More advanced Unix shell - NOTES

- 2016/17 Part III Systems Biology SEB module
- 8 Feb 2017, 10:00-13:00
- Bioinformatics Training Room, Craik-Marshall Building, Downing Site
- Alexey Morgunov

Contents

Introduction

You may want to refresh some basics of Unix shell by referring to my introductory notes here.

Shell scripting

This is where things get real. Shell can be used as a general purpose interpreted scripting language(-ish) with many of the associated features. It is especially powerful for processing text data and passing it between programs, as well as file handling. This it does easier and often faster (citation needed) than Python/Perl/Ruby equivalents.

Shell scripts are usually stored as text files with `.sh` as the extension and starting with a shebang consisting of `#!/bin/sh`.

Variables are prefixed with a dollar sign when called, e.g. `$HOME` and assigned with a single equals sign, e.g. `meaning=42`. To avoid ambiguity in scripts, it is often customary to surround variable names with braces, e.g. `${HOME}`. More on variables here.

Conditional statements:

```
if  [ condition ]; then
 echo "Something"
 elif [ another_condition ]; then
   echo "Something else"
 else
   echo "None of the above"
fi

#example

#!/bin/sh
if [ $1 -gt 0 ]; then
  echo "$1 number is positive"
  else
    echo "$1 number is negative"
fi
```

Note that in the above example, special variable `$1` refers to the first argument supplied when running the script. Save the script to a file named `my_script.sh` and run it using the command `./my_script.sh [number]`. This is an

example of "positional parameter" usage. For more on positional parameters, see here.

Case statements:

```
case $variable in
  "pattern1") echo "pattern1";;
  "pattern2") echo "pattern2";;
  *) echo "none of the above";;
esac

#example

#!/bin/sh
echo "car, van, jeep, bicycle or something entirely different?"
read rental
case $rental in
   "car") echo "For $rental £30 per day";;
   "van") echo "For $rental £50 per day";;
   "jeep") echo "For $rental £60 per day";;
   "bicycle") echo "For $rental £7 per day";;
   *) echo "Sorry, I can not get a $rental for you";;
esac
```

While loops:

```
while [ condition ]; do
  echo "Something"
done

#example

#!/bin/sh
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

For loops:

```
for $variable in {list}; do
  echo "Something"
done

#example

#!/bin/sh
for ((  i = 0;  i <= 5;  i++  )); do
  echo "Welcome $i times"
done
```

For most modern shells, the use of double brackets is recommended instead of single brackets in conditional statements.

There exist too many comparison operators to list here. Use this resource for reference on how to build up conditional statements.

Some further useful things:

```
expr $x +1 #evaluate expression and print results
```

```
read [variable] #interactively supply value
printf "%s\t%d\t%f\t%.1f\n" "${string}" "${integer}" "${float}" "${float.0}" #prints tab
separated variables (note the specifiers) and a new line character
```

Now try some exercises.

---

awk

`awk`:

```
awk '{print $2,$5;}' [file] #print 2nd and 5th columns from file
awk 'BEGIN { Actions at the start ;} { Actions for every line ;} END { Actions in the end ;}'
[file] #do something, then act on every line of the file, then do something else
```

`awk` has been called a language within a language. It has a set of useful inbuilt variables and it is very powerful. See more here. It can even do its own control flow.

There are some `awk` specific exercises here.

Miscellaneous

There are some other things you should definitely know about if you are going anywhere near programming:

Git and GitHub allow version tracking and collaborative development. Don't worry, nobody understands it. But it is still incredibly useful. Start here (official) or even better here or here.

Make is a peculiar little tool that is worth understanding as it can save lots of time and pain by automating very tedious tasks. Check out the shorter and longer tutorials. A good example of what it can do is to automate git processes for a repository to one command `make all` if the following is saved in the `Makefile`:

```
.PHONY: all

all:
	git add --all
	git commit -a -m "pushed through make"
	git pull
	git push
```

Some excellent resources for much more in-depth shell:

- Advanced Bash-Scripting Guide.
- Short-ish tutorial in bash scripting.
- Linux Shell Scripting Tutorial (LSST) v1.05r3.
- Linux Shell Scripting Tutorial (LSST) v2.0.

This tutorial was written in `Markdown`, which is also a useful thing to learn. Start here.

License