Unit 2
Shell Variables

Unit Objectives

After completing this unit, you should be able to:

- Set variables
- Reference variables
- Use positional parameters
- Shift arguments
- Set positional parameters
- Use shell parameters
- Understand inheritance
- List shell variables
- List environment variables

Setting Variables

To assign a value to a variable: **name=value**

```
$ var1=Fri
$ _
```

To "unset" the value to a variable:

```
$ unset var1
```

To protect a variable against further changes:

```
readonly name=value
      - or -
typeset -r name=value
```

```
$ readonly var1=Sun
$ var1=Mon
ksh: var1: This variable is read only
$ _
$ readonly -p
```

displays full list

Referencing Variables

To reference a variable, prefix name with a **$**

```
$ print $var1
Fri
$ _
```

To separate a variable reference from other text use: **${ }**

```
$ print The course ends on $var1day
The course ends on
$ print The course ends on ${var1}day
The course ends on Friday
$ _
```

Positional Parameters

Parameters can be passed to shell scripts as arguments on the command line

```
$ params.ksh arg1 arg2
```

- "arg1" is positional parameter number 1
- "arg2" is positional parameter number 2
- Others are unset

They are referenced in the script by:

- **`$1`** to **`$9`** for the first nine
- **`${10}`** to **`${n}`** for all after the first nine

Setting Positional Parameters

In a shell script the **set** command can:

- Change the values of positional parameters
- Unset positional parameters previously set

```
$ cat first.ksh
print $1 $2 $3
set apple banana
print $1 $2 $3

$ first.ksh a b c
a b c
apple banana

$ _
```

Variable Parameters

Shell scripts set a number of other shell parameters:

$# The number of positional parameters set

$@ Positional parameters in a space separated list

$* Positional parameters in a list separated by the first **I**nternal Field Separator (the default is a space)

In double quotes, $@ and $* behave differently:

"$@" = "$1" "$2" "$3" . . .

"$*" = "$1 $2 $3 . . . "

Some Shell Parameters

Shell parameters that remain fixed for the duration of the script:

$0	The (path)name used to invoke the shell script
$$	The **P**rocess **ID** (PID) of current process (shell)

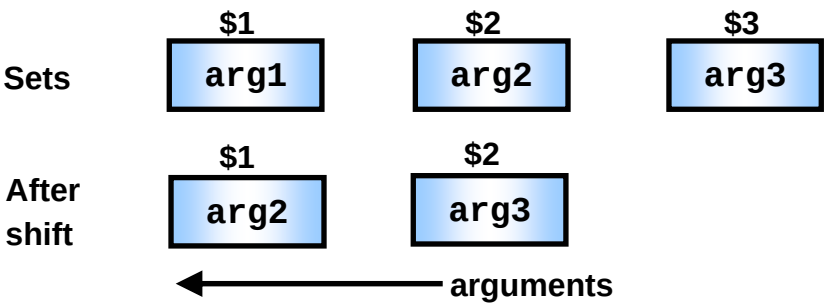
Parameters set as the script executes commands:

$!	The PID of the last background process
$?	The return code from the last command executed

Shifting Arguments

In a shell script the **shift** command moves arguments to the left:

```
$ params.ksh arg1 arg2 arg3
```



- Discarding the first or leftmost argument
- Decrementing the number of positional parameters
- Allowing Bourne shell to reference more than 9 arguments

Parameter Code Example

So, let's put all of it into action in a shell script.

```
$ cat second.ksh
print $$
print $0
print "$# PPs as entered"
print "PP1=$1 PP2=$2 PP3=$3 PP4=$4"
shift
print $0
print "$# PPs after a shift"
print "PP1=$1 PP2=$2 PP3=$3 PP4=$4"
set "$@"
print 'Set "$@" - parameters in double quotes'
print "PP1=$1 PP2=$2 PP3=$3 PP4=$4"
set "$*"
print 'Set "$*" - parameters space separated'
print "PP1=$1 PP2=$2 PP3=$3 PP4=$4"

$ _
```

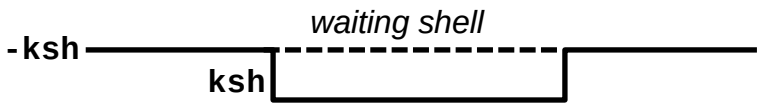
Parameter Output Example

Here's what it does.

```
$ second.ksh Atlanta NYC "Chicago and D.C."
4687
second.ksh
3 PPs as entered
PP1=Atlanta PP2=NYC PP3=Chicago and D.C. PP4=
second.ksh
2 PPs after a shift
PP1=NYC PP2=Chicago and D.C. PP3= PP4=
Set "$@" - parameters in double quotes
PP1=NYC PP2=Chicago and D.C. PP3= PP4=
Set "$* "- parameters space separated
PP1=NYC Chicago and D.C. PP2= PP3= PP4=
$ _
```

This Shell and the Next

What happens to variables when you spawn a Subshell?



Unless you export variables, they will not be passed on.

$ set	to list all variables and values
$ export var	export variable var so that it will
- or -	be inherited by subshells, or
$ typeset -x var	use typeset in the Korn shell
$ declare -x var	use declare in the Bash shell
$ export	to list variables that are exported,
- or -	other variables will be unset in a
$ typeset -x	subshell

Inheritance Example - The export Command

Let's see inheritance in action...

```
$ x=324
$ print "$$: X=$x"
4589: X=324
$ ksh
$ print "$$: X=$x"
4590: X=
$ _ Ctrl-d
$ print "$$: X=$x"
4589: X=324
$ export x
$ ksh
$ print "$$: X=$x"
4591: X=324
$ x=3
$ _ Ctrl-d
$ print "$$: X=$x"
4589: X=324
```

We can set a variable **x** in our current shell

In a subshell, **x** is unset - there is no value to print

Returning to the main shell...

x will have its value restored

If we export **x**, a subshell can inherit the value of **x**

If we change **x** from the subshell the change does not affect the main Shell

Korn Shell Variables

Korn Shell sets certain variables each time they are referenced:

SECONDS	seconds since Shell invocation
RANDOM	random number in the range 0 to 32767
LINENO	current line number within a Shell Script or function
ERRNO	system error number of the last failed system call – a system-dependent value!

Environment Variables

Several variables define the environment of a Shell:

`CDPATH`	a search path for the **`cd`** command
`HOME`	your home directory
`IFS`	input field separators (***space, tab, newline***)
`PATH`	the system command search path
`PS1`	the primary Shell command prompt
`PS2`	a secondary prompt for multi-line entry
`PS3`	prompt for the **`select`** command
`PS4`	debug prompt for **`ksh`** with the **`-x`** option
`PWD`	the current working directory
`OLDPWD`	previous working directory for **`cd -`**

Korn Environment Variables (1 of 2)

Korn Shell specific features require environment variables:

COLUMNS	screen width
LINES	screen length
SHELL	the pathname of the shell
TERM	the terminal type (selects terminfo file)
ENV	program/script to be sourced for each new shell
FCEDIT	an editor for the **fc** command
FPATH	a search path for function definition files
HISTFILE	your history file
HISTSIZE	limit of history commands accessible

Korn Environment Variables (2 of 2)

LC_COLLATE	sorting sequence for pattern ranges
MAIL	the name of your mail file
MAILCHECK	mail check frequency (default 600 seconds)
MAILMSG	the "you have new mail" message
PPID	the parent process ID
REPLY	set by **select** command and the **read** command if no argument is given
EDITOR	the editor for command line editing
VISUAL	a visual editor – overrides **EDITOR**

Korn Shell 93 Variables

- There are several additional variables and variable meanings in ksh93. Here are a few:

`TMOUT`	also used to timeout of select menu
`.sh.version`	identifies version of the shell -- use **`${ }`**

Bash Environment Variables

- Bash variables are the same unless noted here:
- **BASH_ENV** instead of **ENV** program to be sourced for each new interactive shell
- **PS1** has additional features (see below)
- Some additional variables in bash:

BASH_VERSION	version number for the instance of bash
HOSTNAME	name of current host
HOSTTYPE	describes machine bash is running on
SHLVL	shell level - how deeply you are nested

Checkpoint

1. How could we use positional parameter 3 in a shell script?
2. Which variable contains the number of positional parameters?
3. How can we change the value of a variable set in a different process?
4. What is the variable **IFS**?
5. How can we reset **PS1** to show the current directory?
6. By setting a variable, how can we have a command recall facility?

Unit Summary

- Setting variables
- Referencing variables
- Using positional parameters
- Shifting arguments
- Setting positional parameters
- Using shell parameters
- Understanding inheritance
- Shell variables
- Environment variables