IT 6204
Section 5.0

Automating System Administration

© 2012, University of Colombo School of Computing

5.1 Shell Basics

Shells

- The shell is a UNIX program that interprets the commands you enter from the keyboard
- UNIX provides several shells, including the Bourne shell, the Korn shell, and the C shell
- Steve Bourne at AT&T Bell Laboratories developed the Bourne shell as the first UNIX command processor
- The Korn shell includes many extensions, such as a history feature that lets you use a keyboard shortcut to retrieve commands you previously entered
- The C shell is designed for C programmers' use
- Linux uses the freeware Bash shell as its default command interpreter (compatible with Bourne shell, created & distributed by the GNU project)
- You can choose the one that best suites your way of working …..

Choosing Your Shell

- You choose a shell when the system admin sets up your user account
  - Bourne shell – ***sh***
  - Korn shell – ***ksh***
  - C shell – ***csh***
  - Bash – ***bash***
  - Enhanced C shell (a freeware shell derived from the C shell) – ***tcsh***
  - Z shell (a freeware shell derived from the Korn shell) – ***zsh***
- After you choose your shell, the system administrator stores your choice in your account record, and it becomes your assigned shell
- UNIX uses this shell any time you log on (try %***echo $SHELL***)

Choosing Your Shell

- After you choose your shell, the system administrator stores your choice in your account record, and it becomes your assigned shell
- UNIX uses this shell any time you log on (try %***echo $SHELL***)
- However, you can switch from one shell to another by typing the shell's name (such as tcsh, bash, or zsh) on your command line

  (try %***chsh***)

  Example of /etc/passwd file:

```
saman:xxxxx:500:500:Saman Silva:/home/saman:/bin/tcsh
root:xxxxxxxx:0:0:root:/root:/bin/bash
```

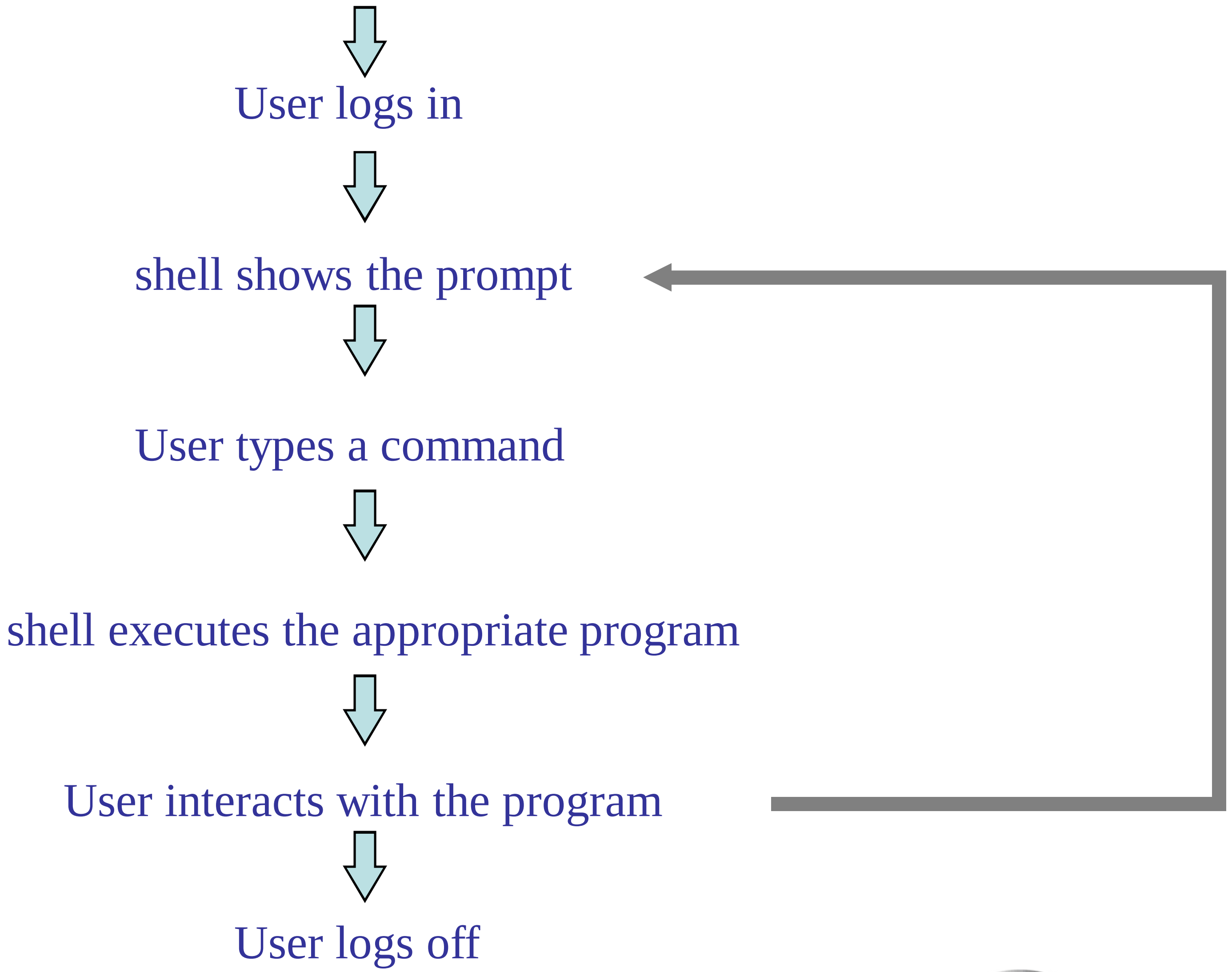
Command-line Editing

- Shells support certain keystrokes for performing command-line editing
- For example, Bash supports the left and right arrow keys, which move the cursor on the command line
- Not all shells support command-line editing in the same manner

Multiple Command Entry

- You may type more than one command on the command line by separating each command with a semicolon(**;**)
- When you press Enter, UNIX executes the commands in the order you entered them
- You can use the ***clear*** command to clear your screen; it has no options or arguments
- You can access the command history with the up and down arrow keys with most shells

User Interaction with the Shell


User logs in
shell shows the prompt
User types a command
shell executes the appropriate program
User interacts with the program
User logs off

5.2 Bash Scripting

Shell Scripts

- What are they for?
  - To automate certain common activities an user performs routinely.
  - They serve the same purpose as batch files in DOS/Windows.
  - Example:
    - ✓rename 1000 files from upper case to lowercase

What are Shell Scripts

- Just text/ASCII files with:
  - a set of standard UNIX/Linux commands (`ls`, `mv`, `cp`, `less`, `cat`, etc.) along with
    - ✓flow of control
      - some conditional logic and branching (`if-then`),
      - loop structures (`foreach`, `for`, `while`), and
    - ✓I/O facilities (`echo`, `print`, `set`, ...).
  - They allow use of variables.
  - They are interpreted by a shell directly.
  - Some of them (`csh`, `tcsh`) share some of C syntax.
  - DOS/Win equivalent - batch files (.bat)

Why not use C/C++ for that?

- C/C++ programming requires compilation and linkage, maybe libraries, which may not be available (production servers).
- For the typical tasks much faster in development, debugging, and maintenance (because they are interpreted and do not require compilation).

Shell Script Invocation

- Specify the shell directly:
  - `% tcsh myshellscript`
  - `% tcsh -v myshellscript`
    (-v = verbose, useful for debugging)
- Make the shell an executable first and then run is a command (set up an execution permission):
  - `% chmod u+x myshellscript`
- Then either this:
  - `% myshellscript`
    (if the path variable has '.' in it; **security issue**!)
- Or:
  - `% ./myshellscript`
    (should always work)

Shell Script Invocation (2)

- If you get an error:
  "myshellscrip: command not found"
  - The probably "." is not in your path or there's no execution bit set.
- When writing scripts, choose unique names, that preferably do not match system commands.
  - Bad name would be test for example, since there are many shells with this internal command.
- To disambiguate, always precede the shell with "./" or absolute path in case you have to name your thing not very creatively.

Start Writing a Shell Script

- The very first line, often called 'shebang' (#!) should precede any other line, to assure that the right shell is invoked.

```
#!/bin/tcsh                          #!/bin/bash
# This is for tcsh                   # For Bourne-Again Shell

#!/bin/sh
# This is for Bourne Shell
```

- Comments start with '#', with the exception of #!, $#, which are a special character sequences.
- Everything on a line after # is ignored if # is not a part of a quoted string or a special character sequence.

Bourne Shell Script Constructs Reference

- System/Internal Variables
- Control Flow (if, for, case)

Internal Variables

`$#`	Will tell you # of command line arguments supplied
`$0`	Ourselves (i.e. name of the shell script executed with path)
`$1`	First argument to the script
`$2`	Second argument, and so on…
`$?`	Exit status of the last command
`$$`	Our PID
`$!`	PID of the last background process
`$-`	Current shell status

© 2012, University of Colombo School of Computing

Internal Variables (2)

- Use `shift` command to shift the arguments one left:
  - Assume intput:
    - `./shift.sh 1 2 foo bar`
      - $0 = <directory-of>/shift.sh
      - **$1 = 1**
      - **$2 = 2**
      - **$3 = foo**
      - **$4 = bar**
    - `shift`:
      - $0 = <directory-of>/shift.sh
      - **$1 = 2**
      - **$2 = foo**
      - **$3 = bar**

© 2012, University of Colombo School of Computing

Environment

- These (and very many others) are available to your shell:
  - $PATH - set of directories to look for commands
  - $HOME - home directory
  - $MAIL
  - $PWD – personal working directory
  - $PS1 – primary prompt
  - $PS2 – input prompt
  - $IFS - what to treat as blanks

Control Flow: if

- General Syntax:

```
if [ <expression> ]; then
    <statements>
elif
    <statements>
else
    <statements>
fi
```

- <expression> can either be a logical expression or a command and usually a combo of both.

if

- Some Logical “Operators”:

-eq	--- Equal
-ne	--- Not equal
-lt	--- Less Than
-gt	--- Greater Than
-o	--- OR
-a	--- AND

- File or directory?

-f	--- file
-d	--- directory

for

- Syntax:

```
for variable in <list of values/words>[;]
do
    command1
    command2
    …
done
```

- List can also be a result of a command.

for

```
for file in *.txt;
do
    echo “File $file:”;
    echo "===START===“;
    cat $file;
    echo "===END===“;
done
```

while

- Syntax

```
while <expression>
do
    command1
    command2
    …
done
```

until

- Syntax

```
until <expression>
do
    command1
    command2
    …
done
```

Exercise

- All the *.conf files in the current directory will be copied with that file name.org

```
for file in *.conf;
        do cp $file $file.org;
done
```

More Examples

```
#!/bin/bash
# This is my script to make a backup of a # .conf file
d=`date +%d%m%y`;
cp -pv $1 $1.$d.org;
echo "Copying Finished";
vi $1
```

```
for i in *.txt;
do
        echo "File name: $i";
        echo "=====START=======";
        cat $i;
        echo "=====END=======";
done;
```

More Examples

```
#!/bin/bash
if [ "${1##*.}" = "tar" ]
then
        echo This appears to be a tarball.
else
        echo At first glance, this does not appear to
be a tarball.
fi
if [ "$2" = "help" ]
then
echo " ===============HELP ============";
fi
```

5.3 Periodic Processes

Cron

- Cron gives the ability to run commands periodically on the system.
- Cron jobs can be set up by the administrator or by users.
- The Cron Table is stored in /etc/crontab
- Users can edit cron jobs with: ***crontab –e***
- List with: ***crontab –l***

© 2012, University of Colombo School of Computing

Cron cont…

- Each entry has 6 fields:
  - Minutes ➔ 00-59
  - Hours ➔ 0-23 (Mid-night is 0)
  - Day of the month ➔ 1-31
  - Month of the year ➔ 1-12
  - Day of the week ➔ 0-6 (Sunday is 0)
  - Job to be executed
- * all legal values
- “,” multiple entries are separated by comma
- # implies comments

© 2012, University of Colombo School of Computing

Cron Example

- Field Rules:
  - single number ie. 1
  - range ie. 1-4
  - ranges w/step ie. 1-100/5
  - list ie. 1,3,5,7
  - wildcard ie. *
- **0   17   *   *   1,2,3,4,5   /usr/backup**
- Run /usr/backup at 5pm Monday-Friday every week, in every month in the year
- Cron daemon starts by ***rc*** files. Once started never terminates. It checks the crontab file every minute (for any changes)
- Cron allow us to schedule programs for periodic execution. However, cron is not a general facility for scheduling program execution off-hours

  – use the ***at*** command

© 2012, University of Colombo School of Computing

More Cron Examples

- **0 6 */2 * * mailq –v | mail –s "Stuck Mails …" nimal**
- Uses ***mailq*** every two days to test whether there is any mail stuck in the mail queue and sends the mail to administrator (nimal@...)
- **0 2 1 */2 * mt –f /dev/rft0 rewind; tar cf /dev/rft0 /etc**
- Runs at 2:00AM on the first day of the month in every other month to backup the /etc to the tape (make sure the tape is in the drive!!)
- The same can be written as:
  0 2 1 jan,mar,may,jul,sep,nov * mt –f /dev/rft0 rewind; tar cf /dev/rft0 /etc

0 0 * * * *cmd*	- Every night at 00:00 hours
5 4 * * 6 *cmd*	- 4:05am on Saturdays
0 1 */5 * * *cmd*	- At 1:00am on every 5th day – 1st, 6th, 11th, so on
0 1 1-15 * * *cmd*	- At 1:00am on every day from 1st to 15th, inclusive
*** * * 12 4,5 *cmd***	- Every December Thu & Fri

© 2012, University of Colombo School of Computing

End of Section 5.0

© 2012, University of Colombo School of Computing