Bash shell scripting: exercises day 1

Exercise list:

- Exercise 1.1 - a complex pipeline
- Exercise 1.2 - a simple shell script
- Exercise 1.3 - tokenizing and quoting
- Exercise 1.4 - parsing
- Exercise 1.5 - parameter expansions
- Exercise 1.6 - brace expansions and command substitution
- Exercise 1.7 - process-substitution

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 part of the 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 1.1 - A complex pipeline

You are given a file that contains genetic sequences in FASTA format that look like the following (see `exercises/data/sample_01.dna` for the full file):

```
>Desulfo_vulgaris_Hilden;tax=d:Bacteria,...,s:Desulfovibrio vulgaris;
ATGGCGAAGAAACCGAGCCTCAGCCCCGAAGAGATGCGCCG
CACCATCGAGCGGAAGTACGGTCTGGGGGCGGTCATGAAAC
TCCCCGTCATCCCCACCGGTTCCATCGGCCTTGACCTCGCA
...
>Mycobac_tubercu_H37Rv;tax=d:Bacteria,...,s:Pseudomonas putida;
ATGACGCAGACCCCCGATCGGGAAAAGGCGCTCGAGCTGGC
CGGCAAAGGTTCGGTGATGCGCCTCGGCGACGAGGCGCGTC
GATCCATCGCACTAGACGTGGCCCTGGGCATTGGCGGCCTG
```

Your task is to **find the number of sequences** of the species with the most

sequences in the file (this may be useful *e.g.* to check for over-representation). The result should be a single number. For instance, if the most represented species has 16 sequences in the file, then your command should output `16`.

Hints:

- In a FASTA file, each sequence starts with a **header line** that can be identified with by its leading `>` character.
- The species name to which each sequence belongs can be found in the header of each sequence, and is prefixed with `s:`. In our example above, the species of the first sequence is `Desulfovibrio vulgaris` - a species of sulfate-reducing bacteria.

Exercise solution

There are a multitude of ways to solve this task, this is just one way to do it:

```
# Commented solution (Note: will not work if you copy/paste because of comments).
grep -o 's:.*;' < data/sample_01.dna | \  # Keep only species names.
  sort | \            # Sort alphabetically. Needed for applying `uniq` to the data.
  uniq -c | \         # Keep only a single occurrence of each value, and add counts.
  sort -n | \         # Sort numerically.
  tail -1 | \         # Keep the last value only (the one with the most counts).
  grep -o '[0-9]\+'   # Keep only the number of counts (not the name of the species).

# The same on a single line:
grep -o 's:.*;' < data/sample_01.dna | sort | uniq -c | sort -n | tail -1 | grep -o '[0-9]\+'
```

Notes:

- The backslashes \ at the end of lines are only for page layout purpose (it's a line continuation character) - it's all really one line. As we shall see, backslashes are a form of *escaping*.
- The second `grep` could be replaced by `sed`, `awk` or `cut`. We decided to show the solution with `grep` because it might be more familiar to most people.

Exercise 1.2 - A simple shell script

Using your favorite text editor - which should be Vim, of course :-) - create a new file named `max_spc.sh` and copy/paste the following lines of shell script into it.

Notes:

- These are just the commands from Exercise 1.1 with a minor change (can you spot it?).
- Feel free to choose a different file name: the `.sh` extension, like all extensions under Unix, has no particular effect and is just a matter of convention (`sh` is short for `shell`), it could also be `.bash`, for example.

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

grep -o 's:.*;' |    # Keep only species names.
  sort |             # Sort alphabetically. Needed for applying `uniq` to the data.
  uniq -c |          # Keep only a single occurrence of each value, and add counts.
  sort -n |          # Sort numerically.
  tail -1 |          # Keep the last value only (the one with the most counts).
  grep -o '[0-9]\+'  # Keep only the number of counts (not the name of the species).
```

Congratulations!, you have just created a **Bash script**.
Let's make it **executable** by using the command: `chmod a+x max_spc.sh`.
You can now try to **run it just as if it were a system command**:

```
./max_spc.sh < ./data/sample_01.dna
```

And of course, you can pass any other FASTA file to it:

```
./max_spc.sh < ./data/sample_02.dna
```

It wouldn't be hard to run our `max_spc.sh` on *all* our FASTA `./data/sample_*.dna` files in a single command. We will see `for` loops later in the course, but here is how you could do it:

```
for file in ./data/sample_0?.dna; do
    echo -n "Max sequence counts for a single species in $file: "
    ./max_spc.sh < "$file"
done
```

Exercise 1.3 - Tokenizing and quoting

The following command works...

```
echo my name is Bond
```

...but this does not:

```
echo my name is O'Donnelly
```

Why is that? and how can we fix this?

Exercise solution

The problem is that the single quote in `O'Donnelly` is not matched by a second (closing) single quote. To fix this, we can either:

- Escape the single quote: `echo my name is O\'Donnelly`.
- Double-quote the whole name (or, indeed, the whole string: `echo "my name is O'Donnelly"`.

Note: since the character we are trying to escape is a single quote `'`, we cannot use single quotes to quote it.

This will *not* work: `echo 'my name is O'Donnelly'`, because there is now again a non-matched single quote (the last one).

Exercise 1.4 - Parsing

1. **Write a command list** that prints the content of a file to standard output (i.e. the terminal) if a file exists, and print `Error: the file '<name of file>' does not exist!"` otherwise.
   Important: make sure your command also works with files whose name contains a space!

2. **Modify your command list** so that the error message gets printed to both the standard output and a hypothetical log file named `tmp.log`. Note: the error messages should *append* to the log file, not overwrite it.
   Hint: the command `tee` allows to both write content to a file and redirect it to standard output.
   Example usage:

   ```
   echo "print me!" | tee test_file.txt
   echo "add another line" | tee -a test_file.txt
   ```

3. **Additional task (if you have time):** prepend the current date and time to the error message.

Exercise solution

Create some test files and run the command list. Please note:

- To work with file names that contain spaces, the variable `$file` must be double quoted to prevent **word spitting** from occurring.
- Inside double quotes, the expansion of variables between single quotes (here `'$file'`) *does* occur.

```
echo "This is the content of the file..." > test_file.txt
echo "This is the content of the file..." > "test file.txt"
file="max_spc.sh"        # Test with a regular file.
file="test file.txt"     # Test with a file containing a white space.
file="missing_file.txt"  # Test with a non-existent file.

[ -f "$file" ] && cat "$file" || echo "Error: file '$file' does not exist!"

# Alternatively, we can also directly run `cat` on the file, and re-direct
# eventual error messages to /dev/null (but this is maybe slightly hacky...).
cat "$file" 2> /dev/null || echo "Error: file '$file' does not exist!"
```

Both print and save the error message to a log file named `tmp.log`.

```
[ -f "$file" ] && cat "$file"  || echo "Error: file '$file' does not exist!" \
  | tee -a tmp.log
```

Prefix the error log messages with the date and time.

```
[ -f "$file" ] && cat "$file" || \
  echo "$(date "+%Y%m%d %H:%M:%S") - Error: file '$file' does not exist!" | \
  tee -a tmp.log
```

Exercise 1.5 - parameter expansion

Using the **`showa` function** from the course slides, try the following (feel free to use any name as long as it has more than one word):

```
# Define the `showa` function if not already done.
showa(){ printf "%d args\n" "$#"; printf "%s\n" "$@";}

# Call the `showa` command.
name='Otocolobus manul'
showa $name
#   2 args
#   Otocolobus
#   manul
```

- Give two ways of passing `$name` to `showa` such that the content of `$name` is **passed as a single argument**. In other words, `showa` should output `Otocolobus manul` on a single line, as shown below.
- If needed, feel free to read this short section of the bash manual about word splitting.

```
showa <your argument here, must involve $name>
# Your output should be:
#   1 args
#   Otocolobus manul
```

Exercise solution

A first solution is to add double quotes around the variable `$name` when passing it to the `showa` function. Note that it must be double quotes - single quotes will not work as they prevent expansion of the `$name` variable.

```
name='Otocolobus manul'
showa "$name"
# 1 args
# Otocolobus manul
```

A second solution is to set the **`IFS` - Internal Field Separator** to an empty string, so the expanded content of `$name` is no longer split on whitespace (the default `IFS`).

Important: when using this option, make sure to **reset the IFS to its default** using the command `unset IFS` when the modified `IFS` is no longer

needed, otherwise you are likely to get strange (but not unexpected!) behaviors in your shell later on (default `IFS` value is: `<spc><tab><newline>`).

```
IFS=          # Same as IFS=""
showa $name
# 1 args
# Otocolobus manul

# Make sure to reset the IFS to its default.
unset IFS
```

Exercise 1.6 - brace expansion and command substitution

Part 1 - brace expansion

We would like to print the numbers 1 to 10 using the following code. Try to run it in your shell... but you should see that it does not give the expected result!

- **Questions:** why does it not work as expected?
- **Note:** we have not yet formally seen the syntax of `for` loops, but it does not really matter here (the syntax of the for loop is correct, the problem is elsewhere).
- **Warning:** make sure that you are using `bash` and not `zsh` - the Z shell which is currently the default on MacOS.

```
n=10
for i in {1..$n}; do
    echo $i
done
```

Exercise solution: part 1

The reason why the above loop does not print the expected result (i.e. numbers from 1 to 10), is because of the **order in which expansions are carried-out by the shell**.

Specifically, brace expansion is **performed before any other expansions**. In our case, this means that the shell will attempt to expand `{1..$n}` before it has actually expanded `$n` to the value `10`. Since `{1..$n}` does not expand to anything, it is left as is, and then `$n` is expanded into `10`.
As a result, the `for` loop iterates over a single value: `{1..$10}`, which is printed to the terminal via `echo {1..$10}`.

Additional Task (if you have time): Part 2 - command substitution

Try to make the above `for` loop work by replacing `{1..$n}` with the output of a call to the command `seq`. The `seq` command can be used to generate a sequence of integer numbers.

Example use of `seq`:

```
seq 1 5  # -> prints the number from 1 to 5.
```

Exercise solution: part 2

To make the loop print numbers from 1 to 10 while expanding the `$n` variable, we can use the **command substitution** `$( seq 1 $n )`:

```
n=10
for i in $( seq 1 $n ); do
    echo $i
done
```

Alternatively, we could also use the **C-style syntax** `for` loop (more on this later in the course):

```
n=10
for ((i=1; i<=$n; i++)); do
    echo $i
done
```

Exercise 1.7 - process substitution

The files `sample_1.tsv` and `sample_2.tsv` are two replicates of a gene expression data.

To make sure that both files contain the

Exercise solution

```
diff -u <( cut -f1 file 1 ) <( cut -f3 file 2)
```

Additional Task (if you have time)

The same thing, but on 2 different columns, and this time they have to additionally be sorted.

Exercise solution: additional task