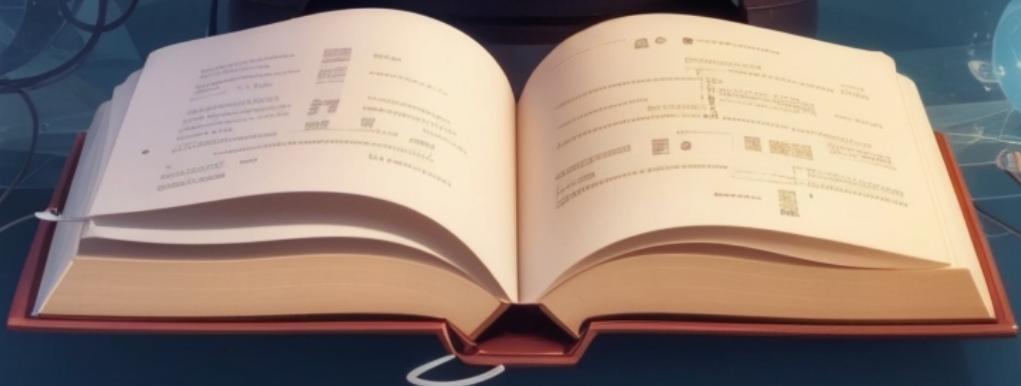




# El Grimorio

Ecosistema y descripción de herramientas de  
ClinSysBioLab (Clinic System Biology Laboratory)

Pedro Seoane, Elena Rojano, Jim Perkins, José Córdoba, Álvaro Esteban, Federico García y Jesús Pérez





# Contents

<b>I</b>	<b>Introduction</b>	<b>9</b>
<b>0.1</b>	<b>Introducción al curso</b>	<b>11</b>
0.1.1	Introducción a la computación .....	11
0.1.2	Sistema operativo .....	13
0.1.3	Introducción a la línea de comandos .....	16
<b>0.2</b>	<b>Configuración del entorno de trabajo</b>	<b>19</b>
0.2.1	Crear cuenta en Picasso .....	19
0.2.2	Cómo conectarse a <i>Picasso</i> .....	20
0.2.3	Facilitar la conexión a <i>Picasso</i> y que se mantenga activa .....	21
0.2.4	Acceder a los archivos de <i>Picasso</i> .....	22
0.2.5	macOS .....	23
0.2.6	Windows .....	24
<b>0.3</b>	<b>Primeros pasos en la línea de comandos</b>	<b>28</b>
<b>0.4</b>	<b>Manipulación de archivos de texto</b>	<b>30</b>
0.4.1	Comandos touch, cat, echo y printf .....	31
0.4.2	Comandos grep, cut, sed y tr .....	32
0.4.3	Comandos head, tail y wc .....	34
0.4.4	Comandos sort, uniq y du .....	35
0.4.5	Otros comandos de interés: tree, ln, awk, gzip, find .....	35
0.4.6	Edición de archivos con vim .....	37
<b>0.5</b>	<b>Herramientas e interacciones</b>	<b>39</b>

<b>1</b>	<b>Picasso y su sistema de colas</b>	<b>41</b>
<b>1.1</b>	<b>Carga de software</b>	<b>41</b>
1.1.1	Software de Picasso .....	41
1.1.2	Cargar software de soft_bio_267 .....	41
<b>1.2</b>	<b>Sistema de colas</b>	<b>41</b>
1.2.1	Enviar trabajos .....	41
1.2.2	Controlar la ejecución .....	41
<b>2</b>	<b>Instalación de herramientas</b>	<b>43</b>
2.0.1	Instalación en Picasso - Grupo BIO267 .....	43
2.0.2	Instalación y configuración del flujo en otros sistemas .....	43
<b>3</b>	<b>GitHub</b>	<b>57</b>
<b>3.1</b>	<b>Gestión de versiones: conceptos fundamentales</b>	<b>57</b>
3.1.1	¿Para qué sirve un gestor de versiones? .....	57
3.1.2	Repositorio local y remoto .....	57
3.1.3	Ramas y bifurcaciones .....	58
<b>3.2</b>	<b>Creación de tokens para verificación</b>	<b>58</b>
<b>3.3</b>	<b>Claves SSH</b>	<b>61</b>
3.3.1	Crear la clave .....	61
3.3.2	Añadir la clave a tu cuenta .....	61
3.3.3	Clave de firma .....	62
3.3.4	Añadir las claves a tu configuración de git .....	62
3.3.5	Crear y enlazar el repositorio remoto .....	62
<b>3.4</b>	<b>Comandos de git</b>	<b>62</b>
<b>II</b>	<b>Sección de usuario</b>	<b>67</b>
<b>4</b>	<b>AutoFlow</b>	<b>69</b>
<b>4.1</b>	<b>Descripción</b>	<b>69</b>
<b>4.2</b>	<b>Forma de uso</b>	<b>69</b>
4.2.1	Crear una plantilla .....	69
<b>4.3</b>	<b>Ejecución por linea de comandos</b>	<b>74</b>
<b>4.4</b>	<b>Aspectos avanzados</b>	<b>76</b>
<b>4.5</b>	<b>Buenas prácticas</b>	<b>77</b>
<b>4.6</b>	<b>Flow logger</b>	<b>78</b>
<b>5</b>	<b>cmdtabs</b>	<b>81</b>
<b>5.1</b>	<b>Descripción</b>	<b>81</b>

<b>5.2</b>	<b>Características y parámetros comunes de los scripts de py_cmtabs</b>	<b>81</b>
5.2.1	aggregate_columns_data .....	82
5.2.2	desaggregate_column_data .....	83
5.2.3	transpose_table .....	83
5.2.4	create_metric_table .....	84
5.2.5	merge_tabular .....	85
5.2.6	tag_table .....	85
5.2.7	intersect_columns .....	85
5.2.8	table_linker .....	86
5.2.9	standard_name_replacer .....	87
5.2.10	column_filter .....	87
5.2.11	excel_to_tabular .....	88
<b>6</b>	<b>report_html</b> .....	<b>89</b>
<b>6.1</b>	<b>Descripción</b>	<b>89</b>
<b>6.2</b>	<b>Script</b>	<b>90</b>
6.2.1	Métodos del report_html .....	91
<b>7</b>	<b>semtools</b> .....	<b>95</b>
<b>7.1</b>	<b>Descripción</b>	<b>95</b>
<b>7.2</b>	<b>semtools.py</b>	<b>95</b>
7.2.1	Ejemplos .....	98
<b>8</b>	<b>NetAnalyzer</b> .....	<b>101</b>
8.0.1	Instalación desde GitHub .....	101
<b>8.1</b>	<b>netanalyzer</b>	<b>101</b>
8.1.1	Inputs .....	102
8.1.2	Proyecciones .....	104
8.1.3	Análisis de cluster .....	105
8.1.4	Visualización de redes .....	106
8.1.5	Embeddings: Kernels y node2vec .....	107
<b>8.2</b>	<b>ranker</b>	<b>108</b>
<b>8.3</b>	<b>integrate_kernels</b>	<b>110</b>
<b>8.4</b>	<b>text2binary_matrix</b>	<b>110</b>
<b>8.5</b>	<b>randomize_clustering</b>	<b>111</b>
<b>8.6</b>	<b>randomize_network</b>	<b>111</b>
<b>8.7</b>	<b>net_explorer</b>	<b>111</b>
<b>9</b>	<b>cdlib_clusterize.py</b> .....	<b>113</b>
<b>9.1</b>	<b>Descripción</b>	<b>113</b>

<b>9.2 Funcionamiento</b>	<b>113</b>
9.2.1 Argumentos del script . . . . .	113
<b>9.3 Métodos de clustering</b>	<b>114</b>
9.3.1 Métodos para comunidades no solapantes . . . . .	114
9.3.2 Métodos para comunidades solapantes . . . . .	115
<b>10 PETS: Patient Exploration Tools Suite</b> . . . . .	<b>117</b>
<b>10.1 Cohort Analyzer</b>	<b>117</b>
10.1.1 Descripción . . . . .	117
10.1.2 <i>Script</i> . . . . .	117
<b>11 ExpHunterSuite</b> . . . . .	<b>121</b>
<b>11.1 Análisis de detección y expresión de miRNAs (smallRNAseq)</b>	<b>122</b>
11.1.1 Detección de miRNA . . . . .	122
11.1.2 Análisis de expresión diferencial de miRNA . . . . .	123
11.1.3 Subir cambios al repositorio ExpHunterSuite . . . . .	124
<b>12 DEG Workflow</b> . . . . .	<b>125</b>
<b>12.1 Análisis de muestras de RNA-seq con DEG Workflow</b>	<b>125</b>
12.1.1 Fase 0: Configuración y preparación de DEG Workflow . . . . .	127
12.1.2 Descarga e indexado de las referencias . . . . .	137
12.1.3 Fase 2: Pre-procesamiento y mapeo . . . . .	138
12.1.4 Fase 3: ExpHunter Suite y comprobación de las muestras . . . . .	140
12.1.5 Fase 4: Enriquecimiento funcional . . . . .	141
12.1.6 Fase 5: Preparación de paquetes . . . . .	141
<b>12.2 Funcionalidades externas al DEG_workflow</b>	<b>142</b>
12.2.1 Preparación de paquetes de trabajo para su entrega . . . . .	142
<b>12.3 Ejemplos de análisis</b>	<b>144</b>
12.3.1 Análisis de datos RNA-Seq en Picasso a partir del SRA . . . . .	144
12.3.2 Análisis de datos RNA-Seq en UPO . . . . .	148
<b>13 coRmiT workflow</b> . . . . .	<b>151</b>
<b>13.1 targets_templates.af</b>	<b>151</b>
13.1.1 Nodo launch_target_analysis . . . . .	151
13.1.2 Nodo functional_analysis . . . . .	152
13.1.3 Preparación de paquetes . . . . .	152
13.1.4 Flujo de trabajo . . . . .	152
<b>14 Utilidades externas</b> . . . . .	<b>155</b>
<b>14.1 Obsidian</b>	<b>155</b>

---

<b>14.2 Descarga de datos</b>	<b>155</b>
14.2.1 gdrivedl . . . . .	155
14.2.2 FTP . . . . .	156
14.2.3 gofile.io . . . . .	157
<b>III Sección de desarrollo</b>	<b>159</b>
<b>15 Desarrollo de una gema</b> . . . . .	<b>163</b>
15.1 Desarrollo de dos o más gemas a la vez	164
<b>16 report_html</b> . . . . .	<b>167</b>
<b>17 Semtools: Documentación para desarrollo.</b> . . . . .	<b>169</b>
17.1 Introducción	169
17.2 Comienzo del análisis: Pinceladas en la command line	171
17.3 Siguiente parada: La clase Ontology	174
17.3.1 Lectura de los archivos OBO . . . . .	178
17.3.2 Operaciones sobre perfiles y términos . . . . .	183
17.4 Apéndice	199
<b>18 Desarrollo en R</b> . . . . .	<b>203</b>
18.1 Descripción	203
18.2 Preparar el espacio de trabajo	203
18.3 Guía de estilo	204
18.4 Estructura general del paquete	204
18.4.1 Hacer el paquete ejecutable dentro de la CMD . . . . .	204
18.4.2 Main . . . . .	205
18.4.3 write_report . . . . .	205
18.4.4 report (Rmd) . . . . .	205
18.4.5 Documentación del paquete (man pages) . . . . .	206
18.5 Construir y comprobar el paquete	206
18.6 Actualizar Readme	207
18.7 Resumen	207
18.8 Crear un paquete desde cero	208
18.8.1 DESCRIPTION y LICENSE . . . . .	208
<b>19 Chuletario</b> . . . . .	<b>211</b>
19.0.1 1 muestra != 1 archivo . . . . .	211
19.0.2 Análisis funcional con archivos proporcionados por el grupo colaborador . . . . .	211



**Part I**

**Introduction**



## 0.1 Introducción al curso

Con este curso se pretende dar las competencias básicas para poder solucionar los problemas que se encuentran a diario en el análisis de datos en bioinformática. Al finalizar este curso, el/la lector/a adquirirá la capacidad de manipular grandes volúmenes de datos, así como extraer la información útil para el estudio científico en el que esté interesado/a. Por ello, se empezará por una amplia sección técnica, en la cual se enseñarán los conceptos y herramientas básicas para la manipulación de información genérica. Más adelante, haciendo uso de esta base conceptual general, se describirán herramientas y protocolos usados en el estudio de problemas biológicos.

### WARNING: Normas del curso:

- No pases nunca al siguiente bloque sin haber comprendido todos los conceptos y completado todos los ejercicios.
- Utiliza la menor ayuda posible. Así aprenderás a ser autodidacta y no coger malas costumbres.
- Cuando escribes texto, sea donde sea, ten en cuenta que no son lo mismo mayúsculas y minúsculas, y que los espacios pueden significar algo.
- Escribe a mano todos los comandos incluidos en el documento PDF, si copias y pegas pueden no funcionar.
- Presta mucha atención a los apartados de Buenas Prácticas.

### 0.1.1 Introducción a la computación

En nuestra sociedad actual, no hay momento del día que no dependamos de algún tipo de aparato con capacidad de computación. Gran parte de nuestro trabajo lo realizamos o gestionamos con un ordenador personal. Para entablar comunicación con nuestras personas cercanas utilizamos lo que se llama hoy en día *smart-phone* o teléfonos inteligentes. También, todos los automóviles actuales poseen un ordenador que controla las condiciones del coche, proporciona toda la información relevante al conductor e incluso lo asiste (con las proyecciones actuales, probablemente lo sustituya con el paso del tiempo). A pesar de este contacto tan cercano e indispensable con la tecnología, poco se sabe de su funcionamiento. No somos conscientes de que, en común, todos estos dispositivos tienen un procesador o máquina para hacer operaciones sobre la información que se les proporcione. De hecho, la noción más avanzada que se suele tener es que se pueden ampliar las capacidades de un dispositivo (ordenador, tablet, móvil, etc) descargando o instalando un programa/aplicación en el dispositivo. Esta noción va a ser empleada como punto de referencia para acercarnos al funcionamiento de estos dispositivos.

La instalación de un programa u aplicación en nuestro dispositivo le permite realizar nuevas funciones. Esto se debe a que en informática existen dos conceptos fundamentales: *hardware* y *software*. El *hardware* es la parte física del dispositivo que realiza las operaciones sobre la información, mientras que el *software* indica qué operaciones y en qué orden deben realizarse con el fin de tratar esta información. Por lo tanto, tenemos una parte fija (*hardware*) y una variable (*software*) en nuestro dispositivo. De esta manera, un mismo dispositivo en distintos usuarios puede configurarse dependiendo de las necesidades de cada uno de ellos. Esto evita la tarea de construir y diseñar dispositivos a medida para cada usuario, lo que sería muy caro y tedioso. Por lo tanto, al trabajar con un dispositivo (por ejemplo un ordenador) en lo único que podemos influir de forma

significativa es en el *software*. De hecho, cuando se compra cualquier dispositivo, por norma general posee un *software* pre-configurado, que es lo que se denomina como Sistema Operativo o SO. El SO permite al *hardware* funcionar de manera coordinada e interactuar con el usuario.

Antes de seguir profundizando en el concepto *software* vamos a ver cuáles son los principales componentes esenciales que forman parte del *hardware* de un dispositivo:

- **Procesador (CPU, Central Processing Unit):** Circuito eléctrico que ejecuta las instrucciones de un programa informático mediante la realización de operaciones aritméticas básicas, lógicas y de entrada/salida del sistema. Un ordenador puede tener más de una CPU y esto se conoce como multiprocesador. Todas las CPU modernas son microprocesadores que contienen un único circuito integrado (chip). Algunos circuitos integrados pueden contener varias CPU en un solo chip; estos son denominados procesadores multinúcleo.

En caso de que el circuito integrado contenga más componentes del dispositivo que la CPU, estaremos ante lo que se denomina un sistema en un chip (SoC). La CPU se descompone a su vez en la **unidad aritmético lógica (ALU)**, que realiza operaciones aritméticas y lógicas, y la **unidad de control (CU)**, que extrae instrucciones de la memoria, las decodifica y las ejecuta, diciendo a la ALU qué operaciones ha de realizar.

- **Memoria principal (RAM, Random Access Memory):** la CPU, para trabajar, necesita que la alimenten con información e instrucciones, pero su arquitectura no reserva espacio para guardar información sino para procesarla. Gracias a ello, la CPU es uno de los componentes más rápidos del ordenador y necesita que el componente que le proporcione la información e instrucciones sea lo más rápido posible con el fin de aprovechar sus capacidades al máximo. Para ello, la memoria RAM es la encargada de almacenar la información e instrucciones que recibe la CPU para trabajar y los resultados finales de las operaciones de la misma.

Cuando el dispositivo se apaga, la memoria pierde toda la información almacenada (tecnología volátil). Este es el precio a pagar por conseguir velocidad a la hora de alimentar la CPU. Es lógico pensar que es mucho más lento escribir estructuras permanentes en un circuito eléctrico que mantener circulando pequeñas corrientes eléctricas representando la información.

- **Almacenamiento secundario (disco duro):** Si la memoria principal o RAM no puede mantener registro de la información, de ello se encarga el disco duro. En comparación con la memoria RAM, este componente es un orden de magnitud más lento que la memoria RAM pero permite almacenar cuantiosos volúmenes de datos que no se pierden con el apagado del dispositivo (tecnología no-volátil). Por lo tanto, se utiliza para almacenar todo el *software* indispensable para controlar el dispositivo y la información que genere el usuario.

Debido a las piezas que forman parte de su estructura (las cuales son diferentes dependiendo del tipo de disco duro, HDD o SSD) y al funcionamiento en fases para la transmisión de la información, la velocidad de los discos duros es inferior a la de otros componentes como la memoria RAM. Además, los discos duros no ejecutan directamente los programas/aplicaciones almacenados en ellos, sino que siguen el siguiente esquema: (1) leer la información a procesar, (2) enviarla a la memoria RAM, (3) ejecutar las instrucciones necesarias para la CPU y (4) escritura de los resultados en el disco duro. Todos los pasos intermedios son ignorados al no presentar interés para el usuario. Esto ahorra una gran cantidad de tiempo al evitar la escritura de datos innecesarios en este componente. Por lo tanto, el disco duro sólo se usa para guardar información de forma definitiva, no temporal.

Ahora que conocemos los componentes necesarios para que un computador pueda trabajar, podemos retomar el concepto de *software* y ver como está estructurado.

### 0.1.2 Sistema operativo

El **sistema operativo (SO)** es conjunto de instrucciones encargado de la interacción entre el usuario, las aplicaciones y el *hardware*. Controla las funciones de entrada y salida, la manipulación de los datos introducidos en el ordenador, el almacenamiento de la RAM y su procesamiento en la CPU. Todos los ordenadores, portátiles, tablets, móviles y servidores necesitan un sistema operativo para realizar sus funciones.

Actualmente, los SO más extendidos en ordenadores son:

- Windows, perteneciente a la empresa Microsoft®.
- OSX, perteneciente a la empresa Apple, Inc.
- Linux, perteneciente a UNIX ®. El núcleo central es código abierto y existen varias versiones según las empresas y organizaciones sin ánimo de lucro que adoptan este núcleo para crear su SO a medida.

#### Funciones del sistema operativo:

El SO de un dispositivo se encarga de varias tareas, sin que el usuario tenga que ser consciente de ello. A continuación se detallan estas tareas.

#### Gestión de procesos:

Un proceso es un conjunto de instrucciones que corresponden a un programa y que son ejecutadas por la CPU. En un programa se pueden ejecutar uno o varios procesos diferentes. La ejecución de un programa necesita recursos del sistema como tiempo de CPU, memoria, archivos y dispositivos de entrada y salida para poder leer y escribir datos. El sistema operativo es el responsable de asignar recursos a los procesos, crearlos y destruirlos, pararlos y reanudarlos y hacer que se comuniquen y sincronicen entre sí.

#### Gestión de la memoria principal:

La memoria principal se optimiza para asignar espacio a los diferentes programas a ejecutar, por lo que se comparte entre los distintos procesos. El espacio de memoria asignado se protege para que un programa no interfiera con otro. El SO es el responsable de gestionar la memoria principal conociendo qué espacios de la misma están siendo utilizados y por qué procesos. En cada momento, el SO decide qué procesos se cargarán en el espacio disponible de la memoria. En esta operación, se asigna y reclama el espacio de memoria para que la ejecución del proceso sea exitosa.

#### Gestión del almacenamiento secundario:

El sistema de almacenamiento secundario es otro de los componentes de un SO. Es un espacio reservado en los discos con el objetivo de almacenar los programas que no necesitan estar en la memoria principal. Permite el intercambio de los datos de programas desde/hacia la memoria principal. El sistema operativo se encarga de planificar los discos, gestionar el espacio libre, asignar el almacenamiento y verificar que los datos se guarden en orden.

#### Sistema de entrada y salida:

El sistema de entrada/salida representa el intercambio de información entre el procesador y los dispositivos periféricos (teclado, ratón, pantalla, impresora y otros) a través del SO. Los dispositivos periféricos solicitan recursos del sistema e introducen/sacan información del dispositivo en función de las operaciones que se estén realizando en el momento.

### Sistema de archivos:

Los archivos representan un conjunto de información almacenada en los discos de un ordenador. Dicha información se almacena de forma relacionada y organizada. Estos archivos almacenan tanto los programas como los datos guardados en el dispositivo. El SO es responsable de construir y eliminar archivos y directorios, manipularlos, establecer la localización física de los archivos en las unidades de almacenamiento y realizar copias de seguridad de archivos esenciales.

### Sistema de protección:

En un SO ocurre lo siguiente:

- Varios **usuarios** pueden ejecutar simultáneamente sus programas.
- Varios **procesos** se pueden ejecutar simultáneamente.
- Varios **programas** se pueden ejecutar al mismo tiempo.
- Varios procesos se pueden **intercalar** para su ejecución simulando una ejecución simultánea.

Normalmente, los SO utilizan métodos de protección de datos. Esto sucede por ejemplo para que un programa no pueda usar o cambiar los datos de otro usuario. Con el sistema de protección se controla el acceso de los programas o el de los usuarios a los recursos del sistema. El SO se encarga de distinguir entre uso autorizado y no autorizado, especificar los controles de seguridad a realizar y forzar el uso de los mecanismos de protección para evitar cualquier corrupción de datos u accesos no autorizados. Los tipos de permisos son:

- **r** read (lectura)
- **w** write (escritura)
- **x** execution (ejecución)

Los permisos vienen dados en bloques de tres. El primer bloque se corresponde a los permisos del usuario, el segundo a los permisos del grupo de usuarios y el tercero al resto de usuarios. Para comprobar los permisos de un archivo se pueden consultar a través de la línea de comandos.

### Sobre los comandos:

Por ahora no tienes que ejecutar ningún comando de la línea de comandos. Los que se describen a continuación son para informar sobre cómo obtener información sobre los permisos de archivos, pero la línea de comandos se explicará en la sección 0.1.3

### SO basados en UNIX:

```
>ls -ltr
```

### SO basados en Windows:

```
>dir
```

Imaginemos que el comando devuelve la siguiente salida:

1	-rwxr-xr-x	2	usuario	sysbiolab	8980	jul 18 22:39	a.txt
2	drwxr-xr-x	2	usuario	sysbiolab	4096	nov 26 11:58	FOD
3	drwx-----	49	usuario	sysbiolab	4096	dic 2 10:23	Dropbox
4	drwxrwxr-x	4	usuario	sysbiolab	4096	oct 10 19:21	VMs

La primera columna se corresponde con los permisos que tiene el archivo. Si la primera letra es una «d» se trata de un directorio (por ejemplo en el caso **drwxr-xr-x**); en caso de ser una «l» sería un enlace a ese directorio/archivo y «b» un archivo ejecutable (binario). Los restantes tres grupos de tres letras se corresponden con los permisos de usuario, grupo y resto descritos anteriormente.

- **-rwxr-xr-x** se corresponde a un archivo que tiene permisos de lectura (r), escritura (w) y ejecución (x) por parte del usuario, y de lectura y ejecución (pero no de escritura) por parte del grupo/otros usuarios (r-xr-x).
- **drwxr-xr-x** tiene los mismos permisos que en el caso anterior pero en lugar de ser un archivo se trata de un directorio.
- **drwx**— es un directorio con permisos de escritura, lectura y ejecución exclusivos del usuario.

La segunda columna (numérica) se refiere al número de **enlaces** al archivo. Los enlaces se explicarán mas adelante en el texto. La tercera y cuarta columna se corresponden con el nombre de usuario y grupo, correspondientemente. La quinta es el tamaño del archivo, la sexta y séptima son las fechas y hora de la última modificación, y la última columna es el nombre del archivo/directorio.

Para cambiar los permisos (por ejemplo, cuando queramos ejecutar un programa que hayamos creado y que sólo tenga permisos de edición y lectura) debemos ejecutar el comando **chmod**, seguido del código del permiso y el archivo al que se lo queremos cambiar. La tabla 1 muestra cómo establecer dichos permisos.

Tabla 1: Código de permisos de lectura (r), escritura (w) y ejecución (x) de archivos, programas y directorios.

Combinación	Valor	Permiso
- - -	0	Ninguno
- - x	1	Sólo ejecución
- w -	2	Sólo escritura
- wx	3	Escritura y ejecución
r - -	4	Sólo lectura
r - x	5	Lectura y ejecución
r w -	6	Lectura y escritura
rwx	7	Todos

Para establecer los permisos a un archivo, al ejecutar el comando **chmod** tenemos que incluir el código numérico dependiendo de cómo los queramos configurar en bloques de 3 (usuario, grupo y resto). Por ejemplo, si queremos tener como usuarios los permisos de lectura, escritura y ejecución (rwx, 7) de un archivo pero sólo queremos dar al resto de usuarios la opción de leer y ejecutar pero no escribir (r-x, 5), el comando sería:

```
> chmod 755 filename
```

Si por ejemplo quisiéramos dar permisos exclusivos al usuario (rwx, 7), sin que el grupo ni el resto pudieran leer, escribir o ejecutar el archivo (- - -, 0), el comando sería:

```
> chmod 700 filename
```

Y si por ejemplo quisiéramos darle permisos de lectura y escritura al usuario (rw-, 6), de ejecución al grupo (- -x, 2) y de sólo lectura al resto, el comando sería (r- -, 4):

```
> chmod 624 filename
```

Confirma tus conocimientos sobre **gestión de permisos** resolviendo estas preguntas:

1. ¿Qué permisos estoy dando a un archivo si ejecuto el comando chmod 767? ¿Y si lo ejecuto con el código 475?
2. ¿Si quiero darle a un archivo permisos de lectura y ejecución al usuario pero ningún permiso al grupo y solo de escritura al resto, cómo sería el comando a ejecutar?

Más información disponible sobre permisos en [https://www.linuxtotal.com.mx/index.php?cont=info\\_admon\\_011](https://www.linuxtotal.com.mx/index.php?cont=info_admon_011)

### **Sistema de comunicaciones:**

El sistema de comunicaciones permite el intercambio de información entre procesos y programas que se ejecutan localmente con los que lo hacen de forma remota. Las tareas de envío y recepción de información las ejecuta el sistema de comunicaciones a través de las interfaces de red. El SO es el responsable de controlar el envío y recepción de la información, crear y mantener la comunicación para que las aplicaciones envíen y reciban información, y crear y mantener conexiones virtuales entre las aplicaciones locales y las remotas.

### **Partes del sistema operativo:**

En cuanto a las **partes del SO** encontramos:

- **Núcleo:** parte del código que interactúa con el *hardware* del ordenador. Este código hace que el ordenador pueda usar todos sus componentes de forma adecuada y que sea «consciente» de todos ellos.
- **Shell:** parte del SO que interactúa con las aplicaciones y el usuario. Actúa como intermediario entre el núcleo y los procesos que se estén ejecutando.
- **Interfaz:** el usuario es capaz de interactuar con el *shell* por medio de la interfaz de línea de comandos (CLI) o por medio de la interfaz gráfica del usuario (GUI). Es aquí donde pasamos todo el tiempo interaccionando con el dispositivo, sobre todo a nivel de la GUI, que solo es una representación de las opciones más básicas que se pueden utilizar en cada uno de los programas que usamos día a día.

### **Interfaz gráfica del usuario (GUI)**

La GUI es un programa que emplea imágenes, iconos y gráficos para representar información y acciones de la interfaz, como por ejemplo carpetas, comandos y/o programas ejecutables, para facilitar al usuario su empleo.

*Un ejemplo son las carpetas del explorador, el botón de guardar o el de subir un directorio.*

Cuando empleamos la CLI o línea de comandos, un nuevo mundo se abre ante nosotros. Pero esto merece un apartado propio que vendrá detallado en la siguiente sección.

#### **0.1.3 Introducción a la línea de comandos**

##### **¿Por qué la línea de comandos?**

La línea de comandos o *command-line* es una interfaz de usuario que carece de representación gráfica y se trabaja con ella mediante instrucciones que se escriben a través del teclado. En la gran mayoría de casos, la interfaz gráfica que usamos todos los días es un envoltorio que transforma en cajas y botones cada una de las instrucciones que se pueden realizar en la línea de comandos. Cuando la

utilizamos, desbloqueamos el potencial de todo el *software* presente en nuestro dispositivo. Pero esto tiene un coste: el usuario DEBE conocer las instrucciones y sus modificadores para realizar las tareas que quiere hacer. Ya no hay menús o botones que indiquen lo que se puede hacer, sino que este conocimiento debe estar en la cabeza del usuario. Y aquí el principiante se topa con el primer escollo, acostumbrado a la facilidad de la interfaz gráfica en contraposición a la carga de trabajo que supone manejar la línea de comandos.

El principiante debe distinguir entre aprender y utilizar la línea de comandos. Para trabajar, se deben conocer las instrucciones exactas. Una vez conocidas, la velocidad con la que se obtienen resultados y la profundidad alcanzada en el análisis de datos no tiene comparación con el trabajo equivalente realizado en la interfaz gráfica. Por lo tanto, la desventaja de usar la línea de comandos frente a la interfaz gráfica es que si se quiere hacer algo por primera vez hay que invertir más tiempo y esfuerzo. Sin embargo, una vez aprendido su uso y conforme más veces consecutivas se emplee, el ahorro de tiempo a la hora de analizar datos, desarrollar programas y ejecutar flujos de trabajo supera con creces el tiempo y esfuerzo invertidos en aprender. Si a eso le sumamos que en Ciencias es muy frecuente repetir los análisis cientos de veces hasta que cuadran los resultados con la realidad observada, las ventajas del trabajo y tiempo invertidos son evidentes. También existe una segunda ventaja: se gana conocimiento y experiencia para situaciones similares futuras (si se cambia de rama de investigación, por ejemplo).

El empleo de programas con interfaz gráfica supone otro problema a la hora de configurar parámetros para llevar a cabo funciones que nos interesen. Puede suceder que no tengan fácilmente disponible opciones que, con el uso de la línea de comandos, sí sean intuitivas de configurar y/o modificar. Con esto se quiere resaltar la importancia del aprendizaje de la línea de comandos para llevar a cabo el trabajo rutinario de análisis de datos. El ahorro de tiempo es tal que la suma de tiempo de aprendizaje + tiempo de trabajo está muy por debajo del invertido en el trabajo equivalente en una interfaz gráfica. Y todo ello si no consideramos lo que puede conseguirse automatizando los procesos para la obtención de resultados biológicos, lo cual se explicará en otras secciones de este manuscrito.

### ¿Cómo usar la línea de comandos?

Todos los sistemas operativos poseen una línea de comandos, pero nosotros vamos a trabajar con la línea de comandos de Linux. Esto se debe a que los SO Linux son derivados de UNIX, uno de los primeros SO en usarse en supercomputación y que además, está desarrollado para entornos científicos. De ahí que la comunidad científica esté muy apagada a Linux, y en comparación existe muy poco desarrollo de herramientas científicas en otros sistemas operativos. Del mismo modo, si se trabaja en el supercomputador Picasso, debemos saber que su SO es una extensión de Linux igualmente.

La forma de acceder a una terminal de Linux depende del sistema del que partamos:

- Si partimos de una distribución de Linux, es sencillo: pulsando la combinación de teclas CTRL + ALT + T.
- Si partimos desde Windows hay varias opciones:
  1. Por un lado se puede usar un programa muy completo llamado MobaXterm. Descarga la versión instalable, no la portable. El MobaXterm, posee una interfaz que permite tener varias terminales organizadas en pestañas (muy útil cuando queramos trabajar en dos carpetas a la vez). Además tiene una ventana (izquierda) que consiste en un explorador gráfico de archivos, que puede ser de gran utilidad para visualizar archivos.
  2. En los últimos años, el sistema operativo Windows 10 ha incorporado un sub-sistema de

Linux, el cual incluye una terminal nativa. Para instalar esta terminal hay que activar primero el sub-sistema Linux de Windows (WLS). Primero hay que abrir la terminal *Windows PowerShell* desde el menú de inicio, y después hay que ejecutar el comando<sup>1</sup>:

```
1 Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-
2 Windows-Subsystem-Linux
```

Una vez finalizado el proceso sólo hay que instalar una terminal desde la *Windows Store*, como por ejemplo la terminal de Ubuntu.

- La terminal propia de OSX (Apple) se abre con la combinación de teclas CMD + T.

Una vez abierta la terminal podemos empezar a escribir instrucciones para ejecutar tareas. Para ello debemos conocer el sistema de instrucciones a usar, ya que hay varios sistemas. En mayoría de terminales de Linux se usa BASH. Dentro de la terminal, hay un intérprete de este lenguaje que procesa todos los comandos que se escriben en la terminal.

### ¿Qué es el Prompt?

Es el conjunto de caracteres que se muestran en una línea de comandos para indicar que está a la espera de órdenes. Generalmente se suele escribir como:

- usuario@máquina:~\$
- \$>
- >

En este manual, recuerda que **A MENOS QUE SE ESPECIFIQUE LO CONTRARIO**, cuando se ejecute un comando, no se tiene que escribir el carácter >, ya que simbolizará una ejecución en la línea de comandos.

Como antes hemos dicho, las «órdenes» que se imparten al ordenador desde la línea de comandos se dan mediante comandos. Estos comandos son palabras y conjuntos de letras con una estructura determinada. Para ejecutar un comando simple, basta con abrir la terminal, escribir un comando y pulsar la INTRO o RETURN (retorno). Como se muestra a continuación:

```
1 >whoami
2 username
```

La gran mayoría de programas se pueden ejecutar definiendo varias opciones llamadas parámetros de entrada. Estos parámetros de entrada se definen después del comando (casi siempre la primera palabra que se escribe en la línea de comandos, que es el nombre del programa invocado) y se escriben en forma de *flags*. Los *flags* se suelen escribir con un <-> precediendo una letra, como por ejemplo <-i>, o con <-> precediendo una o varias palabras separadas por guiones, como por ejemplo <-input>. Se pueden especificar uno o más parámetros de entrada con la sintaxis indicada según necesite el programa. Hay que tener en cuenta que no todos los programas cumplen esta convención. Además, muchos de estos parámetros de entrada requieren que se añadan palabras o números, separados del flag por un espacio. Estas palabras o números adicionales al *flag* se llaman atributos. A continuación se puede ver un ejemplo:

```
>command -l --word atributo
```

<sup>1</sup>Aunque el comando se vea en varias líneas, lo que hay después del signo ">" hay que escribirlo en una sola

Las opciones se pueden combinar de dos formas: Escribiendo los *flag* por separado, que es la más común (ideal cuando los *flags* necesitan de un atributo), o juntos en caso de que haya muchos *flags* que no necesiten parámetros de entrada (esta forma es muy poco frecuente de ver).

```
1 > command -a -b atrb -c  
2 > command -acb atrb
```

Hay programas que presentan multitud de opciones y es imposible conocerlas todas. Por ello, existen varias formas de buscar qué parámetros de entrada se pueden proporcionar a un programa: Se puede buscar el manual con el comando «man», seguido del nombre del programa (no todos los programas tienen)

```
>man ls
```

También se pueden consultar los parámetros de entrada del comando a usar mediante la opción estándar de ayuda. A esta opción se accede mediante el flag «-h», o «--help».

```
1 > grep -h  
2 > grep --help
```

## 0.2 Configuración del entorno de trabajo

En esta sección vamos a aprender a instalar el entorno de trabajo para realizar conexiones en remoto al supercomputador. En este caso, trabajaremos con el clúster *Picasso* perteneciente al Servicio de Supercomputación y Bioinformática de la Universidad de Málaga (SCBI), realizando una conexión mediante cliente SSH y montar directorios remotos dependiendo del SO utilizado.

### WARNING: Antes de conectarte al clúster...

- Recuerda que antes de conectarte con el clúster necesitas tener una **cuenta de usuario** en el mismo. Para ello, sigue las instrucciones provistas en su página [web](#) y cumplimenta con el identificador de grupo «bio\_267\_uma» (necesitarás confirmación previa del jefe de grupo). Para más info, leer sección siguiente.

### 0.2.1 Crear cuenta en Picasso

Para acceder a Picasso, se debe crear una cuenta de usuario mediante registro a través de la plataforma. Para ello, se debe acceder al formulario en la siguiente dirección: [https://www.scbi.uma.es/pab\\_registration/](https://www.scbi.uma.es/pab_registration/)

En el campo Head of research se deben incluir los datos relativos al IP de grupo. El código de nuestro grupo es bio\_267\_uma.

Una vez cumplimentado el registro, se devuelve un PDF que debe ir firmado por el usuario final y el IP de grupo, para enviarlo por email a la dirección [soporte@scbi.uma.es](mailto:soporte@scbi.uma.es), declarando que se quiere dar de alta a un nuevo usuario. Pasados unos días, si hay confirmación, se recibirá una respuesta del administrador de sistemas por email donde se facilitará el nombre de usuario y la contraseña.

Para acceder al supercomputador, en la terminal se debe escribir: ssh *usuario@picasso.scbi.uma.es* (sustituyendo *usuario* por el nombre asignado) y teclear la contraseña. Una vez dentro, el manual de usuario se puede consultar a través de las webs que ofrece el mensaje de bienvenida al supercomputador (<https://www.scbi.uma.es/site/scbi/documentation>).

### 0.2.2 Cómo conectarse a Picasso

A partir de ahora, el supercomputador *Picasso* va a ser el entorno de trabajo principal. Por ello, es un requisito esencial obtener una cuenta en el mismo para continuar con el curso siguiendo los pasos y ejemplos que se explican<sup>2</sup>. Una vez obtenido el acceso, el administrador de sistemas asigna un nombre de usuario (username) y una contraseña aleatoria que posteriormente se puede modificar por la línea de comandos. *Picasso* utiliza un sistema operativo (SO) SUSE (Linux) sin interfaz gráfica, por lo que hay que conectarse al mismo de forma remota mediante un protocolo (conjunto de normas de conexión) SSH.

#### Protocolo de conexión SSH

Las siglas SSH vienen de «*Secure SHell*», que es el nombre de un protocolo de conexión remota entre dos ordenadores, y también del programa que implementa dicho protocolo. Este tipo de conexiones se caracterizan por ser muy seguras ya que la información que se transfiere entre un ordenador y otro se encuentra cifrada.

Para establecer una conexión remota, hay que abrir una terminal de Linux o de macOS en nuestro ordenador local (tal y como se describe en el apartado 0.1.3). Una vez hecho, se debe ejecutar el comando `ssh` e indicarle como atributo un nombre de usuario y un host (en el caso de *Picasso* es `picasso.scbi.uma.es`) separados por el carácter @. Tras usar dicho comando se debe introducir la contraseña. Una vez enviada la contraseña, la terminal enviará las órdenes a *Picasso*.

```
1 > ssh username@picasso.scbi.uma.es
2 Password:
```

#### Cambiar contraseña de la cuenta de usuario

En el clúster Picasso existe un comando para cambiar la contraseña que nos asignó el administrador de sistemas una vez activada la cuenta. Para ello, una vez establecida la conexión, hay que escribir el comando `passwd` y a continuación el nombre de usuario. Se pedirá la actual y nueva contraseña.

Con ello accedemos al sistema (*log in*) y se mostrará información de la fecha de la última conexión realizada, un enlace a un manual de uso (cuya lectura recomendamos encarecidamente para comprender aspectos básicos del supercomputador no reflejados en este manual) y un mensaje de bienvenida. La terminal se habilita de forma que podemos ejecutar comandos a continuación.

```
> username@picasso:~>
```

#### Ejercicio: Cambiar contraseña de usuario

Prueba a cambiar tu contraseña predefinida de usuario en Picasso por una que recuerdes empleando el comando `passwd`. ¡No olvides que **nunca** debes compartir tus contraseñas con nadie!

---

<sup>2</sup>Si se tiene acceso a otro clúster, como el de la Universidad Pablo de Olavide, o al del BSC-CNS, se debe establecer la conexión al mismo utilizando la dirección facilitada por los administradores de sistema.

### Conexión SSH a nodo login con GPU

Si llega el momento en el que necesites experimentar con código que requiera uso de GPU, pero todavía no está depurado y listo para mandar al sistema de colas, existe otro *log in* como al que hemos accedido en Picasso, pero que nos permite ejecutar trabajos de manera interactiva haciendo uso GPU sin necesidad de solicitarla al sistema de colas. Para ello, de manera similar a cómo nos conectamos al sistema de Picasso (pero ya desde dentro de él en este caso) para acceder al login de GPUs, ejecutaremos el comando **ssh loginexa** y solo hará falta introducir nuestra contraseña para acceder. Para comprobar que efectivamente estamos en el *login exa* con GPUs, podremos ver que en el *prompt* de comandos en vez de aparecer el formato user@picasso, aparecerá user@loginexa, y que ejecutando el comando **nvidia-smi** nos devolverá información sobre las GPUs que tiene dicho nodo (si probamos a ejecutar el mismo comando en el *login* de Picasso, no nos aparecerá nada). Podemos salir de este nodo para volver al nodo *login* de Picasso escribiendo **exit**

#### 0.2.3 Facilitar la conexión a *Picasso* y que se mantenga activa

Existe un par de configuraciones que pueden hacer nuestra conexión a *Picasso* algo más rápida y sencilla.

Una de ellas consiste en crear un alias para realizar la conexión ssh, de manera que así no tenemos que recordar la dirección completa del servidor al que nos conectamos. Para ello entramos con un editor de texto (como nano, vim o vi) al siguiente archivo (desde una consola en local, es decir, sin estar conectados a *Picasso*):

```
> nano ~/.ssh/config
```

Y en dicho archivo guardamos una configuración que contendrá el nombre del alias para realizar la conexión (Host), la dirección completa al servidor (Hostname) y el nombre de usuario que tenemos para acceder a *Picasso* (User). En el recuadro de abajo se deja un ejemplo para un usuario que se llame "gojo":

```
1 Host picasso
2 Hostname picasso.scbi.uma.es
3 User gojo
```

De manera que si ahora guardamos el contenido de dicho archivo, podemos proceder a conectarnos a *Picasso* usando el formato ssh alias (en nuestro caso alias sería *picasso*, ya que el alias lo determinada el campo Host), tal como el siguiente ejemplo:

```
> ssh picasso
```

No obstante todavía necesitamos seguir introduciendo la contraseña para poder conectarnos. Podemos facilitar también este paso utilizando un sistema de claves públicas y privadas. Podemos lograr esto cambiando el directorio a */.ssh* y lanzando los comandos *ssh-keygen* (pulsamos varias veces enter para dejar las opciones por defecto que nos salgan) y *ssh-copy-id* (que nos pedirá que introduzcamos nuestra contraseña de acceso a *picasso*) , tal y como se va a mostrar en el siguiente ejemplo, en el que se crearán claves publicas para el alias "picasso" que hemos creado previamente:

```
1 > cd ~/.ssh
2 > ssh-keygen
3 > ssh-copy-id picasso
```

Una vez hecho esto, cuando queramos conectarnos a *Picasso* ya solo tendremos que usar "ssh *picasso*" y podremos conectarnos directamente.

Finalmente para que la conexión con *Picasso* no se corte tras unos minutos de inactividad en la terminal, podemos añadir unas líneas adicionales al archivo de configuración que creamos a la hora de hacer un alias de conexión a *Picasso*. De manera que volvemos a abrir con un editor de texto el archivo que abrimos anteriormente, tal que así:

```
1 > nano ~/.ssh/config
```

Y añadimos las siguientes líneas (antes o después de las que añadimos anteriormente):

```
1 Host *
2   ServerAliveInterval 300
3   ServerAliveCountMax 2
```

De esta manera la conexión a *Picasso* de nuestra terminal se mantendrá activa de manera indefinida, aunque estemos un gran tiempo inactivos en esa terminal.

#### 0.2.4 Acceder a los archivos de *Picasso*

El siguiente paso para construir el entorno de trabajo es el montaje del almacenamiento de *Picasso* el ordenador local. Este paso es muy útil porque permite acceder a los archivos de usuario almacenados en *Picasso* mediante la interfaz gráfica del ordenador local. Cabe destacar que podemos distinguir 2 discos en *Picasso*: *HOME* y *FSCRATCH*. En primer lugar, *HOME* es el disco principal en el que guardaremos el código y aquellas muestras de las que somos responsables y no podemos perder. Por otro lado, *FSCRATCH* es otro disco con una política de borrado, es decir, los datos almacenados se eliminan tras un determinado período de tiempo, de manera que se recomienda almacenar la ejecución y muestras a corto plazo en él.

Dependiendo del SO instalado en el mismo, se usará un método u otro:

##### Ubuntu

Este SO ofrece en su explorador de archivos la forma más sencilla de acceder a un repositorio en remoto, ya que la conexión SSH es nativa de Linux y el sistema de archivos del de *Picasso* es similar al de Ubuntu. En este caso, hay que abrir el explorador de archivos y, en concreto, la pestaña «Otras ubicaciones». Una vez dentro, abajo se habilita una ventana «Conectar al servidor» para introducir el directorio remoto al que conectar. Aquí, empleamos el protocolo SFTP, y conectamos al servidor escribiendo «`sftp://username@host/`». Por ejemplo, para conectar a nuestra cuenta de *Picasso* sería «`sftp://username@picasso.scbi.uma.es/`» (Figura 1).

##### Protocolo de transferencia de archivos SFTP

Las siglas SFTP hacen referencia a «SSH File Transfer Protocol», un método de transferencia de archivos entre sistemas en remoto. SFTP emplea el protocolo SSH para realizar la autenticación de credenciales y establecer una conexión segura.

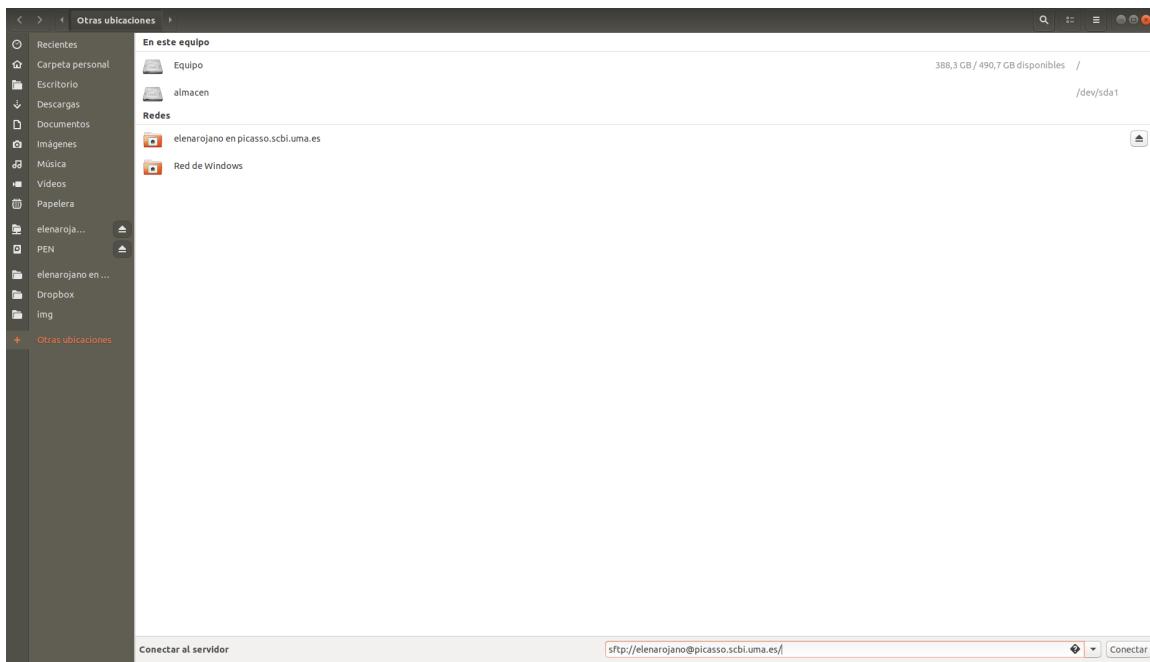


Figura 1: Configuración del protocolo SFTP a Picasso empleando el explorador de archivos de la versión de Ubuntu 18.04.

#### WARNING: Ojo a las versiones de los SOs

Si se tiene instalado un SO inferior a la versión de Ubuntu 18.04 se debería actualizar a esta versión. Y si se prefiere seguir con los métodos de la Edad Media, se debe seguir el procedimiento explicado en [digitalocean.com](#).

#### 0.2.5 macOS

Para instalar el protocolo de transferencia de archivos, tomando como referencia el macOS Catalina, se deben seguir las instrucciones portal OSXFUSE. Se descargan dos utilidades: 1) FUSE para MacOS (OSXFUSE), una imagen de disco (archivo .dmg) con las herramientas necesarias para aumentar las capacidades nativas de manejo de archivos de OSX, y 2) SSHFS, programa (archivo .pkg) para montar sistemas de archivos remotos mediante SSH. Ambas utilidades se descargan desde la misma página web (FUSE for macOS 3.10.3 y SSHFS 2.5.0, versiones estables a fecha de escritura de este manuscrito).

El procedimiento a seguir consiste en primero instalar la imagen de OSXFUSE y posteriormente instalar el paquete SSHFS. Una vez realizado esto, seguir las instrucciones que se dan en la página web de [github](#):

1. Crear una carpeta en local que servirá de punto de conexión con nuestro sistema de carpetas en remoto (al contenido almacenado en *Picasso*). Una vez hecho, se tiene que ejecutar el comando que se especifica a continuación <sup>3</sup> :

```
| ##### Crear la carpeta donde montar el sistema de archivos #####|
```

<sup>3</sup>Aunque el comando se vea en varias líneas, lo que hay después del signo ">" hay que escribirlo en una sola

```
2 >mkdir ruta_de_carpetade_montaje
3 ##### Montar la carpeta de usuario de Picasso #####
4 >sshfs username@picasso.scbi.uma.es:/mnt/home/users/bio_267_uma/
5 username/ ruta_de_carpetade_montaje
```

2. Confirmar que la instalación se ha realizado satisfactoriamente. Una vez creado el punto de conexión se debe configurar el sistema de carpetas en remoto. El ícono de la carpeta debería cambiar una vez establecida la conexión y deberían aparecer las subcarpetas de tu usuario en *Picasso*.

### 0.2.6 Windows

En la primera subsección de este apartado explicamos la excepcionalidad que sucede con Windows. Si quieres saber el por qué hay que emplear utilidades externas para enlazar el sistema de carpetas local-remoto, lee la siguiente subsección. Pero si tienes Windows actualizado a la última versión de 2020, sigue el protocolo explicado en la subsección 0.2.6. Para asegurarte de la versión que tienes, pulsa la tecla de Windows + r y escribe en la consola: winver. Pulsa aceptar y confirma tu versión de SO.

**Windows, método antiguo (usar para versiones de Windows anteriores a Windows 10, versión 2014, compilación 19041.572)**

Las versiones anteriores de 2020 de Windows 10 no tenían un modo nativo para montar un sistema de archivos diferente a los que usan por defecto (FAT o NTFS). Para ello, se desarrolló una utilidad llamada Dokan que facilita el montaje de sistemas de archivos de tipo Linux. Para montar sistemas de archivos de forma remota vía SFTP en Windows se emplea un programa llamado WinSSHFS que usa Dokan. Este programa permite montar y desmontar diferentes sistemas de archivos con una interfaz gráfica bastante sencilla. Para usarlo hay que instalar la versión 1.0.3 de Dokan disponible en GitHub (descargar e instalar con la configuración por defecto el archivo «DokanSetup.exe»).

#### WARNING: Versiones e incompatibilidades

Se deben instalar las versiones exactas de los programas que se indican ya que otras versiones podrían no ser compatibles.

El siguiente paso es descargar y configurar WinSSHFS, disponible en el repositorio de GitHub. Primero hay que descargar la versión 1.6.0.14, la cual es difícil de encontrar dentro del repositorio. Para localizarla, hay que acceder a la versión 1.6.1.13 devel RC5 y descargar el archivo «previous.release-1.6.0-rc3.zip». Este archivo comprimido contiene una versión portable de WinSSHFS, por lo que no es necesaria su instalación, solo una serie de configuraciones. Primero hay que crear la carpeta «C:\Program Files\WinSshFS», después hay que descomprimir el archivo «previous.release-1.6.0-rc3.zip» dentro de esta nueva carpeta y para finalizar, hay que crear un acceso directo del programa en el escritorio. Para ello, *click* derecho con el ratón en el archivo «WinSshFS.exe»; «Enviar a»; «Escritorio (Crear acceso directo)».

Otra peculiaridad de este sistema es que el programa WinSSHFS no funciona correctamente a la hora de realizar la autenticación interactiva, por lo que hay que automatizar el proceso. Para ello, hay que usar un sistema de claves SSH. Este sistema de claves consiste en una clave privada, única de un ordenador (en este caso del ordenador local) y una clave privada que se puede compartir con cualquier servidor. Con este sistema, el ordenador local se puede conectar con cualquier servidor de

forma automática comprobando ambas claves. Para generar estas claves hay que abrir una terminal (Terminal de Ubuntu o MobaXterm) y crear la carpeta «keys» donde se va a guardar la clave privada:

```

1 ##### En el caso de la terminal de Ubuntu #####
2 >mkdir /mnt/c/keys
3 ##### En el caso de MobaXterm #####
4 >mkdir /media/c/keys

```

Luego hay que usar el comando ssh-keygen para crear ambas claves.

```

1 ##### En el caso de la terminal de Ubuntu #####
2 >ssh-keygen -f /mnt/c/keys/id_rsa
3 ##### En el caso de MobaXterm #####
4 >ssh-keygen -f /media/c/keys/id_rsa

```

El comando solicitará una «passphrase», pero no conviene indicarle ninguna, así que solo se debe pulsar la tecla «INTRO» dos veces (una para rehusar la «passpharase» y otra vez para confirmar la elección). Este comando habrá creado una clave privada llamada «id\_rsa» y una clave pública «id\_rsa.pub» dentro de la carpeta «keys». Como la clave privada se encuentra ya en el ordenador local, hay que copiar la clave pública dentro de *Picasso*<sup>4</sup>:

```

1 ##### En el caso de la terminal de Ubuntu #####
2 >scp /mnt/c/keys/id_rsa.pub username@picasso.scbi.uma.es:/mnt/home/users/
   bio_267_uma/username/.ssh/authorized_keys
3 ##### En el caso de MobaXterm #####
4 >scp /media/c/keys/id_rsa.pub username@picasso.scbi.uma.es:/mnt/home/users/
   /bio_267_uma/username/.ssh/authorized_keys

```

Cuando estén las versiones concretas de ambos programas instaladas y las claves correctamente ubicadas, se procederá a configurar el programa WinSSHFS *Picasso*. Primero hay que abrir el programa y añadir un nuevo volumen: para ello, hay que hacer *click* en «Add» y configurarlo, tal y como se muestra en la figura 2, y luego hacer *click* en «Save». El montaje y desmontaje de la carpeta de usuario se realizará automáticamente al abrir y cerrar la aplicación, respectivamente.

## Windows 10, actualización 2020

Para preparar el sistema de carpetas local-remoto hay un nuevo protocolo disponible a través de la plataforma de GitHub SSHFS-Win (<https://github.com/billziss-gh/sshfs-win>). En su página web, describe las instrucciones para descargar, instalar la herramienta en tu sistema y configurar los path remotos a tu ordenador local (apartado Basic Usage, Windows Explorer). Con más detalles:

- Descarga e instala la última versión de WinFsp (<https://github.com/billziss-gh/winfsp/releases/latest>).
- Instala la última versión de SSHFS-Win (<https://github.com/billziss-gh/sshfs-win/releases>). Elige el instalador x64 or x86 dependiendo de la arquitectura de tu ordenador.

Para confirmar la arquitectura de tu ordenador, en panel de control accedes a sistema y seguridad, sub-apartado sistemas y ahí puedes comprobarlo (figura 3).

Una vez instaladas las herramientas, vamos al explorador de archivos y pulsamos en Equipo. En la parte superior del explorador de archivos se habilitará un panel donde podamos hacer *click* sobre «Conectar a unidad de red». Se habilitará un panel (figura 4) donde debemos escribir la ruta para

<sup>4</sup>Aunque el comando se vea en varias lineas, lo que hay después del signo ">" hay que escribirlo en una sola

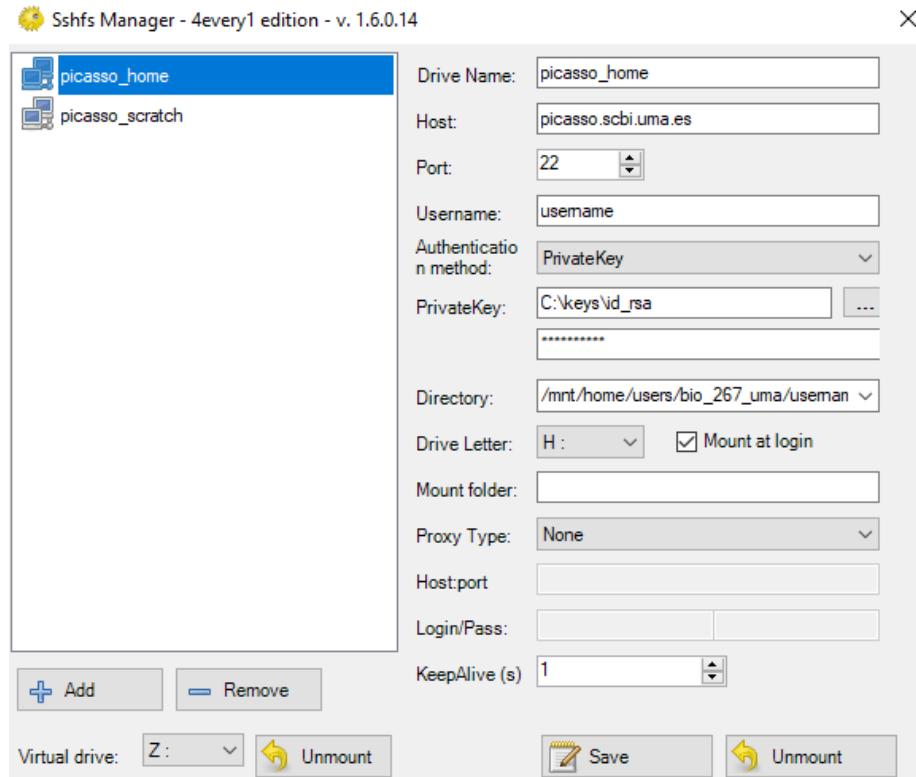


Figura 2: Ejemplo de configuración del programa WinSSHFS para Windows 10. Se recomienda usar la misma configuración, pero sustituyendo «username» por el nombre de usuario correcto.



Figura 3: Panel de información del sistema operativo Windows 10.

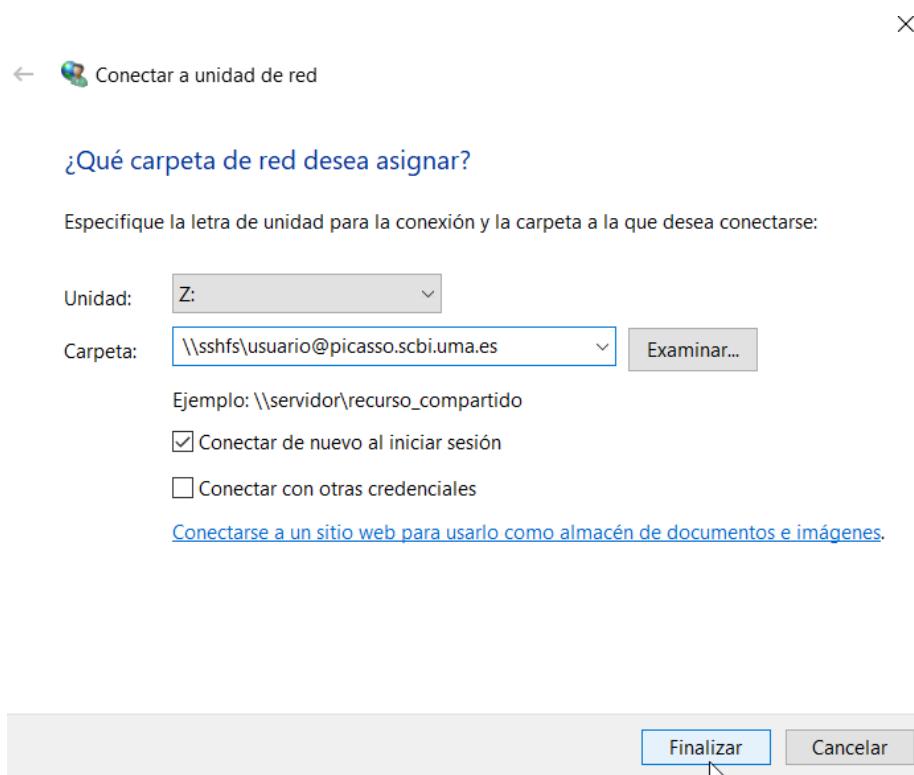


Figura 4: Panel de configuración de conexión remota a Picasso a través de Windows 10.

conectar nuestro equipo local en remoto:

\\\sshfs\usuario@picasso.scbi.uma.es

Para ello, escribe en el campo «carpeta» la ruta a Picasso (cambiando «usuario» por tu nombre de usuario).

Y tendremos habilitado nuestro sistema de carpetas conectado al clúster cada vez que encendamos nuestro equipo (figura 9).

#### Para configurar el FSCRATCH:

En este caso, accedemos desde la terminal al editor de linux *gedit* mediante el comando:  
*gedit ./config/gtk-3.0/bookmarks*.

De esta forma, se abre una nueva ventana que nos muestra los bookmarks guardados en Archivos. A continuación, introducimos el siguiente comando, que en caso de ser del grupo bio\_267\_uma es:

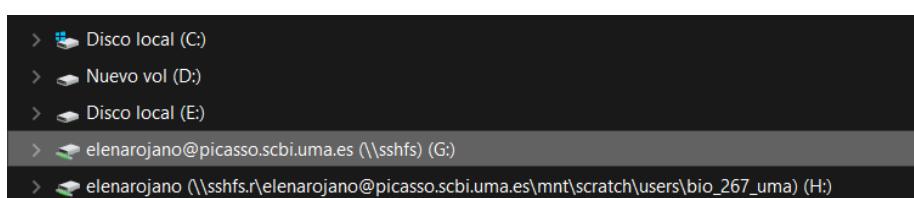


Figura 5: En el explorador de archivos de Windows 10 estará disponible el sistema de carpetas para el usuario preestablecido.

sftp://usuario@picasso.scbi.uma.es/mnt2/fscratch/users/bio\_267\_uma/usuario

Tras escribir el comando, añadiremos el nombre de usuario y fscratch, de manera que quedaría: *comando usuario fscratch*.

*Nota:* Este método se puede llevar a cabo también con el HOME del servidor, añadiendo su ruta correspondiente.

#### Para configurar el FSCRATCH desde Windows:

Repetiendo el proceso empleado para conectar con los archivos del HOME pero incorporando la siguiente ruta: \\sshfs\usuario@picasso.scbi.uma.es\..\..\..\..\mnt2\fscratch\users\bio\_267\_uma

#### Para configurar el SCRATCH:

En caso de querer configurar la carpeta de SCRATCH, se designa una unidad distinta a la del login de Picasso (E:, H:, I:, etc) y al conectar especificamos la carpeta con la ruta:

\\sshfs.r\usuario@picasso.scbi.uma.es\mnt\scratch\users\bio\_267\_uma

### 0.3 Primeros pasos en la línea de comandos

Una vez configurado el entorno de trabajo, el primer paso es familiarizarse con la línea de comandos de Linux y con el clúster *Picasso*. Para ello hay que entrar primero en el clúster con el comando ssh. Por defecto, la carpeta de inicio de sesión de *Picasso* es la carpeta *Home*: esta carpeta es la principal (*root* o raíz) de tu usuario. En todos los sistemas de almacenamiento hay dos tipos de elementos, las carpetas o directorios y los archivos. La diferencia entre ambos es sencilla, los **archivos** son elementos que almacenan información, y las **carpetas y directorios** son elementos que almacenan archivos, u otras carpetas o directorios. Lo primero que haremos en *Picasso* es crear una carpeta llamada «initial\_steps» y una carpeta llamada «folders\_adn\_files» dentro de ella. Para ello usaremos el comando «mkdir» (*make subdirectory*).

```
1 > mkdir initial_steps  
2 > mkdir initial_steps/folders_adn_files
```

Para comprobar que la carpeta se ha creado correctamente usaremos el comando «ls» (*list*) que lista todos los archivos dentro de una carpeta. El argumento por defecto que usa este comando es el directorio actual «./»

```
1 > ls initial_steps  
2 folders_adn_files
```

Para acceder a las carpetas que se han creado con el comando cd (*change directory*) indicando el nombre de la carpeta como argumento (en el caso de que la carpeta se encuentre dentro del directorio actual) o la dirección completa de la carpeta (Figure 6).

```
1 > cd initial_steps
```

Se puede volver a la carpeta «home» de varias formas: i) ejecutando el comando «cd» sin ningún argumento, ii) usando el comando «cd» con la dirección completa de la carpeta «home», el cual está guardado en la variable «\$HOME» o iii) usando el alias de la dirección del \$HOME (o «~»).

```
1 > cd  
2 > pwd  
3 /mnt/home/users/bio_267_uma/username  
4 > cd $HOME
```

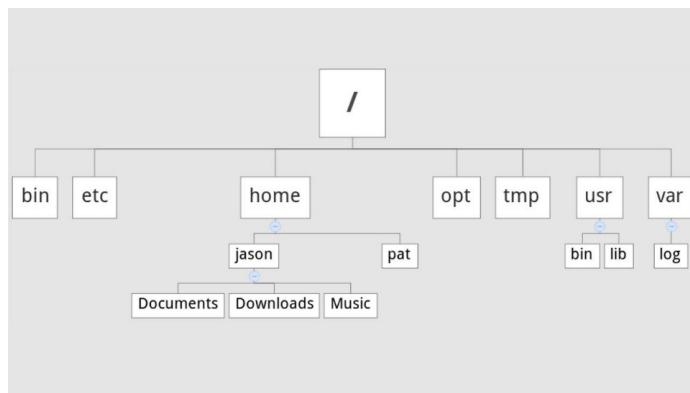


Figura 6: La dirección de un elemento, también llamada «path», es un texto único que indica donde se encuentra ese archivo o carpeta. El «path» se construye indicando toda la jerarquía de carpetas, desde la carpeta raíz (carpeta principal de un sistema Linux que se indica como «/») hasta el directorio actual. En la dirección, cada carpeta de la jerarquía se separa con el carácter «/», que indica que el elemento de la derecha se encuentra dentro del directorio de la izquierda. En este ejemplo, la dirección de la carpeta «Music» es «/home/jason/Music».

```

5 > pwd
6 /mnt/home/users/bio_267_uma/username
7 > cd ~/
8 > pwd
9 /mnt/home/users/bio_267_uma/username
  
```

#### WARNING: Se ha cometido un error!

La carpeta `folders_adn_files` contiene una errata, debería llamarse `folders_and_files`

Hay que corregir este error. Hay dos formas de hacerlo: i) usando el comando «rm» (del inglés *remove*) con la opción «-r [folder]» (la opción «-r» viene del inglés *recursive* y es necesaria para borrar carpetas y subcarpetas), ¡pero muchísimo cuidado de **no borrar elementos importantes!**,

```

1 > rm -r initial_steps/slashfolders_adn_files
2 > mkdir initial_steps/folders_and_files
  
```

ii) usando el comando «mv» (*move*), con el nombre del elemento origen como primer argumento de entrada y el elemento objetivo como segundo.

```

> mv initial_steps/folders_adn_files initial_steps/folders_and_files
  
```

#### «Home» en el sistema de archivos de Picasso

En el sistema de archivos de Picasso, la carpeta «home» de los usuarios del grupo **bio\_267\_uma** es «/mnt/home/users/bio\_267\_uma/username», donde «username» es el nombre de cada usuario.

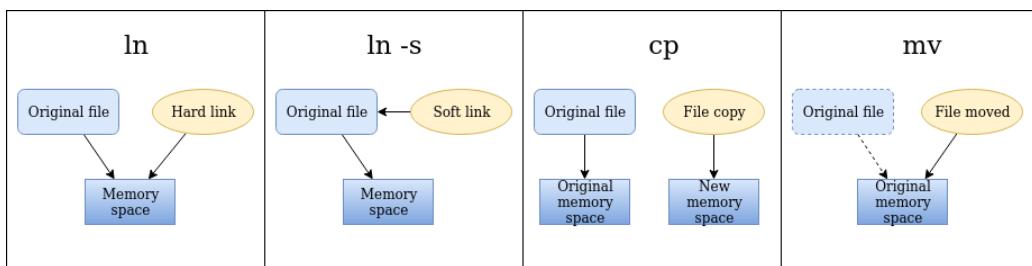


Figura 7: Estructura de carpetas básica del curso. Las cajas de color gris simbolizan carpetas, y las cajas de color verde simbolizan enlaces simbólicos. Las líneas discontinuas hacen referencia a elementos que dejan de existir después de ejecutar el comando.

### Comando «pwd»

El comando «`pwd`», que viene del inglés *Print workdirectory* se usa para imprimir el directorio actual de trabajo.

Además de los archivos y carpetas, hay otro tipo de elementos en el sistema de archivos, denominados enlaces. Hay dos tipos, los **físicos** (*hard links*) y los **simbólicos** (*soft links*). Todos los archivos tienen asignado un espacio de memoria. Los enlaces físicos son aquellos que apuntan a un archivo con acceso a su espacio de memoria designado, obteniendo así las mismas propiedades que el archivo original (Figura 7). Los enlaces físicos se generan con el comando «`ln`», indicándole el archivo original como primer argumento y el *path* del futuro enlace físico como segundo argumento.

```
> ln original_file path_to_hard_link
```

Los enlaces simbólicos son aquellos *paths* alternativos al del archivo original. Cuando se intenta acceder a un enlace simbólico, éste nos redirecciona al archivo original (Figura 7). Cada tipo de enlace tiene su utilidad, pero en este curso se usarán solo los enlaces simbólicos, ya que la modificación de un enlace simbólico no afecta al archivo original. Los enlaces simbólicos se generan de manera similar a los físicos, pero hay que utilizar el parámetro «`-s`».

```
> ln -s original_file path_to_soft_link
```

Además de los comandos para eliminar, enlazar y mover elementos, hay también un comando (comando «`cp`») para crear una copia del archivo, creando a su vez, un espacio de memoria nuevo (Figura 7).

### Ejercicio: Crea las carpetas que se usarán en el curso

En este ejercicio, hay que replicar la estructura mostrada en la figura 8 dentro de *Picasso*. Para ello debes utilizar los comandos «`mkdir`», «`cd`» y «`ln`». Recuerda que si cometes un error lo puedes solucionar con los comandos «`rm`» y «`mv`».

## 0.4 Manipulación de archivos de texto

La información contenida dentro de los archivos se almacena en forma de texto plano. El texto plano es un texto legible que no contiene ningún formato tipográfico. Como se mostró en la sección

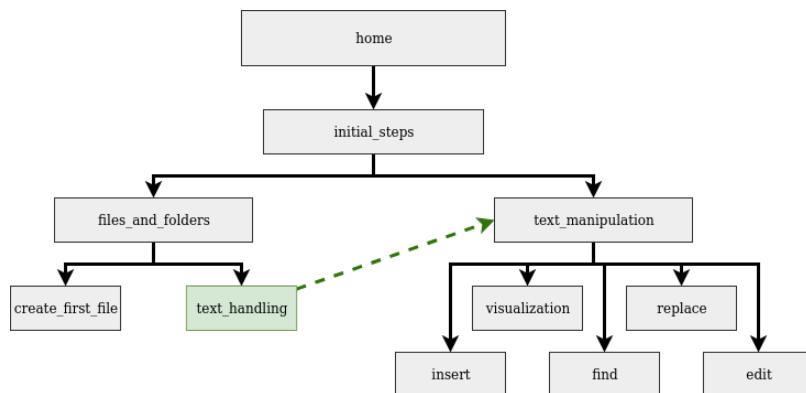


Figura 8: Estructura de carpetas básica del curso. Las cajas de color gris simbolizan carpetas, y las cajas de color verde simbolizan enlaces simbólicos.

0.1.2, los archivos se pueden leer, escribir y ejecutar: en este apartado se mostrará cómo leerlos y escribirlos.

#### 0.4.1 Comandos touch, cat, echo y printf

El primer paso es crear un archivo vacío. Para ello, hay que entrar en la carpeta «create\_first\_file» y ejecutar el comando «`touch my_first_file`» para crear un archivo vacío llamado «`my_first_file`». El siguiente paso consiste en ver el contenido del archivo. Para ello se emplea el comando «`cat` (concatenate), que lee el archivo e imprime en pantalla su contenido.

```

1 >cd files_and_folders/create_first_file
2 >touch my_first_file
3 >cat my first_file
4 >
  
```

#### Redirección de salida de programas.

El resultado del comando «`touch`» es un archivo vacío, por lo que al ejecutar el comando «`cat`» no imprimirá nada en pantalla. Para incluir información en el archivo, se puede emplear el comando «`echo`», el cual imprime cadenas de texto en pantalla. Para que el texto escrito detrás del comando «`echo`» se incluya en el archivo «`my_first_file`», hay que re-direccionarlo. Esta acción se realiza mediante el uso del símbolo operador «`>>`».

#### WARNING:

Generar archivos desde cero con el comando «`touch`» puede suponer un problema dependiendo del SO en el cual se generen. Hay veces que al ejecutar programas creados a partir de un comando «`touch`» no funcionen porque se les asigna caracteres especiales, como el retorno de carro de Windows. Para solucionar este problema, el comando «`dos2unix`» transforma archivos de texto plano en formato DOS o Mac a formato Unix.

### Sobreescribir e incluir información

Cuando se re-direcciona la salida de un programa a un archivo, por ejemplo cuando ejecutamos «echo 'hola' > archivo.txt», se incluye el contenido 'hola' al archivo «archivo.txt». Se puede insertar más información para concatenarla con la existente en un archivo empleando el operador «>>>» en lugar de «>». En caso de ejecutar otro programa y volver a emplear el operador «>» sobre el archivo «archivo.txt», se reescribiría el contenido del mismo.

Estos conceptos de entrada y salida de programas se definen como **descriptores de archivo**. El *STDOUT* es el resultado de un programa que se muestra en la terminal. El *STDIN* es la entrada estándar a un programa y la terminal lo recibe a través de lo que se escribe en el teclado. Existe otro tipo de descriptor de archivo que es el *STDERR*, otro tipo de salida que registra el error producido en un programa y que es esencial para realizar depuraciones de código.

### Comando «printf» en sustitución de «echo»

El comando «printf» suele emplearse en sustitución de «echo» cuando las cadenas de texto son más complejas, ya que funciona mejor cuando se incluyen saltos de línea o tabulaciones, por ejemplo.

## 0.4.2 Comandos grep, cut, sed y tr

Del mismo modo, se puede concatenar la salida de un programa con la entrada a otro. Por ejemplo, se quiere concatenar la salida de un programa y queremos buscar un patrón de texto que nos interese para guardarlo en un archivo nuevo. El comando para buscar un patrón de texto es «grep» (*globally search a regular expression and print*) y toma como argumentos de entrada el patrón a buscar en un archivo determinado. Por lo tanto, este comando o programa requiere dos argumentos de entrada y devuelve uno de salida, que será el resultado del patrón buscado (si hay coincidencias).

### Ejercicio: Incluir información en un archivo

Con los comandos enseñados hasta el momento, incluye en el archivo «my\_first\_file» la siguiente cadena de texto: «el sol se pone por el oeste». A continuación, busca «oeste» y, en caso de que exista, redirige la cadena de texto al completo a un nuevo archivo llamado «busqueda\_string.txt». ¿Sabrías redirigir únicamente la cadena «oeste» a un nuevo archivo de texto? Prueba a ejecutar el argumento de entrada «-w» cuando ejecutes el comando de búsqueda.

Observa que para realizar este ejercicio se ha ejecutado un primer programa (echo), su salida ha sido re-direccionada al archivo «my\_first\_file» y para la búsqueda del contenido se ha usado el programa «grep», re-direccionando su salida a un nuevo archivo. Hay operaciones que se pueden simplificar en una sola línea de ejecución concatenando la entrada y salida de archivos de forma apropiada, como se explica a continuación.

### Concatenación de operaciones.

El *STDOUT* de un programa se puede redirigir a un programa mediante el operador de concatenación «|» (también conocido como *pipe*). Mediante este operador se puede concatenar la salida de un programa y emplearla como entrada para otro, de manera que no haya que escribir los resultados a

disco sino que se guardan en la memoria RAM. Una vez ejecutadas las operaciones, el resultado se puede re-direccionar en caso de que sea necesario con los operadores explicados anteriormente.

A continuación, vamos a generar un archivo llamado «animales.txt» dentro del directorio de trabajo. Dentro de este archivo vamos a incluir unas líneas de texto que van a estar separadas mediante saltos de línea (`\n`). Este código de control indica el movimiento del texto a la siguiente línea de la terminal. Para ello, en lugar de usar el comando «echo» emplearemos el comando «printf».

```

1 > printf "animal\nperro\nbuitre\ncaballo\n" > animales.txt
2 > cat animales.txt

```

Se deben observar dos cosas: 1) Hay que poner un salto de línea (`\n`) al final de la cadena de texto ya que el comando «printf» no la incluye por defecto como «echo»; y 2) El salto de línea no lleva un espacio detrás del texto que se quiere concatenar. En caso de haberlo incluido, el texto tendría justo delante de cada animal un espacio que no nos interesa. Hay que tener mucho cuidado a la hora de manipular cadenas de texto, ya que aunque a vista parezca intuitivo poner espacios para facilitar su lectura, el computador los interpretará y esto podría generar problemas.

Vamos a modificar el contenido del archivo «animales.txt», de manera que lo convirtamos en una tabla tabulada como la mostrada en el ejemplo 0.4.2. Para ello, tendremos que utilizar tabulaciones (`\t`).

animal	nombre	color
perro	toby	blanco
buitre	hawk	negro
caballo	bojack	gris

```

1 > printf "animal\tnombre\tcolor\nperro\ttoby\tblanco\nbuitre\tbuitre\thawk\tnegro\
2 > ncaballo\tbojack\tgris\n" > animales.txt
> cat animales.txt

```

Ahora buscaremos en el archivo «animales.txt» la cadena de texto «buitre»:

```

1 > grep 'buitre' animales.txt
2 buitre hawk negro

```

El programa «grep» ha encontrado la cadena de texto «buitre» y ha devuelto el contenido de la línea dentro del archivo sin devolver el resto del contenido. Imaginemos que queremos sin embargo que nos busque exclusivamente la palabra «buitre» sin tener en cuenta el resto del texto. Esta palabra se encuentra en la primera columna del archivo «animales.txt». Para ello, debemos seleccionar la columna en la que se encuentra el patrón a buscar con «grep» empleando el comando «cut». Este comando corta secciones de un archivo a partir de un carácter delimitador (tabulación por defecto). Requiere como argumentos de entrada el número de la columna a cortar (-f número) y el archivo con el que trabajar. Observa el código a continuación para ver un ejemplo de su uso:

```

1 > grep 'buitre' animales.txt | cut -f 1
2 buitre

```

Se ha empleado en este comando la concatenación de instrucciones con pipes (`|`) como se explicó anteriormente. Primero se ha buscado con el comando «grep» la cadena de texto 'buitre' en el archivo que contiene la información «animales.txt». Una vez localizada la línea en la que está la

palabra, se ha usado esa información como entrada para el comando «`cut`», que ha procesado la información para seleccionar de la primera columna (`-f, field`) el contenido a imprimir en pantalla («`buitre`»).

Imaginemos que nos equivocamos a la hora de escribir nuestro archivo «`animales.txt`» y el nombre de nuestro buitre no era «`hawk`» sino «`vulture`». Para realizar sustituciones de cadenas de texto dentro de un archivo, se utiliza el comando `sed`. Requiere unos argumentos de entrada para especificarle el texto a buscar y la nueva cadena de texto por la que reemplázalo, así como el archivo de entrada con la información (en este caso, «`animales.txt`»).

```

1 > sed 's/hawk/vulture/g' animales.txt
2 animal nombre color
3 perro toby blanco
4 buitre vulture negro
5 caballo bojack gris

```

Si se quiere guardar el contenido del cambio, hay que re-direcccionar la salida del mismo a otro archivo. En el caso de querer cambiar caracteres especiales como saltos de línea y tabulaciones, el comando a emplear es «`tr`».

#### 0.4.3 Comandos head, tail y wc

Los comandos que se explican a continuación sirven para mostrar el número de líneas de un archivo partiendo de su cabecera («`head`»), de su final («`tail`») o el número de líneas, palabras y/o caracteres que presenta («`wc`»). Por ejemplo, si empleamos sobre el archivo «`animales.txt`» el comando «`head`», nos devolverá en pantalla el contenido del archivo al completo. Por defecto, los comandos «`head`» y «`tail`» devuelven las primeras 10 líneas del archivo consultado. Si se quiere definir un número concreto de líneas, se tiene que especificar el argumento `-n` seguido del número de líneas a consultar.

```

1 > head -n 2 animales.txt
2 animal nombre color
3 perro toby blanco
4
5 > tail -n 2 animales.txt
6 buitre vulture negro
7 caballo bojack gris

```

El comando «`tail`» es muy útil cuando queremos omitir cabeceras de archivo. Por ejemplo, en el caso de querer omitir el cabecero del archivo «`animales.txt`»:

```

1 > tail -n +2 animales.txt
2 perro toby blanco
3 buitre vulture negro
4 caballo bojack gris

```

Con respecto al comando «`wc`» (*word count*), se suele utilizar para contar el número de caracteres y/o líneas que presenta un archivo. Cuando se utiliza por defecto (sin argumentos de entrada), cuenta los saltos de línea, palabras y número de bytes del archivo de entrada e imprime este valor en pantalla. Para la manipulación de archivos, generalmente trabajaremos con este comando pidiéndole el número de líneas que tiene un archivo, y para ello se le debe especificar con el argumento de entrada `-l (lines)`:

```

1 > wc -l animales.txt
2 4

```

#### 0.4.4 Comandos sort, uniq y du

Estos tres comandos sirven para ordenar («sort»), obtener elementos únicos («uniq») y comprobar el tamaño de archivos y/o directorios («du»).

El comando «sort» sirve para ordenar cadenas de texto dentro de un archivo. El ordenamiento puede ser tanto de cadenas de caracteres como numéricas, o la combinación de mismas. Sólo hay que especificarle mediante argumentos de entrada al comando el tipo de ordenamiento que se quiere llevar a cabo. Por ejemplo, si queremos ordenar el archivo «animales.txt», el comando «sort» tomará como referencia la primera columna del archivo, específicamente la primera letra de cada palabra por cada fila, y ordenará en base a ello:

```

1 > sort animales.txt
2 animal nombre color
3 buitre hawk negro
4 caballo bojack gris
5 perro toby blanco

```

El comando «uniq» devuelve el número de elementos únicos de una lista. En este caso, compara las filas de un archivo, y si son exactamente iguales, omite los elementos repetidos y devuelve en pantalla las filas únicas.

```

1 > printf "animal\tnombre\tcolor\nperro\ttoby\tblanco\nbuitre\thawk\tnegro\
2 ncaballo\tbojack\tgris\ncaballo\tbojack\tgris\n" > animales.txt
3 > cat animales.txt
4 animal nombre color
5 perro toby blanco
6 buitre hawk negro
7 caballo bojack gris
8 caballo bojack gris
9
10 > uniq animales.txt
11 animal nombre color
12 perro toby blanco
13 buitre hawk negro
14 caballo bojack gris

```

Por su parte, el comando du sirve para calcular el espacio que ocupa un archivo. Esto es importante a la hora de trabajar con archivos de gran volumen. Generalmente se emplea con el argumento de entrada -sh (*summarize, human-readable*), que devuelve un resumen de los cálculos hechos por el comando en un formato legible para el ser humano. Ejecutando este comando con dichos argumentos, se devolvería el tamaño de archivo en las unidades de bytes:

```

1 > du -sh animales.txt
2 4,0K animales.txt

```

#### 0.4.5 Otros comandos de interés: tree, ln, awk, gzip, find

Otros comandos que pueden resultar útiles a la hora de manejar la línea de comandos son:

- «quota» — devuelve una tabla con los archivos y espacio en disco ocupado para las distintas particiones.
- «count\_files» — devuelve los directorios con mayor conteo de archivos.
- «scontrol show nodes» — lista los nodos de Picasso y su estado.

```
lhurtado@picasso:~> tree folder/
folder/
└── subfolder
    ├── file_1
    ├── file_2
    └── subfolder_2
        └── file_3
```

Figura 9: Ejemplo de uso del comando tree para visualizar los contenidos de un directorio.

- «scontrol show reservation» — lista los bloqueos de nodo.
- «tree» — imprime en pantalla un 'árbol' con todos los contenidos de la carpeta que se especifique
- «ln -s» — sirve para crear enlaces simbólicos a un archivo o carpeta, los cuales son una referencia cruzada que se dirige a un archivo original mediante una ruta de referencia.

```
1 > ln -s $source_folder/file $output_folder/linked_file
2
```

- «awk» — sirve para buscar palabras y patrones de palabras y reemplazarlos por otras palabras y/o patrones; hacer operaciones matemáticas; procesar texto y mostrar líneas o columnas que cumplan determinadas condiciones, entre otros. En general, permite procesar y modificar el texto en base a las necesidades. Algunos ejemplos son:

```
1 1. Para imprimir una columna (delimitador por defecto: espacio)
2 > cat file | awk '{print $num_columna}'
3 2. Cambiar delimitador:
4 > cat file | awk -F "delim" '{print $num_columna}'
5 3. Extraer varias columnas de texto a la vez, especificando en
6 "delim" el delimitador:
7 > cat file | awk '{print $num_columna_1"delim"$num_columna_2"
8     "delim"$num_columna_3}'
9 4. Imprimir lineas con un determinado patron:
> file | awk '/expresion/ {print}'
```

- «gzip» — sirve para comprimir archivos en formato .gz, siendo su sintaxis:

```
1 Comprime el archivo y elimina el original:
2 > gzip filename
3 Si queremos quedarnos con el original:
4 > gzip -k filename
5 Para varios archivos:
6 > gzip file1 file2 file3
```

```
7     Para comprimir todos los archivos en un directorio:  
8     > gzip -r directory  
9     Para descomprimir un archivo .gz:  
10    > gzip -d filename.gz o gunzip filename.gz  
11
```

- «find» — sirve para encontrar un archivo o directorio en el sistema. Al utilizar este comando, se empieza una búsqueda recursiva, ya sea en un directorio concreto o en el sistema. El argumento más utilizado es -name, especificando el nombre del archivo.

```
1     Esquema general: > find <ruta> <argumentos>  
2     Para buscar un archivo concreto en el directorio actual:  
3     > find . -name "nombre_archivo"  
4     Para buscar en tu carpeta de usuario:  
5     > find ~ <argumentos>  
6     Para buscar en el sistema entero:  
7     > find / <argumentos>  
8
```

#### 0.4.6 Edición de archivos con vim

El editor de texto Vim (también conocido como vi) permite abrir archivos y modificar su contenido. Al ejecutarlo, abre una ventana dentro de la terminal a modo de editor de textos tipo Word pero mucho más simplificado (figura 10). Permite modificar archivos existentes o crear archivos desde cero. Para su manejo, es necesario conocer cómo funciona con una serie de comandos.

##### WARNING: Precauciones a la hora de modificar archivos

Los comandos de edición de archivos son un tanto complejos y deben ser empleados con cuidado, ya que las modificaciones pueden estar sujetas a errores si no se efectúan con precaución. Esto no quiere decir que no se deban utilizar estos comandos, sino que se preste atención antes de guardar cambios.

```
> vim animales.txt
```

Vim ofrece el modo de llamada de comandos y el de edición de caracteres. En el primer modo, se pueden llamar a una serie de comandos que actúan sobre el texto pero sin insertar caracteres uno a uno como en el segundo modo, de edición. Se puede borrar líneas de texto(*delete*, d), copiar (y), pegar (*paste*, p), deshacer cambios (*undo*, u), guardar (*write*, w), insertar saltos de línea (o), salir (*quit*, q) y salir sin guardar cambios (q!). El modo de edición de caracteres se activa mediante los comandos i (insertar) y a (añadir).

1. **i:** insertar texto. Entra en el modo edición de texto en la posición del texto que se haya seleccionado. Para salir del modo edición, pulsamos la tecla escape (ESC).
2. **a:** insertar texto justo a continuación del carácter especificado a modificar. Para salir del modo edición, pulsamos la tecla escape (ESC).
3. **u:** deshacer. Cuando pulsamos esta tecla en el modo de llamada de comandos, se deshacen los cambios llevados a cabo en el archivo.
4. **d:** eliminar. Pulsar dos veces la tecla d para eliminar una fila del texto en el modo de llamada de comandos.

Figura 10: Ventana abierta con el editor de textos Vim en Ubuntu 18.04 sobre el archivo «animales.txt». El contenido del archivo se muestra arriba, y abajo aparece el nombre del archivo, el total de líneas y el número de caracteres que incluye.

5. **y**: copiar. Pulsar dos veces la tecla y para copiar una fila del texto en el modo de llamada de comandos.
  6. **p**: pegar. Pulsar la tecla p para pegar una fila del texto (previamente copiada pulsando dos veces la tecla y) en el modo de llamada de comandos.
  7. **w**: guardar cambios. Para guardar cambios, desde el modo de llamada de línea de comandos, hay que abrir un submenú escribiendo dos puntos y w (:w). De esta manera se guardan los cambios efectuados. Si se pulsa la tecla w sin abrir este submenú, el puntero salta de una columna a otro dentro del archivo.
  8. **q**: salir. Al igual que en el anterior comando, hay que abrir un submenú escribiendo dos puntos y q (:q). Si se quiere salir del archivo sin guardar cambios, hay que añadir un símbolo de exclamación al final (:q!).
  9. **set list**: mostrar la lista de caracteres especiales dentro de un archivo (tabulaciones, saltos de línea).

## Ejercicio: Editar archivos con Vim

Con los comandos enseñados hasta el momento, incluye en el archivo «animales.txt» una nueva fila con un animal, su nombre y su color, con las columnas separadas por tabulaciones. A continuación, sal del modo edición y elimina la segunda fila, copia la tercera y pégala al final del archivo dos veces. Deshaz el último cambio, guarda todos los cambios y sal del archivo.

Vim también nos puede servir para ver caracteres especiales dentro de un archivo. Para ello,

podemos habilitar el modo «set list», que lista el contenido de todos los caracteres especiales. En la figura 11 se puede ver cómo trabaja este comando. Es un comando interesante de usar, ya que a veces si manipulamos un archivo de manera manual (por ejemplo, copiando y pegando contenido) se pueden sustituir tabulaciones por espacios, haciendo que algunos programas dejen de funcionar correctamente.

The screenshot shows a terminal window with the Vim editor running. The command ':set list' has been entered at the bottom. The output lists various characters and their representations: sample^Itreat^Igroup\$, CTL\_1^ICtrl^Ictrl\$, CTL\_2^ICtrl^Ictrl\$, CTL\_3^ICtrl^Ictrl\$, CTL\_4^ICtrl^Ictrl\$, CTL\_5^ICtrl^Ictrl\$, CTL\_6^ICtrl^Ictrl\$, CTL\_7^ICtrl^Ictrl\$, COVID\_1^ITreat^Itreat\$, COVID\_2^ITreat^Itreat\$, COVID\_3^ITreat^Itreat\$, COVID\_4^ITreat^Itreat\$, COVID\_5^ITreat^Itreat\$, COVID\_6^ITreat^Itreat\$, COVID\_7^ITreat^Itreat\$. Below these, there are seven tilde (~) characters, each preceded by a tab character. At the very bottom, the command ':set list' is visible again, with the cursor positioned after it.

Figura 11: Resultado de la ejecución del comando «set list» dentro de la consola de Vim en un archivo determinado. Las tabulaciones vienen definidas como «^ I» y los saltos de línea como «\$».

## 0.5 Herramientas e interacciones

Ecosistema de herramientas para proyectos de redes y enfermedades<sup>5</sup>



Ejemplo de una anotación

<sup>5</sup>Ejemplo de nota al pie de página

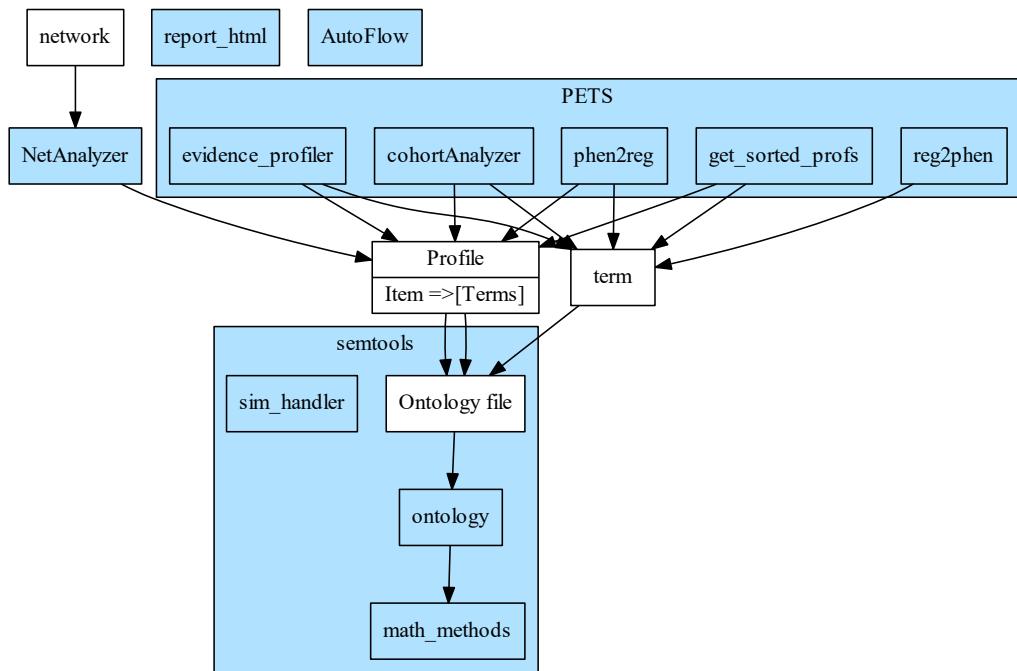
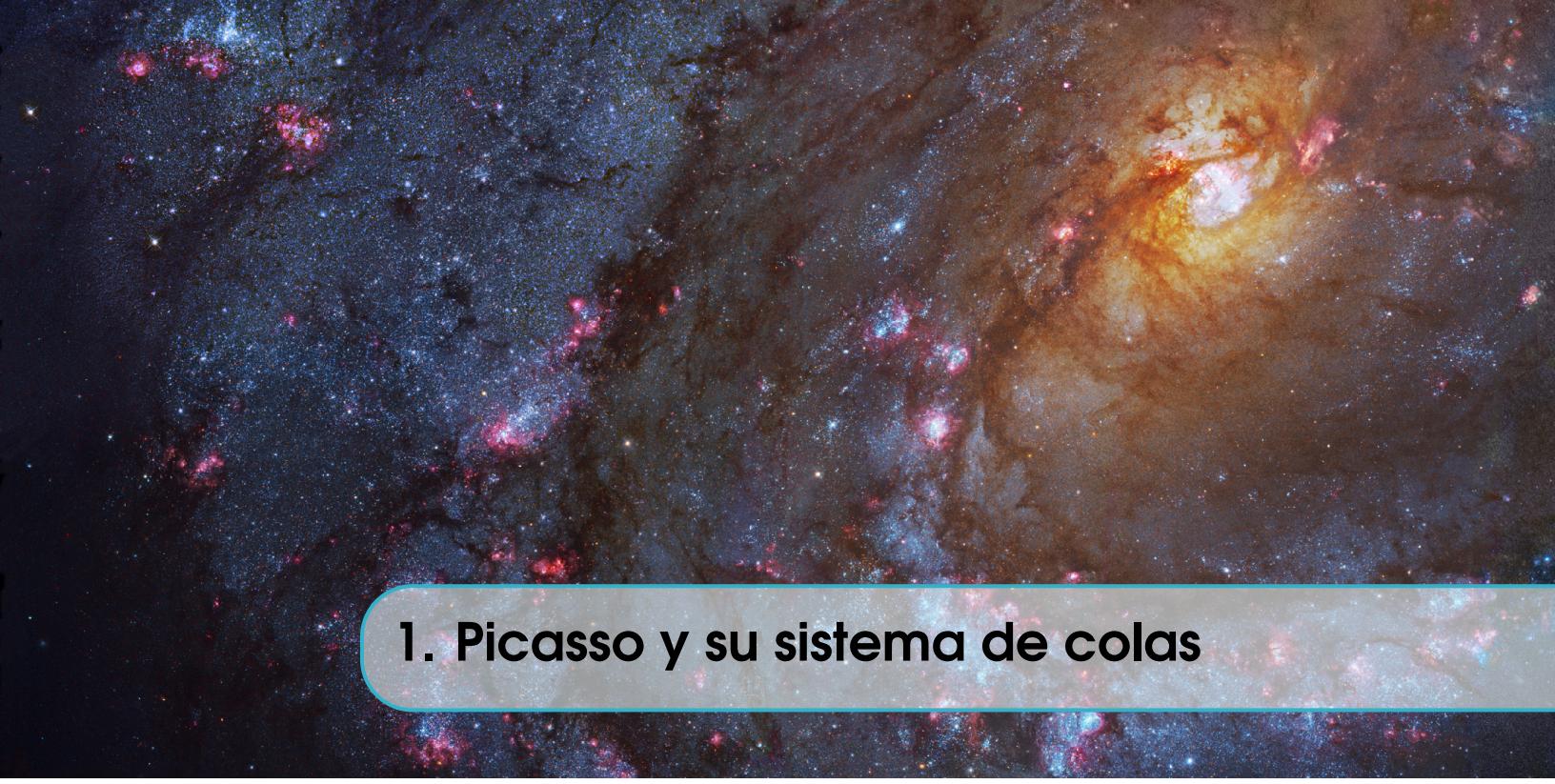


Figura 12: Herramientas desarrolladas para la rama de trabajo del laboratorio centrada en redes, fenotipos y enfermedades

```

1 #this is an example of bash code
2 var="test"
3
4 export $var
5 echo $var

```



## 1. Picasso y su sistema de colas

### 1.1 Carga de software

1.1.1 Software de Picasso

1.1.2 Cargar software de soft\_bio\_267

### 1.2 Sistema de colas

1.2.1 Enviar trabajos

1.2.2 Controlar la ejecución



## 2. Instalación de herramientas

La siguiente parte de este manual se centra en la instalación de las herramientas necesarias para la ejecución del flujo de análisis de expresión basado en la herramienta ExpHunter Suite. Es esencial comprobar que todas las instalaciones con los métodos que se indican se realizan satisfactoriamente.

### 2.0.1 Instalación en Picasso - Grupo BIO267

En el caso de contar con cuenta en Picasso dentro del grupo bio\_267 no es preciso instalar todas las herramientas, sino que se dispone de ellas a partir de los initializes disponibles en la cuenta soft\_bio\_267. Por ejemplo, para cargar todas las dependencias para trabajar con la herramienta ExpHunter Suite, se debe proceder de la siguiente manera:

```
1 source ~soft_bio_267/initializes/init_degenes_hunter
```

### 2.0.2 Instalación y configuración del flujo en otros sistemas

Los pasos a seguir se detallan a continuación:

#### Creación de carpetas de software

Todo el software que se va a descargar e instalar en los siguientes pasos va a ser guardado en la carpeta «software»:

#### Nombre de usuario

Recuerda que tu nombre de usuario en el sistema aparecerá en este bloque del manual como usuario. Asegúrate de cambiarlo por tu nombre de usuario cuando vayas a ejecutar los comandos. Es equivalente a la variable \$HOME.

```
1 > mkdir -p ~usuario/software  
2 > cd ~usuario/software
```

### Creación del sistema initializes

Los programas que se instalen en local van a seguir un sistema de carga mediante *initializes*. Para ello, lo primero que vamos a hacer es crear una carpeta llamada *initializes*. Los *initializes* o *init* son unos archivos de texto que cargan el *environment* y dependencias de un software concreto. Se cargan con el comando «source» o su alias en linux «.». Los *inits* se van a guardar en la carpeta usuario/software/*initializes* y van a seguir la nomenclatura "init\_*#nombre\_del\_programa*".

```
> mkdir -p ~usuario/software/initializes
```

#### Englobar tus *initializes* en uno

En algunas ocasiones, puede ser tedioso tener que llamar a cada uno de los *initializes* para cargar los programas que vamos a usar. Para ello, a veces se suele usar un sistema que englobe todos los *initializes* en uno. Para ello, puedes crear un script que llames, por ejemplo, *global\_init*, e incluir línea a línea un source llamando a los *initializes* necesarios:

```
source usuario/software/initializes/init_ruby  
source usuario/software/initializes/init_sysbiolab_scripts  
source usuario/software/initializes/init_sraToolkit
```

**Recuerda** que cada vez que quieras disponer de tus herramientas, debes hacer un source o al initialize que te interese o a tu *global\_init* o los programas que quieras usar no estarán disponibles.

### Dependencias y utilidades del flujo de trabajo DEG

Para comprobar que los programas estén instalados, en la mayoría de los casos basta con ejecutar la ayuda del programa con el argumento «-h» o «--help». Por ejemplo, para comprobar la instalación de «R», habría que ejecutar «R -h». Se debería ver un desglose de los argumentos que acepta el programa y la función de los mismos.

Curl, R, Perl y git deberían estar instalados en el clúster. Antes de continuar, confirmar esto empleando el comando «module avail» dentro del clúster. En algunos casos, puede que la versión de R no esté disponible dentro del clúster y, para ello, tengas que instalar tu propia versión (para trabajar con ExpHunter Suite es necesaria la versión 4.0.3 de R).

### Descarga e instalación de R (clúster de supercomputación)

La descarga e instalación del intérprete de este lenguaje de programación parte de la web oficial de R. Aquí, tendrás que seguir los siguientes comandos para descargar, descomprimir, generar el *makefile*, compilar e instalar R. A continuación, se describen los comandos a ejecutar y posteriormente en qué consiste cada etapa. **Asegúrate** de que comprendes bien el procedimiento y que en los module de tu clúster no hay una instalación previa de una versión de R que puedas utilizar (mínimo R 4.0.0 en adelante).

### WARNING: Asegúrate:

- En algunos clústeres es necesario tener antes disponible un compilador para poder llevar a cabo la función «make».
- Para ello, puedes preguntar a tu administrador de sistemas por si necesitas ayuda para saber qué versión y cómo llamar al compilador.
- En el caso de Picasso, ya existe una versión de R que se puede cargar directamente para usar:
  - module load R/4.0.2
- En el caso de la UPO, para la instalación de versiones superiores de R 4.0.0, necesitas ejecutar previamente el comando:
  - module load GCC/4.9.2

```

1 > mkdir -p $HOME/software/R
2 > wget https://cran.r-project.org/src/base/R-4/R-4.0.3.tar.gz -O R-4.0.3.
   tar.gz
3 > tar -xvf R-4.0.3.tar.gz
4 > rm R-4.0.3.tar.gz
5 > cd R-4.0.3
6 > ./configure --prefix=$HOME/software/R/4.0.3 --enable-R-shlib
7 > make
8 > make install
9 > echo 'export PATH=$HOME/software/R/4.0.3/bin:$PATH' > $HOME/software/
   initializes/init_R.4.0.3

```

En primer lugar, creamos la carpeta donde vamos a guardar los archivos binarios de la versión de R que vamos a descargar. Esta versión, la 4.0.3, la descargaremos de la web de CRAN a través del archivo <https://cran.r-project.org/src/base/R-4/R-4.0.3.tar.gz>. Cuando tengamos el archivo comprimido, emplearemos el comando «tar» con los argumentos «-xvf» que permitirán la descompresión del archivo. Podemos eliminar el archivo una vez que se haya descomprimido. Ese archivo descomprimido consiste en una carpeta (R-4.0.3) a la que accederemos. En ella, si listamos su contenido, veremos que existe un archivo «configure». Este archivo será el que escanee el sistema y genere la receta de compilación que se guardará en el archivo «makefile» para que, cuando llevemos a cabo la compilación con el comando «make», se conozcan las características del entorno donde se va a llevar a cabo la compilación. El comando «configure» necesita la ruta donde se van a instalar finalmente los archivos compilados. En este caso, y para ser ordenados, lo instalaremos en una subcarpeta dentro de `usuario/software/R/4.0.3` (empleamos el número de versión por si más adelante queremos instalar otra versión más reciente de R, o incluso otra más antigua si es necesario). Cuando termina la generación del archivo «makefile» (tras ejecutar el «configure»), si se lista la carpeta R-4.0.3 aparecerán nuevos archivos que sirven para la compilación (entre ellos el «makefile»). Ahora, iniciamos la compilación con el comando «make», que usará la información almacenada en el «makefile» (este proceso puede tardar un tiempo en ejecutarse). Y con el comando «make install» instalaremos en la carpeta preestablecida con el comando `configure` la versión de R. Para terminar, exportaremos el PATH a la carpeta de instalación de R para tener disponible esta versión de R cuando la necesitemos.

**Recomendación:**

Recomendamos incluir una(s) línea(s) adicional(es) al archivo init\_R.4.0.3 para que siempre pueda tener activas los enlaces al compilador (esto puede requerirse para la instalación de ciertas librerías o paquetes de R). Para ello, abre el archivo init\_R.4.0.3 con Vim o Nano e introduce los comandos «module load» a las librerías que has empleado para compilar R. Más información sobre compiladores al inicio de la sección 2.0.2.

**Descarga e instalación de R (equipo local)**

La descarga e instalación del intérprete de este lenguaje de programación ejecutada en un SO Ubuntu 18.04 LTS. Si partes de otro SO puedes seguir las instrucciones de este enlace.

```

1 > sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
   E298A3A825C0D65DFD57CBB651716619E084DAB9
2 > sudo apt-add-repository "deb https://cloud.r-project.org/bin/linux/
   ubuntu bionic-cran40/"
3 > mkdir $HOME/software/R/last
4 > echo "export R_LIBS=$HOME/software/R/last" >> $HOME/.bashrc
5 > source $HOME/.bashrc
6 > sudo apt-get update
7 > sudo apt-get install r-base --yes

```

Para confirmar que la instalación se ha llevado a cabo satisfactoriamente, hay que llevar a cabo el paso explicado anteriormente, escribiendo «R -help». También se puede escribir en la línea de comandos «R», el cual abre el REPL (*read-eval-print loop*) de R si ha sido instalado correctamente.

**Descarga e instalación de dependencias necesarias de R**

Algunas dependencias deben instalarse manualmente para poder llevar a cabo los análisis de expresión diferencial que se explican en este manual. Para ello, nos aseguramos de cargar la última versión de R que hemos instalado y ejecutamos los siguientes comandos:

```

1 install.packages("R.utils")
2 install.packages("tibble")
3 install.packages("canvasXpress")
4 install.packages("Rmisc")

```

**Descarga e instalación de Ruby**

Para descargar e instalar el intérprete del lenguaje de programación Ruby:

```

1 > mkdir -p $HOME/software/rubies/last
2 > wget https://cache.ruby-lang.org/pub/ruby/stable-snapshot.tar.gz
3 > tar -xvf stable-snapshot.tar.gz
4 > cd stable-snapshot
5 > ./configure --prefix=$HOME/software/rubies/last
6 > make
7 > make install
8 > echo 'export PATH=$HOME/software/rubies/last/bin:$PATH' > $HOME/software
   /initializes/init_ruby

```

Para confirmar que la instalación se ha llevado a cabo satisfactoriamente, se puede o bien escribir «ruby -help», que desglosa la ayuda del intérprete, o el comando «irb», el cual abre el REPL de Ruby si ha sido instalado correctamente.

## Descarga e instalación de gemas (librerías) de Ruby

Al igual que en R, hay que instalar dependencias manualmente. Las de Ruby son las siguientes:

### zip-zip

Para instalar esta gema de Ruby, hay que ejecutar los siguientes comandos:

```
1 > source ~usuario/software/initializes/init_ruby
2 > gem install zip-zip
```

## Autoflow

Para descargar AutoFlow, el gestor de flujos de trabajo necesario para ejecutar el análisis de expresión, hay que instalar previamente el intérprete de Ruby. Una vez hecho, escribir los siguientes comandos en la terminal:

```
1 > source $HOME/software/initializes/init_ruby
2 > gem install autoflow
```

Con el fin de comprobar si la instalación se ha realizado satisfactoriamente, se debe crear un archivo de texto llamado «test\_template.af» que incluya lo siguiente:

```
1 test_node_1{
2 ?
3 echo "test" > echo_file
4 }
5 test_node_2{
6 ?
7 cat test_node_1)/echo_file > final_file
8 }
```

A continuación, se debe ejecutar el siguiente comando:

```
> AutoFlow -w test_template.af -v -o ./exec
```

Para confirmar que la ejecución de la plantilla se ha llevado a cabo satisfactoriamente con AutoFlow, hay que ver si se ha creado una carpeta llamada «exec» con dos subcarpetas: «echo» y «cat».

## Descarga e instalación de los scripts del grupo SysBioLab

A partir de un repositorio en GitHub se pueden descargar los scripts del grupo SysBioLab, algunos de los cuales son necesarios para la ejecución del flujo de análisis. Para ello, se deben ejecutar los siguientes comandos:

```
1 > git clone https://github.com/seoanezonjic/sys_bio_lab_scripts.git
2 > echo 'export PATH=$HOME/software/sys_bio_lab_scripts:$PATH' > $HOME/
   software/initializes/init_sysbiolab_scripts
```

### ¡IMPORTANTE! Sobre las comillas:

En este caso, para ejecutar el contenido que hay entre las comillas rectas y que el resultado lo devuelva el comando echo, las comillas son las que aparecen en el teclado español *qwerty* en la misma tecla que el signo de cierre de interrogación. Ver Figura 2.1.



Figura 2.1: El círculo rojo señala el tipo de comilla usada en bash para interpretar y ejecutar el código incrustado entre ellas.

### Descarga e instalación de SeqtrimBB, bases de datos y creación de índices

SeqtrimBB es el programa de pre-procesado de lecturas. Necesita una serie de dependencias para ser ejecutado correctamente. Para ello, hay que lanzar los siguientes comandos (**Nota:** recuerda que los comandos que hay detrás de las almohadillas no hay que ejecutarlos ya que son comentarios que facilitan la comprensión de las ejecuciones que se están llevando a cabo):

```

1 ## FASTQC ## Sequencing samples quality analyzer
2 > cd $HOME/software
3 > wget "https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
4   fastqc_v0.11.8.zip"
5 > unzip "fastqc_v0.11.8.zip"
6 > rm fastqc_v0.11.8.zip
7 > chmod 544 $HOME/software/FastQC/fastqc
8 > echo 'export PATH=$HOME/software/FastQC:$PATH' > $HOME/software/
9   initializes/init_fastqc
10 ## BBmap ## Mapping Suite required for SeqtrimBB
11 ## It also needs the version 13 of oracle java (oracle-java13-installer &
12 > wget "http://sourceforge.net/projects/bbmap/files/latest/download/
13   BBMap_38.76.tar.gz"
14 > tar -xvf BBMap*
15 > rm BBMap*.tar.gz

```

Una vez instaladas las dependencias, se descarga la versión de SeqtrimBB siguiendo los comandos que se especifican a continuación.

```

1 > source $HOME/software/initializes/init_ruby
2 > gem install bundle

```

Pedir la versión compilada de la gema a los autores del manuscrito (seqtrimbb-2.1.6.gem) mediante petición por correo electrónico. Creamos la carpeta usuario/software/seqtrimbb y dentro guardamos la gema.

```

1 > gem install seqtrimbb-2.1.6.gem
2 > echo 'export PATH=$HOME/software/FastQC:$HOME/software/bbmap:$PATH' >
   $HOME/software/initializes/init_seqtrimbb

```

Además, hay que descargar las bases de datos dentro de la carpeta `usuario/software/seqtrimbb/DB`. Para ello, escribir en la terminal:

```

1 > cd $HOME/software/seqtrimbb
2 > mkdir DB
3 > git clone https://github.com/rafnunser/seqtrimbb-databases.git
4 > mv seqtrimbb-databases $HOME/software/seqtrimbb/DB
5 > echo 'export BBDB=$HOME/software/seqtrimbb/DB' >> $HOME/software/
   initializes/init_seqtrimbb
6 > echo 'export RAMDB=16384' >> $HOME/software/initializes/init_seqtrimbb

```

El último parámetro (`export RAMDB=16384`) sirve para definir memoria fija con el fin de indexar las bases de datos de SeqtrimBB ya descargadas. Se recomienda usar 16GB (debe escribirse en MB). Para confirmar que todo el copiado de carpetas y la descarga ha sido satisfactoria, antes de generar los índices que se explican a continuación, se deben listar las carpetas contenidas en «`l usuario/software/seqtrimbb/DB/fastas`». Debe haber 10 carpetas. Finalmente hay que crear un script llamado «`index_databases.sh`», con el siguiente contenido (**Nota**: en este caso SÍ hay que escribir las almohadillas # dentro del archivo para que se interpreten por el sistema de colas cuando se ejecute el comando, en caso de que sea necesario):

```

1#!/usr/bin/env bash
2#SBATCH --cpus=4
3#SBATCH --mem='17gb'
4source $HOME/software/initializes/init_seqtrimbb
5seqtrimbb -w 4

```

Una vez creado, hay que darle permisos de ejecución («`chmod 755`») y ejecutarlo en el **sistema de colas** empleando el comando «`sbatch index_databases.sh`». Para confirmar que el procedimiento está corriendo en el sistema de colas, ejecutar el comando «`sbatch -u nombreUsuario`». La ejecución en el sistema de colas devolverá un archivo `slurm-#.out`, siendo # el número de ejecución.

### Descarga e instalación de herramientas de mapeo

En primer lugar se instalará la herramienta STAR, que mapea lecturas de RNA-seq con un genoma de referencia. Para ello, ejecutar los siguientes comandos en la terminal:

```

1 > cd $HOME/software
2 > wget "https://github.com/alexdobin/STAR/archive/2.7.3a.tar.gz"
3 > tar -xzf 2.7.3a.tar.gz
4 > mv STAR-2.7.3a STAR
5 > cd $HOME/software/STAR/source
6 > make STAR
7 > cd $HOME/software
8 > echo 'export PATH=$HOME/software/STAR/source:$PATH' > $HOME/software/
   initializes/init_star

```

También se instalará la herramienta Bowtie, un mapper de secuencias específicamente diseñado para trabajar con lecturas de pequeño tamaño (30 pares de bases, aproximadamente). Para llevar a cabo su descarga e instalación, hay que ejecutar los siguientes comandos:

```

1 > cd $HOME/software
2 > git clone https://github.com/BenLangmead/bowtie.git
3 > cd ~usuario/software/bowtie
4 > make
5 > echo 'export PATH=$HOME/software/bowtie:$PATH' > $HOME/software/
   initializes/init_bowtie

```

Puede que se tenga que instalar la librería «libtbb-dev». En tal caso, leer las instrucciones dadas en <https://packages.ubuntu.com/source/bionic/bowtie>.

Asimismo, se debe instalar la herramienta Bowtie2, que realiza la misma tarea que Bowtie pero para trabajar de forma más eficiente con lecturas de mayor tamaño (50-100 pares de bases)

```

1 > wget "https://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.3.5.1/
2   bowtie2-2.3.5.1-linux-x86_64.zip"
3 > unzip bowtie2-2.3.5.1-linux-x86_64.zip
4 > mv bowtie2-2.3.5.1-linux-x86_64 bowtie2
5 > echo 'export PATH=$HOME/software/bowtie2:$PATH' > $HOME/software/
6   initializes/init_bowtie2

```

La herramienta SAMTools para trabajar con archivos de mapeo debe ser también descargada e instalada para llevar a cabo el análisis de muestras de secuenciación. Para ello, hay que ejecutar los siguientes comandos:

```

1 > cd $HOME/software/
2 > wget "https://sourceforge.net/projects/samtools/files/samtools/1.10/
3   samtools-1.10.tar.bz2"
4 > tar -xvf samtools-1.10.tar.bz2
5 > cd samtools-1.10
6 > ./configure --prefix="$HOME/software/samtools"
7 > make
8 > make install

```

### ¡Recuerda! Sobre los comandos configure y make:

El comando configure genera el archivo makefile con la ruta donde se van a instalar los programas. El comando make compila y genera los binarios, y el make install copia los binarios en el path que corresponda.

Para analizar la calidad de los mapeos se debe descargar la herramienta Qualimap. La herramienta se descarga e instala siguiendo estos comandos:

```

1 > wget "https://bitbucket.org/kokonech/qualimap/downloads/qualimap_v2.2.1.
2   zip"
3 > mv qualimap_v2.2.1 qualimap
4 > echo 'export PATH=$HOME/software/qualimap:$PATH' > $HOME/software/
5   initializes/init_qualimap

```

En el caso de alinear nuestras muestras contra un genoma de referencia, la herramienta STAR realizaría el alineamiento y devolvería la tabla de conteo. Sin embargo, si alineamos frente a un transcriptoma (por ejemplo, cuando no se cuenta con un genoma de referencia) empleamos la herramienta Bowtie para realizar el alineamiento y Sam2counts para generar la tabla de conteo. En caso de alinear contra un transcriptoma de referencia, se puede descargar e instalar la herramienta Sam2counts, que convierte los resultados del mapeo (SAM) en un archivo delimitado por tabulaciones que contiene los recuentos de cada secuencia. La descarga e instalación de esta herramienta **no es un paso estrictamente necesario** ya que en el repositorio que previamente se descargó con herramientas del grupo SysBioLab existe un script llamado «sam2counts\_all.py» que realiza la misma función.

Sus dependencias se pueden instalar con los siguientes comandos en la terminal:

```

1 > sudo apt install python-pip --yes
2 > pip install Cython --install-option="--no-cython-compile"
3 > pip install pysam
4 > git clone https://github.com/vsbuffalo/sam2counts.git
5 > chmod 555 $HOME/software/sam2counts/sam2counts.py
6 > echo 'export PATH=$HOME/software/sam2counts:$PATH' > $HOME/software/
    initializes/init_sam2counts

```

Una vez instalados todos los programas y dependencias necesarias para llevar a cabo el análisis de limpieza de lecturas y alineamiento se pueden instalar los programas para el análisis de detección y/o expresión.

### Detección de miRNAs con miRDeep2

En el caso de llevar a cabo la determinación de micro-RNAs (miRNAs) a través de pequeñas librerías de RNA-seq, hay que instalar la herramienta miRDeep2 ejecutando los siguientes comandos:

```

1 > git clone https://github.com/rajewsky-lab/mirdeep2.git
2 > cd $HOME/software/mirdeep2
3 > echo -e '#!/usr/bin/env bash
4     export PATH=$HOME/software/bowtie:$HOME/software/mirdeep2/bin:$PATH
5     export PERL_MB_OPT="--install_base $HOME/software/perl5"
6     export PERL_MM_OPT="INSTALL_BASE=$HOME/software/perl5"
7     export PERL5LIB=$HOME/software/mirdeep2/lib/perl5:$PERL5LIB' >> $HOME/
        software/initializes/init_mirdeep2
8 > source $HOME/software/initializes/init_mirdeep2
9 > ./install.pl

```

### Análisis de expresión diferencial con ExpHunter Suite

ExpHunter Suite está incluida en el paquete de R ExpHunterSuite. Esta herramienta de análisis de expresión diferencial a partir de muestras de RNA-seq requiere una serie de dependencias (pandoc, aptitude, libcurl4-openssl-dev y libxml2-dev) que deben instalarse *a priori*. Antes de nada, **asegúrate de que en tu sesión (terminal) estás usando la versión de R indicada anteriormente (4.0.3)**. Esto es esencial ya que en caso contrario tendrás incompatibilidades que harán que el programa de análisis de expresión diferencial no funcione. Para la instalación de pandoc, se deben ejecutar los siguientes comandos:

```

1 cd $HOME/software
2 wget "https://github.com/jgm/pandoc/releases/download/2.9.2.1/pandoc
      -2.9.2.1-linux-amd64.tar.gz"
3 tar -xzvf pandoc-2.9.2.1
4 mv pandoc-2.9.2.1 pandoc
5 echo 'export PATH=$PATH:$HOME/software/pandoc/bin' > $HOME/software/
    initializes/init_pandoc

```

Y para instalar aptitude, libcurl4-openssl-dev y libxml2-dev hay que usar el gestor de paquetes del sistema operativo correspondiente.

Asegúrate en primer lugar que tienes en tu sistema un módulo donde se llame a la librería XML. Por ejemplo, en muchos clústeres (como Picasso o UPO) aparece dentro del módulo libxml2/2.9.7-GCCcore-6.4.0 (más información al principio de la sección 2.0.2). Para utilizarlo:

```

1 echo 'module load libxml2/2.9.7-GCCcore-6.4.0 > $HOME/software/initializes
      /init_xml'
2 source $HOME/software/initializes/init_xml

```

Una vez hecho esto, comenzaremos la descarga e instalación de ExpHunter Suite. Para ello, seguiremos las instrucciones provistas en la web <https://github.com/seoanezonjic/ExpHunterSuite>.

En primer lugar, y una vez que nos hemos asegurado de tener una versión de R superior a la 4.0.0, así como los compiladores y dependencias requeridas *a priori*, como pandoc y el módulo libxml2/2.9.7-GCCcore-6.4.0 activo (asegúrate ejecutando en tu terminal el comando «module list» y viendo que aparece dicho módulo en la lista), empezaremos la instalación de ExpHunter Suite.

La instalación requiere la herramienta devtools. Para ello, hay que ejecutar el siguiente comando en la terminal:

```
Rscript -e 'install.packages("devtools", repos="http://cran.us.r-project.org")' > log
```

Esta utilidad sirve para descargar paquetes de R desde GitHub. Para asegurarnos de que no haya fallos, redireccionaremos la salida en pantalla de toda la ejecución en un archivo log, que nos servirá para detectar si existen errores al instalar la herramienta.

Para comprobar cualquier tipo de error durante la instalación de devtools (lo cual puede suceder en el caso de emplear una nueva instalación de R), buscaremos en qué líneas ha habido error en el archivo log:

```
grep "ERROR:" log
```

En pantalla, se imprimirá (en caso de que los haya) los errores de instalación de dependencias. En la Figura 2.2 se muestra un ejemplo de errores:

```
[ R]$ grep 'ERROR:' log
ERROR: loading failed
ERROR: dependency 'stringi' is not available for package 'stringr'
ERROR: loading failed
ERROR: dependency 'stringr' is not available for package 'knitr'
ERROR: dependency 'openssl' is not available for package 'credentials'
ERROR: dependency 'openssl' is not available for package 'httr'
ERROR: dependencies 'credentials', 'openssl' are not available for package 'gert'
ERROR: dependency 'httr' is not available for package 'gh'
ERROR: dependency 'httr' is not available for package 'covr'
ERROR: dependencies 'gert', 'gh' are not available for package 'usethis'
ERROR: dependencies 'knitr', 'stringi', 'stringr' are not available for package 'roxygen2'
ERROR: dependencies 'usethis', 'covr', 'httr', 'roxygen2' are not available for package 'devtools'
```

Figura 2.2: Errores de ejecución en la instalación de devtools. En cuadrados rojos se marcan dos de los paquetes de los cuales dependen los demás. Obsérvese que es como un dominó: si la dependencia stringi falla, el paquete stringr no estará disponible, y por ende tampoco se podrá instalar correctamente knitr, y así sucesivamente. Lo mismo sucede con la librería openssl, por lo tanto tendremos que ver la manera de instalar estas librerías manualmente.

### Instalación manual de librerías

En el caso de que existan librerías difíciles de instalar, tenemos varias opciones:

- Recurrir a la instalación manual entrando en la consola de R (escribimos R y pulsamos Enter) y escribiendo el comando «install.packages("nombre\_librería")». En el caso de querer instalar la dependencia stringi (CRAN) y la librería diffcoexp (de Bioconductor):

```
1 install.packages("stringi")
2 BiocManager::install("biomaRt")
3
```

(Automáticamente se abrirá una ventana donde tendremos que seleccionar el servidor de descarga, seleccionar la opción (0) *cloud* (<https://>)).

- Descargando el archivo comprimido de la librería desde su repositorio en CRAN (buscar en *Package source* la fuente al archivo tar.gz) para volver a abrir R y ejecutar el comando «`install.packages("nombre_archivo_comprimido")`». Esto resulta muy útil en el caso de que tengamos que instalar versiones previas de algunas librerías que tengan incompatibilidades con ciertas versiones de R. Por ejemplo, en el caso de querer descargar e instalar la última versión de stringi:

```

1 > cd $HOME/software
2 > wget https://cran.r-project.org/src/contrib/stringi_1.5.3.tar.gz
3 -O stringi_1.5.3.tar.gz
4 > R
5 > install.packages('stringi_1.5.3.tar.gz')

```

- En casos excepcionales donde no se puedan instalar por los dos métodos anteriores, un comando que sirve para instalar las librerías en R es, en la terminal (sin entrar en R) ejecutar:

```

1 > R CMD INSTALL nombre\_paquete.tar.gz --configure-args='--disable-
2   pkg-config'

```

Pondremos un ejemplo real de problema cuando se intenta instalar devtools, y es que en algunos casos devuelve el error documentado en la Figura 2.2.

```

----- ANTICONF -----
Configuration failed to find libgit2 library. Try installing:
* brew: libgit2 (MacOS)
* deb: libgit2-dev (Debian, Ubuntu, etc)
* rpm: libgit2-devel (Fedora, CentOS, RHEL)
If libgit2 is already installed, check that 'pkg-config' is in your
PATH and PKG_CONFIG_PATH contains a libgit2.pc file. If pkg-config
is unavailable you can set INCLUDE_DIR and LIB_DIR manually via:
R CMD INSTALL --configure-vars='INCLUDE_DIR=... LIB_DIR=...'
----- [ERROR MESSAGE] -----
<stdin>:1:18: error fatal: git2.h: No existe el fichero o el directorio
compilación terminada.
-----
```

Figura 2.3: Error en la instalación de devtools. En este caso, la librería usethis está dando problemas para instalarse. En concreto, la última versión. Para solucionar el problema, se debe descargar una versión compatible (en este caso, la 1.6.3).

Para solucionar este problema de incompatibilidad de versiones, tenemos que ir al listado de versiones de la librería usethis en CRAN y descargar el comprimido de la versión anterior a la que nos da problemas. Cuando la localicemos, ejecutamos los siguientes comandos:

```

1 > cd $HOME/software
2 > wget https://cran.r-project.org/src/contrib/Archive/usethis/usethis_1
3 .6.3.tar.gz
4 > R
5 > install.packages('usethis_1.6.3.tar.gz')

```

En algunos casos, esta librería puede devolver un error al no encontrar la dependencia git2r. El error sería algo similar a esto:

```
1 ERROR: dependency 'git2r' is not available for package 'usethis'
```

En este caso, lo que deberíamos hacer es instalar esa librería y posteriormente volver a instalar la librería usethis (en la consola de R):

```
1 > install.packages('git2r')
2 > install.packages('usethis_1.6.3.tar.gz')
```

Con todo ello, tendríamos instalado devtools y poder acceder al siguiente paso, que es instalar ExpHunter Suite.

### Instalación de ExpHunter Suite con devtools

El siguiente paso consiste en instalar el paquete ExpHunterSuite a través del propio repositorio de GitHub. Para ello, en la consola ejecutamos el siguiente código:

```
1 > cd $HOME/software
2 > git clone https://github.com/seoanezonjic/ExpHunterSuite.git
3 > echo 'export DEGHUNTER_MODE=DEVELOPMENT > $HOME/software/initializes/
4 init_ExpHunterSuite
> echo 'export PATH=$PATH:$HOME/software/ExpHunterSuite/inst/scripts' >>
$HOME/software/initializes/init_ExpHunterSuite
```

En el archivo initialize del ExpHunterSuite hay que poner variable de entorno (export DEGHUNTER\_MODE=DEVEL) para que entre en modo desarrollo y que importe las dependencias sin instalar el paquete.

Posteriormente, hay que instalar las dependencias del paquete y hacer un export de la carpeta en la que están los scripts (no instala el paquete pero sí las dependencias). Para instalar las dependencias:

```
1 > cd $HOME/software
2 > Rscript -e 'devtools::install_deps("$HOME/software/ExpHunterSuite",
dependencies=TRUE)"'
```

En algunos casos, la instalación de esta librería falla si no está instalada previamente la librería de biomaRt. Hay que descargar la versión anterior para que pueda funcionar. Para ello, escribimos en la terminal:

```
1 cd $HOME/software
2 wget https://bioconductor.org/packages/3.12/bioc/src/contrib/Archive/
biomaRt/biomaRt_2.46.0.tar.gz
3 R
4 install.packages('biomaRt_2.46.0.tar.gz')
5 devtools::install("ExpHunterSuite")
```

En el momento en el que nos pida actualizar librerías, **NO** las actualizamos, o volveremos a tener el fallo en la instalación de ExpHunterSuite.

#### Aprende de los errores:

Analiza todos los mensajes de error que pudieran estar presentes por si tienes que instalar otras versiones de las dependencias a mano.

Con todas estas dependencias y herramientas instaladas, ya tendremos nuestro sistema puesto a punto para realizar análisis de expresión diferencial de muestras de RNA-Seq. Este manual incluye

ejemplos prácticos que puedes llevar a cabo en la sección 12.1 y de un caso real en la sección 12.3. Asegúrate de leer muy bien lo que se va a llevar a cabo en la siguiente sección (12.1) antes de pasar a ver los ejemplos reales para comprender al detalle cómo se realiza el análisis de muestras con ExpHunter Suite.





## 3. GitHub

Git es un gestor de versiones muy empleado tanto en el mundo académico como empresarial. Lo más habitual, como es también nuestro caso, es usarla junto al servidor de repositorios (GitHub).

### 3.1 Gestión de versiones: conceptos fundamentales

#### 3.1.1 ¿Para qué sirve un gestor de versiones?

En desarrollo de software, el código va evolucionando continuamente. Puedes partir desde cero en un proyecto nuevo o partir desde otro que vas a modificar, pero lo que es seguro es que tu código va a cambiar a lo largo del tiempo.

Un gestor de versiones aporta dos cosas: la primera, un historial de los cambios realizados y por qué se han realizado. La segunda, una forma de volver al estado anterior del código. A medida que vayas ganando experiencia verás que tener estas opciones es fundamental.

#### 3.1.2 Repositorio local y remoto

En control de versiones, la base es el concepto de repositorio. Literalmente, es un lugar donde se guarda algo. Cuando hablamos de código, no es simplemente eso: es también un directorio donde se guarda todo lo relacionado al control de versiones de todo lo que contiene. Por tanto, es la estructura básica sobre la que funciona un gestor de versiones como git.

El repositorio local es un directorio en tu ordenador. Ahí irás haciendo los cambios, y tendrás el control absoluto sobre lo que pase en él. Sólo tiene una pega: sólo tendrá acceso a él quien tenga acceso a tu equipo.

Un repositorio remoto existe en un servidor externo, tal y como puede ser GitHub. Conectando los repositorios local y externo puedes enviar cambios de un sitio a otro, actualizándolos según haga falta, y manteniendo un historial de versiones común. Esto permite a cualquiera con acceso a ese servidor usar tu código, y también posibilita la colaboración entre distintos programadores sobre un mismo repositorio.

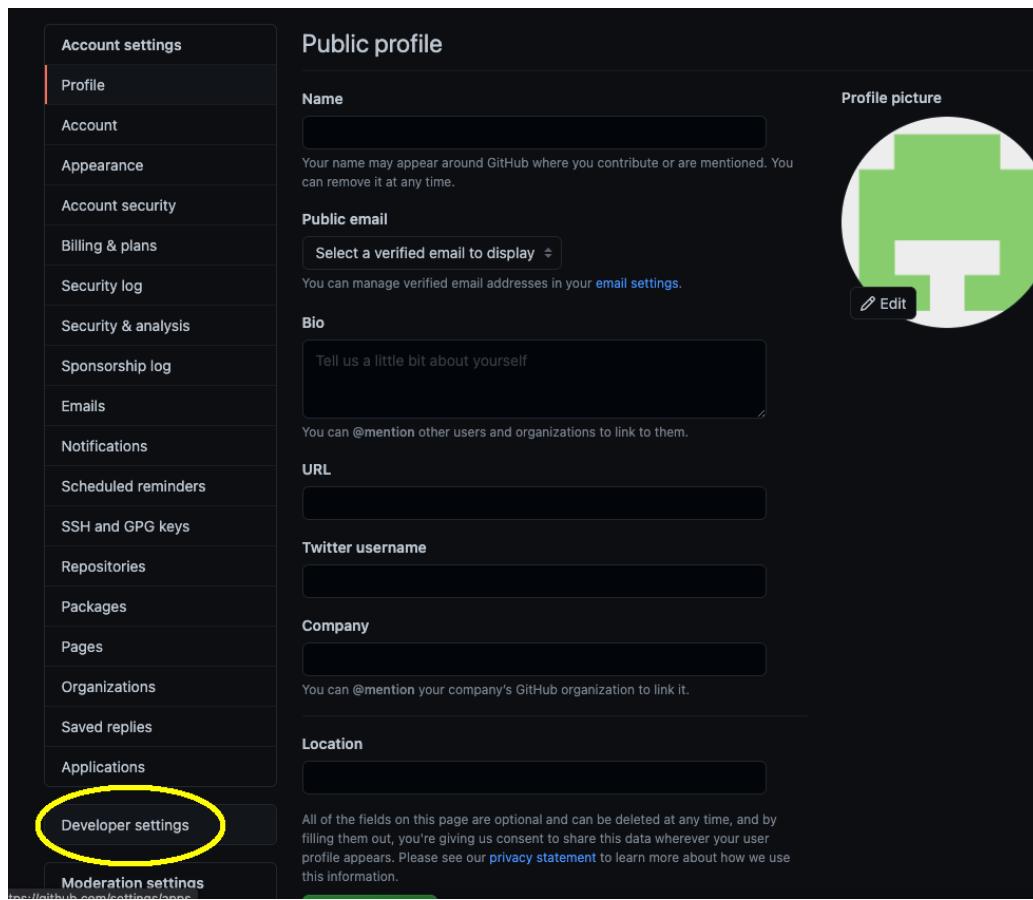
### 3.1.3 Ramas y bifurcaciones

Hablando de colaboraciones, es importante entender cómo se puede organizar un repositorio. Normalmente se va a dividir en «ramas», o «branches», en inglés. Cada una representa, realmente, lo que el programador quiera, pero lo más habitual es que representen etapas de desarrollo. La rama «main» o «master» es la principal, y lo suyo es que represente una versión del programa completamente funcional y lista para usarse. Ramas adicionales representan cambios que se quieren realizar sobre el programa, y se unirán (lo que en git se conoce como «Pull Request») a la principal en cuanto estén completadas y funcionen. Esto permite realizar cambios sobre el programa sin afectar a la versión que ya pueden estar usando otros, incluso colaboradores del repositorio (otros programadores que están trabajando sobre él).

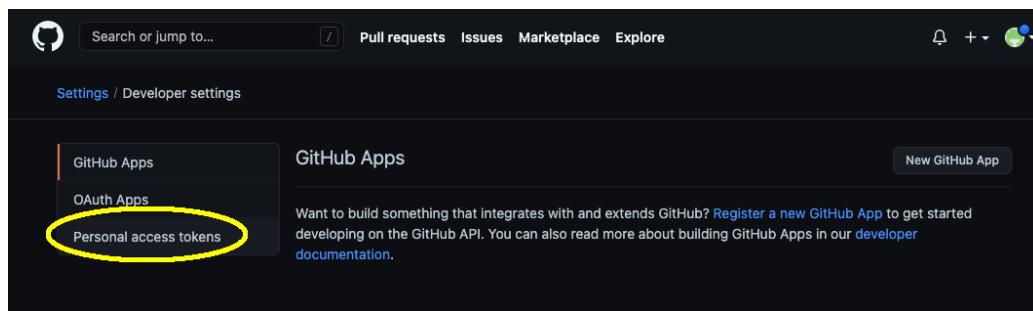
Existe un concepto parecido, que es el de bifurcación (o «fork», en inglés). Mientras que una rama es parte del repositorio original, la bifurcación es una copia que pertenece a otro programador, aunque mantiene su vinculación al repositorio original (estará indicado de qué repositorio e incluso de qué versión de él procede). Trabajar sobre una rama de un repositorio requiere ser un colaborador, o en otras palabras, que los dueños te den permiso para trabajar con ellos. Tratándose de otros grupos, raro es que te lo vayan a dar. Lo que sí es más fácil es que te den permiso para hacer una bifurcación (con habilitar una opción del repositorio se da permiso de bifurcación a todo el que lo deseé), y podrás trabajar en tu propia versión. También puedes solicitar a los dueños del repositorio original que incorporen tus cambios una vez consideres que estén listos (un «Pull Request»).

## 3.2 Creación de tokens para verificación

El 13 de agosto de 2021, Git cambió el sistema de verificación. Antes, era posible identificarse en la línea de comandos mediante usuario y contraseña para subir y descargar contenido de repositorios. Ahora no es posible, y es necesario crear un «Personal Access Token», o «PAT». El propio GitHub tiene un tutorial muy sencillo. Este token es una clave que tendremos que guardar, y será lo que usemos para identificarnos cuando queramos trabajar en un repositorio de GitHub. En primer lugar, accede con tus datos personales a tu cuenta de GitHub. Pulsa en tu ícono de usuario y aparecerá un menú lateral, donde debemos bajar hasta «settings». Deberías llegar a una página llamada «Account Settings». Ahora busca el apartado «Developer Settings».



Aquí encontrarás la pestaña «Personal Access Tokens», donde lo podrás generar.



Hay dos tipos de token: «Fine-grained» (más seguro) y «classic» (más funciones). Recomendamos el «classic», porque, entre otras cosas, permite la opción de que no caduque.

Click over `repo`, or can be used to authenticate to the API over Basic Authentication.

**Note**

`pat_for_BS_projects`

What's this token for?

**Expiration \***

Custom...  08/11/2022

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> <code>repo:status</code>	Access commit status
<input checked="" type="checkbox"/> <code>repo_deployment</code>	Access deployment status
<input checked="" type="checkbox"/> <code>public_repo</code>	Access public repositories
<input checked="" type="checkbox"/> <code>repo:invite</code>	Access repository invitations
<input checked="" type="checkbox"/> <code>security_events</code>	Read and write security events

Para generar

un token, hay que darle un nombre, seleccionar una fecha de caducidad (recomendamos «No expiration»), sólo disponible en «classic», y seleccionar el «Scope», que son los permisos que concederá el token al verificar la sesión en la línea de comandos. Recomendamos marcarlos todos.

What's this token for?

**Expiration \***

Custom...  dd/mm/yyyy

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> <code>repo:status</code>	Access commit status
<input checked="" type="checkbox"/> <code>repo_deployment</code>	Access deployment status
<input checked="" type="checkbox"/> <code>public_repo</code>	Access public repositories
<input checked="" type="checkbox"/> <code>repo:invite</code>	Access repository invitations
<input checked="" type="checkbox"/> <code>security_events</code>	Read and write security events
<input type="checkbox"/> <b>workflow</b>	Run workflows

November 2021 ▾

- 2018
- 2019
- 2020
- 2021
- 2022

Jan	Feb	Mar	Apr
May	Jun	Jul	Aug
Sep	Oct	Nov	Dec

Después de llenar esta información hay que hacer click en «generate token». Así, Github generará un PAT, un código que se puede usar en vez de tu contraseña en la línea de comandos.

**WARNING: GUARDAR EL TOKEN**

MUY IMPORTANTE. Hay que copiar el token y guardarla en un archivo de texto. Será la única forma de acceder a él después de haberlo generado. Guardalo en un sitio personal y seguro, esto sigue siendo una contraseña. **NO lo compartas con NADIE.**

Una vez que hayas generado el token, puedes usarlo cuando trabajes con Git y te pida tus credenciales a modo de contraseña.

```
1 git add archivo_de_ejemplo
2 git commit -m 'mensaje_de_commit'
3 git push origin rama
4 Username: your_username
5 Password: your_token
```

### 3.3 Claves SSH

Existe una manera de automatizar el proceso de verificación, ahorrando tener que introducir el token cada vez que se envía un cambio al repositorio remoto. Viene explicado en la web de GitHub, y aquí ponemos un resumen.

#### 3.3.1 Crear la clave

Para empezar, ejecuta lo siguiente en la terminal:

```
1 ssh-keygen -t ed25519 -C "tu_email@ejemplo.com"
```

Te pedirá un nombre para el archivo donde se guardará, recomendamos no poner nada para que use el archivo por defecto. Si vas a crear más de una clave, sí tendrás que nombrarlos por separado. También te pedirá una contraseña para cuando quieras abrir el archivo.

Esto creará tu clave, asociada al correo que hayas introducido. Debe ser uno de los que hayas asociado a tu cuenta de GitHub, de lo contrario no servirá para nada.

#### 3.3.2 Añadir la clave a tu cuenta

Primero hay que comprobar que el agente está funcionando. CUIDADO AQUÍ: no son comillas, son backticks (`):

```
1 eval `ssh-agent -s`
```

Te debería imprimir en consola «Agent pid XXXXXX», siendo XXXXXX un número distinto cada vez que lo ejecutes. Eso significa que el agente está funcionando. Si no devuelve nada, ejecuta:

```
1 Get-Service -Name ssh-agent | Set-Service -StartupType Manual
2 Start-Service ssh-agent
```

No debería ser necesario, ya que el agente se usa constantemente en Picasso, pero puede servirte si usas otro sistema. Ahora, para añadir la clave, ejecuta:

```
1 ssh-add ~/.ssh/id_ed25519
```

Esto asume que has guardado la clave en el archivo por defecto. Si no es así, modifica el path como sea necesario.

Ahora, tu gestor de ssh está al tanto de la clave, y falta comunicárselo a git y a GitHub.

Al crear la clave ssh, tendrás dos archivos: uno sin extensión («id\_ed25519» por defecto) y otro con extensión (mismo nombre pero acabado en «.pub»). Copia el contenido del .pub en el portapapeles, y vete a ajustes de tu cuenta de GitHub. Vete a **Access** y pincha en **SSH and GPG keys**, y selecciona **New SSH key**. En **Title** pon un nombre descriptivo («Clave Picasso», por ejemplo, si la has generado en el Picasso), y pega lo que has copiado en el portapapeles en **Key**. Asegúrate de que **Key type** marca **Authentication Key**.

### 3.3.3 Clave de firma

Esto te permitirá firmar los commits automáticamente. El proceso es exactamente el mismo que el anterior, así que, al añadir tu clave a GitHub, simplemente añade otra vez la misma llave, pero en este caso en **Key type** marca **Signing key**. La misma clave te servirá.

### 3.3.4 Añadir las claves a tu configuración de git

Esto es muy sencillo, son sólo tres comandos:

```
1 git config --global gpg.format ssh
2 git config --global user.signingkey ~/.ssh/id_ed25519.pub
3 git config --global commit.gpgsign true
```

Le estamos diciendo a git tres cosas: Primero, que nuestras claves de verificación y de firma están en formato ssh. Después, dónde está nuestra clave. Por último, que use esa clave para firmar los commits que hagamos. El flag **-global** le dice que se esto se aplique a todo nuestro git, pero también podemos sustituirlo por **-local** para que sólo afecte al repositorio en el que estamos trabajando.

### 3.3.5 Crear y enlazar el repositorio remoto

Crearlo es muy sencillo. Ve a la página de GitHub y hazte una cuenta si no la tienes ya. Si es el primer repositorio que creas, GitHub te guiará. Es importante que al crearlo marques "Private". Igual no hace falta, pero mejor pecar de mucho cuidado que de poco. Para enlazar el repositorio local al remoto, vete al local en la terminal. Introduce el siguiente comando:

```
1 git remote add origin git@github.com:USER/REPO.git
```

Donde USER es tu usuario de GitHub y REPO el nombre del repositorio remoto. Se puede añadir una URL, pero hacerlo de esta manera le permitirá reconocer las claves ssh de la configuración, y no tendrás que introducir tu usuario y token cada vez que quieras comunicarte con el repositorio remoto.

¡Y ya está listo! Ya le hemos dicho a GitHub qué son esas claves (verificación de nuestro usuario y firma de nuestros commits), así que al recibirlas desde git las reconoce. Ahora no debería pedirte las credenciales cuando quieras enviar o solicitar cambios del repositorio remoto, y todos tus commits estarán verificados (firmados).

## 3.4 Comandos de git

- **git init**. Este comando, ejecutado tal cual y sin argumentos, convierte el directorio actual en un repositorio de Git. Para añadir una dirección remota (y que no exista sólo en local), se debe

ejecutar

```
1 git remote add origin "git@github.com:USER/REPO.git"
```

Realmente no tiene por qué llamarse "origin", pero es lo más habitual y lo que recomendamos. "USER" debe ser tu usuario de "GitHub" y "REPO" el nombre del repositorio remoto. También puedes usar la URL de git, pero eso te impedirá aprovechar la ventaja de tener tus claves incluidas en la configuración de git (es decir, te pedirá tu usuario y tu token cada vez que quieras conectarte al repo remoto).

- **git clone.** Sirve para crear una copia local de un repositorio remoto ya existente.

```
1 git clone https://github.com/OWNER/REPOSITORY.git
```

Esto creará una copia local en la carpeta REPOSITORY. Sólo podremos enviar los cambios realizados si somos colaboradores del repositorio.

- **git branch.** Si se ejecuta sin argumentos, nos mostrará las ramas del repositorio, marcando en verde la rama actual. Si se le pasa un argumento, creará una rama con ese nombre.

```
1 git branch  
2 git branch NEW_BRANCH
```

Si ya existe una rama con ese nombre, el comando fallará.

- **git checkout.** Sirve para cambiar de rama. Si se añade el flag -b, el programa entenderá que esa rama no existe y deberá crearla antes de enviarnos a ella.

```
1 git checkout BRANCH  
2 git checkout -b NEW_BRANCH
```

El comando fallará si se usa con el flag -b con una rama existente.

- **git status.** Muestra los cambios realizados sobre el repositorio. Se ejecuta tal cual, sin argumentos.

- **git diff.** Este sí que lleva argumentos. Se ejecuta de la siguiente manera:

```
1 git diff FILE
```

Abrirá en la terminal el archivo modificado en modo lectura, mostrando en rojo las líneas eliminadas y en verde las nuevas (otra forma de entenderlo, mostrará en rojo lo antiguo y en verde lo nuevo). Muy útil para asegurarte de que no te has dejado nada por el camino (alguna línea que sea sólo para hacer pruebas y no quieras realmente añadir al repositorio).

- **git add.** Guarda el cambio realizado. Toma como argumento los archivos que deseas añadir, pudiendo incluso pasarle el repositorio entero de golpe.

```
1 git add FILE(s)
```

MUY recomendable hacerlo archivo por archivo. Es posible que quieras deshacer este comando. Es decir, no quieras borrar los cambios en el archivo, pero no quieras guardarlos.

ahora mismo (quizás quieras esperar a hacer antes otro «commit», que ya explicaremos qué es). En cualquier caso, la solución es sencilla:

```
1 git restore --staged ADDED_FILE(s)
```

Tu versión local del archivo no cambiará, pero Git dejará de tenerlo almacenado. Al hacer «git status», volverá a aparecerte como cambios no añadidos. En caso de que sí quieras deshacer estos cambios, lo usas sin el flag:

```
1 git restore FILE(s) _TO_RESTORE
```

Este comando vuelve a la última versión guardada (mediante «git commit») del archivo.

- **git stash.** Guarda de forma temporal los cambios para aplicarlos más tarde. Se usa como «git add». Para más detalles de cómo funciona, recomendamos esta guía
- **git commit.** Toma los cambios guardados y los almacena definitivamente (de ahí «commit», que en inglés significa «comprometer»). Se puede deshacer, pero es más complicado. Asegúrate antes de ejecutar el commit. Se puede usar tal cual, en cuyo caso abre un editor de texto para introducir el mensaje del commit, o se puede ejecutar con el flag -m, con el mensaje entre comillas simples o dobles. Mejor hacerlo con dobles, así podrás usar apóstrofes (muy habituales en inglés) dentro del mensaje.

```
1 git commit
2 git commit -m "MESSAGE"
```

Es posible que te hayas equivocado al escribir el mensaje de commit o que se te haya olvidado añadir un archivo. No hay problema, para eso está el flag «--amend». Si ejecutas «git add» antes, añadirás otro archivo al commit, o una actualización a un archivo ya añadido. Si lo usas con el flag -m, actualizarás el mensaje del commit. Se pueden usar juntos.

```
1 git add CHANGED_OR_NEW_FILE(s)
2 git commit --amend -m "CHANGED_MESSAGE"
```

### WARNING: LIMITACIONES Y BUENAS PRÁCTICAS

Esto sólo servirá para el último commit que hayas realizado. Es posible cambiar commits anteriores, pero es más complicado, y es mejor no tener que llegar a esa situación. Por otro lado, es fundamental no utilizar amend con cambios ya publicados. Si eres el único que trabaja con ese repositorio, no pasa nada, pero si hay más gente trabajando, es posible que hayan hecho cambios desde el commit que quieras sustituir. Eso puede dar muchos problemas, y es mejor hacer un commit nuevo indicando que es un arreglo.

- **git push.** Envía los cambios comprometidos (almacenados con «git commit») al repositorio remoto. Tiene dos argumentos: el nombre del repositorio remoto, y la rama que se desea enviar.

```
1 git push origin BRANCH
```

Si has llamado al repositorio remoto de otra manera que no sea "origin", sustitúyelo en el comando.

Es muy importante especificar a qué rama quieres mandar los cambios, ya que lo habitual es separar en ramas diferentes etapas del desarrollo (incluso paralelas). Si no se especifica la rama, se mandará a la principal, que se presupone lista para producción.

- **git pull**. Actualiza el repositorio local con los cambios que haya en el remoto. Se usa igual que «git push».

```
1 git pull origin BRANCH
```

Se le puede pasar el flag –force para saltarse errores y advertencias, y literalmente forzar el push. Esto es **MUY PELIGROSO** (pregunta a Pedro y Elena para más detalles).

- **git reset**. Sirve para deshacer commits. Con el flag «–soft», simplemente vuelves al commit especificado. Los cambios que hayas hecho desde entonces aparecerán al ejecutar «git status». Con el flag «–hard» **BORRARÁS** todos los cambios locales posteriores al commit especificado.

```
1 git reset --soft HEAD~N  
2 git reset --hard HEAD~N
```

Estos comandos retrocederán N commits (NO irán al commit «N», cuidado). Si no se introduce un valor de N, por defecto se tomará 1 (es decir, irá al commit más reciente). **MUCHÍSIMO** cuidado con el «–hard», los cambios locales borrados serán irrecuperables.

Hay un comando parecido a este que conlleva mucho menos riesgo, y es **git revert**. Se usa igual que «git reset», pero sin flags. La diferencia es que este no borra el commit, sino que crea un commit nuevo que deshace el anterior. Esto mantiene el historial del error, por así decirlo, eliminando el riesgo de perder información accidentalmente, ya que el commit deshecho no se borra.



**Part II**

**Sección de usuario**





## 4. AutoFlow

Una Herramienta para gestionarlas a todas. Una Herramienta para modularizarlas, una Herramienta para coordinarlas a todas, integrarlas en las tinieblas del sistema de colas.

### 4.1 Descripción

En el análisis bioinformático es muy frecuente el uso de varios programas enlazados, uno detrás de otro. Estos programas se suelen organizar en **flujos de trabajo**. Los flujos de trabajo se dividen en **tareas** más concretas en las que se ejecutan uno o varios programas. Las tareas de un flujo están normalmente enlazadas entre sí ya que es frecuente que una tarea necesite del resultado de otra para ejecutarse. Esto es lo que denomina una relación de **dependencia** entre tareas.

AutoFlow es un programa de línea de comandos que gestiona flujos de trabajo de forma automática.

### 4.2 Forma de uso

AutoFlow recibe flujos de trabajo en forma de plantilla, los ejecuta de forma controlada tanto en el sistema local o un sistema de colas.

#### 4.2.1 Crear una plantilla

Las plantillas de AutoFlow son archivos de texto plano que se organizan en tareas. Cada tarea comienza con su nombre (que debe ser un nombre único, sin espacios y que resuma la tarea en pocas palabras) seguido de un signo de cierre de paréntesis «)».

Las tareas están compuestas por código de *bash* en el que se llama a los diferentes programas. Este código se escribe en un espacio entre llaves «{ » y « }» que comienza justo después del nombre de la tarea. Además en cada tarea hay dos secciones: i) la sección de iniciación donde se carga todo el software necesario y se preparan los archivos de entrada y ii) la sección principal donde se procesan esos archivos mediante de la ejecución de uno o más programas. Las dos secciones están separadas por el signo de cierre de interrogación «?». **Es muy importante** tener en cuenta que el

primer comando después de «?» será el comando principal ya que el comando principal se usará para darle nombre a la carpeta de salida (por defecto) y, además, se medirá el rendimiento de este comando mediante el comando `time`

```

1 task_name){ #nombre de la tarea delimitado con ')' y comienzo del codigo
2   con '{'
3   module load main_command # initialize: Código para cargar software y
4   preparacion de input
5   ? #caracter que comienza la sección principal
6   main_command --param # Comando principal
7 } # el ')' marca el fin de la tarea
8

```

Al ejecutar Autoflow para una plantilla, este creará un directorio de ejecución. En este directorio se creará una **carpeta para cada tarea**. Por defecto, esta tarea se llamará como el comando principal seguido de un código numérico. Dentro de cada carpeta de tarea se generará un *script* de *bash* (.sh) con el código de la tarea, que se ejecutará en el **entorno** de dicha carpeta.

### Dependencia de tareas

Como se ha mencionado anteriormente, hay tareas que dependen del resultado de otras tareas. Esta relación de dependencia se crea al mencionar el **nombre de una tarea** dentro de otra. Esta llamada a una tarea será **sustituida** por la ruta de la carpeta de la tarea mencionada en el *script* resultado, por eso es importante **tratar la mención como si fuera una ruta**.

#### Plantilla:

```

1 task_A{
2
3   ?
4   echo "May the force be with you"
5   > out
6 }
7
7 task_B{
8
9   ?
10  sed 's/you/u/g' task_A)/out >
11  out_corrected
12 }

```

#### Script resultado de echo\_0000/task\_A.sh:

```

#!/usr/bin/env bash

time echo "May the force be with
you" > out

```

#### Script resultado de sed\_0000/task\_B.sh:

```

#!/usr/bin/env bash

time sed 's/you/u/g' /path/to/exec/
echo_0000/out

```

### Iteración de tareas

En los flujos de trabajo un poco más complejos se suele ejecutar muchas veces una única tarea con diferentes argumentos para un programa, como por ejemplo ejecutar el mismo código para diferentes archivos de entrada. Lo ideal para esta situación es crear una tarea iterativa. Autoflow nos proporciona una manera intuitiva para ello. En primer lugar, el nombre de la tarea a iterar irá acompañado del **iterador**, un conjunto de parámetros, delimitado por corchetes, '[' y ']', y separados por ';':

```

1 task_name_[param_1;param_2;...;param_n]){
2   ?
3   main_command
4 }

```

Tras ello, en la sección «*main*» los parámetros se reemplazarán allí donde aparezca la expresión «(\*)».

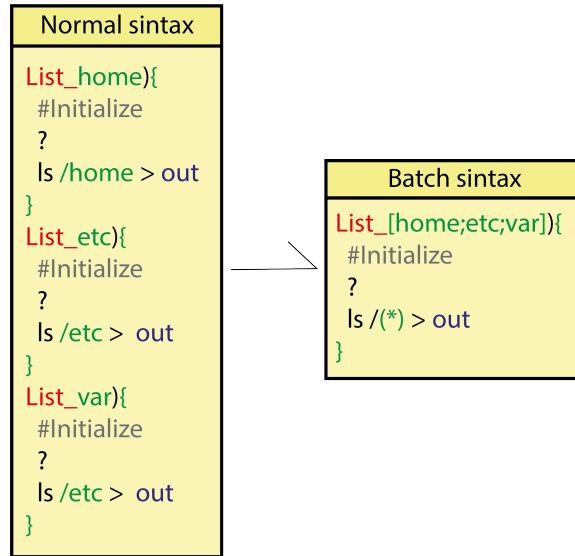


Figura 4.1: Ejemplo de scripts para cada sintaxis.

```

1 #!/usr/bin/env bash
2
3 ls /home > out
4

```

```

1 #!/usr/bin/env bash
2
3 ls /etc > out
4

```

```

1 #!/usr/bin/env bash
2
3 ls /var > out
4

```

### Dependencia de una iteración

Las tareas que dependen de otra tarea iterativa pueden estructurarse de dos formas diferentes.

- **Tarea no iterativa que depende de una que sí es iterativa.** Esta tarea dependerá de **todas** las subtareas generadas por la tarea iterativa anterior y no comenzará hasta que todas ellas terminen. Así, se forma una relación de dependencias de «muchas a una». Esta relación de dependencia se crea al invocar al nombre de la tarea «padre» entre signos de cierre de exclamación «!» y **sin el iterador** dentro de la tarea dependiente, ej: «!iterative\_task\_name\_!». Figura 4.2, derecha. En este caso AutoFlow sustituirá la invocación por las rutas a las carpetas cada una de las tareas iteradas, separadas por espacios.

- **Tarea dependiente iterativa con mismo iterador.** En este caso hay una forma especial de indicar las dependencias para que cada iteración de la tarea dependa de la correspondiente iteración en la tarea «padre». Esta dependencia se invoca de la misma forma que la anterior pero añadiendo un «\*» en el lugar donde debiera estar el iterador, ej: «!iterative\_task\_name\_\*!». Figura 4.2, izquierda.

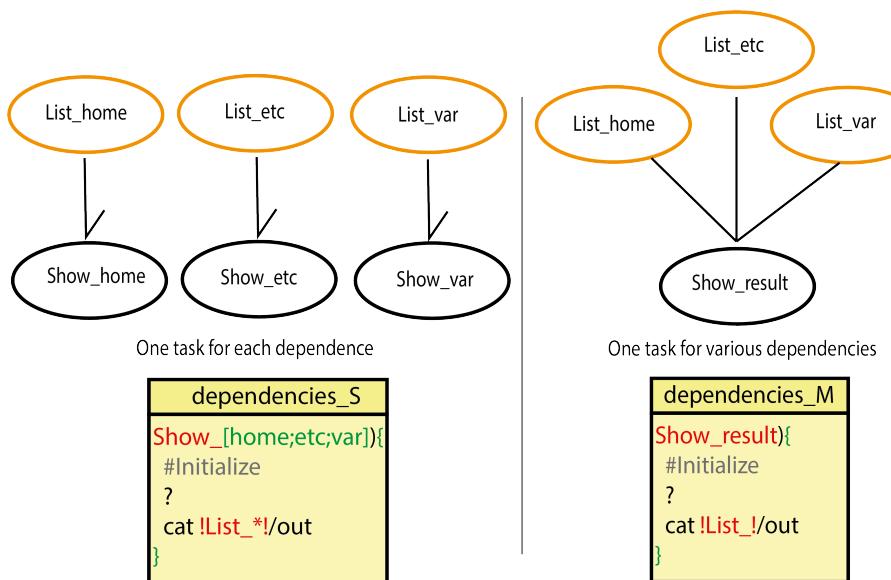


Figura 4.2: Tipo de dependencias de una tarea iterativa. Las tareas ejemplo de esta figura son dependencias de la figura anterior

### Variables y datos de entrada

En Autoflow nos podemos encontrar con dos variables propias de su sistema: (1) **Variables estáticas**, definidas y mencionadas con el prefijo «\$»; (2) **Variables dinámicas**, definidas con el prefijo «@» como variables de entorno, automáticamente. Estas últimas pueden invocarse como una variable de entorno más o modificarse su valor en el «main» de la siguiente manera: «env\_manager "variable\_din=valor\_nuevo"». Así, permiten transferir datos entre tareas.

Dynamic & static variables
<pre> @n_files=0 \$folder=/var List_dir{     #Initialize     ?     ls \$folder &gt; out     num=`ls \$folder wc -l`     env_manager "n_files=\$num" } Show_list{     #Initialize     ?     echo "Num of files: \$n_files"     cat List_dir/out } </pre>

Figura 4.3: Variables y valores de entrada.

### Gestión de recursos necesarios

Distintas tareas requerirán de recursos específicos para llevarlas a cabo. En ocasiones no es tanto una cuestión de necesidad sino más bien de eficiencia. Los recursos generales de todo el script pueden ordenarse desde la línea de comandos de Autoflow. Además, es posible especificar en la sección de *Initialize* de cada tarea empezando por «resources:», seguido de la especificación para cada parámetro.

#### ¿Cuáles serían estos parámetros?

- **-s:** Usar más de un nodo para ejecutar cada trabajo.
- **-c:** Número de CPUs a usar en cada trabajo.
- **-n:** Asignar trabajos a un tipo de cola determinado (bignum, cal o Gigabit).
- **-t:** Tiempo para cada trabajo. Formato: *days-hours:minutes:seconds*.
- **-u:** Número de nodos a usar en cada trabajo.
- **-m:** Memoria que va a necesitar el script/tarea.

#### WARNING: Cuidado a la hora de pedir las cpus

Debe indicarse para que autoflow añada las cpus a la plantilla con una línea que ponga [cpu].

Example
<pre> List_dir{     resources: -s -n cal -c 1 -t 3-03:53:00 -m 100gb     ?     ls &gt; out1 } Show_dir{     resources: -n cal -c 2 -t 3-00:00:00 -m 10gb     ?     cat List_dir/out1 &gt; out2 } </pre>

Figura 4.4: Ejemplos de gestión de recursos por tareas.

### 4.3 Ejecución por linea de comandos

#### Comando base:

Una vez tenemos construida la plantilla, el siguiente paso es ejecutar el flujo de trabajo. Para ejecutarlo en un sistema de colas (será el caso más común) solo hay que indicar con el parámetro «-w» («--workflow») la ruta a la plantilla y con el «-o» («--output») la ruta de salida.

```
AutoFlow -w template_name -o exec
```

#### Variables

Desde la línea de comandos pueden añadirse valores de entradas para las variables de la plantilla con el parámetro «-V» o «--Variables». Esta funcionalidad nos permite tener una plantilla general de un flujo que podremos personalizar para cada proyecto simplemente modificando el comando de AutoFlow, sin necesidad de editar la plantilla.

```
AutoFlow -w template_name -V '$var1=val1,$var2=val2'
```

#### Recursos

Los recursos pueden ser también definidos desde la línea de comandos. Cuando se definen los recursos por la linea de comandos se hace de forma general para **todas las tareas**, pero la configuración de recursos específica de cada tarea sobrescribe a la general. Las opciones que se pueden configurar son las mismas que en el apartado 4.2.1.

```
AutoFlow -w template_name -s -n cat -c 1 -t 3-03:53:00 -m 100gb
```

#### Representación gráfica del flujo

Autoflow da la posibilidad de conseguir una representación esquematizada del flujo de trabajo. La representación puede ser semántica, indicando cada tarea con el nombre introducido por el usuario, o estructural, con el programa usado en el main. Esto ayuda a tener una imagen de conjunto sobre las dependencias entre tareas.

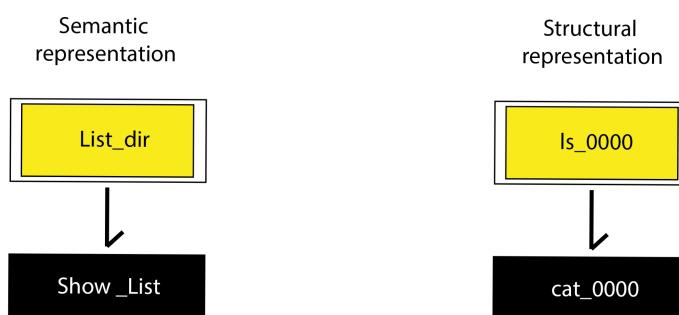


Figura 4.5: Ejemplo de cada tipo de representación gráfica de flujo.

Para conseguirlo, hay que utilizar el parámetro «-g» («--graph») seguido del valor «t» (representación semántica) o «f» (representación estructural).

```
1 #Semantic representation.
2 AutoFlow -w template_name -g t
3
```

```
4 #Structural representation.  
5 AutoFlow -w template_name -g f
```

### Modo verbose

En este modo, AutoFlow crea la estructura de carpetas sin ejecutar nada. Esto es **muy útil** para comprobar que todo es correcto antes de ejecutar. Para activar este modo basta con usar el parámetro «-v» (forma larga «--verbose»).

```
1 AutoFlow -w template_name -v
```

Este comando es especialmente útil al combinarse con el flow logger. Lo explicaremos más adelante.

### Modo local

Con el parámetro «-b» o «--batch» AutoFlow ejecutará el flujo completo en local, es decir, no enviará trabajos al sistema de colas.

```
1 AutoFlow -w template_name -b
```

### Comentar línea principal

Es el flag «-C» («--comment\_main\_command») AutoFlow comentará la primera línea de la tarea, es decir, el comando principal. Hacerlo de forma manual rompe AutoFlow, ya que necesita esa línea internamente. Hacerlo de esta forma evita el problema, permitiendo hacer reejecuciones parciales sin tener que repetir el paso más pesado. Puedes comentar en la plantilla el resto de líneas perfectamente si existen más comandos que te quieras ahorrar. No es una opción que vayas a utilizar frecuentemente, pero es útil conocerla para desarrollo.

```
1 AutoFlow -w template_name -C
```

### Opciones adicionales

Este flag, «-A» o «--additional\_job\_options», permite especificar otras opciones del sistema de colas.

```
1 AutoFlow -w template_name -A "parameter:value"
```

### Pausa entre lanzamientos

Es posible establecer un tiempo de espera entre lanzamientos con el flag «--sleep». Muy útil en flujos grandes, en los que al lanzar una gran cantidad de trabajos es posible que el sistema de colas empiece a rechazarlos.

```
1 #Sleep 0.1 seconds between job submissions  
2 AutoFlow -w template_name --sleep 0.1
```

### Carpetas simbólicas

La organización normal de directorios al lanzar una plantilla de AutoFlow se determina según el comando principal de cada tarea. Según las veces que se ejecute dicho comando, lo que dependerá de cuántas tareas lo tengan como principal y los iteradores de dichas tareas, ese directorio tendrá además un número (por ejemplo: cat\_0000, cat\_0001, etc.). Esto supone un problema: si una de las muestras se elimina del flujo, o si se añaden muestras nuevas, la estructura de estas carpetas se

alterará, pero su contenido previo no se eliminará. Esto mezcla ejecuciones distintas, entorpeciendo al flujo actualizado e impidiendo el correcto funcionamiento de flow\_logger.

Para solucionar este problema se ha añadido el flag -L, que indica a AutoFlow que genere unos nombres estáticos a los que luego hará referencia de la forma habitual mediante simbólicos. Esto quiere decir que nosotros veremos la misma estructura, pero realmente se está usando otra más robusta que resiste estos cambios.

```
AutoFlow -L -w template_name
```

Con el tiempo este será el comportamiento por defecto de AutoFlow. La principal ventaja es que permite cambiar iteradores e incluso el comando principal de las tareas sin obligar a reejecutar el flujo completo para aclarar la estructura de carpetas, lo que también permite usar flow\_logger sin problema alguno.

#### WARNING: FLUJOS CON Y SIN -L SON INCOMPATIBLES

Un flujo lanzado de una forma romperá si se intenta ejecutar de la otra. A partir de ahora, **TODOS** los flujos se deberán ejecutar con -L

## 4.4 Aspectos avanzados

**Fusión de plantillas** Puede suceder que tengamos distintas plantillas con entradas y salidas compatibles. Autoflow ya tiene esto en cuenta, permitiendo la fusión de ambos flujos de trabajo en uno. Para unirlos se utiliza una variable intermedia que conectará una tarea con otra.

Por ejemplo, supongamos que tenemos las siguientes tareas:

<pre>1 List_dir){ 2     #Initialize 3     ? 4     ls &gt; out 5 }</pre>	<pre>1 Show){ 2     #Initialize 3     ? 4     cat \$file 5 }</pre>
---	--

Entonces, podemos *fusionar* ambas plantillas, con el siguiente comando:

```
AutoFlow -w List_template,Show_Template -V '$file=List_dir)/out'
```

**Atributos de tareas especiales.** Por lo general, Autoflow ejecuta cada tarea, creando una carpeta para cada una de ellas. Sin embargo, podemos hilar más fino sobre cómo debe actuar Autoflow frente a una determinada tarea. Para ello, se tiene que añadir un prefijo específico al nombre de la tarea. Así, usaremos: (1) «%» para indicar que la tarea no sea ejecutada, pero se sigan teniendo en cuenta sus resultados; (2) «!» para no crear una carpeta propia para la tarea y (3) «&» con el que se agregan varias tareas entre sí, para constituir un trabajo en el sistema de colas.

**Iteración de iteraciones.** Al igual que podemos hacer con un *bucle for* en ciertos lenguajes de programación, podemos realizar varios bucles sucesivos desde AutoFlow. ¿Cómo llevar esto a cabo? Es tan sencillo como definir una tarea iterativa que contenga otras tareas iterativas. Valga lo siguiente de caso concreto:

Example
<pre>ls_[temp;sys]){ ? scan){ ? pwd &gt; file } show_[file;folder]){ ? echo `cat scan)/file` ls_(+) } }</pre>

Figura 4.6: Iteración de iteraciones. `ls_(+)` corresponde al iterador superior («temp» y «sys»), y `(*)` al inferior («file» y «folder»).

**Repositorio de plantillas.** Se recomienda utilizar un repositorio de plantillas general y otro propio del usuario. Para utilizar estas plantillas sin tener que mencionar el path es necesario escribir el siguiente comando en el `.bashrc`:

```
1 export WORKFLOW_REPOSITORY=folder:$WORKFLOW_REPOSITORY
```

## 4.5 Buenas prácticas

Cuando se esté desarrollando un flujo de trabajo hay que tener en cuenta su utilización en un futuro y los posibles errores que puedan generarse. Por ello, es una buena práctica el mantener un archivo con las versiones utilizadas para cada programa ejecutado durante las tareas, así como mantener un report de las mismas.

**Archivo versions.** Cuando se ejecuta Autoflow se genera automáticamente un archivo `versions` dentro de la carpeta de ejecución con su número de versión. Cada tarea debería tener un comando que capture la versión del programa que se vaya a utilizar y volcarlo al archivo mencionado. Para hacer esto, en la sección *Initialize* de cada tarea bastará con poner:

```
1 [upquote=true]
2 task_A){
3   echo -e "nombredelprograma\t /
4     'linea de comandos para obtener la version'" >> ../versions
5   ?
6   ....
7 }
```

**Archivo report.** Para ello, en la sección «main» de la tarea se ejecutará una línea de comandos que escriba el nombre de la tarea y sus resultados obtenidos, enviándolos al archivo «report».

```
1 task_A{
2   ....
3   ?
4   echo -e "Nombre de la tarea
5   -----
```

```

6     Resultados obtenidos por la tarea
7     " >> ../report
8 }
```

Esto resulta de gran utilidad para depurar errores y recopilar los datos obtenidos durante el proceso.

## 4.6 Flow logger

Esta funcionalidad de AutoFlow permite controlar que la ejecución del flujo haya ido bien, e incluso indicar a AutoFlow que vuelva a lanzar los trabajos que hayan fallado. Tiene dos modos de ejecución: uno de comprobación de ejecución (el que usaremos normalmente) y otro de logging (se usa dentro de AutoFlow, rara vez lo tocaremos). En esta sección explicaremos cada flag, y al final ilustraremos el uso de flow logger con un ejemplo.

### Logging

AutoFlow se encarga de esto, así que rara vez será necesario utilizar estos flags manualmente. De todas formas, es importante saber cómo funciona el programa.

Estos flags son «-s» (o «--start») y «-f» (o «--finish»), y van seguidos del nombre de la tarea. Al ejecutar AutoFlow, cada tarea se lanza desde su propio archivo sh. Este archivo incluye, a efectos de lo que ahora nos interesa, una llamada a flow logger con el flag -s, el cuerpo de la tarea, y una segunda llamada a flow logger con el flag -f. La primera llamada a flow logger deja una marca de que la tarea se ha ejecutado, y la segunda deja otra para indicar que la tarea ha finalizado. Flow logger en el modo de comprobación busca estas dos marcas, y según cuáles encuentre devuelve un estado u otro.

### Workflow execution

Este flag, «-e» o «--workflow\_execution», indica el path que flow logger va a utilizar. En el caso de la comprobación de ejecución, hay que pasarle como argumento los paths que queramos comprobar.

### Report

Indica a flow logger qué estados queremos comprobar. Tiene las formas «-r» y «--report», seguido del estado que queremos comprobar. Lo más frecuente es pasarle «all», para comprobarlo todo, aunque admite cualquier posible estado de tarea.

Con estos dos flags ya es posible ejecutar flow logger en su forma más simple:

```
flow_logger -e exec_folder -r all
```

### Flujo terminado

Su flag corto es «-w», y el largo «--workflow\_finished». No tiene argumento. Si está presente, flow logger asume que el flujo ha terminado, lo que afecta a los nombres de los estados con los que marca la tarea. Por ejemplo, en ausencia del flag, si detecta la marca de comienzo pero no la del final, asigna "PENDING" a la tarea, lo que significa que no se ha ejecutado. Si se añade el flag, marca la tarea como "ABORTED", es decir, que ha fallado y no se ha completado.

```
flow_logger -w -e exec_folder -r all
```

### Lanzar trabajos fallidos

En presencia del flag «-w», se puede añadir este otro («-l» o «-launch\_failed\_jobs»). Tampoco tiene argumento, y lo que hace es llamar a AutoFlow para que vuelva a lanzar las tareas marcadas como ABORT (fallidas, hay marca de comienzo pero no de final). Al ser una ejecución, admite el flag «--sleep» de AutoFlow.

```
flow_logger -w -l -e exec_folder --sleep 0.1
```

### Lanzar trabajos pendientes

Funciona exactamente igual que el flag «-l», sirve para lanzar los trabajos marcados como NOT (no lanzadas, no hay marca de comienzo). El flag es «-p», en su forma larga «--pending». Se puede combinar con el anterior para lanzar todos los trabajos fallidos y pendientes.

```
flow_logger -w -p -e exec_folder --sleep 0.1
```

Este comando es especialmente potente en combinación con el modo verbose de AutoFlow. Pongamos que has ejecutado un flujo que ya tienes desarrollado con un conjunto nuevo de muestras, más grande que las pruebas anteriores, y un conjunto de tareas casi al final rompen por falta de memoria. Es tan fácil como actualizar la plantilla pasando más memoria a las tareas, lanzar AutoFlow en modo verbose y ejecutar flow\_logger con el flag -l. La plantilla se actualizará con los parámetros nuevos de memoria y relanzará todos los trabajos que han fallado. Esto también es válido para cambios más grandes en el código.

### Lanzar trabajos pendientes

Mismo funcionamiento que el anterior, en este caso para lanzar los trabajos que no se han llegado a lanzar (marcadas como "NOT"). Su forma corta es «-p», y la larga es «--pending». Al ser una ejecución, admite el flag «--sleep» de AutoFlow.

```
flow_logger -w -p -e exec_folder -r all --sleep 0.1
```

### No imprimir tamaños

También admite el flag «-n» (forma larga «--no\_size») para que el logger no anote el tamaño de la carpeta de cada tarea.

```
flow_logger -e exec_folder -r all -n
```

### Modo batch

El flag «-b», con forma larga «--batch», funciona igual que el de AutoFlow. Sirve para ejecutar los trabajos en local, sin enviarlos al sistema de colas.

```
flow_logger -b -p -e exec_folder -r all
```

### Report en html

Función en desarrollo, actualmente no funcional. En presencia de este flag («-H» o «--html»), flow logger imprimirá un report mucho más detallado en un fichero html.

```
flow_logger -H -e exec_folder -r all
```





## 5. cmdtabs

**NOTA IMPORTANTE: REFACTORIZACIÓN GORDA POR REDUNDANCIAS, DOCUMENTACIÓN NO DEFINITIVA**

### 5.1 Descripción

Link al report de uso de CMDTabs [https://github.com/seoanezonjic/py\\_cmddata/blob/master/tests/cli\\_example\\_report/Report.html](https://github.com/seoanezonjic/py_cmddata/blob/master/tests/cli_example_report/Report.html)

En cualquier trabajo desarrollado en este laboratorio es necesario el manejo de las tablas<sup>1</sup>, ya sea para separar una tabla por filas o por columnas, fusionar varias tablas o cambiar el formato de las mismas. Normalmente para hacer estas operaciones recurrimos a comandos de *BASH*. Sin embargo, en los casos en los que necesitemos construir un comando muy complejo debemos recurrir a la librería **CMDtabs**:

Actualmente existen dos repositorios de la herramienta CMDtabs en GitHub: programada en Ruby y en Python. La que está disponible en Picasso para su uso es la que está programada en Python. Ambas se pueden descargar asimismo desde sus correspondientes repositorios de GitHub a través de los enlaces:

- Python: [https://github.com/seoanezonjic/py\\_cmddata](https://github.com/seoanezonjic/py_cmddata)
- Ruby: <https://github.com/seoanezonjic/cmddata>

### 5.2 Características y parámetros comunes de los scripts de py\_cmddata

A continuación se describen las características principales de los scripts de la herramienta py\_cmddata. Para acceder a ellos en Picasso, debemos utilizar el comando:

```
source ~soft_bio_267/initializes/init_python
```

Una característica común de todos los scripts que se explicarán en los apartados siguientes, en cuanto a parámetros se refiere, es que:

<sup>1</sup>\*Una tabla es un archivo de texto con columnas delimitadas por un separador (normalmente es la tabulación, "\t")

- Allá donde se soporte, el parámetro **-o** o en forma larga **–output\_file** sirve para indicar el archivo en el que queremos que se escriba el resultado de aplicar la funcionalidad del script concreto utilizado. Si se omite el parámetro, se ejecuta el script en la terminal, devolviendo el flujo de datos al standard output, de manera que pueda ser utilizado por un script posterior en bash por medio de una "pipe". De esta forma, se puede utilizar para encadenar con comandos posteriores formando pipelines.
- Allá donde se soporte, el parámetro **-i** o en forma larga **–input\_file** sirve para indicar el archivo de entrada con el que queremos trabajar. Si en vez de querer indicar que cargue un archivo, queremos que encadene los datos provenientes de un script anterior (pipelines), podemos hacerlo escribiendo como parámetro de entrada '**-i -**' o '**–input\_file -**', estableciendo dicho guión que el archivo que toma con entrada es el resultado del comando anterior.
- Se puede utilizar el parámetro **–transposed** con cualquiera de los scripts siguientes, en cuyo caso la tabla de entrada es traspuesta una vez que se ha cargado. Además, una vez que se realiza la funcionalidad pertinente, se traspone de vuelta los datos resultantes antes de ser escritos a la salida (sea archivo o pipe).
- Se puede utilizar el parámetro **–compressed\_in** si los datos de entrada con los que queremos trabajar (aquellos referenciados por el parámetro **-i**) vienen comprimidos, con independencia de si proceden de un archivo o como resultado de un script anterior encadenado por pipelines en bash.
- Se puede utilizar el parámetro **–compressed\_out** si queremos que los datos de salida (aquellos referenciados por el parámetro **-o**) vayan comprimidos, con independencia de si van a ser escritos en un archivo o van a ser pasados a otro comando mediante pipelines.
- El parámetro **-H** o **–header** se puede utilizar en cualquier script para indicar que la tabla que cargamos con el flag **-i** contiene un cabecera, de manera que se elimina temporalmente a la hora de realizar la función concreta del script, pero se escribe finalmente en el archivo de salida.

Es posible que la funcionalidad de alguno de estos parámetros, comunes a todos los scripts, se vea alterada si el script concreto realice alguna función que así lo requiera, en cuyo caso, se verá explicada esa modificación en el apartado del script pertinente.

### 5.2.1 aggregate\_columns\_data

Algunas de nuestras herramientas principales, como NetAnalyzer y Cohort Analyzer, trabajan con información en la que un identificador (ya sea de paciente, enfermedad, muestra, clúster, etc.) se asocia con varios elementos del mismo tipo, como términos HPO o genes. Este tipo de información se puede guardar en dos formatos diferentes:

- En un formato de dos columnas (o desagregado), donde cada línea corresponde a un identificador y un solo elemento asociado.
- En un formato agregado, donde cada línea incluye un identificador y todos sus elementos asociados delimitados por otro **separador**.

El script `aggregate_column_data.py` nos permite cambiar el formato de la información guardada en un archivo dado, pasándolo de formato desagregado (dos columnas) a agregado.

Los argumentos de entrada del programa son los siguientes:

- **-i, –input\_file** -> Ruta al archivo que contiene la información cuyo formato queremos cambiar. Dicha información vendrá dada en dos columnas separadas por tabulación, siendo una la de identificadores.

- **-x, --column\_index** -> Índice de la columna (en base 1) que contiene los identificadores.
- **-s, --separator** -> Carácter que se usará como separador en la tabla agregada.
- **-a, --column\_aggregate** -> Índice de la columna (en base 1) con los elementos que se desea agregar.
- **-h, --help** -> Muestra una lista de los posibles argumentos que admite nuestro script.

#### Ejemplo de uso.

Imaginemos que tenemos un archivo de dos columnas en el que la primera columna es un identificador de paciente y la segunda un fenotipo. Queremos obtener un archivo que contenga por cada fila el identificador de paciente en la primera columna y en la segunda los fenotipos separados por coma. Para ello, se debería ejecutar el script como se especifica a continuación:

```
aggregate_column_data -i input_file.txt -x 1 -a 2 -s ','
```

#### 5.2.2 desaggregate\_column\_data

Realiza el proceso contrario que el script aggregate\_column\_data.py. De esta manera, nos permite una tabla en formato agregado a desagregado, produciendo un archivo de dos columnas en el que cada identificador se repite tantas veces como características tenga. Por ejemplo, un paciente como dos fenotipos aparecerá en dos líneas, una por cada fenotipo.

- **-i, --input\_file** -> Ruta al archivo que contiene la información cuyo formato queremos cambiar. Dicha información vendrá dada en un formato agregado, en el que aparecerá cada identificador junto a todos sus elementos asociados (los cuales vendrán separados entre sí por un **separador** concreto), separados por tabulación.
- **-x, --column\_index** -> Índice de la columna (en base 1) que contiene los elementos que se desea desagregar.
- **-s, --separator** -> Carácter empleado para separar todos los elementos asociados a un mismo identificador.
- **-h, --help** -> Muestra una lista de los posibles argumentos que admite nuestro script.

#### Ejemplo de uso.

Imaginemos que tenemos un archivo de dos columnas en el que la primera columna es un identificador de paciente y la segunda una lista de fenotipos separados por comas. Queremos obtener un archivo que contenga por cada fila el identificador de paciente en la primera columna y en la segunda un único identificador de fenotipo. Para ello, se debería ejecutar el script como se especifica a continuación:

```
desaggregate_column_data -i input_file.txt -x 1 -s ','
```

#### 5.2.3 transpose\_table

Transpone la tabla dada como entrada. Dado que esta es la única función de este binario, en este caso no es necesario utilizar el flag "transposed", que si pueden utilizar otros de los binarios del cmdtabs para transponer la tabla antes de aplicar su función concreta.

- **-i, --input\_file** -> Ruta al archivo que contiene la tabla a transponer, la cual tendrá sus registros separados por tabulación.
- **-o, --output\_file** -> Ruta al archivo en el que queremos guardar la tabla tabulada. En caso de no utilizar este flag, la tabla se devolverá al standard output (stdout).
- **-h, --help** -> Muestra una lista de los posibles argumentos que admite nuestro script.

### Ejemplo de uso.

Tenemos una tabla, llamada `input_file.txt`, que nos interesa transponer, de manera que las columnas pasen a ser filas, y viceversa. Queremos guardar esa tabla transpuesta en un archivo llamado `output.txt`. Entonces podemos ejecutar el script tal y como se muestra a continuación:

```
transpose_table -i input_file.txt -o output.txt
```

### 5.2.4 create\_metric\_table

Normalmente, en nuestros flujos de trabajo, computamos una serie de medidas para comparar entre sí varias ramas de dicho flujo. En ocasiones, cuando queremos calcular diferentes variables a partir de varios elementos del mismo tipo (por ejemplo, genes), es preferible obtener dichas medidas de manera desglosada, insertando cada una en una línea independiente. Cada elemento se marca con un identificador único seguido del nombre de la variable y su valor. Para representar esta información gráficamente, lo ideal es tabularla, por lo que habrá que ajustar su formato.

El script `create_metric_table.py` nos ofrece la posibilidad de llevar a cabo el cambio de formato de una manera rápida y sencilla. El programa requiere argumentos de entrada de forma posicional, y al final el flag `<-c>` con su argumento. Los argumentos son:

1. **Posición 1** -> Archivo que se desea formatear
2. **Posición 2** -> Atributos separados por comas. En el caso más simple será, por ejemplo, el identificador de la muestra. En casos más complejos, puede ser muestra Y tratamiento. De esa manera, separando por comas, indicamos al script dos cosas. La primera es que hay dos atributos con los que agrupar las muestras, que son su identificador y su tratamiento (que no considere que todos los tratamientos son iguales). En segundo lugar, le indicamos cómo queremos que llame a esos dos atributos (lo razonable sería "muestra" y "tratamiento").
3. **Posición 3** -> Archivo de salida. Contendrá las métricas ya formateadas.
4. **-c, --corrupted (No posicional)** -> Archivo de salida que almacena métricas erróneas.

### Ejemplo de uso.

Tenemos un archivo de métricas que queremos procesar con `create_metric_table.py` para facilitar su interpretación. El siguiente comando indica que queremos procesar el archivo `«all_metrics»`, agrupando por un atributo al que llamaremos `«sample»`, volcando el resultado en el archivo `«metric_table»`, y desviando los errores a `«$out/TEST_file»`.

```
create_metric_table.py all_metrics sample metric_table -c $out/TEST_file
```

### Particularidades del script.

En un flujo de trabajo ideal, haríamos una ejecución con ciertos argumentos de entrada y obtendríamos directamente nuestra salida. En un caso real, rara vez será así, y tendremos de por medio ejecuciones parciales que se han detenido por errores a lo largo del flujo, reejecuciones con cambios de parámetros porque los resultados no son válidos (p-valor demasiado laxo, por poner un ejemplo) y un larguísimo etcétera.

Al ser la generación de las métricas un proceso de concatenación, cada ejecución sucesiva del flujo no sobreescribirá esta información, sino que la añadirá. Este script lo leerá todo, pero en el archivo final sólo reflejará la información más reciente.

Por ejemplo, pongamos que en un flujo de enriquecimiento funcional hemos establecido un p-valor de 0,1 y hemos detectado diez mil términos GO. Esto es demasiado, así que endurecemos el criterio: reejecutamos estableciendo un p-valor de 0,01. Ahora detectamos 27 términos, una cantidad razonable. El archivo de métricas contendrá ambos, pero `create_metric_table.py` sólo considerará el de 27, porque aparece en segundo lugar.

### 5.2.5 merge\_tabular

Durante el desarrollo del trabajo en el laboratorio es habitual tener la necesidad de unir la información contenida en dos archivos diferentes.

El script `merge_tabular` nos permite unir dos o más tablas tabuladas proporcionadas por línea de comandos.

A la hora de ejecutar este script deberemos pasarle como argumentos la ruta de los archivos cuya información queremos unir. Al igual que en el script anterior, dichos argumentos se pasarán por la consola de comandos sin la necesidad de incluir ningún flag, sin importar su orden en este caso.

La información contenida en los archivos pasados como argumentos deberá estar en forma de tabla. La primera columna debe contener el identificador del elemento, y en base a él se realizará la unión, concatenando por columnas. Ejemplo de uso:

```
merge_tabular.py table_1 table_2 ... table_n > merged_table
```

### 5.2.6 tag\_table

En algunas ocasiones es necesario incluir una serie de etiquetas específicas a elementos de una tabla. Para ello se emplea el programa `tag_table.py`.

- **-i, --INPUT\_FILE** -> Ruta al archivo principal que contiene la información tabulada, a la cual, deseamos incluirle una serie de etiquetas.
- **-t, --TAGS** -> String correspondiente a las etiquetas o a los archivos que contendrán las etiquetas que deseamos añadir a nuestro archivo tabulado. Este string empleará un carácter determinado (por defecto, coma) para separar sus elementos. También se puede usar un archivo de entrada, en cuyo caso sólo se utilizará la primera línea del mismo como etiqueta.
- **-s, --SEP** -> Carácter empleado para separar los distintos elementos correspondientes al argumento tags.
- **-H, --header** -> Argumento booleano que nos indicará si la información contenida en nuestro archivo principal contiene, o no, una cabecera. En base a ello añadirá las etiquetas desde la primera linea de nuestro archivo (si no hay cabecera), o desde la segunda (si hay cabecera).
- **-h, --help** -> Muestra una lista de los posibles argumentos que admite nuestro script.

### 5.2.7 intersect\_columns

A menudo, es interesante comparar los elementos de dos tablas distintas y obtener un análisis de los elementos que coinciden o difieren en ellas. Para ello, el programa `intersect_columns.py` nos permite comparar los elementos de dos columnas pertenecientes a dos tablas (archivos) distintas y, dependiendo del interés de nuestro estudio, nos devuelve un análisis, más o menos específico, de dicha comparación.

Los argumentos de entrada se detallan a continuación:

- **-a, -a\_file** -> Ruta al archivo que contiene la tabla correspondiente a la primera columna que queremos comparar.
- **-b, -b\_file** -> Ruta al archivo que contiene la tabla a la que pertenece la segunda columna que queremos comparar.
- **-A, -a\_cols** -> Índice de la columna (en base 1) perteneciente a la primera tabla que contiene los elementos de interés para nuestra comparación.
- **-B, -b\_cols** -> Índice (en base 1) de la columna correspondiente a la segunda tabla, la cual, contiene los elementos que deseamos comparar.
- **-c, -count** -> Parámetro booleano que nos permite configurar la salida de nuestro script. Si deseamos que la ejecución nos devuelva un análisis mas detallado de la comparación dejaremos su valor por defecto (FALSE). Si, por el contrario, únicamente nos interesa saber el número de coincidencias encontradas en nuestra comparación, su activación dará lugar a una salida de la forma:
  - a: número de elementos que aparecen únicamente en la columna perteneciente a la primera tabla introducida por parámetros.
  - b: número de elementos que aparecen únicamente en la columna perteneciente a la segunda tabla introducida por parámetros.
  - c: número de elementos en común en ambas columnas.
- **-k, -keep** -> Cuando nos interesa obtener un análisis más detallado de nuestra comparación deberemos especificar que tipo de resultado deseamos obtener. Para ello hacemos uso de dicho parámetro, cuyo valor por defecto es 'c' y, el cual admite los siguientes valores:
  - 'c': nos devolverá, únicamente, el valor los elementos que ambas columnas comparten (elementos en común).
  - 'a': nos devolverá el valor los elementos que solo aparecen en la columna perteneciente a la primera tabla pasada como parámetro.
  - 'b': nos devolverá el valor los elementos que solo aparecen en la columna perteneciente a la segunda tabla pasada como parámetro.
  - 'ab': nos devolverá el valor de aquellos elementos que difieren en ambas tablas.
- **-s, -SEP** -> Índica el carácter con el que se separan las distintas columnas de la tabla, siendo la tabulación (' ') su valor por defecto.
- **-h, -help** -> Muestra la ayuda.

### 5.2.8 table\_linker

Cuando trabajamos con información almacenada en diferentes archivos, es bastante común tener la necesidad de unir dicha información en una única tabla. El script `table_linker.py` nos permite guardar en un mismo archivo de salida la información extraída de un archivo tabulado, en base a los identificadores de un segundo archivo.

Los argumentos de entrada al script son los siguientes:

- **-i, -INPUT\_FILE** -> Ruta al archivo de entrada con la información que deseamos extraer. Dicho archivo deberá estar en formato tabulado y contener en su primera columna los identificadores.
- **-l, -LINKER\_FILE** -> Ruta al archivo que contiene los identificadores en base a los cuales se llevará a cabo la unión de ambas tablas.
- **-o, -OUTPUT\_FILE** -> Ruta al archivo de salida con las tablas unidas.
- **-s, -SEP** -> Separador de columnas en el archivo de salida con los datos colapsados.

- **-drop** -> Configurar para que se escriban las líneas de los identificadores que han coincidido.
- **-h, -help** -> Muestra la ayuda.

Con respecto al argumento **-drop**, se debe mencionar que el programa imprimirá por defecto en el archivo de salida todos los identificadores del segundo archivo aunque no se hayan encontrado coincidencias. Si se activa, se imprimirán exclusivamente las coincidencias encontradas.

### 5.2.9 standard\_name\_replacer

El script standard\_name\_replacer.py nos permite reemplazar los valores de una tabla en función de un código de valores obtenido a partir de los argumentos pasados por consola. Los argumentos necesarios para ejecutar el programa son los siguientes:

- **-h, -help** -> Muestra la ayuda.
- **-i, -INPUT\_FILE** -> Ruta al archivo de entrada, que contiene la tabla en la que llevaremos a cabo la sustitución.
- **-o, -OUTPUT\_FILE** -> Ruta al archivo de salida.
- **-I, -INDEX\_FILE** -> Ruta al archivo que contiene los índices mediante los que crearemos nuestro código para llevar a cabo la sustitución los valores.
- **-s, -INPUT\_SEPARATOR** -> Carácter mediante el que separaremos los elementos del archivo de entrada. Por defecto: tabulación.
- **-c, -COLUMNS** -> Índices de las columnas, separados por comas, para realizar las traducciones.
- **-f, -from** -> Columna en el archivo de índice cuyos elementos tomaremos como valor de referencia. Su valor predeterminado es 1. La numeración está basada en 1.
- **-t, -to** -> Columna en el archivo de índices de cuyos elementos tomaremos el valor que se utilizará en la sustitución. Su valor predeterminado es 2. La numeración está basada en 1.
- **-u, -remove\_untranslated** -> Activar opción para eliminar los registros que no han sido reemplazados.

#### Ejemplo de uso.

Queremos traducir una lista de identificadores de genes ENSEMBL (ENSG) a identificadores de proteína (ENSP) a partir de un diccionario que contiene la equivalencia entre ambos términos (ENSP-ENSG, archivo de dos columnas que sirve como índice). Para ello, el código debería ejecutarse de la siguiente manera:

```
standard_name_replacer.py -I ensp_ensg_dict_file -i ensg_ids_file -c 1 -f  
2 -t 1 -o ensp_ids_file -s "\t"
```

### 5.2.10 column\_filter

Con este script se pueden filtrar columnas de archivos tabulados cuyos elementos coincidan con un patrón específico.

- **-i, -input\_file** -> en este parámetro se especificará la ruta al archivo de entrada. Es común para los distintos scripts de cmdtabs, de manera que en este caso no es necesario especificarlo al usar el parámetro -t de manera obligatoria (si no usamos -t da error).
- **-x, -column\_index** -> índice de columna (en base 1) para usar como referencia.

- **-H, --header** -> indica si los archivos de entrada tienen cabecera. Por defecto está en falso, de manera que si el archivo tiene cabecera, basta con añadir el flag -H al comando.
- **--transposed** -> mediante este parámetro se especifica que queremos que las operaciones se ejecuten en FILAS en vez de columnas.
- **-t, --table\_file** -> archivo tabulado de entrada. USAR PARÁMETRO -i
- **-c, --column** -> mediante este parámetro especificamos las columnas que queremos que se muestren (en base 1), siguiendo el formato: x,y,z...
- **-f, --col\_filter** -> selecciona columnas filtrando mediante palabras clave (keywords), siguiendo el formato: x,y,z...
- **-p, --separator** -> especifica el separador de columna que se va a usar. Por defecto está el tabulador.
- **-k, --keywords** -> palabras claves para filtrar filas. Formato: key1\_col1&key2\_col1&key1\_col2&key2\_col2
- **-s {c,s}, --search {c,s}** -> 'c' se usa para especificar que queremos match en todas las combinaciones de keyword\_col mencionadas anteriormente (equivalente a un AND), mientras que 's' se usa para especificar que no es necesario que haya match en todas las combinaciones (OR). Por defecto 'c'.
- **-m {i,c}, --match\_mode {i,c}** -> si usamos 'i', el string debe contener la palabra que buscamos, mientras que 'c' especifica que tiene que ser un match exacto. Por defecto: i.
- **-r, --reverse** -> selecciona aquellas palabras que NO coinciden con la especificada.
- **-u, --uniq** -> eliminar elementos redundantes.
- **-h, --help** -> muestra la ayuda.

#### Ejemplos de uso.

En el primer ejemplo, queremos que se realice un filtrado en la primera columna de la tabla, de manera que se muestren las columnas 1 y 2 de aquellas en las que se encuentre la keyword exacta, mientras que en el segundo ejemplo, se acepta con que aparezca la palabra MONDO como parte de la palabra.

```
1    column_filter -t {input_file} -c 1,2 -f 1 -k MONDO:0008995 -s c -m c
2    column_filter -t {input_file} -c 1,2 -f 1 -k MONDO -s c -m i
```

### 5.2.11 excel\_to\_tabular

El script `excel_to_tabular.py` es una herramienta para transformar archivos de excel (.xlsx) en archivos tabulares y facilitar su manejo. **No parsea archivos .csv**. Sus argumentos de entrada son los siguientes:

- **-h, --help** -> Muestra la ayuda.
- **-c, --COLUMNS2EXTRACT** -> número de columna para extraer la información. Por defecto se extrae la información de la primera columna.
- **-i, --INPUT\_FILE** -> archivo con extensión .xlsx del cual extraer la información.
- **-o, --OUTPUT\_FILE** -> archivo de salida. Por defecto está definido como "table.txt"
- **-s, --SHEET\_NUMBER** -> número de la hoja del archivo .xlsx para extraer la información. Por defecto se extrae la información de la primera hoja.



## 6. report\_html

### 6.1 Descripción

Permite hacer informes interactivos a partir de cualquier tipo de información. Debe su influencia a lo que se conoce como *embedded Ruby*, una herramienta que permite escribir una plantilla con texto y que en medio de ese texto se puedan incluir etiquetas que indiquen que se debe ejecutar un fragmento de código en su interior, y que el resultado de éste código se incluya dentro del texto. En Python funciona de manera similar, usando de fondo la librería Mako para hacerlo posible.

Los archivos que contienen este tipo de instrucciones (denominados plantillas) se suelen definir con la extensión *.txt*. En el momento que se detectan las marcas especiales en la plantilla que recibe el script, Python sabe que tiene que *parsear* el texto y ejecutar el código que se encuentra dentro. Las marcas «<%» y «%>» se utilizan para que Python sepa que tiene que ejecutar ese código, manteniendo el resultado en memoria o evaluándolo, pero sin incluirlo en el texto final. Por otro lado, las marcas «\${» y «}» se utilizan para que Python ejecute el código y se incluya en el texto final. Adicionalmente, tiene algunas sentencias especiales para bloques de código condicionales y de iteración, para poder lidiar con los problemas de identación que tendría Python si no, y que nos permiten mostrar contenido de html y otros bloques de resultados dependiendo de si se cumple una determinada expresión dentro del if o no, o para repetir por ejemplo elementos de html de una lista más contenido que nos interese ( ej: «<li>\${item}</li>») siendo item la variable que se recorre dentro de un bucle for.

Ademas, si hemos cargado archivos a la hora de llamar al script, todos ellos se encontrarán disponibles dentro del entorno de ejecución en la plantilla. Serán cargados como lista de listas (emulando una estructura de tabla) dentro de un diccionario que se llama hash\_var y que será accesible a través del objeto "plotter", que será nuestro intermediario a la hora de comunicarnos tanto con los datos como con las funciones de ploteo que tiene la herramienta report\_html. Es decir, que si hemos cargado una tabla llamada "patients.txt", podremos acceder a ella dentro de la plantilla tal que así: «*plotter.hash\_var["patients.txt"]*»

Un ejemplo del uso de este tipo de marcas en un texto HTML sería el siguiente: Imaginemos que tenemos una tabla que provenía de un archivo llamado "patients.txt" que hemos cargado mediante la

llamada a la herramienta `report_html`, de manera que va a estar cargada dentro de `plotter.hash_vars`. Esta tabla tiene 2 columnas con un encabezado (`header`) que tiene los valores "name", "mutations", y vamos a partir de ella a generar una lista de diccionarios en la que se guarden los valores de cada fila de la tabla y a partir de ahí trabajaremos brevemente con esa tabla para enseñar varias de las funcionalidades que hemos comentado anteriormente:

```

1  <%
2      tabla = plotter.hash_vars ["patients.txt"]
3      cohort = []
4      header = tabla[0]
5      for row in tabla[1:]:
6          cohort.append({header[0]:row[0], header[1]: row[1]})

7      #Imaginemos que esta es la lista resultante de nuestro parseo
8      #cohort = [{"name": "Alfonso", "mutations": 2}, {"name": "Javier", "mutations": 0}]
9      %>
10
11      Patients with mutations:
12      <ul>
13          % for patient in cohort:
14              % if patient["mutations"] > 0:
15                  <li> Patient ${patient["name"]} has ${patient["mutations"]}
16                      mutations</li>
17              % else:
18                  <li> Patient ${patient["name"]} has no mutations</li>
19              % endif
20          % endfor
21      </ul>
22

```

En este caso, se crea una variable con una lista de diccionarios con datos de pacientes (pero sin plasmarlo en el report resultante) y posteriormente se usa un tag de lista no ordenada en html (`<ul>`) y se empieza a iterar en la lista de pacientes con el bloque de códigos `% for`. En cada iteración de la lista se mira si el paciente tiene mutaciones (bloque `%if`), en cuyo caso se dice el nombre del paciente y la cantidad de mutaciones que tiene. Si no se cumple esa condición (bloque `%else`), se devuelve el nombre del paciente, informando de que no tiene mutaciones. En ambos casos se genera un tag de lista (`<li>`) en el que se devuelve la evaluación del código dentro de cada bloque `{}$`.

Para un mayor conocimiento de otras funcionalidades que pueda ofrecer la librería Mako se puede acudir a la documentación de dicha librería (<https://docs.makotemplates.org/en/latest/index.html>).

## 6.2 Script

El script `report_html` acepta los siguientes argumentos:

- -t -> Path al template .txt
- -o -> Outputname del archivo que contiene el resultado de parsear la plantilla en Python
- -d -> Path separados por coma de los archivos que queremos tener en memoria para utilizar dentro del report. Por defecto, el formato de archivo utilizado es tsv, y son cargados como un hash/diccionario llamado `hash_var` (dentro del objeto `plotter`) en el que la key es el nombre del archivo y el valor es un array/lista de listas con los datos del archivo. Además, en las

funciones expandidas del report\_html (que se explican en el proximo apartado), podemos dar como argumento «id= nombre\_del\_archivo» para que acceda a los datos de los archivos.

### 6.2.1 Métodos del report\_html

#### Tablas

Crear tablas dentro del report\_html es tan sencillo como saber utilizar la función **table** (a traves del objeto plotter, es decir, plotter.table() ), la cual acepta los siguientes argumentos:

- **id**: Aquí indicaremos el nombre del archivo que deseamos utilizar para tomar los datos. Teniendo en cuenta que esta función la usaremos dentro de una plantilla, el nombre del archivo que introduzcamos debe coincidir con el que se pase por el argumento **-d** a la hora de llamar al script report\_html en la linea de comandos (no la ruta completa, solo el nombre del archivo y extension, es decir si en -d se hubiera especificado ./data/patients.txt desde el directorio de trabajo actual, el id del archivo en hash\_var seria "patients.txt").
- **header**: Acepta los argumentos true o false para indicar si los datos de entrada vienen con cabecero o no.
- **row\_names**: Acepta los argumentos true o false para indicar si los datos de entrada vienen con nombres de fila o no.
- **add\_header\_row\_names**: Se utiliza para que automaticamente añada como nombre de fila o nombre de columna una sucesión de numeros. Por defecto es false.
- **transpose**: Se utiliza para rotar la tabla. Por defecto es false.
- **text**: Sirve para indicar si en la tabla de entrada hay elementos con texto. Por defecto está configurado a false. Si en la tabla hay texto y no se especifica este parámetro a true, los valores de string aparecerán como 0.0.
- **fields**: Sirve para indicar un array de los numeros de las columnas que nos interesan de la tabla de datos de la que partimos.
- **border**: Sirve para indicar el grosor del borde de la tabla. Por defecto es 1
- **cell\_align**: Array en el cual se indica el alineamiento deseado por cada columna (left, center o right). El primer elemento del array se corresponde con la primera columna de la tabla, y sucesivamente iremos indicando el alineamiento que deseamos para cada columna (el array debe tener tantos elementos como columnas).
- **attrib**: Acepta un hash en el que pondremos los atributos que queremos de CSS para la tabla. Por ejemplo, attrib: { 'class' => 'table table-striped', 'style' => "background-color:black" }
- **styled**: Sirve para indicar el estilo de la tabla, y se le pueden pasar 2 valores: 'bs', que utiliza bootstrap para el estilo de la tabla y 'dt', que se utiliza para crear tablas interactivas.
- **func**: Sirve para pasar alguna funcion de formateo o transformacion de los datos de la tabla que queremos realizar antes de que finalmente se muestra ya en el informe de html.

Ademas, una vez cargado el archivo, podemos modificar a nuestro antojo el dataset dentro de su "key" en el diccionario de hash\_var (o crear una version modificada en otro key distinto, ya que cualquier tabla guardada en hash\_var será accesible a través del parámetro "id" de la funcion table o cualquier funcion de ploteo del report\_html). Adicionalmente, tambien podemos definir una funcion que sea aplicada antes de mostrar la tabla como hemos comentado anteriormente a traves del parametro "func" de la funcion table.

Vamos a suponer que tenemos una tabla de datos numéricos y que queremos multiplicar cada uno de los valores por 100 antes de mostrar la tabla, definiendo una funcion para transformar los datos que pasaremos mediante el parámetro "func". Para ello, podríamos diseñar una plantilla como la que se muestra en el siguiente ejemplo:

```

1  <% def transform_table(data_array):
2      for idx, row in enumerate(data_array):
3          if idx == 0: continue #Skipping header
4          for colidx in range(len(row)):
5              row[colidx] = row[colidx] * 100
6      %>
7
8  ${plotter.table(id= 'table1_header.txt', header= True, styled= 'dt',
9      attrib= {'class' => 'table'}, func = transform_table )}
```

## CanvasXpress

Los métodos incluidos en el report\_html pertenecientes a este bloque basan su funcionamiento en CanvasXpress, una librería de JavaScript que permite crear representaciones gráficas interactivas. Actualmente, disponemos de los siguientes métodos:

- **line** -> line(id: 'data.txt', data\_format: 'one\_axis', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **stacked** -> stacked(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **barplot** -> barplot(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **heatmap** -> heatmap(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **boxplot** -> boxplot(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **pie** -> pie(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **corplot** -> corplot(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], correlationAxis: 'samples', config:{} )
- **scatter2D** -> scatter2D(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **scatterbubble2D** -> scatterbubble2D(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )
- **circular** -> circular(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], ring\_assignment: [], ringsType: [], ringsWeight: [], config:{} )
- **circular\_genome** -> circular\_genome(id: 'data.txt', header: false, row\_names: false, add\_header\_row\_names: false, transpose: false, fields: [], config:{} )

Cada método del report\_html perteneciente a este bloque se encarga por debajo de editar al mínimo necesario los parámetros de config del CanvasXpress para obtener la representación necesaria. Los parámetros mínimos para usar cualquiera de estas funciones y generar un gráfico serían el "id" del archivo, y si la tabla no tiene cabecera o nombre de filas, fijar el parámetro add\_header\_row\_names a True y añadir True también en header/row\_names (o ambos) según cual sea el que no tengamos (y necesitamos que add\_header\_row\_names nos llene por defecto). Tambien necesitamos

Cualquier modificación extra que se desee, como el tamaño, orientación, etc, habrá que indicarla específicamente. Las opciones más básicas se pueden pasar como parámetros a la función de ploteo concreta tal y como se muestra arriba (header, transpose, etc), mientras que parámetros

más concretos que sean de CanvasExpress se pasan a través de un diccionario en el parámetro "config" que aceptan dichas funciones (y en el que se añadirán atributos que acepte canvasxpress para configurar los graficos). En el manual del CanvasXpress puede consultarse toda esta información, y además, se pueden modificar de forma interactiva los parámetros de las gráficas en la propia web, lo que nos permite tanto conocer el nombre del parámetro como el valor que queremos poner para que se vea de la forma que deseamos antes de escribirlo en la plantilla para el report (<https://canvasxpress.org/examples/bar-1.html>, hacer click derecho en la gráfica, Explore, Search parameters, y hacer click en la lupa (browse properties by category)).

### Embeber archivos dentro del report

- embed\_img(img\_file, img\_attribs = nil)
- embed\_pdf(pdf\_file, pdf\_attribs = nil)

El primer argumento que reciben estas funciones debe ser un path al nombre del archivo que deseamos embeber con su extensión correspondiente. El segundo argumento recibe opciones de html relativas a la imagen que queremos embeber (height, width, etc). Así, un ejemplo real sería:

```
embed_img('temp/img/pre_clean/pair1/per_base_quality.png', 'width="600px"  
", height="600px")
```





## 7. semtools

### 7.1 Descripción

Semtools es una librería de Python para trabajar con ontologías en formato .obo cargándolas como un objeto y realizando una serie de operaciones en base a los datos que se encuentran en la propia ontología. Tiene múltiples usos, entre los que se incluyen la exploración de la ontología, el análisis y manipulación de datos relacionados con ella, traducción de términos, entre otros. Todas sus funcionalidades se describen en los siguientes apartados.

### 7.2 semtools.py

El script principal, semtools.py, trabaja con archivos de dos columnas, siendo una la que incluye los identificadores (IDs) y la otra las entidades asociadas separadas por punto y coma (;). Por ejemplo, un archivo que incluya los identificadores de pacientes en la primera columna y en la segunda sus perfiles de fenotipos HPO. Los argumentos de entrada al programa son los siguientes:

- **-d, --download** -> Descarga los archivos .obo desde las fuentes oficiales: configurar como MONDO, HPO, GO, EFO. Esta opción tiene un parámetro extra que sirve para listar dónde se encuentran las fuentes de información: list. **¡¡MUY IMPORTANTE!!** Para evitar los temidos conflictos de versiones entre ontologías y para procurar que todos los miembros del grupo empleamos el mismo archivo, al usar la ruta al archivo .obo SIEMPRE hay que recavar la ruta de forma programática a partir de la salida del comando semtools.py -d list. Este comando nos dará el path a los archivos .obo de interés dentro de la gema semtools y evitará posibles problemas en caso de que haya actualizaciones de la librería. La salida del comando se puede redireccionar para su utilización mediante comandos de bash. Por ejemplo, para configurar la ruta al archivo MONDO.obo:

```
1 mondo_path='semtools.rb -d list | grep 'MONDO'
2 echo $mondo_path
3
```

- **-i, --input\_file** -> Path al archivo de dos columnas con datos del perfiles fenotípicos. Debe tener el identificador (ID) de los registros a trabajar en la primera columna y en la segunda las entidades asociadas separadas por punto y coma (;).
- **-o, --output\_file** -> Path al archivo de salida.
- **-I, --IC** -> Cálculo del coeficiente de información (IC). Se puede configurar a 'ont' para que se use la estructura de la ontología o 'prof' usando los perfiles de fenotipos.
- **-O, --ontology\_file** -> Path a un archivo de ontología.
- **-T, --term\_filter** -> Si se especifica un término de la ontología, se considerarán solo los términos descendientes de aquél que ha sido especificado.
- **-t, --translate** -> Configurar para que traduzca términos a códigos o viceversa. Se puede configurar a 'names' o 'codes'.
- **-s, --similarity\_method** -> Método para el cálculo de la similitud semántica entre los perfiles de fenotipos. Usar entre: 'resnik', 'lin' o 'jiang\_conrath'
- **--reference\_profiles** -> Path al archivo tabulado donde la primera columna corresponde con el identificador del paciente y la segunda con los términos de la ontología separados por un elemento. \*\* consultar
- **-c, --clean\_profiles** -> Busca en los perfiles si hay términos con relaciones padre-hijo y términos obsoletos en la ontología para eliminarlos.
- **-r, --removed\_path** -> Path al archivo donde se almacenan los perfiles que han sido eliminados.
- **-u, --untranslated\_path** -> Path al archivo donde se almacenan los términos que no han sido traducidos.
- **-k, --keyword** -> Regex usada para obtener referencias cruzadas (xref) entre términos de la ontología.
- **-e, --expand\_profiles** -> Expandir perfiles de fenotipos usando términos parentales ('parental') o añadiendo nuevos perfiles ('propagate'). Con 'propagate' se anota por cada término qué IDs tiene asociado. Por ejemplo, si tenemos varios pacientes que comparten el término "Intellectual disability, severe", en su expansión también compartirán el término "Intellectual disability". Hay que tener en consideración que esto puede sumar un elevado número de términos para llevar a cabo el análisis.
- **-U, --unwanted\_terms** -> Para la expansión de perfiles, añadir separados por coma aquellos términos de la ontología que no quieran ser empleados para dicha expansión.
- **-S, --separator** -> Separador empleado para los perfiles de términos. Por defecto es punto y coma (';').
- **-n, --statistics** -> Devuelve un archivo de estadísticas básicas calculadas a partir de los perfiles de fenotipos.
- **-l, --list\_translate** -> Cuando se incluye una lista de entrada en lugar de un archivo, traducir a nombres ('names') o a códigos ('codes').
- **-f, --subject\_column** -> En el archivo de entrada, número de la columna que contiene los identificadores (pacientes, enfermedades, genes...)
- **-a, --annotations\_column** -> En el archivo de entrada, número de la columna que tiene las entidades asociadas (términos HPO, GO...)
- **--list\_term\_attributes** -> A partir de la lista de términos, si se configura (TRUE) se imprime en pantalla el término, su traducción y su nivel en la ontología.
- **-R, --root** -> Término a considerarse como raíz de la ontología.
- **--xref\_sense** -> Sentido para usar las referencias cruzadas (xref). Por defecto: xref-ontology. Se puede configurar a ontology-xref.

- **-C, --childs** -> Lista de términos separados por coma para emplearse como términos hijo. Añadiendo un término, busca todos sus hijos. Esto se emplea para la expansión de perfiles. Este argumento tiene varios modificadores:
  - **r** -> Relación padre/hijo; a -> devuelve los ascendentes (que no los términos hijo).
  - **n** -> Devuelve los términos como nombre (no como código).
  - **h + número** -> Indica cuántos saltos de nivel dar en la ontología.
 Para configurar los modificadores se debe especificar sobre qué término hacerlo. Por ejemplo: -C anh5/MONDO:0019200 (busca hasta cinco ascendentes del término MONDO:0019200 y los traduce de código a nombres).
- **-a, --annotations\_column** -> Número de la columna que tiene las entidades asociadas.
- **-C, --childs** -> añadiendo un término a continuación, busca todos los términos hijos dentro de la ontología. Se puede hacer una búsqueda sobre varios términos si se separan por comas. Tiene varios modificadores: r -> relación padre/hijo; a -> te da los ascendentes (que no los términos hijo); n -> te da los términos como nombre (no como término); h + número -> es para indicarle cuántos saltos de nivel dar en la ontología. Para configurar los modificadores se debe configurar sobre qué término hacerlo. Por ejemplo: -C anh5/MONDO:0019200 (busca hasta cinco ascendentes del término MONDO:0019200 y los traduce de código a nombres).
- **-c, --clean\_profiles** -> Eliminar ancestros, descendientes y términos obsoletos de los perfiles.
- **-d, --download** -> Descargar el archivo .obo desde su fuente oficial. Actualmente hay cuatro ontologías disponibles: MONDO, HPO, GO, EFO. Esta opción tiene un parámetro extra que sirve para listar dónde se encuentran las fuentes de información: list. **¡¡MUY IMPORTANTE!!** Para evitar los temidos conflictos de versiones entre ontologías y para procurar que todos los miembros del grupo empleamos el mismo archivo, al usar la ruta al archivo .obo SIEMPRE hay que recavar la ruta de forma programática a partir de la salida del comando semtools.rb -d list. Este comando nos dará el path a los archivos .obo de interés dentro de la gema semtools y evitará posibles problemas en caso de que haya actualizaciones de la gema. La salida del comando se puede redireccionar para su utilización mediante comandos de bash. Por ejemplo, para configurar la ruta al archivo MONDO.obo:

```

1 mondo_path='semtools.rb -d list | grep \'MONDO\''
2 echo $mondo_path
3

```

- **-e, --expand\_profiles** -> Expandir perfiles añadiendo ancestros de los términos presentes. Las opciones son 'parental' para una expansión por términos parentales hasta la raíz y 'propagate' anota por cada término qué IDs tiene asociado. Por ejemplo, si tenemos varios pacientes que comparten el término "Intellectual disability, severe", en su expansión también compartirán el término "Intellectual disability". Hay que tener en consideración que esto puede sumar un elevado número de términos para llevar a cabo el análisis.
- **-f, --subject\_column** -> Número de la columna que tiene los IDs.
- **-I, --IC** -> Obtener IC a nivel de ontología y de frecuencia observada.
- **-i, --input\_file** -> Path al archivo de entrada, que debe ser de dos columnas con el identificador (ID) de los registros a trabajar en la primera columna y en la segunda las entidades asociadas separadas por punto y coma (;).
- **k, --keyword** -> Expresión regular utilizada para capturar los términos que aparecen en la etiqueta xref (cross-reference) a la hora de traducir por xref un archivo de perfiles.
- **-l, --list\_translate** -> Traduce a nombres (names) o a códigos (codes) una lista de entrada (sin

indicar perfiles, sino un término por línea).

- **-list\_term\_attributes** -> Devuelve una lista con los atributos de cada término dentro de la ontología.
- **-n, -statistics** -> Genera unas estadísticas generales sobre el archivo de perfiles de entrada.
- **-O, -ontology\_file** -> Path al archivo de la ontología.
- **-o, -output\_file** -> Path y nombre del archivo de salida.
- **-R, -root** -> Término para ser considerado como raíz de la ontología
- **-r, -removed\_path** -> Path donde escribir el archivo de perfiles eliminados tras la limpieza, en caso de que algún perfil quede vacío.
- **-reference\_profiles** -> Archivo empleado como perfiles de referencia. Tiene los identificadores en la primera columna y las entidades asociadas (perfiles) en la segunda, separados por comas.
- **-S, -separator** -> Separador utilizado en el archivo de entrada para separar los términos en los perfiles. Por defecto se utilizan las comas.
- **-s, -similarity** -> Calcular similitud semántica entre los IDs en base a sus perfiles.
- **-T, -term\_filter** -> Opción para especificar un término concreto de la ontología con el fin de que al realizar una limpieza de perfiles sólo los términos que sean descendientes del término especificado sean conservados.
- **-t, -translate** -> Permite llevar a cabo una traducción de los perfiles sobre un archivo de perfiles. Indicar 'names' para traducir de códigos a nombres y 'codes' para traducir de nombres a códigos.
- **-U, -unwanted\_terms** -> Lista de términos que no deseamos incluir a la hora de expandir los perfiles. Deben ir separando por comas.
- **-u, -untranslated\_path** -> Path donde escribir el archivo de términos no traducidos, en caso de que algún término no haya podido ser encontrado para traducir en la ontología.

### 7.2.1 Ejemplos

Para comprender mejor este script, se listan a continuación diferentes ejemplos de su uso:

- Archivo de entrada (OMIM\_2\_hpnames) con términos OMIM (OMIM ID) en la primera columna y los fenotipos HPO asociados a cada enfermedad separados por punto y comas (;) en la segunda columna.
  - Traducción de fenotipos a códigos HPO:

```
1     semtools.rb -i OMIM_2_hpnames -o OMIM_2_hpcodes -S ';' -O
2     ontologies/hp.obo -t 'codes' -u ./untranslated_terms.txt}
```

- Traducción de fenotipos a nombres y cálculo de la similitud semántica entre los IDs en base a sus perfiles HPO asociados, devolviendo la lista de OMIM IDs no traducidos en el archivo untranslated\_terms.txt:

```
1     semtools.rb -i OMIM_2_hpcodes -o OMIM_2_hpnames -S ';' -O
2     ontologies/hp.obo -t 'names' -u ./untranslated_terms.txt -s
```

- Efectuar la traducción de los fenotipos a códigos y la limpieza de los perfiles fenotípicos, calcular la similitud semántica y el IC a nivel de ontología y de frecuencia observada. Devolver la lista de OMIM IDs no traducidos en el archivo untranslated\_terms.txt:

```
1      semtools.rb -i OMIM_2_hpnames -o OMIM_2_hpcodes_clean -  
2      S ';' -O ontologies/hp.obo -c -r ../removed_profiles.txt -s -I  
      -t 'codes' -u ./untranslated_terms.txt
```

- Para la búsqueda de términos hijo a partir de otros de interés. Por ejemplo, imaginemos que queremos obtener todos los términos que derivan de los MONDO:0019200 (retinitis pigmentosa) y MONDO:0018998 (Leber congenital amaurosis). Semtools nos puede dar la lista de todos los descendientes de los términos incluidos en la búsqueda. Vamos a usar como ejemplo la ontología MONDO:

```
1      semtools.rb -C MONDO:0019200,MONDO:0018998 -O mondo.obo  
2
```

- Del mismo modo, podríamos buscar la lista de términos parentales de ambos términos si añadimos una serie de modificadores al argumento -C como se explicó en su descripción. Vamos a poner como ejemplo buscar los parentales de MONDO:0019200 y MONDO:0018998, que no sean más de 3 y que además los traduzca a nombres. Asimismo, para evitar perder de cuál de los dos términos son las relaciones parentales, añadimos el modificador r, que nos permite conocer si los términos buscados pertenecen a uno u otro MONDO:

```
1      semtools.rb -C 'rank3/MONDO:0019200,MONDO:0018998' -O  
      mondo.obo  
2
```





## 8. NetAnalyzer

NetAnalyzer es una librería de Python (migrada a partir de una gema de Ruby, <https://rubygems.org/gems/NetAnalyzer>) que consiste en una *suite* de herramientas para la gestión y análisis de redes multipartitas y otras operaciones asociadas.

En Picasso, hay que cargar Python para poderla usar. Recuerda asimismo que para llamar a los programas de Python no es necesario añadir su extensión (.py):

```
source ~soft_bio_267/initializes/init_python
```

La librería NetAnalyzer consta de varios scripts que realizan distintas funciones:

- **netanalyzer**: Script principal con el que se pueden realizar distintos análisis de redes.
- **ranker**: Prioriza los nodos (genes) de una matriz de embedding tipo kernel en función de uno o más nodos semilla (seed).
- **text2binary\_matrix**: Transforma una red representada en un archivo de texto plano de tipo parejas o matriz al formato binario [NPY](#) y, además, permite calcular métricas sobre la red.
- **randomize\_clustering**: Para aleatorizar el agrupamiento los nodos de una red.
- **randomize\_network**: Sirve para aleatorizar los nodos o enlaces de una red respetando su estructura.
- **net\_explorer**: Sirve para explorar ciertos nodos dentro de una red de partida.

### 8.0.1 Instalación desde GitHub

```
pip install git+https://github.com/seoanezonjic/NetAnalyzer.git pip install langchain
```

### 8.1 netanalyzer

NetAnalyzer es una librería en Python diseñada para facilitar el análisis de redes biológicas y complejas, integrando funcionalidades avanzadas para la exploración estructural, visualización y modelado de grafos. Esta herramienta es especialmente útil para investigadores en bioinformática, biología de sistemas y ciencia de datos que trabajan con datos de interacción, como redes de proteínas, genes o compuestos.

Esta herramienta permite realizar distintos análisis clave sobre redes, incluyendo la proyección de datos, la detección de comunidades, la visualización interactiva y la obtención de representaciones vectoriales (embeddings) de los nodos.

- **Proyecciones.**

- Proyectar información externa (por ejemplo, genes expresados diferencialmente, módulos funcionales, etc.) sobre una red de referencia disminuye número de capas de una red basándose en las conexiones de una capa excluida.
- Muy útil cuando se buscan relaciones ocultas entre dos conjuntos, facilitando el análisis funcional o la priorización de nodos relevantes en el contexto de la red.
- Ejemplo: Pasar de una red Fenotipo-Paciente-Mutación a una Paciente-Mutación, conectando las dos capas en función al número de nodos comunes entre paciente y mutación.

- **Embedding de grafos.**

- Sirve para definir similitud entre nodos por distintos métodos de procesamiento en la información de contexto en redes.
- La idea es capturar la información de la red para transformarla en números con los que poder trabajar usando algoritmos de machine learning o estadísticas.
- Los principales bloques están en el clásico enfoque por **Kernels** (para análisis de similitud estructural entre nodos o subgrafos, por ejemplo: diffusion kernel, regularized Laplacian, etc.) y el enfoque adicional por *deep learning*: **node2vec** (basado en *random walks* sesgados).
- El resultado es una matriz NumPy (almacenada en binario) de tamaño n x n, donde n es el número de genes (o nodos) considerados en la red.

### Análisis de clústeres.

- Implementa algoritmos de detección de comunidades, tanto disjuntas como superpuestas.
- Soporta análisis jerárquico, modularidad y métodos basados en enlaces.
- Permite realizar métricas específicas sobre clústeres predefinidos<sup>1</sup>.
- **Visualización de redes.**
  - Representación gráfica de redes con soporte para distintos layouts.
  - Posibilidad de exportar imágenes de alta calidad o visualizaciones interactivas.

Entre sus aplicaciones, podemos encontrar el análisis funcional de genes en redes biológicas, la priorización de candidatos terapéuticos en redes moleculares, la exploración de modularidad en datos de interacción, la reducción de dimensionalidad para visualización o clustering y el estudio de relaciones entre nodos a partir de estructuras latentes.

#### 8.1.1 Inputs

NetAnalyzer utiliza por defecto una **representación por parejas** para redes no dirigidas tanto ponderadas como no ponderadas<sup>2</sup>. El otro formato en el que se puede mostrar una red es en **formato matricial**. Esta forma puede darse en texto plano (*colname*, *rowname* y valores en cada casilla, todo en un archivo tabulado) o en formato binario. Además, también acepta como información externa, ontología asociada por capa en la red, para ello se debe utilizar el flag -O, indicando la ontología específica de alguna de las capas (ejemplo; -O layer1:HPO;layer2:MONDO).

---

<sup>1</sup>Generalmente, por herramientas como cdlib\_clusterize.py

<sup>2</sup>Se está pensando en la posibilidad de expandir la herramienta para utilizar también redes dirigidas

**opción -i** Los archivos con las conexiones nodo-nodo, bien en formato parejas, matricial plano o matricial binario.

**opción -f** Para indicar el formato de los datos de entrada, pudiendo ser "bin", "pair", "matrix"

**opción -l** para indicar las layers con su respectivo regex para encontrar los nodos. Formato, layer1,regex1;layer2,regex2;...

**opción -n** indicar el nombre de los nodos asociados a filas o columnas en la matriz. Pudiendo indicarse sólo un archivo, que servirá para filas y columnas, o dos archivos separados por "," que indicarán filas y columnas.

**opción -s** indica el separador utilizado.

**opción -O** Especificar ontologías asociadas por cada capa, en formato layer1,ont1.obo;layer2,ont2.obo.

**opción -A –attributes** Para obtener los atributos de los nodos dentro de una red. Los atributos que se desean pueden especificarse por ",". Devolviéndose un archivo node\_attributes.txt.

**-attributes\_summarize** Incorporando este flag se darán las métricas de los atributos de los nodos de una red, resumiendo los datos a nivel estadístico.

**Opción -graph\_attributes** Indicando los atributos de la red en su conjunto. Indicando las distintas métricas por ",". Devolviéndose un archivo "graph\_attributes.txt".

**-dsl\_script** Función muy versatil con la que añadir de manera custom un conjunto de comandos que quieras que vaya ejecutando el net. Su funcionamiento es: 1) Incorporas la red y parámetros de layers elementales junto al flag dsl\_script y el path al archivo dsl, donde el programa se encargará de interpretar las instrucciones.

**opción -filter\_connected\_components** elimina aquellos nodos que estén en comunidades (componentes conexas) menores o iguales al número de nodos que establezcamos.

### Ejemplo de dsl

Un ejemplo donde pedimos: 1) Obtener el degree de los nodos, 2) hacer una proyección y 3) guardar la matrix de proyección y su estadística.

```

1 get_node_attributes attr_names=['get_degree'] layers='gene' summary=
    True output_filename='preproc_metrics'
2 get_association_values ['gene'] 'group' 'jaccard' add_to_object=True
3 write_matrix ('associations',('gene','gene'),'jaccard')
    output_filename='similarity_matrix_bin'
4 write_stats_from_matrix ('associations',('gene','gene'),'jaccard') ,
    stats_from_matrix_jaccard_1

```

```

1 netanalyzer -i matrix_adj.npy -f "bin" -l "layername" -n matrix_adj.lst
2 netanalyzer -i pair_table -f "pair" -l "layername"
3 netanalyzer -i matrix.txt -f "matrix" -l "layername"

```

**WARNING: Fallos comunes en los inputs**

- El **formato bin** no guarda los nombres de los nodos, así que hay: (1) Un archivo .bin con la información de la matriz y (2) un archivo .lst para cada capa que contienen los nombres de los nodos.
- Al utilizar la **opción -l**, tener en cuenta que los nodos se filtran de manera secuencial y eliminatoria. Así, los nodos cogidos para la primera capa mencionada luego no se cogerán para la capa dos aunque su expresión regular coincida, y así sucesivamente.

### 8.1.2 Proyecciones

Como ya se ha mencionado, las proyecciones de red son una operación usual en el caso de redes multipartitas (aquellas con distintas capas de información). De tal modo, suele ser útil pasar de 3 capas a 2 o de 2 a 1, pasando a una monopartita. Esto nos permite encontrar conexiones entre nodos de una misma capa, de forma indirecta. Ejemplo, Fenotipo-Paciente-Gen -> Fenotipo-Gen.

#### Línea de comandos

**opción -m** para indicar el método con el cual se van a calcular la fuerza de las asociaciones. Para ello, se pueden tomar distintas métricas: simpson; jaccard; pcc; csi; geometric; cosine; hipergeometric(\_bh,\_bf,\_elim,\_weight); counts; transference; correlation; umap; pca; bicm (utilizado para poder quedarnos con conexiones significativas en las proyecciones sobre bipartitas).

**opción -u** para indicar la capa o capas a proyectar (separadas por «,», respecto a una dada (separada por «;»). Por ejemplo, -u 'hpo,genes;diseases', para realizar la proyección FENOTIPO-GENES, utilizando como capa para medir las conexiones DISEASES.

**opción -N** para eliminar las autoconexiones en caso de emplear una mixedlayer sobre la que proyectar una 3 capa. Por ejemplo, si tenemos una red FENOTIPO-PACIENTE-MUTACIÓN, y deseamos determinar la relación FENOTIPO-MUTACIÓN, tendremos que realizar una proyección indicando -N, para eliminar las asociaciones FENOTIPO-FENOTIPO y MUTACIÓN-MUTACIÓN.

**opción -a** para indicar el nombre del output de las asociaciones.

**opción -T** para indicar el número de *cores* a utilizar.

### ¿Por qué hipergeometric tiene distintas modificaciones?

Esta métrica es el -log del p-valor obtenido a partir de la distribución hipergeométrica de los nodos compartidos <sup>a</sup>. De tal modo, un p-valor bajo indica que es recurrente que dos fenotipos salgan juntos en una misma enfermedad. Una primera corrección estadística es que, dado que estamos tratando múltiples hipótesis a la vez, tenemos que efectuar una corrección por bonferroni (por el \_bf) o benjamini hochberg (por el \_bh).

Por otro lado, se puede romper la suposición de que los dos factores son independientes. Por ejemplo, podemos estar estableciendo ontologías que, en realidad, significan conceptos muy semejantes, siendo más probable que estás salgan asociadas por azar. Para paliar esta situación, podemos aprovecharnos de la información que nos proporciona GO. Así, se entiende ahora la existencia del método \_elim, que elimina GO directamente emparentadas, y del \_weigh que pondera respecto al grado de parentesco.

Por supuesto, para conseguir que estos dos últimos métodos funcionen, se tiene que añadir el archivo.go, con el parámetro -O 'nombrecapa:path/a/ontología' .

<sup>a</sup>referenciaaquí

#### 8.1.3 Análisis de cluster.

Esta subsección abarca las funcionalidades de: (1) Descubrimiento de comunidades, (2) Medida por comunidades (en general y por cada comunidad extraída) y comparación de familias de clusters y (3) Expansión de clusters. Hasta la fecha, el descubrimiento y medida de comunidades se basa en las capacidades extraídas de la librería CDLib.

##### Línea de comandos

(1) Descubrimiento de nuevas comunidades Antes de nada, es importante mencionar que las comunidades requieren de especificar la red ambiente en las que se encuentran (por el comando -i para añadir la red sobre la que se está realizando el estudio).

opción -G para especificar clusters que ya se conocen. Hay que meter el *path* de un archivo, con formato: Primera columna, id cluster; Segunda columna, id del nodo de la red. Para casos de clusters de menor tamaño, también se puede utilizar el formato string: idcluster;elm1,elm2,elm2,etc.

opción -b Nombre del algoritmo o método de clustering que se desea implementar. Haciendo los métodos para comunidades no solapantes y los solapantes.

opción -B Opciones adicionales para los métodos de clustering. Cada método tiene una serie de argumentos que pueden darse en este flag y que permiten modificar su comportamiento. El formato de entrada de este argumento es el siguiente: -a "opción1" : valor1, "opción2" : valor2, etc.

opción -seed Con esto se especifica la semilla para la generación del clusterizado (basada en la semilla de aleatorización de numpy).

ón -output\_build\_clusters Output para las comunidades obtenidas.

##### (2) Medida de comunidades

opción -M (output file → 'group\_metrics.txt') Cálculo de diversos tipos de métrica sobre cada cluster. En lo que respecta a las métricas generadas por la propia clase Netanalyzer, nos encontramos

con el *comparative\_degree*, con fórmula  $ratio = \frac{edges_{cluster}}{edges_{total}}$  y con el *avg\_sht\_path(ASP)*, que calcula la distancia mínima media entre los nodos de un cluster. Las demás métricas son todas aquellas disponibles por la librería CDLIB. Las métricas son: 'size', 'avg\_transitivity', 'internal\_edge\_density', 'conductance', 'triangle\_participation\_ratio', 'max\_odf', 'avg\_odf', 'avg\_embeddedness', 'average\_internal\_degree', 'cut\_ratio', 'fraction\_over\_median\_degree', 'scaled\_density'. Excluyendo a, 'surprise', 'significance', 'comparative\_degree', 'avg\_sht\_path', 'node\_com\_assoc'.

opción -R para especificar clusters externos que quieren compararse con los ya añadidos o descubiertos previamente. Uso adecuado, usar -G para añadir grupos externos conocidos y -R cuando se quiera comparar una nueva familia externa con los anteriores. Este flag se usará cuando queramos activar el modo de métricas del cdlib sobre un archivo de clústers previamente creado, por ejemplo, queremos obtener la mutual information entre una familia de clusters y otra de referencia:

```
1 netanalyzer -i 'red.txt' -R 'clusters.txt' -G 'external.txt'
```

#### **Si los clustering son solapantes, se debe especificar con –overlapping\_communities**

opción -S (output file → 'group\_metrics\_summarized.txt') Activa el cálculo de métricas de modo sumarizado para el conjunto de los clusters. Es importante mencionar las métricas disponibles para añadir a este modo son: 'size', 'avg\_transitivity', 'internal\_edge\_density', 'conductance', 'triangle\_participation\_ratio', 'max\_odf', 'avg\_odf', 'avg\_embeddedness', 'average\_internal\_degree', 'cut\_ratio', 'fraction\_over\_median\_degree', 'scaled\_density'. Excluyendo a, 'surprise', 'significance', 'comparative\_degree', 'avg\_sht\_path', 'node\_com\_assoc'.

output\_metrics\_by\_cluster Output para métricas por cluster.

output\_summarized\_metrics Output para métricas descriptivas resumiendo el promedio de todos los clusters.

### **(3) Expansión de clusters**

opción -x sht\_path Para expandir los clusters de la siguiente manera. Incorpora aquellos nodos en el shortest path entre los nodos que forman la comunidad (de momento sólo está implementado sht\_path) (output file → "expand\_clusters.txt")

opción -one\_sht\_pairs Para especificar que sólo se tengan en cuenta una de las shortest path de todas las posibles entre dos nodos dados.

-output\_expand\_clusters Para especificar el output para el caso del cluster expandido.

#### **8.1.4 Visualización de redes.**

opción -g es para dar el nombre del output del archivo de grafo. Esta opción tiene un parámetro a tener en cuenta, *–graph\_options 'attr=attrvalue,attr2=attrvalue2,etc'*, para indicar cómo y con qué programa representar la red. Por ello, uno de los parámetros obligatorios es indicar el programa de visualización con 'method=x', donde x puede ser: sigma, elgrapho, graphviz y cytoscape. En orden de mayor a menor rendimiento para procesar redes de gran tamaño.

opción -r NODE1,NODE2... para indicar nodos de referencia de un color determinado.

### Profundicemos en las opciones de visualización

- Sigma. Alto rendimiento, sirve para redes muy grandes. Usar en vez de el grapho si la red es bastante grande, más de 10k nodos y millones de conexiones. Como mucho acepta **group=layer**, como atributo complementario.
- Elgrapho. Acepta como atributos: layout=hairball, forcedir, chord, radial y cl (para clusters), que acepta a su vez un step=número\_de\_iteraciones (por defecto = 30); group=layer, para colorear las distintas capas.
- Graphviz. Tiene por defecto layout=dot (árbol jerárquico), pero también puede emplearse los valores neato (perspectiva radial) y sfdp (repulsión electroestática). Este último es el único que no genera un report interactivo
- Cytoscape. Es ideal para redes pequeñas y tiene muchas funcionalidades.

#### 8.1.5 Embeddings: Kernels y node2vec

##### Aproximación al concepto de Kernel.

Una de las cuestiones más interesantes que pueden realizarse en una red es: ¿Cuál es el parecido que hay entre dos nodos? La respuesta a esta pregunta puede llegar a tener gran relevancia. Por ejemplo, si encontrásemos una red de genes en la que dos de ellos poseen una alta semejanza, podríamos inferir que tienen funciones semejantes, que comparten ciertos mecanismos en común, etc. Pero ¿Cómo definir la distancia o semejanza entre dos nodos?, para ello vamos a algunos conceptos del Álgebra lineal. En concreto, si a cada nodo lo consideramos como un vector o punto en un espacio vectorial n-dimensional, para cada par de nodos podemos medir el producto escalar o la distancia entre ellos, respectivamente.

En primero lugar, el producto escalar entre vectores haría las veces de "semejanza" entre los mismos, pudiendo valer valores positivos o negativos. Esta operación debe cumplir las condiciones de simetría, linealidad y definida positiva<sup>3</sup>.

Por otro lado, en lo que respecta a la distancia, necesitamos "algo" que asocie a cada par de nodos un número. En concreto, para que sea una distancia, necesitamos que:

- I- El valor devuelto siempre sea no negativo. Siendo cero, sólo en caso de que los nodos sean iguales.
- II- La operación es simétrica ( $\text{dist}(\text{nodeA}, \text{nodeB}) = \text{dist}(\text{nodeB}, \text{nodeA})$ ).
- III- La distancia entre dos nodos nunca será mayor que la suma de las distancias pasando por un tercero. Siguiendo el ejemplo,  $\text{dist}(\text{nodeA}, \text{nodeB})$  nunca será mayor que  $\text{dist}(\text{nodeA}, \text{nodeC}) + \text{dist}(\text{nodeC}, \text{nodeB})$ .

En realidad, esto es algo que ya experimentamos en el día a día, cuando caminamos o medimos en una regla, asociamos dos extremos (o un conjunto) a un valor siguiendo estas condiciones, simplemente hemos abstraído un concepto que ya conocemos y utilizamos.

En concreto, para el caso de las redes necesitamos representar cada nodo en un espacio vectorial. Así, cada nodo va a aparecer representado como un vector, dentro de un espacio vectorial y las medidas de semejanza por producto escalar partirán de la matriz de adyacencia<sup>4</sup>. Es a esta matriz a la que se le realizarán distintos tipos de modificaciones<sup>5</sup>, para obtener un **Kernel**. Entonces

<sup>3</sup>Una operación entre dos vectores es definida positiva si, al realizar la operación entre dos vectores **iguales** el resultado es mayor que cero o cero en caso del vector nulo

<sup>4</sup>Matriz que representa las conexiones presentes en una red. De manera que, cada fila y columna representa los nodos, quedando en cada casilla el valor 0 o 1, en función a la ausencia o presencia de conexión, respectivamente.

<sup>5</sup>las operaciones realizadas sobre la matriz de adyacencia o la laplaciana dependerán del espacio vectorial que quiera

**¿Qué es un kernel?** es una operación matemática capaz de almacenar directamente una medida de semejanza entre vectores (los nodos), por medio del producto escalar, sin necesidad de computar las coordenadas exactas del nodo en el espacio vectorial. En la práctica, el sistema representa un kernel como una matriz nxn, donde n es el número de nodos de la red y cada valor representa el producto escalar entre el nodo de la fila i con el de la columna j.

### Espacios vectoriales y One-hot encoding

Para poder utilizar el kernel, necesitamos codificar los nodos como vectores dentro de un espacio vectorial. Para esto, resulta de gran utilidad el sistema One-hot encoding, una manera de representar datos categóricos (como presencia o ausencia de un gen) por medio de valores binarios dentro de un vector de tamaño n (siendo n el número de nodos de la red).

Por ello, para el nodo s, el vector asociado sería:  $(k_1, \dots, k_s, \dots, k_n)/k_s = 1 \quad y \quad k_i = 0 \quad \forall i \neq s$

### Empleo con NetAnalyzer

Con NetAnalyzer, podemos generar el embedding por medio de kernels (la manera clásica) o por medio de node2vec (una estrategia de deep learning que simula un espacio vectorial optimizado para representar el grafo).

operación -k que operación de embedding utilizar. **Para embedding por kernels:** “el” (Exponential Laplacian difusión), “ct” (commute time kernel), “rf” (random forest Kernel), “vn” (von Neumann difusión kernel). Este último puede ir acompañado del parámetro  $\alpha$ . Así que realmente sería vnx (donde  $x=1, 0.5, 0.1$ ). También están ‘rl’ (Regularized Laplacian kernel matrix), con el número asociado rlX (1, 0.5, 0.1), “me” (Markov exponential difusión kernel), “ka” (Kernelized adjancency matrix). ’md’ (Markov diffusion kernel matrix) (mdX, parámetro t, número de iteraciones llevadas a cabo). A esto se le añade como opción para sistema por *deep learning* "deepwalk" y "node2vec".

operación -u decimos qué capa utilizar para el embedding.

operación -z para una normalización por coseno. Resulta útil para integrar distintos kernels.

—embedding\_add\_options Opciones adicionales para los sistemas de embedding, con el siguiente formato: opt1:value1,opt2:value2,...  
operación -K donde se indica el output del embedding en formato matricial.

operación —coords2sim\_type para seleccionar el modo de pasar de coordenadas a valor de similitud nodo a nodo, sólo válido para node2vec y deepwalk. Actualmente, están las siguientes opciones: dotProduct, normalizedScaling.

## 8.2 ranker

La herramienta NetAnalyzer tiene incorporado asimismo un script llamado ranker que sirve para comparar y priorizar listas de genes entre sí. Por ejemplo, se puede utilizar para comparar dos listas de genes que se hayan agrupado en clústeres y ver cómo de cercanos están esos genes entre sí. Los archivos de entrada deben tener dos columnas, siendo la primera la de identificador de elemento (clúster, por ejemplo) y en la segunda columna los genes asociados separados por comas. Se recomienda emplear identificadores de gen de la HGNC.

Los argumentos de entrada de ranker son los siguientes:

---

ser utilizada como representación de los nodos. En ocasiones será un espacio que trate de preservar el *commute time* entre dos nodos de una red, otras veces representará un proceso de difusión de calor entre los nodos, etc.

- k --input\_kernels -> Archivo con la matriz de embedding a emplear.
- n --node\_file -> Lista de nodos de cada kernel en formato lst. Generalmente suele ser el archivo kernel\_rowIds que genera la ejecución de NetAnalyzer.
- seed\_nodes -> Archivo de dos columnas que incluye los genes semilla empleados para hacer la comparación separados por coma. Puede ser de tres columnas, indicando en la tercera el peso específico para cada uno de los nodos que conforman el seed.
- seed\_presence -> Indicando con "remove" para eliminar las seed correspondientes del propio ranking a la hora de realizar los cálculos o bien con "annotate" para incluirlos con una anotación de si ese gen pertenece o no originalmente al seed.
- minimum\_size -> Cuando se quiere hacer un filtrado sobre aquellos seed con un tamaño menor en el embedding de n.
- whitelist -> Incorporación de un whitelist para filtrar las filas y columnas de una matriz determinada.
- presentation\_seed\_metric -> Para explicar el método con el que hacer el promediado dentro de las filas que representan los nodos de un seed. Actualmente está por defecto a "mean" y puede también utilizarse el "max".
- seed\_sep -> Separador de los nodos semilla si vienen en el archivo de entrada separados por otro string que no sea una coma (,).
- f --filter -> Ruta al archivo con la lista de genes (segunda columna separados por coma) sobre los que hacer la comparación.
- l --cross\_validation -> Habilita una prueba de validación utilizando cross validation.
  - K --k\_fold -> Si se quiere aplicar un k-fold cross validation. Por defecto se haría un leave one out.
  - t --top\_n -> Número máximo de genes a guardar en el archivo de salida.
  - output\_top -> Output name para el archivo de los tops en los rankings.
- o --output\_name -> Path al archivo de salida con los resultados del análisis. Tendrá cuatro o cinco columnas: 1) identificador del gen, 2) valor medio del ranking teniendo en cuenta los valores del kernel, 3) valor del ranking ordenado de 0 a 1 (cuanto más próximo a 0 más cercano a la semilla), 4) valor del ranking real, ordenado por cada clúster de mayor a menor valor, 5) identificador de clúster.
- adj\_matrix -> Para seeds mayores o iguales que 1. Hace un reajuste de la red en base al parámetro de similitud de nodo. Se incluye la matriz de adjacencia para hacer ese ajuste.
- score2pvalue -> Valores a escoger: "logistic"
- representation\_seed\_metric -> Valores a escoger: "fisher"
- 

### Ejemplo de uso de ranker

Imaginemos que tenemos una matriz embebida, creada por ejemplo a partir de STRING, así como la lista de nodos que forman parte de ella, y queremos consultar la distancia de un gen semilla (causal de la enfermedad) en una lista de genes cualesquiera (pongamos en este ejemplo todos los genes asociados a las distintas enfermedades que hay en OMIM). Necesitamos un mínimo de cuatro archivos (red embebida, lista de nodos, gen causal y lista de genes a comparar) para que ranker pueda trabajar:

```
ranker -k kernel.npy -n kernel_rowIds --seed_nodes causalGene -f
omim_genes
```

Con más detalles, el archivo kernel.npy es la matriz embebida resultante de ejecutar NetAnalyzer (y, preferentemente, con el grado normalizado) que únicamente contiene valores numéricos, el archivo kernel\_rowIds es el archivo que contiene los nombres de los nodos (genes en este caso),

el archivo causalGene contiene el nombre del gen causal y el archivo omim\_genes tiene la lista de genes OMIM (separados por comas).

Es necesario comentar que los archivos causalGene y omim\_genes son de dos columnas, donde la primera indica el clúster al que pertenece el nodo, y en la segunda los nodos constituyentes. Si todos los elementos pertenecen al mismo nodo (porque no ha hecho falta hacer un clústering previo), es suficiente con poner el valor 0 separando a los nodos por una tabulación.

El resultado es una tabla de 6 columnas que contiene, ordenado de mayor a menor score, la lista de nodos más próximos al seed. En este caso, los genes rankeados por ser más próximos en la red al gen causal de la enfermedad.

### 8.3 integrate\_kernels

Junto a las herramientas anteriores, nos encontramos con el script de integración de kernels. Cuya funcionalidad es la de combinar distintos embeddings (preferiblemente de mismo tipo) en formato kernel (formato matricial, npy) y devolver un nuevo embedding tipo kernel integrado.

Los argumentos de entrada de integrate\_kernels son los siguientes:

- -k –input\_kernels (**Obligatorio**) -> Paths a los kernels utilizados separados por ";". Ejemplo: mnt.../kernel1.npy;mnt.../kernel2.npy;...
- -n –input\_nodes (**Obligatorio**) -> Lista de nodos de cada kernel en formato lst. Ejemplo: mnt.../kernel1.lst;mnt.../kernel2.lst;...
- -I –kernels\_ids (**Opcional**) -> El nombre para cada kernel separado por ;. Ejemplo: Kernel1; kernel2...
- -i –integration\_type (**Obligatorio**) -> En este flag se especifica como se quieren integrar los kernels. Los valores actualmente disponibles son: "median", "max", "mean" e "integration\_mean\_by\_presence"<sup>6</sup>.
- –asymm (**Opcional, por defecto puesto a False**) -> Para indicar si las matrices que se dan de input pueden ser asimétricas.
- -T –threads (**Opcional, por defecto puesto a 1**) -> Para indicar el número de threads para parallelizar el proceso.
- -o –output\_file (**Opcional, por defecto "general\_matrix"**) -> El nombre de output de la matriz integrada.

### 8.4 text2binary\_matrix

Este script es empleado a la hora de cambiar el formato de la representación de una red de parejas <-> matriz txt <-> matriz npy, pequeñas manipulaciones en los valores y extracción de estadísticas.

#### Línea de comandos

opción -t –input\_type para especificar el formato del input, de la red que queremos analizar. Pudiendo tomar los valores: pair (por defecto), matrix y bin.

-byte\_format para especificar el formato float 64 o 32 de los valores. Por defecto está en float64.

opción -l para indicar, en formato string, los nombres de las capas y regexp que utilizaremos para identificar los nodos de esas capas. P.ej 'phenotypes,HP;diseases,[A-Za-z\_]\*'.

-O, –output\_type Indica el formato de salida ("pair", "bin" (por defecto) y "matrix").

---

<sup>6</sup>La diferencia entre mean e integration\_mean\_by\_presence es que en el primero la media es una división con respecto al número de kernels totales y en otra respecto al número de kernels donde aparece representada esa pareja de nodos

- d, --set\_diagonal Incorporar en la diagonal un valor de 1 en cada casilla.
- B, --binarize Binarizar la matriz en base a un cierto cutoff.
- c, --cutoff Realizar un cutoff sin realizar la binarización.
- s, --get\_stats Obtención de las estadísticas de la matriz de interés procesada.

## 8.5 randomize\_clustering

- opción -i --input\_file Archivo de nodos con comunidades asociadas. Pudiendo estar en formato agregado o desagregado.
- s --node\_sep Para indicar el separador de nodo en caso de que el archivo esté agregado, es necesario especificarlo.
  - a --aggregate\_sep Activa la agregación en el output con el separador especificado.
  - N --node\_column Indicando cuál es la columna que especifica el nodo, indexado en base 1. Por defecto está indicado para la segunda columna.
  - C --cluster\_column Indicando cuál es la columna que especifica la comunidad, indexado en base 1. Por defecto está indicado para la primera columna.
  - r --random\_type Para especificar el tipo de randomizado. Primero tenemos los que respetan el número de comunidades iniciales: "hard\_fixed", número fijo de comunidades al que pertenece un nodo, "soft\_fixed", aproximar el número de comunidades al que un nodo por un modelo de probabilidad, "not\_fixed", no se respecta nada más que el número de comunidades. Finalmente, también está la opción de pedir las comunidades al azar de manera *custom*, "custom:n:s:r|nr" con n el número de comunidades, "s" el tamaño de las comunidades y "r" o "nr" para especificar con o sin reemplazamiento, respectivamente.

## 8.6 randomize\_network

Para randomizar redes por conexiones o por nodos en una red. Los inputs para especificar la red a utilizar son los mismos que los empleados en el netanalyzer.

- opción -r --type\_random Especificar el tipo de randomizado colocando "nodes" o "links".

## 8.7 net\_explorer

El script net\_explorer sirve para explorar ciertos nodos dentro de una red de partida. Por ejemplo, esta herramienta es útil para explorar dónde quedan dentro de una red de interacciones los genes o proteínas de interés en un experimento.

```
net_explorer -i $matrix_paths -n $nodes_paths -s $seeds_file -c $cutoffs
-N
```

Los argumentos de entrada de net\_explorer son los siguientes:

- -i: path a las matrices de entrada (en formato .npy) donde buscar los nodos de interés. Se debe describir qué red es, separado por una coma a su path, y por un punto y coma las distintas redes a incluir: string, \$matrix\_paths=path\_to\_string.npy;hpo,path\_to\_hpo.npy...
- -n: path a las listas de nodos a considerar en cada una de las matrices. Se deben adjuntar de la misma forma que los paths a las matrices de entrada, describiendo en qué red buscar, separado por una coma a su path, y por punto y coma a las distintas listas a considerar. Las listas suelen darse con extensión .lst: nodes\_path=string,path\_to\_string.lst;hpo,path\_to\_hpo.lst...

- -s: path a los seeds de interés.
- -c: valores de corte para filtrar los nodos para cada una de las redes de entrada.
- -N: Si se activa, descarta autorrelaciones en la red.
- –neigh\_level: Argumento numérico, sirve para configurar el número de nodos vecinos a considerar.

Un ejemplo sencillo de uso, empleando las redes de STRING y considerando una lista de nodos (separados por tabulación) sería el siguiente:

```
1 source ~soft_bio_267/initializes/init_python
2
3 matrix_paths="string,similarity_matrix_bin.npy"
4 nodes_paths="string,symbol_nodes.lst"
5 seeds_file=$1
6 cutoffs="string,200"
7
8 net_explorer -i $matrix_paths -n $nodes_paths -s $seeds_file -c $cutoffs
-N
```

Un ejemplo de \$seeds\_file sería el siguiente (categoría separada de seeds por tabulación):

```
1 metabolic_disease ARSK,MECR,NAGA
2 cardiovascular_disorder MYH11,NOTCH1,TMEM43
```

El programa devuelve dos archivos de salida en HTML que incluyen un informe de error y un archivo output\_file.html con las proyecciones de los nodos en las redes.



## 9. cdlib\_clusterize.py

### 9.1 Descripción

CDLib es un paquete de Python que permite la extracción, comparación y evaluación de comunidades en redes complejas. En nuestro caso, esta herramienta se usa para aplicar diferentes algoritmos de clusterización sobre redes, así como evaluar los clústers generando una serie de métricas. Tenemos la posibilidad de generar métricas tanto generales como a nivel de cada cluster individual.

### 9.2 Funcionamiento

El script «cdlib\_clusterize.py» funciona apoyándose en el paquete de Python 'Networkx' para almacenar dentro de un objeto la información de la red que recibe. Posteriormente, los diferentes métodos de clusterización que contiene este paquete son llamados sobre el objeto previamente creado que contiene el grafo.

#### 9.2.1 Argumentos del script

El funcionamiento de este script puede controlarse mediante los siguientes flags:

- -i -> Archivo de entrada con los datos de la red a clusterizar, en formato tsv de parejas de nodos
- -m -> Nombre del algoritmo o método de clustering que se desea implementar
- -a -> Opciones adicionales para los métodos de clustering. Cada método tiene una serie de argumentos que pueden darse en este flag y que permiten modificar su comportamiento. El formato de entrada de este argumento es el siguiente: -a "opcion1" : valor1, "opción2" : valor2, etc.
- -o -> Nombre del archivo de output, que contendrá los clusters generados.
- -s -> Activa el cálculo de métricas sobre los clústers generados.
- -S -> Nombre del archivo de métricas. (default=clusters\_stats.txt)
- -g -> Activa el cálculo de métricas para cada uno de los clústers generados.
- -E -> Nombre del archivo de métricas para cada clúster.

- -e -> Archivo externo de clustering. Este flag se usará cuando queramos activar el modo de métricas del cdlib sobre un archivo de clústers previamente creado, por ejemplo:

```
1 cdlib_clusterize.py -i 'red.txt' -e 'clusters.txt' -m external -s
2
```

## 9.3 Métodos de clustering

A continuación, se listan los métodos de clusterización disponibles para darle al parámetro **-m** junto con las opciones adicionales que podremos pasar con el parámetro **-a**

### 9.3.1 Métodos para comunidades no solapantes

Se dice que una comunidad es **no solapante** cuando está formada por nodos que exclusivamente pertenecen a dicha comunidad, sin poder formar parte de ninguna otra:

- **cpm** - *initial\_membership*<sup>1</sup>, **weights**<sup>2</sup>, *node\_sizes*<sup>3</sup>, *resolution\_parameter*<sup>4</sup>
- **leiden** - *initial\_membership*, **weights**
- **louvain** - **weights**, *resolution*, *randomize*<sup>5</sup>
- **markov\_clustering** - consultar manual<sup>6</sup>
- **label\_propagation** - consultar manual
- **greedy\_modularity** - **weights**
- **rb\_potts** - *initial\_membership*, **weights**, *resolution\_parameter*
- **rber\_potts** - *initial\_membership*, **weights**, *node\_sizes*, *resolution\_parameter*
- **der** - *walk\_len*, *threshold*, *iter\_bound*
- **eigenvector** - consultar manual
- **edmot** - *component\_count*, *cutoff*
- **significance\_communities** - consultar manual
- **gdmp2** - *min\_threshold*
- **walktrap** - consultar manual
- **surprise\_communities** - consultar manual
- **spinglass** - consultar manual

<sup>1</sup>Lista de int. Membership inicial para la partición (número mínimo de miembros para formar un cluster). Si "None", el valor predeterminado es una partición tipo singleton. Default=None

<sup>2</sup>Lista de double, o atributo Weights de los edges. Puede ser tanto un iterable como un atributo. Para algunos métodos, en vez de una lista de double o atributo, se pide un string a usar para reconocer los datos relativos a los pesos en el set de datos dado. (Debido a esta ambigüedad, cuando vaya a usarse un método que acepte pesos, mejor consultar el manual: [https://cdlib.readthedocs.io/en/latest/reference/cd\\_algorithms/node\\_clustering.html](https://cdlib.readthedocs.io/en/latest/reference/cd_algorithms/node_clustering.html)). Default=None en el primer caso y Default='weight' en el segundo.

<sup>3</sup>lista de int, o atributo Sizes de los nodos. Sirve para indicar una ponderación por tamaño de los nodos en caso de que se desee. Default=None

<sup>4</sup>Controla el tamaño de los clusters. Entre 0 y 1, a mayores valores son menores los tamaños de cluster y mayor el número de clusters totales

<sup>5</sup>Int, instancia de RandomState o None, opcional (Default=None). Si es int, random\_state es la semilla utilizada por el generador de números aleatorios; Si es una instancia de RandomState, random\_state es el generador de números aleatorios; Si es None, el generador de números aleatorios es la instancia de RandomState utilizada por np:random.

<sup>6</sup>[https://cdlib.readthedocs.io/en/latest/reference/cd\\_algorithms/node\\_clustering.html](https://cdlib.readthedocs.io/en/latest/reference/cd_algorithms/node_clustering.html)

### 9.3.2 Métodos para comunidades solapantes

Llamamos **comunidades solapantes** a aquellos clusters que contienen al menos un nodo que forma parte de, como mínimo, otra comunidad más:

- **wcommunity** - *min\_bel\_degree, threshold\_bel\_degree, weightName*
- **aslpaw** - No recibe argumentos extra, y funciona tanto para redes con pesos como sin pesos.
- **slpa** - *t, r*
- **nnsed** - *dimensions, iterations, seed*
- **nmmf** - *dimensions, clusters, lambd, alpha, beta, iterations, lower\_control, eta*. Funciona tanto para comunidades solapantes como no solapantes.
- **egonet\_splitter** - *resolution*
- **ego\_networks** - *level*
- **danmf** - *layers, pre\_iterations, iterations, seed, lamb*
- **big\_clam**
- **lais2**





## 10. PETS: Patient Exploration Tools Suite

El PETS engloba una serie de herramientas para analizar y gestionar perfiles de fenotipos en términos de la Human Phenotype Ontology (HPO). Los perfiles fenotípicos pueden ser tanto perfiles fenotípicos de los pacientes de una o varias cohortes, como cualquier otro tipo de agrupación de fenotipos.

Esta herramienta se compone de diversos scripts:

### 10.1 Cohort Analyzer

#### 10.1.1 Descripción

Cohort Analyzer es una herramienta que mide la calidad del fenotipado de cohortes de pacientes. Para ello se calculan una serie de estadísticas que dan una visión general del estado de la cohorte, en lo referido a profundidad y amplitud del fenotipado.

#### 10.1.2 Script

##### Uso básico

Esta herramienta se invoca en la terminal mediante coPatReporter.rb. Para el *input* de este *script* se debe indicar con el *flag* -i un archivo tabulado de tipo **profiles** con dos columnas obligatorias: una columna con un identificador de paciente (que se debe indicar con el *flag* -d) y otra columna donde están agregados todos los términos HPO (que se debe indicar con el *flag* -p) divididos por un carácter separador ('l' por defecto pero se puede cambiar con el *flag* -S).

El archivo de entrada puede ir con *header* o sin el. Por defecto se asume que el archivo lleva una linea de *header*, lo que implica que en los -p y -d se debe indicar el **nombre** de la columna. Si, por el contrario, se activa el *flag* -H, se dará por hecho que el archivo no lleva *header* por tanto se deberá indicar los **índices en base 0** de las columnas. Esto último afecta también a los *flags* -c, -s y -e que se comentarán mas tarde en el manuscrito.

La lista de argumentos de entrada se enumeran a continuación:

- **-a, --coverage\_analysis** -> Desactivar el módulo de análisis de cobertura del genoma. Activo por defecto (true).
- **-b, --bin\_size** -> En el plot de cobertura, máximo de bins a mostrar. Por defecto 50,000.
- **-C, --clusters2show\_detailed\_phen\_data** -> Número máximo de pacientes cuya información detallada mostrar en el informe final. Valor por defecto: 3.
- **-c, --chromosome\_col** -> Nombre de la columna que indica el valor CROMOSOMA. En caso de que no haya cabecera, poner el valor numérico de la posición que corresponda con ese valor en el archivo de entrada.
- **-d, --id\_col** -> Nombre de la columna que indica el valor IDENTIFICADOR DE PACIENTE. En caso de que no haya cabecera, poner el valor numérico de la posición que corresponda con ese valor en el archivo de entrada.
- **-E, --excluded\_hpo** -> Ruta al archivo con los identificadores HPO a omitir en el análisis.
- **-e, --end\_col** -> Nombre de la columna que indica el valor FIN DE COORDENADA GENÓMICA. En caso de que no haya cabecera, poner el valor numérico de la posición que corresponda con ese valor en el archivo de entrada.
- **-M, --minClusterProportion** -> Porcentaje mínimo de pacientes a mostrar por clúster. Valor por defecto: 0.01.
- **-f, --patients\_filter** -> Número mínimo de pacientes que comparten la misma *short overlapping region (SOR)*. Valor por defecto: 2.
- **-g, --clusters2graph** -> Número máximo de clústeres de pacientes a incluir en los plots. Valor por defecto: 30.
- **-G, --genome\_assembly** -> Versión del ensamblaje humano para el análisis de genes. Valores disponibles: hg18, hg19 y hg38. Valor por defecto: hg38.
- **-h, --help** -> Muestra la ayuda.
- **-H, --header** -> Configurar si el archivo tiene cabecera. Activo por defecto (true).
- **-i, --input\_file** -> Ruta al archivo de entrada con datos de pacientes.
- **-m, --clustering\_methods** -> Lista con métodos de clustering. Si son varios, separar por comas sin espacio. Valores disponibles: resnik, jiang\_conrath y lin. Valores por defecto: resnik, jiang\_conrath y lin.
- **-n, --names** -> Definir si los HPO vienen como NOMBRES en lugar de CÓDIGOS. Inactivo por defecto (false).
- **-o, --output\_file** -> Ruta al archivo de salida.
- **-P, --hpo\_file** -> Ruta al archivo con la información de la HPO.
- **-p, --ont\_col** -> Nombre de la columna que indica el valor FENOTIPO. En caso de que no haya cabecera, poner el valor numérico de la posición que corresponda con ese valor en el archivo de entrada.
- **-S, --hpo\_separator** -> Carácter separador de los HPO en el archivo de entrada. Valor por defecto: |.
- **-s, --start\_col** -> Nombre de la columna que indica el valor INICIO DE COORDENADA GENÓMICA. En caso de que no haya cabecera, poner el valor numérico de la posición que corresponda con ese valor en el archivo de entrada.
- **-r, --root\_node** -> Código HPO a considerar como raíz de la ontología. Términos parentales a él no se considerarán en el análisis. Valor por defecto: HP:0000118.
- **-t, --ic\_stats** -> Método para calcular el valor de IC (information content). Valores disponibles: freq (computa el valor de IC considerando la frecuencia del término en la cohorte), freq\_internal (usa un valor de IC precalculado) y onto (computa el valor de IC considerando la frecuencia

- del término en la ontología)). Valor por defecto: freq.
- **-T, -threads** -> Número de threads a configurar para la ejecución. Valor por defecto: 1.
  - **-reference\_profiles** -> Ruta al archivo con perfiles de referencia, un archivo tabulado que en la primera columna tiene el identificador de perfil y en la segunda los términos de la ontología separados.
  - **-sim\_thr** -> Valor a especificar para filtrar pacientes por similitud fenotípica de perfiles. Si no se especifica ningún valor no se aplica este filtro.

### Ejemplos de uso.

Ejecución mínima del comando comando coPatReporter.rb. El archivo de entrada tiene cabecera:

```
coPatReporter.rb -i cohort.txt -o results/results\_cohort.txt -p phenotypes  
-d patient_id
```

Otra ejecución del comando comando coPatReporter.rb. El archivo de entrada no tiene cabecera (-H). Se especifica el cálculo de IC como onto, el separador de fenotipos es “,”, el máximo de clústeres se establece en 25 y el método de clústering a lin:

```
coPatReporter.rb -i cohort.txt -o results/results\_cohort.txt -t onto -p 1 -  
S ',' -T 1 -d 0 -H -C 25 -m lin
```





## 11. ExpHunterSuite

La suite de herramientas ExpHunter Suite fue diseñada con el propósito de unificar distintos métodos de análisis de datos de expresión. Incluye varios flujos de trabajo para llevar a cabo análisis de expresión diferencial a partir de datos de RNA-seq y miRNA-seq, análisis de co-expresión y de anotación funcional empleando distintas fuentes de información. Entre sus características generales, destacan las que se enumeran a continuación:

- Combina hasta cuatro paquetes de análisis de expresión para llevar a cabo análisis de expresión diferencial: DESeq2, limma, NOISeq y edgeR.
- Integra los p-valores de cada método de análisis de expresión mediante el método de Fisher para obtener un p-valor representativo.
- Incluye un análisis funcional empleando diferentes fuentes de información (GO, KEGG, Reactome) para transformar los genes expresados diferencialmente (DEG) en conocimiento biológico.
- Realiza análisis de co-expresión para buscar grupos de genes que co-expresen (CEG) y correlacionen con datos fenotípicos.

En la sección 2 se detalla la instalación de la herramienta en distintos entornos y ejemplos de uso.

### Vignettes

Pincha en este enlace para consultar las vignettes del paquete. Entenderás cómo usarlo y qué funcionalidades tiene.

### Enriquecimiento funcional de clústeres de co-expresión

Al igual que con los genes expresados diferencialmente, los clústeres de co-expresión también se analizan a nivel funcional. El programa `clusters_to_enrichment.R` es el que se encarga de hacer esta función. Sus argumentos de entrada son los siguientes:

- **-i, --input\_file** -> Ruta al archivo de entrada. Debe ser de dos columnas sin cabecera, donde la primera columna corresponde al identificador del cluster, y la segunda columna los elementos

(genes co-expresados en este caso) sobre los que hacer el enriquecimiento funcional, separados por comas y codificados en términos entrez o ENSEMBL (ENSG).

- **-o, --output\_file** -> Ruta al archivo de salida.
- **-f, --funsys** -> Módulos funcionales a activar, separados por comas. MF => GO Molecular Function, BP => GO Biological Process, CC => GO Celular Component, KEGG => KEGG, Reactome => Reactome, DO => Disease Ontology, DGN => DisGeNET.
- **-t, --task\_size** => nmerodeclsteresportarea(por defecto 1)

Partiendo de un archivo de dos columnas, donde la primera tiene los identificadores de cluster y la segunda los genes en códigos ENSEMBL separados por coma y sin cabecera, un ejemplo de uso básico del programa sería el siguiente:

```
1 source ~soft_bio_267/initializes/init_degenes_hunter
2 clusters_to_enrichment.R -i cluster_genes.txt -w 16 -o
functional_results -f BP,CC,MF,KEGG,Reactome -k ENSEMBL
```

#### **add\_annotation.R: adición de columna gene symbol**

Otra funcionalidad que encontramos en ExpHunterSuite consiste en añadir una columna en las tablas creadas con el gene symbol de cada código Ensembl proporcionado por Hunter. Para ello, utilizamos la herramienta add\_annotation.R, cuyos argumentos de entrada son los siguientes:

- **-i, --input\_file** -> ruta al archivo de entrada.
- **-o, --output\_file** -> ruta al archivo de salida con la columna de anotación añadida.
- **-c, --column** -> nombre o índice de la columna que presenta los identificadores de ensembl. Por defecto: 1.
- **-I, --input\_keytype** -> keytype de entrada, por defecto NULL. En nuestro caso: "ENSEMBL".
- **-K, --output\_keytype** -> keytype de salida: "SYMBOL".
- **-O, --organism** -> organismo modelo. Por defecto: Human.
- **-h, --help** -> ayuda.

### 11.1 Análisis de detección y expresión de miRNAs (smallRNAseq)

Cuando se lleva a cabo el análisis de datos de miRNA-seq hay una serie de diferencias con respecto al análisis de datos de RNA-seq. Existen dos etapas claramente diferenciadas:

- Detección de miRNA (miRNA\_det)
- Análisis de expresión diferencial de miRNA detectados (miRNA\_dea)

Se explicarán las dos etapas y cómo se debe configurar el archivo config\_daemon para poder llevar a cabo tanto la detección como el DEA de forma satisfactoria.

#### 11.1.1 Detección de miRNA

En la primera etapa se van a detectar los miRNA presentes en los archivos de lecturas para que puedan servir como referencia en el paso posterior de análisis de expresión diferencial. Para ello, en la plantilla config\_daemon las siguientes variables tienen que configurarse como se especifican a continuación:

```
1 export experiment_type="miRNAseq_detection"
2 export mapping_ref=$CODE_PATH"/references"
3 export only_read_ref=~pedro/references/hsGRc38
4 export TRIM_TEMPLATE=$CODE_PATH"/templates/trimming/miRNA.txt"
```

También asegúrate de que la carpeta **export\_project\_folder** tenga un path donde el nombre de carpeta termine en `_det` para saber que ahí tienes los resultados de la DETECCIÓN de miRNA.

Esto configurará el sistema para que cuando lancemos `./daemon 1a` nos genere la carpeta «references» en la carpeta del flujo. De ahí tomará el programa Bowtie la referencia para hacer el primer alineamiento de las lecturas. Para llevar a cabo dicho alineamiento, ejecutamos `./daemon 2a`. Esto no solo ejecutará la limpieza de las lecturas con seqtrimbb y el alineamiento con Bowtie, sino que además ejecutará el programa miRDeep. Este programa analiza el alineamiento y usa una base de datos de precursores de miRNA para determinar si una lectura pertenece a un miRNA (busca que alinee el precursor) o no analizando sus propiedades (como por ejemplo, que sea capaz de formar una horquilla).

Esta ejecución dará lugar a un archivo fasta que tendrá todas las secuencias de miRNA que se emplearán como referencia para llevar a cabo el alineamiento en la siguiente fase.

Una vez que comprobemos que la limpieza, los alineamientos y la detección se han llevado a cabo correctamente (ejecutando `./daemon 2b`), preparamos un target (que será «fake» a fin de que no nos rompa la ejecución) para que se ejecute el report de mapeo de detección de miRNA. Esto se lanzará con `./daemon 3`. Es conveniente que nos aseguremos de que en nuestra carpeta de resultados tengamos nuestro archivo `mapping_report.html`, así como que los archivos de referencia estén disponibles en la carpeta «references» (`all_miRNA.fasta`, `final_miRNA.coord`, `known_miRNA.coord`, `miRNA_nr.fasta`, `miRNA_nr.fasta.clstr`, `novel_miRNA.coord` y la carpeta `bowtie_index_tr/`).

Con todo ello podremos seguir adelante en el análisis de miRNA.

### 11.1.2 Análisis de expresión diferencial de miRNA

Los archivos fasta de miRNA generados se emplearán como referencia para hacer el alineamiento de las lecturas y poder llevar a cabo el análisis de expresión diferencial. De esta manera podremos ver cuánto se expresan esos miRNA detectados. Ahora copiamos nuestro archivo `config_daemon` para tener dos copias: una de detección (`config_daemon_det`) y otra de expresión diferencial (`config_daemon_dea`). Antes de lanzar `./daemon`, nos aseguraremos de que `config_daemon_dea` vuelva a llamarse `config_daemon`. Mucho cuidado con esto ya que es fácil confundir ambos tipos de daemon.

En el archivo `config_daemon_dea` hay que configurar la variable:

```
export experiment_type="RNAseq_transcriptome"
```

Y también tenemos que asegurarnos de cambiar el nombre a la carpeta de salida de los resultados (en la variable `project_folder`), cambiando el «`_det`» por «`_dea`».

Con ello, volvemos a ejecutar `./daemon 1b` para que se emplee de referencia el fasta de los miRNA generado en el proceso de detección. Antes de llevar a cabo el proceso de alineamiento, tenemos que especificar en la variable `link_path` del `config_daemon`, la ruta a la carpeta de lecturas limpias que obtuvimos en la fase de detección. De esta manera, al ejecutar `./daemon.sh 2a`, evitamos realizar la limpieza de nuevo al crear enlaces simbólicos a estas lecturas. Con ello se deberían alinear las lecturas contra el fasta de miRNA de referencia. Para confirmar que nuestros resultados son correctos, cuando nos aseguremos de que en el sistema de colas no hay trabajos corriendo, ejecutamos `./daemon 2b`.

Una vez hecho este procedimiento pasaremos al análisis de expresión diferencial de los miRNA detectados. Para ello ejecutamos `./daemon 3a` asegurándonos de que hemos configurado los

targets de forma acorde a las comparaciones que queremos llevar a cabo.

#### Análisis de correlación miRNA-diana y análisis funcional

*Sección por completar* Con los resultados, se puede hacer un análisis funcional de las dianas de los miRNA que se han detectado. Del mismo modo, si se tienen datos de RNA-seq del mismo estudio, se pueden llevar a cabo análisis de correlación de miRNA.

En el caso del análisis funcional de las dianas de los miRNA, por cada ejecución es necesario configurar un archivo config\_daemon con los parámetros que se describen a continuación: De los resultados devueltos los archivos de mayor interés para su exploración y entrega son los siguientes:

- miRNA\_target.html: es equivalente al informe de análisis de expresión diferencial para RNA-seq.
- all\_miRNA\_summary.txt: tabla que recoge la información del apartado «Strategy miRNAs summary» con la tabla que incluye información de los targets de miRNAs diferenciales.
- target\_results\_table.txt: detalles de las parejas miRNA-target.
- results/clusters\_func\_report.html: agrupa todos los miRNA para devolver un análisis de enriquecimiento funcional global.
- results\_func\_report.html: por cada miRNA (\*) se incluye un informe detallado del análisis de enriquecimiento funcional llevado a cabo sobre todos sus targets.

#### 11.1.3 Subir cambios al repositorio ExpHunterSuite

Al ser un paquete de R, se rige por lo descrito en SECCIÓN NO CREADA AÚN, siguiendo los pasos de devtools::document(), R CMD check y R CMD build. Allí también se detalla la guía estilo que seguimos en el desarrollo del paquete.



## 12. DEG Workflow

El flujo de trabajo para el análisis de muestras de RNA-seq y miRNA-seq, DEG Workflow, está disponible en [https://github.com/seoanezonjic/DEG\\_workflow](https://github.com/seoanezonjic/DEG_workflow) y su publicación ha sido aceptada en *Briefings in Bioinformatics* bajo el título «Exploring miRNA-target gene pair detection indisease with coRmiT». Incluye programas desarrollados en Bash, Python y llamadas a la herramienta ExpHunter Suite para llevar a cabo el tratamiento de archivos de secuenciación (FASTQ).

Incluye cinco etapas diferenciadas:

1. Descarga e indexado del genoma de referencia.
2. Limpieza (trimming) y alineamiento (mapping) de las lecturas contra la referencia.
3. Generación de informe de calidad de limpieza y alineamiento, análisis de expresión diferencial y co-expresión.
4. Análisis de enriquecimiento funcional.
5. Preparación de paquetes.

En la sección 12.1 se describe paso por paso y al completo desde dónde se obtiene la información de secuenciación y cómo trabaja el flujo de trabajo en las distintas fases para el análisis de muestras de RNA-seq. En el caso de miRNA-seq, en el capítulo ?? se explica el mismo análisis detallado para la detección y análisis de expresión diferencial de miRNA y el uso de coRmiT para llevar a cabo el análisis de diana-target.

### 12.1 Análisis de muestras de RNA-seq con DEG Workflow

Este flujo de trabajo emplea los archivos FASTQ producto del secuenciador (tecnología Illumina) sin preprocessar y devuelve informes de composición de las muestras (cómo ha salido el experimento y la secuenciación), así como informes de resultados de la ejecución de ExpHunter Suite.

### ¿Qué lleva a cabo la herramienta ExpHunter Suite?

- Localiza genes que cambian su expresión entre dos grupos de muestras de una forma significativa.
- Establece módulos de co-expresión, definiendo conjuntos de genes que se expresan de la misma forma a lo largo de las muestras.
- Analiza la función de los genes con expresión diferencial y de los módulos de co-expresión.

### RNA-seq con Illumina:

La secuenciación de Illumina es una técnica que se encarga de leer pequeños fragmentos de DNA (desde 50 nt hasta 200 nt). En el caso particular del RNA-seq se pretenden secuenciar los RNA de una muestra. Antes de ser secuenciado, el RNA de la muestra se transcribe de forma reversa a cDNA que luego se rompe en fragmentos pequeños que pueda leer la máquina. Las lecturas de la máquina se guardan en archivos fastq que además de la secuencia del fragmento de DNA, guardan información de la fiabilidad de cada nucleótido según la máquina. Cada entrada en un fastq tiene cuatro líneas (identificador, secuencia, segundo identificador, calidad), y aquí se representa un ejemplo:

```
1 @SEQ_ID
2 GATTTGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTGTTCAACTCAC
3 +
4 ! ˇ * (((****)) %%%++) (%%%%) . 1*** - +* ˇ )) **55CCF>>>>>CCCC
```

DEG Workflow se compone de varias fases que permitirán realizar un análisis de RNA-seq al completo, con el único input de las muestras de secuenciación e información adicional del experimento que se lanzarán en orden a partir del ejecutable `daemon.sh`. En la fase 0 (sub-apartado 12.1.1) se explica la configuración manual del flujo antes de ejecutar las fases posteriores. En la fase 1 (sub-apartado 12.1.2) se explica la descarga e indexado del transcriptoma o genoma que se vaya a usar como referencia para el mapeo. La fase 2 (sub-apartado 12.1.3) describe el protocolo de pre-procesamiento (limpiar las muestras de lecturas de poca calidad, contaminantes y artefactos de secuenciación) y mapeo (clasificar las lecturas en función del gen al que pertenecen). Después se transforman los datos de mapeo a un formato llamado «matriz de conteos», que es un archivo de texto plano tabulado que indica cuántas lecturas de cada gen hay en cada muestra de secuenciación. Esta matriz de conteos, será el input principal de ExpHunter Suite. La fase 3 (sub-apartado 12.1.4) consiste en la ejecución de la generación del informe de calidad basado en los resultados del pre-procesamiento y alineamiento de las lecturas contra el genoma de referencia y del posterior análisis de expresión diferencial con ExpHunter Suite. La fase 4 (sub-apartado 12.1.5) describe cómo analizar la composición funcional de los resultados obtenidos en la fase 3 con ExpHunter Suite. Por último, la fase 5 (sub-apartado 12.1.6) describe cómo realizar la preparación del paquete de entrega de resultados.

### 12.1.1 Fase 0: Configuración y preparación de DEG Workflow

Para que quede todo el proceso más claro, nos basaremos en el siguiente ejemplo: Imagina un experimento en el que se quiere ver el efecto de una enfermedad hereditaria llamada Examplemia, que es producida por mutaciones en el genA o en el genB. Esta enfermedad se manifiesta cuando los dos alelos del genA o del genB están mutados, es decir, el paciente presenta un fenotipo homocigótico para uno de los dos genes. No se han encontrado pacientes con genotipos homocigóticos patogénicos en ambos genes, así que es posible que esta combinación no sea compatible con la vida. Además se ha observado que los pacientes con mutaciones en el genA pueden vivir más años que los que tienen mutaciones en el genB. Por tanto se diseña un experimento para comparar la RNA-seq de 9 muestras: 3 de ellas pertenecientes a pacientes sin la enfermedad (controles), otras 3 pertenecen a pacientes con genotipo patogénico homocigótico en el genA (tratamiento 1) y las tres restantes pertenecen a pacientes con genotipo patogénico homocigótico en el genB (tratamiento 2). Lo interesante de este experimento es ver 3 cosas: i) Diferencias generales entre los pacientes sanos y los pacientes enfermos (controles vs todos los enfermos), ii) estudiar por separado cada una de las mutaciones (controles vs mutantes de genA y controles vs mutantes de genB), y iii) ver las diferencias entre ambos tipos de enfermos (mutantes del genA vs mutantes del genB).

#### Configuración de carpetas:

En primer lugar se debe crear el sistema de carpetas que guardará los datos y las ejecuciones correspondientes al proyecto. Se aconseja la creación de un directorio «projects» en el \$HOME «mkdir ~/projects». Dentro de esta carpeta se debe crear un directorio específico del proyecto con un nombre representativo, en el ejemplo propuesto, la carpeta podría llamarse «~/projects/examplemia». Dentro de esta carpeta se crearán dos subcarpetas: «~/projects/examplemia/data» (donde se guardarán las muestras de secuenciación) y «~/projects/examplemia/analysis» (donde se guardarán los archivos del flujo, como se explicará a continuación).

Dentro de la carpeta «analysis» se debe clonar el código del flujo a través del repositorio [https://github.com/seoanezonjic/DEG\\_workflow.git](https://github.com/seoanezonjic/DEG_workflow.git), ejecutando el comando:  
«git clone https://github.com/seoanezonjic/DEG\_workflow.git».

Se creará una carpeta llamada DEG\_workflow.

#### Descripción general del flujo de trabajo.

La carpeta DEG\_workflow incluye las siguientes carpetas y archivos:

- Carpeta aux\_parsers -> Incluye programas auxiliares que ayudan a trabajar con los archivos resultantes de ejecuciones del flujo de trabajo. Incluye programas en Ruby, R, Bash y Python.
- Carpeta aux\_sh -> Incluye programas que hacen las llamadas a las templates de autoflow y programas que se encargan de llevar a cabo el análisis de las muestras. Todos son ejecutables de Bash que se llaman a partir del archivo máster de ejecución del flujo (daemon.sh).
- Carpeta templates -> incluye plantillas de AutoFlow para llevar a cabo varios procesos, como la limpieza y mapeo de las lecturas (mapping\_template.txt), la detección de miRNAs (miRNA\_detection.txt), y otras plantillas de markdown y erb con la información para generar los informes de resultados.
- Archivo README.md -> Readme de DEG\_workflow en GitHub.

- Archivo config\_daemon\_default -> Parámetros configurables del flujo de trabajo. Es el archivo que se debe configurar para llevar a cabo el análisis, con las variables y parámetros a modificar dependiendo del tipo de muestras con las que se va a trabajar. Es importante hacer una copia del archivo (no renombrarlo) a config\_daemon para no perder la información de la configuración inicial en caso de que se corrompa el archivo original (copia de seguridad). Importante: el flujo de trabajo **no funciona** si no existe el archivo config\_daemon. Más detalles sobre el archivo de configuración están disponibles en 12.1.1.
- Archivo daemon.sh -> Archivo máster de ejecución del flujo DEG\_workflow. Hay más detalles en el apartado 12.1.1. Su ejecución llama a los programas almacenados en la carpeta aux\_sh. Acepta un argumento de entrada que está configurado para activar las siguientes etapas del flujo (ejecutando programas dentro de la carpeta aux\_sh.):
  - \* ./daemon 1a -> Descarga de la referencia en la carpeta references (se crea en el directorio raíz de DEG\_workflow). Ejecuta el programa download\_files\_for\_index.sh. Si ya hay una referencia indexada y disponible, en el config\_daemon hay que incluir el path a la referencia dentro de la variable only\_read\_ref.
  - \* ./daemon 1b -> Indexado de la referencia. Ejecuta el programa create\_index.sh. Si la referencia ya está indexada y disponible en la variable only\_read\_ref, NO hay que ejecutar de nuevo este indexado.
  - \* ./daemon 2a -> Llamada a la plantilla de Autoflow que se encarga de hacer la limpieza y mapeo de las lecturas contra la referencia. Ejecuta el programa trim\_and\_map.sh.
  - \* ./daemon 2b -> Ejecuta el script check\_wf.sh que llama a la funcionalidad de AutoFlow denominada flow logger 4.6 para revisar el estado de las ejecuciones. Devuelve una tabla para indicar si la limpieza y mapeo se han llevado a cabo de manera satisfactoria. Asimismo, se puede emplear para lanzar trabajos que hayan roto. Para ello, se ha de ejecutar añadiendo el argumento '-lp'.
  - \* ./daemon 3 -> Lanza el programa compare\_all\_samples.sh que, por un lado, llama al programa para crear el informe de limpieza y mapeo contra la referencia de las muestras, y tras ello, ejecuta el análisis de expresión diferencial con ExpHunter Suite.
  - \* ./daemon 4a -> Ejecuta el programa launch\_fun\_hun.sh, que activa la modalidad funcional de ExpHunter Suite (en muestras de RNA-seq).
  - \* ./daemon 4b -> Ejecuta el programa launch\_fun\_hun.sh, que activa la modalidad funcional de ExpHunter Suite (en muestras de miRNA-seq).
  - \* ./daemon 5 -> Permite la creación de paquetes de entrega de resultados mediante la herramienta create\_hunter\_pack.sh.
- Archivo manage\_input\_files\_example.sh -> Ejemplo de preparación de muestras. Renombra los archivos para que sus nombres sean fácilmente interpretables. Se puede renombrar a manage\_input\_files.sh antes de ejecutarlo. En él, hay que cambiar la variable source\_folder al path donde estén los archivos originales, y por cada llama al enlace simbólico asegurarse de que los archivos renombrados corresponden con las muestras originales. Recomendamos que, para ello, se cree un archivo Excel (tabla de equivalencias) que por columnas tenga el identificador de la muestra original y el nombre final de archivo/muestra que usará el flujo de trabajo para el tratamiento de las muestras.

**Configuración de muestras:**

Para empezar a trabajar, necesitamos los datos de secuencias (archivos FASTQ o fastq.gz). Los archivos FASTQ se descargarán o copiarán en la carpeta «original\_data». Las muestras originales suelen tener un nombre por defecto del secuenciador o del servicio de secuenciación y generalmente no suele ser interpretable. Para ello, hay que renombrar los archivos para que se puedan interpretar fácilmente los resultados.

Se debe escoger un nombre corto identificativo del experimento para cada archivo. En el ejemplo, partiendo de muestras llamadas xxx1, xxx2, xxx3 (pertenecientes a pacientes no enfermos); xxx4, xxx5 y xxx6 (pertenecientes a pacientes con el genA mutado); xxx7, xxx8 y xxx9 (pertenecientes a pacientes con el genB mutado), se van a renombrar indicando la condición del paciente: controles = CTL, mutación en el genA = genA y mutación en el genB = genB. y el número de la réplica (1, 2, 3) con el formato [CONDICIÓN]\_[RÉPLICA]. Sólo se pueden usar los caracteres «\_» y «.» como separadores de palabras en los nombres.

Además, para añadir un poco más de complejidad, en el experimento se ha usado la técnica de secuenciación de lecturas pareadas (más información sobre la técnica en Illumina.com), que consiste en secuenciar fragmentos más largos, leyéndolos desde ambos extremos, de tal manera que se generan dos archivos por cada muestra: xxxN\_1.fastq.gz refiriéndose a la lectura de cada fragmento por un extremo y xxxN\_2.fastq.gz para las lecturas secuenciadas desde el otro extremo.

**WARNING: Recuerda:**

- Ten presente que cuando renombres los archivos no debes usar espacios en blanco como separadores, sino ceñirte exclusivamente a emplear barras bajas (\_) o puntos (.)

**Archivo manage\_input\_files.sh**

Para renombrar los archivos, se puede crear un script de BASH dentro de la carpeta «~/projects/examplemia/data/renamed\_data», llamado «manage\_input\_files.sh». En ese script crearemos un enlace simbólico (utilizando el comando ln -s) para cada archivo «fastq.gz» de la carpeta «~/projects/examplemia/data/original\_data» (en el ejemplo son 18 archivos: 9 muestras con dos pareadas cada una). La salida del comando donde se guarden las redirecciones será la carpeta «~/projects/examplemia/data/renamed\_data», donde se guardarán los enlaces simbólicos generados con el nombre de las muestras corregidos.

En el ejemplo, el archivo «manage\_input\_files.sh» quedaría de la siguiente manera:

```

1 #!/usr/bin/env bash
2
3 ln -s ~/projects/examplemia/data/original_data/xxx1_1.fastq.gz ~/projects/examplemia/data/renamed_data/CTL_1_1.fastq.gz
4 ln -s ~/projects/examplemia/data/original_data/xxx1_2.fastq.gz ~/projects/examplemia/data/renamed_data/CTL_1_2.fastq.gz
5
6 ln -s ~/projects/examplemia/data/original_data/xxx2_1.fastq.gz ~/projects/examplemia/data/renamed_data/CTL_2_1.fastq.gz
7 ln -s ~/projects/examplemia/data/original_data/xxx2_2.fastq.gz ~/projects/examplemia/data/renamed_data/CTL_2_2.fastq.gz
8
9 ln -s ~/projects/examplemia/data/original_data/xxx3_1.fastq.gz ~/projects/examplemia/data/renamed_data/CTL_3_1.fastq.gz
10 ln -s ~/projects/examplemia/data/original_data/xxx3_2.fastq.gz ~/projects/examplemia/data/renamed_data/CTL_3_2.fastq.gz
11
12 ln -s ~/projects/examplemia/data/original_data/xxx4_1.fastq.gz ~/projects/examplemia/data/renamed_data/genA_1_1.fastq.gz
13 ln -s ~/projects/examplemia/data/original_data/xxx4_2.fastq.gz ~/projects/examplemia/data/renamed_data/genA_1_2.fastq.gz
14
15 ln -s ~/projects/examplemia/data/original_data/xxx5_1.fastq.gz ~/projects/examplemia/data/renamed_data/genA_2_1.fastq.gz
16 ln -s ~/projects/examplemia/data/original_data/xxx5_2.fastq.gz ~/projects/examplemia/data/renamed_data/genA_2_2.fastq.gz
17
18 ln -s ~/projects/examplemia/data/original_data/xxx6_1.fastq.gz ~/projects/examplemia/data/renamed_data/genA_3_1.fastq.gz
19 ln -s ~/projects/examplemia/data/original_data/xxx6_2.fastq.gz ~/projects/examplemia/data/renamed_data/genA_3_2.fastq.gz
20
21 ln -s ~/projects/examplemia/data/original_data/xxx7_1.fastq.gz ~/projects/examplemia/data/renamed_data/genB_1_1.fastq.gz
22 ln -s ~/projects/examplemia/data/original_data/xxx7_2.fastq.gz ~/projects/examplemia/data/renamed_data/genB_1_2.fastq.gz
23
24 ln -s ~/projects/examplemia/data/original_data/xxx8_1.fastq.gz ~/projects/examplemia/data/renamed_data/genB_2_1.fastq.gz
25 ln -s ~/projects/examplemia/data/original_data/xxx8_2.fastq.gz ~/projects/examplemia/data/renamed_data/genB_2_2.fastq.gz
26
27 ln -s ~/projects/examplemia/data/original_data/xxx9_1.fastq.gz ~/projects/examplemia/data/renamed_data/genB_3_1.fastq.gz
28 ln -s ~/projects/examplemia/data/original_data/xxx9_2.fastq.gz ~/projects/examplemia/data/renamed_data/genB_3_2.fastq.gz

```

El paso siguiente es la ejecución de este script (recuerda que hay que darle a este script permisos de ejecución mediante el comando chmod 755 para que pueda ser ejecutado).

```

1 chmod 755 manage_input_files.sh
2 ./manage_input_files.sh

```

### Archivo experiment\_design

Además de los archivos de secuenciación, necesitamos un archivo de descripción de muestras llamado «experiment\_design». Este archivo **tabulado** tiene en la primera columna el nombre de muestra, en la segunda la condición de cada muestra y, en columnas adicionales, la mayor información posible sobre las muestras (si se incluye en el experimento, la edad de los pacientes o medidas de otras técnicas aplicadas sobre las mismas muestras como, por ejemplo, la medición de la concentración del lípido Z).

Un ejemplo de archivo experiment\_design se ilustra a continuación:

	sample	group	age	lipidZ
1	CTL_1	control	10	0.908
2	CTL_2	control	12	1.243
3	CTL_3	control	13	0.994
4	genA_1	genA	9	0.809
5	genA_2	genA	12	0.945
6	genA_3	genA	14	0.709
7	genB_1	genB	12	0.123
8	genB_2	genB	7	0.540
9	genB_3	genB	11	0.408
10				

Este archivo se debe generar en «~/projects/examplemia/analysis/RNA\_seq/DEG\_workflow». Recomendamos que se usen los identificadores «sample» y «group» para identificar las muestras y grupos, respectivamente. Ambas columnas son **obligatorias** para que se pueda realizar el análisis de los datos.

#### WARNING: Uso de los nombres de las muestras:

- Observa que, a pesar de trabajar con lecturas pareadas, solo incluimos el nombre de la muestra (CTL\_1), y no el identificador de pareada (CTL\_1\_1) que se le dio al hacer los enlaces simbólicos. Esto es esencial para que el flujo de trabajo funcione correctamente.

### Archivo samples\_to\_process.lst

Otro archivo esencial para ejecutar el flujo es una lista de las muestras con los nombres nuevos llamado «samples\_to\_process.lst» y que se debe crear y guardar en la ruta del flujo de trabajo: «~/projects/examplemia/analysis/RNA\_seq/DEG\_workflow/samples\_to\_process.lst».

Este archivo se puede crear a partir de la información almacenada en el archivo «manage\_input\_files.sh» mediante la concatenación de varios comandos:

- cut -f 1 experiment\_design (seleccionamos la primera columna del archivo experiment\_design, que contiene los nombres de las muestras).
- tail -n + 2 (seleccionamos todas las filas menos la primera, que corresponde con el cabecera «sample» y que no es un nombre de muestra a procesar).

Por lo tanto, los comandos a ejecutar para generar el archivo samples\_to\_process.lst serían los siguientes:

```
1 cd ~/projects/examplemia/analysis/RNA_seq/DEG_workflow
2 cut -f 1 experiment_design | tail -n +2 > samples_to_process.lst
```

Este archivo samples\_to\_process.lst contendrá los nombres de los archivos para su preprocesamiento y mapeo en una lista separada por saltos de línea:

```

1 CTL_1
2 CTL_2
3 CTL_3
4 genA_1
5 genA_2
6 genA_3
7 genB_1
8 genB_2
9 genB_3

```

La ventaja de utilizar un sistema de nombres de archivo es que si en un caso hubiera que eliminar una muestra del análisis (por ejemplo, que las lecturas estén corrompidas), con simplemente retirar el identificador de la muestra sería suficiente para re-ejecutar el procedimiento. Recuerda que este archivo sólo se empleará en la etapa de limpieza y mapeo, no en la de análisis de expresión diferencial.

#### **WARNING: Con respecto a la lista de nombres de muestras:**

- En caso de que alguna muestra quisiera eliminarse de una comparación, tendría que emplearse una blacklist y considerarla a la hora de generar los targets en el config\_daemon. Más información en 12.1.1.

#### **Sobre caracteres especiales:**

En muchos casos sucede que, cuando trabajamos con la terminal y copiamos y pegamos contenido, las tabulaciones se malinterpretan y son sustituidas por espacios. Esto podría hacer que haya errores de ejecución del flujo. Lo mismo pasa cuando se manipulan manualmente archivos: es común añadir un salto de línea extra que hace que haya errores de ejecución. Asegúrate bien de no confundirte al generar estos archivos para evitar problemas. Si quieres saber más sobre como asegurarte del contenido de tu archivo, revisa la sección 0.4.6.

Antes de continuar, **asegúrate que tienes los siguientes archivos** preparados y bien configurados en tu ruta de ejecución DEG\_workflow:

- manage\_input\_files.sh — Preparar los enlaces simbólicos a los archivos de lecturas.  
Renombrado de manage\_input\_files\_example.sh
- experiment\_design — Diseño experimental.
- samples\_to\_process.lst — Muestras que se van a emplear para el análisis.

Además, asegúrate también que en la carpeta de archivos renombrados (raw\_data) los enlaces simbólicos están correctamente creados.

#### **Configuración de las opciones del flujo:**

En la carpeta «~/projects/examplemia/analysis/RNA\_seq/DEG\_workflow/» encontraremos dos archivos principales:

- daemon.sh — Ejecución del flujo.
- config\_daemon — Configuración del flujo.

**daemon.sh**

El archivo daemon.sh es el ejecutable principal del flujo, a partir del cual podremos ejecutar todas las fases del flujo. Este archivo **no se debe editar en ningún momento**, sino que solo se ejecutará.

**WARNING: Ten presente:**

- Modificaciones que afecten la estructura del programa daemon.sh harán que el flujo deje de funcionar correctamente. Puedes abrir el archivo para ver su contenido, **pero en ningún momento lo modifiques**.

### **config\_daemon**

El archivo config\_daemon es el archivo de configuración del flujo completo y es el único que se puede editar de forma manual. Al clonar el repositorio DEG\_workflow, el archivo config\_daemon no existe: lo que hay es una plantilla de este archivo llamada «config\_daemon\_default», por lo que el primer paso será crearlo. Dentro de la carpeta del flujo de trabajo:

```
cp config_daemon_default config_daemon
```

El siguiente paso es configurar todas las variables del archivo config\_daemon. Hay explicaciones de cada opción en el archivo (en inglés, comentadas con «#» al final de cada comando). Recomendamos encarecidamente la revisión de las distintas variables para proceder con la edición de las mismas antes de continuar con este manual.

### **Variables del diseño experimental**

- export experiment\_type="RNAseq\_genome" — Definir el tipo de experimento. Por ejemplo, si se va a mapear contra un genoma de referencia (RNAseq\_genome), contra un transcriptoma (RNAseq\_transcriptome) o detección de miRNAs (miRNAseq\_detection).
- export organism="human" — Definir el organismo al que pertenecen las muestras del estudio.
- export mapping\_ref — Definir la ruta donde se descargará, guardará e indexará el archivo de referencia. En caso de utilizar el clúster Picasso y pertenecer al grupo bio\_267, utilizar la configuración por defecto. Referencia indexada (genome.fa) y anotaciones (annotation.gtf) disponibles para *H.sapiens* (versión del ensamblaje hsGRc37 y hsGRc38) y *M. musculus* (mmGRcm38).
- export TRIM\_TEMPLATE="transcriptomics.txt" — Ruta para el archivo del flujo de trabajo que se encarga de hacer la limpieza y mapeo de las muestras. Definir según el experimento a llevar a cabo.
- export transcriptome="" — Definir ruta en caso de usar un transcriptoma de referencia.

### **Variables de ejecución**

- export RESOURCES — Configuración de los recursos del sistema (número de cores, memoria, tiempo de ejecución o tipos de nodo). Configurar de forma acorde al sistema de colas del clúster en el que se lleve a cabo la ejecución.
- export project\_folder — Definir la carpeta donde se guarden los resultados de la ejecución.
- export MAPPING\_RESULTS\_FOLDER=\$project\_folder'/clean\_and\_map' — Nombre de la carpeta donde se guardan los resultados de la limpieza y mapeo contra la referencia.
- export TARGETS\_FOLDER=\$CODE\_PATH'/TARGETS' — Definir carpeta donde se guardarán los archivos de targets (se explicarán en la sección 12.1.1).
- export link\_path="" — Definir, si se tiene, una ruta a una carpeta con muestras pre-procesadas.
- export launch\_login=FALSE — Por defecto, lanza el análisis al sistema de colas. Cambiar a TRUE si se desea ejecutar en el login.

### **Otras variables de interés**

- export min\_too\_short=2 — Cuando en el mapping report el porcentaje de lecturas por muestra que están etiquetadas como Unmapped: too short (que una parte de la lectura no alinea contra el genoma de referencia) supera el valor de esta variable, se activa su

protocolo de análisis. Esto puede suceder porque haya contaminación de las muestras.  
El protocolo reads to run the protocol to inspect them

- export unmapped\_ratio=0.8 — Porcentaje para la selección de las lecturas que no superen ese umbral de alineamiento contra la referencia. Lecturas que lo superen se seleccionarán para el análisis posterior.
- export HUNTER\_RESULTS\_FOLDER=\$project\_folder'/ExpHunterSuite\_results' — Directorio de salida de los resultados del análisis de expresión diferencial.
- export MIN\_READ\_LENGTH=65 — Tamaño mínimo de lectura. Considerar el tamaño de las lecturas en las muestras y una vez realizado el proceso de limpieza.
- export read\_layout='paired' — Configurar a «single» si las lecturas NO son pareadas.
- export clean\_parentals=FALSE — usado en el análisis funcional para eliminar redundancia con los términos parentales.
- keep\_bam=TRUE/FALSE — Para no eliminar el sorted\_mappings.bam del proceso de mapeo en caso de que sean necesarios.

### Generación de targets

La opción «generate\_targets» es la más compleja por lo que se comentará más en detalle. Esta variable contiene código de BASH que se ejecutará siempre a la vez que daemon.sh. Esta sección se encarga de generar una serie de archivos esenciales para el funcionamiento de la herramienta ExpHunter Suite, llamados TARGETS. Estos targets son una modificación del archivo experiment\_design que se generan mediante el script generate\_targets.rb (carpeta aux\_parsers), y le dicen a ExpHunter Suite cómo comparar las muestras, por lo que habrá que generar uno de ellos para cada comparación.

Cómo se comentó antes, en el ejemplo propuesto se hacen 4 comparaciones: CTL vs genA + genB, CTL vs genA, CTL vs genB y genA vs genB. Para generar cada *target*, hay que indicarle a generate\_targets.rb qué muestras del experimento se usarán como controles y cuáles como tratamiento, a través de un texto como argumento del flag «-t».

El texto tiene una estructura concreta especificando el nombre de la comparación, el carácter «>» como separador, el nombre de la columna del «experiment\_design» que se usará para separar entre controles y tratamientos, el carácter «::» como separador, las variables de dicha columna que se usarán como controles (separadas por «/»), el carácter «,» como separador, y por último, las variables que serán usadas como tratamiento (también separadas por «/»).

Un ejemplo de ejecución de generate\_targets sería el siguiente:

```
1 nombre_del_target>columna_del_experiment_design:variable(s)_control,
  variable(s)_tratamiento
```

En el caso de generar los *targets* del ejemplo, habría que ejecutar los siguientes comandos (dentro de la carpeta DEG\_workflow):

```
1 export generate_targets=""
2 generate_targets.rb -o $TARGETS_FOLDER -e experiment_design -t
  'CTL_vs_MUT>group:control,genA/genB'
3 generate_targets.rb -o $TARGETS_FOLDER -e experiment_design -t
  'CTL_vs_genA>group:control,genA'
4 generate_targets.rb -o $TARGETS_FOLDER -e experiment_design -t
  'CTL_vs_genB>group:control,genB'
5 generate_targets.rb -o $TARGETS_FOLDER -e experiment_design -t
  'genA_vs_genB>group:genA,genB'
6 "
```

El programa generate\_targets.rb puede incluir varios argumentos de entrada adicionales para personalizar los targets y llevar a cabo análisis multifactoriales, los cuales se pueden consultar en generate\_targets.rb -h

### Personalización de targets

Para personalizar los targets existen varios argumentos que se explicarán a continuación:

1. Se pueden emplear listas negras con identificadores de muestras que no se quieran emplear en el estudio, o bien porque no nos interesan al ver que dan problemas en el análisis o porque queremos personalizar nuestro target de manera específica (por ejemplo, queremos quitar todos los controles). Para ello se debe crear en el directorio raíz del flujo de ExpHunter Suite un archivo con los identificadores separados por saltos de línea. Recomendamos que el archivo se llame blacklist o similar. Y para que el programa generate\_targets.rb lo reconozca hay que activar el parámetro -b y darle de argumento el nombre del archivo blacklist.

```
1 -b blacklist
```

2. También se puede emplear la antítesis del blacklist, es decir, un whitelist en el que se especifiquen las muestras que queremos incluir en los targets en el caso de que no sean todas. De igual forma que con las blacklist, se crea en el directorio raíz del flujo de ExpHunter Suite un archivo con los identificadores separados por saltos de línea. Se recomienda nombrarlo como whitelist e incluirlo en generate\_targets.rb mediante el argumento -w añadiendo el archivo.

```
1 -w whitelist
```

3. Por otro lado, en el caso de que queramos hacer comparaciones dentro de un grupo de muestras con una característica seleccionada, se usa el parámetro -f de filter. Por ejemplo, imaginemos que tenemos controles y pacientes y que sobre ambos se han aplicado tratamientos. Si queremos ver el efecto del tratamiento sobre los pacientes en exclusiva, tenemos que aislar a los pacientes y comparar los resultados de haber llevado (sí) o no un tratamiento sobre ellos. Para ello, la opción -f permite definir el grupo con el que se quiere trabajar. El argumento de entrada a la opción -f incluye el identificador de la condición y separado por un igual el tipo de condición sobre la que se quiere llevar el análisis. Por ejemplo, si nuestro grupo es control/paciente y queremos seleccionar solo los pacientes:

```
1 -f "group=patient"
```

4. Por último, en el caso en el que tengamos dos grupos de individuos, por ejemplo sano y enfermo, con diferentes condiciones, control y tratamientos, y queramos comparar una condición de un individuo sano frente a otra condición de un individuo enfermo, al escribir el comando se especificarán tanto las condiciones a estudiar, separadas por comas, y los tipos de individuos a estudiar, también separados por comas, y a su vez separadas ambas categorías por punto y coma de la siguiente forma:

```
1 nombre_del_target>col_categoria_1_experiment_design:variable(s)
    _control,variable(s)_tratamiento;
    col_categoria_2_experiment_design:variable(s)_control,variable(
        s)_tratamiento
```

### Análisis multifactorial

Con todo esto comprendido y configurado para llevarse a cabo, podemos empezar con las fases de análisis. En primer lugar, se llevará a cabo la descripción de la descarga e indexado de la referencia, seguido del trimming y mapeo de las muestras frente a la referencia (obtención de la tabla de conteo), el análisis de expresión diferencial y finalmente el análisis de enriquecimiento funcional. Todo ello será ejecutado a través del programa **daemon.sh**.

#### 12.1.2 Fase 1: Descarga e indexado de las referencias

Esta fase es preparatoria para las siguientes. Se encarga de descargar todos los archivos necesarios para ejecutar el mapeo. Por ahora solo está puesta a punto la descarga de las referencias de genoma de humano y ratón, que se ejecutan con «./daemon.sh 1a».

En el caso de que se quiera usar otras versiones de los genomas o el genoma de otro organismo modelo, habrá que descargar de forma manual el archivo FASTA del FTP de Ensembl `ftp://ftp.ensembl.org/pub` y su correspondiente archivo de anotación en GTF. Estos dos archivos deberán guardarse en la carpeta configurada como «\$mapping\_folder» en el archivo «config\_daemon» con los nombres de «genome.fa» y «annotation.gtf».

#### WARNING: Check del flujo

Estos dos archivos son los que se crean cuando se ejecuta el comando «./daemon.sh 1a». En este caso, el mensaje *«[Human or Mouse] genome and annotations has been downloaded»* indicará que todo ha ido de forma correcta.

En el caso de que se use el modo «RNaseq\_transcriptome», hay que descargar la referencia de una fuente fiable. En lo referente al flujo, hay que definir la variable «\$transcriptome» con el path al archivo fasta del transcriptoma. Por norma general, los transcriptomas se usan para análisis de expresión de especies poco estudiadas por lo que suelen ser transcriptomas ensamblados *de novo*, y por lo tanto, no suelen tener identificadores de Ensembl propios de la especie. Lo que suele ocurrir es que el transcriptoma esté anotado con los genes o proteínas de la especie modelo más cercana.

Hay opciones para disponer esos identificadores anotados:

1. Sustituir los identificadores de los transcritos del FASTA por los identificadores de Ensembl de la especie más cercana, de tal manera que el FASTA incluya toda la información.
2. Crear un archivo de anotación que consiste en tabla tabulada sin cabecera, en el que la primera columna son los identificadores de Ensembl y la segunda los correspondientes identificadores del transcriptoma. Habrá que definir la variable «\$annotation\_list» con el nombre de este archivo.

Tanto si estamos en el modo «RNaseq\_genome» como en «RNaseq\_transcriptome» hay que ejecutar «./daemon.sh 1b» para crear un índice de la referencia para que pueda ser manejado por los programas de mapeo.

**Ejecución en del flujo:**

El flujo se puede ejecutar tanto en un clúster de computación como Picasso, como en un ordenador personal que disponga de los recursos necesarios. Sin embargo, en este manual solo se comentará la ejecución en clúster. Para ello se ha configurado la variable «\$launch\_login» del «config\_daemon», que deberá configurarse como «FALSE», así los trabajos se enviarán a un sistema de colas con la excepción del comando «./daemon.sh 1a» que se completará en la sesión del «login» desde la que se ejecute, ya que el sistema de colas no tiene acceso a Internet.

**WARNING: Comprobación del flujo**

Una vez finalizado el indexado habrá que comprobar el archivo de log «index.[id\_del\_trabajo].out» que se genera en la misma carpeta donde se encuentra el «daemon.sh». En el caso del modo «RNAseq\_genome» debe de estar el mensaje “” al final del archivo. En el caso del modo «RNAseq\_transcriptome» el mensaje que debe estar es “Total time for backward call to driver() for mirror index”.

**12.1.3 Fase 2: Pre-procesamiento y mapeo**

En esta fase se pre-procesarán las lecturas de cara a descartar secuencias con poca calidad, contaminantes y RNA ribosómicos que puedan crear ruido en el análisis. Luego, estas lecturas se mapearán sobre la referencia. Todo se hace de forma automatizada mediante sub-flujos creados en AutoFlow<sup>1</sup>. Se ejecuta el sub-flujo por cada muestra por lo que obtendremos una tabla de conteos por cada una de ellas. La forma de ejecutar esta sección es lanzando «./daemon.sh 2a», que enviará todos los trabajos al sistema de colas. Una vez confirmados que todos los trabajos se han ejecutado con el comando squeue, confirmaremos que todos los trabajos se han ejecutado correctamente con el comando ./daemon.sh 2b. Este comando ejecuta un programa que está guardado en la carpeta aux\_sh llamado check\_wf.sh y ejecuta la funcionalidad flow\_logger de Autoflow para confirmar que todos los trabajos se han llevado a cabo de manera satisfactoria.

**WARNING: Comprobación del flujo**

Una vez terminen todos los trabajos del sistema de colas habrá que comprobar que hayan terminado correctamente. Para ello hay que ejecutar «./daemon.sh 2b», que devolverá una tabla con información de cada uno de los trabajos. Los trabajos finalizados correctamente se marcarán con «SUCC», mientras que los que finalicen con errores se marcarán con «ERR». En este último caso habrá que comprobar el fallo en la carpeta de resultados y re-ejecutar antes de continuar.

Cuando trabajamos con un supercomputador hay casos en el que las máquinas pueden cortar tareas y dar lugar a errores de ejecución. Puede pasar que para un experimento tengamos tareas terminadas con algunas muestras (SUCC) y que en otras tengamos una tarea terminada

<sup>1</sup>P. Seoane y otros (2016). AutoFlow, a Versatile Workflow Engine Illustrated by Assembling an Optimised de novo Transcriptome for a Non-Model Species, such as Faba Bean (*Vicia faba*). *Current Bioinformatics* 11(4): 440-450.

y otras que no (ABORT). Caracterizar estos errores es importante y para ello se deben revisar los archivos de log tanto .out como .err para ver, respectivamente, la máquina donde se ha ejecutado la tarea y el error (en caso de que lo haya). En el caso de que haya errores (ABORT) por algunas tareas en concreto, se puede añadir un argumento de entrada al comando ./daemon 2b para que re-lance las tareas que han abortado o no se han lanzado.

```
> ./daemon 2b '-lp'
```

Para que esto sea efectivo y se resuelvan correctamente las tareas hay que asegurarse de que el error ha sido debido a un fallo en las máquinas (generalmente, en la carpeta de resultados los archivos log de error y process\_data vienen vacíos).

#### WARNING: ¡IMPORTANTE!

Dado que el programa que se utiliza para la limpieza de los archivos de lecturas es SeqTrimBB, su configuración hace que para volver a ejecutar una limpieza fallida sea necesario borrar las carpetas temporales que se generan dentro del directorio seqtrimbb\_0000/. Si borramos las carpetas de las ejecuciones en lugar de las específicas de limpieza (lo cual puede ser una opción tentadora por comodidad), la opción -l de flow\_logger no encontrará las carpetas para relanzar y los trabajos nunca entrarán en cola. Por lo tanto, nuestra recomendación es que en el caso de que falle la limpieza se vaya a cada una de las carpetas de ejecución y se borre la carpeta seqtrimbb\_0000/. En el caso de que por imprudencia se hayan borrado las carpetas de las ejecuciones, la solución sería lanzar de nuevo el ./daemon 2a pero configurando el archivo samples\_to\_process.lst incluyendo los identificadores de las muestras cuyo procesamiento ha fallado. Recuerda siempre generar un archivo backup con tu lista al completo de muestras para no perderlo y sustituir de nuevo el archivo samples\_to\_process.lst al original cuando se quiera ejecutar el análisis de expresión diferencial o el funcional.

#### Protocolo de análisis de lecturas too short mapping

En múltiples ocasiones hemos observado que existen muestras contaminadas que nos introducen sesgos a la hora de hacer los análisis de expresión diferencial. Uno de los más frecuentes es la elevada cantidad de muestras que tienen lecturas en las que una parte alinea correctamente contra el genoma de referencia y otra parte que no. Estas lecturas se las denomina como «short reads» (en el informe de calidad de limpieza y mapeo, *mapping\_report.html*).

Cuando se detecta un porcentaje de lecturas «too short» superior al valor configurado en la variable min\_too\_short del config\_daemon, se ejecuta dentro del flujo de trabajo el protocolo de análisis de este tipo de lecturas. Con más detalles, el protocolo incluye las siguientes etapas:

1. STAR se configura y ejecuta para obtener todo lo que no mapea a FASTQ (que no coincide la secuencia o que alinee de forma parcial).
2. Los FASTQ se alinean contra el genoma de referencia usando STAR con parámetros para que no los etiquete como «too short».
3. Se genera un BAM que se parsea con el script get\_too\_short.py. Esto devuelve un .bam y un .fasta con las lecturas que sean iguales o superen el valor configurado en el config\_daemon con la variable unmapped\_ratio (80% por defecto) de la secuencia que no alinea contra el genoma de referencia.
4. Del FASTA se pueden hacer BLAST en el NCBI de la secuencia para averiguar qué son

esas lecturas ( posible localización de contaminantes).

Salvo este parámetro, junto con export generate\_targets (que se explicará a continuación), el resto son intuitivos de seguir, ya que mayoritariamente hacen referencia a valores de filtro (como *P*-valores o logFC, entre otros).

#### 12.1.4 Fase 3: ExpHunter Suite y comprobación de las muestras

Tras conseguir las tablas de conteo por cada muestra hay que combinarlas y ejecutar ExpHunter Suite. Este procedimiento se lleva a cabo gracias a «./daemon.sh 3», que llamará al programa compare\_all\_samples.sh dentro de la carpeta aux\_sh. Lleva a cabo las siguientes funcionalidades:

1. Creación de un report (informe) de mapeo, que describirá el estado de las muestras, y se podrán encontrar errores en la secuenciación, etiquetado y en la elección de parámetros de pre-procesamiento y mapeo. Este report consiste en un HTML llamado «mapping\_report.html» alojado en la carpeta definida en «\$report\_folder».
2. Combinación de las tablas de conteo en una matriz de conteo.
3. Ejecución de ExpHunter Suite. Por cada comparación se creará una carpeta localizada en «\$HUNTER\_RESULTS\_FOLDER» con su respectiva ejecución. Al ejecutar «./daemon.sh 3» se enviará un único trabajo al sistema de colas (en caso de que la variable launch\_login en el archivo config\_daemon esté configurada como FALSE). Al finalizar, estarán disponibles los resultados del análisis de expresión diferencial y de co-expresión en un único informe.

##### Creación de report de mapeo

En primer lugar, el programa create\_metric\_table.rb dentro de la carpeta aux\_parsers generará una tabla con datos recopilados de la ejecución, por cada muestra analizada, sobre valores de referencia de la limpieza y alineamiento. Con ello, se ejecutará el programa create\_report.R, que será el encargado de preparar el mapping\_report.html.

##### Generación de matriz de conteo

A continuación, se llamará al programa maps2DEGhunter.rb, el cual por cada una de las muestras tomará los archivos selected\_counts generados tras la ejecución del mapeo contra la referencia, y creará una tabla de conteo única donde, por columna, se incluirá el identificador de la muestra, y por fila los genes expresados con los valores de expresión por muestra. Asimismo, se llevará a cabo un filtrado de isoformas con el script sum\_counts\_by\_isoform.rb para, finalmente, generar el archivo final\_counts.txt, que servirá como entrada a ExpHunter Suite para hacer el análisis de expresión diferencial según las comparaciones (targets) definidas.

##### Ejecución de ExpHunter Suite

La llamada al módulo de expresión de ExpHunter Suite (degenes\_Hunter.R) tomará los parámetros de configuración definidos en el config\_daemon, el archivo final\_counts.txt y el target definido (se ejecuta tantas veces ExpHunter como targets haya, guardándose los resultados en diferentes carpetas). Se debe obtener, además de archivos auxiliares, un archivo HTML con el report final del análisis de expresión diferencial.

ExpHunter Suite se puede configurar para que ejecute los análisis con los paquetes de expresión que se desee, activar o desactivar el módulo de co-expresión, ajustar los *p*-valores y el logFC, entre otros. Más información sobre la herramienta y su funcionamiento en 11.

**WARNING: Comprobación del flujo**

Una vez termine el trabajo «compare\_all\_samples.sh», dentro de las carpetas de cada *target*, se habrá creado un archivo «DEG\_report.html» correspondiente a los informes del análisis de expresión. Solo se podrá continuar si estos informes están creados y si los resultados son coherentes. En el caso de que los informes no estén, dentro de la carpeta de cada *target* hay un archivo de log llamado «degenes\_Hunter.log» en donde se mostrarán los errores.

**Resultados de ExpHunter Suite**

ExpHunter Suite, con la configuración por defecto del flujo, realiza un análisis de expresión diferencial, es decir, busca el conjunto de genes que se expresan de manera significativa en los tratamientos con respecto a los controles. Además, realiza un análisis de co-expresión que devuelve una serie de módulos o clústeres de co-expresión. Dentro de estos módulos, todos los genes cambian su expresión con la misma tendencia a lo largo de las muestras, con la excepción del módulo 0, que es donde se guardan los genes que no se parecen estadísticamente a ningún otro.

**12.1.5 Fase 4: Enriquecimiento funcional**

Esta fase se encarga de llevar a cabo el enriquecimiento funcional de los DEG y CEG que hayan sido caracterizados durante la etapa de análisis de expresión diferencial. En este caso, ejecutando ./daemon 4a se llama al programa launch\_fun\_hun.sh dentro de la carpeta aux\_sh. Este script prepara los archivos necesarios para llevar a cabo el análisis funcional de los genes expresados diferencialmente y co-expresados con el programa functional\_Hunter.R.

La llamada al programa incluye una serie de argumentos de entrada, entre los cuales se incluye la carpeta con los resultados de la ejecución de degenes\_Hunter.R, los filtros de p-valor, el *background* (si todos los genes de la especie o el total de expresados), las bases de datos a emplear y la especie, entre otros. Más información sobre el uso de la herramienta en 11.

**WARNING: Comprobación del flujo**

Habrá que confirmar la existencia de dos informes HTML que serán los marcadores del funcionamiento correcto del flujo. Los dos informes se llaman «functional\_report.html» y «clusters\_func\_report.html», y se encuentran en la carpeta «functional\_enrichment», dentro de la carpeta de resultados de cada *target*. En el caso de que los informes no estén, dentro de la carpeta de cada *target* hay un archivo de log llamado «functional\_Hunter.log» en donde se mostrarán los errores.

**12.1.6 Fase 5: Preparación de paquetes**

La última etapa del flujo de trabajo consiste en preparar paquetes de trabajo para entregarlos a los grupos de colaboración una vez analizados los datos.

Para ello, una vez se hayan realizado las diferentes comparaciones y estén listas para entregar a los grupos colaboradores, hay que preparar el sistema de carpetas de resultados. Se utilizará el módulo 5 del daemon (./daemon.sh 5). Este programa utiliza la herramienta

create\_hunter\_pack.sh (localizado en /mnt/home/soft/soft\_bio\_267/programs/x86\_64/scripts, por lo que no hay que inicializar nada), la cual selecciona los archivos y carpetas más relevantes o de interés para un grupo de investigación, como los archivos mapping\_report.html, DEG\_report.html, final\_counts.txt y todo el contenido de la carpeta donde se guarda la información del análisis funcional, entre otros.

Se recomienda encarecidamente que el nombre de la carpeta de resultados sea *results* seguido de la fecha de creación, por ejemplo: results\_11\_06\_2024.

Finalmente, el/los programas hayan terminado de recopilar la información, comprimiremos la carpeta empleando el comando zip -r, que tiene como primer argumento el nombre del archivo comprimido y como segundo el path a la carpeta que contiene el paquete de resultados. Por coherencia con lo explicado previamente, se recomienda que el nombre del archivo sea el nombre del proyecto seguido de results y la fecha de creación, con la extensión .zip. Ejemplo: ObesityMiceResults\_11\_06\_2024.zip

```
1 zip -r folder.zip path_to_savefolder
```

*He dejado abajo lo que explicasteis inicialmente. La parte de create\_cormit\_pack.sh no la he visto y no quiero meter la pata (creo que ahora es igual que RNA\_daemon.sh 5?), por si podéis echar un vistazo, que no lo he tocado. Por otro lado, ¿supongo que antes no estaba incluido en Autoflow? Porque ahora con ejecutar daemon.sh 5 ya crea el paquete sin necesidad de dar argumentos (ya están en el propio script).*

## 12.2 Funcionalidades externas al DEG\_workflow

Aún por implementar en el flujo de trabajo, existe la posibilidad de preparar paquetes de trabajo para que se entreguen a los grupos de colaboración una vez que se hayan analizado.

### 12.2.1 Preparación de paquetes de trabajo para su entrega

Una vez que tengamos toda la información lista para su entrega a los grupos de trabajo colaboradores, hay que preparar el sistema de carpetas de resultados. Para ello, en nuestro \$HOME crearemos una carpeta llamada NGS\_results en la cual volcaremos el contenido de los paquetes de entrega. Por cada entrega se deberá crear una subcarpeta dentro de NGS\_results que tenga un nombre identificativo del proyecto. Para volcar la información emplearemos la herramienta create\_hunter\_pack.sh (localizado en /mnt/home/soft/soft\_bio\_267/programs/x86\_64/scripts, por lo que no hay que inicializar nada). Este programa selecciona los archivos y carpetas más relevantes o de interés para un grupo de investigación, como los archivos mapping\_report.html, DEG\_report.html, final\_counts.txt y todo el contenido de la carpeta donde se guarda la información del análisis funcional, entre otros.

De argumentos de entrada le daremos a create\_hunter\_pack.sh en primer lugar el path a donde tenemos nuestras carpetas de resultados de la ejecución (el directorio donde están las carpetas clean\_and\_map, DEGenesHunter\_results y mapping\_reports) y de segundo argumento el path y nombre de nuestra carpeta de resultados. En el caso de hacer un análisis de miRNA, el programa requiere un tercer argumento de entrada donde se indicará el path a la carpeta de detección miRNA. Esto incluirá tres archivos adicionales en la carpeta que se entregará como paquete de trabajo.

Asimismo, si se realiza un análisis funcional de los targets de miRNA, la preparación del paquete se lleva a cabo mediante la ejecución de otro programa, create\_cormit\_pack.sh,

que como primer argumento de entrada debe aportársele la directorio donde está el **resultado** del flujo de ExpHunterSuite\_CorMiT\_benchmark (directorio que incluye los resultados de los miRNA-diana de coRmiT **coRmiT\_0000** y el funcional de las dianas **clusters\_to\_enrichments.R\_0000**). Como segundo argumento de script hay que indicar el nombre de la comparación de degenes\_Hunter que se ha usado, ya que dicho flujo se ejecuta una vez por cada comparación. Estos resultados deberán incluirse en un directorio de resultados creado de forma **manual**.

Se recomienda encarecidamente que el nombre de la carpeta de resultados sea *results* seguido de la fecha de creación, por ejemplo: results\_22\_07\_2022.

Finalmente, cuando el/los programa/s hayan terminado de recopilar la información, comprimiremos la carpeta empleando el comando zip -r, que como primer argumento tendrá el nombre del archivo comprimido y como segundo argumento el path a la carpeta que resulta de la ejecución de create\_hunter\_pack.sh y/o create\_cormit\_pack.sh. Por coherencia con lo explicado previamente, se recomienda que el nombre del archivo coincida con el de la carpeta (*results* seguido de la fecha de creación) y la extensión .zip: results\_22\_07\_2022.zip).

```
1 mkdir path_to_savefolder  
2 create_hunter_pack.sh path_to_results path_to_savefolder  
3 zip -r folder.zip path_to_savefolder
```

## 12.3 Ejemplos de análisis

En esta sección se contemplan una serie de ejemplos de uso para el análisis de datos de expresión con la herramienta ExpHunter Suite. Se explicarán para dos entornos de trabajo distintos: los clúster de Picasso y UPO.

### WARNING: Antes de ponerte con los ejemplos

- Se da por hecho que el lector tiene conocimiento previo del uso básico de Bash (terminal o línea de comandos), incluyendo moverse entre directorios, crear carpetas, leer, eliminar y crear archivos, entre otros básicos. En caso negativo, leer el manual previamente aportado, secciones 0.3 y 0.4.
- Del mismo modo, se da por hecho que el usuario tiene disponibles (Picasso) o instaladas (UPO) previamente las herramientas necesarias para llevar a cabo el análisis de expresión. Para más información, ver la sección 2

### 12.3.1 Análisis de datos RNA-Seq en Picasso a partir del SRA

Vamos a trabajar con muestras de datos de pacientes afectados por COVID e individuos sanos. El estudio en el que nos vamos a centrar tiene una cohorte de 7 pacientes ingresados en UCI con claros síntomas de la enfermedad y confirmados por PCR como positivos al SARS-CoV-2. Las muestras pertenecen a leucocitos aislados. En el caso de los controles corresponden con muestras de 7 individuos sanos, sin síntomas y confirmados con PCR negativa. El estudio se encuentra en revisión y está disponible a través de este enlace.

#### Preparación de datos del estudio

El Sequence Read Archive (SRA) es un servicio del NCBI que ofrece datos de lecturas obtenidas a partir de análisis de secuenciación para estudios (la gran mayoría publicados). El estudio que vamos a usar en este ejemplo tiene siete registros de pacientes confirmados como COVID + por PCR y otros siete COVID - detectados por la misma técnica. El identificador correspondiente en el SRA es el SRP273299, e incluye 14 muestras de acceso público de lecturas single obtenidas mediante la plataforma de secuenciación Illumina (NextSeq 500). Los archivos en el SRA a descargar son los que tienen como extensión .sra.

En primer lugar, vamos a configurar nuestro archivo \$HOME/.bashrc (en el root de nuestra cuenta) para que tengamos siempre disponibles los intérpretes de R y Ruby cuando ejecutemos en la terminal el comando load\_R o load\_ruby, respectivamente. Del mismo modo, incluiremos en la variable \$PATH la ruta a los programas instalados en la cuenta de software del grupo (soft\_bio\_267) para tenerlos siempre disponibles cuando hagamos login (entremos) en Picasso. Para ello, modificamos el archivo incluyendo esta información:

### WARNING: Puede que tengas un archivo similar...

Si ya tienes tu archivo .bashrc o .bash\_rc (se pueden llamar de ambas maneras) modifica su contenido en lugar de crear uno nuevo para evitar conflictos. Por su parte, el archivo .bash\_history guarda los comandos que hayas ejecutado en una sesión hasta su cierre (exit).

```
> alias load_R=' . ~soft_bio_267/initializes/init_R'
```

```
2 > alias load_ruby='. ~soft_bio_267/initializes/init_ruby'
3 > PATH=~/soft_bio_267/programs/x86_64/scripts:$PATH
4 > export PATH
```

Una vez hecho esto, ejecutamos el comando source para hacer efectivos estos cambios en nuestra consola actual:

```
1 > source $HOME/.bashrc
```

A continuación, diseñaremos el sistema de carpetas para trabajar en nuestro proyecto. Trabajaremos tanto en nuestro root o cuenta local (~/) como en el FSCRATCH (\$FSCRATCH) para guardar archivos de gran tamaño (como los de lecturas que descarguemos a través del SRA). En nuestro root creamos una carpeta de proyectos (si no la tenemos ya) y allí crearemos una subcarpeta con el nombre del estudio (/COVID en este caso). Dentro de ella crearemos las carpetas /data y /analysis. En la carpeta /data guardaremos la información necesaria de los datos de lecturas con los que vayamos a trabajar, y en la carpeta /analysis llevaremos a cabo el análisis de expresión con el flujo de ExpHunter Suite. Empezaremos a trabajar la carpeta /data para guardar la información que vayamos generando más adelante, ya que la descarga de datos se realizará automáticamente en el FSCRATCH en una carpeta llamada \$FSCRATCH/sra\_download/sra. A continuación explicamos esto con detenimiento.

Para realizar la descarga de los datos, se puede emplear un protocolo automatizado disponible dentro de las herramientas del SRA toolkit, las cuales se encuentran disponibles en el repositorio de herramientas instaladas de la cuenta soft\_bio\_267. Para cargarlas, ejecutar el siguiente comando:

```
1 > source ~/soft_bio_267/initializes/init_sratookit
```

Esta herramienta habilita una serie de programas que tienen distintas funcionalidades entre las que se incluyen permitir la descarga de información del SRA de manera automatizada o transformar ciertos tipos de archivos para su posterior uso.

A continuación se explicará cómo llevar a cabo la descarga de datos del SRA (servirá tanto para Picasso, UPO como cualquier otro servidor, una vez se hayan instalado o habilitado las herramientas necesarias)

### Descarga de datos del SRA

Para descargar los archivos correspondientes a las muestras de nuestro estudio desde el SRA hay que buscar los 14 identificadores de las muestras, los cuales empiezan por SRR o ERR (en este caso van desde el SRR12313439 al SRR12313452). También funciona con el código GEO de la muestra, que empiezan por GSM (en este caso desde GSM6039206 hasta GSM6039214). En la carpeta ~/projects/COVID/data creamos un archivo de texto que se llame sra\_ids y que tenga los 14 identificadores de las muestras a descargar separados por saltos de línea:

```
1 > SRR12313439
2 > SRR12313440
3 > SRR12313441
4 > SRR12313442
5 > SRR12313443
6 > SRR12313444
7 > SRR12313445
8 > SRR12313446
9 > SRR12313447
```

```
10 > SRR12313448  
11 > SRR12313449  
12 > SRR12313450  
13 > SRR12313451  
14 > SRR12313452
```

Con este archivo sra\_ids preparado, creamos un script que se llame get\_fastq.sh, el cual va a realizar la descarga de los datos llamando a una herramienta provista dentro de la cuenta soft\_bio\_267 llamada ncbi\_sra\_file\_downloader.sh, que tomará como argumento de entrada el contenido de los identificadores del SRA guardados en el archivo sra\_ids. El contenido de get\_fastq.sh sería el que se detalla a continuación:

```
1 > ncbi_sra_file_downloader.sh sra_ids
```

El programa ncbi\_sra\_file\_downloader.sh automáticamente guarda la información descargada en una carpeta que se crea automáticamente en el FSCRATCH y mencionada anteriormente, \$FSCRATCH/sra\_download/sra. Para lanzar este programa y que se ejecute en segundo plano, evitando que nos bloquee el uso de la terminal, lo lanzamos ejecutando el siguiente comando en la terminal:

```
1 > ./get_fastq.sh &>log&
```

Esta ejecución nos permitirá seguir trabajando con la terminal ya que será lanzada en segundo plano (ejecutar el comando top para visualizar si se ha lanzado correctamente, el proceso se denominará prefetch). Para confirmar que todo está funcionando correctamente, se puede listar el contenido de la carpeta \$FSCRATCH/sra\_download/sra y confirmar si el tamaño de los archivos aumenta. Del mismo modo, donde se ejecute el programa get\_fastq.sh se creará un archivo log con el registro de la ejecución.

#### Puede que haya errores:

Desafortunadamente, el servicio que usa el programa ncbi\_sra\_file\_downloader.sh a veces da fallos y no descarga correctamente todos los archivos de lecturas. Para que no cunda el pánico, siempre hay una solución. Si hay algún archivo que se ha quedado en temporal en lugar de con su extensión .sra, se puede borrar los registros completados del archivo sra\_ids, dejando los que no se han completado. Seguidamente, borramos dentro de la carpeta de descarga de datos en el FSCRATCH los registros incompletos y volvemos a ejecutar el programa get\_fastq.sh como se ha indicado previamente. En casos extremos donde no se puedan descargar mediante este sistema los archivos, se puede optar por descargarlos manualmente uno a uno mediante el uso del comando wget empleando el repositorio del SRA donde estén almacenadas las muestras de interés. Recuerda redireccionar la salida del wget con un -O a la carpeta donde guardamos los archivos del SRA para tener centralizadas las descargas en una carpeta común y conserva la extensión del archivo (.sra).

Los archivos del SRA tienen que ser transformados a .fastq para su uso. Tendrán que ser transformados a tales con la herramienta sra2fastq.sh dentro del repositorio sys\_bio\_lab\_scripts. Para ello, creamos un script (por ejemplo, transform\_sra\_to\_fastq.sh) en el que guardemos el siguiente contenido (incluye un cabecera para lanzar al sistema de colas):

```
1 #! /usr/bin/env bash
2 #SBATCH --time=7-00:00:00
3 #SBATCH --mem='30gb'
4 #SBATCH --cpus-per-task=1
5 #SBATCH --constraint=cal
6 #SBATCH --error=job.%J.err
7 #SBATCH --output=job.%J.out
8 output_folder=~/projects/COVID19/data/raw_data
9 mkdir $output_folder
10 sra2fastq.sh sra_ids $FSCRATCH/sra_download/sra $output_folder
```

El programa sra2fastq.sh hace una llamada al siguiente comando (ejecutándolo en el sistema de colas, por cada uno de los archivos que se van a procesar):

```
1 . ~/soft_bio_267/initializes/init_srato toolkit
2 for i in `cat $1`
3   do
4     fastq-dump --split-files --gzip --origfmt $4 --outdir $3 $2"/"${i}" .sra"
5   done
```

Siendo \$1 el archivo con los identificadores del SRA, \$2 el path al archivo con dichos archivos SRA, \$3 es el path de salida y el \$4 las opciones adicionales (en caso de que sean necesarias).

#### ¿Y si tengo archivos en el SRA con dos identificadores por muestra?

En algunos casos, puede suceder que aparezcan dos identificadores de muestras para el mismo run. Para trabajar con estos archivos, a la hora de hacer la conversión del archivo sra a fastq, el argumento –origfmt sirve para recuperar los nombres originales de las lecturas.

#### WARNING: Si tuviste problemas con la descarga del SRA

Asegúrate de que tienes todos los identificadores de las muestras a la hora de ejecutar este script si borraste alguno para hacer la descarga de nuevo de los identificadores, en caso de que te dieran fallo.

Este script se lanzará al sistema de colas (mediante el comando sbatch transform\_sra\_to\_fastq.sh) para que realice la transformación de archivos .sra a .fastq, guardando los resultados en la carpeta ~/projects/COVID19/data/raw\_data.

Con los archivos .fastq disponibles, se puede proceder con el análisis de los mismos. Una vez obtenidos, se pueden borrar los archivos .sra para que no ocupen espacio, ya que no se van a utilizar. **NOTA:** En caso de que se vaya a utilizar DEG\_Workflow, se puede pasar directamente al apartado 12.1 para el análisis de las muestras, a partir del apartado 12.1.1. Si no se va a usar, se procederá según lo establecido a continuación:

Ahora se procederá con la corrección de los nombres de las muestras para que sean representativos. Se harán enlaces simbólicos a los archivos, renombrándolos según las muestras que estamos analizando. Para ello, creamos un archivo manage\_input\_files.sh que tenga el siguiente contenido (**NOTA:** solo se han añadido algunas de las muestras para simplificar el ejemplo, pero hay que incluirlas todas):

```

1  output_folder=~/projects/COVID19/data/raw_data
2  final_folder=~/projects/COVID19/data/final_files
3  mkdir $final_folder
4  cd $final_folder
5  ln -s $output_folder/SRR12313439_1.fastq.gz ctrl_PCR_neg_1.fastq.gz
6  ln -s $output_folder/SRR12313440_1.fastq.gz ctrl_PCR_neg_2.fastq.gz
7  ln -s $output_folder/SRR12313441_1.fastq.gz ctrl_PCR_neg_3.fastq.gz
8  ln -s $output_folder/SRR12313442_1.fastq.gz treat_PCR_pos_2.fastq.gz
9  ln -s $output_folder/SRR12313443_1.fastq.gz treat_PCR_pos_2.fastq.gz
10 ln -s $output_folder/SRR12313444_1.fastq.gz treat_PCR_pos_2.fastq.gz

```

Este sistema de enlaces es necesario por dos motivos:

1. En todo momento sabemos qué es la muestra y podemos interpretar los resultados en consecuencia
2. Si alguna muestra presenta algún problema, los enlaces guardan la información de cuál es el identificador original y nos podemos poner en contacto con el proveedor del archivo para averiguar qué ha pasado con ella, dando el id que se ha usado en la generación de los fastq.

Con ello, ahora hay que seguir las instrucciones provistas en el manual de LaTex en la sección 12.1 para el análisis de las muestras, a partir del apartado 12.1.1. Si te quedan dudas, repasa la sección al completo.

### 12.3.2 Análisis de datos RNA-Seq en UPO

En el caso de utilizar un clúster distinto a Picasso, lo primero que tenemos que hacer es asegurarnos de que tenemos disponibles los módulos que llaman a los intérpretes de ciertos lenguajes de programación, como Ruby o R. Luego, procederíamos con la instalación de las herramientas que necesitamos para realizar el análisis.

Primero, como siempre, preparamos nuestro entorno de trabajo. Una vez que tenemos acceso a nuestra cuenta del clúster de la UPO, comenzamos a crear el sistema de carpetas donde vamos a guardar la información correspondiente a nuestro análisis. En primer lugar, crearemos la carpeta software en nuestro directorio raíz (\$HOME, ~/). Para ello, ejecutaremos el comando mkdir:

```

1 > mkdir -p $HOME/software

```

Vamos a poner de ejemplo el mismo análisis de muestras de pacientes con/sin COVID19 confirmada por PCR descrito en la sección 12.3.1. Empezaremos creando la carpeta de proyecto.

```

1 > mkdir -p $HOME/NGS_projects/COVID19

```

Entramos en la carpeta COVID19 y allí creamos dos carpetas nuevas, una para los datos del SRA y otra para los datos crudos:

```

1 cd /NGS_projects/COVID19
2 mkdir data
3 cd data
4 mkdir raw_data sra

```

En la carpeta sra será donde guardaremos los datos de lecturas descargados del SRA y en la carpeta raw\_data irán los archivos .fastq necesarios para realizar el análisis de expresión

diferencial, que se obtendrán a partir de los archivos .sra.

### Instalación de herramientas.

En la carpeta \$HOME/software instalaremos las herramientas que emplearemos para la descarga de datos y posterior procesamiento y análisis. En algunos casos, solo hay que descomprimir los archivos, en otros casos solo clonar o descargar el repositorio (cuando los binarios estén disponibles) La herramienta SRA toolkit nos permitirá el manejo de datos del SRA. Se puede descargar a través de su GitHub (sigue las instrucciones provistas para su descarga).

Algunos scripts deben descargarse del repositorio del laboratorio: sys\_bio\_lab\_scripts, y están disponibles en este enlace: [https://github.com/seoanezonjic/sys\\_bio\\_lab\\_scripts](https://github.com/seoanezonjic/sys_bio_lab_scripts). Una vez descargados, hay que añadir a nuestro path de esta carpeta (en el archivo .bashrc). Para ello, con nano o vim editamos el archivo .bashrc en el home de la cuenta (aunque no exista el archivo, se crea desde cero):

```
1 > vi $HOME/.bashrc
```

Y el contenido del archivo que sea:

```
1 PATH=$HOME/software/sys_bio_lab_scripts:$PATH
2 export PATH
```

Esto hará que todos los scripts dentro de sys\_bio\_lab\_scripts estén disponibles en tu línea de comandos sin tener que poner constantemente el PATH a ellos. Después de guardar el archivo hay que ejecutar el siguiente comando para que sea efectivo el cambio:

```
1 source $HOME/.bashrc
```

Dentro de sys\_bio\_lab\_scripts se deben editar varias cosas del script ncbi\_sra\_file\_downloader.sh para que funcione en la cuenta de la UPO:

- Borrar la línea que empieza por module
- Añadir las líneas (en sustitución de las que incluyen module):

```
1 PATH=$HOME/software/SRA_Toolkit/bin:$PATH
2 export PATH
```

- Cambiar el valor de la variable aspera\_folder a la carpeta donde has bajado el aspera. Sería tal que así:

```
1 aspera_folder=$HOME/software/aspera/connect
```

- Eliminar la línea con el comando mkdir -p \$SCRATCH'/sra\_download'.
- Sustituir en la línea que incluye el contenido: /repository/user/main/public/root = "'\$SCRATCH'/sra\_download'" > \$HOME/.ncbi/user-settings.mkfg la variable "'\$SCRATCH'/sra\_download'" por la carpeta donde se quieren guardar los resultados de los datos sra de cada estudio (en este caso, por "'\$HOME/NGS\_projects/COVID19/sra'"').

**WARNING: Si las descargas fallan:**

Este sistema a veces da fallos en algunos clústeres. En caso de que cuando se vaya a hacer la descarga de los datos se interrumpa o no devuelva resultados, se puede hacer la descarga de los datos del SRA a mano empleando el servicio de descargas y un script de bash que incluya la información correspondiente a la muestra usando el servicio de descarga del sra (<https://sra-pub-src-3.s3.amazonaws.com>) con un comando wget por cada una de las muestras a descargar.

**Búsqueda de datasets.**

Este apartado es exactamente igual que en el caso descrito en la sección 12.3.1. La única diferencia es el cabecero para ejecutar en el sistema de colas del script transform\_sra\_to\_fastq.sh, que quedaría como viene detallado a continuación:

```
1 #! /usr/bin/env bash
2 #SBATCH -J sra_fastq
3 #SBATCH -p day
4 #SBATCH -N 1
5 #SBATCH -c 1
6 output_folder=$HOME/projects/COVID19/data/raw_data
7 mkdir $output_folder
8 sra2fastq.sh sra_ids $SCRATCH/sra_download/sra $output_folder
```

Siguiendo los pasos descritos anteriormente se obtendrá la información necesaria para lanzar el análisis de las muestras como se describe en la sección 12.1, a partir del apartado 12.1.1. Si te quedan dudas, repasa la sección al completo.



## 13. coRmiT workflow

Este flujo de trabajo sirve para llevar a cabo el análisis, comparación, selección e integración de estrategias de cálculo de correlación de pares miRNA-targets (basados en umbrales de anticorrelación). Está incluido dentro de el paquete de Bioconductor ExpHunterSuite.

### 13.1 targets\_templates.af

La plantilla principal en la que está desarrollado este flujo de trabajo es targets\_templates.af. Contiene dos nodos principales:

1. launch\_target\_analysis: Nodo que utiliza el programa coRmiT.R.
2. functional\_analysis: Nodo que utiliza el programa clusters\_to\_enrichment.R

#### 13.1.1 Nodo launch\_target\_analysis

En este nodo se utiliza el programa coRmiT.R, el cual requiere los siguientes parámetros:

- export output — se establece la ruta de salida de los resultados del análisis.
- export Multimir\_path — actualmente se puede elegir entre dos rutas en base al organismo con el que se trabaje. Corresponde a la base de datos con la que se van a cotejar los resultados obtenidos en el análisis de miRNA.
- export RNAseq\_input — se añade la carpeta con los resultados de la comparación a estudiar. Esta carpeta normalmente se encontrará en FSCRATCH y estará dentro de la carpeta DEGenesHunter\_results/comparación\_elegida
- export miRNA\_seq\_input — se añade la carpeta con los resultados obtenidos mediante el análisis miRNA de la comparación establecida anteriormente.
- export strategies — se establecen las estrategias de análisis (Creo que sería conveniente al menos mencionar los códigos de cada una pero no me los sé: EE:Eh:Ed:hd:hE:hh:dh:dE)
- export CORR\_THR — se indican los umbrales de correlación elegidos para el análisis: -0.55:-0.6:-0.65:-0.7:-0.75:-0.8:-0.85:-0.9:-0.95.
- export Organism — se especifica el organismo de estudio, siendo 'hsa' para el caso de humanos o 'mmu' en el caso de ratones.

- export Add\_opt\_corr — se añaden parámetros adicionales para personalizar el análisis, como puede ser –tag\_filter, el cual permite filtrar tanto los genes como los miRNA incluidos en el análisis en base a su análisis diferencial, eligiendo únicamente aquellos que hayan sido etiquetados como prevalentes (prevalent), prevalentes y posibles (all\_possible) o NOT\_DEG que se encuentren en el mismo módulo de correlación que un prevalent/possible (putative). Otro parámetro de filtrado es -u. Este parámetro especifica el número mínimo de bases de datos necesario que predigan una pareja miRNA-target para que esta sea aceptada. Por último, en el parámetro –output\_pairs, entre otros, se puede especificar si se quiere filtrar por pares presentes en la base de datos multiMiR en general (multimir), o por aquellas predichas (predicted) o validadas (validated).

Seguidamente, el programa add\_mirna\_mature.R coge la tabla de resultados del Hunter y le añade los identificadores de miRNA a los códigos MIMAT. Requiere como argumento obligatorio -i, al cual se le proporcionará la tabla de resultados hunter\_results\_table.txt.

```
1 add_mirna_mature.R -i hunter_results_table.txt
```

### 13.1.2 Nodo functional\_analysis

Este nodo realiza el análisis funcional, tal y como indica su nombre, y requiere los siguientes parámetros:

- export F\_organism — se especifica 'Human' o 'Mouse'
- export funsys — se añaden las bases de datos con las que se quiere realizar el análisis funcional. Por defecto: MF:BP:CC:KEGG:Reactome
- export f\_pval — pvalor. Por defecto: 0.1.
- export Add\_opt\_enr — se pueden especificar parámetros adicionales. Por ejemplo, se añade –custom seguido de archivos adicionales para la realización del análisis funcional con más bases de datos.

### 13.1.3 Preparación de paquetes

Para preparar paquetes de resultados, se utiliza la herramienta create\_cormit\_report.sh, la cual recopila los resultados miRNA-diana de coRmiT (**coRmiT\_0000**) y el funcional de las dianas (**clusters\_to\_enrichment.R\_0000**). De igual forma que en RNA, se genera una carpeta .zip con los resultados recopilados y se nombra según el esquema ProyectoResults\_coRmiT\_dd\_mm\_aaaa.zip. Ej: SarcomaEwingResults\_coRmiT\_15\_05\_2024.zip

```
1 zip -r folder.zip path_to_savefolder
```

### 13.1.4 Flujo de trabajo

Para llevar a cabo el flujo de trabajo se debe clonar del repositorio de github:

```
1 git clone https://github.com/JoseCorCab/ExpHunterSuite\_CorMiT\_benchmark.git
```

Se creará una carpeta de flujo de trabajo similar a la que se ha visto anteriormente. En este caso, se vuelve a realizar una copia del archivo config\_daemon\_default con el nombre de config\_daemon y se procede a su modificación, estableciendo los parámetros previamente descritos para ambos nodos.

Una vez establecidos, en primer lugar se comenta el bloque de análisis funcional en templates/-targets\_templates.af y se ejecuta "./daemon.sh 1". Seguidamente, una vez comprobado que el proceso ha ido bien y observado el informe, se comenta el bloque de análisis y se descomenta el bloque funcional y se vuelve a ejecutar "/daemon.sh 1". Por último, para crear el paquete con los archivos necesarios, se ejecuta "./daemon.sh report", el cual utiliza la herramienta create\_cormit\_report.sh descrita previamente.





## 14. Utilidades externas

En este capítulo se describen algunas de las utilidades externas que se emplean en el grupo de trabajo.

### 14.1 Obsidian

SEcción pendiente

### 14.2 Descarga de datos

En esta sección se van a detallar las herramientas existentes para la descarga de archivos FASTQ que pueden proveer distintos grupos colaboradores para llevar a cabo su análisis.

#### 14.2.1 gdrivecl

Para todas aquellas descargas que vengan a partir de archivos almacenados en Google Drive (GD) emplearemos la herramienta gdrivecl. Más información sobre la herramienta disponible en <https://github.com/matthuisman/gdrivecl>.

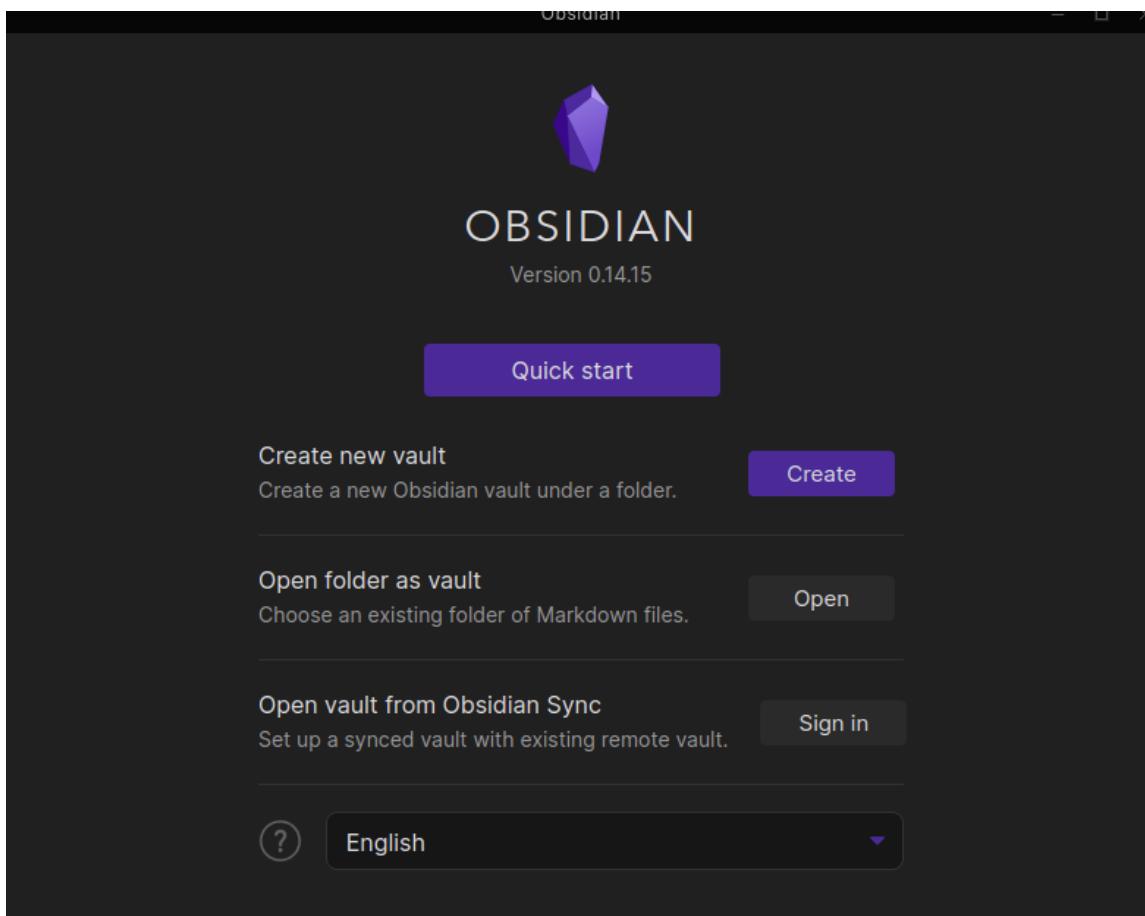
Su carga en Picasso está disponible en los initializes:

```
1 initializes/init\_gdrivecl
```

Un ejemplo de utilización está disponible en /mnt/home/users/pab\_001\_uma/pedro/proyectos/marti\_mit/data/original. En el directorio superior de este ejemplo está la separación de archivos en dos carpetas: original (datos originales) y raw (los que se emplearán en el análisis siguiendo el sistema del Hunter). Si vamos a la carpeta original encontraremos un archivo download.sh con las ejecuciones necesarias que se pueden emplear a modo de ejemplo para descargar archivos de GD:

```
1 #! /usr/bin/env bash
2 . ~soft_bio_267/initializes/init_gdrivecl
3 gdrivecl.py 'https://drive.google.com/folder\_to\_download' -P ./output\_folder
```

Figura 14.1



### 14.2.2 FTP

Para aquellos proyectos, como Meniere o organoides MAGEL2, donde el grupo investigador aporta los FASTQ a través de un servicio de FTP, se debe contar con una información para preparar la descarga, incluyendo la dirección del host, el puerto, el protocolo (FTP o SFTP), el nombre de usuario y la contraseña de acceso.

SFTP es un servicio completamente ajeno a bash. Para echar un primer vistazo de los archivos que nos quieren enviar, debemos introducir el comando:

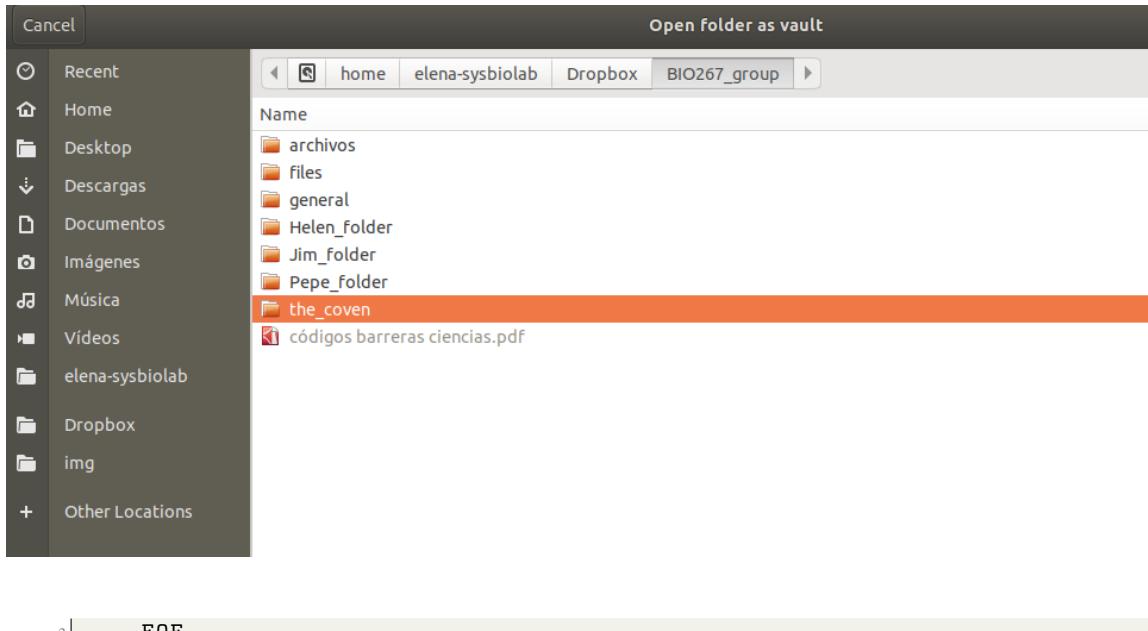
```
1 sftp -P 22 usuario@server.server
```

Esto nos abrirá una terminal SFTP interactiva. Tiene sus comandos específicos, pero `pwd`, `ls` y `cd` son iguales, podemos usarlos para echar un vistazo a los datos y decidir qué queremos descargar y qué no. Sabiendo eso, podemos montar el comando final (usa ‘exit’ para salir de la terminal interactiva).

Al ser el comando ‘`sftp`’ interactivo, para automatizar la descarga tenemos que añadir una estructura especial que permite inyectarle un script. Esto se consigue de la siguiente manera:

```
1 sftp -P 22 usuario@server.server <<EOF
2 reget -r pathToFASTQfiles
```

Figura 14.2



3 EOF

Lo que hay entre "«EOF" y el segundo "EOF" es el script. "reget" es "resume get", el comando para descargar y/o rescatar descargas que se han quedado colgadas, comprobando si tienes ya los archivos que intentas descargar. El flag -r es para descargar recursivamente (siguiendo directorios y subdirectorios), y luego necesita el PATH dentro del servidor sftp a lo que quieras descargar. Ahora debes guardar este script en un sh en la carpeta de proyecto para así saber de dónde vienen los datos.

### 14.2.3 gofile.io

Existe una plataforma de descarga de archivos llamada gofile.io (no confundir con gofile.me) donde los usuarios pueden subir el material que dispongan y nosotros descargarlo con un programa de Python almacenado en la cuenta de software de Picasso:

```
1 ~soft_bio_267/programs/x86\_64/gofile-downloader/gofile-downloader.py
```

Requiere un primer argumento de entrada que es la url del archivo que se quiera descargar, y el initilize para ejecutar este programa se encuentra en:

```
1 ~soft_bio_267/initializes/init_gofile_downloader
```

Un ejemplo de uso sería el siguiente:

```
1 source ~soft_bio_267/initializes/init_gofile_downloader
2 gofile-downloader/gofile-downloader.py url_to_gofile.io
```



**Part III**

**Sección de desarrollo**



En esta sección se explicarán los términos necesarios para agilizar el desarrollo de diversas gemas<sup>1</sup> de Ruby. No obstante, para ello necesitaremos recordar previamente el procedimiento para el desarrollo de una sola gema.

---

<sup>1</sup>nombre usado para referirse a los paquetes en Ruby





## 15. Desarrollo de una gema

Imaginemos que tenemos una gema de interés que está desarrollando el grupo. Por algún motivo, se necesita modificar el código presente en la misma. Para ello, clonamos el repositorio de Github donde se encuentra la gema... ¿Dónde lo clonamos? en una carpeta llamada *dev\_gems* que se deberá haber creado previamente, en el *home*. Es en esta donde se deberían clonar todos los repositorios con las gemas que se estén desarrollando.

Un vez clonado el repositorio, ya podemos comenzar a realizar los cambios, actualizándolos por medio de los comandos de git. Ahora bien ¿y si se necesitan realizar algunas pruebas antes de colgar la nueva versión del código al repositorio? En dicho caso, una primera respuesta podría ser tener una carpeta distinta a la de *dev\_gems*, con la que poder realizar una serie de test manuales, por medio del uso de un pequeño script .sh. No obstante, lo correcto sería que en la propia gema hubiese ya una carpeta donde poder añadir una serie de *testeos* automáticos<sup>1</sup>.

Finalmente, cuando los cambios se hayan realizado, habrá que avisar a la persona correspondiente para actualizar esa gema que está utilizando todo el grupo.

Ahora bien, sabemos el procedimiento general para el desarrollo de una gema. De hecho, esto seguiría siendo válido para el desarrollo de varias gemas independientes a la vez. Sin embargo, podría suceder que estuviésemos desarrollando gemas con algún tipo de dependencia. Por ejemplo, necesitamos desarrollar en *semtools* un método con el que obtener los descriptores estadísticos del número de anotaciones para cada perfil. Para ello, vamos a necesitar utilizar una serie de funciones matemáticas que le corresponden a la gema *exp\_calc*. El problema, esta función aún no está definida, de manera que el desarrollo de ambas gemas deben ir parejos y eso puede resultar una tarea bastante tediosa. Por ese motivo se propone a continuación, una manera de agilizar el proceso.

---

<sup>1</sup>Aquí se hace referencia a ciertas pruebas que podemos realizar con paquetes como *minitest*

## 15.1 Desarrollo de dos o más gemas a la vez

Para disminuir la carga del proceso, vamos a utilizar la gema *bundler*, utilizada para simplificar la definición de dependencias entre las gemas que necesito. A continuación llevaremos a cabo los siguientes pasos.

1- En el repositorio local, buscamos el archivo *Gemfile.lock* que indica la versión y path necesarios para cada dependencia de la gema. Esta información es necesaria y, por ello, si no existe todavía deberá realizarse el siguiente comando:

```
1 bundle install
```

En dicho momento, se debería obtener el archivo *Gemfile.lock* y salir en pantalla una lista de las dependencias usadas o instaladas.

### **WARNING: Cuidado con una lista verde de dependencias**

Si en la terminal se obtiene una lista de dependencias, indicándose en su mayoría que se están instalando, esto no debería ser usual y deberá informarse a la persona correspondiente de que actualice el sistema de gemas del grupo. Pues es posible que estén desactualizadas y, por ese motivo, al realizar el comando anterior, se descarguen las últimas versiones.

Si ya existe un archivo *Gemfile.lock*, sería recomendable borrarlo y volver a realizar el proceso anterior, por si sucede que las gemas necesarias se hubieran actualizado.

### **WARNING:**

No añadir el archivo *Gemfile.lock* al repositorio de Github a la hora de realizar el push

2- Escribimos el siguiente script en la carpeta correspondiente para nuestros tests. En este caso, mostramos el ejemplo para dos gemas. Queremos ejecutar el script *exec.rb* de la gema1, teniendo en cuenta que la versión de la gema2 que vamos a usar también sea la local (la que estamos desarrollando a la par).

```
1 #!/usr/bin/env bash
2 source ~soft_bio_267/initializes/init_ruby
3 bundle config --local local.gema1 ~/dev_gems/gema1
4 bundle config --local local.gema2 ~/dev_gems/gema2
5 bundle config
6 current='pwd'
7 cd ~/dev_gems/gema1
8 bundle exec ruby exec.rb -i $current/file_prueba -o $current/
results.txt
```

### **WARNING: Ajustarse a los comandos presentados en el script anterior**

El comando *bundle exec* debe realizarse dentro del repositorio local de la gema a desarrollar. De lo contrario este procedimiento no funcionará.

Sirvan de ejemplo para cada caso, los archivos de ejecución presentes en los siguientes *paths*:

- Una gema: /mnt/home/users/pab\_001\_uma/pedro/proyectos/pets/pets\_reloaded\_testing/copatReporter/sylientis

- Dos o más gemas: /mnt/home/users/pab\_001\_uma/pedro/proyectos/pets/pets\_reloaded\_testing/copatReporter/ ó /mnt/home/users/bio\_267\_uma/federogc/projects/semtools/statistics\_profiles





## 16. report\_html



## 17. Semtools: Documentación para desarrollo

### 17.1 Introducción

Como ya se ha mencionado en la sección de uso, el semtools es una gema que puede utilizarse como herramienta para tratar ontologías tanto a nivel de términos (obteniendo términos padres e hijos, por ejemplo) como de perfiles (ID-lista de términos). De aquí, se deduce que vamos a requerir como archivos de entrada: Un archivo de ontología .obo (Ontology file) y, o bien un archivo de perfiles (Profile file or PACO) o bien una lista de términos (Terms files, en caso de presentarse como archivo y no desde la línea de comandos).

Así, como el semtools lee la ontología para estudiar términos o perfiles, podríamos resumirlo en "input = Ontología + Perfiles o Términos".

Por un lado, cuando se utilicen **las listas de términos**, estaremos realizando un análisis sobre el propio árbol de la ontología. De tal modo, se pueden realizar las siguientes operaciones:

- Traducir una lista de términos (comando -l)
- Obtener los términos padres o hijos de un conjunto de términos (comando -c + modificadores)
- especificar la ruta que debe asignarse al archivo de ontologías (comando -R)

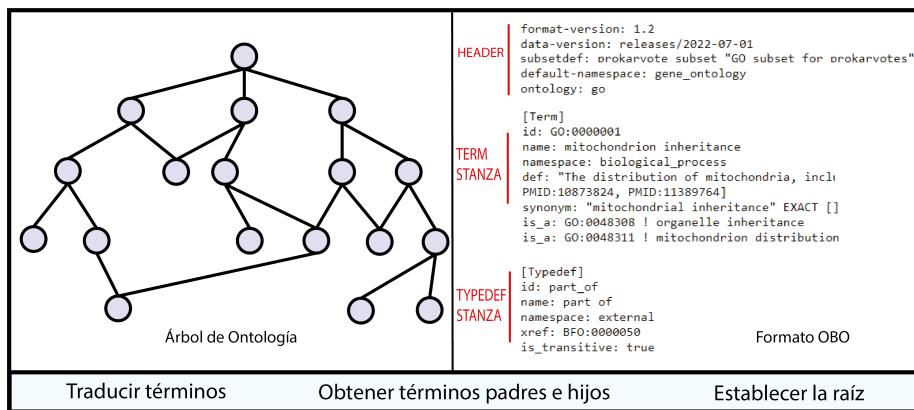


Figura 17.1: Variables y valores de entrada.

Por otro lado, al utilizar un **archivo de perfiles**, las operaciones posibles se enfocarán sobre estos datos, siendo los siguientes:

**(I) Operaciones que alteren los datos de perfiles (ahora llamados operadores modificadores):**

- Limpieza de perfiles (comando -c) incluyendo opcionalmente un término que actúe como filtro (comando -T).
- Expansión de perfiles para ancestros e hijos (comando -e).
- Traducción de los términos de los perfiles (comando -t).

**(II) Operaciones analicen los datos (Operadores analíticos):**

- Obtención del IC de cada perfil (-I).
- Descriptores estadísticos del conjunto de perfiles (-n).
- Cálculo de la similitud semántica entre perfiles (-s).

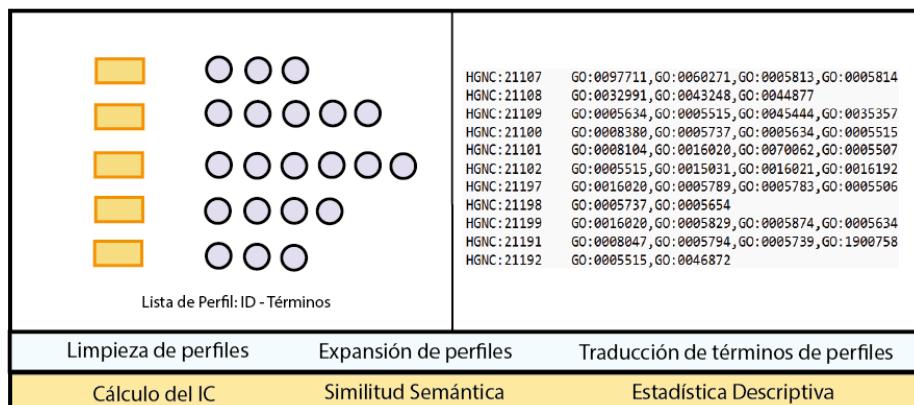


Figura 17.2: Variables y valores de entrada.

### Recordatorio sobre qué es una ontología

Se trata de un conjunto de términos o vocablos conectados entre sí por relaciones (el más conocido, la relación de tipo 'is-a'). Por lo general, la estructura de estas ontologías suelen ser en forma de grafo dirigido acíclico (DAG), o lo que es lo mismo, un árbol con herencias múltiples.

Por lo general, una de las maneras más utilizadas de almacenarlo es en formato OBO, donde encontramos el *header*, los términos y las relaciones. Estos dos últimos definidos en párrafos o unidades de texto, denominado *stanzas*.

Teniendo en cuenta lo anteriormente mencionado. El análisis se va a dividir en dos bloques fundamentales: Lectura de Ontologías y Análisis ontológico (importante ser coherente y constante con los vocablos). No obstante, antes de nada, necesitamos partir de un punto desde donde tirar del hilo. Este comienzo se encuentra en el binario de la gema, que denotaremos por abuso del lenguaje como archivo BIN.

## 17.2 Comienzo del análisis: Pinceladas en la command line

Como se acaba de mencionar, antes de pasar al código presente en la carpeta *lib*, vamos a comenzar desde el script que define la línea de comandos de la gema (en la carpeta *bin*).

Para comprender de una manera metódica el empleo de cada función, se van a seguir el orden de las llamadas para cada tipo de comando<sup>1</sup>. Veamos en vista "cenital" la estructura que tiene el *Main* del script.

En primer lugar, necesitamos descargar los archivos de las ontologías (en formato *obo*). Para ello, podemos indicarlo desde una fuente externa, o bien desde el propio fichero que contiene cada ruta de descarga en función del tipo de ontología. A continuación, se muestra el código comentado en gran detalle para dar una idea de aquello que está sucediendo.

```

1 ## MAIN ##
2 ont_index_file = File.join(EXTERNAL_DATA, 'ontologies.txt') # Path del
   archivo de direcciones de descarga para los obos en funcion a cada
   ontologia.
3
4 if !options[:download].nil? # Comando -d
5   download(ont_index_file, options[:download], options[:output_file]) # 
      <- Desargar la ontologia indicada por options[:download] (GO, HPO,
      MONDO, EFO), en el path indicado por options[:output_file] (si es
      nil, se descarga en external_data).
6   # options[:output_file], comando -o
7   Process.exit
8 end
9
10 if !options[:ontology_file].nil? # Comando -O
11   options[:ontology_file] = get_ontology_file(options[:ontology_file],
12     ont_index_file) # <- Cargar el archivo de ontologia previamente
        descargado en el fichero external_data de la gema, escogiendo entre
        GO, HPO, MONDO o EFO.
12 end

```

<sup>1</sup>Se recomienda encarecidamente que se haya leído previamente el manual de uso para esta gema

Aquí se observan dos funciones, la función *download* y la *get\_ontology\_file*, encargadas de realizar la descarga del obo y almacenar en una variable el path del archivo, respectivamente. En este caso, es aconsejable ver cómo son exactamente estas funciones:

```

1 def download(source, key, output)
2   source_list = load_tabular_file(source).to_h
3   external_data = File.dirname(source)
4   if key == 'list'
5     Dir.glob(File.join(external_data, '*.obo')){|f| puts f} # <- Ante un
6     "list", se despliegan los paths de las ontologías descargadas en
7     la gema
8   else
9     url = source_list[key] # Si el key no es un 'list', busca el url
10    para la ontología correspondiente en el source.
11    if !output.nil? # Con estos condicionales, se asegura que el OBO se
12      descargue por defecto en la gema o donde haya sido indicado.
13      output_path = output
14    else
15      file_name = key + '.obo'
16      if File.writable?(external_data)
17        output_path = File.join(external_data, file_name)
18      else
19        output_path = file_name
20      end
21    end
22  end
23 end

```

```

1 def get_ontology_file(path, source)
2   # Si el archivo existe, dejamos el path como esta
3   # De lo contrario se comprueba si podemos usar la variable path como
4   # un key
5   # que valga GO, HPO, MONDO o EFO.
6   if !File.exists?(path) # Si el archivo existe, dejamos el path como
7     esta
8     ont_index = load_tabular_file(source).to_h
9     if !ont_index[path].nil?
10       path = File.join(File.dirname(source), path + '.obo')
11     else
12       abort("Input ontology file not exists")
13     end
14   end
15   return path
16 end

```

El detalle está en que la función *download* en realidad puede servir también para obtener los *paths* donde la gema tiene almacenada las ontologías y que en *get\_ontology\_file* podemos usar en la variable 'path' una ruta de archivo (si queremos usar una ontología que no esté incluida)

o bien los nombres de las ontologías ya consideradas en la gema (hasta el momento GO, HPO, EFO y MONDO).

Una vez hecho esto, ya podemos pasar a almacenar el árbol de ontologías, indicando el *path* con la variable *options[:ontology\_file]*, que contiene el path al archivo OBO en cuestión.

```

1 ontology = Ontology.new(file: options[:ontology_file], load_file: true)
  # <- Instanciamos un objeto de la clase Ontology, cargando el
  # archivo de la ontología.
2 Ontology.mutate(options[:root], ontology, clone: false) if !options[:root].nil? # <- Modificar la raíz del DAG, en caso de que se pida.

```

En el código de arriba ya puede empezar a intuirse cuál va a ser la *Clase* fundamental en la que va a depender el script *bin*, la clase *Ontology*, de la que ya se utiliza uno de sus métodos de clase para modificar la raíz del DAG (*Ontology.mutate()*). Ahora bien, en ocasiones vamos a querer analizar una lista de perfiles, veamos cómo se cargan:

```

1 #...code...
2 def store_profiles(file, ontology)
3   file.each do |id, terms|
4     ontology.add_profile(id, terms)
5   end
6 end
7 #...code...
8 #MAIN#
9 #...code...
10 if !options[:input_file].nil?
11   data = load_tabular_file(options[:input_file]) # <- Lectura del
12   # archivo, separando por tabs
13   if options[:list_translate].nil? || !options[:keyword].nil?
14     format_tabular_data(data, options[:separator], options[:subject_column], options[:annotations_column]) # <- Se formatea la
15     # variable para colocar las columnas de ID y términos en el orden
16     # adecuado
17     store_profiles(data, ontology) if options[:translate] != 'codes' &&
18     options[:keyword].nil? # <- store_profiles
19   end
20 end

```

Como todo parecía señalar, los perfiles deben ser almacenados en el objeto *ontology* uno a uno, según el método *ontology.add\_profile(id,term)*, que aparece antes del *MAIN*. Así, la lista de perfiles es formateada para poder añadir uno a uno el par id-términos.

### ¿Dónde está el input de los términos?

Por lo general los términos son añadidos desde la propia línea de comandos. El único caso donde es necesario cargar una lista de términos es al utilizar el comando -l para traducirlos, y su carga es la estándar de un archivo de una columna.

Dado que ya se ha conseguido cargar todos los archivos de entrada necesarios, nos quedan por realizar las operaciones o análisis deseadas sobre los mismos. Para conseguir comprender estos hay que adentrarse en la clase 'Ontology' y analizar su código.

Al ser una clase con excesivo número de métodos (en ocasiones ""quizás"" obsoletos) y elevadas dependencias entre ellos, se va a proceder al estudio de una forma reversa. Es decir,

partieramos de cada una de las funciones que podemos realizar desde la línea de comandos del semtools y de ahí iremos indagando especialmente en cada función **relevante o complicada**<sup>2</sup> que aparezca.

Además, la clase está claramente dividida en, al menos, dos fragmentos fundamentales<sup>3</sup>: (I) Lectura de los archivos OBO y (II) Operaciones sobre perfiles y términos. Por esa razón, el análisis se dividirá en ambos fragmentos.

En síntesis la estrategia del análisis son dos: (I) Fragmentar la clase en dos y (II) realizar un estudio reverso<sup>4</sup>

### 17.3 Siguiente parada: La clase Ontology

Antes de ver los métodos, se van a presentar cada una de las variables que constituyen esta clase, junto a su comentario asignado, tal y como aparece en el código.

```

1  # Handled class variables
2  # => @@basic_tags :: hash with main OBO structure tags
3  # => @@allowed_calcs :: hash with allowed ICs and similaritites
4  calcs
5  # => @@symbolizable_ids :: tags which can be symbolized
6  # => @@tags_with_trailing_modifiers :: tags which can include extra
7  info after specific text modifiers
8
9  @@basic_tags = {ancestors: [:is_a], obsolete: :is_obsolete,
10 alternative: [:replaced_by,:consider,:alt_id]}
11 @@allowed_calcs = {ics: [:resnik, :resnik_observed, :seco, :zhou, :sanchez], sims: [:resnik, :lin, :jiang_conrath]}
12 @@symbolizable_ids = [:id, :alt_id, :replaced_by, :consider]
13 @@tags_with_trailing_modifiers = [:is_a, :union_of, :disjoint_from,
14 :relationship, :subsetdef, :synonymtypedef, :property_value]
15 @@multivalue_tags = [:alt_id, :is_a, :subset, :synonym, :xref, :
16 intersection_of, :union_of, :disjoint_from, :relationship, :
17 replaced_by, :consider, :subsetdef, :synonymtypedef, :
18 property_value, :remark]
19 @@symbolizable_ids.concat(@@tags_with_trailing_modifiers)

```

En primer lugar, nos encontramos con las variables de la Clase Ontology, que se encargan de designar y establecer los cálculos disponibles en la clase, así como las etiquetas que pueden ser empleadas. Añadamos algunos comentarios extras a las variables necesarias:

- @@basic\_tags: Utilizada para establecer cuáles son las etiquetas que nos encontraremos a lo largo de la ontología para designar que "T1 tiene de ancestro T2" (:is\_a), que "T1 está obsoleto" (:is\_obsolete), ó "T1 puede referirse con el término T2 (:replaced\_by,:consider,:alt\_id)".
- @@tags\_with\_trailing\_modifiers: En ocasiones, ciertas etiquetas de información pueden ir acompañadas de información extra en el archivo de la ontología (por ejemplo, is\_a: GO:00102010 ! "extracellular location"), esto es algo a tener en cuenta a la hora de procesar la información.

<sup>2</sup>En este caso, lo más óptimo es dar mayor explicación a aquellas funciones un tanto difíciles de comprender a primera vista o de gran relevancia conceptual en el proceso (por ejemplo, el cálculo del IC)

<sup>3</sup>En ocasiones ya se ha comentado que esta clase debería ser fragmentada en al menos dos

<sup>4</sup>El estudio reverso es particularmente útil si lo que se quiere es desarrollar o expandir una determinada funcionalidad de la gema, bastará con que "vaya tirando" del hilo desde la sección del comando específico

- @@multivalue\_tags: En ocasiones, puede suceder que un término pueda llegar a tener varias veces repetido una etiqueta de información. Por ejemplo, un cierto término de la ontología podría tener dos ancestros diferentes (indicado con la etiqueta "is\_a") o podrían haber varias ids alternativas para un sólo término. Sin embargo, otros, como el que indica si el término está o no obsoleto, sólo aparecerá una vez para un término dado.
- En segundo lugar, nos encontramos con las variables de objeto. Que son las siguientes que vemos en el bloque de código:

```

1 # Handled object variables
2 # => @header :: file header (if is available)
3 # => @stanzas :: OBO stanzas {:terms,:typedefs,:instances}
4 # => @ancestors_index :: hash of ancestors per each term handled
5   with any structure relationships
6 # => @descendants_index :: hash of descendants per each term
7   handled with any structure relationships
8 # => @alternatives_index :: has of alternative IDs (include alt_id
9   and obsoletes)
10 # => @obsoletes_index :: hash of obsolesces and it's new ids
11 # => @special_tags :: set of special tags to be expanded (:is_a, :
12   obsolete, :alt_id)
13 # => @structureType :: type of ontology structure depending on
14   ancestors relationship. Allowed: {atomic, sparse, circular,
15   hierarchical}
16 # => @ics :: already calculated ICs for handled terms and IC types
17 # => @meta :: meta_information about handled terms like [ancestors,
18   descendants, struct_freq, observed_freq]
19 # => @max_freqs :: maximum freqs found for structural and observed
20   freqs
21 # => @dicts :: bidirectional dictionaries with three levels <key|
22   value>: I) <tag|hash2>; II) <(:byTerm/:byValue)|hash3>; III)
23   dictionary <k|v>
24 # => @profiles :: set of terms assigned to an ID
25 # => @items :: hash with items relations to terms
26 # => @removable_terms :: array of terms to not be considered
27 # => @term_paths :: metainfo about parental paths of each term

```

En este caso, algunas de las variables más familiares son ics, ancestors\_index, descendants\_index, alternative\_index, obsoletes\_index, ics, profiles, removable\_terms. Sin embargo, hay una que quizás no suene tan familiar, pero que va a parecer en numerosas ocasiones a lo largo del análisis, y es la variable meta, en la que para cada término de la ontología, se especifica sus ancestros, descendientes, frecuencia estructural y frecuencia observada.

**@header:** Contendrá la información mostrada en el header del archivo de la ontología, en formato de diccionario. De modo que, para el caso del header de la figura-z, tendríamos en ruby algo como:

```

1 header = { :"format-version" => "1.2" , :"data-version" =>
2   "releases/2022-07-01", :subsetdef => "prokaryote subset 'Go subset
3   for prokaryotes', :"default-namespace" => "gene_ontology", :
4   ontology => 'go'}

```

**@stanzas:** Diccionario en el que se almacena el resto de la información del archivo de ontología que no es el **header**. Este hash tiene tres *keys* posibles (:terms,:typedefs,:instances) y cada uno de ellos lleva a otro hash. Veamos como ejemplo el caso del término que aparece en

la figura-z, si tratamos de acceder a la información de este archivo, lo haríamos de la siguiente manera:

```
1 term = "GO:0000001"
2 stanzas[:terms][term]
```

De aquí obtendríamos algo como:

```
1 { :id => "GO:0000001" , :name => "mitochondrion inheritance" , :
  namespace => "biological_process" , :def => "The distribution of..." ,
  ... , :is_a => ["GO:0048308" , "GO:0048311"]}
```

De donde se observa que en cada línea sepáramos por ":" y convertimos esto en un par *key-value*. Este almacenamiento es equivalente para los otros dos tipos de hash restantes.

**Las variables acabadas en ".index"**: Todas ellas poseen nombres y comentarios que muestran claramente lo que almacenan. Sin embargo, se va a dar un lista de ejemplos para ver cómo es exactamente esta estructura de los datos:

```
1 # => @ancestors_index
2 # Example :: {"T2"=> ["T3","T1","T7"] , "T67" => ["T33","T1","T69"]
3 "] ,...}
4
4 # => @descendants_index
5 # Example :: {"T1"=> ["T2","T67"] , "T7" => ["T2","T300"] ,...}
6
7 # => @alternatives_index
8 # Example :: {"T1"=> "T1.1" , "T3" => "T3.1",...}
9
10 # => @obsoletes_index
11 # Example :: {"T1"=> "T1.1" , "T7" => "T7.1",...}
12
13 # => @obsoletes
14 # Example :: {"T1"=> true , "T7" => true,...}
```

Como pequeña anotación, mencionar que si un término está obsoleto puede tener o no un id alternativo. Por otro lado, puedes tener un id alternativo sin ser un término obsoleto.

En el proceso, hay unas cuantas variables de objeto adicionales terminadas en ".index" que pueden ser de interés. Por ejemplo, la variable @mica\_index, en la que se indica el mica (Maximum Information Common Ancestor) entre dos términos dados (estructura *Key1*: Term1 => *key2*:Term2 => semantic\_similarity from MICA)

**@special\_tags**: Como puede verse en el siguiente fragmento de código, almacena la información equivalente a la variable @@basic\_tags.

```
1 def initialize(file: nil , load_file: false , removable_terms: [] ,
2 build: true , file_format: nil)
3   # Initialize object variables
4   # ...code...
5   @special_tags = @@basic_tags.clone
6   # ...code...
6 end
```

**@ics**: La estructura de esta variable puede verse según el siguiente ejemplo:

```
1 # Extracting the hash for the :resnik IC:
2 @ics[:resnik]
```

```

3 => { :T1 => 0.7, :T2 => 0.9, :T3 => 1, ... }
4 # Extracting IC from one term and different formulas:
5 @ics[:resnik][:T1]
6 => 0.7
7 @ics[:resnik_observed][:T1]
8 => 0.6
9 @ics[:seco][:T1]
10 => 0.2

```

**@meta:** Variable en la que para cada término de la ontología, se especifica sus ancestros, descendientes, frecuencia estructural y frecuencia observada. La estructura de esta variable puede verse según el siguiente ejemplo:

```

1 # Extracting the meta information from a term:
2 @meta[:T1]
3 => [ 7, 2, 3, 10]

```

**@max\_freqs:** La estructura de esta variable puede verse según el siguiente ejemplo.

```

1 @max_freqs = { :struct_freq => 3, :observed_freq => 7, :max_depth => 9}

```

**@dicts:** Se trata de un "diccionario de diccionarios", donde se almacena información de gran relevancia. La idea es obtener una especie de "diccionario bidireccional", en el que poder buscar en base a términos o valores, según se dé el caso. Por ello, lo primero que debemos saber de @dicts es la siguiente línea de código que la define:

```

1 @dicts[store_tag] = { byTerm: byTerm, byValue: byValue}

```

Es decir, podemos acceder a aquellos diccionarios que utilizan el término como valor de búsqueda, o el valor directamente. Quizás esto último haya resultado un poco confuso, para comprender la idea de su estructura, vayamos con el siguiente ejemplo.

Imaginemos que necesitamos obtener los ancestros directos<sup>5</sup> de un determinado término. En ese caso lo que haríamos sería:

```

1 @dicts[:byTerm][:is_a][T1]
2 => ["T3", "T7"]

```

Por otro lado, también podríamos interesarnos por aquellos términos que poseen un ancestro directo en específico:

```

1 @dicts[:byValue][:is_a][:T7]
2 => ["T1", "T2", "T6", "T55"]

```

Así, de este modo, @dicts es la variable empleada para almacenar información *key-value* que sea interesante en manipular o acceder de manera bidireccional. Por ejemplo, los dos casos en los que esta variable es utilizada son para obtener: los ancestros directos (con ':is-a') y los niveles de la ontología (con ':level').

**@profiles:** Se trata de un diccionario cuyos keys son los IDs de los perfiles y los values un array de los términos asociados a el perfil.

**@items:** Consiste en un hash con key los términos de la ontología y value, valores como el ID del perfil al que está asociado (que puede ser uno o varios, en forma de array).

**@removable\_terms:** Un array con los términos de la ontología que no deben ser considerados.

<sup>5</sup>Aquellos que están asociados directamente, sin ancestros intermedios

**@term\_paths:** La estructura de esta variable puede verse según el siguiente ejemplo. Básicamente, aporta la información relevante sobre todas las rutas posibles hasta llegar a la raíz de la ontología.

```

1 # => @term_paths :: metainfo about parental paths of each term
2
3 @term_paths = { :T1 => { total_paths: 33, largest_path: 15, shortest_path
    : 7, paths: [] }, :T2 => { total_paths: 12, largest_path: 23,
    shortest_path: 3, paths: [] }, ... }

```

### 17.3.1 Lectura de los archivos OBO

Antes de nada, necesitamos saber cómo se lleva a cabo el proceso de lectura de los archivos de ontología. Semtools acepta dos tipos de formato, el formato OBO y el formato JSON. En ambos casos podemos ver el sistema como un conjunto de bloques en el que se almacena la información de un término (*term*), de un tipo de conexión o relación (*typedef*) o, en pocos casos, para para especificar ciertas particularidades y permitir compatibilizar formatos (*instance*). Cada bloque recibe el nombre de *stanza* o "estrofa" por su traducción al español<sup>6</sup>. Todas las estrofas vienen "escritas" en el cuerpo del archivo y está precedido por un encabezado o *header* con toda la información general de la ontología (quizás pueda ayudar la imagen 17.1). La idea sería la siguiente: "Leamos cada línea del documento, almacenando la información de un hash *header* y uno *stanzas*" -> "Para cada tipo de stanzas tendremos una serie de subdiccionarios" -> "En cada subdiccionario se almacena asocia un ID a un conjunto de *key-value*". Así, conseguimos cargar y organizar cada bloque del archivo, en forma de diccionario. Con esta idea en mente, pasemos a ver un poco de código:

```

379 ontology = Ontology.new(file: options[:ontology_file], load_file: true)

86     if load_file
87         fformat = file_format
88         fformat = File.extname(file) if fformat.nil? && !file.nil?
89         if fformat == :obo || fformat == ".obo"
90             load(file, build: build)
91         elsif fformat == :json || fformat == ".json"
92             self.read(file, build: build)
93         elsif !fformat.nil?
94             warn 'Format not allowed. Loading process will not be performed'
95         end
96     end

```

Figura 17.3

<sup>6</sup>La ontología es un conjunto de términos y relaciones que vienen definidos por bloques

**Lectura para formato OBO:**

7.

En el interior del método *initialize* se llama al método *load*. Este se encarga de llamar a los distintos métodos involucrados en el proceso de lectura del archivo OBO y construcción de las variables @header y @stanzas, entre otras.

```

1013     def load(file, build: true)
1014         _, header, stanzas = self.class.load_obo(file) → 237 def self.load_obo(file) #TODO: Send to obo_parser class
1015         @header = header → 238 raise("File is not defined") if file.nil?
1016         @stanzas = stanzas → 239 # Data variables
1017         self.remove_removable() → 240 header = ''
1018         # @removable_terms.each{|removableID| @stanzas[→ 241 stanzas = {terms: {}, typedefs: {}, instances: {}}
1019             self.build_index() if build → 242 # Auxiliar variables
1020         end → 243 infoType = 'Header'
1021         currInfo = [] → 244 curvInfo = []
1022         stanzas_flags = %w[[Term] [Typedef] [Instance]] → 245 # Read file
1023         → 246
1024
1025         File.open(file).each do |line|
1026             line.chomp!
1027             next if line.empty?
1028             fields = line.split(':', 2)
1029             # Check if new instance is found
1030             if stanzas_flags.include?(line)
1031                 header = self.process_entity(header, infoType, stanzas, currInfo) ← 250
1032                 info = self.info2hash(currInfo) → 251
1033                 # Store current info
1034                 if infoType.eql?('Header')
1035                     header = info → 252
1036                 else
1037                     id = info[:id]
1038                     case infoType
1039                     when 'Term' → 253
1040                         stanzas[:terms][id] = info → 254
1041                     when 'Typedef' → 255
1042                         stanzas[:typedefs][id] = info → 256
1043                     when 'Instance' → 257
1044                         stanzas[:instances][id] = info → 258
1045                     end → 259
1046                 end → 260
1047                 # Concat info
1048                 currInfo << fields → 261
1049             end → 262
1050             # Store last loaded info
1051             header = self.process_entity(header, infoType, stanzas, currInfo) if !currInfo.empty? → 263
1052             → 264
1053             # Prepare to return
1054             finfo = {file => file, :name => File.basename(file, File.extname(file))} → 265
1055             return finfo, header, stanzas → 266
1056         end → 267
1057     end → 268

```

Figura 17.4

Como puede verse en la imagen superior 17.4, lo primero es llamar al método de clase *load\_obo* para poder parsear el archivo según su formato. Entre las líneas 247 y 261 de la imagen puede verse el núcleo de la operación que sería algo como "acumula la información de cada línea en el id asignado hasta llegar a un nuevo stanzas". Como puede seguirse en *process\_entity* la información puede ser el *header* o bien distintos tipos de *stanzas*, a saber: *Term*, *Typedef* y *Instance*, todos ellos ya mencionados. Sigamos leyendo hasta llegar al método *remove\_removable* y *build\_index*...

<sup>7</sup>Parte del análisis da por sentado el conocimiento de algunos conceptos básicos sobre el formato OBO que ya se han ido explicando ligeramente a lo largo del documento. Si se desea una guía más profunda se recomienda OBO format

```

1013     def load(file, build: true)
1014         _, header, stanzas = self.class.load_obo(file)
1015         @header = header
1016         @stanzas = stanzas
1017         self.remove_removable()                                     1023 → def remove_removable()
1018         # @removable_terms.each{|removableID| @stanzas[ 1024       @removable_terms.each{|removableID| @stanzas[:terms].delete(removableID) if !@removable_terms.empty?
1019         self.build_index() if build                                1025       end
1020     end

1021
1022     def build_index()
1023         self.get_index_obsoletes      Completa la variable @obsoletes, y parte de @alternative_index y @obsolete_index
1024         self.get_index_alternatives   Completa la variable @alternative_index
1025         self.get_index_child_parent_relations
1026             @alternatives_index.each{|k,v| @alternatives_index[k] = self.extract_id(v)}
1027             ## @alternatives_index.map {|k,v| @alternatives_index[k] = self.stanzas[:terms][v][:id] if k == @alternatives_index.compact!
1028             @obsoletes_index.each{|k,v| @obsoletes_index[k] = self.extract_id(v)}
1029             @obsoletes_index.compact!
1030             @ancestors_index.each{|k,v| @ancestors_index[k] = v.map{|t| self.extract_id(t)}.compact}
1031             @ancestors_index.compact!
1032             @descendants_index.each{|k,v| @descendants_index[k] = v.map{|t| self.extract_id(t)}.compact}
1033             @descendants_index.compact!
1034         self.get_index_frequencies()  Obtiene los datos de número de ancestros, descendientes y frecuencia estructural de la variable @meta y actualiza el @max_freqs
1035         self.calc_dictionary(:name)  Genera el diccionario bidireccional con la etiqueta del stanzas :name
1036         self.calc_dictionary(:synonym, select_regex: /\^(.*\^\^)/) Genera el diccionario bidireccional con la etiqueta del stanzas :synonym
1037         self.calc_term_levels(calc_paths: true) Para el APÉNDICE: Rellena la variable @term_paths
1038     end

```

1173     def extract\_id(text, splitBy: ' ')
1174         if self.exists?(text)
1175             return text
1176         else
1177             splittedText = text.to\_s.split(splitBy).first.to\_sym
1178             return self.exists?(splittedText) ? splittedText : nil
1179         end
1180     end

Si el término existe o está al principio del string, nos lo quedamos, de lo contrario devuelve un nil

Figura 17.5

...Efectivamente, `remove_removable` se encargará de eliminar aquellos términos que hayamos asignado en la variable `@removable_terms`. Lo cierto es que este método no tiene mucho misterio, pasemos al de `build_index`. En este caso, se ve que se llegan a almacenar la información relevante en las variables `@obsoletes`, `@alternative_index`, `@obsolete_index`, `@meta`, `@max_freqs`, `@term_paths` y `@dicts` (para el diccionario de stanzas `:name` y `:synonym`). No obstante, no se ve el procedimiento exacto por el que se realiza. Por este motivo, se pasa a mostrar dos de los métodos que podrían causar más dificultad a la hora de comprender el funcionamiento: `get_index_child_parent_relations` y `get_index_frequencies`.

### `get_index_child_parent_relations`

EL objetivo del proceso es claro, conseguir los ancestros y descendientes directos e indirectos para cada término, así como almacenar el tipo de estructura que posee la ontología. Para ello, primero se obtienen los ancestros (parentales) directos e indirectos gracias, en última instancia, al método recursivo `get_related_ids()`, en el que se almacenan los ancestros totales para un término dado. Un procedimiento que se lleva a cabo para cada término de la ontología. En segundo lugar, añadimos para cada término los descendientes directos indirectos, empleando el hash `parentals` a la inversa. Finalmente, durante todo el procedimiento se ha ido actualizando el tipo de estructura de la ontología, pudiendo ser: `:herierarchical`, `:circular` o `:atomic`.

```

748 def get_index_child_parent_relations(tag: @@basic_tags[:ancestors][0])
749   # Check
750   if @stanzas[:terms].nil?
751     warn('stanzas terms empty')
752   else
753     # Expand
754     structType, parentals = self.class.get_related_ids_by_tag(terms: @stanzas[:terms],
755       target_tag: tag,
756       alt_ids: @alternatives_index,
757       obsoletes: @obsoletes_index.length,
758       reroot: @reroot)
759
760     # Check
761     raise('Error expanding parentals') if (structType.nil?) || parentals.nil?
762
763     # Prepare ancestors structure
764     anc = {}
765     des = {}
766
767     parentals.each do |id, parents|
768       parents = parents - @removable_terms
769       anc[id] = parents
770
771       parents.each do |anc_id| # Add descendants
772         if des.include?(anc_id) # Eliminamos parentales con términos a
773           des[anc_id] = [id]    # eliminar y añadimos la asociación término-
774         else                  # descendientes
775           des[anc_id] << id
776         end
777       end
778
779     # Store alternatives
780     # @alternatives_index.each do |id,alt|
781     #   anc[id] = anc[alt] if anc.include?(alt)
782     #   des[id] = des[alt] if des.include?(alt)
783     # end
784
785     # Check structure
786     if !{:atomic,:sparse}.include? structType
787       structType = structType == :circular ? :circular : :hierarchical
788     end
789
790     # Store
791     @ancestors_index = anc
792     @descendants_index = des
793     @structureType = structType
794
795   end
796
797   # Check
798   if !{:atomic,:sparse}.include? structType
799     structType = structType == :circular ? :circular : :hierarchical
800   end
801
802   # Define structure type
803   structType = :hierarchical
804
805   related_ids = {}
806
807   terms.each do |id, tags|
808     # Check if target tag is defined
809     if !tags[target_tag].nil?
810       # Obtain related terms
811       set_structure, _ = self.get_related_ids(id, terms, target_tag, related_ids, alt_ids)
812
813       # Check structure
814       structType = :circular if set_structure == :circular
815
816     end
817
818   end
819
820   # Check special case
821   structType = :atomic if related_ids.length <= 0
822   structType = :sparse if reroot || (related_ids.length > 0 && ((terms.length - related_ids.length - obsoletes) >= 2))
823
824   # Return type and hash with related_ids
825   return structType, related_ids
826
827 end

```

Obtenemos el tipo de estructura de la ontología y todos los parentales de cada término (directos e indirectos)

eliminamos parentales con términos a eliminar y añadimos la asociación término-descendientes

Función encargada de devolver el tipo de estructura que siguen las relaciones de este término y obtener todas las relaciones, colocándolas en la variable related\_ids por medio de un método de recursión

Figura 17.6

### get\_index\_frequencies

Aquí dos de los aspectos más relevantes son: (I) Asegurarnos de que todos los términos sinónimos o alternativos posean la misma información y (II) Al realizar el conteo de descendientes y ascendentes, no se contarán por duplicado aquellos términos que sean equivalentes. Finalmente, fijémonos que las cuentas sobre la frecuencia se realiza sobre los datos estructurales... ¿A qué se refiere? Son aquellos datos obtenidos desde la propia estructura de la ontología, sin añadir los datos de perfiles gen-términos. Finalmente, se actualiza el máximo de las frecuencias y el denominado :max\_depth, que hace referencia al número de descendientes máximos que posee un término.

```

675     def get_index_frequencies()
676         # Check
677         if @ancestors_index.empty?
678             warn('ancestors_index object is empty')
679         else
680             # Per each term, add frequencies
681             @stanzas[:terms].each do |id, tags|
682                 if @alternatives_index.include?(id) La información de @meta para términos equivalentes debe ser la misma
683                     alt_id = @alternatives_index[id]
684                     query = @meta[alt_id] # Check if exist
685                     if query.nil?
686                         query = {ancestors: 0.0, descendants: 0.0, struct_freq: 0.0, observed_freq: 0.0}
687                         @meta[alt_id] = query
688                     end
689                     @meta[id] = query
690                     # Note: alternative terms do not increase structural frequencies
691                 else # Official term
692                     query = @meta[id] # Check if exist
693                     if query.nil?
694                         query = {ancestors: 0.0, descendants: 0.0, struct_freq: 0.0, observed_freq: 0.0}
695                         @meta[id] = query
696                     end
697                     # Store metadata
698                     query[:ancestors] = @ancestors_index[id].count{|anc| !@alternatives_index.include?(anc)}.to_f : 0.0 ??????????
699                     query[:descendants] = @descendants_index[id].count{|desc| !@alternatives_index.include?(desc)}.to_f : 0.0
700                     query[:struct_freq] = query[:descendants] + 1.0 Definición de la frecuencia estructural
701                     # Update maximums
702                     @max_freqs[:struct_freq] = query[:struct_freq] if @max_freqs[:struct_freq] < query[:struct_freq]
703                     @max_freqs[:max_depth] = query[:descendants] if @max_freqs[:max_depth] < query[:descendants]
704                 end
705             end
706         end
707     end

```

OJO: Tanto el número de ancestros, descendientes y las frecuencias comienzan en 0!

No se tienen en cuenta los términos alternativos al contar

???????????

Figura 17.7

**Lectura para formato JSON:**

En esta ocasión el procedimiento es bastante más *straight-forward* ya que el JSON es un formato de texto sencillo para el intercambio de datos con una estructura semejante a un *hash*, por lo que el parseo (análisis sintáctico), resulta mucho más sencillo. Como puede verse en la imagen de abajo 17.8, se trata de ir traduciendo un diccionario en las distintas variables de tipo *hash* que hemos definido en ruby. Realmente, la principal diferencia se encuentra en el método sustraído de la clase *JSON*, denominado *parse*. Sin embargo, finalmente se vuelve a ejecutar el método *build\_index* de la misma forma que se hacía con el formato OBO.

```

def read(file, build: true)
  # Read file
  jsonFile = File.open(file)
  jsonInfo = JSON.parse(jsonFile.read, :symbolize_names => true)
  # Pre-process (Symbolize some hash values)
  if !jsonInfo[:header].nil?
    aux = jsonInfo[:header].map do |entry,info|
      if info.kind_of?(Array) && @symbolizable_ids.include?(entry)
        [entry,info.map{|item| item.to_sym}]
      else
        [entry,info]
      end
    end
    jsonInfo[:header] = aux.to_h
  end
  jsonInfo[:stanzas].map{|id,info| self.class.symbolize_ids(info)} # STANZAS
  jsonInfo[:stanzas].map{|id,info| self.class.symbolize_ids(info)}
  jsonInfo[:stanzas].map{|id,info| self.class.symbolize_ids(info)}
  # Optional
  jsonInfo[:alternatives_index] = jsonInfo[:alternatives_index].map{|id,value| [id,value.to_sym]}.to_h unless jsonInfo[:alternati
  jsonInfo[:ancestors_index].map{|id,family_arr| family_arr.map{|item| item.to_sym}} unless jsonInfo[:ancestors_index].nil?
  jsonInfo[:descendants_index].map{|id,family_arr| family_arr.map{|item| item.to_sym}} unless jsonInfo[:descendants_index].nil?
  jsonInfo[:obsoletes_index] = jsonInfo[:obsoletes_index].map{|id,value| [id,value.to_sym]}.to_h unless jsonInfo[:obsoletes_index]

#...code...
# Store info
@header = jsonInfo[:header]
@stanzas = jsonInfo[:stanzas]
@ancestors_index = jsonInfo[:ancestors_index]
@descendants_index = jsonInfo[:descendants_index]
@alternatives_index = jsonInfo[:alternatives_index]
@obsoletes_index = jsonInfo[:obsoletes_index]
jsonInfo[:structureType] = jsonInfo[:structureType].to_sym unless jsonInfo[:structureType].nil?
@structureType = jsonInfo[:structureType]
@ics = jsonInfo[:ics]
@meta = jsonInfo[:meta]
@special_tags = jsonInfo[:special_tags]
@max_freq = jsonInfo[:max_freqs]
@dicts = jsonInfo[:dicts]
@profiles = jsonInfo[:profiles]
@items = jsonInfo[:items]
@removable_terms = jsonInfo[:removable_terms]
@term_paths = jsonInfo[:term_paths]

self.build_index() if build
end

```

Figura 17.8: Variables y valores de entrada.

### 17.3.2 Operaciones sobre perfiles y términos

Una vez hecho el estudio sobre la carga de archivos de ontología. Ya es posible tener una idea sobre las principales variables y su estructura. Este conocimiento será utilizado para abordar esta nueva sección, sobre las operaciones sobre términos y perfiles.

#### Atención. No todos los métodos incluidos

Antes de nada, es importante mencionar que este análisis se basa en los métodos de objeto y clase llamados para el **archivo bin** del *semtools*. Sin embargo, también habría que tener en cuenta que existen métodos que este script no utiliza y que puede ser empleadas individualmente, como en el caso del *PETS*. No obstante, se utilizan estos métodos para poder visualizar un cierto esqueleto de la gema.

A continuación, sigamos la lectura partiendo de las operaciones llevadas a cabo para un conjunto de términos deseados. Atención a estos primeros métodos pues, en la sección de operaciones sobre perfiles, serán utilizados de nuevo. Este orden nos permite ver la gema de "abajo->arriba".

#### Operaciones sobre términos

Se recuerda que las distintas operaciones observadas desde el *bin* son:

- Traducir una lista de términos (comando -l)
- Obtener los términos padres o hijos de un conjunto de términos (comando -C + modificadores)
- especificar la raíz que debe asignarse al archivo de ontologías (comando -R)

### Traducir una lista de términos (Comando -l)

Una de las primeras operaciones que podríamos llegar a necesitar realizar a un conjunto de término, sería su traducción pero... ¿A qué nos referimos por traducción? Un término puede aparecer según un código de la ontología (GO:0000001) o bien por medio de su nombre asignado (*mitochondrion inheritance*). Veamo cómo se plantea este problema desde el código del *semtools*.

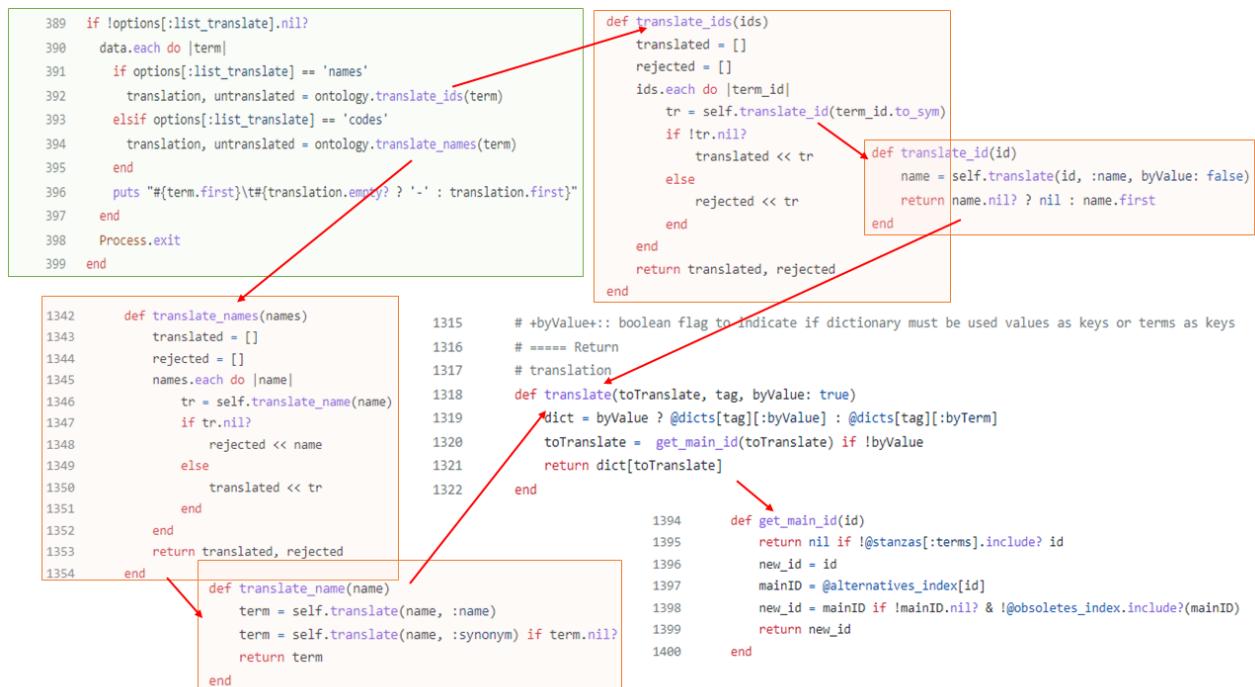


Figura 17.9

Según el esquema mostrado arriba 17.9, se llaman a dos métodos en función del sentido de la traducción, de códigos->nombres tenemos `translate_ids` y de nombres->código tenemos `translate_names`. En ambos casos se acaba dependiendo de la función<sup>8</sup> `translate`. En esta, se toma el subhash presente en la variable `@dicts` (por medio del `tag :name`) indicando el sentido en que se desea traducir (indicado con el parámetro `byValue`) y sustrayendo la información para este código o nombre.

#### get\_main\_id()

Este método puede llegar a ser utilizado con recurrencia en diversas ocasiones, dado que su función es de carácter general. Con este código se pretende obtener el id principal del término en cuestión... "Si tiene un ID alternativo y este no está obsoleto, lo consideraremos el ID principal".

<sup>8</sup>En realidad, lo correcto sería denominarla método de instancia

### Obtener los términos padres o hijos de un conjunto de términos (comando -C + modificadores)

Con el sistema de traducción comprendido, pasamos ahora a centrarnos en la obtención de los términos padres e hijos para un conjunto de términos determinado.

Recordando un poco la sección de usuario del grimorio, recordamos que podemos añadir un conjunto de términos desde la terminal, separándolos por comas. Además, pueden incorporarse una serie de "modificadores" con el fin de especificar exactamente qué se desea realizar con los términos. Ello se indicaba con un "" previamente a los demás términos, quedando algo como (-C "arn/term1,term2,term3,..."). Con estos *terms* y *modifiers* que podemos ver en el código de abajo 17.10, se llama a la función *get\_childs(ontology, terms, modifiers)*.

La función *get\_childs* posee diversos condicionales para cada modificador (*modifier*). Con el fin de hacer más comprensivo el estudio, vamos comenzar por centrarnos en la opción 'a', con la que indicar si se desea obtener los ancestros (método, *get\_ancestors*) o los descendientes (método, *get\_descendants*). Finalmente, tanto para obtener los ancestros como los descendientes, se llama al método *get\_familiar()*, indicando cada caso con el parámetro booleano *return\_ancestors*. Como puede verse, esta función tiene como principal cometido captar la información almacenada por las variables *@ancestors\_index* y *@descendants\_index*.

```

323 options[:childs] = [[], '']
324 opts.on("-C STRING", "-cchilds STRING", "Term code list (comma separated) to generate child list") do |item|
325   if item.include?('/')
326     modifiers, terms = item.split('/')
327   else
328     modifiers = ''
329     terms = item
330   end
331   terms = terms.split(',').map{|t| t.to_sym}
332   options[:childs] = [terms, modifiers]
333 end

454 if options[:childs].first.empty?
455   terms, modifiers = options[:childs]
456   all_childs = get_childs(ontology, terms, modifiers)
457   all_childs.each do |ac|
458     if modifiers.include?('r')
459       puts ac.join("\t")
460     else
461       puts ac
462     end
463   end
464 end

169 def get_childs(ontology, terms, modifiers)
170   #modifiers
171   # - a: get ancestors instead of descendants
172   # - r: get parent-child relations instead of list descendants/ancestors
173   # - hn: when list of relations, it is limited to N hops from given term
174   # - n: give terms names instead of term codes
175   all_childs = []

176   terms.each do |term|
177     if modifiers.include?('a')
178       childs = ontology.get_ancestors(term)
179     else
180       childs = ontology.get_descendants(term)
181     end
182     all_childs = all_childs | childs
183   end

800 def get_ancestors(term, filter_alternatives = false)
801   return self.get_familiar(term, true, filter_alternatives)
802 end

811 def get_descendants(term, filter_alternatives = false)
812   return self.get_familiar(term, false, filter_alternatives)
813 end

823 def get_familiar(term, return_ancestors = true, filter_alternatives = false)
824   # Find into parentals
825   familiars = return_ancestors ? @ancestors_index[term] : @descendants_index[term]
826   if !familiars.nil?
827     familiars = familiars.clone
828     if filter_alternatives
829       familiars.reject!{|fm| @alternatives_index.include?(fm)}
830     end
831   else
832     familiars = []
833   end
834   return familiars
835 end

```

Figura 17.10

Si observamos un poco más en detalle la función del bin anterior, veremos otros métodos que utiliza de la clase *Ontology* como son: *get\_direct\_ancestors*, *get\_direct\_descendants*, y *get\_terms\_levels*. Para las dos primeras basta recordar la variable *@dicts[:is\_a]* donde se almacenaban los datos de las relaciones directas entre los distintos términos. Para el caso del último de los métodos mencionados, el código se puede comprender rápidamente al ver cómo se usa en el método *get\_term\_level* la variable *@dicts[:level][:byValue]* para obtener la

información del nivel de la ontología en el que se encuentra un determinado término.

```

2324  def get_direct_ancestors(term, remove_alternatives: false)
2325    return self.get_direct_related(term, :ancestor, remove_alternatives)
2326  end
2327
2328  def get_direct_descendants(term, remove_alternatives: false)
2329    return self.get_direct_related(term, :descendant, remove_alternatives)
2330  end
2331
2332  def get_direct_related(term, relation, remove_alternatives: false)
2333    if @dicts[:is_a].nil?
2334      warn("Hierarchy dictionary is not already calculated. Returning nil")
2335      return nil
2336    end
2337
2338    target = nil
2339    case relation
2340      when :ancestor
2341        target = :byTerm
2342      when :descendant
2343        target = :byValue
2344      else
2345        warn("Relation type not allowed. Returning nil")
2346    end
2347
2348    return nil if target.nil?
2349    query = @dicts[:is_a][target][term]
2350    return query if query.nil?
2351    query = remove_alternatives_from_profile(query) if remove_alternatives
2352    return query
2353  end
2354
2355  def remove_alternatives_from_profile(profile)
2356    alternatives = profile.select{|term| @alternatives_index.include?(term)}
2357    redundant = alternatives.select{|alt_id| profile.include?(@alternatives_index[alt_id])}
2358    return profile - redundant, redundant
2359  end
2360
2361  def sort_terms_by_levels(terms, modifiers, ontology, all_childs)
2362    term_levels = ontology.get_terms_levels(all_childs)
2363    if modifiers.include?('a')
2364      term_levels.sort!{|t1,t2| t2[1] <=> t1[1]}
2365    else
2366      term_levels.sort!{|t1,t2| t1[1] <=> t2[1]}
2367    end
2368    all_childs = term_levels.map{|t| t.first}
2369    return all_childs, term_levels
2370  end
2371
2372  def get_terms_levels(terms)
2373    termsAndLevels = []
2374    terms.each do |term|
2375      termsAndLevels << [term, get_term_level(term)]
2376    end
2377    return termsAndLevels
2378  end
2379
2380  def get_term_level(term)
2381    return @dicts[:level][:byValue][term]
2382  end

```

Figura 17.11

#### Especificar la raíz que debe asignarse al archivo de ontologías (comando -R)

En este caso, el método utilizado para asignar una nueva raíz a la ontología parece relativamente sencillo 17.12. Basta con tomar todos los descendientes de un determinado nodo (al que queremos asignar como nueva raíz) y, tras ello, quedarnos sólo con los términos y relaciones que están dentro de dicho conjunto.

```

379 ontology = Ontology.new(file: options[:ontology_file], load_file: true)
380 Ontology.mutate(options[:root], ontology, clone: false) if !options[:root].nil?

def self.mutate(root, ontology, clone: true, remove_up: true)
  ontology = ontology.clone if clone
  # Obtain affected IDs
  descendants = ontology.descendants_index[root] 28      # => @descendants_index :: hash of descendants per each term handled with any structure relationships
  descendants <> root # Store itself to do not remove it
  # Remove unnecessary terms
  terms = ontology.stanzas[:terms].select{|id,v| remove_up ? descendants.include?(id) : !descendants.include?(id)}
  ids = terms.keys
  terms.each do |id, term|
    term[:is_a] = term[:is_a] & ids # Clean parental relations to keep only those that exist between selected terms
  end
  ontology.stanzas[:terms] = terms
  ontology.ics = Hash[@@allowed_calcs[:ics].map{|ictype| [ictype, {}]}}
  ontology.max_freqs = {struct_freq => -1.0, observed_freq => -1.0, max_depth => -1.0}
  ontology.dicts = {}
  ontology.removable_terms = []
  ontology.term_paths = {}
  ontology.reroot = true

  ontology.ancestors_index = {}
  ontology.descendants_index = {}
  ontology.alternatives_index = {}
  ontology.obsoletes_index = {}
  ontology.meta = {}
  ontology.profiles = {}
  ontology.items = {}

# Recalculate metadata
ontology.build_index
ontology.add_observed_terms_from_profiles
# Finish
return ontology
end

```

↓

Nos quedamos con los términos parentales de la raíz o sus descendientes

Nos quedamos con las relaciones que mantengan el conjunto de términos anterior

Volvemos a cargar las variables y a construir la ontología con build index, partiendo de la variable @stanzas

Figura 17.12

### Operaciones sobre perfiles

Habiendo ya visto ciertas operaciones que pueden realizarse con un conjunto de términos dado, pasemos a las operaciones sobre perfiles. Es decir, sobre aquellas listas constituidas por un par ID-Conjunto de términos. Por ejemplo, un tipo de archivo de este tipo podría ser:

1 HGNC :21197	MONDO :0012866 , MONDO :0017999
2 HGNC :21143	MONDO :0000133 , MONDO :0013553
3 HGNC :21144	MONDO :0011142 , MONDO :0014236
4 HGNC :21176	MONDO :0013969
5 HGNC :21157	MONDO :0018053 , MONDO :0014619
6 HGNC :21158	MONDO :0013617
7 HGNC :21150	MONDO :0014553
8 HGNC :21219	MONDO :0009696 , MONDO :0012980
9 HGNC :21205	MONDO :0020846
10 HGNC :21208	MONDO :0054636
11 HGNC :21253	MONDO :0019588 , MONDO :0012460
12 HGNC :21237	MONDO :0015448 , MONDO :0014356
13 HGNC :21244	MONDO :0021548 , MONDO :0008925 , MONDO :0018610
14 HGNC :21246	MONDO :0014011 , MONDO :0019306
15 HGNC :21296	MONDO :0019588 , MONDO :0033198

Siendo los códigos con prefijo HGNC, aquellos referentes a un gen, y los que comienzan por MONDO, para los términos de la ontología de enfermedades.

Antes de analizar<sup>9</sup> los perfiles o tratar de sustraer algún tipo de información de estos, puede que se necesite hacer alguna limpieza, traducción o modificación de los perfiles, veámoslo:

**(I) Operaciones que alteren los datos de perfiles (ahora llamados operadores modificadores):**

Recordando un poco teníamos:

- Limpieza de perfiles (comando -c) incluyendo opcionalmente un término que actúe como filtro (comando -T).
- Expansión de perfiles para ancestros e hijos (comando -e).
- Traducción de los términos de los perfiles (comando -t).

### Limpieza de perfiles (comando -c y comando -T)

En primer lugar, desde el bin se llama a un total de 3 funciones encadenadas<sup>10</sup>. En primer lugar, la función *clean\_profiles()* se encargará de mandar a limpiar uno a uno cada perfil de entrada, almacenando en una lista aparte aquellos perfiles que han sido eliminados durante la limpieza (porque se han eliminado todos sus términos). La limpieza de cada perfil individualmente se orquesta desde la función *clean\_profile()*, donde podemos ver que llama al método *clean\_profile\_hard()* y de usar el término dado desde el comando -T para seleccionar aquellos que términos descendientes del mismo.

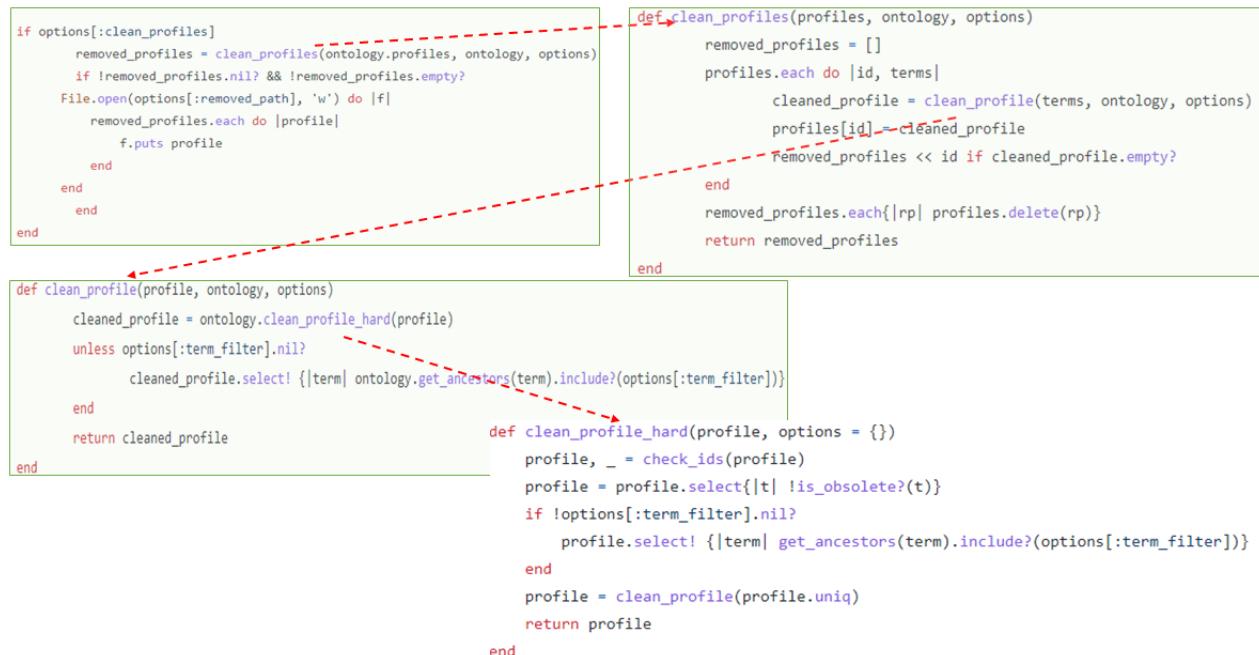


Figura 17.13

Ahora, veamos un poco más en detalle qué está sucediendo con la función *clean\_profile\_hard()*. Según podemos ver en la imagen de arriba 17.13 y abajo 17.14, se empieza utilizando el método *check\_ids()* para obtener los Main IDs de cada término y eliminar aquellos que no estén en la ontología. Con estos, se pasa a eliminar (segunda línea del cuerpo de la función) los

<sup>9</sup>De hecho, puede ser que sólo queramos modificar los perfiles, sin necesidad de un análisis posterior.

<sup>10</sup>Se entiende por encadenadas como un conjunto de funciones formando una cadena lineal y ordenada de dependencias.

términos obsoletos y luego... espera... luego ¿Se vuelve a realizar la misma operación sobre el filtrado con el comando -T? Esto parece una redundancia en el código que podría haberse dejado para permitir compatibilidades con antiguos scripts o porque simplemente...sigamos<sup>11</sup>. Finalmente, se vuelve a realizar un limpiado del perfil llamando al método `clean_profile()`<sup>12</sup>, con el que eliminar términos que sean ancestros o perfiles alternativos.

```

def clean_profile_hard(profile, options = {})

def check_ids(ids, substitute: true)
  checked_codes = []
  rejected_codes = []
  ids.each do |id|
    if @stanzas[:terms].include? id
      if substitute
        checked_codes << self.get_main_id(id)
      else
        checked_codes << id
      end
    else
      rejected_codes << id
    end
  end
  return checked_codes, rejected_codes
end

def clean_profile(profile, remove_alternatives: true)
  warn('Estructure is circular, behaviour could not be which is expected') if @structureType == :circular
  terms_without_ancestors, _ = self.remove_ancestors_from_profile(profile)
  if remove_alternatives
    terms_without_ancestors_and_alternatives, _ = self.remove_alternatives_from_profile(terms_without_ancestors)
  else
    terms_without_ancestors_and_alternatives = terms_without_ancestors
  end
  return terms_without_ancestors_and_alternatives
end

```

Figura 17.14

Por si se desea profundizar un poco más, aquí se dejan los dos métodos involucrados en el último limpiado del perfil.

```

def clean_profile(profile, remove_alternatives: true)

1657 def remove_ancestors_from_profile(prof)
1658   ancestors = prof.map{|term| self.get_ancestors(term)}.flatten.uniq
1659   redundant = prof.select{|term| ancestors.include?(term)}
1660   return prof - redundant, redundant
1661 end

1669 def remove_alternatives_from_profile(prof)
1670   alternatives = prof.select{|term| @alternatives_index.include?(term)}
1671   redundant = alternatives.select{|alt_id| prof.include?(@alternatives_index[alt_id])}
1672   return prof - redundant, redundant
1673 end

```

Figura 17.15

### Expansión de perfiles para ancestros e hijos (comando -e)

En función del valor asignado al parámetro `Options[:expand_profiles]` podemos, o bien expandir los perfiles para añadir los términos parentales ( `Options[:expand_profiles] = 'parental'`), o bien, anotar por cada término qué IDs tiene asociado. Esto es algo que ya sabíamos de la sección de uso del grimorio.

La opción de 'parental' es relativamente sencilla de comprender: "Dado un conjunto de términos para un perfil, tomamos todos los ancestros, directos e indirectos, de los mismos, y lo asociamos al perfil". Sin embargo, se hace un poco más escurridiza la opción de 'propagate', intentemos desgranarla<sup>13</sup>.

<sup>11</sup>O simplemente podría ser que es un fragmento de código obsoleto

<sup>12</sup>CUIDADO: NO ES la MISMA que la FUNCIÓN del BIN.

<sup>13</sup>Usando como intuición lo mencionado por la sección de Usuario del grimorio pero dando prioridad a la información directamente sacada del código.

Como primera aproximación, veamos la variable @items que parece estar constantemente involucrada en el proceso. Esta variable, recordamos, consiste en un hash con key los términos de la ontología y value, el ID del perfil al que está asociado (que puede ser uno o varios). Así, el proceso quedaría algo como: I- *get\_items\_from\_profiles*, "almacena la asociación término-ID en el hash @items"; II- *expand\_items\_to\_parentals()*, "expande la relación término-ID con los padres de los términos"; III- *get\_profiles\_from\_items()*, "actualiza la variable @profiles para incluir los términos nuevos incluidos" y IV- *add\_observed\_terms\_from\_profiles()*, "actualiza el cálculo de los términos observados".



Figura 17.16

### Problemas en la comprensión de expand\_items\_to\_parentals

De momento en este análisis no se ha conseguido terminar de comprender la lógica subyacente al método *expand\_items\_to\_parentals*, clave para la opción de 'propagate'. Pese a ello, se deja una imagen del código con los comentarios que "más o menos" han podido sacarse en claro. El principal problema se basa en el uso del @items, parece que obtiene como values términos y no ID de perfiles... porque las funciones que utilizan lo que aceptan los términos... no termina de quedar claro.

```

def expand_items_to_parentals(ontology: nil, minimum_childs: 2, clean_profiles: true)
  targetKeys = expand_profile_with_parents(@items.keys)  Expandir conjunto de términos a sus respectivos parentales directos e indirectos.
  terms_per_level = list_terms_per_level(targetKeys)          Obtener el hash key: level => Value: terms
  terms_per_level = terms_per_level.to_a.sort([11, 12] 11.first <=> 12.first} # Obtain sorted levels
  terms_per_level.pop # Leaves are not expandable Eliminamos los niveles más bajos registrados

terms_per_level.reverse_each do |lvl, terms| # Expand from leaves to roots
  terms.each do |term|      Para cada término del nivel lvl...
    childs = self.get_descendants(term,true).select{|t| @items.include?(t)}   Sacamos aquellos hijos que están incluidos en los perfiles
    next if childs.length < minimum_childs
    propagated_item_count = Hash.new(0)  El hash que...
    if ontology.nil? # Count how many times is presented an item in childs
      childs.each do |child|
        @items[child].each{|i| propagated_item_count[i] += 1}
      end
    else # Count take into account similarity between terms in other ontology. Not pretty
      while childs.length > 1  Vamos cogiendo los hijos uno a uno hasta que no quede ninguno
        curr_term = childs.shift
        childs.each do |child| Vamos cogiendo los hijos uno a uno hasta que no quede ninguno
          curr_items = @items[curr_term]?????
          child_item = @items[child]
          maxmica_counts = Hash.new(0)
          curr_items.each do |item|
            maxmica = ontology.get_maxmica_term2profile(item, child_items)
            maxmica_counts[maxmica.first] += 1
          end
          child_items.each do |item|
            maxmica = ontology.get_maxmica_term2profile(item, curr_items)
            maxmica_counts[maxmica.first] += 1
          end
          maxmica_counts.each{|t,freq| propagated_item_count[t] += freq if freq >= 2} Si la frecuencia no es menor que uno, se aumenta el conteo del
        end
      end
    end
    propagated_items = propagated_item_count.select{|k,v| v >= minimum_childs}.keys
  end
end
propagated_items = propagated_item_count.select{|k,v| v >= minimum_childs}.keys
if propagated_items.length > 0
  query = @items[term]
  if query.nil?
    @items[term] = propagated_items
  else
    terms = @items[term] | propagated_items
    terms = ontology.clean_profile(terms) if clean_profiles && !ontology.nil?
    @items[term] = terms
  end
end
end
end
end
Se añaden los ítems propagados (?)

```

Figura 17.17

### Traducción de los términos de los perfiles (comando -t)

Este apartado resulta más sencillo, ya que se emplean métodos que ya se han mostrado con anterioridad. De hecho, el código resulta prácticamente autoexplicativo 17.18

```

450  if options[:translate] == 'names'
451    translate(ontology, 'names', options)
452 end

51  def translate(ontology, type, options, profiles = nil)
52    not_translated = {}
53    if type == 'names'
54      ontology.profiles.each do |id, terms|
55        translation, untranslated = ontology.translate_ids(terms)
56        ontology.profiles[id] = translation
57        not_translated[id] = untranslated unless untranslated.empty?
58      end
59    elsif type == 'codes'
60      profiles.each do |id, terms|
61        translation, untranslated = ontology.translate_names(terms)
62        profiles[id] = translation
63        profiles[id] = profiles[id].join("#{options[:separator]}")
64        not_translated[id] = untranslated unless untranslated.empty?
65      end
66    end
67    if !not_translated.empty?
68      File.open(options[:untranslated_path], 'w') do |file|
69        not_translated.each do |id, terms|
70          file.puts([id, terms.join(";")].join("\t"))
71        end
72      end
73    end
74  end

```

Figura 17.18

### (II) Operaciones que analicen los datos (Operadores analíticos):

De momento hemos visto cómo se realizan las operaciones para modificar los términos, podemos utilizar métodos semejantes para los perfiles. No obstante, aún nos queda uno de los aspectos más importantes, que es el análisis de los perfiles en sí.

Recordando el apartado inicial, las modificaciones que podemos realizar sobre estos perfiles eran:

- Obtención del IC de cada perfil (-I).
- Cálculo de la similitud semántica entre perfiles (-s).
- Descriptores estadísticos del conjunto de perfiles (-n).

#### Obtención del IC de cada perfil (-I)

En primer lugar, debemos realizar la cuenta de los términos observados en la lista de perfiles. Para ello, fijándonos en el siguiente bloque de código 17.19, lo que se actualiza es la frecuencia observada. De tal modo que, si el término no se encuentra almacenado previamente en la variable @stanzas el conteo sobre el número de ancestros y descendientes va a ser de "-1", pero las cuentas en lo que respecta a los observados será la "usual".

```

439  if options[:IC]
440    ontology.add_observed_terms_from_profiles
441    by_ontology, by_freq = ontology.get_profiles_results_dual_ICs
442    ic_file = File.basename(options[:input_file], ".") + '_IC_onto_freq'
443    File.open(ic_file, 'w') do |file|
444      ontology.profiles.keys.each do |id|
445        file.puts([id, by_ontology[id], by_freq[id]].join("\t"))
446      end
447    end
448  end
449

1555   # Includes as "observed_terms" all terms included into stored profiles
1556   # ===== Parameters
1557   # +reset::: if true, reset observed freqs already stored before re-calculate
1558   def add_observed_terms_from_profiles(reset: false)
1559     @meta.each{|term, freqs| freqs[:observed_freq] = -1} if reset
1560     @profiles.each{|id, terms| self.add_observed_terms(terms: terms)}
1561   end

1562

415    def add_observed_terms(terms:, increase: 1.0, transform_to_sym: false)
416      # Check
417      raise ArgumentError, 'Terms array given is NIL' if terms.nil?
418      raise ArgumentError, 'Terms given is not an array' if !terms.is_a? Array
419      # Add observations
420      if transform_to_sym
421        checks = terms.map{|id| self.add_observed_term(term: id.to_sym, increase: increase)}
422      else
423        checks = terms.map{|id| self.add_observed_term(term: id, increase: increase)}
424      end
425      return checks
426    end

391  # true if process ends without errors, false in other cases
392  def add_observed_term(term:, increase: 1.0)
393    # Check
394    raise ArgumentError, "Term given is NIL" if term.nil?
395    return false unless @stanzas[:terms].include?(term)
396    return false if @removable_terms.include?(term)
397    # Check if exists
398    @meta[term] = {:ancestors => -1.0, :descendants => -1.0, :struct_freq => 0.0, :observed_freq => 0.0} if @meta[term].nil?
399    # Add frequency
400    @meta[term][:observed_freq] += 0 if @meta[term][:observed_freq] == -1
401    @meta[term][:observed_freq] += increase
402    # Check maximum frequency
403    @max_freqs[:observed_freq] = @meta[term][:observed_freq] if @max_freqs[:observed_freq] < @meta[term][:observed_freq]
404    return true
405  end

```

Si el término no estaba previamente almacenado en @meta, se designa con un -1 al principio...  
...pero para proceder con el conteo, a efectos prácticos la cuenta se realiza normal

Figura 17.19

Con la frecuencia de términos observados actualizados, pasamos a calcular los índices de información (ICs) sobre los términos (con el método `get_IC()`), tanto para un IC estructural como para el IC en base de los términos observados. Una vez obtenido estos valores, se realiza la media de los valores de los ICs (con el método `get_profile_mean_IC()`) 17.20. Enfoquémonos ahora un poco más en el método `get_IC()`.

```

439 if options[:IC]
440   ontology.add_observed_terms_from_profiles
441   by_ontology, by_freq = ontology.get_profiles_resnik_dual_ICs
442   ic_file = File.basename(options[:input_file], ".")+'_IC_onto_freq'
443   File.open(ic_file, 'w') do |file|
444     ontology.profiles.keys.each do |id|
445       file.puts([id, by_ontology[id], by_freq[id]].join("\t"))
446     end
447   end
448 end
1794 → def get_profiles_resnik_dual_ICs
1795   struct_ics = {}
1796   observ_ics = {}
1797   @profiles.each do |id, terms|
1798     struct_ics[id] = self.get_profile_mean_IC(terms, ic_type: :resnik)
1799     observ_ics[id] = self.get_profile_mean_IC(terms, ic_type: :resnik_observed)
1800   end
1801   return struct_ics.clone, observ_ics.clone
1802 end
1803 # +zhou_k+: special coefficient for Zhou IC method
1804 # ===== Returns
1805 # mean IC for a given profile
1806 def get_profile_mean_IC(prof, ic_type: :resnik, zhou_k: 0.5)
1807   return prof.map{|term| self.get_IC(term, type: ic_type, zhou_k: zhou_k)}.inject(0){|sum,x| sum + x}.fdiv(prof.length)
1808 end

```

Figura 17.20

El método `get_IC()` se va a encargar de: (I) comprobar que el tipo de IC que se ha solicitado está permitido ( con la variable `@allowed_calcs[:ics]`), (II) Extraer la información de la variable `@meta` y (III) Calcular el tipo de ICs que se ha solicitado. La siguiente cuestión es: ¿Cuáles son y cómo se obtienen los distintos ICs?...veámoslo 17.22

```

846   def get_IC(termRaw, type: :resnik, force: false, zhou_k: 0.5)
847     term = termRaw.to_sym
848     curr_ics = @ics[type]
849     # Check
850     raise ArgumentError, "IC type specified (#{$type}) is not allowed" if !@allowed_calcs[:ics].include?(type)
851     # Check if it's already calculated
852     return curr_ics[term] if (curr_ics.include? term) && !force
853     # Calculate
854     ic = -1
855     term_meta = @meta[term]
856     case type # https://arxiv.org/ftp/arxiv/papers/1310/1310.8059.pdf ||| https://sci-hub.st/https://doi.org/
857
858       curr_ics[term] = ic
859       return ic
860     end

```

**Optimización que permite no repetir cálculos previamente realizados para los distintos ICs.**

**Bloque de código con los distintos tipos de ICs definidos para cada caso**

Figura 17.21

Hasta la fecha, las métricas definidas en el código pueden estar basadas en el contenido de información o en técnicas híbridas, quedando posibilidad para extender a un mayor número de tipos de fórmulas. Estas quedan comentadas en el código.

Tipos de ICs que no han empezado su desarrollo	Tipos de ICs que ya poseen varias implementaciones Resnik estructural y observados, seco, zhou y sanchez
<pre>##### #### FEATURE-BASED METRICS ##### # Tversky # x-similarity # Rodriguez  ##### #### STRUCTURE BASED METRICS ##### # Shortest path # Weighted Link # Hirst and St-Onge Measure # Wu and Palmer # Slimani # Li # Leacock and Chodorow</pre>	<pre>##### #### INFORMATION CONTENT METRICS ##### when :resnik# Resnik P: Using Information Content to Evaluate Semantic Similarit     # -log(Freq(x) / Max_Freq)     ic = -Math.log10(term_meta[:struct_freq].fdiv(@max_freqs[:struct_freq])) when :resnik_observed     # -log(Freq(x) / Max_Freq)     ic = -Math.log10(term_meta[:observed_freq].fdiv(@max_freqs[:observed_freq])) # Lin # Jiang &amp; Conrath  ##### #### HYBRID METRICS ##### when :seco, :zhou # SECO: An intrinsic information content metric for semantic similarity in WordNet     # 1 - ( log(hypo(x) + 1) / log(max_nodes) )     ic = 1 - Math.log10(term_meta[:struct_freq]).fdiv(Math.log10(@stanzas[:terms].length - @alternatives_index.length))     if :zhou# New Model of Semantic Similarity Measuring in Wordnet         # k*(IC_Seco(x)) + (1-k)*(log(depth(x))/log(max_depth))         @ics[:seco][term] = ic # Special store         ic = zhou_k * ic + (1.0 - zhou_k) * (Math.log10(term_meta[:descendants]).fdiv(Math.log10(@max_freqs[:max_depth])))     end when :sanchez # Semantic similarity estimation in the biomedical domain: An ontology-basedinformation-theoretic perspective     ic = -Math.log10((term_meta[:descendants].fdiv(term_meta[:ancestors]) + 1.0).fdiv(@max_freqs[:max_depth] + 1.0)) # Knappe</pre>

Figura 17.22

### Cálculo de la similitud semántica entre perfiles (-s)

Para efectuar la similitud semántica entre perfiles se va a llevar a cabo una comparación todos contra todos entre ellos. La función `write_similarity_profile_list()` se encargará de escribir la similitud semántica entre cada uno de los pares perfil-perfil, obtenido desde el método `compare_profiles()`. De aquí, podemos ver dos métodos de gran relevancia que van a pasar a explicarse a continuación: `get_mica_index_profiles()` 17.24 y `compare()` 17.25. Comencemos por el primero...

```

426 if !options[:similarity].nil?
427   refs = nil
428   if !options[:reference_profiles].nil?
429     refs = load_tabular_file(options[:reference_profiles])
430     format_tabular_data(refs, options[:separator], 0, 1)
431     refs = refs.to_h
432   refs = clean_profiles(ontology.profiles, ontology, options) if options[:clean_profiles]
433   abort("Reference profiles are empty after cleaning ") if refs.nil? || refs.empty?
434 end
435 write_similarity_profile_list(options[:output_file], ontology, options[:similarity], refs)
436 end

def write_similarity_profile_list(output, onto_obj, similarity_type, refs)
  profiles_similarity = onto_obj.compare_profiles(sim_type: similarity_type, external_profiles: refs)
  File.open(output, 'w') do |f|
    profiles_similarity.each do |pairsA, pairsB_and_values|
      pairsB_and_values.each do |pairsB, values|
        f.puts "#{pairsA}\t#{pairsB}\t#{values}"
      end
    end
  end
end

```

Podemos optar por incorporar los perfiles de referencia o bien limpiar ciertos perfiles con el clean.

```

def compare_profiles(external_profiles: nil, sim_type: :resnik, ic_type: :resnik, bidirectional: true)
  profiles_similarity = {} #calculate similarity between patients profile
  if external_profiles.nil?
    comp_profiles = @profiles
    main_profiles = comp_profiles
  else
    comp_profiles = external_profiles
    main_profiles = @profiles
  end
  # Compare
  @_lca_index = {}
  pair_index = get_pair_index(main_profiles, comp_profiles)
  @_get_lca_index(pair_index)
  @_mica_index = {}
  @_get_mica_index_from_profiles(pair_index, sim_type: sim_type, ic_type: ic_type, lca_index: false)
  main_profiles.each do |curr_id, current_profile|
    comp_profiles.each do |id, profile|
      value = compare(current_profile, profile, sim_type: sim_type, ic_type: ic_type, bidirectional: bidirectional, store_mica: true)
      add2nestHash(profiles_similarity, curr_id, id, value)
    end
  end
  return profiles_similarity
end

```

Los external\_profiles son la referencia, es decir, sólo se harán las comparaciones de perfiles ya guardados previamente en @profiles (perfiles "internos") respecto a los perfiles "externos".

Obtención de un diccionario para determinar los términos a comparar con key [Term1,Term2].sort y valor true

Figura 17.23

... Con este método se calcula las similitudes para todo el conjunto de parejas de términos que aparecen en el archivo de perfiles a analizar. Para ello, necesitamos obtener el MICA (Maximum informative common ancestor) de cada una de las parejas, con el método `get_MICA()`, llevando a cabo al final una de las distintas posibles operaciones de similitud.

El método `get_MICA()` devolverá un array con dos elementos: [Ancestro Común con mayor IC, su IC]. Con este dato del IC del MICA almacenado en la variable `sim_res()` podemos calcular las distintas definiciones de similitud.

```

def get_mica_index_from_profiles(pair_index, sim_type: :resnik, ic_type: :resnik, lca_index: true)
  pair_index.each do |pair, val|
    tA, tB = pair
    value = self.get_similarity(tA, tB, type: sim_type, ic_type: ic_type, lca_index: lca_index)
    value = true if value.nil? # We use true to save that the operation was made but there is no
    add2nestHash(@mica_index, tA, tB, value)
    add2nestHash(@mica_index, tB, tA, value)
  end
end

def get_similarity(termA, termB, type: :resnik, ic_type: :resnik, lca_index: false)
  # Check
  raise ArgumentError, "SIM type specified (#(type)) is not allowed" if !@allowed_calcs[:sims].include?(type)
  sim = nil
  mica, sim_res = get_MICA(termA, termB, ic_type, lca_index)
  if !mica.nil?
    case type
      when :resnik
        sim = sim_res
      when :lin
        sim = (2.0 * sim_res).fdiv(self.get_IC(termA,type: ic_type) + self.get_IC(termB,type: ic_type))
      when :jiang_conrath # This is not a similarity, this is a disimilarity (distance)
        sim = (self.get_IC(termA, type: ic.type) + self.get_IC(termB, type: ic.type)) - (2.0 * sim_res)
    end
  end
  return sim
end

def get_MICA(termA, termB, ic_type = :resnik, lca_index = false)
  termA = @alternatives_index[termA] if @alternatives_index.include?(termA)
  termB = @alternatives_index[termB] if @alternatives_index.include?(termB)
  mica = [nil,-1.0]
  # Special case
  if termA.eql?(termB)
    ic = self.get_IC(termA, type: ic_type)
    mica = [termA, ic]
  else
    get_LCA(termA, termB, lca_index: lca_index).each do |lca| # Find MICA in shared ancestors
      ic = self.get_IC(lca, type: ic_type)
      mica = [lca, ic] if ic > mica[1]
    end
  end
  return mica
end

def get_LCA(termA, termB, lca_index: false)
  lca = []
  if lca_index
    res = @lca_index.dig(termA, termB)
    lca = [res] if !res.nil?
  else
    # Obtain ancestors (include itselfs too
    anc_A = self.get_ancestors(termA)
    anc_B = self.get_ancestors(termB)
    if !(anc_A.empty? && anc_B.empty?)
      anc_A << termA
      anc_B << termB
      lca = anc_A & anc_B
    end
  end
  More than LCAs is the array of CAs
  return lca
end

```

Figura 17.24

En segundo lugar, el método *compare\_profiles()* llama a *compare()* que, como su nombre indica, realiza la comparación entre dos perfiles. Como puede verse en el código de abajo 17.25, se trata de aplicar la fórmula de recuadro en azul, calculando previamente los elementos necesarios para ello.

```

438 def compare(termsA, termsB, sim_type: :resnik, ic_type: :resnik, bidirectional: true, store_mica: false)
439     # Check
440     raise ArgumentError, "Terms sets given are NIL" if termsA.nil? | termsB.nil?
441     raise ArgumentError, "Set given is empty. Aborting similarity calc" if termsA.empty? | termsB.empty?
442     micasA = []
443     # Compare A -> B
444     termsA.each do |tA|
445         micas = []
446         termsB.each do |tB|
447             if store_mica
448                 value = @mica_index[tA][tB]
449             else
450                 value = self.get_similarity(tA, tB, type: sim_type, ic_type: ic_type)
451             end
452             micas << value if value.class == Float
453         end
454         !micasA.empty? ? micasA << micasA.max : micasA << 0
455     end
456     means_sim = micasA.inject{ |sum, el| sum + el }.fdiv(micasA.size) | (I)
457     # Compare B -> A
458     if bidirectional
459         means_simA = means_sim * micasA.size
460         means_simB = self.compare(termsB, termsA, sim_type: sim_type, ic_type: ic_type, bidirectional: false, store_mica: store_mica) * termsB.size | (II)
461         means_sim = (means_simA + means_simB).fdiv(termsA.size + termsB.size)
462     end
463     # Return
464     return means_sim
465 end

```

De estar ya almacenados, se extraen los valores desde la variable `@mica_index`.

(I)

(II)

$$\text{sim}_{lin}(GenA, GenB) := \text{sim}_{lin}(C_A, C_B) = \frac{\sum_{t_a \in C_A} \max_{t_b \in C_B} (\text{sim}_{lin}(t_a, t_b)) + \sum_{t_b \in C_B} \max_{t_a \in C_A} (\text{sim}_{lin}(t_a, t_b))}{\#C_A + \#C_B}$$

Figura 17.25

### Descriptores estadísticos del conjunto de perfiles (-n)

En ocasiones puede ser de utilizada sacar la información estadística sobre un conjunto de perfiles dado. En este caso, la lectura del código resulta clara y no requiere de mayor explicación.

```

474  if options[:statistics]
475    get_stats(ontology.profile_stats).each do |stat|
476      puts stat.join("\t")
477    end
478  end

142  def get_stats(stats)
143    report_stats = []
144    report_stats << ['Elements', stats[:count]]
145    report_stats << ['Elements Non Zero', stats[:countNonZero]]
146    report_stats << ['Non Zero Density', stats[:countNonZero].fdiv(stats[:count])]
147    report_stats << ['Max', stats[:max]]
148    report_stats << ['Min', stats[:min]]
149    report_stats << ['Average', stats[:average]]
150    report_stats << ['Variance', stats[:variance]]
151    report_stats << ['Standard Deviation', stats[:standardDeviation]]
152    report_stats << ['Q1', stats[:q1]]
153    report_stats << ['Median', stats[:median]]
154    report_stats << ['Q3', stats[:q3]]
155    return report_stats
156  end

2562 → def profile_stats
2563   stats = Hash.new(0)
2564   data = @profiles.values.map{|ont_ids| ont_ids.size}
2565   stats[:average] = data.sum().fdiv(data.size)
2566   sum_devs = data.sum{|element| (element - stats[:avg]) ** 2}
2567   stats[:variance] = sum_devs.fdiv(data.size)
2568   stats[:standardDeviation] = stats[:variance] ** 0.5
2569   stats[:max] = data.max
2570   stats[:min] = data.min
2571
2572   stats[:count] = data.size
2573   data.each do |value|
2574     stats[:countNonZero] += 1 if value != 0
2575   end
2576
2577   stats[:q1] = data.get_quantiles(0.25)
2578   stats[:median] = data.get_quantiles(0.5)
2579   stats[:q3] = data.get_quantiles(0.75)
2580
2581   return stats
2582 end

```

Figura 17.26: Método info2hash. Encargada de extraer la información relevante de los *stanzas*.

## 17.4 Apéndice

Con el fin de "rellenar" aquellos métodos cuya explicación hubiese entorpecido el análisis, aquí se dejan una serie de imágenes mostrando código utilizado de manera secundaria en las operaciones anteriores:

```

196     def self.info2hash(attributes, split_char = " ! ", selected_field = 0)
197         # Load info
198         info_hash = {}
199         # Only TERMS multivalue tags (future add Typedefs and Instance)
200         # multivalue_tags = [:alt_id, :is_a, :subset, :synonym, :xref, :intersection_of, :union_of, :disjoint_from, :relationship, :replaced_by, :consider]
201         attributes.each do |tag, value|
202             value.gsub!(/{[\\"A-Z0-9\\.\-,\?&_]+}/, '') if tag == 'is_a' # To delete extra attributes (source, xref) in is_a tag of MONDO ontology
203             # Check
204             raise EncodingError, 'Info element incorrect format' if (tag.nil?) || (value.nil?)
205             # Prepare
206             tag = tag.lstrip.to_sym
207             value.lstrip!
208             value = value.split(split_char)[selected_field].to_sym if @@tags_with_trailing_modifiers.include?(tag)
209
210             # Store
211             query = info_hash[tag]
212             if !query.nil? # Tag already exists
213                 if !query.kind_of?(Array) # Check that tag is multivalue
214                     raise('Attempt to concatenate plain text with another. The tag is not declared as multivalue. [' + tag.to_s + ']' + query + ')')
215                 else
216                     query << value # Add new value to tag
217                 end
218             else # New entry
219                 if @@multivalue_tags.include?(tag)
220                     info_hash[tag] = [value]
221                 else
222                     info_hash[tag] = value
223                 end
224             end
225         end
226         self.symbolize_ids(info_hash)
227     return info_hash
228 end

```

```

302     def self.symbolize_ids(item_hash) JUST SYMBOLIZE THE VALUES!
303         @@symbolizable_ids.each do |tag|
304             query = item_hash[tag]
305             if !query.nil?
306                 if query.kind_of?(Array)
307                     query.map!{|item| item.to_sym}
308                 else
309                     item_hash[tag] = query.to_sym if !query.nil?
310                 end
311             end
312         end
313     end

```

Figura 17.27: Método info2hash. Encargada de almacenar la información extraída de los sntanzas en fomra de hash.

```

717     def get_index_obsoletes(obs_tag: @@basic_tags[:obsolete], alt_tags: @@basic_tags[:alternative])
718         if @stanzas[:terms].empty?
719             warn('stanzas terms empty')
720         else
721             # Check obsoles
722             @stanzas[:terms].each do |id, term_tags|
723                 next if term_tags.nil?
724                 next if self.is_alternative?(id)
725                 query = term_tags[obs_tag]
726                 if !query.nil? && query == 'true' # Obsolete tag presence
727                     next if !obsoletes_index[id].nil? # Already stored
728                     # Check if alternative value is available
729                     alt_ids = alt_tags.map{|alt| term_tags[alt]}.compact
730                     if alt_ids.empty?
731                         alt_id = alt_ids.first.first #FIRST tag, FIRST id
732                         # Store
733                         @alternatives_index[id] = alt_id
734                         @obsoletes_index[id] = alt_id
735                     end
736                     @obsoletes[id] = true
737                 end
738             end
739         end
740     end
741
742     def get_index_alternatives(alt_tag: @@basic_tags[:alternative].last)
743         # Check input
744         raise('stanzas terms empty') if @stanzas[:terms].empty?
745         # Take all alternative IDs
746         alt_ids2add = {}
747         @stanzas[:terms].each do |id, tags|
748             if id == tags[:id] # Avoid simulated alternative terms
749                 # id = tags[:id] # Take always real ID in case of alternative terms simulated
750                 alt_ids = tags[:tag]
751                 if alt_ids.nil?
752                     alt_ids = alt_ids - @removable_terms - [id]
753                     # Update info
754                     alt_ids.each do |alt_term|
755                         @alternatives_index[alt_term] = id
756                         alt_ids2add[alt_term] = $stanzas[:terms][id] if !@stanzas[:terms].include?(alt_term)
757                         @ancestors_index[alt_term] = @ancestors_index[id] if !@ancestors_index[id].nil?
758                     end
759                 end
760             end
761         end
762         @stanzas[:terms].merge!(alt_ids2add)
763     end

```

Figura 17.28: Métodos get\_index\_obsoletes() y get\_index\_alternatives(). Utilizados para obtener los términos obsoletos y los que poseen término alternativo, respectivamente.

```

1809     def calc_term_levels(calc_paths: false, shortest_path: true)
1810         if @term_paths.empty?
1811             if calc_paths
1812                 self.calc_term_paths
1813             else
1814                 warn('Term paths are not already loaded. Aborting dictionary calc')
1815             end
1816         end
1817         if !@term_paths.empty?
1818             byTerm = {}
1819             byValue = {}
1820             # Calc per term
1821             @term_paths.each do |term, info|
1822                 level = shortest_path ? info[:shortest_path] : info[:largest_path]
1823                 if level.nil?
1824                     level = -1
1825                 else
1826                     level = level.round(0)
1827                 end
1828                 byTerm[term] = level
1829                 queryLevels = byValue[level]
1830                 if queryLevels.nil?
1831                     byValue[level] = [term]
1832                 else
1833                     byValue[level] << term
1834                 end
1835             end
1836             @dicts[:level] = {byTerm: byValue, byValue: byTerm} # Note: in this case
1837             # Update maximum depth
1838             @max_freqs[:max_depth] = byValue.keys.max
1839         end
1840     end
1841
1842     def calc_term_paths(only_main_terms=false)
1843         self.calc_ancestors_dictionary if @dicts[:is_a].nil?
1844         visited_terms = {} # PEDRO: To keep track of visited
1845         @term_paths = {}
1846         if [:hierarchical, :sparse].include? @structure_type
1847             @stanzas[terms].each do |term, t_attributes|
1848                 if !only_main_terms && (self.is_obsolete?(term) || self.is_alternative?(term)) # special case (obsoletes)
1849                     special_term = term
1850                     term = self.isObsolete?(term) ? @obsoletes_index[term] : @alternatives_index[term]
1851                     @term_paths[term] = {total_paths: 0, largest_path: 0, shortest_path: 0, paths: []} if !@term_paths.include?(term)
1852                     @term_paths[special_term] = @term_paths[term]
1853                     visited_terms[special_term] = true
1854                 end
1855                 if !visited_terms.include?(term)
1856                     # PEDRO: This code is very similar to expand_path method, but cannot be replaced by it (test fail). We must work to
1857                     path_attr = @term_paths[term]
1858                     if path_attr.nil?
1859                         path_attr = {total_paths: 0, largest_path: 0, shortest_path: 0, paths: []} # create new path data
1860                         @term_paths[term] = path_attr # save path data container
1861                     end
1862                     parentals = @dicts[:is_a][byTerm][term]
1863                     if parentals.nil?
1864                         path_attr[:paths] << [term]
1865                     else
1866                         parentals.each do |direct_parental|
1867                             self.expand_path(direct_parental)
1868                             new_paths = @term_paths[direct_parental][:paths]
1869                             path_attr[:paths] = path_attr[:paths].concat(new_paths.map{|path| path.clone.unshift(term)})
1870                         end
1871                     end
1872                     anc = @ancestors_index[term].each{|anc| visited_terms[anc] = true} if @ancestors_index.include?(term)
1873                     visited_terms[term] = true
1874                 end
1875                 # Update metadata
1876                 path_attr = @term_paths[term]
1877                 path_attr[:total_paths] = path_attr[:paths].length
1878                 paths_sizes = path_attr[:paths].map{|path| path.length}
1879                 path_attr[:largest_path] = paths_sizes.max
1880                 path_attr[:shortest_path] = paths_sizes.min
1881             end
1882             else
1883                 warn('ontology structure must be hierarchical or sparse to calculate term levels. Aborting paths calculation')
1884             end
1885         end
1886     end
1887
1888     def calc_ancestors_dictionary
1889         self.calc_dictionary(:is_a, substitute_alternatives: false, self_type_references: true, multimodel: true)
1890     end

```

Figura 17.29: Métodos calc\_term\_levels junto a calc\_term\_paths(). Empleados para extraer la información de los niveles en los que se encuentra cada término en la ontología. En recuadro rojo se señala al método encargado de almacenar la información en la variable @dicts.





## 18. Desarrollo en R

### 18.1 Descripción

Aquí explicaremos el protocolo del grupo para el desarrollo en R. Centrándonos en el ejemplo del paquete ExpHunterSuite, detallaremos todos los pasos necesarios desde la introducción de los cambios hasta su subida al repositorio de GitHub. También daremos algunas pinceladas sobre estilo, dejando un enlace a una guía que profundiza más. Al final del capítulo daremos indicaciones sobre cómo crear un paquete de R desde cero.

### 18.2 Preparar el espacio de trabajo

Lo primero es crear una carpeta en nuestro usuario de Picasso que se llame dev\_R. Es muy importante tener nuestros directorios ordenados, así que tendremos este directorio para toda la parte de desarrollo en R, no sólo el Hunter. En este directorio clonaremos el repositorio <https://github.com/seoanezonjic/ExpHunterSuite>, y ya estaremos listos para trabajar. Para confirmar que todo ha ido bien, basta con consultar el estado del repositorio:

```
git status
```

Si todo ha ido bien, debería decirnos que estamos en la rama "master" y que todo está actualizado. Si falla, algo hemos hecho mal.

```
aestebanm@picasso:~/dev_R/ExpHunterSuite> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   NAMESPACE
    modified:   man/STC.Rd
    deleted:   man/caught_pairwise_termsim.Rd
    modified:   man/main_clusters_to_enrichment.Rd
    modified:   man/trycatch_pairwise_termsim.Rd
    modified:   man/write_clusters_to_enrichment.Rd
    modified:   man/write_functional_report.Rd

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ..Rcheck/
    man/get_org_db.Rd

no changes added to commit (use "git add" and/or "git commit -a")
```

Figura 18.1: Ejemplo de lo que puede aparecer al ejecutar «git status». Estamos trabajando en la rama **master** del repositorio **ExpHunterSuite**.

Para poder subir los cambios necesitaremos permisos de colaborador, así que es buena idea irlos pidiendo. Si es tu primera vez manejando el Hunter, y sobre todo si es tu primera vez trabajando en un repositorio con más colaboradores, es buena idea hacerte tu propia rama para que los posibles errores no afecten a nadie más. Esto se consigue de la siguiente manera:

```
1 git checkout -b branch_name
```

Donde `branch_name` será el nombre de nuestra rama. Git checkout es el comando que permite cambiar de rama, y el flag `-b` indica que es una rama que aún no existe y se debe crear. Lo habitual en el grupo es ir trabajando directamente en la rama master, comunicándose con los demás para coordinar cambios locales. Esto es lo último que se explicará de Git y GitHub. Si tienes más dudas, consulta el capítulo 3.

### 18.3 Guía de estilo

Seguimos la guía de estilo Hadley Wickham. Si tienes dudas, consúltala.

### 18.4 Estructura general del paquete

Aquí explicaremos la organización general de los scripts y cómo se debe proceder en el desarrollo, usando como ejemplo de uso el flujo de expresión diferencial (`degenes_Hunter`).

#### 18.4.1 Hacer el paquete ejecutable dentro de la CMD

Por cómo trabajamos en el grupo, a menudo es conveniente utilizar nuestras herramientas no como paquetes o librerías, sino llamando a un script desde la línea de comandos. Es sencillito: Verás que hay una carpeta del paquete de R que no hemos tocado, llamada «`inst`». Esta carpeta es especial, puesto que su contenido no se analiza en el check de RMD ni el de BioConductor. Tenemos libertad total en este directorio, no hay ningún estándar que debamos seguir, y por lo tanto será lo que utilicemos para construir esta opción personalizada.

### Modo usuario

- **1. Scripts.** En el initializes crearemos un sh por cada una de las capacidades que queramos tener disponibles en línea de comandos. No serán funciones individuales, como en la carpeta R, sino el flujo al que queramos llamar. En el caso del html\_reportR, el script create\_reportR.sh se llama con los flags «-t» e «-i» (plantillas y tablas de entrada) para renderizar una plantilla con una selección de tablas.
- **2. Initializes.** Crear en la cuenta de software (o, mejor dicho, pedir a Pedro que cree en la cuenta de software) un archivo initializes que exporte el PATH a los scripts definidos en la carpeta inst del paquete.

### Modo desarrollo

- **1. Modo DEVEL.** Pepe me tendrá que echar una mano con esto.

#### 18.4.2 Main

El script R/main\_degenes\_Hunter.R contiene la función principal, es decir, la «main», que tiene el mismo nombre que el archivo. Esta función contiene las llamadas a todas las demás contenidas en este archivo, que utiliza para procesar los datos. Si vas a introducir una nueva función que se vaya a utilizar en expresión diferencial, debe ir definida en este archivo, después de la definición de main. Luego, main debe contener una llamada a esa función con los argumentos necesarios. Si debes añadir argumentos nuevos, no olvides documentarlos en el cabecera del archivo. Fíjate en las líneas con «@param», e imítalas. Esta función devuelve los resultados dentro de la lista final\_results. Si tu función añade un resultado nuevo, deberás crear un elemento nuevo de esta lista que lo contenga.

#### 18.4.3 write\_report

Este script (R/write\_report.R) recoge el objeto final\_results del main y lo «desmonta». Por tanto, lo primero que tenemos que hacer es añadir una línea que haga esto mismo (fíjate en cómo lo hace para otros objetos ya existentes). El uso más habitual que faremos de este script es simplemente añadir elementos a una función ya existente. write\_report llamará entonces a la plantilla de Rmd correspondiente, que tendrá acceso a todas las variables definidas en la función.

#### 18.4.4 report (Rmd)

Es una plantilla de markdown ejecutada desde el script R/write\_report.R, y se encuentra en /inst/templates/main\_report.Rmd. Es en esta plantilla donde deberemos añadir las gráficas para nuestros resultados obtenidos. NO debe haber procesamiento de datos, sólo graficado. Todo esto irá a un archivo html.

**Details of the input data**

- Data quality control (QC)
- Count metrics by sample ranks
- Statistics of expressed genes
- Gene counts variance distribution
- DEgenes Hunter results
- DE detection package specific results
- Detailed package results comparison

**ExpHunterSuite: Differential Expression Report**

**Details of the input data**

**First group of samples (to be referred to as control in the rest of the report)**

Sample Names:

- WT\_NT\_1
- WT\_NT\_2
- WT\_NT\_3

**Second group of samples (to be referred to as treatment in the rest of the report)**

Sample Names:

- rd10\_TR\_1
- rd10\_TR\_2
- rd10\_VH\_1
- rd10\_VH\_2
- rd10\_VH\_3

Note: A positive log fold change shows higher expression in the treatment group; a negative log fold change represents higher expression in the control group.

**Data quality control (QC)**

**Correlation between samples:**

Here we show scatterplots comparing expression levels for all genes between the different samples, for i) all controls, ii) all treatment samples and iii) for all samples together. These plots will only be produced when the total number of samples to compare within a group is less than or equal to 10.

**Correlation between control samples:**

Replicates within the same group tend to have Pearson correlation coefficients  $\geq 0.96$ . Lower values may indicate problems with the samples.

Figura 18.2: Esto es lo primero que verás al abrir el report de **ExpHunterSuite**

#### 18.4.5 Documentación del paquete (man pages)

Las marcas que has estado añadiendo del estilo «param» son marcas para el paquete roxygen2, un intérprete de markdown que genera los manuales a los que accedes cada vez que ejecutas «help(function)» o «?function». Estás, esencialmente, anotando las órdenes que el paquete va a seguir para documentar tu función. Estas órdenes se ejecutan llamando a la función «devtools::document()» desde el directorio raíz del paquete.

#### 18.5 Construir y comprobar el paquete

Una vez hayamos hecho los cambios, es posible que todo parezca funcionar perfectamente, produciendo los resultados esperados y sin romper el resto del código. Sin embargo, aún no estamos listos para subir los cambios. Nos queda la parte más temible de todas: asegurarnos de que estamos respetando la estructura y los requisitos de un paquete de R.

Subiendo de nuevo al directorio dev\_R, toca comprobar que los cambios que hemos introducido respalan los requisitos para elaborar un paquete de R. Este es el comando para construirlo:

```
R CMD build ExpHunterSuite --no-build-vignettes --resave-data
```

#### WARNING: Errores al construir el paquete

Si esto falla, ha habido un error MUY grave. Trázalo bien, y pide ayuda si hace falta.

En flag «--no-build-vignettes», como su nombre indica, evita que se construyan las viñetas del paquete. La elaboración de las viñetas vendrán más adelante, así que no tiene sentido construirlas ahora. «--resave-data» indica al programa que guarde el paquete de la forma más compacta posible (por defecto usará gzip).

Una vez construido, ya podemos comprobar que funciona como paquete y que respeta todos los requisitos que se espera de uno.

Debemos ejecutar la siguiente línea:

```
1 R CMD check ExpHunterSuite_version.tar.gz --ignore-vignettes >
check_results&
```

Donde pone «version» debemos poner la versión del Hunter en la que estemos trabajando (por ejemplo, 1.7.2, la versión en el momento en el que estamos escribiendo esta sección). El flag «--ignore-vignettes» sirve para que no compruebe las viñetas, ya que de eso nos encargaremos más adelante. El «&» del final es algo general de bash, sirve para ejecutar el comando en segundo plano. Tardará bastante, así que es mejor que redirija la salida a un archivo y ejecute en segundo plano, y así nosotros podremos seguir trabajando. Aquí es más frecuente encontrarse errores. Por ejemplo, es posible que hayamos introducido una llamada a un paquete instalado y que nos haya funcionado bien, pero que no se haya indicado en la sección «Imports» del archivo «DESCRIPTION». Al hacer pruebas, nos habrá funcionado perfectamente, pero saltará el error en la comprobación del paquete. Por estas cosas es tan importante este paso.

También es habitual que nos encontremos **WARNINGS**. Esto nos permitirá seguir adelante, pues el paquete funcionará, pero realmente nos estamos saltando las buenas prácticas de un paquete de R, lo cual puede ser más o menos grave. Hay que solucionarlos, aunque no rompa. Lo más habitual es que se haya olvidado documentar un nuevo argumento de entrada de una función (¡no te olvides del roxygen!), o algo por el estilo.

Por último, nos encontraremos con **NOTES**. Son cosas menos graves, pero que también hay que arreglar. Existen algunos que no son errores en sí: por ejemplo, al subir un paquete a CRAN por primera vez, aparece una nota avisando a los responsables del servicio de que ese paquete es nuevo, y por tanto hay que prestar especial atención al revisarlo.

Para más información sobre el desarrollo de paquetes en R, recomendamos el manual de Hadley Wickham.

## 18.6 Actualizar Readme

Ahora mismo, el readme que utiliza el Hunter es el archivo README.Rmd. Este formato nos permite incluir llamadas a GitHub Actions, que pueden aportar diferentes beneficios. Lo que tienes que saber es que, si quieras actualizar el Readme, debes actualizar este. **NO TOQUES README.Md.**

Una vez hayas terminado con tus cambios, inicia R en el root del hunter y ejecuta `usethis::build_readme`. Te actualizará el contenido de README.Md a partir del contenido de README.Rmd.

## 18.7 Resumen

- 1. Procesamiento de datos en main.
- 2. Recoger resultados en `write_report`.
- 3. Dibujar resultados en plantilla (.Rmd).
- 4. Comprobar estilo.
- 5. Documentar funciones nuevas.
- 6.

```
1 R CMD build ExpHunterSuite --no-build-vignettes --resave-data
```

- 7.

```
R CMD check ExpHunterSuite_version.tar.gz --ignore-vignettes >
check_results &
```

- 8. Ver si otro miembro del grupo tiene cambios locales. Si tiene, solucionar conflictos. Si no, push.

## 18.8 Crear un paquete desde cero

Sigue las mismas indicaciones de preparación de espacio de trabajo. Esta vez, en vez de clonar un repositorio, lo vamos a crear aquí utilizando la herramienta devtools de R. Es muy sencillo: carga R y ejecuta:

```
usethis::create_package("nombre_del_paquete")
```

Esto creará el esqueleto mínimo indispensable para el desarrollo del paquete. Estos serán la carpeta R y los archivos NAMESPACE y DESCRIPTION. A continuación, inicializamos el repositorio de Git. También lo haremos desde R, con una función que nos hace la vida mucho más cómoda añadiendo automáticamente ciertos archivos al .gitignore. La función es:

```
usethis::use_git()
```

Te preguntará si quieres añadir esos cambios (los del .gitignore) al commit. Son un poco graciosos y te dan tres opciones: "No", "Nope" y "Yeah". Pulsaremos el 3 para seleccionar "Yeah". A continuación tenemos que seguir las indicaciones del capítulo [git] para configurar el repositorio.

### 18.8.1 DESCRIPTION y LICENSE

El archivo DESCRIPTION se genera automáticamente al generar el paquete, pero tenemos que rellenarlo a mano. El nombre del paquete se añade automáticamente, consultando el nombre de su directorio raíz. En "Title" debemos poner una descripción muy breve. Por ejemplo, el paquete htmlreportR tiene como título "HTML reporting toolkit". Debemos evitar expresiones como "An R package for...", ya que no hace falta aclarar que es un paquete de R. Hadley Wickham explica muy bien esta parte.

Luego nos encontramos el campo VERSION. No lo modificaremos desde aquí.

El archivo DESCRIPTION también contiene un campo para los autores. Se genera con una plantilla muy sencilla de llenar. Más importantes son los roles, de entre los cuales nos interesan:

- «cre», de «creator». Pese a lo que su nombre indica, realmente se utiliza para indicar quién es el encargado del mantenimiento del paquete. CRAN contactará con esta persona en caso de que haga falta arreglar algo. Es el único rol que está obligado a incluir su correo electrónico en la lista de autores.
- «aut», de «author». Han contribuido de forma significativa al paquete.
- «ctb», de «contributor»Contribuciones más pequeñas, como parches.

A continuación encontramos el campo Description, que es una descripción de hasta un párrafo del paquete. Podemos entrar en más detalle en qué hace. No editaremos manualmente el resto de campos de este archivo.

El archivo LICENSE no se genera solo, tenemos que hacerlo nosotros. Para eso, iniciamos R en la terminal, en el directorio raíz del paquete, y usamos un comando de usethis. Para saber cuál, tenemos que decidir qué licencia usar. Para los paquetes del grupo solemos usar la licencia MIT, cuyo comando es el siguiente:

```
| usethis::use_mit_license()
```



# 19. Chuletario

En este capítulo se expondrán particularidades que se han encontrado en diversos proyectos y su solución, con la intención de ayudar en el futuro si se diesen los mismos casos.

## 19.0.1 1 muestra != 1 archivo

En este caso, una muestra del proyecto no corresponde a un archivo .fastq.gz, sino que varios archivos .fastq.gz constituyen una muestra. Para concatenar los distintos archivos en una muestra, se deben descomprimir los .fastq.gz, concatenarlos y volver a comprimirlos. Se puede hacer creando un script con el siguiente comando para cada muestra:

```
gunzip -c muestra_1 muestra_2 muestra_3 | gzip -c - >  
$output_folder/nombre_final_muestra
```

*Truco:* Utilizar Excel para generar los comandos de manera automática y posteriormente pegar los comandos en el script.sh.

## 19.0.2 Análisis funcional con archivos proporcionados por el grupo colaborador

En el caso de que el grupo colaborador proporcione archivos específicos adicionales para realizar el análisis funcional, se procederá de la siguiente forma:

- Análisis funcional RNA: se procederá a copiar los archivos proporcionados por el grupo en la carpeta DEG\_Workflow. Seguidamente, en config\_daemon, en la variable 'export custom\_nomenclature', se añaden dichos archivos separados por comas (y sin espacios):

```
1      export custom_nomenclature="$CODE_PATH/c3.tft.v2023.2.Hs.entrez  
.gmt,$CODE_PATH/c4.all.v2023.2.Hs.entrez.gmt,$CODE_PATH/c6.all.  
v2023.2.Hs.entrez.gmt,$CODE_PATH/h.all.v2023.2.Hs.entrez.gmt"
```

- Análisis funcional coRmiT: en este caso, recordando que los archivos proporcionados se encuentran en la carpeta de análisis de RNA, se procede a configurar config\_daemon modificando el parámetro 'Add\_opt\_enr':

```
1  export Add\_opt\_enr="$CODE_PATH/c3.tft.v2023.2.Hs.entrez.gmt,  
$CODE_PATH/c4.all.v2023.2.Hs.entrez.gmt,$CODE_PATH/c6.all.v2023  
.2.Hs.entrez.gmt,$CODE_PATH/h.all.v2023.2.Hs.entrez.gmt"
```

*Nota:* en CODE\_PATH, se añade la ruta de la carpeta de análisis de RNA donde se encuentran los archivos.

### 19.0.3 Rescatar gen que ha sido excluido por el filtro de cpm

En el caso de que se tenga constancia de que un gen ha sido validado experimentalmente y, sin embargo, no aparezca en nuestro análisis, puede existir la posibilidad de que el filtro de cpm haya sido demasiado estricto. Lo primero que se comprueba es el estado del gen, mediante su código ENSEMBL, en Common\_results/hunter\_results\_table.txt. Mediante esta comprobación, se puede corroborar si el gen aparece como FILTERED\_OUT. Seguidamente, se busca el gen en filtered\_counts\_data.txt, archivo en el que aparecen aquellos genes que han pasado el filtro de expresión. En este caso, dicho gen no debería aparecer. Por último, se hace un grep del gen en selected\_counts para ver sus niveles de expresión.

Una vez realizada la comprobación, se procede a disminuir el filtro de cpm (parámetro min\_reads), el cual está a 2 cpm por defecto, hasta que se rescate dicho gen.