1875-0362/20

The Open Bioinformatics Journal

Content list available at: https://openbioinformaticsjournal.com

LETTER

Extraordinary Command Line: Basic Data Editing Tools for Biologists Dealing with Sequence Data

Magda Mielczarek[1,2,*], Bartosz Czech[1], Jarosław Stańczyk[1], Joanna Szyda[1,2] and Bernt Guldbrandtsen[3]

[1]*Biostatistics Group, Department of Genetics, Wroclaw University of Environmental and Life Sciences; Kozuchowska 7, 51-631Wroclaw, Poland*
[2]*National Research Institute of Animal Production, Krakowska 1, 32-083 Balice, Poland*
[3]*Department of Animal Science, University of Bonn, Endenicher Allee 15, 53115Bonn, Germany*

Abstract:

The command line is a standard way of using the Linux operating system. It contains many features essential for efficiently handling data editing and analysis processes. Therefore, it is very useful in bioinformatics applications. Commands allow for rapid manipulation of large ASCII files or very numerous files, making basic command line programming skills a critical component in modern life science research. The following article is not a guide to Linux commands. In this manuscript, in contrast to many various Linux manuals, we aim to present basic command line tools helpful in handling biological sequence data. This manuscript provides a collection of simple and popular hacks dedicated to users with very basic experience in the area of the Linux command line. It includes a description of data formats and examples of editing of four types of data formats popular in bioinformatics applications.

Keywords: Bash, Command line, Data manipulation, DNA, Linux, Sequence data.

Article History	Received: September 07, 2020	Revised: November 18, 2020	Accepted: November 30, 2020

1. INTRODUCTION

Basic programming skills are critical in life science research [1 - 5]. Among computing environments used by scientists for years, the UNIX and the Linux operating systems have been used the most. Linux can be used *via* Graphical User Interfaces (GUI). However, using Linux textually from the so-called Command-Line Interface (CLI) is far more efficient. This is due to the philosophy that guided the creators of these systems. According to Gancarz [6], the philosophy of Unix can be subsumed into three statements: (i) make each program do one thing and do it well, (ii) write programs to work together, and (iii) write programs to handle text streams (*e.g.*, from text files), because that is a universal interface. The second factor of the popularity of the Linux operating system, or rather, more correctly GNU/Linux [7], is that it is open source software with many free distributions developed by an international community. GUIs tend to constrain interactions to specific workflows and inhibit the integration of tools into pipelines. Therefore, it is still worth dealing with the command line in times of an all-embracing picture culture and graphic interfaces. The CLI gives the user full control over commands issued from the command line. Exploiting the CLI's capacity for the integration of tools makes automation of quite complex tasks accessible to the average user. Additionally, the CLI allows the user to integrate bioinformatics analysis tools with automated data transformations using both specialized tools and Linux's native tools and to integrate transparently and efficiently with queue systems in high-performance computing environments. A final advantage of the CLI is that the set of tools available from the CLI level has been developed for years and is very extensive. Hence, most of the activities performed on the command line can be done in many ways.

The CLI is an environment available on computing clusters. The following article is not a guide to Linux commands. Numerous entry-level guides can easily be found on the internet; instead, we primarily focus on the interaction with GNU/Linux *via* the command line and present a few of the most popular tools useful in processing biological sequence data, especially in processing large-scale text files confirming that bash scripting language is appreciated for solving simple, everyday tasks in bioinformatics [8]. Since Bash programming skills are required to efficiently analyse and present sequence data, which may be a barrier for many researchers [9, 10], this manuscript provides a collection of simple and popular hacks dedicated to users with very basic experience in the area of the Linux command line. It includes a description of elemental biological data formats and examples of how to manipulate this kind of data. All examples (data files, scripts) presented in this

* Address correspondence to this author at Biostatistics Group, Department of Genetics, Wroclaw University of Environmental and Life Sciences; Kozuchowska 7, 51-631 Wroclaw, Poland;
E-mails: magda.mielczarek@upwr.edu.pl

manuscript are available online (https://github.com/bczech/bio-cli).

2. LINUX COMMAND LINE AVAILABILITY

Running or connecting to the GNU/Linux CLI may be done in multiple ways. (1) On a local computer, the easiest way is to install the GNU/Linux operating system and to run a terminal, which is a text-based input and output device for all operations. There is a whole range of distributions (*i.e.*, versions of the operating system) that can be used. (2) Windows operating systems users can run GNU/Linux as a Virtual Machine (VM). There are several virtualization tools available; the most popular are the Oracle VirtualBox (www.virtualbox.org), and the VMWare Workstation (www.vmware.com). Another method is to use a container – operating system virtualization with dedicated software. The most popular platform of the container is Docker (https://www.docker.com/) with a dedicated bioinformatics repository called BioContainers (https://biocontainers.pro/). (3) Nevertheless, since the increasing number of institutions begins to collect large biological datasets, the amount of biological data is huge; therefore often, instead of being stored and processed locally, cloud-based solutions are used [11, 12]. Most GNU/Linux systems run OpenSSH, a Secure Shell service, which represents an application that allows a user to connect remotely to the UNIX-based operating system. On Windows, the PuTTY software is recommended (www.putty.org) to establish OpenSSH connections, as it is free and easy to use. On macOS and Linux, simply entering an ssh command in a terminal window followed by server name or IP address in the terminal is sufficient.

3. SELECTED BIOLOGICAL DATA FORMATS

Numerous biological text-based data formats exist to keep data readable for humans and software. In this chapter, we describe four data formats widely used in bioinformatics applications, which are used here as input in command line data processing. In contrast to regular text, formatted data has to adhere to strict rules of formatting. This allows for automated processing by standard tools.

3.1. FASTA Format

FASTA is one of the most popular and one of the simplest formats in bioinformatics. FASTA is used to store DNA, RNA or protein sequence information (we use DNA information here) as plain text. Sequences can be of any length – from a few bases to whole chromosomes. A typical use of the FASTA format is to contain a whole genome assembly. Chromosomes are listed consecutively. Each chromosome is listed first with a name followed by one or more lines with sequence information. Each sequence starts on a line by itself with the greater-than character (“>”) followed by the identification number or name and description of the sequence. The next line is the first line with the actual nucleotide or amino-acid sequence, with each letter representing one nucleotide or one amino acid. Additional lines of sequence usually follow. The typical extension of a file in FASTA format is “.fasta” or “.fa” [13]. In contrast to Windows, command line applications under Linux do not rely on file extensions; however, the use of extensions is still common in order to identify the format of the file content. An example of a nucleotide sequence belonging to *Ursus arctos* (GenBank ID: AM411403.1) [14] is shown below (*Case 1*). The presented file has only one sequence, a multi-sequence FASTA file is created in Section 4.1 (*Case 12*).

>AM411403.1 Ursus arctos mitochondrial D-loop, isolate TAK4

ACTACTATTTTACTCCATGTCCTATTCATTTCATATATACCATTCTATGTACTGTACTATCACAGTATGT

CCTCGAATACTTTCCCCCCCCTATGTATATCGTGCATTAATGGCGTGCCCCATGCATATAAGCATGTACA

TACTGTGCTTGGTCTTACATGAGGACCTGCATTTTAGAAGTTTATCTCAGGTGTATAGTCTGCAAGCATG

TATTTCACTTAGTCCGGGAGCTTAATCACCAGGCCTCGAGAAACCATCAATCCTTGCGAGT

3.2. FASTQ Format

FASTQ [15] is the most popular format used to store high throughput sequencing data and the corresponding nucleotides qualities. This format contains information about short sequences (called reads) and their features. Each read is represented by four lines. The first line begins with a “@” character followed by information about the read (identifier, sequencing process features). The second line contains the actual nucleotide or protein sequence. The third line begins with a “+” optionally followed by information duplicated from the first line. The fourth line contains information on the phred-scaled quality score of each base in the read, encoded as a single ASCII character. The quality is expressed as the probability of incorrectly called base. The typical extension of a FASTQ file is “.fq” or “.fastq”. Moreover, due to the huge size of sequencing data, FASTQ files are often compressed using the gzip or bzip2 tools, such that the final extension is “.fq.gz”, “.fastq.gz”, “.fq.bz2”, or “.fastq.bz2”. An example of a single read in the FASTQ format (NCBI ID: SRR5078057) [16] is shown below (*Case 2*).

@SRR5078057.1 HWI-ST:6:1101:1149:1947/1

NGCGACCTGAACCTCTACAACAAGGAGTCCAAGCTGTCCTACTTCACCGA

+

#4=DDFFFHHGHHJJJJJJJJJIJJJHIIJIJJJJIJJJJJJJJJJJB

3.3. Variant Call Format (VCF)

Variant Call Format (VCF) [17] is a standard for storing information about DNA polymorphisms (*e.g.*, SNPs, insertions, deletions and structural variants), and information on individuals’ genotypes. The actual list of polymorphisms is preceded by several lines forming a header starting with a “##” string. The header includes descriptions of the chromosomes and provides a definition for variables used to describe polymorphisms in the INFO fields. The header is followed by a single line starting with “#”, which defines the record layout for the following columns consisting of data lines, each with information on one polymorphism. These data lines contain eight tab-delimited, mandatory columns: chromosome or

scaffold identifier (CHROM), the variant position expressed in base pairs (POS), variant ID (ID), a nucleotide(s) corresponding to the reference allele(s) (REF), a nucleotide representing alternative allele(s) (ALT), a quality score quantifying the polymorphism being a false positive (QUAL), filter status (FILTER), and additional information (INFO). The content of the last column is software dependent, but typically, it contains information about the type of polymorphism (*e.g.*, SNP, SNV, indel, duplication). Missing values are coded by a dot character. An example of the VCF format (NCBI ID: GCA_000001405.27) is provided below (*Case 3*).

```
##fileformat=VCFv4.1
##fileDate=20180307
##source=ensembl;version=92;url=http://e92.ensembl.org/Homo_sapiens
##reference=ftp://ftp.ensembl.org/pub/release-92/fasta/Homo_sapiens/dna/
## [further header lines]
#CHROM POS ID REF ALT QUAL FILTER INFO
1 10177 rs367896724 A AC . . dbSNP_150;TSA=insertion;E_Freq;E_1000G;MA=C;MAF=0.425319;MAC=2130;EAS_AF=0.3363;EUR_AF=0.4056;AMR_AF=0.3602;SAS_AF=0.4949;AFR_AF=0.4909
1 10235 rs540431307 T TA . . dbSNP_150;TSA=insertion;E_Freq;E_1000G;MA=A;MAF=0.00119808;MAC=6;EAS_AF=0;EUR_AF=0;AMR_AF=0.0014;SAS_AF=0.0051;AFR_AF=0
[further body file lines]
```

3.4. Variant Effect Predictor Software Input and Output Formats

The Variant Effect Predictor (VEP) tool predicts the functional consequences of a polymorphism [18, 19]. The default input format is a whitespace-separated file containing the following columns: chromosome, variant start position and variant end position, both expressed in base pairs, variant polymorphism and optional information such as the strand and the name of the variant. The example below (*Case 4*) shows an indel variant (line 1), SNPs (lines 2-7), and a structural polymorphism (line 8). The example comes from the official VEP manual website (www.ensembl.org/info/website/upload/var.html).

```
1 881907 881906 -/C +
5 140532 140532 T/C +
12 1017956 1017956 T/A +
2 946507 946507 G/C +
14 19584687 19584687 C/T -
19 66520 66520 G/A + var1
8 150029 150029 A/T + var2
1 160283 471362 DUP
```

The default output format consists of a header starting with a "#" character. The body of the file contains 14 tab-delimited columns including: the variant polymorphism from the input, variant position, variant allele used to assign the consequence, gene ID containing or closest to the variant, transcript ID, Sequence Ontology [20] term assigned to the variant, and the predicted biochemical consequence of the variant. Missing values are indicated by '.'. The example below (*Case 5*) presents the first three lines of a VEP output file. It contains the header (starting with #) and genomic annotation for the structural duplication variant. Based on the presented fragment, the duplication is located close to the gene ENSG00000222623 ("upstream_gene_variant") and overlaps with the gene ENSG00000236601 ("non_coding_transcript_exon_variant,intron_variant"). The output was generated by VEP based on the input shown in the previous example (*Case 4*).

```
#Uploaded_variation Location Allele Gene Feature Feature_type Consequence cDNA_position CDS_position Protein_position Amino_acids Codons Existing_variation Extra
1_160283_duplication 1:160282-471362 duplication ENSG00000222623 ENST00000410691 Transcript upstream_gene_variant - - - - - - IMPACT=MODIFIER;SYMBOL=RNU6-1100P;BIOTYPE=snRNA;DISTANCE=2396;STRAND=-1;SYMBOL_SOURCE=HGNC;HGNC_ID=HGNC:48063
1_160283_duplication 1:160282-471362 duplication ENSG00000236601 ENST00000412666 Transcript non_coding_transcript_exon_variant,intron_variant - - - - - - IMPACT=MODIFIER;SYMBOL=AL732372.1;BIOTYPE=lincRNA;EXON=1-2/2;INTRON=1/1;STRAND=1;SYMBOL_SOURCE=Clone_based_ensembl_gene;TSL=1
[further body lines]
```

4. COMMAND LINE TOOLS

A description of most commands can be displayed directly in the terminal using the ***man*** command. As the example below (*Case 6*) shows, when a terminal is opened, a command prompt (typically) consisting of a "$" character appears, indicating readiness to accept commands. In the example presented, the ***man*** command is used to display the manual of the ***ls*** command. Normally, the manual page is displayed by means of a program called ***less***. One can exit ***less*** using the ***q*** key and move one page forward or backward using the ***f*** and ***b*** keys, respectively.

```
username@hostname:~$ man ls
```

In the following, we show the usage of example commands useful for biological data editing.

4.1. Example 1: Merging Multiple Files Into One (The FASTA Format)

A reference genome is the complete sequence of an official genome of a given species. FASTA files containing a reference genome can be downloaded from publicly available databases, *e.g.*, NCBI (www.ncbi.nlm.nih.gov) or Ensembl (www.ensembl.org). Often separate chromosomes are stored in

separate files, each containing unmasked, masked and soft-masked genomic DNA sequences for a chromosome or a region not (yet) assigned to any chromosome. Although the files can be downloaded using a web browser, it is usually more convenient and faster to use the ***wget*** command followed by the web address of the file we want to download. An example of using the ***wget*** command can be found below (*Case 7*). The command downloads a FASTA file for mouse chromosome 1 from the Ensembl database. The output "Resolving…" shows that ***wget*** is trying to connect to the remote server and about to start the download. The progress bar keeps informing about the status of the process. The last output line indicates that the file is saved.

```
username@hostname:~$ wget ftp://ftp.ensembl.org /pub/release-

94/fasta/mus_musculus/dna/Mus_musculus.GRCm38.dna. chromosome.1.fa.gz

Resolving ftp.ensembl.org (ftp.ensembl.org)... 193.62. 193.139

Connecting to ftp.ensembl.org (ftp.ensembl.org)|193. 62.193.139|:21... connected.

Logging in as anonymous ... Logged in!

==> SYST ... done. ==> PWD ... done.

==> TYPE I ... done. ==> CWD (1) /pub/release-94/fasta/mus_musculus/dna ... done.

==> SIZE Mus_musculus.GRCm38. dna.chromosome.1.fa.gz ... 58345453

==> PASV ... done. ==> RETR Mus_musculus.GRCm38.dna.chromosome.1.fa.gz ... done.

Length: 58345453 (56M) (unauthoritative)

100%[==========================================
==========================================
==========>] 58,345,453 47.9MB/s in 1.2s

2020-10-27 10:12:08 (47.9 MB/s) - 'Mus_musculus.GRCm38.dna.chromosome.1.fa.gz' saved [58345453]
```

The ***wget*** command may be modified to save more files (*Case 8*). The example below downloads all unmasked chromosomes, excluding other genomic elements (*e.g.*, mitochondrial DNA). The chromosome names are provided in curly braces ({}), while [1-9XY] matches any one character between the brackets ([]). The sequence "1-9" matches any one digit. Thus, the command downloads autosomes from 1 to 9 and sex chromosomes X and Y. "?" is a wild card. It matches any single character. Therefore, all autosomes whose names begin with "1" (*e.g.*, 13), followed by any single character, are retrieved. Here, the reference genome files in the FASTA format are named according to the following pattern: (Species).(reference_genome_version).(molecule_type).chrom osome.(chromosome name).fa.gz. Thus, the file ***Mus_musculus.GRCm38.dna.chromosome.1.fa.gz*** represents the DNA sequence of the entire first chromosome of the house mouse corresponding to the GRCm38 version of the reference genome.

```
username@hostname:~$wget ftp://ftp.ensembl.org/pub/release-94/fasta/mus_musculus/dna/ Mus_musculus.GRCm38.dna.chromosome.{[1-

9XY],1?}.fa.gz
```

The file names end with .gz. which indicates that the files are compressed. After downloading, the ***gunzip*** command must be used to uncompress the files (*Case 9*).

```
username@hostname:~$ gunzip Mus_musculus.GRCm38.dna.chromosome.*.fa.gz
```

The asterisk ("*") is a wildcard, which matches zero or more occurrences of any characters. All files with names starting with "Mus_musculus.GRCm38.dna.chromosome." and ending with ".fa.gz" will be uncompressed. Tools that allow processing and viewing of compressed files exist (*e.g.*, the ***zcat*** command). Listing of the content of the current directory is done by the ***ls*** command (*Case 10*).

```
username@hostname:~$ ls

Mus_musculus.GRCm38.dna.chromosome.10.fa Mus_musculus.GRCm38.dna.chromosome.2.fa

Mus_musculus.GRCm38.dna.chromosome.11.fa Mus_musculus.GRCm38.dna.chromosome.3.fa

Mus_musculus.GRCm38.dna.chromosome.12.fa Mus_musculus.GRCm38.dna.chromosome.4.fa

Mus_musculus.GRCm38.dna.chromosome.13.fa Mus_musculus.GRCm38.dna.chromosome.5.fa

Mus_musculus.GRCm38.dna.chromosome.14.fa Mus_musculus.GRCm38.dna.chromosome.6.fa

Mus_musculus.GRCm38.dna.chromosome.15.fa Mus_musculus.GRCm38.dna.chromosome.7.fa

Mus_musculus.GRCm38.dna.chromosome.16.fa Mus_musculus.GRCm38.dna.chromosome.8.fa

Mus_musculus.GRCm38.dna.chromosome.17.fa Mus_musculus.GRCm38.dna.chromosome.9.fa

Mus_musculus.GRCm38.dna.chromosome.18.fa Mus_musculus.GRCm38.dna.chromosome.X.fa

Mus_musculus.GRCm38.dna.chromosome.19.fa Mus_musculus.GRCm38.dna.chromosome.Y.fa

Mus_musculus.GRCm38.dna.chromosome.1.fa
```

If the genome sequence of all chromosomes is required as a single file, the chromosome-specific (as is typically the case for whole genome sequence analysis) FASTA files can be merged using the ***cat*** command, where the asterisk ("*") denotes "any characters" (*Case 11*).

```
username@hostname:~$ cat Mus_musculus.GRCm38.dna.chromosome.*fa > M.musculus.GRCm38.ref.fa
```

By default, the merged files would be written to the terminal. Given the size of a typical genome, this is not desirable. Instead, the ">" character redirects the output to a file, ***Mus_musculus.GRCm38.dna.ref.fa*** merging all files with names starting by "Mus_musculus.GRCm38.dna.chromosome.

" and ending with the ".fa" extension. By default, the ***cat*** command concatenates files in alphabetical order, so particular chromosomes will be ordered as: 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1, 2, 3, 4, 5, 6, 7, 8, 9, X, Y. In order to arrange the files in chromosome order, a ***for*** loop can be used. An example is provided below (*Case 12*).

username@hostname:~$ for chr in {1..19} X Y; do cat Mus_musculus.GRCm38.dna.chromosome.$chr.fa done > M.musculus.GRCm38.ref.num.order.fa

The first command says: in the first iteration of the loop, display the content of the Mus_musculus.GRCm38.dna. chromosome.1.fa file, in the next iteration, display the content of the file Mus_musculus.GRCm38.dna. chromosome.2.fa, and so on. As the files are displayed in the terminal, save them in the new file called M.musculus.GRCm38.ref.num.order.fa. The newly created file is in multi-sequence FASTA format.

4.2. Example 2: File Content Searching (The FASTA Format)

The regular expression is a very useful tool for pattern matching in a sequence of characters. The simplest example may be a string of characters such as "bioinfo" which would match, *e.g.*, "bioinfo" or "bioinformatics", but not "biotechnics", "Bioinfo" (because of the case sensitivity) or anything else that does not contain "bioinfo" string. The ***grep*** command searches files for lines containing a pattern of interest. It may be, for example, used to check the chromosomes order by displaying only those lines of M.musculus.GRCm38.ref.num.order.fa which contain a greater than (">") character, which corresponds to lines containing sequence description (only the first five lines are shown here). Please note, that special characters such as a greater-than character (">") have to be given in quotes (*Case 13*). Otherwise, it indicates a redirection of output as in the previous example (*Case 12*).

username@hostname:~$ grep '>' M.musculus.GRCm38. ref.num.order.fa

>1 dna:chromosome chromosome:GRCm38:1:1:195471 971:1 REF

>2 dna:chromosome chromosome:GRCm38:2:1:182113 224:1 REF

>3 dna:chromosome chromosome:GRCm38:3:1:160039 680:1 REF

>4 dna:chromosome chromosome:GRCm38:4:1:156508 116:1 REF

>5 dna:chromosome chromosome:GRCm38:5:1:151834 684:1 REF

4.3. Example 3: File Reformatting By Column Extraction (The VCF Format)

The file used in this example was downloaded from the Ensembl database (ftp://ftp.ensembl.org/pub/release-94/variation/vcf/homo_sapiens/). It contains all known short DNA polymorphisms in *Homo sapiens*. In the example, we use the first 100,000 lines of this file. Different programmes for data analysis require different inputs, so reformatting the original file is an essential skill. Assume we need some information about SNPs kept in the VCF file (*Case 14*).

username@hostname:~$ grep -v "^#" 1000GENOMES-phase_3.100000.vcf | awk '$5 !~/,/ && $8 ~/SNV/ {print $1,$2,$2,$4"/"$5}' > 1000GENOMES-phase_3.100000.edited.txt

username@hostname:~$ head -n 5 1000GENOMES-phase_3.100000.edited.txt

1 10505 10505 A/T

1 10506 10506 C/G

1 10511 10511 G/A

1 10539 10539 C/A

1 10542 10542 C/T

The pipe character (|) directs the output of a command to the left of the pipe to become the input of the command to the right of the pipe. In the first command, the file header is removed, by ignoring all lines, which start with a "#". It is done using the ***grep -v*** command. The ^ character denotes the beginning of a line, so the pattern "^#" only matches lines starting with a "#" character. The output of the ***grep*** command is not displayed; instead, it forms the input for the next command ***awk***. Next, only lines containing polymorphisms classified as "SNV" (meaning Single Nucleotide Variant) are kept by ***awk***, which checks if column 8 contains the "SNV" pattern ($8 ~/SNV/). Moreover, ***awk*** excludes (***!~***) those lines where column 5 includes the comma character being a separator of alternative alleles ($5 !~/,/). In practice, SNVs with more than one allele are excluded. Columns containing information about chromosome location (column 1), SNV position (column 2), and genotype (columns 4 and 5) are extracted by ***awk*** and saved into a new file named 1000GENOMES-phase_3.100000.edited.txt. This file contains two copies (as start and end positions) of column 2; columns are separated by a space character (by default, ***awk*** separates fields by a comma). The alleles are separated by a "/" character. The newly created file is in one of the input formats accepted by Variant Effect Predictor (VEP) software. In order to check the file content, five lines (***-n*** flag) of the newly created file are displayed on the terminal by the ***head*** command.

4.4. Example 4: Information Extraction (The VEP Output Format)

From the file in the VEP output format, the list of genes (column 6) containing polymorphisms can easily be extracted by ***awk***. Nevertheless, since one polymorphism may be located in multiple transcripts of the same gene, gene names may occur multiple times. To select unique gene names, the ***sort*** and ***uniq*** commands are used. In the following example (*Case 15*), gene names are sorted alphabetically, then unique gene names are saved to a file. The ***uniq*** command removes all but one copy of adjacent identical lines. To remove all duplicates, the file, therefore, needs to be sorted. This is done using the ***sort*** command.

username@hostname:~$ grep -v "^#" 1000GENOMES-phase_3.100000.edited.vep | awk '{print $4}' | grep -v "-" | sort | uniq > 1000GENOMES-phase_3.100000.genes.vep

username@hostname:~$ head -n 5 1000GENOMES-phase_3.100000.genes.vep

ENSG00000008128

ENSG00000008130

ENSG00000067606

ENSG00000078369

ENSG00000078808

The ***wc –l*** command counts the number of lines in the input. This can be used to count the number of genes containing polymorphisms without saving them to a file first. As the following example (*Case 16*) shows, the newly created list of genes is not saved to a file, but rather piped to the command ***wc -l***. This counts the number of lines (genes). It works because, in this particular dataset, each line corresponds to one gene due to the use of ***sort | uniq***.

username@hostname:~$ grep -v "^#" 1000GENOMES-phase_3.100000.edited.vep | awk '{print $4}' | grep -v "-" | sort | uniq | wc -l

4.5. Example 5: Data filtering (the VEP output format)

In the following example (*Case 17*), only genes containing polymorphisms, causing a frameshift in translation (column 7) are processed. The ***awk*** command displays the name of the gene (column 4) with the consequence assigned to "frameshift_variant" (column 7). The ***sort*** and ***uniq*** commands remove duplicate gene names from the list.

username@hostname:~$ awk '$7~"frameshift_variant" {print $4}' 1000GENOMES-phase_3.100000.edited.vep | sort | uniq

4.6. Example 6: Identifying Variants Shared Between Two VCF Files

Assume we would like to find SNPs common to two separate VCF files. Columns defining SNPs are 1 (CHR), 2 (POS), 4 (REF) and 5 (ALT). All of these have to be taken into account in the comparison (Case *18*).

username@hostname:~$ comm -12 <(grep -v '^#' 1000GENOMES-phase_3.100000.vcf | awk '{print $1,$2,$4,$5}' | sort) <(grep -v '^#' investigated_sample.vcf | awk '{print $1,$2,$4,$5}' | sort)

1 1000018 G A

1 1000079 A G

1 1000112 G T

1 1000170 C T

1 1000242 C A

For each file, the header (lines starting with a "#") is removed by ***grep –v.*** Next, columns are extracted by the ***awk*** command. The command ***comm*** with flags -12 prints only lines present in both files "1000GENOMES-phase_3.100000.vcf" and "investigated_sample.vcf". Only the five first lines of output are shown.

4.7. Example 7: Counting The Number Of Reads And Their Average Length In A FASTQ File

The command below (*Case 19*) counts the total number of reads as well as their average length.

username@hostname:~$ awk 'NR%4==2 {reads++; readlen += length($0)} END {print reads; print readlen/reads}' SRR5078057_1.fastq

Since a sequence is stored in the second of four lines corresponding to each read, it may be extracted using modulo operator '%' (remainder of integer division). 'NR' is the line number of the current line read. By the condition 'NR%4==2', we extract only lines whose remainder after division by 4 (the number of lines describing a single read) is equal to 2 (*i.e.*, the line which contains a sequence line). Then, the number of reads is incremented by ***reads***++. The lengths of reads are summed (***readlen += length($0)***). The last thing ***awk*** does is execute the commands following the 'END' condition. Here, ***awk*** prints the number of reads (***print reads***) and the average read length (***print readlen/reads***) in a given FASTQ file. Note that for compressed files, you must first uncompress them or read them using ***zcat*** or ***bzcat*** – depending on the type of compression (Case *20*).

zcat SRR5078057_1.fastq.gz | awk 'NR%4==2 {reads++; readlen += length($0)} END {print reads; print readlen/reads}'

4.8. Example 8: Conversion Of The FASTQ Format To The FASTA Format

Sed is a stream editor used to transform text. In this example, it is used to convert FASTQ to FASTA file by reducing the number of lines corresponding to each nucleotide sequence. Normally, ***sed*** prints the input lines without any modification. With the '-n' option, ***sed*** prints only what is defined by the 'p' option. In the example below, the ***1~4*** pattern selects the first line out of 4. In this line, the initial '@' character is replaced by a '>', since 's' before the pattern indicates substitution. After the semicolon, the ***2~4p*** pattern indicates that every fourth line starting from the second line (*i.e.*, the line containing the sequence) should be printed. The effect is written in the new file called SRR5078057_1.fasta (Case *21*).

username@hostname:~$ sed -n '1~4s/^@/>/p; 2~4p' SRR5078057_1.fastq > SRR5078057_1.fasta

username@hostname:~$ head -n 6 SRR5078057_1.fasta

>SRR5078057.1 HWI-ST:6:1101:1149:1947/1

NGCGACCTGAACCTCTACAACAAGGAGTCCAAGC TGTCCTACTTCACCGA

>SRR5078057.2 HWI-ST:6:1101:2495:1939/1

NGGCCGTCGGACTGCTCTGTGTATCAGCAGCGCTG CTGGTGCGACAGCGG

>SRR5078057.3 HWI-ST:6:1101:2340:1969/1

TGCGCTTACAAACTAATTAATAAATTAATAGTTAG

CTTAAAAAGAGGCTT

4.9. Example 9: Transformation Of Columns Using A "Key File"

A common challenge is that one has to update a column in a text file with information from another source. ***Awk*** can be used to transform values in a given column using information from a "key file", which contains pairs of old and new values. In the following example, the genes.txt file is the "key file" containing the list of Ensembl gene IDs with corresponding genes symbols. The file polymorphisms.txt contains SNPs located in some of these genes, but only Ensembl gene ID is provided. ***Awk*** transforms Ensembl gene IDs to gene symbols in polymorphism_in_genes.txt file (C*ase 22*).

username@hostname:~$ cat genes.txt

ENSG00000227232 FAM39F

ENSG00000278267 hsa-mir-6859-1

ENSG00000284332 hsa-mir-1302-2

ENSG00000284332 MIRN1302-2

ENSG00000237613 F379

username@hostname:~$ cat polymorphism_in_genes.txt

1 14404 14404 G/A ENSG00000227232

1 34570 34570 C/G ENSG00000237613

username@hostname:~$ awk 'BEGIN{OFS="\t"}; NR==FNR{k[$1]=$2;next}; {$5=k[$5]; print}' genes.txt polymorphism_in_genes.txt

The tab character is defined to be an output field separator by the OFS="\t" option. The syntax ***NR==FNR{k[$1]=$2;next}*** concerns the "key file" (genes.txt) and creates the array (***k***) containing gene symbols from the second column (***$2***) indexed by Ensembl gene IDs from the first column (***$1***). Therefore, the number of array elements is the same as the number of lines in the genes.txt file. The syntax ***{$5=k[$5]; print}*** assigns the values from array ***k*** indexed by column 5 of polymorphism_in_genes.txt to this column and prints the file with new values (gene symbols) instead of Ensembl IDs.

5. BASH SCRIPTS

All the commands described may be saved in a text file. Such a file is called a script. The commands in the script are executed as if they were input on the command line. In "*Example 5*," commands are presented in the form of a script. A program executes all the commands in a script. Linux operating system has several shells. The most widely used is the bash (Bourne-Again Shell). Each bash script starts with the string "#!/bin/bash". This indicates that commands in the script should be executed using the bash shell. The ***echo*** command displays the string of characters defined in quotation marks (*Case 17*).

#!/bin/bash

echo "This script looks for frameshifts"

awk '$7~"frameshift_variant" {print $4}' \

1000GENOMES-phase_3.100000.edited.vep| sort | uniq

The above code must be saved in the text file script_eg5.sh. To execute the commands, the name of the file must be entered at the command prompt (*Case 23*). The 2nd line of the above script is executed printing the string "This script looks for frameshifts" by the ***echo*** command. Then the execution of the 3rd line of the script results in printing the list of gene names, which contain a frameshift variant, defined by the ***awk*** command.

username@hostname:~$ bash script_eg5.sh

This script looks for frameshifts

ENSG00000162571

ENSG00000162576

ENSG00000176022

ENSG00000184163

ENSG00000187583

ENSG00000197530

ENSG00000228594

6. BIOAWK

The development of sequencing technology and the huge need for various types of data collecting resulted in many tools dedicated to the processing of sequencing data. One of these tools is ***bioawk***, being an extension of ***awk*** programming language (https://github.com/lh3/bioawk), which enables the processing of several popular biological formats such as BED [21], SAM [22], VCF, GFF (http://mblab.wustl.edu/GTF22.html) and FASTX (FASTA or FASTQ). It is an easy-to-use ***awk*** with predefined separator fields and variables characteristic for a given biological format. In order to process FASTX files, the following variables are predefined: 'name' (text from '>' character to the first whitespace), 'seq' (sequence, *i.e.*, text from the second line to the end of file or to the next '>' character), 'qual' (in FASTQ format, the 4th line of read), and 'comment' (text described in the first line, except the content of the 'name' variable). For the VCF format following variables are predefined: 'chrom', 'pos', 'id', 'ref', 'alt', 'qual', 'filter', and 'info'. The complete list of all available file formats and their predefined variables is available by using '***bioawk -c help***'.

6.1. Example 10: The Sequence Length And Its GC Content Calculations (FASTX)

This command computes the sequence length and its GC content. This is useful in the analysis of sequence complexity (*Case 24*).

username@hostname:~$ bioawk -c fastx '{print ">" $name $comment; print "Length:", length($seq); print "GC%:", gc($seq)}' SRR5078057_1.fasta

The ***-c*** option 'fastx' file format is defined. Then the '>' character is printed together with the sequence ID ('$name') and other information ('$comment') stored in the first line of each sequence header. Then, the length and GC content percent for each sequence are respectively calculated using '***length***'

and '***gc***' functions. Finally, the input file in FASTA format is provided.

6.2. Example 11: Calculating the Average Quality and the Proportion of Nucleotides Satisfying a Quality Threshold (FASTQ)

Quality control is a standard step of high-throughput sequence data analysis. The following command combination (*Case 25*) computes the mean sequencing quality across all reads in a FASTQ file as well as the proportion of nucleotides with a quality above a certain threshold (in this case, set to 30).

```
username@hostname:~$ bioawk -c fastx '{threshold+=qualcount($qual,30); seqLen+=length($seq)} END {print meanqual($qual), threshold/seqLen}' SRR5078057_1.fastq
```

CONCLUSION

In modern life sciences, handling large-scale datasets is an important skill. One of the most fundamental issues is learning how to easily edit and manipulate files using bash command line tools under the UNIX/Linux operating system. An important advantage of using the bash programming language is that it is system independent, so that the vast majority of commands and scripts are directly portable between computers running different UNIX or Linux distributions and versions. Another advantage is that by using the command line, fast processing of data is possible since it does not have to rely on graphical software, which itself consumes some is CPU time and typically requires continuous user interaction. This ensures a good continuity and portability of the application. We believe that starting from using the basic commands presented above, an interested user can easily extend skills and modify them to meet specific needs of their own analysis, even without deep knowledge of computer programming.

LIST OF ABBREVIATIONS

ASCII = American Standard Code for Information Interchange

Bash = Bourne-Again Shell

CLI = command-line interface

NCBI = National Center for Biotechnology Information

Ssh = Secure Shell

VCF = Variant Call Format

VEP = Variant Effect Predictor

ETHICS APPROVAL AND CONSENT TO PARTICIPATE

Not applicable.

HUMAN AND ANIMAL RIGHTS

No human and Animal were used for studies that are the basis of this research.

CONSENT FOR PUBLICATION

Not applicable.

AVAILABILITY OF DATA AND MATERIALS

Not applicable.

FUNDING

None.

CONFLICT OF INTEREST

The authors declare no conflict of interest, financial or otherwise.

ACKNOWLEDGEMENTS

This work was supported by Wroclaw Centre of Biotechnology programme, The Leading National Research Centre (KNOW) for years 2014-2018, as well as Poznan Supercomputing and Networking Centre.

REFERENCES

[1] Ekmekci B, McAnany CE, Mura C. An introduction to programming for bioscientists: A python-based primer. PLOS Comput Biol 2016; 12(6)e1004867 [http://dx.doi.org/10.1371/journal.pcbi.1004867] [PMID: 27271528]

[2] Visser MD, McMahon SM, Merow C, Dixon PM, Record S, Jongejans E. Speeding up ecological and evolutionary computations in R; essentials of high performance computing for biologists. PLOS Comput Biol 2015; 11(3)e1004140 [http://dx.doi.org/10.1371/journal.pcbi.1004140] [PMID: 25811842]

[3] Lee J, Heath LS, Grene R, Li S. Comparing time series transcriptome data between plants using a network module finding algorithm. Plant Methods 2019; 15: 61. [http://dx.doi.org/10.1186/s13007-019-0440-x] [PMID: 31164912]

[4] Kesharwani RK, Chiesa M, Bellazzi R, Colombo GI. CBS-miRSeq: A comprehensive tool for accurate and extensive analyses of microRNA-sequencing data. Comput Biol Med 2019; 110: 234-43. [http://dx.doi.org/10.1016/j.compbiomed.2019.05.019] [PMID: 31207557]

[5] Alberdi A, Gilbert MTP. A guide to the application of Hill numbers to DNA-based diversity analyses. Mol Ecol Resour 2019; 19(4): 804-17. [http://dx.doi.org/10.1111/1755-0998.13014] [PMID: 30947383]

[6] Gancarz Linux and the Unix Philosophy. 2nd ed. Digital Press 2013.

[7] Stallman Free Software Free Society: Selected Essays of Richard M. 3rd ed. Stallman 2015.

[8] Mohammed Y, Palmblad M. Using the object-oriented powershell for simple proteomics data analysis. Methods Mol Biol 2020; 2051: 389-405. [http://dx.doi.org/10.1007/978-1-4939-9744-2_17] [PMID: 31552639]

[9] Ferrero G, Licheri N, Coscujuela Tarrero L, *et al.* Docker4Circ: A framework for the reproducible characterization of circRNAs from RNA-Seq data. Int J Mol Sci 2019; 21(1)E293 [http://dx.doi.org/10.3390/ijms21010293] [PMID: 31906249]

[10] Perampalam P, Dick FA. BEAVR: A browser-based tool for the exploration and visualization of RNA-seq data. BMC Bioinformatics 2020; 21(1): 221. [http://dx.doi.org/10.1186/s12859-020-03549-8] [PMID: 32471392]

[11] Davis-Turak J, Courtney SM, Hazard ES, *et al.* Genomics pipelines and data integration: Challenges and opportunities in the research setting. Expert Rev Mol Diagn 2017; 17(3): 225-37. [http://dx.doi.org/10.1080/14737159.2017.1282822] [PMID: 28092471]

[12] Griffith M, Walker JR, Spies NC, Ainscough BJ, Griffith OL. Informatics for RNA sequencing: A web resource for analysis on the cloud. PLOS Comput Biol 2015; 11(8)e1004393 [http://dx.doi.org/10.1371/journal.pcbi.1004393] [PMID: 26248053]

[13] Lipman D J, Pearson W R. Rapid and Sensitive Protein Similarity Searches. Science (80) 1985; 227(4693): 1435-41. [http://dx.doi.org/10.1126/science.2983426.]

[14] Calvignac S, Hughes S, Tougard C, *et al.* Ancient DNA evidence for the loss of a highly divergent brown bear clade during historical times. Mol Ecol 2008; 17(8): 1962-70. [http://dx.doi.org/10.1111/j.1365-294X.2008.03631.x] [PMID: 18363668]

[15] Cock PJA, Fields CJ, Goto N, Heuer ML, Rice PM. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Res 2010; 38(6): 1767-71. [http://dx.doi.org/10.1093/nar/gkp1137] [PMID: 20015970]

[16] Hua BL, Bell GW, Kashevsky H, Von Stetina JR, Orr-Weaver TL. Dynamic changes in ORC localization and replication fork progression during tissue differentiation. BMC Genomics 2018; 19(1): 623. [http://dx.doi.org/10.1186/s12864-018-4992-3] [PMID: 30134926]

[17] Danecek P, Auton A, Abecasis G, *et al.* The variant call format and VCFtools. Bioinformatics 2011; 27(15): 2156-8. [http://dx.doi.org/10.1093/bioinformatics/btr330] [PMID: 21653522]

[18] McLaren W, Gil L, Hunt SE, *et al.* The ensembl variant effect predictor. Genome Biol 2016; 17(1): 122. [http://dx.doi.org/10.1186/s13059-016-0974-4] [PMID: 27268795]

[19] McLaren W, Pritchard B, Rios D, Chen Y, Flicek P, Cunningham F. Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. Bioinformatics 2010; 26(16): 2069-70. [http://dx.doi.org/10.1093/bioinformatics/btq330] [PMID: 20562413]

[20] Eilbeck K, Lewis SE, Mungall CJ, *et al.* The Sequence Ontology: A tool for the unification of genome annotations. Genome Biol 2005; 6(5): R44. [http://dx.doi.org/10.1186/gb-2005-6-5-r44] [PMID: 15892872]

[21] Quinlan AR, Hall IM. BEDTools: A flexible suite of utilities for comparing genomic features. Bioinformatics 2010; 26(6): 841-2. [http://dx.doi.org/10.1093/bioinformatics/btq033] [PMID: 20110278]

[22] Li H, Handsaker B, Wysoker A, *et al.* The sequence alignment/map format and SAMtools. Bioinformatics 2009; 25(16): 2078-9. [http://dx.doi.org/10.1093/bioinformatics/btp352] [PMID: 19505943]