Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 «Інженерія програмного забезпечення»

На тему: *Розробка програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем*

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-22ск
_____ *Міхєєв О. С.*

Керівник кваліфікаційної роботи _____ */ Стрюк А. М. /*

Завідувач кафедри _____ */ Стрюк А. М. /*

Кривий Ріг
2025

Криворізький національний університет
Факультет: *Інформаційних технологій*
Кафедра: *Моделювання та програмного забезпечення*
Освітньо-кваліфікаційний рівень: *бакалавр*
Спеціальність: *121 "Інженерія програмного забезпечення"*

ЗАТВЕРДЖУЮ
Зав. кафедри
_____*Стрюк А. М.*
«___»_____ 2025__ р.

ЗАВДАННЯ
на кваліфікаційну роботу
студенту групи ІПЗ-22ск Міхєєву Олександру Сергійовичу

1. Тема: *Розробка програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем* затверджено наказом по КНУ №119 від «10» квітня 2025 р.
2. Термін подання студентом закінченого проекту «20» червня 2025 р.
3. Вихідні дані по роботі: *Пояснювальна записка: 110 сторінок, 52 рисунків, 1 додаток, 15 використаних у роботі джерел*
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Актуальність вивчення принципів використання терміналу UNIX-подібних систем. Використання UNIX-систем в сучасній IT-індустрії. Огляд емуляторів UNIX. Огляд емуляторів терміналу, призначених для навчання. Проєктування вимог до програмного забезпечення ігрового тренажеру. Проєктування програмного забепзечення ігрового тренажеру. Проєктування потоків даних. Проєктування діаграми використання ігрового тренажеру. Проєктування баз даних ігрового тренажеру. Розробка програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем. Добір засобів розробки ігрового тренажеру. Приклади роботи ігрового тренажеру.
5. Перелік графічного демонстраційного матеріалу: Приклади існуючих програм. Діаграма потоків даних. Діаграми використання. Блок-схеми основних алгоритмів. Структура бази даних. Приклади роботи розробленої програми.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	*Формулювання мети та задач роботи*	*13.02.2025-20.02.2025*
2	*Аналіз інформаційних джерел*	*21.02.2025-09.03.2025*
3	*Визначення вимог до програмного забезпечення*	*10.03.2025-18.03.2025*
4	*Розробка функціональної схеми*	*19.03.2025-23.03.2025*
5	*Розробка алгоритмів*	*24.03.2025-31.03.2025*
6	*Розробка бази даних*	*01.04.2025-14.04.2025*
7	*Розробка програмного забезпечення*	*15.04.2025-13.05.2025*
8	*Тестування програмного забезпечення*	*14.05.2025-20.05.2025*
9	*Оформлення пояснювальної записки*	*21.05.2025-12.06.2025*
10	*Розробка демонстраційних матеріалів*	*13.06.2025-17.06.2025*

Дата видачі завдання: «_____»_____ 2025 р.
Студент _____ Міхєєв О. С.
Керівник роботи _____ Стрюк А. М.

РЕФЕРАТ

UNIX, ТЕРМІНАЛ, ГРА, ТРЕНАЖЕР, JAVASCRIPT, NODE.JS, XTERM.JS

Кваліфікаційна робота містить 110 сторінок, 52 рисунків, 1 додаток, 15 використаних у роботі джерел.

Метою кваліфікаційної роботи є створення програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем.

В роботі було досліджено актуальність вивчення принципів використання терміналу UNIX-подібних систем. Розглянути використання UNIX-систем в сучасній IT-індустрії. Оглянуто варіанти емуляторів терміналу UNIX. Досліджено приклади емуляторів терміналу, призначених для навчання.

В ході розробки було спроєктовано вимоги до програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX. Розроблено діаграми потоки даних, діаграми використання ігрового тренажеру. Спроєктовано структуру баз даних ігрового тренажеру.

Під час розробки було підібрано засоби розробки, зокрема бібліотеки, що надають можливість реалізувати емулятор терміналу UNIX. На їх основі розроблено веб-застосунок, що в ігровій формі надає можливість вивчати базові команди UNIX.

Ігровий тренажер має не лише зробити знайомство з UNIX-командами цікавішим, але й підвищить ефективність навчання, дозволяючи користувачеві з легкістю застосовувати здобуті знання на практиці.

ABSTRACT

UNIX, TERMINAL, GAME, TRAINER, JAVASCRIPT, NODE.JS, XTERM.JS

The qualification work contains 110 pages, 52 figures, 1 appendix, and 15 sources used in the work.

The goal of the qualification project is to develop software for a gaming trainer designed to teach the fundamentals of using the terminal in UNIX-like operating systems.

The study explored the relevance of learning the principles of using the terminal in UNIX-like systems. It considered the use of UNIX systems in the modern IT industry and reviewed various UNIX terminal emulators. Examples of terminal emulators designed for educational purposes were analyzed.

During the development process, requirements for the software of the gaming trainer were designed, focusing on learning the basics of UNIX terminal usage. Data flow diagrams and user interaction diagrams for the trainer were developed. The database structure of the gaming trainer was designed.

The development tools were selected, including libraries that enable the creation of a UNIX terminal emulator. Based on these, a web application was created that offers a gaming approach to learning basic UNIX commands.

The gaming trainer not only aims to make familiarity with UNIX commands more engaging but also enhances learning efficiency, allowing users to easily apply the acquired knowledge in practice.

ЗМІСТ

ВСТУП

Досвід роботи з терміналом UNIX-подібних систем є важливим для сучасних програмістів, незалежно від їхньої спеціалізації — чи займаються вони розробкою фронтенду, чи працюють над бекендом сучасних застосунків. Термінал дозволяє виконувати широкий спектр завдань, починаючи від базових операцій із файлами та каталогами до складних сценаріїв автоматизації робочих процесів. Програмісти часто використовують його для роботи з системами контролю версій, такими як Git, а також для налаштування середовища розробки або деплойменту додатків, зокрема через інструменти на кшталт Docker або Kubernetes.

Окрім того, знання роботи з терміналом відкриває доступ до багатьох потужних інструментів командного рядка, серед яких текстові редактори (наприклад, Vim або Nano), системи менеджменту пакетів (APT, Yum, Homebrew тощо), а також скриптові мови для автоматизації завдань, такі як Bash або Python. Це істотно прискорює виконання задач і забезпечує програмістів додатковою гнучкістю у розробці.

Навички роботи з терміналом також важливі для взаємодії з віддаленими серверами через SSH, налаштування системи, управління процесами або моніторинг ресурсів, що є невіддільною частиною роботи бекенд-розробників. Фронтенд-програмісти часто використовують термінал для установки бібліотек, запуску середовищ розробки або роботи з інструментами для трансформації та оптимізації коду, такими як Webpack, Parcel або Node.js.

Таким чином, практика роботи з терміналом є ключовою компетенцією в арсеналі сучасного програміста, яка дозволяє ефективно працювати над проектами, взаємодіяти з інфраструктурою, а також створювати масштабовані та продуктивні системи.

Щоб вивчити основні команди терміналу, не обов’язково встановлювати повноцінну UNIX-систему на свій комп’ютер. Існують більш прості та зручні методи для новачків, які бажають освоїти базові навички роботи з командним

рядком. Один із таких методів — використання спрощених емуляторів терміналу або інтерактивних онлайн-платформ.

Ці інструменти дозволяють отримати базові навички взаємодії з терміналом в симуляційному середовищі, що позбавляє потреби налаштовувати складне програмне забезпечення чи ризикувати порушенням роботи операційної системи.

Проте вивчення основних команд UNIX може виявитись доволі нудним та нецікавим, особливо для початківців, які лише знайомляться з операційною системою та її функціональністю. Водночас ефективне оволодіння командним рядком є ключовим для освоєння UNIX-подібних систем. Тому виникає потреба у формах навчання, які враховують особливості сучасного підходу до навчання — інтерактивність, залученість і ігрові елементи.

Одним із рішень може стати створення ігрового тренажера, що допоможе користувачам у цікавій і ненав’язливій формі освоїти основні команди UNIX. Такий тренажер може бути побудований за принципом "гейміфікації" — інтеграції ігрових елементів у освітній процес. Наприклад, користувачеві може пропонуватись виконати серію завдань, які поступово ускладнюються. Виконуючи завдання, він отримає бали або досягнення, що буде мотивувати продовжувати навчання.

Під час проходження таких завдань користувач не тільки запам’ятає синтаксис команд, але й зрозуміє, як і коли їх слід застосовувати. Ігровий формат може включати також різні рівні складності, від базового до просунутого, щоб врахувати можливості й рівень підготовки різних користувачів. Наприклад, новачок може почати з простого задачника, у якому йому пояснять, як навігаційні команди допомагають знайти файли в системі. У той час як досвідчені користувачі можуть отримати завдання зі створення складних сценаріїв автоматизації за допомогою `bash`-скриптів.

Таким чином, метою кваліфікаційної роботи є створення програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем.

Завданнями роботи є:

– Дослідити актуальність вивчення принципів використання терміналу UNIX-подібних систем.

– Розглянути використання UNIX-систем в сучасній IT-індустрії.

– Оглянути варіанти емуляторів терміналу UNIX.

– Дослідити приклади емуляторів терміналу, призначених для навчання.

– Спроєктувати вимоги до програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX.

– Спроєктувати потоки даних.

– Спроєктувати діаграми використання ігрового тренажеру.

– Спроєктувати баз даних ігрового тренажеру.

– Виконати добір засобів розробки ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем.

– Розробити програмне забезпечення ігрового тренажеру.

– Протестувати програмне забезпечення ігрового тренажеру.

Ігровий тренажер не лише зробить знайомство з UNIX-командами цікавішим, але й підвищить ефективність навчання, дозволяючи користувачеві з легкістю застосовувати здобуті знання на практиці.

1 АКТУАЛЬНІСТЬ ВИВЧЕННЯ ПРИНЦИПІВ ВИКОРИСТАННЯ ТЕРМІНАЛУ UNIX-ПОДІБНИХ СИСТЕМ

1.1 Використання UNIX-систем в сучасній IT-індустрії

UNIX – це родина багатозадачних операційних систем, яка розроблена для підтримки одночасного використання багатьма користувачами. Ця потужна операційна система стала основою для численних інших систем, охоплюючи ринок серверів, робочих станцій та суперкомп'ютерів. Її історія почалася наприкінці 1960-х років у Bell Labs, де талановиті програмісти, такі як Кен Томпсон, Денніс Рітчі та їхні колеги, розробили першу версію UNIX на комп'ютері PDP-7.

Ця перша реалізація зосереджувалася на роботі з файлами та процесами, надаючи користувачам нові можливості в управлінні обчислювальними ресурсами. Однією з ключових ідей, які лежать в основі UNIX, є концепція "всієї інформації у файлах", яка дозволяє зручно зберігати і обробляти дані. UNIX також запровадила потужну командну оболонку, яка дозволяє користувачам вводити команди, переміщуватися між файлами та запускати програми.

Протягом 1970-х і 1980-х років UNIX розвивався, виходив у нові версії, а також активно адаптувався компаніями та університетами, що призвело до появи численних варіантів системи, таких як BSD, System V та Apollo. Адаптивність і гнучкість UNIX дозволили йому стати основою для багатьох сучасних операційних систем, включаючи Linux та macOS. З часом, завдяки своїй надійності, стабільності та широкому інструментарію командного рядка, UNIX почала широко використовуватися у сфері науки, освіти та бізнесу. Сьогодні UNIX та його похідні системи залишаються надзвичайно впливовими у світі інформаційних технологій.

Спочатку UNIX був розроблений у 1969 році в дослідницькій лабораторії AT&T Bell Labs як внутрішній проєкт, метою якого було створення багатозадачної операційної системи для міні-комп'ютерів. Оскільки

система виявилася надзвичайно потужною та гнучкою, AT&T вирішила ліцензувати UNIX для сторонніх організацій. Це призвело до появи численних варіантів UNIX, кожен з яких адаптувався відповідно до специфічних потреб і обчислювальних середовищ. Серед найвідоміших варіантів були Berkeley Software Distribution (BSD), яка пропонувала нові функції та вдосконалення; Xenix, версія для мікрокомп'ютерів, що була речистю Microsoft; а також системи, розроблені різними виробниками апаратного забезпечення, такими як SunOS для комп'ютерів Sun Microsystems, HP-UX для систем Hewlett-Packard та AIX від IBM (рис. 1.1).

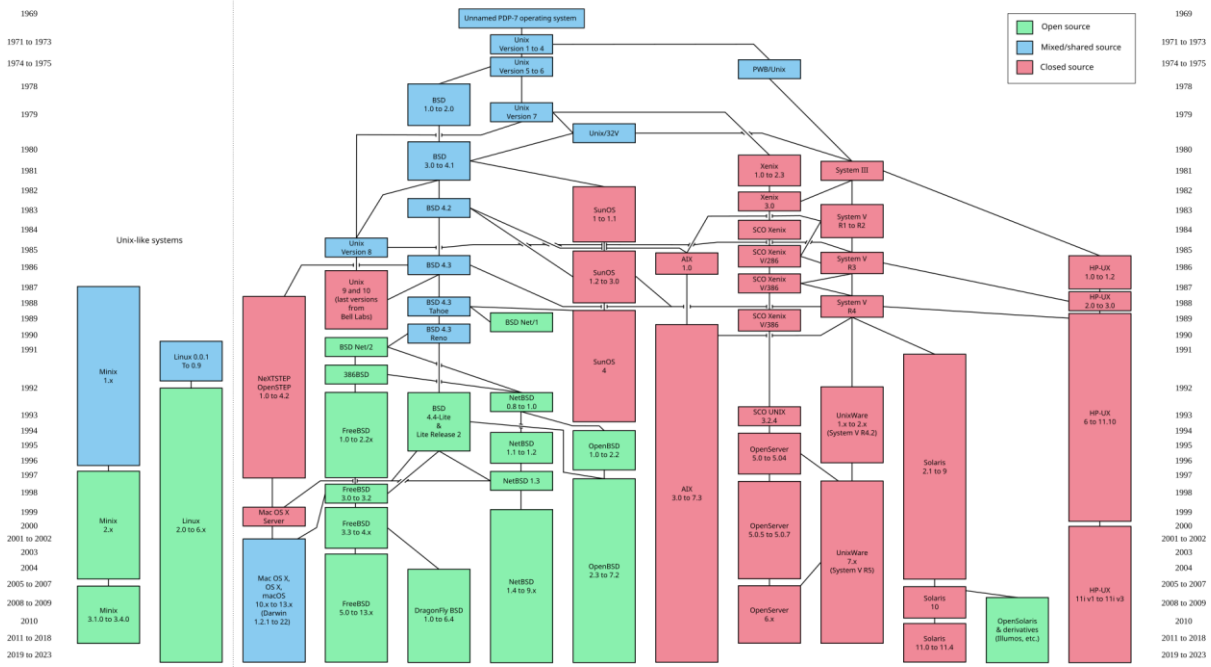

Рисунок 1.1 – Генеалогічне дерево Unix-систем

У 1990-х роках, у результаті змін у корпоративній структурі, AT&T прийняла рішення продати права на UNIX компанії Novell, що призвело до нових комерційних розгортань та розробок системи. Однак згодом Novell зосередилася на розробці програмного забезпечення мережі, і права на UNIX перейшли до The Open Group. Ця організація стала відповідальною за сертифікацію операційних систем відповідно до стандартів Single UNIX Specification. Ця специфікація встановлює стандартні вимоги до функцій,

інтерфейсів та поведінки UNIX-подібних систем, забезпечуючи сумісність між різними реалізаціями та сприяючи розвитку екосистеми UNIX в цілому.

На основі UNIX з'явилися UNIX-подібні системи, зокрема Linux, який був створений Лінусом Торвальдсом у 1991 році. Linux став популярним завдяки відкритому коду, що дозволяє будь-кому використовувати, модифікувати та поширювати його. Це сприяло формуванню активної спільноти розробників, яка регулярно вдосконалює систему, додає нові функції та виправляє помилки. Linux визнаний за свою стабільність, безпеку та гнучкість, завдяки чому він став основою для багатьох дистрибутивів, таких як Ubuntu, Fedora, Debian і CentOS, кожен з яких має свої особливості та цільову аудиторію.

Сьогодні UNIX та його похідні широко використовуються в корпоративних середовищах, де стабільність та безпека є критично важливими. Багато компаній обирають Linux для серверних рішень, оскільки він має менші витрати на ліцензії в порівнянні з комерційними операційними системами. Linux також активно використовується в наукових дослідженнях, вбудованих системах, хмарних обчисленнях та інтернеті речей.

Крім того, дистрибутиви Linux знайшли своє місце і в мобільних пристроях, зокрема в Android, який є найпопулярнішою операційною системою для смартфонів у всьому світі. Незважаючи на те, що Android є модифікованою версією ядра Linux, його успіх демонструє велику гнучкість системи та можливість адаптації під різні потреби користувачів.

Відтак, Linux і інші UNIX-подібні системи відіграють важливу роль у сучасному інформаційному технологічному просторі, підтримуючи різноманітні рішення для підприємств, розробників і звичайних користувачів.

Не зважаючи на неймовірну кількість версій UNIX, UNIX-подібних систем та їх дистрибутивів, всі вони мають спільний набір команд, що дозволяють користувачам взаємодіяти з операційною системою. Це уніфікованість у командах є результатом історичного розвитку системи UNIX, яка з'явилася у 1970-х роках і заклала основи для багатьох наступних систем.

Користувачі можуть виконувати різноманітні операції, такі як маніпуляція файлами, управління процесами, налаштування мережі та автоматизація завдань, завдяки цим базовим командам. Наприклад, команди на кшталт `ls` для переліку файлів у каталозі, `cd` для зміни каталогу, `cp` для копіювання файлів та `rm` для їх видалення є звичними для практично всіх дистрибутивів.

Засіб введення цих команд, термінал, слугує своєрідним містком між користувачем та операційною системою. Він дозволяє вводити текстові команди, отримувати результати їх виконання та взаємодіяти з файловою системою через командний рядок. З терміналом користувачі можуть не лише виконувати команди в режимі реального часу, а й використати скрипти для автоматизації рутинних завдань, що значно підвищує ефективність роботи.

Таким чином, хоча сучасні системи можуть пропонувати графічні інтерфейси, термінал залишається потужним і незамінним інструментом для розробників, адміністраторів та досвідчених користувачів, які прагнуть максимальної контролю та гнучкості в роботі з UNIX-подібними системами.

Для сучасних інженерів-програмістів важливо вміти працювати з терміналом UNIX, оскільки він забезпечує потужний інтерфейс для управління операційною системою та своїми ресурсами. Знання основних команд UNIX-подібних систем є необхідним для ефективної роботи з файлами, процесами, мережевими ресурсами та системними налаштуваннями.

Розуміння даних команд не лише полегшує рутинні завдання, але також підвищує продуктивність у роботі над великими проектами, оскільки дозволяє програмістам швидко взаємодіяти зі системою без необхідності використовувати графічний інтерфейс. Знання терміналу також надає змогу глибше розуміти принципи роботи операційної системи, що є вагомим плюсом в кар'єрі програміста.

Проте, якщо ваше завдання – лише ознайомитися з базовими командами UNIX, встановлювати повноцінний дистрибутив UNIX або Linux може здатися надмірно складним і необов'язковим процесом. На щастя, для цієї мети

можна скористатися спеціальними симуляторами терміналу, які дозволяють виконувати основні команди в комфортному середовищі без необхідності зміни операційної системи або налаштування віртуальних машин. Ці симулятори часто надають користувачам веб-інтерфейс, що дозволяє працювати з командним рядком прямо у браузері, а також забезпечують інтерактивне навчання завдяки підказкам, інтегрованим керівництвам і навчальним прикладам. Таким чином, ви можете швидко освоїти роботу з командами UNIX, ознайомитися із синтаксисом і принципами їх функціонування, а при необхідності – перейти до більш складних завдань у реальному середовищі. До того ж, симулятори терміналу часто мають перевагу у вигляді простішої установки, легкого доступу і відсутності ризику внесення незапланованих змін до вашої основної операційної системи.

1.2 Огляд емуляторів терміналу UNIX

На сьогодні існує багато емуляторів терміналу UNIX, які дозволяють користувачам працювати з командним рядком ефективно, незалежно від операційної системи, яку вони використовують. Ці емулятори забезпечують інтерактивний інтерфейс для вводу команд, виконання скриптів, управління файлами та іншими завданнями, характерними для UNIX-середовища. Вони імітують функціональність традиційного терміналу, дозволяючи запускати команди, працювати з текстовими файлами, встановлювати програмне забезпечення і використовувати різноманітні утиліти.

Такі інструменти не лише полегшують доступ до функціоналу UNIX-систем, але й забезпечують гнучкість для розробників, системних адміністраторів та ентузіастів, які працюють з різними наборами операційних систем. Вибір емулятора часто залежить від індивідуальних вимог користувача до продуктивності, функціональності та зручності інтерфейсу.

Крім того, існують емулятори, адаптовані для використання у веб-браузерах або мобільних пристроях, що дозволяє працювати з командним

рядком навіть у віддалених чи нестандартних умовах. Розглянемо детально деякі з них.

Alacritty – це сучасний емулятор терміналу з відкритим вихідним кодом, який використовує прискорення GPU для забезпечення максимальної продуктивності. Його ключовими особливостями є швидкість, мінімалістичний підхід до дизайну, а також зосередженість на забезпеченні плавної роботи навіть з великими обсягами тексту або під час виконання важких обчислювальних завдань.

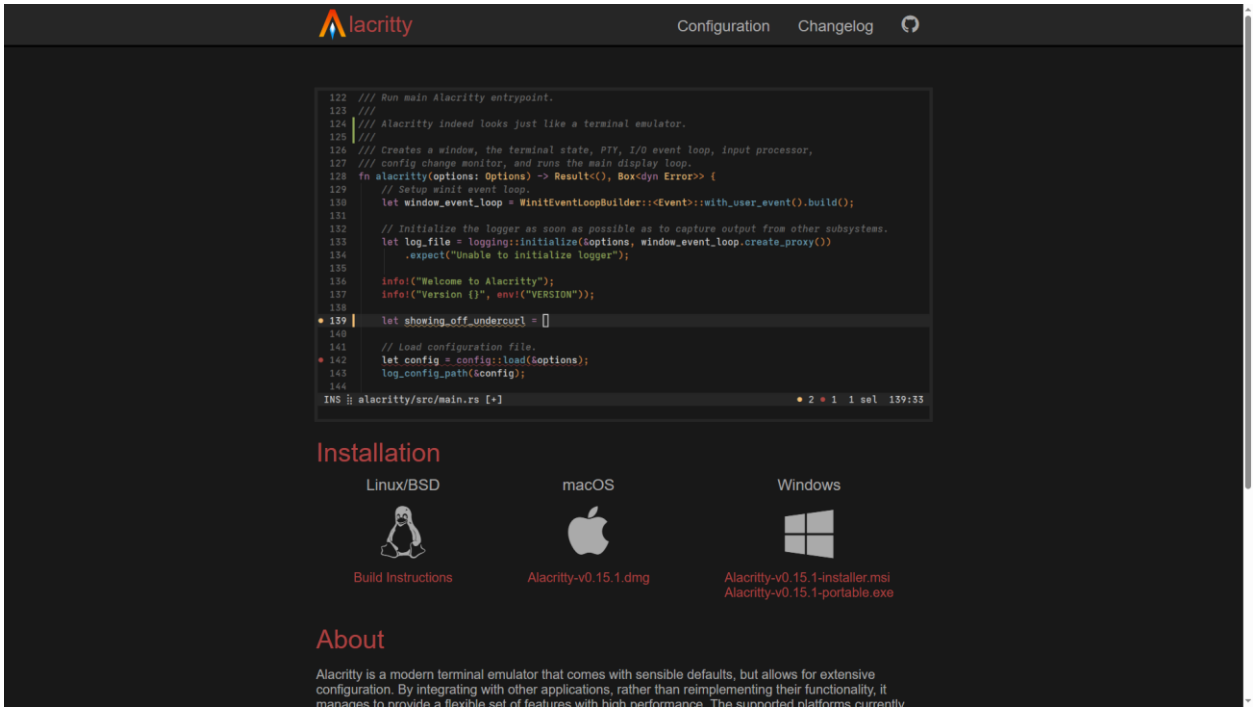

Рисунок 1.2 – Alacritty

Емулятор був розроблений з акцентом на простоту, без перевантаження користувацького інтерфейсу зайвими функціями. Він підтримує конфігурацію за допомогою текстового файлу YAML, що дозволяє користувачам налаштовувати такі параметри, як шрифти, кольорові схеми, підтримку клавіш та інші аспекти відповідно до власних потреб.

Однією з головних переваг Alacritty є його ефективність у використанні апаратних ресурсів, де рендеринг тексту виконується за допомогою графічного процесора (GPU), що значно прискорює обробку та відповідає сучасним вимогам до високої продуктивності. Завдяки цьому Alacritty

ідеально підходить для програмістів, системних адміністраторів та всіх, хто працює з терміналом на професійному рівні.

Alacritty добре інтегрується з іншими популярними інструментами, такими як tmux або vim. Він доступний для різних операційних систем, включаючи Linux, macOS та Windows, що робить його універсальним вибором для користувачів на різних платформах.

Cool Retro Term — це стильний і функціональний емулятор терміналу, який відтворює вигляд старих катодних дисплеїв. Завдяки ретро-дизайну програма створює атмосферу минулого, нагадуючи епоху ранніх комп'ютерів, коли текстові інтерфейси були основним методом взаємодії з машинами. Утиліта пропонує широкий спектр налаштувань, що дозволяє користувачам вибирати серед різних варіацій ретро-стилів, включаючи текстуру екрана, ефекти мерехтіння, вигнуті кути і типове розмиття, яке було характерним для старих моніторів.

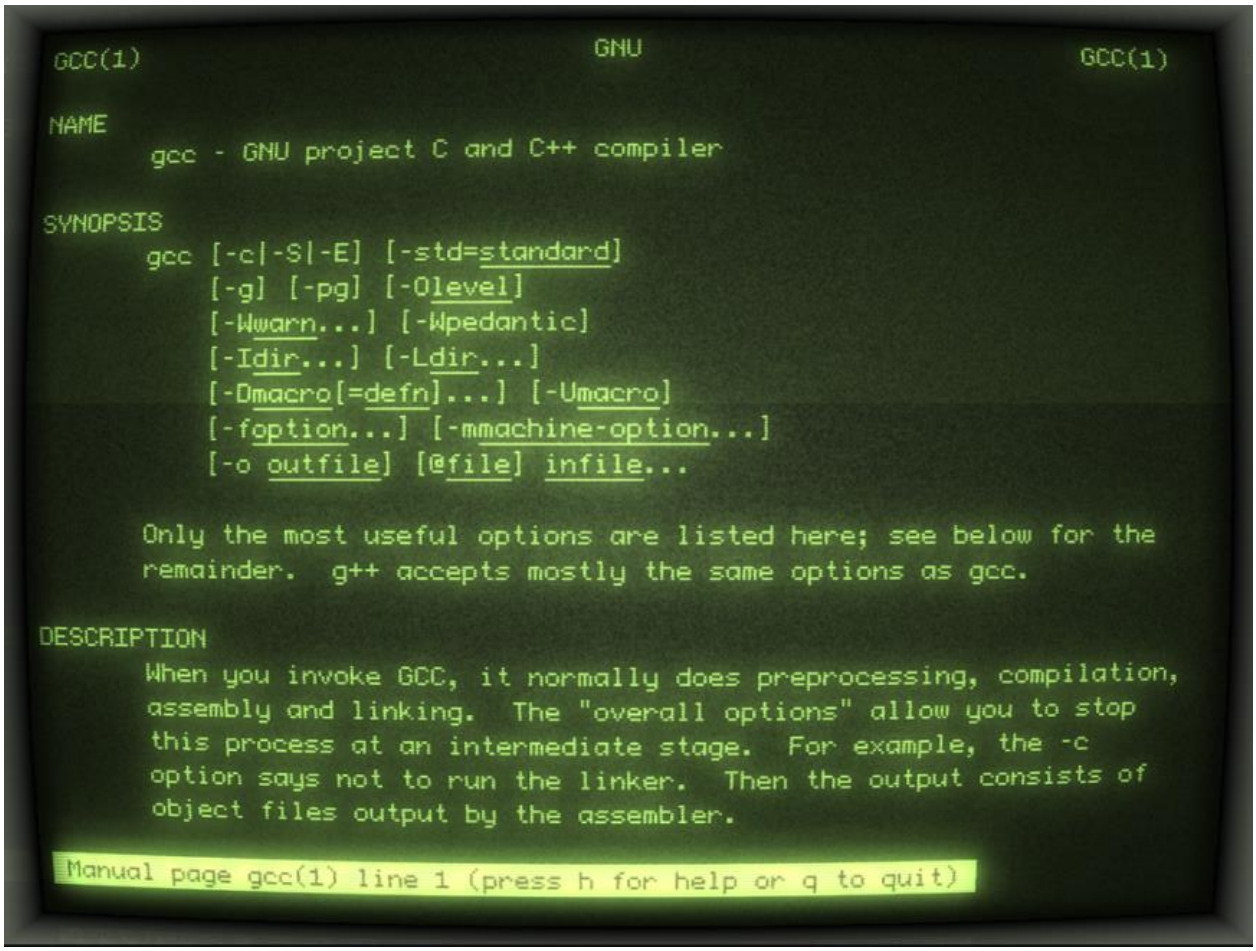

Рисунок 1.3 – Cool Retro Term

Cool Retro Term оптимізований для роботи на сучасних системах і підтримує різні платформи, включаючи Linux та macOS. Він побудований на технології QML і вбудований у фреймворк Qt, що забезпечує швидку і стабільну роботу. Інструмент є ідеальним вибором для тих, хто захоплюється естетикою ретро-технологій або хоче додати стильного вигляду до своїх робочих процесів, зберігаючи при цьому функціональність сучасного терміналу.

Deepin Terminal – це сучасний інструмент для взаємодії з командним рядком, який відомий своїм елегантним дизайном, високою продуктивністю та інтеграцією з робочим середовищем Deepin. Він розроблений для користувачів, які цінують як естетику, так і функціональність. Однією з головних особливостей Deepin Terminal є його можливість підтримувати кілька вкладок, що робить роботу з кількома процесами одночасно зручною.

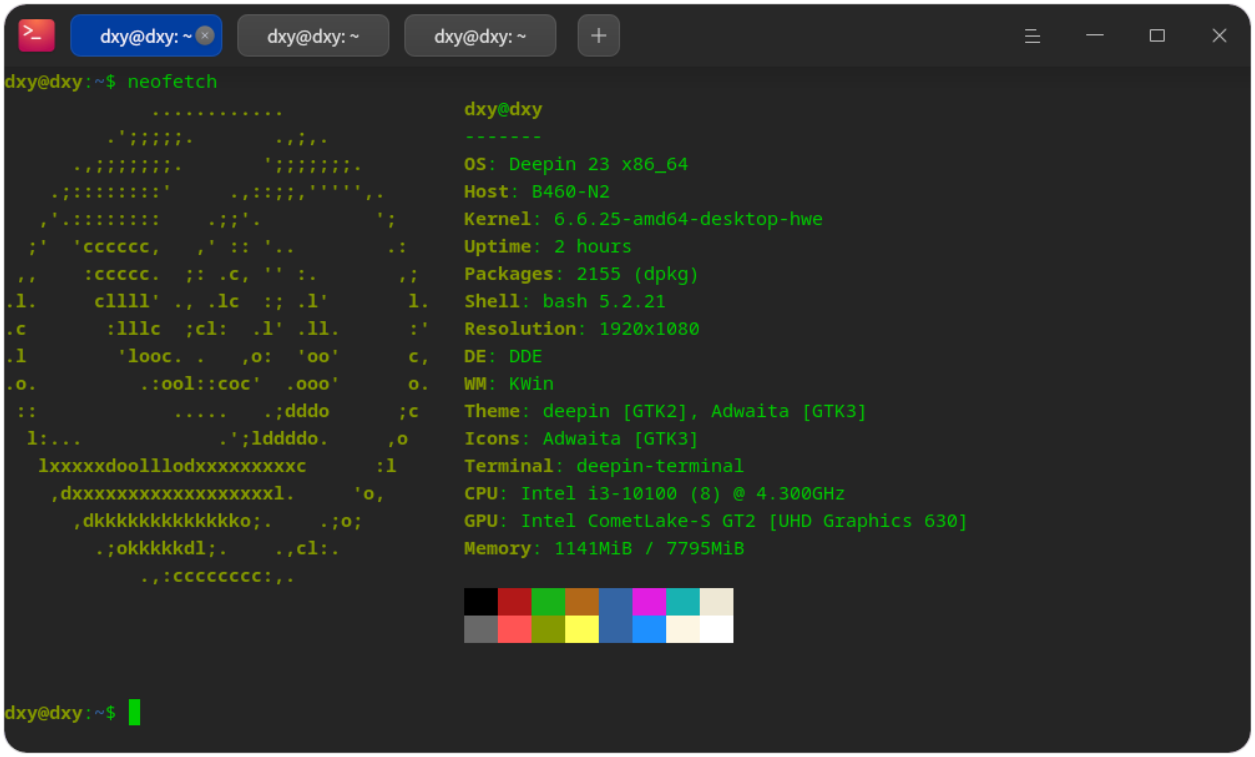

Рисунок 1.4 – Deepin Terminal

Крім того, це програмне забезпечення має функції розділення екрану, які дозволяють розділити вікно на декілька частин для виконання різних команд

паралельно. Інтеграція з файловою системою Deepin робить його чудовим вибором для швидкого доступу до системних інструментів та обробки даних.

Deepin Terminal також підтримує налаштування зовнішнього вигляду, включаючи зміну кольорових схем, шрифтів та фону, що дозволяє користувачам персоналізувати програму під свій стиль роботи. Додатково до цього, він підтримує гарячі клавіші, які прискорюють виконання часто використовуваних команд, і створює максимально зручне середовище для продуктивної роботи.

Ця програма є частиною екосистеми операційної системи Deepin, але може використовуватись і в інших дистрибутивах Linux завдяки відкритому програмному коду. Deepin Terminal залишається вибором багатьох користувачів завдяки своїй простоті у використанні, сучасному інтерфейсу та можливостям, які роблять роботу з терміналом інтуїтивно зрозумілою навіть для новачків.

Extraterm: Це сучасний термінал з розширеними можливостями, який надає широкий набір інструментів для роботи з вашим терміналом та застосунками командного рядка. Він дозволяє користувачам ефективно взаємодіяти з текстовими інтерфейсами, впроваджуючи функції, які не доступні у стандартних терміналах. Зокрема, Extraterm підтримує такі можливості, як зручна організація вкладок, інтерактивний перегляд виводу команд, система маркування для швидкого доступу до важливих секцій тексту, попередній перегляд файлів та інтеграцію з різними інструментами розробника. Завдяки своєму зручному інтерфейсу, мультимедійній підтримці та розширеним функціям, цей інструмент стане незамінним для програмістів, системних адміністраторів та інших IT-фахівців.

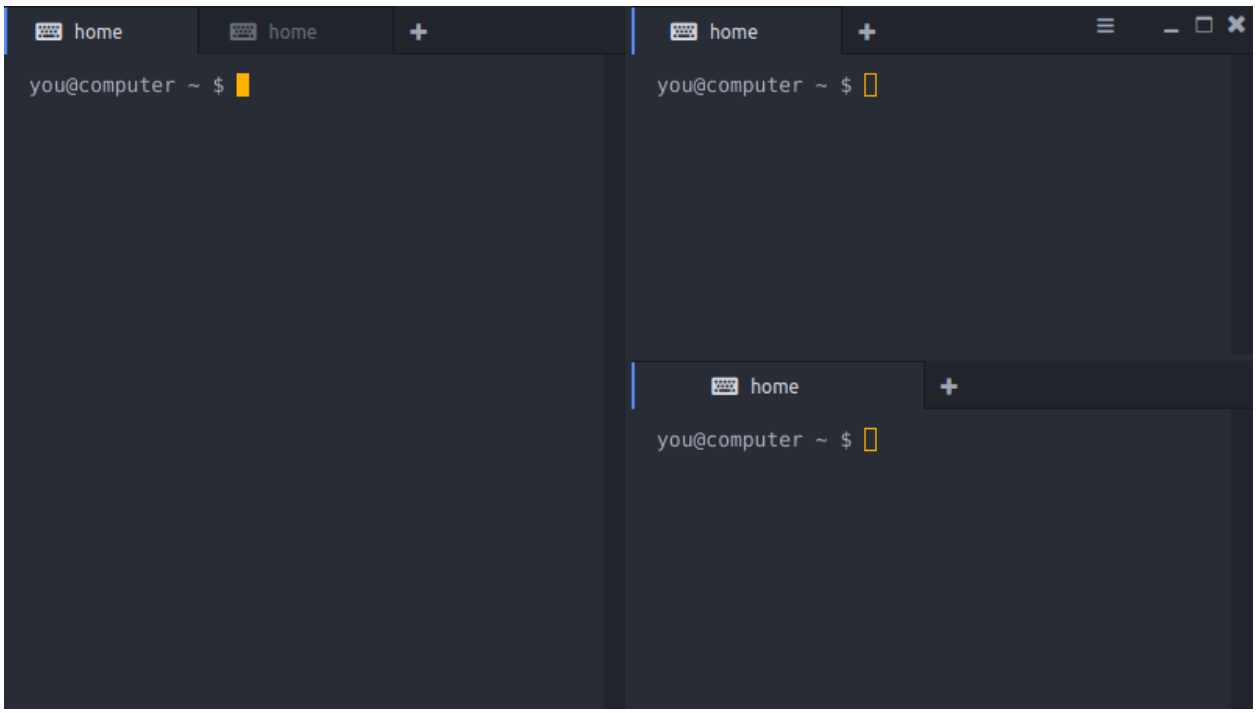

Рисунок 1.5 – Extraterm

Foot: Швидкий емулятор терміналу, оптимізований для роботи в середовищі Wayland. Ця програма створена з акцентом на простоту використання, ефективність та мінімальне споживання системних ресурсів, що ідеально підходить для користувачів, які цінують легкість і швидкодію. Завдяки компактному коду та сучасним технологіям, Foot забезпечує швидкий відгук, плавну роботу і підтримує широкий спектр функцій, таких як кастомізація гарячих клавіш, налаштування інтерфейсу, а також підтримка Unicode і TrueColor.

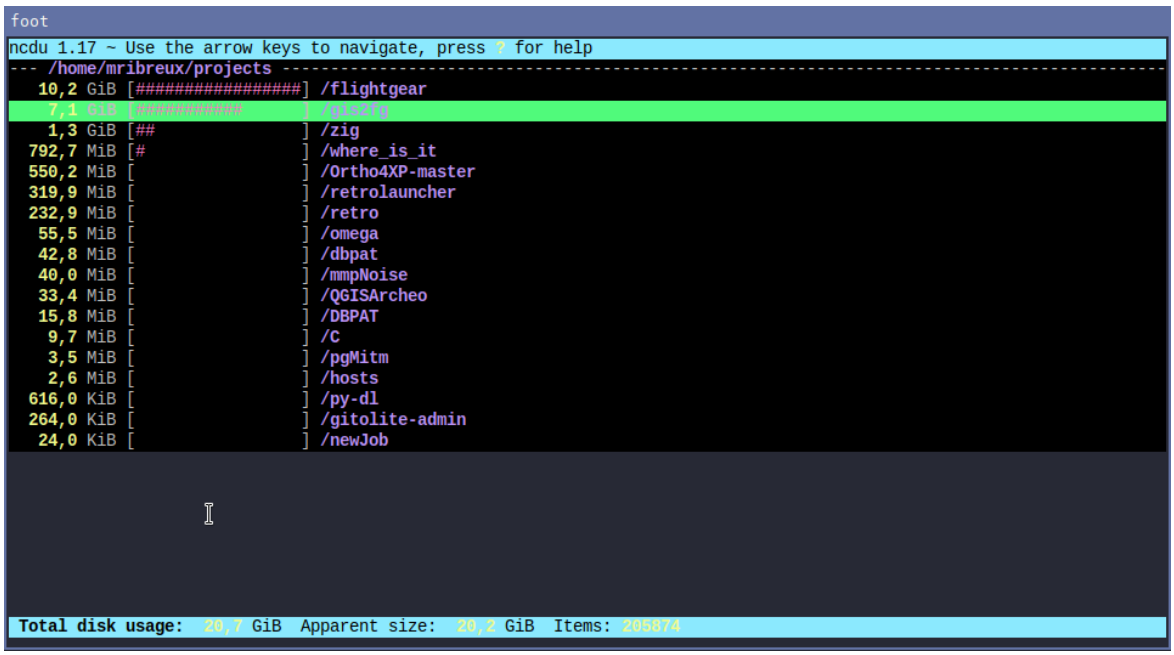

Рисунок 1.6 – Foot

GNOME Terminal – це потужний емулятор термінала, який є стандартною програмою для роботи в командному рядку на GNOME-десктопі, одному з найпопулярніших графічних середовищ Linux. Він надає користувачам швидкий доступ до системних інструментів і функцій через текстовий інтерфейс, забезпечуючи ефективну інтеграцію з іншим програмним забезпеченням GNOME.

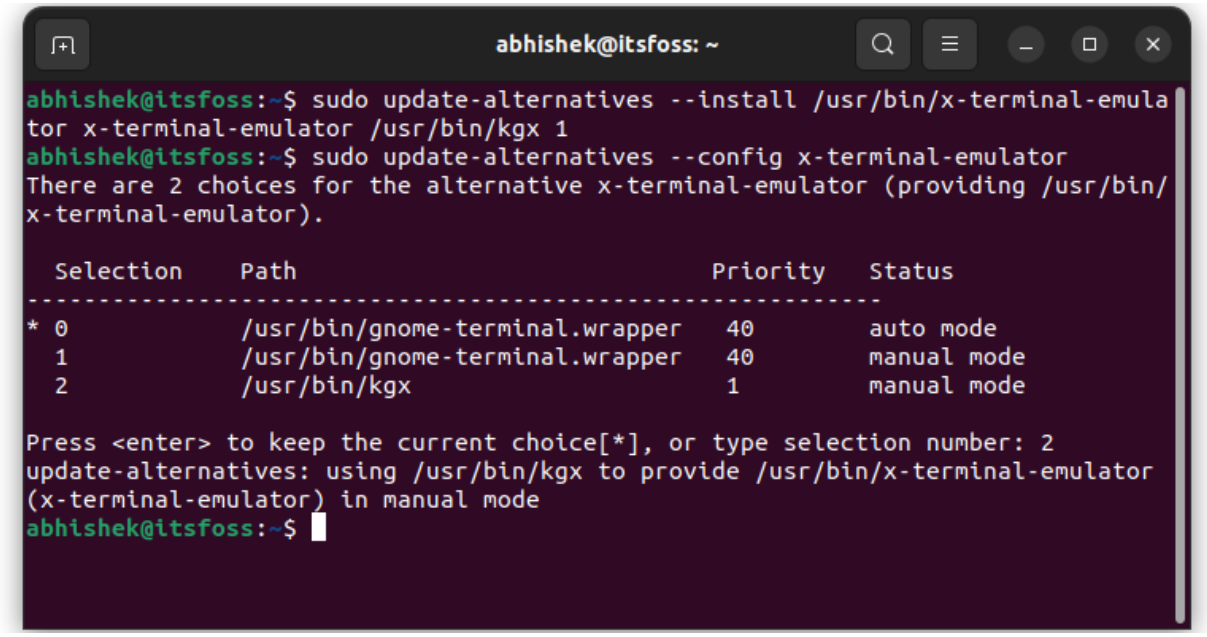

Рисунок 1.7 – GNOME Terminal

Цей термінал вирізняється високою настроюваністю, дозволяючи змінювати його зовнішній вигляд, шрифти, кольорові схеми та гарячі клавіші для оптимізації роботи. GNOME Terminal підтримує вкладки, які значно спрощують одночасну роботу з декількома сесіями командного рядка, а також функцію розділення вікна, щоб переглядати кілька процесів в одному робочому просторі.

Він також пропонує функції, як-от збереження історії команд, можливість створювати скрипти, що автоматизують рутинні завдання, і підтримку розширених налаштувань SSH для віддаленого підключення. Завдяки своєму інтуїтивно зрозумілому дизайну та легкій масштабованості GNOME Terminal стане незамінним інструментом як для новачків, так і для досвідчених користувачів Linux.

Ghostty — це сучасний емулятор терміналу, який відзначається високою швидкістю роботи та широкою функціональністю. Він створений з використанням нативного інтерфейсу платформи, що забезпечує йому органічну інтеграцію в операційну систему та зручність у використанні. Одна з ключових особливостей Ghostty — підтримка апаратного прискорення за допомогою графічного процесора (GPU), що значно підвищує продуктивність і робить роботу з великими обсягами даних швидкою та ефективною.

Емулятор підтримує налаштування під потреби користувача та може працювати з кількома вкладками, надаючи можливість зручно управляти різними сесіями одночасно. Ghostty також підтримує сучасні технології рендерингу, завдяки чому текст, символи та графічні елементи відображаються з високою чіткістю і без затримок. Крім того, інструмент сумісний з популярними оболонками та командними інтерпретаторами, такими як Bash, Zsh, Fish та іншими, що робить його універсальним і гнучким рішенням для розробників, системних адміністраторів і звичайних користувачів.

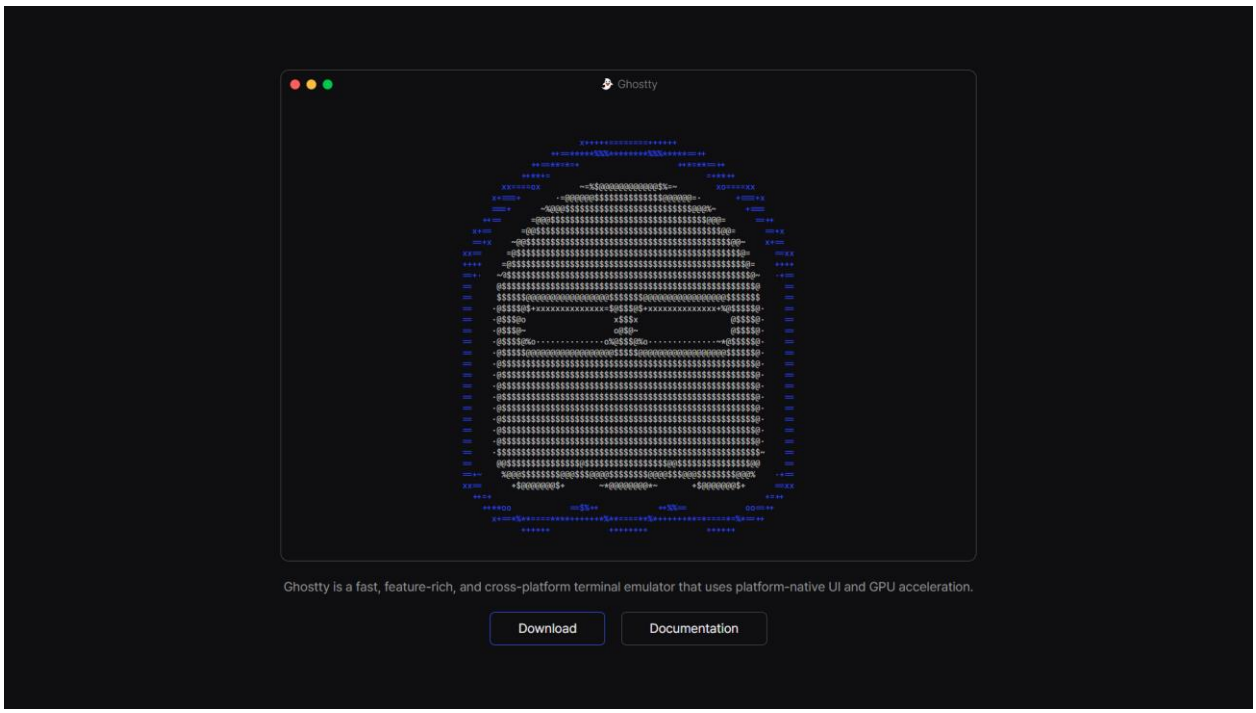

Рисунок 1.8 – Ghostty

Ghostty ідеально підходить для роботи з великими журналами, командами розробників, які потребують швидкої взаємодії з терміналом, або для користувачів, які цінують потужність і простоту в роботі.

Guake – це спливаючий термінал, розроблений спеціально для середовища робочого столу GNOME, який забезпечує зручний і швидкий доступ до командного рядка. Він створений для користувачів, які цінують ефективність і швидке перемикання між завданнями. Основною особливістю Guake є те, що термінал викликається натиском клавіші (зазвичай F12 або іншої заданої гарячої клавіші), після чого він плавно спливає з верхньої частини екрана, нагадуючи консоль у грі.

Guake відомий своїм простим і зрозумілим інтерфейсом, налаштовуваними параметрами зовнішнього вигляду, а також підтримкою функції декількох вкладок, що дозволяє одночасно використовувати кілька сеансів терміналу. Користувачі можуть змінювати розміри вікна термінала, його прозорість та навіть налаштовувати колірну схему відповідно до своїх уподобань.

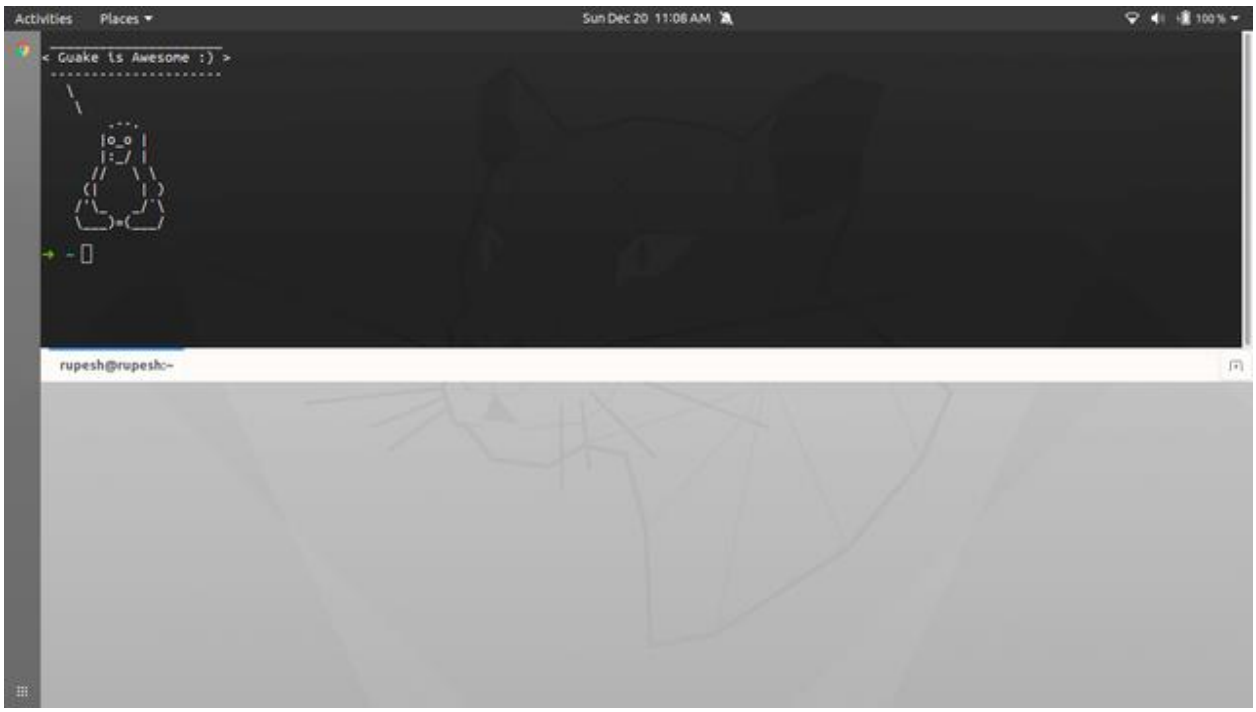

Рисунок 1.9 – Guake

Цей інструмент є ідеальним вибором для розробників, системних адміністраторів та всіх, хто часто працює з командним рядком. Guake підтримує функції копіювання та вставки, швидке перемикання між вкладками за допомогою клавіатури та повну інтеграцію з GNOME, що робить його надзвичайно корисним для виконання завдань у Linux-системах.

Завдяки легкості та гнучкості налаштувань, Guake ефективно вирішує потребу в швидкому доступі до терміналу, водночас не займаючи постійно місця на екрані.

Hyper – це кросплатформенний термінал, розроблений мовами JavaScript, HTML і CSS, що відзначається високою гнучкістю та широкими можливостями для налаштування. Його створено з використанням Electron, що дозволяє працювати з терміналом як з веб-додатком, забезпечуючи інтеграцію сучасних веб-технологій.

Hyper підтримує плагіни та теми, що дає користувачам змогу адаптувати інтерфейс і функціонал під свої потреби. Завдяки відкритому API, розробники можуть створювати власні розширення, додаючи нові функції або модифікуючи існуючі. Він відомий своєю масштабованістю, привабливим

дизайном, а також інтеграцією з інструментами командного рядка та хмарними обчислювальними середовищами.

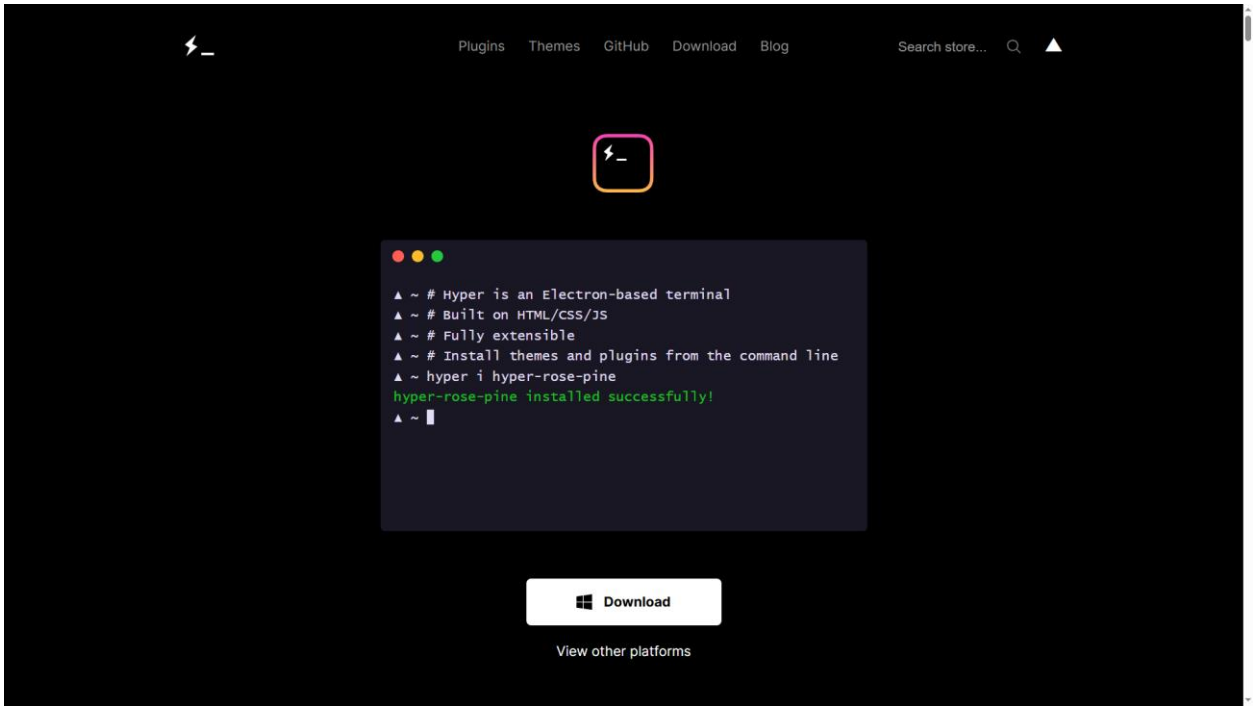

Рисунок 1.10 – Hyper

Крім того, Hyper ідеально підходить для розробників, які працюють в багатозадачному режимі, оскільки дозволяє ефективно управляти вкладками та сеансами, організовувати робочий процес і швидко виконувати різні процеси безпосередньо в терміналі.

Kitty — це сучасний термінальний емулятор, розроблений для роботи на основі GPU, що забезпечує високу швидкість, ефективність та плавність роботи. Завдяки використанню можливостей графічного процесора, Kitty забезпечує покращену продуктивність у порівнянні з традиційними емуляторами, які покладаються виключно на центральний процесор. Kitty підтримує широкий набір функцій, таких як рендеринг зображень і навіть відтворення відео прямо в терміналі, що робить його унікальним серед інших емуляторів.

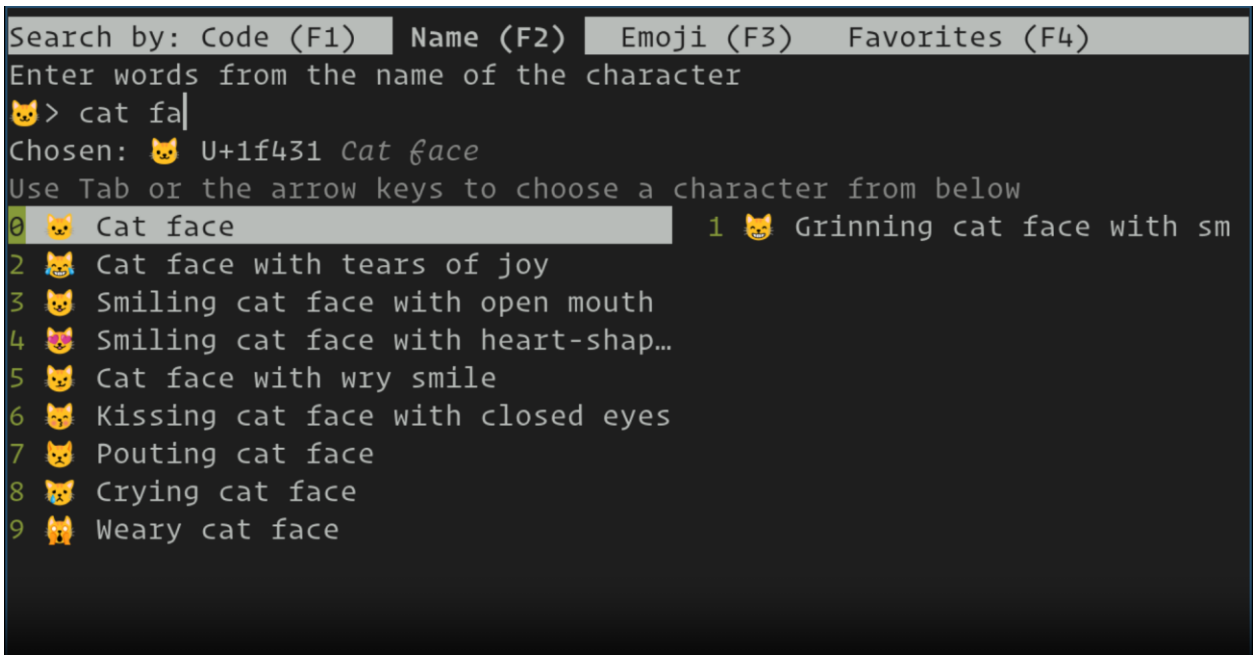

Рисунок 1.11 – Kitty

Цей інструмент підтримує кастомізацію та розширення за допомогою Python-скриптів, дозволяючи налаштовувати робоче середовище під конкретні потреби користувача. Kitty пропонує вдосконалений рендеринг тексту, який гарантує чіткість і якість навіть на дисплеях із високою роздільною здатністю. Крім того, емулятор оптимізований для роботи з сучасними масштабованими дисплеями, такими як 4K або 8K.

Також Kitty має підтримку вкладок, спеціальних макросів, розподілення вікон і багатьох інших функцій, які сприяють продуктивній роботі. Він сумісний із багатьма операційними системами, такими як Linux, macOS і Windows (через WSL), що робить його універсальним вибором для розробників, системних адміністраторів і всіх, хто працює з терміналом.

Konsole – це потужний термінал-емулятор середовища робочого столу KDE, який вирізняється багатофункціональністю, гнучкими налаштуваннями та інтуїтивно зрозумілим інтерфейсом. Однією з головних особливостей цієї програми є підтримка декількох профілів, що дозволяє користувачам налаштовувати зовнішній вигляд, шрифти, кольорові схеми та інші параметри окремо для різних робочих середовищ чи завдань. Інтерфейс із вкладками

надає можливість працювати з кількома терміналами одночасно в межах одного вікна, що допомагає зберігати порядок і швидко перемикатися між різними сеансами.

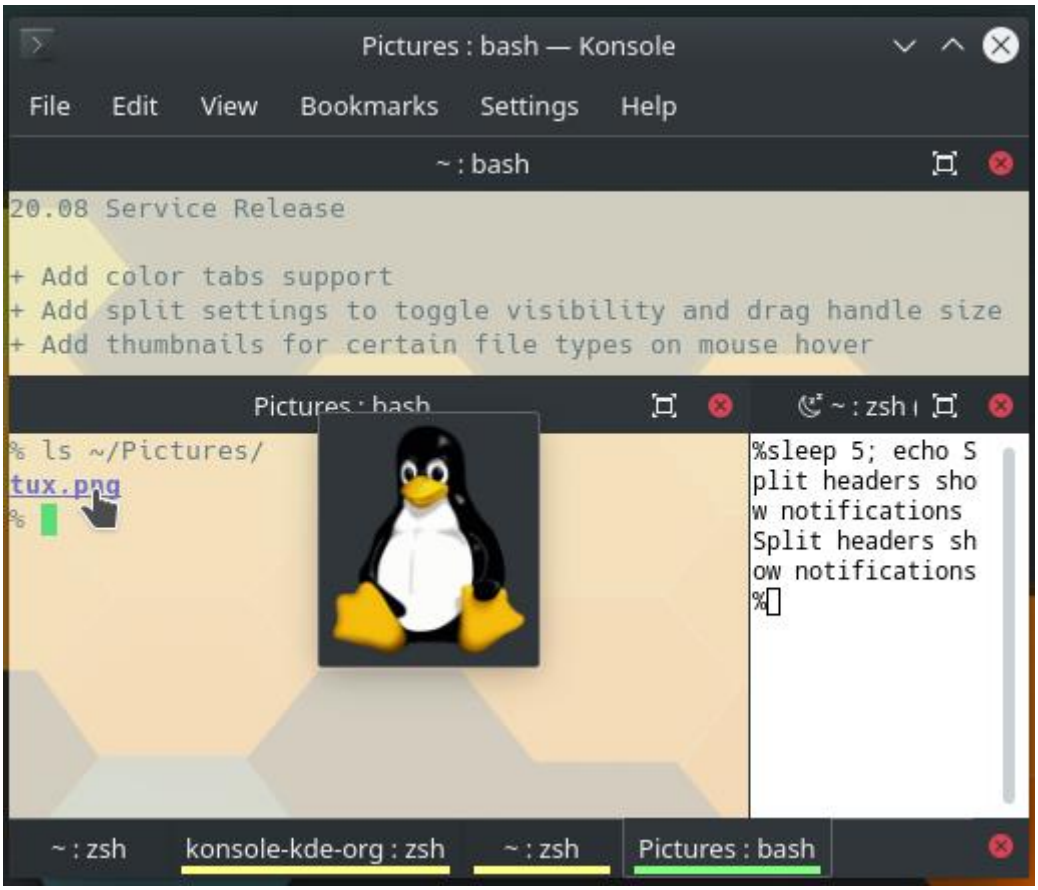

Рисунок 1.12 – Konsole

Окрім цього, Konsole інтегрується з іншими додатками KDE, що робить його зручним для розробників та адміністраторів систем. Наприклад, вікна Konsole можна легко вбудовувати в такі програми, як Kate або Dolphin, для спрощеного доступу до командного рядка під час роботи з текстовими файлами чи файловою системою. Програма підтримує історію команд, клавіатурні скорочення, розділені вікна, а також має можливість виведення вмісту терміналу у форматі тексту або HTML.

Завдяки відкритому коду та активній спільноті розробників Konsole постійно вдосконалюється, отримуючи нові можливості та забезпечуючи стабільну і надійну роботу у різноманітних Linux-дистрибутивах.

LXTerminal — це термінальний емулятор, розроблений спеціально для використання в робочому середовищі LXDE (Lightweight X11 Desktop Environment). Він відзначається легкістю, мінімальними системними вимогами та залежностями, що робить його відмінним вибором для систем із обмеженими ресурсами. У своїй роботі LXTerminal використовує бібліотеку VTE, яка забезпечує базову функціональність емуляції термінала.

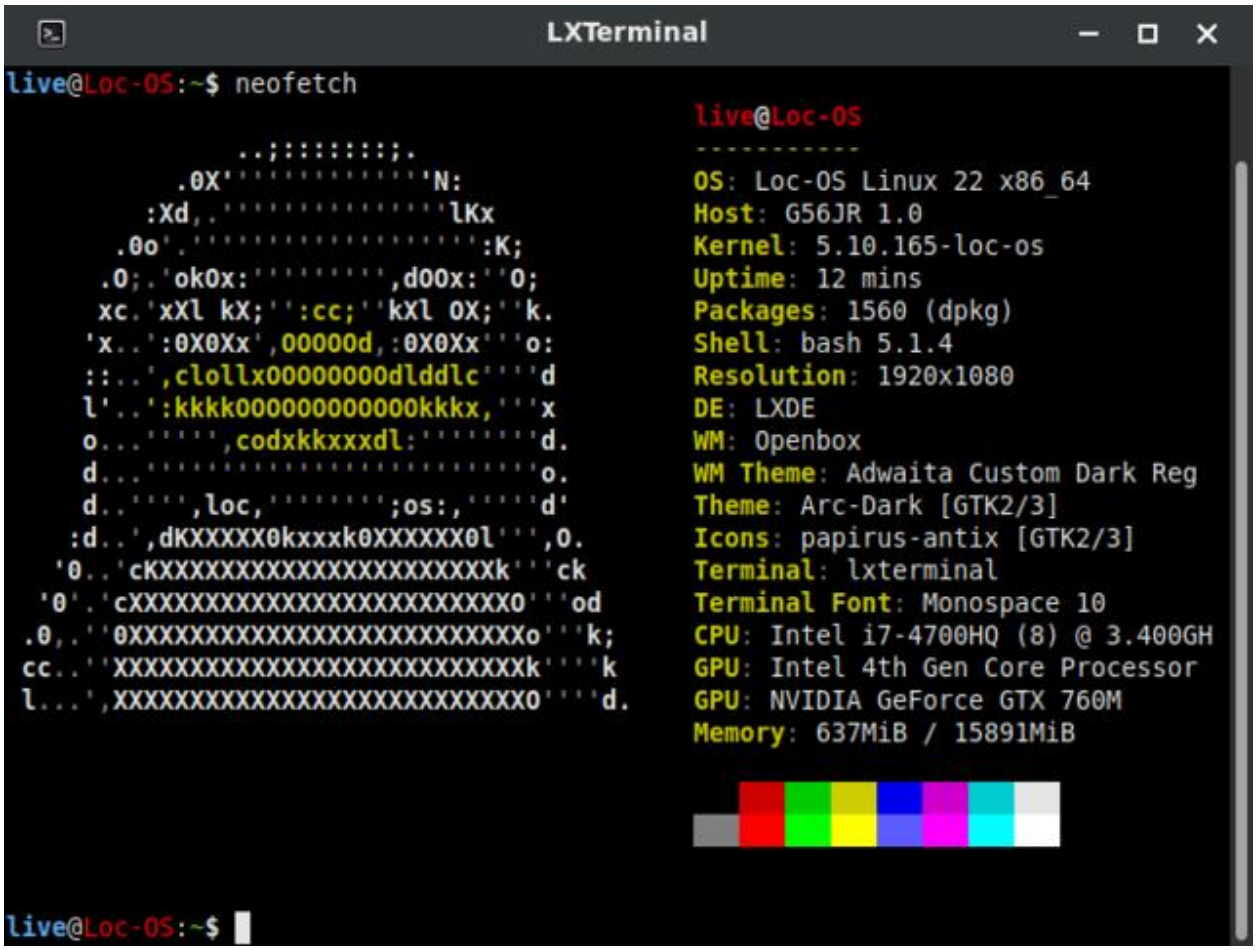

Рисунок 1.13 – LXTerminal

Серед основних особливостей LXTerminal можна виділити:

1. Мінімалістичний дизайн. Інтерфейс програми простий і зрозумілий, без зайвих елементів, що відповідає загальній концепції LXDE як легкого і швидкого середовища.

2. Мінімальні залежності: Програма не вимагає встановлення важких бібліотек або додаткового програмного забезпечення, що сприяє продуктивності і зменшує використання пам’яті.

3. Підтримка вкладок (tabs): Дає змогу працювати з кількома терміналами в одному вікні, надаючи більшу гнучкість для користувачів.

4. Налаштування зовнішнього вигляду: Користувачі можуть змінювати розмір шрифтів, кольорову схему фону та тексту, забезпечуючи комфортний досвід роботи.

5. Інтеграція з LXDE: Повністю підтримує компоненти LXDE, забезпечуючи швидкий і плавний обмін даними з іншими програмами в середовищі.

LXTerminal є ідеальним вибором для користувачів, яким важлива ефективність роботи на слабких системах, таких як старі комп'ютери або пристрої з обмеженими апаратними можливостями.

Mate-Terminal – це сучасний ейтермінал, який є частиною середовища робочого столу MATE, розробленого для забезпечення зручної роботи в системах Linux. Він сумісний з різними версіями Linux, забезпечуючи широкий функціонал і можливість тонкого налаштування параметрів. Завдяки своїй інтуїтивно зрозумілій структурі, Mate-Terminal дозволяє користувачеві легко виконувати текстові команди, запускати скрипти та взаємодіяти з іншими компонентами операційної системи через командний рядок.

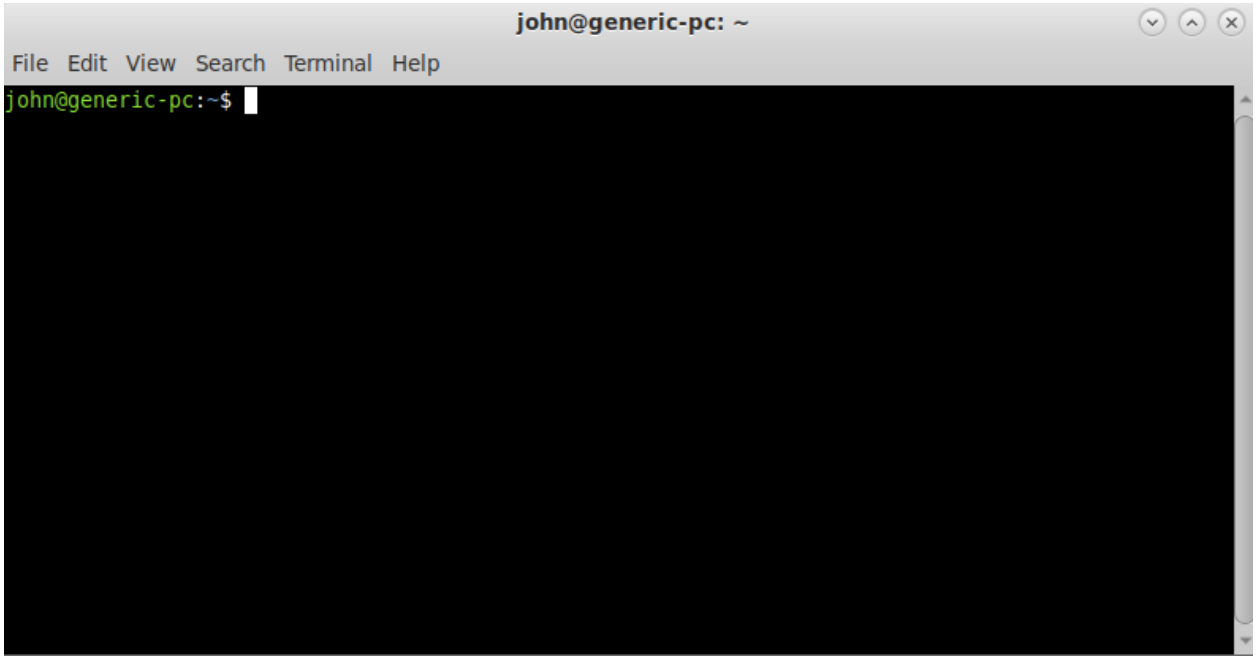

Рисунок 1.14 – Mate-Terminal

Ключовими особливостями Mate-Terminal є його гнучкість у налаштуваннях. Користувач може змінювати шрифти, кольорову схему, поведінку вкладок і навіть встановлювати бажаний рівень прозорості вікна термінала. Ця програма підтримує роботу з кількома вкладками, що дозволяє ефективно виконувати паралельні процеси та організовувати робочі сесії. Крім того, розробники приділили значну увагу продуктивності, забезпечивши швидке виконання команд навіть під високим навантаженням.

Серед переваг Mate-Terminal також слід зазначити його простоту інтеграції з іншими компонентами середовища MATE. Це дозволяє досягти гармонійної взаємодії між програмами, полегшуючи виконання завдань на рівні користувача та системного адміністратора. Зокрема, термінал підходить для роботи як новачкам, так і досвідченим користувачам Linux, надаючи їм інструмент для виконання різноманітних операцій, таких як конфігурування системи, управління файлами, моніторинг процесів і багато іншого.

Mate-Terminal активно підтримується спільнотою розробників, що гарантує його стабільність і актуальність. Завдяки підтримці відкритого коду, ця програма постійно вдосконалюється й оновлюється, зберігаючи високі стандарти якості й сумісності з новими версіями Linux та функціями.

mlterm – це багатофункціональний, гнучкий емулятор термінала для систем X11, який створений для роботи з текстом, що використовує різні набори символів та мови. Основна особливість mlterm полягає в його здатності забезпечувати якісний рендеринг тексту з підтримкою широкого спектра мов, включаючи ті, що використовують складні сценарії написання, як-от китайська, японська, арабська та інші.

Цей емулятор термінала підтримує Unicode, що дозволяє відображати різноманітний текст на основі розширених символів. mlterm також сумісний із технологіями таких міжнародних стандартів, як UTF-8, і підтримує одночасний багатомовний ввід, що є особливо корисним для поліглотів або для роботи в мультикультурному середовищі.

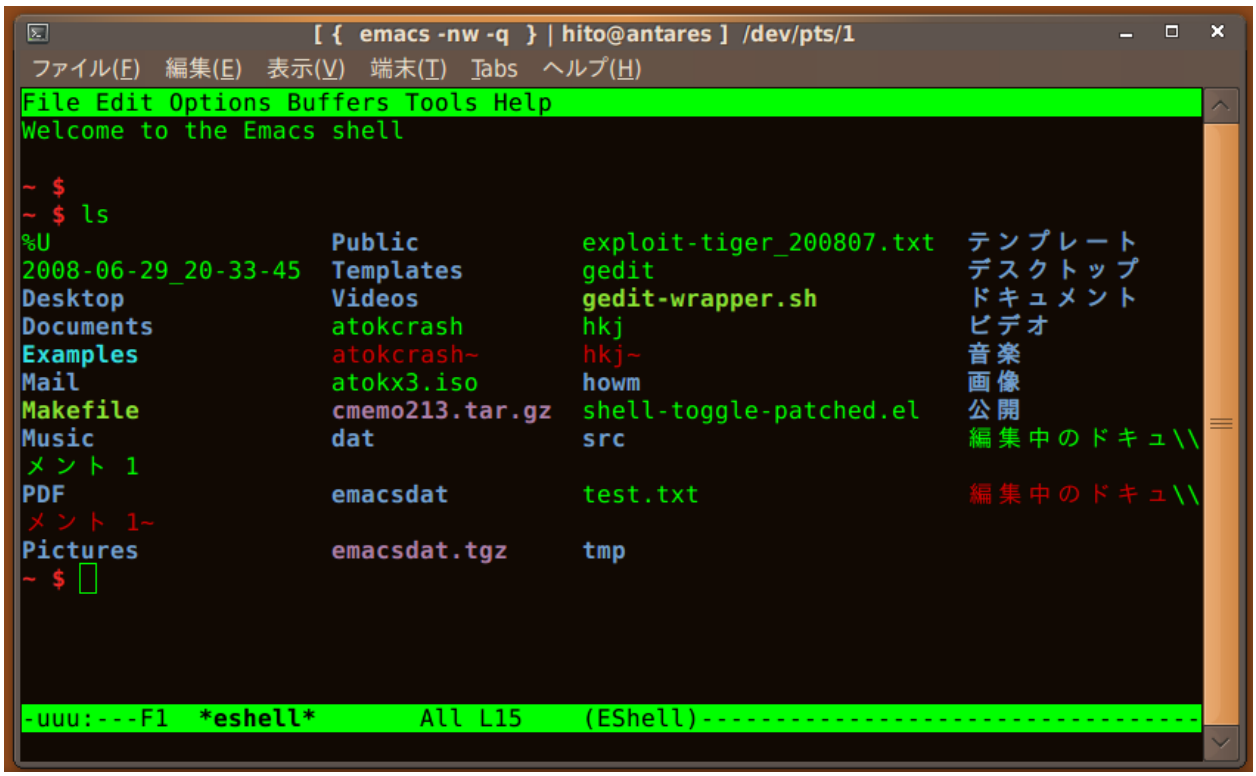

Рисунок 1.15 – mlterm

Крім цього, mlterm має широкий спектр налаштувань, зокрема зміну шрифтів, кольорових схем і параметрів інтерфейсу користувача. Він підтримує функції масштабування вікна, прозорість, а також апаратне прискорення для покращення продуктивності. Завдяки цим характеристикам mlterm стає ефективним інструментом для користувачів, які працюють із різними мовами або завданнями, що вимагають точного відображення тексту.

Це програмне забезпечення популярне серед тих, хто шукає простий, але водночас потужний засіб для роботи з терміналом у середовищах Linux та Unix, де багатомовність та підтримка складних наборів символів є критично важливими.

Ptyxis: Сучасний високопродуктивний термінал-емулятор, що використовує апаратне прискорення GPU для забезпечення максимальної швидкості та плавності роботи. Ця програма створена для використання в середовищах, де продуктивність і ефективність відіграють ключову роль, особливо при роботі з великими обсягами даних або складними командними сесіями. Ptyxis розроблений на мові програмування Rust, що гарантує високу

стабільність, безпеку і продуктивність роботи. Завдяки технологіям GPU-рендерингу, термінал демонструє швидку обробку графічних і текстових елементів, що особливо корисно для розробників, системних адміністраторів і користувачів, які цінують продуктивність в інструментах командного рядка.

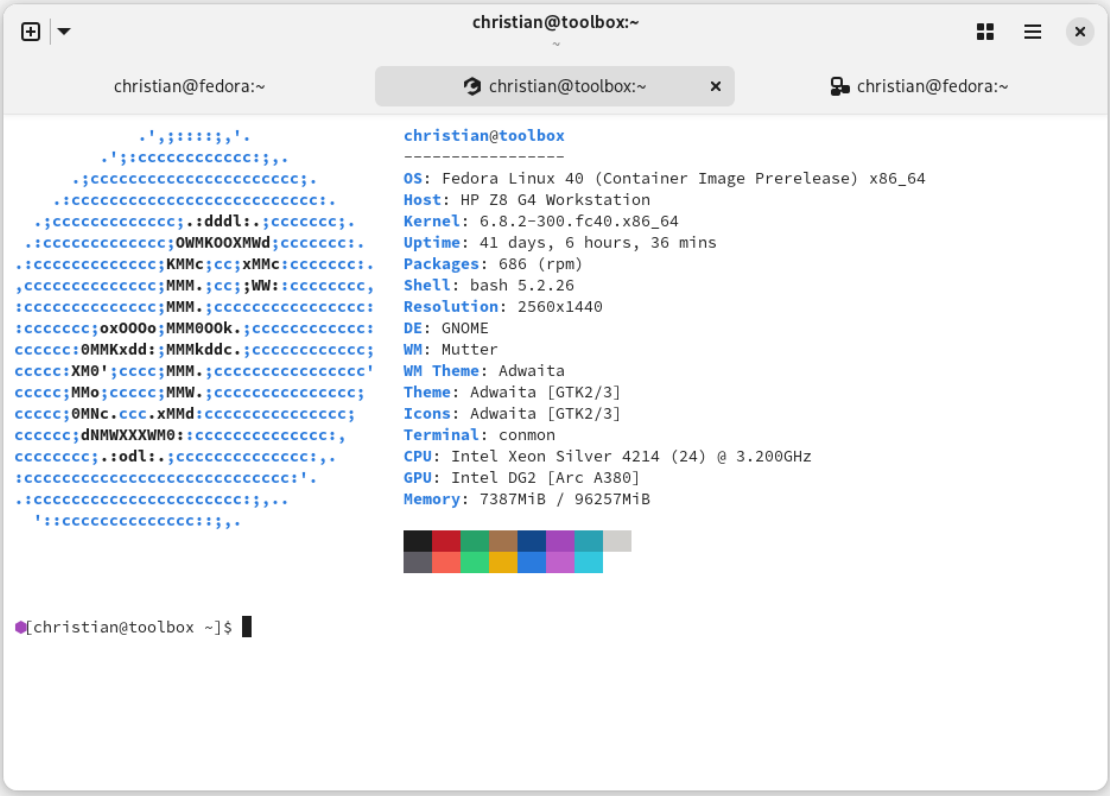

Рисунок 1.17 – Ptyxis

QTerminal — це легкий емулятор термінала, створений на основі бібліотеки Qt, який часто використовується разом із середовищем робочого столу LXQt. Завдяки своїй простоті та функціональності, QTerminal ідеально підходить для користувачів, які шукають мінімалістичне рішення для роботи з командним рядком на Linux.

Основними перевагами цього емулятора є швидкість, низьке споживання системних ресурсів і інтеграція з LXQt, хоча він також може використовуватися незалежно від цього середовища. QTerminal підтримує розділення вікна на вкладки та панелі, що дозволяє виконувати кілька задач

одночасно. Інструмент також надає можливість налаштовувати шрифти, кольори та гарячі клавіші, що допомагає адаптувати програму під індивідуальні потреби користувача.

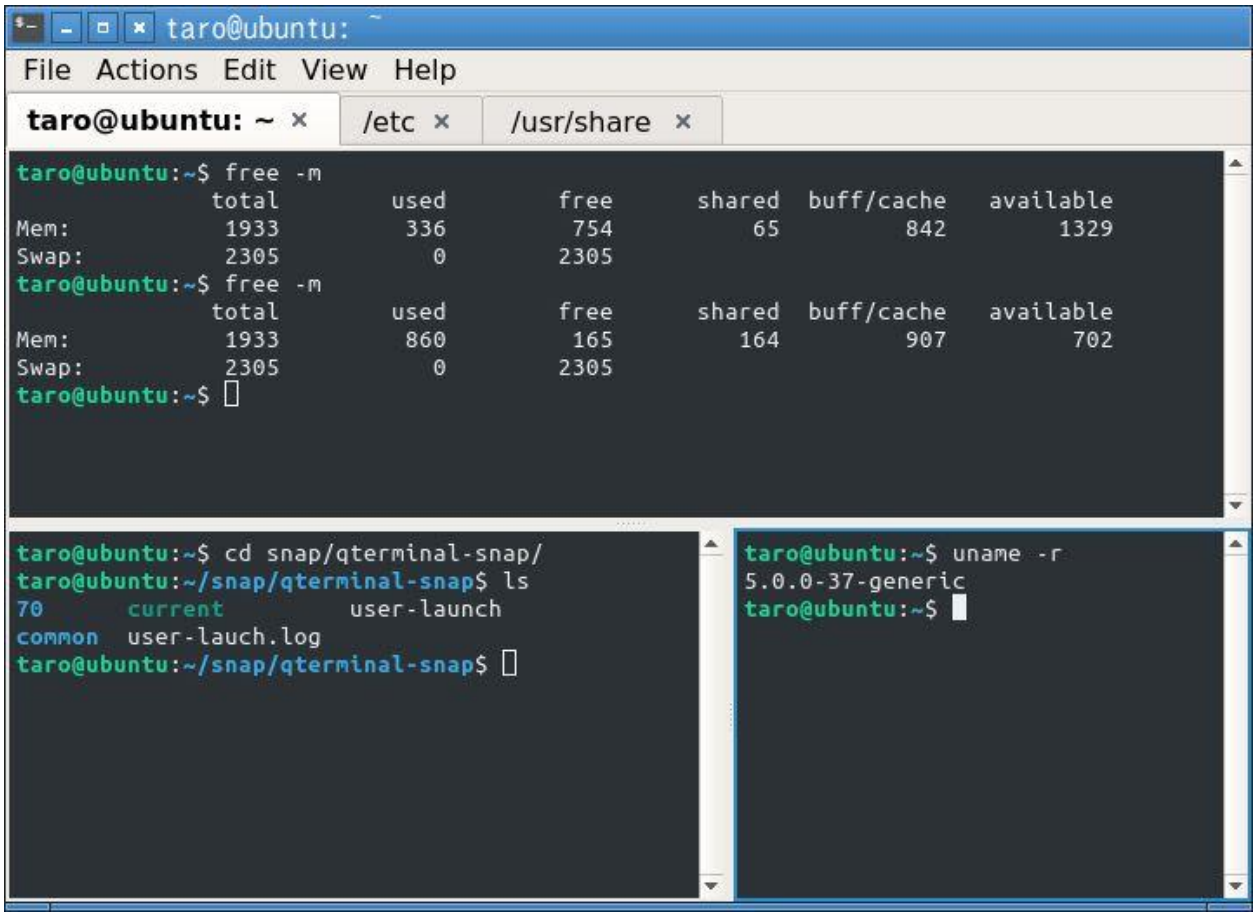

Рисунок 1.18 – QTerminal

Серед додаткових функцій варто відзначити підтримку Unicode, можливість вибору кодової таблиці, автоматичне копіювання тексту при виділенні, роботу з буфером обміну, а також гнучку систему налаштувань. Завдяки своїй модульності, програма легко масштабована, що робить її ефективним інструментом як для новачків, так і для досвідчених користувачів.

QTerminal є частиною LXQt, але його не обов'язково використовувати тільки у цьому середовищі — програма добре працює і з іншими дистрибутивами Linux. Крім того, завдяки відкритому вихідному коду, QTerminal активно розвивається та вдосконалюється спільнотою розробників.

Remmina – це багатофункціональний відкритий клієнт для віддаленого доступу, зокрема до серверів Windows і Unix/Linux, через протокол RDP (Remote Desktop Protocol). Ця програма побудована спеціально для користувачів Linux/Unix, але вона також підтримує інші протоколи, такі як VNC, SSH, NX, XDMCP та SPICE, що розширює її можливості і дозволяє працювати з різними типами віддалених серверів і робочих станцій. Крім того, Remmina може виконувати функції емулятора терміналу, забезпечуючи безпечний доступ до віддалених SSH-серверів, що робить її зручним інструментом для адміністраторів систем і розробників програмного забезпечення. Інтерфейс програми орієнтований на максимальну зручність використання, пропонуючи можливість створення, збереження та швидкого підключення до множинних сеансів.

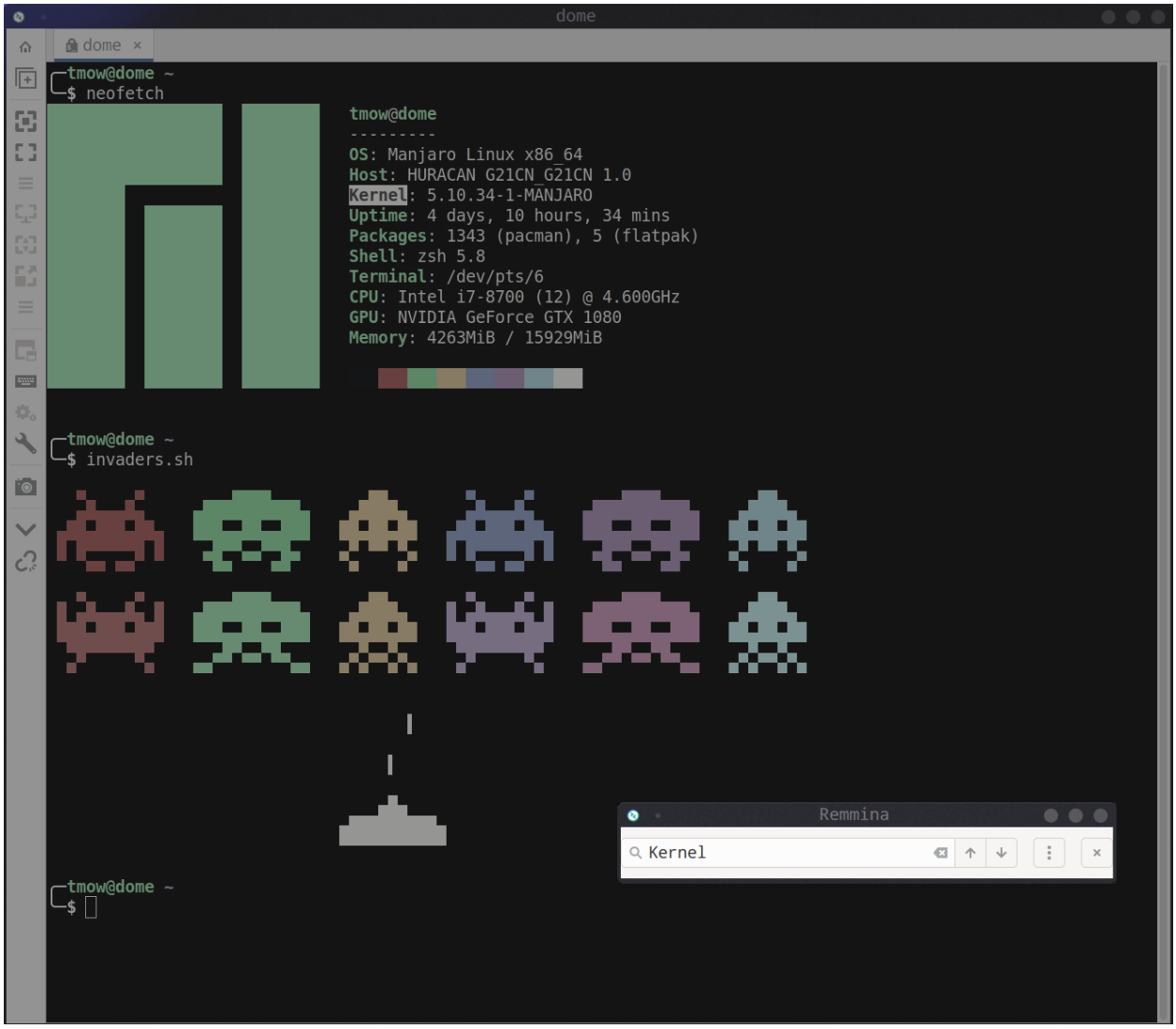

Рисунок 1.19 – Remmina

Roxterm: Термінальний емулятор на основі GTK, створений для користувачів, які шукають потужний і гнучкий інструмент для роботи в командному рядку. Roxterm підтримує налаштування вигляду та поведінки відповідно до індивідуальних потреб, зокрема зміну шрифтів, кольорів і прозорості вікна. Його багатовкладкове середовище дозволяє легко перемикатися між різними сеансами, а функція групування вкладок спрощує організацію роботи. Roxterm також славиться швидким завантаженням і низьким споживанням ресурсів, що робить його ідеальним для користувачів, які працюють на менш потужному обладнанні або потребують ефективності в інтенсивних робочих процесах. Простота інтеграції із зовнішніми утилітами і скриптами дозволяє розширити функціональність терміналу, задовольняючи потреби професіоналів і вимогливих користувачів.

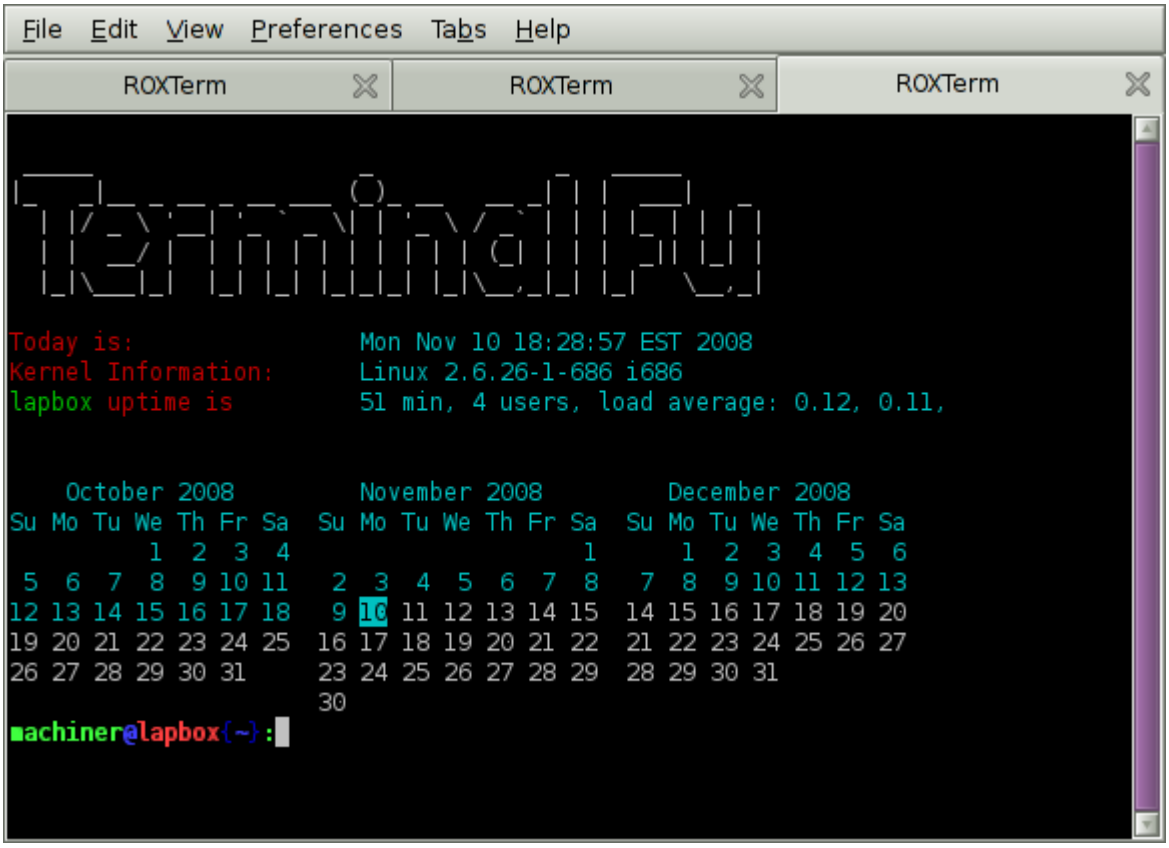

Рисунок 1.20 – Roxterm

St (Simple Terminal) — це мінімалістичний емулятор термінала, розроблений командою Suckless, яка спеціалізується на створенні легких, простих та ефективних програм без зайвої надмірності. Основна мета цього

емулятора — забезпечити базову функціональність термінала, не жертвуючи продуктивністю чи зручністю користування. St виділяється своєю надзвичайно малою кількістю вихідного коду та компактною архітектурою, що робить його ідеальною опцією для любителів мінімалізму та розробників, які хочуть мати контроль над інструментами, які вони використовують.

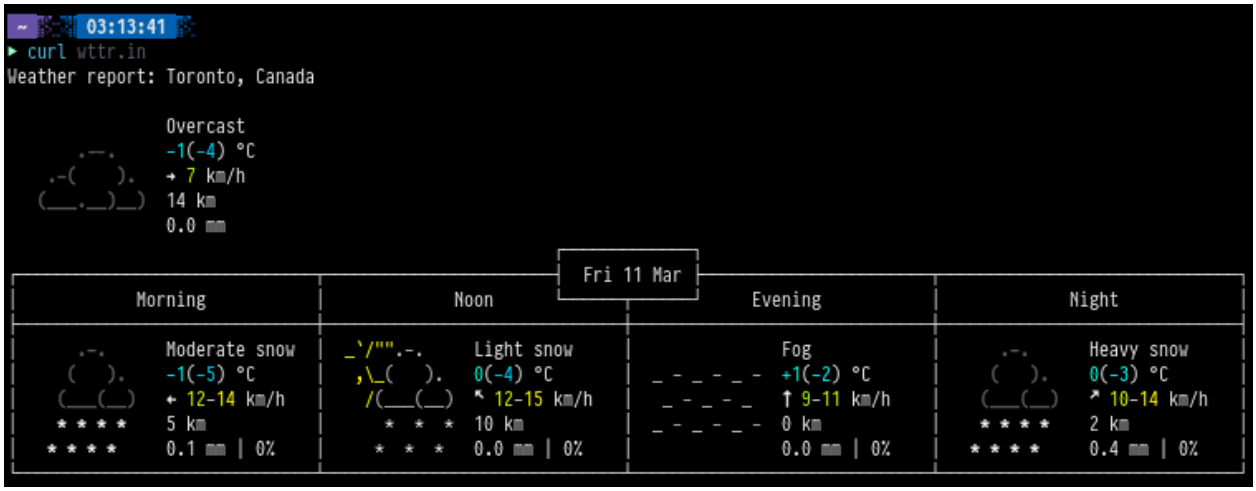

Рисунок 1.21 – St (Simple Terminal)

Багато користувачів віддають перевагу St завдяки його швидкодії та можливості глибокого налаштування. У порівнянні з іншими емуляторами терміналу, такими як Alacritty, GNOME Terminal чи Konsole, St не має графічного інтерфейсу для конфігурації і всі зміни в налаштуваннях вносяться безпосередньо у вихідний код програми перед його компіляцією. Це дає широкі можливості для кастомізації, хоча й вимагає базових знань програмування з боку користувача.

Програма підтримує основні функції, необхідні для роботи в терміналі, включаючи Unicode, кольорові схеми, копіювання і вставку тексту, а також інтеграцію з іншими інструментами. Завдяки модульній структурі, St легко розширюється за рахунок патчів, які розробляє спільнота.

St ідеально підходить для тих, хто цінує максимальну продуктивність, мінімалістичний підхід та стабільність. Він є популярним вибором серед користувачів, які працюють з операційними системами на основі UNIX,

наприклад, Linux і BSD, та шукають простий і настроюваний інструмент для роботи у командному рядку.

Tabby – це сучасний багатоплатформний термінал, створений для того, щоб забезпечити користувачам зручність, швидкість і функціональність під час роботи з командним рядком. Головними перевагами Tabby є його простота у використанні, стильний інтерфейс та можливість гнучкої настройки відповідно до потреб користувача.

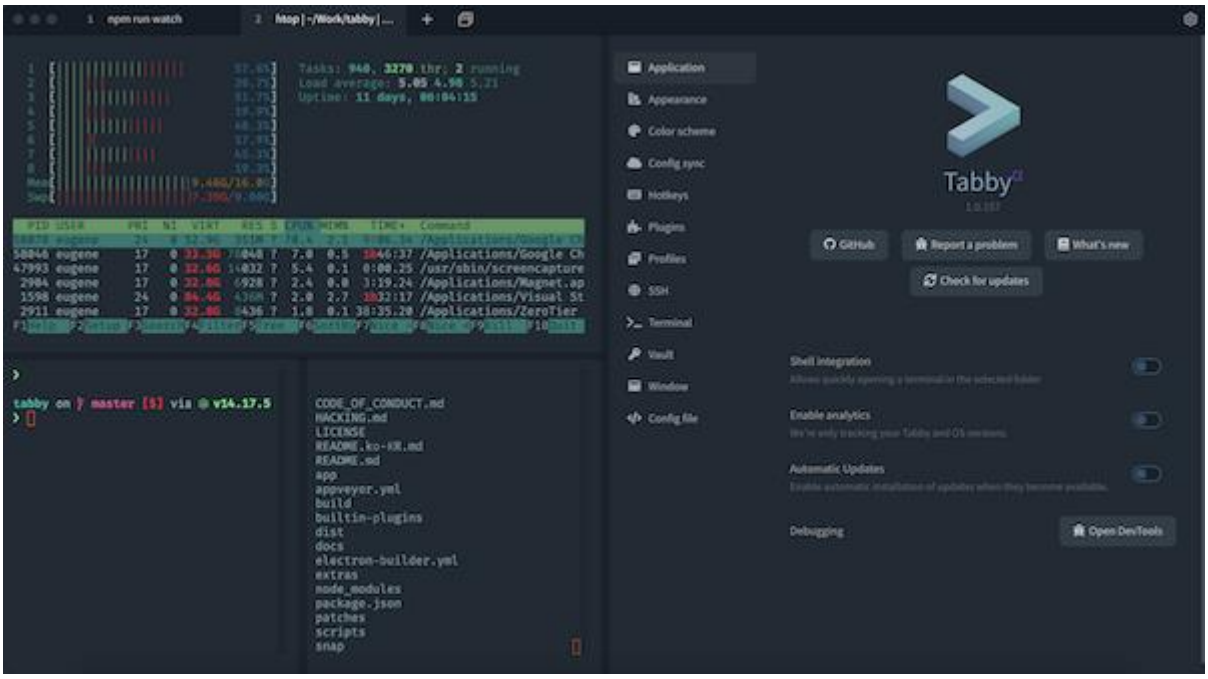

Рисунок 1.22 – Tabby

Цей інструмент підтримує інтеграцію з різними платформами, зокрема Windows, macOS і Linux, що робить його універсальним рішенням для розробників, системних адміністраторів та всіх, хто взаємодіє з терміналом. Завдяки легкому та інтуїтивно зрозумілому дизайну, Tabby об'єднує потужний функціонал з естетикою, забезпечуючи комфортну роботу.

Серед ключових особливостей Tabby:

- підтримка вкладок і багатооконного режиму, що дозволяє ефективно працювати з кількома процесами одночасно;

- можливість кастомізації через зміну тем, шрифтів та кольорових схем;

- інтеграція з SSH, що робить його ідеальним для дистанційного управління серверами;

- відмінна продуктивність навіть на комп'ютерах із середніми ресурсами;

- модулярна структура, яка надає можливість розширення функціоналу за допомогою плагінів.

Будучи відкритим програмним забезпеченням, Tabby активно розвивається спільнотою розробників, що гарантує постійне оновлення та впровадження нових технологій. Це ідеальний вибір для тих, хто цінує сучасний підхід до роботи в терміналі та прагне підвищити продуктивність своєї щоденної діяльності.

Terminator – це потужний і зручний емулятор термінала, який підтримує функцію одночасного відкриття кількох сеансів у межах одного вікна, завдяки чому користувач може розділяти екран на кілька панелей.

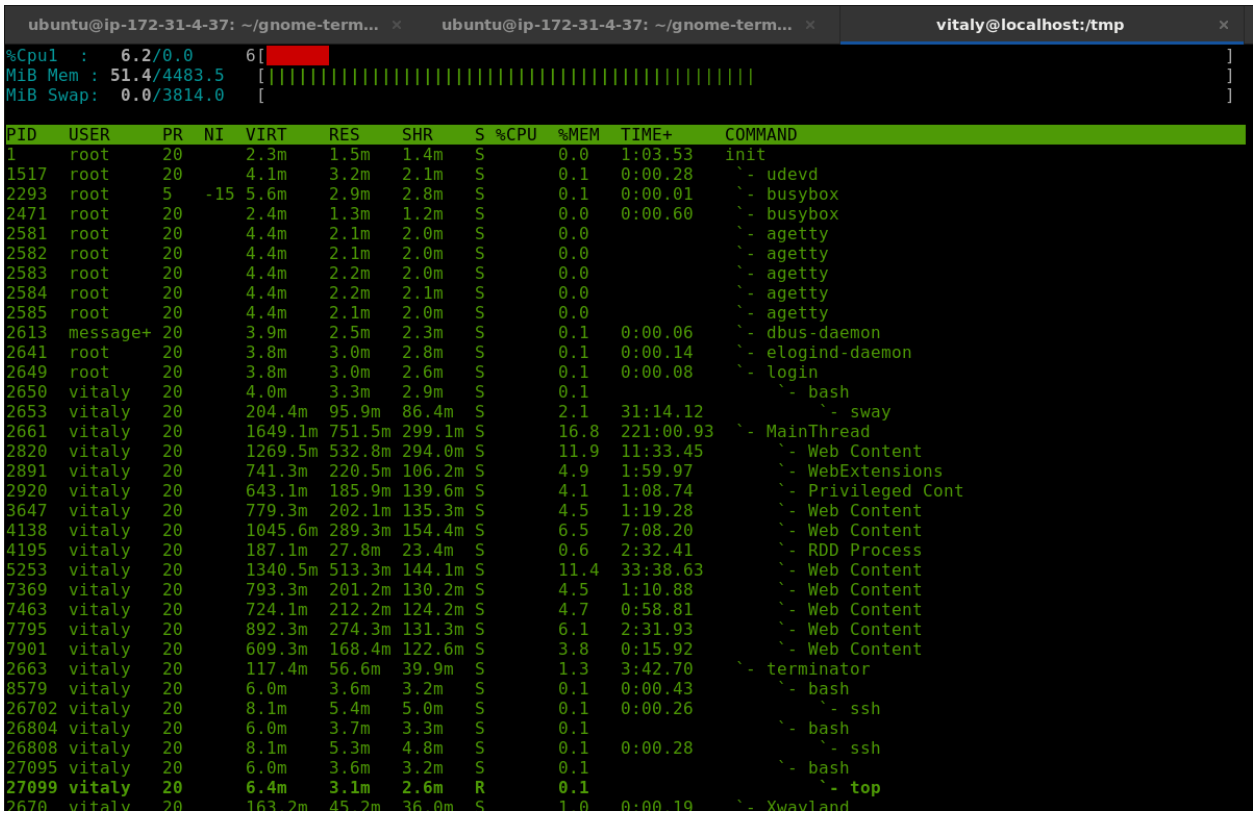

Рисунок 1.23 – Terminator

Ця можливість забезпечує зручну роботу з різними термінальними вікнами відразу, зберігаючи час і підвищуючи продуктивність. Крім того,

Terminator надає широкі можливості по налаштуванню гарячих клавіш, що дозволяє швидко змінювати конфігурацію, відкривати нові панелі, копіювати чи переміщувати текст, а також миттєво виконувати інші часто вживані операції. Інтерфейс програми інтуїтивно зрозумілий і підходить як новачкам, так і досвідченим користувачам, які потребують ефективного інструменту для багатозадачної роботи в командному рядку.

Terminology – це сучасний термінальний емулятор, спеціально створений для робочого середовища Enlightenment (E). Він вирізняється багатофункціональністю та інтерактивним інтерфейсом, який надає користувачеві широкий спектр можливостей, недоступних у стандартних термінальних емуляторах.

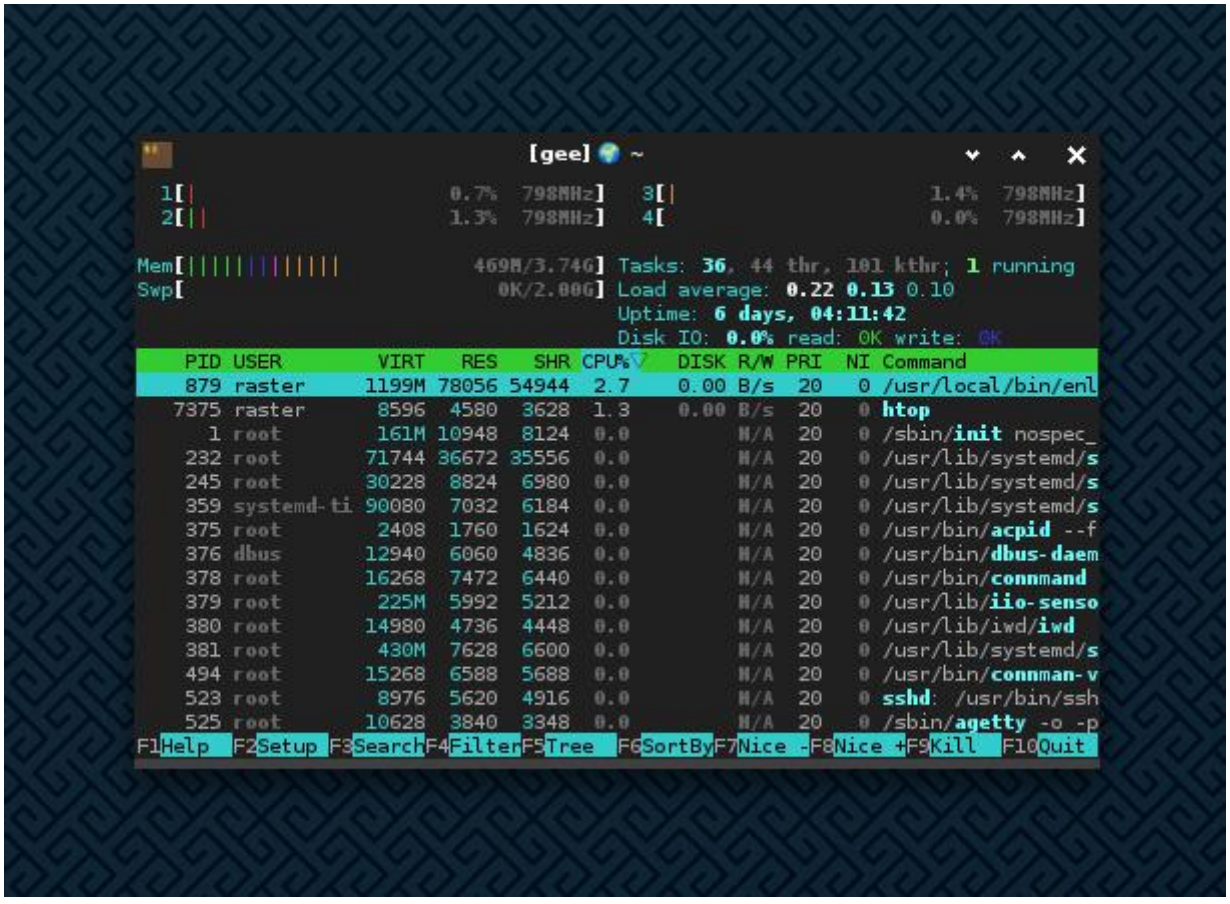

Рисунок 1.24 – Terminology

Terminology дозволяє переглядати зображення, відео, аудіофайли та інші мультимедійні дані прямо в терміналі. Це забезпечує зручність роботи з файлами, не потребуючи використання окремих програм для їх відкриття.

Вбудована підтримка мультимедіа:

- Можливість відтворення відео і аудіо прямо у вікні термінала.

- Відображення зображень у форматах PNG, JPEG, GIF тощо без необхідності виходу зі середовища термінала.

Підтримка посилань та інтерактивних елементів:

- Terminology може розпізнавати веб-посилання, які можна відкривати безпосередньо з терміналу.

- Використання інтерактивних команд дозволяє змінювати робочі опції, виконувати дії над файлами та оперативно отримувати інформацію.

Підтримка тем та графічних ефектів:

- Користувачі можуть налаштувати вигляд термінала за допомогою тем, змінюючи кольорову схему, прозорість і анімаційні ефекти.

- Можливість адаптації інтерфейсу для зручності роботи.

Функціональність вкладок:

- Термінологія дозволяє працювати з декількома вкладками одночасно, що значно підвищує ефективність роботи в багатозадачному режимі.

Легке налаштування та персоналізація:

- Індивідуальне налаштування гарячих клавіш, шрифтів, кольорів і розташування для максимального комфорту використання.

- Дружній інтерфейс для швидкого доступу до всіх функцій.

Продуктивність:

- Швидкість обробки запитів і завдань порівняно з іншими емуляторами термінала завдяки оптимізованій архітектурі програми.

Кросплатформенність:

Terminology повністю сумісна з Linux та іншими операційними системами, які підтримують Enlightenment, що забезпечує гнучкість у виборі платформи.

Terminology є прекрасним вибором для користувачів Enlightenment, які шукають потужне і багатофункціональне термінальне середовище, що виходить за рамки стандартних рішень. Це інструмент, який поєднує в собі

технологічні нововведення, швидкість роботи і простоту використання, створюючи комфортне робоче середовище.

Termius – це сучасний багатоплатформовий емулятор терміналу з вбудованим SSH-клієнтом, орієнтований на професійне віддалене управління серверами, мережевим обладнанням та іншими пристроями. Цей інструмент підтримує роботу на пристроях з ОС Windows, macOS, Linux, а також на мобільних платформах Android та iOS, що забезпечує зручність і доступність незалежно від місця чи умов використання.

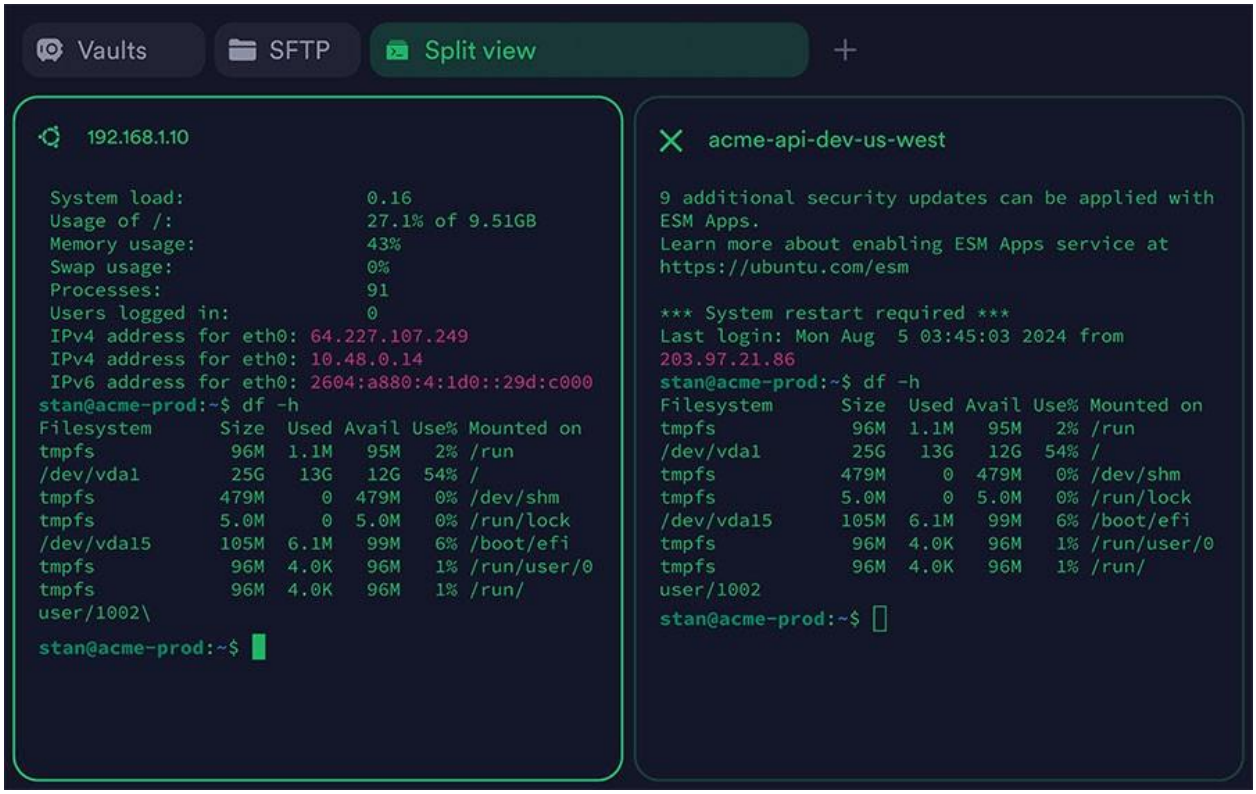

Рисунок 1.25 – Termius

Основною перевагою Termius є можливість зручної синхронізації ключів SSH, налаштувань, інформації про хости та сесії між різними пристроями через захищений хмарний обліковий запис. Програма дозволяє створювати і впорядковувати профілі хостів, групувати їх, налаштовувати індивідуальні параметри підключення, такі як проксі чи порти, та зберігати команди для швидкого доступу.

Додаткові функції Termius включають підтримку протоколів Telnet, SCP і Mosh, двофакторну автентифікацію для підвищення безпеки, можливість використання кастомізованих кольорових схем та шрифтів для зручного відображення. Інструмент має також інтегровану функцію SFTP для передачі файлів між системами.

Для комерційного використання доступна преміум-версія, яка надає розширені функції, такі як зберігання необмеженої кількості хостів, доступ до історії сесій, автозавершення команд, використання проксі SOCKS та інші інструменти для підвищення ефективності роботи.

Завдяки своєму інтуїтивному інтерфейсу та широкому функціоналу Termius став популярним вибором серед системних адміністраторів, DevOps-інженерів, розробників та інших IT-фахівців, які потребують зручного та безпечного рішення для віддаленого управління і моніторингу.

Termux – це потужний інструмент, який надає середовище Linux для пристроїв на базі Android, дозволяючи користувачам запускати командний рядок та встановлювати різні пакети. Завдяки цьому додатку, ваш смартфон або планшет може перетворитися на портативний інструмент для програмування, системного адміністрування, навчання або інших завдань, що потребують використання Linux.

Основні особливості Termux:

1. Мінімалістичне середовище Linux: Додаток надає доступ до стандартного командного рядка, дозволяючи виконувати базові системні команди.

2. Управління пакетами: Використовує власний менеджер пакетів `pkg` (заснований на apt), який дозволяє легко встановлювати, оновлювати та видаляти пакети. Ви можете налаштувати середовище для роботи з Python, Node.js, Ruby, Git, SSH, компіляторами та іншими інструментами.

3. Можливість програмування: Забезпечує зручне середовище для написання та запуску коду мовами, такими як Python, C, C++, Go, Java, JavaScript та інші.

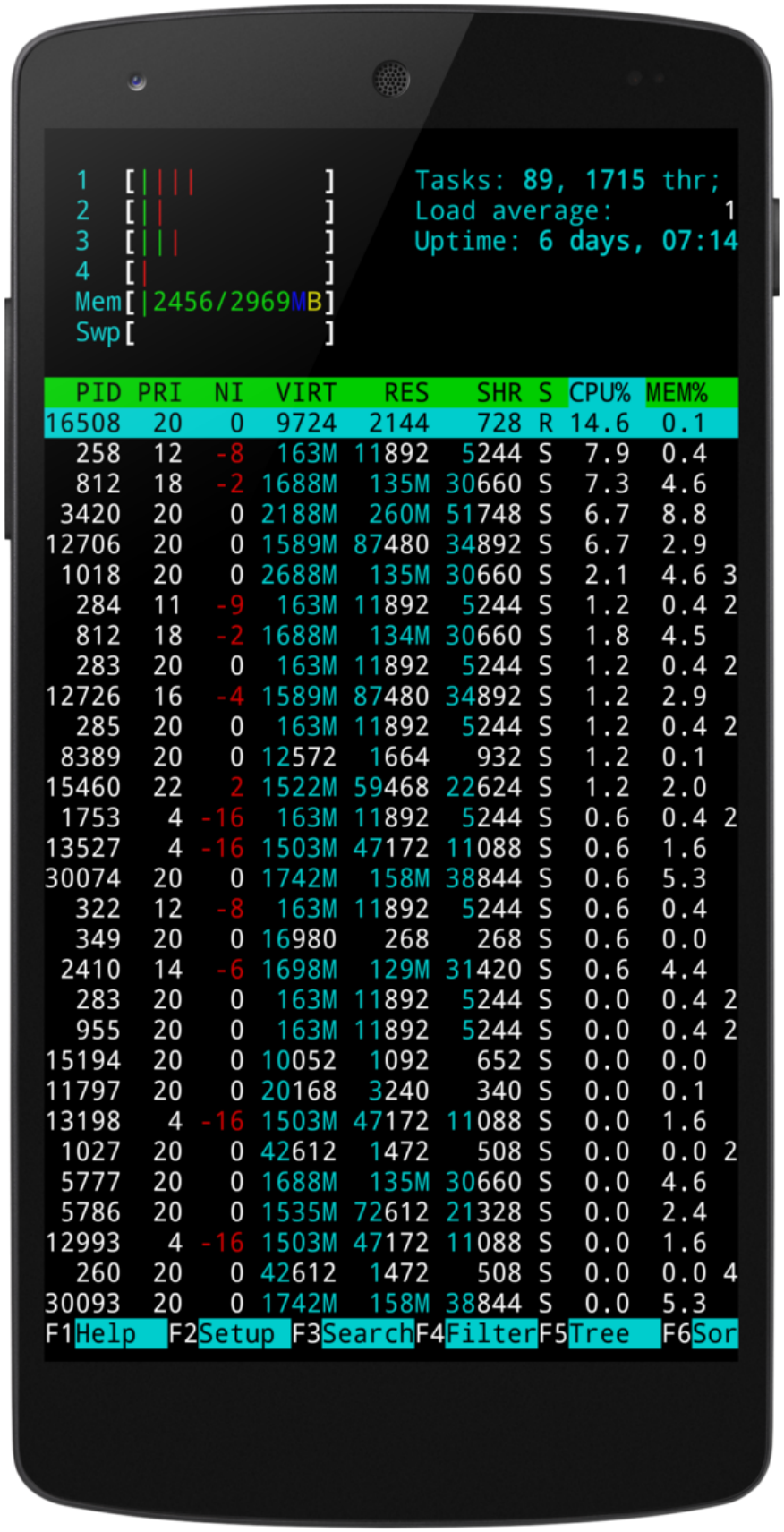

Рисунок 1.26 – Termux

4. Робота з мережею: Підтримує такі інструменти, як OpenSSH, Curl, Wget, які дозволяють керувати серверами та працювати з інтернет-ресурсами.

5. Підтримка редакторів тексту: Можна використовувати популярні текстові редактори, такі як Vim, Nano, або навіть Emacs, безпосередньо на вашому пристрої.

6. Широка розширюваність: Termux підтримує додаткові плагіни, наприклад, для роботи з SSH, доступу до API Android, ведення баз даних SQL тощо.

7. Без необхідності рут-доступу: Для встановлення та використання Termux не потрібні права суперкористувача (root), що робить додаток доступним для всіх користувачів.

Це ідеальне рішення для розробників, ентузіастів Linux або тих, хто тільки знайомиться з принципами роботи цієї операційної системи. Termux можна безкоштовно завантажити з офіційного сайту чи альтернативних джерел, оскільки наразі він недоступний у Google Play через певні політики магазину.

Tilda – це конфігурований термінал для Linux, який працює у вигляді випадаючого меню, створений для забезпечення максимальної зручності та простоти використання. Програму відрізняє мінімалістичний підхід до дизайну та функціоналу, що робить її популярною серед користувачів, які цінують стильний та ненав'язливий інтерфейс. Основна особливість Tilda полягає в тому, що термінал можна викликати за допомогою натискання гарячої клавіші, і він плавно з'являється зверху екрану, як консоль у багатьох відеоіграх. Це дозволяє користувачам швидко запускати команди або перевіряти статус системи без необхідності перемикання між програмами.

Для налаштування Tilda доступний широкий спектр параметрів, включаючи вибір шрифтів, кольорових схем, прозорості та поведінки при виклику. Інструмент також підтримує функцію табів, яка дає змогу працювати з декількома сесіями одночасно. Завдяки своїй легкості у використанні та мінімальній вимогливості до ресурсів системи, Tilda є відмінним вибором для

розробників, системних адміністраторів і досвідчених користувачів Linux, які шукають простий і функціональний термінал для щоденних завдань.

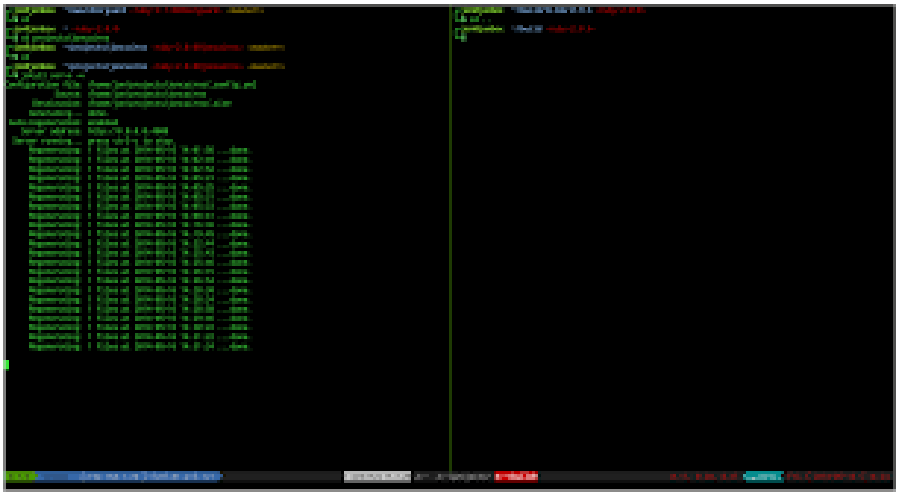

Рисунок 1.27 – Tilda

Tilix – це сучасний емулятор терміналу для Linux, який пропонує розширений механізм укладання вікон з плитками, дозволяючи користувачам організовувати робочу область максимально ефективно.

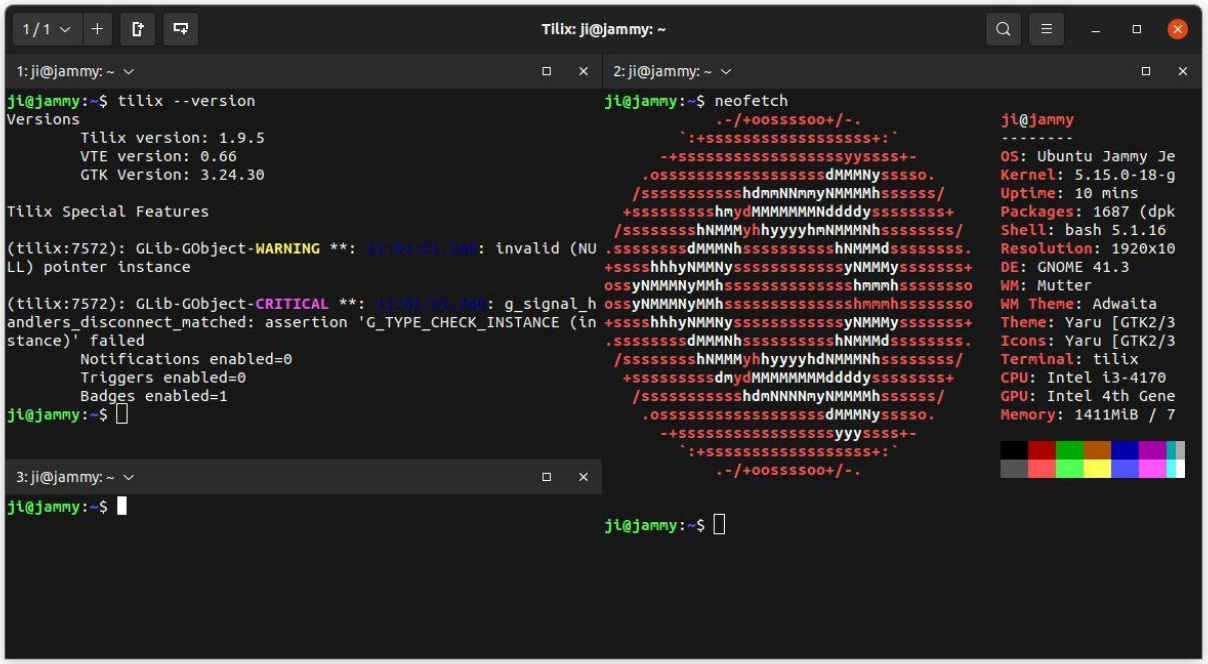

Рисунок 1.28 – Tilix

Завдяки гнучкому розподілу вікон, можна швидко змінювати їх розташування, масштабувати або створювати секції для різних процесів. Tilix

відзначається високим рівнем налаштувань, що дозволяє адаптувати його під індивідуальні потреби: змінювати кольорові схеми, налаштовувати гарячі клавіші, шукати термінали по вкладках і багато іншого. Крім того, програма підтримує збереження й відновлення сеансів, що є особливо корисним для тривалих або критичних робочих процесів. Tilix чудово інтегрується з сучасними середовищами робочого столу, надаючи широкий функціонал для професійного використання.

URxvt (rxvt-unicode) — легкий і потужний емулятор терміналу, розроблений для X11, який відзначається високою продуктивністю та широкими можливостями налаштування. Основною відмінністю URxvt є підтримка Unicode, що забезпечує коректне відображення символів багатьох мов світу, включаючи складні письмові системи, такі як арабська, китайська або японська. Завдяки своєму мінімалістичному дизайну, URxvt ідеально підходить для користувачів, які цінують швидкість роботи та ефективність використання системних ресурсів.

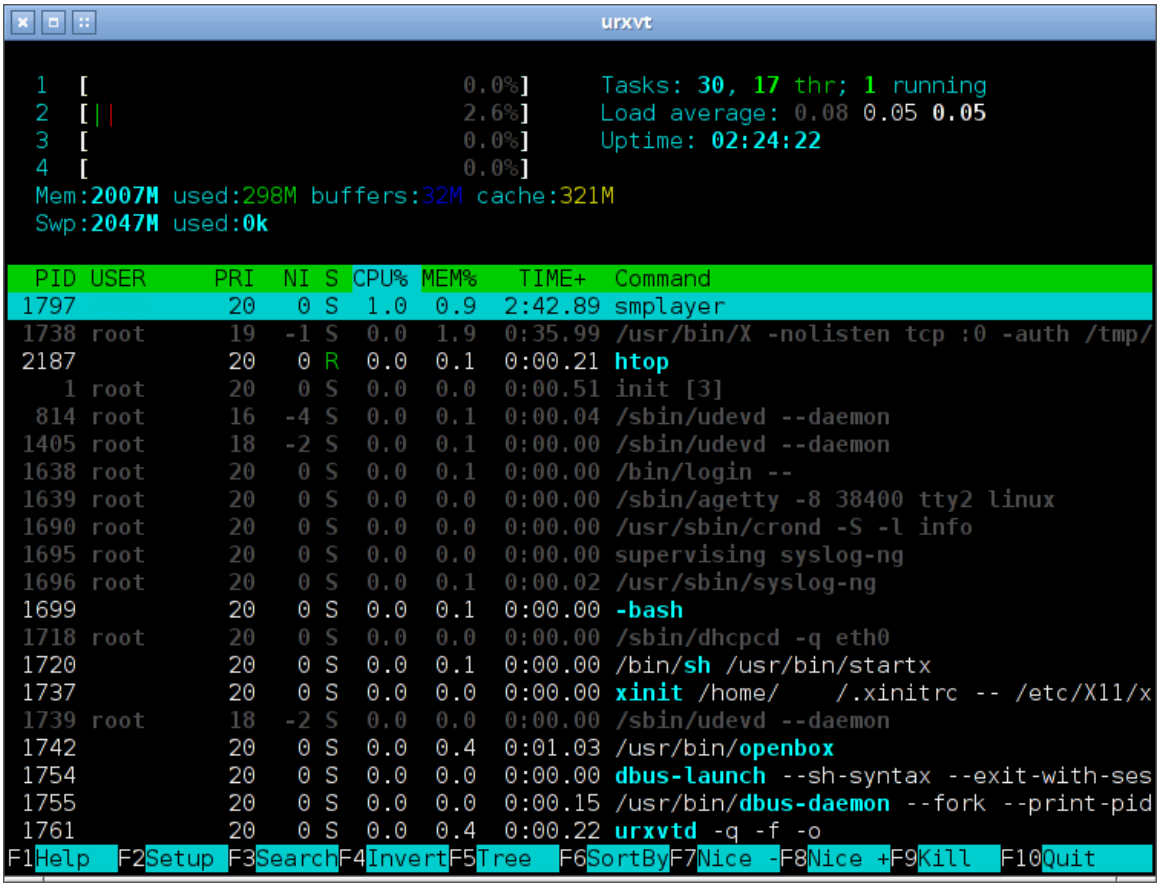

Рисунок 1.28 – URxvt

URxvt підтримує розширення, які дозволяють додатково налаштувати та розширити його функціонал. Наприклад, користувачі можуть додати скрипти для покращення роботи з вкладками, зміни шрифтів або навіть інтеграції із програмами, такими як SSH. Додатковою перевагою є гнучкість у налаштуванні зовнішнього вигляду, що дозволяє змінювати кольори, прозорість та інші аспекти для зручності роботи.

Цей емулятор добре підходить для користувачів операційних систем на базі Linux і Unix, особливо тих, хто працює з програмуванням, адмініструванням серверів або управлінням системою через командний рядок. Його популярність обумовлена простотою, малою витратою оперативної пам'яті і підтримкою багатьох сучасних стандартів. Завдяки своїй стабільності й продуктивності, URxvt залишається одним із найулюбленіших виборів серед ентузіастів Linux.

Warp – це сучасний термінал, створений з використанням мови Rust, яка відома своєю швидкістю, безпекою та ефективністю роботи з системними ресурсами. Окрім стандартних можливостей, які очікуються від термінальних програм, Warp виділяється інтеграцією функцій штучного інтелекту, що суттєво покращують взаємодію користувача та оптимізують процес роботи.

Особливості Warp:

1. Швидкість і продуктивність: Використання мови Rust забезпечує високу швидкодію навіть при роботі з великими даними або складними запитами.

2. Інтуїтивний дизайн: Інтерфейс термінала адаптований до сучасних вимог. Він поєднує текстовий режим із інтерактивними елементами, такими як панелі автопідказок чи історії команд.

3. Інтегровані функції штучного інтелекту:

- Автодоповнення: Термінал пропонує автопідказки на основі контексту, аналізуючи системну інформацію й історію команд.

- Контекстна підтримка: ІІ допомагає розшифровувати помилки, даючи пояснення та рішення прямо в терміналі.

- Пропозиції оптимізації: Штучний інтелект аналізує вашу роботу і пропонує більш ефективні способи виконання задач.

4. Функціональність для командної роботи: Warp підтримує колаборативні функції, такі як обмін командними сесіями або історією. Це особливо корисно для команд розробників чи адміністраторів.

5. Кросплатформенність: Warp доступний для всіх основних операційних систем, таких як macOS, Linux і Windows, що робить його універсальним рішенням для різних середовищ.

6. Висока безпека: Завдяки основі з Rust, термінал має менше вразливостей, пов'язаних із пам'яттю, що додає додатковий рівень безпеки для ваших операцій.

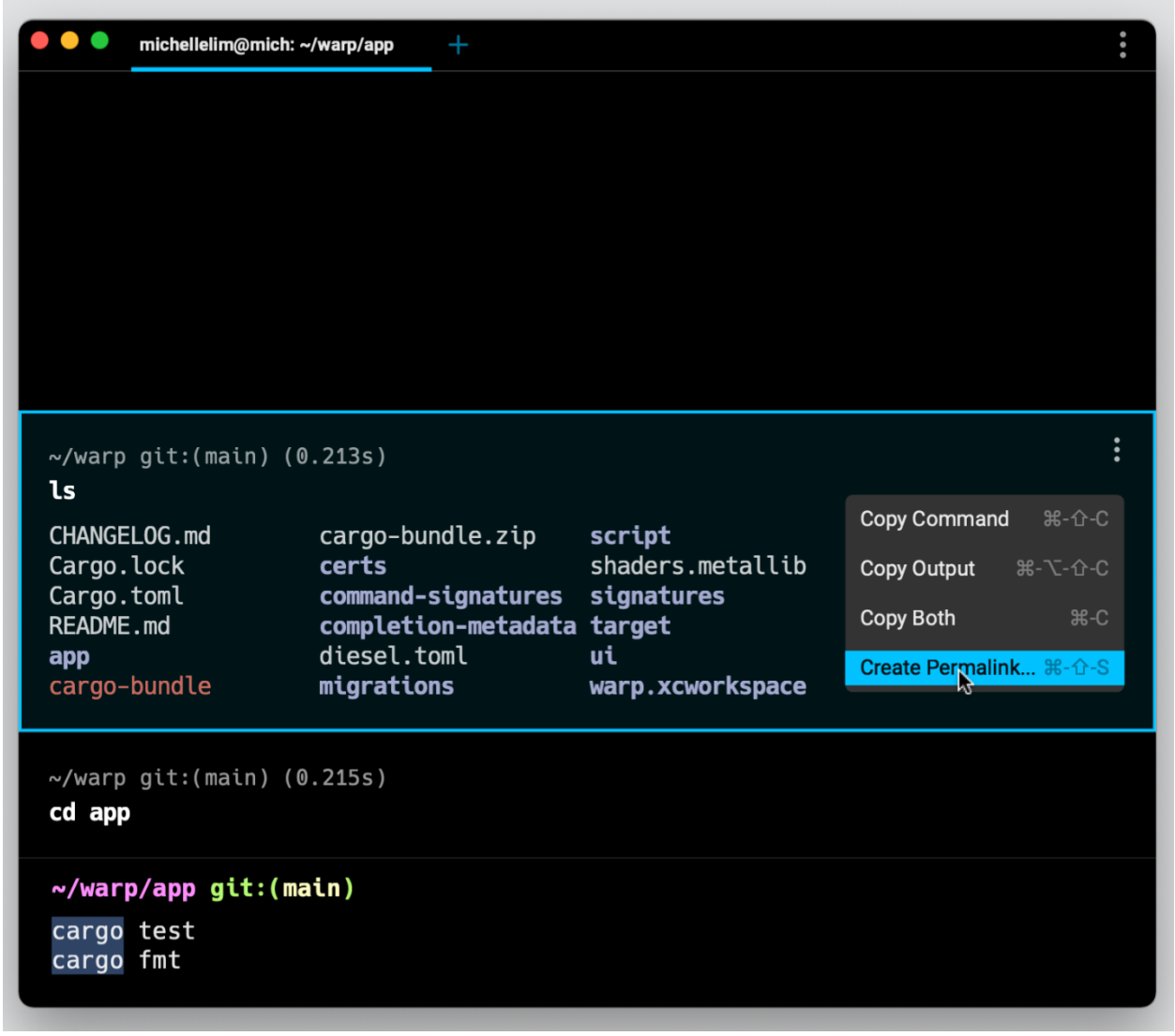

Рисунок 1.29 – Warp

Warp позиціонується як перспективний інструмент для сучасних розробників та системних адміністраторів, пропонуючи не лише стандартні функції терміналу, але й новітні технології, які підвищують продуктивність та комфорт роботи.

Wave: Розроблено для максимальної швидкості роботи та найвищих стандартів сучасного користувацького досвіду (UX). Завдяки інтеграції підтримки апаратного прискорення GPU, інтерфейс забезпечує надзвичайно швидке реагування, а анімації, візуальні елементи та рендеринг виконуються плавно та без затримок, створюючи комфортну і продуктивну взаємодію для користувачів.

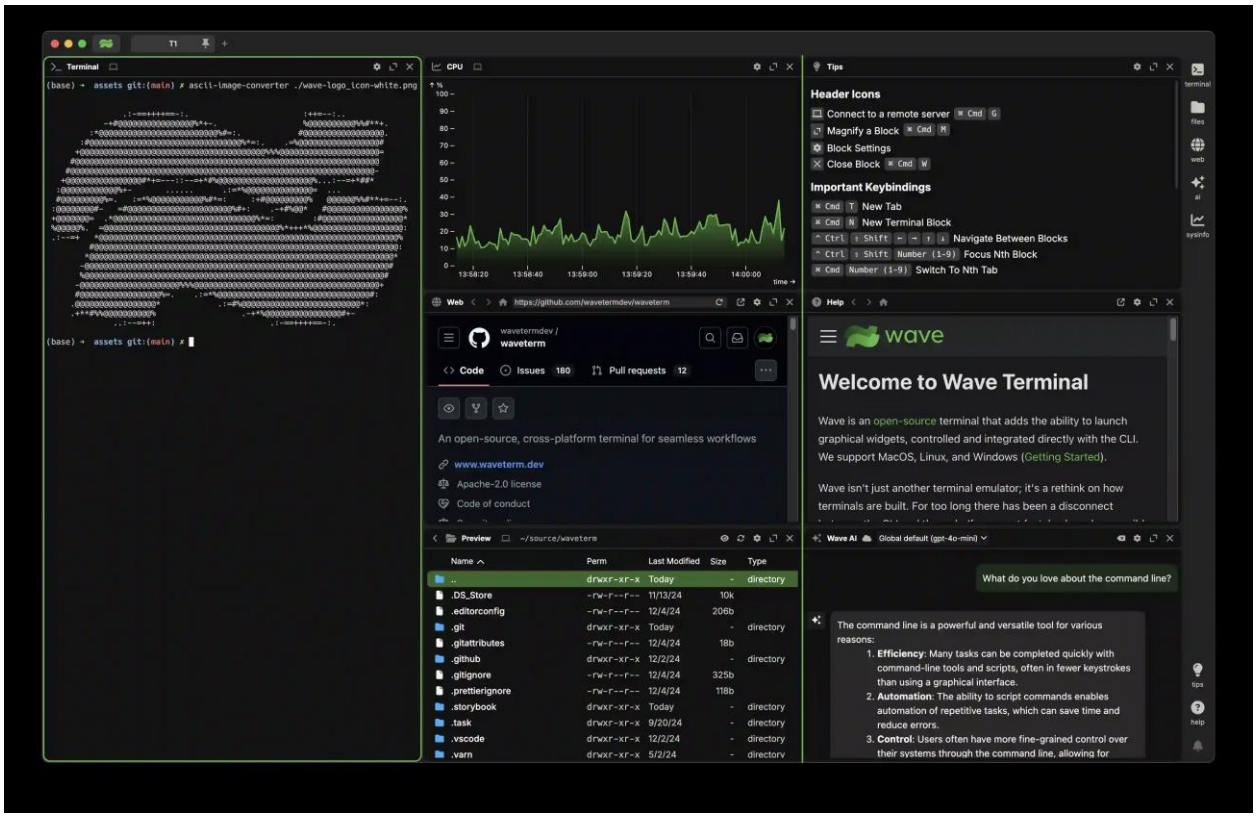

Рисунок 1.30 – Wave

WezTerm — це сучасний, високопродуктивний емулятор терміналу та мультиплексор, що використовує GPU-прискорення для швидшої обробки графічного інтерфейсу. Завдяки кросплатформеній природі, він підтримує роботу на таких операційних системах, як Windows, macOS та Linux, забезпечуючи однаковий досвід використання на всіх платформах. Написаний

на безпечній та продуктивній мові програмування Rust, WezTerm вирізняється високою швидкістю роботи, стабільністю та ефективністю управління системними ресурсами.

WezTerm підтримує сучасні функції, які роблять його зручним інструментом для розробників та системних адміністраторів. Серед них — розширена підтримка Unicode, включаючи роботу з широкими й комбінованими символами, інтегровані можливості для роботи з вкладками, панелями та сплітами. Провідною особливістю є GPU-прискорення, яке мінімізує затримки відображення тексту, дозволяючи розвиватися інтенсивним робочим сесіям без втрати продуктивності.

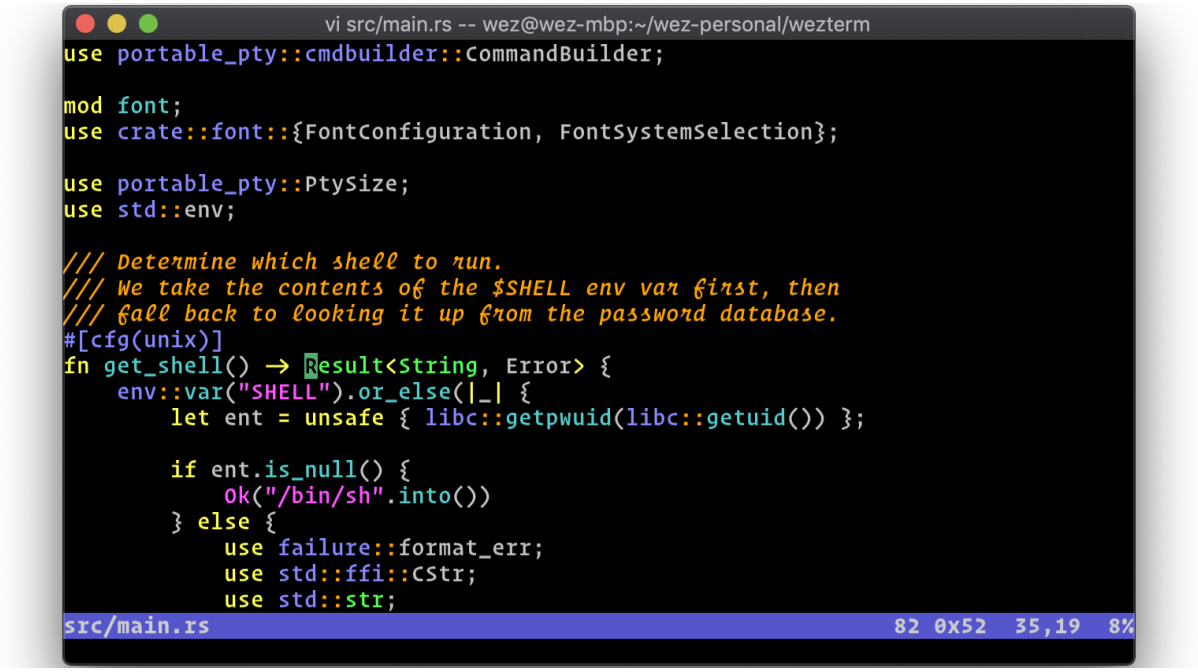

Рисунок 1.31 – WezTerm

WezTerm також включає вбудований Lua-інтерпретатор, який дає змогу користувачам налаштовувати і розширювати функціонал терміналу через створення власних скриптів і конфігурацій. Гнучке налаштування шрифтів, кольорів, гарячих клавіш та багато іншого дозволяє адаптувати термінал під індивідуальні потреби. Завдяки підтримці протоколу SSH, інструмент підходить для віддаленої роботи з серверами.

WezTerm розроблено з урахуванням потреб сучасних користувачів, які цінують швидкість, масштабованість і підтримку новітніх технологій, що роблять роботу з терміналом максимально комфортною та ефективною.

WindTerm — це потужний та багатофункціональний клієнт для SSH, Telnet, SFTP, а також інших мережевих протоколів, створений для забезпечення ефективного управління віддаленими підключеннями. Ця програма є ідеальним вибором для адміністраторів систем, розробників і IT-фахівців, які потребують швидкого доступу до віддалених серверів і передачі даних.

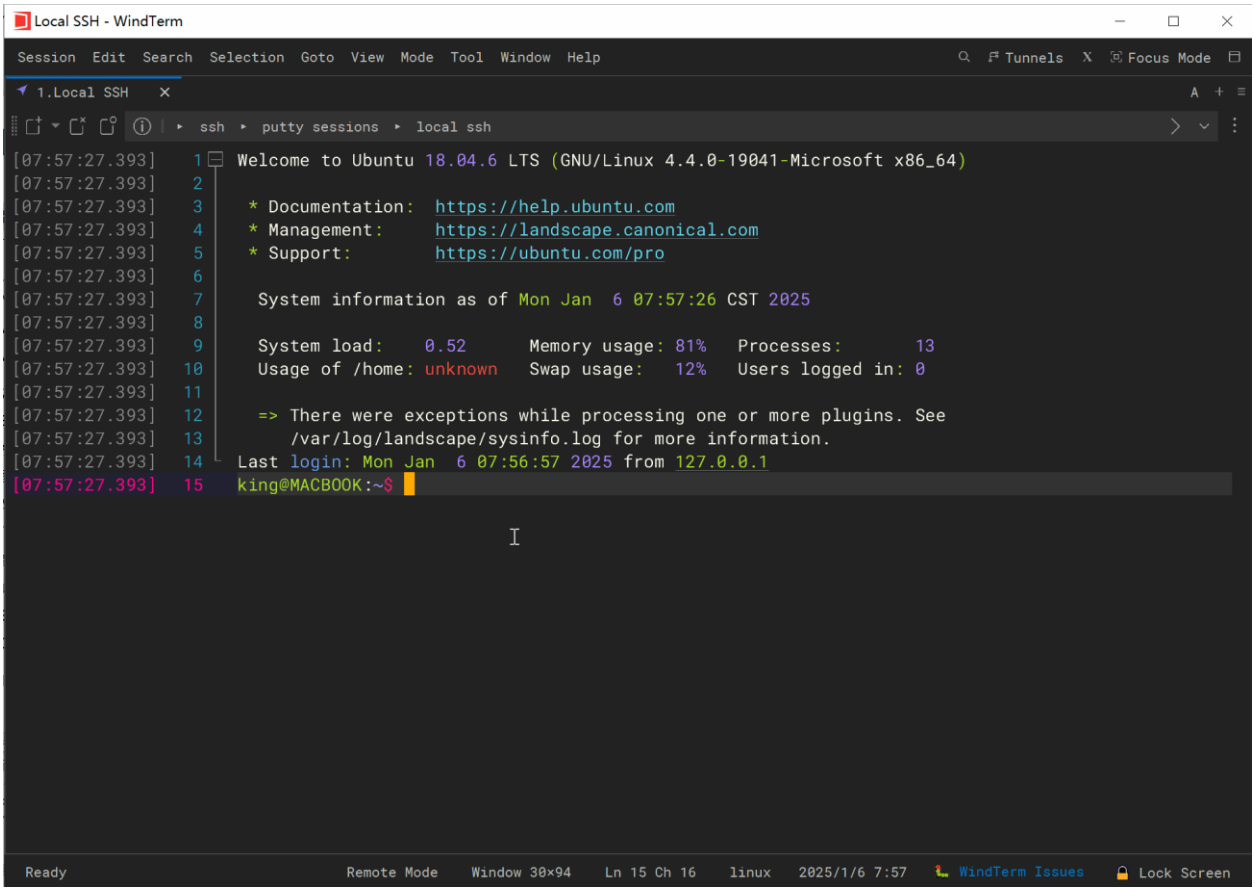

Рисунок 1.32 – WindTerm

Основні функції WindTerm:

1. Управління сеансами: Дозволяє легко організовувати та одночасно керувати великою кількістю активних сеансів. Ви можете створювати групи, зберігати параметри підключень і швидко переключатися між сесіями.

2. Підтримка кількох панелей: Інтерфейс програми забезпечує багатовкладковий підхід, що дозволяє працювати з декількома активними з’єднаннями в одній програмі. Ви можете відкривати окремі сесії в панелях і працювати над різними завданнями паралельно.

3. Автоматизація за допомогою скриптів: Платформа підтримує використання скриптів, що дозволяє автоматизувати рутину, виконувати серії команд чи налаштовувати автоматичне з'єднання з серверами.

4. Оптимізація для великих даних: WindTerm чудово справляється з великими масивами тексту та даних, надаючи коректне рендерування навіть у найскладніших сценаріях.

5. Широка підтримка протоколів: Окрім SSH, Telnet і SFTP, клієнт також підтримує інші стандартні мережеві протоколи, надаючи широкий функціонал для обслуговування різних потреб.

6. Висока продуктивність: Швидкий запуск, низьке споживання ресурсів і стабільна робота навіть під навантаженням.

7. Налаштування інтерфейсу: Програма дозволяє персоналізувати UI, вибираючи теми оформлення, колірні схеми, розташування панелей та інші деталі для зручності користувача.

8. Безпека: WindTerm забезпечує високий рівень захисту даних завдяки підтримці сучасних методів шифрування та автентифікації.

WindTerm є безкоштовним інструментом і є досить популярним серед користувачів, які цінують його простоту, універсальність і швидкість роботи. Він доступний для різних операційних систем, таких як Windows, macOS і Linux, що робить його доступним для широкого кола користувачів.

Xfce4 terminal: Термінал-емулятор за замовчуванням робочого середовища XFCE, що вирізняється легкістю, високою швидкодією та широким функціоналом. Він створений для забезпечення простоти використання, при цьому зберігає багаті можливості для налаштувань, дозволяючи користувачам адаптувати його під свої потреби. Xfce4 Terminal підтримує вкладки, що допомагають організувати робочий процес, дозволяє

змінювати вигляд, шрифти та кольорові схеми, інтегрується з іншими компонентами XFCE. Завдяки використанню бібліотек GTK+, він є компактним і споживає мінімум системних ресурсів, що робить його ідеальним вибором для малопотужних пристроїв або систем з обмеженими можливостями.

Рисунок 1.33 – Xfce4 terminal

XTerm – класичний термінальний емулятор для систем X11, який здобув популярність завдяки своїй ефективності, мінімалізму та широкій підтримці функцій. Він є однією з найстаріших програм у своєму класі і використовувався ще з часу зародження графічних інтерфейсів Unix-систем. XTerm підтримує емуляцію VT-terminal'ів, таких як VT102 і частково VT220, що забезпечує його сумісність із великою кількістю текстових додатків. Незважаючи на свою простоту, XTerm пропонує настроювані шрифти, кольори, клавіші та функцію копіювання/вставки. Завдяки своїй надійності і малому споживанню ресурсів, цей емулятор залишається актуальним для адміністраторів систем, програмістів та користувачів, яким необхідний термінал без зайвих доповнень.

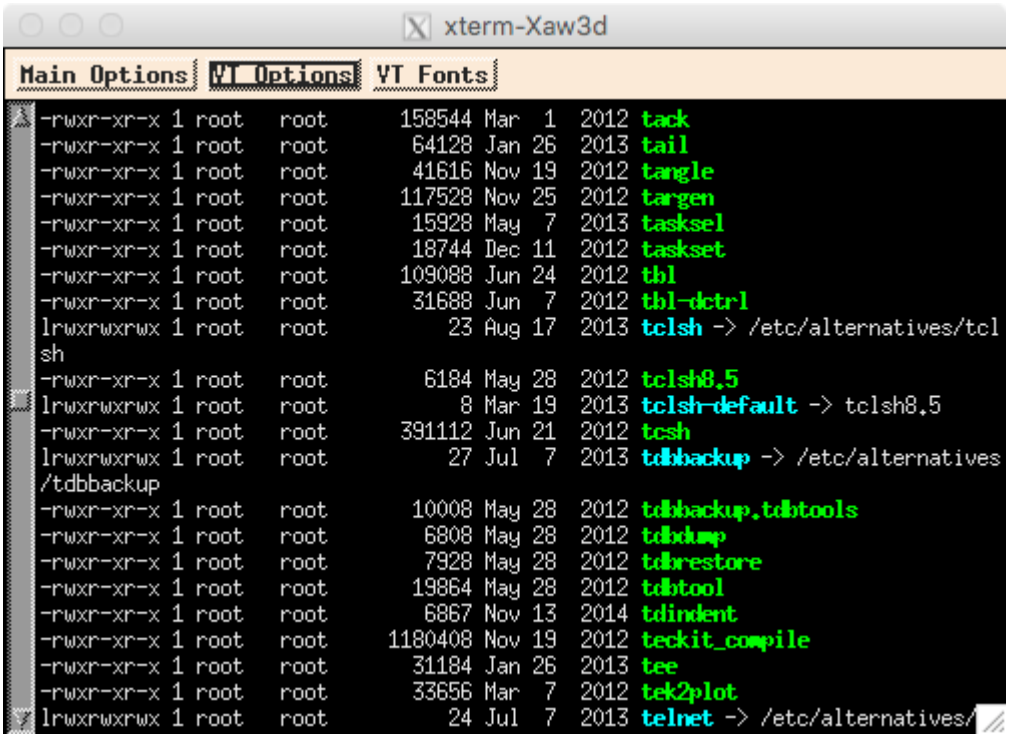

Рисунок 1.34 – XTerm

Yakuake — це потужний емулятор термінала для операційної системи Linux, який натхненний інтерфейсом і стилем спадного консолі з популярної гри Quake. Основним його елементом є зручність і швидкий доступ: термінал "з'являється" зверху екрану в момент натискання спеціальної гарячої клавіші, дозволяючи користувачам миттєво відкривати консольний інтерфейс без необхідності запускати окреме застосування. Завдяки стильному дизайну та можливості налаштування під потреби кожного користувача, Yakuake пропонує широкий спектр функціональності, включаючи роботу з вкладками, налаштування тем оформлення, високу продуктивність і плавність анімацій.

Цей інструмент ідеально підходить як для розробників, так і для системних адміністраторів, які часто працюють з командним рядком. Вбудована підтримка популярних функцій, таких як швидке перемикання між сеансами, настройка гарячих клавіш, і спрощене управління термінальними командами робить Yakuake ще більш зручним у використанні. Завдяки інтеграції з середовищем KDE, застосунок органічно вписується в екосистему

та підтримує широкий спектр можливостей, які забезпечують його високу популярність серед користувачів Linux.

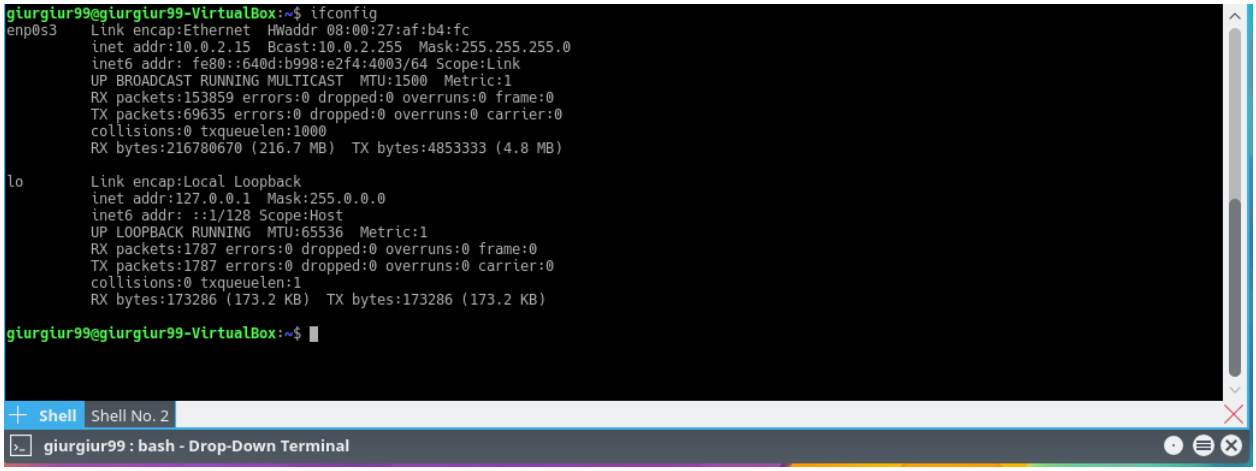

Рисунок 1.35 – Yakuake

1.3 Огляд емуляторів терміналу, призначених для навчання

Вивчення команд UNIX є важливим навиком для роботи в різних IT-сферах. Щоб опанувати цей набір знань, існує безліч онлайн-сервісів, які пропонують інтерактивні уроки, тренування та симуляцію терміналу. Ось перелік найбільш популярних сервісів із детальним описом функціоналу:

1.3.1 OverTheWire

OverTheWire – це платформа, яка пропонує навчальні "військові" ігри для вивчення Linux, команд UNIX та основ безпеки. Завдання пропонуються у формі різних рівнів гри (наприклад, "Bandit" для початківців), де користувачу потрібно використовувати команди UNIX для вирішення реальних проблем. Симуляція терміналу проводиться через SSH-з'єднання, тож користувач працює в середовищі, максимально наближеному до реального.

Особливості:

- Робота через реальний термінал.
- Тематичні квізи з акцентом на базові та просунуті команди UNIX.
- Навчальні вправи з підходом "навчання через гру".

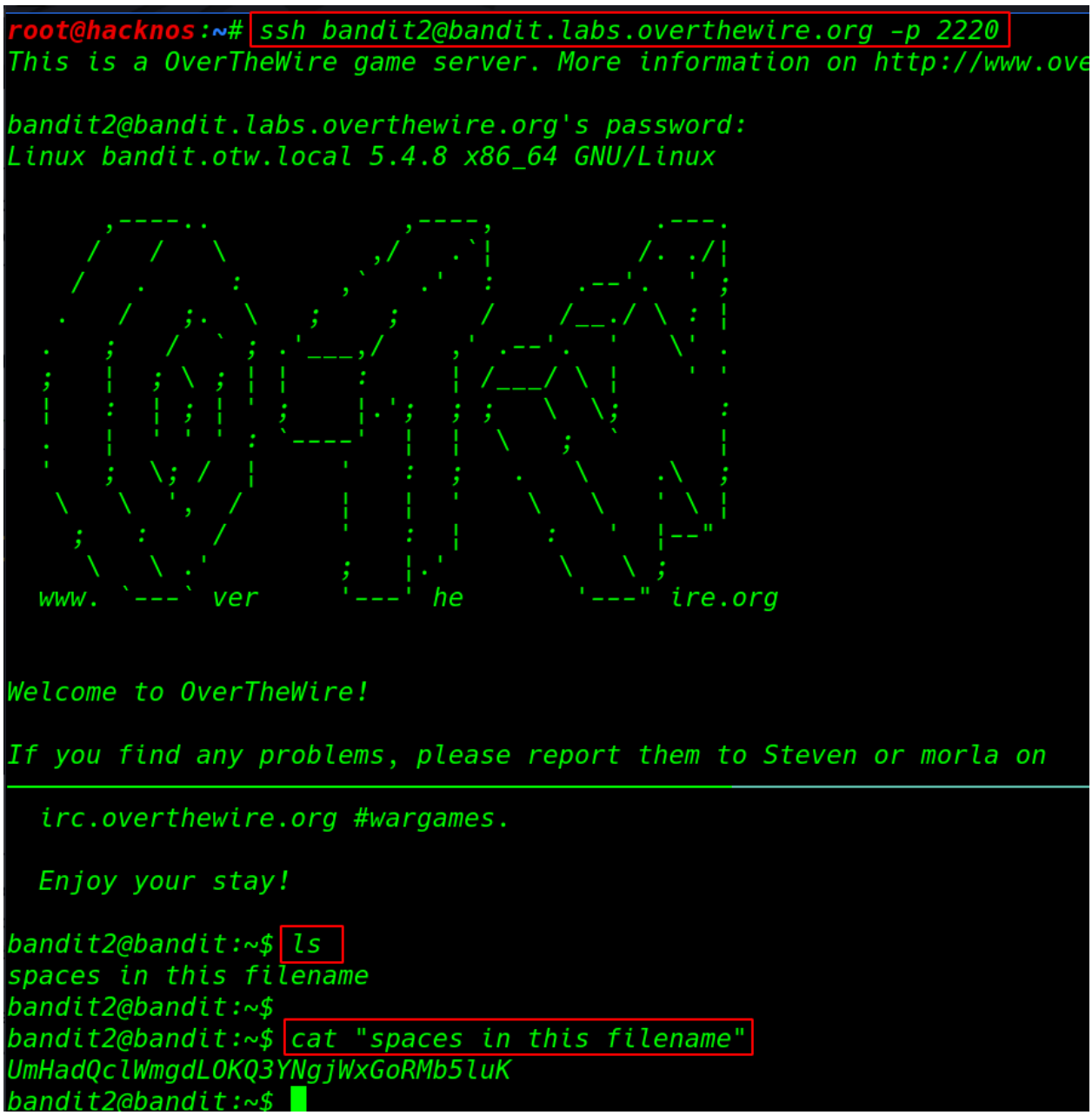

Рисунок 1.36 – OverTheWire

2. Linux Survival

Цей вебсайт пропонує інтерактивні уроки для тих, хто тільки починає знайомитися з Linux. В ньому реалізована емуляція терміналу, яка дозволяє користувачам виконувати команди в безпечному навчальному середовищі. Вони поступово освоюють основи роботи з операційною системою Linux, починаючи з базових команд навігації.

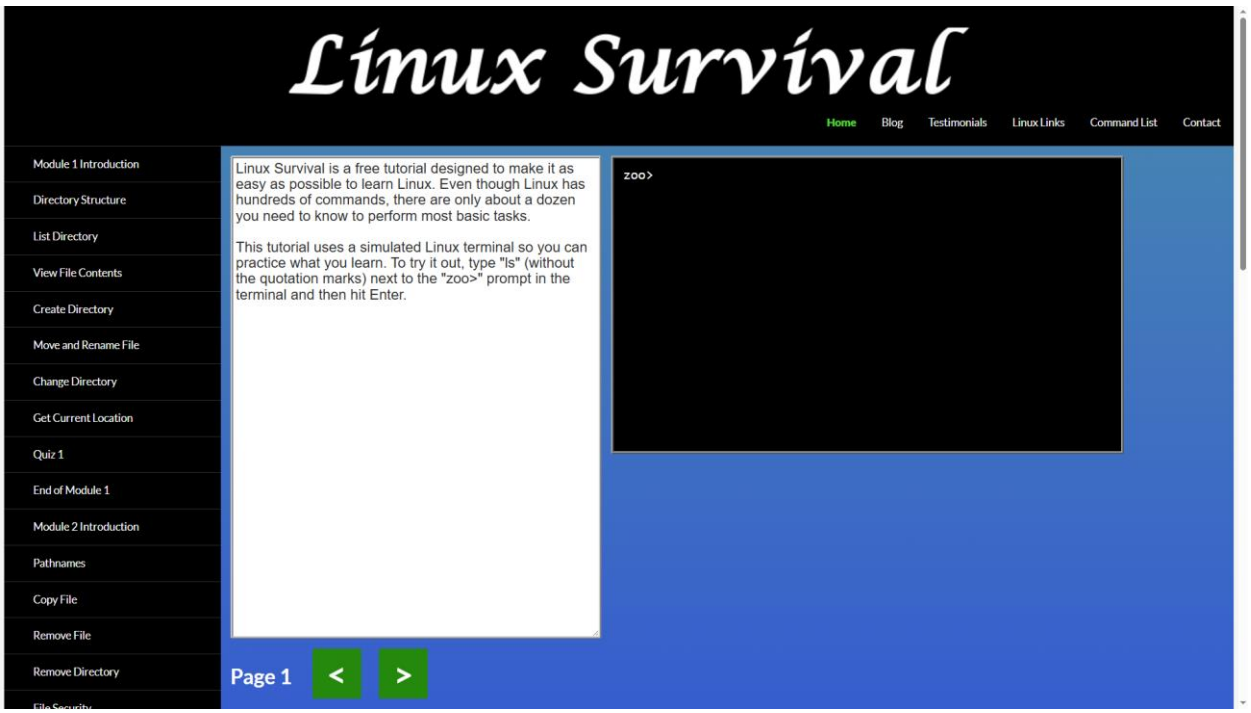

Рисунок 1.37 – Linux Survival

Особливості:

- Вбудована симуляція терміналу.
- Послідовні уроки для поступового освоєння матеріалу.
- Автоматичні підказки для початківців.

3. LearnLinuxOnline

LearnLinuxOnline спеціалізується на навчанні роботи в Linux через браузерні симуляції терміналу. Вебсайт пропонує інтерактивні вправи, де користувачам потрібно виконувати конкретні завдання, як-от створення файлів, редагування тексту, використання команд для навігації та налаштування прав доступу.

Особливості:

- Структуровані навчальні уроки.
- Практика через емуляцію терміналу прямо в браузері.
- Оцінка прогресу та рекомендації щодо подальших завдань.

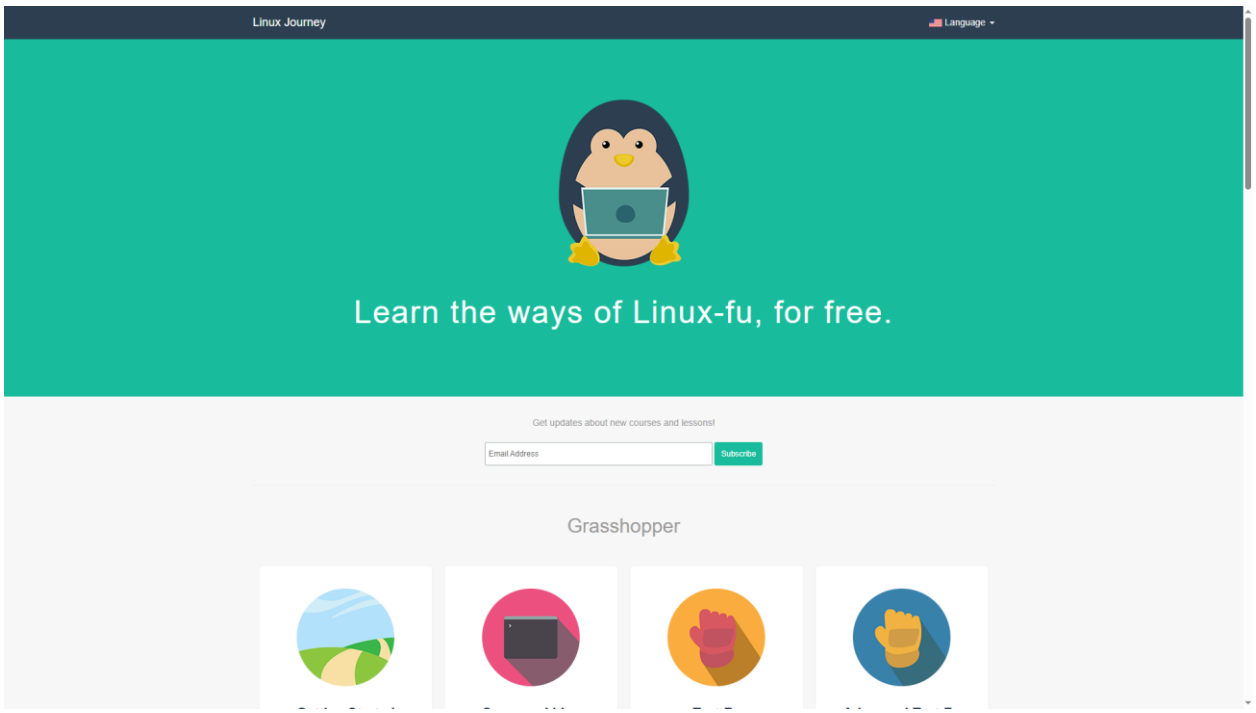

Рисунок 1.38 – LearnLinuxOnline

4. Katacoda

Katacoda – інноваційний сервіс для інтерактивного навчання, який дозволяє вивчати команди UNIX та розуміти роботу системи Linux в інтерактивному середовищі. Тут надається реальне середовище терміналу, а користувачам доступний широкий набір сценаріїв – від базових команд до професійних завдань із адміністрування серверів.

Особливості:

- Взаємодія з реалістичною симуляцією термінала.

- Велика кількість сценаріїв і вправ для різного рівня підготовки.

- Вбудовані інструкції та командні підказки.

5. CmdChallenge

CmdChallenge – це простий і зручний вебсайт для практики команд UNIX. Користувачу пропонується вирішити набір завдань шляхом введення правильних команд у симульованому терміналі. Суть полягає у використанні найефективніших команд для вирішення задач. Це відмінний варіант для закріплення знань і тренування практичних умінь.

Особливості:

- Завдання на оптимізацію команд.

- Симуляція терміналу без необхідності встановлення ПЗ.

- Акцент на ефективності й точності виконання завдань.

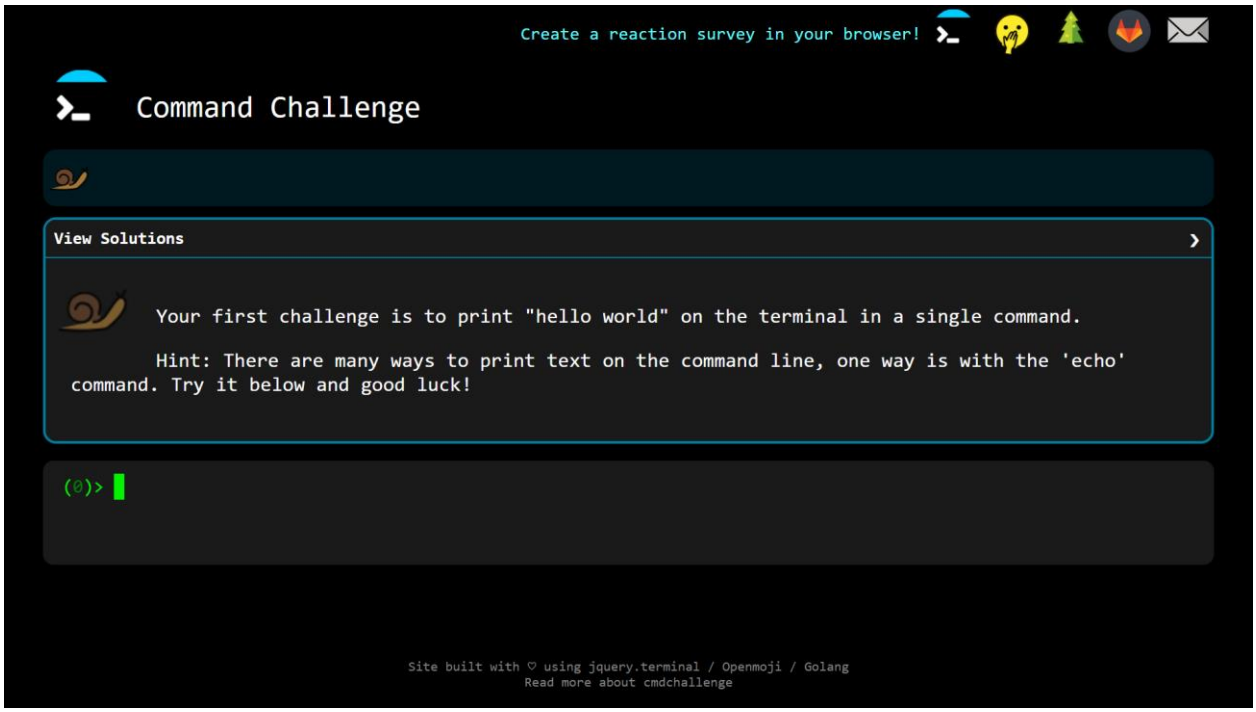

Рисунок 1.39 – CmdChallenge

6. Explainshell

Explainshell більше нагадує освітній інструмент, який допомагає розібратися в синтаксисі команд UNIX. Користувач може вводити будь-яку команду у вікні симуляції, і вебсайт детально пояснює її структуру, включаючи параметри та аргументи. Це чудовий ресурс для тих, хто хоче зрозуміти не лише можливості команд UNIX, а й логіку їх виконання.

Особливості:

- Роз'яснення функціональності кожної команди.

- Інтерактивна демонстрація роботи команд у середовищі.

- Підтримка ведення власного набору завдань.

Рисунок 1.40 – Explainshell

7. TeachMeTheBasics

Цей сервіс орієнтований на абсолютних новачків, які тільки починають освоєння команд UNIX. У ньому реалізовано просте і зручне середовище для емуляції терміналу, а також уроки, що охоплюють основні концепції роботи з файлами, директоріями й правами доступу в Linux.

Особливості:

- Інтуїтивно зрозуміле середовище для навчання.

- Розбиття курсу на невеликі уроки з простими інструкціями.

- Можливість самостійно практикуватись у симульованому терміналі.

8. Webminal

Webminal – це платформа для практичного освоєння Linux та UNIX через симуляцію терміналу прямо в браузері. Сервіс дозволяє інтерактивно вивчати команди, працювати зі скриптами, створювати файли та виконувати інші завдання, пов’язані з адмініструванням Linux.

Особливості:

- Браузерна симуляція реального терміналу.

- Практичні вправи з розв’язання кейсів.

- Легкий доступ для новачків та можливості для професійного зростання.

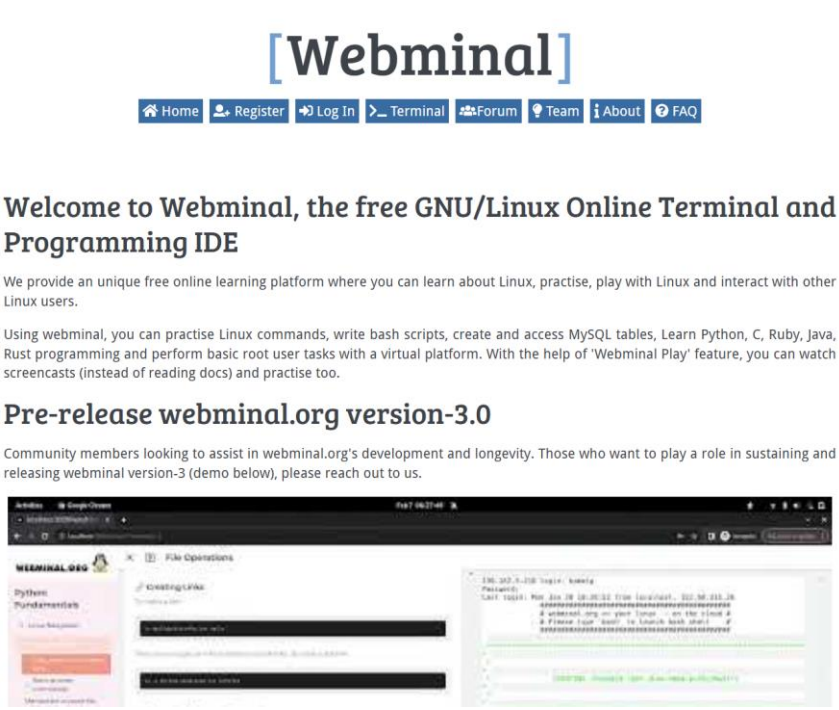

Рисунок 1.41 – Webminal

Вивчення команд UNIX через онлайн-симуляції – це практичний і доступний спосіб отримати необхідні навички без потреби встановлення операційної системи Linux. Використання цих платформ допоможе зрозуміти принципи роботи з терміналом, розібратися в базових командах і відточити свої уміння в безпечному навчальному середовищі.

1.4 Проєктування вимог до програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX

Тож, як ми бачимо, сервісів, які надають можливість вивчати команди UNIX, доволі багато. Проте вони не мають засобів посилення мотивації для навчання. Таке посилення можна було б здійснити, внесши в процес навчання елементи гри і спроєктувати програмне забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем.

Сформулюємо основні вимоги до програмного забезпечення ігрового тренажера для вивчення терміналу UNIX-подібних систем, розділивши їх на функціональні та нефункціональні.

Функціональні вимоги:

Інтерактивне навчання – це сучасний підхід до освіти, який дозволяє користувачам безпосередньо взаємодіяти з навчальним матеріалом у реальному часі. Одним із прикладів такого підходу є використання інтерактивних емуляторів терміналу, де користувачі мають можливість не лише спостерігати за процесами, а й брати активну участь у виконанні завдань, що допомагає глибше зрозуміти сутність конкретних інструментів, команд або програмного забезпечення. Це дозволяє "вчитися через дію", набуваючи практичного досвіду разом із теоретичною підготовкою.

Основні переваги інтерактивного навчання з використанням емуляторів терміналу:

1. Практичність: Користувачі не тільки читають або слухають матеріал, а й одразу застосовують знання на практиці, повторюючи команди у віртуальному середовищі.

2. Безпека: Емулятори створюють ізольоване середовище, де можна протестувати команди чи скрипти без ризику пошкодити реальну систему.

3. Візуальний фідбек: Учасники навчального процесу отримують миттєвий зворотний зв’язок у вигляді результатів виконаних команд або повідомлень про помилки, що стимулює їх швидше навчатися на власних спробах.

4. Індивідуалізація: Користувачі за власним темпом можуть виконувати завдання, повторювати операції або переключатися між різними складнощами вправ.

5. Адаптованість до реальних умов: Після навчання в емуляторі терміналу студенти можуть легко адаптувати свої знання до роботи в реальних системах, оскільки команди, навички та технічні підходи залишаються актуальними.

Таким чином, інтерактивне навчання з використанням емуляторів терміналу – це потужний інструмент для створення практично орієнтованих тренінгів, курсів або навчальних матеріалів. Він допомагає навчити як

новачків, так і просунутих користувачів працювати з системами, здійснювати завдання і знаходити рішення в середовищах, що максимально наближені до реальних умов.

Гейміфікація – це процес інтеграції ігрових механізмів у неігровий контекст для мотивації та залучення користувачів до виконання певних завдань або досягнення цілей. Система гейміфікації повинна бути добре продумана, включаючи такі ключові елементи:

1. Рівні – структура прогресу, яка дозволяє користувачам крок за кроком підійматися, удосконалюючи свої навички та знання. Рівні стимулюють активну участь і створюють відчуття досягнень. Наприклад, кожен рівень може відкривати нові можливості або бонуси для користувачів.

2. Досягнення – специфічні цілі, які користувачі повинні виконати для отримання відзнаки або нових можливостей. Досягнення можуть бути пов’язані з виконанням завдання, накопиченням балів, вивченням нового контенту або участю в додаткових активностях, формуючи мотивацію до вдосконалення.

3. Нагороди – система заохочень, яка може включати віртуальні значки, рейтингові очки, бонуси, привілеї або навіть матеріальні подарунки. Нагороди стимулюють користувачів активно брати участь у процесі й зміцнюють мотивацію. Зазвичай вони обумовлені конкретними діями, що сприяють досягненню мети.

4. Змагальні елементи – дух суперництва через інтеграцію рейтингових таблиць, лідербоардів, турнірів чи командних змагань. Це дозволяє користувачам порівнювати свої результати з іншими, викликаючи у них прагнення стати кращими, поліпшити свої показники та досягти нових вершин.

Гейміфікація допомагає створити захоплюючий досвід для користувачів у будь-якій сфері, будь то освіта, бізнес або соціальні проєкти. Продовжуючи долати рівні, відкриваючи нові досягнення та отримуючи нагороди, людина

відчуває задоволення, мотивацію і бажання взаємодіяти з системою на довгостроковій основі.

Режими навчання створені для забезпечення ефективного освоєння роботи з терміналом UNIX, пропонують два основні варіанти, які відповідають різним етапам навчання та рівням підготовки користувачів:

1. Навчальний режим із підказками. У цьому режимі користувач отримує покрокові інструкції та наочні підказки під час виконання команд у терміналі UNIX. Це підходить для початківців та тих, хто бажає закріпити теоретичні знання на практиці. Функціонал режиму включає:

– Пропонування списку базових команд із поясненнями їхньої ролі та параметрів.

– Інтерактивні вправи, що дозволяють вводити команди з автоматичною перевіркою правильності.

– Підказки в реальному часі, які допомагають уникнути типових помилок і вказують на можливі варіанти застосування тієї чи іншої функції.

– Інформаційні блоки з поясненням концепцій, таких як робота з файловою системою, процесами, правами доступу тощо.

2. Тестовий режим для перевірки знань. Цей режим служить для самоконтролю і перевірки засвоєного матеріалу. Він орієнтований на виконання завдань без підказок, щоб користувач міг оцінити свій рівень підготовки. Основні характеристики тестового режиму:

– Завдання різного рівня складності: від виконання простих операцій (створення файлу, перегляд вмісту каталогу) до складніших сценаріїв (написання bash-скриптів, управління процесами).

– Автоматична система оцінювання, яка аналізує правильність виконаних дій та надає розгорнуті коментарі щодо можливих покращень.

– Часові обмеження на виконання окремих завдань для імітації реальних умов роботи.

– Статистика успішності, що дозволяє відслідковувати прогрес і визначати слабкі місця для подальшого опрацювання.

Обидва режими навчання розроблені таким чином, щоб максимально допомогти користувачу освоїти ключові аспекти роботи з терміналом UNIX, незалежно від початкового рівня знань. Вони можуть використовуватись як самостійно, так і у складі інтерактивних курсів чи навчальних програм.

Навчальна гра з використання терміналу UNIX має підтримувати різні команди – ls, cd, mkdir, rm, grep, chmod тощо.

Сценарії та завдання для користувачів мають бути створені таким чином, щоб максимально імітувати реальні ситуації, з якими вони можуть зіткнутися при роботі з UNIX-подібними системами. Нижче подано приклади практичних завдань із коротким описом, адаптовані під різні рівні підготовки користувачів:

1. Базовий рівень (початковий користувач):

Створення та навігація файловою системою.

– Створіть новий каталог із назвою `project`.

– У цьому каталозі створіть текстовий файл із назвою `notes.txt`.

– Використовуючи текстовий редактор (nano, vim або будь-який інший), додайте до файлу `notes.txt` рядки тексту: "Моє перше завдання".

– Створіть копію файлу `notes.txt` із назвою `backup_notes.txt`.

– Перемістіть обидва файли в інший каталог, наприклад, `archive`.

Ви повинні побачити файли `notes.txt` та `backup_notes.txt` у папці `archive`.

2. Середній рівень (досвідчений користувач):

Робота з правами доступу та пошук файлів у системі.

– Створіть файл із назвою `config.cfg` в каталозі `/home/user/settings`.

– Встановіть для цього файлу права доступу: лише власник може читати та редагувати цей файл.

– Використовуючи команду `find`, знайдіть усі файли розширення `.log` у вашій домашній директорії та скопіюйте їх в нову папку `/home/user/logs_archive`.

– Напишіть у файл `log_report.txt` повний список цих файлів і дату їх останнього оновлення.

Відповідно заданих умов, користувач навчиться налаштовувати доступ до файлів, знаходити їх в системі та формувати звіти про знайдені об'єкти.

3. Просунутий рівень (системний адміністратор):

Автоматизація та робота зі скриптами.

– Створіть Bash-скрипт із назвою `backup.sh`, який виконує наступні дії:

1. Знаходить усі файли старіші ніж 7 днів у заданій директорії (`/var/log`).

2. Створює архів цих файлів із назвою `old_logs.tar.gz`.

3. Переміщує архів у папку `/backup`.

– Налаштуйте запуск цього скрипту щоденно о 2:00 за допомогою планувальника `cron`.

– Додайте логіку до скрипту, щоб він перевіряв вільний простір на диску, і якщо його менше ніж 10 ГБ, повідомляв користувачеві про це через повідомлення в системі.

Користувач отримає комплексні знання з автоматизації завдань, роботи з архівами та налаштування регулярних завдань для підтримки системи.

4. Спеціальні сценарії (розробник чи DevOps інженер):

Управління процесами та мережею.

– Дослідіть список активних процесів у системі за допомогою команди `ps` або `top`. Знайдіть процес із максимальною витратою оперативної пам'яті.

– Використовуючи команду `netstat` або `ss`, знайдіть усі відкриті порти та активні з'єднання.

– Напишіть скрипт, який перевіряє доступність певного хоста (наприклад, google.com) через ping, і записує результат у файл `network_status.log`.

– Налаштуйте правила файрволу (iptables або ufw), щоб обмежити доступ до певного порту (наприклад, 22).

Користувач тренується керувати процесами, діагностувати проблеми в мережі та налаштовувати базовий рівень безпеки.

Додаткові сценарії:

– Резервне копіювання баз даних MySQL із використанням скриптів.

– Моніторинг ресурсів системи (CPU, RAM, дискового простору) за допомогою команди `htop` або `df`.

– Аналіз системних журналів у папці `/var/log` для вирішення помилок та пошуку невідповідностей.

Ці завдання забезпечують комплексне розуміння роботи з UNIX-подібними системами й допомагають користувачам здобувати практичні навички, які необхідні для виконання реальних завдань у сфері IT.

Збереження прогресу – це функціонал, який дозволяє користувачу продовжити навчання, роботу або будь-яку іншу діяльність у цифровому середовищі саме з того місця, де він зупинився в попередній раз. Ця опція особливо корисна у навчальних платформах, онлайн-курсах, іграх, додатках для читання книжок, перегляду контенту або виконання завдань. Завдяки ній забезпечується безперервність процесу і економія часу, оскільки не потрібно щоразу шукати точку, з якої слід відновити роботу.

Інтеграція з навчальними матеріалами в UNIX-подібних системах передбачає доступ до широкого спектра документації та інструкцій, які допомагають користувачам вивчати основи роботи з командним рядком, освоювати адміністративні функції та поглиблювати своє розуміння операційної системи.

Доступ до навчальних матеріалів у UNIX-подібних системах є різноманітним і охоплює потреби як початківців, так і досвідчених користувачів. Завдяки комбінованій інтеграції локального і онлайн-контенту, навчання стає доступним і зручним, сприяючи підвищенню рівня знань та майстерності у роботі з системою.

1.4.2 Нефункціональні вимоги

Продуктивність – швидке виконання команд терміналу UNIX-подібних систем без помітних затримок.

Безпека – захист від некоректного введення UNIX-команд та потенційно небезпечних операцій.

Інтуїтивний інтерфейс – зручний дизайн для новачків та досвідчених користувачів.

Локалізація – підтримка кількох мов, включаючи українську та англійську.

Після визначення вимог ми можемо перейти до проєктування та реалізації програмного забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІГРОВОГО ТРЕНАЖЕРУ З ВИВЧЕННЯ ОСНОВ ВИКОРИСТАННЯ ТЕРМІНАЛУ UNIX-ПОДІБНИХ ОПЕРАЦІЙНИХ СИСТЕМ

2.1 Проєктування потоків даних

Діаграми потоків даних (DFD) допомагають візуалізувати, як інформація рухається через систему. Для програмного забезпечення ігрового тренажеру з вивчення UNIX-подібного терміналу можна розробити кілька рівнів DFD:

Контекстна діаграма (DFD рівня 0) – показує загальну взаємодію системи з користувачем та зовнішніми даними.

DFD рівня 1 – деталізує основні процеси всередині тренажера, такі як введення команд, перевірка відповідей, надання підказок.

DFD рівня 2 – розбиває окремі процеси на менші функціональні блоки, наприклад роботу з інтерпретатором команд, логіку навчального модуля, систему підрахунку балів.

Контекстна діаграма (DFD рівня 0) показує загальну взаємодію між системою тренажеру та зовнішніми елементами. Вона включає основні потоки даних, що надходять у систему та виходять із неї.

Основні компоненти контекстної DFD:

Користувач – вводить команди в термінал.

Ігровий тренажер – обробляє введені команди, надає зворотний зв’язок, перевіряє правильність виконання завдань.

База даних навчальних завдань – містить сценарії навчання, перевіряє виконання завдань.

Система підрахунку балів – фіксує успішність користувача.

Інтерпретатор команд – аналізує введені команди, визначає правильність.

Навчальний модуль – надає підказки та пояснення щодо помилок.

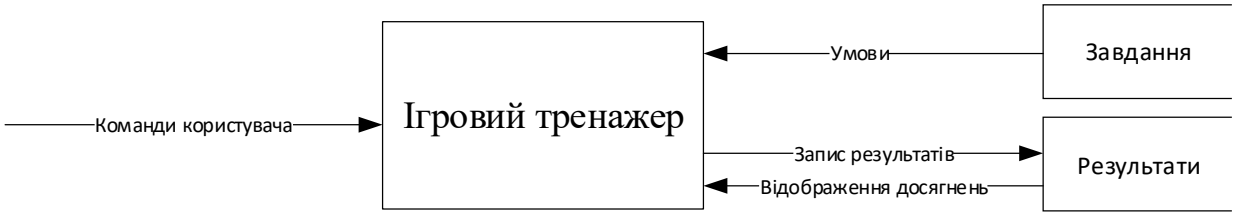

Рисунок 2.1 – Контекстна діаграма ігрового тренажеру

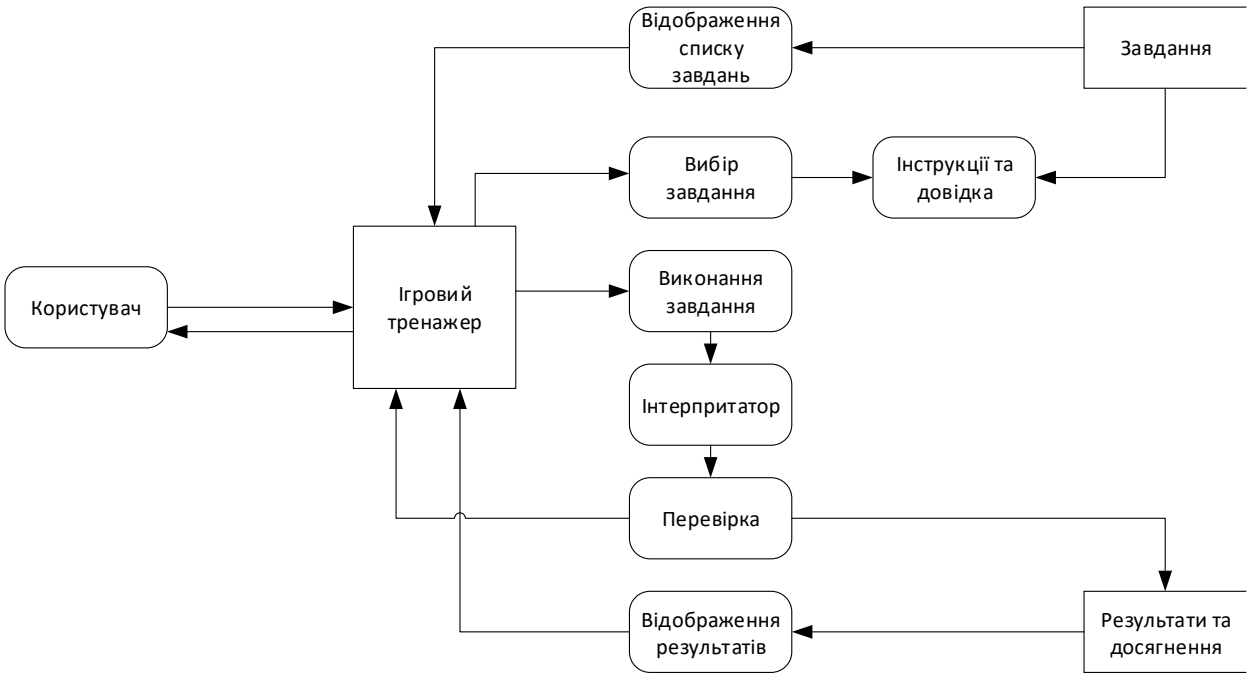

Рисунок 2.2 – Діаграма потоків даних

2.2 Проєктування діаграми використання ігрового тренажеру

Діаграма використання (Use Case Diagram) у UML моделює взаємодію між користувачем та функціональними можливостями системи. Для тренажеру терміналу UNIX-подібних ОС ключовими елементами будуть:

Основні актори:

– Користувач – взаємодіє з тренажером, вводячи команди.

– Ігровий тренажер – перевіряє виконання завдань, надає підказки.

– Система підрахунку балів – оцінює прогрес користувача.

– База даних навчальних завдань – містить сценарії навчання.

– Інтерпретатор команд – аналізує введені команди.

Основні варіанти використання (Use Cases):

– Введення команди (Користувач → Ігровий тренажер)

– Перевірка введеної команди (Ігровий тренажер → Інтерпретатор команд)

– Отримання зворотного зв’язку (Ігровий тренажер → Користувач)

– Нарахування балів (Ігровий тренажер → Система підрахунку балів)

– Отримання навчальних матеріалів (Ігровий тренажер → База даних навчальних завдань)

– Надання підказок (Ігровий тренажер → Користувач)

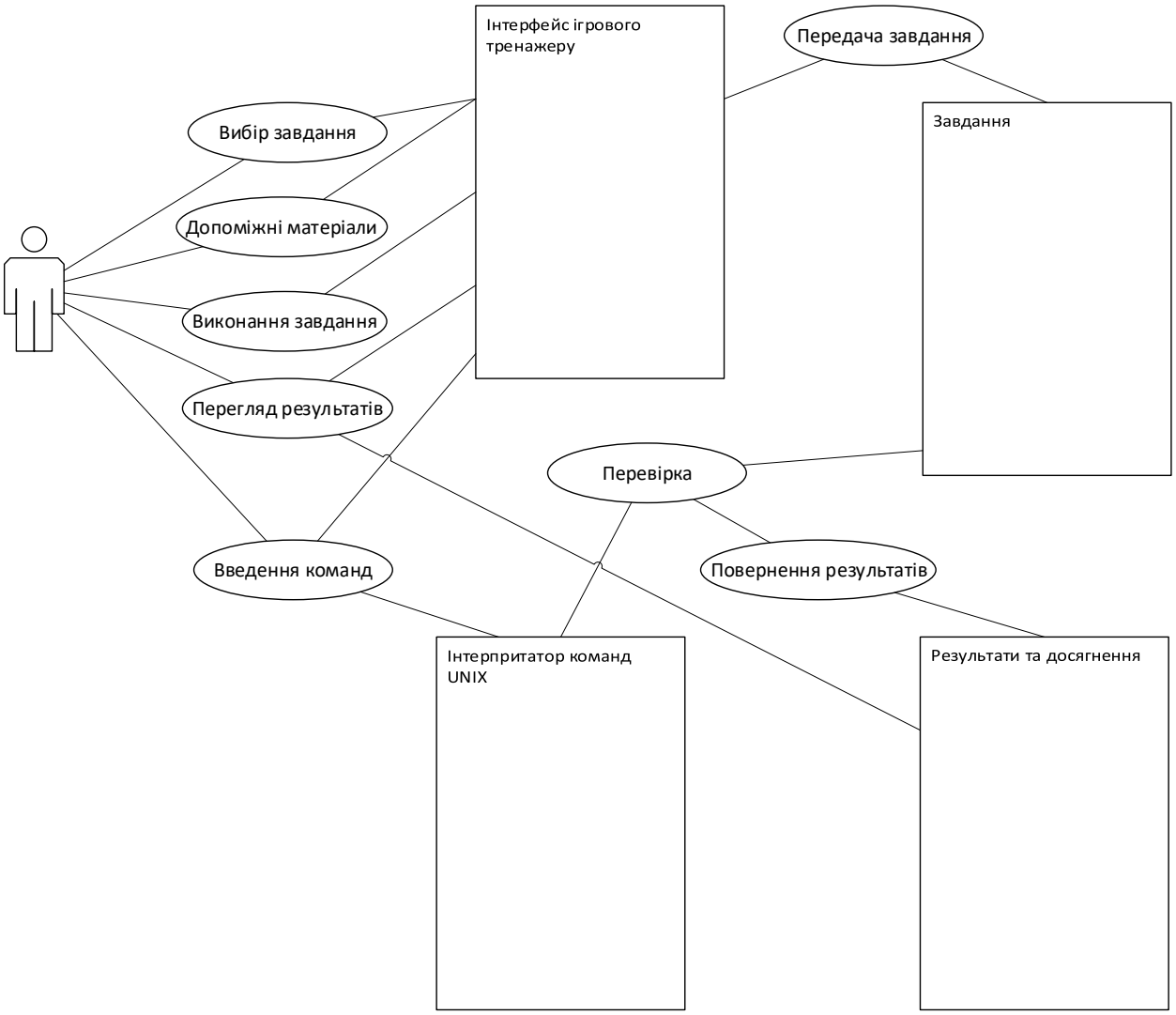

Рисунок 2.3 – Діаграма використання ігрового тренажеру

Також нам необхідно спроєктувати бази даних, що буде використовувати наш ігровий інтерпритатор.

2.3 Проєктування баз даних ігрового тренажеру

Для тренажеру терміналу UNIX-подібних ОС база даних має зберігати інформацію про користувачів, навчальні завдання, команди, результати та підказки. Ось приклад структури таблиць:

1. Таблиця Users (Користувачі)

Зберігає дані про зареєстрованих користувачів.

```
CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    registration_date TIMESTAMP DEFAULT
CURRENT_TIMESTAMP
);
```

2. Таблиця Tasks (Навчальні завдання)

Містить завдання, які потрібно виконати у тренажері.

```
CREATE TABLE Tasks (
    task_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    correct_command VARCHAR(255) NOT NULL
);
```

3. Таблиця Commands (Команди користувача)

Фіксує введені команди та їх статус.

```
CREATE TABLE Commands (
    command_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
```

```
    task_id INT,
    command_text VARCHAR(255) NOT NULL,
    execution_result TEXT,
    is_correct BOOLEAN DEFAULT FALSE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (task_id) REFERENCES Tasks(task_id)
);
```

4. Таблиця Hints (Підказки)

Містить підказки для користувачів щодо виконання завдань.

```
CREATE TABLE Hints (
    hint_id INT PRIMARY KEY AUTO_INCREMENT,
    task_id INT,
    hint_text TEXT NOT NULL,
    FOREIGN KEY (task_id) REFERENCES Tasks(task_id)
);
```

5. Таблиця Scores (Результати користувачів)

Відображає прогрес кожного гравця.

```
CREATE TABLE Scores (
    score_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    total_points INT DEFAULT 0,
    completed_tasks INT DEFAULT 0,
    last_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

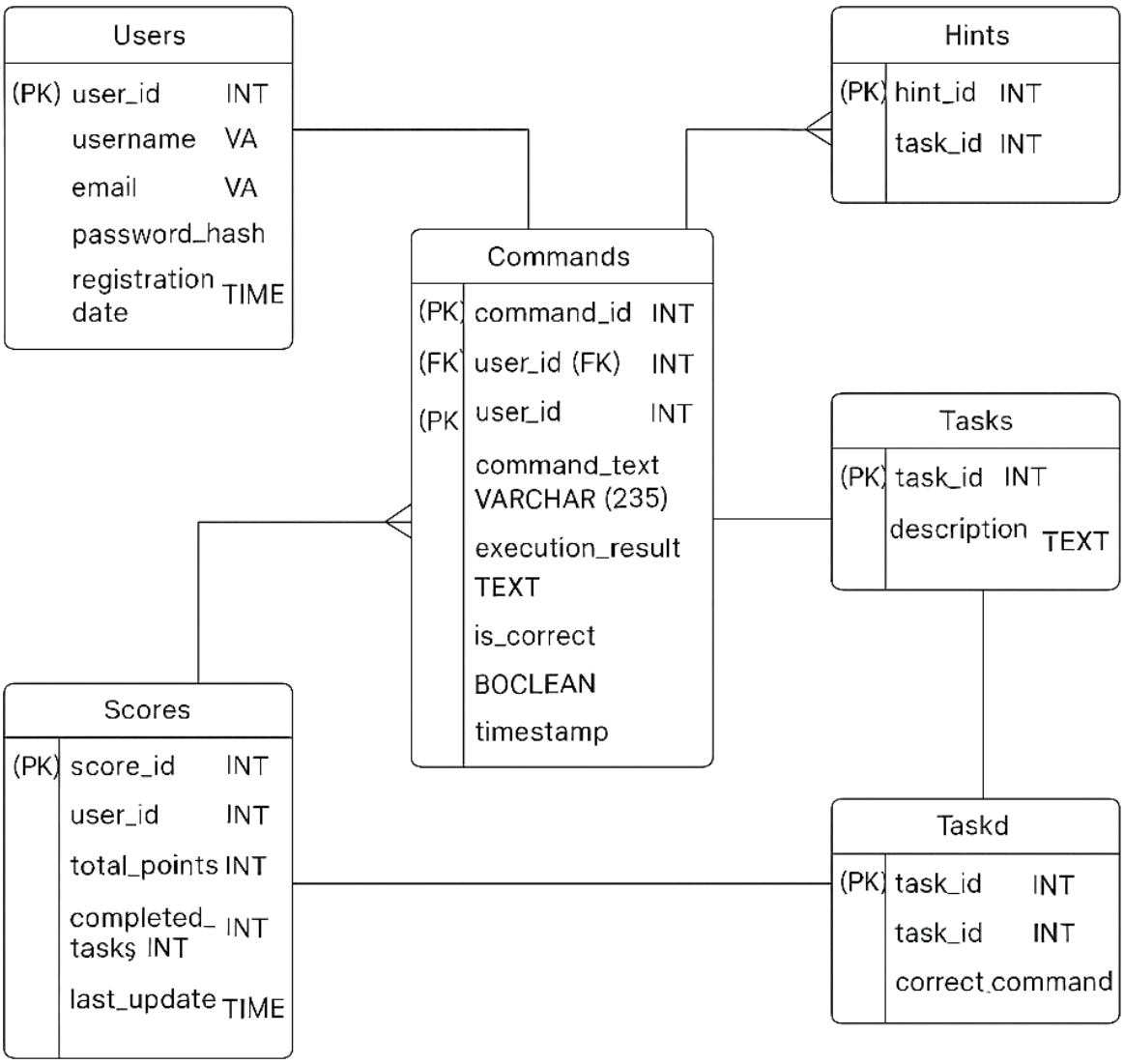

Рисунок 2.4 – ERD-діаграма ігрового тренажеру

Ця структура дозволяє ефективно зберігати дані та організовувати навчальний процес за допомогою ігрового тренажеру.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІГРОВОГО ТРЕНАЖЕРУ З ВИВЧЕННЯ ОСНОВ ВИКОРИСТАННЯ ТЕРМІНАЛУ UNIX-ПОДІБНИХ ОПЕРАЦІЙНИХ СИСТЕМ

3.1 Добір засобів розробки ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем

Перш за все, під час реалізації ігрового тренажеру нам потрібно було вирішити питання, яким чином вбудувати емулятор терміналу UNIX-подібної системи в наш ігровий інтерфейс. Дослідження показало, що це можна реалізувати кількома способами. Розглянемо них.

3.1.1 Використання WebAssembly (WASM)

WebAssembly дозволяє виконувати високопродуктивний код у браузері, що робить його чудовим інструментом для створення складних веб-додатків. Завдяки WebAssembly стало можливим запускати емулятори, такі як xterm.js або shell.js, без необхідності встановлення додаткового програмного забезпечення на комп'ютері користувача.

Ці емулятори відтворюють функціональність терміналів або командних оболонок і дозволяють взаємодіяти з операційною системою чи середовищем через браузер. xterm.js, зокрема, є модульною бібліотекою JavaScript для створення терміналів, яку можна інтегрувати в будь-який веб-додаток. В поєднанні з WebAssembly вона пропонує ще кращу продуктивність, прискорюючи обробку тексту, команд та даних.

Shell.js дозволяє реалізувати повноцінну підтримку командного рядка в браузері, яку можна використовувати для емуляції оболонок, таких як bash або zsh. Використання WebAssembly у цьому контексті допомагає оптимізувати роботу з великими обсягами обчислень, роблячи команди швидкими навіть для складних сценаріїв.

Ці можливості спрощують навчання основам роботи з терміналом, адже користувачам більше не потрібен доступ до фізично встановленого терміналу. Завдяки WebAssembly все це реалізується швидко, безпечно і без обмежень

сумісності. Також є можливість інтегрувати XLinux або інші емулятори, які працюють онлайн.

3.1.2 Використання JavaScript-емуляторів

xterm.js — це популярна бібліотека з відкритим вихідним кодом, яка надає можливість інтегрувати функціональні термінали безпосередньо у веб-додатки. Вона забезпечує потужний, гнучкий і легко налаштований інтерфейс командного рядка, який підтримує основні функціональності терміналу, такі як введення команд, виведення тексту, робота з потоком даних і емуляція багатьох особливостей традиційних терміналів.

Бібліотека написана на JavaScript і працює на основі технологій Canvas API, що дозволяє відображати імітацію терміналу прямо у веб-браузері. Вона широко підтримується на різних платформах, включаючи сучасні версії популярних браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge та інші. Крім того, xterm.js є модульною системою, що дозволяє розширювати її можливості за допомогою плагінів.

xterm.js активно використовується у таких додатках, як платформи для хмарного програмування, адміністрування серверів, DevOps-інструменти, а також різноманітні системи моніторингу, де потрібно управляти пристроями чи сервісами через термінал. Наприклад, бібліотека є основою таких проєктів, як Visual Studio Code (для інтегрованого терміналу) та інших популярних IDE.

Серед основних її переваг можна виділити:

- Високу продуктивність та хорошу підтримку емуляції протоколу UNIX-подібних терміналів.

- Універсальність і легкість інтеграції в існуючі веб-додатки.

- Можливість кастомізації зовнішнього вигляду та функціональності.

- Активну спільноту розробників і регулярні оновлення.

Інший приклад JavaScript-емулятора – це term.js. Це бібліотека, яка дозволяє інтегрувати термінальний інтерфейс прямо у веб-додаток. Використовуючи term.js, можна створити емуляцію терміналу для виконання команд прямо в браузері, що є зручним для створення веб-інструментів,

серверних панелей адміністрування, навчальних матеріалів або просто демонстраційного середовища.

Одним із ключових переваг term.js є його легкість та простота у використанні. Він добре підходить для інтеграції в існуючі веб-додатки, адже його базова функціональність достатньо зрозуміла, а налаштування не потребують складних конфігурацій. Крім того, term.js підтримує кольоровий текст, спеціальні символи та базові функції терміналу, включаючи обробку вводу та виводу. Важливо зазначити, що term.js побудований на основі WebSocket, що дозволяє йому передавати дані між клієнтом і сервером у режимі реального часу.

Використання емульованого терміналу через term.js може бути корисним у таких випадках:

1. Веб-інтерфейси для адміністрування серверів - надання доступу до функціоналу SSH через браузер.

2. Інструменти для навчання програмуванню - інтерактивне середовище для виконання команд і скриптів.

3. Зручна інтеграція в DevOps-додатки - управління проектами без потреби постійно відкривати локальні термінали.

4. Реалізація демонстраційних середовищ - презентація роботи програми у консолі для показу перед користувачами.

Таким чином term.js є корисним інструментом для створення динамічних веб-рішень, які потребують інтеграції термінального інтерфейсу.

3.1.3 Запуск через Docker або серверний бекенд

Ви можете розгорнути контейнер Linux на сервері, використовуючи такі технології, як Docker або Podman, що дозволяють створювати ізольовані середовища для запуску програмного забезпечення. Після розгортання контейнера можна організувати взаємодію між ним та користувачем через веб-інтерфейс, що забезпечить зручний доступ до функціоналу контейнера з браузера. Один із способів реалізації такої функціональності передбачає

використання WebSockets для передачі даних у режимі реального часу між браузером і сервером.

WebSockets забезпечують двосторонній зв'язок, завдяки якому сервер може передавати дані клієнту без необхідності повторних запитів, як це робиться у традиційній моделі HTTP-запитів. Це дозволяє створити інтерактивний веб-інтерфейс для користувача, з яким можна працювати динамічно.

Для реалізації такого підходу нам потрібно буде розгортання контейнера UNIX-подібної системи, наприклад, Linux. Виконати налаштування контейнера за допомогою такого програмного забезпечення, як Docker. На сервері має бути встановлений веб-додаток, який працює як посередник між веб-інтерфейсом у браузері та контейнером. Сервер може бути створений із використанням мови програмування, наприклад, Python, Node.js або Go. У рамках цього додатка потрібно налаштувати обробку WebSocket-з'єднань.

Сервер взаємодіє з контейнером, передаючи команди або отримуючи дані з нього. Наприклад, для доступу до командного рядка контейнера можна використати утиліту docker exec. Сервер надсилає команду до контейнера, а потім повертає результати її виконання до клієнта через WebSockets.

Веб-інтерфейс повинен бути інтерактивним і підтримувати підключення до WebSocket-сервера. Він може бути створений за допомогою HTML, CSS і JavaScript. Наприклад, можна розробити в ньому консоль, у якій користувач матиме можливість вводити команди, а відповіді від контейнера будуть відображатися в режимі реального часу.

Зважаючи на вимоги до нашого ігрового тренажера, для розробки було обране середовище виконання Node.js, що забезпечує високопродуктивний серверний JavaScript з підтримкою асинхронного програмування та широкого спектра бібліотек для різноманітних потреб. Завдяки цьому середовищу створення інтерактивних ігор стає більш ефективним і технологічно гнучким. Для реалізації імітації терміналу було вирішено використати бібліотеку xterm.js, яка надає потужний інструментарій для створення інтерактивного

користувацького інтерфейсу з відтворенням функціоналу реального терміналу. Вона дозволяє обробляти команди введення, відображати текстові результати, моделювати поведінку консолі та інтегруватися з серверною логікою через веб-сокети, що відповідає нашим потребам для створення динамічного навчального середовища. Таким чином, обраний стек технологій забезпечує оптимальну продуктивність, легкість масштабування та високу інтерактивність кінцевого продукту.

Для реалізації бекенду для веб-емулятора терміналу UNIX ми обрали Node.js як основний серверний середовище завдяки його високій продуктивності, асинхронному підходу до обробки операцій вводу/виводу та широкій підтримці необхідних модулів. Одним із ключових компонентів реалізації стало використання WebSockets — технології, що дозволяє встановлювати двосторонній, безперервний зв'язок між клієнтом і сервером. Це забезпечило інтерактивну взаємодію, завдяки якій термінал працює в реальному часі, відображаючи результати команд та приймаючи ввід користувача без затримок.

У серверній частині ми налаштували обробку підключень WebSocket, що дала можливість одразу отримувати введені команди від клієнта і пересилати їх для виконання через дочірній процес. За допомогою модуля `child_process`, який входить до стандартної бібліотеки Node.js, команда виконується безпосередньо на сервері в системній оболонці, а результати виконання спільно з будь-якими повідомленнями про помилки чи статуси передаються назад клієнту через активний WebSocket-з'єднання. Це дозволяє користувачеві взаємодіяти з терміналом, наче він працює на локальній UNIX-системі.

Для клієнтської частини була розроблена інтуїтивно зрозуміла веб-інтерфейс, який імітує вигляд класичного терміналу. Він створений із використанням HTML, CSS та JavaScript, а також включає функції додавання тем оформлення, зміни шрифтів та кольорів для покращення зручності роботи. Складність інтеграції такої функціональності стала можливою завдяки

подібності взаємодії між фронтендом і бекендом, яку забезпечують WebSockets: коли сервер отримує введену команду, клієнт отримує миттєвий результат виконання вже оформлений у відповідному вигляді.

Додатково були впроваджені механізми безпеки. Так, для захисту серверної системи виконувані команди були обмежені за правами доступу, щоб уникнути неконтрольованого виклику критичних функцій. Також реалізована система тайм-аутів в WebSocket-з'єднаннях, яка запобігає надмірному навантаженню сервера через довготривалі та неактивні сеанси.

Таким чином, використання Node.js та WebSockets дозволило створити ефективний, швидкий та безпечний бекенд для веб-емулятора терміналу UNIX, забезпечивши інтерактивну взаємодію й підтримку багатокористувацької роботи.

Для забезпечення доступу до бази даних нашого ігрового тренажеру ми використали Express.js, яка є популярним фреймворком для Node.js, що дозволяє створювати веб-додатки і API з високою продуктивністю та гнучкістю. Завдяки Express.js ми реалізували функціональність для обробки HTTP-запитів, встановили маршрути для взаємодії з базою даних, а також налаштували систему middleware для управління аутентифікацією, логуванням і обробкою помилок.

3.2 Приклади роботи ігрового тренажеру

Гра починається з відображення титульної сторінки (рис. 3.1), на якій ми бачимо коротку інформацію про гру та кнопку для її початку.

Також на головній сторінці є головне меню, за яким можна перейти до основних сторінок і режимів роботи програми.

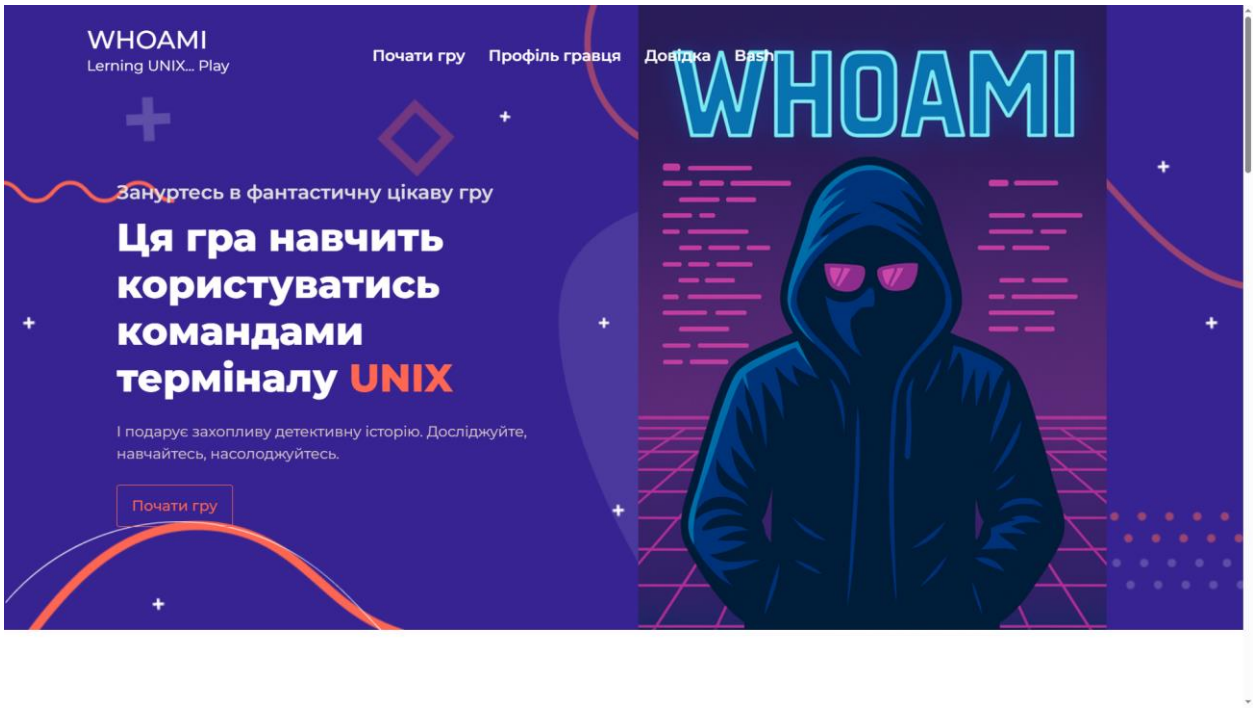

Рисунок 3.1 – Титульна сторінка гри

Почавши гру, ми переходимо до списку рівнів (рис. 3.2). Рівні об’єднані в глави, в яких розкривається основний сюжет історії. В кожній главі є декілька сцен, кожна з яких має познайомити з певними командами терміналу UNIX. Проходження кожної сцени піднімає рівень гравця та надає можливість перейти до наступної сцени або глави сюжету.

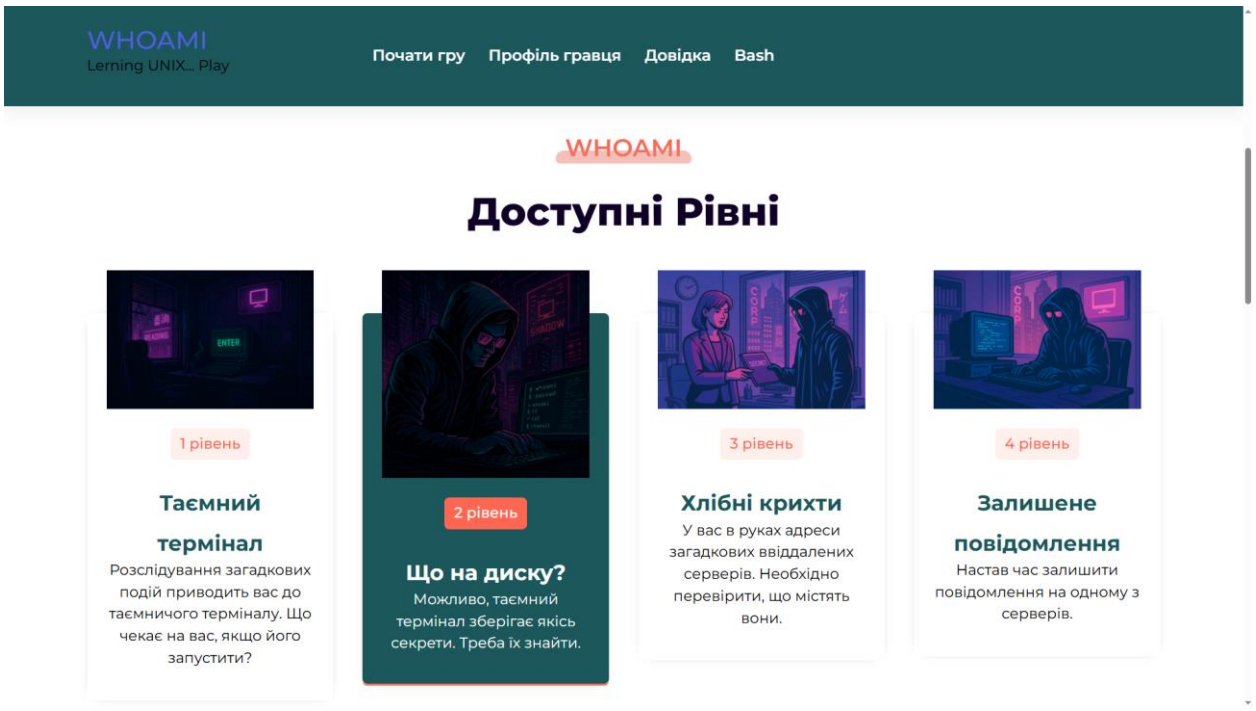

Рисунок 3.2 – Вибір рівня

В кожній сцені (рис. 3.3) гравцю дається завдання, що за сюжетом він має виконати. Це завдання так чи інакше пов’язано з введенням команд UNIX, іноді в комбінації з різними параметрами.

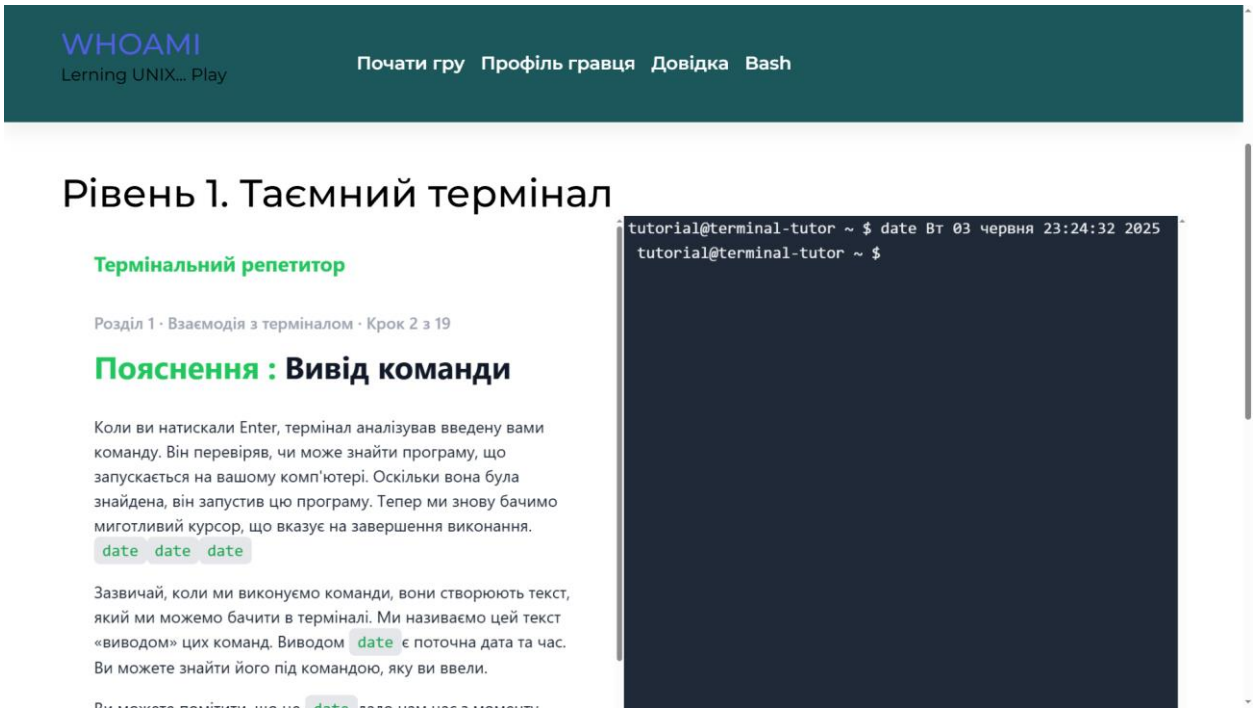

Рисунок 3.3 –Приклад роботи терміналу

Основний ігровий екран (рис. 3.4) розділено на дві частини. В правій знаходиться сам емулятор терміналу, де можна вводити команди та отримувати результати їх виконання.

В лівій частині знаходяться поради до виконання того чи іншого завдання.

За сюжетом наш головний герой – кібер-детектив – має вбудований гаджет з інтелектуальним помічником, який і дає поради, підказки, нагадує про цілі завдання тощо.

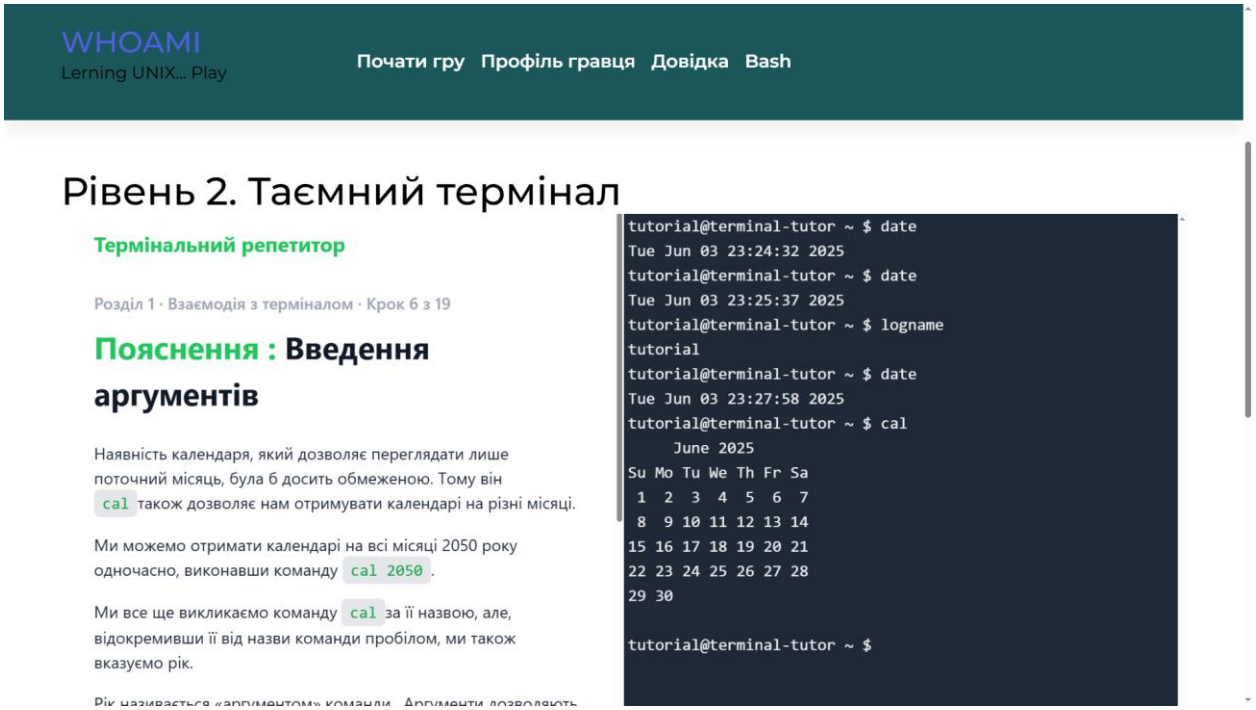

Рисунок 3.4 – Приклад введення команд в терміналі

Після успішного виконання завдання, рівень гравця підвищується. Також гра фіксує, які команди були успішно вивчені.

Свій поточний рівень та досягнення можна переглянути в профілі гравця (рис. 3.5).

Тут в профілі є можливість переглянути короткий опис пройденої історії, а також переглянути ті команди, що вже були вивчені.

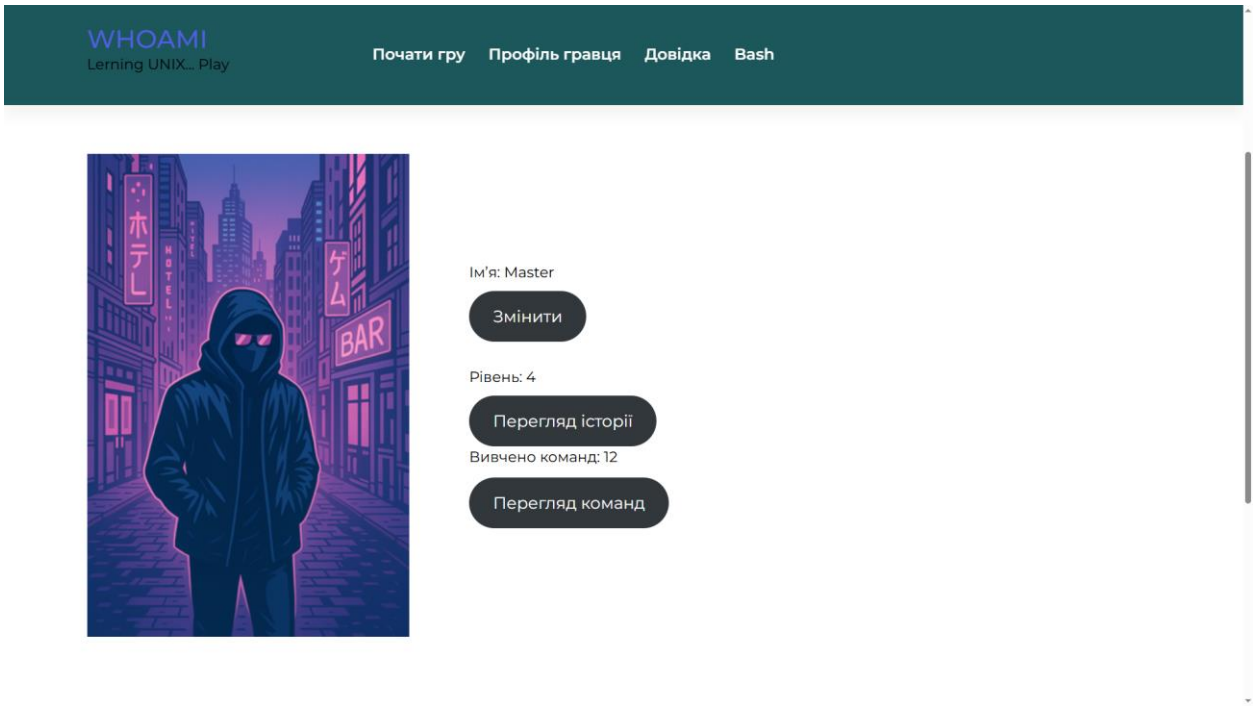

Рисунок 3.5 – Перегляд профілю гравця

Вивчені команди відображаються на окремій сторінці (рис. 3.6).

Також є можливість подивитись довідку по всім командам (рис. 3.7).

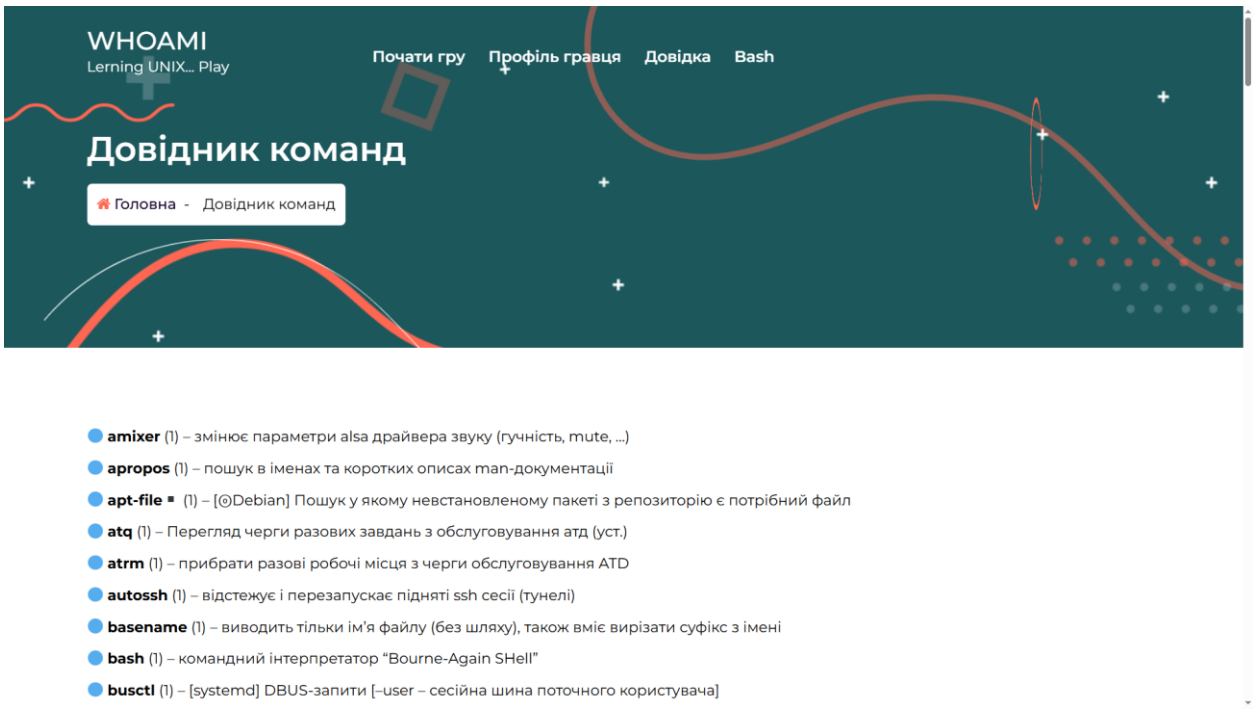

Рисунок 3.6 – Перегляд короткої довідки по командам

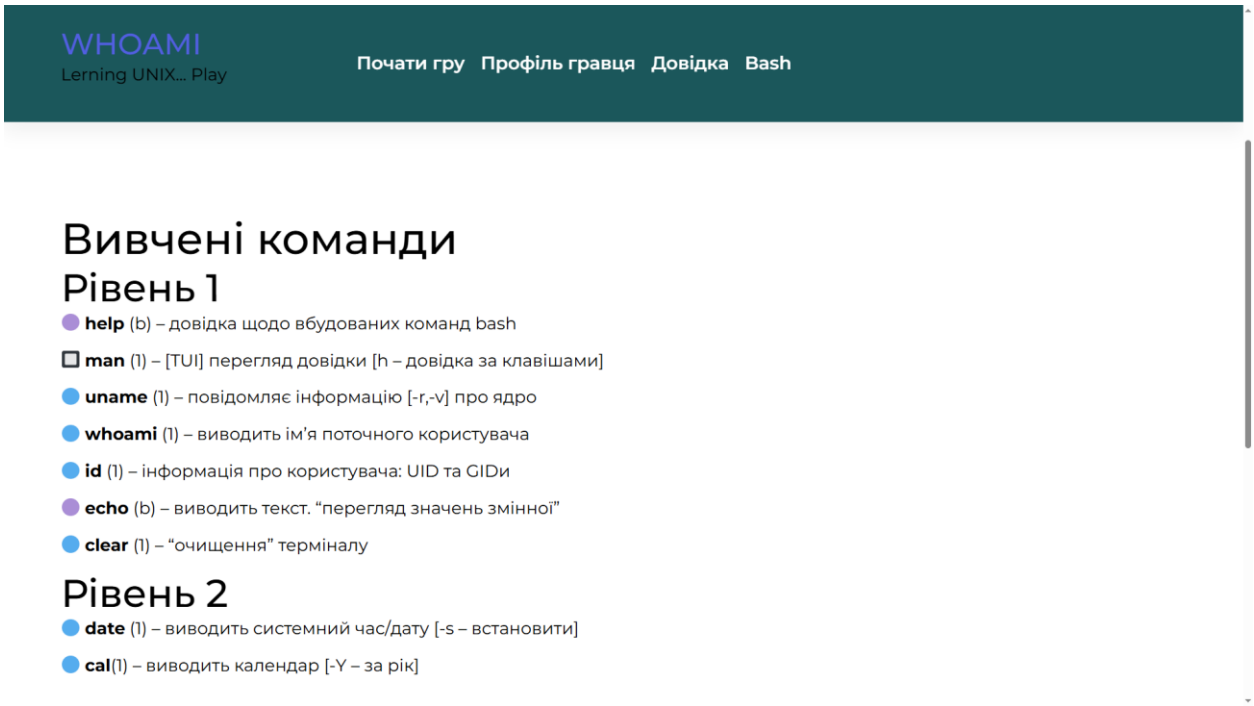

Рисунок 3.7 – Перегляд вивчених в грі команд

Завдяки цьому, цю програму можна використовувати як довідник та підручник по командам Unix-подібних систем.

Таким чином ми створили сюжетну гру, в якій гравець, крок за кроком відкриваючи сторінки цікавої історії, вивчає все нові і нові команди Unix, починаючи з найпростіших, і завершуючи скриптами, що дозволяють автоматизувати значну кількість роботи з операційною системою.

Така гра може зацікавити всіх, хто хоче самостійно опанувати команди UNIX, а також може використовуватись як допоможний матеріал під час викладання дисциплін, що передбачають вивчення UNIX-систем та їх команд.

ВИСНОВКИ

В кваліфікаційній роботі спроєктовано та реалізовано програмне забезпечення ігрового тренажеру, що дозволяє вивчати основні команди UNIX-подібних систем.

Було проведено детальний аналіз актуальності використання UNIX-подібних операційних систем в сучасній IT-індустрії, враховуючи масштаби їх застосування, функціональні можливості та ефективність в різноманітних сферах. Такі системи, як Linux, macOS та інші, отримали широке визнання завдяки стабільності, безпеці та відкритій архітектурі, що дозволяє адаптувати їх під конкретні задачі. UNIX-подібні системи активно використовуються в серверних технологіях, хмарних обчисленнях, розробці програмного забезпечення та адмініструванні мереж, демонструючи високу продуктивність та надійність.

Окрім цього, їх популярність серед розробників продовжує зростати завдяки зручним інструментам командного рядка, широкій підтримці серед фреймворків і бібліотек, а також активній спільноті, яка сприяє постійному розвитку й удосконаленню систем. У контексті кібербезпеки UNIX-подібні системи вирізняються сильними механізмами контролю доступу і можливістю швидкого реагування на загрози завдяки регулярним оновленням.

Таким чином, аналіз доводить, що UNIX-подібні операційні системи залишаються ключовим технологічним елементом для багатьох сучасних IT-проєктів, від стартапів до корпоративних рішень, забезпечуючи високу гнучкість і масштабованість у вирішенні складних завдань.

Оглянуто сучасні емулятори терміналу UNIX, серед яких як традиційні варіанти для досвідчених користувачів, так і інтерактивні емулятори, спеціально спрямовані на навчання новачків. Традиційні емулятори, такі як GNOME Terminal, Konsole і iTerm2, забезпечують широкий спектр функціональності, включаючи налаштування зовнішнього вигляду, підтримку вкладок, інтеграцію з редакторами, як-от Vim або Emacs, а також різні варіанти

перегляду історії команд. Вони націлені на професіоналів, які шукають стабільність, гнучкість і розширену функціональність для роботи в UNIX-середовищі.

Серед емуляторів для навчання, особливу увагу привертають інтерактивні платформи, такі як "Terminus" або "Webminal". Вони дозволяють новачкам освоїти основи командного рядка в ігровій формі, пропонуючи пояснення команд і можливість експериментувати без ризику зробити критичні помилки. Ці платформи також часто включають інтегровані навчальні ресурси, інтерфейс для оцінки прогресу користувача та демонстрації прикладів використання команд у різних сценаріях.

Таким чином, сучасні емулятори терміналу UNIX сьогодні не тільки виконують функцію інструменту для продуктивної роботи, але й слугують ефективним засобом для навчання, відкриваючи доступ до знань про UNIX-системи для широкої аудиторії.

Спроєктовано деталізовані вимоги до програмного забезпечення для створення ігрового тренажера, призначеного для вивчення основ роботи з терміналом UNIX. Цей тренажер має на меті забезпечити інтерактивне навчання для користувачів, які хочуть опанувати базові команди та навчитися ефективно працювати в терміналі.

Здійснено повне проєктування потоків даних для програмного забезпечення, яке використовується в ігровому тренажері, а також створено детальні діаграми використання. Проєктування потоків даних передбачало аналіз всіх ключових сценаріїв взаємодії між модулями системи, включаючи отримання, обробку, передачу та зберігання даних. Особливу увагу приділено інтеграції джерел даних, забезпеченню їхньої точності і безперебійності роботи системи.

Діаграми використання розроблені для візуального представлення функціональності тренажера, опису всіх можливих сценаріїв взаємодії користувача з системою, а також для демонстрації ролей і обов’язків різних компонентів тренажера. До діаграм включені основні функції, такі як

реєстрація користувача, запуск ігрових сесій, управління процесом навчання або тренування, аналіз статистики та збереження результатів.

Враховано потреби різних категорій користувачів: від гравців і тренерів до адміністраторів системи. Це дозволяє забезпечити легкість використання програмного забезпечення та ефективність його функціонування. Таким чином, виконане проєктування створює технологічний фундамент для подальшого впровадження системи ігрового тренажера, спрямованої на навчання, розвиток навичок чи розваги користувачів різного рівня підготовки.

Спроєктована структура бази даних для ігрового тренажера покликана забезпечити ефективне зберігання, організацію й обробку даних, що стосуються користувачів, їхніх досягнень, прогресу, сценаріїв ігрових вправ, результатів тестувань, а також налаштувань гри.

Було проведено детальний аналіз та виконано добір найбільш відповідних засобів для розробки ігрового тренажеру, призначеного для вивчення основ роботи з терміналом UNIX-подібних операційних систем. Вибір інструментів включав оцінку платформ для створення інтерактивного навчального середовища, систем для моделювання командного рядка, а також фреймворків і бібліотек для забезпечення гнучкості та масштабованості.

Програмне забезпечення ігрового тренажеру з вивчення основ використання терміналу UNIX-подібних операційних систем може використовуватись як для самостійного навчання, так і інтегруватись у відповідні навчальні курси.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unix. – URL: https://uk.wikipedia.org/wiki/Unix
2. Що таке Unix? Огляд UNIX-подібних систем . – URL: https://hyperhost.ua/info/uk/shcho-take-unix-oglyad-unix-podibnikh-sistem
3. Історія UNIX: Повний посібник. – URL: https://informatecdigital.com/uk/%D1%96%D1%81%D1%82%D0%BE%D1%80%D1%96%D1%8F-unix-%D0%BF%D0%BE%D0%B2%D0%BD%D0%B8%D0%B9-%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA/
4. 30 Linux Terminal Emulators. – URL: https://linuxblog.io/linux-terminal-emulators/
5. 7 кращих емуляторів терміналу для Ubuntu/Linux Mint – URL: https://linuxthebest.net/uk/7-luchshih-emulyatorov-terminala-dlya-ubuntu-linux-mint/
6. OverTheWire: Wargames – URL: https://overthewire.org/wargames/
7. Linux Survival | Where learning Linux is easy – URL: https://linuxsurvival.com/
8. Command Challenge! – URL: https://cmdchallenge.com/
9. Explain Shell – URL: https://explainshell.com/
10. Webminal - Learn and Practise Linux online, Programming online – URL: https://www.webminal.org/
11. WebAssembly – URL: https://webassembly.org/
12. Use XLinux online Linux - Virtual machine – URL: https://www.offidocs.com/index.php/desktop-online-utilities-apps/xlinux-online-linux
13. Xterm.js – URL: https://xtermjs.org/
14. xtermjs/xterm.js: A terminal for the web – URL: https://github.com/chjj/term.js
15. Node.js — Run JavaScript Everywhere – URL: https://nodejs.org/en

Додаток А

Текст програми

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>UNIX Термінал</title>
    <link                                          rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/xterm/4.19.0/xterm.
min.css">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/xterm/4.19.0/xterm.m
in.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #222;
            color: #fff;
            text-align: center;
        }
        #terminal-container {
            width: 80%;
            height: 500px;
            margin: auto;
            border: 1px solid #555;
            background: #000;
            padding: 10px;
            display: flex; /* Виправлення: використовує flex-
контейнер */
            align-items: center;
            justify-content: center;
        }
    </style>
</head>
<body>
    <h2>Емулятор UNIX Терміналу</h2>
    <div id="terminal-container"></div>

    <script>
        // Ініціалізація терміналу
        const term = new Terminal({
            cursorBlink: true,
            rows: 20,
            cols: 80
        });

        // Відкриваємо термінал у правильному контейнері
        term.open(document.getElementById('terminal-
container'));
```

```
            // Відображення початкового тексту
            term.writeln('Ласкаво просимо до UNIX терміналу!');
            term.writeln('Введіть команду, наприклад `ls`, `pwd`,
`echo "Hello"`');
            term.write('$ ');

            // Обробка введених команд
            term.onData(e => {
                term.write(e); // Відображення тексту команди

                const command = e.trim().toLowerCase();
                if (command === 'ls') {
                    term.writeln('\r\ndocuments        downloads
projects');
                } else if (command === 'pwd') {
                    term.writeln('\r\n/home/user');
                } else if (command.startsWith('echo ')) {
                    term.writeln(`\r\n${command.slice(5)}`);
                } else if (command === 'clear') {
                    term.clear();
                } else {
                    term.writeln('\r\nНевідома команда');
                }

                term.write('\r\n$ '); // Відображення запрошення
після виконання команди
            });
        </script>
    </body>
    </html>

    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <title>UNIX Термінал</title>
        <script      src="https://cdn.jsdelivr.net/npm/terminal-
js"></script>
        <style>
            body {
                font-family: Arial, sans-serif;
                text-align: center;
                background-color: #222;
                color: #fff;
            }
            #terminal-container {
                width: 80%;
                height: 500px; /* Забезпечує щонайменше 20 рядків
*/
                margin: auto;
                border: 1px solid #555;
                background: #000;
                padding: 10px;
```

```
                overflow: auto;
            }
        </style>
    </head>
    <body>
        <h2>Емулятор UNIX Терміналу</h2>
        <div id="terminal-container"></div>

        <script>
            const term = new Terminal();
            term.setHeight(500); // Гарантує мінімум 20 рядків
тексту
            term.open(document.getElementById('terminal-
container'));

            term.print('Ласкаво просимо до емулятора UNIX!');
            term.print('Введіть команду, наприклад `ls`, `pwd`,
`echo "Hello"`');

            function runCommand(command) {
                command = command.trim().toLowerCase();
                if (command === 'ls') {
                    term.print('documents  downloads  projects');
                } else if (command === 'pwd') {
                    term.print('/home/user');
                } else if (command.startsWith('echo ')) {
                    term.print(command.slice(5));
                } else {
                    term.print('Невідома команда');
                }
                term.input('$ ', runCommand);
            }

            term.input('$ ', runCommand);
        </script>
    </body>
    </html>


    #!/usr/bin/env node

    /**
     * term.js
     * Copyright (c) 2012-2013, Christopher Jeffrey (MIT License)
     */

    var http = require('http')
      , express = require('express')
      , io = require('socket.io')
      , pty = require('pty.js')
      , terminal = require('../');

    /**
```

```
 * term.js
 */

process.title = 'term.js';

/**
 * Dump
 */

var stream;
if (process.argv[2] === '--dump') {
  stream = require('fs').createWriteStream(__dirname + '/dump.log');
}

/**
 * Open Terminal
 */

var buff = []
  , socket
  , term;

term = pty.fork(process.env.SHELL || 'sh', [], {
  name: require('fs').existsSync('/usr/share/terminfo/x/xterm-256color')
    ? 'xterm-256color'
    : 'xterm',
  cols: 80,
  rows: 24,
  cwd: process.env.HOME
});

term.on('data', function(data) {
  if (stream) stream.write('OUT: ' + data + '\n-\n');
  return !socket
    ? buff.push(data)
    : socket.emit('data', data);
});

console.log(''
  + 'Created shell with pty master/slave'
  + ' pair (master: %d, pid: %d)',
  term.fd, term.pid);

/**
 * App & Server
 */

var app = express()
  , server = http.createServer(app);

app.use(function(req, res, next) {
```

```
    var setHeader = res.setHeader;
    res.setHeader = function(name) {
      switch (name) {
        case 'Cache-Control':
        case 'Last-Modified':
        case 'ETag':
          return;
      }
      return setHeader.apply(res, arguments);
    };
    next();
  });

  app.use(express.basicAuth(function(user, pass, next) {
    if (user !== 'foo' || pass !== 'bar') {
      return next(true);
    }
    return next(null, user);
  }));

  app.use(express.static(__dirname));
  app.use(terminal.middleware());

  if (!~process.argv.indexOf('-n')) {
    server.on('connection', function(socket) {
      var address = socket.remoteAddress;
      if (address !== '127.0.0.1' && address !== '::1') {
        try {
          socket.destroy();
        } catch (e) {
          ;
        }
        console.log('Attempted connection from %s. Refused.',
address);
      }
    });
  }

  server.listen(8080);

  /**
   * Sockets
   */

  io = io.listen(server, {
    log: false
  });

  io.sockets.on('connection', function(sock) {
    socket = sock;

    socket.on('data', function(data) {
      if (stream) stream.write('IN: ' + data + '\n-\n');
```

```
    //console.log(JSON.stringify(data));
    term.write(data);
  });

  socket.on('disconnect', function() {
    socket = null;
  });

  while (buff.length) {
    socket.emit('data', buff.shift());
  }
});



<!doctype html>
<title>term.js</title>
<!--
  term.js
  Copyright (c) 2012-2013, Christopher Jeffrey (MIT License)
-->
<style>
  html {
    background: #555;
  }

  h1 {
    margin-bottom: 20px;
    font: 20px/1.5 sans-serif;
  }

/*
  .terminal {
    float: left;
    border: #000 solid 5px;
    font-family: "DejaVu Sans Mono", "Liberation Mono",
monospace;
    font-size: 11px;
    color: #f0f0f0;
    background: #000;
  }

  .terminal-cursor {
    color: #000;
    background: #f0f0f0;
  }
*/
</style>
<h1>term.js</h1>
<script src="/socket.io/socket.io.js"></script>
<script src="term.js"></script>
<script>
;(function() {
```

```
      window.onload = function() {
        var socket = io.connect();
        socket.on('connect', function() {
          var term = new Terminal({
            cols: 80,
            rows: 24,
            useStyle: true,
            screenKeys: true,
            cursorBlink: false
          });

          term.on('data', function(data) {
            socket.emit('data', data);
          });

          term.on('title', function(title) {
            document.title = title;
          });

          term.open(document.body);

          term.write('\x1b[31mWelcome to term.js!\x1b[m\r\n');

          socket.on('data', function(data) {
            term.write(data);
          });

          socket.on('disconnect', function() {
            term.destroy();
          });
        });
      };
    }).call(this);
    </script>



    function EventEmitter() {
      this._events = this._events || {};
    }

    EventEmitter.prototype.addListener     =     function(type,
listener) {
      this._events[type] = this._events[type] || [];
      this._events[type].push(listener);
    };

    EventEmitter.prototype.on                                    =
EventEmitter.prototype.addListener;

    EventEmitter.prototype.removeListener    =    function(type,
listener) {
      if (!this._events[type]) return;
```

```
    var obj = this._events[type]
      , i = obj.length;

    while (i--) {
      if (obj[i] === listener || obj[i].listener === listener)
{
        obj.splice(i, 1);
        return;
      }
    }
  };

  EventEmitter.prototype.off                                    =
EventEmitter.prototype.removeListener;

  EventEmitter.prototype.removeAllListeners = function(type) {
    if (this._events[type]) delete this._events[type];
  };

  EventEmitter.prototype.once = function(type, listener) {
    function on() {
      var args = Array.prototype.slice.call(arguments);
      this.removeListener(type, on);
      return listener.apply(this, args);
    }
    on.listener = listener;
    return this.on(type, on);
  };

  EventEmitter.prototype.emit = function(type) {
    if (!this._events[type]) return;

    var args = Array.prototype.slice.call(arguments, 1)
      , obj = this._events[type]
      , l = obj.length
      , i = 0;

    for (; i < l; i++) {
      obj[i].apply(this, args);
    }
  };

  EventEmitter.prototype.listeners = function(type) {
    return this._events[type] = this._events[type] || [];
  };

  /**
   * Stream
   */

  function Stream() {
    EventEmitter.call(this);
```

```
}

inherits(Stream, EventEmitter);

Stream.prototype.pipe = function(dest, options) {
  var src = this
    , ondata
    , onerror
    , onend;

  function unbind() {
    src.removeListener('data', ondata);
    src.removeListener('error', onerror);
    src.removeListener('end', onend);
    dest.removeListener('error', onerror);
    dest.removeListener('close', unbind);
  }

  src.on('data', ondata = function(data) {
    dest.write(data);
  });

  src.on('error', onerror = function(err) {
    unbind();
    if (!this.listeners('error').length) {
      throw err;
    }
  });

  src.on('end', onend = function() {
    dest.end();
    unbind();
  });

  dest.on('error', onerror);
  dest.on('close', unbind);

  dest.emit('pipe', src);

  return dest;
};

/**
 * States
 */

var normal = 0
  , escaped = 1
  , csi = 2
  , osc = 3
  , charset = 4
  , dcs = 5
  , ignore = 6
```

```
      , UDK = { type: 'udk' };

    /**
     * Terminal
     */

    function Terminal(options) {
      var self = this;

      if (!(this instanceof Terminal)) {
        return    new    Terminal(arguments[0],    arguments[1],
arguments[2]);
      }

      Stream.call(this);

      if (typeof options === 'number') {
        options = {
          cols: arguments[0],
          rows: arguments[1],
          handler: arguments[2]
        };
      }

      options = options || {};

      each(keys(Terminal.defaults), function(key) {
        if (options[key] == null) {
          options[key] = Terminal.options[key];
          // Legacy:
          if (Terminal[key] !== Terminal.defaults[key]) {
            options[key] = Terminal[key];
          }
        }
        self[key] = options[key];
      });

      if (options.colors.length === 8) {
        options.colors                                          =
options.colors.concat(Terminal._colors.slice(8));
      } else if (options.colors.length === 16) {
        options.colors                                          =
options.colors.concat(Terminal._colors.slice(16));
      } else if (options.colors.length === 10) {
        options.colors = options.colors.slice(0, -2).concat(
          Terminal._colors.slice(8, -2), options.colors.slice(-
2));
      } else if (options.colors.length === 18) {
        options.colors = options.colors.slice(0, -2).concat(
          Terminal._colors.slice(16, -2), options.colors.slice(-
2));
      }
      this.colors = options.colors;
```

```
        this.options = options;

        // this.context = options.context || window;
        // this.document = options.document || document;
        this.parent = options.body || options.parent
          || (document ? document.getElementsByTagName('body')[0]
: null);

        this.cols = options.cols || options.geometry[0];
        this.rows = options.rows || options.geometry[1];

        // Act as though we are a node TTY stream:
        this.setRawMode;
        this.isTTY = true;
        this.isRaw = true;
        this.columns = this.cols;
        this.rows = this.rows;

        if (options.handler) {
          this.on('data', options.handler);
        }

        this.ybase = 0;
        this.ydisp = 0;
        this.x = 0;
        this.y = 0;
        this.cursorState = 0;
        this.cursorHidden = false;
        this.convertEol;
        this.state = 0;
        this.queue = '';
        this.scrollTop = 0;
        this.scrollBottom = this.rows - 1;

        // modes
        this.applicationKeypad = false;
        this.applicationCursor = false;
        this.originMode = false;
        this.insertMode = false;
        this.wraparoundMode = false;
        this.normal = null;

        // select modes
        this.prefixMode = false;
        this.selectMode = false;
        this.visualMode = false;
        this.searchMode = false;
        this.searchDown;
        this.entry = '';
        this.entryPrefix = 'Search: ';
        this._real;
        this._selected;
```

```
  this._textarea;

  // charset
  this.charset = null;
  this.gcharset = null;
  this.glevel = 0;
  this.charsets = [null];

  // mouse properties
  this.decLocator;
  this.x10Mouse;
  this.vt200Mouse;
  this.vt300Mouse;
  this.normalMouse;
  this.mouseEvents;
  this.sendFocus;
  this.utfMouse;
  this.sgrMouse;
  this.urxvtMouse;

  // misc
  this.element;
  this.children;
  this.refreshStart;
  this.refreshEnd;
  this.savedX;
  this.savedY;
  this.savedCols;

  // stream
  this.readable = true;
  this.writable = true;

  this.defAttr = (0 << 18) | (257 << 9) | (256 << 0);
  this.curAttr = this.defAttr;

  this.params = [];
  this.currentParam = 0;
  this.prefix = '';
  this.postfix = '';

  this.lines = [];
  var i = this.rows;
  while (i--) {
    this.lines.push(this.blankLine());
  }

  this.tabs;
  this.setupStops();
}

inherits(Terminal, Stream);
```

```
/**
 * Colors
 */

// Colors 0-15
Terminal.tangoColors = [
  // dark:
  '#2e3436',
  '#cc0000',
  '#4e9a06',
  '#c4a000',
  '#3465a4',
  '#75507b',
  '#06989a',
  '#d3d7cf',
  // bright:
  '#555753',
  '#ef2929',
  '#8ae234',
  '#fce94f',
  '#729fcf',
  '#ad7fa8',
  '#34e2e2',
  '#eeeeec'
];

Terminal.xtermColors = [
  // dark:
  '#000000', // black
  '#cd0000', // red3
  '#00cd00', // green3
  '#cdcd00', // yellow3
  '#0000ee', // blue2
  '#cd00cd', // magenta3
  '#00cdcd', // cyan3
  '#e5e5e5', // gray90
  // bright:
  '#7f7f7f', // gray50
  '#ff0000', // red
  '#00ff00', // green
  '#ffff00', // yellow
  '#5c5cff', // rgb:5c/5c/ff
  '#ff00ff', // magenta
  '#00ffff', // cyan
  '#ffffff'  // white
];

// Colors 0-15 + 16-255
// Much thanks to TooTallNate for writing this.
Terminal.colors = (function() {
  var colors = Terminal.tangoColors.slice()
    , r = [0x00, 0x5f, 0x87, 0xaf, 0xd7, 0xff]
    , i;
```

```
  // 16-231
  i = 0;
  for (; i < 216; i++) {
    out(r[(i / 36) % 6 | 0], r[(i / 6) % 6 | 0], r[i % 6]);
  }

  // 232-255 (grey)
  i = 0;
  for (; i < 24; i++) {
    r = 8 + i * 10;
    out(r, r, r);
  }

  function out(r, g, b) {
    colors.push('#' + hex(r) + hex(g) + hex(b));
  }

  function hex(c) {
    c = c.toString(16);
    return c.length < 2 ? '0' + c : c;
  }

  return colors;
})();

// Default BG/FG
Terminal.colors[256] = '#000000';
Terminal.colors[257] = '#f0f0f0';

Terminal._colors = Terminal.colors.slice();

Terminal.vcolors = (function() {
  var out = []
    , colors = Terminal.colors
    , i = 0
    , color;

  for (; i < 256; i++) {
    color = parseInt(colors[i].substring(1), 16);
    out.push([
      (color >> 16) & 0xff,
      (color >> 8) & 0xff,
      color & 0xff
    ]);
  }

  return out;
})();

/**
 * Options
 */
```

```
Terminal.defaults = {
  colors: Terminal.colors,
  convertEol: false,
  termName: 'xterm',
  geometry: [80, 24],
  cursorBlink: true,
  visualBell: false,
  popOnBell: false,
  scrollback: 1000,
  screenKeys: false,
  debug: false,
  useStyle: false
  // programFeatures: false,
  // focusKeys: false,
};

Terminal.options = {};

each(keys(Terminal.defaults), function(key) {
  Terminal[key] = Terminal.defaults[key];
  Terminal.options[key] = Terminal.defaults[key];
});

/**
 * Focused Terminal
 */

Terminal.focus = null;

Terminal.prototype.focus = function() {
  if (Terminal.focus === this) return;

  if (Terminal.focus) {
    Terminal.focus.blur();
  }

  if (this.sendFocus) this.send('\x1b[I');
  this.showCursor();

  // try {
  //   this.element.focus();
  // } catch (e) {
  //   ;
  // }

  // this.emit('focus');

  Terminal.focus = this;
};

Terminal.prototype.blur = function() {
  if (Terminal.focus !== this) return;
```

```
    this.cursorState = 0;
    this.refresh(this.y, this.y);
    if (this.sendFocus) this.send('\x1b[O');

    // try {
    //   this.element.blur();
    // } catch (e) {
    //   ;
    // }

    // this.emit('blur');

    Terminal.focus = null;
  };

  /**
   * Initialize global behavior
   */

  Terminal.prototype.initGlobal = function() {
    var document = this.document;

    Terminal._boundDocs = Terminal._boundDocs || [];
    if (~indexOf(Terminal._boundDocs, document)) {
      return;
    }
    Terminal._boundDocs.push(document);

    Terminal.bindPaste(document);

    Terminal.bindKeys(document);

    Terminal.bindCopy(document);

    if (this.isMobile) {
      this.fixMobile(document);
    }

    if (this.useStyle) {
      Terminal.insertStyle(document,         this.colors[256],
this.colors[257]);
    }
  };

  /**
   * Bind to paste event
   */

  Terminal.bindPaste = function(document) {
    // This seems to work well for ctrl-V and middle-click,
    // even without the contentEditable workaround.
    var window = document.defaultView;
```

```
  on(window, 'paste', function(ev) {
    var term = Terminal.focus;
    if (!term) return;
    if (ev.clipboardData) {
      term.send(ev.clipboardData.getData('text/plain'));
    } else if (term.context.clipboardData) {
      term.send(term.context.clipboardData.getData('Text'));
    }
    // Not necessary. Do it anyway for good measure.
    term.element.contentEditable = 'inherit';
    return cancel(ev);
  });
};

/**
 * Global Events for key handling
 */

Terminal.bindKeys = function(document) {
  // We should only need to check `target === body` below,
  // but we can check everything for good measure.
  on(document, 'keydown', function(ev) {
    if (!Terminal.focus) return;
    var target = ev.target || ev.srcElement;
    if (!target) return;
    if (target === Terminal.focus.element
        || target === Terminal.focus.context
        || target === Terminal.focus.document
        || target === Terminal.focus.body
        || target === Terminal._textarea
        || target === Terminal.focus.parent) {
      return Terminal.focus.keyDown(ev);
    }
  }, true);

  on(document, 'keypress', function(ev) {
    if (!Terminal.focus) return;
    var target = ev.target || ev.srcElement;
    if (!target) return;
    if (target === Terminal.focus.element
        || target === Terminal.focus.context
        || target === Terminal.focus.document
        || target === Terminal.focus.body
        || target === Terminal._textarea
        || target === Terminal.focus.parent) {
      return Terminal.focus.keyPress(ev);
    }
  }, true);

  // If we click somewhere other than a
  // terminal, unfocus the terminal.
  on(document, 'mousedown', function(ev) {
    if (!Terminal.focus) return;
```

```
    var el = ev.target || ev.srcElement;
    if (!el) return;

    do {
      if (el === Terminal.focus.element) return;
    } while (el = el.parentNode);

    Terminal.focus.blur();
  });
};



Terminal.prototype.refresh = function(start, end) {
  var x
    , y
    , i
    , line
    , out
    , ch
    , width
    , data
    , attr
    , bg
    , fg
    , flags
    , row
    , parent;

  if (end - start >= this.rows / 2) {
    parent = this.element.parentNode;
    if (parent) parent.removeChild(this.element);
  }

  width = this.cols;
  y = start;

  if (end >= this.lines.length) {
    this.log('`end` is too large. Most likely a bad CSR.');
    end = this.lines.length - 1;
  }

  for (; y <= end; y++) {
    row = y + this.ydisp;

    line = this.lines[row];
    out = '';

    if (y === this.y
        && this.cursorState
        && (this.ydisp === this.ybase || this.selectMode)
        && !this.cursorHidden) {
```

```
          x = this.x;
        } else {
          x = -1;
        }

        attr = this.defAttr;
        i = 0;

        for (; i < width; i++) {
          data = line[i][0];
          ch = line[i][1];

          if (i === x) data = -1;

          if (data !== attr) {
            if (attr !== this.defAttr) {
              out += '</span>';
            }
            if (data !== this.defAttr) {
              if (data === -1) {
                out  +=  '<span  class="reverse-video  terminal-
cursor">';
              } else {
                out += '<span style="';

                bg = data & 0x1ff;
                fg = (data >> 9) & 0x1ff;
                flags = data >> 18;

                // bold
                if (flags & 1) {
                  if (!Terminal.brokenBold) {
                    out += 'font-weight:bold;';
                  }
                  // See: XTerm*boldColors
                  if (fg < 8) fg += 8;
                }

                // underline
                if (flags & 2) {
                  out += 'text-decoration:underline;';
                }

                // blink
                if (flags & 4) {
                  if (flags & 2) {
                    out = out.slice(0, -1);
                    out += ' blink;';
                  } else {
                    out += 'text-decoration:blink;';
                  }
                }
```

```
        // inverse
        if (flags & 8) {
          bg = (data >> 9) & 0x1ff;
          fg = data & 0x1ff;
          // Should inverse just be before the
          // above boldColors effect instead?
          if ((flags & 1) && fg < 8) fg += 8;
        }

        // invisible
        if (flags & 16) {
          out += 'visibility:hidden;';
        }

        // out += '" class="'
        //   + 'term-bg-color-' + bg
        //   + ' '
        //   + 'term-fg-color-' + fg
        //   + '">';

        if (bg !== 256) {
          out += 'background-color:'
            + this.colors[bg]
            + ';';
        }

        if (fg !== 257) {
          out += 'color:'
            + this.colors[fg]
            + ';';
        }

        out += '">';
      }
    }
  }

  switch (ch) {
    case '&':
      out += '&amp;';
      break;
    case '<':
      out += '&lt;';
      break;
    case '>':
      out += '&gt;';
      break;
    default:
      if (ch <= ' ') {
        out += ' ';
      } else {
        if (isWide(ch)) i++;
        out += ch;
```

```
          }
          break;
      }

      attr = data;
    }

    if (attr !== this.defAttr) {
      out += '</span>';
    }

    this.children[y].innerHTML = out;
  }

  if (parent) parent.appendChild(this.element);
};

Terminal.prototype._cursorBlink = function() {
  if (Terminal.focus !== this) return;
  this.cursorState ^= 1;
  this.refresh(this.y, this.y);
};

Terminal.prototype.showCursor = function() {
  if (!this.cursorState) {
    this.cursorState = 1;
    this.refresh(this.y, this.y);
  } else {
    // Temporarily disabled:
    // this.refreshBlink();
  }
};

Terminal.prototype.startBlink = function() {
  if (!this.cursorBlink) return;
  var self = this;
  this._blinker = function() {
    self._cursorBlink();
  };
  this._blink = setInterval(this._blinker, 500);
};

Terminal.prototype.refreshBlink = function() {
  if (!this.cursorBlink || !this._blink) return;
  clearInterval(this._blink);
  this._blink = setInterval(this._blinker, 500);
};
```