UNIX Shell Scripts

The shell should be familiar to all UNIX users as the front-end process that provides access to the operating system. The shell takes input from the keyboard and directs output to the terminal (or a terminal window). The input takes the form of a command which, if properly formed, runs a program. The shell is therefore an interpreter: it parses a line of input for syntactical correctness and passes the information to the operating system for execution. A line of input to the shell, then, is essentially a one-line program.

The shell will also interpret a sequence of commands listed in an ordinary text file. This file is called a "shell script." It provides a convenient method for grouping a set of commonly executed commands. Following is an example of a short shell script:

```
#!/bin/csh
echo Shell script begin.
pwd
echo Shell script end.
# End of script
```

The script is an ordinary text file named, in this case, "my_script". The first line is special; it specifies the shell that will be used to interpret the script. A UNIX system commonly provides a choice of shells to the user: the original AT&T Bourne shell, /bin/sh; the Berkeley C shell, /bin/csh; and we set up new accounts with Ohio State's enhanced C shell, /usr/local/bin/tcsh. The shell you type into is not the process that will actually interpret the script; rather, the shell named in the first line of the script will be forked as a separate process to do that. There are differences between the shell languages, so the choice is significant. In general, stick with /bin/csh unless special needs arise. Also, the '#' can always be used to start a comment in a script. The remaining lines are the actual commands to run. Any valid UNIX commands can be listed, including programs you have written yourself.

Executing the script is a simple matter of typing the script's name at the prompt:

```
> my_script
my_script: Permission denied.
```

Oops! Forgot one little detail. The script must be given execute permission:

```
> ls -l my_script
-rw-r--r--  1 jxxl      79 Mar 18 15:05 my_script
> chmod u+x my_script
> ls -l my_script
-rwxr--r--  1 jxxl      79 Mar 18 15:05 my_script
```

Now the script can be executed:

```
> my_script
Shell script begin.
/work/jxxl
Shell script end.
```

In addition to executing commands that could be typed directly to the shell, a shell script can also contain more advanced programming features like loops, conditional execution, and setting variables. These features are documented in the man page for each shell, but following are a few /bin/csh examples:

- Setting and using variables:

```
# character values
set name = "Bill"
echo $name
# command output into variable
set dirlist = `ls`
# integer values
@ count = 0
```

- While loop with counter:

```
@ count = 0
set dlist = `ls`
while ($count < 3)
  @ count = $count + 1
  echo $dlist[$count]
end
```

- For loop and conditional statement:

```
unalias ls
foreach i (`ls`)
  # if file is directory...
  if (-d $i) then
    echo Removing $i
    rm -r $i
  else
    echo $i is not a directory
  endif
end
```

- Access command line arguments:

```
if ($#argv == 0) then
  echo Usage: my_script additional arguments...
  exit 0
endif
foreach j ($argv)
  echo $j
end
```

- Case statement:

```
set today = ‘date‘
switch ($today[1])
  case “Mon”:
    @ day = 0
    breaksw
  case “Tue”:
    @ day = 1
    breaksw
# ...
  case “Sun”:
    @ day = 6
    breaksw
  default:
    echo Bad date format
endsw
```