SHELL SCRIPTING, CROND, ATD

SHELL SCRIPTING

- Shell scripting involves placing a series of shell commands in a file for later re-use.
  - Simple shell scripts simply run command after command, as if the user typed them in at the command line
  - More complex shell scripts actually make decisions about what commands need to be run, and might even repeat certain sequences to accomplish some task
- Scripts start executing at the top and stop when there are no more commands to execute or when `exit` is called.

EXAMPLE SHELL SCRIPT

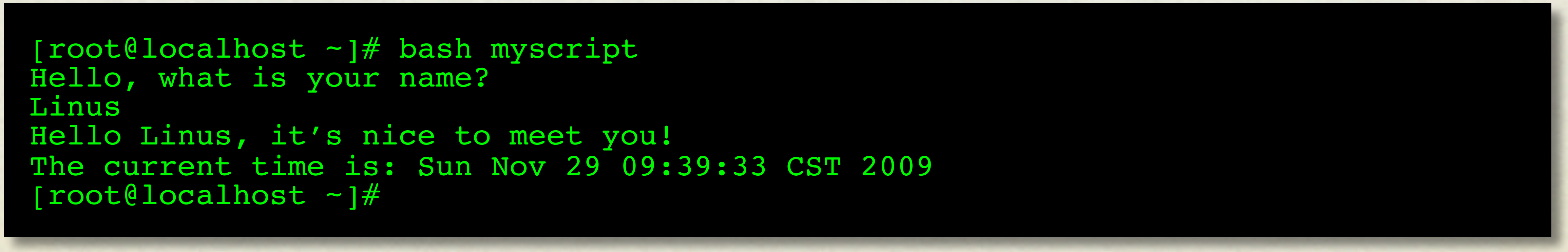
- Here is an example of a very simple shell script:

```
echo “Hello, what is your name?”
read NAME
echo “Hello $NAME, it’s nice to meet you!”
echo -n “The current time is: “
date
```

- Using the `echo` command, this script asks a question.
- The `read` command accepts input from the user and stores it in the environment variable `NAME`
- The script finishes up with a couple more echo statements, greeting the user and announcing today’s date

SHELL SCRIPTING

- If we put the example in a file called `myscript`, we can execute the script as:
  - `bash myscript`
- `bash` will open `myscript` and execute each line as if the user had typed it in manually.



```
[root@localhost ~]# bash myscript
Hello, what is your name?
Linus
Hello Linus, it's nice to meet you!
The current time is: Sun Nov 29 09:39:33 CST 2009
[root@localhost ~]#
```

INTERPRETERS

- In the previous example, we put five commands in a regular file and fed the filename to `bash` on the command line, which in turn executed the commands.

- Running in this way, `bash` operated as an interpreter. Reading each line of the file, `bash` would interpret the words and perform some action.

- There are many interpreted languages available for scripting, including all shells, `python`, `ruby`, `perl`, etc.

EXECUTING SCRIPTS

- To run a script, feed the file to the appropriate interpreter:
  - `bash mybashscript`
  - `perl myperlscript`
- This works fine, but sometimes it's more user-friendly to allow the script to be run directly, removing the need for an external call to the interpreter...
  - `./mybashscript`
  - `myperlscript`

SHEBANG

- This is accomplished with the shebang (#!). Also known as a hash bang, pound bang or hashpling.
- When the kernel is asked to execute a file, it must either be machine code, or a file that starts with the shebang sequence. If the first two characters of the file are a hash mark and an exclamation mark, the rest of the line is expected to be an absolute pathname for an interpreter, which will then be invoked to “run” the file as a script.

SHEBANG

- So, add an appropriate shebang to the example:

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
echo “Hello $NAME, it’s nice to meet you!”
echo -n “The current time is: “
date
```

- Then add execute permissions and the script can be run directly:

```
[root@localhost ~]# chmod 755 myscript
[root@localhost ~]# ./myscript
Hello, what is your name?
Linus
Hello Linus, it’s nice to meet you!
The current time is: Sun Nov 29 09:39:33 CST 2009
[root@localhost ~]#
```

DECISIONS

- More advanced problems require the script to make decisions. There are two basic ways to make decisions with shell scripts:
  - `if` statements
  - `case` statements

TEST COMMAND

- Before we continue talking about decisions, we need to talk about the `test` command. This command actually performs the comparisons necessary to ask a question, such as:
  - `“string1” = “string2”` *Returns true if string1 is identical to string2*
  - `VAR -le 45` *Returns true if VAR is numerically less than or equal to 45*
- See the man page for `test` for additional details

IF STATEMENTS

- Basic syntax:

```
if list;

  then list;

  [ elif list; then list; ]

  ...

  [ else list; ]

fi
```

IF EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
if [ “$NAME” = “Linus” ]
then
  echo “Greetings, Creator!”
elif [ “$NAME” = “Bill” ]
then
  echo “Take your M$ elsewhere!”
  exit
else
  echo “Hello $NAME, it’s nice to meet you!”
fi
echo -n “The current time is: “
date
```

- This script will now base it’s response based on what name the user provides

CASE STATEMENTS

- Basic syntax:

```
case word in
  pattern [| pattern] ) list;;
  ...
esac
```

CASE EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
case $NAME in
   “Linus” )
     echo “Greetings, Creator!”
     ;;
   “Bill” )
     echo “Take your M$ elsewhere!”
     exit
     ;;
   * )
     echo “Hello $NAME, it’s nice to meet you!”
esac
echo -n “The current time is: “
date
```

- This script also bases it’s response based on what name the user provides, but does so using a case statement instead of a large if statement

LOOPING

- Sometimes a certain sequence of commands need to be run repeatedly, either for a set number of times or while some condition is true. This is accomplished with:
    - `while` loops
    - `for` loops

WHILE LOOPS

- Basic syntax:

```
while list;
  do list;
done
```

WHILE EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
while [ “$NAME” != “Linus” ]
do
  echo “I don’t know that person, what is your name?”
  read NAME
done
echo “Greetings, Creator!”
echo -n “The current time is: “
date
```

- This script will loop until the name typed is “Linus”

FOR LOOPS

- Basic syntax:

```
for (( expr1 ; expr2 ; expr3 ))
  do list;
done
```

FOR EXAMPLE

```
#!/bin/bash
echo “Hello, what is your name?”
read NAME
for (( I=0 ; I<3 ; I++ ))
do
  echo “Hello $NAME!!”
done
echo -n “The current time is: “
date
```

- This excitable script repeats your name 3 times before giving you the date and time

SCRIPTING

- There is of course quite a bit more to shell scripting than can be covered in this course. There are a few more structures you can use for looping, and dozens of special metacharacters for achieving all kinds of results.

- With this introduction, though, you should be able to read through light shell scripts and have a handle on what’s going on, as well as be able to write simple ones on your own.

EXERCISES

- Write a simple shell script that prints out the message “Hello world.” Make the script executable and verify it works correctly by running it as “./myscript”
- Browse through the man page on ‘bash’, focusing in a bit on the various scripting elements.

CROND

Scheduled fun

OVERVIEW

- `crond` is the cron daemon. Cron provides for the ability to execute commands on a regular basis.
- Generally used to run hourly, daily and weekly type system maintenance scripts.
- Also useful to run reports, cleanup jobs and much, much more.

USING CRON

- Cron is controlled through crontab files.
    - There are system-wide crons, accessible under /etc/cron.*
    - Every user has their own crontab, accessible through the crontab command

SYSTEM CRONS

- /etc/crontab defines the system cron jobs.
  - Many distributions use the run-parts script to execute all scripts found in /etc/cron.hourly, /etc/cron.daily, etc on the appropriate schedule.
  - /etc/crontab defines the times for each schedule: hourly, daily, weekly, monthly

CRONTAB

- `crontab`: View, edit or remove crontabs
  - The `-l` option prints the crontab. The `-e` option opens the crontab for editing. The `-r` option removes the crontab.
  - Root can work with the crontab for any user by specifying the username on the command line:
    - `crontab -e -u bob`

CRONTAB SYNTAX

- There are two main components to a crontab entry:
  - The timespec specifies when the command should be run
  - The command is what gets executed every time the timespec is matched

CRONTAB TIMESPECS

- The timespec is broken down into 5 fields, separated by spaces:
  - `minute hour day-of-month month day-of-week`
- Each field can contain a number, a range of numbers, a comma-separated list of numbers, an asterisk or a number slash division rate
- Mostly self-explanatory - some examples will help...

TIMESPEC EXAMPLES

- `0 23 * * *` *11pm every day*
- `30 * * * 1-5` *30 minutes after every hour, M-F*
- `0 7 1 * *` *7am, first of every month*
- `* * * * *` *Every single minute*
- `0,10,20,30,40,50 * * * *` *Every 10 minutes*
- `*/5 8-17 * * 1-5` *Every 5 minutes, 8am-5pm, M-F*

EXAMPLE CRONTAB

```
01 4 * * * /usr/local/bin/restart-webserver
00 8 1 * * /usr/bin/mail-report boss@mycompany.com
*/5 * * * * /monitor/bin/check-site -e admin@mycompany.com -o /var/log/check.log
```

- There are various additional options and features available to the cron system. Check the man pages for reference:
    - `cron`, `crontab` (sections 1 and 5)

ATD

ATD OVERVIEW

- `atd` is a simple daemon that executes one-off jobs at a certain time.
- To create an at job:
  - `at <time>`
  - Then you enter all of the commands you want run at the given time, and finish by typing ctrl-d

ATD

- atd is not commonly used these days, but if it's there is can be useful in some situations..
  - If editing the firewall on a machine over the network, it's sometimes nice to put a simple "reset" so if you lock yourself out, you'll be able to get back in the machine:

```
[root@localhost ~]# at now + 10 minutes
at> iptables-save > /iptables.backup
at> iptables -F
at> <EOT>
job 1 at 2009-11-30 10:44 a root
[root@localhost ~]#
```

ATD

- Some additional commands to use with the at system:
    - `atq`: Displays list of at jobs
    - `atrm`: Removes given at job from queue

slideshow.end();