CA170: Week 6

More on Shell

Pipes and redirection in scripts

- See notes on pipes and redirections on command line too
- Multi-line pipe:
  - Shell script can have multi-line pipe. Easier to read. e.g.:

```
cat file |

    grep -i "http:.*dcu.ie"          |

    sed -e "s|COMPAPP|computing|g"   |
    sed -e "s|compapp|computing|g"   |

    sort -u
```

  - This example uses sed to do a "search and replace" operation on text.
- Two output streams:

```
UNIX separates "ordinary output" (standard output, stdout, 1>, >)
from "error output" (standard error, stderr, 2>)

prog > output 2> errors
```

- Null output:

```
> /dev/null      to get rid of some unwanted output
                 e.g. error/warning messages from compilation/search
                 (redirects it into a non-existent file)
```

  - The null device exists on Windows too - link in notes
- To redirect script output to a file, from command line

```
script > file
```

- To redirect script output to a file from within the script, put this at start of script:

```
exec > file
```

  - Then run:

```
script
```

- Program detect if output going to screen or not:
  - Have you noticed this:

```
ls           (gives multi-column output)
ls > file    (gives single-column output)
ls | prog    (gives single-column output)
```

  - You can detect if output is going to terminal, file or pipe, and adjust output accordingly.
  - Shell script to detect where output is going:

```
if [ -t 1 ]
then
 echo stdout
else
 echo pipe or file
fi
```

  - Test it:

```
prog
prog | cat
prog > file
```

- Advanced Bash-Scripting Guide - link in notes
  - Redirection - link in notes

Arguments and returns (more)

```
$1            1st argument
$*            all arguments

$0          name of prog
$#          no. of args

shift       shift args leftwards
            this is useful if you want to remove some of the first args, then
            "shift" a couple of times,
            and then do "for i in $*" with the remaining args
            e.g. grep (switches) (string) file1 ... filen

exit 20           exit with a return code

$?          return code of last prog executed
            e.g. quiet grep:
             grep > /dev/null
            and then check $?
            though grep may have -q (quiet) option anyway
```

Environment variables

- Like global vars for all programs.
- Note any environment vars that are declared within a program are local to that program only.

```
env
printenv
set               may display shell functions too


var=value               set environment variable
                        N.B. no spaces!
echo var                print the string "var"
echo $var               print value of environment variable


echo $HOME              get into the habit of using these
                        instead of the actual hard-coded values,
                        - makes scripts more portable
echo path is $PATH
echo $USER
```

uname and arch

```
echo `hostname`         recall backquotes

uname           show hardware, OS, etc.

arch            same as "uname -m"

echo `arch`     recall backquotes
```

- Uname - link in notes
- man uname - link in notes
- "arch" on DCU Linux may give x86_64 or i686 - link in notes
- "arch" on DCU Solaris may give sun4 or i86pc - link in notes

- Example of using arch in config files:
    1. share the same set of files across a number of Unix/Linux systems running on different hardware.
    2. I collect binaries for each platform, but keep them in separate directories underneath the $home/bin directory.
    3. Then at login I set the path to automatically include the correct directory.
        - e.g. On my C shell system I put this in the .cshrc file:
          `set path = ( $home/bin/`arch` ... )`

Strings and echo

```
echo                    print something on screen, followed by new line
echo -n                 print with no new line


printf                  print with no new line
printf "\n"             print with new line


On some platforms, echo -e exists (interpret special backslash chars)
On DCU Linux:

echo -e "\n text \n\n"        print multiple new lines


echo "string"
echo 'string'

      It is useful to have 2 choices for string - single quote and double quote.
      If using one for something else, surround with the other.
      e.g. To search for single quote in file:

      grep ' file         syntax error (why?)
      grep "'" file       surround with quotes and it works
      grep '"' file       to search for the double quote itself




The 2 forms of string are not equal:

echo "--$HOME--"        --/users/group/humphrys--
echo '--$HOME--'        --$HOME--
echo '--'$HOME'--'      --/users/group/humphrys--
```

File wildcards

```
echo *                  echo all files
echo f*                 all files beginning with f
echo */*                files in next layer
*/*/*                   etc.
```

- Important to realise it is the shell that interprets "*" and passes the result to echo or ls or your program. It is not actually echo or ls itself that parses it.

```
grep string *

# grep does not understand *

# but that's fine because grep does not actually RECEIVE *
# what happens is:
# the shell EXPANDS * to a list of files and passes these to grep
# so grep actually receives:

grep string f1 f2 .. fn
```

- To see that it is the shell that expands it, assign it to an environment variable. Try these:

```
echo *
echo "*"

x=*
echo $x
echo "$x"

x="*"
echo $x
echo "$x"

x=`echo *`
echo $x
echo "$x"
```