

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Linux

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для
здобувачів ступеня бакалавра
спеціальності F2 Інженерія програмного забезпечення
спеціальності F6 Інформаційні системи та технології
спеціальності F7 Комп'ютерна інженерія

Укладачі: М. С. Хмелюк

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 004.45 (075.8)

Укладачі: *Хмелюк Марина Сергіївна*

Рецензент *Лісовиченко О.І., к.т.н., доцент кафедри інформатики та програмної інженерії КПІ ім. Ігоря Сікорського*

Відповідальний редактор *Амонс О.А., к.т.н., доцент кафедри інформаційних систем та технологій КПІ ім. Ігоря Сікорського*

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 05.03.2026 р.)
за поданням вченої ради факультету Інформатики та обчислювальної техніки
(протокол № 7 від 23.02.2026 р.)*

Linux. [Електронний ресурс] : лаб. практикум : навч. посіб. для здобувачів ступеня бакалавра за спец. F2 Інженерія програмного забезпечення, F6 Інформаційні системи та технології, F7 Комп'ютерна інженерія / КПІ ім. Ігоря Сікорського ; уклад.: М. С. Хмелюк. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 228 с.

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Linux». Посібник містить роз'яснення щодо виконання лабораторного практикуму, передбаченого робочою програмою дисципліни «Linux». В посібнику викладено інформаційні матеріали та приклади налаштування, адміністрування операційної системи з використанням командної оболонки Bash в Linux. Посібник може бути корисним студентам відповідних спеціальностей при вивченні дисциплін, пов'язаних з побудовою, налаштуванням та адмініструванням мережевої інфраструктури, IT-інфраструктури з використанням Linux.

УДК 004.45 (075.8)

Реєстр. № НП 25/26-233. Обсяг 11,4 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© Марина ХМЕЛЮК, 2025
© КПІ ім. Ігоря Сікорського, 2025

ЗМІСТ

ВСТУП	4
ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ.....	6
ЛАБОРАТОРНИЙ ПРАКТИКУМ №1 Установка, налаштування Linux системи Ubuntu	9
ЛАБОРАТОРНИЙ ПРАКТИКУМ №2 Інтерпретатор. Команди інтерпретатора. Змінні оточення, оболонки	41
ЛАБОРАТОРНИЙ ПРАКТИКУМ №3 Структура файлової системи. Управління каталогами/файлами Ubuntu	69
ЛАБОРАТОРНИЙ ПРАКТИКУМ №4 Стандартні потоки Linux.....	87
ЛАБОРАТОРНИЙ ПРАКТИКУМ №5 Управління правами доступу до файлів і каталогів Linux	117
ЛАБОРАТОРНИЙ ПРАКТИКУМ №6 Ідентифікація користувачів. Umask в Linux. Жорсткі і символічні посилання	149
ЛАБОРАТОРНИЙ ПРАКТИКУМ №7 Управління процесами	162
ЛАБОРАТОРНИЙ ПРАКТИКУМ №8 Обробка даних за допомогою мови shell. Скрипти в Linux. Планувальник задач в Linux	183
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	228

ВСТУП

Будь-яка обчислювальна система складається з апаратного забезпечення та програмного забезпечення. Програмне забезпечення включає в себе системне програмне забезпечення - забезпечення, яке необхідне для коректного функціонування системи та, прикладне програмне забезпечення, яке необхідне для роботи користувача.

До системного програмного забезпечення можна віднести операційну систему основними функціями операційної системи є організація діалогу між користувачем і апаратним забезпеченням комп'ютера та управління апаратною та програмною складовими обчислювальної системи, організація взаємодії складових системи, розподіл ресурсів таких як процесор пам'ять пристрої вводу виведення даних між процесами які конкурують за ці ресурси.

Таким чином, можна сказати, що операційна система - це набір програм, які призначені для організації діалогу між користувачем та комп'ютером та програм для управління комп'ютером.

Найбільш поширеними у використанні є система Windows від компанії Microsoft та операційна система Linux, представлена різними дистрибутивами. Операційна система Linux використовується в якості користувацької системи, серверного варіанту та мобільної операційної системи.

Дисципліна «Linux» належить до Вибіркових освітніх компонент спеціальностей: F2 Інженерія програмного забезпечення, F6 Інформаційні системи та технології, F7 Комп'ютерна інженерія. Вивчення дисципліни базується на знаннях, навичках та уміннях, отриманих під час вивчення студентами дисциплін «Операційні системи», «Комп'ютерні мережі».

Основна мета вивчення дисципліни – володіння знаннями щодо принципів роботи операційної системи Linux та застосовування знань в адмініструванні, конфігуруванні системи Linux. В результаті вивчення дисципліни повинен вміти

інсталювати операційну систему Linux, працювати в системі за допомогою командного рядка Linux, керувати даними зі стандартних потоків Linux, конфігурувати файлову систему в залежності від потреб застосування, управляти правами доступу до файлів та каталогів, адмініструвати користувачів, групи, управляти процесами в Linux, налаштовувати журнали операцій, писати скрипти мовою Shell, планувати виконання задач в Linux, налаштовувати операційну систему для роботи з апаратним забезпеченням, мережею в Linux, налаштовувати графічне середовище в Linux.

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Посібник призначений для виконання комп'ютерних практикумів з дисципліни «Linux». Він складається з 8 тем комп'ютерного практикуму, де для кожної із тем систематизовані теоретичні матеріали, приклади практичного засвоєння теоретичного матеріалу, завдання, та рекомендована література. В процесі вивчення даної навчальної дисципліни студенти засвоюють основні принципи роботи з ОС Linux. В комп'ютерному практикумі основна увага приділяється налаштуванню, адмініструванню операційної системи Linux за допомогою командного рядка командної оболонки Shell.

Для виконання комп'ютерних практикумів використовується комп'ютер з встановленою операційною системою Windows 7, 8, 10, 11 або Linux [1]. Для роботи з операційною системою Linux використовується дистрибутив Ubuntu. Для розгортання та безпечної роботи з Ubuntu, систему потрібно встановити на віртуальну машину, для цього використовується програма для віртуалізації Oracle VirtualBox або VMware Workstation.

При вивченні конкретної теми слід уважно ознайомитися з поставленим завданням, пропрацювати матеріали комп'ютерного практикуму по цій темі. Лабораторний практикум відпрацьовується студентами групи індивідуально, протягом практикуму студенти виконують всі заплановані практикуми один за одним. Кожна практична робота комп'ютерного практикуму оформляється у вигляді звітнього документу. Звіт повинен містити наступні розділи:

- стандартний титульний лист (вказати: ВНЗ, факультет, кафедру; тему та номеру поточної роботи; код групи та П.І.Б. виконавця; П.І.Б. викладача що перевірятиме роботу);
- назва практикуму та мета роботи;
- завдання до практикуму;
- тези теоретичних відомостей;

- результати виконання;
- висновки по роботі.

Операційна система Linux - це багатокористувацька, багатозадачна система, яка добре зарекомендувала себе, як користувацька система, серверна система, в якості файлового серверу, серверу баз даних, поштового серверу, серверу друку та мобільна операційна система [2]. Для прикладу, операційна система Android побудована на ядрі Linux. Операційна система Linux прекрасно працює на більшості апаратних платформ та інтегрується з багатьма інструментами, Має велику кількість дистрибутивних версій, наприклад Debian, Ubuntu, Fedora, Kali Linux, Arch Linux, Red Hat, Linux сумісна з багатьма файловими системами, які розроблені для операційних систем Linux, також і операційної системи Windows.

Більшість дистрибутивів поширюється на безоплатній основі, що є привабливим у використанні даної системи як користувацької системи так і серверного варіанту. Особливістю системи Linux є те що вона поширюється у вихідних відкритих вихідних кодах, це дає можливість користувачу модифікувати системи, розширювати функціональні системи, доповнювати систему програмами, виправляти помилки. Навколо системи Linux утворилося велике співтовариство розробників, які постійно вдосконалюють систему, розробляють нові варіанти та модифікації системи, програмне забезпечення під систему Linux. Операційна система Linux побудована таким чином, що виключає роботу вірусних програм, що робить систему надійною та безпечною.

За стандартом POSIX (Portable Operating System Interface for uniX) [3], який описує інтерфейси між операційною системою та прикладним програмним забезпеченням, обов'язковим компонентом операційної системи є командна оболонка, вона ж інтерпретатор мови команд Shell - це користувацький інтерфейс розміщений в просторі користувацьких процесів і забезпечений набором засобів для спілкування з ядром операційної системи та утилітами

операційної системи. Таким чином, командний рядок є найважливішим засобом для спілкування з операційною системою. Будь-які дії в системі виконуються шляхом передачі відповідних командних директив. Всі команди які набираються в командному рядку можуть бути використані в командних (виконуваних) файлах, які використовуються інтерпретатором Shell та навпаки, дії, що виконуються в командному інтерпретаторі в подальшому можуть бути оточені графічною оболонкою і деталі їх виконання будуть прихованими від кінцевого користувача. Засоби, які дають можливість користувачу працювати в графічному режимі забезпечує кросплатформенна система X Windows System. Ця система не прив'язана ні до апаратної платформи комп'ютера, ні до будь-якої операційної системи, вона не має відношення до жодного з дистрибутивів Linux.

ЛАБОРАТОРНИЙ ПРАКТИКУМ №1 Установка, налаштування Linux системи Ubuntu

Мета: навчитись працювати з програмою віртуалізації, створювати та налаштовувати віртуальну машину та інсталиувати операційну систему.

Завдання

Завантажити Oracle VM VirtualBox. Інсталиувати на комп'ютер. Налаштувати. Завантажити дистрибутив Linux - Ubuntu. Інсталиувати на віртуальну машину.

Теоретичні відомості

Віртуалізація дає змогу пристрою однієї операційної системи емулювати іншу операційну систему. Операційна система (ОС), всередині якої запускається інша операційна система, називається хост-системою host, а ОС, яка працює в віртуальному оточенні - гостьова guest.

Віртуалізація - це створення ізольованих оточень в рамках одного фізичного пристрою (комп'ютера). Кожне оточення при цьому виглядає як окремий комп'ютер зі своїми «логічними» ресурсами, такими як пам'ять, процесор тощо. Таке оточення називається віртуальною машиною [4].

Поняття віртуалізації пов'язане зі спеціальною програмою, яка має назву «гіпервізор», яка займається створенням віртуальних машин і їх управлінням.

Гіпервізор забезпечує ізоляцію операційних систем однієї від іншої, захист і безпеку, поділ ресурсів між запущеними ОС.

Гіпервізор першого типу працює безпосередньо на фізичному апаратному забезпеченні хост-машини (рисунок 1.1).

Гіпервізор другого типу встановлюється на існуючу ОС (рисунок 1.1).

Перший тип відповідає апаратній віртуалізації, а другий - програмній. Апаратна віртуалізація працює завдяки підтримці з боку апаратного забезпечення комп'ютера - процесора. На відміну від програмної віртуалізації, гостьові операційні системи управляються гіпервізором безпосередньо без участі хостової ОС. Програмна віртуалізація працює через основну операційну систему, встановлену на хост-машину.

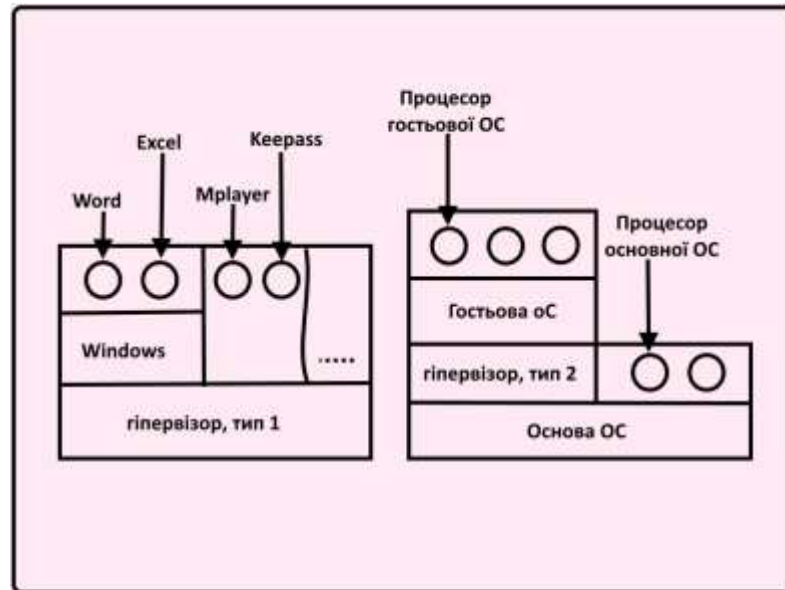


Рисунок 1.1 – Гіпервізори 1-го та 2-го типу

Програмна віртуалізація, в свою чергу, ділиться на кілька підтипів:

- програмна паравіртуалізація - операційна система взаємодіє з програмою гіпервізор, який надає їй гостьовий інтерфейс API, замість використання безпосередньо такого ресурсу, як пам'ять;
- програмна повна віртуалізація (динамічна трансляція) - при динамічній (бінарній) трансляції проблемні команди гостьової операційної системи перехоплюються гіпервізором.

Окремо можна виділити контейнерну віртуалізацію. На відміну від попередніх видів, вона не пов'язана з запуском ОС в ізольованому оточенні. При

контейнерній віртуалізації ізоляція відбувається на рівні процесу операційної системи.

Контейнерна віртуалізація - метод віртуалізації, при якому ядро операційної системи підтримує кілька ізольованих примірників простору користувача замість одного. Ці примірники називаються контейнерами або зонами і з точки зору користувача повністю ідентичні окремому екземпляру операційної системи.

Ядро забезпечує повну ізольованість контейнерів, тому програми з різних контейнерів не можуть впливати одна на одну..

На відміну від апаратної віртуалізації, при якій емулюється апаратне оточення і може бути запущено кілька різних гостьових операційних систем, в контейнері може бути запущений екземпляр операційної системи тільки з тим же ядром, що і у хостової операційної системи. При цьому при контейнеризації відсутні додаткові ресурсні витрати на емуляцію віртуального обладнання та запуск повноцінного примірника операційної системи, характерні при апаратної віртуалізації. Найбільш відомі проекти: OpenVZ, Docker, LXC.

Ще одним прикладом віртуалізації є віртуальна машина Java (рисунок 1.2).

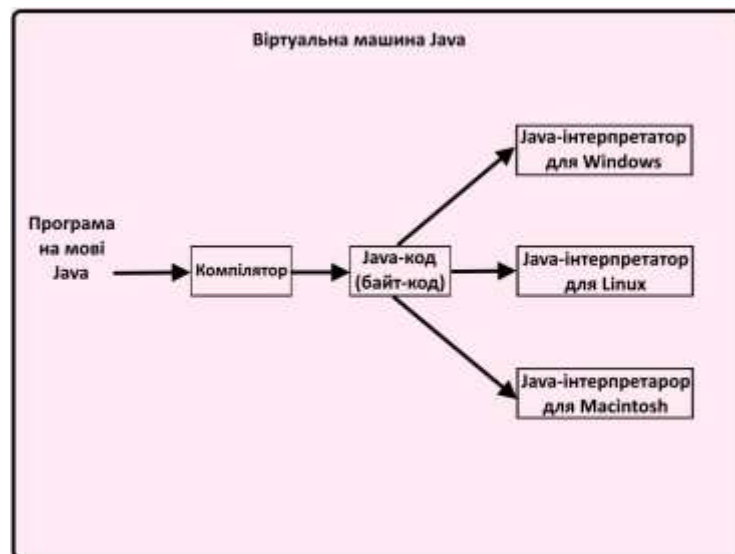


Рисунок 1.2 – Віртуальна машина Java

На персональних комп'ютерах поширена саме програмна віртуалізація. На відміну від установки двох операційних систем на одну машину поруч, **віртуалізація** - це більш безпечний метод. У будь-який момент можна все видалити і перевстановити. Також можна створити необхідну, для користувача, кількість віртуальних машин. Віртуалізація ізолює машини одну від одної, а це виключає можливість одних користувачів нашкодити системі або іншим користувачам.

Віртуальна машина - це програма, яка емулює фізичний комп'ютер з усіма його компонентами (процесором, оперативною пам'яттю, пристроями зберігання та вводу і виводу інформації, мережевими адаптерами, тощо). На такий віртуальний комп'ютер можна встановити, наприклад, операційну систему, драйвери, програми. Таким чином, можна запустити на своєму реальному комп'ютері ще кілька віртуальних комп'ютерів з такою ж або іншою операційною системою. Є можливість без проблем здійснювати обмін даними між реальним і віртуальним комп'ютером.

Віртуальну машину використовують для різних цілей і завдань:

- установка другої / іншої операційної системи;
- тестування програмного забезпечення;
- безпечний запуск підозрілих програм;
- емуляція комп'ютерної мережі;
- запуск застосунків, які не можна запустити з основної операційної системи.

Віртуальна машина - це дуже зручно, тобто можна тестувати різні незнайомі програми у віртуальному середовищі, щоб не наражати на небезпеку і не захаращувати основний фізичний комп'ютер.

Наприклад, на ваш комп'ютер може бути встановлена операційна система Windows 10, а на віртуальну машину можна поставити і протестувати інші

операційні системи, такі як Windows 8 або Linux, а також встановлювати і тап використовувати різні програми і утиліти.

Існує велика кількість різних програм для створення і управління віртуальними комп'ютерами: Oracle VM VirtualBox, VMware Workstation, Microsoft, Citrix, Open Virtualization Alliance (OVA), Hyper-V, Xen, kVM та інші.

VirtualBox - це безкоштовна віртуальна машина, на яку можна встановити всі найпопулярніші операційні системи [5]. VirtualBox підтримує роботу з Windows, Linux, FreeBSD, Mac OS.

VirtualBox підтримує як 32-х, так і 64-розрядні версії операційних систем. VirtualBox підтримує роботу з віртуальними комп'ютерами, створеними в програмі VMware Workstation.

Налагодження та робота з VirtualBox дуже зручна і проста. Програма продуктивна і стабільна має широкий функціонал, зручний інтерфейс і вона безкоштовна. VirtualBox - віртуальна машина, яка добре підходить для домашнього використання.

Перш ніж встановлювати VirtualBox в вашу систему, потрібно перевірити чи увімкнена підтримка віртуалізації на вашому комп'ютері. Для цього зайдіть в BIOS комп'ютера або ноутбука і в розділі налаштувань CPU увімкніть підтримку віртуалізації.

Завантажити програму можна на офіційному сайті на сайті Oracle за посиланням <https://www.oracle.com/cis/virtualization/virtualbox/> (рисунок 1.3).

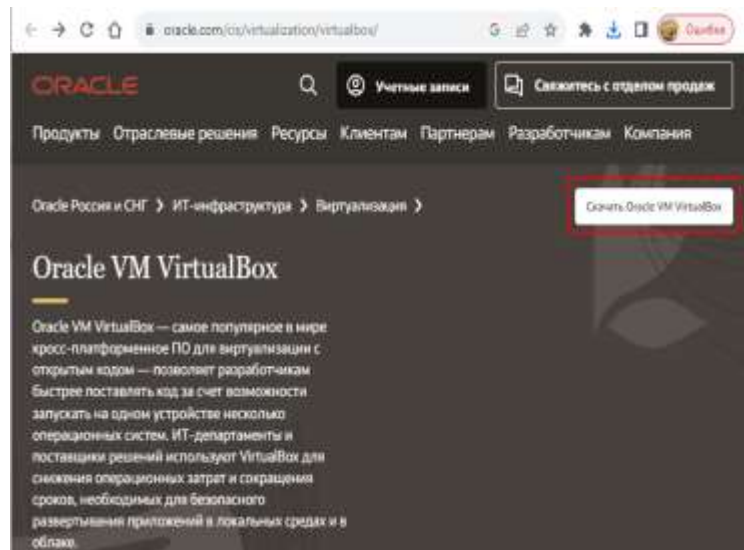


Рисунок 1.3 - Офіційний сайт Oracle

В наступному вікні можна побачити версії програми та які платформи підтримуються (рисунок 1.4)



Рисунок 1.4 - Вікно вибору дистрибутива

Скачайте дистрибутив на комп'ютер та запустіть установку VirtualBox (рисунок 1.5).



Рисунок 1.5 – Запуск установки VirtualBox

Виберіть компоненти VirtualBox для установки (рисунок 1.6).

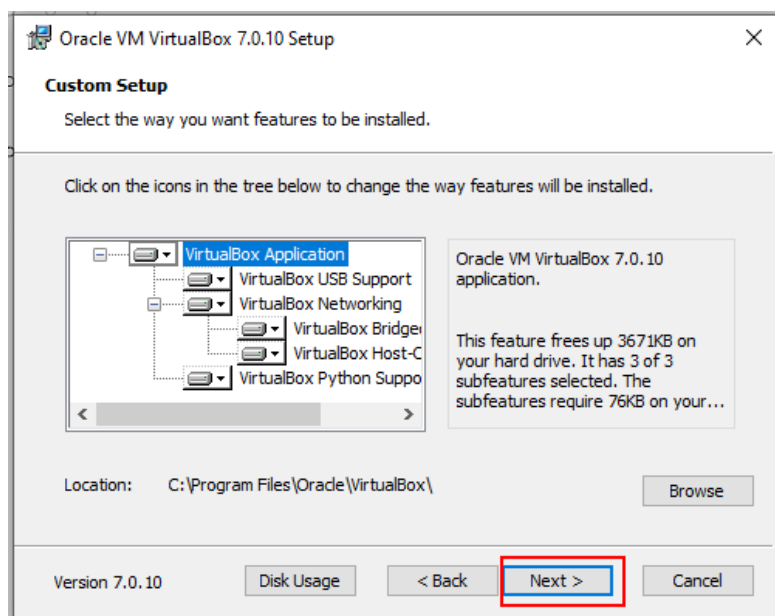


Рисунок 1.6 – Компоненти для установки

Для доступу віртуальних машин до мережних інтерфейсів хоста будуть встановлені Network драйвери VirtualBox (рисунок 1.7).



Рисунок 1.7 – Вікно установки мережеских драйверів

Натисніть «Інсталювати» для продовження установки (рисунок 1.8).

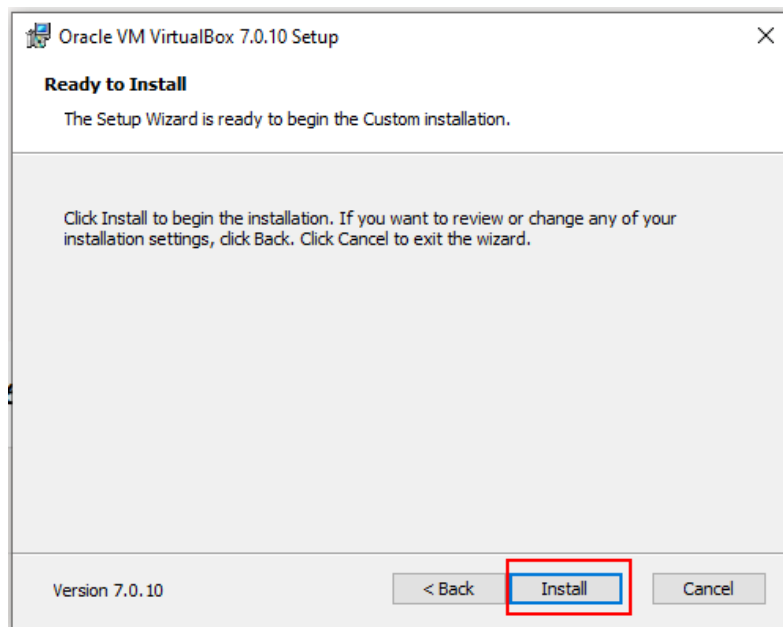


Рисунок 1.8 – Підготовка до інсталяції

Процес установки. Це може зайняти кілька хвилин (рисунок 1.9)

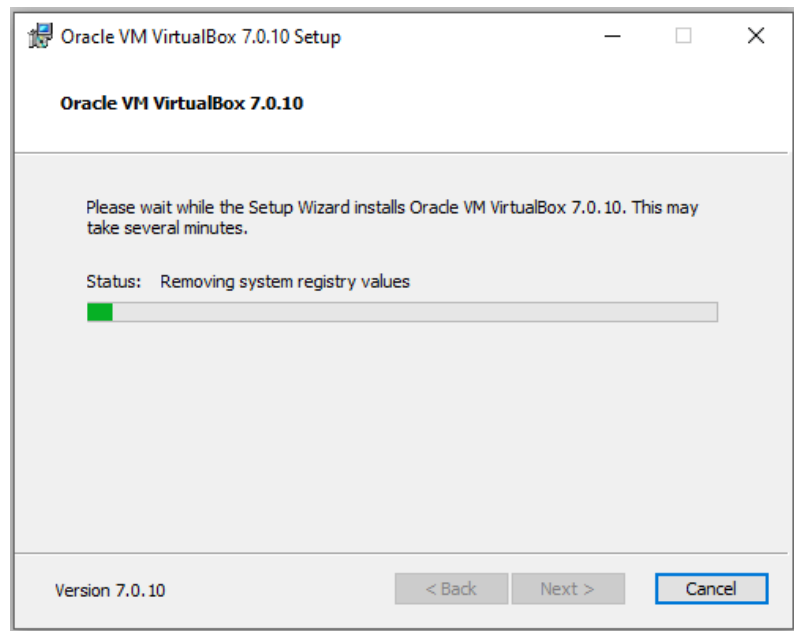


Рисунок 1.9 – Процес установки VirtualBox

Установка Oracle VM VirtualBox завершена. Натисніть «Завершити» (рисунок 1.10).



Рисунок 1.10 – Завершення установки

Запустіть Oracle VM VirtualBox менеджер (рисунок 1.11).

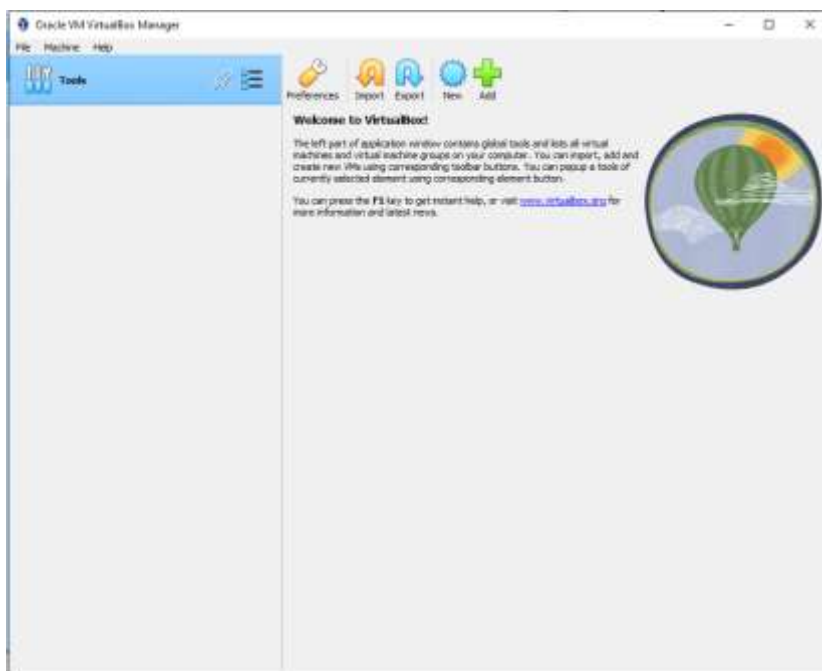


Рисунок 1.11 – Головне вікно VirtualBox

Установка Ubuntu на віртуальну машину

Завантажити ISO-образ Ubuntu можна на офіційній сторінці Ubuntu <https://ubuntu.com>.

На сторінці, яка завантажиться, оберіть вкладку Products, пункт Ubuntu Desktop для робочого столу, відкриється вікно, натисніть на «Download Ubuntu Desktop», якщо ви хочете завантажити образ Ubuntu Desktop <https://ubuntu.com/desktop> (рисунок 1.12).

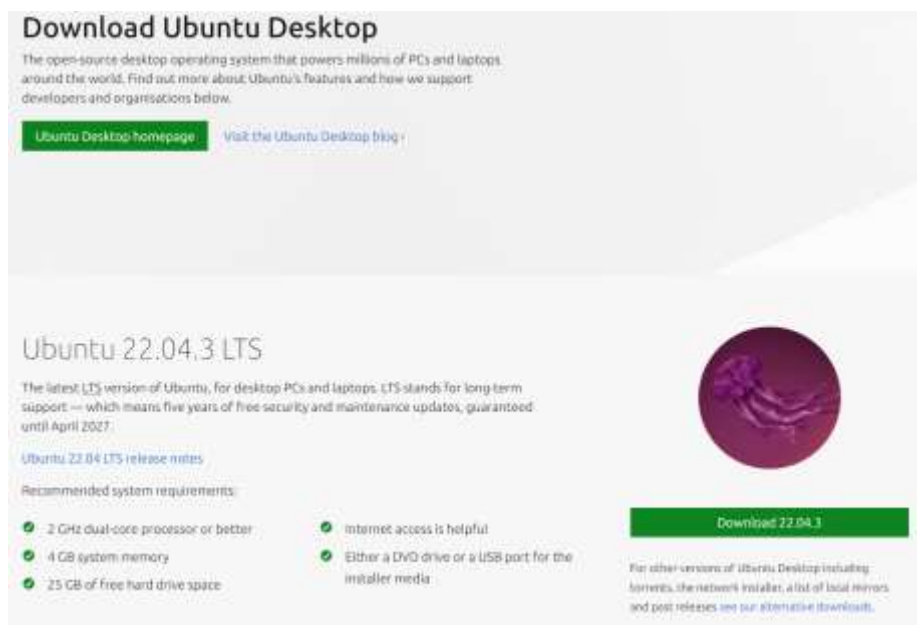


Рисунок 1.12 – Вікно завантаження Ubuntu Desktop

Якщо ви хочете завантажити образ Ubuntu Server, на сторінці, яка завантажиться, оберіть вкладку Products, пункт Ubuntu Server для сервера, відкриється вікно, натисніть на «Download Ubuntu Server» <https://ubuntu.com//server> (рисунок 1.13) .



Рисунок 1.13 - Вікно завантаження Ubuntu Server

Натисніть на Download... і завантаження буде відбуватись автоматично (рисунок 1.14) або можна це зробити в ручному режимі. Це може зайняти деякий час.



Рисунок 1.14 – Завантаження образу

Далі необхідно створити віртуальну машину. Для цього відкрийте VirtualBox і оберіть пункт меню Machine > New ... (рисунок 4.15)

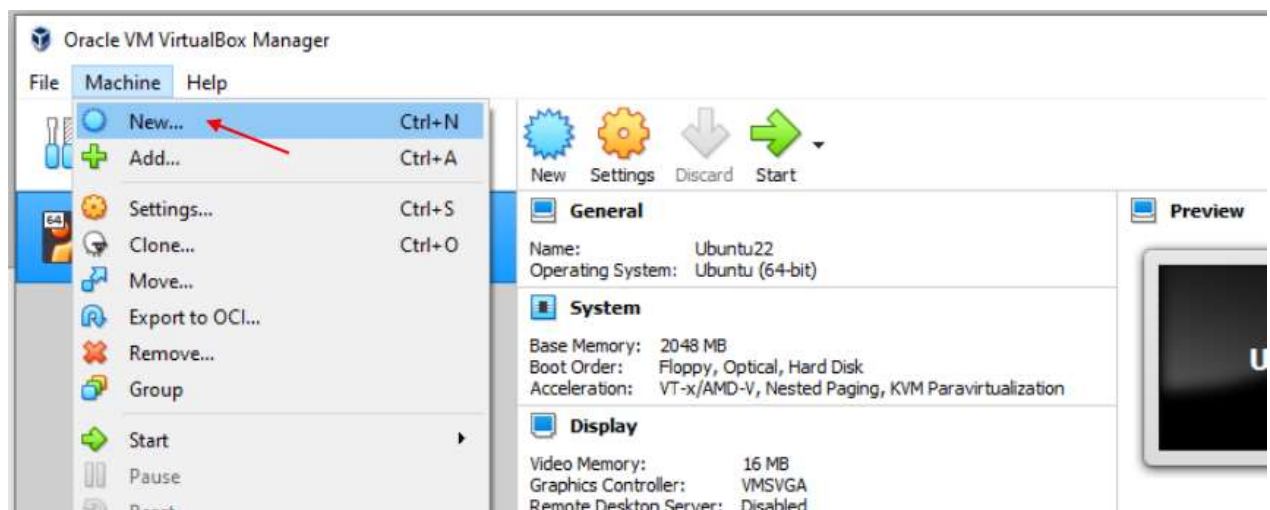


Рисунок 1.15 – Створення віртуальної машини

Відкриється вікно, в якому введіть ім'я для віртуальної машини, шлях, де буде створена віртуальна машина, виберіть тип Linux в спадаючому списку і Ubuntu (64-розрядної версії) в спадаючому списку «Версії». Після чого натисніть «Далі» (рисунок 1.16).

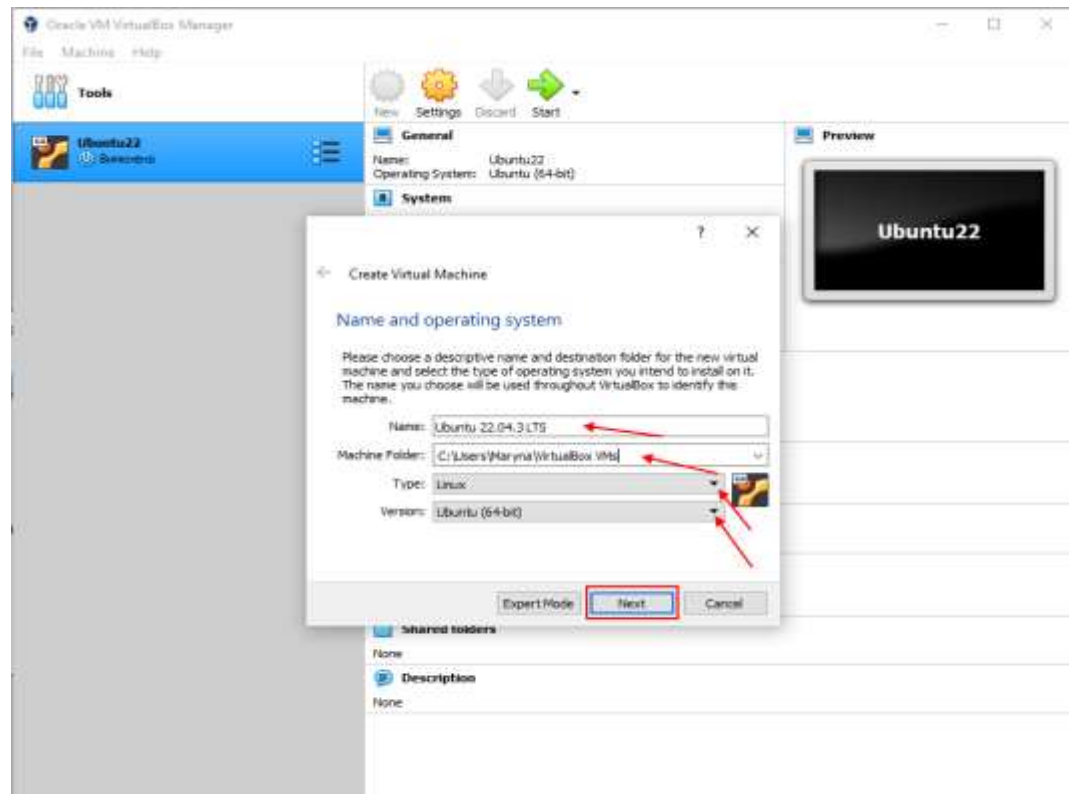


Рисунок 1.16 – Ім'я і операційна система

На наступному етапі ви повинні встановити розмір оперативної пам'яті для віртуальної машини. Для Ubuntu Desktop 22.04.3 LTS та 24.04.1 LTS він повинен бути не менше 2048 МБ (2 ГБ). Для Ubuntu Server 22.04.3 LTS та 24.04.1 LTS він повинен бути не менше 512 МБ. Натисніть «Далі» (рисунок 1.17).

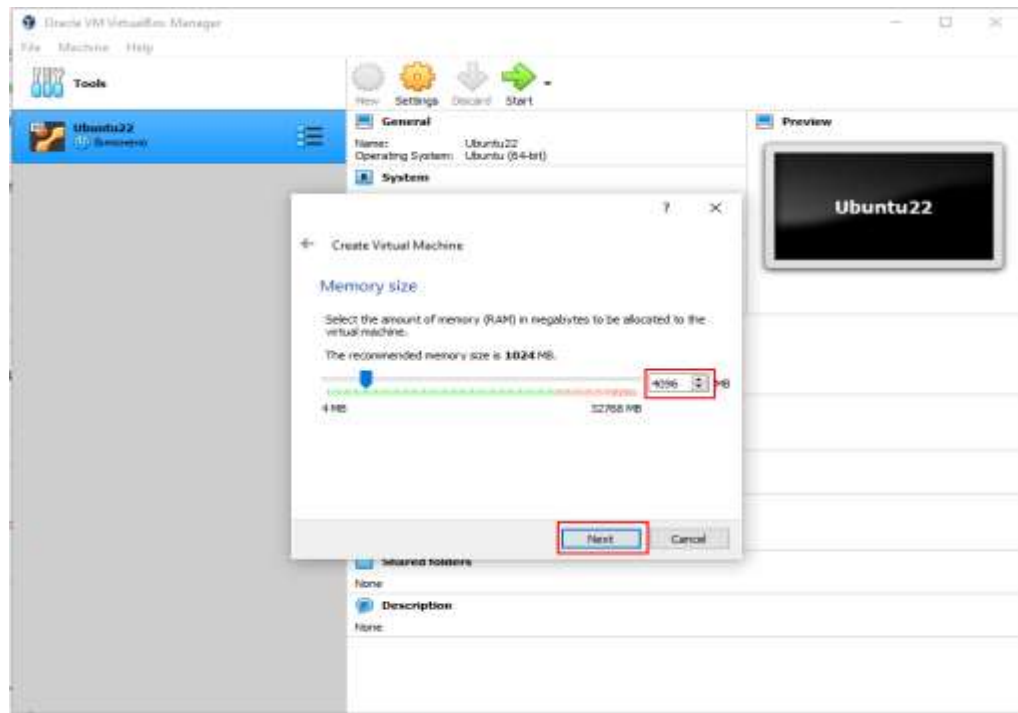


Рисунок 1.17 – Розмір оперативної пам'яті

В наступному вікні оберіть «Створити віртуальний жорсткий диск зараз» і натисніть «Створити» (рисунок 1.18).

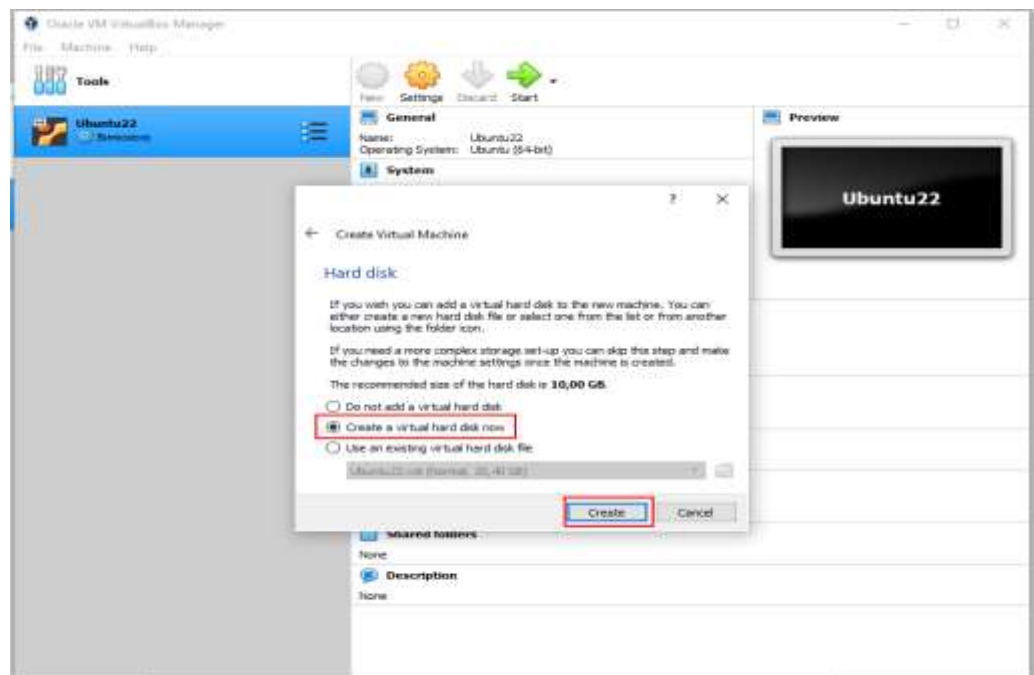


Рисунок 1.18 – Створення віртуального жорсткого диску

В наступному вікні потрібно обрати тип віртуального диску. Є наступні типи:

- Virtual Disk Image (VDI) - власний формат віртуальних дисків у VirtualBox, копіює існуючий жорсткий диск, включаючи всі дані та структурні елементи. Працює на хостах macOS, Windows, Linux та Solaris та підтримує різні гостьові операційні системи, включаючи Windows, DOS, Linux, Solaris та OpenSolaris, OS/2 та OpenBSD;

- Virtual Machine Disk (VMDK) - відкритий формат, який використовується в багатьох інших продуктах віртуалізації, наприклад в VMware, Hyper-V и VirtualBox;

- Virtual Hard Disk (VHD) - формат, розроблений Microsoft, включаючи Microsoft Virtual PC та Virtual Server.

В розширених налаштуваннях «Експертний режим» є додаткові типи:

- Parallels Hard Disk (HDD) - формат програми віртуалізації Parallels. Це віртуальна машина для запуску систем сімейства MS Windows на хості MacOS;

- QEMU Copy-On-Write (QCOW) - це швидкий кросплатформовий емулятор машини з відкритим вихідним кодом, який може емулювати безліч апаратних архітектур. Copy-On-Write - тіньове копіювання. Коли дані модифікуються, вихідний файл копіюється в тіньове сховище і може бути швидко відновлений в будь-який час. QCOW2 – це оновлена версія формату QCOW. QCOW2 підтримує шифрування AES. Відмінність від оригінальної версії полягає в тому, що QCOW2 підтримує кілька моментальних знімків, використовуючи гнучкішу модель їх зберігання;

- QEMU enhanced disk (QED) - заміна QCOW2, формат образу диска, який відмовляється від функцій, що є в qcow2, на користь вищого рівня продуктивності та цілісності даних.

Оберіть VDI (образ диска VirtualBox) і натисніть «Далі» (рисунок 1.19).

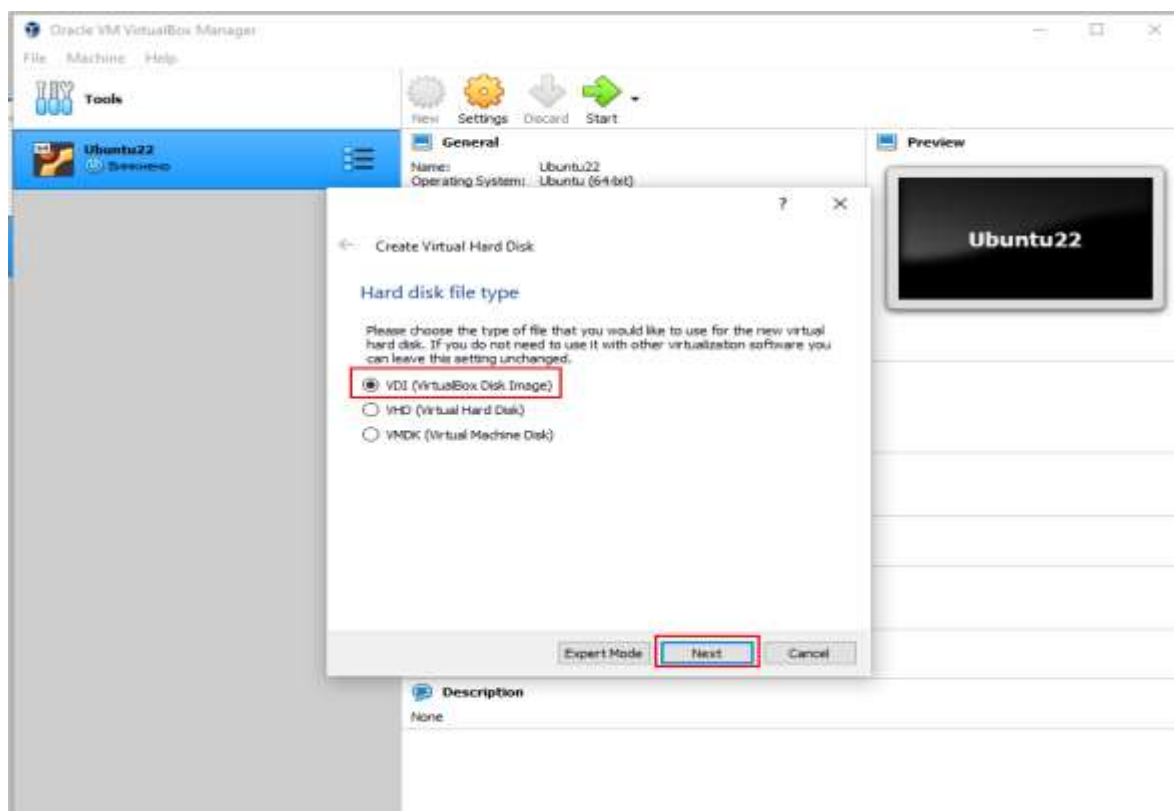


Рисунок 1.19 – Тип віртуального диску

Віртуальні диски, створені за допомогою VirtualBox, мають розширення .vdi. VDI переноситься і може працювати з іншими програмами віртуалізації. Він дозволяє використовувати як фіксований розмір сховища, так і сховище, що виділяється динамічно.

Динамічний диск - починається з мінімального розміру (поточна заповненість диску) та розширюється разом із розміром файлів у гостьовій ОС.

Віртуальний диск - статичний і не змінюється в розмірі. Одразу виділяється фіксований великий розмір.

В наступному вікні потрібно обрати тип сховища на фізичному жорсткому диску (рисунок 1.20).

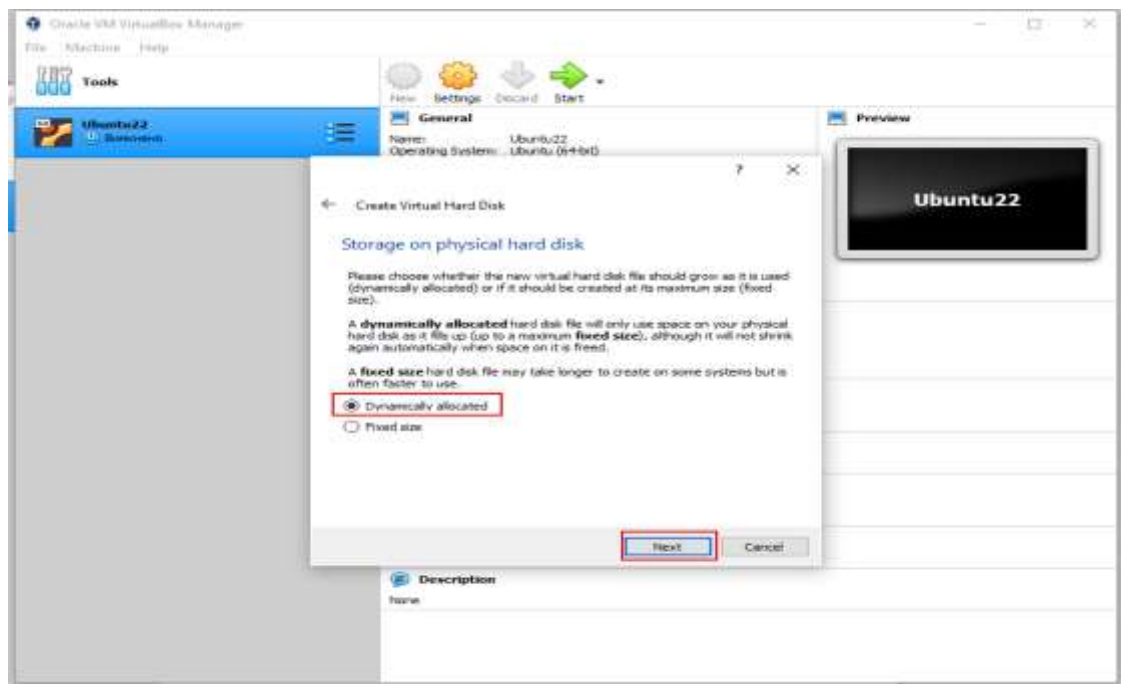


Рисунок 1.20 – Тип сховища на фізичному жорсткому диску

В прикладі обираємо «Динамічно розподілений» і натискаємо «Далі». Після цього встановить розмір віртуального жорсткого диска для віртуальної машини. Бажано встановити не менше 20,0 ГБ (рисунок 1.21).

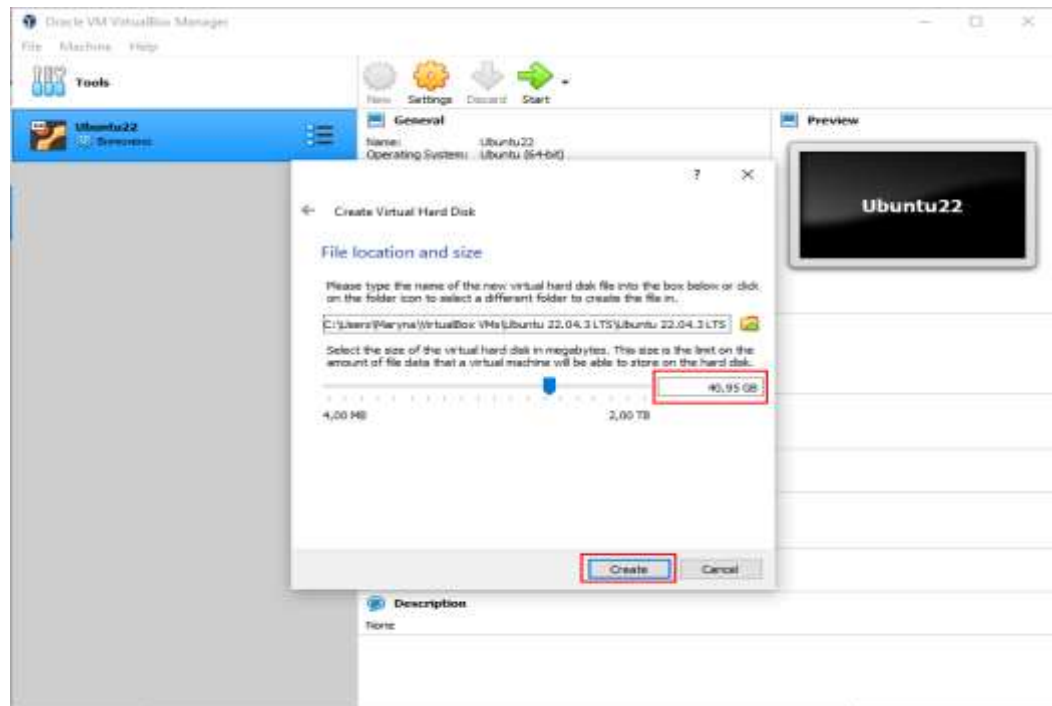


Рисунок 1.21 – Розмір віртуального жорсткого диска

Натисніть кнопку «Створити». В головному вікні VirtualBox ви побачите створену віртуальну машину (рисунок 1.22).

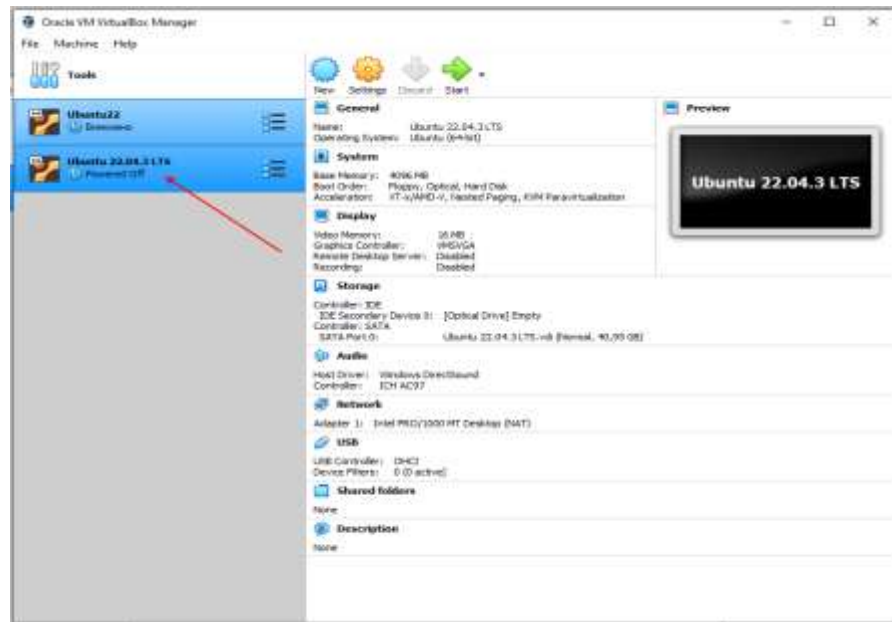


Рисунок 1.22 – Створена віртуальна машина

Далі потрібно обрати створену віртуальну машину і натиснути «Налаштування» для неї (рисунок 1.23).

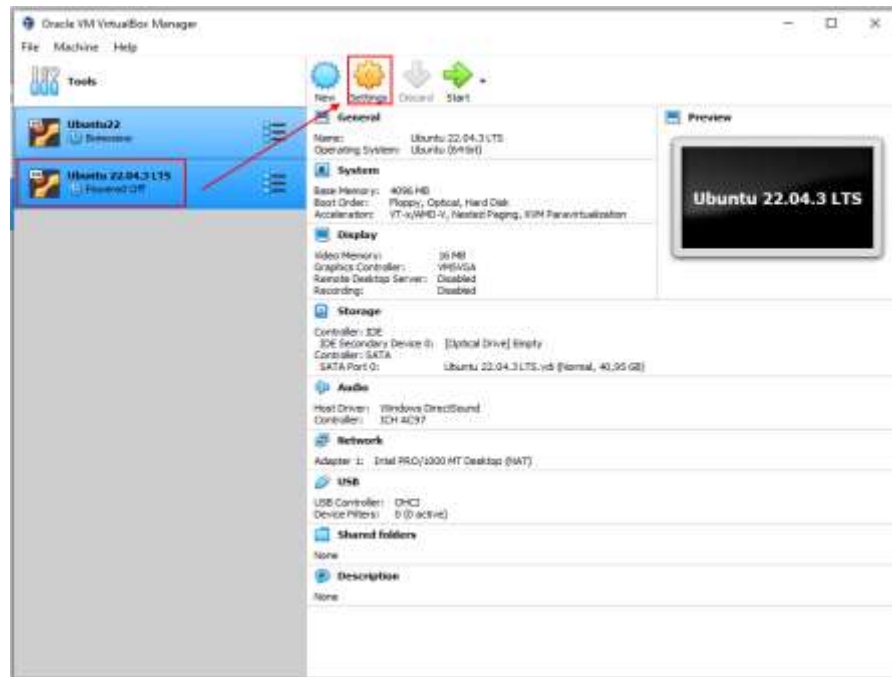


Рисунок 1.23 – Налаштування віртуальної машини

Наступним кроком є приєднання ISO-образу Ubuntu до віртуальної машини. Перейдіть на вкладку «Сховище», виберіть порожній пристрій IDE, клацніть значок компакт-диска і натисніть «Вибрати файл віртуального оптичного диска» Setting > Storage > Choose/Create a Virtual Optical Disk (рисунок 1.24) .

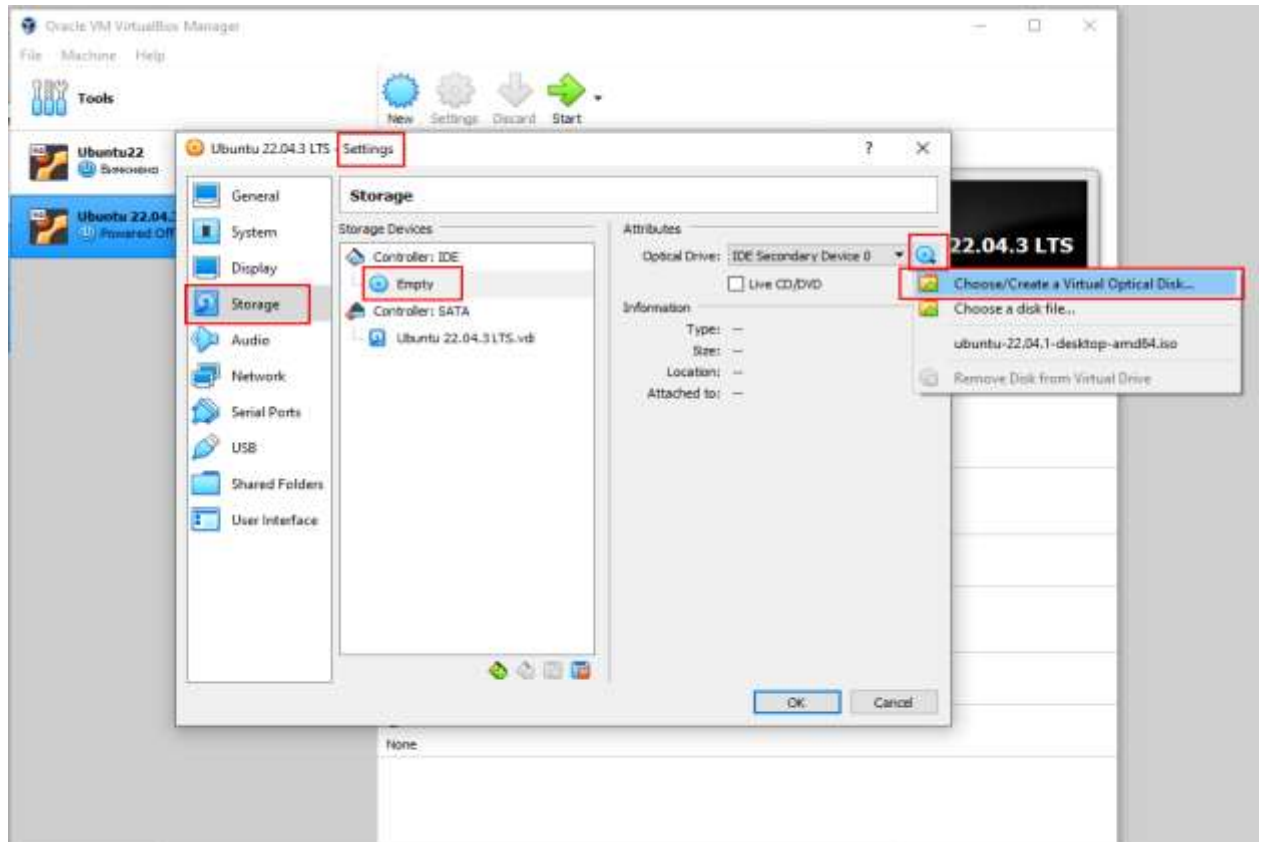


Рисунок 1.24 – Приєднання ISO-образу до віртуальної машини

В наступному вікні натисніть кнопку «Додати» та оберіть ISO-образ (рисунок 1.25).

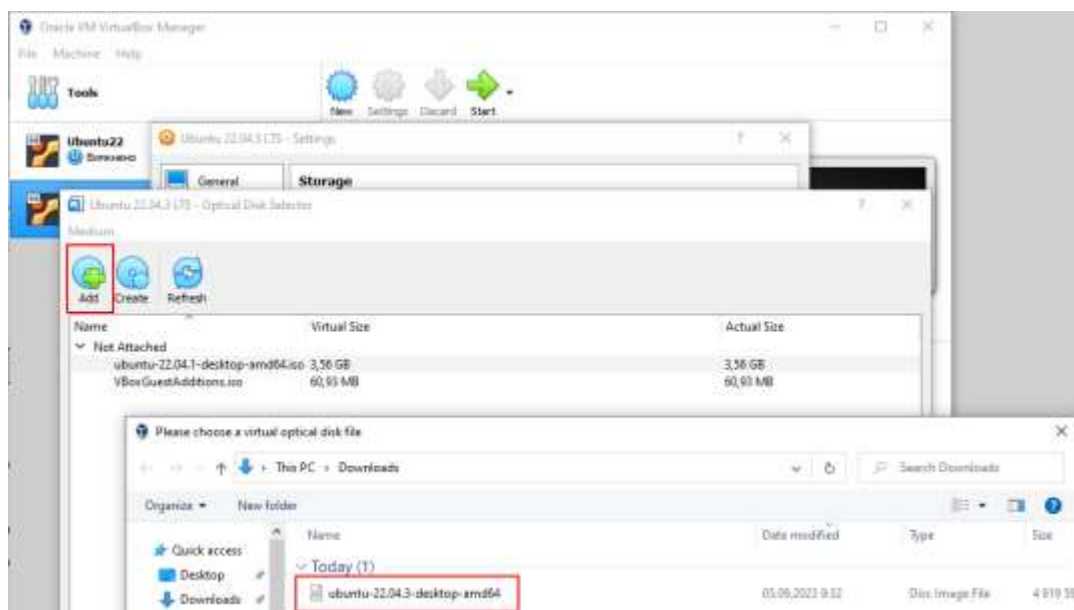


Рисунок 1.25 – Вибір ISO-образу

Натисніть кнопку «Обрати» (рисунок 1.26).

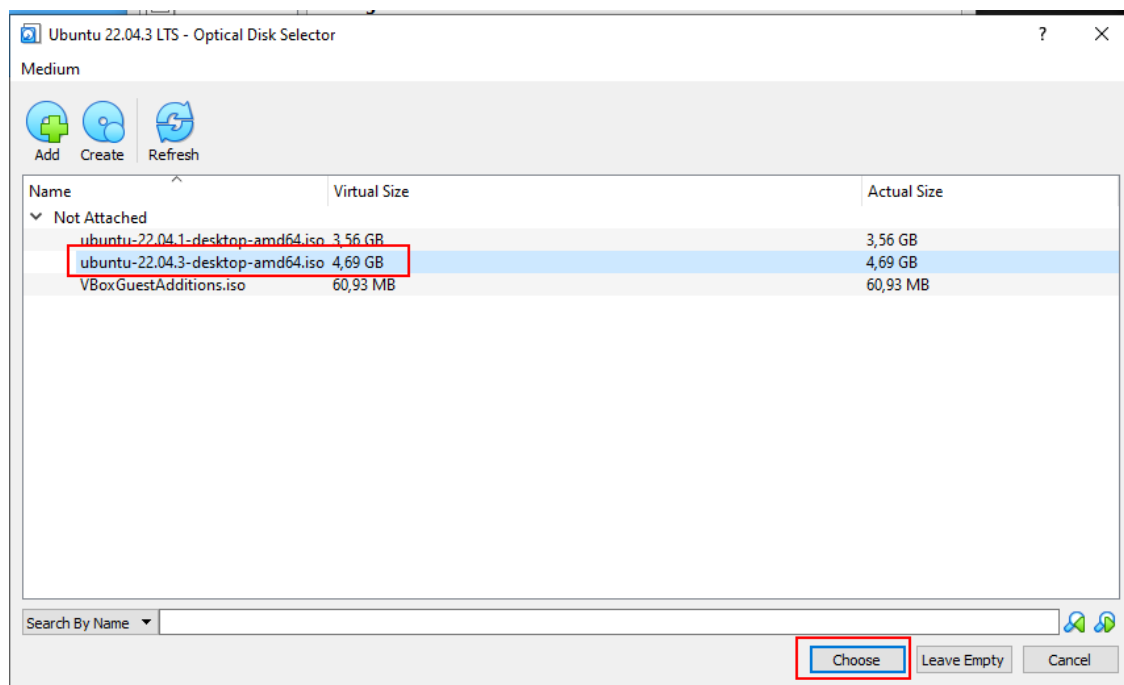


Рисунок 1.26 - Вибір ISO-образу

Оберіть підключений ISO-файл і натисніть «Так» (рисунок 1.27).

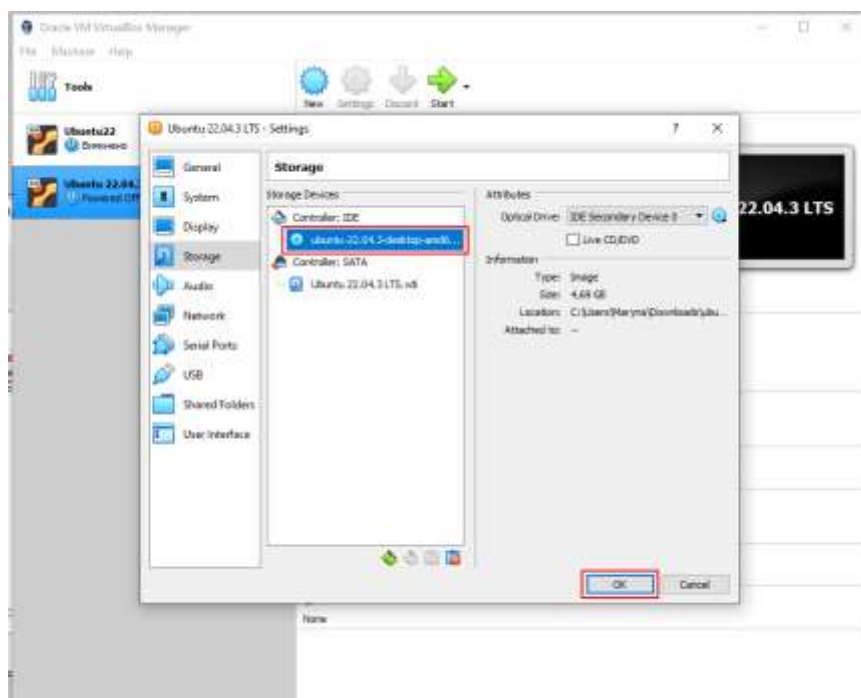


Рисунок 1.27 – Вибір під'єданого ISO-образу

Наступним кроком є запуск віртуальної машини:

Оберіть віртуальну машину і натисніть «Старт» (рисунок 1.28).

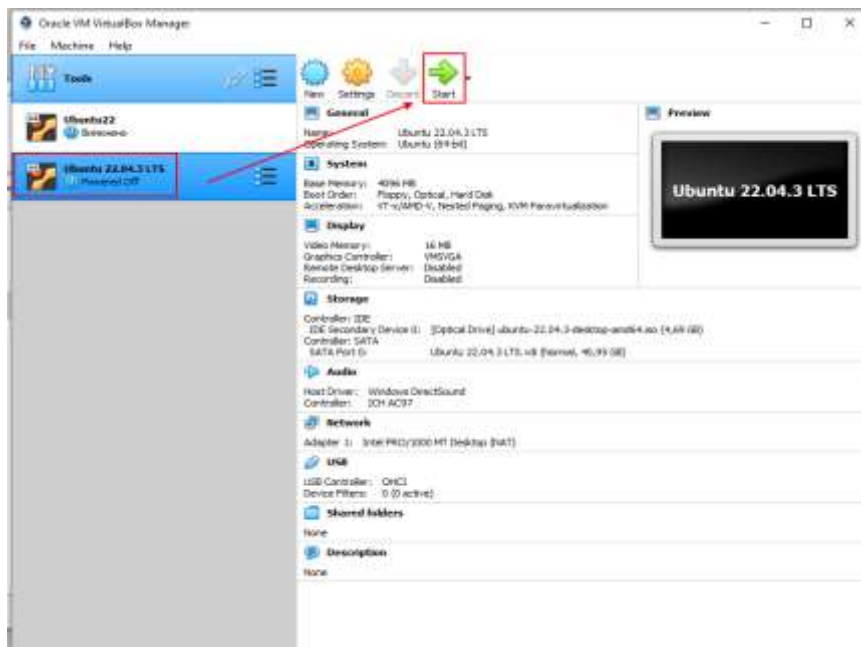


Рисунок 1.28 – Запуск віртуальної машини

Віртуальна машина повинна запускатися і завантажуватися з ISO-образу.
В даному прикладі Ubuntu 22.04.3 LTS (рисунок 1.29).

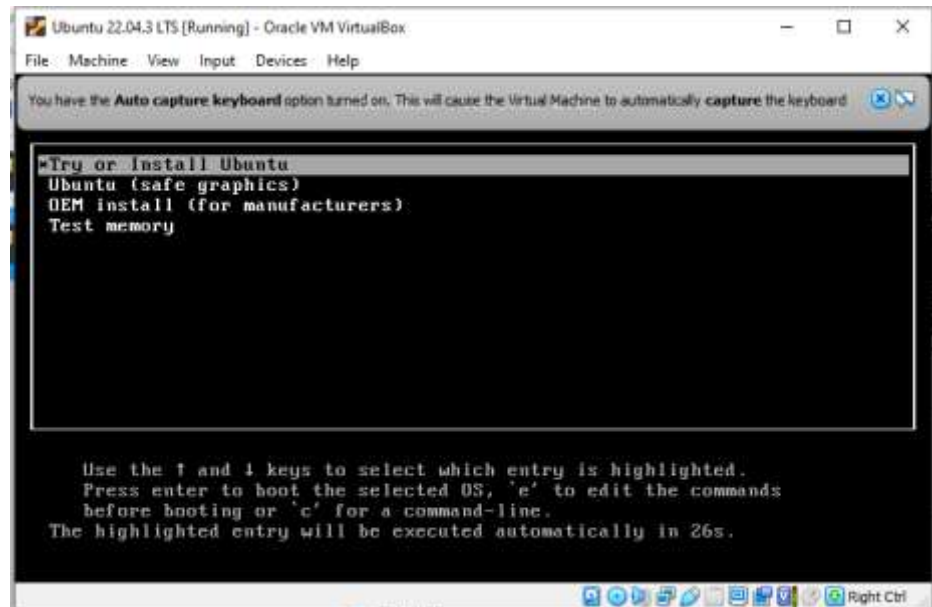


Рисунок 1.29 – Вибір інсталяції Ubuntu

Через деякий час установник Ubuntu 22.04.3 LTS повинен запуститися (рисунок 1.30). Тепер натисніть «Інсталювати Ubuntu».

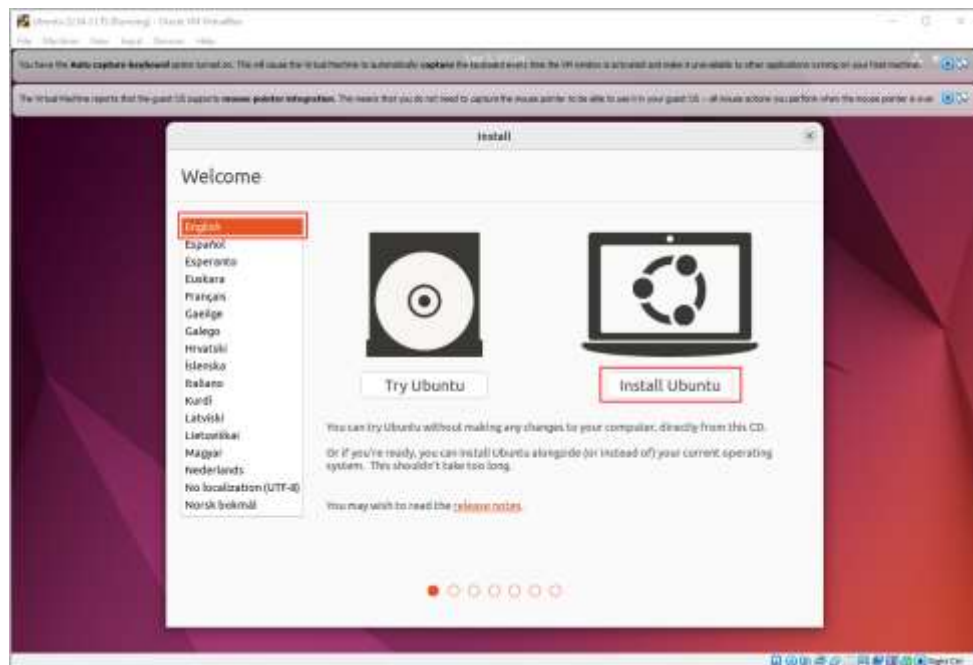


Рисунок 1.30 – Вікно-привітання установника Ubuntu

Наступним кроком є Установка Ubuntu у віртуальній машині:

Встановимо Ubuntu Desktop 22.04.3 LTS на віртуальну машину VirtualBox.

Якщо ви хочете встановити Ubuntu Server 22.04.3 LTS, процес установки буде трохи відрізнятися, так як Ubuntu Server 22.04.3 LTS використовує інсталятор на основі терміналу.

Оберіть розкладку клавіатури і натисніть «Продовжити» (рисунок 1.31).

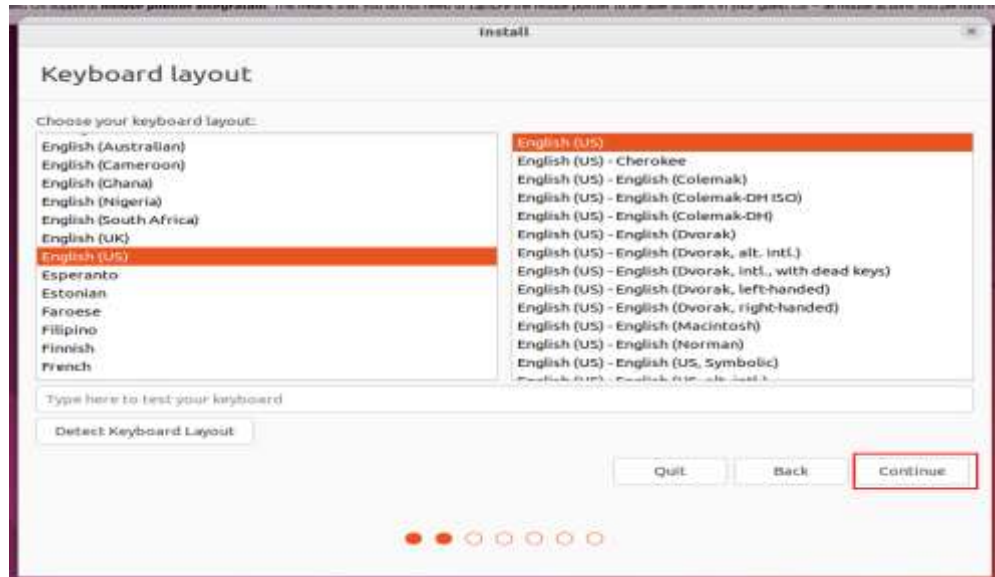


Рисунок 1.31 – Вибір розкладки клавіатури

Звідси ви можете виконати нормальну або мінімальну установку.

Нормальна установка йде з усіма застосунками, як зазвичай.

Мінімальна установка поставляється з обмеженою кількістю програм, що економить дисковий простір.

Якщо у вас є підключення до Інтернету на вашому комп'ютері, ви можете перевірити Завантаження оновлень при установці Ubuntu, щоб завантажити всі необхідні оновлення при установці Ubuntu на ваш комп'ютер.

Натисніть «Продовжити» (рисунок 1.32).

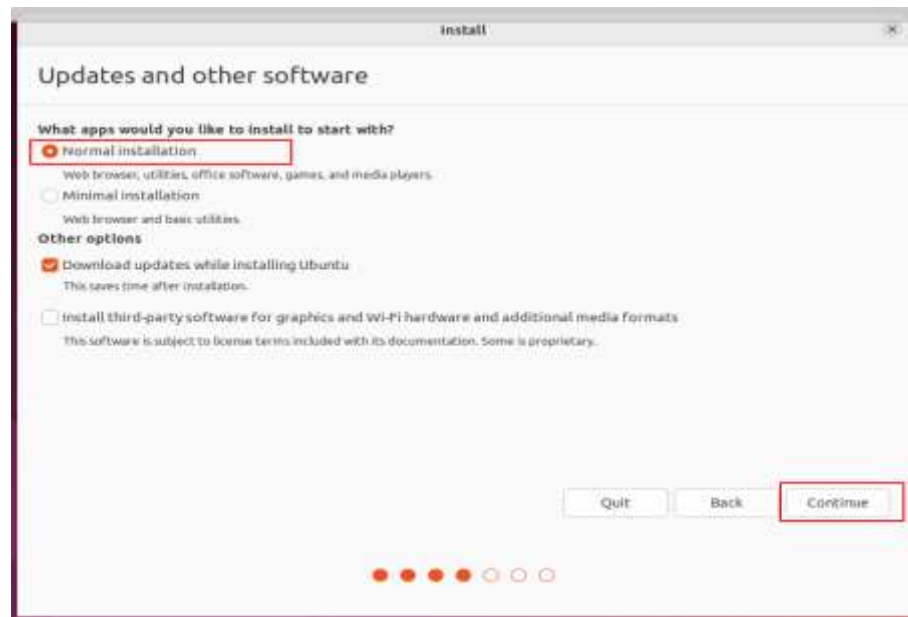


Рисунок 1.32 – Установки та оновлення Ubuntu

В наступному вікні оберіть «Очистити диск» та інстальювати Ubuntu», оскільки це віртуальна машина і натисніть «Встановити зараз». Установник Ubuntu автоматично створить всі необхідні розділи для роботи системи на вашому віртуальному жорсткому диску (рисунок 1.33).

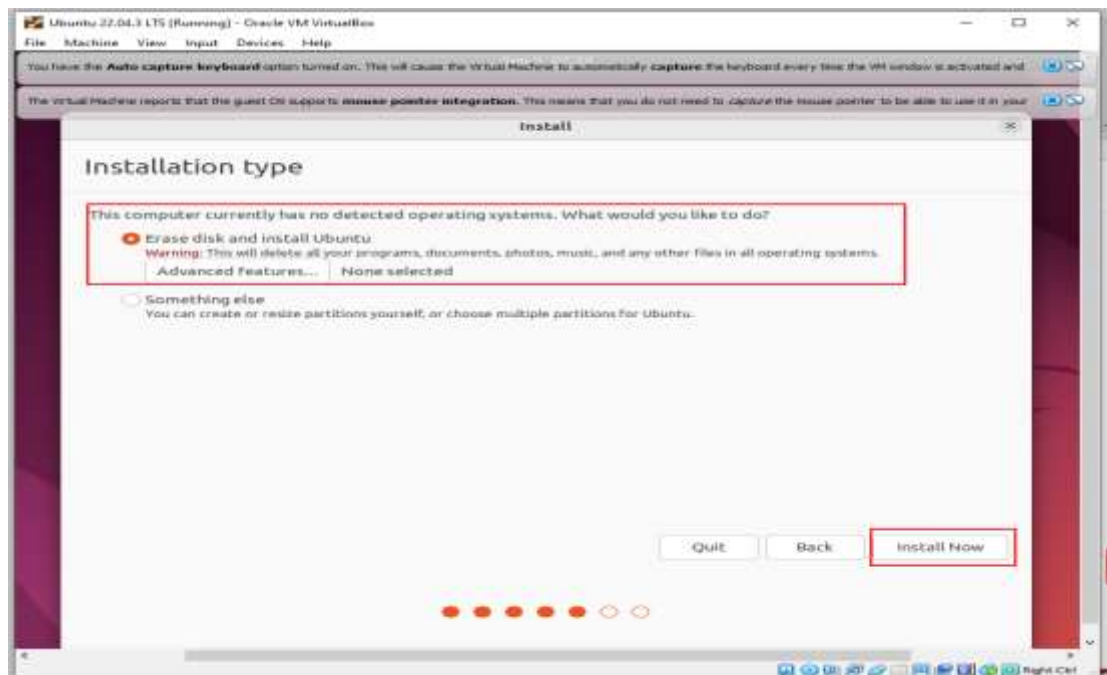


Рисунок 1.33 – Тип Інсталяції

Далі натисніть «Продовжити», щоб записати на диск зміни розподілу на розділи (рисунок 1.34).

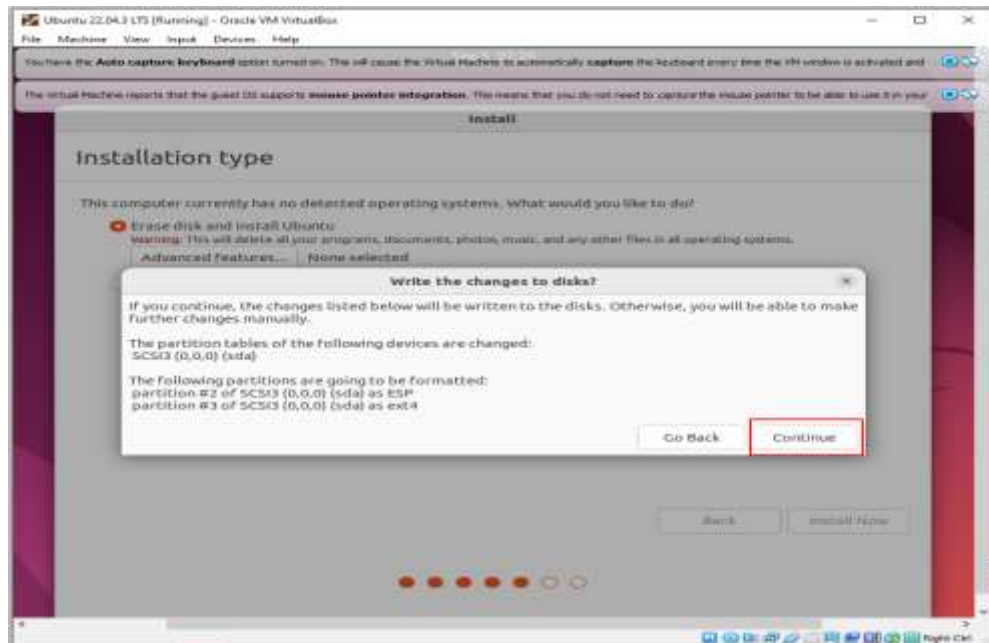


Рисунок 1.34 – Підтвердження розбивки диска на розділи

Оберіть часовий пояс і натисніть «Продовжити» (рисунок 1.35).

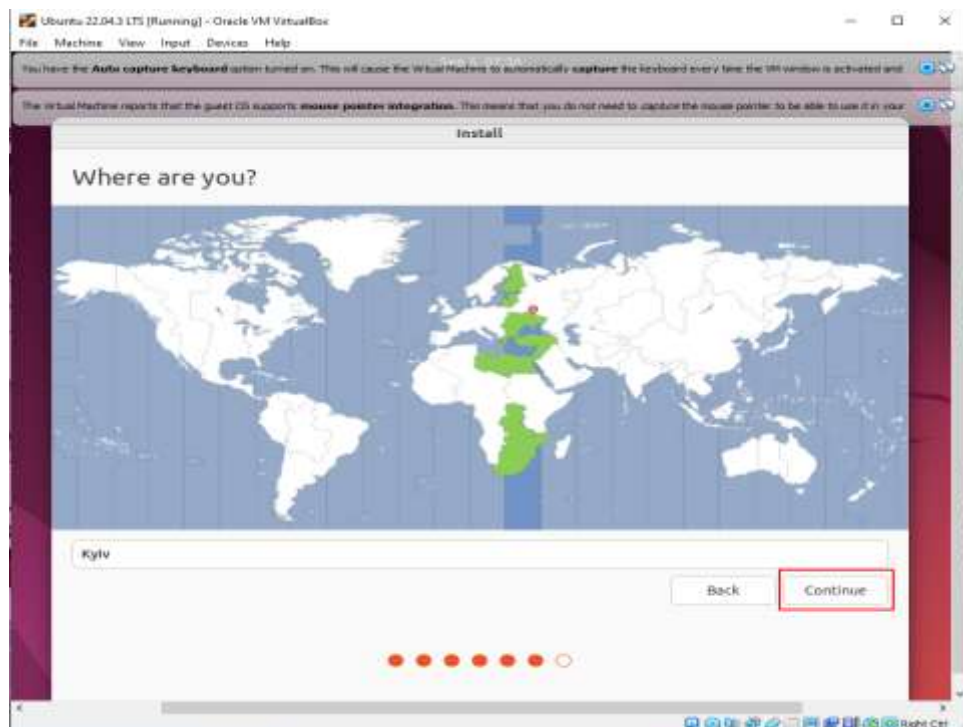


Рисунок 1.35 – Вибір часового поясу

В наступному вікні введіть свої особисті дані та натисніть «Продовжити» (рисунки 1.36-1.37).

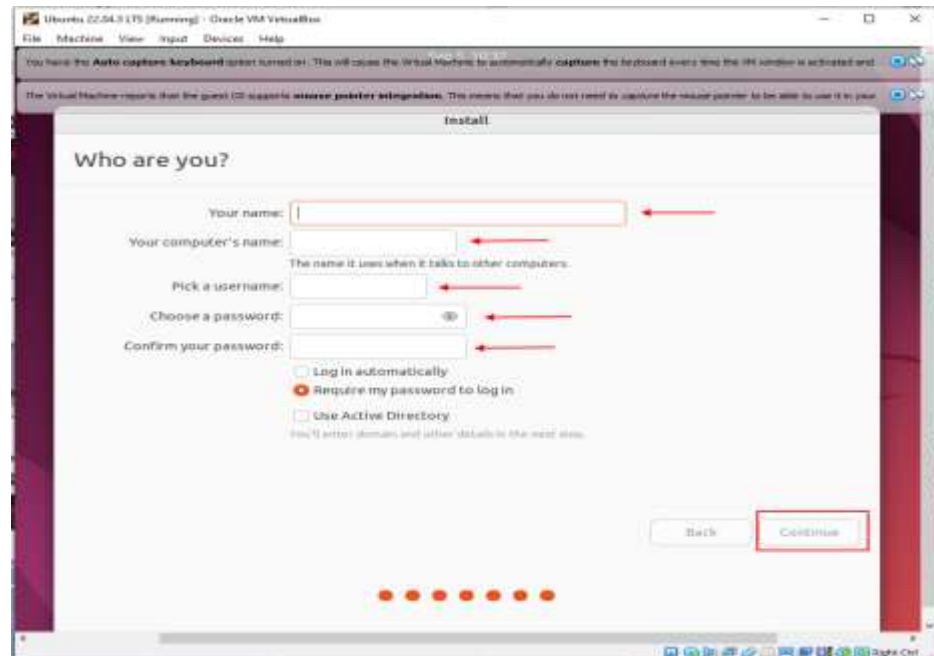


Рисунок 1.36 – Особисті дані

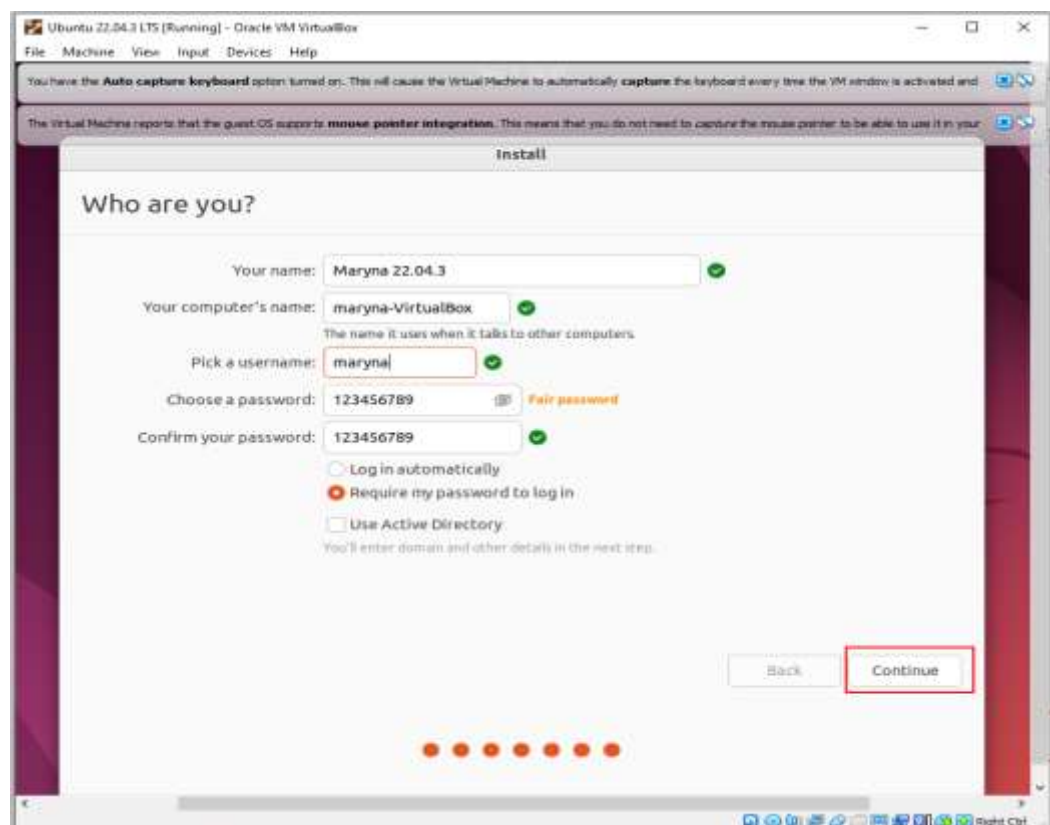


Рисунок 1.37 – Особисті дані

Установник Ubuntu повинен почати копіювання всіх необхідних файлів на ваш віртуальний жорсткий диск. Це може зайняти деякий час (рисунк 1.38).



Рисунок 1.38 – Копіювання даних на віртуальний диск

Після завершення установки натисніть «Перезавантажити зараз» (рисунк 1.39).

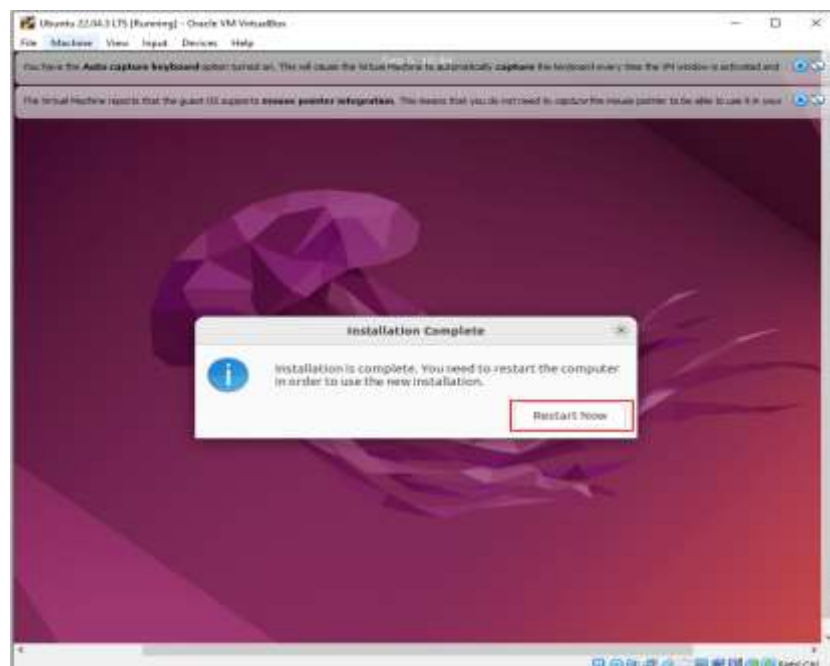


Рисунок 1.39 - Перезавантаження системи

Після завантаження віртуальної машини ви повинні побачити наступне меню GRUB (завантажувач операційної системи). Оберіть користувача і натисніть Enter (рисунок 1.40).

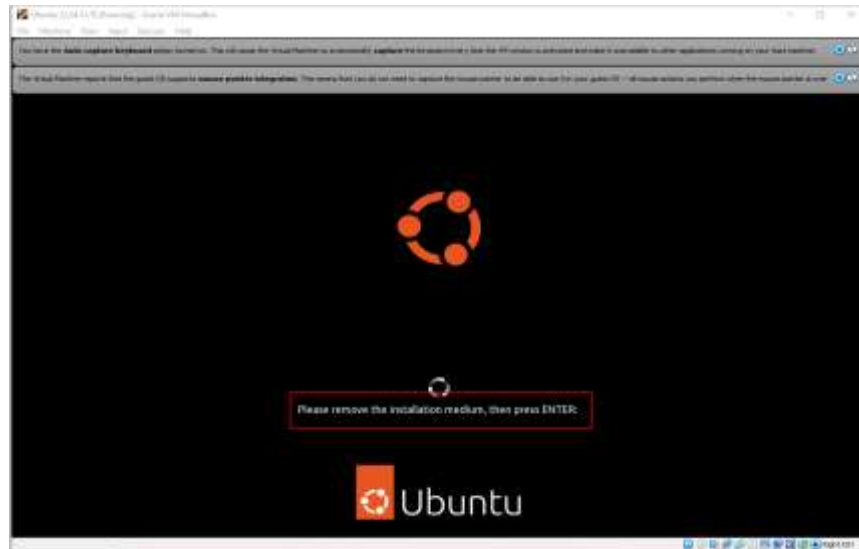


Рисунок 1.40 – Меню GRUB

Оберіть користувача і натисніть Enter (рисунок 1.41).

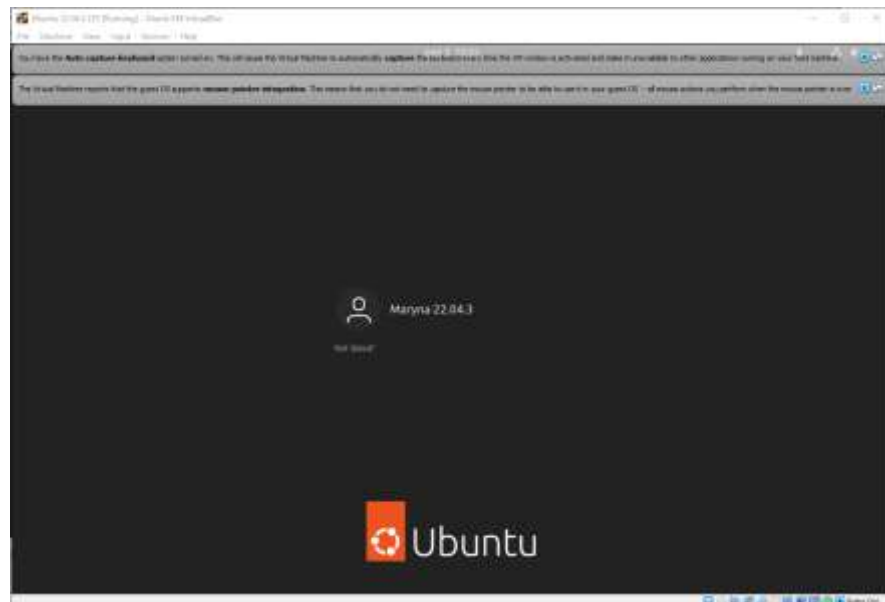


Рисунок 1.41 – Вибір користувача для входу в систему

Тепер ви можете увійти, використовуючи пароль, який ви встановили під час установки системи (рисунок 1.42).

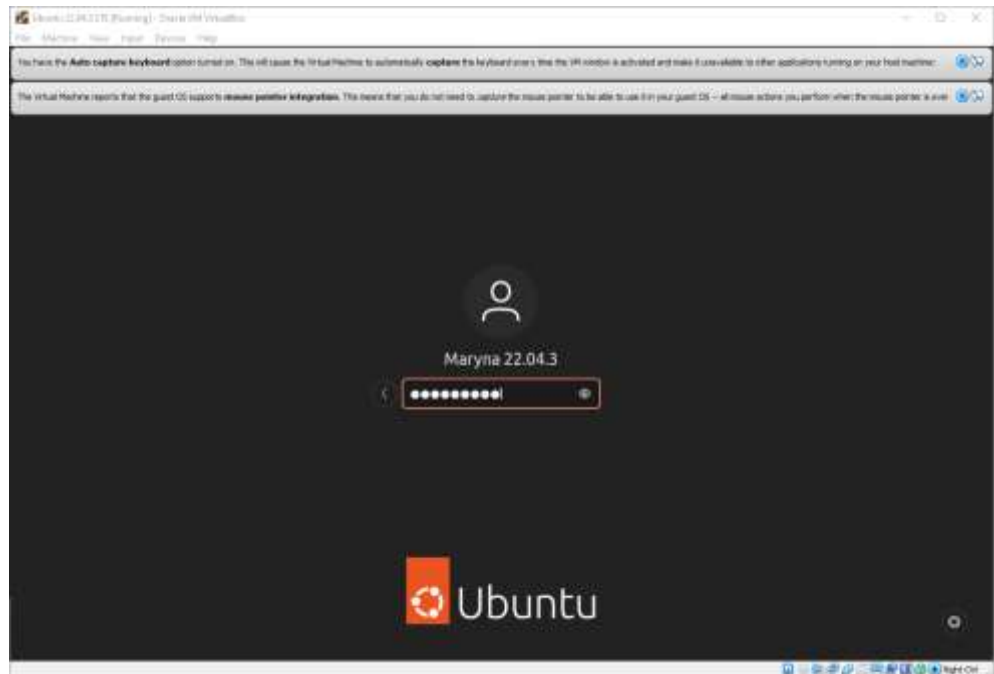


Рисунок 1.42 – Вікно вводу пароля користувача

Після входу в систему ви зможете використовувати Ubuntu (рисунок 1.43).



Рисунок 1.43 – Успішний вхід в систему

Для оновлення системи Linux, використовуються менеджери пакетів - це набір програмного забезпечення для установки, оновлення, видалення програм [7]. Для різних дистрибутивів використовуються свої пакетні менеджери (Dpkg - в Debian подібних системах, Yum та RPM - в Red Hat подібних системах, Pacman - в Arch Linux і т.д.).

Для оновлення системи Ubuntu, використовується менеджер пакетів – apt.

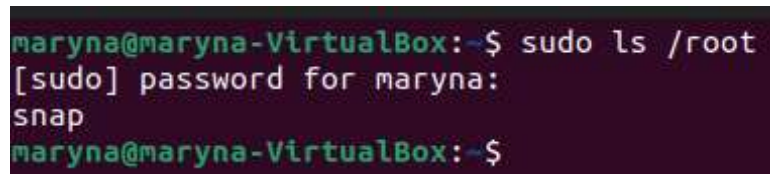
Dpkg (Debian Package) - це система управління пакетами. Apt (Advanced Packaging Tool) - надбудова над командним рядком, дозволяє працювати із системою управління пакетами dpkg, А саме – шукати їх, завантажувати, оновлювати, видаляти програмні пакети в операційних Ubuntu, Mint.

Оновлення системи, установку програм необхідно виконувати в правами адміністратора/суперкористувача. В Linux системах це користувач root, який створюється системою автоматично при установці.

Root або суперкористувач - це спеціальний обліковий запис і група користувачів з ідентифікатором, власник якого має право на виконання всіх операцій.

Команди sudo (рисунок 1.44) і su (рисунок 1.45) - це два різні способи отримати привілеї суперкористувача (root), що володіє максимальними правами.

Основна відмінність між su і sudo полягає в тому, що su вимагає пароль цільового облікового запису (наприклад, користувача root) і перемикає вас на нього, в той час як sudo вимагає пароль поточного користувача і запускає від його імені лише одну (або кілька) команд, для виконання яких потрібні права користувача root.



```
maryna@maryna-VirtualBox:~$ sudo ls /root
[sudo] password for maryna:
snap
maryna@maryna-VirtualBox:~$
```

Рисунок 1.44 – Використання команди sudo

Команда зміни користувача:

su - username і натиснути клавішу Enter.

Залогінітись як root без зміни середовища

su

зі зміною середовища (додати тире)

su -

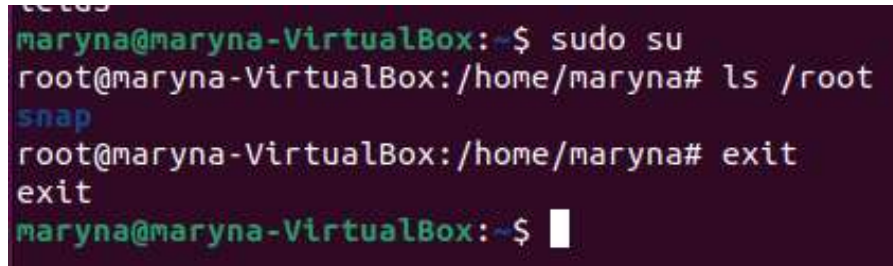
A screenshot of a terminal window with a dark background. The prompt is 'maryna@maryna-VirtualBox:~\$'. The user enters 'sudo su', and the prompt changes to 'root@maryna-VirtualBox:/home/maryna#'. The user then enters 'ls /root', and the output 'snap' is displayed. Next, the user enters 'exit', and the prompt returns to 'maryna@maryna-VirtualBox:~\$'.

Рисунок 1.45 – Використання команди su

Оновити систему, встановити програмне забезпечення, видалити програмне забезпечення можна кількома способами, один з варіантів - з пункту меню "Центр застосунків", але можна це зробити командами:

- `sudo apt update` - синхронізує інформацію з репозиторію з локальним кешем. Оновлення до наступної версії. Цю команду необхідно виконувати перед установкою будь-яких пакетів;
- `sudo apt upgrade` - оновлення до останньої версії без видалення/оновлення всіх залежностей;
- `sudo apt full-upgrade` - оновить всі пакети, для яких вийшли нові версії, в тому числі встановить необхідні залежності і видалить пакети, які заважають, а також оновиться ядро системи, якщо вийшла нова версія;
- `apt install program` - встановити / оновити program;
- `apt remove program` - видалити program, зберігши файли конфігурації;
- `apt purge program` - видалити program і файли конфігурації;
- `apt list` – список пакетів;

- apt search program – пошук пакетів за іменем;
- apt show program – детальна інформація про пакет (рисунок 1.46).

```
maryna@maryna-VirtualBox:/$ apt show nano
Package: nano
Version: 6.2-1ubuntu0.1
Priority: standard
Section: editors
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Jordi Mallach <jordi@debian.org>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 881 kB
Depends: libc6 (>= 2.34), libncursesw6 (>= 6), libtinfo6 (>= 6)
Suggests: hunspell
Conflicts: pico
Breaks: nano-tiny (<< 2.8.6-2)
Replaces: nano-tiny (<< 2.8.6-2), pico
Homepage: https://www.nano-editor.org/
Task: standard
Download-Size: 280 kB
APT-Manual-Installed: no
APT-Sources: http://ua.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages
Description: small, friendly text editor inspired by Pico
```

Рисунок 1.46 – Перегляд детальної інформації про програму nano

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу.

ЛАБОРАТОРНИЙ ПРАКТИКУМ №2 Інтерпретатор. Команди інтерпретатора. Змінні оточення, оболонки

Мета: навчитись працювати в командній оболонці, ознайомитись та опрацювати вбудовані команди середовища, створювати, редагувати змінні середовища та оболонки, які використовуються для налаштування системи.

Завдання

Зайти в систему під користувачем, якого створено в попередній роботі. В терміналі опрацювати команди: env, export, who, path, alias, bind, cd, dirs, echo, enable, history, man, pwd, date, uname.

Теоретичні відомості

Однією з обов'язкових складових операційної системи Linux є **командна оболонка**, вона ж – **інтерпретатор мови команд** або **shell**. Будь-які дії в будь-якій системі виконуються прямо або за допомогою різноманітних менеджерів, шляхом віддачі відповідних командних директив. І тому інтегруючою надбудовою над всім виступає командна оболонка. Різні програми, платформи з інтерфейсами - лише надбудови над елементарними shell-командами і їх комбінаціями.

Всі команди, які набираються в командному рядку, можуть бути використані в командних файлах, що виконуються інтерпретатором shell, і навпаки.

Дії, що виконуються в командному інтерпретаторі, в подальшому можуть бути оточені графічною оболонкою і деталі їх виконання, таким чином, виявляються прихованими від кінцевого користувача.

При кожному вході в систему користувач потрапляє в середовище так званого домашнього інтерпретатора користувача, що виконує налагоджувальні

дії для користувацької сесії і в подальшому здійснює інтерактивне спілкування з користувачем.

Вихід з призначеної для користувача сесії завершує роботу інтерпретатора і породжених з нього процесів.

Існує цілий набір командних інтерпретаторів [2], здатних бути користувацькою оболонкою і засобом виконання командних файлів:

- sh - Bourne-Shell, прашур всіх інших командних інтерпретаторів;
- csh - C-Shell, інтерпретатор для системи BSD з C-подібним синтаксисом керуючих операторів. Має розширені засоби для роботи в інтерактивному режимі, засоби управління завданнями, але робота з командними файлами не надто коректна;
- ksh - Korn-Shell, інтерпретатор, розроблений Девідом Корном (David Korn). Програмно сумісний з Bourne-Shell, включає в себе засоби редагування командних рядків;
- bash - Bourne-Again-Shell, в достатній мірі сумісний з Bourne-Shell, що включає в себе як засоби інтерактивної роботи, запропоновані в C-Shell, так і редагування командних рядків;
- tcsh - Tenex-C-Shell, подальший розвиток C-Shell з розширеним інтерактивним інтерфейсом і дещо поліпшеним програмним;
- zsh - Z-Shell, включає в себе всі напрацювання Bourne-Again-Shell і Tenex-C-Shell, а також деякі суттєві їх розширення (втім, не такий популярний, як останні);
- pdksh - Public-Domain-Korn-Shell, вільно розповсюджуваний аналог Korn-Shell з деякими доповненнями.

Таким чином, інтерфейс командного рядка Linux називається «Shell» (оболонка). Оболонка являє собою інтерфейс між користувачем і операційною системою.

bash – один з інтерпретаторів командного рядка, який виконує команди, одержувані зі стандартного потоку введення або з файлу.

Список встановлених в вашій системі оболонок знаходиться в файлі **/etc/shells**.

/etc – це один з найбільш важливих каталогів систем Linux, бо він містить всі конфігураційні файли системи, індивідуальні для кожного хоста.

Оболонка знаходиться між користувачем та операційною системою.

Оболонка забезпечує оточення (середовище) всередині вікна терміналу (Ctrl+Alt+T), яке дозволяє вводити команди та запускати програми.

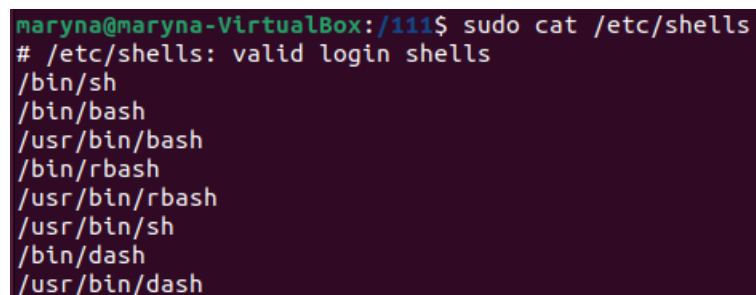
Оточення - це область, яку оболонка створює кожен раз при запуску сеансу, що містить змінні, що визначають системні властивості (часовий пояс в системі, шляхи до певних файлів, додатки за замовчуванням, локалі тощо). Змінні оточення також можуть використовуватися в програмах оболонки або в підоболонках для виконання різних операцій.

У Linux є багато різних оболонок-інтерпретаторів (bash, dash, ksh...). Усі вони дозволяють виконувати одні й ті ж основні завдання:

- досліджувати файлову систему;
- працювати з файлами;
- запускати програми;
- запускати сценарії.

Встановлені оболонки в системі – **/etc/shells**.

Командою **cat** можна переглянути вміст файлу /etc/shells (рисунок 2.1)



```
maryna@maryna-VirtualBox:/111$ sudo cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/usr/bin/sh
/bin/dash
/usr/bin/dash
```

Рисунок 2.1 – Вміст файлу /etc/shells

Також існує змінна оточення **\$SHELL**, яка містить назву поточної оболонки (наприклад **bash**) та шлях до виконуваного файлу. Значення змінної можна вивести за допомогою команди **echo**, коли ви звертаєтесь до змінної, перед ім'ям змінної обов'язково потрібно поставити знак **\$** (рисунок 2.2).

```
maryna@maryna-VirtualBox:/111$ echo $SHELL  
/bin/bash
```

Рисунок 2.2 – Значення змінної **SHELL**

Для запуску іншої оболонки потрібно в терміналі написати назву оболонки, наприклад **dash** (рисунок 2.3). Ця зміна діє в рамках поточної сесії або до моменту зміни оболонки на іншу

```
maryna@maryna-VirtualBox:/111$ dash  
$ echo new shell dash  
new shell dash
```

Рисунок 2.3 – Запуск оболонки **dash**

Для зміни поточної оболонки в конфігураційному файлі, використовується команда **chsh**

chsh шлях до нової оболонки (приклад, **chsh -s /usr/bin/dash**)

-s --shell оболонка - ім'я нової оболонки користувача.

Якщо встановити порожнє значення, то буде використано оболонку за замовчуванням.

Змінні оточення та змінні оболонки

Змінні мають наступний вигляд:

KEY=value

KEY="Some other value"

KEY=value1: value2

При створенні змінних необхідно дотримуватись наступних правил:

- змінні оточення повинні бути написані великими літерами (UPPER CASE). Імена змінних чутливі до регістру;
- якщо змінній присвоєно кілька значень, то значення розділяються двокрапкою;
- навколо символу = немає пробілу.

Змінні можна розділити на дві категорії:

- змінні оточення (Environmental Variables) – це змінні, які визначені для поточної оболонки і успадковуються будь-якими пов'язаними оболонками або процесами. Змінні оточення використовуються для передачі інформації в процеси, які породжуються з оболонки;
- змінні оболонки (Shell Variables) – це змінні, які містяться виключно в оболонці, в якій вони були встановлені або визначені. Вони часто використовуються для відстеження даних, наприклад, поточного робочого каталогу.

Змінні оточення (або змінні середовища) – це набір пар «ключ-значення», визначені оболонкою, які зберігаються в вашому Linux і використовуються процесами (програмами) для виконання певних операцій. Вони відповідають за стандартну поведінку системи і додатків.

При взаємодії з вашою системою через оболонку, інформацію обробляє оболонка, щоб визначити поведінку системи та доступи.

Деякі з цих параметрів (змінна=значення) містяться в налаштуваннях конфігурації системи, а інші визначаються призначеним для користувача введенням.

Системні змінні оточення Linux визначаються системою і використовуються програмами системного рівня.

Оболонка відстежує всі ці параметри і налаштування через оточення.

Змінні оточення за радіусом їхньої дії та часом «життя» можуть бути трьох типів :

- локальні змінні оточення;

Ці змінні визначені лише для поточної сесії. Вони будуть безвідновно стерті після завершення сесії, незалежно від того, чи це віддалений доступ, чи локальний термінал. Вони не зберігаються ні в яких файлах, а створюються і видаляються за допомогою спеціальних команд.

- користувальницькі змінні оточення;

Ці змінні оточення в Linux визначаються для конкретного користувача і завантажуються кожного разу, коли користувач входить в систему за допомогою локального терміналу або ж підключається віддалено. Такі змінні, як правило, зберігаються в файлах конфігурації: `.bashrc`, `.bash_profile`, `.bash_login`, `.profile` або в інших файлах (ім'я файлу залежить від дистрибутива), розміщених в директорії користувача.

- системні змінні оточення.

Ці змінні доступні і діють в рамках всієї системи, для всіх користувачів. Вони завантажуються при старті системи з системних файлів конфігурації: `/etc/environment`, `/etc/profile`, `/etc/profile.d/` `/etc/bash.bashrc` ((ім'я файлу залежить від дистрибутива).

Налаштування змінних середовища визначається за допомогою загальносистемних і користувацьких файлів конфігурації, наприклад в Ubuntu, `/etc/profile` (рисунок 2.4) і `.bashrc` (рисунок 2.5) в `/etc` та в домашньому каталозі користувача (рисунок 2.6).

```
maryna@maryna-VirtualBox:/111$ cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
```

Рисунок 2.4 – Конфігураційний файл /etc/profile

Системні налаштування:

- /ect;
- /etc/profile;
- /etc/bash.bashrc;
- /etc/bashrc;
- /etc/password.

```
maryna@maryna-VirtualBox:/111$ cat /etc/bash.bashrc
# System-wide .bashrc file for interactive bash(1) shells.

# To enable the settings / commands in this file for login shells
# this file has to be sourced in /etc/profile.

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

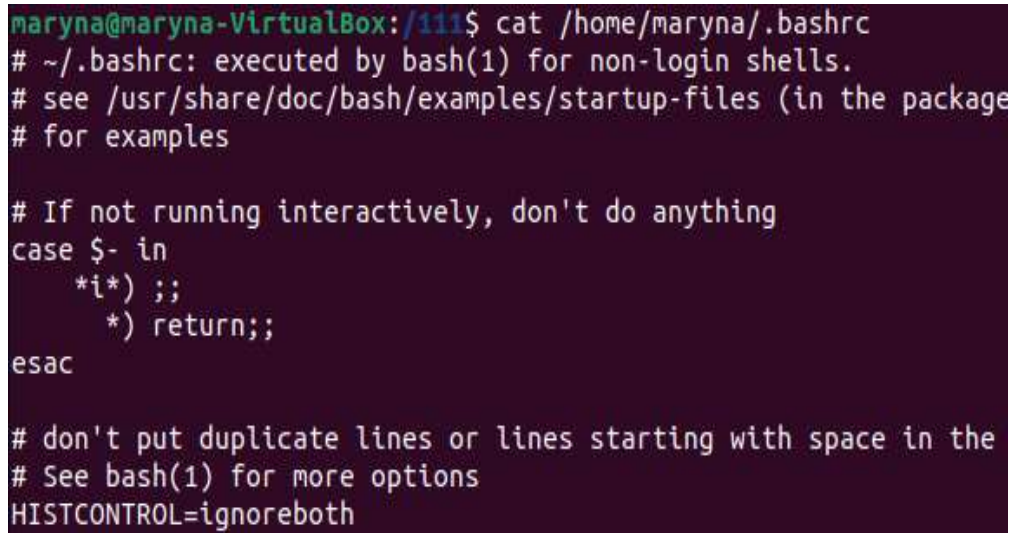
# set variable identifying the chroot you work in (used in the pro
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
  debian_chroot=$(cat /etc/debian_chroot)
```

Рисунок 2.5 – Конфігураційний файл /etc/ bash.bashrc

Користувацькі налаштування:

- /home/user;
- /home/user/.bashrc;

- /home/user/bash-profile;
- /home/user/.bash-logout.



```
maryna@maryna-VirtualBox:/111$ cat /home/maryna/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the
# See bash(1) for more options
HISTCONTROL=ignoreboth
```

Рисунок 2.6 – Конфігураційний файл /home/user/ bash.bashrc

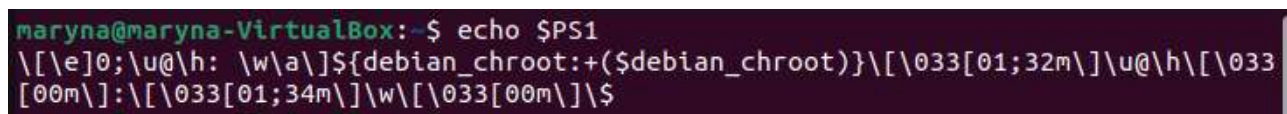
Інтерпретатор командного рядка BASH

BASH - один з найбільш популярних командних інтерпретаторів для ОС LINUX і їй подібних [2].

Синтаксис команд: **команда <ключі> <параметри>**

Команди інтерпретатора можуть вимагати зазначення обов'язкових параметрів і мати ключі для уточнення або розширення виведеної інформації. Використання певних ключів також може вимагати додаткових параметрів (наприклад, імен файлів, IP-адреси і т.д.). При роботі з інтерпретатором командного рядка користувач бачить знак запрошення .

Зазвичай це знак «\$» або «>». Запрошення командної оболонки можна змінити, воно міститься в змінній PS1 (рисунок 2.7).



```
maryna@maryna-VirtualBox:~$ echo $PS1
\[ \e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\[ \033[01;32m\]\u@\h\[ \033[00m\]:\[ \033[01;34m\]\w\[ \033[00m\]\$
```

Рисунок 2.7 – Значення змінної PS1

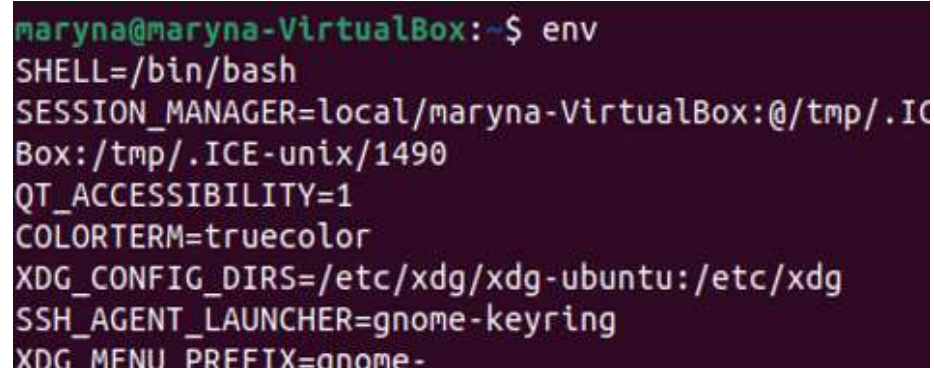
Робота в командному рядку:

- крапка «.» - важливий символ в командному рядку Linux, який означає поточний каталог;
- відносний шлях визначається відносно поточного каталогу, а абсолютний шлях – відносно кореневого каталогу, який позначається слешом «/»;
- тільда «~» в шляху до файлів означає домашній каталог користувача.
- Enter «порожня команда» - використовується для відділення тексту;
- скролінг ↑ або ↓ вгору/вниз команд, які нещодавно виконувались
- Tab – доповнення імені команди або імені файлу. При наявності декількох імен файлів або команд, що включають вже набраний текст, доповнює до максимального загального фрагмента. Доповнення команд або імен файлів здійснюється в залежності від попереднього параметра:
 - 1) якщо текст йде від знака запрошення - доповнюються імена команд;
 - 2) якщо перед текстом вже є команда, текст з командою розділені пробілом доповнюються імена файлів.
- Tab + Tab (double Tab) - виводить перелік усіх доступних команд або імен файлів, які починаються з тексту, який ви пишете;
- \$PATH - змінна оточення в UNIX-подібних ОС або Linux, яка містить список каталогів, де розташовані виконувані файли. Каталоги /bin, /usr/bin і /usr/local/bin зазвичай включені в більшість користувацьких налаштувань \$PATH, так як в них розташована найбільша кількість виконуваних файлів;
- оболонка bash має кілька операторів команд. Найбільш часто використовуються наступні оператори: < > | || & && ; ;; ().

Команди Bash

Команда **env** - використовується або для виведення змінних оточення або для запуску іншої утиліти з певними значеннями параметрів оточення без їх зміни в поточному середовищі (рисунок 2.8).

\$ env



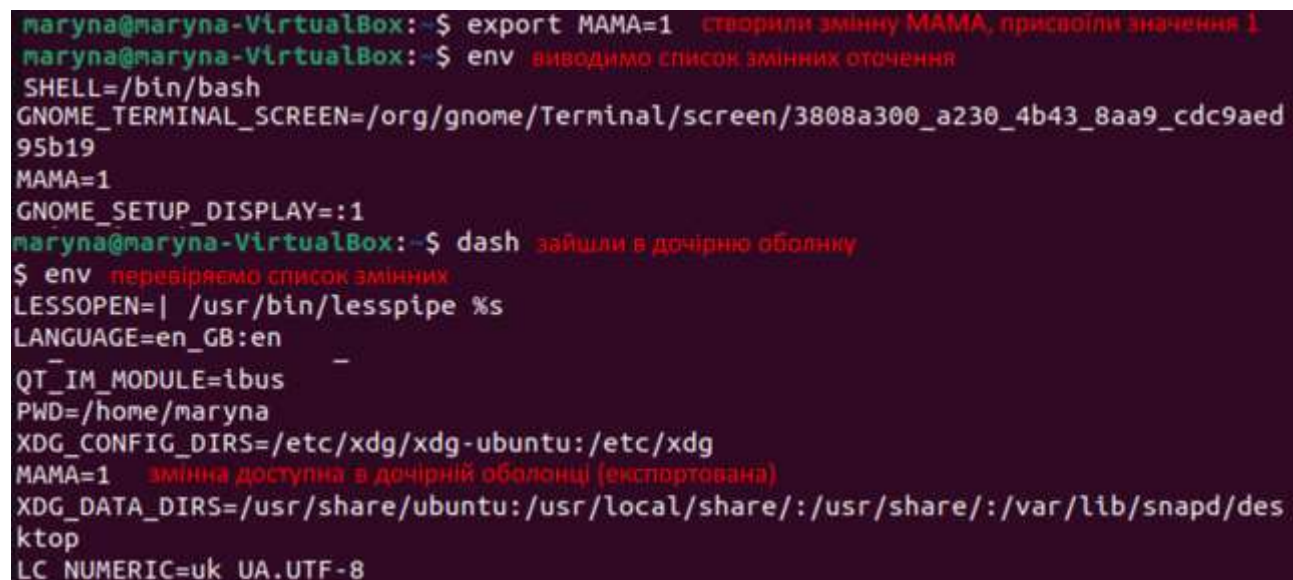
```
maryna@maryna-VirtualBox:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/maryna-VirtualBox:@/tmp/.ICE-unix/1490
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
```

Рисунок 2.8 – Команда env

Команда **export** - призначена для створення та експорту змінних і функцій поточного процесу в дочірній процес, у всі підоболонки (рисунок 2.9).

\$ export Variable = Value

Додає до переліку змінних оточення змінну з ім'ям Variable і значенням Value.



```
maryna@maryna-VirtualBox:~$ export MAMA=1 створили змінну MAMA, присвоїли значення 1
maryna@maryna-VirtualBox:~$ env виводимо список змінних оточення
SHELL=/bin/bash
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/3808a300_a230_4b43_8aa9_cdc9aed95b19
MAMA=1
GNOME_SETUP_DISPLAY=:1
maryna@maryna-VirtualBox:~$ dash зайшли в дочірню оболонку
$ env перевіряємо список змінних
LESSOPEN=| /usr/bin/lesspipe %s
LANGUAGE=en_GB:en
QT_IM_MODULE=ibus
PWD=/home/maryna
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
MAMA=1 змінна доступна в дочірній оболонці (експортована)
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
LC_NUMERIC=uk_UA.UTF-8
```

Рисунок 2.9 – Команда export

Приклад використання команди **export**, для створення змінної в поточній оболонці bash, запуску програми (скрипта), яка виводить значення цієї змінної в підоболонці sh (рисунок 2.10).

```
maryna@maryna-VirtualBox:~/SSS/SGID1$ export GIDNAME=SGID
maryna@maryna-VirtualBox:~/SSS/SGID1$ cat > SGID1.sh
#!/bin/sh

echo $GIDNAME
maryna@maryna-VirtualBox:~/SSS/SGID1$ sh
$ ls
mar1.txt  mar2.txt  SGID1.sh  SGID.sh  test1.txt  test2.txt
$ exit
maryna@maryna-VirtualBox:~/SSS/SGID1$ sudo chmod a=rwx SGID1.sh
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l SGID1.sh
-rwxrwxrwx 1 maryna root 25 жов  3 15:44 SGID1.sh
maryna@maryna-VirtualBox:~/SSS/SGID1$ sh
$ ls -l
total 8
-rw-rw-r-- 1 maryna maryna  0 жов  3 15:07 mar1.txt
-rw-rw-r-- 1 maryna root    0 жов  3 15:09 mar2.txt
-rwxrwxrwx 1 maryna root   25 жов  3 15:44 SGID1.sh
-rw-rw-r-- 1 maryna maryna 21 жов  3 15:35 SGID.sh
-rw-rw-r-- 1 test1  root    0 жов  3 15:08 test1.txt
-rw-rw-r-- 1 test1  test1   0 жов  3 15:13 test2.txt
$ ./SGID1.sh
SGID
```

Рисунок 2.10 – Приклад використання команди export

Команда **set** - відображає та встановлює імена та значення змінних оболонки та поточного середовища, використовується для надання значень позиційним параметрам.

Позиційний параметр - це змінна у програмі оболонки і на її значення посилаються як $\${N}$, де N – цифра, яка означає позицію параметра. Позиційні параметри використовуються в скриптах.

Команда **set** без параметрів відображає (рисунок 2.11) та встановлює (рисунок 2.11) імена та значення змінних оболонки та поточного середовища

```
maryna@maryna-VirtualBox:~$ set
BASH=/usr/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullqu
:force_ignores:globasciiranges:histappend:int
rs:sourcepath
BASH_ALIASES=()
BASH_ARGC=( [0]="0" )
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSIONINFO=( [0]="2" [1]="11" )
```

Рисунок 2.11 – Команда set, відображення змінних

Для того щоб встановити значення позиційним параметрам, використовуємо команду set, аргументами якої є значення для позиційних параметрів (рисунок 2.12).

```
maryna@maryna-VirtualBox:~$ set red blue
maryna@maryna-VirtualBox:~$ echo $1
red
maryna@maryna-VirtualBox:~$ echo $2
blue
maryna@maryna-VirtualBox:~$ set --
maryna@maryna-VirtualBox:~$ echo $1

maryna@maryna-VirtualBox:~$
```

Рисунок 2.12 – Команда set, встановлення імен та значень змінних

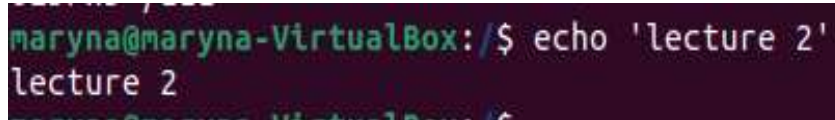
Команда **unset** - для видалення (вимкнення) змінної (рисунок 2.13). Перевіри існування змінної можна за допомогою команди envn або set і переконатись, що в списку змінних видалена змінна буде відсутня.

```
maryna@maryna-VirtualBox:~$ unset MYENV
maryna@maryna-VirtualBox:~$ env
SHELL=/bin/bash
```

Рисунок 2.13 – Команда unset

Команда **echo** - одна з найпростіших, але дуже корисних команд. Вона виводить зазначений рядок на екран (рисунок 2.14). Часто використовується в різних скриптах або для відправки значень в конфігураційні файли.

\$ echo «рядок»



```
maryna@maryna-VirtualBox:/$ echo 'lecture 2'
lecture 2
```

Рисунок 2.14 – Команда echo

Також дану команду можна використовувати для виведення значень змінних (рисунок 2.15). Коли ми звертаємось до змінної, перед нею потрібно вказати знак \$ без пробілу, інакше команда echo сприйме ім'я змінної як звичайний текст і виведе його.

\$ echo \$PATH - виведення значення змінної PATH - це змінна середовища, яка використовується для вказівки оболонці, де шукати виконувані файли.



```
maryna@maryna-VirtualBox:~$ echo $MYENV
1
maryna@maryna-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
maryna@maryna-VirtualBox:~$ echo PATH
PATH
maryna@maryna-VirtualBox:~$
```

Рисунок 2.15 – Команда echo для виведення значень змінних

Команда **who** – дозволяє дізнатися, скільки користувачів використовують систему або ввійшли в систему (рисунок 2.16). Аналогічну інформацію виводить команда **w** (рисунок 2.19). Команда має набір опцій, за допомогою яких можна вивести додаткову інформацію про користувачів в системі (час останнього завантаження системи, час перебування користувача в системі, перелік користувачів).

\$ who «опції»

```
maryna@maryna-VirtualBox:~$ who
maryna    tty2                2024-10-28 12:45 (tty2)
```

Рисунок 2.16 – Команда who

Опція -b використовується для перегляду часу останнього завантаження системи (рисунок 2.17).

```
maryna@maryna-VirtualBox:~$ who -b
system boot 2024-10-28 12:44
```

Рисунок 2.17 – Команда who

Опція -u використовується для перегляду користувачів, які увійшли в систему (рисунок 2.18).

```
maryna@maryna-VirtualBox:~$ who -bu
system boot 2024-10-28 12:44
maryna    tty2                2024-10-28 12:45 01:03      2751 (tty2)
```

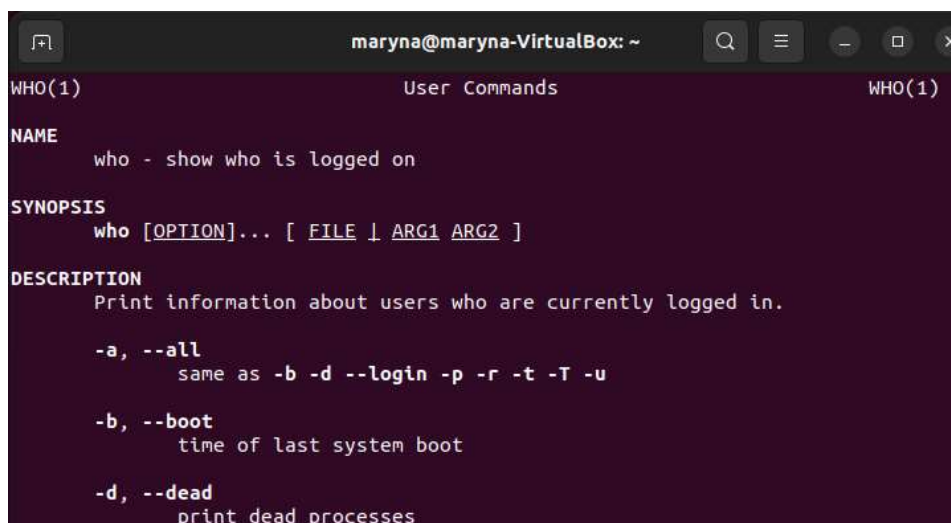
Рисунок 2.18 – Команда who

```
maryna@maryna-VirtualBox:~$ w
13:56:32 up 1:11, 1 user, load average: 0,00, 0,00, 0,00
USER      TTY      FROM          LOGIN@      IDLE        JCPU        PCPU WHAT
maryna    tty2     tty2          12:45       1:11m      0.01s       0.01s /usr/libexec/gn
```

Рисунок 2.19 – Команда w

Більше опцій команди who можна переглянути в довідці командою **man** (рисунок 2.20).

\$ who man



```
maryna@maryna-VirtualBox: ~
WHO(1) User Commands WHO(1)
NAME
  who - show who is logged on
SYNOPSIS
  who [OPTION]... [ FILE | ARG1 ARG2 ]
DESCRIPTION
  Print information about users who are currently logged in.

  -a, --all
      same as -b -d --login -p -r -t -T -u
  -b, --boot
      time of last system boot
  -d, --dead
      print dead processes
```

Рисунок 2.20 – Довідка команди who

Оболонка має вбудовані команди, які мають різне призначення: source, alias, bind, cd, declare, dirs, echo, enable, exit, bg, fg, history, help, kill, let, type, test, popd, printf, pushd, pwd, read, suspend, umask, time, wait та ін.

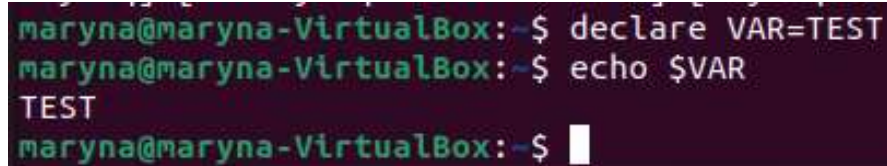
Команда **source** - дозволяє прочитати і виконати всі команди із зазначеного файлу в поточній командній оболонці. Це означає, що всі задані в ньому змінні оточення будуть видні в усіх скриптах або командах, які ви будете виконувати. Якщо ви запускаєте звичайний скрипт і оголошуєте в ньому змінну, то вона буде доступна тільки йому, бо для скриптів створюється окремий екземпляр оболонки. Він успадковує змінні батьківської оболонки, але не навпаки. Ця команда часто застосовується для оновлення оточення після змін в налаштуваннях системи, тобто зміни вступають в силу без перезавантаження системи.

\$ source шлях до файлу налаштувань (\$ source /etc/profile)

Команда **declare** – вбудована команда bash, дозволяє оголошувати змінні (рисунок 2.21).

\$ declare «опції»

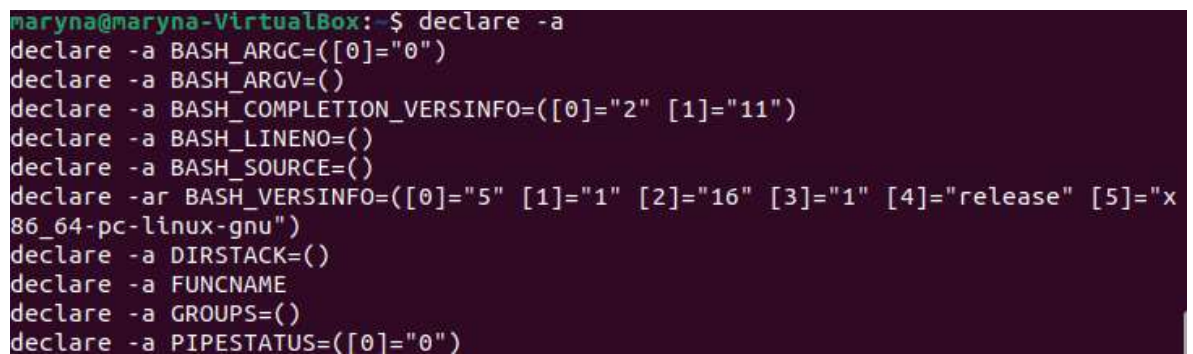
\$ declare VAR=value



```
maryna@maryna-VirtualBox:~$ declare VAR=TEST
maryna@maryna-VirtualBox:~$ echo $VAR
TEST
maryna@maryna-VirtualBox:~$
```

Рисунок 2.21 – Команда declare

Опція -a дає змогу вивести всі вже існуючі змінні оточення (рисунок 2.22). На відміну від звичайного оголошення можна задати різні параметри, наприклад, зробити змінну тільки для читання за допомогою опції -r. Також можна оголошувати функції.



```
maryna@maryna-VirtualBox:~$ declare -a
declare -a BASH_ARGC=([0]="0")
declare -a BASH_ARGV=()
declare -a BASH_COMPLETION_VERSION=([0]="2" [1]="11")
declare -a BASH_LINENO=()
declare -a BASH_SOURCE=()
declare -ar BASH_VERSION=([0]="5" [1]="1" [2]="16" [3]="1" [4]="release" [5]="x
86_64-pc-linux-gnu")
declare -a DIRSTACK=()
declare -a FUNCNAME
declare -a GROUPS=()
declare -a PIPESTATUS=([0]="0")
```

Рисунок 2.22 – Команда declare (вивід змінних оточення)

Команда **alias** дозволяє створювати псевдоніми для інших команд. Якщо викликати команду без параметрів, то вона виведе список вже заданих псевдонімів (рисунок 2.23).

\$ alias

\$ alias псевдонім='команда' (створення псевдоніму для команди)

```

maryna@maryna-VirtualBox:~$ alias
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]} && echo terminal || echo error
)" "${history|tail -n1|sed -e '\''s/^s*[0-9]\+\s*//;s/[\;&|]\s*alert$//'\''}"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
maryna@maryna-VirtualBox:~$ alias myls='ls'
maryna@maryna-VirtualBox:~$ myls
Desktop Documents Downloads Music Pictures Public snap Templates Videos
maryna@maryna-VirtualBox:~$

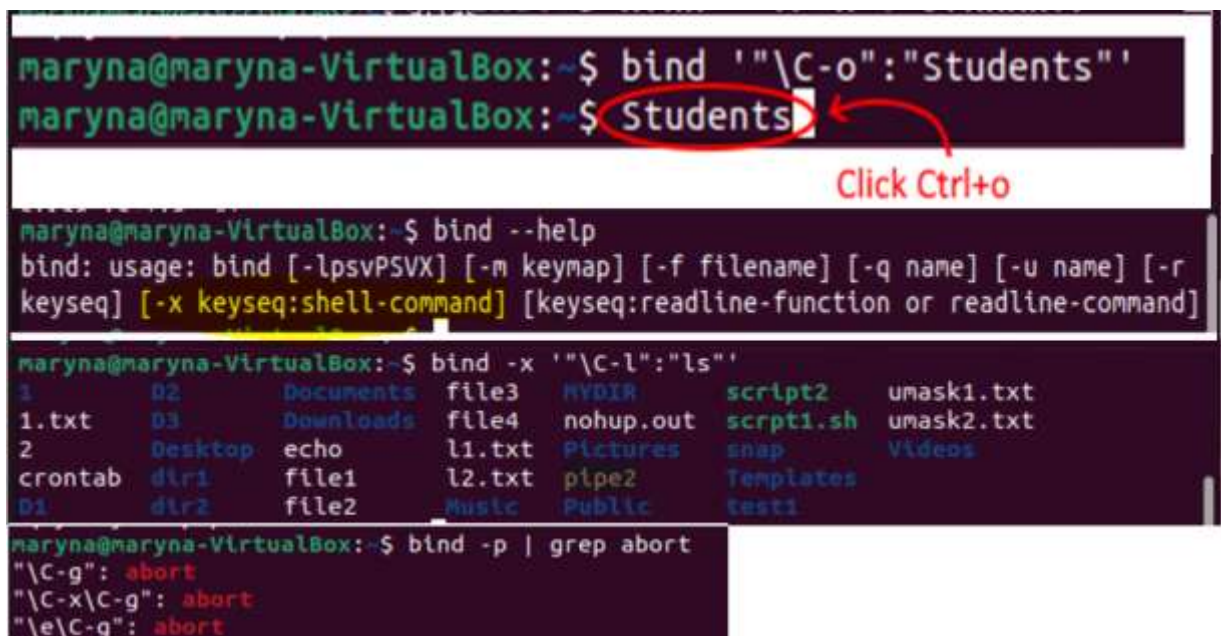
```

Рисунок 2.23 – Команда alias

Команда **bind**. За допомогою цієї команди ви можете пов'язати натискання клавіші з певною командою або функцією.

\$ bind «опції» """комбінація клавіш":»"команда"""

В прикладі по натисканню комбінації клавіш Ctrl+o буде виведено слово «Students», по натисканню комбінації клавіш Ctrl+l буде виконуватись команда ls – виведення вмісту поточного каталогу (рисунок 2.24).



```

maryna@maryna-VirtualBox:~$ bind "\C-o":"Students"
maryna@maryna-VirtualBox:~$ Students
maryna@maryna-VirtualBox:~$ bind --help
bind: usage: bind [-lpsvPSVX] [-m keymap] [-f filename] [-q name] [-u name] [-r
keyseq] [-x keyseq:shell-command] [keyseq:readline-function or readline-command]
maryna@maryna-VirtualBox:~$ bind -x "\C-l":"ls"
1      02      Documents  file3     MYDIR     script2   umask1.txt
1.txt   03      Downloads  file4     nohup.out script1.sh umask2.txt
2       Desktop echo       l1.txt    Pictures  snap      Videos
crontab dir1      file1     l2.txt    pipe2     Templates
01      dir2      file2     Music     Public    test1
maryna@maryna-VirtualBox:~$ bind -p | grep abort
"\C-g": abort
"\C-x\C-g": abort
"\e\C-g": abort

```

Рисунок 2.24 – Команда bind

Команда **cd** використовується при роботі з каталогами в терміналі. Вона дозволяє змінити поточну директорію на іншу (рисунк 2.25). Вона може використовуватись в скриптах, також вона необхідна при компіляції.

За замовчуванням, поточний каталог для користувача – домашній каталог `home/ім'я користувача (home/maryna)/`.

\$ cd /шлях/

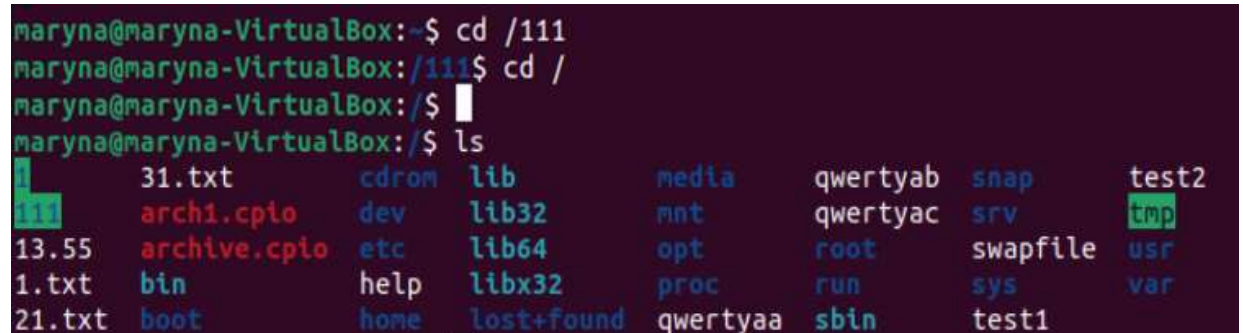
\$ cd перейти в домашній каталог

\$ cd / перейти в кореневий каталог

\$ cd ~user (перейти в домашній каталог користувача user)

\$ cd ../.. перейти в каталог на 2 рівні вище

\$ cd – перейти в каталог, в якому ви перебували до переходу до поточного каталогу

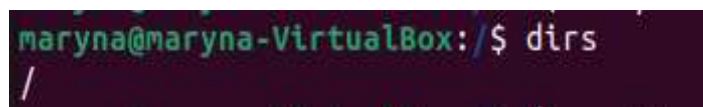


```
maryna@maryna-VirtualBox:~$ cd /111
maryna@maryna-VirtualBox:/111$ cd /
maryna@maryna-VirtualBox:/$ 
maryna@maryna-VirtualBox:/$ ls
111  31.txt      cdrom  lib       media     qwertyab  snap     test2
    arch1.cpio dev     lib32    nnt       qwertyac  srv       tmp
13.55 archive.cpio etc     lib64    opt       root      swapfile  usr
1.txt bin         help    libx32    proc      run       sys       var
21.txt boot        home    lost+found  qwertyaa  sbin      test1
```

Рисунок 2.25 – Команда cd

Команда **dirs** відображає історію зміни поточного каталогу (рисунк 2.26). Кожний попередній каталог відображається з нового рядка. За допомогою опції `-n` можна вказати, скільки останніх елементів виводити.

\$ dirs «опції»



```
maryna@maryna-VirtualBox:/$ dirs
/
```

Рисунок 2.26 – Команда dirs

Команда **enable** дозволяє увімкнути або вимкнути вбудовані команди **bash**. Для увімкнення вбудованої команди потрібно передати її в параметрах. Для вимкнення команди, потрібно використовувати опцію **-n**, після цього замість вбудованої команди, буде виконуватися утиліта з файлової системи.

\$ enable (виводить список увімкнених вбудованих команд)

\$ enable -n команда (вимкнути команду)

\$ enable команда (увімкнути команду)

В прикладі використовуємо команду **alias** і бачимо що вона працює, далі вимикаємо її **enable -n alias** і команда не доступна, після увімкнення **enable alias** – команда знову доступна і її можна використовувати (рисунк 2.27).



```
maryna@maryna-VirtualBox:/$ alias
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]} && echo terminal || echo error)"
"$$(history|tail -n1|sed -e '\''s/^\[0-9\]\+\s*//;s/[:;&]\s*alert$//'\''"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
alias myls='ls'
maryna@maryna-VirtualBox:/$ enable -n alias
maryna@maryna-VirtualBox:/$ alias
Command 'alias' not found, did you mean:
  command '0alias' from deb 0install-core (2.16-2)
Try: sudo apt install <deb name>
maryna@maryna-VirtualBox:/$ enable alias
maryna@maryna-VirtualBox:/$ alias
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]} && echo terminal || echo error)"
"$$(history|tail -n1|sed -e '\''s/^\[0-9\]\+\s*//;s/[:;&]\s*alert$//'\''"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
alias myls='ls'
maryna@maryna-VirtualBox:/$
```

Рисунок 2.27 – Команда **enable**

В наступному прикладі використовуємо команду **echo**, переглядаємо список вбудованих команд **enable** і бачимо що вона є, далі вимикаємо її **enable -n echo**, переглядаємо список і команда не доступна, використовуємо команду **echo**, бачимо результат виконання, але в цьому випадку працює не вбудована команда

echo, а одноіменна утиліта, яка працює так само, як і вбудована команда echo (рисунок 2.28).

```
maryna@maryna-VirtualBox:~$ enable
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
enable builtin
enable caller
enable cd
enable command
enable compgen
enable complete
enable compopt
enable continue
enable declare
enable dirs
enable disown
enable echo
enable enable
enable eval
enable exec
maryna@maryna-VirtualBox:~$ enable -n echo
maryna@maryna-VirtualBox:~$ eanble
eanble: command not found
maryna@maryna-VirtualBox:~$ enable
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
enable builtin
enable caller
enable cd
enable command
enable compgen
enable complete
enable compopt
enable continue
enable declare
enable dirs
enable disown
enable enable
enable eval
enable exec
maryna@maryna-VirtualBox:~$ echo working
working
```

Рисунок 2.28 – Команда enable

Командою **type** можна перевірити тип команди, яка використовується (рисунок 2.29).

```
maryna@maryna-VirtualBox:~/$ type echo
echo is a shell builtin
maryna@maryna-VirtualBox:~/$ enable -n echo
maryna@maryna-VirtualBox:~/$ type echo
echo is /usr/bin/echo
maryna@maryna-VirtualBox:~/$
```

Рисунок 2.29 – Команда type

Команда **exit** – універсальна команда, яка дозволяє закрити сеанс оболонки, вийти з терміналу або завершити поточну сесію (рисунок 2.30).

```
maryna@maryna-VirtualBox:/111$ dash
$ echo new shell dash
new shell dash
$ exit
```

Рисунок 2.30 – Команда exit

Команда **history** дозволяє подивитися історію команд, які ви виконували раніше. Тут зберігається понад 1000 останніх команд, ви можете все їх побачити, виконувати по них пошук, виконувати певні команди з історії за порядковим номером, видаляти команди з історії команд . Можна задати необхідну кількість записів шляхом передачі команді потрібного значення (рисунок 2.31).

\$ history «опції»

\$ history 5

```
maryna@maryna-VirtualBox:/$ history 8
197  enable alias
198  alias
199  enable -n echo
200  enable
201  enalbe echo
202  enable echo
203  enable
204  history 8
```

Рисунок 2.31 – Команда history

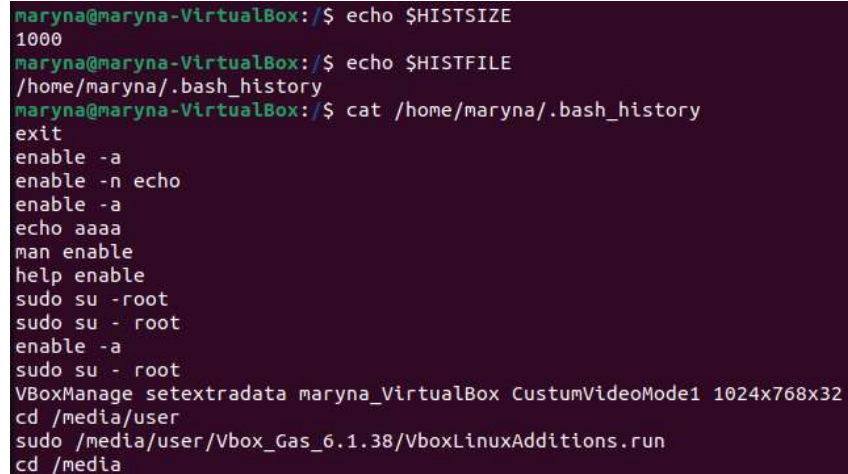
Команда history зберігає історію для кожного користувача в прихованому файлі `.bash_history` (рисунок 2.32).

Змінна оточення `HISTSIZE` задає оболонці bash кількість рядків для збереження в історії.

Змінна оточення HISTFILE задає оболонці bash ім'я файлу для збереження історії введення команд.

Детальну інформацію про команду history, можливості та опції можна отримати в довідці MAN:

\$ man history

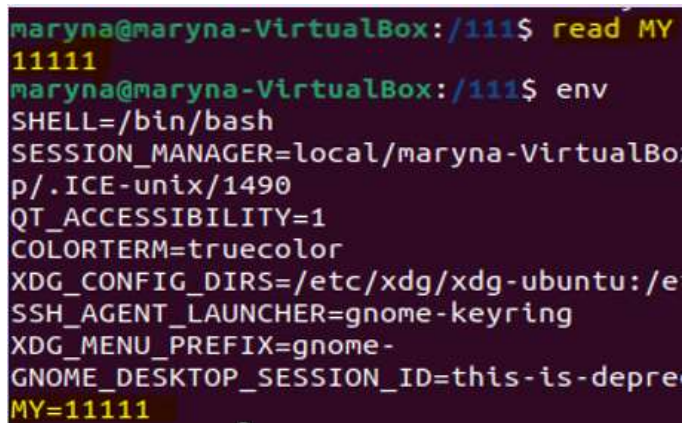


```
maryna@maryna-VirtualBox:/$ echo $HISTSIZE
1000
maryna@maryna-VirtualBox:/$ echo $HISTFILE
/home/maryna/.bash_history
maryna@maryna-VirtualBox:/$ cat /home/maryna/.bash_history
exit
enable -a
enable -n echo
enable -a
echo aaaa
man enable
help enable
sudo su -root
sudo su - root
enable -a
sudo su - root
VBoxManage setextradata maryna_VirtualBox CustumVideoMode1 1024x768x32
cd /media/user
sudo /media/user/Vbox_Gas_6.1.38/VboxLinuxAdditions.run
cd /media
```

Рисунок 2.32 – Файл .bash_history

Команда **read**. За допомогою цієї команди ви можете прочитати рядок зі стандартного вводу або файлу в змінну (рисунок 2.33). Команді можна передати файл для зчитування, а також кілька змінних. Команда використовується в скриптах.

\$ read VAR



```
maryna@maryna-VirtualBox:/111$ read MY
11111
maryna@maryna-VirtualBox:/111$ env
SHELL=/bin/bash
SESSION_MANAGER=local/maryna-VirtualBo
p/.ICE-unix/1490
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/e
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-depre
MY=11111
```

Рисунок 2.33 – Команда read

Команда **pwd** дозволяє відобразити поточний робочий каталог:

```
$ pwd
```

Кожна програма при запуску отримує поточний каталог, в якому будуть виконуватися всі операції з файлами, якщо для них не вказаний інший шлях. Всі відносні шляхи теж будуть будуватись від цього каталогу.

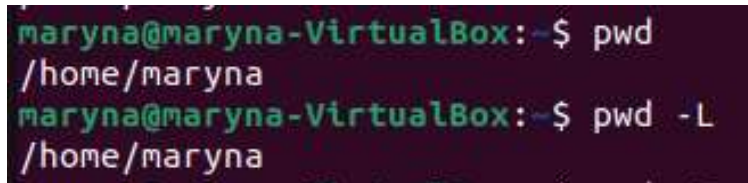
При запуску терміналу поточним каталогом є домашній каталог користувача (рисунок 2.34). Якщо програму запустити з певного каталогу, то її поточним каталогом буде каталог запуску.

```
$ pwd «опції»
```

Основні опції:

-L, --logical – брати директорію із змінної оточення PWD;

--help – довідка по команді.

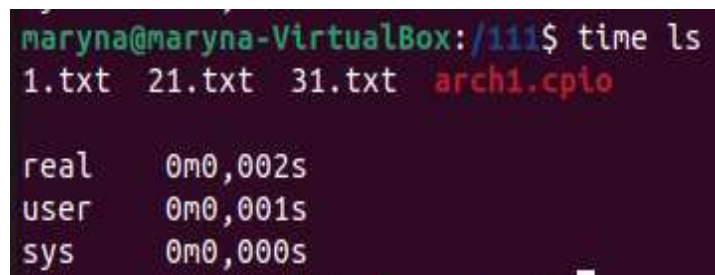


```
maryna@maryna-VirtualBox:~$ pwd
/home/maryna
maryna@maryna-VirtualBox:~$ pwd -L
/home/maryna
```

Рисунок 2.34 – Команда pwd

Команда **time** дозволяє заміряти час виконання будь-якого скрипта або програми. Вимірюється окремо час виконання програми і час введення користувачем (рисунок 2.35).

```
$ time ls
```



```
maryna@maryna-VirtualBox:~/111$ time ls
1.txt  21.txt  31.txt  arch1.cpio

real    0m0,002s
user    0m0,001s
sys     0m0,000s
```

Рисунок 2.35 – Команда time

Команда **date** працює з датою та часом (рисунок 2.36).

\$ date «опції»... «+формат»

Має декілька опцій

-d STRING --date=STRING – виведення дати за вказаним рядком (наприклад 'yesterday', 'tomorrow', '1 year ago').

Аргумент ФОРМАТ відповідає за форматування виведення дати. Щоб задати формат, потрібно поставити знак "+" і написати потрібний формат.

Найбільш популярні формати:

%D – вивід дати mm/dd/yy

%Y – рік (2024)

%m – місяць (01-12)

%B – довге ім'я місяця (November)

%b – коротке ім'я місяця (Nov)

%d – день місяця (e.g., 01)

%j – день року (001-366)

%u – день тижня (1-7)

%A – повне ім'я дня (Friday)

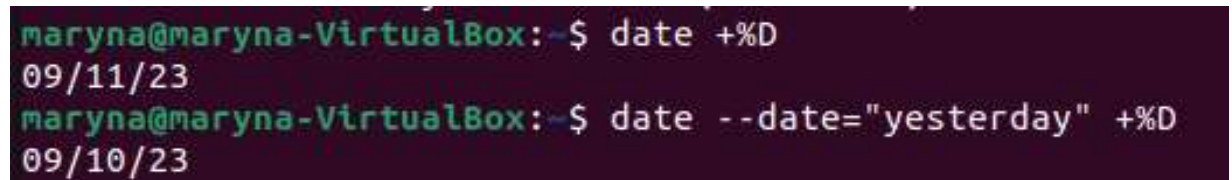
%a – коротке ім'я дня (Fri)

%h – години (00-23)

%i – години (01-12)

%m – хвилини (00-59)

%s – секунди (00-60)



```
maryna@maryna-VirtualBox:~$ date +%D
09/11/23
maryna@maryna-VirtualBox:~$ date --date="yesterday" +%D
09/10/23
```

Рисунок 2.36 – Команда date

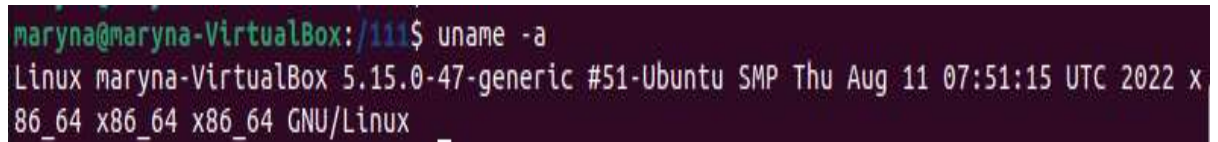
Команда **uname** друкує основну інформацію про ім'я операційної системи і системне обладнання (рисунок 2.37). Найчастіше використовується для визначення архітектури процесора, імені хоста системи, версії ядра.

\$ uname «опції»

Можливі опції:

- s – виводить ім'я ядра;
- n – виводить ім'я вузла системи (ім'я хоста)- це ім'я, яке система використовує в мережі;
- r – виводить реліз ядра;
- v – виводить версію ядра;
- m – виводить назву архітектури машини;
- p – виводить архітектуру процесора;
- i – виводить апаратну платформу;
- o – виводить назву операційної системи;
- a – при використанні опції -a uname поводитьься так само, як uname з опціями -snrvmo.

Без параметрів uname друкує ім'я ядра, результат, такий, як з опцією -s.



```
maryna@maryna-VirtualBox:/111$ uname -a
Linux maryna-VirtualBox 5.15.0-47-generic #51-Ubuntu SMP Thu Aug 11 07:51:15 UTC 2022 x
86_64 x86_64 x86_64 GNU/Linux
```

Рисунок 2.37 – Команда uname

Команду **exec** можна використовувати для запуску іншої програми, яка замінює поточну оболонку. В прикладі запуск текстового редактора nano (рисунок 2.38).

exec >filename перенаправляє вивід на стандартний потік stdout в заданий файл. Після цього, весь вивід від команд, який зазвичай направляється на stdout, буде виводиться в цей файл.

`exec <filename` перенаправляє введення зі стандартного потоку `stdin` в файл. Всі введення, замість `stdin` (наприклад, з клавіатури), будуть братись з цього файлу. Це дає можливість читати вміст файлу, рядок за рядком, і аналізувати кожен введений рядок за допомогою, наприклад утиліти для пошуку текстових потоків та заміни - `sed`.



Рисунок 2.38 – Команда `exec`

Команда **man** дозволяє отримати доступ до загальної бази довідки по команді, функції або програмі. Для отримання довідки, програмі потрібно передати назву команди (рисунок 2.39).

`$ man «розділ» «назва сторінки»`

Зазвичай назва сторінки збігається з ім'ям команди або назвою програми. Вся довідка розділена на кілька розділів. За замовчуванням, сторінки довідки виводяться з усіх розділів, але ви можете виводити сторінки довідки з певного розділу. Для цього потрібно передати номер розділу команді.

Основні розділи:

- 1 – команди оболонки або програми;
- 2 – системні виклики (функції, які реалізуються ядром);
- 3 – функції бібліотек (функції, які реалізуються бібліотеками програм);

- 4 – файли з каталогу (наприклад /etc);
- 5 – формати файлів;
- 6 – ігри;
- 7 – різне (включаючи макропакети);
- 8 – команди адміністрування системи;
- 9 – процедури ядра.

```
maryna@maryna-VirtualBox:/11$ man uname
NAME
  uname - print system information
SYNOPSIS
  uname [OPTION]...
DESCRIPTION
  Print certain system information.  With no OPTION, same as -s.
  -a, --all
    print all information, in the following order, except omit -p and
    if unknown:
  -s, --kernel-name
    print the kernel name
  -n, --nodename
    print the network node hostname
  -r, --kernel-release
    print the kernel release
  -v, --kernel-version
    print the kernel version
  -m, --machine
    print the machine hardware name
  -p, --processor
    print the processor type (non-portable)
Manual page uname(1) line 1/64 46% (press h for help or q to quit)
```

Рисунок 2.39 – Команда man

Команда/опція **help** використовується для виведення інформації про команду в термінал користувача (рисунок 2.40).

«ім'я команди» --help

```
maryna@maryna-VirtualBox:/$ cd --help
cd: cd [-L|[-P [-e]] [-@]] [dir]
  Change the shell working directory.

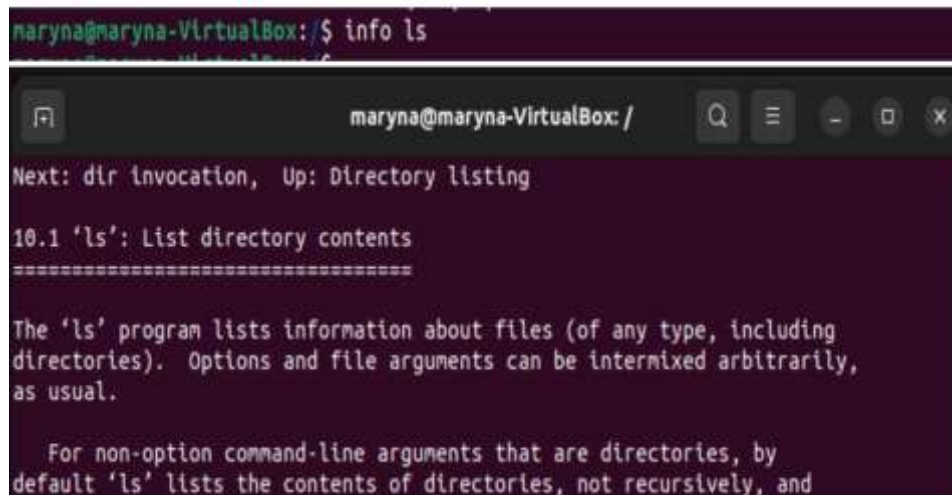
  Change the current directory to DIR.  The default DIR is the value of the
  HOME shell variable.

  The variable CDPATH defines the search path for the directory containing
  DIR.  Alternative directory names in CDPATH are separated by a colon (:).
  A null directory name is the same as the current directory.  If DIR begins
  with a slash (/), then CDPATH is not used.
```

Рисунок 2.40 – Команда/опція help

Опція `help` є не у всіх команд, тому рекомендується використовувати команду `man` або `info` (рисунок 2.41).

info «ім'я команди»



```
maryna@maryna-VirtualBox:/$ info ls
Next: dir invocation, Up: Directory listing

10.1 'ls': List directory contents
=====

The 'ls' program lists information about files (of any type, including
directories). Options and file arguments can be intermixed arbitrarily,
as usual.

For non-option command-line arguments that are directories, by
default 'ls' lists the contents of directories, not recursively, and
```

Рисунок 2.41 – Команда `info`

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №3 Структура файлової системи.

Управління каталогами/файлами Ubuntu

Мета: ознайомитись з структурою файлової системи, за допомогою команд в командному інтерпретаторі навчитись створювати, редагувати, копіювати, переміщувати, архівувати, видаляти, переглядати файли та каталоги, здійснювати пошук файлів та каталогів, використовувати символи для підстановки імен (file-globbing) при роботі з командами [7].

Завдання

Зайти в систему під користувачем, якого створено в першій роботі. Навчитися користуватися команди (з ключами): `cd`, `ls`, `mkdir`, `rm`, `mv`, `cp`, `touch`, `file`, `find`, застосувати символи для підстановки (file-globbing) при роботі з командами. Ознайомитися з утилітами для стиснення даних: `tar`, `gzip/bzip2` та утилітою копіювання та конвертації файлів `dd`, навчитися користуватись даними утилітами.

Теоретичні відомості

Файлова система Linux має структуру, в основі якої лежить кореневий каталог, який позначається символом `"/`, а за ним знаходяться інші каталоги. В Linux є набір стандартних каталогів які уніфіковані відповідно до стандарту Filesystem Hierarchy Standard (рисунок 3.1). Стандарт ієрархії файлової системи FHS в Linux описує призначення каталогів в дереві файлової системи.

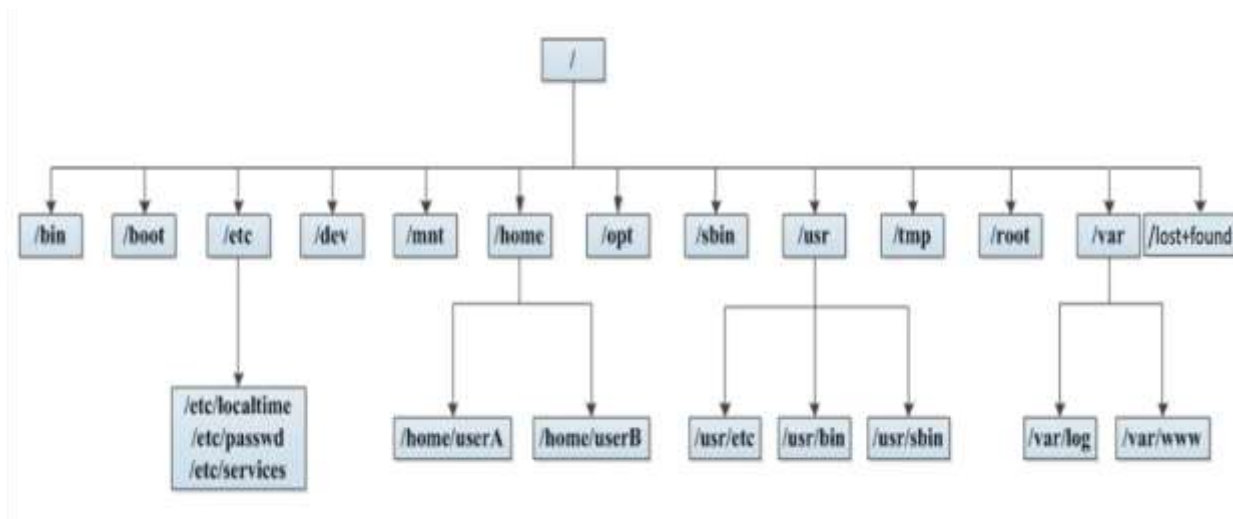


Рисунок 3.1 – Команда info

Структура [2]:

/bin - основні виконувані модулі (стандартні утиліти);

/boot - містить конфігураційні файли завантажувача GRUB (GRand Unified Bootloader), образи ядра та інші допоміжні файли завантажувача;

/dev - містить файли пристроїв (в Linux все підключені пристрої представлені у вигляді файлів і звернення до них відбувається саме через файли в цьому каталозі);

/etc - містить конфігураційні файли операційної системи і всіх мережевих служб, тут знаходяться саме загальносистемні конфігураційні файли, а призначені для користувача конфігураційні файли знаходяться в домашньому каталозі кожного користувача;

/home - тут знаходяться домашні каталоги всіх користувачів, які зареєстровані в системі, а саме зберігаються призначені для користувача дані і конфігураційні файли конкретного користувача. Звичайний користувач має права запису тільки в своєму домашньому каталозі, а щоб змінювати інші файли в системі потрібно мати права доступу root;

/lib - тут знаходяться різні бібліотеки і модулі ядра, саме сюди встановлюються різні залежності необхідні для коректної роботи додатків;

/lost + found - в цей каталог система буде поміщати пошкоджені файли, надалі їх можна спробувати відновити;

/media - підключаються носії, коли вставиш в комп'ютер флешку в цьому каталозі створиться підкаталог з ім'ям тома твого пристрою та використовуючи його можна отримати доступ до вмісту пристрою;

/mnt - в цей каталог можна монтувати зовнішні або додаткові файлові системи

/opt - каталог містить допоміжні пакети, туди можуть встановлюватися додаткові пакети програмного забезпечення;

/proc - містить файли з інформацією про систему і процесах;

/root - домашній каталог користувача root;

/run - в цьому каталозі застосунки можуть зберігати допоміжні файли, в тому числі ідентифікатори процесів;

/sbin - містить виконувані файли, необхідні для адміністрування системи;

/srv - зберігаються дані сервісних служб;

/sys - інформація про систему;

/tmp - тимчасові файли;

/usr - знаходяться програми та файли, які використовуються користувачами;

/var - містить найбільш часто змінюються файли, наприклад, кеші програм, файли системних журналів, тимчасові файли і т.д.

Одна з команд яка застосовується дуже часто це **cd** - це команда зміни каталогу, вона дозволяє переміщатися між каталогами:

cd /home/user перейти в зазначений каталог

cd / перейти в кореневий каталог

cd перейти в домашній каталог поточного користувача

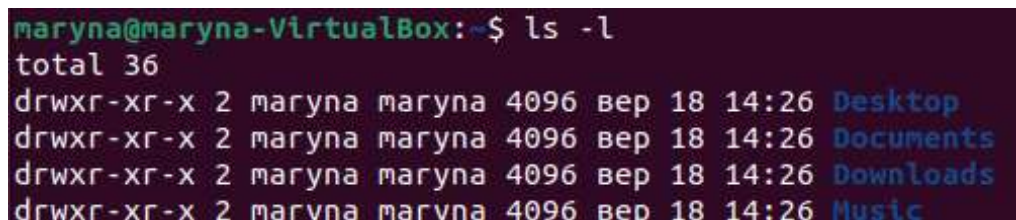
cd ../.. перейти в каталог на два рівні вище

cd - перейти в каталог в якому знаходився до переходу в поточний.

Команда: **ls** - перегляд вмісту каталогу:

ls покаже файли і каталоги в цій папці

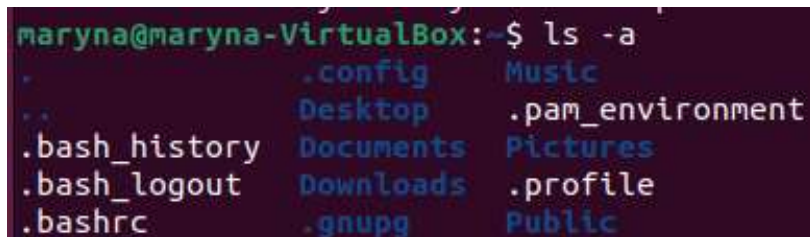
ls -l виведе вміст з детальною інформацією (дата створення, розмір, права доступу і т.д.) (рисунок 3.2).



```
maryna@maryna-VirtualBox:~$ ls -l
total 36
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Desktop
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Documents
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Downloads
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Music
```

Рисунок 3.2 – Команда **ls** - вміст каталогу з детальною інформацією

ls -a покаже вміст включаючи приховані файли (починаються з символу ".") (рисунок 3.3).



```
maryna@maryna-VirtualBox:~$ ls -a
.      .config  Music
..     Desktop .pam_environment
.bash_history Documents Pictures
.bash_logout Downloads .profile
.bashrc  .gnupg   Public
```

Рисунок 3.3 – Команда **ls** - вміст каталогу, включаючи приховані файли

Ключі команди **ls**:

-a не ігнорувати файли, що починаються з крапки

-l вивести індексний дескриптор файлу

-l довгий формат виведення

-r --reverse звернути порядок при сортуванні

-R --recursive рекурсивно виводити підкаталоги

- t впорядкувати за часом зміни
- d вивести тільки каталоги
- s впорядкувати за розміром
- h виводити розмір в форматі К, М, G - байт.

Команда **mkdir** – створення нового каталогу. Команді потрібно передати назву нового каталогу (потрібно пам'ятати, що можна створювати нові каталоги тільки всередині каталогу, в якому у користувача є право на запис).

Приклади створення каталогів (рисунок 3.4):

mkdir newfolder створить каталог з ім'ям newfolder

mkdir new new1 створить два каталоги з іменами new і new1

mkdir -p new / new1 / new2 створить вказане дерево директорій

```
maryna@maryna-VirtualBox:~$ mkdir myD1
maryna@maryna-VirtualBox:~$ mkdir myD2 myD3
maryna@maryna-VirtualBox:~$ ls
1.txt  3.txt  Documents  Music  myD2  Pictures  snap  Videos
2.txt  Desktop Downloads  myD1  myD3  Public  Templates
maryna@maryna-VirtualBox:~$ mkdir -p myD4/myD5
maryna@maryna-VirtualBox:~$ ls
1.txt  3.txt  Documents  Music  myD2  myD4  Public  Templates
2.txt  Desktop Downloads  myD1  myD3  Pictures  snap  Videos
maryna@maryna-VirtualBox:~$ cd myD4
maryna@maryna-VirtualBox:~/myD4$ ls
myD5
maryna@maryna-VirtualBox:~/myD4$ cd
maryna@maryna-VirtualBox:~$
```

Рисунок 3.4 – Команда mkdir

Команда **touch** - без застосування опцій створює порожній файл розміром 0 байт з вказаним ім'ям. Можна створити кілька файлів – потрібно вказати назви всіх файлів (рисунок 3.5).

touch опції... [файл]...

```

maryna@maryna-VirtualBox:~$ touch 1.txt
maryna@maryna-VirtualBox:~$ ls
1.txt  Documents Music  Public Templates
Desktop Downloads Pictures snap Videos
maryna@maryna-VirtualBox:~$ touch 2.txt 3.txt
maryna@maryna-VirtualBox:~$ ls
1.txt 3.txt Documents Music Public Templates
2.txt Desktop Downloads Pictures snap Videos

```

Рисунок 3.5 – Команда touch

Також команда **touch** використовується для зміни тимчасової мітки файлу. За допомогою цієї команди можна змінювати час доступу до файлу і час його зміни (рисунок 3.6). Якщо вказаний файл не існує, touch створить файл нульового розміру з вказаним ім'ям.

Ключі команди touch:

- a змінити час доступу
- m змінити час модифікації
- t замість поточного часу використовувати timestamp в форматі YYMMDDhhmm (Зміна mtime)
- d замість поточного часу використовувати різні формати дати і часу аналогічні для такого параметра команди date
- r установка часу за прикладом
- зміна позначки часу для файлу, пов'язаного зі стандартним виведенням даних.

```

maryna@maryna-VirtualBox:~$ ls -l 1.txt
-rw-rw-r-- 1 maryna maryna 4 жов 2 14:27 1.txt
maryna@maryna-VirtualBox:~$ touch - 1.txt
maryna@maryna-VirtualBox:~$ ls -l 1.txt
-rw-rw-r-- 1 maryna maryna 4 жов 2 15:13 1.txt
maryna@maryna-VirtualBox:~$ touch -r 1.txt 2.txt
maryna@maryna-VirtualBox:~$ ls -l 2.txt
-rw-rw-r-- 1 maryna maryna 4 жов 2 15:13 2.txt
maryna@maryna-VirtualBox:~$

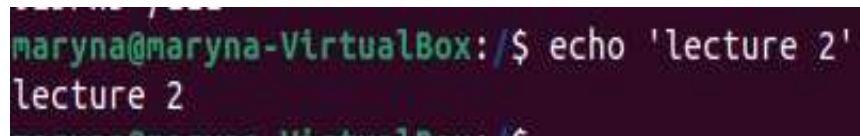
```

Рисунок 3.6 – Команда touch - установка часу за прикладом

Команда **echo** - виводить зазначений рядок на екран. Часто використовується в різних скриптах або для відправки значень в конфігураційні файли (рисунок 3.7).

echo "Це рядок"

echo -e "рядок1 \nрядок1 \nрядок1" – виведе кілька рядків (\n для перенесення тексту на новий рядок)



```
maryna@maryna-VirtualBox:/$ echo 'lecture 2'
lecture 2
```

Рисунок 3.7 – Команда echo

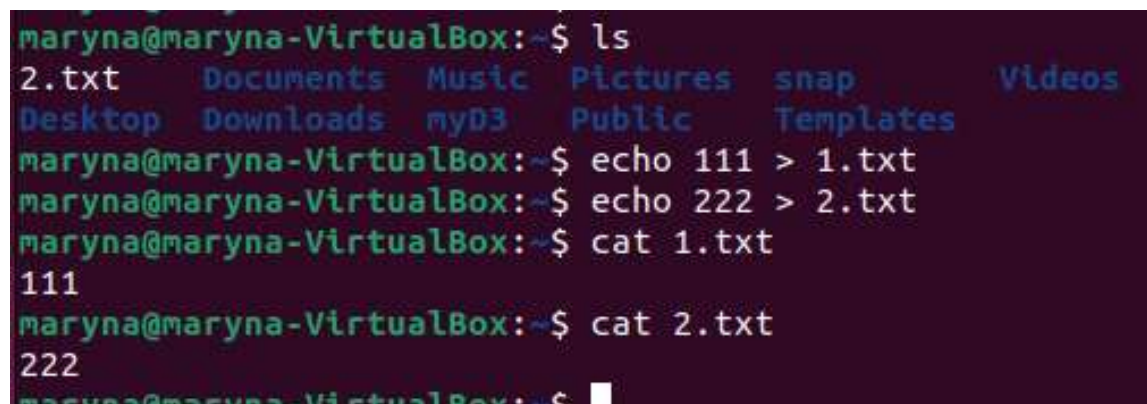
Для запису тесту у файл використовуються символи перенаправлення:

> - перенаправлення у файл виводу заданої команди (перезапис)

>> - перенаправлення у файл виводу заданої команди (додавання в кінець файлу).

Створення файлу з вмістом на допомогою команди echo зображено на рисунку 3.8.

Команда **cat** - виводить вміст файлу.



```
maryna@maryna-VirtualBox:~$ ls
2.txt  Documents  Music  Pictures  snap  Videos
Desktop Downloads myD3    Public   Templates
maryna@maryna-VirtualBox:~$ echo 111 > 1.txt
maryna@maryna-VirtualBox:~$ echo 222 > 2.txt
maryna@maryna-VirtualBox:~$ cat 1.txt
111
maryna@maryna-VirtualBox:~$ cat 2.txt
222
maryna@maryna-VirtualBox:~$
```

Рисунок 3.8 – Створення файлу з вмістом

Команда **file** відображає тип файлу. Це корисно, коли потрібно дізнатися тип файлу, який ви ніколи раніше не бачили або файл без розширення (рисунок 3.9).

```
maryna@maryna-VirtualBox:/$ file test1
test1: empty
maryna@maryna-VirtualBox:~$ file 1.txt
1.txt: ASCII text
```

Рисунок 3.9 – Команда file

Команда **rmdir** - видалення каталогів (рисунок 3.10).

rmdir опції... [імя_діректорії]

```
maryna@maryna-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
maryna@maryna-VirtualBox:~$ mkdir newdir1
maryna@maryna-VirtualBox:~$ ls
Desktop  Downloads  newdir1  Public  Templates
Documents Music    Pictures snap    Videos
maryna@maryna-VirtualBox:~$ rmdir newdir1
maryna@maryna-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
```

Рисунок 3.10 – Команда rmdir

Команда **rm** - видалення файлів і каталогів (рисунок 3.11).

Ключі команди rm:

- f --force ігнорувати неіснуючі файли, виключити підтвердження
- i --interactive завжди питати перед видаленням
- r -R --recursive видаляти каталоги і їх вміст рекурсивно
- v --verbose виводити звіт про виконані дії.

rm file1 видалити file1

rm -r folder1 видалити каталог folder1 і його вміст

```

maryna@maryna-VirtualBox:~$ rm 3.txt
maryna@maryna-VirtualBox:~$ ls
1.txt Desktop Downloads myD1 myD3 Pictures snap Videos
2.txt Documents Music myD2 myD4 Public Templates
maryna@maryna-VirtualBox:~$ rm myD4
rm: cannot remove 'myD4': Is a directory
maryna@maryna-VirtualBox:~$ rm -r myD4
maryna@maryna-VirtualBox:~$ ls
1.txt Desktop Downloads myD1 myD3 Public Templates
2.txt Documents Music myD2 Pictures snap Videos

```

Рисунок 3.11 – Команда rm

Команда **mv** - перейменування / переміщення файлі та каталогів (рисунок 3.12).

mv file1 newname перейменувати файл (каталог) file1 в newname

mv file1 folder1 перемістити файл file1 в каталог folder2

mv folder1 folder2 перемістити каталог folder1 в каталог folder2

Ключі команди mv:

-f --force не перепитувати перед перезаписом

-i --interactive перепитувати перед перезаписом

```

maryna@maryna-VirtualBox:~$ ls
1.txt Desktop Downloads myD1 myD3 Public Templates
2.txt Documents Music myD2 Pictures snap Videos
maryna@maryna-VirtualBox:~$ mv 1.txt 111.txt
maryna@maryna-VirtualBox:~$ ls
111.txt Desktop Downloads myD1 myD3 Public Templates
2.txt Documents Music myD2 Pictures snap Videos
maryna@maryna-VirtualBox:~$ mv 111.txt myD1
maryna@maryna-VirtualBox:~$ ls
2.txt Documents Music myD2 Pictures snap Videos
Desktop Downloads myD1 myD3 Public Templates
maryna@maryna-VirtualBox:~$ cd myD1
maryna@maryna-VirtualBox:~/myD1$ ls
111.txt
maryna@maryna-VirtualBox:~$ ls
2.txt Documents Music myD3 Public Templates
Desktop Downloads myD1 Pictures snap Videos
maryna@maryna-VirtualBox:~$ cd myD3
maryna@maryna-VirtualBox:~/myD3$ ls
maryna@maryna-VirtualBox:~/myD3$ cd ..
maryna@maryna-VirtualBox:~$ mv myD1 myD3
maryna@maryna-VirtualBox:~$ ls
2.txt Documents Music Pictures snap Videos
Desktop Downloads myD3 Public Templates
maryna@maryna-VirtualBox:~$ cd myD3
maryna@maryna-VirtualBox:~/myD3$ ls
myD1

```

Рисунок 3.12 – Команда mv

Команда **cp** - копіювання файлів, вмісту файлів та каталогів (рисунок 3.13).
cp file1 file2 скопіювати файл file1 в файл file2 (якщо файли не в поточному каталозі необхідно вказувати повний шлях до файлів)

cp file1 /home/user1/ скопіювати файл file1 в каталог user1

cp -r folder1 folder2 скопіювати каталог folder1 в каталог folder2 (якщо каталогу folder2 НЕ існує він буде створений) (folder1/2 це повний шлях до каталогу).

Ключі команди cp:

-i --interactive завжди питати перед перезаписом

-p збереження режиму доступу, власника і тимчасових міток

-r -R --recursive копіювати каталоги і їх вміст рекурсивно

-v --verbose виводити звіт про виконані дії.

```
maryna@maryna-VirtualBox:~$ cp 1.txt 2.txt
maryna@maryna-VirtualBox:~$ cat 1.txt
111
maryna@maryna-VirtualBox:~$ cat 2.txt
111
maryna@maryna-VirtualBox:~$ cd myD3
maryna@maryna-VirtualBox:~/myD3$ ls
myD1
maryna@maryna-VirtualBox:~/myD3$ cd ..
maryna@maryna-VirtualBox:~$ cp 2.txt myD3
maryna@maryna-VirtualBox:~$ cd myD3
maryna@maryna-VirtualBox:~/myD3$ ls
2.txt  myD1
maryna@maryna-VirtualBox:~/myD3$ cp -r myD1 myD4
maryna@maryna-VirtualBox:~/myD3$ ls
2.txt  myD1  myD4
maryna@maryna-VirtualBox:~/myD3$ cd myD4
maryna@maryna-VirtualBox:~/myD3/myD4$ ls
111.txt
```

Рисунок 3.13 – Команда cp

Команда **find** представляє собою універсальний інструмент пошуку: вона дозволяє шукати файли і каталоги, переглядати всі каталоги в системі або тільки поточний каталог.

find шлях_імя параметри –опції

шлях_імя - це каталог, з якого необхідно починати пошук. Символ '.' служить для позначення поточного каталогу, символ / - кореневого каталогу, а символ «~» - записаного в змінній \$HOME початкового каталогу поточного користувача.

Параметри команди:

-P - ніколи не відкривати символічні посилання

-L - отримує інформацію про файли за символьними посиланнями.

Важливо для подальшої обробки, щоб оброблялося не посилання, а сам файл.

-maxdepth - максимальна глибина пошуку по підкаталогам, для пошуку тільки в поточному каталозі становить 1.

-depth - шукати спочатку в поточному каталозі, а потім в підкаталогах

-mount шукати файли тільки в цій файлової системи.

-version - показати версію утиліти find

-print - виводити повні імена файлів

-type - пошук з шаблоном

-type f - шукати тільки файли

-type d - пошук папки в Linux.

Аргументи ключа type:

b блоковий пристрій (пристрої зберігання)

c символічне пристрій

d каталог

p іменованний канал

f звичайний файл

l символічне посилання.

Опції команди find:

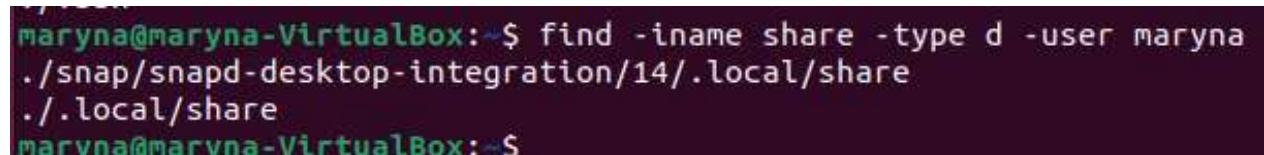
- name - пошук файлів по імені
- group - пошук по групі
- mtime - пошук по часу модифікації файлу
- atime - пошук файлів за датою останнього читання
- nogroup - пошук файлів, які не належать жодній групі
- nouser - пошук файлів без власників
- ctime час останньої зміни дескриптора файлу
- size n [c] пошук за розміром файлу, за замовчуванням в блоках. Для пошуку в байтах - с і +/- (більше / менше) вказаного розміру
- path пошук в шляху файлу
- perm шукати файли з зазначеними правами доступу
- iname – без врахування регістру при пошуку
- type n шукати файли зазначеного типу, наприклад: l - для символічних посилань
- user n шукати файли, які належать користувачу
- xdev шукати файли тільки на пристрої знаходження шляху пошуку/

Приклади використання команди find (рисунок 3.14):

find . -maxdepth 1 -name "*.php" // Пошуку файлів по імені в Linux тільки в цій папці

find ./test ./test2 -type f -name "*.c" // Шукати в двох каталогах одночасно

find . -user 111 // Знайти всі файли, що належать користувачеві



```
maryna@maryna-VirtualBox:~$ find -iname share -type d -user maryna
./snap/snapd-desktop-integration/14/.local/share
./.local/share
maryna@maryna-VirtualBox:~$
```

Рисунок 3.14 – Команда find

File-globbing – підстановка імен файлів.

Bash, сам по собі, не розпізнає регулярні вирази.

file-globbing – це спосіб оболонки надавати шаблони регулярних виразів.

Основна відмінність між регулярними виразами і глобінгом в тому, що Регулярні вирази використовуються в командах/функціях для зіставлення зі зразком у тексті, тоді як Глобування імен файлів використовується оболонками для зіставлення імен файлів та каталогів за допомогою підстановочних символів.

Фактично, Bash може виконувати підстановку імен файлів, цей процес називається "globbing", але при цьому не використовується стандартний набір регулярних виразів.

Замість цього, при виконанні підстановки імен файлів, проводиться розпізнавання і інтерпретація шаблонних символів.

Символи для підстановки:

***** - один і більше символів. ***ab** відповідає **ab**, **lab**, **aaab**, **1ab** і т.д.

? - тільки один символ. **?ab** відповідає **cab** і **1ab**

[at] - будь-який символ із зазначених в дужках. **[at] ab** відповідає **aab** і **tab**

[!c-z] - будь-який символ окрім тих, що вказані в дужках. **[!cz]** відповідає **aab** і **bab**

{0,1,9} - містить символи 0,1 і 9. **fl{0,1,9}** відповідають **fl0**, **fl1** і **fl9**

Приклади:

\$ cp file{1,2}.txt /dir1 // **file{1,2}.txt** – маска для імен файлів, які будуть скопійовані з поточного каталогу в каталог **dir1**

\$ rm -v file*.txt // видалення **'file1.txt'**, видалення **'file2.txt'**. **file*.txt** – маска для імен файлів, які видаляється.

\$ cp *.jpg /111/222 // використання оператора *****. Тут ***.jpg** це шаблон **globb**, який відповідає всім файлам, що закінчується на **.jpg** в поточному каталозі.

Це еквівалентно і набагато простіше, ніж перерахування поточного каталогу і введення всіх файлів поіменно вручну:

\$ ls

cat .jpg dog.jpg drawing.png recipes.txt zebra.jpg

\$ cp cat.jpg dog.jpg zebra.jpg /111/222

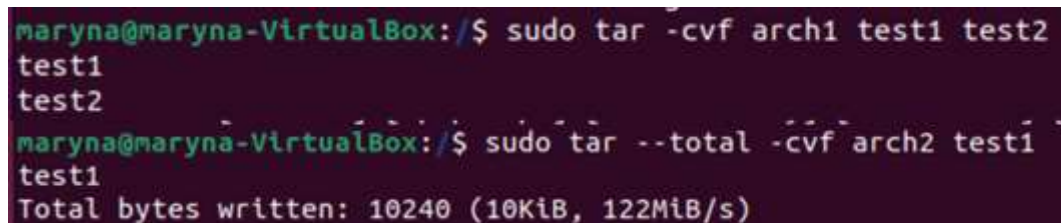
Архівація даних в Linux здійснюється різними утилітами.

tar Linux - вбудований архіватор.

Ключі команди tar:

- c створити архів
- f вказати ім'я архіву
- j стискати і розпаковувати файли в форматі bzip2
- r додати файли в існуючий архів
- v вивести детальну інформацію процесу
- t вивести список всіх файлів, що зберігаються в архіві
- u оновити існуючий архів
- x розархівувати файл
- z стискати і розпаковувати файли в форматі gzip

\$ tar --totals -cvf archive.tar file1 file2 file3 (рисунок 3.15)



```
maryna@maryna-VirtualBox:/$ sudo tar -cvf arch1 test1 test2
test1
test2
maryna@maryna-VirtualBox:/$ sudo tar --total -cvf arch2 test1
test1
Total bytes written: 10240 (10KiB, 122MiB/s)
```

Рисунок 3.15 – Команда tar – створення архіву

Команда **сріо** також використовується для архівації даних з відповідним ім'ям. Ця утиліта підтримує як власний бінарний формат архівів сріо, так і популярний формат архівів tar.

Ключі команди сріо:

- i** режим сору-ін для копіювання файлів з архіву (розархівування)

-o режим `cru-out` для створення архіву
-p режим `cru-pass`; не створює архів, тільки копіює файли з одного каталогу в інший.

Приклади використання (рисунок 3.16):

\$ cpio -i <archive.cpio // розархівування файлів з архіву формату CPIO в поточну директорію

\$ ls | cpio -o > archive.cpio // створення архіву формату CPIO з перенаправленням стандартного потоку виводу

\$ cpio -iv < arch.cpio file1.txt // вивільнення одного файлу з архіву



```
root@maryna-VirtualBox:/111# ls | cpio -ov > arch1.cpio
1.txt
21.txt
31.txt
arch1.cpio
1 block
root@maryna-VirtualBox:/111# cpio -iv < arch1.cpio
cpio: 1.txt not created: newer or same age version exists
1.txt
cpio: 21.txt not created: newer or same age version exists
21.txt
cpio: 31.txt not created: newer or same age version exists
31.txt
cpio: arch1.cpio not created: newer or same age version exists
arch1.cpio
1 block
root@maryna-VirtualBox:/111# cd /
root@maryna-VirtualBox:/# cpio -iv < /111/arch1.cpio
1.txt
21.txt
31.txt
arch1.cpio
1 block
```

Рисунок 3.16 – Команда `cpio`

Програми стиснення даних `gzip`, `bzip2`, `xz`

Програми `gzip`, `bzip2`, `xz` призначені для стиснення одиначних файлів. Стиснені файли мають розширення `.gz`, `bz2`, `xz` відповідно (рисунок 3.17).

Крім того, розпізнаються розширення `.tgz`, `.tbz`, `.txz`, є скороченням від подвійних розширень `.tar.gz`, `.tar.bz2`, `.tar.xz`.

Для розпакування використовуються ці ж програми з опцією `-d` (`decompress`).

1. При стисненні файлу оригінальний файл видаляється і створюється стиснений файл, до імені якого додається відповідне розширення.

2. При розпакуванні розширення відкидається, створюється нестиснений файл, а стиснений файл видаляється.

3. Винятком є випадки, коли ім'я файлу не задано, замість імені файлу стоїть "-" або додатково до імені файлу опція -с.

У перших двох випадках вихідні дані читаються зі стандартного вводу, а результат виводиться в стандартний вивід, в третьому - вихідні дані читаються з файлу, файл не видаляється, результат виводиться на стандартний вивід.

У програм стиснення є альтернативні імена. Запуск за цими іменами еквівалентний запуску з деяким набором опцій.

gunzip, bunzip2, unxz // запуск gzip, bzip2, xz з опцією -d

Ключі команди gzip і gunzip:

-d розпакування (застосовується тільки з командою gzip)

-f примусове стиснення / розпакування, навіть якщо файл має багато зв'язків, або навіть якщо файл вже існує

-r рекурсивне стиснення / розпакування

-v вказати детальний режим

Ключі команди bzip2 і bunzip2:

-d розпакування (застосовується тільки з командою bzip2)

-f примусове стиснення / розпакування, навіть якщо файл має багато зв'язків, або навіть якщо відповідний файл вже існує

-r рекурсивне стиснення / розпакування

-v вказати детальний режим

-z стиснення (застосовується тільки з командою bunzip2).

```
maryna@maryna-VirtualBox:/1$ ls -l
total 16
drwxr-xr-x 2 root root 4096 жов 21 13:46 11
-rw-rw-r-- 1 maryna maryna 6 жов 21 14:00 1.txt
drwxr-xr-x 2 root root 4096 жов 21 13:49 22
-rw-rw-r-- 1 maryna maryna 6 жов 21 14:00 2.txt
-rw-r--r-- 1 root root 0 жов 21 13:50 aaa
maryna@maryna-VirtualBox:/1$ gzip 2.txt
maryna@maryna-VirtualBox:/1$ ls -l
total 16
drwxr-xr-x 2 root root 4096 жов 21 13:46 11
-rw-rw-r-- 1 maryna maryna 6 жов 21 14:00 1.txt
drwxr-xr-x 2 root root 4096 жов 21 13:49 22
-rw-rw-r-- 1 maryna maryna 32 жов 21 14:00 2.txt.gz
-rw-r--r-- 1 root root 0 жов 21 13:50 aaa
maryna@maryna-VirtualBox:/1$ gzip -d 2.txt.gz
maryna@maryna-VirtualBox:/1$ ls -l
total 16
drwxr-xr-x 2 root root 4096 жов 21 13:46 11
-rw-rw-r-- 1 maryna maryna 6 жов 21 14:00 1.txt
drwxr-xr-x 2 root root 4096 жов 21 13:49 22
-rw-rw-r-- 1 maryna maryna 6 жов 21 14:00 2.txt
-rw-r--r-- 1 root root 0 жов 21 13:50 aaa
```

Рисунок 3.17 – Утиліта gzip

Утиліта **dd** - копіювання даних з одного місця в інше на двійковому рівні.

Досить часто системним адміністраторам доводиться копіювати різні двійкові дані. Наприклад, іноді може знадобитися зробити резервну копію жорсткого диска, створити порожній файл, заповнений нулями для організації простору підкачки або інший віртуальної файлової системи.

Для вирішення всіх цих завдань використовується утиліта **dd linux**.

Приклад використання:

```
$ dd if =/dev/urandom of =/dev/null bs=100M count=5
```

Параметри:

if вказує на джерело, тобто на те, звідки копіюємо. Вказується файл, який може бути як звичайним файлом, так і файлом пристрою.

of вказує на файл призначення. Те ж саме, писати можемо як в звичайний файл, так і безпосередньо в пристрій.

bs кількість байт, які будуть записані за раз. Можна представляти цей параметр, як розмір блоків даних, які будуть записані або прочитані.

count число, яке вказує: скільки блоків буде скопійовано.

Таким чином, описана команда читає $5 * 100$ мегабайт з пристрою `/dev/urandom` в пристрій `/dev/null`. Надаючи цій команді смислове навантаження виходить, що система згенерує 500 мегабайт випадкових значень і запише їх у `null` пристрій.

Приклад створення образу диска:

\$ dd if = /dev/cdrom of = image.iso

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №4 Стандартні потоки Linux

Мета: навчитись працювати зі потоками даних, обробляти текстові потоки за допомогою фільтрів, використовувати регулярні вирази для обробки текстових даних (пошуку, заміни, видалення і т.д.).

Завдання

Зайти в систему під користувачем, якого створено в першій роботі. Ознайомитись з потоками Linux (stdin, stdout, stderr), перенаправленням даних в потоки/ з потоків. Навчитися користуватися перенаправленням даних використовуючи символи перенаправлення даних; командою пошуку `grep`, використовуючи регулярні вирази.

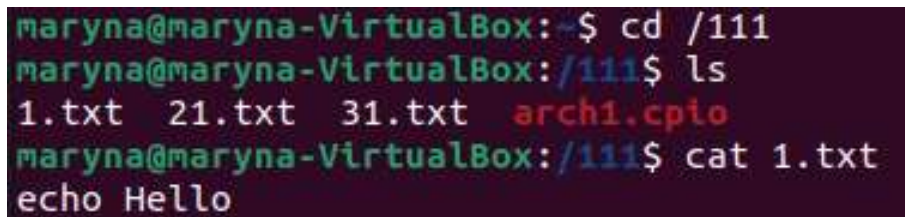
Теоретичні відомості

При роботі з текстовими файлами можливо здійснювати пошук, фільтрацію і перенаправлення інформації в інші файли [7].

Команда **cat** використовується для виведення вмісту текстового файлу на екран. Якщо команді передати кілька файлів, то вони виведуться разом.

cat опції... [файл]...

cat mytext.txt



```
maryna@maryna-VirtualBox:~$ cd /111
maryna@maryna-VirtualBox:/111$ ls
1.txt  21.txt  31.txt  arch1.cpio
maryna@maryna-VirtualBox:/111$ cat 1.txt
echo Hello
```

Рисунок 4.1 – Команда cat

Також команда **cat** використовується для створення файлу (рисунок 4.2).

Потрібно:

- написати команду **cat > filename**, Натиснути Enter,
- написати один або кілька рядків тексту, Натиснути Enter,
- для завершення вводу та збереження тексту в файл - натиснути **Ctrl+D**

cat > mytext.txt

```
123333333maryna@maryna-VirtualBox:~$ cat > 1.txt
1234
asdf
asfdg
zxcvb
maryna@maryna-VirtualBox:~$ cat 1.txt
1234
asdf
asfdg
zxcvb
maryna@maryna-VirtualBox:~$ █
```

Рисунок 4.2 – Створення файлу з вмістом за допомогою команди cat

Команда **cut** використовується для вибірки окремих полів з рядків файлу (рисунок 4.3). Вибірка полів проводиться за допомогою роздільників.

cut опції... [файл]...

Опції команди:

- b - номер байта, набір або діапазон байтів, які підлягають вирізанню
- c - символ, який слід вирізати. Також можна вказувати набір або діапазон символів
- d - користувач встановлює свій роздільник замість стандартного TAB
- f - перелік полів для вирізання
- s - cut не виводить рядки, де нема роздільника.

```
maryna@maryna-VirtualBox:/111$ cat 1.txt
echo Hello
maryna@maryna-VirtualBox:/111$ cut -c 1-3 1.txt
ech
maryna@maryna-VirtualBox:/111$ cut -b 2-6 1.txt
cho H
```

Рисунок 4.3 – Команда cut

Команда **expand** призначена для заміни символів табуляції на символи пробілів в текстових файлах (рисунок 4.4).

Команда **unexpand** призначена для заміни символів пробілів на символи TAB в текстових файлах (рисунок 4.4).

```
maryna@maryna-VirtualBox:~/MYDIR$ expand temp.txt
111      222      333      555      6666
maryna@maryna-VirtualBox:~/MYDIR$ expand -t 15 temp.txt
111              222              333              555              6666
```

Рисунок 4.4 – Команды expand, unexpand

Команда **fmt** використовується для форматування текстових файлів.

fmt -w 50 1.txt (-w – ширина ряда) (рисунок 4.5).

[illegible]

Рисунок 4.5 – Команда fmt

Команда **head** виводить перші 10 рядків з вказаного файлу (рисунк 4.6). Якщо вказано декілька імен файлів, команда друкує дані по кожному файлу окремо.

head [опції]... [ФАЙЛ]...

Опції команди:

-n --lines показує вказану кількість рядків

-c --bytes показує вказану кількість байт

```

maryna@maryna-VirtualBox:/111$ cat 21.txt
ssssssssssssssssssss
ssss
ssss dgsdgsfgdsfg sdgfasgdfsgds asdfgsadgsdag
1
2
3
4
5
6
7
8
3
4
5
76
354
36
67

maryna@maryna-VirtualBox:/111$ head 21.txt
ssssssssssssssssssss
ssss
ssss dgsdgsfgdsfg sdgfasgdfsgds asdfgsadgsdag
1
2
3
4
5
6
7

maryna@maryna-VirtualBox:/111$ head -n 3 21.txt
ssssssssssssssssssss
ssss
ssss dgsdgsfgdsfg sdgfasgdfsgds asdfgsadgsdag
maryna@maryna-VirtualBox:/111$

```

Рисунок 4.6 – Команда head

Команда **tail** виводить останні 10 рядків з вказаного файлу (рисунок 4.7). Якщо вказано декілька імен файлів, даним кожного файлу передують його ім'я.

tail [опції]... [ФАЙЛ]...

Опції команди:

-n --lines показує вказану кількість рядків

-c --bytes показує вказану кількість байтів

```

maryna@maryna-VirtualBox:/111$ cat 21.txt
ssssssssssssssssssss
ssss
ssss dgsdgsfgdsfg sdgfasgdfsgds asdfgsadgsdag
1
2
3
4
5
6
7
8
3
4
5
76
354
36
67

maryna@maryna-VirtualBox:/111$ tail 21.txt
6
7
8
3
4
5
76
354
36
67

maryna@maryna-VirtualBox:/111$ tail -n 5 21.txt
5
76
354
36
67

```

Рисунок 4.7 – Команда tail

Команда **od** в Linux використовується для перетворення вмісту файлу в різні формати, вісімковий форматат -формат за замовчуванням (рисунок 4.8). Ця команда особливо корисна під час налагодження сценаріїв Linux на наявність небажаних змін або символів.

od [опції]... [файл]...

Опції:

-b показує вміст файлу у вісімковому форматі

-c показує вміст файлу в символьному форматі.

```
maryna@maryna-VirtualBox: /111$ od -b 21.txt
00000000 163 163 163 163 144 144 144 144 144 144 144 144 144 144 144
00000020 144 144 163 012 163 163 163 163 012 163 163 163 040 144 147
00000040 163 144 147 144 163 146 147 144 163 146 147 040 163 144 147 146
00000060 141 163 147 144 146 163 147 144 163 040 141 163 144 146 147 163
00000080 141 144 147 163 144 141 147 012 061 012 062 012 063 012 064 012
00000100 065 012 066 012 067 012 070 012 063 012 064 012 065 012 067 066
00000120 012 063 065 064 012 063 066 012 066 067 012
00000140
00000153
maryna@maryna-VirtualBox: /111$ od -c 21.txt
00000000 s s s s d d d d d d d d d d
00000020 d d s \n s s s s \n s s s s d d g r
00000040 s d g d s f g d s f g s d f g s
00000060 a s g d f s g d s \n 1 \n 2 \n 3 \n 4 \n 5 \n 6
00000080 s \n 6 \n 7 \n 8 \n 3 \n 4 \n 5 \n 7 \n 6
00000100 \n 3 5 4 \n 3 6 \n 6 7 \n
00000120
00000140
00000153
maryna@maryna-VirtualBox: /111$ cat 21.txt
ssssssssssssssssss
ssss
ssss dgsgdsfgdsfg sdgfasgdfsgds asdfgsadgsdag
1
2
3
4
5
6
7
8
3
4
5
76
354
36
67
```

Рисунок 4.8 – Команда od

Команда **join** - для об'єднання рядків двох файлів на основі ключового поля, що є в обох файлах (рисунок 4.9).

Наприклад, є два файли, і є потреба об'єднати два файли. Файл 1, який містить імена, файл 2 – ідентифікатори, і потрібно об'єднати обидва файли таким чином, щоб імена та відповідні ідентифікатори відображалися в одному рядку.

join [опції]... [файл1] [файл2]

якщо кількість рядків в файлах різна,

У таких випадках можна використати ключ `-a`, до звичайного виведення буде додано рядки з одного файлу, для яких нема відповідних рядків в другому файлі. Цей параметр вимагає від користувача передачі номера файлу (1 або 2), щоб інструмент знав, про який файл йдеться.

```
maryna@maryna-VirtualBox:~/MYDIR$ cat file1.txt
1 A
2 B
3 C
4 D
5 E
maryna@maryna-VirtualBox:~/MYDIR$ cat file2.txt
1 101
2 102
3 103
4 104
maryna@maryna-VirtualBox:~/MYDIR$ join file1.txt file2.txt
1 A 101
2 B 102
3 C 103
4 D 104
maryna@maryna-VirtualBox:~/MYDIR$ join file1.txt file2.txt -a 1
1 A 101
2 B 102
3 C 103
4 D 104
5 E
maryna@maryna-VirtualBox:~/MYDIR$ join file1.txt file2.txt -a 2
1 A 101
2 B 102
3 C 103
4 D 104
```

Рисунок 4.9 – Команда `join`

Команда **`nl`** використовується для нумерації рядків, приймаючи вхідні дані з файлу або з потоку вводу. Вона копіює кожен вказаний файл у потім виводу із додаванням номерів рядків перед рядками (рисунок 4.10).

`nl [опції]... [файл]...`

Опції:

`-b a` параметр використовується для нумерації всіх рядків, порожніх чи непорожніх.

```

maryna@maryna-VirtualBox:~/MYDIR$ cat file1.txt
A
B
C

D

E
maryna@maryna-VirtualBox:~/MYDIR$ nl file1.txt
  1  A
  2  B
  3  C

  4  D

  5  E
maryna@maryna-VirtualBox:~/MYDIR$ nl -b a file1.txt
  1  A
  2  B
  3  C
  4
  5  D
  6
  7  E
maryna@maryna-VirtualBox:~/MYDIR$

```

Рисунок 4.10 – Команда nl

Команда **paste** дозволяє об'єднувати рядки файлів по горизонталі (рисунок 4.11). Вона виводить рядки, що складаються з рядків кожного з файлів, вказаного як аргумент, розділених табуляцією.

Якщо вихідні файли не задано або якщо як аргумент задано “-”, команда буде зчитувати зі стандартного вводу.

paste [опції]... [файл]...

Опції:

-s об'єднує всі рядки у файлі. якщо роздільник не вказано, для розділення стовпців використовується TAB за замовчуванням.

-d використовується для визначення роздільника. Комбінація **-d, -s** усі рядки у файлі об'єднуються в один рядок, розділені комою.

```

maryna@maryna-VirtualBox:~/MYDIR$ paste file1.txt
1 A
2 B
3 C
4 D
5 E
maryna@maryna-VirtualBox:~/MYDIR$ paste -s file1.txt
1 A      2 B      3 C      4 D      5 E
maryna@maryna-VirtualBox:~/MYDIR$ paste -d, -s file1.txt
1 A,2 B,3 C,4 D,5 E
maryna@maryna-VirtualBox:~/MYDIR$ paste file2.txt
1 101
2 102
3 103
4 104
maryna@maryna-VirtualBox:~/MYDIR$ paste file1.txt file2.txt
1 A      1 101
2 B      2 102
3 C      3 103
4 D      4 104
5 E
maryna@maryna-VirtualBox:~/MYDIR$ paste -d! file1.txt file2.txt
1 A!1 101
2 B!2 102
3 C!3 103
4 D!4 104
5 E!

```

Рисунок 4.11 – Команда paste

Команда **sort** використовується для впорядкування текстових рядків в певному порядку (рисунок 4.12).

Команда має певні особливості:

- рядки з номерами будуть передувати алфавітним рядкам.
- ті рядки, які мають малі літери, будуть відображені раніше, ніж рядки, які містять такі самі символи у верхньому регістрі.

sort [опції]... [файл]...

Опції:

- r - сортувати у зворотному порядку
- n - сортувати рядки як числові значень
- u – ігнорувати повторювані рядки

```

maryna@maryna-VirtualBox:~/MYDIR$ cat file1.txt
334
65
1
maryna@maryna-VirtualBox:~/MYDIR$ sort file1.txt
1
334
65
maryna@maryna-VirtualBox:~/MYDIR$ sort -n file1.txt
1
65
334

```

Рисунок 4.12 – Команда sort

Команда **wc** використовується для визначення кількості рядків, кількості слів, кількості байт і символів у файлах (рисунок 4.13). Команда часто використовується в скриптах

wc [опції]... [файл]...

wc без параметрів відображає загальну кількість рядків, слів і символів у файлі

Опції команди:

- c – розмір файлу в байтах
- m - кількість символів в файлі
- l - кількість рядків
- w - кількість слів

```

maryna@maryna-VirtualBox:~/MYDIR$ cat file1.txt
cat
cut
but
34
maryna@maryna-VirtualBox:~/MYDIR$ wc file1.txt
 4  4 15 file1.txt
maryna@maryna-VirtualBox:~/MYDIR$ wc -l file1.txt
4 file1.txt
maryna@maryna-VirtualBox:~/MYDIR$ wc -w file1.txt
4 file1.txt
maryna@maryna-VirtualBox:~/MYDIR$ wc -c file1.txt
15 file1.txt
maryna@maryna-VirtualBox:~/MYDIR$ wc -m file1.txt
15 file1.txt

```

Рисунок 4.13 – Команда wc

Команда **uniq** допомагає виявити сусідні повторювані рядки, а також видаляє повторювані рядки. **uniq** відфільтровує суміжні відповідні рядки з вхідного файлу і записує відфільтровані дані у вихідний файл. (рисунок 4.14)

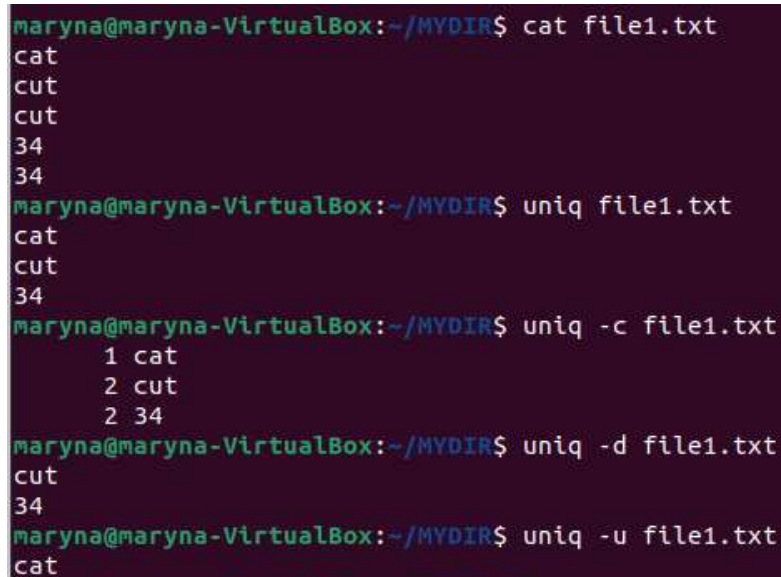
uniq [опції] [input[output]]

Опції команди:

-c - на початку кожного рядка виводить число, яке означає кількість повторів

-d – виводить тільки повторювані рядки

-u – виводить унікальні рядки



```
maryna@maryna-VirtualBox:~/MYDIR$ cat file1.txt
cat
cut
cut
34
34
maryna@maryna-VirtualBox:~/MYDIR$ uniq file1.txt
cat
cut
34
maryna@maryna-VirtualBox:~/MYDIR$ uniq -c file1.txt
 1 cat
 2 cut
 2 34
maryna@maryna-VirtualBox:~/MYDIR$ uniq -d file1.txt
cut
34
maryna@maryna-VirtualBox:~/MYDIR$ uniq -u file1.txt
cat
```

Рисунок 4.14 – Команда **uniq**

Команда **split** розбиття великого файлу на менші (рисунок 4.15).

split [опції] [input[output/префікс імені]]

Опції:

-b розділити файл за розміром

-n число розділити файл на певну кількість файлів

-l розділити файл, встановивши максимальну кількість рядків в результируючих файлах

--verbose – виводить інформацію про нові файли перед створенням.

```
maryna@maryna-VirtualBox:~/MYDIR$ split -l 2 --verbose file1.txt
creating file 'xaa'
creating file 'xab'
creating file 'xac'
maryna@maryna-VirtualBox:~/MYDIR$ split -b 2 --verbose file1.txt
creating file 'xaa'
creating file 'xab'
creating file 'xac'
creating file 'xad'
creating file 'xae'
creating file 'xaf'
creating file 'xag'
creating file 'xah'
creating file 'xai'
maryna@maryna-VirtualBox:~/MYDIR$ split -n 3 --verbose file1.txt
creating file 'xaa'
creating file 'xab'
creating file 'xac'
maryna@maryna-VirtualBox:~/MYDIR$ split -n 3 --verbose file1.txt /qwerty
creating file '/qwertyaa'
split: /qwertyaa: Permission denied
maryna@maryna-VirtualBox:~/MYDIR$ sudo split -n 3 --verbose file1.txt /qwerty
[sudo] password for maryna:
creating file '/qwertyaa'
creating file '/qwertyab'
creating file '/qwertyac'
```

Рисунок 4.15 – Команда split

Команда **more** використовується для перегляду текстових файлів у командному рядку, відображаючи по одному екрану, якщо файл великий.

Команда more також дозволяє користувачу прокручувати сторінку вгору та вниз.

more [-опції]... [файл]

Команда **less** дозволяє читати вміст текстового файлу по одній сторінці (один екран) за раз. якщо файл великий, відкривається не весь файл, а сторінка за сторінкою.

Наприклад, якщо великий файл і ви читаєте його за допомогою будь-якого текстового редактора, то весь файл буде завантажено в основну пам'ять. Команда less завантажує файл частину за частиною, що робить його швидшим.

less [файл]

Стандартні потоки в Linux

У Linux є три стандартні потоки, які є каналами зв'язку для процесів в системі:

- стандартний потік вводу STDIN
- стандартний потік виводу STDOUT
- стандартний потік помилок STDERR

Кожен потік має свій номер, який називається файловий дескриптор. Файловий дескриптор - це невід'ємне ціле число. Коли створюється новий потік введення-виведення, ядро повертає процесу, який створив потік введення-виведення, його дескриптор.

Стандартний потік введення (STDIN). Канал приймає дані для обробки та подальшої передачі на канал STDOUT і / або STDERR. Стандартним потоком введення (stdin) є клавіатура. У більшості випадків, потік вхідних даних вводиться саме через клавіатуру. Номер файлового дескриптора - 0.

Стандартний потік виведення (STDOUT) - це потік, в який програми виводять результати роботи. Зазвичай ці дані відображаються на екрані монітора. Номер файлового дескриптора – 1.

Стандартний потік помилок (STDERR) - потік, використовуваний для виведення повідомлень про помилки і діагностики. Цей потік незалежний від стандартного потоку виведення і може перенаправлятися окремо. Номер файлового дескриптора – 2.

Для перенаправлення потоків даних в терміналі, застосовують певні символи.

> Створює новий файл, який містить дані стандартного потоку виведення. Якщо вказаний файл існує, то він перезаписується.

\$ ls >1.txt // перенаправлено stdout команди ls в файл 1.txt (рисунк 4.16)

```
maryna@maryna-VirtualBox:~/MYDIR$ ls
1hl.txt 2sl.txt 3.txt 4.txt 5.txt 6.txt 7.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls > 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ cat 1.txt
1hl.txt
1.txt
2sl.txt
3.txt
4.txt
5.txt
6.txt
7.txt
```

Рисунок 4.16 – Перенаправлення stdout команди ls в файл

>> Створює новий файл, який містить дані стандартного потоку виведення. Якщо вказаний файл існує, то він дописується.

\$ cat >> 2.txt

2> Створює новий файл, який містить дані стандартного потоку помилок. Якщо вказаний файл існує, то він перезаписується.

Перенаправимо stderr команди rm:

\$ rm example.txt 2> 1.txt

Тут використовується номер потоку (дескриптор) stderr (2). За замовчуванням оператор > перенаправляє потік stdout, який має номер 1. Щоб направити інший потік, треба перед оператором > поставити його номер (рисунок 4.17).

```
maryna@maryna-VirtualBox:~/MYDIR$ ls
1hl.txt 1.txt 2sl.txt 3.txt 4.txt 5.txt 6.txt 7.txt
maryna@maryna-VirtualBox:~/MYDIR$ rm example.txt 2> 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ cat 1.txt
rm: cannot remove 'example.txt': No such file or directory
maryna@maryna-VirtualBox:~/MYDIR$
```

Рисунок 4.17 – Перенаправлення stderr в файл

2 >> Додає інформацію зі стандартного потоку виводу в існуючий файл (рисунок 4.18). Якщо вказаний файл не існує, то він створюється.

```
maryna@maryna-VirtualBox:~/MYDIR$ rm example.txt 2>> 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ cat 1.txt
rm: cannot remove 'example.txt': No such file or directory
rm: cannot remove 'example.txt': No such file or directory
```

Рисунок 4.18 – Перенаправлення stderr в файл (додавання)

&> або **>&** Створює новий файл, який містить дані стандартного потоку виведення і потоку помилок. Якщо вказаний файл існує, то він перезаписується.

Інша форма запису **filename 2>&1** виводить стандартні потоки виведення і помилок в один файл.

&> filename ~ >& filename ~ filename 2>&1

\$ rm exmple.txt> 1.txt 2> & 1 // направлено потік stdout в файл 1.txt, а потім направили stderr туди ж, куди спрямований stdout за допомогою оператора **&** перед номером потоку.

< Перенаправляє вміст файлу як стандартний потік введення.

Приклад: є два файли: **list1.txt** і **list2.txt**, кожен з яких містить невідсортований список рядків. У кожному зі списків є унікальні для нього елементи, але деякі з елементів списку збігаються. Ми можемо знайти рядки, які є і в першому, і в другому списках, застосувавши команду **comm**, але перш ніж її використовувати, списки треба впорядкувати.

Команда **comm** – утиліта, яка зчитує файл1 і файл2, які мають бути попередньо відсортовані і генерує результат, що складається з трьох стовпчиків:

- рядки, знайдені тільки у файлі 1;
- рядки, знайдені тільки у файлі 2;
- та рядки, спільні для обох файлів.

Використаємо команду **sort**, яка повертає відсортований список в термінал без збереження відсортованих даних в файл, з якого вони були взяті.

\$ sort list1.txt > list11.txt

\$ sort list2.txt > list22.txt

\$ comm list1.txt list2.txt

Можна скористатися символом `<` для перенаправлення відсортованої версії кожного файлу команді `comm` (рисунок 4.19).

\$ comm <(sort list1.txt) <(sort list2.txt)



```
maryna@maryna-VirtualBox:~/MYDIR$ cat >> list1.txt
2
5
1
4^Z
[1]+  Stopped                  cat >> list1.txt
maryna@maryna-VirtualBox:~/MYDIR$ cat >> list2.txt
5
7
3^Z
[2]+  Stopped                  cat >> list2.txt
maryna@maryna-VirtualBox:~/MYDIR$ comm <(sort list1.txt) <(sort list2.txt)
1
2
7      5
```

Рисунок 4.19 – Перенаправлення `<`

`<<` Перенаправлення з файлу в файл без перезапису файлу.

`|` (pipe) Перенаправляє стандартний потік виведення однієї команди в стандартний потік введення іншої команди.

\$ command1 | command2

В даному прикладі виконується команда `command1`, її потік виведення використовується, як потік введення при виконанні `command2`.

Команда **tr** використовується для заміни, доповнення або видалення символів зі стандартного потоку вводу, з виведенням результату в стандартний потік виведення. Програма обробляє текст довільно.

tr [опції]... [набір1] [набір2]

Опції:

-d - видаляє всі символи, які перераховані в наборі. Наприклад видалити символ "e", `echo "Hello" | tr -d "e"`

-s - замінює послідовність повторюваних символів на один такий символ

-с - доповнює набір символів у рядку, тобто операції застосовуються до символів, які не входять до набору

-х – заміна символу з першого набору, символом з другого

Для перетворення з нижнього регістру на верхній регістр можна використовувати попередньо визначені набори в tr:

```
$ cat 1.txt | tr "[a-z]" "[A-Z]"
```

```
$ cat 1.txt | tr "[:lower:]" "[:upper:]"
```

```
$ echo "HELLO" | tr -d 'L' // видалення символу з першого вказаного набору
```

Використання символу | (pipe) для перенаправлення результату команди echo команді tr (рисунок 4.20):



```
maryna@maryna-VirtualBox: ~/MYDIR$ cat file1.txt
cat
cut
cut
cut
34
34
maryna@maryna-VirtualBox: ~/MYDIR$ cat file1.txt | tr c 1
1at
1ut
1ut
34
34
maryna@maryna-VirtualBox: ~/MYDIR$ cat file1.txt | tr -d "3"
cat
cut
cut
4
4
maryna@maryna-VirtualBox: ~/MYDIR$ cat file1.txt | tr "[a-z]" "[A-Z]"
CAT
CUT
CUT
CUT
34
34
```

Рисунок 4.20 – Перенаправлення | та команда tr

Утиліти (команди) для роботи з потоками

Утиліта(команда) **tee** для роботи з потоками

Команда tee читає зі стандартного вводу і записує як в стандартний вивід, так і в один або кілька файлів одночасно.

tee [опції]... [файл]...

Опції команди:

-a або -append - для запису виведення в кінець існуючого файлу
-i або -ignore-interrupts - використовується, щоб ігнорувати сигнали переривання

-help - для показу всіх можливих операцій.

Для збереження результату команди можна передати один або кілька файлів.

Приклади використання **tee** (рисунок 4.21):

\$ ls -la | tee 1.txt // команда tee використовується для збереження виводу **ls -la** в файл **1.txt**

\$ pwd | tee -a 1.txt // вивід команди **pwd** буде додано в кінець файлу **1.txt** (ключ -a)

\$ date | tee 1.txt 2.txt // тут збереження виведення **date** в два файли: **1.txt** та **2.txt**

\$ ls | tee 4.txt | wc -lcw // вивід команди tee перенесений в іншу команду. Команда використовується для запису виводу **ls** в файл **4.txt** і підрахунку загального числа рядків, слів та символів у файлі **4.txt**.

```
maryna@maryna-VirtualBox: ~/MYDIR$ ls -la | tee tee.txt
total 48
drwxrwxr-x+ 2 maryna maryna 4096 жов 16 16:25 .
drwxr-x--- 17 maryna maryna 4096 жов 12 15:08 ..
-rw-rw-r-- 1 maryna maryna 825 жов 16 16:25 123.txt
-rw-rw-r-- 1 maryna2 maryna 5 жов 12 15:49 1hl.txt
-rw-rw-r-- 1 maryna maryna 825 жов 16 16:26 1.txt
lrwxrwxrwx 1 maryna maryna 5 жов 12 15:56 2sl.txt -> 2.txt
-rw-rw-r-- 1 maryna maryna 8 жов 12 14:33 3.txt
-rw-rw-r-- 1 maryna maryna 6 жов 12 14:45 4.txt
-rw-r-- 1 maryna maryna 6 жов 12 14:49 5.txt
-rw-rw-r-- 1 maryna maryna 7 жов 12 15:11 6.txt
-rw-r-- 1 maryna maryna 7 жов 12 15:11 7.txt
-rw-rw-r-- 1 maryna maryna 6 жов 16 12:45 list1.txt
-rw-rw-r-- 1 maryna maryna 4 жов 16 12:46 list2.txt
-rw-rw-r-- 1 maryna maryna 0 жов 16 16:26 tee.txt
maryna@maryna-VirtualBox: ~/MYDIR$ date | tee -a tee.txt
неділя, 16 жовтня 2022 16:28:00 +0300
maryna@maryna-VirtualBox: ~/MYDIR$ cat tee.txt
total 48
drwxrwxr-x+ 2 maryna maryna 4096 жов 16 16:25 .
drwxr-x--- 17 maryna maryna 4096 жов 12 15:08 ..
-rw-rw-r-- 1 maryna maryna 825 жов 16 16:25 123.txt
-rw-rw-r-- 1 maryna2 maryna 5 жов 12 15:49 1hl.txt
-rw-rw-r-- 1 maryna maryna 825 жов 16 16:26 1.txt
lrwxrwxrwx 1 maryna maryna 5 жов 12 15:56 2sl.txt -> 2.txt
-rw-rw-r-- 1 maryna maryna 8 жов 12 14:33 3.txt
-rw-rw-r-- 1 maryna maryna 6 жов 12 14:45 4.txt
-rw-r-- 1 maryna maryna 6 жов 12 14:49 5.txt
-rw-rw-r-- 1 maryna maryna 7 жов 12 15:11 6.txt
-rw-r-- 1 maryna maryna 7 жов 12 15:11 7.txt
-rw-rw-r-- 1 maryna maryna 6 жов 16 12:45 list1.txt
-rw-rw-r-- 1 maryna maryna 4 жов 16 12:46 list2.txt
-rw-rw-r-- 1 maryna maryna 0 жов 16 16:26 tee.txt
неділя, 16 жовтня 2022 16:28:00 +0300
```

Рисунок 4.21 –Команда tee

Команда **xargs** – потужна утиліта, зчитує потоки даних зі стандартного вводу, а потім генерує і виконує командні рядки. Використовується для об’єднання команд.

За допомогою утиліти xargs вивід попередньої команди можна передати, як аргументи, наступній команді .

Всі пропуски і пробіли у вхідних даних сприймаються як роздільники, порожні рядки ігноруються.

[Команда-джерело даних] | xargs [опції xargs] [команда-аргумент]

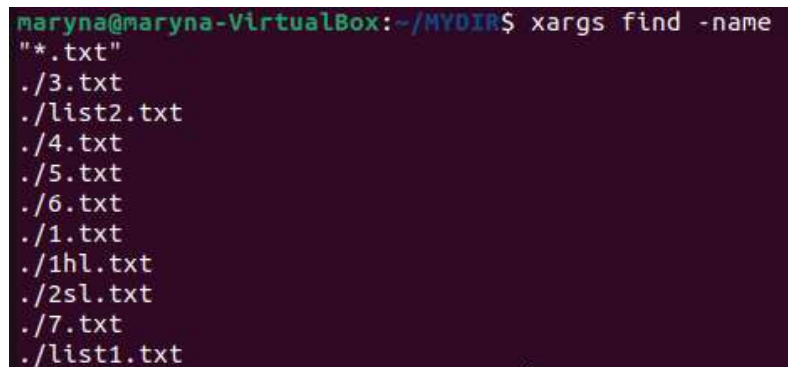
Якщо команда в якості аргументу не передається, за замовчуванням утиліта виконує команду echo, але можна в явному вигляді вказати будь-яку іншу команду.

Наприклад, можна передати в якості аргументу команду find з опцією -name, а потім ввести в стандартному потоці введення ім'я файлу (або тип файлів), який ви хочете знайти (рисунок 4.22).

\$xargs find -name

“*.txt”

// натиснути Ctrl+D



```
maryna@maryna-VirtualBox:~/MYDIR$ xargs find -name
"*.txt"
./3.txt
./list2.txt
./4.txt
./5.txt
./6.txt
./1.txt
./1hl.txt
./2sl.txt
./7.txt
./list1.txt
```

Рисунок 4.22 –Команда xargs

Наприклад, команда виконується без опцій і введено текст 'Hello' в стандартний потік введення. Після натискання Ctrl+D (щоб повідомити xargs

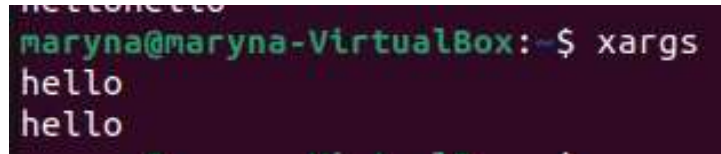
про завершення введення), була автоматично виконана команда `echo`, і на екран знову був виведений текст 'Hello' (рисунок 4.23) .

\$ xargs

Hello /введемо текст

Ctrl+D

Hello /на екран виведеться цей текст



```
maryna@maryna-VirtualBox:~$ xargs
hello
hello
```

Рисунок 4.23 –Команда `xargs`

Опції `xargs`:

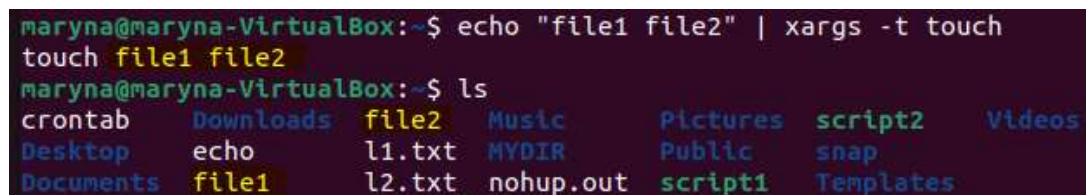
`-0` (нуль) – використовувати як роздільник нульовий символ `NULL` («\0»).
За замовчуванням роздільники: пробіл, переведення рядка, `tab`, які означають кінець збереженого рядка

`-L [число]` - `max-lines` - вказує кількість рядків, що передаються в одну команду, за замовчуванням - всі

`-n [число]` вказує максимальну кількість аргументів, які можна передати команді за одну підстановку

\$ echo "file1 file2" | xargs -n1 touch

`-t` увімкнути подробиці про виконання: команда і кожен побудований список аргументів перед виконанням виводиться в стандартний потік помилок (рисунок 4.24).



```
maryna@maryna-VirtualBox:~$ echo "file1 file2" | xargs -t touch
touch file1 file2
maryna@maryna-VirtualBox:~$ ls
crontab  Downloads  file2      Music      Pictures  script2    Videos
Desktop  echo       l1.txt    MYDIR      Public    snap
Documents file1     l2.txt    nohup.out  script1   Templates
```

Рисунок 4.24 –Команда `xargs`

-p - режим із запрошенням: xargs перед кожним викликом команди запитує підтвердження, друкується виклик команди, який повинен бути виконаний, а за ним запрошення ?

Відповідь Y призводить до виконання команди, при будь-якій іншій відповіді, цей виклик ігнорується (рисунок 4.25).

```
maryna@maryna-VirtualBox:~$ echo "file4 file3" | xargs -p touch
touch file4 file3?...Y
maryna@maryna-VirtualBox:~$ ls
crontab      Downloads  file2      l1.txt     MYDIR      Public     snap
Desktop      echo      file3      l2.txt     nohup.out  script1    Templates
Documents    file1     file4      Music      Pictures   script2    Videos
```

Рисунок 4.25 –Команда xargs

Прикладі: пошук всіх файлів з розширенням txt, що містять текст 'abc', можна виконати наступним чином:

\$ find -name "*.txt" | xargs grep "abc"

В даному випадку команда xargs отримує вхідні дані від команди find. І по черзі застосовує назву файлу до команди grep. Яка в свою чергу вже шукає рядок в файлі (рисунок 4.26).

```
maryna@maryna-VirtualBox:~/MYDIR$ cat list1.txt
2
5
1
maryna@maryna-VirtualBox:~/MYDIR$ cat list2.txt
5
7
maryna@maryna-VirtualBox:~/MYDIR$ find -name "*.txt" | xargs grep "5"
./list2.txt:5
./5.txt:55555
grep: ./2sl.txt: No such file or directory
./7.txt:567567
./list1.txt:5
```

Рисунок 4.26 –Команда xargs

\$ find . -print | xargs -n2 echo // перегляд файлів в поточному робочому каталозі і всіх підкаталогах в двох стовпчиках

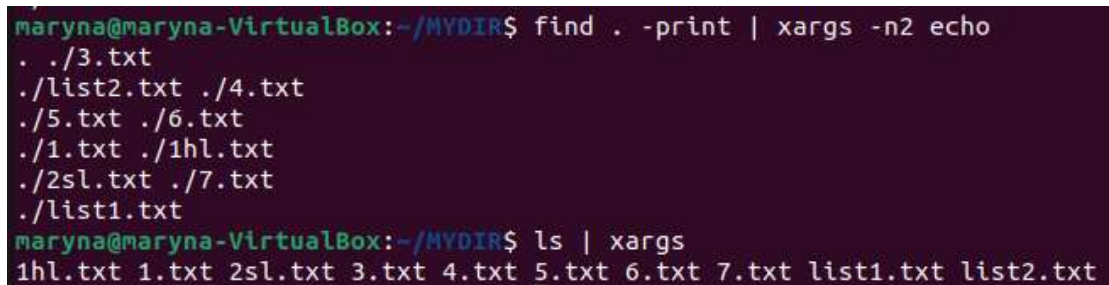
\$ find . -group root | xargs chgrp user // зміна групи користувачів, ця команда змінить групу власників з root на user для всіх файлів і каталогів в поточній директорії.

Можна перетворити виведення безлічі рядків з команди ls в один рядок, використовуючи команду xargs:

\$ ls | xargs

\$ man xargs // отримати додаткову інформацію по команді xargs

На основі цих прикладів можна будувати вже свої конструкції з використання (рисунок 4.27).



```
maryna@maryna-VirtualBox:~/MYDIR$ find . -print | xargs -n2 echo
. ./3.txt
./list2.txt ./4.txt
./5.txt ./6.txt
./1.txt ./1hl.txt
./2sl.txt ./7.txt
./list1.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls | xargs
1hl.txt 1.txt 2sl.txt 3.txt 4.txt 5.txt 6.txt 7.txt list1.txt list2.txt
```

Рисунок 4.27 –Команда xargs

Регулярні вирази

Утиліта **grep** (generalized regular expression processor), яка приймає регулярні вирази та ім'я файла для пошуку текстових рядків в файлах.

Джерелом даних для неї може бути стандартний потік введення stdin.

Регулярні вирази ("Regex" або "regex")- інструмент для пошуку тексту за шаблоном, обробки та зміни рядків.

Регулярні вирази дозволяють виконувати:

- перевірку введення тексту;
- пошук та заміну тексту у файлі;
- пакетне перейменування файлів;
- перевірку рядка на відповідність шаблону та інше.

Основна відмінність між регулярними виразами і глобінгом в тому, що регулярні вирази використовуються в командах/функціях для зіставлення зі зразком у тексті, тоді як Глобування імен файлів використовується оболонками для зіставлення імен файлів та каталогів за допомогою підстановочних символів.

Використання регулярних виразів: **sed**, **grep**, **egrep**, **fgrep**:

grep - стандартна для Linux та Unix систем утиліта командного рядка, яка знаходить рядки, які відповідають заданому регулярному виразу, і виводить їх.

egrep і **fgrep** - окремі випадки утиліти **grep** призначені для роботи з розширеним синтаксисом регулярних виразів і для пошуку фіксованих рядків, відповідно.

sed - утиліта для пошуку, заміни, вставки, видалення тексту.

Символи,, які дають можливість будувати шаблони:

' Змінні і команди всередині одинарних лапках не обробляються

" Змінні всередині подвійних лапок обробляються, а команди — не обробляються

` Усередині зворотних лапок обробляються і змінні, і команди

\ Екранування спеціальних символів

~ Символ тільди відповідає домашньому каталогу користувача

. Будь-який символ

[] Відповідає будь-якому символу, який знаходиться всередині дужок

[^] Відповідає будь-якому символу, окрім того, який знаходиться всередині дужок

* Будь-яка кількість символів

{ } Визначення допустимої кількості повторень виразу

^ Символ ^ позначає початок рядка

\$ Символ \$ відповідає кінцю рядка

() Угрупування виразів

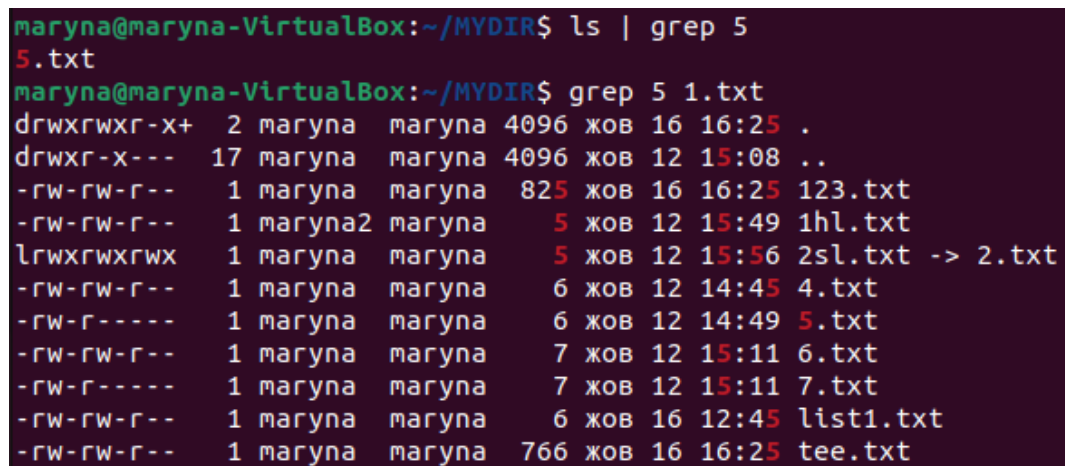
Опції команди grep:

- e Розширений синтаксис (Extended)
- f Зіставлення з фіксованими рядками (Fixed)
- i Відключення чутливості до регістру
- l Список файлів, вміст яких збігається з виразом
- n Виводить номери рядків, в яких знайдено збіг
- r Рекурсивний пошук
- v Інверсія результатів пошуку

Приклади роботи з grep

Якщо потрібно знайти конкретну папку або один файл серед сотні інших, то можна передати вивід команди `ls` в `grep` через вертикальну риску (`|`), а вже `grep` у параметром передати потрібне слово (рисунок 4.28).

\$ ls | grep Documents



```
maryna@maryna-VirtualBox:~/MYDIR$ ls | grep 5
5.txt
maryna@maryna-VirtualBox:~/MYDIR$ grep 5 1.txt
drwxrwxr-x+  2 maryna  maryna 4096 жов 16 16:25 .
drwxr-x--- 17 maryna  maryna 4096 жов 12 15:08 ..
-rw-rw-r--  1 maryna  maryna  825 жов 16 16:25 123.txt
-rw-rw-r--  1 maryna2  maryna   5 жов 12 15:49 1hl.txt
lrwxrwxrwx  1 maryna  maryna   5 жов 12 15:56 2sl.txt -> 2.txt
-rw-rw-r--  1 maryna  maryna   6 жов 12 14:45 4.txt
-rw-r----- 1 maryna  maryna   6 жов 12 14:49 5.txt
-rw-rw-r--  1 maryna  maryna   7 жов 12 15:11 6.txt
-rw-r----- 1 maryna  maryna   7 жов 12 15:11 7.txt
-rw-rw-r--  1 maryna  maryna   6 жов 16 12:45 list1.txt
-rw-rw-r--  1 maryna  maryna  766 жов 16 16:25 tee.txt
```

Рисунок 4.28 –Команда grep

Якщо ж потрібно знайти не одне слово, а словосполучення або ціле речення, то параметр команди `grep` має бути виділено лапками. `Grep` підтримує як одинарні, так і подвійні лапки.

\$ ls | grep 'My Documents'

Незважаючи на те, що команда `grep` частіше використовується як свого роду фільтр для інших команд, але її також можна використовувати окремо.

\$ `grep 'Class 1' Students.txt`

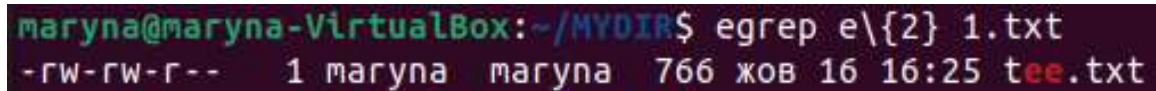
Команді `grep` можна передавати не один параметр, а кілька. Для цього перед кожним аргументом пишеться ключ `e`. Цю команду система розуміє, як "або-або" і виводить всі входження зазначених слів, лапками виділений тільки рядок, який містить пробіл.

\$ `grep -e 'Class 1' -e Lesson Students.txt`

Наступні дві команди еквівалентні і виводять всі рядки, в яких є дві поспіль літери "р" (рисунок 4.29).

\$ `egrep p\{2} fruits.txt`

\$ `grep -E p\{2} fruits.txt`



```
maryna@maryna-VirtualBox:~/MYDIR$ egrep e\{2} 1.txt
-gw-gw-g-- 1 maryna maryna 766 жов 16 16:25 tee.txt
```

Рисунок 4.29 –Команда `egrep`

Щоб вести пошук за вказаним шаблоном серед всіх файлів у всіх папках і підпапках, команду `grep` потрібно запустити з ключем `r`. Команда виведе всі файли, де знайдено збіг із зазначеним шаблоном, а також шлях до них. За замовчуванням пошук ведеться в поточній директорії і піддиректоріях.

\$ `grep -r pattern /directory/to/search`

Якщо в шаблоні пошуку міститься пробіл, то потрібно виділяти рядок лапками. Це можна використовувати для пошуку пробілів і знаків табуляції в файлі.

\$ `grep " " sample.txt`

Але якщо комбінувати клавіші `ctrl+v`, а потім натиснути `TAB`, то система сприйме це як знак табуляції.

\$ grep " " sample.txt

[] використовуються щоб перевірити на відповідність одному із зазначених символів (рисунок 4.30).

\$ grep "Class [123]" Students.txt

[-] знак дефіса означає діапазон значень. Це можуть бути як букви, так і цифри.

\$ grep "Class [1-3]" Students.txt

Друга команда вивела той же, що і перша, але тут обійшлися знаком діапазону.

^ Каретка використовується для пошуку рядків, які починаються з вказаного шаблону. Команда нижче виведе всі рядки, які починаються з літери "A" (рисунок 4.30).

\$ grep "^A" Students.txt

[^] Між квадратними дужками зміст каретки змінюється. Тут він виключає з пошуку наступні за нею символи або діапазон символів. (рисунок 4.30)

\$ grep "Class [^1-2]" Students.txt

\$ Знак долара означає кінець рядка. Команда виведе тільки ті рядки, в кінці яких зустрічає вказаний шаблон (рисунок 4.30).

\$ grep "1\$" Students.txt

. крапка позначає один будь-який символ. Щоб вказати кілька будь-яких символів, можна написати символ крапка потрібну кількість разів (рисунок 4.30).

\$ grep "A.a" Students.txt

Існує дуже багато різних варіантів використання умовного оператора АБО (OR) з командою grep.

\$ grep -E 'string1 | string2' filename

\$ egrep 'string1 | string2' filename

```

maryna@maryna-VirtualBox:~/MYDIR$ cat mygrep.txt
qwerty
123
q123
maryna@maryna-VirtualBox:~/MYDIR$ grep "q[w]" mygrep.txt
qwerty
maryna@maryna-VirtualBox:~/MYDIR$ grep "q[1]" mygrep.txt
q123
maryna@maryna-VirtualBox:~/MYDIR$ grep "q[^1]" mygrep.txt
qwerty
maryna@maryna-VirtualBox:~/MYDIR$ grep "y$" mygrep.txt
qwerty
maryna@maryna-VirtualBox:~/MYDIR$ grep "q..." mygrep.txt
qwerty

```

Рисунок 4.30 –Команда grep

Ігнорування регістру вирішується шляхом додавання ключа **i** до команди пошуку.

\$ grep -i string filename

Шукати точне співпадіння

\$ grep -w "world" fruits.txt

Знайти не тільки точний збіг по рядках, але і дізнатися інформацію: яка йде після них

\$ grep -A3 "word" fruits.txt

яка йде до них

\$ grep -B2 "word" fruits.txt

навколо ключового слова

\$ grep -c2 "word" fruits.txt

Щоб переглянути вміст файлу з виключенням конкретного рядка до запиту потрібно додати ключ **v** (рисунок 4.31).

\$ grep -v string-to-exclude filename

Щоб показати номер рядка, де був знайдений шуканий рядок, команду grep потрібно використувати ключ **n** (рисунок 4.31).

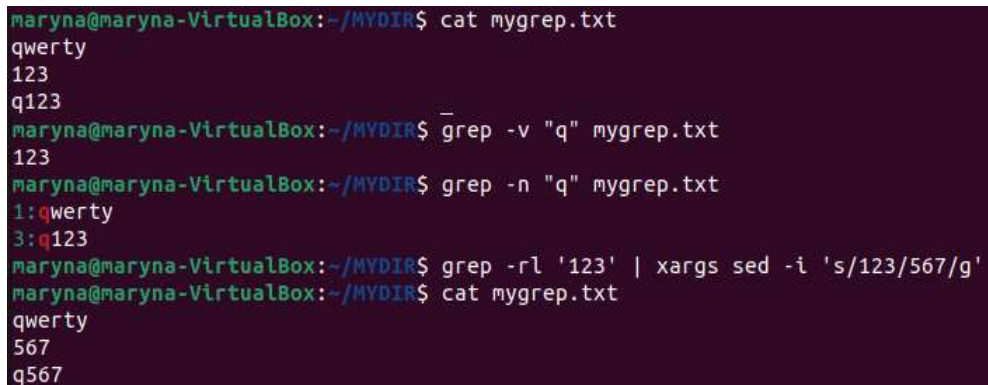
\$ grep -n string filename

Щоб відсортувати вивід команди `grep` в будь-якому порядку, за нею потрібно прописати команду `sort` через вертикальну риску (`|`). За замовчуванням `sort` сортує результат за алфавітом.

\$ `grep text 1.txt | sort`

Для пошуку і заміни одного тексту іншим командою `grep`, потрібно запустити її, а вивід передати команді `sed`. Команда нижче замінить всі входження рядка "string1", на "string2" у всіх файлах в поточній директорії (рисунок 4.31).

\$ `grep -rl 'string1' ./ | xargs sed -i 's / string1 / string2 / g'`



```
maryna@maryna-VirtualBox:~/MYDIR$ cat mygrep.txt
qwerty
123
q123
maryna@maryna-VirtualBox:~/MYDIR$ grep -v "q" mygrep.txt
123
maryna@maryna-VirtualBox:~/MYDIR$ grep -n "q" mygrep.txt
1:qwerty
3:q123
maryna@maryna-VirtualBox:~/MYDIR$ grep -rl '123' | xargs sed -i 's/123/567/g'
maryna@maryna-VirtualBox:~/MYDIR$ cat mygrep.txt
qwerty
567
q567
```

Рисунок 4.31 –Команда `grep`

Команда **`sed`** – це редактор потоків і може виконувати багато функцій у файлі: пошук, пошук та заміна, вставка або видалення. Найчастіше команда `sed` в Linux використовується для пошуку та заміни.

За допомогою `sed` можна редагувати файли, навіть не відкриваючи їх, що є набагато швидшим способом знайти та замінити щось у файлі.

Команда `sed` підтримує регулярні вирази, що дозволяє задавати складні шаблони.

`sed [опції] -e [команди] [файл]`

Опції команди `sed`:

-p - опція `print` виводить по рядкам вміст файлу;

-n - пригнічує автоматичний вивід кожного рядка;

-nr - вивід N рядка. Наприклад `sed -n '1p'` виведе тільки перший рядок.

Якщо в команді вказано діапазон, то `sed` виконує всі задачі тільки на рядках, які входять в цей діапазон

-i - зробити резервну копію файла;

-e - додавання команди в сценарій до набору виконуваних команд при обробці вводу;

Кожній команді можна передати адресу, яка вказуватиме на рядки, для яких вона буде виконана.

Адреси sed:

номер - дозволяє вказати номер рядка, у якому треба виконувати команду;

\$ - останній рядок у файлі;

/регулярний вираз/ - будь-який рядок, який підходить за регулярним виразом. Модифікатор **l** (л) вказує, що регулярний вираз має бути не чутливим до регістру;

номер, номер - починаючи з рядка з першої частини та закінчуючи рядком з другої частини;

номер, /регулярний_ввираз/ - починаючи з рядка першої частини до рядка, який буде відповідати регулярному виразу;

номер, +кількість - починаючи від номера рядка зазначеного в першій частині та плюс кількість рядків після нього;

Якщо для команди не було задано адресу, то вона буде виконана для всіх рядків. Якщо передано одну адресу, команда буде виконана тільки для рядка за цією адресою. Можна також передати діапазон адрес. Тоді адреси розділяються комою та команда буде виконана для всіх адрес діапазону.

Команди редагування sed:

-s заміна тексту на основі шаблону з використанням регулярних виразів

-g заміна всіх запропонованих варіантів. За замовчуванням замінюється лише перший знайдений збіг

-i виконує пошук без урахування регістру

s/що_замінювати/на_що_замінювати/опції - заміна символів, підтримуються регулярні вирази;

y/символи/символи – дозволяє замінити символи з першої частини на відповідні символи з другої частини (посимвольно).

Команда **sed** замінює слово «**old_word**» на «**new_word**» у файлі

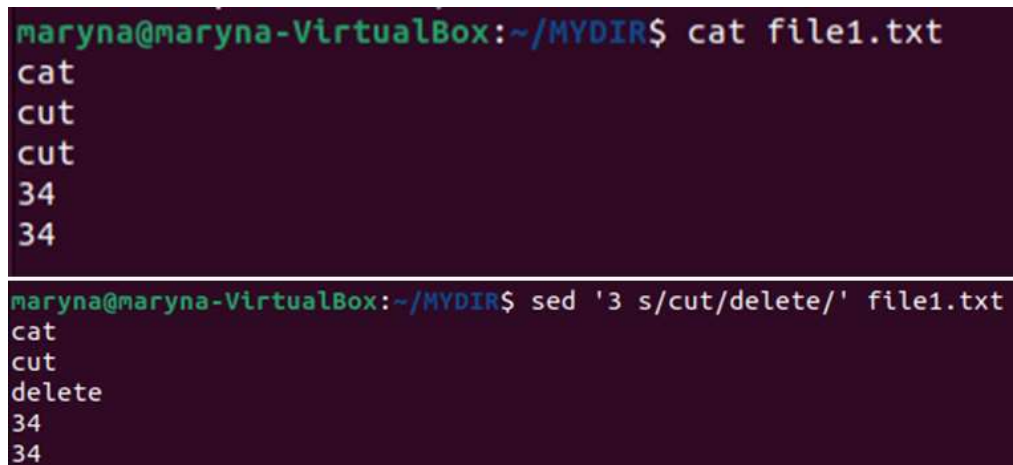
's/old_word/new_word/gi'

-s заміна тексту на підставі шаблону за допомогою регулярних виразів

-g заміна всіх запропонованих варіантів. За замовчуванням замінюється лише перший знайдений збіг

-i робить пошук без урахування регістру

sed 's/unix/linux/' 1.txt // команда **sed** замінює слово «unix» на «linux» у файлі (рисунок 4.32)



```
maryna@maryna-VirtualBox:~/MYDIR$ cat file1.txt
cat
cut
cut
34
34

maryna@maryna-VirtualBox:~/MYDIR$ sed '3 s/cut/delete/' file1.txt
cat
cut
delete
34
34
```

Рисунок 4.31 –Команда sed

Потрібно використовувати прапорці **/1**, **/2**, щоб замінити перше, друге вхідження шаблону в рядку.

sed 's/unix/linux/2' 1.txt // команда замінює друге входження слова «unix» на «linux» у рядку

Команда sed замінює рядки діапазоном від 1 до 5

sed '1,5 s/unix/linux/' 1.txt // замінює рядки діапазоном від 1 до 5

sed '3,\$ s/unix/linux/' 1.txt // sed замінює текст із третього рядка на останній у файлі (тут \$ вказує на останній рядок у файлі)

sed '\$d' 1.txt // видаляє останній рядок (-d команда)

sed '2a\новий рядок' 1.txt // команда sed додає текст після 2-го рядка (-a команда)

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №5 Управління правами доступу до файлів і каталогів Linux

Мета: навчитись, за допомогою команд, створювати користувачів, групи, встановлювати, змінювати права доступу до файлів і каталогів для користувачів, груп користувачів, оволодіти механізмами, які дозволяють захистити файли і каталоги ОС UBUNTU від несанкціонованого доступу.

Завдання

1.

1) Створіть у вашому HOME-каталозі один текстовий файл. Виведіть на екран вміст каталогу.

2) Створіть користувача, групу. Додайте власника файлу та нового користувача в групу.

3) Проаналізуйте і поясніть, які права доступу до цього файлу має власник файлу, його група і інші користувачі.

4) Проаналізуйте права доступу до вашого головного каталогу. Чи є обмеження на роботу з файлами в цьому каталозі?

5) Виведіть на екран вміст файлу. Поясніть, чому операція виконалася успішно.

6) Забороніть права на читання файлу власнику і групі, спробуйте вивести на екран текст файлу. Поясніть, чому операція не виконується.

7) Видаліть права на запис в файл. Спробуйте додати до файлу текст і видалити його. Поясніть результат.

8) Видаліть право на модифікацію каталогу. Повторіть операцію видалення. Поясніть результат.

9) Створіть підкаталог. Додайте в ньому текстовий файл. Проаналізуйте права доступу до підкаталогу і поясніть можливості по використанню підкаталогу.

10) Видаліть право власника на «виконання» підкаталогу.

11) Спробуйте зробити підкаталог поточним. Поясніть результат.

12) Перегляньте вміст підкаталогу. Поясніть результат.

2.

Ознайомитись з бітами SUID, SGID і sticky bit. Навчитись ними користуватись.

Теоретичні відомості

В операційній системі Linux існують три типи користувачів:

- користувач root;
- системні (фіктивні) користувачі;
- звичайні користувачі.

Кожному користувачу окрім імені присвоюється числовий ідентифікатор користувача (UID, User IDentificator). UID - це номер, призначений кожному користувачеві Linux, це уявлення користувача в ядрі Linux.

UID використовується для ідентифікації користувача в системі і для визначення того, до яких системних ресурсів користувач може отримати доступ. Ось чому ідентифікатор користувача повинен бути унікальним.

Інформація про користувачів зберігається в файлах:

/etc/passwd користувачі (рисунок 5.2)

/etc/group групи користувачів (рисунок 5.3)

/etc/shadow зашифровані паролі «тіньові паролі». Зроблено це для безпеки, так як файл **/etc/passwd** може читатися ким завгодно, а файл **/etc/shadow** може прочитати тільки root (рисунок 5.4)

/etc/gshadow зашифровані паролі «тіньові паролі» груп (рисунок 5.6)

Файл **/etc/passwd**, який можна використовувати, щоб відкрити список користувачів в системі Linux та знайти **UID** і **GID**.

Опис файлу **/etc/passwd**

Зазвичай перший рядок описує користувача **root**, за яким ідуть системні та звичайні облікові записи користувачів. Нові записи додаються в кінець файлу.

Кожен рядок файлу **/etc/passwd** містить сім полів, розділених двокрапками (рисунок 5.1):

[Username]:[Password]:[UID]:[GID]:[GECOS]:[Home directory]:[shell]

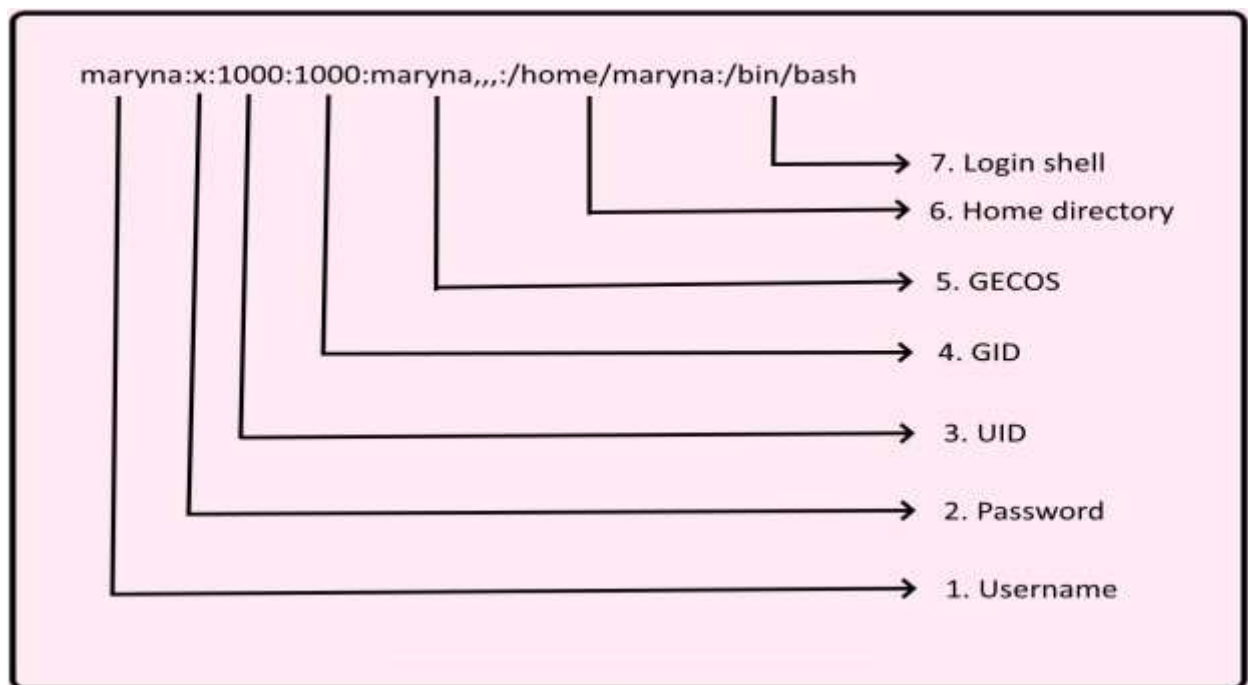


Рисунок 5.1 – Структура файлу **/etc/passwd**

[Username] Рядок, який ви вводите при вході в систему. Кожне ім'я користувача має бути унікальним рядком на комп'ютері. Максимальна довжина імені користувача обмежена 32 символами.

[Password] У більшості систем це поле має значення **x** і пароль користувача зберігається у файлі **/etc/shadow**. У старих системах Linux зашифрований пароль користувача зберігався у файлі **/etc/passwd**.

[UID] ідентифікатор користувача – це номер, призначений для кожного користувача. Він використовується операційною системою для звернення до користувача.

[GID] Номер ідентифікатора групи користувача, який належить до основної групи користувача. Коли користувач створює файл, група файлу встановлюється на цю групу. Як правило, ім'я групи збігається з ім'ям користувача. Інші групи перелічені у файлі **/etc/groups**.

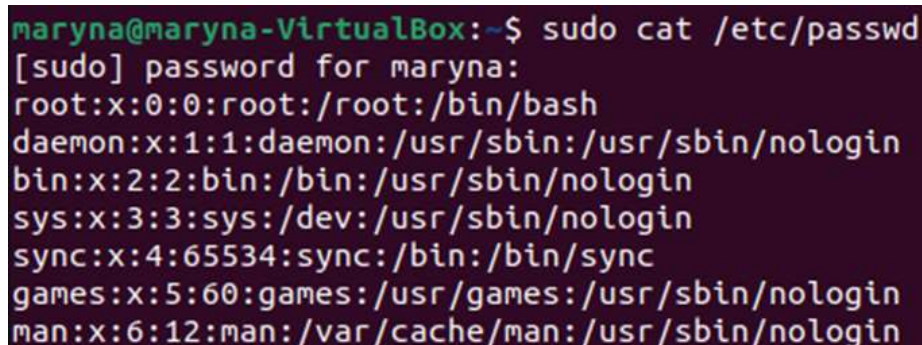
[GECOS] або повне ім'я користувача. Це поле містить список значень через кому з наступною інформацією:

- повне ім'я користувача;
- робочий номер телефону;
- домашній номер телефону;
- інша контактна інформація.

Це поле є необов'язковим, тому його можна залишити пустим.

[Home director] Абсолютний шлях до домашнього каталогу користувача. Він містить файли користувача та конфігурації. За замовчуванням домашні каталоги користувачів називаються за іменем користувача і створюються в каталозі **/home**.

[shell] Абсолютний шлях до оболонки користувача. Це оболонка, яка запускається, коли користувач входить в систему. У більшості дистрибутивів Linux оболонкою входу за замовчуванням є **Bash**.



```
maryna@maryna-VirtualBox:~$ sudo cat /etc/passwd
[sudo] password for maryna:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

Рисунок 5.2 – Файл **/etc/passwd**

Опис файлу **/etc/group**:

[Group name]:[Group password]:[GID]:[Group members]

[Group name] - це ім'я групи.

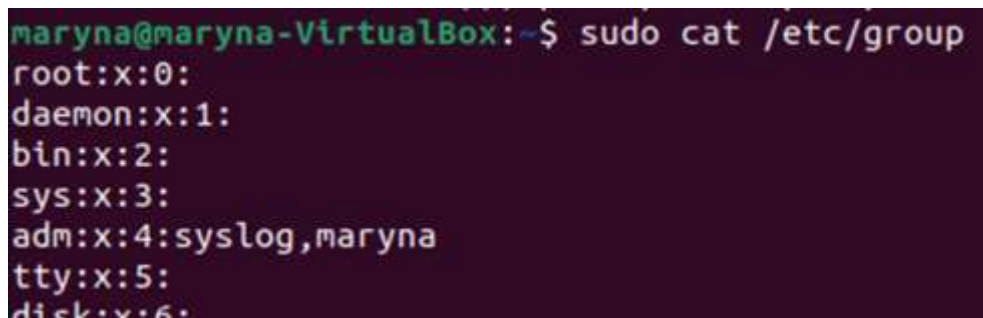
X в **[Group password]** вказує, що групові паролі не використовуються.

[GID] той же, що і в **/etc/passwd**.

[Group members]: список користувачів, які є членами **[Group name]**.

Всі елементи запису розділені двокрапкою і пишуться без пробілів.

Список користувачів групи розділяється комою.



```
maryna@maryna-VirtualBox:~$ sudo cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,maryna
tty:x:5:
disk:x:6:
```

Рисунок 5.3 – Файл **/etc/passwd**

Опис файлу **/etc/shadow**:

[username]:[password]:[lastchg]:[min]:[max]:[warn]:[inactive]:[expire]:

[username] - ім'я користувача

[password] - пароль в зашифрованому вигляді

Пароль повинен складатися щонайменше з 8–12 символів, включаючи спеціальні символи, цифри, літери нижнього регістру тощо.

Порожній запис :: показує, що для входу пароль не потрібен (погана ідея),

запис :*: або :!: показує, що вхід заблокований за паролем, але перемикання на користувача доступне

```
maryna@maryna-VirtualBox:~$ sudo cat /etc/shadow
[sudo] password for maryna:
root:!:19253:0:99999:7:::
daemon:!:19213:0:99999:7:::
bin:!:19213:0:99999:7:::
sys:!:19213:0:99999:7:::
sync:!:19213:0:99999:7:::
games:!:19213:0:99999:7:::
man:!:19213:0:99999:7:::
lp:!:19213:0:99999:7:::
```

Рисунок 5.4 – Файл `/etc/shadow`

Пароль використовує формат `$type$salt$hashed` (рисунок 5.5).

`$type` є методом криптографічного алгоритму хешування і може мати наступні значення:

`$1$` - MD5

`$2a$` - Blowfish

`$2y$` - Eksblowfish

`$y$` - yescrypt

`$5$` - SHA-256

`$6$` - SHA-512

```
maryna:$y$j9T$mlPL7YEAPgFRCfPOBzILV0$WtWbLm4oBPDLflwkeBf3y.KyXDKjzH23EaHschxSzx2
:19253:0:99999:7:::
maryna2:!:19271:0:99999:7:::
maryna1:!:19271:0:99999:7:::
```

Рисунок 5.5 – Зашифрований пароль у файлі `/etc/shadow`

Для вирішення проблеми використання готових хешів зломисниками, розробники ОС Linux почали використовувати сіль (salt), тобто набір із кількох додаткових байт, які додавалися до паролю перед хешуванням.

Таким чином, використання солі (salt) для шифрування паролів ускладнює перебір, оскільки не дозволяє використовувати готові таблиці.

`[lastchg]` - час, що минув з останньої зміни пароля (кількість днів з 01.01.1970)

[min] - мінімальний термін служби пароля; якщо більше нуля, то пароль може бути змінений протягом цієї кількості днів

[max] - максимальний термін служби пароля; зазвичай встановлюється рівним 99999 днів, що відповідає 275 рокам

[warn] - попередження про термін служби; користувачі попереджаються за вказану кількість днів до закінчення максимального терміну служби пароля

[inactive] - час переходу в неактивний стан; аккаунт вважається неактивним через задану кількість днів після закінчення терміну служби пароля

[expire] - час через який аккаунт повинен бути відключений, вказується в кількості днів, що минули з 01.01.1970

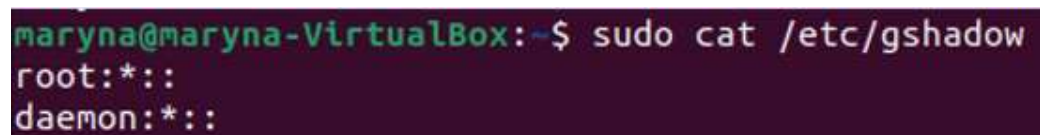
9 поле - невикористане. Це поле ігнорується. Воно зарезервоване для майбутнього використання.

Опис файлу /etc/gshadow:

[groupname]:[encrypted-password]:[group-admins]:[group-users]

[groupname] - ім'я групи;

[encrypted-password] - зашифрований пароль групи. Якщо заданий, то користувачі, які не є членами групи, можуть приєднатися до групи ввівши пароль цієї групи використовуючи команду **newgrp**.



```
maryna@maryna-VirtualBox:~$ sudo cat /etc/gshadow
root:*::
daemon:*::
```

Рисунок 5.6 – Файл /etc/gshadow

Якщо значення цього поля один ! Або *, то користувачі не зможуть приєднатися до групи командою **newgrp**:

newgrp group_name

Значення **!!** означає те ж саме, що і **!**, а також те, що пароль ще не було встановлено.

Якщо значення порожньо - в групу можуть логінитися тільки члени групи;

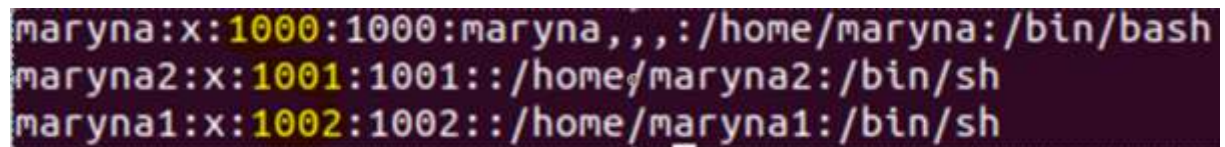
[group-admins] - члени групи, які перераховані в цьому полі, можуть додавати або видаляти членів групи використовуючи командою **gpasswd**:

gpasswd [опції] група

[group members] - члени групи, перераховані в цьому полі, є звичайними користувачами даної групи.

В більшості дистрибутивів Linux **UID 1-500** зазвичай зарезервовані для системних користувачів. В Ubuntu і Fedora **UID** для нових користувачів починаються з **1000**.

Наприклад, якщо ви використовуєте команду **useradd** або **adduser** для створення нового користувача, він отримає наступний доступний номер після 1000 в якості свого UID (рисунок 5.7).



```
margyna:x:1000:1000:margyna,,,:/home/margyna:/bin/bash
margyna2:x:1001:1001::/home/margyna2:/bin/sh
margyna1:x:1002:1002::/home/margyna1:/bin/sh
```

Рисунок 5.7 – Файл **/etc/passwd**

У Linux **UID - 0** і **GID - 0** зарезервовані для користувача **root**.

useradd - це низькорівнева утиліта для створення користувачів в Linux.

Команда **adduser** - це більш просте рішення для створення користувачів і є надбудовою над **useradd**, **groupadd** і **usermod**.

Створити нового користувача з використанням команди **adduser**:

sudo adduser User5

Після запуску цієї команди ви повинні ввести пароль для нового користувача. Потім буде запропоновано ввести додаткові відомості про користувача: ім'я, номер кімнати, телефони та коментар. Вводити цю

інформацію необов'язково. достатньо натиснути Enter, щоб пропустити введення даних (рисунок 5.8).

```
maryna@maryna-VirtualBox:~$ sudo adduser users5
[sudo] password for maryna:
Adding user `users5' ...
Adding new group `users5' (1003) ...
Adding new user `users5' (1001) with group `users5' ...
Creating home directory `/home/users5' ...
Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
Changing the user information for users5
Enter the new value, or press ENTER for the default
    Full Name []: User
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
```

Рисунок 5.8 – Створення користувача

У результаті виконання команди **adduser** будуть виконані наступні дії:

- створюється новий користувач з ім'ям, яке ви вказали при виконанні команди;
- створюється група з тим же ім'ям;
- створюється домашній каталог користувача в директорії /home/ім'я користувача;
- в домашній каталог копіюються файли з директорії /etc/skel. В цій директорії зберігаються файли, які копіюються у домашні каталоги усіх нових користувачів.

Команда **useradd** приймає в якості аргументу ім'я користувача, а також різні параметри.

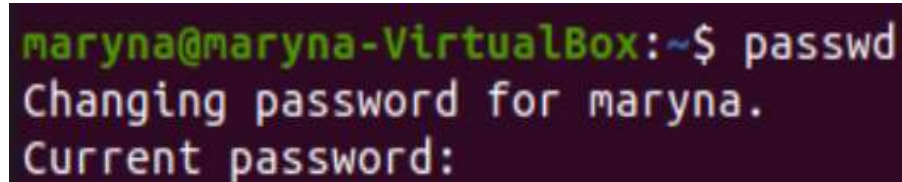
useradd username – створення нового користувача

sudo useradd user3

Ця команда створює нового користувача із системними параметрами за замовчуванням, які прописані у файлі /etc/default/useradd.

Щоб користувач міг увійти в систему, необхідно задати для нього пароль. Для цього використовуємо команду **passwd** (рисунок 5.9):

sudo passwd user3



```
maryna@maryna-VirtualBox:~$ passwd
Changing password for maryna.
Current password:
```

Рисунок 5.9 – Створення паролю користувача

Потім потрібно створити домашню директорію користувача

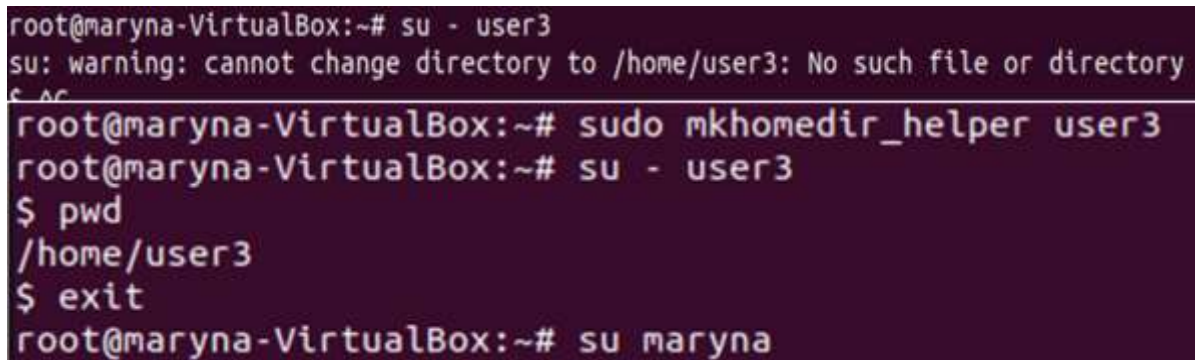
sudo useradd -m user3

Домашня директорія створюється за замовчуванням у каталозі /home. Ім'я директорії співпадає з іменем користувача.

Також створити директорію можна за допомогою команди **mkhomedir_helper** (рисунок 5.10):

sudo mkhomedir_helper user3

Якщо не створити домашню директорію користувача і спробувати змінити користувача, буде повідомлення, що такої директорії не існує



```
root@maryna-VirtualBox:~# su - user3
su: warning: cannot change directory to /home/user3: No such file or directory
root@maryna-VirtualBox:~# sudo mkhomedir_helper user3
root@maryna-VirtualBox:~# su - user3
$ pwd
/home/user3
$ exit
root@maryna-VirtualBox:~# su maryna
```

Рисунок 5.10 – Створення домашньої директорії користувача

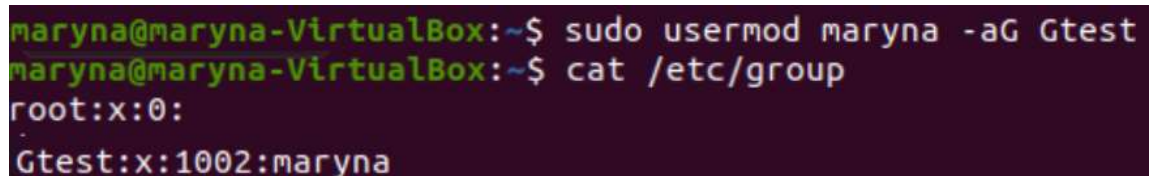
Ключі команди **useradd (adduser):**

- s** додати коментарі в файл `/etc/passwd` до поля даного користувача
- d** вказати домашній каталог, замість наданого за замовчуванням
- e** дата закінчення терміну дії облікового запису
- f** задати кількість днів, що пройдуть після закінчення терміну дії пароля, коли обліковий запис повинен бути відключений
- g** вказати ідентифікатор групи (GID) для користувача, замість використовуваного за замовчуванням
- G** вказати додаткові групи за іменами
- m** створити домашній каталог для користувача, якщо він ще не існує
- s** вказати оболонку для користувача
- u** вказати UID, замість використовуваного за замовчуванням.

Щоб змінити параметри облікового запису користувача, використовується команда **usermod**.

Ключі команди **usermod:**

- aG** ім'я групи; додати користувача в зазначену групу або список груп (рисунок 5.11)
- l** (л) задати користувачу нове ім'я
- L** заблокувати обліковий запис
- U** розблокувати обліковий запис
- u** змінити UID

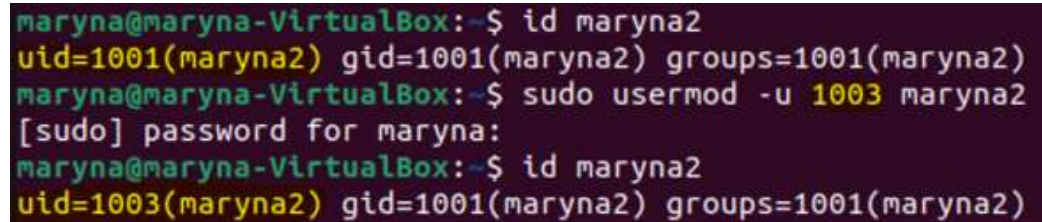


```
maryna@maryna-VirtualBox:~$ sudo usermod maryna -aG Gtest
maryna@maryna-VirtualBox:~$ cat /etc/group
root:x:0:
Gtest:x:1002:maryna
```

Рисунок 5.11 – Додавання користувача в групу

Можна змінити UID, змінивши користувача за допомогою команди **usermod** наступним чином (для цього потрібно мати привілеї суперкористувача) (рисунок 5.12):

usermod -u 1004 user_2



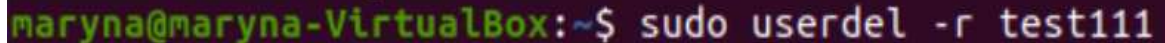
```
maryna@maryna-VirtualBox:~$ id maryna2
uid=1001(maryna2) gid=1001(maryna2) groups=1001(maryna2)
maryna@maryna-VirtualBox:~$ sudo usermod -u 1003 maryna2
[sudo] password for maryna:
maryna@maryna-VirtualBox:~$ id maryna2
uid=1003(maryna2) gid=1001(maryna2) groups=1001(maryna2)
```

Рисунок 5.12 – Зміна ідентифікатора користувача

Команда **userdel** використовується для видалення облікового запису користувача

Ключі команди **userdel**:

-r видаляє користувача і його домашній каталог



```
maryna@maryna-VirtualBox:~$ sudo userdel -r test111
```

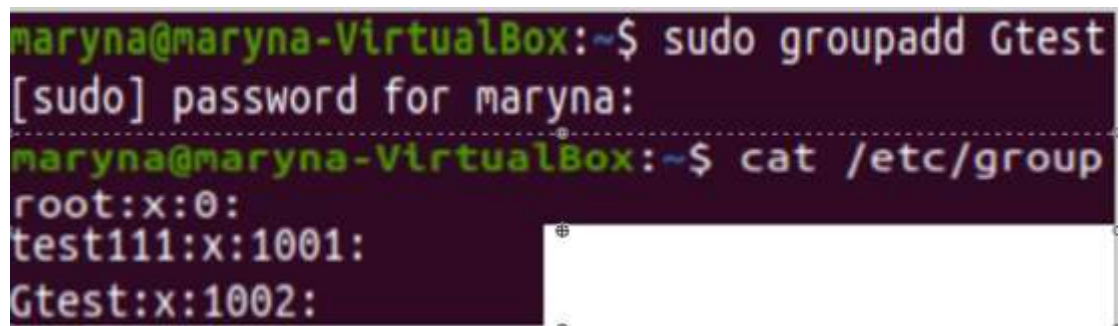
Рисунок 5.13 – Видалення користувача і його домашнього каталогу

Для управління групами використовуються команди:

addgroup (groupadd) - створити нову групу (рисунок 5.14, 5.15).

groupadd - не виводить нічого при створенні групи

addgroup - виводить gid при створенні групи.



```
maryna@maryna-VirtualBox:~$ sudo groupadd Gtest
[sudo] password for maryna:
maryna@maryna-VirtualBox:~$ cat /etc/group
root:x:0:
test111:x:1001:
Gtest:x:1002:
```

Рисунок 5.14 – Створення групи

```

user5@maryna-VirtualBox:~$ groupadd group1
groupadd: Permission denied.
groupadd: cannot lock /etc/group; try again later.
user5@maryna-VirtualBox:~$ sudo groupadd group1
[sudo] password for user5:
user5 is not in the sudoers file. This incident will be reported.

maryna@maryna-VirtualBox:~$ sudo adduser user5 sudo
Adding user `user5' to group `sudo' ...
Adding user user5 to group sudo
Done.
maryna@maryna-VirtualBox:~$ su - user5
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

user5@maryna-VirtualBox:~$ sudo su - user5
[sudo] password for user5:
user5@maryna-VirtualBox:~$ sudo addgroup group3
[sudo] password for user5:
Adding group `group3' (GID 1009) ...
Done.

```

Рисунок 5.15 – Створення групи під різними користувачами

Команда **Groupdel (delgroup)** видаляє групу (рисунок 5.16).

groupdel - не виводить нічого при видаленні групи

delgroup - виводить повідомлення про видалення групи.

```

maryna@maryna-VirtualBox:~$ sudo groupdel Gtest
maryna@maryna-VirtualBox:~$ cat /etc/group

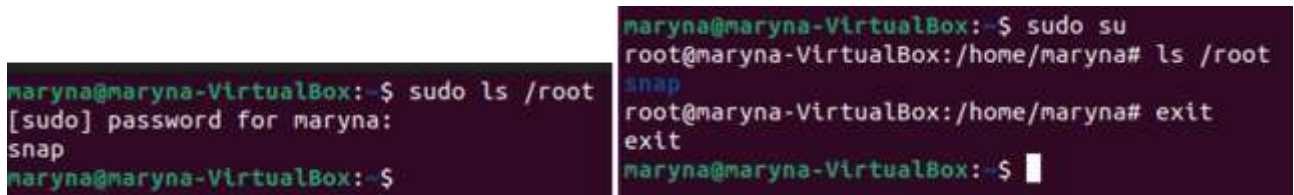
```

Рисунок 5.16 – Видалення групи

Суперкористувач або **root** - це спеціальний обліковий запис і група користувачів з ідентифікатором, власник якого має право на виконання всіх операцій.

Команди **su** і **sudo** - це два різні способи отримати привілеї суперкористувача (root), що володіє максимальними правами (рисунок 5.17).

Основна відмінність між **su** і **sudo** полягає в тому, що **su** вимагає пароль цільового облікового запису (наприклад, користувача root) і перемикає вас на нього, в той час як **sudo** вимагає пароль поточного користувача і запускає від його імені лише одну (або кілька) команд, для виконання яких потрібні права користувача root.



```
maryna@maryna-VirtualBox:~$ sudo ls /root
[sudo] password for maryna:
snap
maryna@maryna-VirtualBox:~$

maryna@maryna-VirtualBox:~$ sudo su
root@maryna-VirtualBox:/home/maryna# ls /root
snap
root@maryna-VirtualBox:/home/maryna# exit
exit
maryna@maryna-VirtualBox:~$
```

Рисунок 5.17 – Команди su, sudo

Команда зміни користувача **su - username** (з пробілами), натиснути клавішу Enter.

Залогінітись як root без зміни середовища **su**, зі зміною середовища (додати тире) **su -**.

Управління правами доступу до файлів і каталогів

Для доступу до файлів у Linux використовуються дозволи.

У Linux кожному файлу відповідає набір прав доступу, представлений у вигляді 9-ти біт режиму.

Режим визначає, які користувачі мають право читати файл, записувати в нього дані або виконувати його.

Разом з іншими трьома бітами, що впливають на запуск виконуваних файлів, цей набір утворює код режиму доступу до файлу.

12 біт режиму зберігаються в 16-бітовому полі індексного дескриптора разом з 4-ма додатковими бітами, що визначають тип файлу.

Останні 4 біта встановлюються при створенні файлів і не підлягають зміні.

Біти режиму (далі права) можуть змінюватися або власником файлу, або суперкористувачем за допомогою команди **chmod**.

У Linux у кожного файлу та кожного каталогу є два власники: користувач та група.

Ці власники встановлюються під час створення файлу чи каталогу.

Користувач, який створює файл, стає власником цього файлу, а первинна група, в яку входить цей же користувач, так само стає власником цього файлу. Щоб визначити, чи є у вас як у користувача права доступу до файлу чи каталогу, оболонка перевіряє володіння ними.

Це відбувається в наступному порядку:

1) оболонка перевіряє, чи ви є власником файлу, до якого ви хочете отримати доступ. Якщо ви є цим власником, ви отримуєте дозволи та оболонка припиняє перевірку.

2) якщо ви не є власником файлу, оболонка перевірить, чи є учасником групи, яка має дозвіл на цей файл. Якщо ви є учасником цієї групи, ви отримуєте доступ до файлу з дозволами, які встановлені для групи, та оболонка припиняє перевірку.

3) якщо ви не є ні користувачем, ні власником групи, ви отримуєте права інших користувачів (Other).

Існує три шляхи управління доступом до файлу або каталогу, кожен файл повинен мати власника (**owner**), групового власника (**group owner**), а також може знадобитися доступ для всіх інших користувачів (**other** - включає всі облікові записи, які не є власником файлу і не входять в групу-власника).

Щоб побачити поточні права власника, ви можете скористатися командою **ls -l** (рисунок 5.18). Ця команда показує користувача та групу-власника.

```

maryna@maryna-VirtualBox:~$ ls -l
total 40
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Desktop
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Documents
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Downloads
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Music
drwxrwxr-x 2 maryna maryna 4096 жов 6 16:34 MYDIR
drwxr-xr-x 2 maryna maryna 4096 вер 18 14:26 Pictures

```

Рисунок 5.18 – Команда ls -l

Реалізація управління доступом до файлів і каталогів в Linux дозволяє або забороняє доступ за трьома прапорами:

- прапор читання (**Read**)
- прапор запису (**Write**)
- прапор виконання (**eXecute**).

Ефект цих дозволів відрізняється при застосуванні до файлів чи каталогів.

Для призначення прав використовуються три групи прапорів:

- перша визначає права для власника
- друга - права для основної групи користувача
- третя - для всіх інших користувачів в системі

Вони представляються наступним чином (рисунок 5.19):

flag	user	group	other
#	rwX	rwX	rwX

Рисунок 5.19 – Прапори для трьох груп

Прапор типу (flag) може бути одним з наступних:

- звичайний файл (відсутність прапора);

d каталог (directory);

l символічне посилання (symbolic link);

- b** блоковий пристрій (block device);
- c** символьний пристрій (character device);
- p** іменованний канал, пристрій fifo (fifo device);
- s** сокет (domain socket).

Розглянемо відмінності при застосуванні прав доступу до файлів та каталогів (рисунок 5.20):

Для файлів:

- r** - право на читання з файлу;
- w** - дозволяє запис в файл (зокрема перезапис або зміну);
- x** - дозволяє виконати файл.

Для каталогів, прапори **r w x** мають дещо відмінний зміст:

- r** - дозволяє читати тільки імена файлів в каталозі;
- x** - дозволяє мати доступ до самих файлів і їх атрибутів (але не імен);
- w** має сенс тільки в поєднанні з **x**, і дозволяє (в доповнення до **x**) маніпулювати файлами в каталозі (створювати, видаляти і перейменовувати);
- w** без **x** - не має ніякого ефекту.

Символьне представлення	Вісімкове значення	Двійкове значення	Права на файл	Права на каталог
rwX	7	111	повні права	повні права
rw-	6	110	права на читання і запис	тільки читання імен файлів
r-X	5	101	права на читання і виконання	читання імен файлів і доступ до файлів і їх атрибутів
r--	4	100	права на читання	тільки читання імен файлів
-wX	3	011	права на запис і виконання	все, окрім доступу до імен файлів
-w-	2	010	права на запис	відсутність прав
--X	1	001	права на виконання	доступ до файлів і їх атрибутів
---	0	000	відсутність прав	відсутність прав

Рисунок 5.20 – Значення прав доступу до файлів і каталогів

Є 2 варіанти призначення прав: вісімковий (777) та символний (rwxrwxrwx).

Звичайні права доступу можуть розширюватися додаванням прапорів: **SUID, SGID і sticky bit**.

SUID, SGID впливають на запуск файлу, а Sticky впливає на визначення власника об'єктів в каталозі.

При їх застосуванні необхідно використовувати не 3 цифри у вісімковому форматі, а 4, наприклад 0744. Багато хто прагне не використовувати спеціальні біти, нарікаючи на безпеку, але, в деяких ситуаціях без них не обійтися.

Приклад прав доступу для файлу:

-rwx----- 1 111 group1 34 2022-02-20 20:42 ubuntu.txt

Перший символ (прапор) порожній: «-» - для файлів.

Наступні три символи (rwx) позначають права для власника файлу, в даному випадку повні права для користувача 111.

Наступні три (---) - визначають права для групи group1, в нашому прикладі для всіх користувачів групи-власника group1 доступ заборонений.

Останні три символи (---) визначають права для всіх інших користувачів, в нашому випадку доступ заборонений.

Вісімкове позначення прав для файлу ubuntu.txt: 0700.

Приклад прав доступу для каталогу:

drwxr-xr-x 2 111 group1 4096 2021-02-20 20:30 dir

Для власника каталогу 111 - повні права (rwx).

Для групи group1 - права на лістинг каталогу і доступ до файлів (r-x).

Для користувача «всі інші» - права на лістинг каталогу і доступ до файлів (r-x).

Вісімкове значення в цьому прикладі: 0755.

Команда **chmod** підтримує установку прав як в вісімковому поданні, так і в символічному (маска режиму доступу) (рисунки 5.21).

chmod <опції> <права> <об'єкт або регулярний вираз>

Опції:

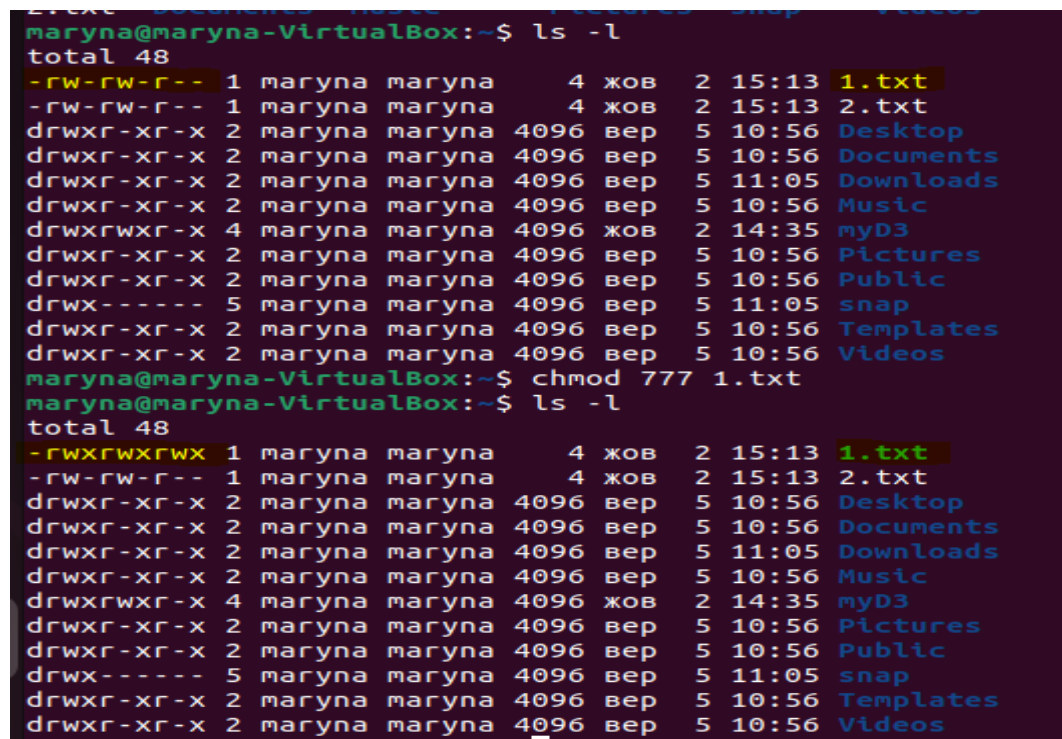
-R - рекурсивне призначення прав. Тобто призначити права всім об'єктам, керуючись регулярним виразом.

`chmod 744 1.txt` // встановить права для файлу 1.txt - (rwx r-- r--);

`chmod -R 775 11` // встановить права на каталог 11 і на всі об'єкти, що всередині цього каталогу, включаючи вміст підкаталогів (rwx rwx r-x);

`chmod -R 755 *` // встановить прав всім об'єктам поточного каталогу, включаючи підкаталоги.

`chmod -R 700 z*` // встановить повні права для власника і виключити права для групи-власника і всіх інших для всіх об'єктів, імена яких починаються з z, що знаходяться в поточному каталозі і його підкаталогах.



```
maryna@maryna-VirtualBox:~$ ls -l
total 48
-rw-rw-r-- 1 maryna maryna    4 жов  2 15:13 1.txt
-rw-rw-r-- 1 maryna maryna    4 жов  2 15:13 2.txt
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Desktop
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Documents
drwxr-xr-x 2 maryna maryna 4096 вер  5 11:05 Downloads
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Music
drwxrwxr-x 4 maryna maryna 4096 жов  2 14:35 myD3
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Pictures
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Public
drwx----- 5 maryna maryna 4096 вер  5 11:05 snap
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Templates
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Videos
maryna@maryna-VirtualBox:~$ chmod 777 1.txt
maryna@maryna-VirtualBox:~$ ls -l
total 48
-rwxrwxrwx 1 maryna maryna    4 жов  2 15:13 1.txt
-rw-rw-r-- 1 maryna maryna    4 жов  2 15:13 2.txt
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Desktop
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Documents
drwxr-xr-x 2 maryna maryna 4096 вер  5 11:05 Downloads
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Music
drwxrwxr-x 4 maryna maryna 4096 жов  2 14:35 myD3
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Pictures
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Public
drwx----- 5 maryna maryna 4096 вер  5 11:05 snap
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Templates
drwxr-xr-x 2 maryna maryna 4096 вер  5 10:56 Videos
```

Рисунок 5.21 – Перегляд та зміна прав доступу до файлу у вісімковому форматі

Інший спосіб призначення прав - це використання маски режиму доступу (символьне подання) (рисунок 5.22).

Окрім прав ми повинні вказати, кому ми збираємося ці права надати:

u - власнику об'єкта;

g - групі об'єкту;

o - користувачеві «всі інші»;

a - все перераховане вище.

Для призначення прав використовуються три знаки: мінус, плюс або дорівнює:

- ЗАБРАТИ зазначені права з об'єкта;

+ додати зазначені права до існуючих прав об'єкта;

= замінити права об'єкта на зазначені.

Приклад:

chmod g+w 1.txt // додасть користувачам групи файлу 1.txt права на запис в цей файл;

chmod a=rwx 2.doc // замінить існуючі права на файлі 2.doc на повні права всім;

chmod o-w 3.cgi // прибере право на запис для користувача «**other**».

chmod ug=rw 4.doc // виставить права на читання і запис файлу 4.doc для власника і групи-власника. Зверніть увагу, що якщо у користувача «**other**» були будь-які права, вони збережуться в незмінному вигляді.

chmod u+x,g+w-x 1.txt // додасть власнику файлу 1.txt права на його виконання, користувачам групи дозволити запис і заборонити виконання і залишити права інших користувачів без змін;

chmod u=rwx,g+w,go-x 2.doc // встановить повні права для власника файлу, дозволити користувачам групи запис і заборонити виконання всім користувачам, крім власника файлу.

```
maryna@maryna-VirtualBox:~$ ls -l
total 48
-rwxrwxrwx 1 maryna maryna 4 жов 2 15:13 1.txt
-rw-rw-r-- 1 maryna maryna 4 жов 2 15:13 2.txt
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Desktop
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Documents
drwxr-xr-x 2 maryna maryna 4096 вер 5 11:05 Downloads
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Music
drwxrwxr-x 4 maryna maryna 4096 жов 2 14:35 myD3
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Pictures
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Public
drwx----- 5 maryna maryna 4096 вер 5 11:05 snap
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Templates
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Videos
maryna@maryna-VirtualBox:~$ chmod ugo=rw 1.txt
maryna@maryna-VirtualBox:~$ ls -l
total 48
-rw-rw-rw- 1 maryna maryna 4 жов 2 15:13 1.txt
-rw-rw-r-- 1 maryna maryna 4 жов 2 15:13 2.txt
drwxr-xr-x 2 maryna maryna 4096 вер 5 10:56 Desktop
maryna@maryna-VirtualBox:~$ ls -l 1.txt
-rw-rw-rw- 1 maryna maryna 4 жов 2 15:13 1.txt
maryna@maryna-VirtualBox:~$ chmod u=rwx,g+x,o-w 1.txt
maryna@maryna-VirtualBox:~$ ls -l 1.txt
-rwxrwxr-- 1 maryna maryna 4 жов 2 15:13 1.txt
maryna@maryna-VirtualBox:~$
```

Рисунок 5.22 – Перегляд та зміна прав доступу до файлу у символічному форматі

Біти SUID, SGID і Sticky

Linux відслідковує не символічні імена власників і груп, а їх ідентифікатори (**UID** - для користувачів і **GID** для груп). Ці ідентифікатори зберігаються в файлах **/etc/passwd** і **/etc/group** відповідно.

Символьні еквіваленти ідентифікаторів використовуються тільки для зручності, наприклад, при використанні команди **ls**, ідентифікатори замінюються відповідними символічними позначеннями.

Що стосується процесів, то з ними пов'язано не два ідентифікатора, а 4-и: реальний і ефективний користувальницький (**UID**), а також реальний і ефективний груповий (**GID**).

Реальні номери застосовуються для обліку використання системних ресурсів, а ефективні для визначення прав доступу до процесів.

Як правило, реальні та ефективні ідентифікатори збігаються.

Власник процесу може посилати йому сигнали, а також змінювати пріоритет.

Процес не може явно змінити жодного зі своїх чотирьох ідентифікаторів, але є ситуації коли відбувається непряма установка нових ефективних ідентифікаторів процесу.

Для цього існують два спеціальних біта: **SUID** (Set User ID - біт зміни ідентифікатора користувача) і **SGID** (Set Group ID - біт зміни ідентифікатора групи).

Коли користувач або процес запускає виконуваний файл, власником якого є інший користувач і для даного файлу встановлений один з цих бітів, файлу тимчасово призначаються права його власника або групи (в залежності від того, який біт заданий).

Таким чином, користувач може навіть запускати файли від імені суперкористувача.

Вісімкові значення для SUID і SGID - 4000 і 2000.

Символьні: u+s і g+s.

Установка бітів SUID або SGID дозволить користувачам запускати виконувані файли від імені власника (або групи) файлу, який запускається.

SGID може застосовуватись і до каталогу.

SGID може бути корисним, тому що ви можете використовувати його для встановлення власника групи за замовчуванням для файлів та підкаталогів, створених у цьому каталозі.

За замовчуванням, коли користувач створює файл, його ефективна первинна група встановлюється як власник групи цього файлу.

Встановити SUID і SGID можна командою **chmod**:

chmod 4755 1.pl // встановлює на файл 1.pl біт SUID і замінює звичайні права на 755 (rwxr-xr-x).

chmod u+s 1.pl // те ж саме, тільки звичайні права не перезаписувати.

chmod 2755 1.pl // встановлює на файл 1.pl біт SGID і замінює звичайні права на 755 (rwxr-xr-x).

chmod g+s 1.pl // те ж саме, тільки звичайні права не перезаписувати.

Зняти встановлені біти можна різними способами:

chmod u-s 1.pl // прибираємо SUID

chmod g-s 1.pl // прибираємо SGID

chmod 0644 1.pl // прибираємо всі додаткові біти і міняємо права на 644.

Відображення SGID і SUID (рисунок 5.23):

ls -l

-rwxrwsrwx 1 root root 0 2021-00-21 19:42 aaaa

Видно, що для файлу aaaa встановлений SGID, про що свідчить символ «s» (-rwxrwsrwx).

Символ «s» може бути як мала літера (s), так і велика (S).

Регістр символу дає тільки додаткову інформацію про налаштування, тобто чи були встановлені права на виконання, до установки **SGID** біта (rwxrwsrwx).

Якщо s рядкова, то права на виконання у групи цього файлу були до установки **SGID**.

Якщо S прописна, то група для цього файлу раніше не мала прав на виконання до установки **SGID**.

```

maryna@maryna-VirtualBox:~/MYDIR$ sudo chmod u+s /bin/chmod
maryna@maryna-VirtualBox:~/MYDIR$ ls -l /bin/chmod
-rwsr-xr-x 1 root root 55808 лют  7  2022 /bin/chmod
maryna@maryna-VirtualBox:~/MYDIR$ sudo chmod g+s /bin/chmod
maryna@maryna-VirtualBox:~/MYDIR$ ls -l /bin/chmod
-rwsr-sr-x 1 root root 55808 лют  7  2022 /bin/chmod

maryna@maryna-VirtualBox:~/MYDIR$ sudo chmod g-s /bin/chmod
maryna@maryna-VirtualBox:~/MYDIR$ sudo chmod u-s /bin/chmod
maryna@maryna-VirtualBox:~/MYDIR$ ls -l /bin/chmod
-rwxr-xr-x 1 root root 55808 лют  7  2022 /bin/chmod
-rwxr-xr-x 1 root root 55808 лют  7  2022 /bin/chmod

maryna@maryna-VirtualBox:~/MYDIR$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59976 бер 14  2022 /usr/bin/passwd
-rwsr-xr-x 1 root root 59976 бер 14  2022 /usr/bin/passwd

```

Рисунок 5.23 – Відображення SGID і SUID в правах доступу

Приклад використання SUID

Наприклад, команду `chmod` за замовчуванням може запускати тільки `root`. Якщо встановити SUID на виконуваний файл `/bin/chmod`, то звичайний користувач зможе використовувати цю команду без використання `sudo`, так, що вона буде виконуватися від імені користувача `root`. За таким принципом працює команда `passwd`, за допомогою якої користувач може змінити свій пароль.

```

maryna@maryna-VirtualBox:~$ ls -l /bin/chmod
-rwxr-xr-x 1 root root 55808 лют  7  2022 /bin/chmod
maryna@maryna-VirtualBox:~$ chmod u=rw /bin/chmod
chmod: changing permissions of '/bin/chmod': Operation not permitted
maryna@maryna-VirtualBox:~$ sudo chmod u=rw /bin/chmod
maryna@maryna-VirtualBox:~$ ls -l /bin/chmod
-rw-r-xr-x 1 root root 55808 лют  7  2022 /bin/chmod
maryna@maryna-VirtualBox:~$ chmod u=rwx /bin/chmod
chmod: changing permissions of '/bin/chmod': Operation not permitted
maryna@maryna-VirtualBox:~$ sudo chmod u+s /bin/chmod
maryna@maryna-VirtualBox:~$ ls -l /bin/chmod
-rwsr-xr-x 1 root root 55808 лют  7  2022 /bin/chmod
maryna@maryna-VirtualBox:~$ chmod g=rwx /bin/chmod
maryna@maryna-VirtualBox:~$ ls -l /bin/chmod
-rwsrwxr-x 1 root root 55808 лют  7  2022 /bin/chmod

```

← chmod працює без sudo

Рисунок 5.24 – Приклад використання SUID

При виконанні файлу, він буде працювати від імені власника (root), навіть, якщо його запускає інший користувач.

Приклад 1 використання SGID (рисунок 5.25, 5.26)

Наприклад, каталог SGID1 створено під користувачем root. На каталог встановлено SGID. У файлів та каталогів в SGID1, які створені іншими користувачами, власник буде користувач, який створив цей файл/каталог, а група власника буде root

```
maryna@maryna-VirtualBox:~/SSS$ sudo mkdir SGID1
maryna@maryna-VirtualBox:~/SSS$ ls -l
total 16
drwxrwxr-x 2 maryna maryna 4096 жов 3 14:37 GID
-rwxr-sr-x 1 root SGIDgroup 0 жов 3 14:37 sgid
drwxr-xr-x 2 root root 4096 жов 3 14:56 SGID1
drwxrwsr-x 2 root root 4096 жов 3 14:10 SGIDdir
drwxrwxrwt 2 maryna maryna 4096 жов 3 13:04 Stickydir
maryna@maryna-VirtualBox:~/SSS$ sudo chmod a=rwx SGID1
maryna@maryna-VirtualBox:~/SSS$ ls -l
total 16
drwxrwxr-x 2 maryna maryna 4096 жов 3 14:37 GID
-rwxr-sr-x 1 root SGIDgroup 0 жов 3 14:37 sgid
drwxrwxrwx 2 root root 4096 жов 3 14:56 SGID1
drwxrwsr-x 2 root root 4096 жов 3 14:10 SGIDdir
drwxrwxrwt 2 maryna maryna 4096 жов 3 13:04 Stickydir
maryna@maryna-VirtualBox:~/SSS$ cd SGID1
maryna@maryna-VirtualBox:~/SSS/SGID1$ touch mar1.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l
total 0
-rw-rw-r-- 1 maryna maryna 0 жов 3 15:07 mar1.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ cd ..
maryna@maryna-VirtualBox:~/SSS$ sudo chmod g+s SGID1
maryna@maryna-VirtualBox:~/SSS$ cd SGID1
```

Рисунок 5.25 – Приклад 1 використання SGID

```

maryna@maryna-VirtualBox:~/SSS/SGID1$ sudo -u test1 touch test1.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l
total 0
-rw-rw-r-- 1 maryna maryna 0 жов  3 15:07 mar1.txt
-rw-rw-r-- 1 test1 root    0 жов  3 15:08 test1.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ touch mar2.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l
total 0
-rw-rw-r-- 1 maryna maryna 0 жов  3 15:07 mar1.txt
-rw-rw-r-- 1 maryna root    0 жов  3 15:09 mar2.txt
-rw-rw-r-- 1 test1 root    0 жов  3 15:08 test1.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ sudo chmod g-s SGID1
chmod: cannot access 'SGID1': No such file or directory
maryna@maryna-VirtualBox:~/SSS/SGID1$ cd ..
maryna@maryna-VirtualBox:~/SSS$ sudo chmod g-s SGID1
maryna@maryna-VirtualBox:~/SSS$ ls -l
total 16
drwxrwxr-x 2 maryna maryna 4096 жов  3 14:37 GID
-rwxr-sr-x 1 root    SGIDgroup  0 жов  3 14:37 sgid
drwxrwxrwx 2 root    root      4096 жов  3 15:09 SGID1
drwxrwsr-x 2 root    root      4096 жов  3 14:10 SGIDdir
drwxrwxrwt 2 maryna maryna 4096 жов  3 13:04 Stickydir
maryna@maryna-VirtualBox:~/SSS$ cd SGID1
maryna@maryna-VirtualBox:~/SSS/SGID1$ sudo -u test1 touch test2.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l
total 0
-rw-rw-r-- 1 maryna maryna 0 жов  3 15:07 mar1.txt
-rw-rw-r-- 1 maryna root    0 жов  3 15:09 mar2.txt
-rw-rw-r-- 1 test1 root    0 жов  3 15:08 test1.txt
-rw-rw-r-- 1 test1 test1   0 жов  3 15:13 test2.txt
maryna@maryna-VirtualBox:~/SSS/SGID1$

```

Рисунок 5.26 – Приклад 1 використання SGID (продовження)

Приклад 2 використання SGID (рисунок 5.27, 5.28)

При виконанні файлу (SGID1.sh), він буде працювати від імені групи власника (root), навіть, якщо його запускає інший користувач. Це корисно для сценаріїв, де потрібно, щоб всі програми завжди працювали від імені певної групи.

```

maryna@maryna-VirtualBox:~/SSS$ ls -l
total 16
drwxrwxr-x 2 maryna maryna 4096 жов 3 14:37 GID
-rwxr-sr-x 1 root SGIDgroup 0 жов 3 14:37 sgid
drwxrwsrwx 2 root root 4096 жов 3 15:35 SGID1
drwxrwsr-x 2 root root 4096 жов 3 14:10 SGIDdir
drwxrwxrwt 2 maryna maryna 4096 жов 3 13:04 Stickydir
maryna@maryna-VirtualBox:~/SSS$ cd SGID1
maryna@maryna-VirtualBox:~/SSS/SGID1$ cat > SGID1.sh
#!/bin/sh

echo SGID
maryna@maryna-VirtualBox:~/SSS/SGID1$ cat SGID1.sh
#!/bin/sh

echo SGID
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l
total 8
-rw-rw-r-- 1 maryna maryna 0 жов 3 15:07 mar1.txt
-rw-rw-r-- 1 maryna root 0 жов 3 15:09 mar2.txt
-rw-rw-r-- 1 maryna root 21 жов 3 15:39 SGID1.sh
-rw-rw-r-- 1 maryna maryna 21 жов 3 15:35 SGID.sh
-rw-rw-r-- 1 test1 root 0 жов 3 15:08 test1.txt
-rw-rw-r-- 1 test1 test1 0 жов 3 15:13 test2.txt

```

Рисунок 5.27 – Приклад 2 використання SGID

```

maryna@maryna-VirtualBox:~/SSS/SGID1$ sudo chmod a=rwx SGID1.sh
maryna@maryna-VirtualBox:~/SSS/SGID1$ ls -l SGID1.sh
-rwxrwxrwx 1 maryna root 25 жов 3 15:44 SGID1.sh
maryna@maryna-VirtualBox:~/SSS/SGID1$ sh
$ ls -l
total 8
-rw-rw-r-- 1 maryna maryna 0 жов 3 15:07 mar1.txt
-rw-rw-r-- 1 maryna root 0 жов 3 15:09 mar2.txt
-rwxrwxrwx 1 maryna root 25 жов 3 15:44 SGID1.sh
-rw-rw-r-- 1 maryna maryna 21 жов 3 15:35 SGID.sh
-rw-rw-r-- 1 test1 root 0 жов 3 15:08 test1.txt
-rw-rw-r-- 1 test1 test1 0 жов 3 15:13 test2.txt
$ ./SGID1.sh
SGID
maryna@maryna-VirtualBox:~/SSS/SGID1$ sudo su test1
test1@maryna-VirtualBox:/home/maryna/SSS/SGID1$ sh
$ ls
mar1.txt mar2.txt SGID1.sh SGID2.sh SGID.sh test1.txt test2.txt
$ ./SGID1.sh
SGID

```

Рисунок 5.28 – Приклад 2 використання SGID (продовження)

Використання sticky-біта в каталогах

Каталог з встановленим **sticky-бітом** означає, що видалити файл з цього каталогу може тільки власник файлу або каталогу, в якому знаходиться файл.

Інші користувачі позбавляються права видаляти файли. Встановити **sticky-біт** в каталозі може тільки користувач з правами root.

Sticky-біт каталогу, на відміну від **sticky-біта** файлу, залишається в каталозі до тих пір, поки власник каталогу або користувач з правами root не вилучить каталог явно або не застосує до нього **chmod**. Власник може видалити **sticky-біт**, але не може його встановити.

Вісімкове значення stiky-біта 1000

Символьне: + t

Встановити sticky-біт на каталог можна використовуючи команду **chmod** (рисунок 5.29)

chmod 1755 111 - з заміною прав;

chmod +t 111 - додавання до поточних прав.

Прибрати sticky-біт на каталог можна:

chmod -t 111

Відображення sticky-біт:

ls -ld

drwxr-xr-t 2 root root 4096 2021-00-21 20:42 111

-rwxr - r-T 1 root root 0 2021-00-21 20:50 aaaa

Видно, що sticky-біт встановлений на каталозі 111, а також на файлі aaaa, про що свідчить символ (t). Символ «t» може бути як мала літера (t), так і велика (T).

Рядкова буква (t) відображається в тому випадку, якщо перед установкою **sticky bit** довільний користувач уже мав право на виконання (x), а прописна (T) - якщо такого права у нього не було. Кінцевий результат один і той же, але регістр символу дає додаткову інформацію про вихідні налаштування.

```

maryna@maryna-VirtualBox:~$ sudo chmod +t /home/maryna/MYDIR
maryna@maryna-VirtualBox:~$ ls -ld /home/maryna/MYDIR
drwxrwxr-t 2 maryna maryna 4096 жов  6 17:12 /home/maryna/MYDIR
maryna@maryna-VirtualBox:~$ sudo chmod -t /home/maryna/MYDIR
maryna@maryna-VirtualBox:~$ ls -ld /home/maryna/MYDIR
drwxrwxr-x 2 maryna maryna 4096 жов  6 17:12 /home/maryna/MYDIR

```

Рисунок 5.29 – Встановлення та відображення sticky bit в правах доступу

Отже, використання **sticky** дозволяє реалізувати, наприклад, каталоги-смітники. Користувачі зможуть писати файли в такі каталоги, але не зможуть видаляти чужі файли.

Приклад створення каталогу смітника з використанням Sticky (рисунок 5.30, 5.31)

```

maryna@maryna-VirtualBox:~/SSS$ ls -l
total 4
drwxrwxrwx 2 root root 4096 жов  2 19:55 SGIDdir
maryna@maryna-VirtualBox:~/SSS$ mkdir Stickydir
maryna@maryna-VirtualBox:~/SSS$ cd Stickydir
maryna@maryna-VirtualBox:~/SSS/Stickydir$ touch maryna1.txt
maryna@maryna-VirtualBox:~/SSS/Stickydir$ sudo -u test1 touch test1.txt
[sudo] password for maryna:
touch: cannot touch 'test1.txt': Permission denied
maryna@maryna-VirtualBox:~/SSS/Stickydir$ cd ..
maryna@maryna-VirtualBox:~/SSS$ ls -l
total 8
drwxrwxrwx 2 root root 4096 жов  2 19:55 SGIDdir
drwxrwxr-x 2 maryna maryna 4096 жов  3 12:43 Stickydir
maryna@maryna-VirtualBox:~/SSS$ sudo chmod o+w Stickydir
maryna@maryna-VirtualBox:~/SSS$ ls -l
total 8
drwxrwxrwx 2 root root 4096 жов  2 19:55 SGIDdir
drwxrwxrwx 2 maryna maryna 4096 жов  3 12:43 Stickydir
maryna@maryna-VirtualBox:~/SSS$ cd Stickydir
maryna@maryna-VirtualBox:~/SSS/Stickydir$ sudo -u test1 touch test1.txt
maryna@maryna-VirtualBox:~/SSS/Stickydir$ sudo -u test2 touch test2.txt
maryna@maryna-VirtualBox:~/SSS/Stickydir$ sudo -u test1 touch test11.txt
maryna@maryna-VirtualBox:~/SSS/Stickydir$ sudo -u test2 touch test22.txt
maryna@maryna-VirtualBox:~/SSS/Stickydir$ ls -l
total 0
-rw-rw-r-- 1 maryna maryna 0 жов  3 12:43 maryna1.txt
-rw-rw-r-- 1 test1 test1 0 жов  3 12:47 test11.txt
-rw-rw-r-- 1 test1 test1 0 жов  3 12:46 test1.txt
-rw-rw-r-- 1 test2 test2 0 жов  3 12:47 test22.txt
-rw-rw-r-- 1 test2 test2 0 жов  3 12:46 test2.txt

```

Рисунок 5.30 – Приклад використання sticky bit

```

maryna@maryna-VirtualBox:~/SSS/stickydir$ sudo -u test1 rm test2.txt
rm: remove write-protected regular empty file 'test2.txt'? y
maryna@maryna-VirtualBox:~/SSS/stickydir$ ls -l
total 0
-rw-rw-r-- 1 maryna maryna 0 жов  3 12:43 maryna1.txt
-rw-rw-r-- 1 test1  test1  0 жов  3 12:47 test11.txt
-rw-rw-r-- 1 test1  test1  0 жов  3 12:46 test1.txt
-rw-rw-r-- 1 test2  test2  0 жов  3 12:47 test22.txt
maryna@maryna-VirtualBox:~/SSS/stickydir$ cd ..
maryna@maryna-VirtualBox:~/SSS$ sudo chmod +t stickydir
maryna@maryna-VirtualBox:~/SSS$ ls -l
total 8
drwxrwxrwx 2 root  root  4096 жов  2 19:55 SGIDdir
drwxrwxrwt 2 maryna maryna 4096 жов  3 12:49 stickydir
maryna@maryna-VirtualBox:~/SSS$ cd stickydir
maryna@maryna-VirtualBox:~/SSS/stickydir$ sudo -u test2 rm test1.txt
rm: remove write-protected regular empty file 'test1.txt'? y
rm: cannot remove 'test1.txt': Operation not permitted
maryna@maryna-VirtualBox:~/SSS/stickydir$ rm test1.txt
rm: remove write-protected regular empty file 'test1.txt'? y
maryna@maryna-VirtualBox:~/SSS/stickydir$ sudo rm maryna1.txt
maryna@maryna-VirtualBox:~/SSS/stickydir$ ls -l
total 0
-rw-rw-r-- 1 test1 test1 0 жов  3 12:47 test11.txt
-rw-rw-r-- 1 test2 test2 0 жов  3 12:47 test22.txt

```

Рисунок 5.31 – Приклад використання sticky bit (продовження)

Користувачі можуть створювати файли в каталозі, але видаляти і модифікувати можуть лише власні файли. Інші користувачі не можуть видаляти і модифікувати файли, створені іншими користувачами. Видалили може власник каталогу root. Корисно в загальних каталогах

Розширені права, додають корисну функціональність до основних дозволів в Linux, але це не дозволяє надавати дозволи більш ніж одному користувачу або одній групі в одному файлі.

Існують додаткові можливості для роботи з првами доступу, **ACL - Access Control List (setfacl, getfacl)** у Linux - списки контролю доступу. ACL дозволяють надавати дозволи більш ніж одному користувачу або одній групі в одному файлі (рисунок 5.32).

Окрім того, вони дозволяють пропонують цю функцію адміністраторам встановлювати стандартні дозволи складним способом, при якому встановлені дозволи можуть відрізнятися в різних каталогах.

Але не завжди є стандартна підтримка для ACL у файловій системі.

Якщо під час налаштування списків ACL для файлової системи ви отримуєте повідомлення «operation not supported», можливо, у вашій файловій системі відсутня підтримка ACL.

Щоб це виправити, потрібно додати опцію `acl mount` у файлі `/etc/fstab`, щоб файлова система була змонтована за допомогою ACL за замовчуванням.

Для встановлення ACL потрібна команда **setfacl**.

Щоб побачити поточні дозволи ACL - **getfacl**.

Опції, які часто використовуються:

-m - Модифікує зазначені ACL на об'єкті. Інші існуючі ACL зберігаються.

-x - Видалення зазначених ACL прав з об'єкта. Стандартні права LinUx не змінюються.

-R - Рекурсивне призначення (видалення) дозволів.

Синтаксис команди:

setfacl <опції> <ключ> <список правил> <об'єкт>

u:<uid>:<perms>*

- призначає ACL доступ заданому користувачу. Тут можна вказати ім'я або UID користувача.

setfacl -m u:maryna2:rw myfile.txt

g:<gid>:<perms>*

- призначає ACL для доступу для заданої групи. Тут можна вказати ім'я або групу GID

setfacl -m g:GROUP:r myfile.txt

o:<perms>*

- призначає ACL для доступу користувачам «other» `setfacl -m o:myfile.txt` (Прибирає всі права (відсутність прав)).

<perms> - правила для користувача або групи можуть приймати значення (r), (x), (w), або поєднання один з одним.

```
maryna@maryna-VirtualBox:~$ sudo useradd maryna1
maryna@maryna-VirtualBox:~$ getfacl /home/maryna/MYDIR
getfacl: Removing leading '/' from absolute path names
# file: home/maryna/MYDIR
# owner: maryna
# group: maryna
user::rwx
user:maryna2:rw-
group::rwx
mask::rwx
other::r-x

maryna@maryna-VirtualBox:~$ setfacl -m u:maryna1:rx /home/maryna/MYDIR
maryna@maryna-VirtualBox:~$ getfacl /home/maryna/MYDIR
getfacl: Removing leading '/' from absolute path names
# file: home/maryna/MYDIR
# owner: maryna
# group: maryna
user::rwx
user:maryna2:rw-
user:maryna1:r-x
group::rwx
mask::rwx
other::r-x
```

Рисунок 5.32 – Приклад використання sticky bit

ACL додає функціональні можливості, але вона має один недолік: не всі утиліти підтримують її. Для прикладу, утиліта `tar` не підтримує ACL. Можна використовувати `star` замість `tar`.

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №6 Ідентифікація користувачів. Umask в Linux. Жорсткі і символічні посилання

Мета: навчитись працювати під різними користувачами, змінювати власника, групу для файлів і каталогів, керувати правами доступу до новостворюваних файлів і каталогів, створювати жорсткі і символічні посилання для файлів/каталогів та розуміти відмінності між ними.

Завдання

- 1) Ознайомитись з командами `id`, `whoami`, `chown` для файлу/каталогу/посилання. Навчитись ними користуватись.
- 2) Ознайомитись з командою `Umask`. Створити файл і каталог, переглянути `umask`, пояснити. Змінити `umask` для новостворюваних файлів.
- 3) Ознайомитись з жорсткими та символічними посиланнями. Створити жорстке і символічне посилання. Пояснити різницю.

Теоретичні відомості

Кожному користувачу окрім імені присвоюється числовий ідентифікатор користувача (UID, User IDentificator).

UID використовується для ідентифікації користувача в системі і для визначення того, до яких системних ресурсів користувач може отримати доступ. Ідентифікатор користувача повинен бути унікальним.

Знайти `UID /etc/passwd`.

Це той же файл, який можна використовувати, щоб відкрити список користувачів в системі Linux.

`root:x:0:0:root:/root:/bin/bash`

`news:x:9:9:news:/var/spool/news:/usr/sbin/nologin`

111:x:1000:1000:111,,,:/home/helder:/bin/bash

davmail:x:127:65534::/var/lib/davmail:/usr/sbin/nologin

Третє поле тут представляє ідентифікатор користувача або UID, четверте поле - GID.

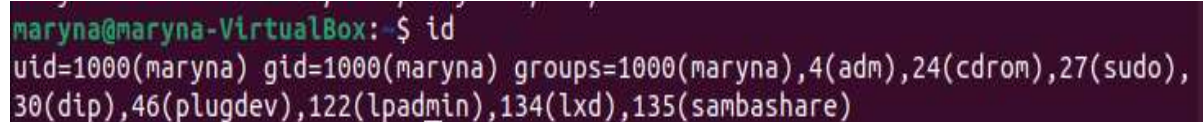
Групи та членів групи можна знайти в файлі /etc/group

daemon:x:1:root,bin,daemon

Третє поле тут представляє ідентифікатор групи користувача або GID.

Команда **id** в Linux відобразить UID, GID, групи, до яких належить поточний користувач (рисунок 6.1):

uid = 1000 (111) gid = 1000 (111) groups = 1000 (111), 4 (adm)...



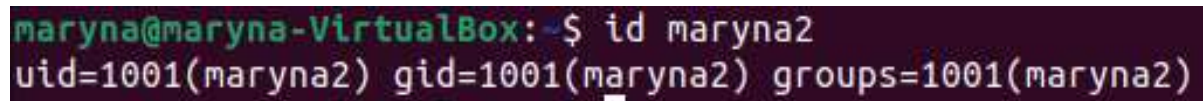
```
maryna@maryna-VirtualBox:~$ id
uid=1000(maryna) gid=1000(maryna) groups=1000(maryna),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),122(lpadmin),134(lxd),135(sambashare)
```

Рисунок 6.1 – Команда id

Також можна вказати імена користувачів за допомогою команди **id**, щоб отримати UID будь-якого користувача Linux (рисунок 6.2):

id username

uid = 1001 (username) gid = 1001 (username) groups = 1001 (username)



```
maryna@maryna-VirtualBox:~$ id maryna2
uid=1001(maryna2) gid=1001(maryna2) groups=1001(maryna2)
```

Рисунок 6.2 – Команда id

Можна змінити UID, змінивши користувача за допомогою команди **usermod** наступним чином (для цього потрібно мати привілеї суперкористувача) (рисунок 6,3):

usermod -u 1004 user_2

```

maryna@maryna-VirtualBox:~$ id maryna2
uid=1001(maryna2) gid=1001(maryna2) groups=1001(maryna2)
maryna@maryna-VirtualBox:~$ sudo usermod -u 1003 maryna2
[sudo] password for maryna:
maryna@maryna-VirtualBox:~$ id maryna2
uid=1003(maryna2) gid=1001(maryna2) groups=1001(maryna2)

```

Рисунок 6.3 – Зміна UID користувача

Команда **whoami**, яка друкує ім'я користувача з чинним ідентифікатором користувача (рисунок 6.4).

```

maryna@maryna-VirtualBox:~$ whoami
maryna
maryna@maryna-VirtualBox:~$ sudo su
[sudo] password for maryna:
root@maryna-VirtualBox:/home/maryna# whoami
root
root@maryna-VirtualBox:/home/maryna# su -maryna
su: invalid option -- 'a'
Try 'su --help' for more information.
root@maryna-VirtualBox:/home/maryna# su - maryna
maryna@maryna-VirtualBox:~$ whoami
maryna

```

Рисунок 6.4 – Команда whoami

Команда **whoami** може бути використана в скриптах для перевірки імені користувача, що запустив сценарій.

В операційних системах Linux кожен файл пов'язаний з власником і групою.

Команда **chown** в Linux – для зміни власника (рисунок 6.5). Для зміни власника потрібно мати права суперкористувача. За допомогою параметрів **chown** можна змінити власника файлів, каталогів і посилань. Якщо звичайний користувач хоче внести певні зміни в файл, привілейований користувач може використовувати команди **chown**, щоб змінити власника і дозволити доступ.

chown filename / dir/ link

Основний формат команди, щоб змінити власника:

chown user filename (s)

chown user /dir

chown user symlink

Приклад, для файлу **1.txt** змінимо власника з root на нового користувача з ім'ям **111**:

chown 111 1.txt

Щоб перевірити зміну власника, можна знову використати команду **ls -l**:

-rw-r - r-- 1 111 root 0 Feb 20 17:45 1.txt

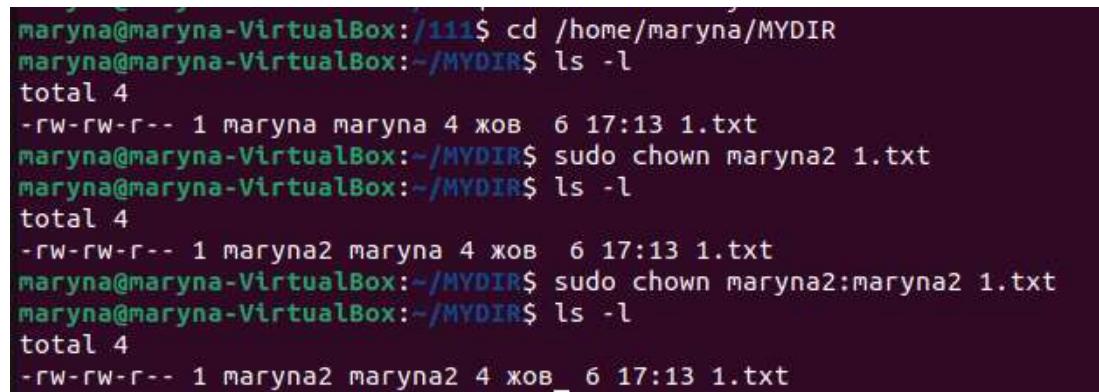
Щоб змінити право власності на файл **1.txt** на нового власника з ім'ям **111** і групу **ggg**, команда буде такою:

chown user[:group] filename(s)

chown 111: ggg 1.txt

Щоб перевірити зміну власника та групи, використовуйте **ls -l**. Вивід цієї команди:

-rw-r - r-- 1 111 ggg 0 Feb 20 17:50 1.txt



```
maryna@maryna-VirtualBox:/111$ cd /home/maryna/MYDIR
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 4
-rw-rw-r-- 1 maryna maryna 4 жов  6 17:13 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ sudo chown maryna2 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 4
-rw-rw-r-- 1 maryna2 maryna 4 жов  6 17:13 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ sudo chown maryna2:maryna2 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 4
-rw-rw-r-- 1 maryna2 maryna2 4 жов  6 17:13 1.txt
```

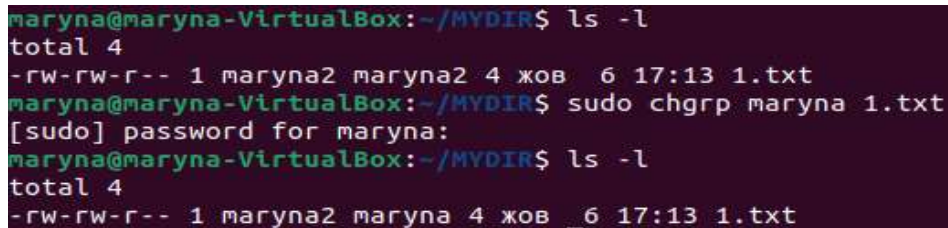
Рисунок 6.5 – Команда chown

Команда **chgrp** дозволяє змінити групи користувачів, які володіють файлом або директорією (рисунок 6.6).

chgrp [options] <groupname> <filename or dirname>

Для того, щоб змінити групу власників директорії, потрібно вказати ім'я групи користувачів і шлях до необхідної директорії:

\$ chgrp root / root



```
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 4
-rw-rw-r-- 1 maryna2 maryna2 4 жов  6 17:13 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ sudo chgrp maryna 1.txt
[sudo] password for maryna:
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 4
-rw-rw-r-- 1 maryna2 maryna 4 жов  6 17:13 1.txt
```

Рисунок 6.6 – Команда chgrp

Команда Umask в Linux

В операційних системах Linux все нові файли створюються з набором дозволів за замовчуванням. Утиліта umask дозволяє переглядати і встановлювати маску режиму створення файлу, який визначає дозволи біт для новостворюваних файлів або каталогів.

За замовчуванням в системах Linux дозволи 666 на створення за замовчуванням призначені для файлів, які дають права на читання і запис для користувача, групи та інших, а також 777 для каталогів, що означає дозвіл на читання, запис і виконання для користувача, групи та інших. Linux не дозволяє створювати файл з дозволами на виконання.

Дозволи на створення за замовчуванням можна змінити за допомогою утиліти umask. umask впливає тільки на поточне середовище оболонки. У більшості дистрибутивів Linux значення umask за замовчуванням для всієї системи встановлюється у файлі pam_umask.so або /etc/profile (рисунок 6.7).

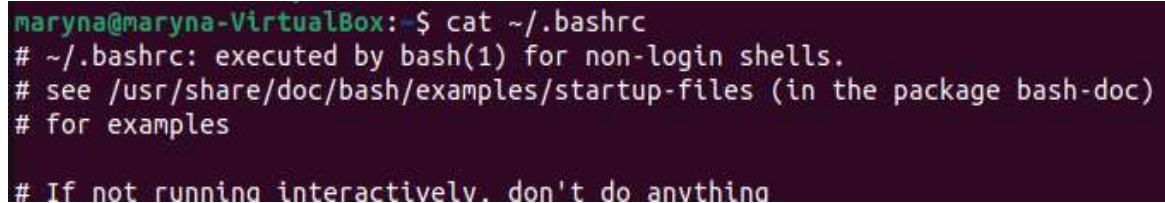


```
maryna@maryna-VirtualBox:~$ cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "${PS1:-}" ]; then
    umask 022
```

Рисунок 6.7 – Конфігураційний файл /etc/profile

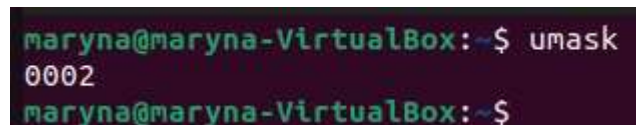
Якщо ви хочете вказати інше значення для кожного користувача, відредагуйте файли конфігурації оболонки користувача, такі як `~/.bashrc` або `~/.zshrc` (рисунок 6.8).



```
maryna@maryna-VirtualBox:~$ cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples
# If not running interactively, don't do anything
```

Рисунок 6.8 – Конфігураційний файл `~/.bashrc`

Щоб переглянути поточне значення маски, потрібно ввести `umask` без аргументів (рисунок 6.9):



```
maryna@maryna-VirtualBox:~$ umask
0002
maryna@maryna-VirtualBox:~$
```

Рисунок 6.9 – Значення маски

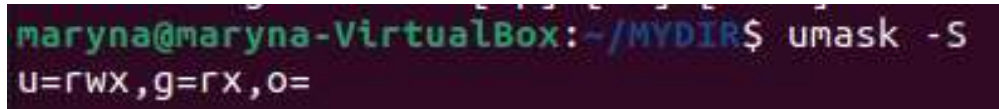
Це значення `umask` містить біти прав доступу, які НЕ будуть встановлені у новостворених файлах і каталогах.

Дозволи на створення файла `666` і для каталогів `777`. Щоб обчислити біти прав доступу для нових файлів, потрібно відняти значення `umask` із значення за замовчуванням.

Наприклад, щоб розрахувати, як `umask 022` вплине на новостворені файли і каталоги, використовуйте:

- файли: $666 - 022 = 644$. Власник може читати і змінювати файли. Група та інші можуть тільки читати файли;
- каталоги: $777 - 022 = 755$. Власник може перейти в каталог і переглянути список, змінити, створити або видалити файли в каталозі.

Можна відобразити значення маски в символічному записі, використовуючи опцію -S (рисунок 6.10):



```
maryna@maryna-VirtualBox: ~/MYDIR$ umask -S  
u=rwx,g=rX,o=
```

Рисунок 6.10 – Значення маски в символічному записі

На відміну від числового запису, значення символічного запису містить біти прав доступу, які будуть встановлені в новостворюваних файлах і каталогах.

Установка значення маски

Маска створення файлу може бути встановлена з використанням вісімкового або символічного запису. Щоб зробити зміни постійними, встановіть нове значення `umask` в глобальному файлі конфігурації, такому як файл `/etc/profile`, який буде впливати на всіх користувачів, або в файлах конфігурації оболонки користувача, таких як `~/.profile`, `~/.bashrc` або `~/.zshrc` які будуть впливати тільки на користувача. Призначені для користувача файли мають пріоритет над глобальними файлами.

Щоб встановити нове значення для всієї системи, відкрийте файл `/etc/profile` в текстовому редакторі: **`sudo nano /etc/profile`** і змініть або додайте наступний рядок на початку файлу `/etc/profile`: **`- umask 022`** (рисунок 6.11).

Щоб зміни вступили в силу, потрібно перезавантажити систему або виконати наступну `source` команду: **`source /etc/profile`**.

Перш ніж вносити зміни в значення `umask`, переконайтеся, що нове значення не представляє потенційну загрозу безпеці. Значення менш суворі, ніж `022` слід використовувати з великою обережністю. Наприклад, `umask 000` означає, що будь-який користувач матиме права на читання, запис і виконання для всіх новостворюваних файлів.

```
GNU nano 6.2 /etc/profile *
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

mask 022

if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then

maryna@maryna-VirtualBox:~$ source /etc/profile
maryna@maryna-VirtualBox:~$ echo 222 > umask3.txt
maryna@maryna-VirtualBox:~$ ls -l umask3.txt
-rw-r--r-- 1 maryna maryna 4 гпу 18 13:36 umask3.txt
maryna@maryna-VirtualBox:~$ umask
0022
```

Рисунок 6.11 – Установка маски для всіх користувачів

Для перевірки нових налаштувань, створимо один новий файл і каталог, використовуючи `mkdir` і `touch`:

`mkdir newdir`

`touch newfile`

Якщо перевірити дозволи за допомогою команди `ls`, можна помітити, що новий файл має 640 і новий каталог 750 дозволів:

`drwxrwxr-x 2 maryna maryna 4096 гпу 18 13:24 newdir`

`-rw-rw-r-- 1 maryna maryna 0 гпу 18 13:24 newfile`

Інший спосіб встановити маску створення файлу - використовувати символічний запис (рисунок 6.12). Наприклад так `umask u=rwx, g=rx, o=` теж же, як `umask 027`.

```
maryna@maryna-VirtualBox:~/MYDIR$ umask u=rwx,g=rx,o=
maryna@maryna-VirtualBox:~/MYDIR$ umask
0027
```

Рисунок 6.12 – Установка маски в символічному записі

Приклад: змінити маску на 027, яка буде діяти в рамках поточної сесії (рисунок 6.13).

```
maryna@maryna-VirtualBox:~$ umask
0002
maryna@maryna-VirtualBox:~/MYDIR$ echo 123123 > 6.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 24
-rw-rw-r-- 1 maryna2 maryna 4 жов 6 17:13 1.txt
-rw-rw-r-- 1 maryna maryna 9 жов 12 14:29 2.txt
-rw-rw-r-- 1 maryna maryna 8 жов 12 14:33 3.txt
-rw-rw-r-- 1 maryna maryna 6 жов 12 14:45 4.txt
-rw-r----- 1 maryna maryna 6 жов 12 14:49 5.txt
-rw-rw-r-- 1 maryna maryna 7 жов 12 15:11 6.txt
maryna@maryna-VirtualBox:~/MYDIR$ umask 027
maryna@maryna-VirtualBox:~/MYDIR$ echo 567567 > 7.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 28
-rw-rw-r-- 1 maryna2 maryna 4 жов 6 17:13 1.txt
-rw-rw-r-- 1 maryna maryna 9 жов 12 14:29 2.txt
-rw-rw-r-- 1 maryna maryna 8 жов 12 14:33 3.txt
-rw-rw-r-- 1 maryna maryna 6 жов 12 14:45 4.txt
-rw-r----- 1 maryna maryna 6 жов 12 14:49 5.txt
-rw-rw-r-- 1 maryna maryna 7 жов 12 15:11 6.txt
-rw-r----- 1 maryna maryna 7 жов 12 15:11 7.txt
maryna@maryna-VirtualBox:~/MYDIR$ umask
0027
```

Рисунок 6.13 – Зміна маски для поточної сесії

Жорсткі і символічні посилання

Для уникнення багаторазового копіювання одного і того ж файлу, якщо він необхідний в декількох місцях, Linux використовує посилання.

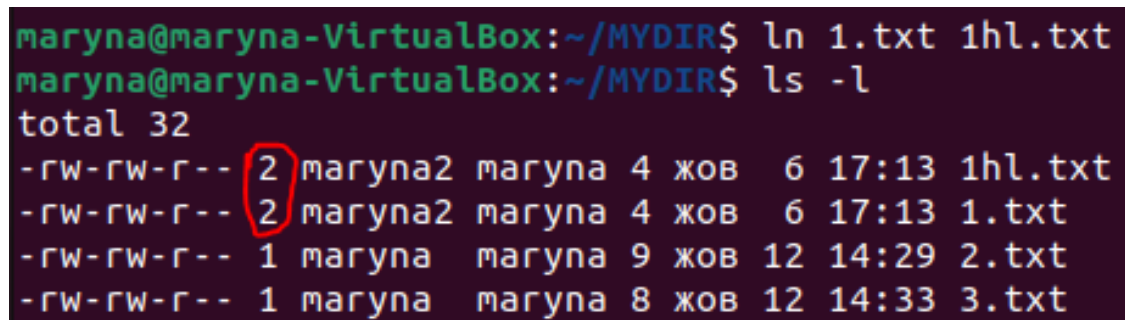
Linux використовує два різних типи посилань - жорсткі і символічні.

Жорсткі посилання:

- жорстке посилання працює як дзеркальна копія оригінального файлу
- жорсткі посилання використовують одні і ті ж індексні дескриптори inode (використовують один номер inode);
- зміни в одному з файлів, пов'язаних жорстким посиланням, відібуваються в іншому;
- при видаленні оригінального файлу жорстке посилання буде показувати вміст;

- у файлів посилань є реальний вміст;
- видалення будь-якого посилання, не впливає на інші посилання;
- жорсткі посилання не можуть пов'язувати різні файлові системи (вони повинні знаходитися в одній файловій системі з файлом, на який посилаються);
- жорсткі посилання можуть вказувати тільки на файли (не на каталоги);
- для створення посилання використовується команда **ln** (рисунок 6.14);
- команда `ls -l` показує всі посилання, а в стовпці посилань виводить їх кількість.

ln <джерело> <посилання>



```

maryna@maryna-VirtualBox:~/MYDIR$ ln 1.txt 1hl.txt
maryna@maryna-VirtualBox:~/MYDIR$ ls -l
total 32
-rw-rw-r-- 2 maryna2 maryna 4 жов  6 17:13 1hl.txt
-rw-rw-r-- 2 maryna2 maryna 4 жов  6 17:13 1.txt
-rw-rw-r-- 1 maryna maryna 9 жов 12 14:29 2.txt
-rw-rw-r-- 1 maryna maryna 8 жов 12 14:33 3.txt
  
```

Рисунок 6.14 – Жорстке посилання

Наприклад, створимо директорію MYDIR і файл 1.txt всередині неї, створимо жорстке посилання на 1.txt з ім'ям 1hl.txt:

\$ mkdir MYDIR

\$ cd MYDIR

\$ echo "file" > 1.txt

\$ ln 1.txt 1hl.txt

За допомогою опції -і команди ls можна переглянути номери inode для обох файлів (рисунок 6.15):

```
$ ls -il 1.txt 1hl.txt
```

Отримаємо наступний результат:

```
396074 -rw-rw-r-- 2 maryna2 maryna 4 жов  6 17:13 1hl.txt
396074 -rw-rw-r-- 2 maryna2 maryna 4 жов  6 17:13 1.txt
```

Рисунок 6.15 – Один inode у файла і посилання

З результату видно, що у обох файлів один номер іноди (396074). Окрім того, у файлів однаковий розмір і права доступу.

Тепер видалимо оригінальний 1.txt: **\$ rm 1.txt**

Після видалення пов'язаного жорсткого посилання оригінального файлу перевіримо вміст «посилання» 1hl.txt: **\$ cat 1hl.txt**

Повинно бути виведено вміст файлу (рисунок 6.16):

```
maryna@maryna-VirtualBox:~/MYDIR$ rm 1.txt
maryna@maryna-VirtualBox:~/MYDIR$ cat 1hl.txt
file
```

Рисунок 6.16 – Після видалення фалу, посилання робоче

Символічне посилання

Символічне посилання (symlink, softlink) – це покажчик на файл за його іменем;

- символічне посилання містить шлях до оригінального файлу, але не його вміст;
- символічні посилання використовують різні номери inode;
- символічні посилання працюють і з файлами і з каталогами;
- якщо вихідний файл, на який вказує посилання видалити, то всі символічні посилання стануть пошкодженим ;

- символічне посилання вказує на оригінальний файл, тому при видаленні цього файлу символічне посилання перестане працювати;
- при видаленні символічного посилання з файлом нічого не станеться. це пов'язано з тим, що inode самого файлу або директорії відрізняється від inode файлу, що створюється для символічного посилання;
- видалення символічного посилання ні на що не впливає;
- символічні посилання можуть пов'язувати різні файлові системи (можуть вказувати на файл, розташований в іншій файлової системі);
- для створення символічного посилання необхідно використовувати опцію -s в команді ln (рисунок 6.17);
- команда ls -l показує всі посилання зі значенням 1 у другому стовпці, а посилання вказує на оригінальний файл.

ln -s <джерело> <посилання>

```
marynaasus@marynaasus-VirtualBox:~/MYDIR$ echo "link" > 2.txt
marynaasus@marynaasus-VirtualBox:~/MYDIR$ ln -s 2.txt 2sl.txt
marynaasus@marynaasus-VirtualBox:~/MYDIR$ ls -il
total 4
1456529 lrwxrwxrwx 1 marynaasus marynaasus 5 лис 12 11:46 2sl.txt -> 2.txt
1456524 -rw-rw-r-- 1 marynaasus marynaasus 5 лис 12 11:46 2.txt
```

Рисунок 6.17 – Символічне посилання

Наприклад, створимо символічне посилання (опція -s) на 2.txt за допомогою наступної команди:

\$ ln -s 2.txt 2sl.txt

Відобразимо номери inode для обох файлів (рисунок 6.18):

\$ ls -il 2.txt 2sl.txt

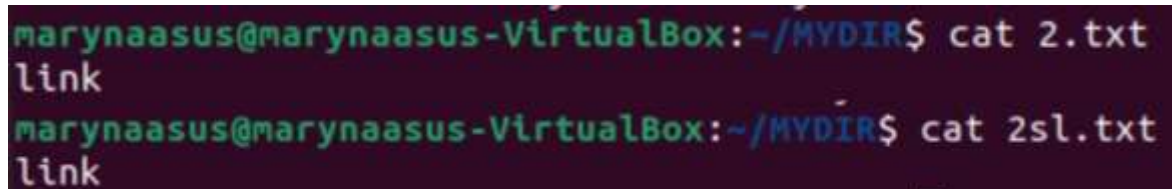
Результат буде таким:

```
1456529 lrwxrwxrwx 1 marynaasus marynaasus 5 лис 12 11:46 2sl.txt -> 2.txt
1456524 -rw-rw-r-- 1 marynaasus marynaasus 5 лис 12 11:46 2.txt
```

Рисунок 6.18 –inode-и та права доступу у файла і посилання різні

Видно, що номери inode різні, а у символічному посиланні інші права доступу (rwxrwxrwx), перед якими зазначено l, так як це символічне посилання.

Тепер можна вивести вміст файлу і символічного посилання (рисунок 6.19):



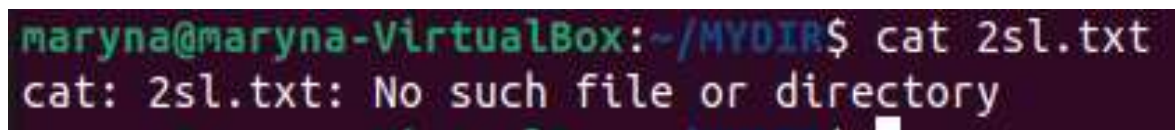
```
marynaasus@marynaasus-VirtualBox:~/MYDIR$ cat 2.txt
link
marynaasus@marynaasus-VirtualBox:~/MYDIR$ cat 2sl.txt
link
```

Рисунок 6.19 – Вміст файлу і посилання співпадає

Видалимо оригінальний файл: **\$ rm 1.txt**

Перевіримо вміст директорії MYDIR: **\$ ls**

У ній ще буде символічне посилання **2sl.txt**, але при спробі вивести вміст **\$ cat 1sl.txt**, ви отримаєте повідомлення про відсутність такого файлу або директорії (рисунок 6.20):



```
maryna@maryna-VirtualBox:~/MYDIR$ cat 2sl.txt
cat: 2sl.txt: No such file or directory
```

Рисунок 6.20 – Після видалення фалу, посилання не робоче

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №7 Управління процесами

Мета: ознайомитись з поняттями процес, дочірній процес потік в ОС та їх характеристиками та станами, в яких вони можуть перебувати, навчитись керувати процесами: переглядати, створювати, завершувати процеси, керувати пріоритетністю процесів.

Завдання

Розібратися з командами управління процесами, яким виділяються необхідні ресурси обчислювальної системи (ps, bg, fg, jobs, kill, killall, nohup, SIGHUP, SIGTERM, SIGK, pgerp, pkill, screen, htop, top, free, nice, renice)..

Теоретичні відомості

У Linux для кожної окремої програми, при її запуску створюється процес. Неважливо запускаєте програму ви вручну або це робить система або ядро.

Процес

Процес - це кожна програма. Для кожної програми, що запускається створюється окремий процес. В рамках процесу програмі виділяється процесорний час, оперативна пам'ять та інші системні ресурси.

Процеси в linux можна описати як контейнери, в яких зберігається вся інформація про стан і виконання програми.

Якщо програма працює добре, то все нормально, але якщо вона зависла або вам потрібно налаштувати її роботу, може знадобитися управління процесами в Linux.

Процеси поділяються на три типи:

- системні процеси - є частиною ядра і завжди розташовані в оперативній пам'яті;

- демони - це неінтерактивні процеси, які запускаються шляхом завантаження в пам'ять відповідних їм програм (виконуваних файлів), і виконуються у фоновому режимі;

- прикладні (користувацькі) процеси. до прикладних процесів відносяться всі інші процеси, запущені в системі.

В ОС Linux кожен процес обов'язково знаходиться в одному з станів і може бути переведений з одного стану в інший, системою або командами користувача. Розрізняють наступні стани процесів:

- TASK_RUNNING - процес готовий до виконання або виконується (runnable). Позначається символом R;

- TASK_INTERRUPTIBLE - чекаючий процес (sleeping). Цей стан означає, що процес заблокований і очікує сигналу для відновлення і переходу в стан виконання (S);

- TASK_STOPPED - виконання процесу зупинено (stopping). Будь-який процес можна зупинити. Це може робити як система так і користувач (T);

- TASK_ZOMBIE - завершився процес (zombie). Процеси даного стану виникають у разі, коли батьківський процес не чекаючи завершення дочірнього процесу, завершує працювати. Процеси з таким станом позначаються символом Z. такі процеси більше не виконуються системою, але далі продовжують споживати ресурси;

- TASK_UNINTERRUPTIBLE -неперерваний процес (uninterruptible). Процеси в цьому стані очікують завершення операції введення-виведення з прямим доступом в пам'ять. Такий процес не можна завершити (перервати сигналом), поки не завершиться операція введення/виведення (D). Стан аналогічний TASK_INTERRUPTIBLE, за винятком того, що процес не відновлює виконання при отриманні сигналу.

У кожного процесу є свій ідентифікатор, Process ID або просто PID, по них найчастіше і визначаються процеси Linux. PID визначається не випадково, програма ініціалізації systemd, яка запускається відразу після завершення завантаження ядра теж має свій процес з PID 1, а кожна наступна запущена програма - на одиницю більше. Таким чином PID користувацьких програм доходить вже до кількох тисяч.

Процеси можна переглядати та управляти ними. Для перегляду процесів можна перейти в кореневий каталог, відкрити каталог /proc. В ній є підкаталоги з номерами. Ці номери - PID всіх запущених процесів. У кожному з цих каталогів знаходиться вся інформація про процес.

Цей PID показує, як система ідентифікує конкретний процес. Systemd - це перший запущений процес з PID 1 в Linux.

Якщо переглянути каталог процесу 1, в ньому є підкаталоги і багато файлів. Файл cmdline містить інформацію про команду запуску процесу (рисунок 7.1):

```
$ cat /proc/1/cmdline
```

```
$ /usr/lib/systemd/systemd
```

```
maryna@maryna-VirtualBox:/proc$ ls
1      1436  1660  183  272  636  84      devices      mounts
10     1442  1666  184  273  637  840     diskstats    mtd
100    1452  1670  1841 275  640  85      dma          mtrr
1051   1472  1672  1865 277  643  87      driver       net
108    1481  1673  19  280  650  88      dynamic_debug pagetypeinfo
11     1483  1674  1934 281  652  89      execdomains  partitions
111    1496  1675  1959 283  653  9      fb           pressure
112    15  1676  1981 2846 674  90      filesystems  schedstat
117    1501  1679  3  3  693  93      fs           scsi

maryna@maryna-VirtualBox:/proc/1$ ls
ls: cannot read symbolic link 'cwd': Permission denied
ls: cannot read symbolic link 'root': Permission denied
ls: cannot read symbolic link 'exe': Permission denied
arch_status      cwd              mem              patch_state      stat
attr             environ          mountinfo         personality       statm
autogroup         exe              mounts            projid_map        status
auxv              fd               mountstats        root              syscall
cgroup            fdinfo          net               sched             task
clear_refs        gid_map         ns                schedstat         timers_offsets
cmdline           io              numa_maps         sessionid          timers
comm              limits          oom_adj           setgroups          timerslack_ns
coredump_filter   loginuid        oom_score         snaps              uid_map
cpu_resctrl_groups map_files       oom_score_adj     smaps_rollup      wchan
cpuset            maps            pagemap           stack
```

```
maryna@maryna-VirtualBox:/proc/1$ cat cmdline
/sbin/init splash
maryna@maryna-VirtualBox:/proc/1$
```

Рисунок 7.1 – Каталог /proc та каталог 1-ого процесу

За допомогою каталогу `/proc` можна зробити все. Але це дуже незручно, особливо з огляду на кількість запущених процесів в системі. Тому для реалізації потрібних завдань існують спеціальні утиліти, які дозволяють реалізувати управління процесами в Linux.

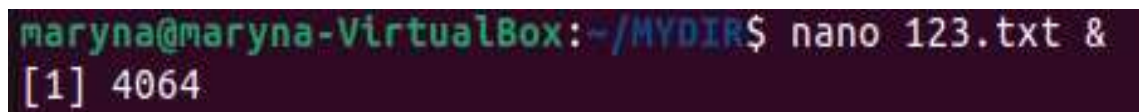
Управління процесами в LINUX

У Linux є дуже велика кількість утиліт для вирішення різних завдань (створення, відстеження, знищення, зміни пріоритету) з управління процесами. Це і такі багатофункціональні рішення, як `htop`, `top`, а також прості утиліти: `ps`, `bg`, `fg`, `jobs`, `kill`, `killall`, `nohup`, `SIGHUP`, `SIGTERM`, `SIGK`, `pgerp`, `pkill`, `screen` та інші.

Коли ви відкриваєте нову оболонку, ви маєте можливість створювати свої процеси. Фактично це те, що відбувається, коли ви вводите будь-яку команду (або серію команд) в термінал.

На рівні користувача з оболонки можна вручну відправляти завдання для роботи у **фоновий режим**, виводити їх на **передній план** або **зупиняти** (за допомогою `ctrl+z`).

Відправити завдання працювати у фоновому режимі можна додаванням до своєї команди символу `&` (рисунок 7.2). Ця команда повертає номер завдання у дужках. Наступний номер це ідентифікатор процесу PID, що використовується ядром.



```
maryna@maryna-VirtualBox:~/MYDIR$ nano 123.txt &  
[1] 4064
```

Рисунок 7.2 – Переведення процесу у фоновий режим

Команда **jobs** - список поточних процесів, прикріплених до оболонки.

Якщо ви спробуєте ввести команду `jobs`, ви можете бачити, що нічого не відбувається. Це тому, що у оболонці не було активних процесів. коли ви маєте запуснене завдання, ви можете запусити команду завдання успішно.

Якщо ви маєте запуснене завдання, команда `jobs` буде успішно виконана (рисунок 7.3).

```
maryna@maryna-VirtualBox:~/MYDIR$ nano 123.txt &  
[1] 4064  
maryna@maryna-VirtualBox:~/MYDIR$ jobs  
[1]+  Stopped                  nano 123.txt
```

Рисунок 7.3 – Команда `jobs`

Ключі команди `jobs`:

-l показати список процесів і їх PID (рисунок 7.4).

```
maryna@maryna-VirtualBox:~/MYDIR$ jobs -l  
[1]+  4064 Stopped (tty output)    nano 123.txt
```

Рисунок 7.4 – Команда `jobs -l`

Команда **fg** - виводить процес на передній план (відновлює виконання процесу в інтерактивному режимі) (рисунок 7.5).

Команда **bg** - перемістить процес на задній план (відновлює виконання процесу в фоновому режимі) (рисунок 7.5).

Ключі команд **fg**, **bg**:

Ctrl+z призупинення виконання процесу і переміщення в фон.

% звернення до процесу по його імені.

%? зверненням до процесу по частині його імені.

& - запусає процес у фоновому режимі

```

maryna@maryna-VirtualBox:~/MYDIR$ gedit
^Z
[1]+  Stopped                  gedit
maryna@maryna-VirtualBox:~/MYDIR$ bg %1
[1]+  gedit &
maryna@maryna-VirtualBox:~/MYDIR$ jobs
[1]+  Running                  gedit &
maryna@maryna-VirtualBox:~/MYDIR$ fg %1
gedit
^Z
[1]+  Stopped                  gedit
maryna@maryna-VirtualBox:~/MYDIR$

```

Рисунок 7.4 – Команди bg, fg

Команда **kill** використовується для знищення процесу з командної оболонки через його PID. Команда kill за замовчуванням посилає сигнал SIGTERM який ініціалізує процедуру завершення процесу, дозволяє коректно його зупинити, також будуть зупинені і все дочірні процеси.

Ключі команди kill:

-l показати список можливих сигналів (рисунок 7.5).

```

maryna@maryna-VirtualBox:/$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO        30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX

```

Рисунок 7.5 – Сигнали команди kill

При знищенні процесу потрібно вибрати сигнал для відправки. Сигнал повідомляє ядру, як ми хочемо завершити процес. Найбільш часто використовувані сигнали:

SIGTERM (terminate), числовий ідентифікатор 15. ядро дає процесу знати, що він має незабаром зупинитися.

SIGTERM дає процесу можливість виходу і виконання операцій безпечного виходу.

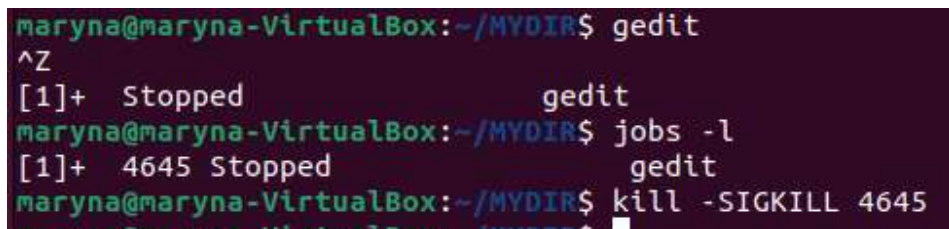
SIGHUP (hang up), числовий ідентифікатор 1. більшість демонів перезапускається при відправці **SIGHUP**. Це часто використовується в процесах, коли були внесені зміни в файл конфігурації.

SIGKILL (kill), числовий ідентифікатор 9. Коли процес зависає, опція **SIGKILL** використовується для явного завершення процесу. Це означає, що після завершення процесу за допомогою **sigkill**, дочірні процеси залишаються в пам'яті і стають «осиротілими» (orphaned), на диску можуть залишитися тимчасові файли процесу (рисунок 7.6).

SIGINT (Interrupt), числовий ідентифікатор 2. Використовується для зупинки процесу користувачем (аналог комбінації Ctrl-C).

SIGCONT (Continue), числовий ідентифікатор 18. Використовується для відновлення виконання процесу.

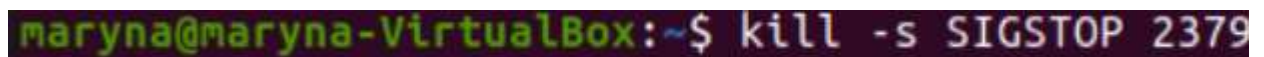
SIGTSTP (TTY stop), числовий ідентифікатор 20 Надсилається для призупинення виконання процесу (аналог комбінації Ctrl-Z).



```
maryna@maryna-VirtualBox:~/MYDIR$ gedit
^Z
[1]+  Stopped                  gedit
maryna@maryna-VirtualBox:~/MYDIR$ jobs -l
[1]+  4645 Stopped              gedit
maryna@maryna-VirtualBox:~/MYDIR$ kill -SIGKILL 4645
```

Рисунок 7.6 – Сигнали команди kill

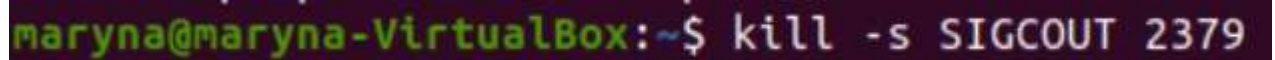
Щоб зупинити виконання деякого процесу, з ідентифікатором 2379, можна скористатися наступною командою (рисунок 7.7)



```
maryna@maryna-VirtualBox:~$ kill -s SIGSTOP 2379
```

Рисунок 7.7 – Команда kill -SIGSTOP

Для його відновлення, можна застосувати таку команду (рисунок 7.8)



```
maryna@maryna-VirtualBox:~$ kill -s SIGCOUT 2379
```

Рисунок 7.8 – Сигнали команди kill -SIGCOUT

Команда **sleep** - для очікування протягом зазначеного числа секунд:

sleep <час очікування>

\$ sleep 2m 30s

Команда **killall** - вбиває всі процеси. Потрібно бути обережним, використовуючи **killall** як користувач root, так як він знищить всі процеси для всіх користувачів.

Команда **pkill** - дозволить адміністратору відправляти сигнал знищення по імені процесу.

Команда **pgrep** - переглядає активні процеси в системі та видає ідентифікатори процесів (рисунок 7.9).

Pgrep process_name - виводить PID процесів, які містять назву процесу (process_name).

Ключі команд **pgrep**, **pkill**:

-l показати ім'я шуканого процесу

-U пошук по uid користувача. Здається як числовий так і символічний ідентифікатор користувача

-u пошук по user effective id користувача. Здається як числовий та й символічний ідентифікатор користувача

-G пошук по gid груп. Здається як числовий та й символічний ідентифікатор користувача

-g пошук по group effective id групи. Здається як числовий та й символічний ідентифікатор користувача.

```
maryna@maryna-VirtualBox:~/MYDIR$ pgrep -u maryna
1358
1359
1366
1367
```

Рисунок 7.9 – Команда pgrep

Команда **nohup**. При роботі з оболонки або терміналу, за замовчуванням всі процеси і задачі, прикріплені до оболонки завершуються, коли оболонка закривається або користувач виходить з системи.

При використанні nohup процес триватиме, якщо користувач закриє оболонку, до якої приєднано процес (рисунок 7.10).

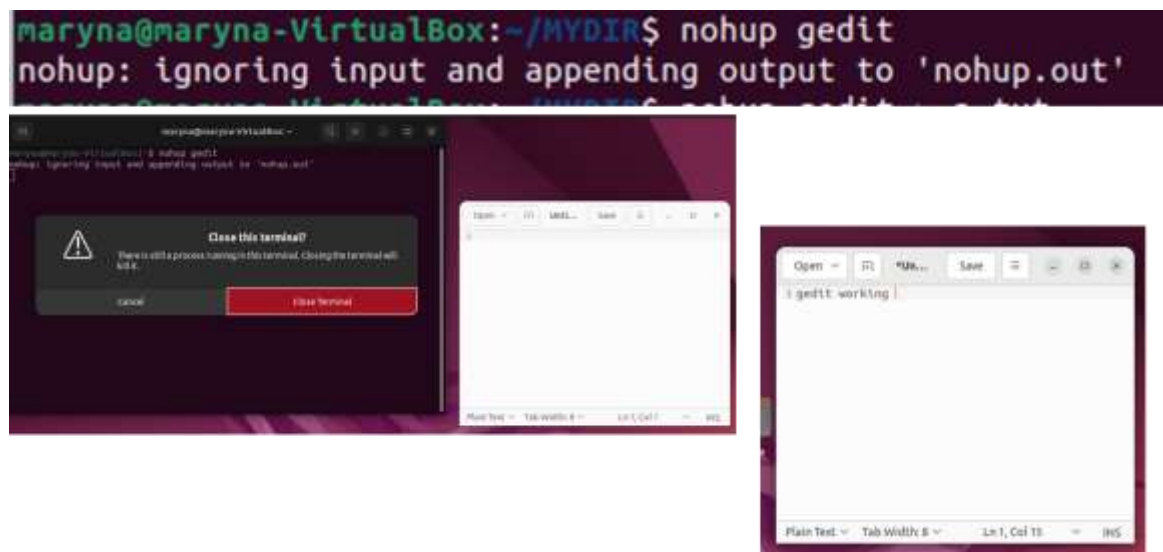


Рисунок 7.10 – Команда nohup

Якщо вивід програми має бути у термінал, nohup перенаправить його у файл nohup.out у поточній директорії (рисунок 7.11).

```
maryna@maryna-VirtualBox:~$ ls
1      D2      Documents  file3  MYDIR      Public      Templates
1.txt  D3      Downloads  file4  myflesh.rules  script2     test1
2      Desktop  echo       l1.txt  nohup.out   script1.sh  umask1.txt
crontab  dir1     file1      l2.txt  Pictures    snap        umask2.txt
D1     dir2     file2      Music   pipe2       SSS         Videos
maryna@maryna-VirtualBox:~$
```

Рисунок 7.11 – Команда nohup, файл nohup.out для виводу програми

Команда **screen** це термінальний мультиплексор, він дає нам можливість всередині діючої сесії відкрити скільки завгодно багато віртуальних вікон / терміналів. Що важливо - процес, запущений всередині сесії через screen, триватиме навіть тоді, коли ви відключитеся від самої першої сесії.

Найпростіший запуск здійснюється командою: **\$ screen** (рисунок 7.12).

У вас відкриється нова сесія в новому вікні. Вже тут ви можете вводити всі необхідні shell команди.

```
maryna@maryna-VirtualBox:~$ screen -ls
There are screens on:
  5552.qqqq      (19.10.22 13:56:49)      (Attached)
  5511.pts-0.maryna-VirtualBox (19.10.22 13:46:46)      (Detached)
  5501.pts-0.maryna-VirtualBox (19.10.22 13:46:07)      (Detached)
  5254.pts-0.maryna-VirtualBox (19.10.22 13:41:07)      (Detached)
4 Sockets in /run/screen/S-maryna.
```

Рисунок 7.12 – Перелік сесій

Згорнути запущений скрін разом з усім, що в ньому працює можна натиснувши **Ctrl+A**, після чого натиснути **D**.

Ctrl+A переведе screen в командний режим, клавіша **D** дасть команду згорнути поточний скрін.

Тепер можна закрити термінал - запущений в згорнутому скрині процес продовжить роботу в фоновому режимі.

Для того, щоб повернутися до згорнутого скріна, досить запустити:

\$ screen -r

Ключ -r вкаже програмі, що потрібно розгорнути згорнутий скрін.

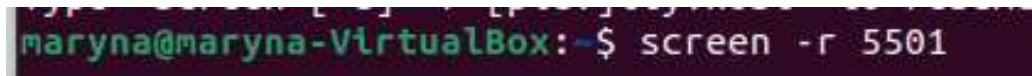


Рисунок 7.13 – Підключення до конкретної сесії

Далі потрібно вийти з сесії за допомогою команди `exit`. `screen` повідомить про завершення своєї роботи. Таким чином, досить вийти з запущеного в скріну процесу, а `screen` вимкнеться сама.

Ключі команди **screen**:

`screen -ls` вивести список сесій `screen`

`screen -r` підключитися до сесії `screen`

`screen -S` присвоїти ім'я сесії `screen`

`ctrl + a d` відключитися від поточної сесії `screen`, залишивши її працюючою.

`ctrl + a k` завершити поточну сесію `screen`

`ctrl + a ?` відобразити список команд і їх параметри за замовчуванням

`ctrl + a :` введення команди для `screen`

`ctrl + a "` список вікон

`ctrl + a 0` відкрити вікно 0

`ctrl + a A` перейменувати поточне вікно

`ctrl + a c` створити нове вікно

`ctrl + a S` розділити поточне вікно на два регіони

`ctrl + a tab` перемикнути фокус введення на наступний регіон

`ctrl + a Q` закрити всі регіони крім поточного

`ctrl + a X` закрити поточний регіон.

Але зараз ця команда мало використовується, бо в Linux можна відкрити кілька вікон/терміналів.

Моніторинг процесів

Команда **top** - це стандартна програма, яка використовується для моніторингу завантаження системи в режимі реального часу (рисунок 14). У команді **top** можна відстежувати програми, в залежності від різних споживаних ресурсів, таких як завантаження процесора, використання пам'яті, ім'я користувача і так далі.

```

maryna@maryna-VirtualBox:/$ top

top - 10:57:52 up 32 min,  1 user,  load average: 0,55, 0,26, 0,50
Tasks: 172 total,   1 running, 171 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1,0 us,   0,7 sy,   0,0 ni, 98,3 id,   0,0 wa,   0,0 hi,   0,0 si,   0,0 st
MiB Mem :  3935,9 total,  1113,0 free,   986,1 used,  1836,9 buff/cache
MiB Swap:  2048,0 total,  2048,0 free,    0,0 used.  2681,6 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
  115 root        0 -20     0     0     0  I   0,3   0,0   0:01.44 kworke+
 1587 maryna    20   0 628832  82404 49340  S   0,3   2,0   0:19.71 Xorg
 1775 maryna    20   0 3464308 356476 129108 S   0,3   8,8   1:16.96 gnome-+
 2369 maryna    20   0 949668  88880 62468  S   0,3   2,2   0:09.79 shotwe+
 2821 maryna    20   0 827264  52656 39144  S   0,3   1,3   0:06.76 gnome-+
 13709 maryna    20   0  23324   3996  3480  R   0,3   0,1   0:00.25 top
    1 root        20   0 103604  13288  8460  S   0,0   0,3   0:11.04 systemd
    2 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kthrea+
    3 root        0 -20     0     0     0  I   0,0   0,0   0:00.00 rcu_gp
    4 root        0 -20     0     0     0  I   0,0   0,0   0:00.00 rcu_pa+
    6 root        0 -20     0     0     0  I   0,0   0,0   0:00.00 kworke+
    9 root        0 -20     0     0     0  I   0,0   0,0   0:00.00 mm_per+
   10 root        20   0     0     0     0  S   0,0   0,0   0:01.04 ksoftt+
   11 root        20   0     0     0     0  I   0,0   0,0   0:03.76 rcu_sc+
   12 root        rt   0     0     0     0  S   0,0   0,0   0:00.03 migrat+
   13 root       -51   0     0     0     0  S   0,0   0,0   0:00.00 idle_l+
   14 root        20   0     0     0     0  S   0,0   0,0   0:00.00 cpuhp/0
   15 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kdevtm+
  
```

Рисунок 7.14 – Команда **top**

Перший рядок (**top**) дає нам уявлення про завантаження системи:

поточний час

up – тривалість роботи системи з моменту запуску

user – поточна кількість користувачів системи

load average – середня завантаженість системи одну хвилину тому, п'ять та 15 відповідно.

Другий рядок даних дає інформацію про загальну кількість процесів у системі, а також про їх статус.

Далі інформація про стан процесора (%Cpu), ОЗП (Mem), SWAP.

PID - номер процесу

USER - ім'я користувача, що його запустив

PR – пріоритет, який має процес в даний момент («rt» означає, що завдання запущено у розкладі пріоритетів у реальному часі (це найвищий пріоритет)).

NI - пріоритет, присвоєний йому командою NICE

VIRT – віртуальна пам'ять, яку використовує процес

RES – фізична пам'ять, зайнята цим процесом

SHR – загальний обсяг пам'яті, яку цей процес ділить з іншими

S – поточний статус процесу: R – running; S – sleeping, Z – zombie, I – IDLE (не виконується)

%CPU – відсоток використовуваного часу центрального процесора

%MEM – відсоток ОЗП, що використовується процесом

TIME+ – тривалість роботи процесу з моменту запуску

COMMAND – назва команди (програми), яка ініціювала процес.

Базове сортування даних здійснюється за використання часу **CPU** (%CPU).

Щоб виконати у **top** сортування за використаною пам'яттю (%MEM), потрібно ввести **Shift+M** у режимі роботи команди **top**.

Щоб побачити, який із процесів найдовше працює, потрібно натиснути **Shift+T** (TIME+).

Можна відсортувати процеси за номером (PID), потрібно ввести **Shift+N** (рисунок 15).

Щоб повернутись до режиму сортування за рівнем споживання ресурсів процесора - **Shift+P**.

Додавання стовпців та сортування таблиці **top**

Щоб відкрити меню вибору полів - комбінація **Shift+F** (рисунок 16).

За допомогою клавіш навігації знаходимо потрібний параметр, клавішею “d” закріплюємо його додавання до загальної таблиці команди **top** (як підтвердження вашого вибору поруч з'явиться символ “*”).

Щоб задати сортування за необхідним параметром, натискаємо “s” і виходимо з меню (ESC).

```

top - 19:45:03 up 18 min, 1 user, load average: 0,00, 0,00, 0,04
Tasks: 186 total, 1 running, 185 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2,1 us, 0,3 sy, 0,0 ni, 97,6 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 1968,6 total, 98,5 free, 829,3 used, 1040,8 buff/cache
MiB Swap: 2190,0 total, 2081,5 free, 108,5 used. 948,8 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR  S  %CPU  %MEM    TIME+  COMMAND
 4920 maryna    20   0   15976    4480   3712  R   0,0   0,2   0:00.03 top
 4913 maryna    20   0   13988    5632   4096  S   0,0   0,3   0:00.00 bash
 4895 maryna    20   0   556124  52328  39848  S   0,0   2,6   0:00.29 gnome-t+
 4880 root       20   0      0      0      0  I   0,0   0,0   0:00.00 kworker+
 4879 root       20   0      0      0      0  I   0,0   0,0   0:00.00 kworker+
 4804 maryna    20   0  2799076  60296  45700  S   0,0   3,0   0:00.22 gjs
 4795 root       20   0      0      0      0  I   0,0   0,0   0:00.00 kworker+
 4371 maryna    20   0  497024   31872  23936  S   0,0   1,6   0:00.10 update-+
 3412 root       20   0  414448   48716  24020  S   0,0   2,4   0:00.37 fwupd
 3337 maryna    20   0  346848   19460  12164  S   0,0   1,0   0:00.08 xdg-des+
 3294 maryna    20   0   155008   2076   1792  S   0,0   0,1   0:00.04 VBoxCli+
 3293 maryna    20   0   22904    1436   1152  S   0,0   0,1   0:00.00 VBoxCli+
 3285 maryna    20   0   221708    3204   2816  S   0,0   0,2   0:01.29 VBoxCli+
 3284 maryna    20   0   22904    1540   1280  S   0,0   0,1   0:00.00 VBoxCli+
 3276 maryna    20   0   221192    3200   2688  S   0,0   0,2   0:00.29 VBoxCli+
 3275 maryna    20   0   22904    1536   1280  S   0,0   0,1   0:00.00 VBoxCli+
 3235 maryna    20   0  717684  178960  53988  S   0,0   8,9   0:02.57 snap-st+

```

Рисунок 7.15 – Команда top, сортування за номером

```

Fields Management for window 1:Def, whose current sort field is %CPU
Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
'd' or <Space> toggles display, 's' sets sort. Use 'q' or <Esc> to end!

* PID      = Process Id          PGRP    = Process Group      OOMs      = OOMEM Score c
* USER     = Effective User    TTY      = Controlling T       ENVIRON   = Environment v
* PR       = Priority          TPGID    = TTY Process G       vmj       = Major Faults
* NI       = Nice Value        SID       = Session Id          vMn       = Minor Faults
* VIRT     = Virtual Image     nTH       = Number of Thr       USED      = Res+Swap Size
* RES      = Resident Size     P         = Last Used CPU       nsIPC     = IPC namespace
* SHR      = Shared Memory     TIME      = CPU Time            nsMNT     = MNT namespace
* S        = Process Statu     SWAP      = Swapped Size       nsNET     = NET namespace
* %CPU     = CPU Usage         CODE      = Code Size (Ki     nsPID     = PID namespace
* %MEM     = Memory Usage     DATA     = Data+Stack (K    nsUSER    = USER namespac
* TIME+    = CPU Time, hun     nMaj     = Major Page Fa   nsUTS     = UTS namespace
* COMMAND  = Command Name/    nMin     = Minor Page Fa     LXC       = LXC container
PPID      = Parent Proces     nDRT     = Dirty Pages C   RSAn      = RES Anonymous
UID       = Effective User    WCHAN    = Sleeping in F      RSfd      = RES File-base
RUID      = Real User Id     Flags    = Task Flags <s     RSlk      = RES Locked (K
RUSER     = Real User Nam    CGROUPS  = Control Group     RSsh      = RES Shared (K
SUID      = Saved User Id    SUPGIDS  = Supp Groups I   CGNAME    = Control Group
SUSER     = Saved User Na    SUPGRPS  = Supp Groups N  NU        = Last Used NUM
GID       = Group Id         TGID     = Thread Group
GROUP     = Group Name       OOMa     = OOMEM Adjustm

```

Рисунок 7.16 – Команда top, меню виводу полів

Ключі команди **top**:

-d визначає затримку (в секундах) між оновленням списку процесів

-i ігнорувати процеси, які простоюють

-n показати n ітерацій і потім вийти

-s запуск команди в захищеному режимі

-u фільтрувати процеси, запущені від імені певного користувача

Hot-keys команди **top**

spacebar оновлення екрану

h вивід довідкової інформації

k завершити процес

q завершення роботи команди **top**

f додавання або видалення полів

o сортування порядку виведення інформації

\$ top -n 10 -s 3-o cpu // виведе таблицю з не більше ніж 10 рядків, в якій процеси будуть відсортовані за спаданням використовуваного ними процесорного часу, причому оновлюватися таблиця буде кожні 3 секунди.

Команда **htop** - це вдосконалена версія команди **top** (рисунк 7.17). За замовчуванням вона не встановлена в системі, але її можна встановити, виконавши команду "**sudo apt-get install htop**"



The screenshot shows the htop interface. At the top, system statistics are displayed: CPU usage at 1.4%, memory usage at 893M/1.93G, and swap usage at 91.4M/2.14G. System tasks are 135, with 238 threads and 1 running. Load averages are 0.00, 0.00, 0.00, and uptime is 04:25:14. Below this is a table of running processes with columns for PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The processes listed include the shell (PID 1522), htop (PID 6718), and various system services like systemd-journald, systemd-udevd, systemd-oomd, systemd-resolved, systemd-timedate, systemd-timesyncd, acpid, avahi-daemon, cron, dbus-daemon, and NetworkManager.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1522	maryna	20	0	3976M	320M	98416	S	2.7	16.2	2:40.15	/usr/bin/gnome-shell
6718	maryna	20	0	22520	4496	3600	R	1.4	0.2	0:00.13	htop
1558	maryna	20	0	3976M	320M	98416	S	0.7	16.2	0:03.92	/usr/bin/gnome-shell
1	root	20	0	164M	13000	7968	S	0.0	0.6	0:01.32	/sbin/init splash
224	root	19	-1	65160	23572	21956	S	0.0	1.2	0:00.32	/lib/systemd/systemd-journald
283	root	20	0	26604	6200	4140	S	0.0	0.3	0:00.08	/lib/systemd/systemd-udevd
426	systemd-o	20	0	14824	6044	5320	R	0.0	0.3	0:13.15	/lib/systemd/systemd-oomd
427	systemd-r	20	0	25392	12340	8948	S	0.0	0.6	0:00.19	/lib/systemd/systemd-resolved
428	systemd-t	20	0	89376	6500	5708	S	0.0	0.3	0:00.05	/lib/systemd/systemd-timesyncd
473	systemd-t	20	0	89376	6500	5708	S	0.0	0.3	0:00.00	/lib/systemd/systemd-timesyncd
606	root	20	0	245M	7416	6540	S	0.0	0.4	0:00.31	/usr/libexec/accounts-daemon
687	root	20	0	2812	1148	1068	S	0.0	0.1	0:00.25	/usr/sbin/acpid
610	avahi	20	0	7628	3324	2988	S	0.0	0.2	0:00.04	avahi-daemon: running [ma
611	root	20	0	20984	2540	2288	S	0.0	0.1	0:00.02	/usr/sbin/cron -f -P
612	messagebu	20	0	11100	6632	4812	S	0.0	0.3	0:00.70	dbus-daemon --system --a
614	root	20	0	338M	16412	13344	S	0.0	0.8	0:00.50	/usr/sbin/NetworkManager

Рисунок 7.17 – Команда **top**, меню виводу полів

Команда **ps** відображає список запущених процесів поточного користувача, з-під якого запускається команда (рисунок 7.18)..

```
maryna@maryna-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2861 pts/1        00:00:00 bash
 2873 pts/1        00:00:00 ps
```

Рисунок 7.18 – Команда ps

Для виведення списку процесів для всіх користувачів потрібно скористатися командою **ps -A** (рисунок 7.19)..

```
maryna@maryna-VirtualBox:/$ ps -A
  PID TTY          TIME CMD
    1 ?            00:00:08 systemd
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 rcu_gp
    4 ?            00:00:00 rcu_par_gp
    5 ?            00:00:01 kworker/0:0-events
    6 ?            00:00:00 kworker/0:0H-kblockd
    8 ?            00:00:00 kworker/u2:0-ext4-rsv
    9 ?            00:00:00 mm_percpu_wq
   10 ?            00:00:00 ksoftirqd/0
```

Рисунок 7.19 – Команда ps -A

Опції команди ps:

-d покаже системному адміністратору всі процеси, за винятком так званих «лідерів» (рисунок 7/20). Лідерство - це коли один процес запускає інші. При запуску процесу з даними ключем користувач не побачить перший сервіс, але йому покажуть залежні від нього.

```
maryna@maryna-VirtualBox:~$ ps -d
  PID TTY          TIME CMD
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 pool_workqueue_release
    4 ?            00:00:00 kworker/R-rcu_g
    5 ?            00:00:00 kworker/R-rcu_p
    6 ?            00:00:00 kworker/R-slub
```

Рисунок 7.20 – Команда ps -d

Ключі команди ps

- a вивід інформації про всі процеси, пов'язані з конкретним терміналом
- l детальний вивід (long format). Виводить пріоритет, ідентифікатор батьківського процесу і іншу інформацію (рисунок 7.21)

```
maryna@maryna-VirtualBox:~$ ps -l
F S  UID      PID     PPID  C  PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000     4913     4895  0   80   0  -  3502 do_wai pts/0        00:00:00 bash
0 T   1000     4920     4913  0   80   0  -  3994 do_sig pts/0        00:00:00 top
0 T   1000     5669     4913  0   80   0  -  3994 do_sig pts/0        00:00:00 top
0 R   1000     5679     4913  0   80   0  -  3883 -      pts/0        00:00:00 ps
maryna@maryna-VirtualBox:~$ ps -l -p 4920
F S  UID      PID     PPID  C  PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 T   1000     4920     4913  0   80   0  -  3994 do_sig pts/0        00:00:00 top
```

Рисунок 7.21 – Команда ps -l

- u користувацький вивід (user format). Виводить час старту процесу і ім'я користувача, що запустив процес
- x вивід процесів-демонів і інших процесів, запущених не в поточній сесії терміналу
- c вивід інформації про процеси, пов'язані з зазначеною командою.
- t Для відображення сервісів, які запущені в поточній термінальній сесії.
- r для перегляду тільки працюючих в даний момент процесів.
- e виведення інформації про всі процеси
- o задати формат виведення (рисунок 7.22)

```
maryna@maryna-VirtualBox:~$ ps -eo pid,ni,pri,cmd
PID  NI  PRI  CMD
1    0   19  /sbin/init splash
2    0   19  [kthreadd]
3    0   19  [pool_workqueue_release]
4  -20   39  [kworker/R-rcu_g]
5  -20   39  [kworker/R-rcu_p]
6  -20   39  [kworker/R-slab_1]
```

Рисунок 7.22 – Команда ps виведе вказані поля

Якщо потрібно відобразити інформацію про конкретний сервіс, то вказуємо його ідентифікатор (PID):

```
$ ps 1721
```

Якщо потрібно показати декілька процесів, то вказується PID кожного через кому.

Можна сортувати процеси - використовуючи ім'я групи. Для цього використовується ключ **-G** або **-Group**.

```
$ ps --group users
```

Пошук за ідентифікатором групи **-g** або **-group**.

Переглянути перелік процесів, які запустив конкретний користувач, використовуємо ключ **-U**.

```
$ ps -U name_user
```

Щоб отримати детальний опис, використовуємо ключ **-eF** (рисунок 7.23).

```
maryna@maryna-VirtualBox:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1         0  0  11:33 ?        00:00:01 /sbin/init splash
root           2         0  0  11:33 ?        00:00:00 [kthreadd]
root           3         2  0  11:33 ?        00:00:00 [rcu_gp]
```

Рисунок 7.23 – Команда ps -eF

Команда **pstree** відображає ієрархічну деревоподібну структуру всіх запущених процесів (рисунок 7.24):

Ключі команди pstree:

- a вивід аргументів командного рядка
- c відключити згортання ідентичних піддерев процесів
- p вивід ідентифікаторів процесів (PIDs)
- u вивід змін ідентифікаторів користувачів (відносно ідентифікатора батьківського процесу)
- V відображення інформації про версію.

Пріоритети процесів

У linux є два показники пріоритету процесу:

Ni -налаштування користувача

Pri -реальний пріоритет, який використовує ядро системи.

Значення пісе можуть змінюватися від -20 (найвищий пріоритет) до 19 (найнижчий пріоритет).

Значення ргі можуть змінюватися від 0 (найнижчий пріоритет) до 99 (найвищий пріоритет).

Всього—140 пріоритетів.

Значення пі впливає на ргі. Чим вищий пріоритет пі, тим вищий пріоритет буде ргі, тобто процесу буде надано більше процесорного часу.

Для процесів з більш високим пі пріоритетом (меншим значенням пісе, до -20) буде виділено більше системних ресурсів в порівнянні з іншими процесами з меншим пріоритетом (до 19), надаючи їм більше число циклів процесора.

Якщо пріоритет пі не заданий, кожен процес буде запускатися з пріоритетом за замовчуванням - 0 («базовим» пріоритетом розподілу машинного часу).

Всі користувачі, окрім root, можуть тільки понижувати пріоритет пі своїх власних процесів в діапазоні від 0 до 19.

Суперкористувач (root) для будь-якого процесу може встановити будь-яке значення пріоритету пі.

Команда **nice** дозволить встановити пріоритет планування процесу з точки зору використання процесора. те, як ядро буде планувати виділення процесорного часу для процесу або задачі (рисунок 7.27). За замовчуванням передбачається, що процесу надається рівний доступ до ресурсів процесора.

```
maryna@maryna-VirtualBox:~$ sleep 500 &
[1] 4417
maryna@maryna-VirtualBox:~$ ps -l
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     3728     3710  0  80   0 - 3497 do_wai pts/0      00:00:00 bash
0 S  1000     4417     3728  0  80   0 - 2804 hrtime pts/0      00:00:00 sleep
0 R  1000     4418     3728  0  80   0 - 3883 -      pts/0      00:00:00 ps

maryna@maryna-VirtualBox:~$ sudo nice --20 sleep &
[1] 6144
maryna@maryna-VirtualBox:~$ ps -l
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     6133     6115  0  80   0 - 3497 do_wai pts/1      00:00:00 bash
0 R  1000     6145     6133  0  80   0 - 3883 -      pts/1      00:00:00 ps

[1]+  Stopped                  sudo nice --20 sleep

maryna@maryna-VirtualBox:~$ sudo nohup nice --20 ping www.google.com &
[3] 6057
```

Рисунок 7.27 – Команда nice

Команда **renice** дозволяє нам змінити поточний пріоритет процесу, який вже запущений (рисунок 7.28). Команда **sleep** створює затримку виконання команд в системі та скриптів.

```
maryna@maryna-VirtualBox:~$ sleep 300 &
[3] 6153
maryna@maryna-VirtualBox:~$ ps -l
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000     6133     6115  0  80   0 - 3502 do_wai pts/1      00:00:00 bash
0 T  1000     6152     6133  0  80   0 - 2804 do_sig pts/1      00:00:00 sleep
0 S  1000     6153     6133  0  80   0 - 2804 hrtime pts/1      00:00:00 sleep
0 R  1000     6154     6133  0  80   0 - 3883 -      pts/1      00:00:00 ps

maryna@maryna-VirtualBox:~$ sudo renice -10 -p 6153
6153 (process ID) old priority 0, new priority -10
maryna@maryna-VirtualBox:~$ renice 10 -p 6153
6153 (process ID) old priority -10, new priority 10
maryna@maryna-VirtualBox:~$ renice -10 -p 6153
renice: failed to set priority for 6153 (process ID): Permission denied
maryna@maryna-VirtualBox:~$ sudo renice -10 -p 6153
6153 (process ID) old priority 10, new priority -10
maryna@maryna-VirtualBox:~$
```

Рисунок 7.28 – Команда renice

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №8 Обробка даних за допомогою мови shell. Скрипти в Linux. Планувальник задач в Linux

Мета: ознайомитись командною мовою Shell, навчитись писати скрипти, використовуючи мову Shell та знання, набуті протягом виконання попередніх робіт. Ознайомитись з планувальниками, їх характеристиками і особливостями застосування та навчитись налаштовувати виконання різних задач по адмініструванню і роботі з Linux за допомогою скриптів та планувальників.

Завдання

1. (по варіантам)

1) Написати скрипт, який видаляє всі порожні каталоги і виводить на екран повідомлення про кожен видалений каталог.

2) Написати скрипт пошуку однакових по їх вмісту файлів в двох каталогах, наприклад, Dir1 і Dir2. Користувач задає імена Dir1 і Dir2 в якості першого і другого параметру командного рядка. В результаті роботи програми файли, наявні в Dir1, порівнюються з файлами в Dir2 по їх вмісту. На екран виводяться число переглянутих файлів і результати порівняння.

3) Написати скрипт, що знаходить в заданому каталозі і всіх його підкаталогах всі файли, власником яких є заданий користувач. Ім'я власника і каталог задаються користувачем в якості першого і другого параметру командного рядка. Скрипт виводить результати в файл (третій аргумент командного рядка) у вигляді "повний шлях, ім'я файлу, його розмір".

4) Написати скрипт для пошуку файлів заданого розміру в заданому каталозі (ім'я каталогу задається користувачем в якості третього параметру командного рядка). Діапазон (min - max) розмірів файлів задається користувачем в якості першого і другого параметрів командного рядка.

5) Написати скрипт з використанням циклу `for`, що виводить на консоль розміри і права доступу для всіх файлів в заданому каталозі і всіх його підкаталогах (ім'я каталогу задається користувачем в якості першого параметру командного рядка).

6) Написати скрипт, який створює в зазначеному каталозі 3 нових каталоги. Поміщає в створені каталоги файли з вихідного каталогу: в перший - файли, що містять один рядок із заданим словом, у другій - файли з двома такими рядками, в третій - з трьома. Імена всіх файлів, які не включені в нові каталоги, виводить на екран.

7) Написати скрипт, який обчислює значення арифметичного виразу, заданого першими 7 параметрами, порівнює отримане значення з числом, що вводиться при виконанні скрипта, при збігу результатів порівняння виводить на екран заданий вираз і його значення

8) Написати скрипт, читає вміст першого файлу, переданого в якості першого параметра, читає вміст другого файлу, переданого в якості другого параметра, якщо число рядків в першому і другому файлах однакове, то виводить на екран кожні 5 секунд по черзі рядки з першого і другого файлів

9) Написати скрипт, який читає вміст першого файлу, переданого в якості першого параметра, читає вміст другого файлу, переданого в якості другого параметра, виводить на екран кожні 7 секунд по черзі 2 рядки з першого і 1 рядок з другого файлу, переміщаючись по файлах циклічно

10) Написати скрипт, який читає вміст файлу, що передається в якості першого параметра (в файлі в кожному рядку представлено одне ціле число), перевіряє правильність формату вмісту файлу, при помилках виводить відповідні повідомлення і завершує роботу, підраховує суму всіх розміщених у файлі чисел, виводить на екран отриману суму

11) Написати скрипт, який вводять передану в якості 1-го параметра кількість символічних рядків, в кожному введеному рядку шукає підрядок,

переданий в якості другого параметра, замінює кожен знайдений підрядок на рядок, переданий в якості третього параметра, виводить на екран кожен введений рядок і відповідний йому новий рядок.

12) Написати скрипт, що виводить в файл (назва файлу задається користувачем в якості першого параметру командного рядка) імена всіх файлів із заданим розширенням (третій параметр командного рядка) з заданого каталогу (ім'я каталогу задається користувачем в якості другого параметру командного рядка).

13) Написати скрипт, що вводить кілька коротких чисел у вигляді параметрів, підраховує їх суму, результат виводить на екран.

14) Написати скрипт, який аналізує вказаний параметром каталог і виводить на екран число файлів різного типу (звичайні, директорія, приховані). Тип задається параметром.

15) Написати скрипт, який видаляє з заданого першим параметром каталогу і всіх підкаталогів файли, дата останньої модифікації яких передують поточну дату мінус число днів, передане в якості другого параметра.

16) Написати скрипт, який створює новий підкаталог і копіює туди з батьківського каталогу файли заданого параметром типу.

17) Написати скрипт, що запитує послідовне введення декількох чисел із знаками і виводить на екран два списки чисел – додатних і від'ємних

18) Написати скрипт, який у заданому першим параметром каталозі знаходить всі файли, число посилань на які більше 3, і видаляє їх.

2. Налаштувати автоматичний запуск скрипта в заданий час

Теоретичні відомості

Командна мова shell є мовою програмування дуже високого рівня [8]. Цією мовою користувач здійснює управління комп'ютером.

Програмування на мові Shell відбувається у вікні терміналу, а самі операції виконуються відразу, як тільки натискається клавіша Enter після введення тексту функції.

Для того щоб, працюючи в терміналі, переключитися в середовище Shell, необхідно ввести команду **sh**. Ознакою того, що оболонка (shell) готова до прийому команд, є на екрані знак "\$" (рисунок 8.1).

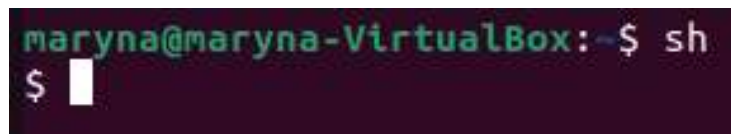


Рисунок 8.1– Запрошення в середовищі Shell

Як і в інших мовах програмування, в ній використовуються змінні для зберігання інформації.

Ім'я shell-змінної - це послідовність літер, цифр і підкреслень.

Значення shell-змінної - рядок символів.

Для присвоювання значень змінним може використовуватися оператор присвоювання "=".

var_1=13 - "13" - це НЕ число, а рядок з двох цифр.

var_2="ОС UBUNTU" - тут подвійні лапки ("") необхідні, так як в рядку є пробіл.

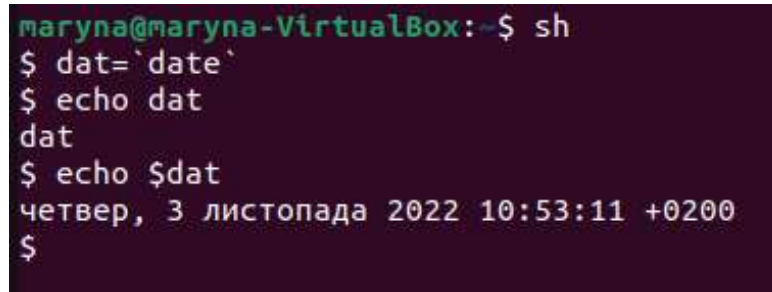
ВАЖЛИВО: як змінна, так і її значення повинні бути записані без пробілів відносно символу "=".

Можливі й інші способи присвоювання значень shell змінним:

\$ dat=`date`

Спочатку виконується команда "date" (зворотні лапки говорять про те, що спочатку повинна бути виконана укладена в них команда), а результат її

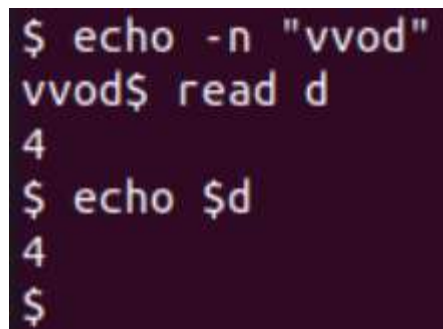
виконання замість видачі на стандартний вивід приписується як значення змінної, в даному випадку **"dat"** (рисунок 8.2).



```
maryna@maryna-VirtualBox:~$ sh
$ dat=`date`
$ echo dat
dat
$ echo $dat
четвер, 3 листопада 2022 10:53:11 +0200
$
```

Рисунок 8.2– Значенням змінної є результат виконання команди

Можна присвоїти значення змінної і за допомогою команди **read**, яка забезпечує прийом значення змінної з клавіатури/дисплея в діалоговому режимі (рисунок 8.3). Зазвичай команді **read** в командному файлі передуює команда **"echo"**, яка дозволяє попередньо видати якесь повідомлення на екран.



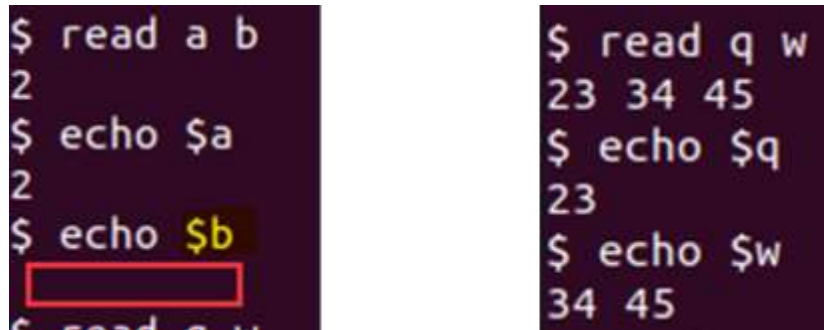
```
$ echo -n "vvod"
vvod$ read d
4
$ echo $d
4
$
```

Рисунок 8.3 – Текст введений з клавіатури, присвоєно змінній за допомогою команди **read**

При виконанні цього фрагмента командного файлу, після виведення на екран повідомлення «vvod», інтерпретатор зупиниться і буде чекати введення значення з клавіатури. Те число, яке користувач введе з клавіатури, стане значенням змінної **"d"**.

Одна команда **read** може прочитати (присвоїти) значення відразу для декількох змінних. Якщо змінних в команді **read** більше, ніж їх введено (через

пробіл), тим які залишилися - присвоюється порожній рядок (рисунок 8.4). Якщо переданих значень більше, ніж змінних в команді read, то зайве запишеться в останню змінну.

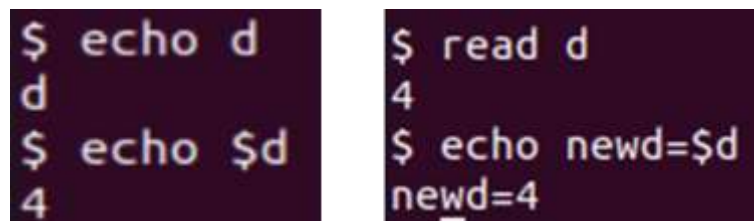


```
$ read a b
2
$ echo $a
2
$ echo $b
$ read q w
23 34 45
$ echo $q
23
$ echo $w
34 45
```

Рисунок 8.4 – Команда read

Інтерпретатор для продовження роботи чекає лише натискання клавіші. Введене вами число сприймається їм НЕ як число, а як послідовність символів (!). Інтерпретатор не перевіряє, що ви ввели. Тому в якості значення змінної може бути введено будь-що або порожній рядок.

При зверненні до shell-змінної необхідно перед ім'ям ставити символ "\$" для отримання значення цієї змінної (рисунок 8.5). Команда echo може за один раз виводити на екран кілька змінних.



```
$ echo d
d
$ echo $d
4
$ read d
4
$ echo newd=$d
newd=4
newd=4
```

Рисунок 8.5 – Звернення до змінної

Команда **set** встановлює значення позиційних параметрів (рисунок 8.6). Це буває дуже зручно. Позиційний параметр - це змінна у програмі оболонки і на її значення посилаються як \${N}, де N – цифра, яка означає позицію параметра.

```
maryna@maryna-VirtualBox:~$ date
четвер, 31 жовтня 2024 15:25:55 +0200
maryna@maryna-VirtualBox:~$ set `date`
maryna@maryna-VirtualBox:~$ echo $1 $3 $5
четвер, жовтня 15:26:07
```

Рисунок 8.6 – Команда set

Команда date видає на екран поточну дату виведе на екран (рисунок 8.7).

```
четвер, жовтня 15:26:07
```

Рисунок 8.7 – Результат команди date

Команда set дозволяє також здійснювати контроль над виконанням програми, наприклад:

- set -v** в термінал виводяться рядки, які зчитує shell;
- set + v** скасовує попередній режим;
- set -x** в термінал виводяться команди перед виконанням;
- set + x** скасовує попередній режим.

Команда set без параметрів виводить в термінал стан програмного середовища (рисунок 8.8).

```
$ set
set
+ set
COLORTERM='truecolor'
DBUS_SESSION_BUS_ADDRESS='unix:path=/run/user/1000/bus'
DESKTOP_SESSION='ubuntu'
DISPLAY=':0'
```

Рисунок 8.8 – Стан програмного середовища

Обчислення арифметичних виразів

Можна обчислювати вирази, укладені або в квадратні дужки, або в подвійні круглі дужки. Перед дужками повинен стояти символ \$, а результати

виразів можна передавати, як аргумент будь-якій команді або призначати змінній (рисунок 8.9).

```
$ echo $((10*10))
echo $((10*10))
+ echo 100
100
$
```

Рисунок 8.9 – Обчислення арифметичних виразів

Команда `echo` виводить у стандартний потік виведення рядок, зазначений в якості аргументу.

Наступний приклад демонструє присвоєння результату арифметичного виразу змінній (рисунок 8.10):

```
$ a=5
a=5
+ a=5
$ b=$((a/2))
b=$((a/2))
+ b=2
$ echo $b
echo $b
+ echo 2
2
$
```

Рисунок 8.10 – Присвоєння змінній результату арифметичного виразу

Тут змінній `a` присвоєно значення 5. Далі проводиться присвоєння змінній `b` результаті ділення значення `a` на 2. В силу того, що в `bash` можливе обчислення тільки цілочисельних виразів, то в результаті значенням `b` стало число 2.

Перевірка заданих умов

Команда **test** ("["]")

Команда **test** перевіряє виконання умови. З використанням цієї (вбудованої) команди формуються оператори вибору і циклу мови shell.

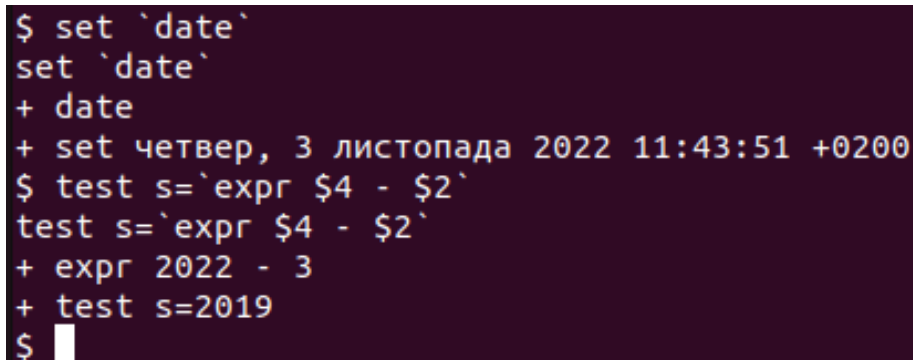
Два можливих формати команди:

test умова або **[умова]**

Пробіли повинні бути і між значеннями і символом порівняння або операції (як і в команді **expr**).

Команда **expr** дає програмі змогу сприймати ту чи іншу змінну саме як число. В shell тип даних визначається після того, як значення було записано в змінну і може змінюватися в тілі програми.

Приклад роботи з командами test та expr (рисунок 8.11)



```
$ set `date`
set `date`
+ date
+ set четвер, 3 листопада 2022 11:43:51 +0200
$ test s=`expr $4 - $2`
test s=`expr $4 - $2`
+ expr 2022 - 3
+ test s=2019
$
```

Рисунок 8.11 – Перевірка виконання умови

Види перевірок, які виконуються командою:

- перевірка файлів на предмет виконання заданих умов;
- порівняння файлів;
- перевірка установки опцій оболонки;
- порівняння рядків;
- порівняння цілих чисел.

Якщо тест виконаний успішно, команда test передає нульовий код повернення. В іншому випадку - ненульовий.

Використовуючи опцію `-e` команди `test`, можна перевірити існування файлу (рисунок 8.12)

```
$ test -e /etc/passwd
test -e /etc/passwd
+ test -e /etc/passwd
$ echo $?
echo $?
+ echo 0
0
$ test -e not_existent_file
test -e not_existent_file
+ test -e not_existent_file
$ echo $?
echo $?
+ echo 1
1
```

Рисунок 8.12 – Приклад перевірки існування файлу

Щоб дізнатися, яке значення повернула команда **test**, використовувався спеціальний параметр `$?`. У першому випадку команда `test` повернула нульовий код повернення, оскільки файл `/etc/passwd` існує. У другому випадку команда повернула код помилки.

Команда `test` в скриптах (сценаріях) зазвичай викликається в іншій формі.

У цій формі замість рядка **test** вказують квадратні дужки і умова в них: **[умова]**.

Попередній приклад, може бути вирішений іншим способом (рисунок 8.13):

команда `[ -e /etc/passwd ]` еквівалентна команді `test -e /etc/passwd`.

```
$ test -e /etc/passwd
test -e /etc/passwd
+ test -e /etc/passwd
$ echo $?
echo $?
+ echo 0
0
```

```
$ [ -e /etc/passwd ]
[ -e /etc/passwd ]
+ [ -e /etc/passwd ]
$ echo $?
echo $?
+ echo 0
0
```

Рисунок 8.13 – Команда test / [умова]

Порівняння цілих чисел проводиться за допомогою наступних опцій команди test:

- eq рівність;
- ne нерівність;
- lt менше;
- le менше або дорівнює;
- gt більше;
- ge більше або дорівнює.

Приклад порівняння чисел (рисунок 8.14):

```
$ [ 1 -lt 2 ]
[ 1 -lt 2 ]
+ [ 1 -lt 2 ]
$ echo $?
echo $?
+ echo 0
0
$ [ 1 -eq 2 ]
[ 1 -eq 2 ]
+ [ 1 -eq 2 ]
$ echo $?
echo $?
+ echo 1
1
```

Рисунок 8.14 – Порівняння чисел

У shell використовуються умови різних типів"

Умовний оператор **if**

У загальному випадку оператор "if" має структуру

if умова; echo \$0

then список

elif умова

then список

else список

fi

\$0 може використовуватись для виводу повідомлень (умова true).

elif, скорочений варіант від **else if**, може бути використаний поряд з повним, тобто допускається вкладення довільного числа операторів **if** (як і інших операторів). «список» в кожному випадку повинен бути осмислений.

Конструкції, не є обов'язковими:

elif умова then список

та

else список

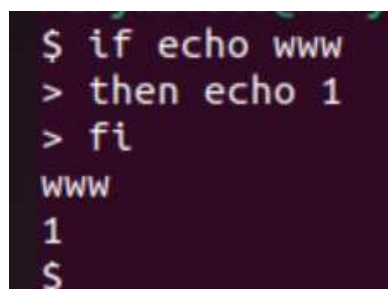
Коротка структура цього оператора

if умова

then список

fi

Приклад 1 використання оператора if (рисунок 8.15):



```
$ if echo www
> then echo 1
> fi
www
1
$
```

Рисунок 8.15 – Використання оператора if

Коротка структура цього оператора

if умова

then список

fi

Структура обов'язково завершується службовим словом **fi**.

Приклад 2 використання оператора if (рисунок 8.16):



```
четвер, 3 листопада 2022 12:50:25 +0200
$ if [ $1 -gt $2 ]
> then echo 7
> else echo hello
> fi
sh: 14: [: Illegal number: четвер,
hello
$ if [ $4 -gt $2 ]
> then echo yes
> else echo no
> fi
yes

$ if [ $4 -gt $2 ]
> then echo $0: yes
> else echo no
> fi
sh: yes
$ if [ $2 -gt $2 ]
> then echo yes
> else echo $0: false
> fi
sh: false
```

Рисунок 8.16 – Використання оператора if

Список логічних операторів, які використовуються для if | then | else:

- z - рядок порожній;
- n - рядок не порожній;
- =, (==) - рядки рівні;
- != - рядки нерівні;
- eq – рівно;
- ne – нерівно;
- lt, (<) – менше;
- le, (<=) - менше або дорівнює;
- gt, (>) – більше;
- ge, (>=) - більше або дорівнює;
- ! - заперечення логічного виразу;

- a, (&&) - логічне «І»;
- o, (||) - логічне «АБО».

Оператор вибору case

```
case $змінна in
    шаблон-1)
        список команд
        ;;
    шаблон -2)
        список команд
        ;;
    шаблон -3| шаблон -4)
        список команд
        ;;
    шаблон -N)
        список команд
        ;;
    *)
        список команд
        ;;
esac
```

Тут **case**, **in** і **esac** - службові слова. «змінна» (це може бути і один символ) порівнюється з «шаблоном». Потім виконується «список команд». Службове слово **esac** необхідне для завершення структури.

« ; » в кінці рядків вибору, але написати тут « ; » було б помилкою. Для кожної альтернативи може бути виконано кілька команд. Якщо ці команди будуть записані в один рядок, то символ « ; » буде використовуватися як роздільник команд.

Зазвичай останній рядок вибору має шаблон « * », що в структурі **case** означає «будь-яке значення». Цей рядок вибирається, якщо не відбувся збіг

значення змінної з жодним раніше записаним шаблоном, обмежених дужкою «)».

Значення проглядаються в порядку запису.

Приклад 1 **case**:

echo -n "Тест:"

read z

case \$z in

5) echo 111 ;;

4) echo 222 ;;

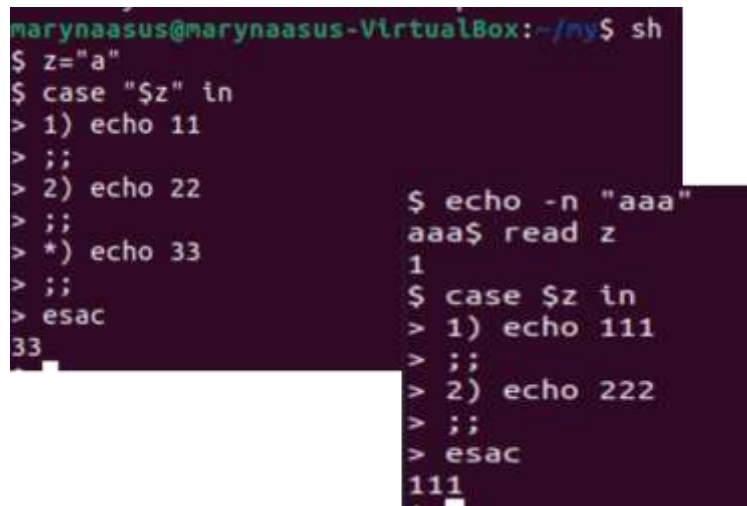
3) echo 333 ;;

2) echo 444 ;;

***) echo 555 ;;**

esac

Приклад 2 **case** (рисунок 8.17):



```
marynaasus@marynaasus-VirtualBox:~/my$ sh
$ z="a"
$ case "$z" in
> 1) echo 11
> ;;
> 2) echo 22
> ;;
> *) echo 33
> ;;
> esac
11

$ echo -n "aaa"
aaa$ read z
1
$ case $z in
> 1) echo 111
> ;;
> 2) echo 222
> ;;
> esac
222
```

Рисунок 8.17 – Приклад використання оператора **case**

Оператор циклу з перерахуванням **for**

for ім'я in список значень

do

список команд

done

де **for** - службове слово що визначає тип циклу, **do i done** - службові слова, що виділяють тіло циклу. Фрагмент in список значень" може бути відсутнім.

Приклад роботи з оператором for (рисунок 8.18):

```
$ $ a=1
$ b=2
$ c=3
$ for i in $a $b $c
> do
> echo "ffff"
> done
ffff
ffff
ffff
$
```

Рисунок 8.18 – Приклад роботи з оператором for

Оператор циклу з істинним умовою **while**

Структура while, також забезпечує виконання розрахунків, краще тоді, коли невідомий заздалегідь точний список значень параметрів або цей список повинен бути отриманий в результаті обчислень в циклі.

while умова

do

список команд

done

де **while** - службове слово що визначає тип циклу з істинною умовою. Список команд в тілі циклу (між **do i done**) повторюється до тих пір, поки зберігається істинність умови (тобто код завершення останньої команди в тілі циклу дорівнює «0») або цикл не буде перерваний зсередини спеціальними командами (**break**, **continue** або **exit**). При першому вході в цикл умова має виконуватися.

Приклад: обчислення виразу $a = \text{`expr $a + 1`}$ в циклі, тобто при кожній ітерації значення **a** збільшується на 1 (рисунок 8.19).

```
$ a=1
$ b=6
$ while [ $a -lt $b ]
> do
> a=`expr $a + 1`
> echo $a
> done
2
3
4
5
6
$
```

Рисунок 8.19 – Приклад роботи з оператором **while**

Оператор циклу з помилковою умовою **until** (рисунок 8.20)

until умова

do

список команд

done

де **until** - службове слово що визначає тип циклу з хибною умовою. Список команд в тілі циклу (між **do** і **done**) повторюється до тих пір, поки зберігається хибність умови або цикл не буде перерваний зсередини спеціальними командами (**break**, **continue** або **exit**). При першому вході в цикл умова не повинно виконуватися.

Відмінність від оператора **while** полягає в тому, що умова циклу перевіряється на неправдивість (на ненульовий код завершення останньої команди тіла циклу) перевіряється ПІСЛЯ кожного (в тому числі і першого!) виконання команд тіла циклу.

Приклад:

```
$ until false
> do
> read z
> if [ $z=5 ] ; echo $0
> then echo enough ; break
> else echo more
> fi
> done
```

Рисунок 8.20 – Приклад роботи з оператором until

Програма з нескінченним циклом чекає введення слів (повторюючи на слово «more»), поки не буде введено «5». Після цього видається «enough» і команда «break» припиняє виконання циклу.

Функції Shell

У shell терміни «процедура» та «функція» використовуються як синоніми.

Процедури (або скрипти) shell - це тільки один спосіб згрупувати команди під одним ім'ям (ім'я файлу з набором команд) .

Функції shell також є серіями команд під одним ім'ям, але вони зберігаються в середовищі активного процесу shell.

Процедури/функції shell виконуються, коли ви набираєте їх ім'я і вони дозволяють вам не набирати повторно одні і ті ж команди.

Проте, процедури shell відрізняються від функцій shell.

Процедура shell зберігається в файлі, батьківський shell інтерпретує її командні рядки, і процес нащадок виконує їх. А функція вже визначена для процесу shell, так що той же shell, який інтерпретує функцію і виконує її.

У роботі з функціями не потрібно створювати процес нащадок shell. З цієї точки зору функції більш ефективні.

Так як функції shell доступні тільки для поточного shell, вони існують тільки поки існує shell. Ви не може експортувати функції; замість цього ви повинні перевизначити функцію для іншого shell, який її потребує.

Розглянемо два способи написання функцій.

З використанням службового слова function

function ім'я_функції

{

команда1

команда2

}

без використання службового слова function

ім'я_функції()

{

команда1

команда2

}

Щоб виконати функцію, потрібно набрати її ім'я (**ім'я_функції**)

Щоб мати постійний доступ до функції потрібно визначити її в файлі користувача **profile**.

Припустимо ви хочете змінити значення вашої змінної PS1, щоб відобразити поточний каталог. Процедура shell не може зробити цього, тому що її буде виконувати процес нащадок. Коли завершується процес нащадок, змінна PS1 змінюється. У цій ситуації вам потрібна функція.

Приклад створення і виклику функції:

My_fn()

{

echo "my function"

}

My_fn # виклик

Результат:

my function

Функції в shell можуть приймати параметри.

Вони передаються як аргументи при виклику і доступні всередині функції через спеціальні змінні: \$1, \$2, \$3 і так далі.

\$1, \$2, \$3, ... - позиційні аргументи

\$@ - всі аргументи, які передаються як окремі рядки

\$* - всі аргументи, які передаються як один рядок

\$# - кількість аргументів, які передаються

Приклад:

My_fn()

```
{  
    echo "my function $1, my function $2"  
    echo "How many my functions? $#"  
}
```

My_fn first second // виклик

Результат:

my function first, my function second

How many my functions? 2

Змінні у функціях

За замовчуванням, змінні, оголошені всередині функції, є глобальними і доступні в усьому скрипті.

Щоб обмежити область видимості змінної лише функцією, її потрібно змінну оголосити зі словом local.

```
var1 = "global"
```

```
My_fn()
```

```
{
```

```
    local var1 = "local"
```

```
    echo "$var1"
```

```
    echo "$var2"
```

```
}
```

```
My_fn # виклик
```

```
echo "$var1"
```

```
echo "$var2" // тут буде помилка, бо змінна локальна
```

Результат:

```
global
```

```
Local
```

```
global
```

(помилка або пустий рядок)

Функції можуть повертати результат за допомогою команди **return**, але на значення, а код завершення. Код 0 означає успішне виконання, а будь-яке інше число - помилку. Цей код можна отримати зі змінної \$? після виклику функції.

Для повернення текстового результату можна використати команди **echo**.

Функції можуть повертати результат за допомогою команди **return**, але на значення, а код завершення. Код 0 означає успішне виконання, а будь-яке інше число - помилку. Цей код можна отримати зі змінної \$? після виклику функції.

Для повернення текстового результату можна використати команди **echo**.

```
My_fn()
```

```
{
```

```
    if [ 1 -lt 2 ]; then
```

```
        return 0 // успіх
```

```
    else return 1 // помилка
```

```

    fi
}
My_fn // виклик
if [ $? -eq 0 ]; then
    echo "1 менше 2"
else
    echo " 1 не менше 2"
fi

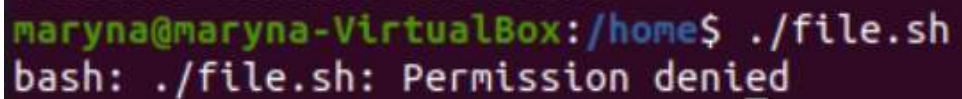
```

Скрипти

Скрипти в ОС на базі Linux - це набір системних та вбудованих команд, записаних в файл. Це робиться з метою швидкого і зручного виклику послідовності цих команд. Скрипти можуть виконувати найрізноманітніші завдання - від автоматизації рутинних дій системного адміністратора до реалізації складних алгоритмів для ІТ інфраструктури. При цьому результати роботи команд можуть служити вхідними даними для інших завдань (команд).

Для того, щоб перетворити файл з набором команд в скрипт, знадобляться **дві речі**. **По-перше**, в першому рядку порожнього файлу необхідно прописати шлях до командної оболонки, яка буде виконувати даний файл. У більшості дистрибутивів Linux даною оболонкою є Bash, і відповідно першим рядком буде **#!/bin/bash**. В інших рядках символ решітки використовується для позначення коментарів, які не будуть оброблені оболонкою. Однак, перший рядок - це особливий випадок, тут решітка, за якою слідує знак оклику (цю послідовність називають шебанг) і шлях **/bin/bash**, вказують системі що даний скрипт створений саме для оболонки bash. **По-друге**, щоб скрипт запустити як виконуваний файл, йому необхідно дати права на виконання, інакше, спроба його запуску дасть помилку Permission denied.

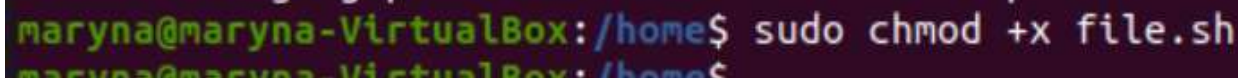
Приклад помилки:



```
maryna@maryna-VirtualBox:/home$ ./file.sh
bash: ./file.sh: Permission denied
```

Рисунок 8.21 – Нема прав на виконання файлу

Права на виконання даються командою `chmod`:



```
maryna@maryna-VirtualBox:/home$ sudo chmod +x file.sh
maryna@maryna-VirtualBox:/home$
```

Рисунок 8.22 – Додавання прав на виконання для файлу

Дана команда дає всім користувачам операційної системи право на запуск файлу з ім'ям `file.sh`.

Право на запуск того ж файлу тільки власнику можна дати за допомогою такої команди: **`chmod u + x file.sh`**.

Розширення `.sh` не обов'язково, але таким чином легше визначати, який файл є скриптом.

Скрипти оболонки можуть бути виконані двома різними шляхами:

- якщо файл сценарію доступний для читання, то ім'я файлу сценарію вказується в якості аргументу командного рядка при явному запуску виконуваного файлу оболонки.

- якщо файл скрипта доступний для читання і виконання, то скрипт можна запустити так само, як і звичайні системні команди, наприклад,

Сценарій створюється за допомогою команди **`cat`**.

Наприклад, створимо сценарій, що містить всього один рядок і обчислює добуток

```
$ cat > myscr1.sh
```

```
echo $ ((10 * 20))
```

Після закінчення введення коду сценарію, треба натиснути комбінацію клавіш **Ctrl+D (Cтр+Z)**. Та ж команда, але без символу перенаправлення виведення «>», дозволяє переглянути код сценарію:

```
$ cat myscr1.sh
```

```
echo $ ((10 * 20))
```

В **bash** використовуються наступні спеціальні параметри:

\$* - рядок, складений зі значень всіх аргументів командного рядка;

\$@ - містить рядок, який складається зі значень всіх аргументів командного рядка, розділених пробілами (аналогічно, але не тотожно **\$***);

\$# - кількість аргументів командного рядка;

\$? - код повернення попередньої команди;

\$\$ - PID оболонки

Найбільш поширений спосіб передачі даних сценаріями полягає у використанні параметрів командного рядка. Викликавши сценарій з параметрами, ми передаємо йому інформацію, з якою він може працювати.

```
$ ./script1 2 4
```

сценарію передано два параметри - "2" і "4".

Оболонка **bash** призначає спеціальним змінним, що називаються позиційними параметрами, параметри командного рядка, які введені при виклику скрипта:

\$0 – ім'я скрипта, **\$1** – перший параметр, **\$2** - другий параметр - і так далі, аж до змінної **\$9**, в яку потрапляє дев'ятий параметр.

Ось як можна використовувати параметри командного рядка у скрипті за допомогою цих змінних:

```
maryna@maryna-VirtualBox:~$ cat script1
#!/bin/bash
echo $0
echo $1
echo $2
```

Рисунок 8.23 –Параметри, як змінні

Запустимо скрипт із параметрами:

```
maryna@maryna-VirtualBox:~$ ./script1 2 4
./script1
2
4
maryna@maryna-VirtualBox:~$ ./script1 2
./script1
2
maryna@maryna-VirtualBox:~$
```

Рисунок 8.24 – Запуск скриптів з параметрами

Приклад використання параметрів: знайдемо суму чисел, переданих сценарієм (рисунок 8.25):

```
maryna@maryna-VirtualBox:~$ cat script1
#!/bin/sh

res=$(( $1 + $2 ))
echo first param $1
echo second param $2
echo Result $res

maryna@maryna-VirtualBox:~$ ./script1 3 5
first param 3
second param 5
Result 8
```

Рисунок 8.25 – Приклад використання параметрів

Параметри командного рядка не обов'язково мають бути числами. Сценарієм можна передавати рядки (рисунок 8.26):

```
maryna@maryna-VirtualBox:~$ cat script2
#!/bin/sh

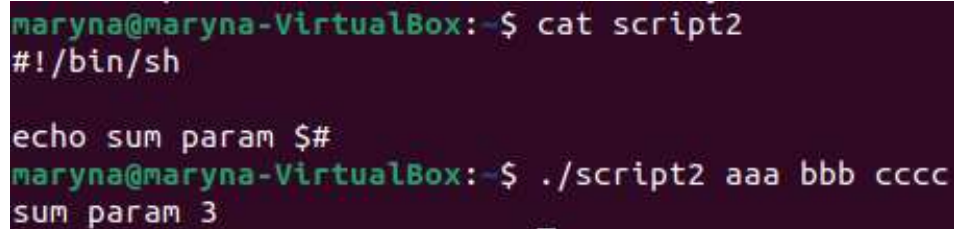
echo first param $1
maryna@maryna-VirtualBox:~$ ./script2 aaa
first param aaa
```

Рисунок 8.26 – Параметрами можуть бути і рядки

Якщо параметр містить пробіли, а потрібно обробляти його як самостійний фрагмент даних, то застосовуються лапки.

Якщо скрипту потрібно більше дев'яти параметрів, при зверненні до них номер у імені змінної треба укласти у фігурні дужки: **`${10}`**

Приклад: підрахунок кількості переданих параметрів (рисунок 2.27):

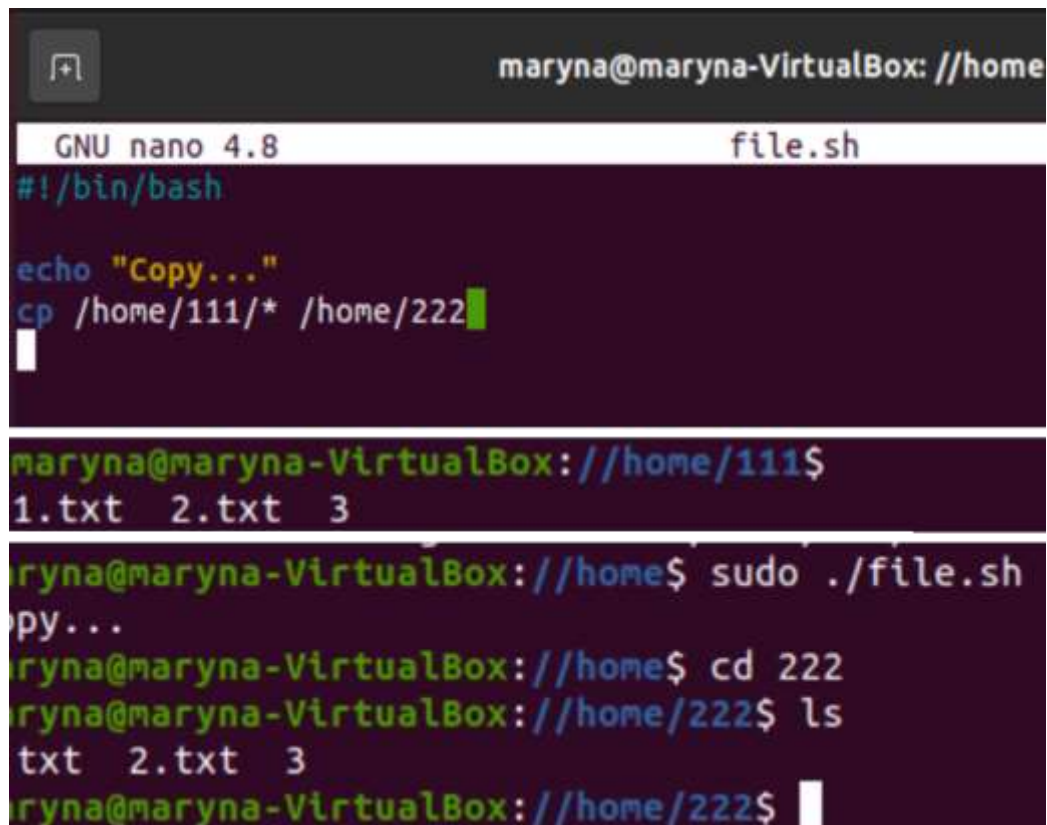


```
maryna@maryna-VirtualBox:~$ cat script2
#!/bin/sh

echo sum param $#
maryna@maryna-VirtualBox:~$ ./script2 aaa bbb cccc
sum param 3
```

Рисунок 8.27 – Підрахунок кількості параметрів

Приклад простого скрипта, що копіює всі файли з папки /home/111 в папку /home/222 (рисунок 2.28):



```
maryna@maryna-VirtualBox: //home
GNU nano 4.8 file.sh
#!/bin/bash

echo "Copy..."
cp /home/111/* /home/222

maryna@maryna-VirtualBox: //home/111$
1.txt 2.txt 3

maryna@maryna-VirtualBox: //home$ sudo ./file.sh
Copy...
maryna@maryna-VirtualBox: //home$ cd 222
maryna@maryna-VirtualBox: //home/222$ ls
1.txt 2.txt 3
maryna@maryna-VirtualBox: //home/222$
```

Рисунок 8.28 – Копіювання файлів

При роботі зі скриптами можуть використовуватися змінні. Вони дозволяють зберігати в файлі сценарію різну інформацію, наприклад, результати роботи одних команд для подальшого їх використання для інших команд.

Виконання окремих команд без зберігання результатів роботи обмежують їхні можливості.

Існують два типи змінних, які використовуються в bash-скриптах:

змінні середовища - використовуються при необхідності роботи з системними даними в командах оболонки.

Найбільш важливі змінні оточення і їх вміст:

HOME - шлях до домашнього каталогу користувача;

LOGNAME і USER - ім'я користувача;

MAIL - шлях до поштової скриньки користувача;

PATH - шлях пошуку файлів, і значення цієї змінної змінюється при кожному запуску програми cd.

PS1 - вид запрошення оболонки;

PWD - ім'я поточного каталогу;

OLDPWD - ім'я попереднього каталогу;

HOSTNAME - ім'я хоста;

SHLVL - номер оболонки і т.д.

Дуже часто зміна значення будь-якої змінної оточення призводить до зміни поведінки програми, що запускається.

Наприклад, якщо значення змінної HOME встановлено невірно, то команда cd ~ не повертатиме в домашній каталог.

echo "current user: \$HOME"

Можна використовувати **\$HOME** в подвійних лапках, що не завадить її розпізнавання системою.

Змінна оточення **PS1** містить вид запрошення, яке bash виводить, коли очікує введення чергової команди користувачем.

При бажанні вид запрошення, а точніше значення змінної оточення PS1 можна змінити (рисунок 8.29).

При цьому можна використовувати як будь-які символи, що вводяться з клавіатури, так і деяке число спеціальних символів, які при формуванні рядка запрошення декодуються відповідно до:

`\d` дата в форматі «день, місяць, число», наприклад, Срд Жов 18

`\u` ім'я користувача

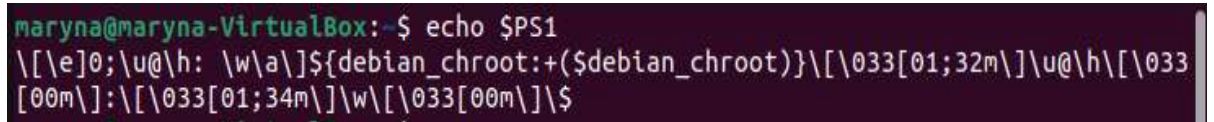
`\t` поточний час в 24-годинному форматі: hh: mm: ss

`\\` зворотний слеш

`\$` символ #, якщо оболонка запущена суперкористувачем, і символ \$, якщо оболонка запущена звичайним користувачем

`\w` повне ім'я поточного робочого каталогу (починаючи з кореня)

`\W` поточний робочий каталог (без вказівки шляху)

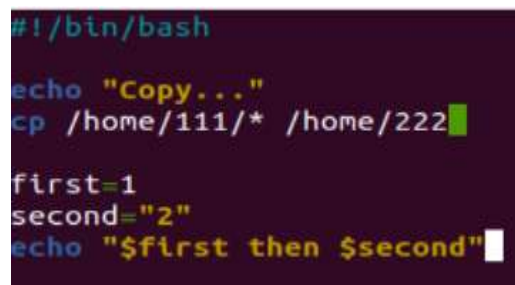
A terminal window with a dark background. The prompt is 'maryna@maryna-VirtualBox:~\$'. The command 'echo \$PS1' has been entered, and the output is a complex escape sequence: '\[\e]0;\u@\h: \w\a\>\${debian_chroot:+(\$debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\\$'.

```
maryna@maryna-VirtualBox:~$ echo $PS1
\[\e]0;\u@\h: \w\a\>${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
```

Рисунок 8.29 – Налаштування запрошення

Користувацькі змінні

bash-скрипти дозволяють задавати і використовувати в скрипті свої власні змінні. Подібні змінні зберігають значення до тих пір, поки не завершиться виконання сценарію. Як і у випадку з системними змінними, до призначених для користувача змінних звертаються використовуючи знак долара \$ (рисунок 8.30).

A terminal window showing a bash script. The script sets 'first=1' and 'second="2"', then echoes '\$first then \$second'.

```
#!/bin/bash

echo "Copy..."
cp /home/111/* /home/222

first=1
second="2"
echo "$first then $second"
```

Рисунок 8.30 – Користувацькі змінні в скрипті

Ім'я змінної може бути словом, що складається з алфавітних символів, цифр і знаків підкреслення (тільки перший символ цього слова не може бути цифрою), а також число або один з наступних спеціальних символів: *, @, #, ?, -, \$, !, 0, _.

Доповнимо відповідним чином скрипт, написаний раніше - додається змінна для зберігання шляху до каталогу, в який здійснюється копіювання (рисунок 8.31).



```
GNU nano 4.8 file.sh
#!/bin/bash

echo "Copy..."
cp /home/111/* /home/222

#!/bin/bash

echo "Copy..."
our_dir="/home/111"
cp /home/222/* $our_dir
```

Рисунок 8.31 – Користувацькі змінні в скрипті

Планувальник задач в Linux

Часто виникають ситуації, в яких доводиться автоматизувати різні завдання по обслуговуванню і роботі з Linux за допомогою скриптів. У цьому випадку зручно, якщо скрипт виконує необхідні команди без участі користувача. Для цього налаштовується автоматичний запуск необхідного скрипта в заданий час.

Для зазначеного налаштування в Linux використовується системний сервіс **Cron** - планувальник, який дозволяє виконувати необхідні скрипти раз на годину, раз в день, тиждень або місяць, а також в будь-який заданий час або через

будь-який інтервал часу. Сервіс cron також часто використовується іншими службами операційної системи.

cron запускається при старті системи і працює у фоновому режимі. Його основне завдання - виконувати необхідні процеси в заданий час.

Існує кілька конфігураційних файлів, з яких береться інформація про те, що і коли потрібно виконувати.

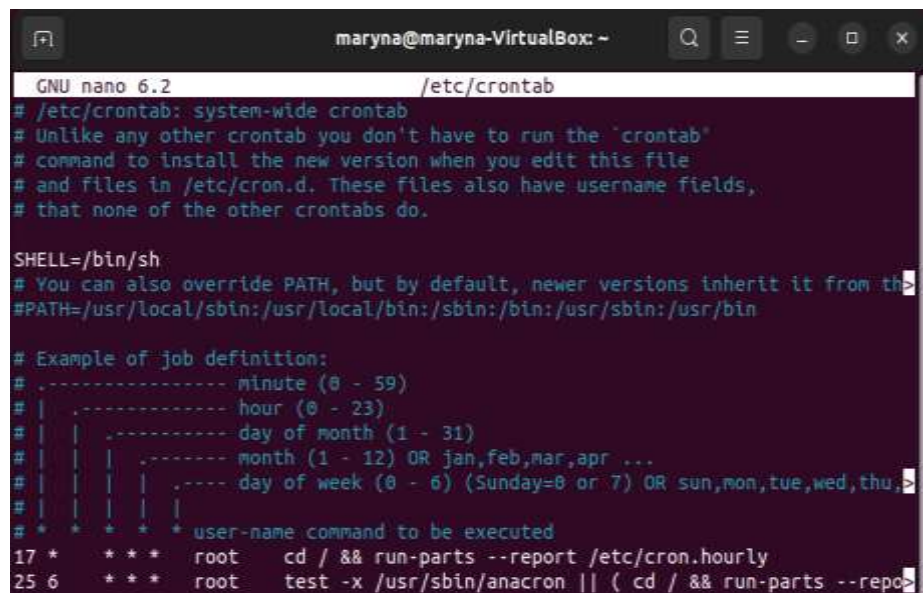
Cron постійно перевіряє файл **/etc/crontab** і каталог **/etc/cron.***/. Він також перевіряє каталог **/var/spool/cron/**.

Команда **Crontab** - це команда, яка використовується для установки, видалення або виведення файлу конфігурації **cron**, використовуваного для управління демоном **cron**. Кожен користувач може мати свій власний файл завдань crontab, і хоча ці файли зберігаються в **/var/spool/cron/crontabs**, вони не призначені для прямого редагування. Потрібно використовувати команду **crontab** для редагування або настройки ваших власних завдань, щоб планувальник cron працював коректно.

Існують різні типи файлів конфігурації:

Системний crontab: Зазвичай використовується системними службами та критичними завданнями, які вимагають привілеїв root доступу. Шосте поле файлу **/etc/crontab** - це ім'я користувача, яке за замовчуванням присвоєно команді. Це дає системі можливість запускати команди від різних користувачів (рисунок 8.32).

Користувацький crontab: Користувач може встановити свої власні завдання в програмі планувальнику Cron, використовуючи команду **crontab**. Шосте поле - це команда для запуску, і всі команди будуть виконуватися від користувача, який поставив їх в своєму власному **crontab** (рисунок 8.33).

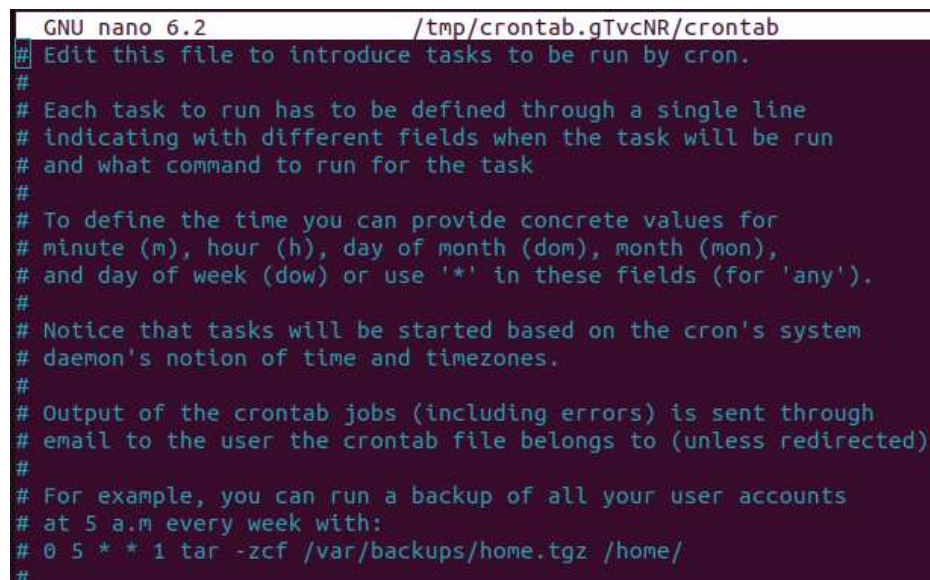
A terminal window titled 'maryna@maryna-VirtualBox: ~' shows the nano 6.2 editor editing the file '/etc/crontab'. The file contains system-wide crontab instructions, including comments about installing new versions, setting SHELL to /bin/sh, and an example job definition for a cron.hourly task. The last line of the file is '17 * * * * root cd / && run-parts --report /etc/cron.hourly'.

```
GNU nano 6.2 /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
# You can also override PATH, but by default, newer versions inherit it from the
#PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# ----- minute (0 - 59)
# | ----- hour (0 - 23)
# | | ----- day of month (1 - 31)
# | | | ----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report
```

Рисунок 8.32 – Системний crontab

A terminal window shows the nano 6.2 editor editing a user crontab file located at '/tmp/crontab.qTvcNR/crontab'. The file contains instructions for editing the file to introduce tasks, defining task lines with time fields and commands, and an example task for backing up user accounts at 5 a.m. every week using the command 'tar -zcf /var/backups/home.tgz /home/'.

```
GNU nano 6.2 /tmp/crontab.qTvcNR/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected)
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
```

Рисунок 8.33 – Користувацький crontab

Створення або редагування, видалення власних завдань cron

Щоб відредагувати файл crontab, потрібно ввести наступну команду в командному рядку Linux:

\$ crontab -e

Можна перевірити створені завдання за допомогою команди:

\$ crontab -l

Для видалення всіх записів в crontab використовується команда:

\$ crontab -r

Ключ -u дозволяє виконувати вищеописані дії для конкретного користувача:

\$ crontab -u username -e / -l / -r

Але потрібно врахувати, що -u можна скористатися лише запускаючи crontab від імені суперкористувача.

Демон cron зчитує необхідну для виконання завдань інформацію з каталогу /var/spool/cron. Для кожного користувача, який має свої завдання для демона cron, в цьому каталозі створюється відповідний файл.

Доступ до демона cron контролюється адміністратором в файлах /etc/cron.allow і /etc/cron.deny

Синтаксис crontab:

1 2 3 4 5 (user) /path/to/command arg1 arg2

1 хвилини 0..59

2 години 0..23

3 день місяця 1..31

4 місяць 1..12

5 день тижня 0..7, 0 - неділя

6 користувач, від якого виконується команда (для системного crontab)

7 команда і її параметри

Використання операторів

Оператор дозволяє вказати кілька значень в поле. Існує чотири оператора:

Зірочка (*): цей оператор задає всі можливі значення для поля. Наприклад, зірочка в поле годин буде еквівалентна кожній годині, а зірочка в поле місяця буде еквівалентна кожному місяцю і т.д.

Кома (,): цей оператор задає список значень, наприклад: «1,5,10,15,20,25».

Тире (-): цей оператор задає діапазон значень, наприклад: «5-15» днів, що еквівалентно набору «5,6,7,8,9, 13,14,15», при використанні оператора «Кома».

Слеш (/): цей оператор задає значення кроку, наприклад: «0-23 />» можна використовувати в полі годин, щоб вказувати виконання команди кожну годину. Кроки також дозволені після зірочки, тому, якщо ви хочете виконувати щось кожні дві години, можна використати «* / 2».

Приклади (значення часу в /etc/crontab):

05 15 * * *	кожного дня о 15:05
05 15 17 2 *	кожне 17 лютого о 15:05
05 15 * * 6	щосуботи о 15:05
05 15 1 1 1	кожне 1-е січня, якщо випадає на понеділок о 15:05
05,40 * * * *	о 5 і 40 хв. кожної години
* / 15 * * * *	кожні 15 хвилин
05 15 * * 1-5	у будні о 15:05

Можна використовувати спеціальні рядки. Замість перших п'яти полів можна використовувати будь-який з восьми спеціальних рядків. Планувальник Cron дозволяє таке налаштування.

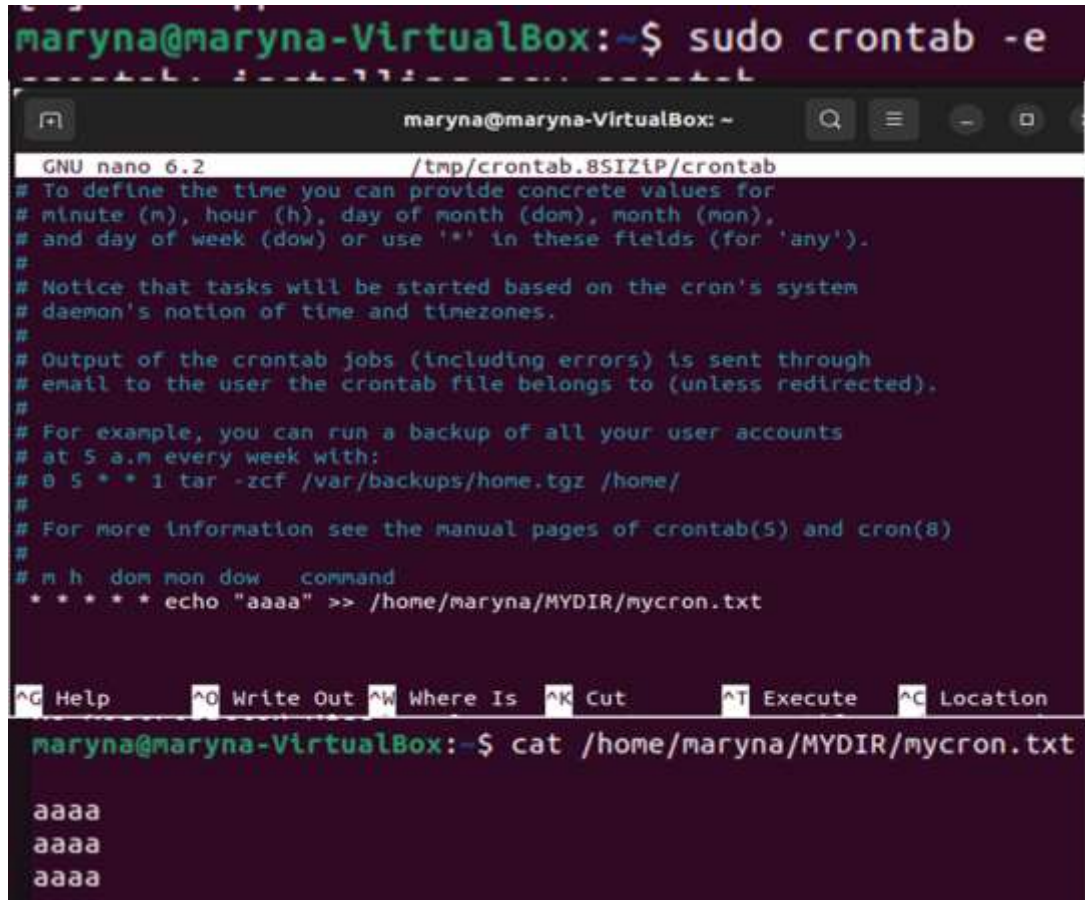
@reboot	Запускається тільки один раз при запуску.
@yearly	Запускається раз на рік, «0 0 1 1 *».
@annually	Таке ж, як у @yearly
@monthly	Запускається раз в місяць, «0 0 1 * *».
@weekly	Виконується раз в тиждень, «0 0 * * 0».
@daily	Виконується один раз в день, «0 0 * * *».
@midnight	Таке ж, як у @daily.
@hourly	Запускається один раз на годину, «0 * * * *».

Приклад:

#Запуск my.sh кожен годину #

@hourly my.sh ~ 0 * * * * my.sh

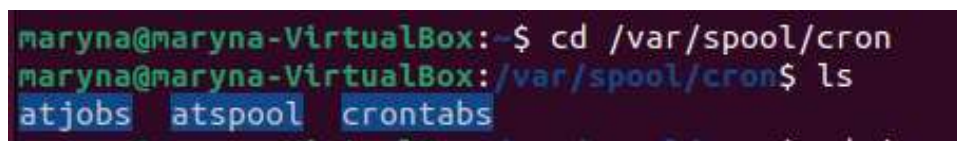
Приклад налаштування запуску команди echo, яка виконується кожну хвилину і результат записується у файл (рисunek 8.34).



```
maryna@maryna-VirtualBox:~$ sudo crontab -e
GNU nano 6.2 /tmp/crontab.8SIZiP/crontab
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
* * * * * echo "aaaa" >> /home/maryna/MYDIR/mycron.txt
^G Help      ^O Write Out ^W Where Is  ^K Cut      ^T Execute  ^C Location
maryna@maryna-VirtualBox:~$ cat /home/maryna/MYDIR/mycron.txt
aaaa
aaaa
aaaa
aaaa
```

Рисунок 8.34 – Налаштування користувацького crontab

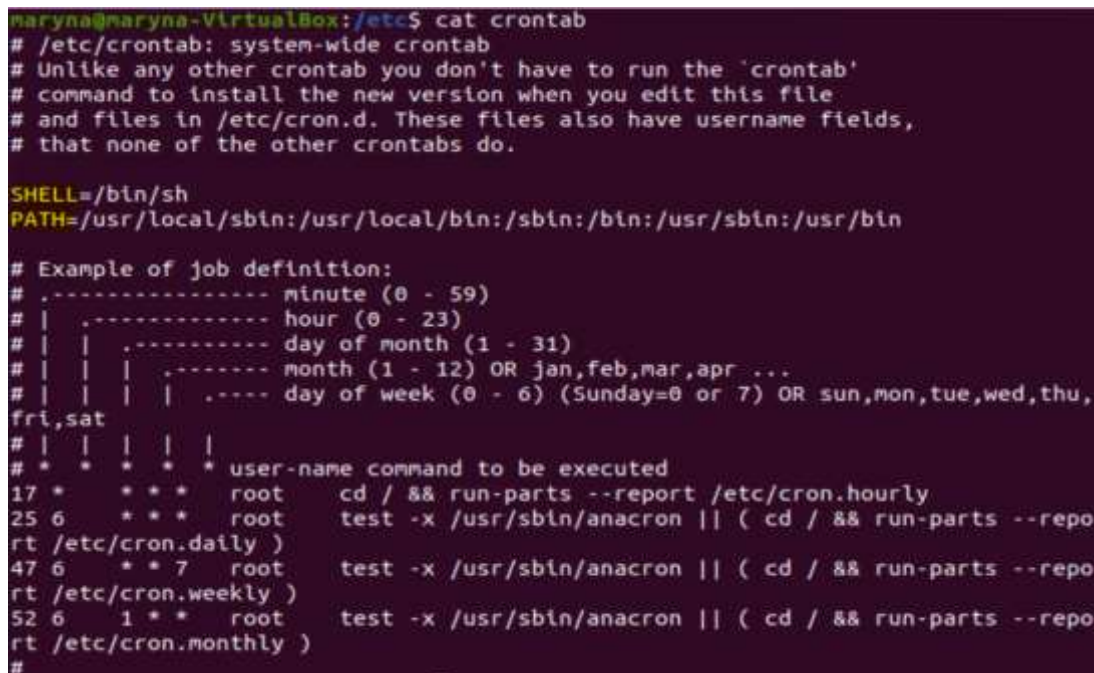
/etc/crontab - це файл crontab системи. Зазвичай використовується тільки користувачем root або для настройки системних задач. /var/spool/cron/ або /var/cron/tabs/ - це каталог для особистих файлів користувача crontab (рисunek 8.35).



```
maryna@maryna-VirtualBox:~$ cd /var/spool/cron
maryna@maryna-VirtualBox:/var/spool/cron$ ls
atjobs  atspool  crontabs
```

Рисунок 8.35 – Каталог для особистих файлів користувача crontab

Записи файлу /etc/crontab (рисунок 8.36):

A terminal window showing the contents of the /etc/crontab file. The text includes comments about system-wide crontab, SHELL and PATH settings, and an example of job definitions for hourly, daily, weekly, and monthly tasks, all executed as root using run-parts.

```
maryna@maryna-VirtualBox:/etc$ cat crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

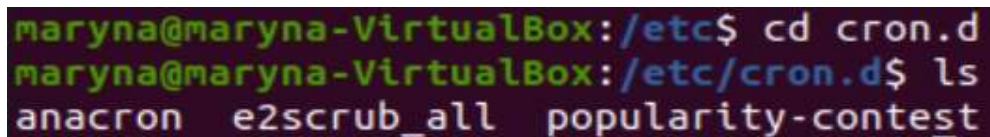
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo
rt /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo
rt /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo
rt /etc/cron.monthly )
#
```

Рисунок 8.36 – Файл /etc/crontab

Насамперед необхідно визначити середовище. Якщо вказівник на оболонку **SHELL** не прописаний, cron буде використовувати значення за замовчуванням sh. Якщо змінна **PATH** не вказана, розташування файлів буде абсолютним. Якщо змінна **HOME** не вказана, cron буде використовувати домашній каталог.

Також cron зчитує файли в каталозі /etc/cron.d/.

A terminal window showing the command to change directory to /etc/cron.d and then listing the files in that directory. The files listed are anacron, e2scrub_all, and popularity-contest.

```
maryna@maryna-VirtualBox:/etc$ cd cron.d
maryna@maryna-VirtualBox:/etc/cron.d$ ls
anacron  e2scrub_all  popularity-contest
```

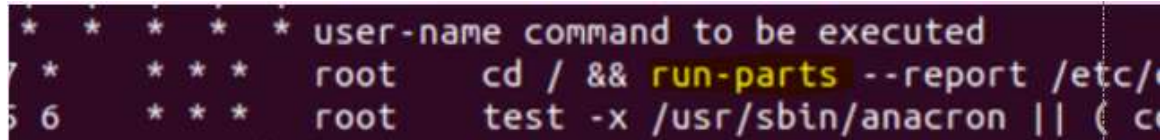
Рисунок 8.37 – Каталог /etc/cron.d/

В якості користувача root, для налаштування завдань cron, можна використовувати каталоги /etc/cron.d/, /etc/cron.daily/, /etc/cron.hourly/,

/etc/cron.monthly/ /etc/cron.weekly/. Тут можна безпосередньо запускати свої скрипти.

Команда **run-parts** запускає скрипти або програми в каталозі через /etc/crontab файл (рисунок 8.38):

/etc/cron.d/ помістіть в цей каталог всі скрипти і викличте їх з файлу /etc/crontab.



```
* * * * * user-name command to be executed
7 * * * * root cd / && run-parts --report /etc/cron.d/
5 6 * * * root test -x /usr/sbin/anacron || ( cd
```

Рисунок 8.38 – Команда run-parts

/etc/cron.daily/ запуск всіх скриптів один раз в день

/etc/cron.hourly/ запуск всіх скриптів один раз на годину

/etc/cron.monthly/ запуск всіх скриптів один раз на місяць

/etc/cron.weekly/ запуск всіх скриптів один раз в тиждень

Для налаштування команд на багаторазове/регулярне виконання може використовуватися додатковий засіб **anacron**.

Anacron використовується для періодичного запуску команд з частотою, визначеною в днях. Його робота трохи відрізняється від роботи cron; припускаючи, що машина не буде включена весь час.

Anacron підходить для виконання щоденних, щотижневих та щомісячних запланованих завдань, зазвичай виконуваних cron, але на машинах, які не будуть працювати 24/7.

Якщо у вас є заплановане завдання (наприклад, сценарій резервного копіювання), яке буде запускатися з використанням cron кожну ніч, коли комп'ютер вимкнений. Ваш скрипт не буде виконано.

Однак, якщо ви використовуєте `anacron`, ви можете бути впевнені, що при наступному включенні комп'ютера у вас обов'язково буде виконаний скрипт резервного копіювання.

Завдання `anacron` перераховані в `/etc/anacrontab` (рисунок 8.39), і завдання можна запланувати в наступному форматі (коментарі всередині файлу `anacrontab` повинні починатися з #):

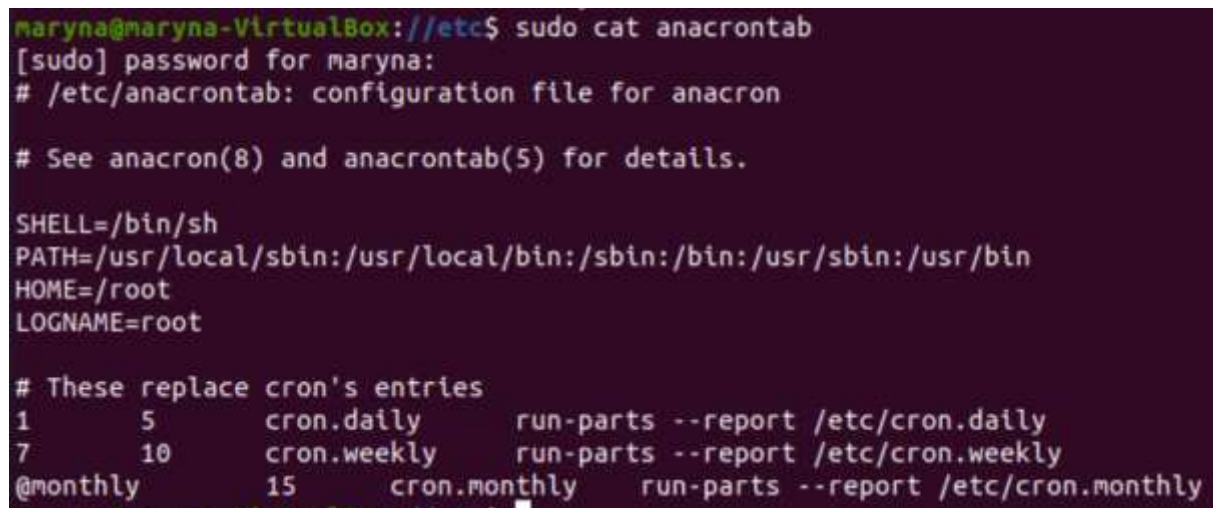
period delay job-identifier command

де, **period** - це частота виконання завдання, зазначена в днях або у вигляді `@daily`, `@weekly` або `@monthly` для одного разу на день, тиждень або місяць відповідно. Ви також можете використовувати числа: 1 - щодня, 7 - щотижня, 30 - щомісяця і N - кількість днів.

delay - це кількість хвилин очікування перед виконанням завдання.

job-id - це ім'я для завдання, записане в файлах журналу.

command - це команда або сценарій оболонки, який буде виконаний.



```
maryna@maryna-VirtualBox:~/etc$ sudo cat anacrontab
[sudo] password for maryna:
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
HOME=/root
LOGNAME=root

# These replace cron's entries
1      5      cron.daily      run-parts --report /etc/cron.daily
7      10     cron.weekly     run-parts --report /etc/cron.weekly
@monthly 15     cron.monthly    run-parts --report /etc/cron.monthly
```

Рисунок 8.39 – Файл `/etc/anacrontab`

Файли завдань (рисунок 8.40):

```
maryna@maryna-VirtualBox: //etc$ ls -l /var/spool/anacron
total 12
-rw----- 1 root root 9 бер 23 12:29 cron.daily
-rw----- 1 root root 9 лют 25 13:07 cron.monthly
-rw----- 1 root root 9 бер 22 17:18 cron.weekly
```

Рисунок 8.40 – Файли завдань

Anacron працює наступним чином:

- перевірить, чи виконано завдання за вказаний період в поле `period`. Якщо ні, він виконує команду, зазначену в полі `command`, після очікування кількості хвилин, зазначеного в полі `delay`;
- після виконання завдання він записує дату в файл мітки часу в каталозі `/var/spool/anacron` з ім'ям, зазначеним в поле `id` завдання (ім'я файлу мітки часу).

Наступний приклад буде запускати скрипт `/home/aron/bin/backup.sh` кожного дня:

@daily 10 example.daily / bin / bash /home/aron/bin/backup.sh

Якщо машина вимкнена, коли очікується виконання завдання `backup.sh`, `anacron` запустить її через 10 хвилин після увімкнення машини, не чекаючи наступного дня.

Порівняння `cron` і `anacron`, для розуміння, коли їх використовувати в таблиці 8.1.

Таблиця 8.1– Порівняння `cron` і `anacron`

Cron	Anacron
Це служба	Це не служба
Підходить для серверних машин	Підходить для настільних комп'ютерів / ноутбуків

Дозволяє запускати заплановані завдання кожну хвилину	Дозволяє запускати тільки заплановані завдання на щоденній основі
Не виконується заплановане завдання, коли машина вимкнена	Якщо машина вимкнена, коли заплановано завдання, вона виконає заплановане завдання при наступному включенні машини
Може використовуватися як звичайними користувачами, так і користувачем root	Може використовуватися тільки користувачем root, якщо не вказано інше (включено для звичайних користувачів з конкретними налаштуваннями)

Команда **at**

В якості альтернативи планувальником завдань **cron** існують різні команди, включаючи команду **at**, яка дозволяє планувати виконання команди один раз в заданий час без редагування файлу конфігурації.

Програма **at** є частиною набору, що складається з чотирьох програм: **at**, **batch**, **atq** і **atrm**:

at виконує завдання в призначений час.

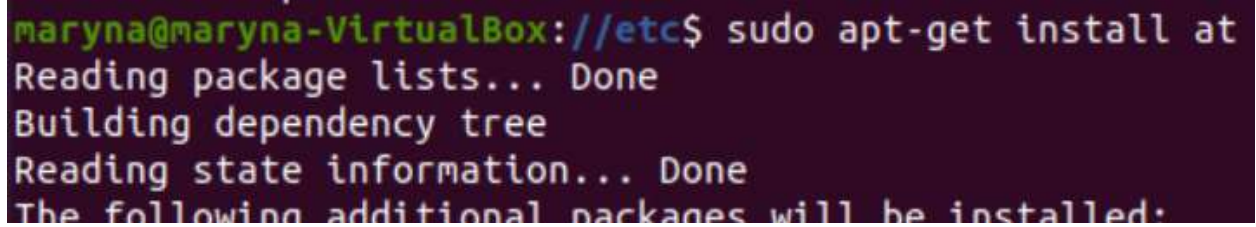
atq виводить список задач, які очікують виконання, для кожного користувача; в разі використання суперкористувачем, виводяться всі задачі, які знаходяться в режимі очікування.

atrm видаляє завдання, задані ідентифікаторами.

batch виконує завдання під час періодів низької завантаженості системи; іншими словами, коли середній рівень завантаження системи падає нижче значення 1.5 або того значення, яке задано при виклику **atd**.

at потрібно інсталиувати (рисунок 8.41) і включити її виконання:

\$ sudo apt-get install at для Debian / Ubuntu



```
maryna@maryna-VirtualBox:~/etc$ sudo apt-get install at
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

Рисунок 8.41 – Інсталювання at

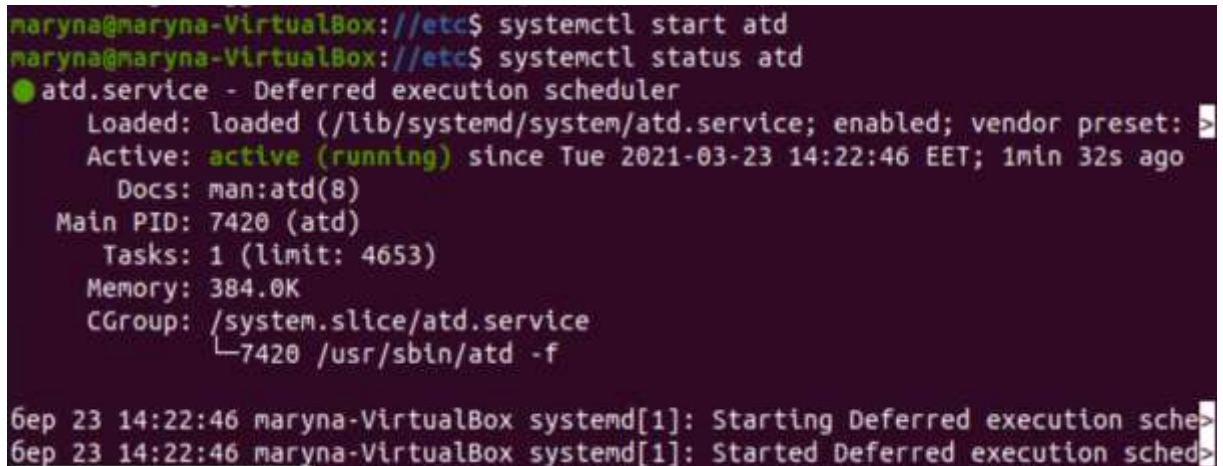
Потім потрібно запустити і додати службу (рисунок 8.42) at в автозавантаження.

\$ systemctl start atd

\$ systemctl enable atd

Перевірити стан at (рисунок 8.42):

\$ systemctl status atd



```
maryna@maryna-VirtualBox:~/etc$ systemctl start atd
maryna@maryna-VirtualBox:~/etc$ systemctl status atd
● atd.service - Deferred execution scheduler
   Loaded: loaded (/lib/systemd/system/atd.service; enabled; vendor preset: >
   Active: active (running) since Tue 2021-03-23 14:22:46 EET; 1min 32s ago
     Docs: man:atd(8)
   Main PID: 7420 (atd)
    Tasks: 1 (limit: 4653)
   Memory: 384.0K
    CGroup: /system.slice/atd.service
            └─7420 /usr/sbin/atd -f

ber 23 14:22:46 maryna-VirtualBox systemd[1]: Starting Deferred execution sche>
ber 23 14:22:46 maryna-VirtualBox systemd[1]: Started Deferred execution sched>
```

Рисунок 8.42 – Запуски служби та перевірка стану at

Після запуску at можна ввести послідовність команд для виконання. Щоб закінчити введення команд, слід використовувати комбінацію клавіш **CTRL+D**.

Опис основних параметрів командного рядка at:

at [-m] [-q чергу] [-f файл] час

Значення цих параметрів:

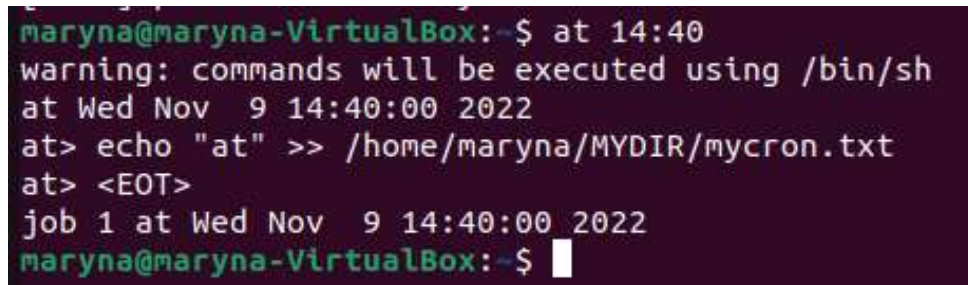
-q використовується для вказівки черги;

-m дозволяє відправити користувачеві повідомлення по електронній пошті після виконання завдання навіть в тому випадку, коли виконана програма нічого не вивела:

-f дозволяє прочитати команди з файлу, а не зі стандартного введення.

Час at приймає рядки в форматах, сумісних зі стандартом POSIX.2. Приймається рядок, який вказує час в форматі ГГ: ХХ, що дозволяє виконати команду в призначений час протягом дня.

Приклад роботи at (рисунок 8.43):



```
maryna@maryna-VirtualBox:~$ at 14:40
warning: commands will be executed using /bin/sh
at Wed Nov 9 14:40:00 2022
at> echo "at" >> /home/maryna/MYDIR/mycron.txt
at> <EOT>
job 1 at Wed Nov 9 14:40:00 2022
maryna@maryna-VirtualBox:~$
```

Рисунок 8.43 –

Щоб створити завдання batch (рисунок 8.44), потрібно передати команди, які ви хочете виконати:

echo "hello" | batch

завдання, створені за допомогою batch, плануються в черзі.

Запити можуть мати ім'я від a до z та A до Z. Черги з малими літерами мають пріоритет перед великими літерами.

Щоб встановити завдання у чергу b, потрібно виконати:

at now +2 hours -q b

The image shows a terminal window and a text file. The terminal window displays the following commands and output:

```
maryna@maryna-VirtualBox:~$ echo "111111" | batch
warning: commands will be executed using /bin/sh
job 7 at Wed Nov 9 14:52:00 2022

maryna@maryna-VirtualBox:~$ at now +2 minutes -q B
warning: commands will be executed using /bin/sh
at Wed Nov 9 14:49:00 2022
at> echo "batch" >> /home/maryna/MYDIR/mycron.txt
at> <EOT>
job 6 at Wed Nov 9 14:49:00 2022
maryna@maryna-VirtualBox:~$ atq
6          Wed Nov 9 14:49:00 2022 B maryna
```

The text file, named `mycron.txt` and located in `~/MYDIR`, contains the following content:

```
1
2 aaaa
3 aaaa
4 at
5 batch
```

Рисунок 8.44 – batch

Можна задати день для виконання команди за допомогою рядка, що вказує дату в форматі назви місяця і дня з необов'язковим зазначенням року або задавши дату рядком формату ММДДРР або ММ/ДД/РР або дд.мм.рр. та час (рисунок 8.45).

The image shows a terminal window with the following command and output:

```
maryna@maryna-VirtualBox:~/home$ sudo at -f file.sh 14:44
warning: commands will be executed using /bin/sh
job 1 at Wed Mar 24 14:44:00 2021
```

Рисунок 8. 45— Завдання на певний час

Приклади планування завдань за допомогою команди `at`:

`at now + 5 minutes` почати виконання завдання через 5 хвилин

`at now + 10` почати виконання завдання через 10 годин

`at now + 2 days` почати виконання завдання через 48 годин

`at now + 1 week` почати виконання завдання через 7 днів

`at teatime` почати виконання завдання о 4 годині дня

`at 14.00 3/14/25` почати виконання завдання о 2 годині дня 14 березня 2025 року.

Для перегляду списку запланованих або виконуваних в даний момент задач можна використовувати програму **atq** (рисунок 8.46) в якості альтернативи команді **at -l** (рисунок 8.47).

```
maryna@maryna-VirtualBox:~/home$ sudo atq  
1      Wed Mar 24 14:44:00 2021 a root
```

Рисунок 8.46 – Команда atq

```
maryna@maryna-VirtualBox:~/home$ sudo at -l  
1      Wed Mar 24 14:44:00 2021 a root
```

Рисунок 8.47 – Команда at -l

Для того, щоб побачити, що буде виконуватися в рамках задачі, можна використовувати наступну команду (рисунок 8.48):

at -c ідентифікатор_задачі

```
maryna@maryna-VirtualBox:~/home$ sudo at -c 1  
#!/bin/sh  
# atrun uid=0 gid=0  
# mail maryna 0  
umask 22  
COLORTERM=truecolor; export COLORTERM  
LC_ADDRESS=uk_UA.UTF-8; export LC_ADDRESS  
LC_NAME=uk_UA.UTF-8; export LC_NAME
```

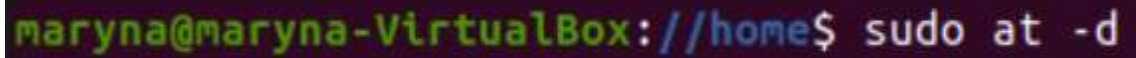
Рисунок 8.48 – Команда at -c

Вона виведе довгий список змінних оточення для виконання команди і саму команду (рисунок 8.49):

```
SUDO_GID=1000; export SUDO_GID  
cd /home || {  
    echo 'Execution directory inaccessible' >&2  
    exit 1  
}  
#!/bin/bash  
  
echo "Copy..."  
our_dir="/home/111"  
cp /home/222/* $our_dir
```

Рисунок 8. 49– Результат команди at -c

Видалення завдань з черги за допомогою програми **atrm** або команди **at -d** для видалення певної задачі (рисунк 8.50).



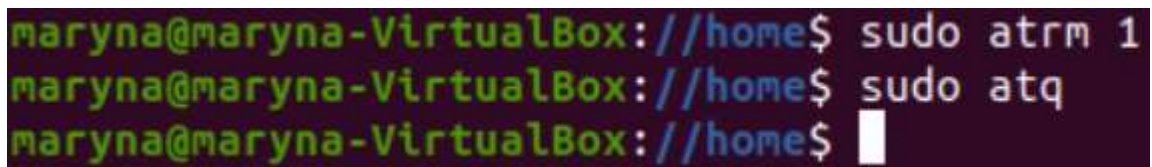
```
maryna@maryna-VirtualBox: //home$ sudo at -d
```

Рисунок 8.50 – Команда at -d

Наприклад, для видалення завдання з ідентифікатором 1, необхідно ввести команду: **atrm ідентифікатор_задачі**

\$ atrm 1 (at -d 1)

Дана команда нічого не виводить в консоль, але після її виконання ви можете переконатися в видаленні задачі за допомогою команди **atq** (рисунк 8.51).



```
maryna@maryna-VirtualBox: //home$ sudo atrm 1
maryna@maryna-VirtualBox: //home$ sudo atq
maryna@maryna-VirtualBox: //home$
```

Рисунок 8.51 – Команда atrm

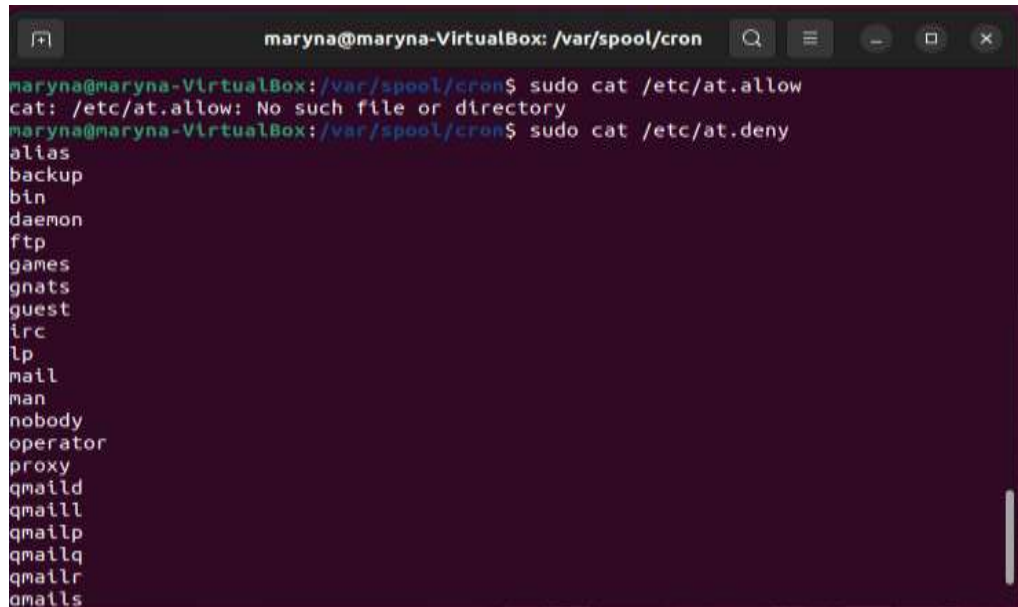
Обмеження списку користувачів програми at

Файли **/etc/at.allow** і **/etc/at.deny** встановлюють користувачів, які можуть планувати завдання для подальшого виконання за допомогою програм **at** і **batch**. В якості формату цих файлів використовується простий список імен користувачів по одному в кожному рядку. Використання прогалів в цих файлах не допускається.

Користувач **root** може використовувати програми **at** і **batch** при будь-яких умовах.

У разі існування файлу **/etc/at.allow**, на його основі встановлюється списку користувачів, яким дозволено планування завдань, але зазвичай в системах цього файлу не існує.

Якщо файлу `/etc/at.allow` не існує, перевіряється файл `/etc/at.deny`, який зазвичай містить довгий список "системних користувачів", таких як `bin`, `backup`, для яких планування завдань заборонено (рисунок 8.52).

A screenshot of a terminal window titled 'maryna@maryna-VirtualBox: /var/spool/cron'. The terminal shows the command 'sudo cat /etc/at.allow' which returns 'cat: /etc/at.allow: No such file or directory'. Then, the command 'sudo cat /etc/at.deny' is executed, displaying a list of system users: alias, backup, bin, daemon, ftp, games, gnats, guest, irc, lp, mail, man, nobody, operator, proxy, qmaild, qmail, qmailp, qmailq, qmailr, and qmails.

```
maryna@maryna-VirtualBox: /var/spool/cron$ sudo cat /etc/at.allow
cat: /etc/at.allow: No such file or directory
maryna@maryna-VirtualBox: /var/spool/cron$ sudo cat /etc/at.deny
alias
backup
bin
daemon
ftp
games
gnats
guest
irc
lp
mail
man
nobody
operator
proxy
qmaild
qmail
qmailp
qmailq
qmailr
qmails
```

Рисунок 8.51 – Файли дозволів та обмежень

Порядок виконання лабораторної роботи

- 1) ознайомитися з теоретичними відомостями;
- 2) виконати завдання;
- 3) оформити звіт;
- 4) продемонструвати результат на комп'ютері і захистити роботу

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andrew S. Tanenbaum, Herbert Bos. Modern operating systems – Pearson, 2022 – 1184 p
2. Linux Essentials Version: 1.6. [Електронний ресурс] – Режим доступу: <https://learning.lpi.org/uk/learning-materials/010-160/>
3. POSIX™ Certification. [Електронний ресурс] – Режим доступу: <https://posix.opengroup.org/>
4. Nelson Ruest, Danielle Ruest. Virtualization, A Beginner's Guide - McGraw Hill, 2009 – 442 p
5. VirtualBox Documentation. [Електронний ресурс] – Режим доступу: <https://www.virtualbox.org/wiki/Documentation>
6. Linux Tutorial. [Електронний ресурс] – Режим доступу: <https://www.geeksforgeeks.org/linux-unix/linux-tutorial/>
7. Ubuntu Tutorials. [Електронний ресурс] – Режим доступу: <https://ubuntu.com/tutorials>
8. Jayant Varma Pro Bash: Learn to Script and Program the GNU/Linux Shell 3rd ed. Edition – Apress, 2023 – 328 p