Introduction to 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 Unpack the exercise files (do that first time only):

From Blackboard Course Documents find file linux-exercise.tar.gz Place it in Linux directory.

Open a terminal, cd to the folder where you placed the archive, unzip and untar the file:

```
$ tar xzvf linux-exercise.tar.gz
```

The **v** (verbose) flag in the tar command shows the files with the path that are untarred. You see, that you'll get a number of files in a subdirectory named linux-exercises. Go to that directory with the **cd** command.

1 Moving around in the directory tree

Metadata: commands in this exercise: **cd, mkdir, ls, mv, more, less, cat, tar.**
Metadata: The aim of this exercise is very simple: learn to move around with cd, look at file contents, create directories and move files around.

0) Unpack the dirs.tar.gz file

```
$ cd moving-around
$ tar zxvf dirs.tar.gz
```

1) Find out what directories and files were created

```
$ ls
$ ls -l
```

```
$ cd inputs
$ ls -l
$ cd ..
Etc.
```

2) Which output file is about caffeine?

Go to the *outputs* subdirectory (which itself has subdirectories *result{1,2,3}*). The pdb-files have a line with "TITLE *somename"*. Below are some commands that you can try find out which file is about caffeine. What are the other structures about?

```
$ grep caffeine *.pdb
$ grep caffeine */*.pdb
$ grep title */*.pdb
$ grep TITLE */*.pdb
```

You can view the complete file contents with either **`more`**, **`less`** or **`cat`**.

3) Create subfolders

There are only three **`result*`** -directories. Create new ones: **`result4`** and **`result5`** for outputfiles 4 and 5 and move the outputfiles (**`out_4.pdb`**, **`out_5.pdb`** which are in the **`result3`** directory) to those new directories.

- **`cd`** to the outputs -directory
- create a new subdirectory with **`mkdir result4`**
- move the **`out_4.pdb`** with the move command **`mv result3/out_4.pdb result4`**
- repeat for **`result5`** and **`out_5.pdb`**

4) Create a new compressed tar file of the new directory hierarchy

- Go to the *moving-around* directory where the original tar file (**`dirs.tar.gz`**) is with **`cd`**.

```
$ tar zvcf dirs-new.tar.gz inputs outputs
```

5) Confirm the tar file contents

```
$ tar ztf dirs-new.tar.gz
```

2 Use the man command to find flags for ls

Metadata: commands in this exercise: **`man, ls`**
Metadata: learn how to find detailed info about flags, and sort **`ls`** output

1) Open the ls man page

```
$ man ls
```

This opens the man page for **`ls`**. As there are a lot of options for **`ls`**, it is useful to search the man page. Search is triggered by pressing **`/`** and then writing a (start of) a keyword. Pressing "enter" triggers the

search and pressing "`n`" proceeds to the next occurrence of the keyword. You can also scroll the screen with arrow keys when needed. Exit from the man page with "`q`"

- Look for a flag to *sort* the `ls` output

`/sort` and press "enter" (note, you need to give this command while in the man page, not from command prompt)

2) Find the flag that sorts the output by file size

Press `n` as many times as needed until you find the flag for sorting by file size. You can also use some other keyword to find that (e.g. size).

3) Sort directory contents by file size

Go to the moving-around/inputs directory and sort the files by size

```
$ ls -S
```

4) Additional flags

Search for the flag that will reverse the sort order (*i.e.* print the largest file last). You can give the flags to the `ls` command together (e.g. `ls -la` instead of `ls -l -a`).

Search for the flag that will show the file size in "human readable format" i.e. kB/MB/GB instead of bytes (the default).

Tip: you can also search for the meaning of flags directly by `/-S` (which will look for occurrence of `"-S`" (or even better `"-S` " (trailing space)) if you want to know what that flag does.

3 Using wildcards

Metadata: commands in this exercise: ls, cp
Metadata: targeting many files with wildcards

Linux enables wildcards or regular expressions to match files or strings that differ only in some controlled ways.

1) Limit listing files with wildcards

Go to the moving-around/inputs directory. Check all contents.

```
$ ls
```

List only those files that have an "a" in the name and end in ".pdb"

```
$ ls *a*.pdb
```

List those files that have a name with seven characters and end in ".pdb".

```
$ ls ???????.pdb
```

2) Limit listing of the output files containing a range of numbers

Go to the moving-around/outputs directory. List all pdb files in the subdirectories.

```
$ ls */*.pdb
```

In this command the first "*" tells to look at all subdirectories in the current directory, and the second "*" all strings Now limit the list to only those out-files that have a number 2-5 in their name:

```
$ ls */out_[2-5].pdb
```

- What is the difference to this command?

```
$ ls */*[2-5].pdb
```

4 Simple backup scripts

Metadata: commands in this exercise: **`cp, mkdir, date, echo`**
Metadata: simple script that makes backup of a sup-directory to freshly created sub-directory

Change into the sub-directory **`linux-exercises/backupscript`**. In that directory you should have the following files that are the solutions (so don't open) for the following exercises:
`enhancedbackupscript.sh  simplebackupscript.sh`

1) A simple backup script

Based on the start from our **`befriendly.sh`** example, create a script that copies all files with a suffix (e.g., **`test.dat`**) from your home directory automatically to a directory in **`/tmp/homebackup`** that is first created by the same script. Use wildcards for that. Try to place some verbosity into the script by using the **`echo`**-command, e.g.,

```
$ echo "Starting copying files"
```

Take also of the advantage of the possibility to preserve the date of the file by adding a **`–p`** to the **`cp`** command. N.B.: if you also add a **`-u`** to the copy command, then you would make sure that upon multiple runs of the script only files of same newer would replace the one in the backup directory. This would be kind of an (on terms of whole files) incremental backup.

2) More versatile backup script

Using the possibility to store the output of a command in a local variable, create a directory-name that includes the current date:

```
destination=/tmp/homebackup_$(date +%Y-%m-%d)
```

And rewrite the script to create dedicated backups that are distinguishable by this date. Hint: You can then use the variable in connection with the **`mkdir`** command simply by **`$destination`**.

5 Linux command line exercises

Metadata: commands in this exercise: `grep, tar, cat, cut, more, less, awk, sed, wc, sort, gnuplot`
Metadata: example files from computational chemistry

Change into the `linux-exercises/chem` directory. In that directory you should now have these files:

`dimer.log` : Gaussian quantum chemistry geometry optimization log file for a water dimer
`dimer_scan.log` : Gaussian log file for relaxed potential energy scan for stretching water dimer distance
`freq.log` : Gaussian log file for a frequency calculation
`cp2k.out` : cp2k calculation ascii output file
`cp2k.xyz` : xyz format molecular structure file of liquid water

1) have a look at the contents of some files

```
$ more cp2k.out
$ nano cp2k.out
…
```

2) How did the total energy change in the water dimer optimization run? (dimer.log)

The part in the log file where the energy has converged is shown below. The final energy is printed on the shadowed line.

```
 Cycle  12  Pass 1  IDiag  1:
 E= -152.637052486060     Delta-E=       -0.000000000001 Rises=F Damp=F
 DIIS: error= 7.20D-08 at cycle   5 NSaved=   5.
 NSaved= 5 IEnMin= 5 EnMin= -152.637052486060     IErMin= 5 ErrMin= 7.20D-08
 ErrMax= 7.20D-08  0.00D+00 EMaxC= 1.00D-01 BMatC= 2.02D-13 BMatP= 1.38D-12
 IDIUse=1 WtCom= 1.00D+00 WtEn= 0.00D+00
 Coeff-Com:  0.795D-03 0.448D-01 0.134D+00 0.411D+00 0.410D+00
 Coeff:      0.795D-03 0.448D-01 0.134D+00 0.411D+00 0.410D+00
 Gap=     0.182 Goal=   None    Shift=    0.000
 RMSDP=8.86D-09 MaxDP=9.24D-08 DE=-1.08D-12 OVMax= 9.92D-08

 SCF Done:  E(RPBE-PBE) =  -152.637052486     A.U. after   12 cycles
            NFock= 12  Conv=0.89D-08     -V/T= 2.0059
 KE= 1.517485854820D+02 PE=-4.434279894947D+02 EE= 9.792090969701D+01
 Leave Link  502 at Thu Jan 29 09:51:27 2015, MaxMem=    33554432 cpu:         2.3
 (Enter /appl/chem/G09RevD.01/g09/l601.exe)
 Copying SCF densities to generalized density rwf, IOpCl= 0 IROHF=0.
```

How to get all those lines out?

```
$ grep "SCF Done:" dimer.log
```

What is the shortest string to `grep` that gives only these lines?

3) Was there something in the output that needs attention? (cp2k.out)

Programs often print out messages in case something goes wrong or the user has chosen questionable options. Is there anything in cp2k.out that we need to worry about?

Are there any warnings?

```
$ grep warning cp2k.out
```

What if the warning was capitalized? (or uppercase)

```
$ grep -i warning cp2k.out
```

4) look for the development of the convergence criteria, which of these is satisfied last? (dimer.log)

```
        Item               Value     Threshold  Converged?
 Maximum Force            0.057661     0.000450     NO
 RMS     Force            0.022508     0.000300     NO
 Maximum Displacement     0.218107     0.001800     NO
 RMS     Displacement     0.108003     0.001200     NO
```

Try some of these, what do they do?

```
$ grep "Maximum Force" dimer.log
$ grep "m Forc" dimer.log
$ grep " Force" dimer.log
$ grep " Displa" dimer.log
$ grep -E "RMS     |Maximum " dimer.log
$ grep -A 4 "Threshold" dimer.log
```

> *Why bother? Sometimes the geometry optimizations with "floppy" modes (flat potential energy surface) fail to converge and just wiggle around. Looking at the RMS force and Maximum displacement relative to the total energy can reveal that this is actually happening.*

5) Did the frequency calculation succeed? (freq.log)

Are there errors or warnings? Was the preceding geometry optimization successful *i.e.* the structure is a minimum on a potential energy surface? (*hint. were forces and displacements converged?*)

6) How many oxygen atoms there are in the cp2k.xyz file?

A line that specifies the coordinates for an oxygen atom looks like this:

```
  O         7.1808680000        1.7902530000        3.7253460000
```

Get all lines that have a capital O in them

```
$ grep O cp2k.xyz
```

How many lines was that?

```
$ grep O cp2k.xyz | wc
```

What does `wc` (short for *word count*) print out? Can we give it some flags? (*try* `man wc`, *or google for it*)

How many atoms in total? (*we know it's only hydrogen and oxygen atoms, i.e. O and H*)

```
$ grep -E "O|H" cp2k.xyz | wc -l
```

You could also count all the lines in the file and subtract the first two, which don't represent atoms.

```
$ wc -l cp2k.xy
```

What if your structure had also osmium atoms (Os). How would you change your commands?

7) How long did one iteration in cp2k.out take?

- The part in the output file that shows timing is like this:

```
 ***************************************************************************
 ENSEMBLE TYPE                =                                          NVE
 STEP NUMBER                  =                                            1
 TIME [fs]                    =                                     0.500000
 CONSERVED QUANTITY [hartree] =                        -0.880615921354E+04

                                          INSTANTANEOUS             AVERAGES
 CPU TIME [s]                 =                  290.20                290.20
 ENERGY DRIFT PER ATOM [K]    =    -0.810224161417E+02   0.000000000000E+00
 POTENTIAL ENERGY[hartree]    =    -0.880838552803E+04  -0.880838552803E+04
 KINETIC ENERGY [hartree]     =     0.222631448393E+01   0.222631448393E+01
 TEMPERATURE [K]              =                305.326               305.326
 ***************************************************************************
```

You could try this to get the timing:

```
$ grep TIME cp2k.out
```

But that gives too many hits. To get only the line that has the wall clock time spent at each time step give:

```
$ grep "CPU TIME" cp2k.out
```

Is energy drift per atom speeding up? How to get access to those lines?

8) How to plot those times?

First direct them to a file

```
$ grep "CPU TIME" cp2k.out > times
```

Confirm they are there and count in which column (white space separated):

```
$ more times
```

Start gnuplot with `$ gnuplot` and give

```
plot 'times' using 0:5 with points
```

or with lines

```
plot 'times' using 0:5 with lines
```

Explanation: in gnuplot the command is "plot", followed with the filename that has the data to be plotted (as it is a string, it needs to be quoted), "using" tells gnuplot to use the following columns in that file, "0" means the line number (first line=1, second line=2,... this will be the x-axis), ":" means to plot the second column as the function of the first column, "5":th column will be the y-axis, "with" is followed by what to plot at the coordinates, now it's "points" i.e. some symbols.

Exit with `quit`

How to plot the energy drift per atom?

9) What is the average temperature based on cp2k.out?

The part in the output file that has this information looks like this:

```
 ***************************************************************************
 ENSEMBLE TYPE                =                                          NVE
 STEP NUMBER                  =                                            1
 TIME [fs]                    =                                     0.500000
 CONSERVED QUANTITY [hartree] =                        -0.880615921354E+04

                                        INSTANTANEOUS             AVERAGES
 CPU TIME [s]                 =                290.20               290.20
 ENERGY DRIFT PER ATOM [K]    =  -0.810224161417E+02   0.000000000000E+00
 POTENTIAL ENERGY[hartree]    =  -0.880838552803E+04  -0.880838552803E+04
 KINETIC ENERGY [hartree]     =   0.222631448393E+01   0.222631448393E+01
 TEMPERATURE [K]              =              305.326              305.326
 ***************************************************************************
```

First we want to `grep` all lines with the temperature:

```
$ grep TEMPERATURE cp2k.out
```

To get the average we can process the output of the `grep` command with `awk` and create a cumulative sum of the column with the temperature (`+=` adds the value in column 4 to the current value in variable `a`), then print that instantaneous value (in column 4), the number of records processed `NR` and the average up to that point:

```
$ grep TEMPERATURE cp2k.out | awk '{a+=$4;print $4,NR,a/NR}'
```

This could also be done without separate `grep` command with

```
$ awk '/TEMPERATURE/{a+=$4;n++;print $4,n,a/n}' cp2k.out
```

Explanation: here we use a counter "n", which is incremented each time the condition is met (the line has the string TEMPERATURE). n can then be used to calculate the cumulative average. In this case NR counts all lines in the file cp2k.out so we can't use that to calculate the average.

How to plot both the instantaneous temperature and the cumulative average? (*hint: in gnuplot if you use replot instead of plot, the previous plot is retained*)

10) What is the smallest x-coordinate in cp2k.xyz?

X-coordinate is the first number, *i.e.* in the second column on the file. You can sort the file numerically (`-n`) according to the column (`-k`) you want.

```
$ cat cp2k.xyz | sort -n -k 2 | head
```

What are the coordinates of the O atom that has the smallest z-coordinate? (in the 4th column)

11) Working with data columns and fixing data format

Often it is necessary to change files slightly to use them in different analysis programs. This exercise simulates some typical changes you need to do. For example, NGS data from different sources may come in different syntax and to use them together needs fixing one or the other. This exercise shows an example on how to accomplish that.

Get file `hsa.gff3` from mirbase.org:

```
$ wget ftp://mirbase.org/pub/mirbase/CURRENT/genomes/hsa.gff3
```

1. Remove comment lines (lines starting with `#`)

```
$ grep -v "#" hsa.gff3 > tmp_nohash
```

2. Remove all lines that include tag '`miRNA_primary_transcript`'

```
$ grep -v "miRNA_primary_transcript" tmp_nohash > tmp_nomir
```

3. Change chromosome names (1st column) from format `chr1, chr2`, .. to format `1, 2,...`

```
$ cut -c 4- tmp_nomir > tmp_noscr
```

> *This command "cuts" i.e. prints everything from the 4th character on each line, i.e. cuts away the first three characters. This could also be done with* `sed`*. The following command would replace the first occurrence of "chr" if the line starts with it (^ matches the start of the line) on each line with nothing (what is between the second and third slash), i.e. remove those.* `$ sed s/^chr// tmp_nomir > tmp_noscr`

4. The 9. column is now format

```
'ID=MIMAT0027618;Alias=MIMAT0027618;\
      Name=hsa-miR-6859-5p;Derives_from=MI0022705'.
```

Change the first item to format `'gene_id "MI0006363_1"'`

```
$ sed s/ID=/'gene id "'/ tmp_noscr > tmp_gene_id
```

> *Note that here we already print out the first double quote " around the gene_id. We'll print the second " at the next stage.*

5. Leave out the last three columns i.e. `Alias, Name` and `Derives` entries and add the " after the `gene_id`.

```
$ awk -F ";" '{print $1 "\""}' tmp_gene_id > tmp_trimmed
```

First we tell `awk` *to use ; as the field separator.* `$1` *now matches everything up to the first ; (i.e. until the* `gene_id` *code). Getting the* `"` *in place is a bit tricky. As* `"` *has a special meaning (it is not just a character) we need to escape it with* `\` *to mean just-the-character-*`"` *and not the meaning of* `"` *and finally quote that with* `"`*:s. An alternative way to do this in two steps is to use* `cut` *to leave out everything after the first occurrence of* `;` *and then print the trailing " with* `awk` *(as above).*

```
$ cut -d ";" -f 1 tmp_gene_id | awk '{print $0 "\""}' > tmp_trimmed
```

6. Sort the file by chromosome and by miRNA start position (4. column). Make sure to sort the chromosomes in numerical order (`-n`), not in alphabetical (*i.e.* 1,2,3... not 1,10,11..)

```
$ sort -k1n,1 -k4n,4 tmp_trimmed > hsa.edited
```

Extra task. Make another file that only has entries from chromosome 2

It's possible to do the above in single command line, i.e. passing the output from the previous command as input to the next, but usually it is safer to use temporary files (until you know each of the steps work). Here is a one-liner that does steps 1-6:

```
$ grep -v "#" hsa.gff3 | grep -v "miRNA_primary_transcript" | cut -c 4- | \
      sed s/ID=/'gene id "'/ | awk -F ";" '{print $1"\""}' | \
      sort -k1n,1 -k4n,4 > hsa.edited
```