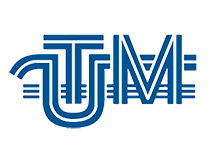
****

*Technical University of Moldova*

**Cybersecurity**

Lab 2

**Submitted by**:Emil Terman *FAF 161*

**Professor**: Cazac M.

**2019**

**Ex 1**

**How to include the result of the command into a file?**

To write the output of a command to a file, we can use I/O redirection. We can use ‘**>**’ or ‘**>>**’. The first one redirects the **stdout** to the destination file, overwriting the contents if it already exists. The second one, appends the output. In both cases, the file is created if it doesn’t exist.

To redirect both **stdout** and **stderr**, we can use **2>&1**:

echo ‘test str’ > my\_file 2>&1

**Ex 2**

**How to compile a program from within vi?**

To run a command directly from vim, we can use: `:!<command>`.

To compile a C file, the **gcc** compiler can be used.

To compile a file directly from vim, we run: `:!gcc %`.

% - refers to the current file.

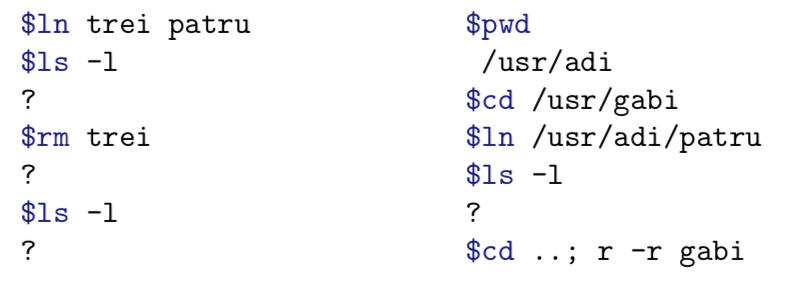
**Ex 3**

**How to quickly edit and run a file?**

1. $ echo ‘echo test’ >> my\_script.sh && sh my\_script.sh
2. From vim, we can run: `:!sh %` or `:!gcc % -o a.out && ./a.out`

**Ex 4**

**In current directory there are 3 files: unu, doi, trei. Explain the result of the commands below. What is the difference between ln and cp?**

****

1. **$**ln trei patru
2. **$**ls -l

We should have 4 files: **unu**, **doi**, **trei** and **patru**, where **trei** and **patru** are hard links with the same inode.

1. **$**rm trei
2. **$**ls -l

We should have 3 files: **unu**, **doi**, **patru**. This time, all files have different inodes.

1. **$**pwd

/usr/adi

1. **$**cd /usr/gabi
2. **$**ln /usr/adi/patru
3. **$**ls -l

We should see the file **patru** with 2 hard links.

**/usr/gabi/patru** and **/usr/adi/patru** have the same inode

* **`ln`** creates links. In our case, we created hard links.
* **`cp`** creates copies

Having file **f00** and **f01** - hard linked: modifying **f00** will “affect” **f01** as well.

Having file **f00** copied to **f01:** modifying **f00** will not affect **f01**.

**Ex 5**

**What do these commands do?**

1. **$**ls -l m[a-df-z]\*

Will long list all files in the current directory that start with ‘m’ and is followed by 0 or more letters from [‘a’; ‘d’] ⋃ [‘f’; ‘z’]

1. **$**cp ../fis\* .

Copies all files from the parent directory that start with ‘fis’ into the current dir.

1. **$**echo \*[!0-9]

Prints all files from the current directory that don’t end with a number and are not hidden

1. **$**echo [A-Z]\*

Prints all files from the current directory that start with a Captial english letter and are not hidden

1. **$**mv \*.\* ./adi

Moves all non hidden files from the current directory containing a dot into ./adi

1. **$**rm \*.o ???

In the current directory, for the non hidden files:

* Removes the files that end with ‘.o’
* Removes the files that named with 3 letters

1. **$**cat f\*.c

In the current directory: print the contents of all the files that start with **‘f’** and end with **‘.c’**

1. **$**ln unu doi

Creates a hard link from unu to doi

**Ex 6**

**What do these ed commands do?**

1. / ... /

Find three characters long sequence surrounded by spaces from both sides.

1. 1,$s/^/ /

Adds 4 spaces to the beginning of each line.

1. 8,9s/da$/na/

In lines from 8 to 9 replace all ”da” that are at the end of the line with ”na”.

1. .,$s/\.$/ /p

In lines from the current line to the last line replace ”.” at the end of the line with space then print.

1. /^[A-Z]/

Search for line that has capital letter at the start of it.

1. 1,$-3s/[^a-zA-Z]//g

From the first to the end of the files - 3, remove every character that is not a lowercase or capital letter.

1. .,+5s/^.\{10\}//

From the current line to the 5 lines ahead remove the 10 characters at the start of line if this line is at least 10 characters long.

1. g/ee/s/^./#&/p

For all lines that contains ”ee”, add the ”” character to the start of the line.

**Ex 7**

**What do the following regular expressions mean?**

1. ^$

Empty line

1. e.\*e

Start and end with the letter ‘e’

1. []a-z]

Matches a lowercase letter from **‘a’** to **‘z’** or the symbol **‘]’**

1. [A-Za-z]\{4,7\}

Matches any lowercase or uppercase letter from 4 to 7 times. Ex: aaaa, AbCdE

1. .\{7,\}

Matches any character at least 7 times. Ex: abcdefg

1. ^\(.\)

Capture the first character from a line

1. ^\(.\).\*\1$

Capture the first character from a line that has at least 2 characters and the last one is ‘1’

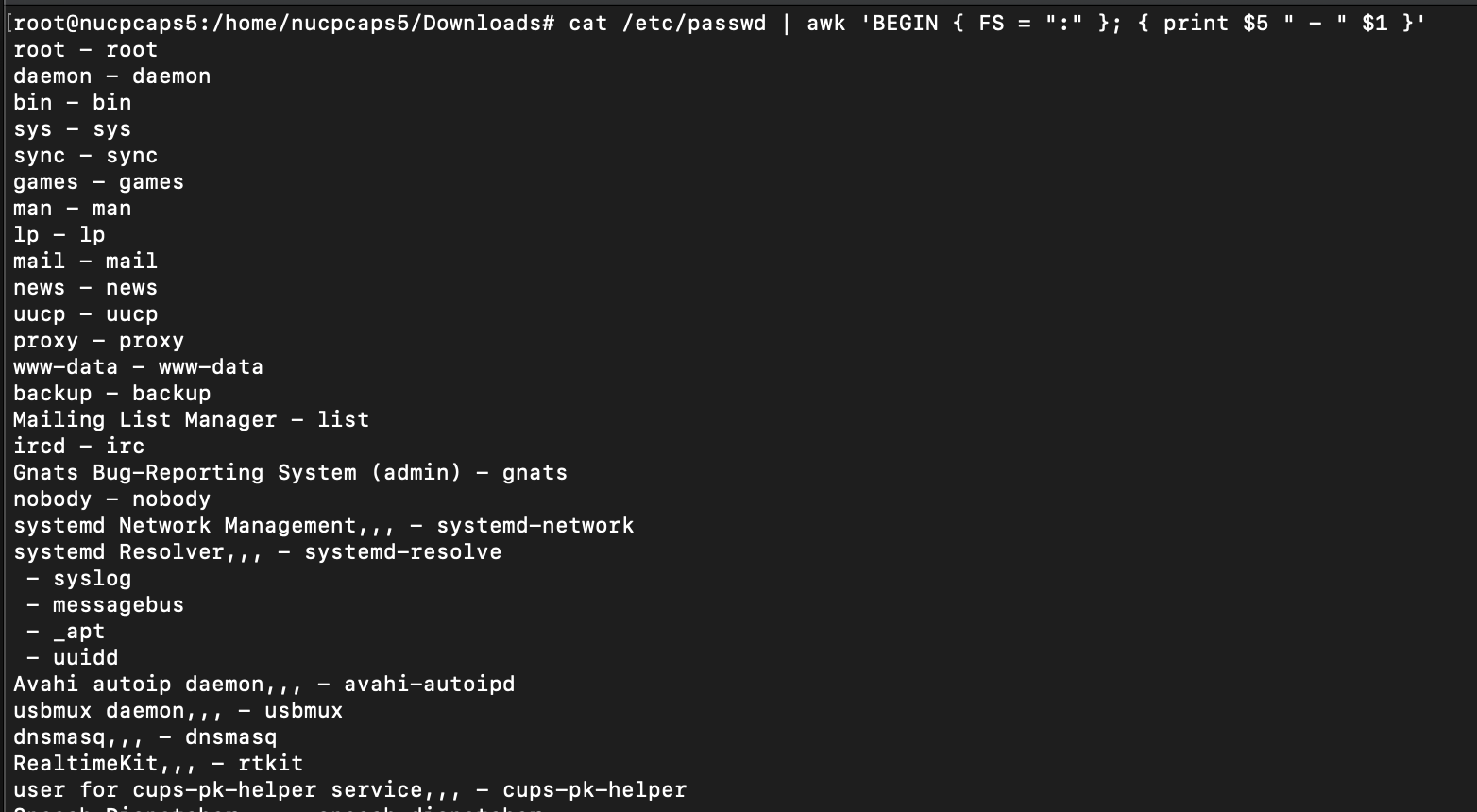
1. /[Tt]he/

“The” or “the”

**Ex 8**

**Using Unix commands and editor, acquire one file that contains every user from the system with his full name and login.**

**$**cat /etc/passwd | awk 'BEGIN { FS = ":" }; { print $5 " - " $1 }' > full\_name\_and\_login



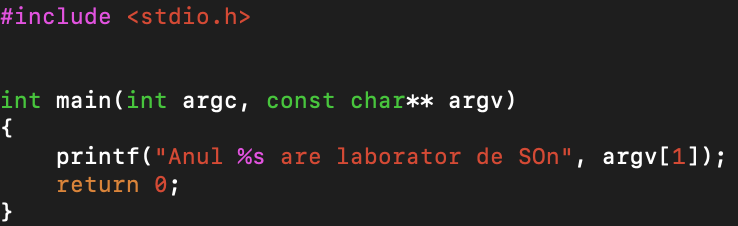
**Ex 9**

**Correct the program below using one of the studied editors.**

Initial version:



Fixed version, using VI:

****

**Ex 10**

**Knowing commands cp and mkdir write a sequence of commands that**

**copy one existent directory as a subdirectory of the current directory.**

$cp -r magic\_box/dir\_to\_copy .

**Ex 11**

**Write the program in C using vim, that works with matrices.**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

typedef **struct** s\_matrix

{

**float**\*\* tab;

**size\_t** width;

**size\_t** height;

} t\_matrix;

typedef **float** (\*t\_matrix\_operation)(**float**, **float**);

**int** output\_is\_being\_redirected()

{

return isatty(STDOUT\_FILENO) == 0;

}

**void** handle\_critical\_errno(**const** **char**\* reason)

{

if (!errno)

return;

perror(reason);

exit(1);

}

t\_matrix init\_matrix(**const** **size\_t** width, **const** **size\_t** height)

{

t\_matrix matrix;

matrix.width = width;

matrix.height = height;

matrix.tab = calloc(height, sizeof(matrix.tab));

handle\_critical\_errno("calloc");

for (**size\_t** i = 0; i < height; i++)

{

matrix.tab[i] = calloc(width, sizeof(\*matrix.tab));

handle\_critical\_errno("calloc");

}

return matrix;

}

**void** free\_matrix(t\_matrix\* matrix)

{

if (matrix->tab == NULL)

return;

for (**size\_t** i = 0; i < matrix->height; i++)

{

if (matrix->tab[i])

{

free(matrix->tab[i]);

matrix->tab[i] = NULL;

}

}

free(matrix->tab);

matrix->tab = NULL;

}

**void** read\_matrix\_tab(t\_matrix matrix)

{

for (**size\_t** i = 0; i < matrix.height; i++)

for (**size\_t** j = 0; j < matrix.width; j++)

scanf("%f", &matrix.tab[i][j]);

}

t\_matrix read\_matrix()

{

**size\_t** width;

**size\_t** height;

t\_matrix matrix;

if (!output\_is\_being\_redirected())

printf("Input the size of the matrix: Width x Height:\n");

scanf("%lu%lu", &width, &height);

matrix = init\_matrix(width, height);

read\_matrix\_tab(matrix);

return matrix;

}

**void** print\_matrix(**const** t\_matrix\* **const** matrix)

{

for (**size\_t** i = 0; i < matrix->height; i++)

{

for (**size\_t** j = 0; j < matrix->width; j++)

printf("% 10.2f", matrix->tab[i][j]);

putchar('\n');

}

}

t\_matrix apply\_operation(

**const** t\_matrix\* **const** matrix1,

**const** t\_matrix\* **const** matrix2,

**const** t\_matrix\_operation op)

{

t\_matrix result;

if (matrix1->width != matrix2->width || matrix1->height != matrix2->height)

{

fprintf(stderr, "Invalid matrix sizes to apply operation\n");

exit(1);

}

result = init\_matrix(matrix1->width, matrix1->height);

for (**size\_t** i = 0; i < matrix1->height; i++)

for (**size\_t** j = 0; j < matrix1->width; j++)

result.tab[i][j] = op(matrix1->tab[i][j], matrix2->tab[i][j]);

return result;

}

**float** add(**const** **float** a, **const** **float** b) { return a + b; }

**float** sub(**const** **float** a, **const** **float** b) { return a - b; }

**float** mul(**const** **float** a, **const** **float** b) { return a \* b; }

**float** divide(**const** **float** a, **const** **float** b) { return a / b; }

t\_matrix\_operation extract\_operation(**const** **int** argc, **const** **char**\*\* argv)

{

if (argc <= 1)

{

fprintf(stderr, "Operation not provided\n");

exit(1);

}

switch (argv[1][0])

{

case '+': return add;

case '-': return sub;

case '\*': return mul;

case '/': return divide;

}

fprintf(stderr, "Invalid operation\n");

exit(1);

}

**int** main(**const** **int** argc, **const** **char**\*\* argv)

{

t\_matrix matrix1;

t\_matrix matrix2;

t\_matrix result;

t\_matrix\_operation op;

op = extract\_operation(argc, argv);

matrix1 = read\_matrix();

matrix2 = read\_matrix();

result = apply\_operation(&matrix1, &matrix2, op);

print\_matrix(&result);

free\_matrix(&matrix1);

free\_matrix(&matrix2);

free\_matrix(&result);

return 0;

}