Advanced Linux - Exercises

In these instructions the *first* character "**$**" in the command examples should not be typed, but it denotes the command prompt.

Some command lines are too long to fit a line in printed form. These are indicated by a backslash "\" at the end of line. It should not be included when typing in the command. For example

```
$ example command \
continues \
and continues
```

Should be typed in as:

```
example command continues and continues
```

0 Download and unpack the exercise files (do that first time only):

Moodle page and download exercise tar-file. Untar and cd into the directory.

First Day

1 Find shell scripts

Metadata: commands in this exercise: **`file`**, **`grep`**, **`wc`**

Metadata: Demonstrate the usage of pipes in simple tasks like finding the number of certain files in a directory.

1) How many shell script files there are in /usr/bin directory?

There are quite a bunch of files in */usr/bin*, but which of those files are actually shell scripts? To find out which type a file is, you will need the **`file`** command, which quite accurately will tell you what is inside a particular file.

2 Bash scripting

Metadata: commands in this exercise: **`bash, convert, animate`**

Metadata: These exercises demonstrate the scripting capabilities of the bash shell in case of automatized renaming and manipulation of files

1) Intro

Create a script that defines two variables, and outputs their value, e.g. 30 and 40. Impose that the script runs in the bash shell. Comment your script to explain what the script does.

2) Loops

Write a bash script that uses:
a) A **`for`** loop to count from 1 to 20000.
b) A **`while`** loop to count from 5 to 10.
c) An **`until`** loop to count down from 20 to 15.

3) Tests

a) Write a bash script that counts the files ending in **`.txt`** in the current directory.
b) Use **`if else`** statements in the script in case there are none (zero) files ending in **`.txt`**.

4) Parameters

Write a bash script that receives:
a) 3 parameters, and outputs them in reverse order.
b) 3 filenames as parameters and checks whether those files exist.

5) I/O

a) Write a bash script that asks for two numbers, and outputs the sum and product, e.g.,
`Sum: 3 + 4 = 7, Product: 3 x 4 = 12`.
b) Ask the numbers to be between 1 and 10, **`exit`** with an error if other number is typed.

6) Functions

Create a function that receives two numbers as parameters and outputs their sum.

3 Writing a conversion script

Metadata: commands in this exercise: **cp, convert, tar, animate.**

Metadata: The aim is to present a scripting solution of systematically manipulating a large set of similarly named files

0) Unpack the jpegs.tar file

```
$ cd FileRename
$ tar xvf jpegs.tar
```

Find out, whether ImageMagik - a legacy graphics program in Linux – containing the programs **`convert`** (file conversion), **`display`** (rendering of pictures) and **`animate`** (rendering of films) is installed

```
$ which convert
```

If it cannot be found, install

```
$ sudo apt-get install imagemagick
```

1) Find out what directories and files were created

```
$ ls *.jpg
```

You find out that the naming is numbered, but the listing is not in order. Hence, if you produce an animated gif from this bunch of files using the ImageMagik command convert:

```
$ convert –delay 30 –loop 2 *.jpg animation_small.gif
```

you will get strange results, as the input jpeg's are ordered by the first digit, i.e. {0, 10, 11, …, 19, 1, 20, 21, …, 29, 2, 30, …, 50, 5, 6, 7, 8, 9}. Try it:

```
$ animate animation_small.gif
```

2) Resize and convert files

This means we want to add a heading 0 to the single-digit numbers to produce the correct order {00,01,02,03,…,09,10,11,…,50}. Doing that by hand would be tedious, as one would have to make 10 shell commands similar to that one

```
$ mv 0_singleframe_small.jpg 00_singleframe_small.jpg
```

Additionally, we also want to resize the output using the ImageMagik command **convert**. In a combined way that would read for single digit numbers (0-9, here using 0):

```
$ convert –resize 200% 0_singleframe_small.jpg 00_singleframe_large.jpg
```

as well as for double digit numbers (10-50, here using 10):

```
$ convert –resize 200% 10_singleframe_small.jpg 10_singleframe_large.jpg
```

which would force us to give 51 shell commands. But we can use the power of loops within bash! You need to embed the commands using a variable for the counter within two loops, one from 0 to 9 and one from 10 to 50. Use the following syntax as a starting point and morph in the commands above:

```
$ for i in {0..9}; do ls ${i}_singleframe_small.jpg;\
echo "converting to 0${i}_singleframe_small.jpg"; done
```

You can (later on!!) look up the solution in convert.sh.

Second Day

1 Spread-sheet operations using awk

Metadata: commands in this exercise: **awk**

Metadata: The aim of this exercise is to demonstrate the abilities of **awk** to work as a command-line spread sheet application

The file **rovaniemi.txt** contains some meteorological information about Lapland's capital.

0) Move to data-subdirectory

```
$ cd exercises/awk/Meteo
```

1) Inspect the input file

```
$ less rovaniemi.txt
```

2) Inquire total yearly precipitation and monthly average daily precipitation

The 4th column contains rainfall (in mm m^{-2}). Inquire the over-all sum of precipitation. Using the 5th column (Rainy days) inquire the average daily precipitation of a rainy day for each month. Hint: you need to exclude the first line (header) of the file, best by using a regular expression inside your awk-script.

3) Compute yearly average minimum and maximum temperatures

Column 2 and 3 contain the monthly averaged temperatures. Compute the yearly averages (you do not have to weight months by their different length).

2 Scaling and deforming existing FEM mesh using awk

Metadata: commands in this exercise: **awk**

Metadata: The aim of this exercise is to demonstrate the abilities of awk to operate on structured data and selectively change entries

0) Move to data-subdirectory

```
$ cd Exercises/awk/fem
```

1) Make a listing

```
$ ls
$ ls unitsquare
```

You will see the files describing a Finite Element Method (FEM) Mesh:
`unitsquare/mesh.{header,nodes,elements,boundary}`

The files describe a unit-length square mesh structure. The aim is to systematically change the mesh-node coordinates, while keeping the structure of the mesh, which in practice means that we will not touch anything else than the file **`unitsquare/mesh.nodes`**, which contains the mesh coordinates; the other files contain information on how these nodes are ordered into bulk- and boundary-elements.

2) Copy original mesh to a new directory

Lets start with copying the original mesh into the new one:

```
$ cp -r unitsquare deformedmesh
```

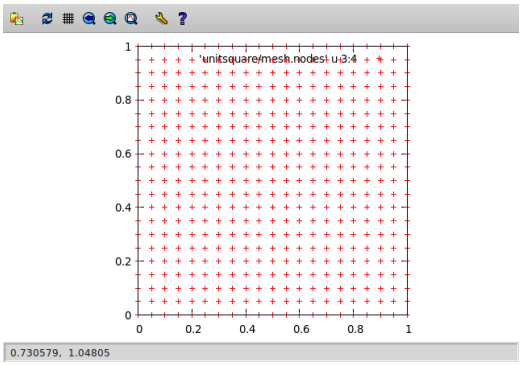
3) Visualize the node coordinates using gnuplot

Open a new terminal in the same directory (hint: Shift+Ctrl +N in the current terminal does the job) and launch **`gnuplot`** within:

```
$ gnuplot
```

In **`gnuplot`** we want to be sure that the length-scales are equal on both axis:

```
gnuplot> set size ratio -1
```



Further, we want to plot the x-y coordinates in the file **`unitsquare/mesh.nodes`**

```
gnuplot> plot 'unitsquare/mesh.nodes' u 3:4 w p
```

This is the syntax that plots contents of **`unitsquare/mesh.nodes`**

using (**`u`**) the 3rd and 4th column (which are the coordinates) of the file with (**`w`**) points (**`p`**).

4) Transform coordinates using awk

Now we come to the core task of the exercise: We want to stretch the x-axis with a factor 10.0 and the y-axis with a factor 0.5 using an **`awk`**-script. Hence, you have to write a script **`deform_mesh.awk`** that, if called as:

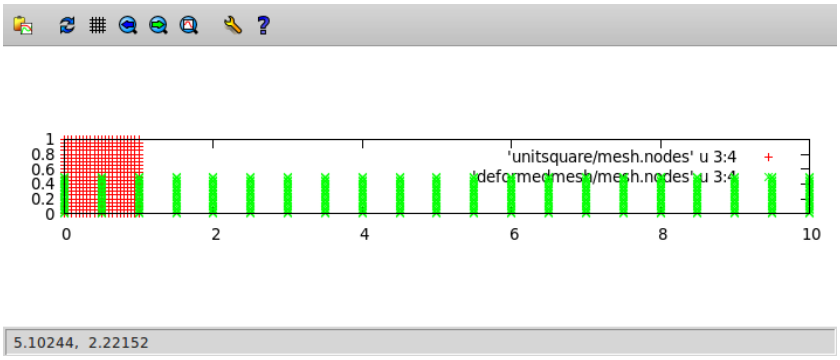
```
$ awk -f deform_mesh.awk unitsquare/mesh.nodes > deformedmesh/mesh.nodes
```

morphs **`unitsquare/mesh.nodes`** into the scaled mesh **`deformedmesh/mesh.nodes`** (you can overwrite the latter). So, your task is to write that script that operates on column 3 and 4 (while keeping column 1, 2 and 5 unchanged) of the input file.

5) Visualize the new node coordinates using gnuplot

In order to compare, we will plot the new coordinates above the old. To achieve that, you can use the history of the previously given **gnuplot** command (upward pointing arrow on keyboard) and simply add (mind that everything is a one-liner)

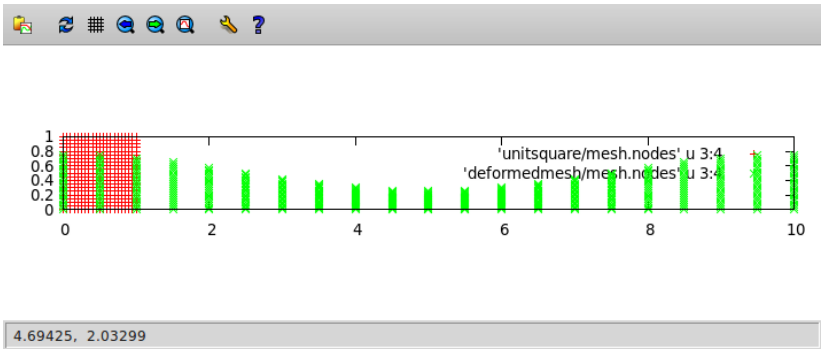
```
gnuplot> plot 'unitsquare/mesh.nodes' u 3:4 w p, 'deformedmesh/mesh.nodes' u 3:4 w p
```



The green points show the by a factor 10 and 0.5 stretched new mesh.

6) Extra: Scaling as a function of coordinate

If you find the time (and interest) you can expand the previous exercise: Changing the previous file, change the y-direction scaling as a cosine-function between 0.25 and 0.75 with a wavelength of the whole x-interval (1 in the original file or 10 in the final one). After visualizing, the output should look like



Hint: the argument for the cosine-function in **awk** is in radiant, hence you have to project the interval [0:1] to [0:2π], with π≈3.1415927.

3 Geographic data manipulation

Metadata: command in this exercise: **wc**, **head**, **tail**, **cat**, **sort**, **gnuplot**, pipes

Metadata: The aim is to create use shell text utilities to find certain values in a dataset.

We will use the dataset from NSDIC that is https://nsidc.org/data/NSIDC-0119/versions/1 and systematically investigate some properties (max elevation, ice thickness) using UNIX tools, only. The area (should you be interested) we investigate is part of the Antarctic ice sheet (see picture right), called Marie Bird Land. If you want to further use the data in any way, please check the NSDIC web services for their conditions. Data could be downloaded using the command (just as an example. **DON'T DO THAT**! – we do not want to blow up our and NSDIC's network).

```
$ wget ftp://sidads.colorado.edu/pub/DATASETS/AGDC/luyendyk_nsidc_0119/*
```

INSTEAD untar-gzip the provided file:

```
$ tar xvzf luyendyk_nsidc_0119.tgz
```

Now we have two ASCII files:

```
$ ls -l *.txt
-rwxrwxrwx 1 root root 170M Mar  9  2004 srfelev.txt
-rwxrwxrwx 1 root root 114M Mar  9  2004 icethick.txt
```

They are large. Check the number of entries (= number of lines) using the **wc** command. We just want to work with a smaller sub-set of the data. Hence we reduce the size to the first 10000 lines, only.

```
$ head –n 10000 srfelev.txt > srfelev_reduced.txt
$ head –n 10000 icethick.txt > icethick_reduced.txt
```

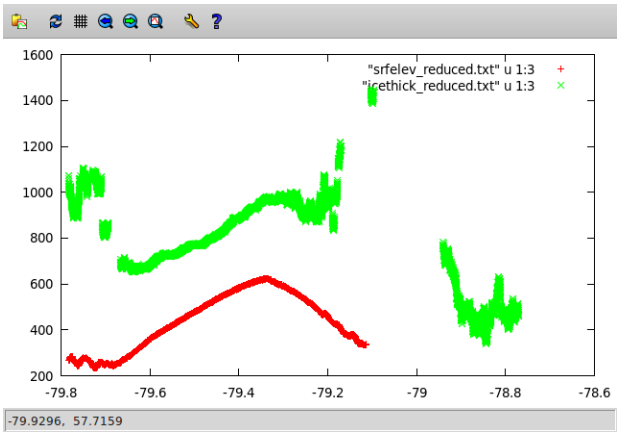
The problem now is, that we do not have a consistent dataset at places (see gnuplot script)

```
$ gnuplot showdata1.gp
```

If nothing shows, you might have to install the X11 version of gnuplot, which is done with

```
$ sudo apt-get install gnuplot-X11
```

(just answer "Y" if asked to download the packages and install).



Hence we would like to reduce the range of both datasets to be confined within -79.6 and -79.2 degrees (southern latitude). Let's inquire the position within the file for the lower bound:

```
$ cat -n srfelev_reduced.txt |grep "79.6000"
```

Output:

```
  2879      -79.600072 -144.39008  360.9 1137.4 1998 358 10256.30 RTZ8/32\
Wy-Y11a
```

And the same for ice thickness:

```
$ cat -n icethick_reduced.txt |grep "79.6000"
  1962      -79.600072 -144.39008  681.5 1998 358 10256.30 RTZ8/32\
 Wy-Y11a
```

So, we have to use the last (10000 – 2879) = 7121 and (10000 – 1962) = 8038 entries.

```
$ tail -n 7121 srfelev_reduced.txt > srfelev_reduced2.txt
$ tail -n 8038 icethick_reduced.txt > icethick_reduced2.txt
$ gnuplot showdata2.gp
```

Now, the same for the upper bound of -79.2 degrees:

```
$ cat -n srfelev_reduced2.txt |grep "79.2000"
  5892      -79.200076 -147.74868  …
$ cat -n icethick_reduced2.txt |grep "79.2000"
  5739      -79.200076 -147.74868  …
```

But now we need the heading lines, not the trailing and spare us the math!

```
$ head -n 5892 srfelev_reduced2.txt > srfelev_reduced3.txt
$ head -n 5739 icethick_reduced2.txt > icethick_reduced3.txt
$ gnuplot showdata3.gp
```

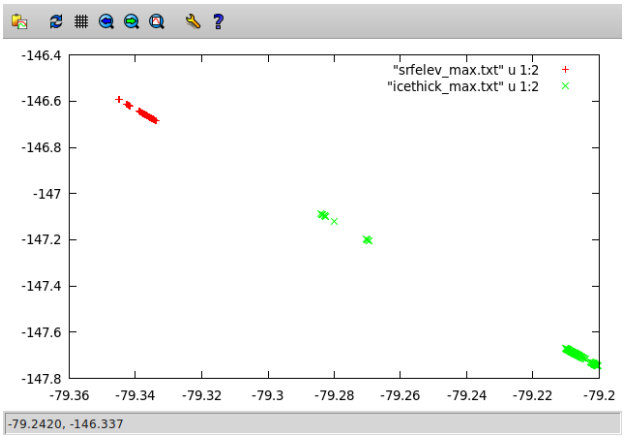
NB.: from the deviation of these two figures you can already tell that you still have 153 missing ice thickness entries.

Let's sort the lines according to the thickest ice and the highest elevation and extract the 100 maximum values

```
$ sort -n -k3 -r srfelev_reduced3.txt|head -n 100 > srfelev_max.txt
$ sort -n -k3 -r icethick_reduced3.txt|head -n 100 > icethick_max.txt
```

Let's check how max ice thickness and elevation correlate in their positions (they don't)

```
$ gnuplot showdata4.gp
```



Give the exact value of maximum thickness and elevation.

Do the same for minimum values and check this correlation.