Practical Linux examples: Exercises

1. Login (ssh) to the machine that you are assigned for this workshop. Prepare working directory, and copy data files into the working directory

```
mkdir /workdir/$USER
cd /workdir/$USER
cp /shared_data/Linux_workshop2/* ./
```

2. Inspect the contents of a compressed gff3 file (human.gff3.gz), using these Linux functions: zcat, head, tail, cut, less

- Inspect the first and last 100 lines of the file, using "head" and "tail" functions to retrieve first and last lines of the file;
- Retrieve line number 1001 to 2000 from the file and write these lines into a new file " mynew.gtf ";
- Inspect the columns 2,3,4,5 and 8 of lines 3901 to 4000, using "cut" function to specify the columns.
- Compare "cut" with "less -S" function. If you use "less" function, remember to exit "less" by pressing "q".

```
zcat human.gff3.gz | head -n 100
zcat human.gff3.gz | tail -n 100
zcat human.gff3.gz| head -n 2000 | tail -n 1000 > mynew.gtf
zcat human.gff3.gz| head -n 4000 | tail -n 100 | cut -f 2-5,8
zcat human.gff3.gz| head -n 4000 | tail -n 100 | less -S
```

3. Count the number of genes listed in the file, using these Linux functions: awk, uniq

- Count the total number of lines in the file using "wc -l" function;
- Count the number of genes listed in the file. First, you need to use "awk" to retrieve lines which 3rd column equals to "gene", then count these lines with "wc -l";
- Count the numbers of features (genes, exons, rRNA and miRNA) in this GFF3 file. The 3rd column of the GFF3 file defines feature categories. The 'grep -v "^#"' command would remove lines starting with "#", then "cut" out the 3rd column of the file, then use "sort | uniq -c" to count the occurrences of each of the categories. Note: "uniq" requires sorted input stream. Always run "sort" before "uniq -c",

```
zcat human.gff3.gz | wc -l
zcat human.gff3.gz | awk '{if ($3=="gene") print}' |wc -l
zcat human.gff3.gz | grep -v "^#" | cut -f 3 | sort | uniq -c
```

4. Convert the GFF3 file to BED file, using only the lines which third column equals to "gene". Then add "chr" to the chromosome name, so that chromosome 1 would be "chr1" instead of "1". Use Linux functions awk and sed for this exercise.

- The BED file requires a minimum of 3 columns (chromosome, start position and end position). It is important that the "start" and "end" positions of BED and GFF3 files are defined differently. In GFF3 file, the start and end positions are both 1-based (the first nucleotide of a sequence is at position 1). In BED file, the start position is 0-based (the first nucleotide of a sequence is at position 0), and the end position is 1-based. When converting GFF3 to BED, you need to subtract 1 from the start positions. In the follow command, the "\" characters are used to split a long command into multiple lines. The expression "BEGIN {OFS = "\t"};'" is to specify that the output stream uses tab as delimiters.
- In this command, the output from awk is piped into "sed", which adds "chr" to the chromosome names. The "^" character in "sed" command is to specify the beginning of a line.

```
#The following three lines are in one single command
zcat human.gff3.gz | \
awk 'BEGIN {OFS = "\t"};{if ($3=="gene") print  $1,$4-1,$5}' | \
sed "s/^/chr/"> mygenes.bed
```

5. Get the size distribution of all genes, using the Linux functions: awk, sort and uniq

- Calculate the nucleotide sequence sizes for each of the genes. The size of a gene is calculated by subtracting "gene start position" (column4) from "gene end position" (column 5), then adding 1 because the GFF3 coordinate system is 1-based. (If a BED file is used to calculate the gene size, you do not need to add 1 because the start position is 0-based in the BED file).
- To get the size distribution, you need to do two more things: 1) Use the expression " int(($5-$4+1)/1000) " to convert the sizes from "base-pair" to "kilo-base-pair"; 2) The expression " LC_ALL=C sort -n | uniq -c" is used to get the histogram. The parameter "-n" is to tell "sort" to do numerical sorting; “-S 2G” is to set the buffer size to 2 gb to speed up the sorting.
- The output is in the file "gene_dist.txt".

```
zcat human.gff3.gz | awk '{if ($3=="gene") print $5-$4+1}'
#The following three lines are in one single command
zcat human.gff3.gz | \
awk '{if ($3=="gene") print int(($5-$4+1)/1000)'} | \
LC_ALL=C sort -S 2G -n | uniq -c > gene_dist.txt
```

6. Count the number of genes and pseudogenes in sliding windows across the whole chromosome, using Bedtools

- BED, GFF3/GTF, SAM/BAM and VCF files are all tab delimited text files for define different kind of features on chromosomes. Software like BEDTools, BEDOPS, VCFTools, SAMtools, BAMtools, Deeptools et al. are often used in combination with basic Linux functions to process these files. In this exercise, you will use BEDTools integrate two files.
- First, you will first generate a text file with sliding windows across the chromosome. The input file for the "makewindows" function is a text file with the length of each chromosomes (hg19.txt). The "-w" and "-s" options specify the window and step size for the sliding windows. In this example, the sliding window size is 1 mb.

- Next, you will count the number of genes and pseudo-genes in each sliding window. To do this, you can use "awk" to select lines which columns 3 equals "gene" or "pseudogene"; then pipe into "bedtools coverage" to count the number of genes in each sliding window. The sorting step after "bedtools coverage" is necessary because bedtools tends to mess up the order. In this case, you sort the file by two columns: column 1(chromosome name) and column 2(position). Note that you need to use "version" style sorting for column 1 (-1,1V) and numerical sorting for column 2 (-k2,2n). The two numbers in "-k1,1V" indicate start and end columns for sorting.
- The "paste" function is used to concatenate the columns, followed by "cut" to output selected columns.

```
bedtools makewindows -g hg19.txt -w 1000000 -s 1000000 > win1mb.bed

zcat human.gff3.gz | \
awk 'BEGIN {OFS = "\t"}; {if ($3=="gene") print $1,$4-1,$5}' | \
bedtools coverage -a win1mb.bed -b stdin -counts | \
LC_ALL=C sort -k1,1V -k2,2n > gene.cover.bed


zcat human.gff3.gz | \
awk 'BEGIN {OFS = "\t"}; \
{if (($3=="processed_pseudogene") || ($3=="pseudogene")) print $1,$4-1,$5}' | \
bedtools coverage -a win1mb.bed -b stdin -counts | \
LC_ALL=C sort -k1,1V -k2,2n > pseudogene.cover.bed


paste gene.cover.bed pseudogene.cover.bed | \
cut -f 1,2,3,4,8 > genecounts.txt
```

7. In this exercise, you will use the fastx software to trim sequence adapters from a fastq file, then get size distribution of the trimmed sequencing reads. You will use Linux functions grep, wc -l, awk.
   • Estimate the percentage of sequencing reads that contain the adapter sequence " AGATCGGAAGAGC". You can use the first 10,000 sequencing reads to estimate. Sometimes the first 10,000 reads are all low quality reads, then this estimation would not be accurate. In that case, you might want to use reads from the middle of the file by using "head -n xxxxx | tail -n xxxxx" command) ;

   • Remove the adapter sequences. Several software can do this, e.g. cutadapt and bbduk. In this exercise, you will use a pretty old tool called fastx_clipper, and write the output into a new fastq file "clean.fastq".
   • Calculate the read length distribution. In a fastq file, each sequence record has 4 lines and the second line of the record is the actual DNA sequence. The "awk" function has a variable "NR" that records the line number for each row. The expression (NR%4) gives you the remainder of NR divided by 4. The statement "if (NR%4 == 2) print length($0)" means "output the size of the second line in every sequence record". The output of awk can then be piped into "LC_ALL=C sort -n | uniq -c" to get the read size distribution.

```
zcat SRR836349.fastq.gz |head -n 40000 | grep AGATCGGAAGAGC | wc -l

zcat SRR836349.fastq.gz | fastx_clipper -a AGATCGGAAGAGC -Q33 > clean.fastq

awk '{if (NR%4 == 2) print length($0)}' clean.fastq | LC_ALL=C sort -n | uniq -c
```

8. Create a shell script .
   In this exercise, you will create a shell script like this:

gzip reads_1.fastq
gzip reads_2.fastq
gzip reads_3.fastq
gzip reads_4.fastq
gzip reads_5.fastq

There are many different ways to create this shell script. Here you will use the “yes” and “paste” commands to create this script "myscript.sh".

```
yes "gzip" |head -n 5 > t1
ls -1 reads*fastq > t2
paste -d " " t1 t2 > myscript.sh
```

· yes "gzip" |head -n 5 > t1 : Create a text file with 5 lines of “gzip”

· ls -1 reads*fastq > t2: Create a text file "t2" with all file names ("1" is the number 1);

paste -d " " t1 t2: Join t1 and t2 files horizontally, with space character as delimiter.