FREE eBook

LEARNING bioinformatics

Free unaffiliated eBook created from
Stack Overflow contributors.

#bioinformatics

Table of Contents

About ... 1

Chapter 1: Getting started with bioinformatics ... 2

Remarks ... 2

Examples ... 2

Definition ... 2

.GFF file parser (as buffer) with filter to keep only rows ... 2

Using mapping of DNA sequences to answer biological questions ... 3

Chapter 2: Basic Samtools ... 5

Examples ... 5

Count number of records per reference in bamfile ... 5

Convert sam into bam (and back again) ... 5

Chapter 3: BLAST ... 6

Examples ... 6

Create a DNA blastdb ... 6

Extract fasta sequences from a nucl blastdb ... 6

Install blast on ubuntu ... 6

Extract GI and taxid from blastdb ... 7

Chapter 4: Common File Formats ... 8

Examples ... 8

FASTA ... 8

Mutation Annotation Format (MAF) ... 8

GCT ... 9

Sequence Writing In fasta Format ... 9

Chapter 5: Linearizing a FASTA sequence. ... 11

Examples ... 11

Linearize a FASTA sequence with AWK ... 11

Reading line by line ... 11

Linearize FASTA sequences from Uniprot ... 11

Chapter 6: Linearizing a fastq file ... 13

Examples ... 13

Using Paste . . . 13
Using Awk . . . 13
Chapter 7: Sequence analysis . . . 15
Examples . . . 15
Calculate the GC% of a sequence . . . 15
Credits . . . 16

About

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

Chapter 1: Getting started with bioinformatics

Remarks

Bioinformatics is an interdisciplinary field that develops methods and software tools for understanding biological data.

Topics within bioinformatics include:

- Sequence analysis
- Phylogenetics
- Molecular modeling
- Analysis of gene and protein expression

Examples

Definition

> (Wikipedia) Bioinformatics is an interdisciplinary field that develops methods and software tools for understanding biological data. As an interdisciplinary field of science, bioinformatics combines computer science, statistics, mathematics, and engineering to analyze and interpret biological data. Bioinformatics has been used for in silico analyses of biological queries using mathematical and statistical techniques.

.GFF file parser (as buffer) with filter to keep only rows

```
""" 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> \
    }
```

Using mapping of DNA sequences to answer biological questions

Many biological questions can be translated into a DNA sequencing problem. For instance, if you want to know the expression level of a gene you can: copy its mRNAs into complementary DNA molecules, sequence each of the resulting DNA molecules, map those sequences back to the reference genome, and then use the count of alignments overlapping the gene as a proxy of its expression (see RNA-seq). Other examples include: determining the 3D structure of the genome, locating histone marks, and mapping RNA-DNA interactions. A not up-to-date list of biological questions addressed by clever DNA-sequencing methods can be found here.

Typically, the wet-lab scientists (the people wearing white coats and goggles) will design and perform the experiments to get the sequenced DNA samples. Then, a bioinformatician (the people using computers and drinking coffee) will take these sequences --encoded as FASTQ files-- and

will map them to a reference genome, saving the results as BAM files.

Going back to our gene expression example, this is how a bioinformatician would generate a BAM file from a FASTQ file (using a Linux system):

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

Where STAR is a spliced-tolerant aligner (necessary for the exon-intron junctions that may be present on the mRNA).

PS: Once the mapping results are obtained, the creative part begins. Here is where bioinformaticians devised statistical test to check whether the data is showing biologically meaningful patterns or spurious signals born out of noise.

Read Getting started with bioinformatics online:
https://riptutorial.com/bioinformatics/topic/3960/getting-started-with-bioinformatics

Chapter 2: Basic Samtools

Examples

Count number of records per reference in bamfile

```
samtools idxstats thing.bam
```

Convert sam into bam (and back again)

Samtools can be used to convert between sam and bam:

- `-b` indicates that the input file will be in BAM format
- `-S` indicates that the stdout should be in SAM format

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

And to convert between sam and bam:

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

This will produce a sorted, indexed bam. This will create the files `thing.bam` and `thing.bam.bai`. To use a bam you must have an index file.

Read Basic Samtools online: https://riptutorial.com/bioinformatics/topic/6886/basic-samtools

Chapter 3: BLAST

Examples

Create a DNA blastdb

In order to compare query sequences against reference sequences, you must create a blastdb of your reference(s). This is done using `makeblastdb` which is included when you install blast.

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

So if you had a file `reference.fasta` containing the following records:

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

You would run the following:

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

This would create the following files:

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

Note, the database files are labelled with the `-out` argument.

Extract fasta sequences from a nucl blastdb

You can extract fasta sequence from a blastdb constructed from a fasta file using `blastdbcmd` which should be installed when you install `makeblastdb`.

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

If you had a database called `my_database` which contained the files `my_database.nhr`, `my_database.nsq`, `my_database.nin` and you wanted your fasta output file to be called `reference.fasta` you would run the following:

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

Install blast on ubuntu

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

You can check the version that will be installed in advance here:

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

Extract GI and taxid from blastdb

Data can be extracted from a blastdb using `blastdbcmd` which should be included in a blast installation. You can specify from the options below as part of `-outfmt` what metadata to include and in what order.

From the man page:

```
-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
```

The example snippet shows how gi and taxid can be extracted from blastdb. The NCBI 16SMicrobial (ftp) blastdb was chosen for this example:

```
# 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
```

Which will produce a file `16SMicrobial.gi_taxid.tsv` that looks like this:

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

Read BLAST online: https://riptutorial.com/bioinformatics/topic/5371/blast

Chapter 4: Common File Formats

Examples

FASTA

The FASTA file format is used for representing one or more nucleotide or amino acid sequences as a continuous string of characters. Sequences are annotated with a comment line, which starts with the `>` character, that precedes each sequence. The comment line is typically formatted in a uniform way, dictated by the sequence's source database or generating software. For example:

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

The above example illustrates the amino acid sequence of an isoform of the human AKT1 genes, as fetched from the NCBI protein database. The header line specifies that this sequence may be identified with the GI ID `62241013` and the protein transcript ID `NP_001014431.1`. This protein is named `RAC-alpha serine/threonine-protein kinase` and is derived from the species, `Homo sapiens`.

Mutation Annotation Format (MAF)

The MAF file format is a tab-delimited text file format intended for describing somatic DNA mutations detected in sequencing results, and is distinct from the Multiple Alignment Format file type, which is intended for representing aligned nucleotide sequences. Column headers and ordering may sometimes vary between files of different sources, but the names and orders of columns, as defined in the specification, are the following:

```
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
```

Many MAF files, such as those available from the TCGA, also contain additional columns expanding on the variant annotation. These columns can include reference nucleotide transcript IDs for corresponding genes, representative codon or amino acid changes, QC metrics, population statistics, and more.

GCT

The GCT file format is a tab-delimited text file format used for describing processed gene expression or RNAi data, typically derived from microarray chip analysis. This data is arranged with a single annotated gene or probe per line, and a single chip sample per column (beyond the annotation columns). For example:

```
#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
```

In this example, the first line specifies the version of the GCT file specification, which in this case is `1.2`. The second line specifies the number of rows of data (`22215`) and the number of samples (`2`). The header row specifies two annotation columns (`Name` for the chip probe set identifiers and `Description` for the gene symbols the probe set covers) and the names of the samples being assayed (`Tumor_One` and `Normal_One`). Each row of data beyond the header lists a single probe set identifier (in this case, Affymetrix gene chip probe sets), its corresponding gene symbol (if one exists), and the normalized values for each sample. Sample data values will vary based upon assay type and normalization methods, but are typically signed floating point numeric values.

Sequence Writing In fasta Format

This a python example function for sequence writing in fasta format.

Parameters:

- filename(String) - A file name for writing sequence in fasta format.
- seq(String) - A DNA or RNA sequence.
- id(String) - The ID of the given sequence.
- desc(String) - A short description of the given sequence.

```
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()
```

Another way is using `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');
```

Read Common File Formats online: https://riptutorial.com/bioinformatics/topic/4034/common-file-formats

Chapter 5: Linearizing a FASTA sequence.

Examples

Linearize a FASTA sequence with AWK

Reading line by line

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

one can read this awk script as:

- if the current line (`$0`) starts like a fasta header (`^>`). Then we print a carriage return if this is not the first sequence. `(N>0?"\n":"")` followed with the line itself (`$0`), followed with a tabulation (`\t`). And we look for the next line (`next;`)
- if the current line (`$0`) does not start like a fasta header, this is the default awk pattern. We just print the whole line without carriage return.
- At the end (`END`) we only print a carriage return for the last sequence.

Linearize FASTA sequences from Uniprot

download and linearize the 10 first FASTA sequences from 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
```

Read Linearizing a FASTA sequence. online:
https://riptutorial.com/bioinformatics/topic/4194/linearizing-a-fasta-sequence-

Chapter 6: Linearizing a fastq file

Examples

Using Paste

```
$ 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
```

Using 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
```

Read Linearizing a fastq file online: https://riptutorial.com/bioinformatics/topic/4286/linearizing-a-fastq-file

Chapter 7: Sequence analysis

Examples

Calculate the GC% of a sequence

In molecular biology and genetics, GC-content (or guanine-cytosine content, GC% in short) is the percentage of nitrogenous bases on a DNA molecule that are either guanine or cytosine (from a possibility of four different ones, also including adenine and thymine).

Using 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
```

Using BioRuby:

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

==> 33
```

Using R:

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

# [1] 0.3333333
```

Using Awk:

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

# atgcatgcaaaa : GC= 0.333333
```

Read Sequence analysis online: https://riptutorial.com/bioinformatics/topic/4015/sequence-analysis

Credits

S. No	Chapters	Contributors
1	Getting started with bioinformatics	BioGeek, Community, hello_there_andy, Marcelo, Pierre
2	Basic Samtools	amblina
3	BLAST	amblina
4	Common File Formats	Razik, woemler
5	Linearizing a FASTA sequence.	Pierre
6	Linearizing a fastq file	Pierre
7	Sequence analysis	BioGeek, Pierre, zx8754