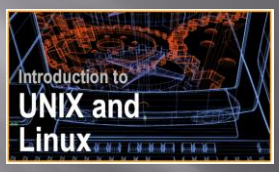


CST 126 – LESSON 9

Specifying Instructions to the Shell
Chapter 8

Overview

- An overview of shell.
- Execution of commands in a shell.
- Shell command-line expansion.
- Customizing the functioning of the shell.
- Employing advanced user features.

An Overview of Shell

- A shell interprets and executes the syntax of the command-lines in a specific way.
- The kernel is the core program of UNIX/Linux, which schedules processes, allocates memory, and handles input/output and other peripherals.
- User cannot directly communicate with the kernel.

An Overview of Shell

- The shell interacts with the kernel to execute a request.
- The shell is the middleman between the user and the kernel.
- A shell translates a user's requests into kernel calls.
- The login shell is started when a user logs in and exits when the user logs out.

An Overview of Shell

- A shell is the interface between the user, utilities, the file system, and the kernel.
- The shell's primary function is to read the command-line, examine its component, and interpret it according to its rules.
- The shell performs the given task and returns the prompt for further requests.

Interacting with the Shell

- Entering a command from the keyboard is the basic way of communicating with the shell.
- For each utility requested by the user, the shell starts a new child process to execute the code of that utility.
- The child process inherits the environment variables like pid, user, etc.
- The "ps" utility can be used for obtaining the process identification numbers.

Communicating with the Shell

- The shell proceeds through a series of specific steps after a user issues commands.
- The complete command-line is first interpreted by the shell.
- The shell interprets the ENTER key as the completion of a command.
- The shell interprets "\" as an instruction not to interpret the special meaning of the single character that immediately follows it.

Communicating with the Shell

- The commands entered at the shell prompt usually include several words or tokens.
- The shell interprets some tokens as utilities and others as filenames.
- The command line interprets the ">", "|", and "<" as special characters that control the input and output of a file.
- The shell uses white space to identify the words or tokens of a command-line.
- The "$" sign is recognized by the shell as the start of a new variable.

Identifying Utilities for Output Redirection

- The shell interprets the first word in the command-line of the shell as a utility.
- The shell interprets the token following the pipe as a utility and the token following the redirection operator as a file.
- The "semicolon" can be used to indicate the end of one pipeline.
- A shell can run one pipeline after another on a single command-line by separating them with semicolons.
- The first token after the semicolon begins a new pipeline, and hence must be a utility.

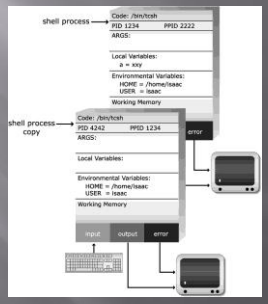
Identifying Utilities in Pipelines

- The logical AND (&&) operator can instruct the pipeline to run the next utility based on the success or failure of the preceding pipeline.
- The command-line is successful only if both are executed.
- A token following the && operator is interpreted as a utility.
- The logical OR (||) operator executes only one of the two utilities in the command-line.
- The shell can easily interpret a variable in all tokens since a $ sign precedes them.
- The "-x" option tells the shell to explain how it interprets the command-line before executing it.

Starting Processes to Run Utilities

- The shell is an active process and runs in the foreground.
- The resources allocated to a running process are called process space or process image.
- The shell makes an exact copy of the process space, including environment variables, when running a utility.
- A new child process space is an exact copy of the shell.
- The child process inherits the input, output, error destination, and variable information from the parent.

Starting Processes to Run Utilities



Redirecting Input and Output

- The input, the output and the error files all are connected to the default output, the monitor.
- An error message is displayed on the screen if a command is not able to execute.
- An error message can be redirected to a file by using the "2>" and a filename to the command-line.

Ex: ls –l practice xxxxA > lsoutput 2> lserror

Redirecting Input and Output

- The bash, ksh, and sh uses ">" or "1>" to redirect output to a file.
- The standard error and output can both be redirected to the same file using the ">&" and specifying the filename in the command line.

Ex: ls –l practice xxxxA > outerr 2>&1

The Exit Code Status After a Utility Execution

- The shell interprets the variable "?" as the exit code of the last process.
- Exit codes other than zero are error codes.
- Every time a process completes its execution and exits, it informs its parent about the status of the exit code.

Using Shell Characters to Expand Filenames

- Some characters are interpreted by the shell as wildcard characters, while others can be used for specifying a range of characters.
- The filename expansion of the filename-matching feature allows the selection of many filenames while entering only one name with special characters embedded.
- The * and ? are interpreted by the shell as special characters.

Using Shell Characters to Expand Filenames

- The asterisk (*) character can be used for matching any number of characters, while the question mark (?) is used only for matching a single character.
- The shell also allows a range of letters or characters to be specified with the help of square brackets.
- The curly brace characters, "{" and "}", are also used by the bash shell and modern ksh shells for matching and creating multiple filenames from one pattern.
- The curly braces match existing filenames if each match is specified in the braces, but does not expand ranges.

Creating and Using Local Variables

- Local and environmental are the two different kinds of variables identified by the shell.
- The "set" or "env" command lists the variables that are set in the shell's memory.
- In a csh or tcsh shell, the set command is used for declaring a variable and assigning a value to it.

Creating and Using Local Variables

- In ksh, bash, or sh shell, a variable is directly defined and assigned a value without the set command.
- The shell interprets the $ character as an instruction to locate in the shell's memory a variable that has the name of the character string that follows the $.
- The variable must be enclosed in single quotes if it includes any spaces.

Passing Environment Variables to Child Processes

- The "unset" command can be used for removing a local variable.
- An environmental variable can be removed with the help of the "unsetenv" command.
- The "export" command is used for making a local variable available to a child process.

Passing Environment Variables to Child Processes

- An environmental variable modified by the child process is not reflected in the parent's environmental variables.
- The shell also allows a variable to be created and exported at the same time. Ex: export newvar=enoughalready
- The variables set in a child process are lost once the child process exits.

Shell Variable Manipulation

Shell	ksh, bash, sh	tcsh, csh
Create local variable	a=xxx	set a=xxx or set a = xxx
Create environmental variable	b=yyy export b or export b=yyy	setenv b yyy
Remove local (C shell family)	-	unset a
Remove environmental (C shell family)	-	unsetenv b
Remove variable regardless of whether local or environmental (sh family)	unset a unset b	-
List ALL variables (sh family)	set	-
List environmental variables	env	env
List local variables (C shell family)	-	set
Make local into environmental	export	-

Using and Modifying the Search Path

- The "path" or "PATH" variable is searched when a user requests for a utility.
- The path is a local variable and is usually assigned a value in the startup script.
- To add a new directory to the path in the Korn shell use the : and append to the end of the existing PATH.

Ex: PATH=$PATH:~/new_directory

Ex: PATH=$PATH:.

Employing Advanced User Features

Completing filenames:

- The variable filec in a tcsh shell, when set in the environment, instructs the shell to search for matching filenames.
- When a shell cannot distinguish between two existing files, it either displays all matching files or simply flashes or produces beeps.
- Filename completion can also be used for files, directories, and executables.

Employing Advanced User Features

Completing filenames (continued):

- The filename-completion variable can be set in the Korn shell by executing either the "set –o vi" or the "set –o vi-tabcomplete" command.
- The "set –o vi" or "set +o posix" commands can be used for turning on the file-completion feature if it is not working.
- Many C shells include filename completion, but use the ESC key to trigger completion of filenames.

Employing Advanced User Features

Evaluating shell variables:

- The bash and ksh shells also provide built-in variables that are useful in interacting with the shell.
- The SECOND shell variable can be used for determining the number of seconds since the shell was started.
- In a bash shell, the PROMPT_COMMAND variable allows a user to execute any command just before it displays the prompt.
- A dot file is a run-control file for a specific utility or shell.

Employing Advanced User Features

Customizing shell startup files:

- The csh shell can be customized with the help of the .cshrc file in the /etc directory since it is always read at startup.
- The bash shell reads the file .bashrc whenever it starts.
- A system setup to start a ksh file reads the .kshrc file at startup.

Employing Advanced User Features

Customizing shell startup files (continued):

- A system setup to start a ksh file reads the .kshrc file at startup.
- The .kshrc file is not read if the ENV environmental variable is not set.
- The ksh shell is programmed to read at startup whatever file is the value of the ENV variable.

Summary

- A variety of command line options exist for interacting with shells.
- A child process started by the shell for each utility execution inherits the input, output, and error destinations, as well as environmental variables.
- Environment and user defined shell variables provide user flexibility for interacting within the unix environment.