${Unix_Tools}
– exercises and solution notes

Markus Kuhn

Computer Science Tripos – Part Ib

1 The shell

Exercise 1: Write a shell command line that appends `:/usr/X11R6/man` to the end of the environment variable `$MANPATH`.

Answer:

```
$ MANPATH="$MANPATH:/usr/X11/man"
```

Exercise 2: Create a new subdirectory and in it five files with unusual filenames that someone unfamiliar with the shell will find difficult to remove. Ask a fellow student to write down for each file the command line that will remove it.

[Your MCS Linux home directory sits on a Windows NTFS/CIFS file server, which does not use a Unix-style file system and has many additional restrictions on filenames. Therefore, better perform this experiment in a local subdirectory, which you can create with "`mkdir /tmp/$USER-odd-names/`".]

Answer:

```
$ mkdir /tmp/$USER-odd-names ; cd /tmp/$USER-odd-names
$ touch -- -i ' ' "\t" 'test.txt ' '...' $(echo -e "\177\177\177")
```

Exercise 3: Given a large set of daily logfiles with date-dependent names of the form `log.`*yyyymmdd*, write down the shortest possible command line that concatenates all files from 1 October 1999 to 7 July 2002 into a single file `archive` in chronological order.

You can fill a directory with 1000 test files using the following `make-logs.pl` Perl script:

```
#!/usr/bin/perl
use POSIX;
for $i (1..1000) {
  $t=rand 60*60*24*365*45;
  $f=strftime("log.%Y%m%d\n", localtime($t));
  $a=localtime($t);
  `echo "$a" >$f`;
}
```

Answer:

```
$ cat log.19991??? log.200[01]???? log.20020[1-6]?? log.2002070[1-7] >archive
```

Exercise 4: Write down the command line that appends the current date and time (in Universal Time) and the Internet name of the current host to the logfile for the respective current day (local time), using the above logfile naming convention.

Answer:

```
$ date -u >>log.$(date +'%Y%m%d') ; hostname >>log.$(date +'%Y%m%d')
```

or

```
$ { date -u ; hostname ; } >>log.$(date +'%Y%m%d')
```

Exercise 5: What outputs will be the result of typing in the following shell command lines (in that order)? Explain why.

```
$ a=1
$ a=2 echo $a ; echo $a
$ a=3 ; echo $a ; echo $a
$ ( a=4 ; echo $a ) ; echo $a
$ { a=5 ; echo $a ; } ; echo $a
$ a=6 bash -c 'echo $a'
$ a=7 ; bash -c 'echo $a'
$ bash -c "echo $a"
$ export a
$ bash -c 'echo $a'
```

Answer:

```
$ a=1
$ a=2 echo $a ; echo $a
1
1
```

The assignment `a=2` merely passes on the environment variable `a=2` to the first invocation of `echo`, which ignores it. It does not affect the shell variable `a` that is substituted in the command-line argument. It does not affect the next invocation of echo at all.

```
$ a=3 ; echo $a ; echo $a
3
3
```

Here, with the added semicolon, the first command now assigns a new value to the shell variable `a`. It does so before any parameter expansion takes place in the second command, therefore all subsequent commands are affected by the new value.

```
$ ( a=4 ; echo $a ) ; echo $a
4
3
```

The assignment of a new value to a shell variable only affects the subshell created by the parenthesis, but not the next command outside this subshell.

```
$ { a=5 ; echo $a ; } ; echo $a
5
5
```

The curly braces do not create a new subshell, therefore any assignments to shell variables inside them persist beyond the closing brace.

```
$ a=6 bash -c 'echo $a'
6
```

An environment variable `a` is passed on to a new `bash` process, which imports it as its own shell variable `a`, applying its value in parameter substitution when executing the command line “`echo $a`”.

```
$ a=7 ; bash -c 'echo $a'
```

Setting the (non-exported) shell variable `a` does not affect a newly started `bash` process, which considers its own shell variable `a` to be empty. The single quotation marks pass "`echo $a`" on to the new `bash` process without first applying parameter substitution.

```
$ bash -c "echo $a"
7
```

The double quotation marks mean that the current shell applies parameter substitution *before* invoking the new `bash` process.

```
$ export a
$ bash -c 'echo $a'
7
```

Marking the shell variable `a` as exported causes it to be passed on to new processes as an environment variable, resulting in the new `bash` process to import its value into its own shell variable `a`, and using it in parameter substitution.

Exercise 6: Configure your MCS Linux account, such that each time you log in, an email gets sent automatically to your Hermes mailbox. It should contain in the subject line the name of the machine on which the reported login took place, as well as the time of day. In the message body, you should add a greeting followed by the output of the "`w`" command that shows who else is currently using this machine.

Answer: Add the following line to one of your startup scripts:

```
{ echo 'Hello there!' ; w ; } | \
  mail ${LOGNAME}@cam.ac.uk -s "Login on `hostname` on `date`"
```

Add this line to the file '`.bash_profile`', which will be executed whenever you log in via the text console (press Alt-Ctrl-F1) or remotely (e.g. `ssh linux.ds.cam.ac.uk`). Where to add it such that it gets executed when you log in via the X Window System depends a lot on the specific installation that you use (usually you have to use `~/.xsession`).

[The backslash at the end of the first line says that this command line continues in the next line.]

Exercise 7: Explain what happens if the command "`rm *`" is executed in a subdirectory that contains a file named "`-i`".

Answer: The file '`-i`' is placed by the pathname expansion as one of the first words onto the command line. The rm command will recognize it as the command line option that asks for the interactive confirmation of each filename before it is deleted. (Some people place an empty '`-i`' file in important directories as a safeguard against an accidentally executed '`rm *`')

Exercise 8: Write a shell script "`start_terminal`" that starts a new "`xterm`" process and appends its process ID to the file `~/.terminal.pids`. If the environment variable `$TERMINAL` has a value, then its content shall name the command to be started instead of "`xterm`".

Answer:

```
#!/bin/bash
if [ "$TERMINAL" != '' ] ; then
  $TERMINAL &
else
  xterm &
fi
echo $! >>~/.terminal.pids
```

Exercise 9: Write a further shell script "`kill_terminals`" that sends a SIGINT signal to all the processes listed in the file generated in the previous exercise (if it exists) and removes it afterwards.

Answer:

```
#!/bin/bash
if [ -f ~/.terminal.pids ] ; then
  for i in `cat ~/.terminal.pids` ; do
    kill $i
  done
  rm ~/.terminal.pids
fi
```

---

2 Text tools

Exercise 10: Write down the command line of the single `sed` invocation that performs the same action as the pipe

```
head -n 12 <input | tail -n 7 | grep 'with'
```

Answer:

```
$ sed -e '6,12!d' -e '/with/!d' input
```

---

3 File tools

4 Revision control

Exercise 11: Generate a Subversion repository and place all your exercise solution files created so far into it. Then modify a file, commit the change, and create a patch file that contains the modification you made. And finally, retrieve the original version of the modified file again out of the repository.

5 Build tools

Exercise 12: Add a `Makefile` with a target `solutions.tar.gz` that packs up all your solutions files into a compressed archive file. Ensure that calling `make solutions.tar.gz` will recreate the compressed package only after you have actually modified one of the files in the package.

Exercise 13: Write a C program that divides a variable by zero and execute it. Use `gdb` to determine from the resulting `core` file the line number in which the division occurred and the value of the variable involved.

Answer:

```
$ ulimit -S -c unlimited            # enable generation of core files
$ cat >t.c <<EOF
#include <stdio.h>
int main(void) {
  int a = 0;
  a = 1/a;
  printf("a = %d\n", a);
  return 0;
}
EOF
$ gcc -ggdb -o t t.c
$ ./t
Floating point exception (core dumped)
$ gdb t core
[...]
Core was generated by `./t'.
Program terminated with signal 8, Arithmetic exception.
[...]
#0  0x08048347 in main () at t.c:4
4         a = 1/a;
(gdb) p a
$1 = 0
(gdb) q
```

---

6 Perl

Exercise 14: When editing sentences, users of text editors occasionally leave some word duplicated by by accident. Write a Perl script that reads plain text files and outputs all their lines that contain the same word twice in a row. Extend your program to detect also the cases where the two occurrences of the same word are separated by a line feed.

Answer:

```
#!/usr/bin/perl
# spot consecutive repetition of the same word
while (<>) {
    # split line into array of words, separated by space or punctuation
    @words = split /[\s.,\'\`\(\)\[\]\{\}]+/;
    # compare neighboring words
    for ($i = 0; $i < $#words; $i++) {
        if ($words[$i] eq $words[$i+1]) {
            print "$_ => double '$words[$i]'\n";
        }
    }
    # compare first word of this line with last word of previous line
    if ($words[0] eq $prevword) {
        print "$prevline$_ => double '$prevword'\n";
    }
    $prevword = $words[$#words];
    $prevline = $_;
}
```
