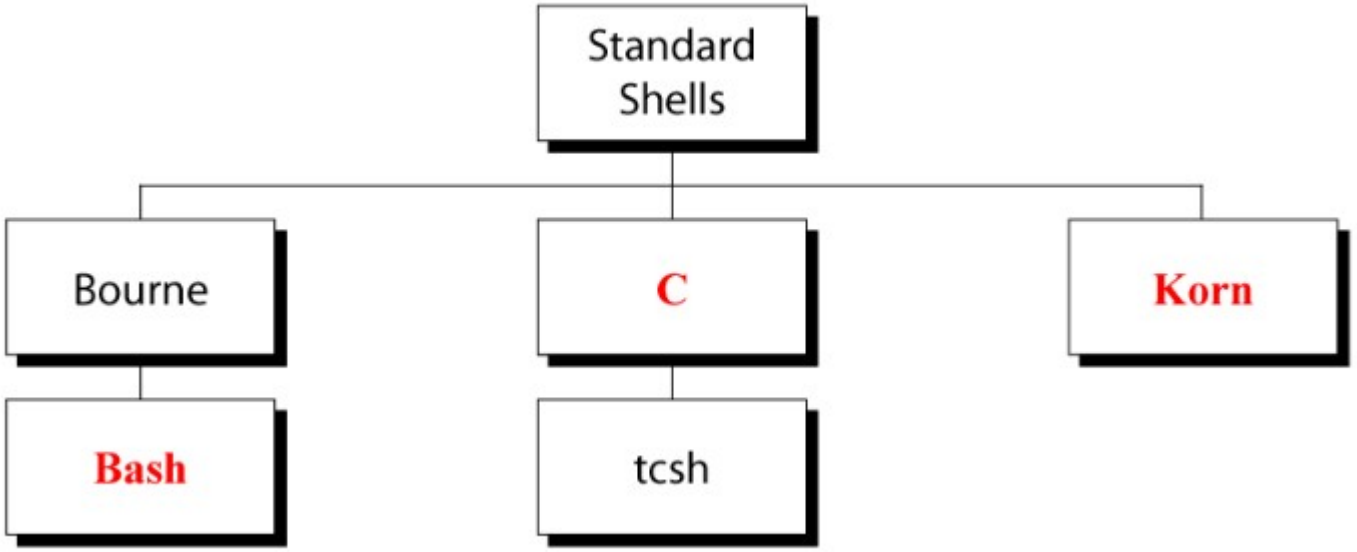
Bash Scripting

Bash

- Bash is a command language interpreter.
- It is widely available on various operating systems and is a default command interpreter on most GNU/Linux systems.
- The name is an acronym for the ‘Bourne-Again SHell’.

Shell

- Shell is a macro processor which allows for an interactive or non-interactive command execution.



Scripting

- Scripting allows for an automatic commands execution that would otherwise be executed interactively one-by-one.
- Shell programming is one of the most powerful features on any UNIX system
- If you cannot find an existing utility to accomplish a task, you can build one using a shell script

Structure of a shell program

- A shell program contains high-level programming language features:
  - Variables for storing data
  - Decision-making control (e.g. if and case statements)
  - Looping abilities (e.g. for and while loops)
  - Function calls for modularity
- A shell program can also contain:
  - UNIX commands
  - Pattern editing utilities (e.g. grep, sed, awk)

Good Practices

- Naming of shell programs and their output
  - Give a meaningful name
  - Program name example: findfile.csh
  - Do not use: script1, script2
  - Do not use UNIX command names
- Repository for shell programs
  - If you develop numerous shell programs, place them in a directory (e.g. bin or shellprogs)
  - Update your path to include the directory name where your shell programs are located

Scripting Steps

- Specify shell to execute program
  - Script must begin with #! (pronounced “shebang”) to identify shell to be executed
  - Examples:
    - #! /bin/sh
    - #! /bin/bash
    - #! /bin/csh
    - #! /usr/bin/tcsh
- Make the shell program executable
  - Use the “chmod” command to make the program/script file executable

Formatting scripts

- Formatting of shell programs
  - Indent areas (3 or 4 spaces) of programs to indicate that commands are part of a group
  - To break up long lines, place a \ at the end of one line and continue the command on the next line
- Comments
  - Start comment lines with a pound sign (#)
  - Include comments to describe sections of your program
  - Help you understand your program when you look at it later

Example script

```
#! /bin/bash
echo "Hello $USER"
echo "This machine is `uname -n`"
echo "The calendar for this month is:"
cal
echo "You are running these processes:"
ps
```

Program output

```
./hello.sh
Hello hari
This machine is hari-HP-348-G5
The calendar for this month is:
   September 2020
Su Mo Tu We Th Fr Sa
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

Current date and time is:
Tuesday 22 September 2020 08:36:46 PM IST
```

Startup and Shutdown scripts

- /etc/profile
- ~/.bash_profile
- ~/.bash_login
- ~/.profile

} executed for login shell

- /etc/bash.bashrc
- ~/.bashrc

} executed for non-login shell

- options:
  - --norc — don't run initialization files
  - -l — run as login shell

- ~/.bash_logout

Predefined Shell Variables

Shell Variable	Description
PWD	The most recent current working directory.
OLDPWD	The previous working directory.
BASH	The full path name used of the bash shell.
RANDOM	Generates a random integer between 0 and 32,767
HOSTNAME	The current hostname of the system.
PATH	A list of directories to search of commands.
HOME	The home directory of the current user.
PS1	The primary prompt (also PS2, PS3, PS4).

User Defined Shell Variables

- Syntax:

  varname=value

  - Example:

    rate=7.65

    echo “Today’s rate is: $rate”

- Use double quotes if the value of a variable contains white spaces
  - Example:

    name=“NIELIT Calicut”

Numeric Variables

- Syntax:
  - let varname=value
- Can be used for simple arithmetic:
  - let count=1
  - let count=$count+10
  - let count+=1

Array Variables

- Syntax:
  - varname=(list of words)
- Accessed via index:
  - ${varname[index]}
  - ${varname[0]} first word in array
  - ${varname[*]} all words in array

Accessing Array values

- `ml=(karthik hari vimala sini ardra)`
- `echo ${ml[*]}`
  - `john hari vimala sini prasoon`
- `echo ${ml[2]}`
  - `vimala`

Environment variables

- Contain information about your login session, stored for the system shell to use when executing commands.
- Environment variable is created by exporting shell variable
- Syntax:

  export varname(s)

  declare –x varname(s)

Commands (for variables)

- To delete both local and environment variables

```
unset varname
```

- To prohibit change

```
readonly varname
```

- list all shell variables (including exported)

```
set
```

Extracting a portion from a variable

- Extract portion of a variable's value via:
  - ${name:offset:length}
  - name – the name of the variable
  - offset – beginning position of the value
  - length – the number of positions of the value
- Example:
  - `SSN="123456789"`
  - `password=${SSN:5:4}`
  - `echo $password`
  - `6789`

Quoting

- Quoting is used to remove the special meaning of certain characters or words to the shell.
- To do this you must use any of the following symbols:
  - Backslash (\)
  - Single quote (‘)
  - Double quote (“)

Command substitution

- Used to substitute the output of a command in place of the command itself
- Two forms of command substitution:
  - $(command)
  - `command`
- Examples:
- `echo "User $(whoami) is on $(hostname)"`

```
user hari is on hari-HP-348-G5
```

- `echo "Today is" `date``

```
Today is Tuesday 22 September 2020
10:14:27 PM IST
```

Questions?