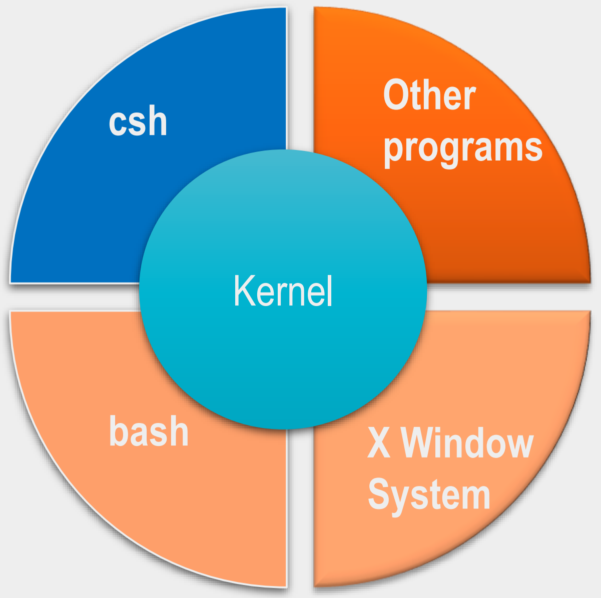
Persistent

UNIX

The Unix Shell

Shell Basics

- A shell is a program that acts as the interface between the user and the system, allowing the user to enter commands for the OS to execute
- It resembles a Windows command prompt, but is much more powerful
- On Unix, it's quite feasible to have multiple shells installed, with different users able to pick the one they prefer
- Most of the shells are derived from the original Bourne shell



Shell Types

- There are different types of shells available, and its likely that most of these will be already present in your Unix system
  - sh (Bourne) - the original shell from early version of Unix
  - csh, tcsh, zsh - the C shell, and its derivatives
  - ksh, pdksh - the Korn shell and its public domain version
  - bash - Bourne Again SHell, is open source and is ported to most of the Unix variants. It is similar to Korn shell

Bash Shell

- bash shell is extremely popular shell. Its advantages are
  - it offers environment variables to configure every aspect of your user experience within the shell
  - it supports command history & command completion
  - it offers built-in arithmetic function
  - wildcard expressions help in finding the exact name of a program or see all the programs with a specific string in the file name
  - command line editing helps you use Emacs and vi commands to make edits at the prompt
  - bash has a long list of built-in commands that make machine administration simple and much easier
  - scripts for bash are compatible with older Bourne shell

Understanding Path

- Every file has a unique location, and this location is called as the path name and specifies the file’s unique place within the entire file system
- For example, the *ls* command usually has the full path name */bin/ls* and means that *ls* command is stored in the */bin* directory, which itself is stored in the root (*/*)
- Typing full path names can be tedious in regular use, especially if you’re working with programs and documents that are stored deep within nested directory
- Another problem is you might want to use a program but you don’t know its location
- The end result is lot of effort is wasted in searching the file system.

PATH Variable

- The solution to these is the *PATH* environment variable, that contains a list of directories where executable files might be located
- For example if the *ar* command is stored in */usr/bin* directory, you can simply type *ar* at the command prompt to invoke the program instead of using the complete path */usr/bin/ar*
- You can view the PATH through the following command - *echo $PATH*
- The value of *PATH* variable is usually set at the system-wide level in a configuration file such as */etc/profile*

PATH Variable

- It is possible to add your own values to the *PATH* variable
- When you login, the shell’s configuration files are executed. You can add any value to the *PATH* variable there
- To append values to *PATH* variable, use the following syntax

  PATH=$PATH:new value
- The changes are apparent immediately after executing the statement above

Configuring Shell

- Following are the main elements of shell configuration
  - run control files
  - environment variables
  - aliases

Run Control Files

- These files are executed as soon as the shell boots up
- The first run control file that the shell checks is a global configuration file and more than one configuration file may be used, depending on the shell
- After the global configuration files are parsed, the shell then parses any existing personal configuration files stored in the user account
- There might be more than one personal configuration file

bash Shell Configuration Files

- */etc/profile* - This is the first file that is read when bash is invoked as a login shell
- *~/.bash_profile* - The commands in this file are read everytime you login into the system. The commands in this file define the basic environment for your login account
- *~/.bash_login & ~/.profile* - These are synonym files for *~/.bash_profile*. Only one of these is read when you log in. If *.bash_profile* does not exist, then *.bash_login* is read, otherwise *.profile* is read
- *~/.bashrc* - This file is read when a new subshell is started (from the command line or through some program)
- *~/.bash_logout* - This file is read every time a login shell exits

Shell Configuration Changes

- Note that whatever you add to your .*bash_profile* won't take effect until the file is re-read by logging out and then logging in again.
- However, you can apply the changes by sourcing the file through the source command

  #source .bash_profile

- *source* executes the commands in the specified file in the current shell
- You can also use the . (dot) command to source a configuration file

Shell Variables

- A shell variable is a name with value associated with it. There are several built-in variables; shell programmers can add their own
- These shell variables are used to configure almost every element of a given shell's behavior
- You can see the variables defined in any shell with the *set, env or printenv* commands
- The value of a particular shell variable can be displayed by using the echo command

  echo $PATH

- This command will display the directories in the *PATH* variable. The *$* sign tells the shell to display the value of the variable instead of the variable itself

Shell Variables

- You can define your own shell variables
- For example, if your project is in */home/PM/work/cl18/lib/source*, and you frequently require this path, you can create a shell variable for it as follows

  PROJDIR=/home/PM/work/cl18/lib/source
- To navigate to this directory from home, type the following - *cd $PROJDIR*
- To remove the value of this variable use the *unset* command as - *unset PROJDIR*
- Notice the absence of *$* on the shell variable
- To export the shell variables to sub-shells, use the *export* command

Common Shell Variables

VARIABLE	Meaning
HOME	contains the absolute pathname of your login directory. The shell itself uses this information to determine the directory to change to when you type cd with no argument
LOGNAME	contains your login name. It is set automatically by the system
PWD	a special variable that gets set automatically to your present working directory
PATH	lists the directories in which the shell searches to find the program to run when you type a command
MAILCHECK	tells the system how frequently, in seconds, to check for new mail
HISTSIZE	tells the shell how many commands to save in your history file
HISTFILE	specifies the location of your history file
SHELL	contains the name of your shell program
MAIL	contains the name of the file in which your newly arriving mail is placed

Prompting Variables

- The bash prompt can be customized through four prompt strings that are stored in variables *PS1, PS2, PS3 & PS4*. The first one is the primary prompt, second is the secondary prompt and the rest are used for debugging
- The primary prompt string can be customized with different commands e.g. to make your prompt display the current working directory, do the following

  PS1=“[\u@\h \w]\$”

PS1 Options

- Other options for PS1 are :

VARIABLE	Meaning
\@	the current time in 12-hour am/pm format
\d	the date in "Weekday Month Date" format (e.g., "Tue May 26")
\!	the history number of this command
\#	the command number of this command
\u	User name
\h	Host name
\w	Current directory with entire path
\W	Current directory

Aliases

- You can also customize your working environment by using alias command. It is a way of substituting one term for another
- For example, instead of using *ls -l* every time, you can do the following:

  alias ll=“ls -l”

  Now *ll* becomes an alias for *ls -l* command
- In most distributions, for root user, the destructive commands like *rm, rmdir, mv,* etc are aliased with switches so that a confirmation is taken from the user before committing the action (this is usually done for the root account)
- To view the current aliases, simply run the alias command without any arguments

Persistent

Command History

- Most of the shells keep a list of all commands that are entered during a session
- history command displays all previously entered commands
- The commands are usually stored in a file in the users home directory (.bash_history in Linux)
- You can know the name of the history file through the HISTFILE variable

Command	Function	Command	Function
history	list commands in history	!!	repeat previous command
history n	list n most recent commands	!n	repeat n command
history -c	deletes all the entries		

Links for objective multiple choice questions.

- http://www.sanfoundry.com/linux-command-mcq-1/
- http://www.sanfoundry.com/linux-command-mcq-2/
- http://www.sanfoundry.com/linux-command-mcq-3/
- http://www.indiabix.com/computer-science/unix/
- http://www.avatto.com/computer-science/test/mcqs/questions-answers/unix/153/1.html
- http://www.gkseries.com/computer-engineering/unix/multiple-choice-questions-and-answers-on-unix-and-shell-programming
- http://www.withoutbook.com/online_test.php?quiz=38&quesNo=10&subject=Top%2010%20UNIX%20Online%20Practice%20Test%20%7C%20Multiple%20Choice

Persistent