H3ABioNet

Pan African Bioinformatics Network for H3Africa

Introduction to AWK

Sumir Panji / Amel Ghouila

Material adapted from: H3ABioNet and WT NGS Bioinformatics Course Africa 2021

AWK

- Scripting language with text processing capabilities for data extraction, comparison, transformation
- Similar to sed, AWK is available on most unix operating systems
- Named after the initials of its inventors Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan
- Used when one wants to extract fields, make comparisons, filter data and general data wrangling

Some features of AWK

- AWK is great as it allows one to work with delimited data
- Similar to sed, it reads in files line by line
- Different to sed, it splits the line into fields – allows for columns
- A lot of data formats in bioinformatics are delimited with a tab (\t) being a common field separator
- AWK has inbuilt functions that allow one to manipulate these fields – unlike sed i.e. allows one to work with columns within a dataset

Basic AWK syntax

- awk - options ‘optional_selection_criteria { action} ’ input_file

- E.g. awk –F “\t” '{ print $1 }' genes.gff

chr1
chr1
chr1
chr2
chr2
chr3
chr4
chr10
chr10
chrX

- The –F flag indicates the field delimiter – in this case a tab

Basic AWK syntax

- Similar to sed, awk prints the output to the screen, if you want to save the output then will need to redirect it to an outfile
- Different to sed – awk has inbuilt variables called $1, $2, $3 …. that map to the fields separated by the \t delimiter when specified
- Usually useful to determine the number of fields a file has first
- E.g. awk '{print NF}' genes.gff
- Number of Fields (NF) is an inbuilt awk variable that is defined each time awk reads in a line

Basic AWK syntax

- E.g. awk '{print NF}' genes.gff

9
10
9
8
10
9
9
9
9
9

- Strange that they are 2 records in line 2 and 5 that have 10 fields, one in line 4 that has 8 fields and the rest have 9 – any thoughts as to why?

Basic AWK syntax

- Let's look at the records with other records to compare with by using sed to print the first 6 lines of the file:

- sed -n '1,6p' genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
chr1    source5    repeat  10000  14000  1      +    .    name=ALU
chr2    source2    gene    10000  1200   0.95   +    0
chr2    source1    gene    50     900    0.4    -    0
        name=gene2;product=gene2 protein
chr3    source1    gene    200    210    0.8    .    0    name=gene3
```

- Looks like there is a space in fields 2 and 5 between the product name and protein e.g. RecA protein
- The annotation column for record 4 is empty

Basic AWK syntax

- Looks like there is a space in fields 2 and 5 between the product name and protein
- The annotation column for record 4 is empty
- The file is tab separated, in the previous construct we did not tell awk to split the file according to a delimiter: awk '{print NF}' genes.gff
- E.g. awk –F “\t” '{print NF}' genes.gff

9
9
9
8
9
9
9
9
9
9

AWK usage

- awk - options ‘optional_selection_criteria { action} ’ input_file
- Let’s use the optional_selection_criteria to do some filtering on the genes.gff file
- awk -F "\t" '$1 > "chr1" {print $0}' genes.gff

```
chr2    source2  gene     10000  1200  0.95  +  0
chr2    source1  gene     50     900   0.4   -  0
        name=gene2;product=gene2 protein
chr3    source1  gene     200    210   0.8   .  0  name=gene3
chr4    source3  repeat   300    400   1     +  .  name=ALU
chr10   source2  repeat   60     70    0.78  +  .  name=LINE1
chr10   source2  repeat   150    166   0.84  +  .  name=LINE2
chrX    source1  gene     123    456   0.6   +  0
        name=gene4;product=unknown
```

AWK usage

- awk -F "\t" '$1 > "chr1" {print $0}' genes.gff
- awk recognizes mathematical operators such as the greater than sign
- The construct above does two things:
  - Optional_selection_criteria is to use field 1 of the line being read in and check if it is greater then chr1
  - As awk reads in the file line by line, it will print the line ($0) only when the condition is met
- Useful for extracting lines based on a field from a file e.g. all entries for chromosome 2 only
- awk -F "\t" ‘$1 == "chr2" {print $0}' genes.gff

AWK usage

- Also a great way to extract fields from a file and put the input into a new one
- E.g. awk -F "\t" '{print $1,$3,$7}' genes.gff

```
chr1 gene +
chr1 gene -
chr1 repeat +
chr2 gene +
chr2 gene -
chr3 gene .
chr4 repeat +
chr10 repeat +
chr10 repeat +
chrX gene +
```

- Printed out the columns I wanted, and I can send the output to a new file
- Problem - the output does not seem to be tab delimited?

AWK usage

- Problem - the output does not seem to be tab delimited
- To get to the output in \t format, need to change awk's default behaviour – can use the Output Field Separator (OFS)
- E.g. awk -F "\t" 'BEGIN {OFS="\t"} {print $1,$3,$7}' genes.gff

```
chr1	gene	+
chr1	gene	-
chr1	repeat	+
chr2	gene	+
chr2	gene	-
chr3	gene	.
chr4	repeat	+
chr10	repeat	+
chr10	repeat	+
chrX	gene	+
```

AWK usage

- E.g. awk -F "\t" 'BEGIN {OFS="\t"} {print $1,$3,$7}' genes.gff
- BEGIN is an awk variable that tells awk to execute the action in the first set of {} once the first line is read in
- In this case, to set the Output Field Separator variable to be a \t
- awk can also be used to replace every value in a specified field
- E.g. awk -F"\t" 'BEGIN {OFS="\t"} {$2=”H_sapiens"; print $0}' genes.gff

```
chr1    H_sapiens   gene    100     300     0.5     +       0
        name=gene1;product=unknown
chr1    H_sapiens   gene    1000    1100    0.9     -       0
        name=recA;product=RecA protein
chr1    H_sapiens   repeat  10000   14000   1       +       .       name=ALU
chr2    H_sapiens   gene    10000   1200    0.95    +       0
```

AWK usage

- Can combine multiple patterns using the && to mean do if meets criteria 1 “and” criteria 2

- E.g. awk -F"\t" '$1=="chr1" && $3=="gene"' genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
```

- Can also meet criteria 1 “and” criteria 2 “and” criteria 3
- E.g. awk -F"\t" '$1=="chr1" && $3=="gene" && $7=="+"' genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
```

AWK usage

- E.g. awk -F"\t" '$1=="chr1" && $3=="gene"' genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
```

- Can use the || as an “or” condition to mean do if meets criteria 1 “or” criteria 2

- E.g. awk -F"\t" '$1=="chr1" || $3=="gene"' genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
chr1    source5    repeat  10000  14000  1      +    .    name=ALU
chr2    source2    gene    10000  1200   0.95   +    0
chr2    source1    gene    50     900    0.4    -    0
        name=gene2;product=gene2 protein
chr3    source1    gene    200    210    0.8    .    0    name=gene3
chrX    source1    gene    123    456    0.6    +    0
        name=gene4;product=unknown
```

AWK usage

- One can combine multiple conditions, and filter based on numerical values instead of just strings as we have done

- E.g. awk -F"\t" '$1=="chr1" && $3=="gene" && $4 < 1100' genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
```

AWK basic arithmetic

- As awk recognizes mathematical operators, can use it to preform basic calculations based on some criteria

- E.g. to find the length of repeats in the genes.gff file - awk -F"\t" '$3=="repeat" {print $5 - $4 + 1}' genes.gff

```
4001
101
11
17
```

AWK basic arithmetic

- E.g. to find the length of repeats in the genes.gff file - awk -F"\t" '$3=="repeat" {print $5 - $4 **+ 1**}' genes.gff

- The +1 addition is due to the General Feature Format where the sequence numbering starts at 1

(https://www.ensembl.org/info/website/upload/gff.html)

- Different to the BED file format where the sequence numbering starts at 0

(https://m.ensembl.org/info/website/upload/bed.html)

AWK basic arithmetic

- Can use awk to add up the total length of the repeats by using a variable
- E.g. awk -F"\t" 'BEGIN{sum=0} $3=="repeat" {sum = sum + $5 - $4 + 1} END{print sum}' genes.gff → 4130
- A variable called “sum” is set at zero before awk reads in the file
- Each time the line repeat is found, the calculated length of the repeat is added to variable sum
- The END statement tells awk what to do once all the lines in the file have been read – in this instance to print the final value of the variable sum
- Can also use awk’s += operator as a counter e.g. awk -F"\t" 'BEGIN{sum=0} $3=="repeat" {sum =+ $5 - $4 + 1} END{print sum}' genes.gff → 4130

AWK basic arithmetic

- Can use awk to calculate the mean scores of the genes in column 6 of the genes.gff file

- E.g. awk -F"\t" 'BEGIN{sum=0; count=0} $3=="gene" {sum =+ $6; count++} END{print sum/count}' genes.gff → 0.1

- We use a second variable called count is set to zero and adds 1 each time the term gene is matched – this keeps track of the number of matches to gene

- The END statement tells awk divide the total value of sum (0.6) by the number of matches to gene (6) = 0.1

More info and examples on using awk (syntaxes / usage might differ)

- https://www.tutorialspoint.com/awk/index.htm
- https://bioinformatics.cvr.ac.uk/category/awk/
- https://linuxhint.com/category/awk/
- https://www.shortcutfoo.com/app/dojos/awk/cheatsheet

H3ABioNet

Thank you!!!

Acknowledgements: Amel Ghouila and course material adapted from H3ABioNet WT NGS Bioinformatics Course Africa 2021

H3ABioNet

Pan African Bioinformatics Network for H3Africa

Introduction to BASH

Sumir Panji / Amel Ghouila

Material adapted from: H3ABioNet and WT NGS Bioinformatics Course Africa 2021

BASH

- So far we are using different single commands to do various operations on input data files
- One generally tends to do similar / same operations when new input data files are generated
- Would be nice to organize these different commands to be run once as a script, rather then run each command individually
- BASH (Bourne Again Shell) scripting is suitable for putting these command line commands into a script – mainly command line commands
- BASH is useful, but be aware that they are more modern programming / scripting languages such as Python and R that are suitable for more re-used and complicated tasks as they have numerous inbuilt functions and libraries – mainly used for calling command line programs

Some BASH basics

- One creates a bash script using a text editor such as nano, vim, emacs etc

- A bash script should be saved / ends with a .sh extension

- A bash script should have a header line that provides the path the interpreter to execute the script and let unix know that it is a bash script – pretty much standard on most unix systems: #!/usr/bin/env bash

- Should change the .sh bash script file permissions to be executable e.g. chmod +x bash_script.sh

Some BASH basics

- Let's create a simple bash script to print out "Hello World!" called hello.sh

```
#!/usr/bin/env bash
echo "Hello World"

bash hello.sh
Hello World

./hello.sh
bash: ./hello.sh: Permission denied

chmod +x hello.sh

./hello.sh
Hello World
```

Some BASH basics

- There are a couple of ways to run a bash script:
  - bash script_name.sh
  - ./script_name.sh
  - script_name.sh

- The first option is when the script file does not have executable permissions – we generally do want scripts / program files to be executable

- The second option is when the script file has executable permissions

- The first two ways of running a bash script assume you are in the same directory as the script

- One might want to run the script on multiple files in multiple locations – e.g. script_name.sh

Some BASH basics

- One might want to run the script on multiple files in multiple locations

- Can add the location of the script to your PATH variable so it will be globally accessible while you are logged into your terminal and just type in the script name:

```
pwd
/home/manager/course_data/unix/practical/Notebooks/advanced_bash/scripts

cd ../
/hello.sh
bash: ./hello.sh: No such file or directory

export
PATH=$PATH:/home/manager/course_data/unix/practical/Notebooks/advanced_bash/scripts

hello.sh
Hello World
```

Some BASH basics

- The general format of running a bash script would be script_name.sh input_file

- Will use the input file called test_file

test file line 1
test file line 2
another line
more lines
more lines
penultimate line
last line
pwd

Script name: options.sh
#!/usr/bin/env bash
echo filename is: $1
echo
echo First $2 lines of file $1 are:
head -n $2 $1

Some BASH basics

```
#!/usr/bin/env bash
echo filename is: $1
echo
echo First $2 lines of file $1 are:
head -n $2 $1

./options_example.sh test_file 3
filename is: test_file

First 3 lines of file test_file are:
test file line 1
test file line 2
another line
```

Some BASH basics

```
#!/usr/bin/env bash
echo filename is: $1
# store the file name in a variable called $1 and print out the file name
filename is: test_file

echo
#Print out space for formatting

echo First $2 lines of file $1 are:
# store the number of lines provided by the user in a variable called $2
First 3 lines of file test_file are:

head -n $2 $1
# use the values in the variables $2 and $1 for obtaining the number of lines
entered by the user and print them out
test file line 1
test file line 2
```

Some BASH basics

- The variables $1, $2 are user defined and are not specific to bash, unlike the way they work for awk

- Can make the script more user friendly and easier to read by using good variable names:

```
#!/usr/bin/env bash
filename=$1
number_of_lines=$2
echo filename is: $filename
echo
echo First $number_of_lines lines of file $filename are:
head -n $number_of_lines $filename
```

General scripting good practice

Should use inbuilt error flags within bash to terminate the program incase it encounters an error at any line (by default the script will continue to run till the end) using the set function:

```
#!/usr/bin/env bash
set -eu
```

- Should provide help / error messages that inform the user of the options if run incorrectly
- Should have comments – lines in a script that begin with # are ignored by the interpreter and are used for commenting about code
- Use of good / informative variable names

e.g. './options_example.2.shfilename is:

First lines of file are:
head: option requires an argument -- 'n
Try 'head --help' for more information.

General scripting good practice

```
#!/usr/bin/env bash
set -eu
# check that the correct number of options was given.
# If not, then write a message explaining how to use the
# script, and then exit.
if [ $# -ne 2 ]
then
echo "usage: options_example.3.sh filename number_of_lines"
echo
echo "Prints the filename, and the given first number of lines of the file"
exit
fi
# Use sensibly named variables
filename=$1
number_of_lines=$2
# check if the input file exists
if [ ! -f $filename ]
then
echo "File '$filename' not found! Cannot continue"
exit
fi
# If we are still here, then the input file was found
echo filename is: $filename
echo
echo First $number_of_lines lines of file $filename are:
head -n $number_of_lines $filename
```

General scripting good practice

- An inbuilt variable $# which should be equal to 2 options provided to the script, if not equal to 2 (-ne 2), then execute this block of code that starts from then then and terminates at fi (a variable that closes and if statement) and exits providing usage instructions.

```
if [ $# -ne 2 ]
then
echo "usage: options_example.3.sh filename number_of_lines"
echo
echo "Prints the filename, and the given first number of lines of the file"
exit
fi
```

- A check to see if the filename exists / provided correctly, if not then exit the program

```
if [ ! -f $filename ]
then
echo "File '$filename' not found! Cannot continue"
exit
fi
```

General scripting good practice

Without user options:
'./options_example.2.sh
filename is:

First lines of file are:
head: option requires an argument -- 'n
Try 'head --help' for more information

With user options:
./options_example.3.sh

usage: options_example.3.sh filename number_of_lines

Prints the filename, and the given first number of lines of the file

For loop

- Usually working with multiple files and would like to run the same operations on these files at one go, rather then individually
- The for command is very useful for this
- Basic usage: for files in directory/*; do operation $files; done
- for filename in loop_files/*; do wc $filename; done

```
2  8 28 loop_files/file.1
 5 20 70 loop_files/file.2
 6 24 84 loop_files/file.3
 1  4 14 loop_files/file.4
0 0 0 loop_files/file.5
```

Running bioinformatics programs

- A lot of bioinformatics software programs are run via the command line

- They require input files to do computations and then write the results out to an output file

- A good bioinformatics program / software package usually has:
  - A version number and software license
  - Options and parameters to choose from
  - Provides information on the options and usage of the program
  - Documentation on how to use the software the various options
  - Usually a good user base and support

Running bioinformatics programs

- Usually bioinformatics programs are installed on a unix system and are available system wide – might have to set the variable $PATH to access the program outside of the program directory

- Can access by typing in the tool's name e.g. samtools

Program: samtools (Tools for alignments in the SAM format)
Version: 1.10 (using htslib 1.10.2)
Usage: samtools <command> [options]

- bcftools

Program: bcftools (Tools for variant calling and manipulating VCFs and BCFs)
License: GNU GPLv3+, due to use of the GNU Scientific Library
Version: 1.10.2 (using htslib 1.10.2)
Usage: bcftools [--version|--version-only] [--help] <command> <argument>

Running bioinformatics programs

- The usual syntax for running a tool is:

Tool name – command from the tool – optional parameters for that specific command – input file – output file (or print to screen if no file is specified)

- The optional parameters for the specific command are usually provided in the format of unix style flags with a – (option) e.g. sort –u

- Spend some time going through the documentation for a bionformatics tool to determine what it does, required input and output formats, various options, parameters and its usage syntax as they do differ

Common errors

- Common errors when running bioinformatics tools include:
  - Not providing the correct path to the input file – try running the program in the directory which has the input files, or provide the complete path to the file(s)
  - Not specifying an output file or output directory
  - The input file might not have content, or be correctly formatted for the tool to use
- Most bioinformatics tools provide some error messages e.g. bcftools mpileup myfile

Error: mpileup requires the --fasta-ref option by default; use --no-reference to run without a fasta reference

- Do look at these error messages as they will help you determine what the issue is e.g. bcftools mpileup --no-reference myfile

[E::hts_open_format] Failed to open file "myfile" : No such file or directory
[mpileup] failed to open myfile: No such file or directory

Useful links

- http://people.duke.edu/~ccc14/duke-hts-2018/cliburn/Bash_in_Jupyter.html
- https://data-skills.github.io/unix-and-bash/03-bash-scripts/index.html
- https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008645
- https://scg.dgist.ac.kr/wp-content/uploads/2017/02/20170330.pdf
- https://wikis.utexas.edu/display/bioiteam/Shell+Script
- https://bioinformatics-core-shared-training.github.io/shell-genomics/06-writing-scripts/index.html
- http://williamslab.bscb.cornell.edu/?page_id=235

H3ABioNet

Thank you!!!

Acknowledgements: Amel Ghouila and course material adapted from H3ABioNet WT NGS Bioinformatics Course Africa 2021

H3ABioNet

Pan African Bioinformatics Network for H3Africa

Introduction to SED

Sumir Panji / Amel Ghouila

Material adapted from: H3ABioNet and WT NGS Bioinformatics Course Africa 2021

Stream EDitor (SED)

- You can find patterns in files using grep
- What happens if you want to change some characters or patterns?
- What happens if you only want to print certain lines in a file to use for another software program or to check?
- Can be done with SED allows one to this quite easily
- Find SED very useful for finding, substituting and formatting text and files e.g. fasta headers

Basic SED syntax

- sed ‘s/pattern to find/pattern to replace/’ input_file
- The s after sed in the command is for substitution
- E.g. sed 's/chr/Chromosome/' practical/Notebooks/awk/genes.gff
- sed has a couple of default ways of working:
  - sed reads in the file looks for matches of the pattern line by line
  - The output is sent to the standard out / screen line by line
- The output of the above command provides:

```
Chromosome1     source1    gene      100     300     0.5     +     0
        name=gene1;product=unknown
Chromosome1     source2    gene      1000    1100    0.9     -     0
        name=recA;product=RecA protein
Chromosome1     source5    repeat    10000   14000   1       +     .
        name=ALU
Chromosome2     source2    gene      10000   1200    0.95    +     0
```

Basic SED syntax

- Fantastic!!! But a stream of characters my screen does not help me as I need an actual modified file for use

- One generally redirects the output from the screen to a new file

- sed ‘s/pattern to find/pattern to replace/’ input_file > output_file

- E.g. sed 's/chr/Chromosome/' practical/Notebooks/awk/genes.gff > sed_output_genes.gff

Basic SED syntax

- The previous example is a bit misleading for a reason:

- Say I need to format the genes.gff file from its current tab-delimited format to a comma separated one for another program

- sed ‘s/\t/,/' practical/Notebooks/awk/genes.gff

```
chr1,source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1,source2    gene    1000    1100    0.9    -    0
        name=recA;product=RecA protein
chr1,source5    repeat    10000    14000    1    +    .
        name=ALU
```

Basic SED syntax

- Only the first tab was substituted by a comma, and not the rest of the tabs

- Why?

Basic SED syntax

- Only the first tab was substituted by a comma, and not the rest
- Why?
- Recall SED works by reading lines and matching to the first pattern it finds in the line
- For the sed 's/chr/Chromosome/' example – chr appears once on each new line
- For the sed 's/\t/,/' – the tab character appears multiple times on each new line
- sed's default behaviour is to substitute the first match on each new line

Basic SED syntax

- What if we want to replace all the matches to the pattern regardless of the number of times it appears in a new line?

- Use the global flag

- E.g. sed 's/\t/,/**g**' practical/Notebooks/awk/genes.gff

chr1,source1,gene,100,300,0.5,+,0,name=gene1;product=unknown
chr1,source2,gene,1000,1100,0.9,-,0,name=recA;product=RecA protein
chr1,source5,repeat,10000,14000,1,+,.,name=ALU
chr2,source2,gene,10000,1200,0.95,+,0

Counting and extracting lines with SED

- One can use sed to print out specific lines in a file e.g a row
- Say I wanted to extract lines 1, 2 and 3 only from the genes.gff file
- sed '1,3p' practical/Notebooks/awk/genes.gff
- Notice that the substitution flag and slashes are not present as we are just extracting lines, not matching and modifying any characters

Counting and extracting lines with SED

sed '1,3p' practical/Notebooks/awk/genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
chr1    source5    repeat  10000  14000  1      +    .    name=ALU
chr1    source5    repeat  10000  14000  1      +    .    name=ALU
chr2    source2    gene    10000  1200   0.95   +    0
chr2    source1    gene    50     900    0.4    -    0
        name=gene2;product=gene2 protein
chr3    source1    gene    200    210    0.8    .    0    name=gene3
chr4    source3    repeat  300    400    1      +    .    name=ALU
chr10   source2    repeat  60     70     0.78   +    .    name=LINE1
chr10   source2    repeat  150    166    0.84   +    .    name=LINE2
chrX    source1    gene    123    456    0.6    +    0
        name=gene4;product=unknown
```

Counting and extracting lines with SED

- In this case sed printed out the whole file and added lines 1 and then 3 within the file

- Why?

Counting and extracting lines with SED

- Recall sed's default behavior is to print everything out onto the screen

- We can use the -n option to prevent sed's default behaviour of printing everything to the screen

- E.g sed -n '1,3p' practical/Notebooks/awk/genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
chr1    source5    repeat  10000  14000  1      +    .        name=ALU
```

Counting and extracting lines with SED

- “sed -n ‘1,3,p' practical/Notebooks/awk/genes.gff” prints out a range of lines from 1 to 3
- How do I get it to print out specific lines e.g 1-3 and then 5 and 7
- sed -n ‘1,3p; 5p; 7p' practical/Notebooks/awk/genes.gff

```
chr1    source1    gene    100    300    0.5    +    0
        name=gene1;product=unknown
chr1    source2    gene    1000   1100   0.9    -    0
        name=recA;product=RecA protein
chr1    source5    repeat  10000  14000  1      +    .    name=ALU
chr2    source1    gene    50     900    0.4    -    0
        name=gene2;product=gene2 protein
chr4    source3    repeat  300    400    1      +    .    name=ALU
```

Special characters for SED

- I mainly use sed for pattern matching and substitution and formatting of files

- Sed provides a number of useful characters to provide more control over its pattern matching:

- ^ match the start of the line
- $ match the end of the line
- [a-z] characters of the alphabet – used to change cases using the U& (for upper case) and L& for lower case (note can also use the unix command tr for case changing)
- sed -n 'p;n' – print out odd number lines
- sed -n 'n;p' – print out even number lines

Special characters for SED

- sed 's/^/Organism_/g' genes.gff

```
Organism_chr1   source1   gene   100    300    0.5   +   0
        name=gene1;product=unknown
Organism_chr1   source2   gene   1000   1100   0.9   -   0
        name=recA;product=RecA protein
```

- sed 's/$/_Organism/g' genes.gff

```
chr1   source1   gene   100    300    0.5   +   0
        name=gene1;product=unknown_Organism
chr1   source2   gene   1000   1100   0.9   -   0
        name=recA;product=RecA protein_Organism
```

- sed 's/[a-z]/\U&/g' genes.gff
- CHR1 SOURCE1 GENE 100 300 0.5 + 0
  NAME=GENE1;PRODUCT=UNKNOWN
- CHR1 SOURCE2 GENE 1000 1100 0.9 - 0
  NAME=RECA;PRODUCT=RECA PROTEIN

Special characters for SED

- For more specific control, can use the pattern to be matched e.g.
- sed ‘s/^chr*/Organism_/g' genes.gff

```
Organism_chr1   source1   gene   100    300    0.5   +   0
        name=gene1;product=unknown
Organism_chr1   source2   gene   1000   1100   0.9   -   0
        name=recA;product=RecA protein
```

- sed 's/$/_Organism/g' genes.gff

```
chr1   source1   gene   100    300    0.5   +   0
        name=gene1;product=unknown_Organism
chr1   source2   gene   1000   1100   0.9   -   0
        name=recA;product=RecA protein_Organism
```

- sed 's/[a-z]/\U&/g' genes.gff
- CHR1 SOURCE1 GENE 100 300 0.5 + 0
  NAME=GENE1;PRODUCT=UNKNOWN
- CHR1 SOURCE2 GENE 1000 1100 0.9 - 0
  NAME=RECA;PRODUCT=RECA PROTEIN

More info and examples on using SED (syntaxes / usage might differ)

- https://bioinformaticsworkbook.org/Appendix/Unix/unix-basics-4sed.html#gsc.tab=0
- https://dasher.wustl.edu/chem478/software/unix-tools/sed.html
- https://www.grymoire.com/Unix/Sed.html
- https://gist.github.com/ssstonebraker/6140154

H3ABioNet

Thank you!!!

Acknowledgements: Amel Ghouila and course material adapted from H3ABioNet WT NGS Bioinformatics Course Africa 2021