

## ***Laboratory works on OS Unix (Linux)***

### Laboratory work No 1

#### ***Introduction to Unix***

##### *Basic items:*

1. Beginning/finishing work with system. Users, groups.
2. Files and directories. Protection of files (directories).
3. Process control.
4. Editing text.
5. Redirection of input/output. Pipes, filters.

##### *Basic commands of Unix:*

- ✓ Users and groups: login, exit, who, newgrp, man.
- ✓ Files and directories: ls, cat, cp, mv, rm, pwd, cd, mkdir, rmdir, find (file, ln, diff, comm, cmp).
- ✓ Protection: chmod, chown, chgrp, umask.
- ✓ Editing files: ed, vi.
- ✓ Processes: ps, &, kill, nice, at, nohup, renice.
- ✓ Filters: wc, sort, grep, sed, cut, paste, pr (awk).

##### *Task:*

1. Execute typical task "Introduction to Unix".
2. Create a command that extends functionality of Unix (according to your variant). Use a pipe.

##### *Variants of task (item 2):*

- 1) lx – list of executable files of a specified directory.
- 2) pu – list of processes started by a specified user.
- 3) nx – number of executable files in a specified directory.
- 4) npu – number of processes started by a specified user.
- 5) np – number of processes running a specified file.
- 6) mp – list of users who started a specified (or greater) number of processes.
- 7) kn – delete all processes running a specified command.
- 8) bp – print information on a specified number of processes having the greatest time of running on processor.
- 9) bf – print information on a specified number of files having the greatest size.
- 10) kp – delete all stopped processes of a specified user.
- 11) ml – print information on a specified number of files having the greatest number of links.
- 12) ll – list of users who are owners of files in a specified directory.

## Typical task "Introduction to Unix"

### A. Files and directories

1. Create the following structure of directories and files. Give Your name to the top directory. This directory will be your home directory.

```

[Your name]
!
vespa      !      vaz volvo
ford daf   !      citroen
Mashini
!
-----!-----
!              !
Razni          Francuz
```

2. Stay in Your home directory.
3. Move *ford* to directory *Mashini*.
4. Move all files, which names begin from “v” to directory *Razni*.
5. Move *vaz* to directory *Mashini*.
6. Stay in directory *Mashini*.
7. Copy *daf* to this directory.
8. Stay in directory *Razni*.
9. Move *vespa* to directory *Francuz* and rename it into *vespa2*.
10. Create three files with arbitrary names.
11. Go to directory *Mashini*.
12. Copy two files from *Razni* to *Francuz* using only one command.
13. Stay in Your home directory.
14. Show on the screen all levels under Your home directory.

### B. Protection of files

1. Stay in Your home directory and execute the command

```
ls -al > LIST
```

which creates a file *LIST* showing initial protection of files.

2. Let files *.profile* and *fruits* and directory *practice* be situated in Your home directory. They will be used further.
3. Change protection of file *.profile* that no other category of users except you can read the file content. Check the result.
4. Change protection of file *fruits* that only You or members of Your group can read file and write to it but all other users cannot do it.
5. Check which group files of Your directory belong to. Check which group you belong to. Change using a command the group of a file. Check in file */etc/group* who belongs and who does not belong to Your group.
6. Cancel all access rights for all categories of users for directory *practice*. Execute command `ls -al practice`. Cancel Your own x-right. Execute command `ls -al practice`.

### C. Processes

1. Determine who works in system and what processes are started by each user. For each process determine its state and priority. Find processes unattached to no one terminal.
2. Try to delete processes started by other users.
3. Create a command *dummy* which incessantly prints into standard output stream its name. Start the command. Interrupt its running and delete the process.
4. Start process *dummy* in background mode. Redirect its output into a file. Find the corresponding process in the processes list. Determine its state. Delete it.
5. Do the same operations decreasing the process priority using command *nice*. Change priority of running process using command *renice*.
6. Start a few copies of the process *dummy* in background mode; use redirection of output into a dummy device. Find started processes in the processes list. Exit from the system. Enter the system. Check the processes list.
7. Do the same operations starting processes in uninterrupted mode *nohup*. Compare process states after repeated entering into the system. Delete started processes.
8. Enter commands of starting three dummy processes in various instances of time (for example, after 3, 5, and 7 minutes). Watch starting the processes then delete them. Do processes start after a user exits the system?

### D. Filters: sort

1. Create a file named *family* that contains surnames, names and birth dates of a few persons. Each line has the following form:

smith john    3/1/50

Separate name and birth date with a tabulation (<tab>).

2. Use command *sort* to sort the file in alphabetic order.
3. Use an option (key) of command *sort* to sort the file in reverse order.
4. Create another file named *work* which contains surnames, names and birth dates of a few colleagues (group mates) having the same format as file *family*. Include Your personal data into the file.
5. Sort both files simultaneously by command *sort*. Pay your attention that your name is printed twice. Use an option (key) of command *sort* to print your name only once.
6. Create sorted files from files *work* and *family*. Use the key “-o” or redirection arrow “>”. If You use “>” please choose a new file name for sorted file.
7. Try using the key “-m” of command *sort* to sort source files *work* and *family*. Then try to sort with the key “-m” already sorted files. Pay attention that this key does not sort files but merges sorted files.
8. Sort Your files by the alphabetic order of name.
9. If a few persons have the same surname the program do not sort them automatically according to names. To overcome it, insert the key “-2” which enlarges sorting field till the column number 3.
10. Sort files in the order of the birth month. Try it with the key “+2”. Look how it has been done for people born in October, November, December. Try doing it with the key “+2n”.

### E. Filters: grep

1. Create a file named price.veg with the following data

Lettuce	.89
tomatoes	0.89
broccoli	.79
cauliflower	0.89
parsely	2.14
avocado	1.19
carrots	0.69
ctltry	0.59
string-beans	1.29
onions	0.29
asparagus	1.39
corn	0.85
cabbage	0.99
lima-beans	1.29
etlflow-beans	1.17
mushrooms	1.09
peppers	1.39
potatoes	0.39
artichokes	1.59
black-beans	1.89
blacheyed-beans	1.49

2. Sort the file twice: once in alphabetic order and twice according to prices. Write results into two new files.

3. Separate all beans in a file.

4. Find vegetables which names start with letter "s".

5. Find vegetables which names start with a vowel.

6. Find vegetables which price ends with digit "9".

7. Find vegetables which price ends with digits "09", "39", or "79".

8. Find vegetables which price does not end with digit "9".

9. Find vegetables which names do not start with letters from "a" to "m".

10. Find vegetables which price ends with digit "9" but does not end with "39", "59", "79", and "89".

### Laboratory work No 2

#### ***Unix: Programming in Shell language***

##### ***Basic items:***

1. Structure of *shell* command line, interpretation of meta-symbols, patterns.
2. Variables of *shell*, built-in variables.

3. Arguments and options of commands; return codes; substitution of results of a command execution.
4. Control of operations sequences: branching and loops. Checking conditions.
5. Subroutines and signals processing in shell scripts.

*Basic constructs of Shell language:*

Meta-symbols:

```
> >> < <<s <<\s <<'s' |
. * ? [abc] [d-f]
; (command) {commands} & command1&&command2 command1||command2
'string' "string" `command`
```

Variables and parameters:

```
variable=value read variable echo $variable
$variable ${variable}
${variable-value} ${variable=value}
${variable?string} ${variable+value}
$1 $2 etc
shift
set argument1 argument2 etc
```

Built-in variables:

```
$# $* @$ $- $? $$ $!
$HOME $PATH $TERM $MAIL $PS1 $PS2 $IFS
```

Control of operations sequence:

```
if condition then commands else commands fi
if condition then commands fi
case string in pattern1) commands ;; etc esac
for variable in values do commands done
while condition do commands done
until conditions do команды done
exec file
break continue exit
```

Testing files, numerical and symbolic values: test parameters

Processing signals: trap commands signals; basic signals: 1 2 9 15

Arithmetic expressions: expr expression

Keys of Shell: -x -v etc

*Task:*

1. Develop a program in Shell language (script) according to a variant.
2. Test script work.

*Variants of task:*

1. Track users who started more than specified number of processes.
2. Track changing the number of file links in the specified sub-tree of directories.

3. Track users who started more than specified number of processes from one terminal.
4. Track changing access rights in the specified sub-tree of directories.
5. Track creating new files/directories in the specified sub-tree of directories.
6. Track changing free disk space for a specified disk.
7. Track changing access rights in in the specified sub-tree of directories.
8. Track login/logout of users into the system from specified terminal printing the time of user work.
9. Track deletion of files/directories in the specified sub-tree of directories.
10. Track changing processes priorities in specified range.
11. Track the list of terminals attached to processes with specified state.
12. Track processes which used the amount of processor time exceeding the specified value.

*Guidelines to implementation of the tasks:*

- ✓ Implement tracking via periodic (after 1-5 minutes) repeating of certain actions and comparing the current state with the previous state saved in an auxiliary temporary file.
- ✓ For periodic start use either command *sleep* or *at*.
- ✓ Create auxiliary temporary files in directory */tmp*, provide unique names of temporary files (using built-in variable *\$\$*).
- ✓ Provide correct completion of the process (cleaning temporary files etc) on its deletion via processing of corresponding signals.
- ✓ The source of additional information – man-pages of shell (bash) and commands.

