Shell Tutorial

Janos Binder[1]

[1]European Molecular Biology Laboratory (EMBL),
Heidelberg, Germany
`janos.binder@embl.de`

May 31, 2014

Contents

1 Getting started

First you the necessary softwares to work remotely. On windows you need the necessary clients such as `Putty` and the `WinSCP` program. On Linux and Mac OSX you can use `ssh`:

```
> ssh server.name
```

Let's get familiar with the shell. You will see something like:

```
jbinder@red:~>
```

Where we are now? The `pwd` command will tell us. Try it out! Check also `man` or `info` out.

```
man pwd
```

Press `q` to exit from the manual.

2 Setting up the project environment

We organize our work into folders. How does it work from the console.

In the modern operating system every user has her or his own home directory. One can reach it by:

```
cd ~
```

The `cd` command changes the current directory and the `~` is a special character, whcih refers to the home directory. What is in this directory exactly? The following command will tell:

```
ls
```

`-la` flags are widely used to list the hidden files and the details.

```
ls -la
```

The `.` denote the current directory and `..` denote the parent directory. The classroom projects should be stored in `projects`. It can be created by:

```
mkdir projects
cd projects
```

Some directories are no longer used and they need to be removed. The `rmdir` is the opposite command of `mkdir`.

```
mkdir wrong_directory
ls -la
ls -la wrong_directory
rmdir wrong_directory
ls -la
```

Task: Create a project directory. Preference: `~/projects/compartments`

3 Manipulating files

First we need the human knowledge dataset from http://compartments.jensenlab.org/Downloads. It can be be downloaded:

```
cd ~/projects/compartments
wget http://download.jensenlab.org/human_compartment_knowledge_full.tsv
ls -la
```

You can also try:

```
cd ~/projects/compartments
curl http://download.jensenlab.org/human_compartment_knowledge_full.tsv > hkcf2.tsv
ls -la
```

A few file manipulation. Copying (try out the `TAB` button when typing):

```
cp human_compartment_knowledge_full.tsv hckf.tsv
ls -la
```

Rename one of the copied files (be lazy, try to use `TAB` button again):

```
mv hckf.tsv hc.tsv
ls -la
```

Delete them:

```
mv hc.tsv
ls -la
```

The `-v` stands for verbose, and it is useful to see what is happening. The `*` character matches to everything.

```
rm -v hc*.tsv
```

3.1 Manipulating files

Bioinformatics data comes often in text format. To get a glance, try the following commands.

```
head human_compartment_knowledge_full.tsv
tail human_compartment_knowledge_full.tsv
```

The columns in `tsv` files are separated with a tab character (`\t`). These files are closely related to the comma separated files (`csv`).

Let see what is inside:

```
less human_compartment_knowledge_full.tsv
```

You can use the cursors, the space and `G` and `g` button, which goes to the end and the beginning of the file respectively. Use `q` to exit the program. We need some statistics about the file, the `wc` command answers how many lines, words and characters exist. We can limit it further by using the flags (`-l -w` or `c-`).

```
wc human_compartment_knowledge_full.tsv
```

We can also filter specific lines, for example what proteins are localized in the mitochondria. The Gene Ontology identifier for mitochondrion is `GO:0005739`. It can be done by:

```
grep 'GO:0005739' human_compartment_knowledge_full.tsv
```

It seems to be too long, let us see a sample from the output. For this we *pipe* the output of the command into another command.

```
grep 'GO:0005739' human_compartment_knowledge_full.tsv | head
```

If it seems good, we can store the mitochondrial proteins.

```
grep 'GO:0005739' human_compartment_knowledge_full.tsv > mitochondrial_proteins.tsv
```

We can also get the non-mitochondrial proteins. The `-v` flag expresses to list all lines, which do not contain the pattern. One example:

```
grep -v 'GO:0005739' human_compartment_knowledge_full.tsv | head
```

We need a file for the non-mitochondrial proteins as well.

```
grep -v 'GO:0005739' human_compartment_knowledge_full.tsv > non_mitochondrial_proteins.tsv
```

Specific columns can be also selected. Now we see how a file with pair of Ensembl identifiers and GO terms can be created.

```
cut -f1,3 human_compartment_knowledge_full.tsv | head
```

If works then save the results:

```
cut -f1,3 human_compartment_knowledge_full.tsv > ensembl_go_pairs.tsv
```

3.2 Advanced operations with AWK

AWK is an interpreted programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems. Let us see some examples.

Printing specific columns with AWK

As the first exercise let us see how can we print the columns with AWK. We use the GNU variant of AWK.

```
gawk -F '\t' '{print $1 "\t" $3;}' human_compartment_knowledge_full.tsv | head
```

The `-F '\t'` expresses the separator between the columns. In the curly backets we have the print statement. It says print the first and the third columns separated by a tab.

Filtering on specific values

We can also select the mitochondrial proteins with AWK:

```
gawk -F '\t' '$3 == "GO:0005739"' human_compartment_knowledge_full.tsv | head
```

Here there are no curly backets used, so it is evaluated as a conditional (printed only if the expression is true). We can combine it with the previous example.

```
gawk -F '\t' '($3 == "GO:0005739"){print $1 "\t" $3;}' human_compartment_knowledge_full.tsv | head
```

More filtering possibilities

It is often neccesary to filter a file based on some numerical threshold. In the example the last column denotes how reliable is the localization evidence. Filtering on high confidence scores ($= 5$):

```
gawk -F '\t' '$7 > 4' human_compartment_knowledge_full.tsv | head
```

One can combine multiple conditionals. Here we examine the mitochondrial proteins with highly reliable evidence.

```
gawk -F '\t' '$3 == "GO:0005739" && $7 > 4' human_compartment_knowledge_full.tsv | head
```

The `&&` denotes that both statement needs to be true, while `||` denote that at least one statement needs be true.

4 Shell programming

We will write a few script during the second part. First we learn how editing works remotely.

Coding the first script – Hello BioWorld!

```
pico 1_hello_bioworld.sh
```

Write the following lines:

```
#!/bin/sh

echo "Hello BioWorld!"
```

Exit works with `CTRL+X`. Don't forget to save!

You have make the script runnable and then you can test it.

```
chmod a+x 1_hello_bioworld.sh
./1_hello_bioworld.sh
```

Using variables

Variables are useful to store intermediate results and often used text and numbers.

```
#!/bin/bash

HW="Hello World!"

echo $HW
echo "It is boring to print" $HW
```

Using conditionals

A branch of a script is executed only if the statement is true. A simple example:

```
#!/bin/bash

FILE="3_basic_conditionals.sh"

if [ -f $FILE ]; then
    echo "FILE:" $FILE "exists!"
fi
```

A more complicated example:

```
#!/bin/bash

FILE="3a_complex_conditionals"

if [ -f $FILE ]; then
    echo "FILE:" $FILE "exists!"
else
    echo "FILE:" $FILE "does not exists!"
fi
```

Task: Try to modify this script to run the other branch of the conditional.

Using loops

A loop is a sequence of statements which is specified once but which may be carried out several times in succession. The code "inside" the loop is obeyed a specified number of times, or once for each of a collection of items, or until some condition is met, or indefinitely.

```
#!/bin/sh

echo "We count to ten!"

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

The double bracket denote that `i` refers to a number.

Task: Modify the script to count back from 10.

There is another solution to the previous example with the `while` keyword.

```
#!/bin/sh

echo "We count to ten!"

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

Task: Modify the script to count back from 10.

The shell can also store the output of a command. Here is an example, which writes the files out:

```sh
#!/bin/sh

FILES=`ls -1`

for f in $FILES; do
    echo $f "is a file"
done
```