A simple file manager script

Creating scripts

First, let's understand how to properly create scripts in Linux.

To create a script, type the following command in the console,

```
$ nano file_name
```

Where file_name is the name of the script.

Then the file editor opens, where we will enter the code of our script.

My script looks as follows,

```
#!/bin/sh


if [ -z "$1" ]; then
echo "Error: you did not specify a directory as a parameter."
echo "Usage: $0 <path_to_directory>"
exit 1
fi


if [ ! -d "$1" ]; then
echo "Error: the specified directory '$1' does not exist."
exit 1
fi


if [ ! -w "$1" ]; then
echo "Error: you do not have write permissions to the '$1' directory."
exit 1
fi


cd "$1"
mkdir -p ValKun
cd ValKun
```

```
touch numbers.txt


shuf -i 1-100 -n 5 > "numbers.txt"


cat numbers.txt

echo "file with the generated numbers:"

sort -n numbers.txt >> sort_numbers.txt

echo "file sorted by ascending order:"

cat sort_numbers.txt

sort -n -r numbers.txt >> resort_numbers.txt

echo "down-sorted file:"

cat resort_numbers.txt
```

and then go through each action of my script.

Script description

- Always start writing a script with:

```
#!/bin/sh
```

is a **shebang**, a special line at the beginning of a script that tells the operating system what program to use to execute the script.

Piece by piece:

#! - shebang characters, indicating that they are followed by the path to the interpreter.

/bin/sh - path to the sh (Bourne Shell) command interpreter on the system.

After running the script, the system reads the first line, finds the sh and passes the script execution to it.

- The following lines will write my script and what it should do.

```
if [ -z "$1" ]; then

echo "Error: you did not specify a directory as a parameter."

echo "Usage: $0 <path_to_directory>"

exit 1

fi
```

This line uses the **if_then** construct to check whether a parameter has been passed when running the script. This construct should always be closed with a **fi** character, as this marks the end of the construct, and all subsequent code will not be included in **if_then**.

-z „$1” - checks whether the parameter $1 has been passed. If the parameter is missing, an error message is printed and the script terminates.

$0 – The first parameter entered.In this case,the name of the script.

echo " your text" - displays the text on the terminal.

exit 1 - terminates the execution of the script.

```
if [ ! -d "$1" ]; then
echo "Error: the specified directory '$1' does not exist."
exit 1
fi
```

This line uses the **if_then** construct to check whether the specified directory exists.

! -d "$1" - checks if the directory specified in the $1 parameter exists. If it does not exist, an error is printed.

```
if [ ! -w "$1" ]; then
echo "Error: you do not have write permissions to the '$1' directory."
exit 1
fi
```

This line uses the **if_then** construct to check whether the current user has write permissions to the $1 directory. If not, an error is printed.

```
cd "$1"
mkdir -p ValKun
cd ValKun
touch numbers.txt
```

cd - command to change the current directory (Change Directory).

$1 - The first argument passed to the script (positional parameter). It contains the path to the directory to which the script should go.

Important! In this case, $1 is in quotation marks, so in this parameter we are revealing the name of the directory and if there is a space in the name, the script will not work or will work, but not as we need it.

mkdir is a command that creates a directory (Make Directory).

-p is an important flag:

- Creates a directory if it does not yet exist.
- Does not cause an error if the directory already exists.
- Creates all parent directories if they do not exist.

ValKun - the name of the subdirectory to be created in the current directory (**$1**).

cd - command to change the directory again.

- **ValKun** - The script moves to the ValKun subdirectory created in the previous step.
- Moving to the directory is done relative to the current location.

touch - command to create a new empty file or update the time when an existing file was last modified.

- **numbers.txt** - the name of the file to be created in the current directory (**ValKun**).

```
shuf -i 1-100 -n 5 > "numbers.txt"
```

shuf to narzędzie wiersza poleceń systemu Linux, które

- Shuffles strings of input data,
- Generates random numbers from a specified range.

-i 1-100 - number range

- **-i** - flag, which tells shuf to work with a range of numbers (not strings from a file).
- **1-100** - number range from 1 to 100.

-n 5 - number of numbers to be written out

- **-n** is a flag specifying the number of numbers to be selected from the jumbled list.

- **5** is the number of numbers to choose.

> „numbers.txt” - redirect the output to a file

> - operator to redirect the standard output (stdout) to a file.

„numbers.txt” - the name of the file to which the result will be saved.

```
cat numbers.txt

echo "file with the generated numbers:"

sort -n numbers.txt >> sort_numbers.txt

echo "file sorted by ascending order:"

cat sort_numbers.txt

sort -n -r numbers.txt >> resort_numbers.txt

echo "down-sorted file:"

cat resort_numbers.txt
```

cat is a command that displays the contents of a file in the terminal (concatenates and displays).

```
sort -n numbers.txt >> sort_numbers.txt
```

sort - command to sort rows in a file.

-n - flag ordering numerical sorting (instead of lexicographic sorting).

numbers.txt - file with numbers to sort.

>> - append operator.

sort_numbers.txt is the file in which the result of sorting is stored.

- If the **sort_numbers.txt** file already exists, the new data will be added to the end of the file.
- If the file does not exist, it will be created automatically.

```
sort -n -r numbers.txt >> resort_numbers.txt
```

sort - string sorting command.

-n - numerical sorting.

-r - flag specifying sorting in descending order.

At this point, my script is fully written and ready to go. We can exit the editor.

Last steps

In order for our script to start working, we need to give it the proper permissions. To do this, we type the following line in the terminal:

```
chmod u+x script_name
```

chmod - change file access rights

- **chmod** is a command used to change the access rights to a file or directory in Unix/Linux.
- It stands for "change mode".

u - for whom the rights are changed

- **u** - means the user who owns the file (user).

- In addition to **u**, you can use
  - **g** - group
  - **o** - others
  - **a** – all

+x - jakie prawa są dodawane

- **+** - addition.
- **x** - execute.
- Instead of +, you can use:
  - - - take away.
  - = - sets exact rights (overwrites existing rights).

x (execute) makes the file executable - allows you to run it as a program or script.

After giving the proper permissions to our script, it's time to run it. This is done using the following line:

```
./script_name path_to_directory
```

. - current directory.

/ - separator between directory and file name.

script_name - the name of the executable file or script located in the current directory.

This is the end of my presentation. Thank you for your attention!