Shell scripting

Unit objectives

After completing this unit, you should be able to:

- Invoke shell scripts in three separate ways and explain the difference
- Pass positional parameters to shell scripts and use them within scripts
- Implement interactive shell scripts
- Use conditional execution and loops
- Perform simple arithmetic

What is a shell script?

- A *shell script* is a collection of commands stored in a text file.

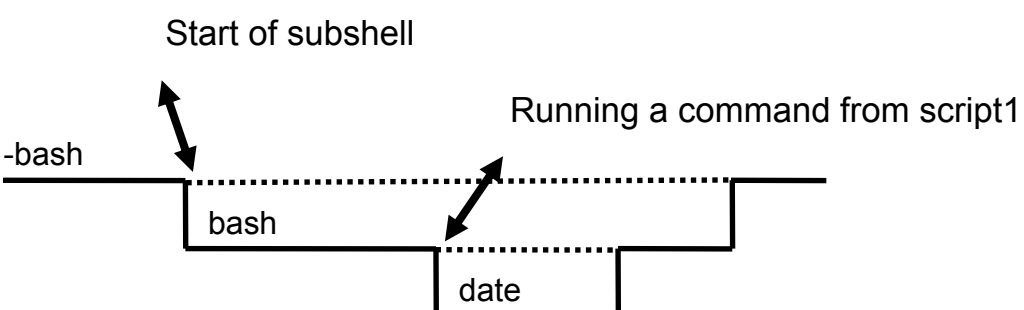
```
$ pwd
$ date
$ ls -l

$ cat script1
pwd
date
ls -l
$
```

Invoking shell scripts (1 of 3)

- The script does not have to be marked executable, but it must be readable.
- bash invokes a script in a child shell.

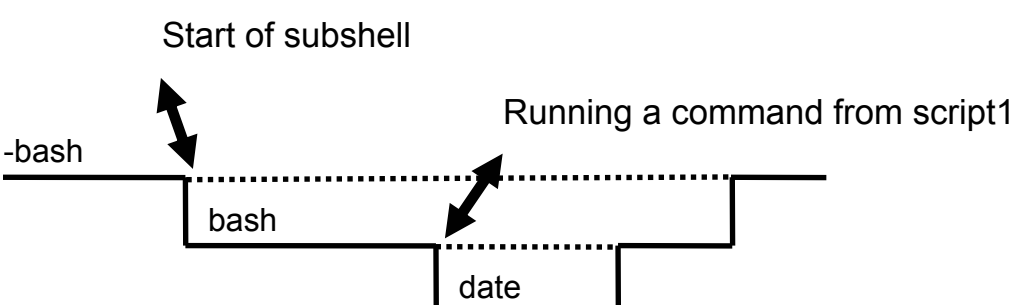
```
$ cat script1
date
$ bash script1
```



Invoking shell scripts (2 of 3)

- Use the **chmod** command to make the script executable.
- Then run the script as if it were a command.
- The script is run in a child shell.

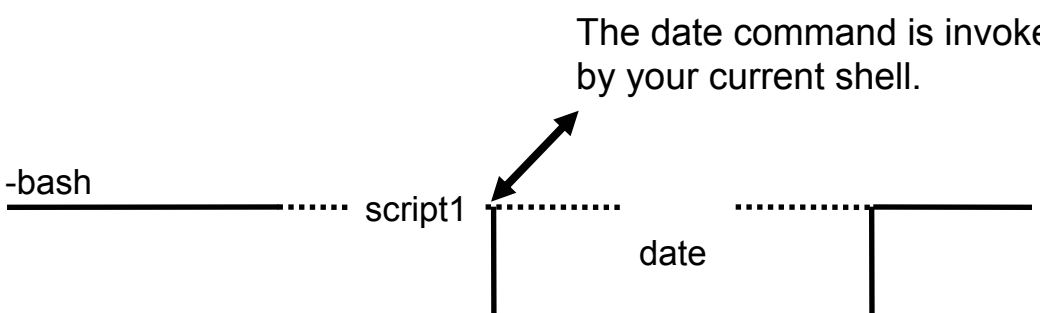
```
$ chmod 755 ./script1
$ ./script1
```



Invoking shell scripts (3 of 3)

- Use the **.** (dot) or **source** command to execute the script in your current shell environment.
- Scripts executed with the dot command can change your current environment.

```
$ . script1
$ source script1
```



Invoking shell scripts in another shell

- To make sure the shell script always runs in the shell it was intended for (sh, bash, csh), use the following on the first line of your script:

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

- The script will now always runs in bash even if the user's default shell is something else.

Typical shell script contents

- Handling of shell script arguments
- Complex redirection
- Conditional execution
- Repeated execution
- User interfacing
- Arithmetic

Shell script arguments

- Parameters can be passed to shell scripts as arguments on the command line.
- These arguments are stored in special shell variables.
  - $1, $2, $3 ... refers to each of the arguments
  - $@ is "$1" "$2" "$3"
  - $* is "$1 $2 $3"
  - $# is the number of parameters

```
$ cat ascript
#!/bin/bash
echo First parameter: $1
echo Second parameter: $2
echo Number of parameters: $#
$ ascript ant bee
First parameter: ant
Second parameter: bee
Number of parameters: 2
```

Complex redirection

- To redirect fixed text into a command, use << END.

```
$ cat << END > cities
Atlanta
Chicago
END
```

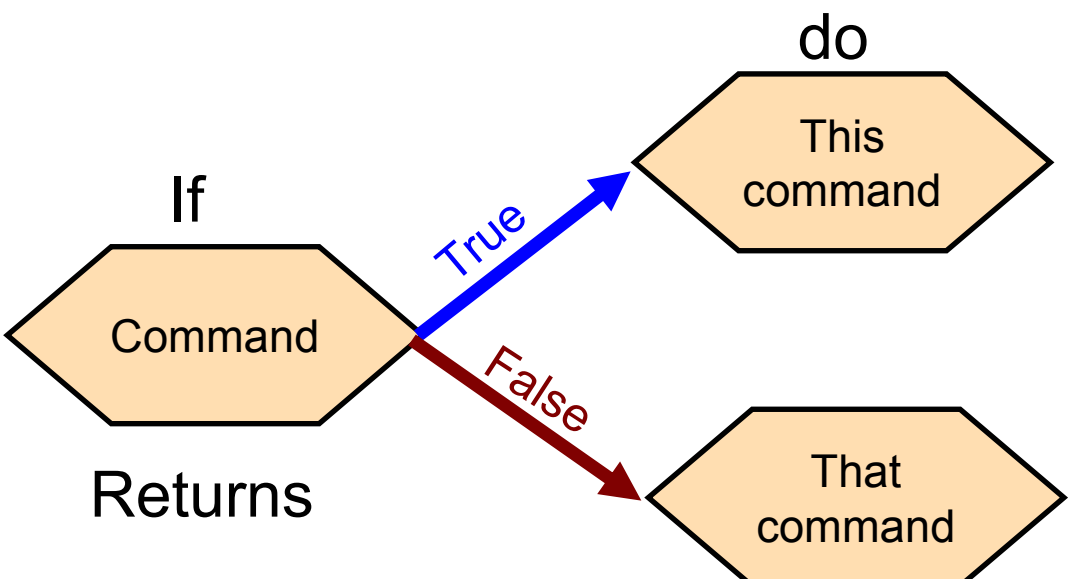
- To avoid large argument lists, use xargs.

```
$ rm *.txt
-bash: /bin/rm: Argument list too long
$ find . -name "*.txt" | xargs rm
$
```

- Avoids large argument lists by running the command multiple times, if necessary.
- -0 (-zero) option to find and xargs needed when file names contain blanks or new lines.

Conditional execution

- The return code from a command or group of commands can be used to determine whether to start the next command.



The test command (1 of 2)

- The **test** command allows you to test for a given condition.
- Syntax:

```
test expression  ...or: [ expression ]
```

```
$ test -f myfile.txt
$ echo $?
0
```

- Expressions to test file status:
  - `-f <file>` file is an ordinary file
  - `-d <file>` file is a directory
  - `-r <file>` file is readable
  - `-w <file>` file is writable
  - `-x <file>` file is executable
  - `-s <file>` file has non-zero length

The test command (2 of 2)

- String tests

```
-n <string>            string is not empty
-z <string>            string is empty
<string> == <string>   strings are equal
<string> != <string>   strings are not equal
```

- Arithmetic tests

```
<value> -eq <value>    equals
<value> -ne <value>    not equal
<value> -lt <value>    less than
<value> -le <value>    less than or equal
<value> -gt <value>    greater than
<value> -ge <value>    greater than or equal
```

The && and || commands

- **&&** and **II** (two vertical bars) can be used to conditionally execute a single command.

  `command1 && command2`

  if (command1 successful) then do (command2)

  `command1 || command2`

  if (command1 not successful) then do (command2)

```
$ [ -f testfile ] && rm testfile
$ [ -f lockfile ] || touch lockfile
$ [ "$TERM" = "xterm" ] && echo This is no tty
$ cat doesnotexist 2>/dev/null || echo \
> "Oh boy, this file does not exist."
```

The if command

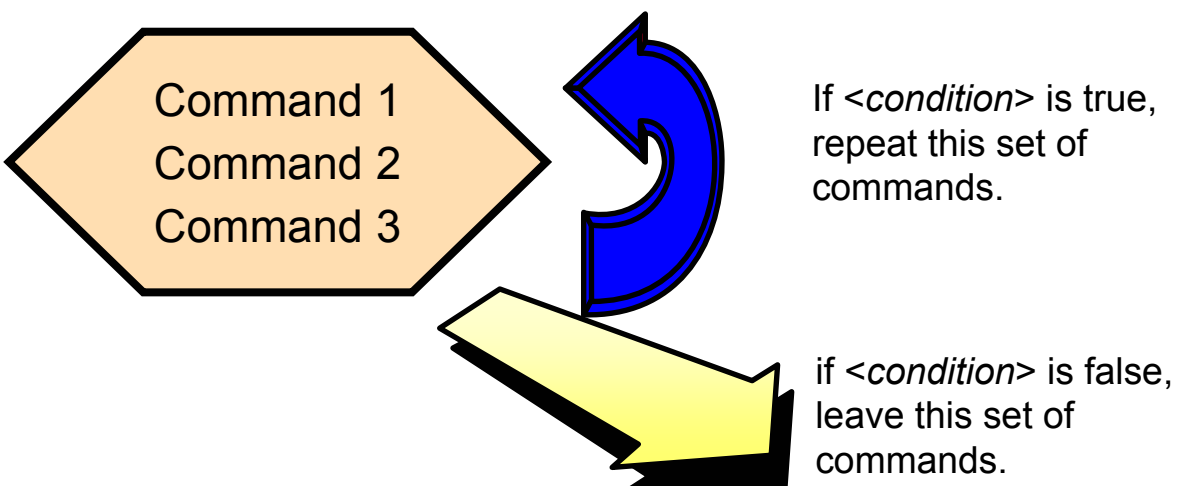
- The structure of the basic if statement is:

```
if command-sequence returns true (0)
then
carry out this set of actions
else
carry out this set of actions
fi
```

```
$ cat myscript
if [ "$MY_VALUE" -eq 10 ]
then
echo MY_VALUE contains the value 10
else
echo MY_VALUE is not 10
fi
$
```

Command repetition

- A *loop* is a set of commands that is executed over and over.
  - Until or while a certain condition is true
  - Or for each item from a list



The while command

- The syntax of the **while** command:

```
while command-sequence-returns-true (0)
do
commands
done
```

```
$ cat myloop
while true
do
echo "It is now $(date)"
echo "There are `ps aux | wc -l` processes"
sleep 600
done
$
```

Note that the command *true* always returns true (0)!

The for command

- The structure of the **for** loop is:

```
for identifier in list
do
commands to be executed on $identifier
done
```

```
$ cat my_forloop
for file in /tmp/mine_*
do
cp $file /other_dir/$file
done
$
```

Shifting shell script arguments

- If you expect a large number of shell arguments (for example, file names), use the **shift** command in a **while** loop to handle them all.

Variables:	$1	$2	$3	$4	$5	$# (count)
At start:	arg1	arg2	arg3	arg4	arg5	5
After first loop:	arg2	arg3	arg4	arg5	*unset*	4
After second loop:	arg3	arg4	arg5	*unset*	*unset*	3

```
$ cat make_backup
while [ $# -gt 0 ]
do
  cp $1 $1.bak
  shift
done
$
```

User interaction: The read command

- The **read** command reads one line from STDIN and assigns the values read to a variable.

```
$ cat delfile
#!/bin/bash
#Usage delfile
echo Please enter the file name:
read name
if [ -f $name ]
then
rm $name
else
echo $name is not an ordinary file -
echo so it is not removed
fi
```

The expr command

- If your shell does not support `$(())` or **let**, use the **expr** command for integer arithmetic.
  - Not a shell built-in, therefore about 10 times slower (it rarely matters)
- Same operators as let:

```
$ echo `expr 3 + 5`
8
```

- Beware of the shell metacharacters *, (and)!

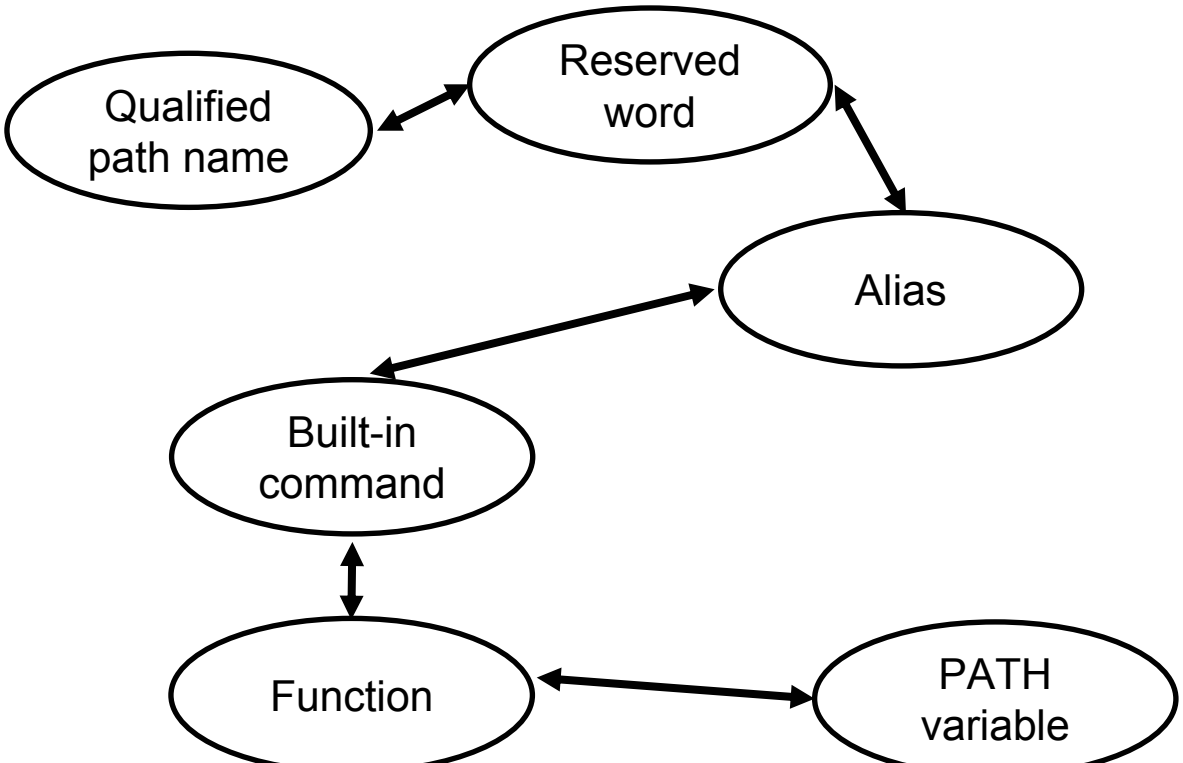
```
$ expr 3 * ( 3 + 5 )
bash: syntax error near unexpected token `(´
$ expr 3 \* \( 3 + 5 \)
24
```

Arithmetic using let

- The bash shell can perform simple arithmetic on integers using the built-in **let** command or the `$(( expr ))` notation.
  - Operators: *, /, +, -, %

```
$ let x=2+3
$ echo $x
5
$ echo $(( 2+3 ))
5
$ let x=3*(3+5)
$ echo $x
24
$ let x=3*3+5
echo $x
14
$ x=$(( 3 * ( 3 + 5 ) ))
```

Command search order



Unit review

- Positional parameters are used to pass to scripts the values from the invoker; they are also in $* or $@.
- To test for a particular condition, the **test** command can be used. This feature is frequently coupled with the **if** statement to control the flow of a program and allow for conditional execution within scripts.
- The **read** command can be used to implement interactive scripts.
- The **while** and **for** commands are used to create loops in a script.
- Simple integer arithmetic can be performed by the **expr** or **let** commands or the `$(( ))` notation.

Checkpoint (1 of 2)

1. What will the following piece of code do?

```
TERMTYPE=$TERM
if [ -n "$TERMTYPE" ]
then
   if [ -f /home/tux1/custom_script ]
   then
            /home/tux1/custom_script
   else
            echo No custom script available!
   fi
else
   echo You don't have a TERM variable set!
fi
```

Checkpoint solutions (1 of 2)

1. What will the following piece of code do?

```
TERMTYPE=$TERM
if [ -n "$TERMTYPE" ]
then
    if [ -f /home/tux1/custom_script ]
    then
              /home/tux1/custom_script
    else
              echo No custom script available!
    fi
else
    echo You don't have a TERM variable set!
fi
```

The answer is if TERMTYPE is set to a non-null value and if custom_script exists as a normal file, the script attempts to execute it. If the script does not exist, this is reported.
If the TERMTYPE is null or not set, that is reported.

Checkpoint (2 of 2)

2. Write a script that will multiply any two numbers together.

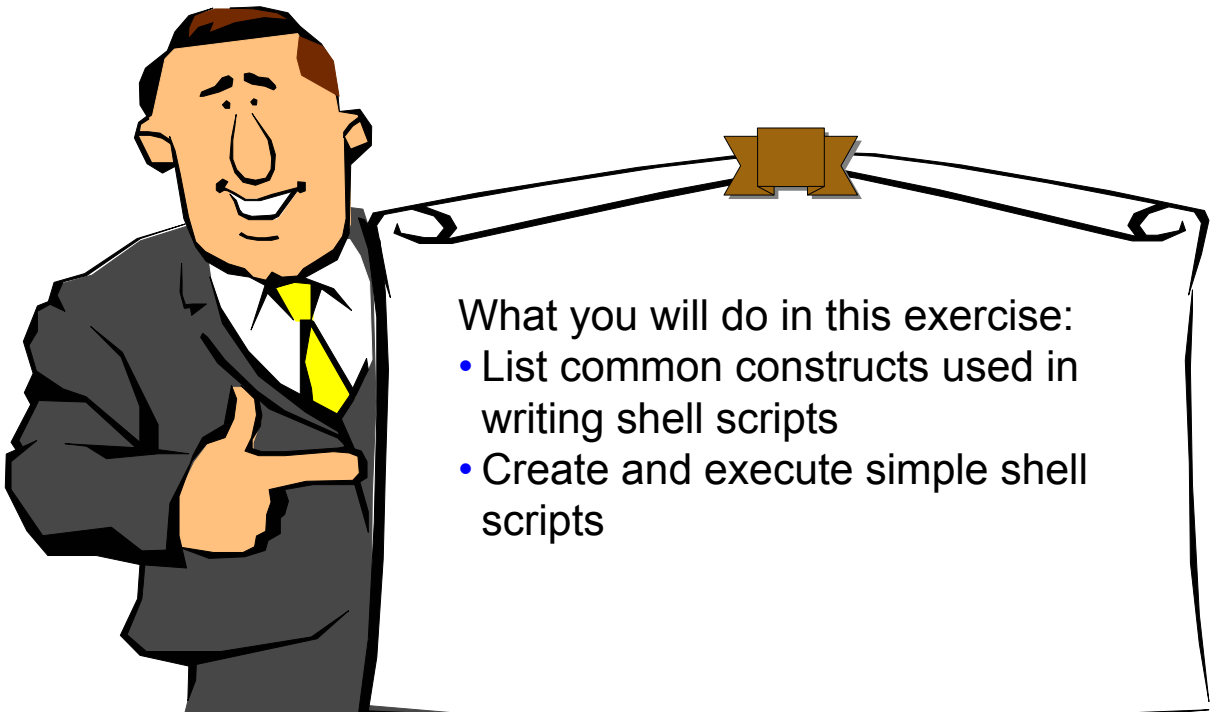
Checkpoint solutions (2 of 2)

2. Write a script that will multiply any two numbers together.

   The answer is:

```
#!/bin/bash
echo $(($1 * $2))
```

Exercise: Shell scripting



What you will do in this exercise:

- List common constructs used in writing shell scripts
- Create and execute simple shell scripts

Unit summary

Having completed this unit, you should be able to:

- Invoke shell scripts in three separate ways and explain the difference
- Pass positional parameters to shell scripts and use them within scripts
- Implement interactive shell scripts
- Use conditional execution and loops
- Perform simple arithmetic