"Kyiv Professional College of Communication"

Cycle Commission of Computer Engineering

REPORT ON THE IMPLEMENTATION

LABORATORY WORK №2

in the discipline: "Operating systems"

Topic: «Introduction to basic CLI commands in Linux»

Performed by a student

of the group BICS-13

Los Zlata Volodymyrivna

Checked by the teacher

Sushanova V.S.

Kyiv 2024

**Objectives:**

1. Familiarization with the basic commands of the CLI mode in Linux.

2. Familiarity with basic text commands in terminal mode in different OS.

**Material support of classes:**

1. Computer such as IBM PC.

2. Windows operating system and Virtual Box (Oracle) virtual machine.

3. GNU/Linux OS (any distribution).

4. The Cisco Network Academy website netacad.com and its online Linux courses.

**Tasks for preliminary preparation.**

1. \*Read the brief theoretical information for the laboratory work and make a small dictionary of basic English terms on the purpose of commands and their parameters.

**A shell (or "command shell")** is not only a command prompt that provides an interface between the user and the operating system kernel, but also a kind of programming language that contains such constructs as conditional branching statements, loops, variables, and much more.

**A command** is the name of a program that a user enters in the terminal to perform a specific task. Commands can have additional data and parameters/options. Commands in Linux/Unix are case sensitive. The entered command is executed only after pressing the Enter key.

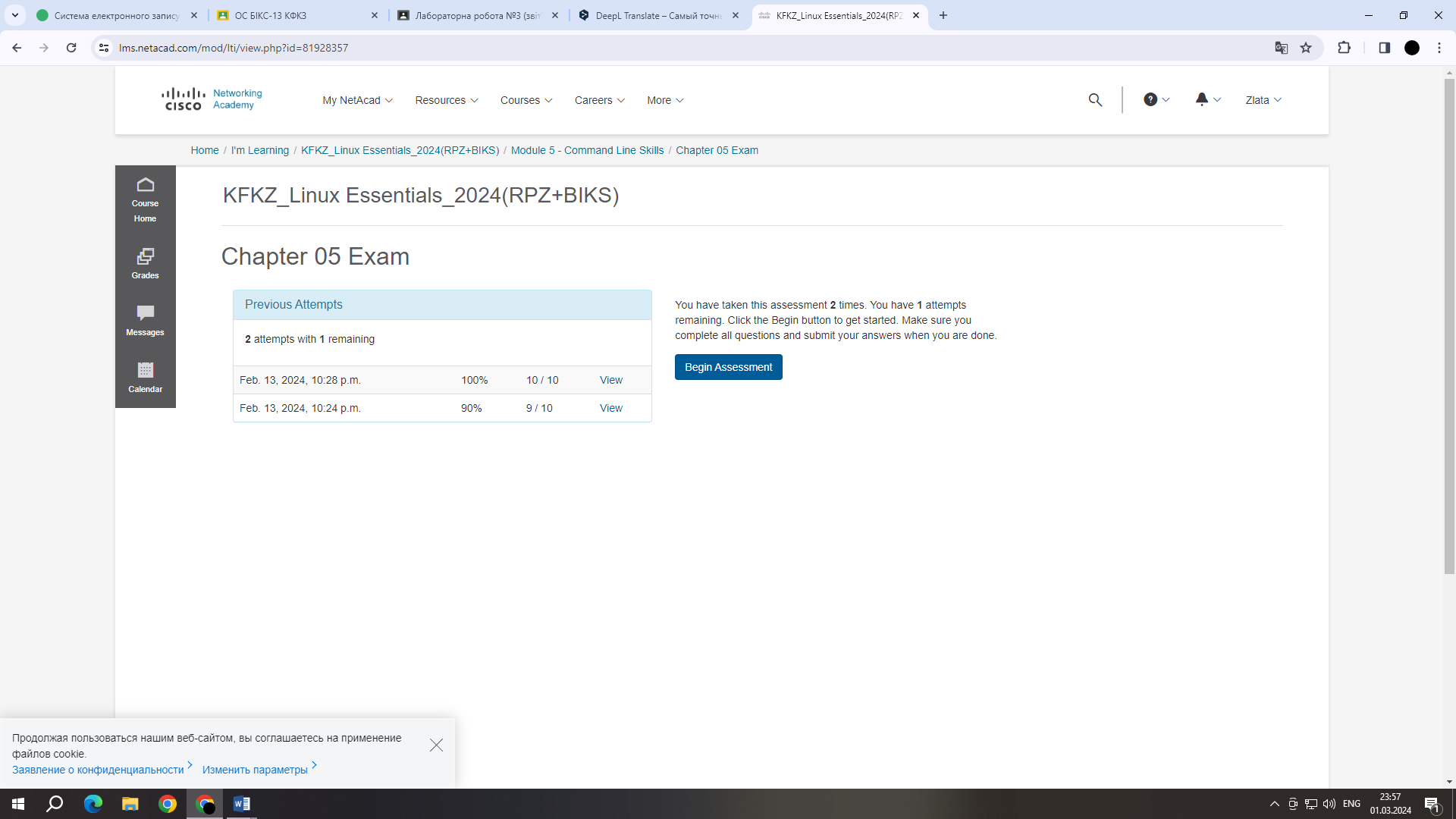
**Command line arguments** are parameters that are passed to the script during its execution in the bash shell. In Linux, they are also known as positional parameters.

2. Take the NDG Linux Essentials online course at Cisco Academy:

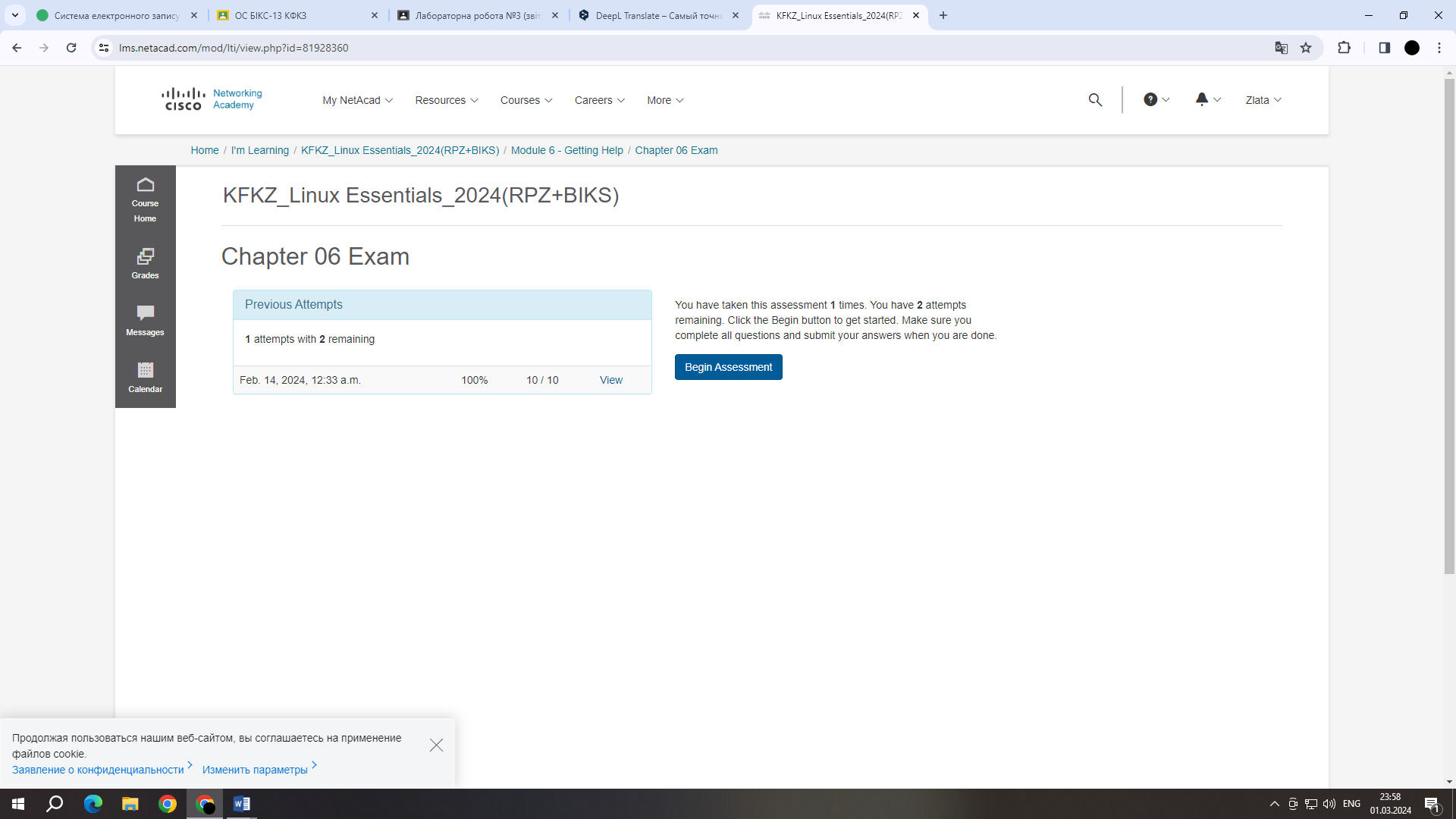
* Chapter 5 - Command Line Skills
* Chapter 6 - Getting Help

3. Take the NDG Linux Essentials course tests on the following topics:

- Chapter 05 Exam



- Chapter 06 Exam



**4. \* Define the following terms:**

- **A command prompt** is a program that provides user interaction with an operating system.

- **A shell** is a program that provides textual communication between the operating system and the user. It is usually a text-based program that reads user input and interprets it as commands to the system.

- **A command** is the name of a program that the user enters in the terminal to perform a specific task. Commands can have additional data and parameters/options.

**5. \*\*Answer the following questions:**

**- What basic information does the prompt prompt line provide?**

The prompt prompt provides the following basic information:

- The current user.

- The name of the host (computer).

- the current working directory.

- the prompt character: usually `$' for regular users and `#' for users with administrator privileges.

**- Why does the command need parameters and arguments?**

Parameters and arguments in commands are used to provide commands with additional information or instructions that they should execute. They allow users to exercise more flexible and precise control over the behavior of programs, scripts, or commands on the command line. Here are some key aspects of what parameters and arguments are for:

- Customization of actions: Allows the user to specify which action or set of actions to perform. For example, when copying files, you can specify where to copy from and where to copy to.

- Flexibility: Provide the ability to use the same command for different scenarios by changing its behavior with different parameters and arguments.

- Efficiency: Allows you to perform complex tasks with a single command instead of using a long sequence of commands.

- Accuracy: Parameters and arguments help to define precise criteria or objects for operations, reducing the likelihood of errors.

- Automation: In scripts or programs, they allow you to pass specific data or settings, making it possible to automate tasks without having to manually change the code.

- Versatility: Parameters and arguments allow a single command or program to serve many users with different needs, giving everyone the ability to customize it for their tasks.

**- What is the purpose of the ls command, what parameters and arguments can it have? Give 3 examples.**

The ls command is used to display a list of files and directories in the current directory or a specified directory. This command allows users to view the contents of directories, including file and folder information such as permissions, number of links, owner, group, size, and last modified time.

Parameters and arguments of the ls command:

The 'ls' command has numerous parameters and arguments that allow you to change its behavior or output format. Here are some of the most commonly used options:

- '-l': Displays a detailed list of files and directories, including permissions, number of links, owner, group, size, and last modified time.

- '-a': Lists all files and directories, including hidden files (anything starting with a dot, such as '.bashrc').

- '-h': Used with '-l' to display the size of files in an easy-to-read format (e.g., KB, MB).

**Examples of using the ls command:**

- Easily view the contents of a directory: 'ls'

Displays a list of files and folders in the current directory.

- Detailed view of the contents of the directory: 'ls -l'

Displays detailed information about files and folders, such as permissions, number of links, owner, size, and last modified time.

- View all files, including hidden ones, with detailed information and in a convenient format: 'ls -lah'

Combines the '-l' option for detailed viewing, '-a' for including hidden files, and '-h' for displaying file sizes in an easy-to-read format.

**- How can you use team history and what are the benefits?**

How to use team history:

- View command history: Use the 'history' command to display a list of the last commands you ran.

- Searching the history: You can search for specific commands by using the 'Ctrl + R' combination and typing the command snippet to search for.

- Using commands by number: Use '!number' to execute a command by its number in the history list.

- Repeat the last command: The '!!' command allows you to repeat the last command.

Advantages:

1. Efficiency.

- Quick access to previously executed commands: Instead of re-entering long or complex commands, you can quickly find and reuse them, which saves you a lot of time.

- Ability to edit commands: Easily modify previous commands for new tasks without having to enter them from scratch.

2. Explore and analyze

- Analysis of command history: Allows you to view commands executed during a session or over a specific period to better understand the operations performed.

- Learning new commands: Reviewing previously used commands can help you learn and memorize them.

3. Recover and fix errors

- Quickly fix errors: If a previous command contained an error, you can quickly access and fix it.

- Recovery from failures: If a session ends suddenly, you can easily find and repeat the necessary commands from the history.

**- What is the purpose of the echo command?**

The echo command is used to display a line of text to a standard output, which is usually a terminal. This command has a wide range of uses, from displaying simple messages to more complex scenarios such as passing values in shell scripts.

**- Describe the concept of a variable in the Bash shell, what types of variables does it support?**

In the Bash shell, a variable is a named object that is assigned a value. Variables are used to store and manipulate data in scripts and on the command line. The main characteristics of variables in the Bash shell include:

**Declaring a variable:**

- Variable name: A variable can have any name, but it must begin with a letter or underscore ' \_'.

- The variable name must not contain spaces.

**Assigning a value to the variable:**

- Use the equal sign '=' to assign a value to a variable.

- There are no spaces between the variable name, the equal sign, and the value.

`name="Zlata"

**Types of variables:**

- String variables: Store textual data. For example: `name="Zlata"`.

- Integer variables: Can hold integers. For example: `count=10`.

- Array variables (Arrays): Allow you to store multiple values in an array. For example: `numbers=(1 2 3)`.

- Variables with other shell objects (for example, commands or functions).

**Getting the value of a variable:**

- Use `$' before a variable name to get its value. For example: `echo $name`.

**Variables and quotes:**

- String values can be defined as either double `` or single `` quotes.

- Double quotes allow the use of special characters (e.g., variables), while single quotes are treated literally.

**- What is the purpose of the env, export, and unset commands?**

**The env command** allows you to run another program in the user environment without changing the current environment. If used without an argument, it displays a list of variables of the current environment.

**The export command** creates an environment variable.

**The unset command** deletes shell variables and environment variables.

**- What commands do you know to get help on commands in the terminal?**

**1. 'man' (man pages):**

Purpose: Used to display help pages for commands and programs.

Syntax: 'man command'.

Example: 'man ls' displays help information for the 'ls' command.

**2. '--help' or '-h' options:**

Purpose: Many commands support the '--help' or '-h' options, which print out brief help on how to use the command.

Syntax: 'command -help' or 'command -h'.

Example: 'ls -help' or 'ls -h' will display help information for the 'ls' command.

**3. 'info':**

Purpose: Some programs provide information through the 'info' system. 'info' allows you to view detailed help in text mode.

Syntax: 'info command'.

Example: 'info ls' displays information in text mode for the 'ls' command.

**6. Prepare an initial version of the report in electronic form:**

- Cover sheet, topic and purpose of the work

- Glossary of terms

- Answers to p.4 and p.5 of the preliminary preparation tasks

**Work in progress**

1. Work through all of the sample commands presented in the NDG Linux Essentials Lab 5: Command Line Skills and Lab 6: Getting Help. Create a table to describe these commands

|  |  |
| --- | --- |
| Team name | Its purpose and functionality |
| ls | Displays information about directories and files. By default, without arguments, displays information for the current directory |
| ls -l | Using the -l option in the ls command allows you to display information about files located in the current working directory in a long format that provides more extensive additional information |
| ls -l /tmp | Using the /tmp argument in conjunction with the -l option in the ls command allows you to display detailed information about the files in the /tmp directory. |
| cd | Changes the context of the current working directory of the shell. |
| cd .. | Moves the user up one level in the directory hierarchy. |
| cd /tmp | Moves the user to the /tmp directory, which is usually used for temporary files. |
| pwd | Displays the full path to the current working directory. |
| mkdir | Creates a new catalog. |
| rmdir | Deletes a directory. The directory must be empty to be deleted. |
| rmdir /tmp/newdir | Deletes the newdir directory in the /tmp directory if it is empty. |
| touch | Changes the last time a file was accessed and/or modified. If the file does not exist, creates an empty file. |
| touch file.txt | Creates or updates the time of the file.txt file. |
| cat file.txt | Displays the contents of the file.txt file on the screen. |
| more file.txt | Shows the contents of the file.txt file page by page. |
| head file.txt | Displays the first 10 lines of the file.txt file. The number of lines can be changed with the -n option. |
| tail file.txt | Prints the last 10 lines of the file.txt file. The number of lines can be changed with the -n option. |
| grep "pattern" file.txt | Searches for lines containing "pattern" in file.txt and prints them. |
| sort file.txt | Sorts the lines in file.txt and prints the result. |
| uniq file.txt | Outputs or stores only unique strings from the input file. Usually used after sort. |
| man ls | Displays the man page for the ls command, which is used to display the contents of a directory. |
| help | Displays help for built-in shell commands, for example, in Bash or another shell. Using a command with a name, such as help cd, will provide information about the cd command. |
| mkdir -p /tmp/newdir | Creates the newdir directory in the /tmp directory, including all necessary parent directories (-p option). If the directories already exist, no error is displayed. |

**Control questions**

**1. What types of commands exist in the Bash shell?**

*1.1 Built-in commands:*

Built-in commands are handled directly by the Bash shell itself. They execute faster than external commands because you do not need to create a new process. Examples of built-in commands include 'cd' (change directory), 'history' (history of commands), 'echo' (emit text), and 'export' (export environment variables).

*1.2 External commands:*

External commands are programs that reside outside of the Bash shell, usually located in standard directories such as '/bin', '/usr/bin', '/sbin', '/usr/sbin', etc. When you call an external command, Bash creates a new process to execute it. Examples of external commands include 'ls' (listing files), 'grep' (searching text), and 'wget' (downloading files from the network).

*1.3 Aliases:*

Aliases allow users to define abbreviations for long commands or groups of commands. An alias is created with the 'alias' command and is only valid in the current shell session unless it is added to a configuration file, such as '.bashrc. For example, 'alias ll='ls -la' creates an alias 'll' for the command 'ls -la'.

*1.4 Shell Functions:*

Shell functions allow users to define more complex commands using combinations of existing commands. Functions behave like mini-scripts within the current shell and can take arguments and return values. They are declared using the syntax 'function\_name() { commands; }'.

*1.5 Keywords:*

Keywords are words that have a special meaning in Bash. They control the behavior of the shell and are used to define conditional statements, loops, and other control structures. Examples of keywords include 'if', 'else', 'fi', 'while', 'do', 'done', 'for', 'in', 'case', 'esac', 'function'.

*1.6 Checking the type of command:*

To determine the type of a particular command in Bash, you can use the 'type' command. For example, 'type cd' will show that 'cd' is a built-in shell command*.*

**2. What are environmental variables? What are they? How can they be viewed in the terminal?**

Environment variables in operating systems are special variables that store information about the user's and the system's working environment. These variables can include paths to important directories, details about the current user, information about network usage, and so on. They are used by both system processes and user-run programs to obtain the configuration and properties of the environment.

Environmental variables can be classified into two main categories:

*- System environment variables:* These are global variables that are defined for the entire system and all users. They include paths to system directories, OS versions, system details, etc. Examples of system environment variables include 'PATH', 'HOME', 'OS', 'USER'.

*- User environment variables:* These variables are defined at the individual user level and can be customized to change the behavior or configuration of applications for a specific user. For example, a user can define an environment variable to store the path to their scripts or configuration files.

There are several ways to view environment variables:

- printenv: The 'printenv' command displays a list of all environment variables along with their values. If you want to view a specific variable, you can use 'printenv VARIABLE\_NAME', where 'VARIABLE\_NAME' is the name of the variable you want to view.

- env: The 'env' command also displays all environment variables, but in addition it can be used to run the program in a modified environment.

- echo: To view a specific environment variable, you can use the 'echo' command along with the variable name, for example, 'echo $HOME' will display the current user's home directory.

- set: In Bash, the 'set' command is used to display a list of all variables (including environment variables) and functions available in the current shell session. It displays more information than 'env' or 'printenv' because it includes not only environment variables but also local shell variables.

**3. \*Describe the variable $PS1. How to view its contents in the terminal?**

The '$PS1' (Prompt String 1) variable is one of the built-in variables of the Bash shell that defines the basic format of the command line prompt that is displayed to users before entering each command. The '$PS1' variable allows you to customize the appearance of the command prompt to increase its information content and usability, including such data as username, hostname, current working directory, and more.

*Contents of the $PS1 variable*

The $PS1 variable can contain plain text and special escape sequences that are interpreted by the shell. Here are some of the most common escape sequences:

**- '\u'** - The name of the current user.

**- '\h'** - Part of the hostname up to the first dot.

**- '\W'** - Current working directory (only the name of the directory, no path).

**- '\w'** - The full path of the current working directory.

**- '\d'** - Date in the format "day of week, month, day".

**- '\t'** - Current time in 24-hour format HH:MM:SS.

**- '\n'** - New line.

**- '\s'** - The name of the shell.

**- '\#'** - The number of the current command in the session.

**- '\!'** - The number of the current command in the command history.

**- '\$'** - Displays the $ symbol for normal users and # for root users.

To view the current value of the $PS1 variable in the terminal, use the echo command: 'echo $PS1'

**4. \*How can I change the value of the $PS1 variable? What will happen in the bash prompt line (the prompt line before each command). How to change the value of this variable not for the current session, but by default?**

To change the value of the $PS1 variable and thereby change the appearance of the invitation string in Bash, you can run the assignment command in the terminal. For example: **'PS1='[\u@\h \W]\$ ''**

This command will set the prompt string to display the username ('\u'), the hostname up to the first dot ('\h'), the current working directory ('\W'), and then the '$' symbol for a normal user or '#' for a user with superuser (root) privileges. All this will be displayed in square brackets.

The changes made using the command above are temporary and will only be valid until the end of the current terminal session. To make the changes permanent, you need to add an assignment command to one of the shell configuration files that are executed at startup. For Bash, this is usually the '.bashrc' file in the user's home directory.

1. Open the '.bashrc' file in a text editor. For example, using 'nano': 'nano ~/.bashrc'.

2. Add the command to assign the variable '$PS1' to the end of the file. For example: 'PS1='[\u@\h \W]\$ ''

3. Save the file and exit the editor. For 'nano', this can be done by pressing 'Ctrl+O', 'Enter', and then 'Ctrl+X'.

4. To apply the changes, restart the terminal or re-read the configuration file using the 'source' command: 'source ~/.bashrc'.

**5. \*What are quotation marks used for in the Bash shell?**

In the Bash shell, quotes are used to control the interpretation of characters and strings. There are three main types of quotes:

**Single quotes (')**

- Single quotes are used to specify strings in which all characters between them are interpreted literally. This means that no character within the single quotes is considered special and cannot be interpreted by the shell.

**Double quotes (")**

- Double quotes allow for variable expansion (i.e., replacing variables with their values), executing commands in back quotes or '$()', and interpreting special characters (e.g., '\n' for a newline), but ignore other special characters, such as the dollar sign ('$'), unless it is used to denote a variable.

**Back quotes (`)**

- Back quotes are used to execute commands and replace the output of those commands at the point where they are used. However, this syntax is old and is often replaced by the more modern '$()' syntax, which has the same functions but better supports nesting of commands.

**6. \*\*** **What are control statements used for, and what types of them do you know?**

Control statements in the Bash shell are used to create conditional structures, loops, check the results of command execution, and otherwise control the progress of a script. They allow you to create complex scripts that can make decisions, repeat operations, and handle a variety of execution scenarios. Here are some basic types of control statements in Bash:

**Cycles**

* 'for' - used to iterate through a list of values or a range of numbers

- 'while' - executes a block of commands while the condition is true

- 'until' - executes a block of commands until the condition is true

**Selection**

- 'case' - simplifies multiple selection and comparison with fixed values

**Controlling execution**

- 'break' and 'continue' - used to exit the loop or skip the iteration of the loop, respectively

**7. \*\*** **What is the difference if the bash prompt line ends with a $ or # character? For example, on the screen we see the following entries**

* **[centos@localhost Desktop]$**

If you see the '$' symbol, it means that you are logged in as a regular user (not root or another user with privileges).

This symbol at the end of the prompt indicates that your commands are executed with limited privileges, and you need to use the 'sudo' command or enter root mode if you need administrative privileges.

* **[root@localhost Desktop]#**

If you see the '#' symbol, it means that you are logged into the shell as a user with privileges (usually root or another administrator).

This symbol indicates that your commands are executed with full administrative privileges and you do not need to use 'sudo'.

**8. \*\*** **What is the purpose of the whereis and locate commands? What is the difference between them?**

The whereis and locate commands are used to search for files and directories in the system, but they work a little differently.

- 'whereis':

The 'whereis' command is designed to quickly locate binary files that exist in system execution paths. It searches for files in standard system locations such as '/bin', '/sbin', '/usr/bin', '/usr/sbin', and others.

- 'locate':

The 'locate' command searches for files and directories in a database, which is usually built based on file system indexes. This allows it to run faster because it doesn't have to rescan the file system in real time. However, it is important to note that the file system indexing is not updated in real time, the indexes are often updated periodically by background processes or task schedulers.

**The difference between 'whereis' and 'locate':**

Source of information:

* 'whereis' uses standard system paths to look up binary files.
* 'locate' uses an indexed database, usually built from the file system.

Update frequency:

* 'whereis' uses real-time, up-to-date file system data.
* 'locate' may show outdated results because the database may be updated periodically rather than in real time.

Performance:

* 'locate' can be faster to search because it uses an indexed database.
* 'whereis' performs real-time searches and may take longer to complete, but it displays more accurate and up-to-date results.

Conclusion: I got acquainted with the basic commands of the CLI mode in Linux, got acquainted with the basic text commands in the terminal mode in different OS.