Lecture #10 - Interactive Korn Shell (Chapter 11)

- Background

  Usually ksh
  Programming interface is a superset of Bourne shell
  Adopted many of the features from both Bourne and C shells

  Adds report formatting, built-in math, data types, control flow, etc

- Startup Files

  /etc/profile, then .profile (just like Bourne shell)
  ENV (usually set to .kshrc)
  Since .profile is used by Bourne shell as well, we often keep KSH specifics in ENV script

- Korn shell options

  set -o option (turn on option)
  set +o option (turn off option)
  Examples: allexport, vi, emacs, ignoreeof

  PS1 is prompt variable like Bourne shell
  Example: $ PS1='! $PWD> ' gives command number followed by working dir

- Order of processing commands

  1. Keywords (if, while, until, etc.)
  2. Aliases
  3. Bult-in commands
  4. Functions
  5. Scripts and executables

- Korn shell aliases

  similiar to c-shell
  alias [-x] name=command
  x option exports the alias for child processes
  t option makes the alias tracked (done after alias is created)
  tracked aliases use absolute paths for cmds to avoid path lookup

- Command line history

HISTFILE	controls the name of the history file
HISTSIZE	controls the number of commands stored in history

```
$  history                # same as fc –l
1   ls
2   vi file1
3   df
4   history

$  history 8              # List from the 8th command to present
8   echo $USER
9   set
10  history
11  history –n
```

Commands can be re-executed with the “r” command:

```
$  r date
date
Mon May 15 12:27:35 PST 2004
$  r 3
ls
file1    file2    file3
```

- Command line editing

As with TCSH, Korn shell provides command line editing with vi or emacs.

To enable with “vi”, use one of the following:

```
$  set –o vi
$  VISUAL=vi
$  EDITOR=vi
```

Command commands:

<ESC> k	Move up the history list
<ESC> j	Move down the history list
/string	Search upward through the history list
?	Search downward through the history list

- Common features

Job control and filename generation are the same as C shell.

- Variables

  consists of letters, digits, and underscore
  cannot start with digit
  common to use UPPERCASE for exported variables and mixed case for everything else.

  assigned and referenced like Bourne shell

  ```
  $ VAR=value
  $  echo $VAR
  ```

- Variable Expressions

${var:-word}	If var is set and nonnull, substitute its value, otherwise use word
${var:=word}	If var is not set or is null, set it to word
${var:+word}	If var is set and nonnull, substitute word, otherwise nothing
${var:?word}	If var is set and nonnull, substitute its value, otherwise print word and exit from shell.

Example:

```
$  fruit=peach
$  print ${fruit:-plum}
peach
$  print ${newfruit:-apple}
apple
$  print $newfruit

$  print ${TERM:-vt120}
dtterm
```

${var%pattern}	Matches smallest trailing portion of value and removes it
${var%%pattern}	Matches largest trailing portion of value and removes it
${var#pattern}	Matches smallest leading portion of value and removes it
${var##pattern}	Matches largest leading portion of value and removes it

Example:

```
$  pathname=”/usr/bin/local/bin”
$  print ${pathname%bin*}
/usr/bin/local
$  print ${pathname%%bin*}
/usr
```

- Variable Attributes

typeset -u NAME	(convert values to uppercase)
typeset -l NAME	(convert values to lowercase)
typeset -i NAME	(integer)
typeset -i# NAME	(integer with base #)
typeset -x NAME	(same as export)
-Lwidth	(left justify within width)
-Rwidth	(right justify)
-LZwidth	(left justify within width and strip leading zero)
-RZwidth	(Right justify, fill with leading zeroes if value starts with a digit)
-Zwidth	(same as -RZ)

Examples:

```
$  typeset –u name=”john doe”
$  print $name
JOHN DOE
$  typeset –L4 name
$  print $name
JOHN
$  typeset –R2 name
$  print $name
HN
```

- Operations on variables

${name:startpos}	
${name:startpos:len}	
${name/pattern/replace}	replace first occurence of pattern
${name/#pattern/replace}	pattern must be at first of string
${name/%pattern/replace}	pattern must be at end of string
${name//pattern/replace}	replace all occurences of pattern

- Arrays and Compound variables

one dimensional arrays with index from 0 to 511

```
$ PRES[0]=Washington
$ PRES[1]="Adams, J"
$ PRES[2]=Jefferson
```

OR

```
set -A PRES Washington "Adams, J" Jefferson
```

```
$ echo $PRES[2]
Jefferson
$ echo $PRES[*]
Washington Adams, J. Jefferson
```

Associative arrays (arrays with arbitrary indices)

```
$ VICEPRES=([Washington]="Adams, J." [Jefferson]="Burr, Clinton")
$ echo ${VICEPRES[Washington]}
Adams, J.
```

Compound variables (similar to C structures or Pascal records)

```
$ name=""
$ name.first=Robert
$ name.last=Stevenson
```

OR

```
$ name=(first=Robert last=Stevenson)
```

- Arithmetic

```
$ let COUNT=COUNT+1 VALUE=VALUE*10+NEW
$ echo There are $((60*60*24*365)) seconds in a non-leap year

$ x=23
$ y=37
$ echo $((2*x + 3*y))
157
$ echo $((2*$x + 3*$y))
157
```

floating point (typeset -F X)

- Command substitution

New syntax is available:

```
$  dir=$(pwd)
```

Easier to read that the backquotes, and more orthogonal as well (why?)

- Functions

```
func_name()          like Bourne shell
{
  cmds
}

function func_name
{
   cmds
}
```

The first form is executed in same env as current shell.

Functions can be removed with unset -f func_name

typeset command can be used in functions to make variables local

Example:

```
$  count=10
$  function count_down
>  typeset count
>  count=$1
>  while [[ $count > 0 ]]
>  do
>     echo $count ...
>     count=`expr $count - 1`
>  done
>  echo Blast Off!
>  return
}
$  echo $count
10
$  count_down 6
6 ...
5 ...
4 ...
3 ...
2 ...
1 ...
Blast Off!
$  echo $count
10
```