Shell Scripts I

1. 05a - Shell Scripts I

1.1 Shell Scripts

1.2 Introduction to Shell Scripts

Introduction to Shell Scripts

- Shell scripts can do what can be done on command line
- Shell scripts simplify recurring tasks
  - if you cannot find an existing utility to accomplish a task, you can build one using a shell script
- Much of UNIX administration and house keeping is done via shell scripts

1.3 Shell Script features

Shell Script features

- Variables for storing data
- Decision-making control (e.g. if and case statements)
- Looping abilities (e.g. for and while loops)
- Functions for modularity
- Any UNIX command:
  - file manipulation: cat, cp, mv, ls, wc, tr, …
  - utilities: grep, sed, awk, …
- comments: lines starting with "#"

1.4 Shell Script: the basics

Shell Script: the basics

- 1. line for shell script: (shebang line)

```
#! /bin/bash
```

or: `#! /bin/sh`

- to run:

```
% bash script
```

- or:
  - make executable: `% chmod +x script`
  - invoke via: `% ./script`

1.5 bash Shell Programming Features

bash Shell Programming Features

- Variables
  - string, number, array
- Input/output
  - echo, printf
  - command line arguments, read from user
- Decision
  - conditional execution, if-then-else, case
- Repetition
  - while, until, for
- Functions

1.6 User-defined shell variables

User-defined shell variables

Syntax:

```
varname=value
```

no spaces

Example:

```
rate=moderate
echo "Rate today is: $rate"
```

- use double quotes if value of variable contains white spaces

Example:

```
name="Thomas William Flowers"
```

1.7 Output via echo command

Output via echo command

- Simplest form of writing to standard output

Syntax: echo [-ne] argument[s]

-n suppresses trailing newline
-e enables escape sequences:
- \t horizontal tab
- \b backspace
- \a alert
- \n newline

1.8 Examples: shell scripts with output

Examples: shell scripts with output

```
#! /bin/bash
echo "You are running these processes:"
ps

#! /bin/bash
echo -ne "Dear $USER:\nWhat's up this month:"
cal
```

1.9 Command line arguments

Command line arguments

- Use arguments to modify script behavior
- command line arguments become positional parameters to shell script
- positional parameters are numbered variables:
  $1, $2, $3 …

1.10 Command line arguments

Command line arguments

	Meaning
$1	first parameter
$2	second parameter
${10}	10th parameter
	{ } prevents "$1" misunderstanding
$0	name of the script
$*	all positional parameters
$#	the number of arguments

1.11 Example: Command Line Arguments

Example: Command Line Arguments

```
#! /bin/bash
# Usage: greetings name1 name2

echo $0 to you $1 $2
echo Today is `date`
echo Good Bye $1
```

1.12 Example: Command Line Arguments

Example: Command Line Arguments

- make sure to protect complete argument

```
#! /bin/bash
# counts lines in directory listing
ls -l "$1" | wc -l
```

1.13 Arithmetic expressions

Arithmetic expressions

Syntax:

```
$((expression))
```

- can be used for simple arithmetic:

```
% count=1
% count=$((count+20))
% echo $count
```

1.14 Array variables

Array variables

Syntax:

```
varname=(list of words)
```

- accessed via index:

```
${varname[index]}
${varname[0]}        first word in array
${varname[*]}        all words in array
```

1.15 Using array variables

Using array variables

Examples:

```
% ml=(mary ann bruce linda dara)
% echo $ml
mary
% echo ${ml[*]}
mary ann bruce linda dara
% echo ${ml[2]}
bruce
% ml[2]=john
% echo ${ml[*]}
mary ann john linda dara
```

1.16 Output: printf command

Output: printf command

Syntax: printf format [arguments]

- writes formatted arguments to standard output under the control of "format"
- format string may contain:
  - plain characters: printed to output
  - escape characters: e.g. \t, \n, \a …
  - format specifiers: prints next successive argument

1.17 printf format specifiers

printf format specifiers

%d number

also: %10d 10 characters wide

%-10d left justified

%s string

also: %20s 20 characters wide

%-20s left justified

1.18 Examples: printf

Examples: printf

% printf “random number”

% printf “random number\n”

% printf “random number: %d” $RANDOM

% printf “random number: %10d\n” $RANDOM

% printf “%d for %s\n” $RANDOM $USER

1.19 User input: read command

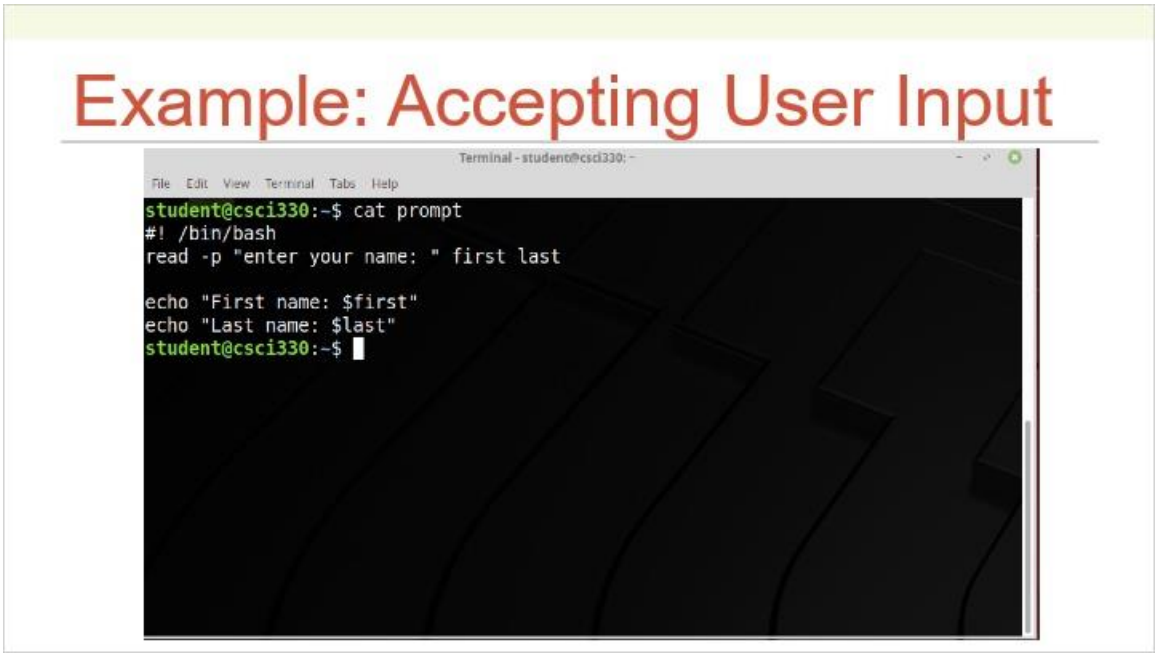
User input: read command

Syntax:

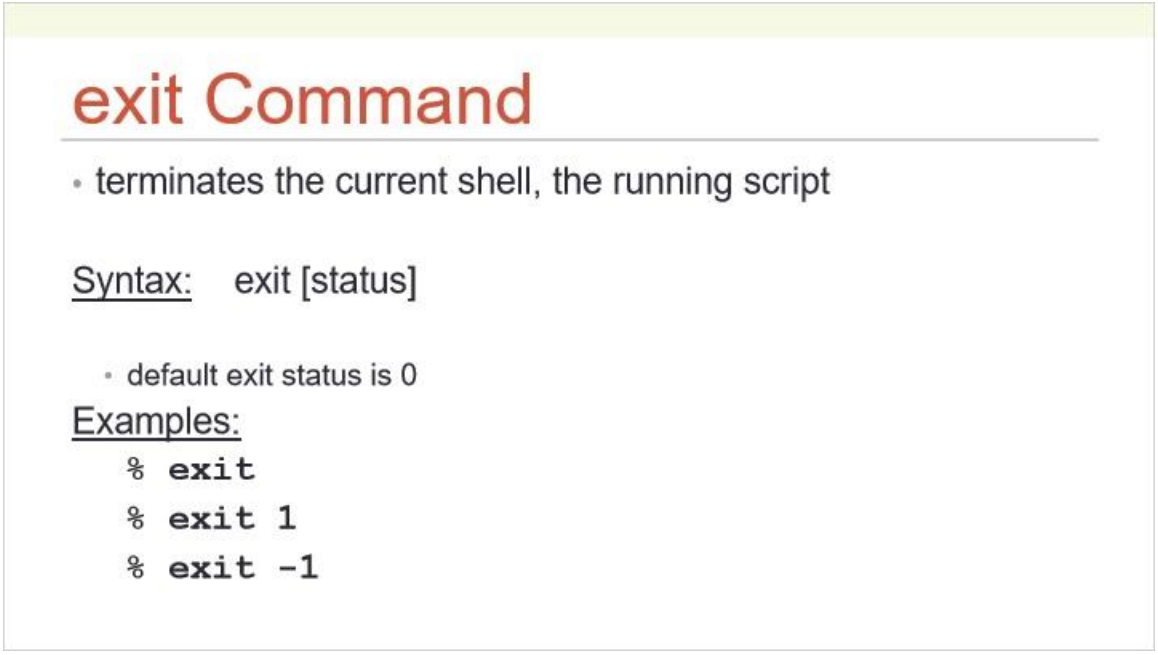
```
read [-p "prompt"] varname [more vars]
```

- words entered by user are assigned to `varname` and “`more vars`”
- last variable gets rest of input line

1.20 Example: Accepting User Input



1.21 exit Command



1.22 Exit Status

Exit Status

- also called: return status
- predefined variable "?" holds exit status of last command
- "0" indicates success, all else is failure

Examples:

```
% ls > /tmp/out
% echo $?
% grep -q "root" /var/log/auth.log
% echo $?
```

1.23 Conditional Execution

Conditional Execution

- operators `||` and `&&` allow conditional execution

Syntax:

`cmd1 && cmd2`	cmd2 executed if cmd1 succeeds
`cmd1 || cmd2`	cmd2 executed if cmd1 fails

- performs boolean "or" "and" on exit status

1.24 Conditional Execution: Examples

Conditional Execution: Examples

```
% grep $USER /etc/passwd  && echo "$USER found"

% grep student /etc/group || echo "no student group"
```

1.25 test command

test command

Syntax:

```
test expression
[ expression ]
```

- evaluates 'expression' and returns true or false

Example:

```
if test $name = "Joe"
then
  echo "Hello Joe"
fi
```

```
if [ $name = "Joe" ]
then
  echo "Hello Joe"
fi
```

1.26 if statements

if statements

```
if [ condition ]; then
 statement
fi

if [ condition ]; then
    statements-1
else
  statements-2
fi
```

```
if [ condition ]; then
      statements
elif [ condition ]; then
    statements
else
    statements
fi
```

1.27 test Relational Operators

test Relational Operators

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eq	=
Not equal	-ne	!=
String length is zero		-z str
String length is non-zero		-n str
file1 is newer than file2		file1 -nt file2
file1 is older than file2		file1 -ot file2

1.28 Compound logical expressions

Compound logical expressions

! expression

true if expression is false

expression -a expression

true if both expressions are true

expression -o expression

true if one of the expressions is true

also (via conditional execution):

&& and

| | or

1.29 Example: compound logical expressions

Example: compound logical expressions

```
• if [ ! "$Years" -lt 20 ]; then
    echo "You can retire now."
  fi
• if [ "$Status" = "H" ] && [ "$Shift" = 3 ]; then
     echo "shift $Shift gets \$$Bonus bonus"
  fi
• if [ "$Calls" -gt 150 ] || [ "$Closed" -gt 50 ]; then
     echo "You are entitled to a bonus"
  fi
```

1.30 File Testing operators

File Testing operators

	Meaning
-d file	true if 'file' is a directory
-f file	true if 'file' is a regular file
-r file	true if 'file' is readable
-w file	true if 'file' is writable
-x file	true if 'file' is executable
-s file	true if length of 'file' is nonzero

1.31 Example: File Testing

Example: File Testing

Terminal - student@csci330: ~

File Edit View Terminal Tabs Help

```
student@csci330:~$ cat check
#!/bin/bash
read -p "Enter a filename: " filename
if [ ! -r "$filename" ]; then
   echo "File is not read-able"
   exit 1
fi
student@csci330:~$
```

1.32 Summary

Summary

- Shell scripts can do what can be done on command line
- Shell scripts simplify recurring tasks