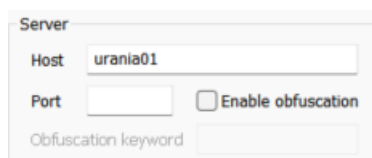


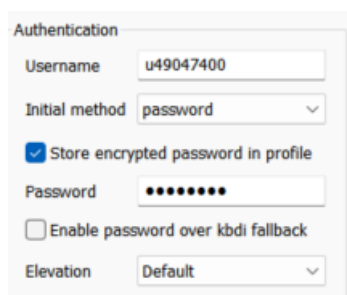
# Protocolo de obtención de alineamientos de péptidos a partir de reads

## Paso 1: Instalación y conexión de Bitvise SSH Client

En el apartado “Server” debemos rellenar el campo de host con “urania01”.



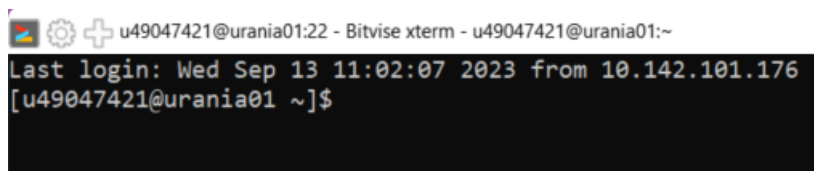
En el apartado “Authentication” debemos introducir nuestro usuario y contraseña.



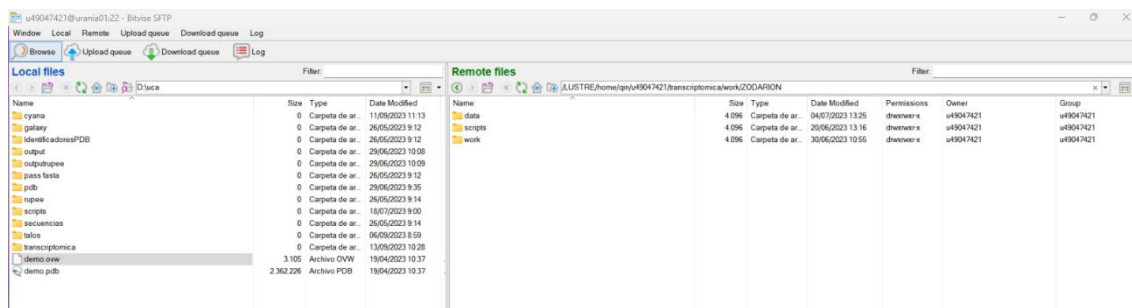
Luego pulsamos el botón “Log in” para conectarnos.

Una vez conectado, aparecen dos botones a la izquierda: “New terminal console” y “New SFTP window”.

Desde la consola podremos navegar a través del sistema de ficheros del clúster utilizando los comandos de Linux.



Una opción más cómoda es utilizar el servicio SFTP, donde a la izquierda tendremos el sistema de archivos de nuestro PC y a la derecha el del clúster. Podremos navegar a través de ellos y arrastrar archivos de un lado a otro para copiarlos, crear carpetas, etc.



Podemos modificar directamente los ficheros desde aquí abriéndolos con un editor de texto (Notepad++) o un editor de código (Visual Studio Code).

Para ejecutar los scripts debemos hacerlo desde la consola.

Utilizamos el comando “cd nombreDelDirectorio” para acceder a un directorio. Con el comando “cd ..” volvemos atrás.

Visualizamos el contenido del directorio con el comando “ls”.

Ejecutamos el script mediante el comando “sbatch nombreDelScript”.

Para comprobar el estado de la ejecución, utilizamos el comando “squeue”. Aparecerá una lista de los trabajos que se están ejecutando o están pendientes de hacerlo. Cada trabajo tiene asociado un ID.

Para cancelar la ejecución de un script debemos usar el comando “scancel” seguido del ID correspondiente.

## **Paso 2: Preparación del directorio de trabajo**

Abre tanto la ventana SFTP como la consola, ya que se trabajará desde ambos lados.

Dentro de tu directorio personal del clúster, crea un directorio llamado “transcriptomica” si aún no lo tienes:

`mkdir transcriptomica` (También puedes crear carpetas desde la ventana SFTP)

Entramos en él:

```
cd transcriptomica
```

Copia el directorio “template” con el siguiente comando:

```
cp -r /LUSTRE/home/qin/u49047421/transcriptomica/transcripto-  
filter/template .
```

Nota: el punto “.” final de la línea anterior sirve para indicar el directorio actual, es decir, a donde se copiará “template”.

Este directorio es una plantilla de trabajo preparada para ejecutar todos los scripts.

Cambia el nombre del directorio por otro más adecuado para tu experimento:

```
mv template experimento1
```

Entramos dentro:

```
cd experimento1
```



```
u49047421@urania01:22 - Bitwise xterm - u49047421@urania01:~/protocolo/transcriptimica/experimento1
[u49047421@urania01 protocolo]$ mkdir transcriptimica
[u49047421@urania01 protocolo]$ cd transcriptimica/
[u49047421@urania01 transcriptimica]$ cp -r /LUSTRE/home/qin/u49047421/transcriptomica/template .
[u49047421@urania01 transcriptimica]$ ls
template
[u49047421@urania01 transcriptimica]$ mv template experimento1
[u49047421@urania01 transcriptimica]$ ls
experimento1
[u49047421@urania01 transcriptimica]$ cd experimento1/
[u49047421@urania01 experimento1]$
```

Copia al directorio “reads” los dos ficheros de partida procedentes de la secuenciación. Estos ficheros tienen un formato fastq, aunque probablemente estén comprimidos. No es necesario descomprimirlos.



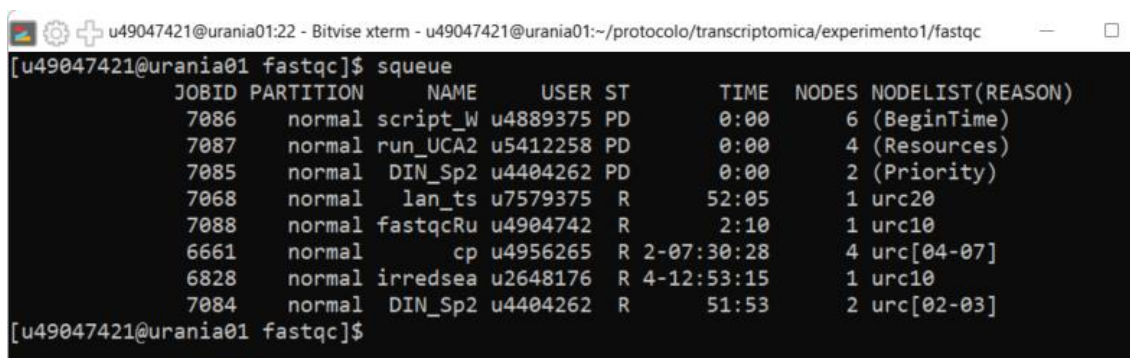
Name	Size	Type	Date Modified
22ID00797_S97_R1_001.fastq.gz	4.484.405.7...	Archivo WinR...	24/05/2023 16:45
22ID00797_S97_R2_001.fastq.gz	4.653.675.7...	Archivo WinR...	24/05/2023 16:47

### Paso 3: Análisis de calidad con FastQC

Entra en el directorio “fastqc” y ejecuta el script con el comando “sbatch” pasando como argumentos los dos ficheros de partida.

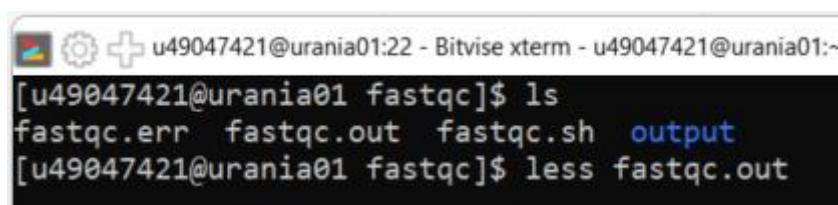
```
[u49047421@urania01 fastqc]$ ls
fastqc.sh
[u49047421@urania01 fastqc]$ sbatch fastqc.sh ../reads/22ID00797_S97_R1_001.fastq.gz ../reads/22ID00797_S97_R2_001.fastq.gz
Submitted batch job 7088
[u49047421@urania01 fastqc]$
```

Con el comando “squeue” puedes comprobar el estado de la ejecución del script desde la cola de trabajo del clúster.

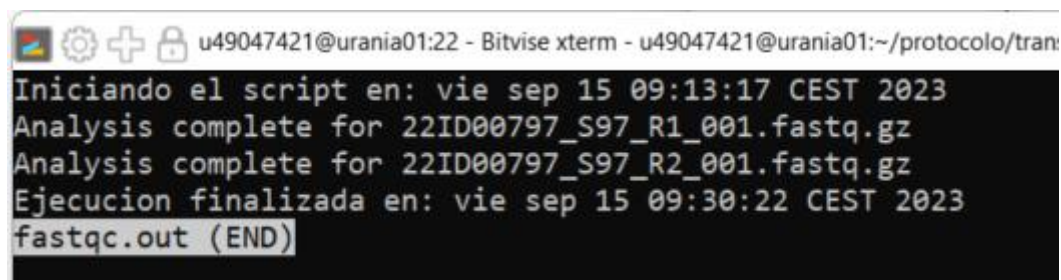


```
[u49047421@urania01 fastqc]$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
7086 normal script_W u4889375 PD 0:00 6 (BeginTime)
7087 normal run_UCA2 u5412258 PD 0:00 4 (Resources)
7085 normal DIN_Sp2 u4404262 PD 0:00 2 (Priority)
7068 normal lan_ts u7579375 R 52:05 1 urc20
7088 normal fastqcRu u4904742 R 2:10 1 urc10
6661 normal cp u4956265 R 2-07:30:28 4 urc[04-07]
6828 normal irredsea u2648176 R 4-12:53:15 1 urc10
7084 normal DIN_Sp2 u4404262 R 51:53 2 urc[02-03]
[u49047421@urania01 fastqc]$
```

Para comprobar si la ejecución ha terminado, además de consultar la cola de trabajo del clúster, puedes visualizar con el comando “less” el fichero “fastqc.out” que se ha generado.



```
[u49047421@urania01 fastqc]$ ls
fastqc.err fastqc.out fastqc.sh output
[u49047421@urania01 fastqc]$ less fastqc.out
```



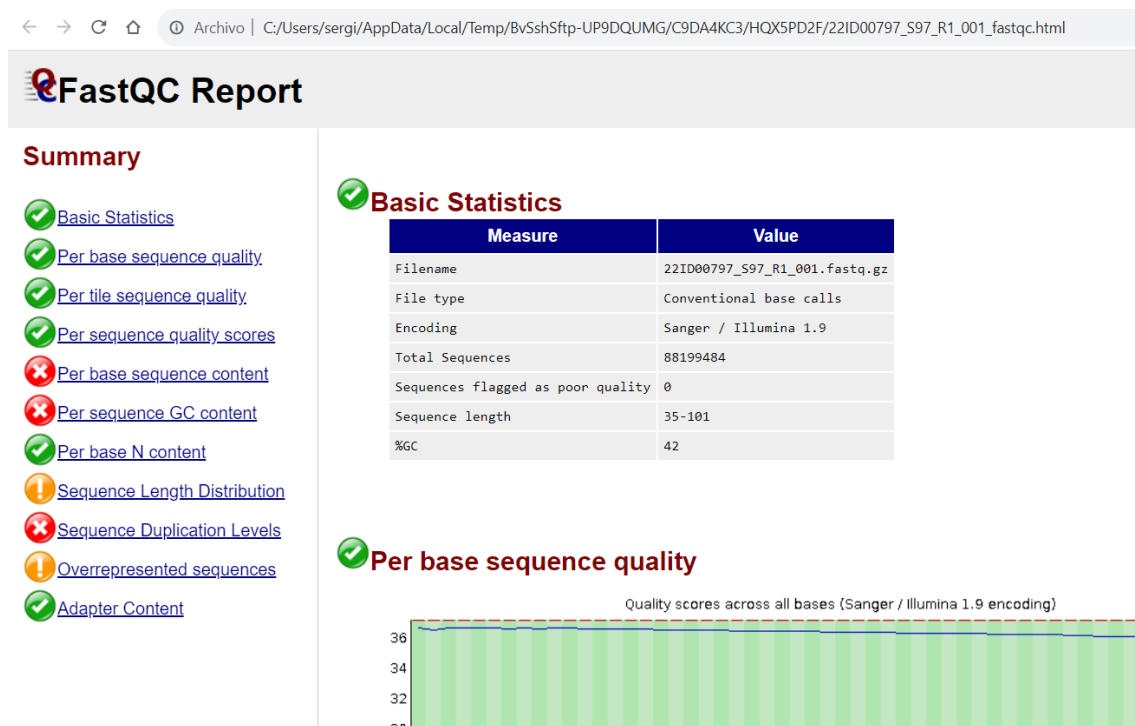
```
Iniciando el script en: vie sep 15 09:13:17 CEST 2023
Analysis complete for 22ID00797_S97_R1_001.fastq.gz
Analysis complete for 22ID00797_S97_R2_001.fastq.gz
Ejecucion finalizada en: vie sep 15 09:30:22 CEST 2023
fastqc.out (END)
```

Para salir pulsa la tecla q.

En el directorio “output” generado aparecerán los ficheros con el análisis de calidad de los reads.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1
[u49047421@urania01 fastqc]$ ls
fastqc.err fastqc.out fastqc.sh output
[u49047421@urania01 fastqc]$ cd output/
[u49047421@urania01 output]$ ls
22ID00797_S97_R1_001_fastqc.html 22ID00797_S97_R2_001_fastqc.html
22ID00797_S97_R1_001_fastqc.zip 22ID00797_S97_R2_001_fastqc.zip
[u49047421@urania01 output]$
```

Puedes descargarlos desde la ventana SFTP o abrirlos directamente.



#### Paso 4: Ensamblaje con Trinity

Una vez comprobada la calidad de los reads, el siguiente paso es el ensamblaje para obtener un fichero fasta con todas las secuencias.

Entramos en el directorio “trinity” y ejecutamos con el comando “sbatch” el script “trinity.sh” pasándole como argumentos los dos ficheros de partida del directorio “reads”.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1/trinity
[u49047421@urania01 trinity]$ sbatch trinity.sh ../reads/22ID00797_S97_R1_001.fastq.gz ../reads/22ID00797_S97_R2_001.fastq.gz
Submitted batch job 7090
[u49047421@urania01 trinity]$
```

Se generarán los ficheros “trinity.err” y “trinity.out” que podrás visualizar para hacer un seguimiento de la ejecución.

Al finalizar la ejecución satisfactoriamente se creará un fichero con los resultados llamado “trinity\_out\_dir.Trinity.fasta”.

## Paso 5: Control de calidad del ensamblaje con BUSCO

Para ejecutar BUSCO desde un ambiente offline necesitamos una base de datos de linaje como referencia. En nuestro caso: metazoa\_odb10.

Puedes descargarla manualmente desde <https://busco-data.ezlab.org/v5/data/lineages/> y guardarla en un nuevo directorio dentro de “transcriptomica”.

Por ejemplo, en “transcriptomica/data/BUSCO\_DB/ metazoa\_odb10”

Este fichero normalmente será común para todos los experimentos. Existe una copia en:

/LUSTRE/home/qin/u49047421/transcriptomica/data/BUSCO\_DB/metazoa\_odb10

Esta copia ya está referenciada dentro del script “busco.sh”. En caso de querer utilizar otra, es necesario modificar el script.

Accede al directorio “busco” y dentro de él ejecuta con “sbatch” el script “busco.sh” pasando como argumento el fichero fasta obtenido con Trinity.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1/busco
[u49047421@urania01 busco]$ sbatch busco.sh ../trinity/trinity_out_dir.Trinity.fasta
Submitted batch job 7092
[u49047421@urania01 busco]$
```

Tras la ejecución se creará un directorio llamado “busco\_output”.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1/busco/
[u49047421@urania01 busco]$ ls
busco_downloads  busco.err  busco.out  busco_output  busco.sh
[u49047421@urania01 busco]$ cd busco_output/
[u49047421@urania01 busco_output]$ ls
logs                short_summary.specific.metazoa_odb10.busco_output.json
run_metazoa_odb10  short_summary.specific.metazoa_odb10.busco_output.txt
[u49047421@urania01 busco_output]$
```

Dentro de él podemos consultar el *short summary* con el comando “less” para visualizar los resultados.

```
***** Results: *****

C:76.9%[S:56.1%,D:20.8%],F:10.1%,M:13.0%,n:954
733    Complete BUSCOs (C)
535    Complete and single-copy BUSCOs (S)
198    Complete and duplicated BUSCOs (D)
96     Fragmented BUSCOs (F)
125    Missing BUSCOs (M)
954    Total BUSCO groups searched
```



## Paso 6: BLASTX

El siguiente paso es utilizar Blastx para comparar las secuencias con una base de datos de péptidos.

Si se trata de una nueva base de datos, primero necesitamos construirla. Para ello necesitamos guardar el fichero fasta con la base de datos en un directorio común para todos los experimentos.

Por ejemplo, si vamos a utilizar el fichero “CTX-May23bis\_completeseq.fasta” como base de datos. Podemos guardarlo en:

transcriptomica/data/BLAST\_DB/CTX\_DB/CTX-May23bis\_completeseq/ CTX-May23bis\_completeseq.fasta

Para construir la base de datos copia el script “makeblastdb.sh” que se encuentra en el directorio “blastx” al directorio donde se encuentra el fichero fasta.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1/blastx
[u49047421@urania01 blastx]$ ls
blastx.sh  makeblastdb.sh
[u49047421@urania01 blastx]$ cp makeblastdb.sh ../../data/BLAST_DB/CTX_DB/CTX-May23bis_completeseq/
q/
[u49047421@urania01 blastx]$
```

Una vez copiado y situado dentro del directorio, ejecuta el script con “sbatch” pasándole el fichero fasta como argumento.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/data/BLAST_DB/CTX_DB/CTX-May23bis_completeseq
[u49047421@urania01 CTX-May23bis_completeseq]$ ls
CTX-May23bis_completeseq.fasta  makeblastdb.sh
[u49047421@urania01 CTX-May23bis_completeseq]$ sbatch makeblastdb.sh CTX-May23bis_completeseq.fasta
Submitted batch job 7094
[u49047421@urania01 CTX-May23bis_completeseq]$
```

Se generarán diferentes ficheros con los que Blastx trabajará automáticamente.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/data/BLAST_DB/CTX_DB/CTX-May23bis_completeseq
[u49047421@urania01 CTX-May23bis_completeseq]$ ls
blastdb.err          CTX-May23bis_completeseq.fasta.pjs
blastdb.out          CTX-May23bis_completeseq.fasta.pot
CTX-May23bis_completeseq.fasta  CTX-May23bis_completeseq.fasta.psq
CTX-May23bis_completeseq.fasta.pdb  CTX-May23bis_completeseq.fasta.ptf
CTX-May23bis_completeseq.fasta.phr  CTX-May23bis_completeseq.fasta.pto
CTX-May23bis_completeseq.fasta.pin  makeblastdb.sh
```

Una vez que ya disponemos de una base de datos construida, podemos entrar en el directorio “blastx” y ejecutar el script “blastx.sh” con el comando “sbatch”. Es necesario pasarle 2 argumentos en el orden correcto.

1. QUERY: Fichero fasta obtenido en Trinity.
2. DB: Fichero fasta de la base de datos de péptidos previamente construida. La base de datos mencionada anteriormente ya se encuentra construida en:  
/LUSTRE/home/qin/u49047421/transcriptomica/data/BLAST\_DB/CTX\_DB/CTX-May23bis\_completeseq/CTX-May23bis\_completeseq.fasta

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1/blastx
[u49047421@urania01 blastx]$ sbatch blastx.sh ../trinity/trinity_out_dir.Trinity.fasta /LUSTRE/home/qin/u49047421/transcriptomica/data/BLAST_DB/CTX_DB/CTX-May23bis_completeseq/CTX-May23bis_completeseq.fasta
Submitted batch job 7096
[u49047421@urania01 blastx]$
```

Tras la ejecución se generará un fichero “blastx\_out.csv”.

### Paso 7: Extracción de alineamientos

Accede al directorio “alignments” y ejecuta con “sbatch” el script “alignments.sh”. Es necesario pasarle 3 argumentos en el orden correcto.

1. BLAST\_CSV: Fichero csv obtenido en Blastx.
2. TRINITY\_FASTA: Fichero fasta obtenido en Trinity.
3. DB: Fichero fasta de la base de datos de péptidos previamente construida.

```
u49047421@urania01:22 - Bitvise xterm - u49047421@urania01:~/protocolo/transcriptomica/experimento1/alignments
[u49047421@urania01 alignments]$ sbatch alignments.sh ../blastx/blastx_out.csv ../trinity/trinity_out_dir.Trinity.fasta /LUSTRE/home/qin/u49047421/transcriptomica/data/BLAST_DB/CTX_DB/CTX-May23bis_completeseq/CTX-May23bis_completeseq.fasta
Submitted batch job 7098
[u49047421@urania01 alignments]$
```

Para verificar que la ejecución ha finalizado correctamente consulta el fichero “alignments.out”.

Se creará un directorio llamado “Alineamientos\_mafft” en donde se encontrarán todos los ficheros con la extensión “mafft.fasta” que contendrán los alineamientos detectados.

También puede ser útil el fichero “extracted\_sequences.fasta”.

Descarga los resultados desde la ventana SFTP a tu PC para continuar trabajando con los alineamientos detectados.

### Paso 8: Filtrado de los alineamientos

Accede al directorio “curation\_filter”. Antes de ejecutar el script es necesario revisar los parámetros de configuración del filtro. Éstos se encuentran en el fichero “config.txt” dentro del directorio “python\_scripts”. Para modificar la configuración puedes editar el fichero o sustituirlo por otro con el mismo nombre y formato.

Una vez revisado los parámetros, ejecuta con “sbatch” el script “filter.sh”, no es necesario indicar ningún argumento. La ejecución puede durar alrededor de 20 minutos. Tras finalizar accede al directorio “Alineamientos\_filtrados”.

Dentro de este directorio se encuentran todos los alineamientos que han pasado el filtro y un fichero llamado “informe.tsv”. Puedes descargar este fichero para consultar toda la información del filtrado.

El fichero contiene las siguientes columnas:

- **Seq\_file:** El número del fichero original mafft.fasta analizado.
- **Frame\_ID:** El identificador del marco de lectura analizado.
- **Ref\_ID:** El identificador de la secuencia de la base de datos con la que se ha intentado alinear el marco de lectura.

- **Pasa\_filtro:** Indica si el alineamiento entre las dos secuencias comparadas ha pasado el filtro.
- **Hay\_segmento\_de\_subsecuencias\_validas:** Criterio de filtrado 1 (debe ser "True"). Hace referencia a que existen partea de las dos secuencias que alinean correctamente.
- **longitud\_minima\_total\_subseqs\_superada:** Criterio de filtrado 2 (debe ser "True"). Hace referencia a la cantidad mínima de aminoácidos que deben alinearse.
- **ratio\_minimo\_longitud\_superado:** Criterio de filtrado 3 (debe ser "True"). Hace referencia al porcentaje mínimo que la parte alineada debe abarcar en la secuencia de la base de datos.
- **Stop\_codon\_en\_mitad\_de\_dos\_segmentos\_subsecuencias\_validas:** Criterio de filtrado 4 (debe ser "False"). Hace referencia a la presencia de un codón de parada en mitad de la secuencia analizada.
- **Vector\_alineamiento:** Representación binaria del alineamiento entre las dos secuencias: 1(aa alineado), 0 (aa no alineado), \* (codón de parada).

### **Paso 9: Filtro de Metionina y clasificación de alineamientos**

Accede al directorio "metionine\_filter" y ejecuta con "sbatch" el script "filter.sh"

Una vez finalizada la ejecución, en el directorio "resultados" se encontrarán los alineamientos clasificados.

- **Alineamientos\_Perfectos:** La metionina inicial en las secuencias de referencia coincide con la metionina inicial del frame.
- **Alineamientos\_Limpios:** Se trata de los alineamientos perfectos ya formateados sin guiones, comenzando por la metionina inicial y finalizando en el codón de stop correspondiente.
- **Alineamientos\_M\_Previa:** La metionina inicial en las secuencias de referencia coincide con la metionina inicial del frame, pero existe una metionina anterior que podría ser el verdadero inicio de la secuencia.
- **Alineamientos\_Multiframe:** Son los ficheros en donde más de un frame alinea con las secuencias de referencia.
- **Alineamientos\_Revision\_Manual:** Necesitan ser revisados manualmente.



### Modo automático: Ejecución de autolauncher

Todo el flujo de trabajo descrito anteriormente se puede realizar de forma automática. Existen dos puntos de partida:

- Punto de partida inicial: Indicando los dos **reads** y la **base de datos de referencia**.
- Punto de partida después del ensamblaje: Indicando el **fichero fasta ensamblado** con Trinity y la **base de datos de referencia**.

Se realizan los pasos 1, 2 y 3 descritos anteriormente para preparar nuestro directorio de trabajo con los dos reads.

Si disponemos del fichero fasta ensamblado debemos copiarlo al directorio trinity

Desde el directorio de trabajo ejecutamos “`sbatch autolauncher.sh`” seguido de las opciones con los argumentos necesarios.

Las opciones son las siguientes:

```
--r1 ruta/del/read1  
--r2 ruta/del/read2  
--bd ruta/de/la/base/de/datos  
--trinity ruta/del/fichero/ensamblado
```

Ejemplos de uso:

```
@urania01 imperialis_optimizado]$ sbatch autolauncher.sh --r1 reads/SRR2609542_1.fastq.gz  
--r2 reads/SRR2609542_2.fastq.gz --db /LUSTRE/home/qin/u49047421/transcriptomica/data/BLAST_DB/CTX_D  
B/CTX-Oct23_cleanseq/CTX-Oct23_cleanseq.fasta
```

```
sbatch autolauncher.sh --db /LUSTRE/home/qin/u49047421/t  
ranscriptomica/data/BLAST_DB/ZODARION_DB/Zodarion_new/Zodarion_new.fasta --trinity trinity/trinity_o  
ut.fasta
```

Los ficheros “t-filter.err” y “t-filter.out” contienen detalles de la ejecución del script.