

Instructions

Complete the problems assigned below and turn in your answers by the due time above. To receive full credit, you must show both the relevant command(s) and output. If a question asks about what a command does, or what the output would be, give both the command and the output. Demonstrate that you actually performed the command, and didn't just guess.

Requirements - All of your shell programs henceforth must:

- a. use the shell template
- b. check for valid arguments
- c. output sensible *usage* or syntax messages
- d. be appropriately commented
- e. exit with an appropriate exit status

Homework Problem

1. Create a graphical kill process program **gkill** that presents the current user's list of running processes. The program allows the user to interactively select from this list of processes those that are to be marked for later termination. The program must meet the following requirements:
 - a. The program outputs the user's processes, one per line. Each line should minimally show the process **name**, **tty**, and **PID** of the process.
 - b. The program repeatedly accepts the following user input (upper and lowercase for letters) and correctly handles bad input:
 - i. **1** to **N** toggles the selection of a line (which represents a single process). *N* is the number of processes the current user has (with the limitation mentioned below). For example, if 4 is entered, then line 4 of the process list should be selected (to be killed). Selecting item 4 again will de-select the process so that it will not be marked for termination. Number each process is to be numbered, starting at 1 and ending at *N* (see examples below).
 - ii. **K** kills all of the currently selected process(es) and quits the program
 - iii. **Q** quits the program without killing any processes
 - c. Each time an item is selected, the program must redraw the screen, presenting the list with the selected items highlighted.
 - d. The program must not present itself in the list of processes to be killed (notice there is no **gkill** process in the examples below).
 - e. Run **ps** only once, and do *not* pipe the output of **ps** (this will have undesired affects).
 - f. Use the /tmp filesystem for any temporary files (no guarantee that the current directory is writable).
 - g. Use \$\$ as part of the filename for any temporary files – don't clobber other user's temporary files.
 - h. Clean up all temporary files.

Examples

Example output from the **gkill** program might be:

```
[1]: bash on pts/2 (pid: 2214)
[2]: longjob on pts/2 (pid: 2320)
[3]: sleep on pts/2 (pid: 2321)
[4]: cat on pts/2 (pid: 2329)
[5]: vim on pts/2 (pid: 2331)
Enter command: [1 - 5 selects; Q - aborts, K - Kills selected] ?
```

If the user enters **4**, the program should select line 4 (the **cat** process) by highlighting it in reverse video:

```
[1]: bash on pts/2 (pid: 2214)
[2]: longjob on pts/2 (pid: 2320)
[3]: sleep on pts/2 (pid: 2321)
[4]: cat on pts/2 (pid: 2329)
[5]: vim on pts/2 (pid: 2331)
Enter command: [1 - 5 selects; Q - aborts, K - Kills selected] ?
```

Selecting **2**, the program will highlight the **longjob**:

```
[1]: bash on pts/2 (pid: 2214)
[2]: longjob on pts/2 (pid: 2320)
[3]: sleep on pts/2 (pid: 2321)
[4]: cat on pts/2 (pid: 2329)
[5]: vim on pts/2 (pid: 2331)
Enter command: [1 - 5 selects; Q - aborts, K - Kills selected] ?
```

Selecting **4**, would deselect line 4:

```
[1]: bash on pts/2 (pid: 2214)
[2]: longjob on pts/2 (pid: 2320)
[3]: sleep on pts/2 (pid: 2321)
[4]: cat on pts/2 (pid: 2329)
[5]: vim on pts/2 (pid: 2331)
Enter command: [1 - 5 selects; Q - aborts, K - Kills selected] ?
```

And finally entering **K** will kill the processes from the selected lines (in this case, only line 2) and quit:

```
[1]: bash on pts/2 (pid: 2214)
[2]: longjob on pts/2 (pid: 2320)
[3]: sleep on pts/2 (pid: 2321)
[4]: cat on pts/2 (pid: 2329)
[5]: vim on pts/2 (pid: 2331)
Enter command: [1 - 5 selects; Q - aborts, K - Kills selected] ?
killing 2320
[1] Killed longjob
```

Hints

- Use **ps -u \$LOGNAME** to obtain the user's current processes
- Use the **clear** command to clear the screen each time through the loop so that the list is drawn at the top of the screen.
- You can highlight text by outputting the characters that are returned from the command **tput smso** and disable highlighting with **tput rmso**
- Use **kill -9** to kill the process(es).
- Consider creating and using the following functions: **getprocs**, **showprocs**, **markprocs**, and **killmarked**
- Use a temporary file to hold the current process list and iterate over every line in the file. Each line might contain the processes *cmd*, *tty*, and *pid*.
- Consider some technique for marking each selected process in the temporary file. All marked processes will be highlighted when shown by **showprocs**.
- Create dummy processes by running, for example, **cat &** without any arguments. There other ways as well to generate processes that will not go away – consider **sleep 500**.

The following solution is for instructive purposes only, and is *not* intended to imply that this is the only solution. There are many possible implements - the larger the problem, the more possible impenetations there are. Review the solution below and compare it to yours. See if you can understand each line, why I chose to implement the solution the way I did, and if you have better solutions. Post any questions and comments in the Forum please.

```
$ cat gkill
#!/bin/sh

# gkill
#
# Interactively shows the current user's processes and allows user to
# select processes to be killed.
#
# Supported Options
#     none
#
# Error codes: 0 = success; non-zero otherwise

prog=`basename $0`
usage="usage: $prog"

if [ $# -ne 0 ] ; then
    echo $prog: too many arguments
    echo $usage
    exit 1
fi

# Save the characters for hilighting
hiOn=`tput smso`
hiOff=`tput rmso`

# ps output fmt
#   PID TTY          TIME CMD
#  3144 pts/3        0:01 bash
#  8724 pts/3        0:02 vim

procfile=/tmp/$0.$$_1
tmpfile2=/tmp/$0.$$_2
nprocs=0

# Create a temporary file containing the users process information.
# The information given to us by ps -u is satisfisfactory enough.
# The process information file will always contain the correct process
# information. Each line consists of the following four fields: PID, TTY,
# TIME, and CMD; an optional word KILL will be added as the 5th firled
# to be used as the selection marker (a null value means no-kill).
# The KILL word will be added when a process is selected to be
# killed. Later, the killprocs function will kill any lines marked with KILL
getprocs() {
    ps -u $LOGNAME > $procfile
    # read and output each line, except the ps header and own process.
    # Doing this in a read loop makes matching our own PID easier, since
    # the words are separated and assigned automatically by read.
    while read pid tty time cmd ; do
        if [ $pid != PID -a $pid != $$ ] ; then
            echo $pid $tty $time $cmd
        fi
    done < $procfile > $tmpfile2
    mv -f $tmpfile2 $procfile
    nprocs=`cat $procfile | wc -l`
}

# Toggles the KILL marker on the line number specified by the function's
# single argument $1 (which is an integer line number > 0).
markprocs() {
    i=1;
```

```

while read pid tty time cmd selected ; do
    if [ $i = $1 ] ; then
        # toggle the selection
        if [ "$selected" = KILL ] ; then
            selected=
        else
            selected=KILL
        fi
    fi
    i=`expr $i + 1`
    echo $pid $tty $time $cmd $selected
done < $procfile > $tmpfile2
mv -f $tmpfile2 $procfile
}

# Kills the proceses marked with KILL - rather than call kill several times,
# the to-be-killed PIDs are grouped into a list, and this list is given
# as arguments to the kill command. Note that the PIDs cannot be assigned
# to a variable inside the loop since a redirected loop is run in a sub-shell.
killmarked() {
    i=1;
    echo > $tmpfile2
    while read pid tty time cmd selected ; do
        if [ "$selected" = KILL ] ; then
            echo $pid >> $tmpfile2
        fi
    done < $procfile
    pids=`cat $tmpfile2`
    echo killing $pids
    kill -9 $pids
}

# Outputs the processes to STDOUT, highlighting those marked by KILL
showprocs() {
    i=1;
    clear
    while read pid tty time cmd selected ; do

        if [ "$selected" = KILL ] ; then
            echo "$hiOn[$i]: $cmd on $tty (pid: $pid)$hiOff"
        else
            echo "[${i}]: $cmd on $tty (pid: $pid)"
        fi

        i=`expr $i + 1`

    done < $procfile
}

# The work starts here. Get the processes, continuing showing them, and
# marking them, based on user selection. When requested by the user, kill
# the marked procs and exit, or just exit.
getprocs

while : ; do
    showprocs
    echo 'Enter command: [1 -'$nprocs' selects; Q - aborts, K - Kills selected ] ? \c'
    read input
    case "$input" in
        q|Q) break;;
        k|K) killmarked ; break;;
        *) echo $input | grep '[^0-9]' > /dev/null
           if [ $? -eq 0 ] ; then
               echo "Invalid input - please select a line number."
               sleep 2
           elif [ $input -gt $nprocs -o $input -le 0 ] ; then
               echo "Invalid selection: line $input does not exist. Please make another
selection."

```

```
        sleep 2
    else
        markprocs $input
    fi
;;
esac
done

rm -f $procfile $tmpfile2

exit 0
```