“Київський фаховий коледж зв’язку”

Циклова комісія Комп’ютерної інженерії

**ЗВІТ ПО ВИКОНАННЮ**

**ЛАБОРАТОРНОЇ РОБОТИ №3**

з дисципліни: «Операційні системи»

**Тема: “Знайомство з базовими командами CLI-режиму в Linux”**

Виконав ли студенти)

групи КСМ 13А

Команда 1: ВВС

Панчук О.С. Петрик С.С

Перевірив викладач

Сушанова В.С.

Київ 2023

**Мета роботи:**

1. Отримання практичних навиків роботи з середовищами віртуальних машин та операційними системами різних типів та сімейств – їх графічною оболонкою, входом і виходом з системи, ознайомлення зі структурою робочого столу, вивчення основних дій та налаштувань при роботі в системі.

**Матеріальне забезпечення занять**

1. ЕОМ типу IBM PC.

2. ОС сімейства Windows (Windows 7).

3. Віртуальна машина – Virtual Box (Oracle).

4. Операційна система GNU/Linux – CentOS.

**Завдання для попередньої підготовки**

***Готував матеріал студент Панчук О.С.***

1. Прочитайте короткі теоретичні відомості до лабораторної роботи та зробіть невеличкий словник базових англійських термінів з питань класифікації ОС.

|  |  |
| --- | --- |
| Термін англійською | Термін українською |
| **Operating System** | Операційна система |
| **Arguments** | аргументи |
| Command Line Skills | Навички роботи з командним рядком |
| Inline editing | Редагування на місці |
| Scripting | сценарії |
| The Shell | Командний рядок |
| Bash shell | оболонка Bash |
| Prompt structure | Ім'я користувача |
| Variables | Змінні |
| Aliases | Псевдонім |

1. Прочитавши матеріал з коротких теоретичних відомостей дайте відповіді на наступні питання:

***Готував матеріал студент Панчук О.С.***

* 1. Охарактеризуйте поняття «гіпервізор». Які бувають їх типи?

**Командний інтерпретатор**-this is software that executes commands and instructions entered by the user through a command-line interface or text interface. It is an interactive shell for interacting with the operating system and performing various tasks.

The main functions of a command interpreter include:

1. Command Execution: The interpreter processes and executes commands entered by the user or launched from scripts.
2. Program Execution: Users can run executable files, programs, and services from the command line.
3. File and Directory Operations: The interpreter provides the ability to manage files and directories, such as creating, moving, deleting, and changing access rights.
4. Input/Output Streams: The interpreter manages input (stdin) and output (stdout) streams for user interaction and command execution result processing.
5. Variables and Aliases: Users can define variables and aliases to simplify command invocation and store data.
6. Scripts: The command interpreter can execute scripts that contain sequences of commands and instructions for task automation.

**Оболонка–**(Shell) in programming refers to an interactive interface that allows interacting with the operating system or executing commands and programs through a text-based interface. Shells in programming are used for automating operations, processing data, executing commands, and running scripts.

In the context of a command-line or script, the shell executes commands entered by the user or contained in executable files. The shell interprets and carries out the commands, ensuring their execution in the operating system.

Examples of shells in programming include Bash, PowerShell, Command Prompt, Python shell, and many others. Shells help automate tasks, create scripts, and interact with the operating system through a text-based interface.

**"Сommand**" - instructions or operators that instruct a computer to perform a specific action or operation. Commands in programming are used to carry out specific tasks or control the execution of a program.

Commands can be part of the program code and contain instructions for calculations, data reading and writing, control of program execution flow, and much more. Depending on the programming language, commands may have different syntax and functionality.

Відповіді на запитання

1. The prompt provides basic information for the user, typically displaying the current directory (working directory), user information, system information, instructions, or other contextual information that helps the user interact with the command interpreter.
2. Commands require parameters and arguments to specify the actions they should perform. Parameters provide additional information to modify the behavior of a command, while arguments are the specific inputs or data that a command operates on.
3. The “ls” command in Bash is used to list files and directories in the current directory. It can take various parameters and arguments to customize its behavior. Here are three examples:

* “ls: Lists files and directories in the current directory.
* “ls -l: Lists files and directories in long format, showing detailed information.
* “Is -a /path/to/directory”: Lists all files and directories in the specified directory, including hidden ones.

1. You can use command history by pressing the "Up" arrow key on your keyboard to cycle through previously entered commands. This allows you to quickly access and reuse commands without retyping them. The advantages of using command history include increased efficiency, reduced typing errors, and the ability to recall and repeat complex commands.
2. The “echo” command is used to display messages or text on the terminal. It is often used for printing output, displaying variables, or creating simple scripts. For example, “echo” "Hello, World!" will display "Hello, World!" on the terminal.
3. In the Bash shell, a variable is a symbol that represents a value or text. Bash supports various types of variables, including:

* Scalar variables: Hold single values, such as numbers or strings.
* Array variables: Store multiple values in indexed arrays.
* Associative arrays: Store key-value pairs.
* Environment variables: Used to configure the shell's behavior and provide information to running programs.

1. The “env” command displays the current environment variables, while the “export” command is used to set or modify environment variables. Environment variables are global variables that are accessible to all processes running in the shell session. The “unset” command is used to remove environment variables, effectively unsetting their values. These commands are essential for configuring the shell's behavior and providing information to programs
2. man: The “man” command is used to display the manual pages (documentation) for various commands and system functions. You can use it as follows:

man command\_name

Replace “command\_name” with the name of the command you want to learn more about. You can navigate through the manual pages using arrow keys, and press “q” to exit.

“--help” option: Many command-line utilities and programs support the “—help” option, which provides a brief summary of the command's usage and available options. You can use it like this:

command\_name –help

Replace “command\_name” with the name of the command you want to get help for.

“Help” (for shell built-ins): If you're looking for help on shell built-in commands (commands that are part of the shell itself, such as “cd” or” echo”), you can use the “help” command followed by the built-in command's name. For example:

“help cd”

**Хід роботи**

***Готував матеріал студент Петрик С.С.***

2.1

Зображення, що містить текст, електроніка, знімок екрана, програмне забезпечення

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, програмне забезпечення, монітор

Автоматично згенерований опис

**2.2**

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

**2.3**

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

**2.4**

Unfortunately, the program gives me numerous errors when I try to input the code correctly, so I'll just attach the code itself.

var\_name1="Stanislav"

var\_name2="Oleksandr"

(2.1)

echo "Students:"

echo $var\_name1

echo $var\_name2

alias mycal1="cal 9 2005"

alias mycal2="cal 10 2005"

echo "Календар для року народження:"

mycal1

mycal2

(2.2)

var\_name1="Stanislav"

var\_name2="Oleksandr"

birth\_year1=2005

birth\_year2=2005

echo "Students:"

echo $var\_name1

echo $var\_name2

echo "Year of birth:"

echo $birth\_year1

echo $birth\_year2

**2.5**

Зображення, що містить текст, знімок екрана, програмне забезпечення

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, програмне забезпечення, Шрифт

Автоматично згенерований опис

**Відповіді на контрольні запитання**

***Готував матеріал студент Петрик С.С***

# 1.

In the Bash shell, there are several types of commands:

Built-in commands: These are commands that are built directly into the Bash shell. They are executed internally and do not require calling an external program. Examples of built-in commands include cd, pwd, echo, and so on.

External programs: These are commands that are separate executable programs or scripts not part of the Bash shell itself. They are invoked as separate processes. Examples of external programs include system utilities like ls, grep, and user-written scripts.

Aliases: Aliases are user-defined shortcuts for longer commands or command options. They are set using the alias command and can simplify frequently used commands.

Functions: Bash allows you to define your own functions using the function keyword or by using a simpler syntax. Functions can encapsulate multiple commands and are often used to create reusable code.

Scripts: You can create shell scripts, which are executable files containing a series of Bash commands. These scripts can be executed just like any other external program.

# 2.

Environment variables are a fundamental part of the Unix and Linux operating systems. They are dynamic named values that can affect the way running processes behave or provide information about the system's environment. Here's an overview of environment variables, the types of environment variables, and how to view them in a terminal:

* Types of Environment Variables:
  + System Environment Variables: These variables are set at the system level and apply to all users and processes. They often contain information about system configurations and paths to system binaries. Examples include PATH, HOME, and LANG.
  + User Environment Variables: These variables are specific to each user and are typically set when a user logs in. They can override system variables for that user's session. Examples include USER, SHELL, and TERM.
  + Custom Environment Variables: Users and applications can define their own custom environment variables to store specific configuration settings or information needed for their operation.
* Viewing Environment Variables:
  + To view all environment variables in a terminal, you can use the env command or printenv command.
  + This command will display a list of all environment variables and their values.
  + To view the value of a specific environment variable, use echo with the variable's name preceded by a dollar sign ($).
  + You can also use the printenv command followed by the variable name to display its value
  + To set or modify an environment variable for the current session, you can use the export command. For example, to set a custom variable
  + To make changes to environment variables persistent, you can add them to shell configuration files like ~/.bashrc or ~/.bash\_profile for Bash, or system-wide configuration files for system-wide variables.

Environment variables play a crucial role in configuring the behavior of programs and providing essential information to the system and user processes. Understanding how to work with them is essential for effective system administration and development on Unix-based systems.

# 3.

The $PS1 environment variable in a Unix-like shell, such as Bash, defines the primary prompt string. This string is displayed as the command prompt in your terminal and provides information about the current shell session. It typically includes information like the username, hostname, current working directory, and other details.

Here's a breakdown of the $PS1 variable:

* \u: Represents the username of the current user.
* \h: Represents the hostname up to the first '.'.
* \w: Represents the current working directory.
* $: Displays a '#' symbol for the root user and a '$' symbol for regular users. It indicates whether the user has superuser (root) privileges.
* \n: Represents a newline character, creating a new line in the prompt.
* : Encloses non-printing characters to indicate they won't take up space on the screen. This is necessary to ensure proper line wrapping.

This command will display the current value of the $PS1 variable, which defines your command prompt. You can also check the content of $PS1 in your shell's configuration files, such as ~/.bashrc or ~/.bash\_profile, where it is often customized to change the appearance of your shell prompt.

# 4.

To change the value of the $PS1 variable temporarily for your current session in Bash, you can use the export command to assign a new value to it. For example, to change the prompt to display "CustomPrompt> " you can do the following:

*export PS1="CustomPrompt> "*

After running this command, your command prompt will be changed to "CustomPrompt> " until you exit the current shell session. This change won't persist beyond the current session.

If you want to change the $PS1 variable permanently, you need to modify your shell's configuration file. The specific file depends on your shell and system configuration. For Bash, the common configuration files are ~/.bashrc for individual users and /etc/bash.bashrc for system-wide settings.

To make a permanent change to the $PS1 variable:

* Open your shell configuration file in a text editor. For example:

*nano ~/.bashrc*

* Find the line that sets the $PS1 variable. It may look something like this:

*PS1="\u@\h:\w\$ "*

* Modify the value within double quotes as desired. For example:

*PS1="CustomPrompt> "*

* To apply the changes, either open a new terminal or run source ~/.bashrc to reload the configuration file in your current terminal session.

After making this change and reloading your shell configuration, the new $PS1 value will be used as the default prompt in all future Bash sessions for your user.

# 5.

In the Bash shell, quotation marks (single or double) are used to control how strings are interpreted. Here's a brief explanation of their usage:

Single Quotes (' '): When you enclose a string in single quotes, like 'example', everything within the single quotes is treated as a literal string. Variables and special characters within single quotes are not expanded or interpreted. This is useful when you want to preserve the exact text without any variable substitution or special character interpretation.

Double Quotes (" "): Double quotes allow for variable expansion and interpretation of certain special characters. When you enclose a string in double quotes, variables within the quotes are expanded to their values, and certain special characters like $, \, and ! are interpreted.Be cautious when using double quotes with special characters, as they can have different meanings or effects depending on the context.

Backticks (` `): Backticks are used for command substitution. When you enclose a command in backticks, Bash executes the command and replaces it with the output of that command.

The choice of quotation marks depends on your specific needs. Single quotes are typically used when you want to preserve text exactly as it is, while double quotes are used when you want variable expansion and interpretation of certain special characters. Backticks or $() are used for command substitution.

# 6.

Control statements, or control structures, are used in programming to manage the flow of a program. Common types include:

* Conditional Statements: Used for decision-making, such as if and switch statements.
* Looping Statements: For repeating actions, like for, while, and do-while loops.
* Branching Statements: Used to control loops, including break, continue, and return.
* Exception Handling: For dealing with errors, often with try-catch blocks.
* Jump Statements (avoided in modern code): goto allows unconditional jumps.
* Control Flow Statements: return and exit control program flow.

These structures enable programmers to create flexible and structured code.

# 7.

In Bash, the symbol at the end of the prompt, whether it's a $ or #, conveys important information about your current shell session. Here's the difference:

* *$ (Dollar Sign):*
  + If your Bash prompt ends with a $, it typically indicates that you are running the shell as a regular user, which means you do not have superuser (root) privileges.
  + This is the standard prompt for regular users in most Unix-like systems, and it signifies limited access to system-level commands and configuration changes.
* *# (Hash or Pound Sign):*
  + If your Bash prompt ends with a #, it typically indicates that you are running the shell with superuser (root) privileges.
  + This root prompt signifies that you have administrative access to system-level commands and can make changes that affect the entire system.

The difference between these symbols is significant in terms of the level of access and control you have within the shell. The $ prompt is associated with a regular user, while the # prompt is associated with the root user who has elevated privileges. It serves as a visual indicator of whether you are operating with standard user rights or as a system administrator.

# 8.

In summary:

* whereis is used to locate program-related files (binaries, source code, manuals).
* locate is used to find files and directories by name or pattern.
* locate is generally faster for finding files and uses an indexed database.

**Висновки  
*Готував матеріал студент Панчук О.С***

Getting acquainted with basic CLI (Command Line Interface) commands in Linux is an essential part of working with the Linux operating system. It can be beneficial for various tasks, including administration, configuration, automation, and more.

Key CLI commands in Linux include operations such as viewing files and directories (using the “ls” command), navigating the file system (using the “cd” command), creating directories (with “mkdir”), deleting files and directories (“rm”), renaming and moving files (“mv”), copying files (“cp”), working with text files (“cat”, “”less”, “nano”, etc.), and managing file and directory permissions (“chmod”).

These commands allow users to interact with the operating system, perform tasks, and execute routine operations without the need for a graphical interface. They are a powerful tool for automating tasks, configuring the system, and quickly accessing information while maintaining control over files and directories.

In conclusion, knowledge and proficiency in using basic CLI commands in Linux are crucial for efficiently working with this operating system. They enhance productivity, provide greater flexibility for system configuration and management, and make Linux a more powerful tool for professional administrators, developers, and regular users.