template

header
- item1
- item2

- item3

header
- make some files of our own (text for now, programming later)
- put them somewhere.

Session: Shell Scripts and Programming
Topic: Shell Programming

Daniel Chang

Shell Scripts

- An executable ordinary file containing

  1. Zero or more UNIX commands

  2. Zero or more Programming structures

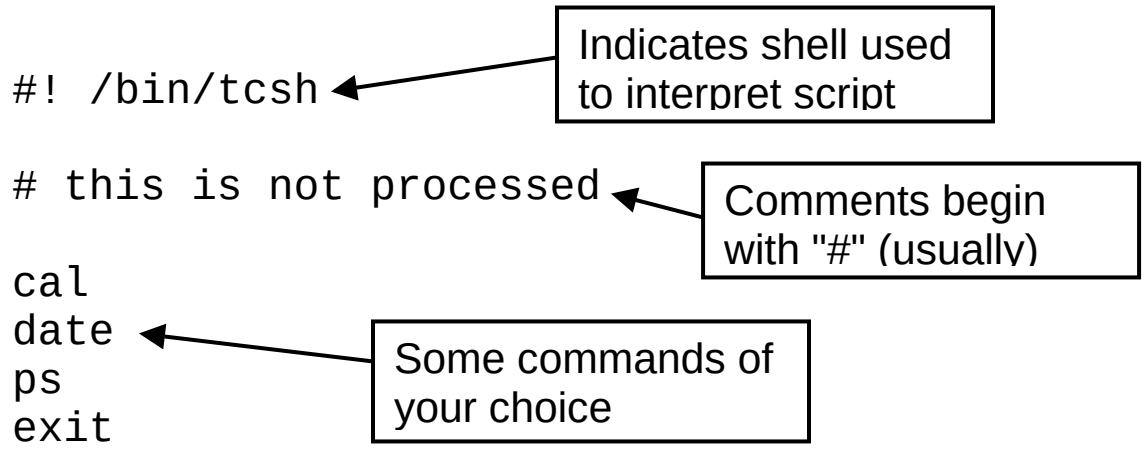
- A basic form of "program"

- Commands and Programming Structures are specific to various shells

- Script should specify the shell used to interpret the script

- Useful when operations must be repeated

Basic Shell Script Contents



- Note the "#!" before the shell indicator
- The "exit" command immediately quits the shell script (which otherwise quits at end)

Executing Shell Scripts

- Use the shell as a command with the script

```
> sh myscript    # uses bourne shell
> tcsh myscript  # t-cshell
```

- Set "execute" permission on the script file, then run as a command

```
> chmod 755 myscript
> myscript
```

Variables

Environment Variables

- Variables provided by shell as part operation
- E.g. USER, HOME, PATH, SHELL, HOSTNAME
- You can change them, but you do not initially create them (exist at startup)
- The "setenv" command views and sets environment variables

```
setenv path $path:\home\here\bin
```

(Sets "path" variable to its current value plus \home\here\bin)

User-Defined Variables

- Variables you specify yourself (can be used in a script)
- Use the "set" command, specifying a variable name not already in use

```
set mywebpath = ~dchang/public_html
```

- To use the value inside the variable place a "$" in front of the variable name

```
ls -l $mywebpath
```

- Typically User-Defined variables are in all lower case to distinguish them from Environment Variables and other Shell Variables.

`setenv` `[variable [value]]`

Description: Sets the value of an environment variable. With no parameters "setenv" displays current environment variables and their values.

Options:

- [variable] Some UNIX environment variable.
- [value] A string value

Example:

```
setenv path $path:\home\here\bin
```

(Sets the path variable to its current value plus \home\here\bin)

`set` `[variable = value]`

Description: Sets the value of a user shell variable. With no parameters "set" displays the current user shell variables and their values. Specifying a variable name not currently in use will create a new user shell variable. Variable settings will stay active until you logout or change the value.

Options:

- [variable] Some UNIX shell variable.
- [value] A string value

Examples:

```
set workdir=\~dchang\sample
```

Positional Parameters

- Also called "read-only" parameters
- Numbered $1 - $9 and correspond to the parameters entered as part of the execution of the command (shell script)
- $0 is always the command itself, and each additional parameter is separated by a space
- $# contains the actual number of parameters, starting with the first parameter (not the command itself)
- $* is a string that contains the entire command with all parameters

Strings and Quotes

- Certain "special characters" have special meanings

```
& * \ | [ ] { } $ < > ( ) # ? ' " / ; ^ ! ~ % `
```

- To use one of these characters in a string you must "escape" the character, meaning to precede it with a backslash (\)
- The backslash only applies to the character immediately following it

```
echo What \#\!\* time is it\?
```

Quote Mark	Effect
\	Cancels the special meaning of a single character within a string
'somestring'	Cancels the special meaning of any and all special characters within the quote marks
"somestring"	Cancels the special meaning of any special characters within the quote marks except $, ``, and \
`somestring`	Run the string specified within the quote marks as a command. The output of the command will replace `somestring`. (Note: the ` is a backquote.)

Arithmetic Operations

- The "expr" shell command can perform simple integer math

`expr` `value1 operation value2`

Description: Performs a simply integer operation on two values, then outputs the result

Options:

- [value1], [value2] Some integers
- [operation] Mathematical operations (*) (+) (-) (/) (%). When necessary the operator must be escaped with a backslash.

Example:

```
set sum = `expr 1 + 2`
set final = `expr $sum \* 2`
echo $final
```

Programming Structures

```
if test then
  commands1
[else
  commands2]
endif
```

Description: "test" is evaluated, and if true (non-zero) "commands1" will be executed. If the "else" clause is specified, "commands2" will be executed if "test" evaluates to false (zero).

Options:

- [commands1], [commands2] Any number of shell commands

Example:

```
    if ($1 == $2) then
      echo "$1 is equal to $2"
    else
      echo "$1 is not equal to $2"
      exit
    endif
```

"Test" Expressions

Test	Effect
`val1 == val2`	Compares two values for equality
`val1 != val2`	Compares two values for inequality
`val1 > val2`	Tests if "val1" is greater than "val2"
`val1 < val2`	Tests if "val1" is less than "val2"
`val1 >= val2`	Tests if "val1" is greater than or equals "val2"
`val1 <= val2`	Tests if "val1" is less than or equals "val2"
`-e filename`	Tests if "filename" exists
`-d filename`	Tests if "filename" is a directory
`-f filename`	Tests if "filename" is an ordinary file
`-r filename`	Tests if "filename" is readable
`-s filename`	Tests if "filename" size is greater than 0
`-w filename`	Tests if "filename" is writeable
`-x filename`	Tests if "filename" is executable

Shell Script Parameters

- You will receive a "Missing file name" error if you try to pass in and use a parameter to your shell script that starts with a comparison flag (e.g. "`-domore`")
- Unused test expression flags: `a, h, i, j, n, q, v, y`
- Use double quotes and single quotes or double-aliasing for flags that would otherwise be treated as Test Expressions

```
if ("$1 == '-r') then
  set rflag = true;
  echo "you chose option -r"
else

if ("x$1 == "x-d") then
  echo "you chose option -d"
else
```

```
while test
  commands
end
```

Description: "test" is evaluated, and if true "commands" will be executed. This process will then be repeated until "test" evaluates to false.

Options:

- [commands] Any number of shell commands

```
foreach item (wordlist)
  commands
end
```

Description: For each item in "wordlist", the single item is assigned to "item" and then "commands" are executed.

Options:

- [item] A variable that will be assigned the word from the "wordlist" that is in current use.
- [wordlist] any list of one or more words (for example, a directory listing)
- [commands] Any number of shell commands

Example:

```
foreach filename (*)
  if (-d $filename) then
    echo $filename is a directory
  endif
end  # foreach filename
```

```
switch (string)
  case string1
    commands1
    breaksw
  case string2
    commands1
    breaksw
  [...]
  default
    commandsdef
endsw
```

Description: "string" is matched against each "stringX" in the cases. For the first case that matches, execution will jump to the "commandsX" within that case, then the switch statement ends. There can be multiple cases. The "default" case is executed if no match occurs.

A "switch" essentially works like a multiple "if-then-else" statement.

```
goto label
```

Description: Immediately transfers execution to the instruction following "label". Labels must be on a line by themselves, with a trailing colon

Example:

```
    goto enditall
    [...]
    enditall:
    exit
```

'sh' or 'bash' - Shell Programming

read *variable*

Description: Prompts for input and sets "variable" to the input. Can be used to input values during the execution of a shell script.

Example:

```
echo "What is your name?"
read name
echo "Your name is $name"
```

set string1 ...

Description: Within a shell script, will set the positional parameters one by one to whatever strings are specified.

Example:

```
set `date`
echo "Time: $4 $5"
```

(`date` will be replaced by the output of the "date" command, then each part of the date command will be assigned to a positional parameter)

'sh' or 'bash' - Shell Programming

```
if test
then
  commands1
[ elif elseif-test
  then
    commands2]
[else
  commands3]
fi
```

Description: This format allows for multiple tests to be performed using an "else if" structure.

Options

- [test] Must be performed using the "test" command, followed by the actual test,
- Note that some "tests" are different than for the c-shell.

Test	Effect
`val1 -eq val2`	Compares two numeric values for equality
`val1 -ne val2`	Compares two numeric values for inequality
`val1 -gt val2`	Tests if "val1" is greater than "val2"
`val1 -lt val2`	Tests if "val1" is less than "val2"
`val1 -ge val2`	Tests if "val1" is greater than or equal to "val2"
`val1 -le val2`	Tests if "val1" is less than or equal to "val2"
`str1 = str2`	Compares two strings for equality
`str1 != str2`	Compares two strings for inequality

'sh' or 'bash' - Shell Programming

```
for item (wordlist)
do
  commands
done
```

Example:

```
    for test in "one" "two" "three"
    do
      echo "$test"
    done  # for
```

```
while condition
do
  commands
done
```

```
until condition
do
  commands
done
```