Creating & Testing Command Scripts

1 Getting Started with Command Scripts

1. Create a file called `Hello` with the following line:
   `echo "Hello, world."`
2. Execute `Hello`.
`Hello: Permission denied.`
3. Make Hello executable:
   `chmod u+x Hello`
4. Execute `Hello`.
`Hello, world`

2 Ensuring That the C Shell Executes Your Script

- Insert the following as the last line in your `.cshrc` file:
  `echo "Goodbye, world."`

5. execute `Hello`.
`Hello, world`

- On my system, the `Hello` shell script was executed by the **Bourne** shell.
- To ensure that the **C** shell executes the shell insert the following 1st line into `Hello`:
  `#! /bin/csh`

6. Execute `Hello`.
```
Goodbye, world.
Hello, world.
```
7. Delete the echo from your `.cshrc` before you forget.

- You can invoke **options** from the first line of your script.

8. Modify the 1st line of Hello to be:
   `#! /bin/csh -v`
   This turns on the verbose option.

- Another way to ensure that the C shell is executing your script is the execute:
  `csh Hello`
  or
  `csh -v Hello`
  That is, execute the **command** `csh` [with options specified].

3 Comments and Continuation Lines

- Comments, which can begin **anywhere** on a line:
  - begin with `#`;

- end with a newline;
- can be continued with a `\` (as the last character on the line).

4 Passing Arguments to the Command Script

- Create a 1-line shell script, called backup, as follows:

```
cp $1 Back.$1
```

- `$0` refers to the command itself; `$1` refers to the 1st command-line argument, etc.
- `$#argv` refers to the number of arguments on the command line (including the command itself).

5 Debugging Command Scripts

- C shell options:

  `-n`: report only the syntax of the script.

  `-v` : echo each command line before execution, but **after** history substitutions have been made.

  `-e`: terminate (with extreme prejudice, whatever that means?) the command script, when any command line terminates abnormally or returns a non-0 exit status.

  `-f`: bypass the execution of the .cshrc file when executing this script.

  `-X, -V`: verbose, as `-v`, but take place even in the `.cshrc` file. (Good for debugging changes to the `.cshrc` file.)

- Execute `man csh` for more details and other command line options.

6 Serializing Commands

- Commands can be executed without waiting for their termination by appending an `&` on the command line.

  ```
  cc test.c &
  ```

  executes the compilation in the background.

- Conditional pipes can be formed:

  ```
  command1 || command2
  ```

  If (execution of `command1` terminates abnormally)

  then `command2` is executed. (and **only** if)

  Similarly,

  ```
  command1 && command2
  ```

  If (execution of `command1` terminates successfully) then command2 is executed. (and **only** if)

```
s> false && echo Wow!
s> true && echo Wow!
Wow!
```

- Each of the commands above can be replaced by a **sequence** of commands by **enclosing them in paranzstheses**.

```
s> true && (echo A || echo B)
A
```

7 Including a File

The analog of the #include preprocessor command is the source command.

One can put shell scripts withing shell scripts.

By using a source command, you ensure that the commands in the sourced script file are executed within the same shell, rather than as a subshell.

This is useful when you want to have access to variables that are defined in the parent shell.

```
Help:
   set xxx = Help
   SayWhat

SayWhat:
   echo $xxx
```

Executing Help produces:

```
xxx: Undefined variable.
```

If we change the SayWhat command to source SayWhat:

```
Help:
   set xxx = Help
   source SayWhat
```

When we execute Help, we get:

```
Help
```

- Sourcing is used much the same as #include is used —to bring in common variables.

8 Good Programming Practices for C Shell Scripts

- The 1st line of a command script should contain the name of the shell under which it is to run (C, Bourne, Korn).

```
#! /bin/csh
```

- Then you should comment:
  - **what** the script does (not how),
  - its command syntax and what the arguments are (like a miniature **man page**).
- The 1st section of executable code should test each of the command line arguments to verify that they are as expected.

```
#! /bin/csh
#
# This script copies 1 file to another.
#
# The 1st argument is the input file;
# The 2nd argument is the output file.
#
if ($#argv < 3) then
   echo "Expected syntax is:"
   echo " copyit InputFile OutputFile"
   echo " "
   echo " where:"
   echo "InputFile is the name of copied file"
   echo "OutputFile is the name of copy"
   exit (1)
endif
# remainder of script goes here
```

- When you exit abnormally , as above, you should exit with a non-0 value;
- When you exit normally, you should exit with a 0 value (just say `exit`).
- Be careful about referencing commands in a script. Depending on the working directory and path, you may get undesired results. One way to avoid this is to set the path within the script.

```
# Save current path for later
set original = $path
# Prepend path with directories used
set path = (/desired/directory $path)
#
#  execute commands
#
# restore original path
set path = ( $original )
```