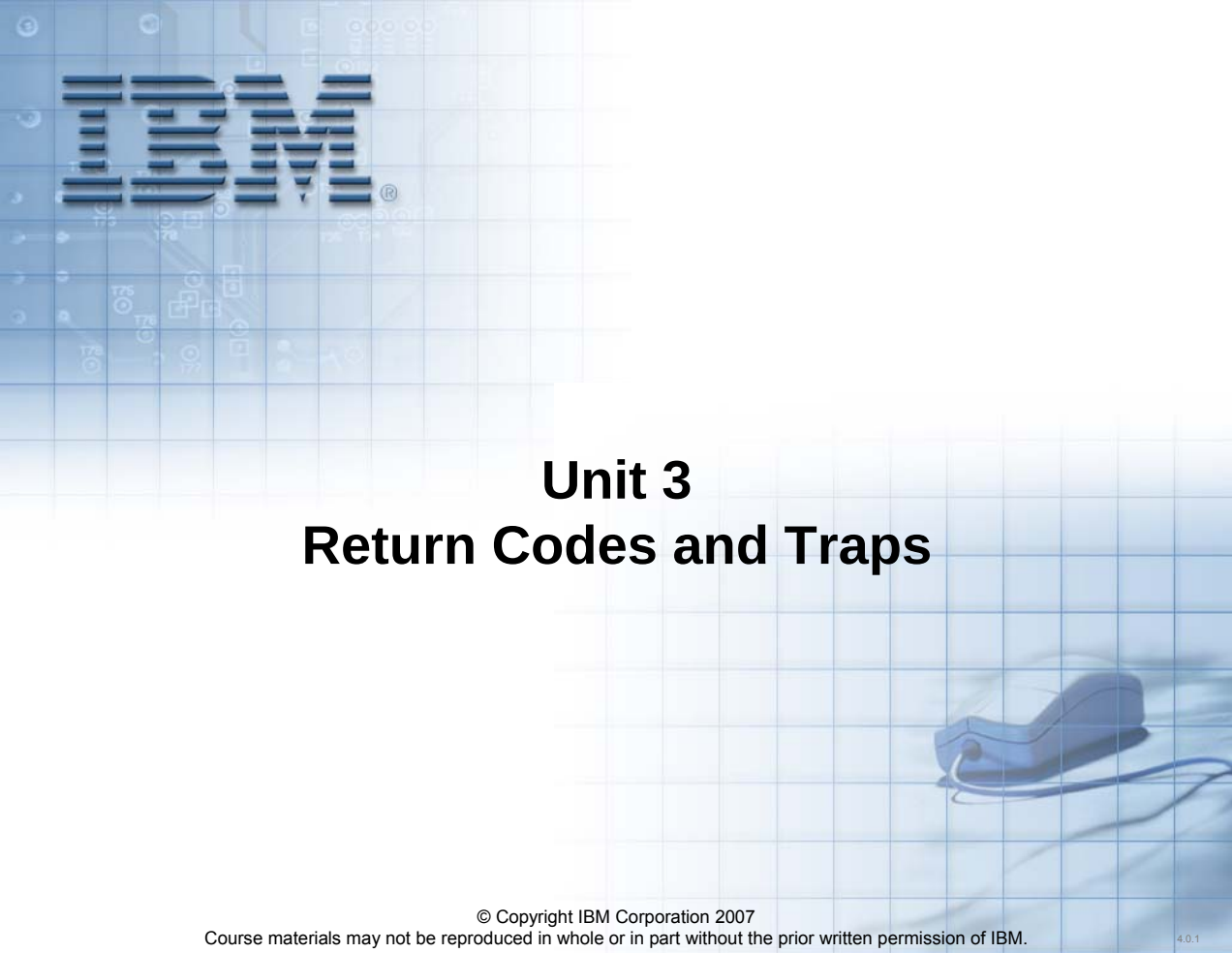

Unit 3
Return Codes and Traps

Unit Objectives

After completing this unit, you should be able to:

- Recognize return values
- Identify exit codes
- Identify conditional execution
- Use the `test` command
- Understand compound expressions
- Examine file `test` operators
- Use numerical expressions
- Understand string expressions
- Understand shell `test` operators
- Use shell `[[ ]]` expressions
- Handle signals
- Understand sending signals
- Understand catching signals

Return Values

Each command, pipeline, or group of commands returns a value to its parent process.

$? contains the value of the return code

- **zero** means success
- **non-zero** means an error occurred

The single value returned by a pipeline is the return code of the last command in the pipeline.

For grouped commands – that is, **( )** or **{ }** – the return code is that of the last command executed in the group.

Exit Status

A shell script provides a return code using the **exit** command.

```
$ print $$                    check the shell process id
879
$ ksh                         start a new subshell
$ print $$                    and check its process id
880
$ exit                        quit the subshell
$ print $?                    and print the return code
0
$ print $$
879
$ ksh                         begin another subshell
$ print $$
890
$ exit 101                    exit with a value to set
$ print $?                    the return code
101
$ print $$
879
$ _
```

Conditional Execution

A return code (or exit status) can be used to determine whether or not to execute the next command.

- If command1 **is** successful execute command2

```
command1 && command2
```

```
$ rm -f file1 && print file1 removed
```

- If command1 **is not** successful execute command2

```
command1 || command2
```

```
$ who|grep marty || print Marty logged off
```

The test Command

The test command is used for expression evaluation

```
test expression
```

- or

```
[ expression ]
```

- Returns zero if the expression is true
- Returns non-zero if the expression is false

The Korn and Bash shells provide an improved version

```
[[ expression ]]
```

- Easier syntax
- Includes same functionality as **`test`**
- Additional operators
- Shell expansions prevented

File Test Operators

File status can be examined using several operators.

Operator:	*True if ...:*
`-s file`	file has a size greater than zero
`-r file`	file exists and is readable
`-w file`	file exists and is writable
`-x file`	file exists and is executable
`-u file`	file exists and has the SUID bit set
`-g file`	file exists and has the SGID bit set
`-k file`	file exists and has the SVTX sticky bit set
`-e file`	file exists
`-f file`	file exists and is an ordinary file
`-d file`	file exists and is a directory
`-c file`	file exists as a character special file
`-b file`	file exists and is a named pipe file
`-p file`	file exists and is a named pipe file
`-L file`	file exists and is a symbolic link

Numeric Expressions

For arithmetic expressions and integer values use:

Expression:	*True if ...:*
exp1 -eq exp2	**exp1** is equal to **exp2**
exp1 -ne exp2	**exp1** is not equal to **exp2**
expl -lt exp2	**exp1** is less than **exp2**
exp1 -le exp2	**exp1** is less than or equal to **exp2**
exp1 -gt exp2	**exp1** is greater than **exp2**
exp1 -ge exp2	**exp1** is greater than or equal to **exp2**

String Expressions

To examine strings use one of the following:

Expression:	*True if ...:*
`-n str`	`str` is non-zero in length
`-z str`	`str` is zero in length
`str1 = str2`	`str1` is the same as `str2`
`str1 != str2`	`str1` is not the same as `str2`

More Shell Test Operators

The shell provides a number of additional **`test`** operators.

Expression:	*True if ...:*
`file1 -ef file2`	**`file1`** is another name for **`file2`**
`file1 -nt file2`	**`file1`** is newer than **`file2`**
`file1 -ot file2`	**`file1`** is older than **`file2`**
`-t des`	**`file`** descriptor **`des`** is open and associated with a terminal device

Shell [[ ]] Expressions

When using the shell **[[ ]]** syntax there are a few extra expressions.

Expression:	*True if ...:*
str = pattern	**str** matches pattern
str != pattern	**str** does not match pattern
str1 < str2	**str1** is before **str2** in the ASCII collation seq.
str1 > str2	**str1** is after **str2** in the ASCII collation seq.
-o opt	option **opt** is on for this shell

You may use shell metacharacters in the patterns.

Compound Expressions

For the `[ ]` or `test` command

`exp1 -a exp2`	binary **`and`** operation
`exp1 -o exp2`	binary **`or`** operation
`! exp`	logical negation
`\(  \)`	used to group expressions

For the `[[  ]]` syntax

`exp1 && exp2`	true if both expressions are true - the second is **only evaluated** if the first is true
`exp1 || exp2`	true if either expression is true - the second is **only evaluated** if the first is false
`! exp`	logical negation
`(  )`	used to group expressions

Practice Test

```
$ [[ -s /etc/passwd || -r /etc/group ]]
$ print $?          True or False?

$ test -f /etc/motd -a ! -d /home
$ print $?          True or False?

$ x="005"
$ y=" 10"
$ test "$y" -eq 10
$ print $?          True or False?

$ [ "$x" = 5  ]
$ print $?          True or False?

$ [[  -n "$x"  ]]
$ print $?          True or False?

$ test  -S  /dev/tty0
$ print $?          True or False?

$ [[  1234 = +([0-9])  ]]
$ print $?          True or False?
```

Signals

- The kernel sends ***signals*** to processes during their execution
  - Certain system events issue signals when they
    - Run out of paging space
    - Receive special key sequences like **`<Ctrl-c>`**
  - The **`kill`** command sends a specific signal to a process

What You Can Do with Signals

Signals sent to processes may be:

- Caught — the process deals with it
- Ignored — nothing happens
- Defaulted — use default *handlers*

The Kill Command

- To send a signal to a process:

```
kill -sig pid       -or-       kill -s sig pid
```

- To list all defined signals

```
kill -l       (lowercase "ell")
```

- To list a specific signal

```
kill -l #     (replace # with a number)
```

- To list the signal that caused an exit error

```
kill -l $?
```

Signal List (1 of 2)

Here is a list of some useful signals.

Signal:	*Event:*
0 EXIT	issued when a process or function completes (shell specific)
1 HUP	you logged out while the process was still running – sent to sub-shells too
2 INT	interrupt pressed **<Ctrl-c>**
3 QUIT	quit key sequence pressed **<Ctrl-\>**
9 KILL	special 'force' signal, cannot be ignored
15 TERM	default kill command signal
18 TSTP	process suspend **<Ctrl-z>**

Signal List (2 of 2)

Signal:		*Event:*
19	CONT	continue if stopped – issued by **`kill`** to a suspended process before TERM or HUP
29	PWR	power failure imminent – save data now!
33	DANGER	paging space low
63	SAK	you pressed **<Ctrl-x>** and **<Ctrl-r>** the SAK sequence

Catching Signals with Traps

The **trap** command specifies any special processing you want to do when the process receives a signal:

To process signals

```
$ trap 'rm /tmp/$$; print signal!; exit 2' 2 3
```

To ignore signals

```
$ trap '' INT QUIT
```

To reset signal processing

```
$ trap - INT QUIT            -or-        trap 2 3
```

To list traps set

```
$ trap
```

Trap Example

```
#!/usr/bin/ksh
# ps_monitor
# monitor processes using ps -elf at intervals
# of 30 seconds for 2 minutes.  If interrupted,
# a summary report is produced by executing
# psummary.
#
trap 'print $0: interrupt received ;
        ./psummary ;
        exit' 2 3 15
ps -elf > /tmp/pdata
sleep 30
ps -elf >> /tmp/pdata
sleep 30
ps -elf >> /tmp/pdata
sleep 30
ps -elf >> /tmp/pdata
sleep 30
ps -elf >> /tmp/pdata
trap - 2 3 15
```

Checkpoint

1. How can you tell whether a command you have just entered was successful?
2. How can you test if file *datafile* is non-empty?
3. How can you check if you have been logged on for more than 20 minutes, and if so, print out a suitable message?
4. How could you log off, using the kill command?
5. If you are a DBA is this a desirable command to terminate the <oracle_server>? **`kill -KILL <oracle_server>`**
6. What does this command do? **`trap echo you did <Ctrl-c> 2`**
7. How could you get <Ctrl-c> to log you off?

Unit Summary

- Return values
- Exit status
- Conditional execution
- The `test` command
- Compound expressions
- File `test` operators
- Numerical expressions
- String expressions
- Shell test operators
- Shell `[[ ]]` expressions
- Signals
- Sending signals – `kill` command
- Catching signals – `trap` command