario podría modificar o eliminar el archivo o directorio que tenga asignado este permiso.

Tabla AI.9: Permisos comunes para asignar a archivos y directorios.

Permisos	Notación Numérica	Significado
rw-------	600	Solo el *propietario* tiene permisos de lectura y escritura.
rw-r--r--	644	Solo el *propietario* tiene permisos de lectura y escritura; el *grupo* y *otros* tienen permiso solo de lectura.
rwx------	700	Solo el *propietario* tiene permisos de lectura, escritura, y ejecución.
rwxr-xr-x	755	El *propietario* tiene permisos de lectura, escritura, y ejecución; el *grupo* y *otros* tienen permisos solo de lectura y ejecución.

Permisos	Notación Numérica	Significado
`rwx--x--x`	`711`	El *propietario* tiene permisos de lectura, escritura, y ejecución; el *grupo* y *otros* solo tienen permisos de ejecución.
`rw-rw-rw-`	`666`	Todos pueden realizar lectura y escritura en el archivo. ¡Cuidado con estos permisos!.
`rwxrwxrwx`	`777`	Todos pueden realizar lectura, escritura, y ejecución. ¡Estos permisos pueden ser riesgosos!.

AI.4.8. Desplegar contenido. `cat`, `more`, `less`, `wc`, `head` y `tail`

El *shell* de GNU/Linux cuenta con comandos que nos permiten visualizar el contenido de una entrada de datos (desde el teclado o desde uno o más archivos) en la salida estándar. De manera predeterminada la salida estándar es la pantalla de la computadora.

Nota. En lo siguiente, se trabajará con los archivos `contigs.fasta`, `archivo01.txt` y `archivo02.txt` que están disponibles para descarga en: https://bit.ly/repositorio-tesis-maestria-RPA.

Comando `cat`. El comando **`cat`** (*catenate* en inglés) nos permite imprimir y concatenar el contenido de uno o más archivos. Si le indicamos más de un archivo, el comando **`cat`** mostrará el contenido del primer archivo e inmmediatamente después mostrará el contenido del segundo archivo, así sucesivamente con cada uno de los archivos. A esto se le conoce como concatenación.

Sintaxis:

```
cat [archivo1] [archivo2] ...
```

El Ejemplo AI.38 muestra en pantalla el contenido del archivo `contigs.fasta`.

Ejemplo AI.38: Muestra en pantalla el contenido del archivo contigs.fasta.

```
rogelio@workstation:~$ cat contigs.fasta
>contig1
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
...
```

Ejemplo AI.39: Muestra en pantalla el contenido de dos archivos.

```
rogelio@workstation:~$ cat archivo01.txt archivo02.txt
Rogelio Prieto
Facultad de Informática Culiacán
```

El Ejemplo AI.39 muestra en pantalla, secuencialmente y según el orden especificado, el contenido de dos archivos. Es decir, concatena el contenido de ambos archivos y los muestra en pantalla.

Existen otros comandos que nos facilitan la visualización del contenido de archivos al mostrarlo pantalla por pantalla. A continuación revisaremos los comandos **more** y **less**.

Comando more. El comando **more** muestra el contenido de uno o más archivos pantalla por pantalla de manera secuencial.

Sintaxis:

```
more [-número] [archivo1] [archivo2] ...
```

Una vez que aparece el contenido, el comando **more** nos brinda algunas opciones para continuar con la visualización:

Tabla AI.10: Comando `more`. Opciones de navegación.

Atajo de teclado	Descripción
Espacio	Muestra la siguiente pantalla. Si se especifica el parámetro `-número` entonces avanza esa cantidad de líneas.
↵	Muestra la siguiente línea.
q o ZZ	Salir.

Ejemplo AI.40: Muestra pantalla por pantalla el contenido del archivo contigs.fasta.

```
1 rogelio@workstation:~$ more contigs.fasta
```

Ejemplo AI.41: Muestra en pantalla el contenido de un archivo en bloques de 10 líneas.

```
1 rogelio@workstation:~$ more -10 contigs.fasta
```

El Ejemplo AI.40 muestra en pantalla el contenido de un archivo pantalla por pantalla. Al visualizar la primera pantalla, usted puede avanzar a la siguiente pantalla con la tecla Espacio o utilizar alguna otra tecla de navegación incluida en la Tabla AI.10. También es posible mostrar el archivo en bloques de determinado número de líneas. El Ejemplo AI.41 muestra el contenido en pantalla del archivo de `contigs.fasta` en bloques 10 líneas.

Comando `less`. El commando `less` permite visualizar el contenido de un archivo pantalla por pantalla hacia adelante y hacia atrás.

Sintaxis:

```
less [-número] [archivo1] [archivo2] ...
```

El comando `less` es similar al comando `more` pero tiene más opciones de interacción. Una vez que aparece el contenido, el comando `less` nos brinda opciones navegación y

búsqueda que pueden ser activadas utilizando los atajos de teclado que nos muestran las Tablas AI.11, AI.12.

Tabla AI.11: Comando `less`. Opciones de navegación.

Atajo de teclado	Descripción
Espacio o AvPág o Ctrl + F	Muestra la pantalla anterior. Si se especifica el parámetro `-número` entonces retrocede esa cantidad de líneas.
RePág o Ctrl + B	Muestra la pantalla anterior. Si se especifica el parámetro `-número` entonces retrocede esa cantidad de líneas.
↵	Muestra la siguiente línea.
`g`	Ir al inicio del texto.
`G`	Ir al final del texto.
`q` o `ZZ`	Salir de `less`.

Tabla AI.12: Comando `less`. Opciones de búsqueda.

Atajo de teclado	Descripción
`/texto-a-buscar`	Realiza la búsqueda del texto indicado. Ejemplo: `/AATG`.
`n`	Ir a la próxima coincidencia (hacia adelante) de la búsqueda.
`N`	Ir a la previa coincidencia (hacia atrás) de la búsqueda.
`&texto-a-buscar`	Muestra solamente las líneas que contienen el texto.
`&!texto-a-buscar`	Muestra solamente las líneas que no contienen el texto.

Ejemplo AI.42: Muestra en pantalla el contenido de contigs.fasta.

```
1 rogelio@workstation:~$ less contigs.fasta
```

El Ejemplo AI.42 muestra en pantalla el contenido de un archivo pantalla por pantalla. Al visualizar la primera pantalla, usted puede navegar por el contenido del archivo utilizando los atajos de teclado incluidos en la Tabla AI.11. Usted también puede realizar búsquedas utilizando los atajos de teclado de la @ AI.12.

El comando **less** también permite mostrar el archivo en bloques de determinado número de líneas. El Ejemplo AI.43 muestra el contenido en pantalla del archivo de `contigs.fasta` en bloques 10 líneas.

Ejemplo AI.43: Muestra en pantalla el contenido de contigs.fasta de 10 en 10 renglones.

```
rogelio@workstation:~$ less -10 contigs.fasta
```

Comando wc. El comando **wc** cuenta las líneas, palabras y caracteres.

Sintaxis:

```
wc [opciones]  [archivo1] [archivo2] ...
```

Opciones:

`-l` Cuenta las líneas

`-w` Cuenta las palabras

`-m` Cuenta los caracteres

`-c` Cuenta los bytes

Ejemplo AI.44: Contar las líneas, las palabras y los bytes del archivo contigs.fasta.

```
rogelio@workstation:~$ wc contigs.fasta
  99   99 6833 contigs.fasta
```

Es posible ejecutar el comando **wc** proporcionando como parámetro únicamente el nombre del archivo que analizará. En este caso, de manera predeterminada, **wc** muestra como resultado la cantidad de líneas, palabras y bytes del archivo. El Ejemplo AI.44 muestra este tipo de ejecución.

El Ejemplo AI.45 muestra cómo obtener el conteo por separado de líneas, palabras, caracteres y bytes del archivo `contigs.fasta`.

Ejemplo AI.45: Uso del comando wc con los parámetros más frecuentes.

```
rogelio@workstation:~$ wc -l contigs.fasta
99 contigs.fasta
rogelio@workstation:~$ wc -w contigs.fasta
99 contigs.fasta
rogelio@workstation:~$ wc -m contigs.fasta
6833 contigs.fasta
rogelio@workstation:~$ wc -c contigs.fasta
6833 contigs.fasta
```

Comando `head`. El comando **`head`** muestra las primeras líneas de un archivo. De manera predeterminada muestra las primeras 10 (diez) líneas.

Sintaxis:

```
head [opciones] [archivo1] [archivo2] ...
```

Opciones:

`-n [-]número` Muestra las primeras líneas, desde la uno hasta que en total se muestren «**`número`**» líneas.

`-c número` Muestra los primeros «**`número`**» caracteres, desde el primero hasta que en total se muestren **`número`** caracteres.

Ejemplo AI.46: Mostrar las primeras 10 líneas de contigs.fasta.

```
rogelio@workstation:~$ head contigs.fasta
>contig1
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
...
...
```

En el Ejemplo AI.47, las dos instrucciones son similares. La línea 5 muestra una manera abreviada de ejecutar la instrucción.

Ejemplo AI.47: Mostrar las primeras 3 líneas de contigs.fasta.

```
rogelio@workstation:~$ head -n 3 contigs.fasta
>contig1
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
rogelio@workstation:~$ head -3 contigs.fasta
>contig1
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
```

Ejemplo AI.48: Mostrar los primeros 5 caracteres de contigs.fasta.

```
rogelio@workstation:~$ head -c 5 contigs.fasta
>controgelio@workstation:~$
```

El Ejemplo AI.48 muestra cómo obtener desde el carácter uno hasta el carácter 5. En la línea 2, se observa el resultado e inmediatamente después aparece el *prompt*.

Ejemplo AI.49: Comando head. Caso Especial. Mostrar todas el archivo excepto las últimas 2 líneas de contigs.fasta.

```
rogelio@workstation:~$ head -n -2 contigs.fasta
...
ATGCCTGGCTTTGGTGAACAGATGCGCCAGATCAGCCTGCATTTTGTACCAACTGCGATCCTTTCGCGTC
AGGTGGGCGTGATTCGCAAACAGGCGCTGATCCTTAACTTACCCGGTCAGCCGAAGTCTATTAAAGAGAC
GCTGGAAGGTGTGAAGGACGCTGAGGGTAACGTTGTGGTACACGGTATTTTTGCCAGCGTACCGTACTGC
ATTCAGTTGCTGGAAGGGCCATACGTTGAAACGGCACCGGAAGTGGTTGCAGCATTCAGACCGAAGAGTG
```

El Ejemplo AI.49 muestra el uso del comando **head** con una notación especial para obtener «las primeras líneas del archivo, excepto las últimas i líneas de un archivo».

Comando `tail`. El comando **tail** muestra la parte final de un archivo. De manera predeterminada muestra las últimas 10 líneas de un archivo.

Sintaxis:

```
tail [opciones] [archivo1] [archivo2] ...
```

Opciones:

`-n [+]número` Muestra las últimas «**número**» líneas.

`-c número` Muestra los últimos «**número**» caracteres.

Ejemplo AI.50: Mostrar las últimas 10 líneas de contigs.fasta.

```
1 rogelio@workstation:~$ tail contigs.fasta
2 ...
3 GCTGGAAGGTGTGAAGGACGCTGAGGGTAACGTTGTGGTACACGGTATTTTTGCCAGCGTACCGTACTGC
4 ATTCAGTTGCTGGAAGGGCCATACGTTGAAACGGCACCGGAAGTGGTTGCAGCATTCAGACCGAAGAGTG
5 CAAGACGCGACGTTAGCGAATAAAAAAATCCCCCCGAGCGGGGGGATCTCAAAACAATTAGTGGGATTCA
6 CCAATCGGCAGAACGGTGCGACCAAACTGCTCGTTCAGTACTTCACCCATCGCCAGATAG
```

Ejemplo AI.51: Mostrar las últimas 3 líneas de contigs.fasta.

```
1 rogelio@workstation:~$ tail -n 3 contigs.fasta
2 ATTCAGTTGCTGGAAGGGCCATACGTTGAAACGGCACCGGAAGTGGTTGCAGCATTCAGACCGAAGAGTG
3 CAAGACGCGACGTTAGCGAATAAAAAAATCCCCCCGAGCGGGGGGATCTCAAAACAATTAGTGGGATTCA
4 CCAATCGGCAGAACGGTGCGACCAAACTGCTCGTTCAGTACTTCACCCATCGCCAGATAG
5 rogelio@workstation:~$ tail -3 contigs.fasta
6 ATTCAGTTGCTGGAAGGGCCATACGTTGAAACGGCACCGGAAGTGGTTGCAGCATTCAGACCGAAGAGTG
7 CAAGACGCGACGTTAGCGAATAAAAAAATCCCCCCGAGCGGGGGGATCTCAAAACAATTAGTGGGATTCA
8 CCAATCGGCAGAACGGTGCGACCAAACTGCTCGTTCAGTACTTCACCCATCGCCAGATAG
```

En el Ejemplo AI.51, las dos instrucciones son similares. La línea 2 muestra una manera abreviada de ejecutar la instrucción.

Ejemplo AI.52: Mostrar los últimos 5 caracteres de contigs.fasta.

```
1 rogelio@workstation:~$ tail -c 5 contigs.fasta
2 ATAG
```

El Ejemplo AI.52 muestra cómo obtener los últimos 5 caracteres de un archivo. Observe que se muestran 4 nucleótidos y el quinto carácter es el salto de línea (no visible).

Ejemplo AI.53: Comando tail. Caso especial. Mostrar desde un número de línea en específico contigs.fasta.

```
1  rogelio@workstation:~$ tail -n +2 contigs.fasta
2  AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
3  TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
4  TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
5  ..
6  rogelio@workstation:~$ tail +2 contigs.fasta
7  AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
8  TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
9  TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
10 ...
```

En el Ejemplo AI.53, se muestra el contenido del archivo `contigs.fasta` a partir de la línea 2, es decir, se elimina la línea 1. Este ejemplo es muy útil cuando se trabaja con archivos de datos cuya primera línea contiene encabezados o nombres de columna que requieren ser eliminados. En el Ejemplo AI.53, las instrucciones de las líneas 1 y 6 son equivalentes.

AI.4.9. Redireccionamiento de la salida. > y >>

Cuando se procesa un comando, la entrada predeterminada se espera que provenga del teclado y la salida predeterminada es enviada a la pantalla.

En GNU/Linux la entrada predeterminada se le llama `STDIN` y a la salida predeterminada se le llama `STDOUT`. La entrada y la salida predeterminadas pueden ser redirigidas utilizando una notación especial al escribir las instrucciones en el *shell*.

Es posible usar `<` para redirigir la entrada, se utiliza `>` para redirigir la salida del comando e incluso `>>` para redirigir la salida y agregarla a un archivo existente. En sentido figurado, imagine que estos operadores «son una flecha» que apuntan o dirigen el flujo de información.

Operador <

El operador `<` para redirigir la entrada de un archivo hacia un comando.

Sintaxis:

```
comando [opciones] < archivo
```

Ejemplo AI.54: Contar la cantidad de líneas, palabras y caracteres del archivo contigs.fasta.

```
1 rogelio@workstation:~$ wc contigs.fasta
2   5    6 6739 contigs.fasta
3 rogelio@workstation:~$ wc < contigs.fasta
4   5    6 6739
```

En el Ejemplo AI.54, en la línea 1 se ejecuta el comando **wc** que recibe un parámetro. La línea 2 muestra el resultado: hay 5 líneas, 6 palabras y 6739 caracteres en el archivo `contigs.fasta`. En cambio, en la línea 3 se ejecuta el comando **wc** el cual recibe la entrada redirigida. Note que el resultado es similar, un cambio sutil es debido a que se recibe la información redirigida no conoce el nombre del archivo y por tanto, no imprime el nombre del archivo.

Operador `>`

El operador `>` permite redirecccionar la salida a un archivo. Si el archivo no existe, lo crea. Si el archivo existe lo reemplazará.

Sintaxis:

```
comando [opciones] > archivo
```

En el Ejemplo AI.55, la línea 1 y 6 obtienen el contenido de la carpeta `/bin/`. La línea 1 muestra el resultado en pantalla, en cambio la línea 6 almacena el resultado en el archivo `listado.txt`.

En el Ejemplo AI.56 el resultado del comando **head** es almacenado en el archivo `primeras.fasta`.

Ejemplo AI.55: Listar el contenido del directorio /bin/ y almacenar el resultado en el archivo listado.txt.

```
rogelio@workstation:~$ ls /bin/
'['                                  nl
 411toppm                            nm
 aa-enabled                          nm-applet
...
rogelio@workstation:~$ ls /bin/ > listado.txt
```

Ejemplo AI.56: Obtener las primeras 4 líneas del archivo contigs.fasta y almacenar el resultado en el archivo primeros.fasta.

```
rogelio@workstation:~$ head -n 4 contigs.fasta > primeras.fasta
```

En el Ejemplo AI.57 el resultado del comando **tail** es almacenado en el archivo `ultimas.fasta`.

Operador >>

El operador >> nos permite agregar la salida a un archivo; si el archivo no existe lo crea. Si el archivo existe, agrega el contenido al final del archivo.

Sintaxis:

```
comando [opciones] >> archivo
```

El Ejemplo AI.58 muestra el uso del operador >> para obtener las primeras 4 líneas y las últimas 6 líneas del archivo `contigs.fasta`; almacena el resultado de cada instrucción en el archivo `algunos.fasta`. El archivo `algunos.fasta` permanece con el contenido de la última instrucción ejecutada.

AI.4.10. Copiar, mover y renombrar. cp, mv, rm

El sistema operativo GNU/Linux nos permite manipular archivos, es posible copiar, mover o renombrar archivos.

Ejemplo AI.57: Mostrar las últimas 4 líneas caracteres de contigs.fasta.

```
rogelio@workstation:~$ tail -n 4 contigs.fasta > ultimas.fasta
```

Ejemplo AI.58: Uso del operador » para agregar contenido al final del archivo algunos.fasta.

```
rogelio@workstation:~$ head -n 4 contigs.fasta >> algunos.fasta
rogelio@workstation:~$ tail -n 6 contigs.fasta >> algunos.fasta
```

Comando `cp`. El comando `cp` se utiliza para copiar archivos o directorios.

Sintaxis:

```
cp  [-rv] [archivo o directorio de origen] [Destino]
```

Opciones:

`-r` Copia archivos recursivamente.

`-v` Muestra el estado de la copia. Muestra los nombres de los archivos a medida que se copian.

Ejemplo AI.59: Copia un archivo a un directorio.

```
rogelio@workstation:~$ cp contigs.fasta misdatos/secuencias
```

En el Ejemplo AI.59 se copia el archivo `contigs.fasta` al directorio `misdatos/secuencias`. El archivo copiado tendrá el mismo nombre.

El Ejemplo AI.60 ejemplo utiliza el parámetro `-r` para indicar que debe copiarse todo el contenido del directorio de origen, incluyendo sus subdirectorios y sus respectivos contenidos. El parámetro `-v` nos permite que el comando `cp` nos brinde más detalles del proceso a medida que los archivos se copian.

El Ejemplo AI.61 una una expresión regular para copiar todos los archivos que terminen con `.fasta`.

Ejemplo AI.60: Copia un directorio a otro directorio.

```
rogelio@workstation:~$ cp -rv misdatos/secuencias ecolli/2022
```

Ejemplo AI.61: Copiar todos los archivos con extensión .fasta al directorio /home/rogelio/respaldo-datos.

```
rogelio@workstation:~$ cp *.fasta /home/rogelio/respaldo-datos
```

El Ejemplo AI.62 usa la expresión regular `*` para referirse a todos los archivos y directorios del directorio actual y copiarlos al directorio `/home/rogelio/respaldo-datos`.

Comando `mv`. El comando `mv` nos permite mover o renombrar archivos y directorios.

Sintaxis:

```
mv [v] [archivo o directorio] [Destino]
```

Opciones:

`-v` Muestra el estado del proceso.

En el Ejemplo AI.63 se mueve el archivo `contigs.fasta` al directorio `misdatos/secuencias`. El archivo tendrá el mismo nombre, dejará de existir en la ubicación de origen y se almacenará en la ubicación `misdatos/secuencias`.

En el Ejemplo AI.64 se mueve el directorio `misdatos/secuencias` al directorio `ecolli/2022`, es decir `misdatos/secuencias` estará ubicado dentro del directorio `ecolli/2022`.

El Ejemplo AI.65 cambia el nombre de `misdatos`, ubicado en el directorio actual, a `datos2022`. Observe que tanto `misdatos` como `datos2022` tiene la misma ubicación, por este motivo, el comando `mv` realiza la acción de renombrar.

Comando `rm`. El comando `rm` se utiliza para eliminar archivos y directorios.

Ejemplo AI.62: Copiar todo el contenido del directorio actual al directorio /home/rogelio/respaldo-datos.

```
1 rogelio@workstation:~$ cp * /home/rogelio/respaldo-datos
```

Ejemplo AI.63: Moviendo un archivo a un directorio.

```
1 rogelio@workstation:~$ mv contigs.fasta misdatos/secuencias
```

Sintaxis:

```
rm [opciones] [archivo o directorio]
```

Opciones:

-r Elimina archivos recursivamente. Se utiiza para eliminar directorios. El parámetro `r` indica que eliminará todo el contenido del directorio, incluyendo sus subdirectorios y sus respectivos contenidos.

-v Brinda más detalles del proceso, imprime un mensaje por cada archivo eliminado.

-f Forzar la eliminación de un archivo o directorio, sin pedir confirmación ni mostrar mensajes de error.

-i Solicita confirmación antes de eliminar cada archivo.

El Ejemplo AI.66 muestra cómo borrar el archivo `contigs.fasta`.

El Ejemplo AI.67 elimina el contenido del directorio `datos 2022` a la vez que va mostrando detalles de cada archivo, directorio o subdirectorio eliminado.

En Ejemplo AI.68 las dos instrucciones (línea 1 y 2) permitirían eliminar todos los archivos con extensión `.fasta`. En la línea 1 se eliminan todos los archivos sin solicitar confirmación e incluso ignorando posibles mensajes de error. En cambio la instrucción en la línea 2 solicita confirmación antes de eliminar cada archivo.

Ejemplo AI.64: Moviendo un directorio a otro directorio.

```
1 rogelio@workstation:~$ mv misdatos/secuencias ecolli/2022
```

Ejemplo AI.65: Cambiar de nombre a un directorio.

```
1 rogelio@workstation:~$ mv misdatos datos2022
```

AI.4.11. Buscar archivos. `find`

El *shell* de GNU/Linux cuenta con un comando que permite realizar búsquedas de archivos y directorios que cumplan con un determinado criterio de búsqueda.

`find`. El comando **`find`** busca, a partir de la ruta especificada, los archivos y directorios que coincidan con el criterio de búsqueda. Si se omite la `ruta`, entonces el comando **`find`** realiza la búsqueda a partir del directorio actual.

Sintaxis:

```
find [ruta] [opciones]
```

Opciones:

`-name patrón` Realiza una búsqueda por los archivos y directorios cuyo nombre coincida con una cadena o patrón de búsqueda.

`-perm modo` Realiza una búsqueda por el permiso que tiene asignado el archivo o directorio. El modo de acceso se puede especificar con la notación corta utilizando dígitos octales.

`-type tipo` Realiza una búsqueda para un tipo específico de elemento. Se utiliza `f` para archivo y `d` para directorio.

Los comandos del Ejemplo AI.69 se ejecutan teniendo como directorio actual: `/home/rogelio/`, por tanto, en este ejemplo la línea 1 y 4 ejecutan la misma acción. Ambas instrucciones obtienen los mismos resultados.

Ejemplo AI.66: Borrar un archivo.

```
1 rogelio@workstation:~$ rm contigs.fasta
```

Ejemplo AI.67: Borrar un directorio.

```
1 rogelio@workstation:~$ rm -rv datos2022
```

La Ejemplo AI.70 muestra cómo obtener un listado de los archivos y directorios que tengan *exactamente* permiso de lectura y escritura para su usuario propietario y grupo, pero que otros usuarios pueden leer pero no escribir. Las instrucciones de las líneas 1,6,11 y 16 son equivalentes.

En el Ejemplo AI.71 un archivo con permisos `665` o `777` formaría parte del resultado. En este ejemplo se obtiene como resultado el listado de nombres de archivos y directorios que cumplen con el criterio de búsqueda; a diferencia del Ejemplo AI.70, también aparecerán en el resultado archivos y carpetas que tengan los permisos descritos y adicionalmente activado el permiso de ejecución para su usuario propietario, grupo u otros. Observe la sintaxis, se utiliza el carácter `-` antes de especificar el permiso en formato numérico.

El Ejemplo AI.72 realiza una búsqueda de los archivos cuyo nombre inicia con la letra `b`. Se incluye en la búsqueda solamente archivos, se excluyen los directorios. Las líneas 2 y 4 muestran los resultados obtenidos al ejecutar las líneas 1 y 3 respectivamente.

El Ejemplo AI.73 realiza una búsqueda de los directorios cuyo nombre inicia con la letra `D`.

Ejemplo AI.68: Borrar todos los archivos .fasta en la ubicación actual.

```
rogelio@workstation:~$ rm -f *.fasta
rogelio@workstation:~$ rm -i *.fasta
```

Ejemplo AI.69: Obtener un listado de los archivos y directorios cuyo nombre inicie con la letra D.

```
rogelio@workstation:~$ find /home/rogelio -name "D*"
/home/rogelio/Descargas
/home/rogelio/Documentos
rogelio@workstation:~$ find -name "D*"
/home/rogelio/Descargas
/home/rogelio/Documentos
```

Ejemplo AI.70: Obtener un listado de los archivos y directorios que tengan permisos específicos.

```
rogelio@workstation:~$ find /home/rogelio -perm 664
/home/rogelio/.local/share/gnome-settings-daemon/input-sources-converted
/home/rogelio/.local/share/gnome-shell/gnome-overrides-migrated
/home/rogelio/.local/share/gnome-shell/application_state
/home/rogelio/.local/share/session_migration-ubuntu
...
rogelio@workstation:~$ find . -perm 664
/home/rogelio/.local/share/gnome-settings-daemon/input-sources-converted
/home/rogelio/.local/share/gnome-shell/gnome-overrides-migrated
/home/rogelio/.local/share/gnome-shell/application_state
/home/rogelio/.local/share/session_migration-ubuntu
...
rogelio@workstation:~$ find -perm 664
/home/rogelio/.local/share/gnome-settings-daemon/input-sources-converted
/home/rogelio/.local/share/gnome-shell/gnome-overrides-migrated
/home/rogelio/.local/share/gnome-shell/application_state
/home/rogelio/.local/share/session_migration-ubuntu
...
```

Ejemplo AI.71: Obtener un listado de los archivos y directorios que al menos tengan permiso de lectura y escritura para su usuario propietario y grupo, pero que otros usuarios pueden leer pero no escribir.

```
rogelio@workstation:~$ find /home/rogelio -perm -664
/home/rogelio/.local/share/gnome-settings-daemon/input-sources-converted
/home/rogelio/.local/share/icc
/home/rogelio/.local/share/gnome-shell/gnome-overrides-migrated
...
rogelio@workstation:~$ find -perm -664
/home/rogelio/.local/share/gnome-settings-daemon/input-sources-converted
/home/rogelio/.local/share/icc
/home/rogelio/.local/share/gnome-shell/gnome-overrides-migrated
...
```

Ejemplo AI.72: Obtener un listado de los archivos cuyo nombre inicie con la letra b. Excluir en el listado a los directorios.

```
rogelio@workstation:~$ find /home/rogelio -type f -name "b*"
/home/rogelio/.config/gtk-3.0/bookmarks
rogelio@workstation:~$ find -type f -name "b*"
./.config/gtk-3.0/bookmarks
```

Ejemplo AI.73: Obtener un listado de los directorios cuyo nombre inicie con la letra D.

```
rogelio@workstation:~$ find /home/rogelio -type d -name "D*"
/home/rogelio/Descargas
/home/rogelio/Documentos
rogelio@workstation:~$ find -type d -name "D*"
./Descargas
./Documentos
```

Bibliografía

[1] IEEE Working Group 1003.2(POSIX.2). Ieee standard for information technology–portable operating system interfaces (posix(r))–part 2: Shell and utilities. *IEEE Std 1003.2-1992/INT, Dec. 1994 Ed.*, pages 1–64, 1993.

[2] Alfred V Aho, Brian W Kernighan, and Peter J Weinberger. *The AWK programming language*. Pearson, Upper Saddle River, NJ, January 1988.

[3] R B Altman. A curriculum for bioinformatics: the time is ripe. *Bioinformatics*, 14(7):549–550, 08 1998.

[4] George Beekman. *Computación e informática hoy : una mirada a la tecnología del mañana*. Addison Wesley Iberoamericana Pearson Educación, México, 1995.

[5] David R Bentley, Shankar Balasubramanian, Swerdlow, et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, November 2008.

[6] Babraham Bioinformatics. Fastqc a quality control tool for high throughput sequence data, 2010.

[7] Babraham Bioinformatics. Fastqc a quality control tool for high throughput sequence data, 2010.

[8] S. R. Bourne. Unix time-sharing system: the unix shell. *The Bell System Technical Journal*, 57(6):1971–1990, 1978.

[9] T. Brown. *Genomes 4*. Garland Science, paperback edition, 5 2017.

[10] Vince Buffalo. *Bioinformatics Data Skills: Reproducible and Robust Research with Open Source Tools*. O'Reilly Media, paperback edition, 8 2015.

[11] Roger Bumgarner. Overview of DNA microarrays: types, applications, and their future. *Curr Protoc Mol Biol*, Chapter 22:Unit 22.1., January 2013.

[12] Tim Carver, Simon R. Harris, Matthew Berriman, Julian Parkhill, and Jacqueline A. McQuillan. Artemis: an integrated platform for visualization and analysis of high-throughput sequence-based experimental data. *Bioinformatics*, 28(4):464–469, 12 2011.

[13] Center for Food Safety CFSAN and Applied Nutrition. Food safety modernization act (fsma), Feb 2021.

[14] Center for Food Safety CFSAN and Applied Nutrition. Food safety partnership between the u.s. and mexico reports progress, Feb 2021.

[15] Center for Food Safety CFSAN and Applied Nutrition. Fsma rules & guidance for industry, Jan 2022.

[16] Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, 12 2009.

[17] David Coil, Guillaume Jospin, and Aaron E. Darling. A5-miseq: an updated pipeline to assemble microbial genomes from Illumina MiSeq data. *Bioinformatics*, 31(4):587–589, 10 2014.

[18] SADER. Secretaría de Agricultura y Desarrollo Rural. Gobierno Federal de México, editor. *Análisis de la Balanza Comercial Agroalimentaria de México, 2021.* SAGARPA, 2021.

[19] Project Debian. Package: Dash (0.5.11+git20200708+dd9ef66-5) [essential], 2022.

[20] DNA Data Bank of Japan - DDBJ. DNA Data Bank of Japan - DDBJ, November 2022. [En línea; consultado en noviembre del 2022].

[21] Ubuntu Documentation Project. rbash - manpage, Apr 2020.

[22] Ubuntu Documentation Team. Ubuntu documentation, Apr 2020.

[23] Embl. Heidelberg, November 2022. [En línea; consultado en noviembre del 2022].

[24] National Center for Biotechnology Information. File Format Guide, December 2022. [En línea; consultado en noviembre 2022].

[25] National Center for Biotechnology Information. National Center for Biotechnology Information, November 2022. [En línea; consultado en noviembre del 2022].

[26] Inc. Free Software Foundation. Gnu gnulib manual, 2004.

[27] Jeffrey E F Friedl. *Mastering Regular Expressions*. O'Reilly Media, Sebastopol, CA, 3 edition, August 2006.

[28] M. Garrels. *Bash Guide for Beginners*. The Linux Document Project, 2004.

[29] Machtelt Garrels. *Introduction to Linux - A Hands on Guide*. The Linux Document Project, paperback edition, June 2008. The url is: https://tldp.org/guides.html.

[30] GenBank. GenBank and WGS Statistics, November 2022. [En línea; consultado en noviembre del 2022].

[31] Alice Maria Giani, Guido Roberto Gallo, Luca Gianfranceschi, and Giulio Formenti. Long walk to genomics: History and current approaches to genome sequencing and assembly. *Computational and Structural Biotechnology Journal*, 18:9–19, 2020.

[32] Adam Gleave and Christian Steinruecken. Making compression algorithms for unicode text. In *2017 Data Compression Conference (DCC)*, pages 441–441, 2017.

[33] Prometeus GmbH. Top500. 58th annual edition, November 2021.

[34] Project GNU. Gnu bash manual - appendix b, Dec 2020. Appendix B.

[35] LSB Group. Lsb workgroup, the linux foundation, Mar 2015. https://refspecs.linuxfoundation.org/fhs.shtml.

[36] Anuj Kumar Gupta and U.D. Gupta. Chapter 19 - next generation sequencing and its applications. In Ashish S. Verma and Anchal Singh, editors, *Animal Biotechnology*, pages 345–367. Academic Press, San Diego, 2014.

[37] Doug Hyatt, Gwo-Liang Chen, Philip F Locascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, 11:119, March 2010.

[38] IEEE. Iso/iec international standard for information technology–portable operating system interface (posix)–test methods for measuring conformance to posix–part 2:shell and utilities. *ISO/IEC 14515-2_2003 IEEE Std 2003.2-1996*, pages 1–1410, 2004.

[39] Illumina. An introduction to Next-Generation Sequencing Technology, November 2022. [En línea; consultado en noviembre del 2022].

[40] Free Software Foundation Inc. Gnu general public license v2.0, Jun 1991.

[41] Red Hat Inc. Deployment guide red hat enterprise linux 5, Nov 2020.

[42] National Human Genome Research Institute. Secuenciación de adn. en línea, Feb 2023.

[43] Andrew D. Johnson. An extended IUPAC nomenclature code for polymorphic nucleic acids. *Bioinformatics*, 26(10):1386–1389, 03 2010.

[44] Neil C. Jones and Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms (Computational Molecular Biology)*. The MIT Press, hardcover edition, 8 2004.

[45] Harry Katzan. *Operating systems; a pragmatic approach*. Van Nostrand Reinhold Co, New York, 1973.

[46] Michael Kerrisk. Signal numbering for standard signals. online, Oct 2022.

[47] Diana L. Kolbe and Sean R. Eddy. Fast filtering for RNA homology search. *Bioinformatics*, 27(22):3102–3109, 09 2011.

[48] Ian Korf, Mark Yandell, and Joseph Bedell. *BLAST*. O'Reilly Media, Inc., Sebastopol, CA, USA, July 2003.

[49] Karin Lagesen, Peter Hallin, Einar Andreas Rødland, Hans-Henrik Staerfeldt, Torbjørn Rognes, and David W Ussery. RNAmmer: consistent and rapid annotation of ribosomal RNA genes. *Nucleic Acids Res*, 35(9):3100–3108, April 2007.

[50] Large Scale Genomics work stream of the Global Alliance for Genomics & Health (GA4GH). HTS format specifications, November 2022. [Enlínea; consultado en noviembre del 2022].

[51] Dean Laslett and Bjorn Canback. ARAGORN, a program to detect tRNA genes and tmRNA genes in nucleotide sequences. *Nucleic Acids Res*, 32(1):11–16, January 2004.

[52] Ivica Letunic and Peer Bork. Interactive tree of life (iTOL): an online tool for phylogenetic tree display and annotation. *Bioinformatics*, 23(1):127–128, October 2006.

[53] Leon S. Levy. A walk through awk. *SIGPLAN Not.*, 18(12):69–85, dec 1983.

[54] John Lewis. *Java software structures : designing and using data structures*. Pearson Education Limited, Harlow, Essex, 2014.

[55] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, June 2009.

[56] Linux Foundation Linux Foundation Annual Report 2021. *New Horizons for Open Source,*. Linux Foundation, Dec 2021.

[57] K M Majidha Fathima and N. Santhiyakumari. A survey on evolution of cloud technology and virtualization. In *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pages 428–433, 2021. https://ieeexplore.ieee.org/document/9388639.

[58] Diego Mariano, Mívian Ferreira, Bruno L. Sousa, Lucianna H. Santos, and Raquel C. de Melo-Minardi. A brief history of bioinformatics told by data visualization. In *Advances in Bioinformatics and Computational Biology: 13th Brazilian Symposium on Bioinformatics, BSB 2020, São Paulo, Brazil, November 23–27, 2020, Proceedings*, page 235–246, Berlin, Heidelberg, 2020. Springer-Verlag.

[59] Fernando Martínez-Plumed, Lidia Contreras-Ochando, Cèsar Ferri, José Hernández-Orallo, Meelis Kull, Nicolas Lachiche, María José Ramírez-Quintana, and Peter Flach. Crisp-dm twenty years later: From data mining processes to da-

ta science trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 33(8):3048–3061, 2021.

[60] Richard McDougall and Jennifer Anderson. Virtualization performance: Perspectives and challenges ahead. *SIGOPS Oper. Syst. Rev.*, 44(4):40–56, dec 2010.

[61] Dinesh Mehta. *Handbook of data structures and applications*. Chapman & Hall/CRC, Boca Raton, Fla, 2005.

[62] Barry Merriman, Ion Torrent R&D Team, and Jonathan M. Rothberg. Progress in ion torrent semiconductor chip based sequencing. *ELECTROPHORESIS*, 33(23):3397–3417, dec 2012.

[63] Jason R Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, March 2010.

[64] Binh Nguyen. *Linux Filesystem Hierarchy*. The Linux Document Project, July 2004. https://tldp.org/guides.html.

[65] Binh Nguyen. *Linux Dictionary*, April 2005. The url is: https://tldp.org/guides.html.

[66] National Human Genome Research Institute NHGRI. Genómica comparada, 2019. https://www.genome.gov/es/about-genomics/fact-sheets/Genomica-comparada.

[67] National Human Genome Research Institute NHGRI. Bioinformática, 2023. https://www.genome.gov/es/genetics-glossary/Bioinformatica.

[68] OEC Observatory of Economic Complexity. Mexico (mex) exports, imports, and trade partners, 2020.

[69] Brian D. Ondov, Todd J. Treangen, and Adam M. Phillippy. Harvesttools, 2014.

[70] Corporation Oracle. Oracle® vm virtualbox® user manual, 2022. https://www.virtualbox.org/manual/.

[71] World Trade Organization, editor. *World Trade Statistical Review 2021*. World Trade Organization, paperback edition, 12 2021. Original URL: https://www.wto.org/spanish/res_s/statis_s/wts2021_s/wts21_toc_s.htm.

[72] W R Pearson and D J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[73] Thomas Nordahl Petersen, Søren Brunak, Gunnar von Heijne, and Henrik Nielsen. SignalP 4.0: discriminating signal peptides from transmembrane regions. *Nat Methods*, 8(10):785–786, September 2011.

[74] Jonathan Pevsner. *Bioinformatics and Functional Genomics*. John Wiley I& Sons, Nashville, TN, 3 edition, October 2015.

[75] Mihai Pop. Genome assembly reborn: recent computational challenges. *Brief Bioinform*, 10(4):354–366, May 2009.

[76] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, jul 1974.

[77] Anthony Rhoads and Kin Fai Au. PacBio sequencing and its applications. *Genomics, Proteomics & Bioinformatics*, 13(5):278–289, oct 2015.

[78] James T. Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S. Lander, Gad Getz, and Jill P. Mesirov. Integrative genomics viewer. *Nature Biotechnology*, 29(1):24–26, Jan 2011.

[79] Rusty Russell, Daniel Quinlan, and Christopher Yeoh. Filesystem hierarchy standard, Jan 2004.

[80] Samtools organisation for next-generation sequencing developers. Samtools organisation and repositories, September 2018. [En línea; consultado en septiembre del 2022].

[81] Torsten Seemann. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*, 30(14):2068–2069, 03 2014.

[82] Torsten Seemann. Prokka: Rapid prokaryotic genome annotation. *Bioinformatics*, 30(14):2068–2069, 2014.

[83] William Shotts. *The Linux command line : a complete introduction.* No Starch Press, San Francisco, 2012.

[84] Ellen Siever, Stephen Figgins, Robert Love, and Arnold Robbins. *Linux in a Nutshell: A Desktop Quick Reference.* O'Reilly Media, paperback edition, oct 2009.

[85] Barton E. Slatko, Jan Kieleczawa, Jingyue Ju, Andrew F. Gardner, Cynthia L. Hendrickson, and Frederick M. Ausubel. "first generation" automated dna sequencing technology. *Current Protocols in Molecular Biology*, 96(1):7.2.1–7.2.28, 2011.

[86] Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 01 2014. 10.1093/bioinformatics/btu033.

[87] LLC SUSE LLC. Administration guide - suse linux enterprise server 15 sp3, Apr 2022. https://documentation.suse.com/sles/15-SP3/pdf/book-administration_color_en.pdf.

[88] Andrew Tanenbaum. *Sistemas operativos modernos.* Prentice Hall Hispanoamericana, México, 1993.

[89] Andrew Tanenbaum. *Modern operating systems*. Pearson, Boston, 2015.

[90] Debian Documentation Team. Debian policy - fhs compliance, Aug 2021. https://www.debian.org/doc/debian-policy/ch-opersys.html.

[91] Helga Thorvaldsdóttir, James T. Robinson, and Jill P. Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in Bioinformatics*, 14(2):178–192, 04 2012.

[92] Ebony Torrington. Bioinformaticians: the hidden heroes of the COVID-19 pandemic. *Biotechniques*, 72(5):171–174, April 2022.

[93] UniProt. UniProt, November 2022. [En línea; consultado en noviembre del 2022] url: https://www.uniprot.org/uniprotkb/.

[94] Dejana T. Vojnak, Borislav S. Đorđević, Valentina V. Timčenko, and Svetlana M. Štrbac. Performance comparison of the type-2 hypervisor virtualbox and vmware workstation. In *2019 27th Telecommunications Forum (TELFOR)*, pages 1–4, 2019.

[95] World Wide Web Consortium (W3C). Html encoding (character sets), 2022.

[96] World Health Organization WHO. Inocuidad de los alimentos, Apr 2020.

[97] Michał Wojcieszek, Magdalena Pawełkowicz, Robert Nowak, and Zbigniew Przybecki. Genomes correction and assembling: present methods and tools. In Ryszard S. Romaniuk, editor, *SPIE Proceedings*. SPIE, nov 2014.

[98] Borislav Đorđevic, Miloš Marjanović, and Valentina Timčenko. Performance comparison of native host and hyper-based kvm virtualization. In *2020 28th Telecommunications Forum (TELFOR)*, pages 1–4, 2020.