Introduction to the shell and task automation

2015-04-22

What is the shell?

In brief...

- Command line interface, looks old-fashioned but very convenient
- Main interface when you want to login to CSC servers or remote servers
- Also present in Linux distributions for personal computers and Mac
- With Windows, the `cmd` prompt is a bit similar (text-based) but not as powerful

What is the shell?

In brief...

- Command line interface, looks old-fashioned but very convenient
- Main interface when you want to login to CSC servers or remote servers
- Also present in Linux distributions for personal computers and Mac
- With Windows, the `cmd` prompt is a bit similar (text-based) but not as powerful

Usage

- Often the only interface For remote connections
- Automate repetitive tasks
- Powerful built-in commands
- Shell scripts to reproduce data manipulation

Where can we find the shell?

To find a shell...

- On Linux and MacOS systems: open a terminal. This will provide you with a Unix-like shell on both systems
- On Windows: run `cmd.exe` or `cmd`. This shell is quite different from the Unix-like shell found in Linux and MacOS. To obtain a Unix shell on Windows, one can install the Cygwin tools.
- It is strongly recommended to learn how to use a Unix shell since it is very likely it is this type of shell you will be exposed to when you connect to a remote server.

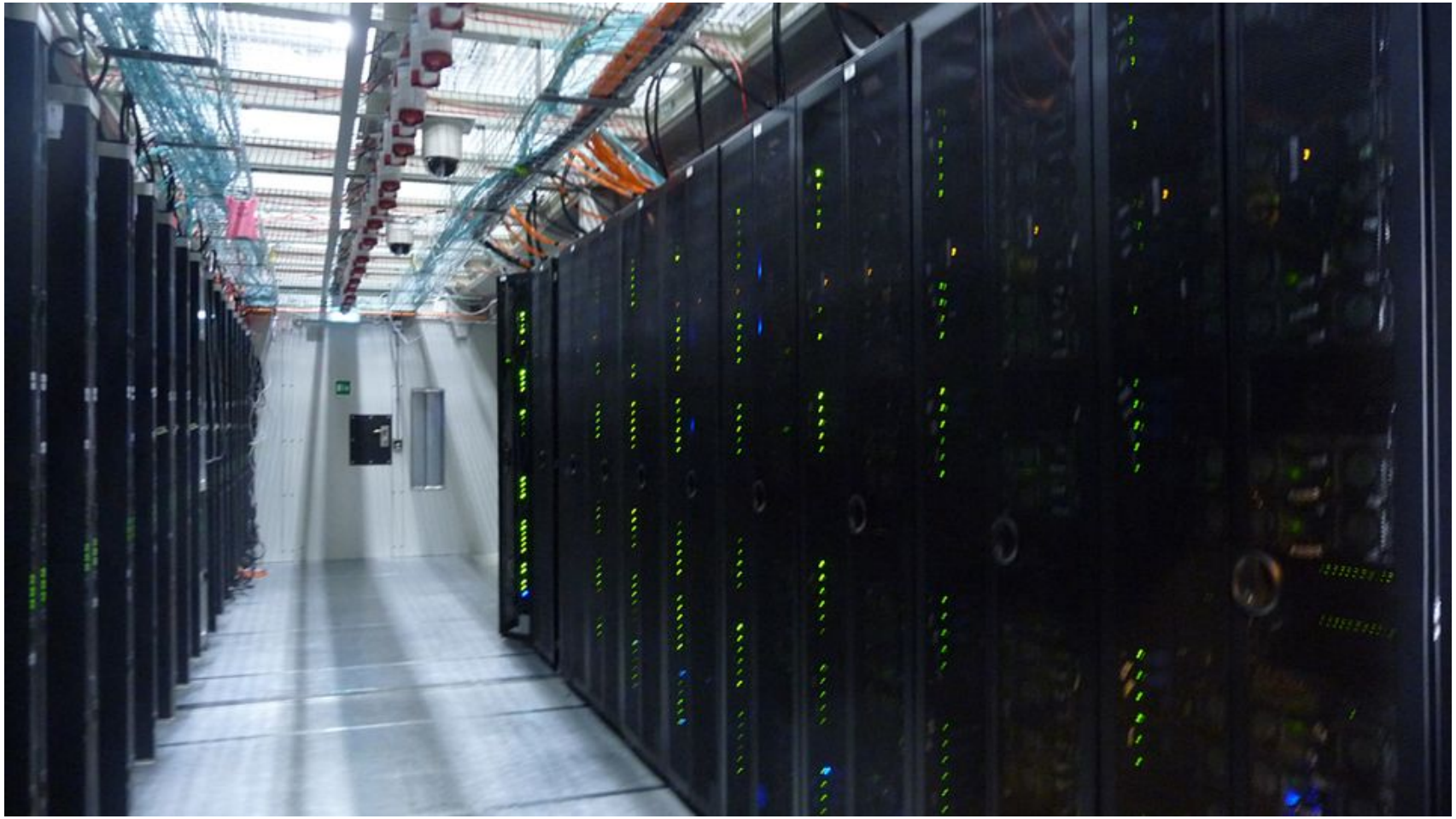
Where can we find the shell?

One shell or several shells?

- A shell: a program providing an interface between the user and the computer. Different shells exist.
- The most popular and widely used shell is probably bash. It is the default shell in most GNU/Linux distributions.
- If you learn how to use bash, you will be able to use most remote servers you'll have to connect to, and also the terminal from MacOS or the Cygwin tools on Windows

The CSC center in Kajaani

Meet the Taito cluster (`taito.csc.fi`)



Connection to a remote shell

The plan

- Using the CSC server Taito in Kajaani (student account)
- Tools: putty (windows) or ssh (Mac and GNU/Linux)
- A word about ssh and the security of connections?

Student account

- Logins: `jyybio01` to `jyybio20`
- Password: on the whiteboard

Connection to a remote shell

The plan

- Using the CSC server Taito in Kajaani (student account)
- Tools: putty (windows) or ssh (Mac and GNU/Linux)
- A word about ssh and the security of connections?

Student account

- Logins: `jyybio01` to `jyybio20`
- Password: on the whiteboard

Connection

- From a terminal (Mac or GNU/Linux):

  `ssh jyybioxx@taito.csc.fi`

  where `xx` is your student number.
- From Putty: ask a teacher if needed

First contact with the shell

Just after connection

What you see after connection in the shell prompt. It tells you the shell is ready to receive your input:

```
jyybioxx@taito-login3$
```

`jyybioxx` is your username, `taito-login` is the host server to which you are connected. The number after `taito-login` can vary because Taito has several login nodes.

First contact with the shell

Execute a command (`ls`)

The shell reads and executes commands you enter at the prompt, and prints the output. Type `ls` and press `RETURN`. You should see:

```
appl_taito
```

First contact with the shell

Execute a command (`ls`)

The shell reads and executes commands you enter at the prompt, and prints the output. Type `ls` and press `RETURN`. You should see:

```
appl_taito
```

You just ran the `ls` command which produces an output: the list of files and folders present in the current directory.
Try another command: `whoami`. What does this command do?

First contact with the shell

Execute a command (`pwd`)

When you login to a server, you are automatically sent to your home folder. You can see where you are by typing `pwd`, which produces:

```
/homeappl/home/jyybioxx
```

So you are now in the folder `jyybioxx`, which is itself contained in `home`, which is contained in `homeappl`, which is at the root of the file system (`/`, there is no parent directory above).

Adding options to a command

Using `ls` options

You can add options to a command with the dash sign `-`:

```
ls -l
```

(this is -l, not -1)
This runs the `ls` command with the `-l` option, which produces a detailed output:

```
total 4
drwx------ 2 jyybio20 jyybio 4096 Apr 15 12:15 appl_taito
drwx------ 2 jyybio20 jyybio 4096 Apr 15 15:22 ecoli-data
```

Now you can see the date of last modification of the folders and some other information. We'll see later in more details what the other information means.

Clone the Git repository for the practicals

Clone the Git repository

Before going further, you should clone a Git repository containing the data which was prepared for you (Git is installed on Taito). The repository is hosted on GitHub.
Check that you are in your home folder with `pwd`. You should see:

```
/homeappl/home/jyybioxx
```

If not, go back to your home folder by typing simply `cd` without any argument.
Clone the Git repository with:

```
git clone https://github.com/mdjbru-teaching-material/practica
```

and run `ls`. What happened?

Data content and motivation

The data files

Each file corresponds to one *Escherichia coli* strain for which a complete or draft genome sequence is available. The name of the strain is contained in the file name. Each file contains the peptide sequences from all translations resulting from Ensembl known or novel gene predictions for that given *E. coli* strain.
Files are in the FASTA format. The original address is `ftp://ftp.ensemblgenomes.org/pub/current/bacteria/fasta/`.

Motivation

We want to determine the amino acod content of all proteins of each strain, and compare the results between strains. We already have a Python script ready which can determine the amino-acid composition for protein sequences.

Basic folder navigation

`cd` command

We can navigate from folder to folder using the `cd` command:

```
cd practicals
ls
cd ecoli-data
ls
```

We could have go directly to the second subfolder with `cd practicals/ecoli-data`
You can see there are already some files in this folder. Let's ask for more details:

```
ls -l
```

How many files are there? How large are they?

Basic folder navigation

Combining options for `ls`

We can ask for more human-readable sizes with:

```
ls -l -h
```

Can you see the difference with `ls -l`? What does `ls -h` do?
We could also combine both options to `ls`: `ls -lh`

Basic folder navigation

Moving to the parent directory

We can go back through the parent folders using `cd ..`:

```
pwd     # Where are you at this point?
cd ..
pwd     # And now?
ls
cd ..
pwd     # And here?
ls
```

Basic folder navigation

Going back to the home directory

A faster way to go back to your home directory, from any starting directory, is just to type `cd` without any argument. Now go back to the `ecoli-data` subfolder and back again to your home directory using `cd`.

Shortcut for the home folder

Another way to go to the home folder is to use the `~` character: this is automatically replaced by the path to your home folder by `bash`:

```
cd
cd practicals
cd ~
cd appl_taito
cd ~/practicals
```

Creating folders

The `mkdir` command

Go back to the `practicals` folder and create a new folder in it:

```
cd ~/practicals
mkdir results
cd results
ls
```

Exercise

Create the following directory structure:

`~/practicals/scripts/python/modules/seqAnalysis`

and go back to your home folder.

Auto-completion

The magic TAB key

Let's go into `seqAnalysis` folder=, but let's be lazy:

```
cd     # Start from your home folder
cd pr  # Press TAB at this point
```

Do you understand what happened? Use this feature to quickly go to `seqAnalysis`. What is the minimum number of keystrokes you have to use to go there from your home folder?

Remember!

When you press `TAB`, the shell tries to complete what you just typed by itself. This auto-completion feature of the shell is very convenient and will save you a lot of typing!

Auto-completion

Test auto-completion

Now create a folder:

`~/practicals/scripts/python/modifiedSources`

Go back to your home folder, and go into `modifiedSources` using the `TAB` completion as much as you can. What do you notice?

Auto-completion

Double TAB

Now create the folder

```
~/practicals/scripts/python/modularComponents
```

and type:

```
cd ~/practicals/scripts/python/mod  # Press =TAB= twice here
                                    # Type "ule" and press =TAB= again
```

Do you understand how TAB completion works? This also works for command names.

Copying, moving and removing files

Creating an empty file

Go the the `seqAnalysis` folder and type:

```
touch DNA-analysis.py
ls
```

What happened?

Moving a file

Now type:

```
mv DNA-analysis.py ../modularComponents
```

What happened? Did you use the `TAB` key? (you should!) Explore the directory structure to find `DNA-analysis.py` again.

Copying, moving and removing files

Copying a file

Go to the `modularComponents` subfolder and type:

```
cp DNA-analysis.py ../modules
```

What happened?

Removing a file

From `modularComponents` folder, type:

```
rm DNA-analysis.py
```

What happened?

Creating a directory hierarchy

Moving a folder

From the `scripts` folder, move `modularComponents` into `modules`:

```
mv modularComponents modules
tree
```

Copying a folder

Go to the `practicals` folder and make a copy of `scripts`:

```
cp -r scripts scripts-backup
```

Note the `-r` option used for recursive copy inside the directories.

Creating a directory hierarchy

Removing a folder

Remove the newly created folder with:

```
rm -r scripts-backup
```

Again, note the `-r` option to work on folders.

Creating a directory hierarchy

Exercise

Now that you have experience, create the exact following directory structure (only folders shown):

```
.
+-- appl_taito
‘-- practicals
    +-- ecoli-data
    |   ‘-- [...]
    +-- results
    |   ‘-- 2015-04-22
    ‘-- scripts
        +-- python
        |   +-- popGenetics
        |   +-- proteinStructure
        |   ‘-- seqAnalysis
        ‘-- R
```

Viewing a file

cat command

Go to the `ecoli-data` folder and type:

```
cat README
```

Try also `cat` on one of the fasta files. What happened?

head and tail commands

```
head Escherichia_coli_o5_k4_l_h4_str_atcc_23502.GCA_000333195.
tail Escherichia_coli_o5_k4_l_h4_str_atcc_23502.GCA_000333195.
head -n 30 Escherichia_coli_o5_k4_l_h4_str_atcc_23502.GCA_0003
tail -n 3 Escherichia_coli_o5_k4_l_h4_str_atcc_23502.GCA_00033
```

Do you understand what those commands do?

Viewing a file

`less` command

`less` is very useful to examine large file. You can navigate using the up and down arrows or `B` and `SPACE` keys, and you can exit with `Q`.

```
less Escherichia_coli_o5_k4_l_h4_str_atcc_23502.GCA_000333195.
```

Useful tools: `wc`

`wc` to count words

Go to the `ecoli-data` folder and type:

```
wc Escherichia_coli_o55_h7_str_06_3555.GCA_000617385.1.26.pep.
```

which produces:

```
26318   51865 1824223 Esch...
```

We can have only the number of lines with `wc -l` (try it).

Wildcards

Try:

```
wc -l *.fa
```

What happened?

Redirection

The > operator

When a command produces some output, it can be redirected to a file instead of to the terminal:

```
wc -l *.fa > lineCounts
cat lineCounts
```

> is a redirection operator, and automatically creates a new file or erases an existing file.

The » operator

To redirect output and append it to an existing file, we can use the » operator:

```
wc -l README >> lineCounts
cat lineCounts
```

Useful tools: grep

grep to search for matches

:grep "flagellin" Escherichia$_{colio55h7str063555}$.GCA$_{000617385}$.1.26.pep.all.fa
:grep –color=always "flagellin"
Escherichia$_{colio55h7str063555}$.GCA$_{000617385}$.1.26.pep.all.fa :grep -n
–color=always "flagellin"
Escherichia$_{colio55h7str063555}$.GCA$_{000617385}$.1.26.pep.all.fa :grep -c
–color=always "flagellin"
Escherichia$_{colio55h7str063555}$.GCA$_{000617385}$.1.26.pep.all.fa Do you understand what each of the `grep` options do?

Exercise

Use `grep` to extract all the sequence names from one of the fasta file and store them in a file called `proteinNames`.

Useful tools: grep

grep is versatile

```
grep -c flagellin *.fa
grep -c flagel *.fa
```

Do you understand the output?

Exercise

How would you count the number of proteins in each fasta file?

Useful tools: cut

cut to get columns

```
grep -c flagel *.fa > flagelCounts
cat flagelCounts
cut -d "_" -f 1 flagelCounts
cut -d "_" -f 3 flagelCounts
cut -d "_" -f 3,5 flagelCounts
cut -d ":" -f 2 flagelCounts
```

Do you understand what `cut` does and the roles of the `-d` and `-f` options?

Useful tools: `sort`

`sort` to sort things

Use `sort` to sort the line counts from `lineCounts`:

```
sort lineCounts
```

Is everything correct? What if you try `sort -n lineCounts`? Can you see a difference? Try also `sort -r lineCounts`

Exercise

Using `grep` and `sort` and an intermediate files, sort the bacterial proteomes by decreasing number of proteins. Hint: `sort` supports two interesting options, `-t` to specify a field separator and `-k` to specify which field to use for sorting, so that `sort -t"_" -k2 myFileName` would sort the lines in `myFileName` using the second field using "_" to delimit the fields.

Combining tools with pipes

Pipes can connect an output and an input streams

When we did `sort lineCounts`, we used `sort` of the output of `wc`, but we used an intermediate file. The shell offers a powerful way to connect directly the output of a command to the input of another: the pipe operator:

```
wc -l *.fa | sort -n
```

Exercises

The `w` output the list of connected users on the server:

```
w
w | head
w | less
```

Use a pipe to find all the users whose login contains "jyy". Extend the same pipe to count how many there are.

Python script to determine amino acid composition

Test the Python script

The script `seqComposition.py` takes a fasta file and produces a table containing the amino-acid composition of each protein in the file. To run the script, type:

```
module load python-env/3.4.1   # This is specific to the serve
python3 seqComposition.py myFastaFile # Use the fasta file you
```

The output is sent to the terminal. Propose at least two ways to have a look at this output.

Python script to determine amino acid composition

Exercise

Using only the Unix tools you know, the Python script and pipes, determine the distribution of the number of histidines per protein in the proteome of the strain of your choice.

One step towards wizardry: shell scripts

Reusing your tool pipeline

Let's use `nano` to store your pipeline in a file:

```
nano getHistDistrib.sh
```

(the usage of `nano` will be demonstrated live) The idea is to be able to produce the histidine distribution results just by typing:

```
bash getHistDistrib.sh myFastaFile
```

Test your pipeline with a few files

Test your pipeline for 5 strains. How would you feel about doing it for 2000 strains?

One step towards wizardry: shell scripts

Making a general purpose listing script

Create a shell script (`testListing.sh`) with this content:

```
listFiles=‘ls *.fa‘
echo $listFiles
for myFile in $listFiles; do
    echo $myFile
    echo $myFile.results
done
```

Run it with `bash`. What does this do?

Exercise: final script

Combine the script with your pipeline and the listing script into a single script to get the histidine distribution for all the fasta files in this folder.