SIB Bash Scripting Course — Exercises, series 3

Fasta -> TSV Converter

Note See `project.pdf` for an intro to the problem with FastA.

These exercises are designed to introduce you to the process of writing a real-world script from scratch, and to make use of the concepts you learned in the course.

Exercise list:

- Exercise 3.0 - Design and Testing
- Exercise 3.1 - Hello, world!
- Exercise 3.2 - Read the First Line
- Exercise 3.4 - Read all the Lines
- Exercise 3.4 - Distinguish Header from Sequence Lines
- Exercise 3.5 - Handle the two Kinds of Lines
- Exercise 3.6 - Two Last Glitches
- Exercise 3.7 - Use the Script

Notes:

- **Exercise material:** all exercise material is found in the `exercises/` directory of this Git repository. We suggest entering this directory when doing the exercises: `cd exercises/`.
- Some of the exercises build on previous exercises, so it's best if they are done in order.
- At the end of some exercises, you will find an **Additional Tasks** section. These sections contain additional tasks for you to complete, **if you still have the time after having completed the main exercise**. These sections will in principle not be corrected in class.
- **Exercise solutions:** all exercises have their solution embedded in this document. The solutions are hidden by default, but you can reveal them by clicking on the drop-down menu, like the one here-below. We encourage you to *not* look at the solution too quickly, and try to solve the exercise without it. Remember you can always ask the course teachers for help.

Exercise solution example

This would reveal the answer...

Exercise 3.0 - Design and Testing

Before we start happily coding away, we should have an idea of what our script needs to do (*specifications*) and how it will do it (*program logic*). It will also help if we have an easy way of making sure our script works correctly (*testing*).

Solution

Specifications

The specifications could look like this:

1. The script should accept FastA on its standard input.
2. It should transform each (multi-line) FastA record into a single-line output record with two fields: (i) the contents of the header (`>`) line, and (ii) the concatenation of the contents of all sequence lines, in order.
3. The two fields should be separated by a single TAB character.
4. The leading `>` of header lines should be dropped.

Example: the following input

```
>DNA 1
ATGCATGC
GAATTCGAATTC
>Protein 2
ALEAIA
CTAEST
```

should be converted to

```
DNA 1\tATGCATGCGAATTCGAATTC
Protein 2\tALEAIACTAEST
```

Testing

In fact, these two snippets of data can be used to *test* our script: put the first into a file `fasta2tsv-test-in.fas` and the second into `fasta2tsv-test-out.tsv`, we can check that the script works by doing

```
diff <(./fasta2tsv.sh < project-test-in.fas) project-test-out.tsv
```

As long `diff` reports a difference, we're not done. Of course, there would typically be more than one test, and they would all be run automatically (guess how this could be done...).

Logic

As for the program logic, here is a possibility in pseudocode:

1. Read the first line, strip the leading '>' and trailing newline, and print it, followed by a TAB

2. While there are still lines to read:
    1. Read one line and strip the trailing newline
    2. - If the line starts with `>` (header):
        1. print a newline (this starts a new record)
        2. print the header line (skipping the `>`), followed by a TAB newline, then print the line
       - Otherwise (sequence), just print the line

NOTE that the *first* header line is treated slightly differently from the other headers: it is not preceded by a newline, since we do not have to separate it from preceding records.

Exercise 3.1 - Hello, world!

Task: create a Bash script named `fasta2tsv.sh`. For now, it should just print the traditional "Hello, world!" (or any message of your choice). It should work like this:

```
./fasta2tsv.sh
Hello, world! # or something
```

Solution

Using your favourite text editor[1], create a file named `fasta2tsv.sh` with the following contents:

```
#!/usr/bin/env bash

printf 'Hello, world!' # could also use echo
```

Then make it executable:

```
chmod +w fasta2tsv.sh
# or: chmod +100 fasta2tsv.sh (geekier)
```

And run it:

```
./fasta2tsv.sh
Hello, world!
```

Exercise 3.2 - Read the First Line

Now we need to get input into our script. First, let's try to read just the first line of input: once we can read one line, it shouldn't be too hard to read them all.

Task: read the first line of FastA input into a variable called `line`, then print that line.

[1] Did I mention it should be Vim?

Solution

```
#!/usr/bin/env bash

read line
printf "%s\n" "$line"
```

Exercise 3.4 - Read all the Lines

Now, let's read *all* the lines, one at a time, and print each line just after we've read it. If we just wanted to copy the lines, we could simply call `cat`, but later one we're going to do different things with different lines, so we need to be able to read each individual line.

Task: read all the lines, one by one, and print each line after reading it.

Solution

```
#!/usr/bin/env bash

while read line; do
  printf "%s\n" "$line"
done
```

This is a repetitive task, so it's a perfect job for a loop. We are going to do "something" (that is, print a line) *while* there is still lines to read, so a `while` loop seems most useful. The `read` command, as we know, will store the next line in the `line` variable; it will fail if there is nothing to read, ending the loop.

Exercise 3.4 - Distinguish Header from Sequence Lines

As discussed in Exercise 3.0 - Design and Testing, we have to treat header lines differently from sequence lines.

Task: for each line, determine if it is a header or a sequence line, and print out "header" or "sequence" accordingly.

Solution

```
#!/usr/bin/env bash

while read line; do
  if [[ ${line:0:1} = '>' ]]; then
    echo header   # simple enough for echo
  else
    echo sequence
  fi
done
```

We have to make a true/false decision, and we need to do something in either situation: this is the textbook use case for `if ... then ... else ... fi`. The criterion is whether the first character of the `line` variable is a `>` or not, so we use it (`${line:0:1}` in a string comparison (`[[ ... = ... ]]`).

Exercise 3.5 - Handle the two Kinds of Lines

Now that we can tell headers apart from sequence lines, we are going to output them differently, as discussed in the beginning.

Task:

- if the current line is a header, (i) print a newline, (ii) print the line itself, followed by a TAB
- if the current line is a sequence line, just print it.

Solution

```
#!/usr/bin/env bash

while read line; do
  if [[ ${line:0:1} = '>' ]]; then
    printf "\n"
    printf "%s\t" "${line:1}"
  else
    printf "%s" "$line"
  fi
done
```

As seen when discussing parameter expansion, `${line:1}` expands to `$line`, but starting from the second character — effectively dropping the leading `>`.

Exercise 3.6 - Two Last Glitches

We're almost there — test the program with `diff` as shown above —, but there are still two problems:

1. There is an extra empty line at the beginning of the input
2. Thee should be a newline at the end of the input.

Task: fix these two problems. Hint: look again at the specifications.

Solution

```
#!/usr/bin/env bash

read line
printf "%s\t" "${line:1}"

while read line; do
```

```
  if [[ ${line:0:1} = '>' ]]; then
    printf "\n"
    printf "%s\t" "${line:1}"
  else
    printf "%s" "$line"
  fi
done

printf "\n"
```

The *first* line (which we'll assume to be a header) must be treated differently from the other headers, so we read it *before* the `while` loop, and print it just like the other headers, **except** that we do not output a newline just before.

To fix the second problem, we simple output a newline after the loop.

Exercise 3.7 - Use the Script

Congratulations! The script should now work as advertised (but you should still do a final check). Now the script can be put to good use.

Try the following tasks:

1. From the file `./data/Spo0A.msa`, extract the record corresponding to ID `Bsph_3503` (as TSV)
2. Find the longest sequence in

Solution

Task 1

```
~/bin/fasta2dsv.sh < Spo0A.msa | grep Bsph_3503
```

Task 2

The only slightly tricky bit is to get the sequence lengths. This can be done e.g. by calling Awk:

```
$ ~/bin/fasta2tsv.sh < sample_01.dna \
  | awk -F $'\t' -v OFS=$'\t' '{print $1, $2, length($2)}' \
  | sort -t $'\t' -k3,3n | tail -1
```

Or we could use Bash itself to compute the length.

Further Developments

Here are some ideas to make the script more realistic:

- Headers — The first liner of a TSV/CSV file typically consists of field names rather than data themselves. Our script could be modified to include them.
- *Optional* headers — But sometimes, headers get in the way (e.g. when sorting by species), so perhaps they should be printed at the user's discretion.
- Other separators: for now fields are TAB-separated, but it would be convenient to output CSV (comma-separated) or for that matter to let the user specify the field separator.