Computer Science 340

Operating Systems

Assignment 1

SOME SHELL SCRIPT EXAMPLES

NOTICE that all the files I use in these examples have names which begin with the letters ss. If your files have names which don't begin with ss, then they should be safe.

At the end of this document, you will find a set of shell scripts called { ss*n*, 1 ≤ *n* ≤ 9 }. These are supposed to be a set of graded examples illustrating some of the properties of the shell language which you might find useful, both for general understanding of Unix and for the first assignment. If you think they might help, copy them (most easily from the copy of this document in the WWW course pages) into your directory, look at them, and try them. To execute a script, first make sure it's executable (use **chmod** – see the *DOING THINGS WITH UNIX* sheet), then just type its name.

To get best value from the examples, make sure that you know what each script does before moving on to the next; some of them illustrate in passing properties of the shell or of the Unix instructions additional to their main purposes. In each case, look at the script, look at what it does when executed, and work out how it happened. Do not be afraid to change the script if it will help you to understand it.

I think I've fitted in rather more information than all you need to answer the assignment; don't try to find an answer which uses everything that's here. There are several possible answers, and I think that everything here could be used in one of them, but once you find a good answer it is wise to stick with it. If you need more information, don't forget man. Experiment with the methods in the examples to see how far they will stretch. Then try making some shell scripts of your own.

Observe that in all the examples the text begins with a line containing only the character #. That's to make sure that if the files are run from the C shell, they are interpreted by the C shell; if you begin in the C shell, and the first character isn't #, your script is interpreted by the Bourne shell. Different shells have different characteristics, and these scripts might not run on shells other than the C shell. A more general way of ensuring that a script is run by a shell other than the current shell is to put a line like

```
#!<shell_pathname>
```

as the first line of the script; this works in the C shell, and seems to work in some others as well. Check the documentation for your shell to be sure.

THE EXAMPLES, WITH NOTES.

ss1 : Using the echo instruction to display things on the screen – and, incidentally, to build files. No, it isn't a very efficient way to build files, but it works.

```
#
# ss1 : Building files with "echo".
echo FIRST LINE > sslogfile
echo Second line > sslogfile
echo Third line >> sslogfile
# ">" starts a new file; ">>" appends.
cat sslogfile
rm sslogfile
```

---

ss2 : Using the echo variable to see what happens when you execute a shell script. The echo variable has nothing whatever to do with the echo instruction.

```
#
# ss2 : setting the echo variable. This file does the same thing twice.
echo List all the "ss" files.
ls ss*
# Repeat with the "echo" flag on.
set echo
echo List all the "ss" files.
ls ss*
```

---

ss3 : Building and running a shell script. You (as an intelligent agent) would not normally use a "here document" to do it, but in this case the agent is the shell.

```
#
# ss3 : Making a Shell script.
cat > sstestscript << *
ls t*
cat ss3
*
# The bit between "<< *" and "*" is a "Here document";
# it's used as the input to ( the second ) cat.
sstestscript
# That didn't work. You can use "csh sstestscript" if you wish.
echo MUST MAKE THE SCRIPT EXECUTABLE !
chmod a+x sstestscript
# "chmod" changes the protection mode.
sstestscript
echo Here\'s sstestscript :
cat sstestscript
rm sstestscript
```

---

ss4 : Processes leave behind result codes, or condition codes, in the status variable. Sometimes these tell you something about what happened during the process.

```
#
# ss4 : The status variable - did the programme work ?
rm sstestscript
cat sstestscript
echo status is $status
cat > sstestscript << *
 A line.
*
cat sstestscript
echo status is $status
rm sstestscript
```

---

ss5 : Testing things. Notice that the equality operator is ==, = is reserved for assignment. The primary mechanism for evaluation is string substitution, so the line

```
if ( x$1 == x ) then
```

is first expanded by substituting the value of the first argument for $1, then evaluating the logical expression in the parentheses.

```
#
# ss5 : Conditions and arguments.
if ( x$1 == x ) then
   echo Try again with some arguments : ss5 one two three.
else
   echo The first argument was \"$1\".
   echo The full argument string was \"$*\".
   echo There are $#argv arguments.
   echo Try an argument with wildcard characters - ss5 \*.
endif
```

---

ss6 : Loops, variables, and some wild characters. The line

```
else if( -e ss$count ) then
```

illustrates a way of getting information about files into the process. ss$count is a file name; –e is an operator which means "it exists". The condition is therefore "if the file exists ...". Other operators of similar type are :

- -d the file is a directory;
- -f the file is an ordinary file;
- -r the file is readable.

Conditions can be combined with && (and) or || (or), and negated with !.

```
#
# ss6 : Iteration and variables.
if ( x$1 == x ) then
   echo Try again with some arguments : ss6 one two three.
   exit
else
   echo The full argument string was \"$*\".
# "thing" is the name of a variable; "$thing" is its value.
   foreach thing ( $* )
      echo The next argument is $thing.
   end
endif
# - but "*" is all the files, and "$*" is all the arguments -
foreach thing ( * )
      echo The next file is $thing.
   end
# A counting loop.
set count = 1
while ( $count <= 8 )
   if( $count == 6 ) then
      echo I know about ss6 !
   else if( -e ss$count ) then
      echo Have you tried ss$count yet \?
   else
      echo What happened to ss$count \?
   endif
# Add 1 to count.
   @ count++
end
```

---

ss7 : Using `grep` to look for strings. The programme `grep` will search a file or files for occurrences of a string, and report those that it finds.

```
#
# ss7 : grep : find strings in files; /dev/null is a black hole.
#
# This line does not contain
# grep, but this one does.
# The next line
# containsgrepaspartofsomethingelse.
# Now look for all the lines containing GREP -
# perhaps.
grep "grep" ss7
echo End of the first grep.
echo For the next trick, we''ll sort the output using a pipeline.
grep "grep" ss7 | sort
echo End of the next trick.
# /dev/null is a "null device" : things sent to it vanish.
grep "grep" ss7 > /dev/null
echo grep status is $status.
grep "grep" ss6 > /dev/null
echo grep status is $status.
```

---

ss8 : Using `sed` to edit things. The name `sed` is an abbreviation for "stream editor", and that's primarily what it is; it is intended for use as a Unix "filter", not as a conventional text editor, so the results of its operations come out at the standard output, leaving the original file (if there is one – it's just as happy to receive all its input from the standard input) unchanged.

```
#
# ss8 : sed, the stream editor.
set stars = zzz
sed "1,/z$stars/ d" ss8 > ss8test
set echo
sed '3,6 d' ss8test
sed '/six/,$ d' ss8test
sed '/o/ s/line/LINE/' ss8test
unset echo
rm ss8test
exit
zzzz
line one
line two
line three
line four
line five
line six
line seven
line eight
line nine
line ten
```

---

ss9 : Using `sed` again, this time taking its instructions from a script. The principle is the same, but you can put a lot more instructions in the script than makes sense as a parameter.

`sed` is a bit temperamental; you have to get the instructions exactly right. I spent a day trying to find out why one of two apparently identical script files worked and one didn't. I finally found that the one that didn't had a space at the end of a delete instruction after the d. Then I tried it on the `sed` that comes with Linux, and it worked perfectly.

```
#
# ss9 : sed, the stream editor, and scripts.
set echo
set stars = zzz
sed "/y$stars/,/z$stars/ w ss9script" ss9
sed "1,/z$stars/ d" ss9 > ss9test
sed -f ss9script ss9test
rm ss9test ss9script
exit
# yzzz
/o/ s/line/LINE/
/five/,7 d
2,4 d
# zzzz
line one
line two
line three
line four
line five
line six
line seven
line eight
line nine
line ten
```

Alan Creak,
July, 1998.