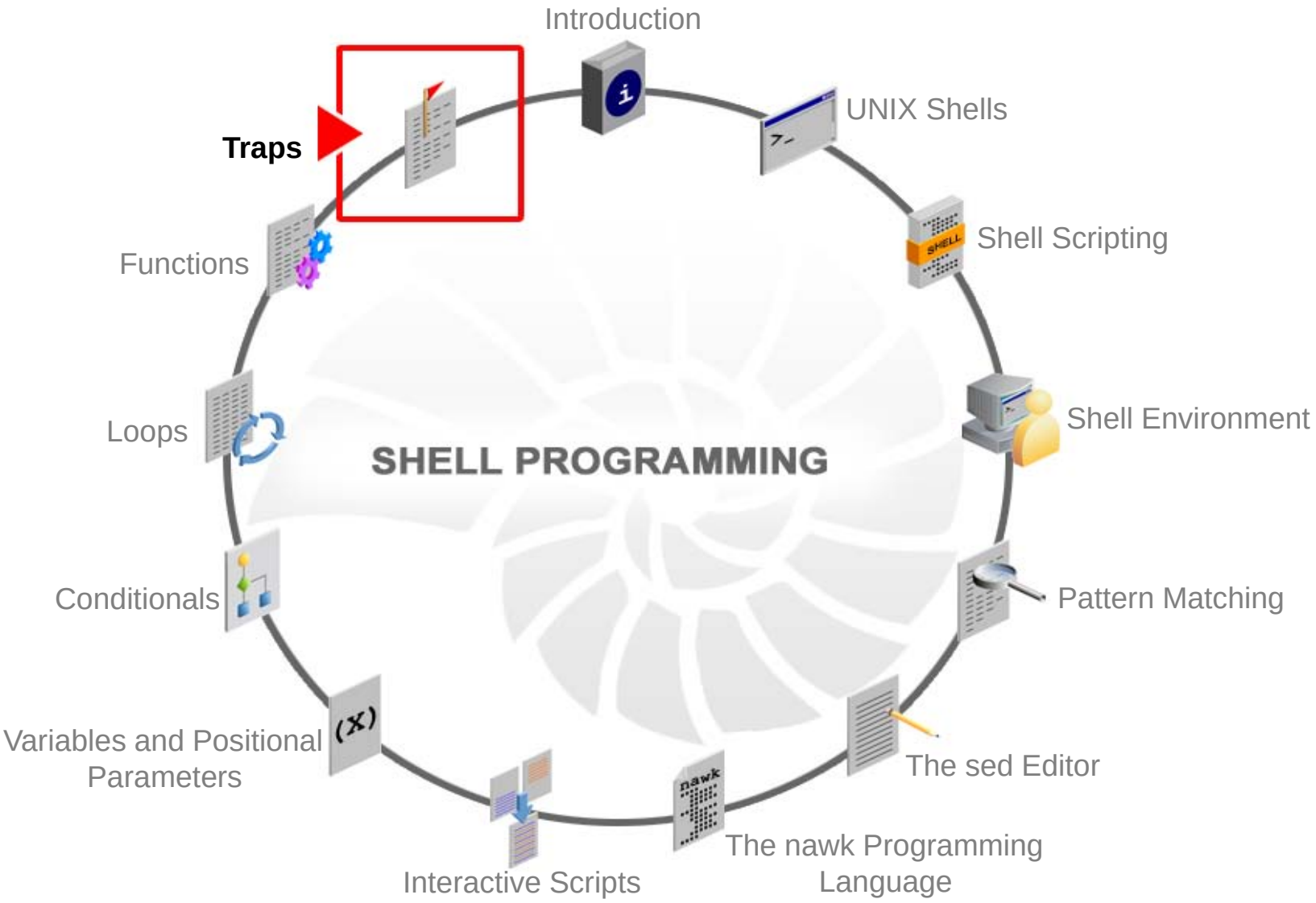
13

Traps



Objectives

After completing this lesson, you will be able to:

- Describe the role of shell signals in interprocess communication
- Catch signals and user errors with the `trap` statement

Agenda

- Describing the role of shell signals in interprocess communication
- Catching signals and user errors with the `trap` statement

Shell Signals

- A shell signal is a message sent from one process to another indicating abnormal event.
- You can also send your own signals through:
  - Keyboard sequences such as `Control-C`
  - The `kill` command at the shell level

Signals Using Keyboard Sequence

The following are some signals that can be sent from the keyboard:

- Signal 2 (`INT`) by pressing `Control-C`
- Signal 3 (`QUIT`) by pressing `Control-\`
- Signal 23 (`STOP`) by pressing `Control-S`
- Signal 24 (`TSTP`) by pressing `Control-Z`
- Signal 25 (`CONT`) by pressing `Control-Q`

Signals Using the `kill` Command

- You can send a signal to processes running on the system by using the `kill` command.

```
kill -signal pid
```

- For example, the `-9` option sends the `KILL` signal to the process.

```
kill -9 pid
kill -KILL pid
```

Note: When you do not specify a signal name or number, the `TERM` signal, signal `15`, is sent to the process.

Shell Signal Values

- The bash shell has 72 defined signals.
- Each signal has a name and a number associated with it.
- The following is a list of shell signal values generated by the `kill-l` command:

```
$ kill -l
1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE    14) SIGALRM     15) SIGTERM     16) SIGUSR1
17) SIGUSR2    18) SIGCHLD     19) SIGPWR      20) SIGWINCH
21) SIGURG     22) SIGIO       23) SIGSTOP     24) SIGTSTP
25) SIGCONT    26) SIGTTIN     27) SIGTTOU     28) SIGVTALRM
29) SIGPROF    30) SIGXCPU
..... (output omitted)
```

Quiz

Which of the following signals is sent to a process when you do not specify a signal name or number with the `kill` command?

a. Signal 2 (`INT`)
b. Signal 3 (`QUIT`)
c. Signal 15 (`TERM`)
d. Signal 23 (`STOP`)
e. Signal 24 (`TSTP`)
f. Signal 25 (`CONT`)

Agenda

- Describing the role of shell signals in interprocess communication
- Catching signals and user errors with the `trap` statement

The `trap` Statement

- Most signals sent to a process executing a shell script cause the script to terminate.
- You can use the `trap` statement to avoid having the script terminate from specified signals.
- Syntax:

```
trap 'action' signal [ signal2 ... signalx ]
```

- Example:

```
trap ’echo "Control-C not available"’ INT
```

Example

```
$ cat trapsig.sh
#!/bin/bash

# Script name: trapsig.sh

trap 'print "Control-C cannot terminate this script."' INT
trap 'print "Control-\ cannot terminate this script."' QUIT
trap 'print "Control-Z cannot terminate this script."' TSTP

print "Enter any string (type 'dough' to exit)."
while (( 1 ))
do
        print -n "Rolling..."
        read string
        if [[ "$string" = "dough" ]]
        then
                break
        fi
done
        print "Exiting normally"
```

Example

```
$ ./trapsig.sh
Enter any string (type 'dough' to exit).
Rolling...
Rolling...d
Rolling...s
Rolling...Rolling...Rolling...Rolling...Rolling...4
Rolling...^c
Control-C cannot terminate this script.
Rolling...^\
Control-\ cannot terminate this script.
Rolling...^z
Control-Z cannot terminate this script.
Rolling...dough
Exiting normally
$
```

Catching User Errors

- In addition to catching signals, you can use the `trap` statement to take specified actions when an error occurs in the execution of a script.
- The syntax for this type of `trap` statement is:

```
trap 'action' ERR
```

- The value of the `$?` variable in the script indicates when the `trap` statement is to be executed.
- It holds the `exit` (error) status of the previously executed command or statement.
- The `trap` statement is executed whenever `$?` becomes nonzero.

Catching User Errors

```
$ cat traperr1.sh
#!/bin/bash
# Script name: traperr1.sh
declare -i num
while [ 1 ]
do
        echo "Enter any number ( 0 to exit ):"
        read num
        if [ $num -eq 0 ]
        then
                echo "Exiting normally ..."
                exit 0
        else
                print "Square of $num is $(( num * num )). \n"
        fi
Done
$ ./traperr1.sh
Enter any number (-1 to exit): r
traperr1.sh[9]: r: bad number
Square of 2 is 4.
Enter any number (-1 to exit): -1
```

The ERR Signal

- The standard error messages are printed to the screen.
- You can however redirect the error messages, such that, if an error occurs, the user sees just the message that you set up with the `trap` statement.

The ERR Signal

```
$ cat trapsig2.sh
#!/bin/bash
# Script name: trapsig2. sh
declare -i num
exec 2> /dev/null
trap 'print "You did not enter an integer.\n"' ERR
while (( 1 ))
do
    print -n "Enter any number ( -1 to exit ): "
    read num
    status=$?
    if (( num == -1 ))
    then
            break
    elif (( $status == 0 ))
    then
            print "Square of $num is $(( num * num )). \n"
    fi
done
print "Exiting normally"
```

The ERR Signal

```
$ ./trapsig2.sh
Enter any number ( -1 to exit ): 3
Square of 3 is 9.

Enter any number ( -1 to exit ): r
You did not enter an integer.

Enter any number ( -1 to exit ): 8
Square of 8 is 64.

Enter any number ( -1 to exit ): -1
Exiting normally
```

Declaring the `trap` Statement

- To trap a signal any time during execution, define the `trap` statement at the start of the script.
- To trap a signal only when certain command lines are executed, turn on the `trap` statement before the lines, and then turn off the `trap` statement after the lines.
- If a loop is being used, a `trap` statement can include the `continue` command to make the loop start again from its beginning.
- You can also trap the `EXIT` signal so that certain commands are executed only when the shell script is being terminated with no errors.

Declaring the `trap` Statement

```
#ident "@(#)profile 1.18 98/10/03 SMI "/* SVr4.0 1.3 */
# The profile that all logins get before using their own
.profile.
trap "" 2 3 # trap INT (Control-C) and QUIT (Control-\)
# and give no feedback
export LOGNAME PATH
if [ "$TERM" = "" ]
then
        if /bin/i386
        then
                TERM=sun-color
        else
                TERM=sun
        fi
        export TERM
fi
```

Declaring the `trap` Statement

```
# Login and -su shells get /etc/profile services.
# -rsh is given its environment in its .profile.
case "$0" in
-sh | -ksh | -jsh)
if [ ! -f .hushlogin ]
then
/usr/sbin/quota
# Allow the user to break the Message-Of-The-Day only.
# The user does this by using Control-C (INT).
# Note: QUIT (Control-\) is still trapped (disabled).
trap "trap '' 2" 2
/bin/cat -s /etc/motd
trap "" 2 # trap Control-C (INT) and give no feedback.
```

Declaring the `trap` Statement

```
/bin/mail -E
case $? in
0)
echo "You have new mail."
;;
2)
echo "You have mail."
;;
esac
fi
esac
umask 022
trap 2 3 # Allow the user to terminate with Control-C (INT) or
# Control-\(QUIT)
```

Quiz

The `trap` statement does not work if you attempt to trap the `KILL` or `STOP` signals.

a. True
b. False

Summary

In this lesson, you should have learned how to:

- Describe the role of shell signals in interprocess communication
- Catch signals and user errors with the `trap` statement

Practice 13 Overview: Traps

This practice covers the following topic:

- Using Traps