# UNX510/DPS918 - Unix BASH Shell Scripting

## Lecture 9 - Signals; Functions

### Signals

* signals are sometimes called interrupts, because they interrupt the normal flow of a program
* here is an example of a script that won't terminate without receiving a signal:
* while :
* do
* echo -n .
* sleep 1

done

- note that ":" is a synonym for "true"

* signals are used to handle events that are unexpected (such as <Ctrl>-c) or are asynchronous (have an unpredictable timing, such as death of a child process)
* when the OS recognizes such an event, it sends a signal to the affected process
* a process can also send another process a signal, if it has permission, using kill
* most signals have a default behaviour (such as aborting the process), or can be caught and handled
  + a process can be programmed to ignore a signal, or a "signal handler" can be used to process it
* signals have a name and number, either can be used
  + most signals are documented in man 7 signal
* here are the signals most useful in scripting:
* Signal Value Description
* SIGHUP 1 Hangup detected on controlling terminal or death of controlling process
* SIGINT 2 Interrupt from keyboard (Ctrl-c)
* SIGQUIT 3 Quit from keyboard (Ctrl-\)
* SIGALRM 14 Timer signal from alarm
* SIGTERM 15 Termination signal ("kill" default)
* SIGUSR1 User-defined signal 1
* SIGUSR2 User-defined signal 2

EXIT Process exit detected

### Trapping Signals

* trap - specifies action to take when a signal is detected
  + trap 'rm -f /tmp/\*$$\*; exit' SIGINT - action for <Ctrl>-c
  + note that any of the following signal identifiers can be used: SIGINT, INT, sigint, int, 2
  + trap - INT - reset action for <Ctrl>-c to default
  + trap 'rm -f /tmp/\*$$\*; exit' SIGHUP SIGINT SIGTERM - can trap multiple signals
  + trap "echo The script is now exiting" EXIT - action when script is exiting
* here is an example of a script that traps all the signals mentioned above:
* echo "PID is $$"
* rm sig\* exit 2>/dev/null
* trap "echo 'hup received'; touch sighup" hup
* trap "echo 'int received'; touch sigint" int
* trap "echo 'quit received'; touch sigquit" quit
* trap "echo 'alrm received'; touch sigalrm" alrm
* trap "echo 'term received'; touch sigterm" term
* trap "echo 'usr1 received'; touch sigusr1" usr1
* trap "echo 'usr2 received'; touch sigusr2" usr2
* trap "echo 'exit received'; touch exit" exit
* quit=
* while [ "$quit" != q ]
* do
* read -p "Enter something: " quit

done

* for example, in another terminal window, try: kill -alrm PID   (with the appropriate PID)
* note that kill -9 will not trigger any of these signals, it cannot be trapped
* here is an example of resetting a signal to default behaviour:
* interrupts=0
* trap "((interrupts++)); echo -e \"\nOuch! That's \$interrupts Cntl-C\"" SIGINT
* while :
* do
* if [ $interrupts = 3 ]
* then
* trap - SIGINT
* fi
* echo -n .
* sleep 1

done

### Functions

* three equivalent styles of function definition:
* function\_name() {
* statements
* }
* function function\_name() {
* statements
* }
* function function\_name {
* statements

}

* must be defined before use, usually at the beginning of the script
* invoked by using name, can pass arguments
* arguments replace the $1, $2, ... variables during function execution
* "return" statement can be used to return a numeric value, same as "exit" from a script
* to return strings, either place in a variable, or use command substitution when invoking the function
  + note that all variables are global by default

### Returning Values

* an example, the function placing the result into a variable:
* add() {
* sum=0
* for num
* do
* sum=$((sum + num))
* done
* }
* add 23 45 -17

echo $sum

* a similar example, the function writing the result to standard output:
* add() {
* sum=0
* for num
* do
* sum=$((sum + num))
* done
* echo $sum
* }

add 23 45 -17

* another similar example, using command substitution to retrieve the result from the function:
* add() {
* sum=0
* for num
* do
* sum=$((sum + num))
* done
* echo $sum
* }
* xxx=$(add 23 45 -17)

echo $xxx

* yet another similar example, using the **return** statement:
* add() {
* sum=0
* for num
* do
* sum=$((sum + num))
* done
* return $sum
* }
* add 23 45 -17

echo $?

### Local Variables

* variables can be made local using declare -l or local, here's an example:
* f1() {
* declare -l a=1 b
* b=2
* local c d=4
* c=3
* e=5
* echo "Inside the function: a=$a b=$b c=$c d=$d e=$e"
* }
* a=0; b=0; c=0; d=0; e=0
* f1

echo "Outside the function: a=$a b=$b c=$c d=$d e=$e"

### Recursive Functions

* this example sends each line in a file to a recursive function which capitalizes each word in the line:
* capitalize\_line() {
* echo $1 | cut -c1 | tr 'a-z' 'A-Z' | tr -d '\n'
* echo "$1" | sed -r 's/^.([^ ]\* \*).\*$/\1/' | tr -d '\n'
* if echo $1 | grep ' ' >/dev/null
* then
* capitalize\_line "$(echo "$1" | sed -r 's/^[^ ]+ +//')"
* else
* echo
* fi
* }
* [ $1 ] && exec < $1
* while read line
* do
* capitalize\_line "$line"

done

* note that this script reads from standard input if no filename is specified:   
    
  capitalize cars   
    
  capitalize < cars   
    
  grep ford cars | capitalize

### Function Libraries

* when developing large, multi-script systems (such as the online labs for this course), it's desireable to keep a central library of commonly-used functions
* a function library is simply a script containing functions, invoked within each script requiring access to those functions
* this example is functionally the same as the previous one, with some improvements:
* exit\_success() {
* # No arguments
* rm /tmp/\*$$\* 2>/dev/null
* exit 0
* }
* exit\_failure() {
* # Argument 1 - exit-status (optional, defaults to 1)
* # Argument 2 - error-message (optional, defaults to null)
* rm /tmp/\*$$\* 2>/dev/null
* [ "$2" != "" ] && echo "$2" >&2
* [ "$1" = "" ] && set 1
* exit $1
* }
* capitalize\_line() {
* # Argument 1 - string to be capitalized
* echo $1 | cut -c1 | tr 'a-z' 'A-Z' | tr -d '\n'
* echo "$1" | sed -r 's/^.([^ ]\* \*).\*$/\1/' | tr -d '\n'
* if echo $1 | grep ' ' >/dev/null
* then
* capitalize\_line "$(echo "$1" | sed -r 's/^[^ ]+ +//')"
* else
* echo
* fi
* }
* trap "exit\_failure 2 'Command interrupted'" INT HUP TERM
* if [ "$1" != "" ]
* then
* if [ ! -f "$1" ]
* then
* exit\_failure 3 "File $1 is not an existing ordinary file"
* elif [ ! -r "$1" ]
* then
* exit\_failure 4 "No read permission for file $1"
* else
* exec 0<$1
* fi
* fi
* while read line
* do
* capitalize\_line "$line"
* done

exit\_success

* assuming the functions may be useful in other related scripts, they may be placed into a separate script, for example "common\_functions":
* exit\_success() {
* # No arguments
* rm /tmp/\*$$\* 2>/dev/null
* exit 0
* }
* exit\_failure() {
* # Argument 1 - exit-status (optional, defaults to 1)
* # Argument 2 - error-message (optional, defaults to null)
* rm /tmp/\*$$\* 2>/dev/null
* [ "$2" != "" ] && echo "$2" >&2
* [ "$1" = "" ] && set 1
* exit $1
* }
* capitalize\_line() {
* # Argument 1 - string to be capitalized
* echo $1 | cut -c1 | tr 'a-z' 'A-Z' | tr -d '\n'
* echo "$1" | sed -r 's/^.([^ ]\* \*).\*$/\1/' | tr -d '\n'
* if echo $1 | grep ' ' >/dev/null
* then
* capitalize\_line "$(echo "$1" | sed -r 's/^[^ ]+ +//')"
* else
* echo
* fi

}

* the following will NOT work, because "common\_functions" will be executed in a child process:
* common\_functions
* trap "exit\_failure 2 'Command interrupted'" INT HUP TERM
* if [ "$1" != "" ]
* then
* if [ ! -f "$1" ]
* then
* exit\_failure 3 "File $1 is not an existing ordinary file"
* elif [ ! -r "$1" ]
* then
* exit\_failure 4 "No read permission for file $1"
* else
* exec 0<$1
* fi
* fi
* while read line
* do
* capitalize\_line "$line"
* done

exit\_success

* the following WILL work, "common\_functions" will be executed in the current process by using source or .:
* source common\_functions
* trap "exit\_failure 2 'Command interrupted'" INT HUP TERM
* if [ "$1" != "" ]
* then
* if [ ! -f "$1" ]
* then
* exit\_failure 3 "File $1 is not an existing ordinary file"
* elif [ ! -r "$1" ]
* then
* exit\_failure 4 "No read permission for file $1"
* else
* exec 0<$1
* fi
* fi
* tempfile=/tmp/capitalize.$$.line
* while read line
* do
* capitalize\_line "$line"
* done

exit\_success

* note that execute permission is NOT required if a script is invoked using source or .
* use an absolute pathname with source or . so that the functions can be run regardless of the current path