CIT 352 BASH Scripting Tips

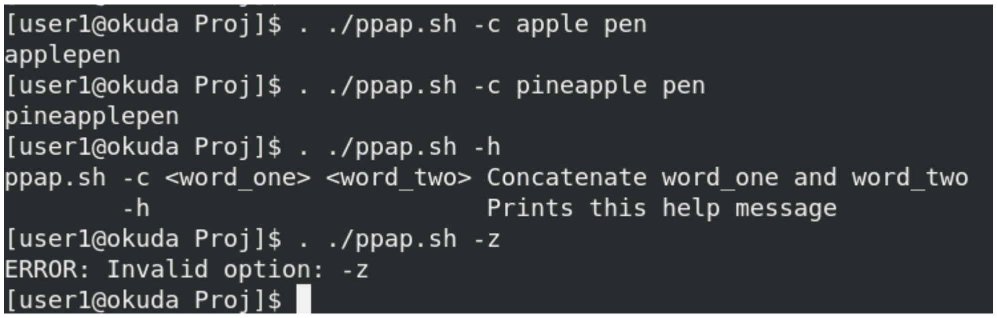
1. Useful system calls for the project (check the man pages)

- `getent passwd $login (to verify if $login exists in /etc/passwd)`
- `chsh -s`
- `userdel -r`
- `useradd -m -s`
- `echo "$password" | passwd --stdin "$login" (Assign the password $password to the user $login)`

2. Parse the command line options

Here's an example of how to parse the command line options. In this example, a user is expected to type either "-c" followed by additional parameters or "-h" for a help message. If anything other than "-c" or "-h" is typed, an error message will be printed.

```
$ cat ppap.sh
#! /bin/bash
#
# Concatenate words according to the following options
# ppap.sh -c <word_one> <word_two>     Concatenate word_one and
word_two
#         -h                           Prints this help message
#

parse_input () {

    case "$1" in
    -c )
        echo "$2$3"
    ;;
    -h )
        echo "ppap.sh -c <word_one> <word_two> Concatenate word_one and
word_two
"
        echo "        -h                      Prints this help
message"
    ;;
    * )
        echo "ERROR: Invalid option: $1"
    ;;
    esac

parse_input "$@"
```

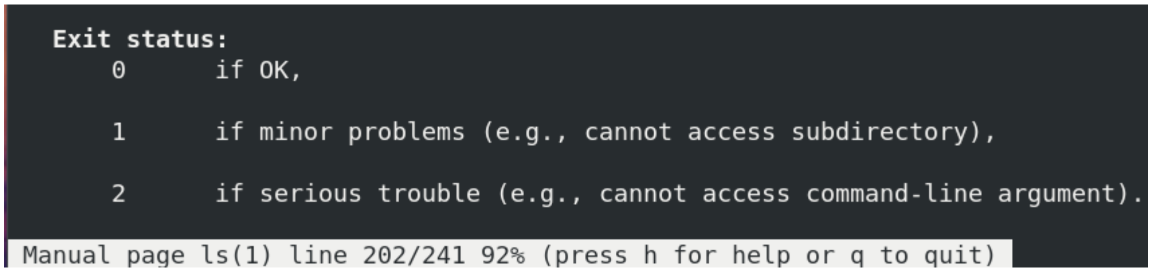


```
[user1@okuda Proj]$ . ./ppap.sh -c apple pen
applepen
[user1@okuda Proj]$ . ./ppap.sh -c pineapple pen
pineapplepen
[user1@okuda Proj]$ . ./ppap.sh -h
ppap.sh -c <word_one> <word_two> Concatenate word_one and word_two
        -h                       Prints this help message
[user1@okuda Proj]$ . ./ppap.sh -z
ERROR: Invalid option: -z
[user1@okuda Proj]$ 
```

3. System call exit status

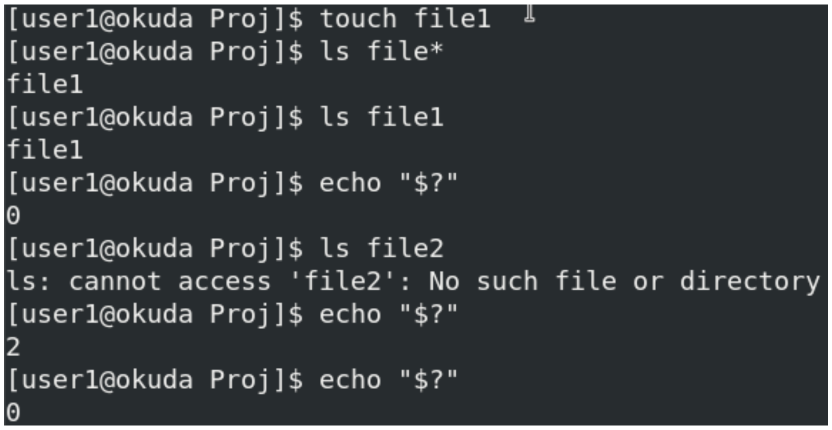
Exit status of a system call can be tested (whether the operation succeeded or resulted in an error) by testing the variable "$?" immediately after the system call execution. If the system call exits successfully, it will return 0 and "$?" will be set to 0. Otherwise, none-zero value will be set. Here's what the man page says about the exit status of "ls" command:

$ man ls



```
   Exit status:
       0      if OK,

       1      if minor problems (e.g., cannot access subdirectory),

       2      if serious trouble (e.g., cannot access command-line argument).

 Manual page ls(1) line 202/241 92% (press h for help or q to quit)
```

Below is an example of displaying the exit status of the ls command under successful and unsuccessful conditions. Note that the last line returned "0" because the previous echo command succeeded.



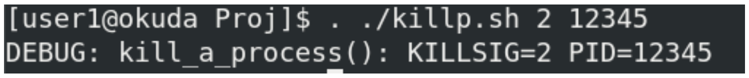
```
[user1@okuda Proj]$ touch file1
[user1@okuda Proj]$ ls file*
file1
[user1@okuda Proj]$ ls file1
file1
[user1@okuda Proj]$ echo "$?"
0
[user1@okuda Proj]$ ls file2
ls: cannot access 'file2': No such file or directory
[user1@okuda Proj]$ echo "$?"
2
[user1@okuda Proj]$ echo "$?"
0
```

4. Function call Example 1 (Print a debug message)

To debug a shell script, you will need to rely on inline debug messages you insert. Below is an example of printing a debug message. It helps you to know that the execution pointer entered kill_a_process () and what the values of the two positional parameters that are passed in.

```
$ cat killp.sh
```

```
#! /bin/bash
#
# Kill a specified process
#
# Usage: killp.sh <killsig> <PID>
#

function kill_a_process (){

      echo "DEBUG: kill_a_process(): KILLSIG=$1 PID=$2"
}

kill_a_process "$1" "$2"
```



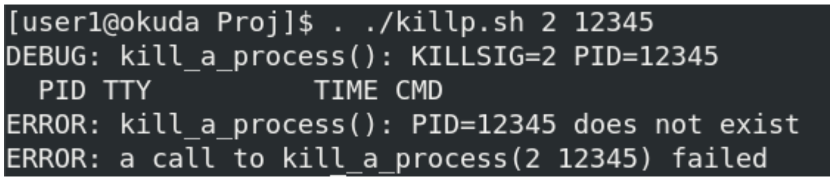
```
[user1@okuda Proj]$ . ./killp.sh 2 12345
DEBUG: kill_a_process(): KILLSIG=2 PID=12345
```

5. Function call Example 2 (Print a debug message. Check for the existence of the specified process)

   killp.sh tries to kill a process with the specified kill signal. Before attempting to kill the process, it is a good idea to test whether the specified process actually exists. You can give “ps -p <pid>” command and test the exit status to accomplish this task. The script below also shows that a function call can return an exit status and be tested by the caller.

```
$ cat killp.sh
#! /bin/bash
#
# Kill a specified process
#
# Usage: killp.sh <killsig> <PID>
#

function kill_a_process () {

      echo " DEBUG: kill_a_process(): KILLSIG=$1 PID=$2"

      # check to see if the process exists
      ps -p "$2"
      if (("$?" != 0))
      then
            echo "ERROR: kill_a_process(): PID=$2 does not exist"
            return 1
      fi

      return 0
}

kill_a_process "$1" "$2"
if (("$?" != 0))
then
      echo "ERROR: a call to kill_a_process($1 $2) failed"
      return 1
fi
```

```
return 0
```

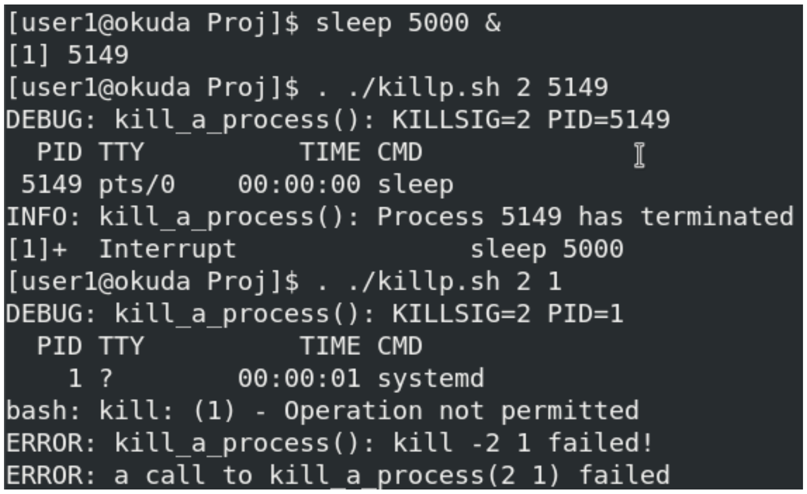


```
[user1@okuda Proj]$ . ./killp.sh 2 12345
DEBUG: kill_a_process(): KILLSIG=2 PID=12345
  PID TTY          TIME CMD
ERROR: kill_a_process(): PID=12345 does not exist
ERROR: a call to kill_a_process(2 12345) failed
```

6. Function call Example 3 (Print a debug message. Check for the existence of the specified process. Kill the process. If kill fails, print an error message.)

   An implementation of attempting to kill a process is shown below.

```
$ cat killp.sh
#! /bin/bash
#
# Kill a specified process
#
# Usage: killp.sh <killsig> <PID>
#

function kill_a_process () {

      echo " DEBUG: kill_a_process(): KILLSIG=$1 PID=$2"

      # check to see if the process exists
      ps -p "$2"
      if (("$?" != 0))
      then
            echo "ERROR: kill_a_process(): PID=$2 does not exist"
            return 1
      fi

      # kill the process
      kill -"$1" "$2"
      if (("$?" != 0))
            then
                  echo "ERROR: kill_a_process(): kill -$1 $2 failed!"
            return 1
      fi

      echo "INFO: kill_a_process(): Process $2 has terminated"

      return 0

}

kill_a_process "$1" "$2"
if (("$?" != 0))
then
      echo "ERROR: a call to kill_a_process($1 $2) failed"
      return 1
fi

return 0
```



```
[user1@okuda Proj]$ sleep 5000 &
[1] 5149
[user1@okuda Proj]$ . ./killp.sh 2 5149
DEBUG: kill_a_process(): KILLSIG=2 PID=5149
  PID TTY          TIME CMD
 5149 pts/0    00:00:00 sleep
INFO: kill_a_process(): Process 5149 has terminated
[1]+  Interrupt               sleep 5000
[user1@okuda Proj]$ . ./killp.sh 2 1
DEBUG: kill_a_process(): KILLSIG=2 PID=1
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
bash: kill: (1) - Operation not permitted
ERROR: kill_a_process(): kill -2 1 failed!
ERROR: a call to kill_a_process(2 1) failed
```