

Санкт-Петербургский Государственный Политехнический Университет  
Кафедра «Прикладная математика»

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

По курсу “Операционные системы”

Выполнил студент гр. 4057/2:

Зенцев Ф.К.

Преподаватель:

Тимофеев Д.А.

Санкт-Петербург 2010

**Формулировка задания (1.6):** Напишите скрипт, управляющий списком текущих дел. Скрипт должен хранить список дел в файле `.todo` в текущем каталоге и предоставлять возможность добавления задачи, удаления задачи, отметки задачи как выполненной и просмотра всех задач. Режим работы задается параметром командной строки. Предусмотрите также параметр, при использовании которого скрипт будет работать с файлом `.todo` в домашнем каталоге пользователя независимо от наличия файла `.todo` в текущем каталоге. Все скрипты должны выдавать справку о своем назначении и поддерживаемых параметрах командной строки. Справка печатается на экран, если среди параметров есть ключ «`-help`» или «`-h`», никаких других действий в этом случае выполнять не нужно.

**Реализация:** На языке оболочки BASH. Первая реализация скрипта существенно отличалась от приведенной ниже. Для удаления задания, добавления нового, а также отметки выполненного задания использовались AWK и SED. Для подсчета строк использовались WC и TR, также возможно использование WC и CUT. Для вывода списка задания - CAT. Камнем преткновения стало то, что мне удалось реализовать задуманную пометку выполненного задания - дописывание в конец строки слова (`done`), приведенные средства всегда добавляли слово после перевода строки. В итоге я решил отказаться вообще от использования специальных команд, а реализовать задание на «чистом» BASH.

В начале работы скрипта файл загружается в массив построчно, в конце - этот массив сбрасывается в файл (`.todo` или `~/todo` в зависимости от режима работы скрипта). Ясно, что при таком подходе, во-первых, нет нужды хранить номера заданий - их можно просто дописать при выводе массива в файл (также отпадает необходимость перенумеровывать задания идущие вслед за удаляемым), а во-вторых все необходимые функции реализуются наиболее прозрачно. Добавить строку (`done`)? Просто сконкатенировать две строки. Как удалить конкретное задание? Выполнить команду UNSET, а затем учесть это при выводе. Вывести на экран? Тривиальный цикл.

## Исходный код скрипта

Listing 1: taskmgr.sh

```
1 #!/bin/bash
2
3 # TODO: Comment the code
4
5 # Global help message variable
6 HELP_MSG="Usage: ./taskmgr.sh [-h] [-g] [-a|-r|-d|-l] \nOptions: \n
7 -g, --global \t Work with ~/todo file , otherwise .todo file in current directory will be
   used \n\n
8 -a, --add \t Add new task \n
9 -r, --remove \t Remove task from the list \n
10 -d, --done \t Mark certain task as 'done' \n
11 -l, --list \t Display present tasks \n
12 -h, --help \t Display this message \n"
13
14 args=("$@")
15
16 declare file_name
17 declare line_count
18 declare -a file_array
19
20 function readFile()
21 {
22     let count=0
23
24     while read LINE; do
25
26         file_array[$count]=$LINE
27         ((count++))
28     done <$file_name
29
30     line_count=${#file_array[@]}
31 }
32
33 # Find argument in array function
```

```

34 function _findArg()
35 {
36     if [ -z "$1" ]; then
37         return
38     fi
39
40     for i in ${args[@]}
41     do
42         if [ $i == $1 ]; then
43             return 1
44         fi
45     done
46
47     return 0
48 }
49
50 function findParameter()
51 {
52     _findArg $1
53     local ret_1=$?
54
55     _findArg $2
56     local ret_2=$?
57
58     if [ $ret_1 -ne 0 ] || [ $ret_2 -ne 0 ]; then
59         return 1
60     else
61         return 0
62     fi
63 }
64
65 # Check which file should be modified
66 function determineFile()
67 {
68     findParameter -g --global
69     local ret_value=$?
70
71     if [ $ret_value -ne 0 ]; then
72         file_name=$HOME"/.todo"
73     else
74         file_name=".todo"
75     fi
76
77     if [ ! -e $file_name ]; then
78         touch $file_name
79     fi
80
81     readFile
82 }
83
84 function addTask()
85 {
86     local task_message
87     echo -e "Write_task_message:"
88     read task_message
89
90     file_array[$line_count]=$task_message
91     ((line_count++))
92 }
93
94 # Tests whether *entire string* is numerical.
95 # In other words, tests for integer variable.
96 function isDigit()
97 {
98     [ $# -eq 1 ] || return 0
99
100     case $1 in
101         *[!0-9]*|") return 0;;
102         *) return 1;;
103     esac
104 }

```

```

105
106 function deleteTask()
107 {
108     listTasks
109     echo -e "What_task_will_be_removed?(number)"
110     read number
111
112     isDigit $number
113
114     if [ $? -eq 0 ]; then
115         echo "Invalid_number"
116         return
117     else
118         if [ $number -gt $line_count ] || [ $number -eq 0 ]; then
119             echo "Invalid_number"
120             return
121         fi
122
123         ((number--))
124         unset file_array[$number]
125     fi
126 }
127
128 function markTaskDone()
129 {
130     listTasks
131     echo -e "What_task_is_done?(number)"
132     read number
133
134     isDigit $number
135
136     if [ $? -eq 0 ]; then
137         echo "Invalid_number"
138         return
139     else
140         if [ $number -gt $line_count ] || [ $number -eq 0 ]; then
141             echo "Invalid_number"
142             return
143         fi
144
145         ((number--))
146
147         if [[ ${file_array[$number]} =~ done ]]; then
148             echo "Task_#$(($number+1))_has_been_already_marked_as_done"
149             return
150         fi
151     fi
152
153     file_array[$number]+="_(done)"
154 }
155
156 function listTasks()
157 {
158     for ((i=0; i<$line_count; i++)); do
159         if [ $# -eq 0 ]; then
160             echo "${i+1})" "${file_array[$i]}"
161         else
162             if [ -z "${file_array[$i]}" ]; then
163                 continue
164             else
165                 echo "${file_array[$i]}"
166             fi
167         fi
168     done
169 }
170
171 function processArguments()
172 {
173     determineFile
174
175     local command

```

```

176
177 #TODO: check -g order
178 if [ -z ${args[1]} ]; then
179     command=${args[0]}
180 else
181     command=${args[1]}
182 fi
183
184 case "$command" in
185     -a|--add)
186         addTask
187         ;;
188     -l|--list)
189         listTasks
190         ;;
191     -r|--remove)
192         deleteTask
193         ;;
194     -d|--done)
195         markTaskDone
196         ;;
197     *)
198         echo -e "Invalid_arguments\n"$HELP_MSG
199         ;;
200 esac
201 }
202
203 # Entry point for the script
204 function work()
205 {
206     if [ ${#args[@]} -eq 0 ]; then
207         echo -e "No_argument_provided\n"$HELP_MSG
208         return
209     fi
210
211     if [ ${#args[@]} -eq 1 ]; then
212         if [ ${args[0]} == "-g" ] || [ ${args[0]} == "--global" ]; then
213             echo -e "No_command_provided\n"$HELP_MSG
214             return
215         fi
216     fi
217
218     # If help argument provided, nothing more is required
219     findParameter -h --help
220     return_val=$?
221
222     if [ $return_val -ne 0 ]; then
223         echo -e $HELP_MSG
224     else
225         # If not, going to process another arguments
226         processArguments
227     fi
228
229     if [ -z "$file_name" ]; then
230         return
231     else
232         listTasks "fake_param" > $file_name
233     fi
234 }
235
236 # Begin to work
237 work

```