EBook Gratis

APRENDIZAJE bioinformatics

Free unaffiliated eBook created from
Stack Overflow contributors.

#bioinformatics

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: bioinformatics

It is an unofficial and free bioinformatics ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official bioinformatics.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Comenzando con la bioinformática

Observaciones

La bioinformática es un campo interdisciplinario que desarrolla métodos y herramientas de software para comprender los datos biológicos.

Los temas dentro de la bioinformática incluyen:

- Análisis de secuencia
- Filogenia
- Modelado molecular
- Análisis de expresión de genes y proteínas.

Examples

Definición

> (Wikipedia) La bioinformática es un campo interdisciplinario que desarrolla métodos y herramientas de software para comprender los datos biológicos. Como campo interdisciplinario de la ciencia, la bioinformática combina la informática, las estadísticas, las matemáticas y la ingeniería para analizar e interpretar datos biológicos. La bioinformática se ha utilizado para el análisis in silico de consultas biológicas utilizando técnicas matemáticas y estadísticas.

Analizador de archivos .GFF (como búfer) con filtro para mantener solo las filas

```
""" A [GFF parser script][1] in Python for [www.VigiLab.org][2]

    Description:
        - That performs buffered reading, and filtering (see: @filter) of .GFF input file
(e.g. "[./toy.gff][3]") to keep only rows whose field (column) values are equal to
"transcript"...

    Args:
        - None (yet)

    Returns:
        - None (yet)

    Related:
        - [1]: https://github.com/a1ultima/vigilab_intergeneShareGFF/blob/master/README.md
        - [2]: http://www.vigilab.org/
        - [3]: https://github.com/a1ultima/vigilab_intergeneShareGFF/blob/master/toy.gff

"""
```

```
gene_to_field = {}  # dict whose keys: genes represented (i.e. later slice-able/index-able) as
1..n, values, where n = 8 total #fields (cols) of a gff row, whose version is unknown but
example is: https://github.com/a1ultima/vigilab_intergeneShareGFF/blob/master/toy.gff

gene_i = 0

with open("./toy.gff", "r") as fi:

    print("Reading GFF file into: gene_to_field (dict), index as such: gene_to_field[gene_i],
where gene_i is between 1-to-n...")

while True: # breaks once there are no more lines in the input .gff file, see "@break"

    line = fi.readline().rstrip() # no need for trailing newline chars ("\n")

    if line == "":  # @break
        break

    line_split = line.split("\t") # turn a line of input data into a list, each element =
different field value, e.g. [...,"transcript",...]

    if line_split[2] != "transcript": # @@filter incoming rows so only those with "transcript"
are not skipped by "continue"
        continue

    gene_i += 1  # indexing starts from 1 (i.e. [1] = first gene) ends at n

    ##@TEST: sometimes 4.00 instead of 4.0 (trivial)   # some @deprecated code, but may be
useful one day
    #if not (str(line_split[5])==str(float(line_split[5]))):
    #    print("oops")
    #    print("\t"+str(line_split[5])+"___"+str(float(line_split[5])))


    # create a dict key, for gene_to_field dict, and set its values according to list elements
in line_split

    gene_to_field[gene_i] = { \
        "c1_reference_seq":line_split[0],# e.g. 'scaffold_150' \
        "c2_source":line_split[1],# e.g. 'GWSUNI' \
        "c3_type":line_split[2],# e.g. 'transcript' \
        "c4_start":int(line_split[3]),# e.g. '1372' \
        "c5_end":int(line_split[4]),# e.g. '2031' \
        "c6_score":float(line_split[5]),# e.g. '45.89' \
        "c7_strand":line_split[6],# e.g. '+' \
        "c8_phase":line_split[7],# e.g. '.' @Note: codon frame (0,1,2) \
        "c9_attributes":line_split[8]# e.g. <see @gff3.md> \
    }
```

Usando mapeo de secuencias de ADN para responder preguntas biológicas

Muchas preguntas biológicas pueden traducirse en un problema de secuenciación del ADN. Por ejemplo, si desea conocer el nivel de expresión de un gen, puede: copiar sus ARNm en moléculas de ADN complementarias, secuenciar cada una de las moléculas de ADN resultantes, mapear esas secuencias de nuevo al genoma de referencia y luego usar el conteo de alineaciones superpuestas El gen como proxy de su expresión (ver RNA-seq). Otros ejemplos incluyen: determinar la estructura 3D del genoma , localizar marcas de histonas y mapear las interacciones ARN-ADN . Aquí puede encontrar una lista no actualizada de preguntas biológicas abordadas por

métodos inteligentes de secuenciación de ADN.

Por lo general, los científicos de laboratorios húmedos (las personas que usan batas blancas y gafas) diseñarán y realizarán los experimentos para obtener las muestras de ADN secuenciadas. Luego, un bioinformático (las personas que usan computadoras y toman café) tomarán estas secuencias, codificadas como archivos FASTQ , y las asignarán a un genoma de referencia, guardando los resultados como archivos BAM .

Volviendo a nuestro ejemplo de expresión genética, así es como un bioinformático generaría un archivo BAM desde un archivo FASTQ (utilizando un sistema Linux):

```
STAR --genomeDir path/to/reference/genome --outSAMtype BAM --readFilesIn my_reads.fastq
```

Donde STAR es un alineador tolerante al empalme (necesario para las uniones exón-intrón que pueden estar presentes en el ARNm).

PD: Una vez que se obtienen los resultados del mapeo, comienza la parte creativa. Aquí es donde los bioinformáticos diseñaron una prueba estadística para verificar si los datos muestran patrones biológicamente significativos o señales espurias nacidas del ruido.

Lea Comenzando con la bioinformática en línea:
https://riptutorial.com/es/bioinformatics/topic/3960/comenzando-con-la-bioinformatica

Capítulo 2: Análisis de secuencia

Examples

Calcula el% GC de una secuencia

En biología molecular y genética, el contenido de GC (o contenido de guanina-citosina,% GC en resumen) es el porcentaje de bases nitrogenadas en una molécula de ADN que son guanina o citosina (de una posibilidad de cuatro diferentes, que también incluyen adenina y timina).

Utilizando BioPython:

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> from Bio.SeqUtils import GC
>>> my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPAC.unambiguous_dna)
>>> GC(my_seq)
46.875
```

Utilizando BioRuby:

```
bioruby> require 'bio'
bioruby> seq = Bio::Sequence::NA.new("atgcatgcaaaa")
==> "atgcatgcaaaa"
bioruby> seq.gc_percent

==> 33
```

Utilizando R:

```
# Load the SeqinR package.
library("seqinr")
mysequence <- s2c("atgcatgcaaaa")
GC(mysequence)

# [1] 0.3333333
```

Usando Awk:

```
echo atgcatgcaaaa |\
awk '{dna=$0;  gsub(/[^GCSgcs]/,""); print dna,": GC=",length($0)/length(dna)}'

# atgcatgcaaaa : GC= 0.333333
```

Lea Análisis de secuencia en línea: https://riptutorial.com/es/bioinformatics/topic/4015/analisis-de-secuencia

Capítulo 3: EXPLOSIÓN

Examples

Crear un blastdb de ADN

Para comparar las secuencias de consulta con las secuencias de referencia, debe crear un blastdb de su (s) referencia (s). Esto se hace usando `makeblastdb` que se incluye cuando instala blast.

```
makeblastdb -in <input fasta> -dbtype nucl -out <label for database>
```

Así que si tienes un archivo `reference.fasta` contiene los siguientes registros:

```
>reference_1
ATCGATAAA
>reference_2
ATCGATCCC
```

Ejecutarías lo siguiente:

```
makeblastdb -in reference.fasta -dbtype nucl -out my_database
```

Esto crearía los siguientes archivos:

- `my_database.nhr`
- `my_database.nin`
- `my_database.nsq`

Tenga en cuenta que los archivos de la base de datos están etiquetados con el argumento `-out` .

Extraer secuencias fasta de un nucl blastdb

Puede extraer la secuencia fasta de un blastdb construido a partir de un archivo fasta usando `blastdbcmd` que debe instalarse cuando instale `makeblastdb` .

```
blastdbcmd -entry all -db <database label> -out <outfile>
```

Si tuviera una base de datos llamada `my_database` que contuviera los archivos `my_database.nhr` , `my_database.nsq` , `my_database.nin` y quisiera que su archivo de salida fasta se llamara `reference.fasta` , ejecutaría lo siguiente:

```
blastdbcmd -entry all -db my_database -out reference.fasta
```

Instalar blast en ubuntu

```
apt-get install ncbi-blast+
```

Puedes consultar la versión que se instalará de antemano aquí:

http://packages.ubuntu.com/xenial/ncbi-blast+

Extraer GI y taxid de blastdb

Los datos se pueden extraer de un blastdb utilizando `blastdbcmd` que debe incluirse en una instalación de `blastdbcmd` . Puede especificar a partir de las opciones a continuación como parte de `-outfmt` qué metadatos incluir y en qué orden.

De la página del manual:

```
-outfmt <String>
  Output format, where the available format specifiers are:
      %f means sequence in FASTA format
      %s means sequence data (without defline)
      %a means accession
      %g means gi
      %o means ordinal id (OID)
      %i means sequence id
      %t means sequence title
      %l means sequence length
      %h means sequence hash value
      %T means taxid
      %X means leaf-node taxids
      %e means membership integer
      %L means common taxonomic name
      %C means common taxonomic names for leaf-node taxids
      %S means scientific name
      %N means scientific names for leaf-node taxids
      %B means BLAST name
      %K means taxonomic super kingdom
      %P means PIG
```

El fragmento de ejemplo muestra cómo gi y taxid pueden extraerse de blastdb. Se eligió el blastdb NCBI 16SMicrobial (ftp) para este ejemplo:

```
# Example:
# blastdbcmd -db <db label> -entry all -outfmt "%g %T" -out <outfile>
blastdbcmd -db 16SMicrobial -entry all -outfmt "%g %T" -out 16SMicrobial.gi_taxid.tsv
```

Que producirá un archivo `16SMicrobial.gi_taxid.tsv` que se ve así:

```
939733319 526714
636559958 429001
645319546 629680
```

Lea EXPLOSIÓN en línea: https://riptutorial.com/es/bioinformatics/topic/5371/explosion

Capítulo 4: Formatos de archivo comunes

Examples

FASTA

El formato de archivo FASTA se utiliza para representar una o más secuencias de nucleótidos o aminoácidos como una cadena continua de caracteres. Las secuencias se anotan con una línea de comentario, que comienza con el carácter `>` , que precede a cada secuencia. La línea de comentarios suele tener un formato uniforme, dictada por la base de datos fuente de la secuencia o el software de generación. Por ejemplo:

```
>gi|62241013|ref|NP_001014431.1| RAC-alpha serine/threonine-protein kinase [Homo sapiens]
MSDVAIVKEGWLHKRGEYIKTWRPRYFLLKNDGTFIGYKERPQDVDQREAPLNNFSVAQCQLMKTERPRP
NTFIIRCLQWTTVIERTFHVETPEEREEWTTAIQTVADGLKKQEEEEMDFRSGSPSDNSGAEEMEVSLAK
PKHRVTMNEFEYLKLLGKGTFGKVILVKEKATGRYYAMKILKKEVIVAKDEVAHTLTENRVLQNSRHPFL
TALKYSFQTHDRLCFVMEYANGGELFFHLSRERVFSEDRARFYGAEIVSALDYLHSEKNVVYRDLKLENL
MLDKDGHIKITDFGLCKEGIKDGATMKTFCGTPEYLAPEVLEDNDYGRAVDWWGLGVVMYEMMCGRLPFY
NQDHEKLFELILMEEIRFPRTLGPEAKSLLSGLLKKDPKQRLGGGSEDAKEIMQHRFFAGIVWQHVYEKK
LSPPFKPQVTSETDTRYFDEEFTAQMITITPPDQDDSMECVDSERRPHFPQFSYSASGTA
```

El ejemplo anterior ilustra la secuencia de aminoácidos de una isoforma de los genes AKT1 humanos, como se obtiene de la base de datos de proteínas NCBI. La línea del encabezado especifica que esta secuencia puede identificarse con el ID de GI `62241013` y el ID de transcripción de la proteína `NP_001014431.1` . Esta proteína se llama `RAC-alpha serine/threonine-protein kinase` y se deriva de la especie, `Homo sapiens` .

Formato de anotación de mutación (MAF)

El formato de archivo MAF es un formato de archivo de texto delimitado por tabulaciones destinado a describir mutaciones somáticas de ADN detectadas en los resultados de secuenciación, y es distinto del tipo de archivo de Formato de Alineación Múltiple, que está destinado a representar secuencias de nucleótidos alineadas. Los encabezados y el orden de las columnas a veces pueden variar entre los archivos de diferentes fuentes, pero los nombres y órdenes de las columnas, tal como se definen en la especificación, son los siguientes:

```
Hugo_Symbol
Entrez_Gene_Id
Center
NCBI_Build
Chromosome
Start_Position
End_Position
Strand
Variant_Classification
Variant_Type
Reference_Allele
Tumor_Seq_Allele1
Tumor_Seq_Allele2
```

```
dbSNP_RS
dbSNP_Val_Status
Tumor_Sample_Barcode
Matched_Norm_Sample_Barcode
Match_Norm_Seq_Allele1
Match_Norm_Seq_Allele2
Tumor_Validation_Allele1
Tumor_Validation_Allele2
Match_Norm_Validation_Allele1
Match_Norm_Validation_Allele2
Verification_Status4
Validation_Status4
Mutation_Status
Sequencing_Phase
Sequence_Source
Validation_Method
Score
BAM_File
Sequencer
Tumor_Sample_UUID
Matched_Norm_Sample_UUID
```

Muchos archivos MAF, como los disponibles en la TCGA, también contienen columnas adicionales que se expanden en la anotación de variante. Estas columnas pueden incluir ID de transcripción de nucleótidos de referencia para genes correspondientes, codones representativos o cambios de aminoácidos, métricas de control de calidad, estadísticas de población y más.

GCT

El formato de archivo GCT es un formato de archivo de texto delimitado por tabuladores que se utiliza para describir la expresión génica procesada o los datos de ARNi, típicamente derivados del análisis de chip de microarrays. Estos datos se organizan con un único gen o sonda anotada por línea y una muestra de chip por columna (más allá de las columnas de anotación). Por ejemplo:

```
#1.2
22215    2
Name    Description    Tumor_One    Normal_One
1007_s_at    DDR1    -0.214548    -0.18069
1053_at    RFC2    0.868853    -1.330921
117_at    HSPA6    1.124814    0.933021
121_at    PAX8    -0.825381    0.102078
1255_g_at    GUCA1A    -0.734896    -0.184104
1294_at    UBE1L    -0.366741    -1.209838
```

En este ejemplo, la primera línea especifica la versión de la especificación del archivo GCT, que en este caso es `1.2` . La segunda línea especifica el número de filas de datos (`22215`) y el número de muestras (`2`). La fila del encabezado especifica dos columnas de anotación (`Name` para los identificadores de conjuntos de sondas de chip y `Description` para los símbolos de genes que abarca el conjunto de sondas) y los nombres de las muestras que se están analizando (`Tumor_One` y `Normal_One`). Cada fila de datos más allá del encabezado enumera un único identificador de conjunto de sondas (en este caso, conjuntos de sondas de chip de genes de Affymetrix), su correspondiente símbolo de gen (si existe) y los valores normalizados para cada muestra. Los

valores de los datos de muestra variarán según el tipo de ensayo y los métodos de normalización, pero normalmente son valores numéricos de punto flotante con signo.

Secuencia de escritura en formato fasta

Esta es una función de ejemplo de python para escritura de secuencias en formato fasta.

Parámetros:

- filename (String): un nombre de archivo para la secuencia de escritura en formato fasta.
- seq (String) - Una secuencia de ADN o ARN.
- id (String): el ID de la secuencia dada.
- desc (String) - Una breve descripción de la secuencia dada.

```
import math

    def save_fsta(filename,seq,id,desc):
        fo = open(filename+'.fa',"a")
        header= str(id)+' <'+desc+'> \n'
        fo.write(header)
        count=math.floor(len(seq)/80+1)
        iteration = range(count)
        for i in iteration:
            fo.write(seq[80*(i):80*(i+1)]+'\n')
        fo.write('\n \n')
        fo.close()
```

Otra forma es usando `textwrap`

```
import textwrap

def save_fasta(filename,seq, id, desc):
    filename+='.fa'
    with open(filename, 'w') as f:
        f.write('>'+id+' <'+desc+'>\n');
        text = textwrap.wrap(seq,80);
        for x in text:
            f.write(x+'\n');
```

Lea Formatos de archivo comunes en línea:
https://riptutorial.com/es/bioinformatics/topic/4034/formatos-de-archivo-comunes

Capítulo 5: Linealizando un archivo fastq

Examples

Utilizando pegar

```
$ gunzip -c input.fastq.gz | paste - - - - | head

@IL31_4368:1:1:996:8507/2    TCCCTTACCCCCAAGCTCCATACCCTCCTAATGCCCACACCTCTTACCTTAGGA    +
FFCEFFFEEFFFFFFFFEFFEFFFEFCFC<EEFEFFFCEFF<;EEFF=FEE?FCE
@IL31_4368:1:1:996:21421/2    CAAAAACTTTCACTTTACCTGCCGGGTTTCCCAGTTTACATTCCACTGTTTGAC    +
>DBDDB,B9BAA4AAB7BB?7BBB=91;+*@;5<87+*=/*@@?9=73=.7)7*
@IL31_4368:1:1:997:10572/2    GATCTTCTGTGACTGGAAGAAAATGTGTTACATATTACATTTCTGTCCCCATTG    +
E?=EECE<EEEE98EEEEAEEBD??BE@AEAB><EEABCEEDEC<<EBDA=DEE
@IL31_4368:1:1:997:15684/2    CAGCCTCAGATTCAGCATTCTCAAATTCAGCTGCGGCTGAAACAGCAGCAGGAC    +
EEEEDEEE9EAEEDEEEEEEEEEECEEAAEEDEE<CD=D=*BCAC?;CB,<D@,
@IL31_4368:1:1:997:15249/2    AATGTTCTGAAACCTCTGAGAAAGCAAATATTTATTTTAATGAAAAATCCTTAT    +
EDEEC;EEE;EEE?EECE;7AEEEEEE07EECEA;D6D>+EE4E7EEE4;E=EA
@IL31_4368:1:1:997:6273/2    ACATTTACCAAGACCAAAGGAAACTTACCTTGCAAGAATTAGACAGTTCATTTG    +
EEAAFFFEEFEFCFAFFAFCCFFEFEF>EFFFFB?ABA@ECEE=<F@DE@DDF;
@IL31_4368:1:1:997:1657/2    CCCACCTCTCTCAATGTTTTCCATATGGCAGGGACTCAGCACAGGTGGATTAAT    +
A;0A?AA+@A<7A7019/<65,3A;'''07<A=<=>?7=?6&)'9('*%,>/(<
@IL31_4368:1:1:997:5609/2    TCACTATCAGAAACAGAATGTATAACTTCCAAATCAGTAGGAAACACAAGGAAA    +
AEECECBEC@A;AC=<AEEEEAEEEE>AC,CE?ECCE9EAEC4E:<C>AC@EE)
@IL31_4368:1:1:997:14262/2    TGTTTTTTCTTTTTCTTTTTTTTTTTGACAGTGCAGAGATTTTTTATCTTTTTAA    +
97'<2<.64.?7/3(891?=(6??6+<6<++/*..3(:'/'9::''&(1<>.(,
@IL31_4368:1:1:998:19914/2    GAATGAAAGCAGAGACCCTGATCGAGCCCCAGAAAGATACACCTCCAGATTTTA    +
C?=CECE4CD<?8@==;EBE<=0@:@@92@???6<991>.<?A=@5?@99;971
```

Usando Awk

```
$ gunzip -c input.fastq.gz | awk '{printf("%s%s",$0,((NR+1)%4==1?"\n":"\t"));}' | head

@IL31_4368:1:1:996:8507/2    TCCCTTACCCCCAAGCTCCATACCCTCCTAATGCCCACACCTCTTACCTTAGGA    +
FFCEFFFEEFFFFFFFFEFFEFFFEFCFC<EEFEFFFCEFF<;EEFF=FEE?FCE
@IL31_4368:1:1:996:21421/2    CAAAAACTTTCACTTTACCTGCCGGGTTTCCCAGTTTACATTCCACTGTTTGAC    +
>DBDDB,B9BAA4AAB7BB?7BBB=91;+*@;5<87+*=/*@@?9=73=.7)7*
@IL31_4368:1:1:997:10572/2    GATCTTCTGTGACTGGAAGAAAATGTGTTACATATTACATTTCTGTCCCCATTG    +
E?=EECE<EEEE98EEEEAEEBD??BE@AEAB><EEABCEEDEC<<EBDA=DEE
@IL31_4368:1:1:997:15684/2    CAGCCTCAGATTCAGCATTCTCAAATTCAGCTGCGGCTGAAACAGCAGCAGGAC    +
EEEEDEEE9EAEEDEEEEEEEEEECEEAAEEDEE<CD=D=*BCAC?;CB,<D@,
@IL31_4368:1:1:997:15249/2    AATGTTCTGAAACCTCTGAGAAAGCAAATATTTATTTTAATGAAAAATCCTTAT    +
EDEEC;EEE;EEE?EECE;7AEEEEEE07EECEA;D6D>+EE4E7EEE4;E=EA
@IL31_4368:1:1:997:6273/2    ACATTTACCAAGACCAAAGGAAACTTACCTTGCAAGAATTAGACAGTTCATTTG    +
EEAAFFFEEFEFCFAFFAFCCFFEFEF>EFFFFB?ABA@ECEE=<F@DE@DDF;
@IL31_4368:1:1:997:1657/2    CCCACCTCTCTCAATGTTTTCCATATGGCAGGGACTCAGCACAGGTGGATTAAT    +
A;0A?AA+@A<7A7019/<65,3A;'''07<A=<=>?7=?6&)'9('*%,>/(<
@IL31_4368:1:1:997:5609/2    TCACTATCAGAAACAGAATGTATAACTTCCAAATCAGTAGGAAACACAAGGAAA    +
AEECECBEC@A;AC=<AEEEEAEEEE>AC,CE?ECCE9EAEC4E:<C>AC@EE)
@IL31_4368:1:1:997:14262/2    TGTTTTTTCTTTTTCTTTTTTTTTTTGACAGTGCAGAGATTTTTTATCTTTTTAA    +
97'<2<.64.?7/3(891?=(6??6+<6<++/*..3(:'/'9::''&(1<>.(,
@IL31_4368:1:1:998:19914/2    GAATGAAAGCAGAGACCCTGATCGAGCCCCAGAAAGATACACCTCCAGATTTTA    +
C?=CECE4CD<?8@==;EBE<=0@:@@92@???6<991>.<?A=@5?@99;971
```

Lea Linealizando un archivo fastq en línea:
https://riptutorial.com/es/bioinformatics/topic/4286/linealizando-un-archivo-fastq

Capítulo 6: Linealizando una secuencia FASTA.

Examples

Linealizar una secuencia FASTA con AWK

Lectura línea por línea

```
awk '/^>/ {printf("%s%s\t",(N>0?"\n":""),$0);N++;next;} {printf("%s",$0);} END
{printf("\n");}' < input.fa
```

Uno puede leer este script awk como:

- si la línea actual (`$0`) comienza como un encabezado fasta (`^>`). Luego imprimimos un retorno de carro si esta no es la primera secuencia. `(N>0?"\n":"")` seguido de la línea (`$0`), seguido de una tabulación (`\t`). Y buscamos la siguiente línea (`next;`)
- Si la línea actual (`$0`) no comienza como un encabezado fasta, este es el patrón awk predeterminado. Acabamos de imprimir toda la línea sin retorno de carro.
- Al final (`END`) solo imprimimos un retorno de carro para la última secuencia.

Linearizar secuencias FASTA de Uniprot

descargue y linealice las 10 primeras secuencias FASTA de UniProt:

```
$ curl -s
"ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.fasta
|\
  gunzip -c |\
  awk '/^>/ {printf("%s%s\t",(N>0?"\n":""),$0);N++;next;} {printf("%s",$0);} END
{printf("\n");}'  |\
  head

>sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate Goorha)
GN=FV3-001R PE=4 SV=1
MAFSAEDVLKEYDRRRRMEALLLSLYYPNDRKLLDYKEWSPPRVQVECPKAPVEWNNPPSEKGLIVGHFSGIKYKGEKAQASEVDVNKMCCWVSKFKDAMRRY

>sp|Q6GZX3|002L_FRG3G Uncharacterized protein 002L OS=Frog virus 3 (isolate Goorha) GN=FV3-
002L PE=4 SV=1
MSIIGATRLQNDKSDTYSAGPCYAGGCSAFTPRGTCGKDWDLGEQTCASGFCTSQPLCARIKKTQVCGLRYSSKGKDPLVSAEWDSRGAPYVRCTYDADLIDT

>sp|Q197F8|002R_IIV3 Uncharacterized protein 002R OS=Invertebrate iridescent virus 3 GN=IIV3-
002R PE=4 SV=1
MASNTVSAQGGSNRPVRDFSNIQDVAQFLLFDPIWNEQPGSIVPWKMNREQALAERYPELQTSEPSEDYSGPVESLELLPLEIKLDIMQYLSWEQISWCKHPW

>sp|Q197F7|003L_IIV3 Uncharacterized protein 003L OS=Invertebrate iridescent virus 3 GN=IIV3-
003L PE=4 SV=1
MYQAINPCPQSWYGSPQLEREIVCKMSGAPHYPNYYPVHPNALGGAWFDTSLNARSLTTTPSLTTCTPPSLAACTPPTSLGMVDSPPHINPPRRIGTLCFDFC
```

```
>sp|Q6GZX2|003R_FRG3G Uncharacterized protein 3R OS=Frog virus 3 (isolate Goorha) GN=FV3-003R
PE=3 SV=1
MARPLLGKTSSVRRRLESLSACSIFFFLRKFCQKMASLVFLNSPVYQMSNILLTERRQVDRAMGGSDDDGVMVVALSPSDFKTVLGSALLAVERDMVHVVPKY

>sp|Q6GZX1|004R_FRG3G Uncharacterized protein 004R OS=Frog virus 3 (isolate Goorha) GN=FV3-
004R PE=4 SV=1    MNAKYDTDQGVGRMLFLGTIGLAVVVGGLMAYGYYYDGKTPSSGTSFHTASPSFSSRYRY
>sp|Q197F5|005L_IIV3 Uncharacterized protein 005L OS=Invertebrate iridescent virus 3 GN=IIV3-
005L PE=3 SV=1
MRYTVLIALQGALLLLLLIDDGQGQSPYPYPGMPCNSSRQCGLGTCVHSRCAHCSSDGTLCSPEDPTMVWPCCPESSCQLVVGLPSLVNHYNCLPNQCTDSSQ

>sp|Q6GZX0|005R_FRG3G Uncharacterized protein 005R OS=Frog virus 3 (isolate Goorha) GN=FV3-
005R PE=4 SV=1
MQNPLPEVMSPEHDKRTTTPMSKEANKFIRELDKKPGDLAVVSDFVKRNTGKRLPIGKRSNLYVRICDLSGTIYMGETFILESWEELYLPEPTKMEVLGTLES

>sp|Q91G88|006L_IIV6 Putative KilA-N domain-containing protein 006L OS=Invertebrate iridescent
virus 6 GN=IIV6-006L PE=3 SV=1
MDSLNEVCYEQIKGTFYKGLFGDFPLIVDKKTGCFNATKLCVLGGKRFVDWNKTLRSKKLIQYYETRCDIKTESLLYEIKGDNNDEITKQITGTYLPKEFILD

>sp|Q6GZW9|006R_FRG3G Uncharacterized protein 006R OS=Frog virus 3 (isolate Goorha) GN=FV3-
006R PE=4 SV=1    MYKMYFLKDQKFSLSGTIRINDKTQSEYGSVWCPGLSITGLHHDAIDHNMFEEMETEIIEYLGPWVQAEYRRIKG
```

Lea Linealizando una secuencia FASTA. en línea:
https://riptutorial.com/es/bioinformatics/topic/4194/linealizando-una-secuencia-fasta-

Capítulo 7: Samtools básicos

Examples

Número de registros por referencia en bamfile

```
samtools idxstats thing.bam
```

Convertir sam en bam (y viceversa)

Samtools se puede utilizar para convertir entre sam y bam:

- `-b` indica que el archivo de entrada estará en formato BAM
- `-S` indica que la salida estándar debe estar en formato SAM

```
samtools view -sB thing.bam > thing.sam
```

Y convertir entre sam y bam:

```
samtools view thing.sam > thing.bam
samtools sort thing.bam thing
samtools index thing.bam
```

Esto producirá un bam ordenado e indexado. Esto creará los archivos `thing.bam` y `thing.bam.bai` . Para usar un bam debes tener un archivo de índice.

Lea Samtools básicos en línea: https://riptutorial.com/es/bioinformatics/topic/6886/samtools-basicos

Creditos

S. No	Capítulos	Contributors
1	Comenzando con la bioinformática	BioGeek, Community, hello_there_andy, Marcelo, Pierre
2	Análisis de secuencia	BioGeek, Pierre, zx8754
3	EXPLOSIÓN	amblina
4	Formatos de archivo comunes	Razik, woemler
5	Linealizando un archivo fastq	Pierre
6	Linealizando una secuencia FASTA.	Pierre
7	Samtools básicos	amblina