# RESUMEN BIOINFORMATICA 1 2021- primer parcial

## COMANDOS PARA LA TERMINAL DE LINUX (Clases 16, 19, 23 y 26 agosto)

### Pwd

* Donde estoy parado desde la barra de home, en que directorio, etc. Ejemplo: /home/jovyan/Bioinfo1\_2021/Material\_Alineamientos

### **mkdir**

* Lo que hace es crear un nuevo directorio (carpeta) con el nombre que yo le coloque, si quiero poner un nombre con espacio debo colocar el guion bajo. Ejemplo: mkdir creando\_carpeta

### **ls**

* Muestra dentro de la carpeta donde estoy parado que es lo que hay. A diferencia de pwd que te muestra con barras, ls lo que hace es como mostrarte una interfas.
* Completar: Con el tabulador completo el directorio (carpeta) o el archivo que quiero.
* Recuperar código: con las flechitas
* Condiciones ( | ): La barra lo que me permite hacer es poner más de una condición.

### tree

* me muestra todos los directorios y carpetas como si fuera un esquema de árbol.

### **Touch**

* Poniendo el comando y el nombre del archivo que queremos crear, creamos el archivo.

### mv

* Este comando sirve para mover archivos que se encuentren dentro de un directorio. Por ejemplo, dentro de Bioinfo1\_2021 yo puedo mover un archivo que se encuentra en Materiales\_Alineamientos a BRH. Para esto el código es:

**mv** [Archivo a mover] [espacio] /home/jovyan/Bioinfo1\_2021/BRH

Este comando si se aplica en la misma carpeta lo que me sirve también es para cambiar el nombre de un archivo. Ya que pongo el comando un nombre distinto al final.

### LS –L

* me muestra los accesos de cada uno de los archivos. Si le agregas \*palabra\* busca específicamente los archivos q la contengan. Los Permisos

Con chmod

* +x o -x [Archivo] lo que estamos haciendo es habilitar los permisos del archivo (+x) o denegar los permisos (-x). Tenemos 3 niveles de permisos.

1. Solo lectura (r)
2. Escritura (w) además de leerlo puedo escribirlo
3. Ejecución (x) permite que yo pueda ejecutar ese programa. Esta esto ya que quiero que ciertas personas ejecuten el programa y otras no, o lo que sea.

##### Borrar

Rmdir BORRAMOS directorios si están vacíos sino te avisa

rm es para BORRAR archivos.

##### Códigos para visualizar archivos:

### head

* me muestra las primeras 10 por defecto, puedo ponerle -2 o -4 o lo que sea para cambiar las que quiero que me muestre

### **tail**

* Lo mismo pasa con este código, pero me muestra desde las ultimas.

### **cat**

* te muestra todo el archivo.

### more

* y el nombre del archivo, me lo muestra al archivo en páginas moviéndome con las flechas.

### grep

* Comando que me permite buscar en un archivo o en parte de un archivo en base a cierto patrón. Ejemplos:

1. head p53.fas | grep">“ Estoy buscando en las primeras 10 líneas del archivo p53. Fas picos.
2. cat p53.fas | grep ">" Lo mismo que el anterior, pero estoy buscando en todo el archivo.

grep -v 🡪 Tiene la misma función del comando grep solo que me filtra por la negativa. Por ejemplo:

1. head p53.fas |grep -v "W" Dentro de las 10 primeras líneas del archivo p53.fas estoy buscando las líneas que NO TIENEN el aminoácido W

grep -c 🡪 Me dice el NUMERO que hay de tal condición.

1. cat p53.fas | grep -c ">" Dentro del archivo completo p53.fas cuantos picos tenemos.

grep -n 🡪 Me dice en que numero de líneas se encuentra la condición que estoy buscando. Ejemplo

1. cat hb.fas | grep -n ">"

### wc

* Es un comando que nos permite ver cuantas líneas, palabras y caracteres (en ese orden) tenemos. Podemos acoplarlo a una condición para que nos dé aún más información.

### sed

* Me permite sustituir algún carácter por otro. Tengo que tener en cuenta el formato que es: [comilla] [ s ] / [carácter a cambiar] / [ nuevo carácter] / [comilla]. Ejemplos

1. head p53.fas | sed ‘s/>/\*/’ Estoy cambiando en las primeras 10 líneas del archivo p53.fas los picos por unos asteriscos.

También me permite ELIMINAR cierta columna. Por ejemplo, si ponemos head tabla | sed 2d eliminamos la columna 2.

### awk

🡪 Nos permite seleccionar COLUMNAS en base a algún criterio. Podemos seleccionar alguna columna entera y printearlas con alguna condición. Ejemplos:

1. head tabla | awk ‘{print $2}’ Acá estamos printeando toda la columna 2
2. head tabla | awk ‘$3>0.5’ Acá estamos printeando de la columna 3 aquellos valores mayores a 0.5
3. head tabla | awk '$4 == "C"' Aca estamos printeando de la cuarta columna aquellos valores que sean iguales a la C.

Tener en cuenta además que si se quiere poner mas de unas condiciones podemos poner el &&

### cat

* Nos permite escribir archivos de texto, recordando que se crean con touch. Luego tenemos que poner un > si es la primera vez que estamos escribiendo. Pero tener cuidado que luego con >> es que tenemos que hacer para sobrescribir.

Recordar: cat [espacio] [>> o >] [espacio] [Archivo]

### cut

* Separa por columnas al poner por ejemplo cat tabla | cut -f1 te corta la columna 1 por defecto, por tabuladores. Sin embargo, si yo quiero determinar otras referencias apara que haga el corte le pongo:

Cat tabla | cut -d ”lo que sea” -f1.

Determino en primera instancia que carácter divide las columnas y luego corto. Por ejemplo, si pongo -d7 me cortara cuando encuentre un 7 y en ese momento elimina lo anterior.

### sort

* Te sirve para ordenar en base a la columna escogiada. Si son números ordena de mayor a menor y así el criterio.

Con los numeros decimales debemos agregar al final un -d

También puede ser que para los números los ordena de no forma correcta y por tanto debe utilizar -n

Ejemplos: head tabla | sort -k2

### Uniq

* Dice que caracteres o letras hay en una lista, si se usa con –c tambien indica cuantas veces aparece cada uno de los caracteres

## ENTORNO Y LENGUAJE DE PROGRAMACION R. (Clases 30/8 y 2/9)

##### VARIABLES DENTRO DE R:

Con **class** (variable) nos dice que tipo de variables estamos hablando.

1. Vectores.

Pueden ser numéricos o de letras. Ej: vector=c("a","e","i","o","u") o vnum=c(1,2,3,4,5,6). Los vectores numéricos, a diferencia de los con letras, no necesitan de comillas. Podemos aplicar operaciones matemáticas sobre cada uno de los elementos del vector ( +, -, \*, /).

Podemos invocar cierto elemento dentro del vector [x] o poner un intervalo de elementos [x:y]. Tambien si ponemos vector[x]=z estamos cambiando ese elemento dentro del vector. Si no queremos ver el primer elemento por ejemplo, vector[-1]. Tambien puedo colocar un x mayor al numero de elementos en el vector, lo que agrandara el mismo.

Podemos aplicar sobre el vector la función length.

Vectores Predefinidos:

|  |  |  |  |
| --- | --- | --- | --- |
| Vector de letras | Vector de aminoácidos | Idem a() con nombres |  |
| letters | a() | aaa() |  |

1. Matrices.

En las matrices lo que hay que haces es poner un intervalo numérico que vallan a ser los elementos de la matriz. Luego podemos especificar o el numero de filas, o el numero de columnas, o ambos. Además, tambien agregando byrow = TRUE lo que estamos queriendo decir es que queremos que los números se ordenen de izquierda a derecha. Poniendo tab igual me va diciendo todas esas cosas.

A=matrix(1:16, nrow = 4, ncol = 2, byrow = TRUE) . Aqui temenos un ejemplo.

Sobre las matrices se pueden hacer operaciones matemáticas tambien. Además, podemos calcular la transpuesta como t(A) en este caso.

1. Data Frame.

Consiste en generar una tabla con filas y columnas con datos aportados por el usuario. Debo poner data.frame = y con tab voy poniendo las opciones. Ejemplo:

B=data.frame("primera"=1:5,"segunda"=letters[1:5],"tercera"=letters[6:10])

1. Listas.

Comenzar diciendo que cuando leemos una secuencia FASTA esta siempre es una lista. Donde cada elemento de la lista está conformado por los cdrs o secuencias codicantes.

##### Para leer una secuencia FASTA

library(seqinr)

#Primero témenos que abrir el paquete previamente instalado seqinr (install.packages)

Fasta = read.fasta("ARCHIVO.fasta")

# Luego témenos que invocar el código read.fasta y leer el archivo que debe estar en formato FASTA.

##### Función length(lista)

La función length me da el largo. Pero el largo de muchas cosas puede darme

1. Si pongo length(fasta) me dará el número de elementos de la lista “Fasta”, en este caso, el numero de picos, el numero de cdrs o secuencias codificantes.
2. Si por otro lado invoco uno de los elementos de esa lista con [[ ]] y a la vez aplico la función length lo que estoy haciendo es contar las bases de ese elemento (secuencia codificante) que escogí,length(fasta[[3]]).

##### Función sapply(lista, función)

Sapply hace como si fuera un loop un for i in range en pythone, lo que hace es aplicar una función determinada a todos los objetos de la lista. Por ejemplo, podríamos saber el largo de todas las secuencias codificantes. Todos\_los\_largos = sapply(fasta, length).

##### Función GC(lista[x])

Esta función se encuentra dentro del paquete seqinr y lo que me permite es calcular el %GC de una secuencia nucleotídica. Para esto, si tengo una lista con muchas secuencias debo dar el elemento.

Ejemplos: 1) PorcentajeGC\_Sec1 = GC(fasta[[1]])

2) GCs = sapply(fasta, GC)

Ademas podemos tener los %GCs de los 1°, 2° y 3° nucelotidos invocando de forma similar a la GC las funciones; GC1, GC2, GC3

Funciones que se pueden aplicar a **listas numéricas**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Media | Resumen | Max y Min | suma |  |
| mean(lista) | summary(lista) | Min(lista) Max(lista) | sum(lista) |  |

Ejemplo bueno: summary(sapply(fasta, GC)) Como varían los %GCs de las secuencias

##### Función unlist(lista)

Esta función lo que hace es compactar toda la información de los elementos de la lista en 1 solo, por lo que podemos guardar la información en una variable y luego aplicarle otras funciones. Ej:

Todo\_junto = unlist(fasta)

length(Todo\_junto)

##### Función translate(lista[[x]])

Esta es otra función de seqinr que lo que hace es traducir SECUENCIAS, por tanto, tenemos que mandarle con la lista el elemento que queremos traducir o poner sapply para traducir todo o usar unlist… Ejemplos:

1. sec\_trad = sapply(fasta, translate)
2. f\_1\_traducida = translate(fasta[[1]])

##### Función table(secuencia traducida)

Acoplada con la función anterior de translate, esta función de seqinr funciona sobre secuencias traducidas, y lo que hace es devolverme una lista con la cantidad de cada uno de los aminoácidos presentes en la secuencia.

##### Función count( secuencia traducida)

También una función dentro de lo que es el paquete seqinr. Me permite contar cuantos aminoácidos hay de cada uno como la función anterior table() pero con más parámetros.

Count(seq = , wordsize = , start = , by = , freq =, alphabet = )

El seq es la secuencia que estoy viendo, el wordsize pongo cada cuanto quiero que cuente, lo otro son otras cosas. El alphabet pongo a() que es un vector con todos los aminoacidos. con el -1 pongo que no quiero el stop.

## ALINEACIONES

##### DOTBLOT

**library(seqinr)**

# Vamos a levantar dos secuencias que están ya, en R, como muestra

**aafile <- system.file("sequences/seqAA.fasta", package ="seqinr**")

**dnafile <- system.file("sequences/malM.fasta", package ="seqinr")**

**AA<-read.fasta(aafile)**

**NN<-read.fasta(dnafile)**

**aa <- AA[[1]]**

**nn <- NN[[1]]**

**?dotPlot** #ayuda

**dotPlot(a,a)**

**dotPlot(n,n)**

**Para mejorar ruido** 🡪 bajar tamaño de w y s (como hacer zoom)

**DNA** 🡪 ¼ posibilidades coincidencia aleatorio = más ruido

**PROTEINA** 🡪 1/20 posibilidades coincidencia aleatoria = menos ruido

##### https://lh5.googleusercontent.com/AkARtiGbMvs7AqncAf_bRDjLXa615gX1yVZ7CJg0SAkhZRCpSEMZ2ir143VAKlMPH9NyoGtXvO_ZFI5wQwxRMFIyQj4N9u0pCoUZFUg-btCbKALengs87eJ-b1DuWuP_FM7AMs3I=s0Global: Needle-Wurnsch

1. Ver match (1) o mismatch (-1)
2. Sumar a 1/-1 el número de arriba, diagonal y costado
3. La opción que de score mayor es la dirección q gana

##### https://lh4.googleusercontent.com/VM0wg0hIYNsGMxvX0BmF8kSmK9rHrMPhFLlR9QGae-uIzxj5S5ZFpSIhCExGNbsAHGA9cvOn9R9ChTufdLTgt1OVcp_kuhhsnTVAxjPiNcv1Jcsgtj2gGGlnyxtqcocuqMj5FfOp=s0Local: Smith – Waterman

\*En este no hay negativos, se usa cero en esos casos.

##### **En R**:

library(Biostrings)

library(seqninr)

sec1 <- read.fasta(‘path del fasta’, as.string=T)

sec2 <- read.fasta(‘path del fasta’, as.string=T)

sec1 <- DNAStringSet(as.character(sec1[[1]])

sec2 <- DNAStringSet(as.character(sec2[[1]])

mat <- nucleotideSubstitutionMatrix(match=1, mismatch=-1,base0nly=TRUE)

ali1 <- pairwiseAligment(patter =sec1, subject=sec2, type=’global’ o ‘local’, substitutionMatrix=mat, gap0pening=3, gapExtension=2)

##### En terminal LOCAL (Emboss)

jovyan@jupyter-238275:~/Bioinfo1\_2021/Material\_Alineamientos$ water hb.fas p53.fas

Smith-Waterman local alignment of sequences

Gap opening penalty [10.0]:

Gap extension penalty [0.5]:

Output alignment [human0.water]: alineamiento\_prueba

jovyan@jupyter-238275:~/Bioinfo1\_2021/Material\_Alineamientos$ more alineamiento\_prueba

##### En terminal GLOBAL (Emboss)

jovyan@jupyter-238275:~/Bioinfo1\_2021/Material\_Alineamientos$ needle hb.fas p53.fasNeedleman-Wunsch global alignment of two sequences

Gap opening penalty [10.0]:

Gap extension penalty [0.5]:

Output alignment [human0.needle]: needle\_prueba

jovyan@jupyter-238275:~/Bioinfo1\_2021/Material\_Alineamientos$ more needle\_prueba

##### En terminal (Clustalw)

jovyan@jupyter-238275:~/Bioinfo1\_2021/Material\_Alineamientos$ clustalw hb.fas

\*Devuelve listas con scores

jovyan@jupyter-238275:~/Bioinfo1\_2021/Material\_Alineamientos$ clustalw

Your choice: 1

Enter the name of the sequence file : hb.fas

\*Devuelve lista con cantidad de aa

Your choice: 2

Your choice: 1

\*Luego de preguntarte nombre de archivo output

CLUSTAL 2.1 multiple sequence alignment \*te muestra las secuencias aa alineadas

## MATRICES DE SUBSTITUCIÓN

\*Para dale sentido biológico a la asignación de scores, según qué cambio de aa sea, y la distancia evolutiva entre ellos

##### PAM:

* tiene en cuenta tasas de mutaciones
* diferentes %
* % indica la cantidad de cambios cada 100 aa
* 100% cambios ni indica 0% identidad, puede haber varios cambios en una posición
* + identidad =PAM bajo 🡪 para proteínas muy similares (de humano xej) entre si

##### BLOSUM:

* Se ven frecuencias de sustitución en 500 alineaciones
* Número del nombre = distancia evolutiva
* Secuencia más similar= BLOSUM 🡪 para proteínas muy similares (de humano xej) entre s

## BLAST

1. Substituir nombres de fastas por unos mas fáciles

**library(seqinr)**

**fas=read.fasta('atp.fasta')**

**len=length(fas)**

**names(fas)**

**nmOriginales=names(fas)** #guarda los nombres originales

#de las seq de la lista fasta ingresada

**names=c()**

**for ( i in 1:len){names[i]=paste('Atp',i,sep='.')}**

**names**

#empiezo a crear los nuevos archivos con las seq con nombre simplificado

**write.fasta(fas,names=names,file.out = 'atp.cds')** #crea el fasta con los nuevos nombres

# puestos (en este ej: atp1,atp2...)

**faspep=lapply(fas,translate)** #traducir todas las seq en el fasta

**write.fasta(faspep, names=names, file.out = 'atp.pep')** #creo fasta con las seq traducidas

1. En terminal crear los archivos necesarios para correr el blast, dento de carpeta donde están los archivos fasta

**makeblastdb -in atp.cds -dbtype nucl -out atp.cds.base**

nombre archivo subject, a partir del que se crea la base de datos

tipo de base de datos que es, nucleótidos (nucl) o proteínas (prot)

como se llaman los archivos de salida (base por base de datos)

hace bases de datos de blast

1. Correr el blast (blastn –help es la ayuda del blast, podemos ver los outfmt)

**blastn -query secuencia.fasta -db a atp.cds.base -outfmt 6 -out tablast.secuencia-atp -num\_threads 2 -evalue 1e-5**

blast a usar (nucl contra nucl, prot contra prot, nucl contra prot o prot contra nucl)

query, sec a comparar

nombre base de datos de paso 2, no es el nombre en si, sino el sufijo q tienen todos

formato de salida:

* + - 6 =tabular, mas liso posible, pero no dice nombre de columnas
    - **-outfmt '6 std qlen slen',** opcion para personalizar las columnas
    - 3= muestra donde estas las alineaciones, en secuencia, comparandolas

nombre archivo salida

cuantos procesadores se van a usar

evalue, umbral, los que tienen más grandes no se toman

1. Se pueden realizar varias funciones para analizar la tabla y filtrarla
   * **C**ontar cuantas líneas hay

jovyan@jupyter-238275:~/Bioinfo1\_2021/ejemplo\_blast$ **wc tablast.secuencia-atp**

8 112 800 tablast.secuencia-atp

* + Para filtrar de la columna 3 cuales superan el 90% de identidad

jovyan@jupyter-238275:~/Bioinfo1\_2021/ejemplo\_blast$ **cat tablast.secuencia-atp | awk '$3 > 90' | wc -l**

2 28 200

* + Largo de alineamiento ($4), sea por lo menos el 90% del largo de una de las secuencias

jovyan@jupyter-238275:~/Bioinfo1\_2021/ejemplo\_blast$ **cat tablast.secuencia-atp | awk '{print $13,$14}'**

50784 1521

50784 1521

50784 1521

50784 1521

50784 1521

50784 1521

50784 1521

50784 1521

jovyan@jupyter-238275:~/Bioinfo1\_2021/ejemplo\_blast$ **cat tablast.secuencia-atp | awk '$4>0.9 \* $13' #nofunciono**

##### BLAST (BRH) en R: busqueda ortologos

1. Igual que el que se corre por terminal hay que simplificar nombres de archivos, esta vez de dos archivos de especies diferentes. Los de atp se crean de la manera que detalla arriba, los de secuencia.fasta, de la misma manera, cambiando el nombre de archivo, asi:

**library(seqinr)**

**fas=read.fasta('secuencia.fasta')**

**len=length(fas)**

**names(fas)**

**nmOriginales=names(fas)** #guarda los nombres originales

#de las seq de la lista fasta ingresada

**names=c()**

**for ( i in 1:len){names[i]=paste('Sec',i,sep='.')}**

**names**

#empiezo a crear los nuevos archivos con las seq con nombre simplificado

**write.fasta(fas,names=names,file.out = 'secuencia.cds')** #crea el fasta con los nuevos nombres #puestos (en este ej: atp1,atp2...)

**faspep=lapply(fas,translate)** #traducir todas las seq en el fasta

**write.fasta(faspep, names=names, file.out = 'sec.pep')** #creo fasta con las seq traducidas

1. En terminal, en este caso se corrió un blast con proteínas (.pep) y devuelve un archivo tblast.query.basedatos:

jovyan@jupyter-238275:~/Bioinfo1\_2021/ejemplo\_blast$ ./BRH-Copy1.bash atp.pepsec.pep

Building a new DB, current time: 10/18/2021 00:34:12

New DB name: /home/jovyan/Bioinfo1\_2021/ejemplo\_blast/sec.pep.base

New DB title: sec.pep

Sequence type: Protein

Keep MBits: T

Maximum file size: 1000000000B

Adding sequences from FASTA; added 59 sequences in 0.00880599 seconds.

Warning: [blastp] Examining 5 or more matches is recommended

Building a new DB, current time: 10/18/2021 00:34:23

New DB name: /home/jovyan/Bioinfo1\_2021/ejemplo\_blast/atp.pep.base

New DB title: atp.pep

Sequence type: Protein

Keep MBits: T

Maximum file size: 1000000000B

Adding sequences from FASTA; added 59 sequences in 0.00723696 seconds.

Warning: [blastp] Examining 5 or more matches is recommended

1. Para corer BlasP desde R:

**system('./runBlastp.bash atp.pep sec.pep')** #es como teclear en terminal linux

**t=read.table("tablast.nc.Ecoli.pep-nc.Ecoli.pep")**

**colnames(t)= c("query","subject","%id","len\_aln","mism","gapop",**

**"qstart","qend","sstart","send","evalue","score","qlen","slen")head(t)**

# Podemos filtrar la tabla desde aquí

**borrar=which(t$query == t$subject)**

**t2=t[-borrar,]**

**dim(t)**

**dim(t2)**

**borrar=which(t2$p\_id\_aln<50)**

**t3 = t2[-borrar,]**

**dim(t3)**

**borrar=which(t3$len <0.9 \* t3$slen)**

**t4=t3[-borrar,]**

**dim(t4)**

## SCRIPT DE BRH

# en gedit poner este archivo en modo resaltado para sh o bash (menú Ver-> Modo resaltado -> scripts -> sh)

# $1 es el fasta con los genes del organismo 1

# $2 es el fasta con los genes del organismo 2

# hago el blast 1 vs 2

**makeblastdb -in $2 -dbtype prot -out $2.base**

**blastp -query $1 -db $2.base -outfmt "6 std qlen slen" -out tablast.$1.$2 -max\_target\_seqs 1 -num\_threads 2 -evalue 1e-5**

# el resultado está en tablast.$1.vs.$2. Ahora hago el blast inverso.

**makeblastdb -in $1 -dbtype prot -out $1.base**

**blastp -query $2 -db $1.base -outfmt "6 std qlen slen" -out tablast.$2.$1 -max\_target\_seqs 1 -num\_threads 2 -evalue 1e-5**

# el resultado está en tablast.$2.vs.$1. Ahora me quedo con las dos primeras columnas de tablast.

**cat tablast.$1.$2 |cut -f1,2 > $1.BH**

**cat tablast.$2.$1 |cut -f1,2 > $2.BH**

# uno tiene los genes 1 a la izquierda y los genes 2 a la derecha, y el otro a la inversa. Invierto el segundo.

# lo que hay entre las comillas (al lado del print) es un tab, para que se mantenga la estructura gen-tab-gen

**cat $2.BH |awk '{print $2" "$1}' > $2.HB # ojo que es un tab lo que hay entre las comillas dobles.**

# lo llamo $2.HB para representar que es el mismo que $2.BH pero con las columnas intercambiadas.

# ahora los junto en un archivo, y hago un sort|uniq -c, que me da cuántas veces aparece cada par.

# si un par "a b" aparece dos veces, significa que el best hit de a era b, y el best hit de b era a.

**cat $1.BH $2.HB | sort | uniq -c | grep -v " 1 "| awk '{print $2" "$3}' > BRH.$1-$2**

# Acá también, verificar que lo que está entre el $2 y el $3, entre comillas, es un TAB.

# para entender este largo pipe-line hay que ir haciéndolo por partes (justo hasta antes de cada |) e ir viendo qué pasa.

# muevo los archivos que tal vez use a una carpeta que se llamará "archivos", para no tener tanta cosa junta. Si la carpeta

# ya está hecha (de una corrida anterior) te sale un warning, pero no importa.

**mkdir archivos**

**mv \*.BH \*.HB archivos**

## 1 SCRIPT REPASO R

**library(seqinr)**

**l=read.fasta("nc.Salm.cds")**

**class(l[[1]])**

**length((l[[1]])) #lista 1 del conjunto de listas L**

**l[[1]]**

**l**

**length(l)** #largo de lista da cuantas seq se tienen

**sapply(l,length) -> largos**

**head(largos)**

**max(largos)**

**mean(largos)**

**min(largos)**

**summary((largos))**

**boxplot(largos, outline=F)**

**hist(largos)**

**GC(l[[1]])**

**GC(l[[2]])**

**GC(l[[3]])**

**gcontent=sapply(l,GC)**

**gcontent**

**summary(gcontent)**

**max(gcontent)**

**hist(gcontent,100)**

**supersecuencia=unlist(l)**

**length(supersecuencia)**

**media=GC(supersecuencia)**

**grep(TRUE, gc>media)** #encuentra en q lugar estan los mayores a la media

**length(grep(TRUE, gc>media))** #q cantidad de genes por debajo de la media

**length(grep(FALSE, gc>media**)) #q cantidad de genes por arriba de la media

**head(largos)**

a= l[[1]]

**count(seq=a,wordsize = 1, start = 0, by=1,freq=FALSE)**

**count(seq=supersecuencia,wordsize = 1, start = 0, by=1,freq=FALSE)**

**count(seq=supersecuencia,wordsize = 1, start = 0, by=1,freq=FALSE, alphabet =c('a','t'))**

**sec=sample(s2c('atg'),100,replace=TRUE)**

**count(seq=sec,wordsize = 1, start = 0, by=1,freq=FALSE)**

## 2 SCRIPT REPASO R

**setwd("~/Bioinfo1\_2021")**

**#SETEAR LIBRERIA**

**library(seqinr)**

**secuencia <- read.fasta("sequence.txt")**

**length(secuencia)**

**length(secuencia[[1]])**

**class(secuencia)**

**sapply(secuencia, length)**

**largos <- sapply(secuencia, length)**

**class(largos)**

**mean(largos)**

**max((largos))**

**min(largos)**

**summary(largos)**

**todos\_CDS<- unlist(secuencia)**

**length(todos\_CDS)**

**table((todos\_CDS))**

**sum(largos)**

**traduccion<-sapply(secuencia,translate)**

**traduccion**

**class(traduccion)**

**todos\_aa <- unlist(traduccion)**

**table(todos\_aa)**

**table(traduccion[[2]])**

**count(seq = todos\_aa,wordsize = 1,start = 0,by = 1,freq = F,alphabet = a())**

**library(Biostrings)**

**library(seqninr)**

**sec1 <- read.fasta(‘path del fasta’, as.string=T)**

**sec2 <- read.fasta(‘path del fasta’, as.string=T)**

**sec1 <- DNAStringSet(as.character(sec1[[1]])**

**sec2 <- DNAStringSet(as.character(sec2[[1]])**

**mat <- nucleotideSubstitutionMatrix(match=1, mismatch=-1,base0nly=TRUE)**

**ali1 <- pairwiseAligment(patter =sec1, subject=sec2, type=’global’ o ‘local’, substitutionMatrix=mat, gap0pening=3, gapExtension=2)**

## QUIZZIZ

##### 30/9

1. Base de datos primaria: directo de experimentos, NCBI, EMBL DDBJ
2. Alineamientos: global (Needle) o local (Smith Waterman)
3. Matriz de sustitución (o de mutacion de péptidos) creada por Margaret Dayhoff fue PAM
4. x <- 4

2\*x+1

9

1. llamar librería de paquete ya instalado 🡪 library(). Para instalar 🡪 install.packages()
2. Directorio de trabajo? 🡪 getwd()
3. Cambiar el directorio de trabajo 🡪 setwd()
4. Ortólogos 🡪 sec de diferentes especies, con gen común ancestral, generadas por evento de especiación
5. Paralogos 🡪 sec homologas de misma especie, originadas por evento duplicación
6. estaba mal
7. dot plot de duplicación en tándem

****

1. Dotplot de inversión

****

1. PAM1 🡪 PAM 1% 🡪 1 cambio cada 100 aminoacidos, se ve la distancia evolutiva de aa, no de nucleotidos
2. MSA: Dominios conservados de la secuencia: los que están en rojo, la que tenga menos rojo es la que menos cambio evolutivo tuvo

##### 4/10

1. BLAST se basa en algoritmo Smith Waterman, alineamiento local
2. BLAST 🡪 Basic Local Alignment Search Tool
3. BLASTN 🡪 buscar query de nucleótidos contra base de nucleótidos

BLASTP 🡪 buscar proteínas contra proteínas

BLASTX🡪 proteínas traducidas contra proteínas

TBLASTX🡪 proteínas traducidas contra proteínas traducidas

1. Como funciona algoritmo blast?
   1. Cortar palabras del query para formar palabras más chicas
   2. Si el largo es 3, se va cortando el query de a 3, corriendo la primera letra

V T A L W G K V N V D 🡪 VTA/ TAL/ ALW/ LWG…. Etc

1. High Score Segment Pair🡪 pares de alineamiento con mejor score, desde el alineamiento primario de secuencia corta (seed), extendiéndolo hacia extremos hacia derecha e izquierda, hasta que el score queda por debajo del umbral
2. Cuanto más bajo sea el e value más significativo es el alineamiento. si e value es 10, habrá 10 alineamientos al azar que tendrán el mismo score alto que los válidos. Si e value es 0.1 solo 0.1 de los alineamientos con score valido será al azar