Advanced Unix – scripting

The previous tutorial introduced the richness of the command line. It allows through the use of simple applications as building blocks and pipes as the plumbing the possibility to construct complex applications on the fly.
It is probably the case that some of the operations you are carrying out you will want to do again, and typing all the commands is long and tedious.

This tutorial introduces the concept of scripting, or programming. That is preparing a set of commands for the computer to run. We will start where the previous tutorial finished, saving the command to a file so that we can run it again. We will then learn a bit more about the sort of things that can be done and take some first steps into programming. Once again this tutorial doesn't aim to turn you into a competent programmer, but instead aims to indicate the sort of things that can be done.

Creating a script file

Previously we had run the command

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | grep THR |
awk ' BEGIN{ print "Threonine residues"; counter=0;} { if ($10>50)
{print $4, $5; counter = counter+1}}  END{print "Total:", counter,
"atoms"}' | uniq
```

which is quite a mouthfull. Instead of writing this command every time we want to run it we can save a copy in a text file and ask the shell to run this instead. This saves typing and errors.

To get the computer to read a file as a list of commands there are two things that need to be done. Firstly we have to tell the computer which *command interpreter* to use. Secondly we have to tell the computer that it can execute (run) the file as a small program.

To do the first we need to find the appropriate command interpreter on the system. At a prompt type:

```
% which sh
/bin/sh
```

This should give the location of the program `sh` which is the command interpreter we will be using. In this case it is located in `/bin/sh` (you will need the full path).

Open your favourite text editor. On the first line of the file write

```
#!/bin/sh
```

This tells the Unix shell that it should use `/bin/sh` to interpret the commands we have written. #! must be the first two characters on the line and there should be no spaces before the path to the program.

Now write in the command. Blank lines are ignored by the command interpreter. Your file should now look like this (remembering to put all the command on one line):

```
#!/bin/sh

grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | grep THR |
awk ' BEGIN{ print "Threonine residues"; counter=0;} { if ($10>50)
{print $4, $5; counter = counter+1}}  END{print "Total:", counter,
"atoms"}' | uniq
```

Now save the file as `aacounter.sh`

For the file to be run by the system it needs to be executable. Wen need to set the permissions appropriately.

```
% chmod u+x aacounter.sh
```

And now we can run our script.

```
% ./aacounter.sh
```

And we should get the same results as typing the command in our script at the prompt.

This is fine and saves a lot of typing but every time we want to use a different structure we have to edit our script to change the pdb file we are searching. Instead of this we can modify our script to read the argument from the command line.

Every shell script places the arguments from the command line into variables. You can use the variable $1 for the first argument, $2 for the second and so on.

Lets edit our file to use command line variables.

Open up aacounter.sh in your text editor. The changes have been highlighted with bold. We have added a few 'echo' commands to put out some information.

```
#!/bin/sh

echo aacounter.sh: counting atoms in THR for $1

grep ATOM $1 | grep -v ATOMS | grep -v ' ATOM' | grep THR | awk '
BEGIN{ print "Threonine residues"; counter=0;} { if ($10>50) {print
$4, $5; counter = counter+1}}  END{print "Total:", counter,
"atoms"}' | uniq

echo Atoms counted in $1. Bye!
```

Now when we run this program we can use the command

```
% ./aacounter.sh 1boy.pdb
```

and can run this with any file you wish.

So far we are only counting atoms in threonine residues. Maybe we can do other things too?

Exercises
Try altering the program to search for a user specified amino acid type. (hint: the command line should look a bit like `% aacount.sh 1boy.pdb SER`)

Can it be modified to count atoms in all types of mino acids and give a table of the results?
(e.g.
```
ALA   432
ARG   529
...
```
)

There are other scripting languages and many more commands. This has just been a first 'toe in the water' for bioinformatics. If you are interested in progressing further then the book *'Beginning perl for Bioinformatics*' from O'Reilly press is strongly recommended.