ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра комп'ютерної інженерії

ОПЕРАЦІЙНА СИТЕМА LINUX:
ПРИНЦИПИ РОБОТИ З ФАЙЛОВОЮ СИСТЕМОЮ

Навчальний посібник

Київ 2021

УДК 004.45.

Укладачі: В.М. Черевик, Л.І. Танцюра, С.С. Коротков, В.О. Сосновий.

Затверджено на засіданні Вченої ради Навчально-наукового інституту інформаційних технологій.

“26“ лютого 2021 року, протокол № 7.

Рецензенти:

д. т. н. Макаренко А.О.

к. т. н. Варфоломеєва О.Г.

Навчальний посібник “Операційна ситема Linux: принципи роботи з файловою системою ” / Уклад.: В.М. Черевик, Л.І. Танцюра, С.С. Коротков, В.О. Сосновий. - К.: ДУТ, 2021. 147 с.

Посібник розкриває зміст сеансу роботи в Linux, виводить основні поняття терміналу і командного рядку, надає структуру файлової системи Linux, показує основні принципи роботи з файловою системою, має інформацію про процеси та доступи до об’єктів обробки, знайомить з основними прийомами роботи з текстовими даними та можливостями командної оболонки і описує базові відомості про текстові редактори vi і emacs.

Ця основа знань і умінь надає можливість утворити базу на якій створюються умови якісно проводити подальші навчання в області операційних середовищ, систем і оболонок, а також, адміністрування системами.

ЗМІСТ

ПЕРЕДМОВА

Системне програмне забезпечення - комплекс програм, які забезпечують ефективне управління такими компонентами обчислювальної системи, як процесор, оперативна пам'ять, пристрої введення-виведення, мережеве обладнання та ін.

Системне програмне забезпечення виступає в ролі " інтерфейсу". З одного боку цього "міжшарового інтерфейсу" - апаратура комп'ютера, а з іншого - додатки користувача. На відміну від прикладного програмного забезпечення, системне ПЗ не вирішує конкретних прикладних задач користувача, а лише забезпечує роботу інших програм, управляє апаратними ресурсами обчислювальної системи і т.д.

При вивченні дисциплін професійного спрямування за спеціальністю *Комп'ютерна інженерія* має приділятися увага операційним системам, середовищам і системам програмування. Саме в такому ключі в основному і будується цей навчальний посібник.

Основне завдання навчального посібника: вивчення головних критеріїв управління ком'ютерними системами, оптимальне використання їх ресурсів та підтримка роботи систем у заданих режимах, які використовуються у практичній діяльності за фахом.

Основні знання та вміння, на які орієнтовно посібник

знання:

– призначення та устрій операційних систем, принципи функціонування основних модулів операційних систем;

– операційну систему Linux і мати навички, необхідні для її підтримки;

вміння:

– встановлювати, налаштовувати та обслуговувати операційну систему Linux;

– проводити профілактичні роботи по обслуговуванню операційної системи Linux;

– виконувати пошук і усунення несправностей в операційній системі Linux.

1 СЕАНС РОБОТИ В ОС LINUX

1.1 Користувачі системи

Між включенням живлення комп'ютера і моментом, коли система готова до роботи з користувачем, відбувається процедура завантаження системи. У процесі завантаження буде запущено основну управляючу програму (ядро), визначено і ініційовано обладнання яке є, активовані мережеві з'єднання, запущені системні служби. В Linux під час завантаження, на екран виводяться діагностичні повідомлення про дії, які відбуваються, якщо все у порядку і не виникало ніяких помилок, завантаження завершиться виводом на екран запрошення «login:». Воно може бути оформлено по різному, в залежності від налаштування системи, воно може відображатися у красиво оформленому вікні або у вигляді простого текстового рядку зверху екрану. Це запрошення до реєстрації у системі: система очікує, що у відповідь на запрошення буде введено вхідне ім'я користувача, який починає роботу. Звісно, має сенс вводити таке ім'я, яке вже відомо системі, щоби вона могла «впізнати», з ким має працювати, команди «невідомого» Linux виконувати відмовиться.

1.1.1 Багатокористувацька модель розмежування доступу

Процедура реєстрації в системі *обов'язкова* для Linux, робити в системі не зареєструвавшись під тим або іншим ім'ям користувача, просто не *можливо*. Для кожного користувача визначена сфера його повноважень в системі: програми, які він може запускати, файли, які він має право продивлятися, змінювати, знищувати. При спробі зробити щось, що виходить за рамки його повноважень, користувач отримає повідомлення про помилку. Така строгість може здатися необов'язковою, якщо користувачі комп'ютера довіряють один одному, особливо якщо у комп'ютера один користувач. Така ситуація дуже поширена в наш час. Але користувач, як правило, не може працювати цілком ізольовано. Як правило, він виконує якусь частку загальної задачі. Крім того, основна маса користувачів є учасниками мережевого простору. Відокремленості робочого місця від інформаційного середовища практично не існує. Тому, вимагається, слідкувати за тим, щоби не вийшло випадкового втручання користувачів в чужу роботу і псування чужих даних (файлів). Треба виділяти кожному машинний час (по можливості, уникати простої), виділяти простір на диску і не допускати узурпації усіх ресурсів одним користувачем і його задачею, а рівномірно ділити ресурси між усіма. Для такої системи принципово важливо знати, кому належать задачі і файли, тому виникла необхідність надавати доступ до ресурсів системи тільки після того, як користувач *зареєструється в системі* під тим або іншим іменем.

Така модель реалізована в багатокористувацькій операційній системі Unix. Саме від неї Linux – також багатокористувацька система – успадкував принципи роботи з користувачами. Це не просто данина традиції або намагання до універсальності: багатокористувацька модель дозволяє вирішити ряд задач, вельми актуальних і для сучасних комп'ютерів, и для серверів, які працюють в локальних і глобальних мережах, і в загалі в любих системах, одночасно виконуючих *різні* завдання, відповідають за які *різні* люди.

Комп'ютер – це всього лише інструмент для рішення різного роду прикладних задач: від набору і роздрукування текстів до обчислень. Складність є в тому, що для зміни цього інструменту і для роботи з його допомогою використовуються одні і ті ж операції: зміна файлів, виконання програм. Виходить, якщо недотримуватися обережності, побічним результатом роботи може стати вихід із ладу самої системи. В багатокористувацькій моделі ця задача вирішується дуже просто: розділяються звичайні користувачі і адміністратори. У повноваження звичайного користувача входить все необхідне для виконання прикладних задач, проще кажучи, для роботи, але йому заборонено виконувати дії, які змінюють саму систему. Таким чином можна уникнути псування системи в результаті помилки користувача або помилки у програмі, або навіть із злого наміру. Повноваження адміністратора зазвичай не обмежені.

Для персонального комп'ютера з яким працює декілька людей, досить важливо забезпечити кожному незалежне середовище. Це знижує вірогідність випадкового псування чужих даних, а також дозволяє кожному користувачу налаштувати зовнішній вид робочого середовища на свій смак, наприклад, зберегти розташування відкритих вікон між сеансами роботи. Ця задача очевидним чиним вирішується в багатокористувацькій моделі: організовується домашній каталог, де зберігаються дані користувача, налаштування зовнішнього вигляду і поведінки його системи і т.п., доступ інших користувачів до цього каталогу обмежується.

Якщо комп'ютер підключено до глобальної або локальної мережі, вірогідно, що якусь частину ресурсів, які зберігаються на ньому, є сенс зробити публічною і доступною по мережі. Навпаки, частину даних, скоріше всього, робити публічною на слід (наприклад, особисте листування). Обмеживши публічний доступ користувачів до персональних даних один до одного, ми і вирішимо цю задачу.

Саме завдяки гнучкості багатокористувацькій моделі розмежування доступу вона використовується сьогодні не тільки на серверах, але і на домашніх персональних комп'ютерах. В самому простому варіанті – для персонального комп'ютера, на якому працює тільки одна людина – ця модель зводиться до двох користувачів: звичайному користувачу для повсякденної роботи і адміністратору – для налаштування, оновлення, доповнення системи и виправлення помилок. Але, навіть в такому скороченому варіанті, це дає цілий ряд названих вище переваг.

1.1.2 Облікові записи

Звичайно, система може бути «знайома» з людиною тільки в переносному вигляді: в ній повинен зберігатися запис про користувача з таким іменем і про зв'язану з ним системну інформацію – обліковий запис. Англійський еквівалент терміну обліковий запис – account, «рахунок». Саме з обліковими записами, а не з самими користувачами працює система. В дійсності співвідношення облікових записів і користувачів-людей в Linux звичайно не є однозначним: декілька людей можуть використовувати один обліковий запис – система не може їх розпізнати. І в той же час в Linux є облікові запаси для системних користувачів, від імені яких працюють деякі програми які не призначені для роботи людей.

*Обліковий запис – о*б'єкт системи, за допомогою якого Linux веде облік роботи користувача. Обліковий запис вміщує дані про користувача, необхідні для реєстрації в системі і подальшій роботі з нею.

Облікові запаси можуть бути утворені під час встановлення системи або після установки. Головне для людини в обліковому записі – його назва, вхідне ім'я користувача. Саме про нього запитує система, виводячи запрошення «login:». Крім вхідного імені в обліковому записі утримується деякі відомості про користувача, необхідні для роботи з ним. Нижче наведено список цих відомостей.

*Вхідне ім'я – н*азва облікового запису користувача, яку треба вводити при реєстрації користувача в системі.

1.1.2.1 Ідентифікатор користувача

Linux зв'язує вхідне ім'я з ідентифікатором користувача в системі – UID (User ID). UID – це натуральне число, або нуль, по якому система відслідковує користувачів. Зазвичай це число вибирається автоматично при реєстрації облікового запису, але воно не може бути зовсім довільний. В Linux є деякі домовленості відносно того, яким типам користувачів можуть бути видані ідентифікатори з того або іншого діапазону. Зокрема, UID від «0» до «100» зарезервовані для псевдо користувачів.

*Ідентифікатор користувача – у*нікальне число, яке однозначно ідентифікує обліковий запис користувача в Linux. Таким числом забезпечені всі процеси Linux і всі об'єкти файлової системи. Використовується для персонального обліку дій користувача і визначення прав доступу до інших об'єктів системи.

Ідентифікатор групи.

Крім ідентифікаційного номера користувача з обліковим записом зв'язаний ідентифікатор групи. Групи користувачів застосовуються для організації доступу декількох користувачів до деяких ресурсів. У групи, також, як у користувача, є ім'я і ідентифікаційний номер – GID (Group ID). В Linux

користувач повинен належати як мінімум до одної групи – групі за замовченням. При утворенні облікового запису користувача зазвичай утворюється і група, ім'я якої співпадає з вхідним іменем, саме ця група буде використовуватися як група за замовченням для цього користувача. Користувач може входити більш ніж в одну групу, але в обліковому записі вказується тільки номер групи за замовченням.

Повне ім'я. Крім вхідного імені в обліковому записі утримується і повне ім'я (ім'я і фамілія) людини, яка використовує даний обліковий запис. Звісно, користувач може вказати що завгодно, в якості імені і фамілії. Повне ім'я необхідно не стільки системі, скільки людям – щоби мати можливість визначити, кому належить обліковий запис.

1.1.2.2 Домашній каталог

Файли всіх користувачів в Linux зберігаються роздільно, у кожного користувача є свій особистий домашній каталог, в якому він може зберігати свої дані. Доступ інших користувачів домашнього каталогу користувача може бути обмеженим. Інформація про домашній каталог обов'язково повинна бути присутня в обліковому записі, тому що саме з нього починає свою роботу користувач, який зареєструвався у системі.

1.1.2.3 Командна оболонка

Кожному користувачу треба представити спосіб взаємодії з системою: передавати їй команди і отримувати її відповіді. Для цієї цілі служить спеціальна програма – командна оболонка (або інтерпретатор командного рядка), вона повинна бути запущена для кожного користувача, який зареєструвався у системі. Оскільки Linux доступно декілька різних командних оболонок, в обліковому записі вказано, яку з командних оболонок треба запустити для даного користувача. Якщо спеціально не вказувати командну оболонку при утворенні облікового запису, вона буде призначена за замовченням, вірогідніше всього це буде bash.

Інтерпретатор командного рядка – програма, яку використовує Linux для організації діалогу людини і системи. Командний інтерпретатор має три основних частини:

- редактор і аналізатор команд у *командному рядку*;
- високорівнева системно-орієнтована мова програмування;
- засіб організації взаємодії команд між собою і системою.

1.1.2.4 Поняття «адміністратор»

В Linux є рівно один користувач, повноваження якого в системі принципово відрізняються від повноважень інших користувачів – це користувач з ідентифікатором «0». Зазвичай, обліковий запис користувача з UID=0називається root (англ. «корінь»). Користувач root – це адміністратор системи Linux, обліковий запис для root обов'язково присутній у любій системі Linux, навіть, якщо в ній немає ніяких інших облікових записів. Користувачеві з таким UID дозволено виконувати *любі* дії в системі, я це означає, люба помилка

або не вірна дія може пошкодити систему, знищити дані і привести до інших сумних наслідків. Тому, *категорично* не рекомендується реєструватися в системі під іменем root для повсякденної роботи. Працювати в root слід тільки тоді, коли це дійсно необхідно: при налаштуванні і оновлені системи, відновленні після збоїв. Саме root володіє достатніми повноваженнями для утворення нових облікових записів.

1.2 Реєстрація в системі.

Повертаємося до нашої завантаженої операційної системи Linux, яка як і раніше, очікує відповіді на своє запрошення «login:». Якщо ваша система налаштована таким чином, що це запрошення оформлено у вигляді графічного вікна в центрі екрану, натисніть комбінацію клавіш Ctrl+Alt+F1 – відбудеться переключення відео режиму і ви побачите перед собою чорний екран з приблизно наступним текстом:

Приклад 1.1 Початкове запрошення до реєстрації

```
Welcome to Some Linux / tty1 localhost login:
```

Ми переключилися у так званий текстовий режим, в якому нам недоступні можливості графічних інтерфейсів: малювання вікон довільної форми і розміру, підтримка мільйонів кольорів, малювання зображень. Всі можливості текстового режиму обмежені набором текстових і псевдографічних символів та декількома десятками базових кольорів. Одначе в Linux в текстовому режимі можна виконувати практично любі дії в системі (крім тих, потребують безпосереднього *перегляду* зображень). Текстовий режим в Linux – це повнофункціональний засіб управління системою. В різних реалізаціях Linux робота в графічному режимі може виглядати по-різному, більше того, графічний режим може бути навіть не доступним після встановлення системи без спеціального налаштування. Текстовий режим, навпаки, доступний в любій реалізації Linux і завжди виглядає практично однаково. Саме тому, всі подальші приклади і вправи, ми будемо робити в текстовому режимі, можливостей якого буде достатньо для засвоєння всього матеріалу який викладається в даному курсі.

Перший рядок в прикладі – це просто запрошення, він носить інформаційний характер. Є багато різних реалізацій Linux. І в кожній з них прийнятий свій формат першого рядка запрошення, і хоча напевно там буде вказано з якою версією Linux ви маєте справу, і, можливо, будуть присутні ще деякі параметри.

В наших прикладах мі будемо використовувати умовну реалізацію Linux – «Some Linux».

Другий рядок починається з імені хоста – саме імені системи, встановленої на даному комп’ютері. Це ім’я суттєве в тому випадку, якщо комп’ютер

підключено до локальної і глобальної мереж, якщо комп'ютер не з ким не зв'язано, це ім'я може бути любим. Зазвичай, ім'я хоста вже визначається при встановленні системи, одначе, в нашому випадку цього зроблено не було, і використовується варіант по замовченню – «localhost». Закінчується цій рядок запрошенням до реєстрації в системі – словом «login:».

Зрозуміло, що у відповідь на запрошення ми повинні ввести вхідне ім'я, для якого є відповідний обліковий запис в системі. В правильно встановленій операційній системі Linux повинна існувати як мінімум один обліковий запис для звичайного користувача. В усіх подальших прикладах у нас буде приймати участь користувач, володар облікового запису «methody» в системі «Some Linux». Ви можете користуватися для виконання прикладів любим обліковим записом, який утворений у вашій системі (звісно, крім запису root). Мефодій – користувач. Отже користувач вводить своє вхідне ім'я у відповідь на запрошення системи:

Приклад 1.2. Реєстрація в системі

```
Welcome to Some Linux / tty1 localhost login: Methody
Password: Login incorrect
login:
```

У відповідь на це система запитую пароль. Пароль користувача нам не відомо, оскільки він його нікому не говорить. Коли користувач вводив свій пароль, на екран монітора він не відображався (це зроблено, щоб пароль не можна було піддивитися), одначе користувач точно знає, що не зробив друкарської помилки. Проте, система відмовила йому у реєстрації, видавши повідомлення про помилку. («Login incorrect»). Якщо уважно на ім'я , яке введено користувачем, можна помітити, що воно починається з великої літери, той час як обліковий запис називається «metody». Linux завжди робить різницю між великими і малими літерами, тому «Metody» для нього – вже інше ім'я.

Тепер користувач повторить спробу.

Приклад 1.3. Успішна реєстрація в системі

```
login: methody
Password:
[methody@localhost methody]$
```

В цей раз реєстрація пройшла успішно, про що свідчить останній рядок прикладу – **запрошення командного рядка**. Запрошення – це підказка, яка виводиться **командною оболонкою** і свідчить проте, що система готова приймати команди користувача. Запрошення може бути оформлено по різному, більше того, користувач може сам керувати видом запрошення. Маже завжди, в запрошенні вміщуються **вхідне ім'я** та **ім'я хоста** в нашому прикладі це

«*metody*» і «*localhost*» відповідно. Закінчується запрошення частіше всього, символом «$». Це **командний рядок**, в якому будуть відображатися всі команди, які введені користувачем з клавіатури, а при натисканні на клавішу *Enter* вміст командного рядка буде передано на виконання системі.

1.2.1 Ідентифікація (authentication)

Після того, коли система виводить на екран запрошення командного рядка, і правильно введені ім'я користувача та пароль, закінчується **ідентифікація користувача** (authentication, «перевірка справжності»). Пароль може здаватися зайвою складністю, але у системи немає іншого засобу запевнитися, що за монітором знаходиться та людина, яка має право на використання даного облікового запису.

Звісно процедура ідентифікації має очевидне значення для систем, до яких мають безпосередній або мережевий доступ багато не зв'язаних один з одним користувачів. Процедура ідентифікації дає впевненість, що до такої системи не отримає доступ випадкова людина, яка не має права на використання її ресурсів і інформації, яка там зберігається. Одночасно вона дає певну гарантію безпеки від зловмисного втручання: навіть якщо нашкодити спробує користувач, який має обліковий запис, його дії будуть зареєстровані в системі, і зловмисника можна буде знайти і зупинити.

Для тих користувачів, кому процедура ідентифікації здається стомлюючою і не обов'язковою (наприклад, єдиним користувачам персональних комп'ютерів) існує можливість отримати доступ до системи, обминаючи процедуру ідентифікації. Для цієї цілі служить програма **autologin**. Вона дає доступ до роботи з графічним інтерфейсом відразу після завантаження системи, не запитуючи ім'я і пароль користувача. В дійсності autologin запускає всі програмові імені одного користувача, зареєстрованого в системі. Наприклад , ми могли би використати свій обліковий запис methody для автоматичного входу у систему. Одначе у цього підходу є свої мінуси:

- губиться можливість визначити, хто і коли робив у системі, тому що всі реальні користувачі працюють з одним обліковим записом, з точки зору системи всі вони – один і той же користувач.
- вся особиста інформація користувача стає загальною.
- пароль легко забувається (пароль все рівно є у любого користувача), тому що йог не треба вводити кожний день. При цьому autologin дає доступ тільки людині, яка сидить перед монітором і тільки для роботи з графічним інтерфейсом. Якщо ж буде потрібно, наприклад, скопіювати файли з вашого комп'ютера по мережі, пароль все рівно треба бути вводити.

Враховуючи усі мінуси, які перераховано, можна укласти, що використовувати autologin розумно тільки у тих системах, які не підключено до локальних або глобальних мереж, і до яких, при цьому, відкрито публічний доступ (наприклад, у бібліотеці).

1.2.2 Зміна пароля

Якщо обліковий запис був утворений самим користувачем, а не адміністратором багатокористувацької системи, скоріше за все було вибрано тривіальний пароль, з тим розрахунком, що користувач його змінить при першому вході у систему. Поширені тривіальні паролі «123456», «empty» і т.п. Оскільки пароль – це єдина гарантія, що вашим обліковим записом не скористується ніхто, крім вас, є сенс вибирати у якості пароля неочевидні послідовності символів. В Linux немає серйозних обмежень на довжину пароля або символи які в нього входять (зокрема, використовувати пробіл *можна*), але не має сенсу робити пароль дуже довгим – зразу росте небезпека його забути. Надійність дає його *непередбаченість*, а не довжина. Наприклад, пароль, представляючи собою ваше ім'я, або який повторює назву облікового запису, *дуже передбачений*. Найменш передбачені паролі представляють собою випадкову комбінацію великих і малих літер, цифр, знаків перетину, але їх най трудніше запам'ятати.

Користувач може у любий момент змінити свій пароль, припустимо, ми придумали більше вдалий пароль і вирішили його замінити. Він уже зареєстрований у системі, тому йому треба тільки набратив командному рядку команду **passwd** і натиснути *Enter*.

Приклад 1.4. Зміна паролю

[methody@localhost methody]$ passwd Changing password for methody.

Enter current password:

You can now choose the new password or passphrase.

A valid password should be a mix of upper and lower case letters, digits, and other characters. You can use an 8 character long password with characters from at least 3 of these 4 classes, or a 7 character long password containing characters from all the classes. An upper case letter that begins the password and a digit that ends it do not count towards the number of character classes used.

A passphrase should be of at least 3 words, 12 to 40 characters long and contain enough different characters.

Alternatively, if noone else can see your terminal now, you can pick this as your password: "spinal&state:buy".

Enter new password:

Набравши в командному рядку «passwd», ми запустили програму passwd, яка призначена саме для заміни інформації про пароль в обліковому записі користувача. Вона вивела запрошення ввести поточний пароль («Enter current password»), а потім у відповідь на вірний пароль, запропонувала підказку про грамотне складання паролю і навіть варіант надійного паролю, який ми можемо використати, якщо ніхто у даний момент не дивиться на монітор. При кожному

запуску програми passwd генерується новий випадковий пароль і пропонується користувачу. Одначе ми не використали підказку і придумали свій пароль самі.

Приклад 1.5. Зміна паролю (продовження)

Enter new password:

Weak password: not enough different characters or classes for this length. Try again.

Enter new password:

В даному випадку операція не вдалася, оскільки з точки зору passwd пароль, який ми придумали, виявився дуже простим. Наступного разу нам прийдеться ввести більше складний пароль. passwd запитує новий пароль двічі, щоби запевнитися, що в перший раз не було зроблено друкарської помилки, якщо усе в порядку, вона видає повідомлення про те, що операція зміну паролю пройшла успішно, і завершить роботу, повернувши нам запрошення командного рядка.

Приклад 1.6. Пароль змінено

Enter new password: Re-type new password:

passwd: All authentication tokens updated successfully [methody@localhost methody]$

Прискіпливість, з якою passwd відноситься к паролю користувача, не випадкова, Пароль користувача – одне із самих важливих і часто і самих слабих місць безпеки системи. Той хто відгадав наш пароль, отримає доступ до ресурсів системи рівно в тому об’ємі, в якому він надається нам, зможе читати і знищувати наші файли і т.п. Особливо це важливо у випадку пароля адміністратора, тому що його повноваження у системі значно ширше, а дії від його імені можуть пошкодити і саму систему. Звичайному користувачу при деяких обставинах також можуть бути надані повноваження адміністратора, у такому випадку не менш важливо, щоби його пароль був надійним.

Пароль користувача *root* від самого початку, назначається при встановленні системи, одначе, його може бути змінено у любий момент в подальшому, точно так як і пароль звичайного користувача.

1.3 Одночасний доступ до системи

Те що Linux – багатокористувацька і багатозадачна система, виявляється не тільки у розмежуванні прав доступу, але і в організації робочого місця. Кожний комп’ютер, на якому працює Linux, надає можливість зареєструватися і отримати доступ до системи одночасно декільком користувачам. Навіть, якщо у розпорядження усіх користувачів є тільки один монітор і одна системна

клавіатура, ця можливість є корисна: одночасне реєстрування в системі декількох користувачів дозволяє працювати по черзі без необхідності кожний раз завершувати все задачі, які розпочато, і потім їх поновлювати. Більше того, немає ніяких перепон зареєструватися у системі декілька разів під одним і тим же **вхідним іменем**. Таким чином, можна отримати доступ до одних і тих же ресурсів і організувати паралельну роботу над декількома задачами.

1.3.1 Віртуальні консолі

Характерний для Linux засіб організації паралельної роботи користувачів – **віртуальні консолі**. Припустимо, що ми хочемо зареєструватися в системі ще раз щоби мати можливість слідкувати за виконанням двох задач одночасно. Ми можемо зробити це не покидаючи текстового режиму: достатньо нажати комбінацію клавіш *Alt+F2* і на екрані з'явиться нове запрошення до реєстрації в системі.

Приклад 1.7. Друга віртуальна консоль

```
Welcome to Some Linux / tty2 localhost login: methody Password:
[methody@localhost methody]$
```

Ми ввели свій пароль і отримали запрошення командного рядка, аналогічне тому, яке ми вже бачили у попередніх прикладах. Натиснувши комбінацію *Alt+F1* ми повернемося до тільки що покинутого нами командного рядка, який виконував команду passwd для зміни паролю. Запрошення в обох випадках виглядає однаково, і це не випадково – обидва командних рядки надають зовсім еквівалентний доступ до системи, в любоу з них можна виконувати любі доступні команди.

Звернемо увагу, що в останньому прикладі перший рядок запрошення закінчується словом «*tty2*». «*tty2*» - це позначення другої **віртуальної консолі**. Можна переключатися між віртуальними консолями, так як би ми переходили з одного монітора з клавіатурою до другого, подаючи, час від часу, команди і слідкуючи за програмами, які там виконуються. За замовченням, в Linux доступно не менше 6-ти віртуальних консолей, переключатися між якими можна за допомогою сполучення клавіші *Alt* з одною з функціональних клавіш (*F1 –F6*), з кожним сполученням зв'язана відповідна за номером консоль. Віртуальні консолі позначаються «*ttyN*», де «*N*» - номер віртуальної консолі.

Віртуальна консоль

Віртуальні консолі – це декілька програм, які паралельно виконуються операційною системою, і надають користувачеві можливість зареєструватися у системі в текстовому режимі і отримати доступ до командного рядка.

У багатьох дистрибутивах Linux одна з віртуальних консолей за замовченням не може бути використана для реєстрації користувача, одначе вона не менше, якщо не більше, корисна. Якщо ми натиснемо Alt+F12, ми побачимо консоль, яка заповнена множиною повідомлень системи про події, що

відбуваються. Зокрема, там ми можемо виявити два записи про те, що в системі зареєстрований користувач «methody». На цю консоль виводяться повідомлення про всі важливі події у системі: реєстрації користувачів, виконання дій від імені адміністратора (root) , підключенні пристроїв і завантаженні драйверів до них та багато іншого.

Приклад дванадцятої віртуальної консолі показує, що віртуальні консолі - доволі гнучкий механізм, який підтримується Linux, за допомогою якого можна вирішувати різні задачі, а не тільки організацію одночасного доступу до системи. Для того, щоби на віртуальній консолі з'явилося запрошення *login:* після завантаження системи, для кожної такої консолі повинна бути запущена програма *getty*. Якщо натиснути A*LT+F10* – з великою вірогідністю ми побачимо просто чорний екран. Десята віртуальна консоль підтримується системою, одначе, чорний екран означає що для цієї консолі не запущено жодної програми, тому скористатися її існуванням не вийде. Для яких консолей буде запущена програма *getty* – визначається налаштуванням конкретної системи. В подальшому, це налаштування може бути змінено користувачем.

1.3.2 Графічні консолі

Втім, як не є широкі можливості текстового режиму, Linux ними не обмежений. Зараз детально розбирати графічний режим не будемо. Зараз важливо помітити , що якщо при завантаженні системи запрошення *login:* було представлено у вигляді графічного вікна можна повернутися до цього запрошення, нажавши комбінацію клавіш *Ctrl+Alt+F7*. Процедура реєстрації тут буде зовсім аналогічна реєстрації в текстовому режимі. З тою різницею, що після **ідентифікації** користувача (вірно введеного імені і пароля), ми побачимо не запрошення командного рядка, а графічне робоче середовище. Як вона буде виглядати – залежить від того, яку систему ми використовуємо, і як вона налаштована.

Крім того, декілька користувачів можуть робити паралельно на різних віртуальних консолях, вони можуть паралельно зареєструватися і працювати паралельно у різних графічних середовищах. Зазвичай, в стандартно налаштованій Linux – системі можна організувати не менше трьох графічних консолей, які працюють одночасно. Переключатися між ними можна за допомогою сполучення клавіш *Ctrl+Alt+F7 - Ctrl+Alt+F9.*

Щоб переключитися з графічного режиму в одну з текстових віртуальних консолей, достатньо натиснути комбінацію клавіш *Ctrl+Alt+FN,* де «*N*» - номер необхідної віртуальної консолі.

1.4 Найпростіші команди ОС Linux

Робота в Linux за допомогою командного рядка, нагадує діалог з системою: користувач вводить команди (репліки), отримуючи від системи

відповідні репліки, які мають відомості про проведені операції, додаткові запитання до користувача, повідомлення про помилки або просто мовчазне погодження виконати наступну команду.

Найпростіша команда в Linux складається з одного слова – назви програми, яку необхідно виконати. Одну таку команду (passwd) ми використовували для того, щоби змінити свій пароль. Тепер ми вирішили повернутися на одну з віртуальних консолей, на якій ми зареєструвалися і спробувати виконати декілька простих команд.

Приклад 1.8. Команда whoami

```
[methody@localhost methody]$ whoami methody
[methody@localhost methody]$
```

Назва цієї команди походить від англійського виразу «Who am I?» («Хто я?»). У відповідь на цю команду система вивела тільки одне слово: «methody» і завершила свою роботу, про що свідчить **запрошення командного рядка**, яке знову з'явилося. Програма whoami повертає назву облікового запису, від імені якого вона була виконана. Ця команда корисна в системах, в яких працює багато різних користувачів, щоби не скористатися помилково чужим обліковим записом. Одначе, в запрошенні командного рядка часто вказується ім'я користувача, тому без команди whoami можна обійтися. Наступний приклад демонструє програму, яка виведе вже більше корисної інформації: who («Кто?»).

Приклад 1.9. Команда who

```
[methody@localhost methody]$ who
methody          ttyl       Sep 23 16:31 (localhost)
methody          tty2       Sep 23 17:12 (localhost)
[methody@localhost methody]$ [methody@localhost methody]$ who am i
methody    tty2          Sep 23 17:12 (localhost)
[methody@localhost methody]$
```

who виводить список користувачів, які в даний момент зареєстровані в системі. Дана програма виводить по одному рядку на кожного зареєстрованого користувача: у першій колонці вказується ім'я користувача, в другій – «точка входу» в систему, далі іде дата і час реєстрації і ім'я хоста. Із виведеної who інформації можна укласти, що в системі двічі користувач «methody», який спочатку зареєструвався на першій віртуальній консолі (*tty1*), а приблизно через 40 хвилин – на другий (*tty2*). Звісно, ми і так це знаємо, одначе адміністратору великих систем, коли користувачі можуть зареєструватися з багатьох комп'ютерів і навіть по Мережі, програма who може бути дуже корисна.

Ще одна програма, яка видає інформацію про користувачів, які працювали в системі останнім часом – *last*. Рядки, які виводяться цією програмою,

нагадують вивід програми who, з тою різницею, що тут перераховані і ті користувачі, які вже завершили роботу.

Приклад 1.10. Команда last

```
[methody@localhost methody]$ last
methody   tty2         localhost          Tru  Sep 23  17:12     still  logged in
methody   tty1         localhost          Tru  Sep 23  16:31     still  logged in
coachman  ???          localhost          Tru  Sep 23  16:15 - 16:17  (00:01)
coachman  ???          localhost          Tru  Sep 23  16:08 - 16:08  (00:00)
cyrus     ???          localhost          Tru Sep 23  16:08 -16:08   (00:00)
reboot    system boot  2/4/26-std-up-al   Tru  Sep 23  16:03         (04:13)
```

В цьому прикладі ми знайшли, крім себе, невідомих нам користувачів *cacheman* і *cyrus* – ми дійсно точно знаємо, що не утворював облікових записів з такими іменами. Це **псевдо користувачі** (**системні користувачі**) спеціальні облікові записи, які використовуються для роботи деякими програмами. Оскільки ці «користувачі» реєструються в системі без допомоги монітору і клавіатури, їх «точка входу» в систему не визначена (в другій колонці записано «???»). В виведені програми *last* з'являється навіть користувач *reboot* (перезавантаження). В дійсності, такого облікового запису немає, програма *last* таким засобом виводить інформацію про те, коли була завантажена система.

1.5 Вихід з системи

В рядках, які виведені програмою *last*, вказано не тільки момент реєстрації користувач в системі, але і момент завершення роботи. Можна уявляти Linux як закрите приміщення: щоби почати працювати треба спочатку *увійти* у систему, а після того як робота закінчена , треба *вийти* з системи. В тому випадку, якщо в систему увійшло декілька користувачів, кожен з них повинен вийти, завершивши роботу, при чому зовсім не має значення, різні це користувачі чи копії одного і того ж.

Вхід користувача в систему означає, що треба приймати і виконувати його команди, і повертати йому звіти про виконані дії, наприклад, надавши йому інтерфейс командної оболонки. Вихід означає. Що робота від імені даного користувача завершена і більш не слід приймати від нього команди. Весь процес взаємодії користувача з системою з моменту реєстрації до виходу називається **сеансом роботи**. Причому, якщо користувач входить в систему декілька разів під одним і тим же іменем, йому будуть доступні декілька *різних* сеансів роботи, не пов'язаних між собою.

В наших прикладах ми реєструвалися в системі двічі: на першій і другій віртуальних консолях. Щоби завершити роботу на любій з них, нам достатньо у відповідному командному рядку набрати команду *logout*.

Приклад 1.11. Команда logout

[methody@localhost methody]$ logout Welcome to Some Linux / tty1 localhost login:

У відповідь на цю команду замість чергового запрошення командного рядка поновлюється запрошення до реєстрації в системі. На даній віртуальній консолі з користувачем завершена, і тепер тут знову може зареєструватися любий користувач.

Є ще інший, більше не багатослівний засіб повідомити системі, що користувач бажає завершити поточний сеанс роботи. Натиснувши *Alt+F2* ми попадаємо на другу віртуальну консоль, де все ще відкрито сеанс для користувача «methody» і натиснемо сполучення клавіш *Ctrl+D*, щоби припинити цей сеанс. Комбінація *Ctrl+D* приведе не до передачі чергового символу, а до закриття поточного **вхідного потоку даних**. Грубо кажучи, командна оболонка вводить команди користувача з консолі, так якби вона читала їх по рядкам із файлу. Натиск *Ctrl+D* дає сигнал їй про те. Що цей «файл» закінчився, і тепер оболонці немає звідки зчитувати команди. Такий засіб завершення повністю аналогічний явному завершенню командної оболонки командою *logout*.

2 ТЕРМІНАЛ І КОМАНДНИЙ РЯДОК ОС LINUX

2.1 Термінал

Як стало відомо з попередньої лекції, основний засіб спілкування з Linux – системна клавіатура та екран монітора, який працює у **текстовому режимі**. Текст, який вводиться користувачем, миттєво відображається на моніторі відповідними літерами, одначе, може і не відображатися, як у випадку вводу пароля. Для *управління* вводом використовуються деякі *нетекстові* клавіші на клавіатурі. Натискання на ці клавіші не приводить до відображення символу, замість цього текст, який вводиться, обробляється системою тим або іншим засобом.

Приклад 2.1. Повідомлення про помилку

```
[methody@localhost methody]$ data
-bash: data: command not found
[methody@localhost methody]$ date
Вск Сен 12 13:59:36 MSD 2004
```

Спочатку ми помилилися, і замість команди *date* написали *data*. У відповідь ми отримали **повідомлення про помилку**, оскільки такої команди система не розуміє. Якщо набрати *data* і не натискати *Enter*, можна виправити становище, натиснувши *Backspace*, знищити останнє «*a*» і замість нього поставити «*e*», перетворивши *data* на *date*. Така команда в системі є, і на екрані виникне поточна дата.

Діалог машини з користувачем не просто так виглядає як обмін текстами. Саме письмову мову використовують люди для постановки і опису рішення задач в наперед визначеному, **формалізованому вигляді**. Тому і задача управління системою може повністю бути представлена і вирішена у вигляді формалізованого тексту – програми. При цьому машині відводиться роль акуратного виконувача цієї програми, а людині – роль автора. Крім того, людина аналізує текст, який отримала від системи: запитану ним інформацію і так звані повідомлення, текст, який описує стан системи в процесі рішення задачі.

Текстовий принцип роботи з машиною дозволяє відволіктися від конкретних частин комп'ютера, на шталт системної клавіатури і відео карти з монітором, розглядаючи як єдиний кінцевий пристрій, за допомогою якого користувач бачить текст і передає його системі, а система виводить користувачу необхідні йому дані і повідомлення. Такий пристрій називається **терміналом**. Взагалі термінал – це *точка входу* користувача в систему, яка має властивість передавати текстову інформацію. Терміналом може бути окремий зовнішній пристрій, який підключається до комп'ютера послідовної передачі даних. В ролі терміналу може працювати програма (наприклад, *xterm* або *ssh*). Нарешті,

віртуальні консолі Linux – теж термінали, тільки організовані програмно за допомогою, пристроїв сучасного комп'ютера які підходять.

Термінал

Пристрій послідовного вводу і виводу символьної інформації, здатний сприймати частину символів як управляючі для редагування вводу, посилання сигналів і т.п. Використовується для взаємодії користувача і системи.

Для прийому і передавання тексту достатньо вміти приймати і передавати символи, з яких цей текст складається. Більше того, бажано, щоби одиницею обміну з комп'ютером був саме байт (один acsii символ). Тоді, кожна буква, яка набрана на клавіатурі, може бути передана системі для обробки, якщо це знадобиться. З іншого боку, типовий спосіб управління системою в Linux – робота в командному рядку – вимагає порядкового режиму роботи, коли набраний текст передається комп'ютеру тільки після натискання на клавішу *Enter* (що відповідає символу кінця рядка). Розмір такого рідка у байтах передбачити, звісно, не можна тому термінал, який працює в порядк**о**вому режимі нічим, по суті, не відрізняється від терміналу, який працює в по символьному режимі, за виключення того, що активність системи по обробці даних, які приходять з цього терміналу падає в декілька разів (обмін ведеться не байтами а цілими рядками).

Властивість терміналу передавати тільки символьну інформацію призводить до того, що деякі символи, які передаються, повинні сприйматися не як текстові, я як **управляючі** (наприклад, символи, які передаються клавішами *Backspace* і *Enter*). Насправді управляючих символів більше: частина з них призначена для екстреного передавання команд системі, частина – для редагування тексту, який вводиться. Багато з цих символів не мають *спеціальної* клавіші на клавіатурі, тому їх необхідно вилучати за допомогою **клавіатурного модифікатора** *Ctrl*.

Увага

Команди. Які подаються з клавіатури за допомогою Ctrl, як і символи, які передаються при цьому системі, прийнято позначати знаком «^» , після якого слідує ім'я клавіші, яка натискається разом з Ctrl: наприклад, одночасне натискання Ctrl і «a» позначається «^A».

Так, для завершення роботи програми cat, яка зчитує порядково дані з клавіатури і виводить їх на термінал, можна скористатися командною «^C» або «^D»:

Приклад 2.2. Як завершити роботу cat

```
[methody@localhost methody]$ cat
Any Text
Any Text
^C
[methody@localhost methody]$ cat
```

```
Any Text agaim^ [ [Dn
Any Text again
^D
[methody@localhost methody]$
```

Один рядок виду «Any Text …» ми вводимо з клавіатури (що відображається на екрані), і після того, як ми натискаємо Enter, він миттєво виводиться програмою cat (що теж відображається на екрані). З кожним наступним рядком програма поступила би аналогічно, але в прикладі ми обидва рази завершили роботу програми, в першому випадку натиснувши «*^C*», а в другому – «*^D*». Ефект команди виявився однаковим, а працюють вони по різному: «*^C*» посилає програмі, яка зчитує з клавіатури, **сигнал** аварійного завершення роботи, «*^D*» повідомляє їй , що вивід даних з клавіатури закінчено, можна продовжувати роботу далі (оскільки програма *cat* більше нічого не робить, вона завершується самостійно природнім шляхом). Можна вважати, що «*^C*» - це скорочення від «***C****ancel*», а «*^D*» - від «***D****one*».

В приклад не попало як набираючи перший *cat*, ми знову помилилися і написали *ccat* замість *cat*. Щоби виправити положення, ми скористувалися клавішами із стрілочками: За допомогою клавіші «*клавіша вліво*» підвели курсор до одного «c» і натиснули «*Backspace*», а потім «*Enter*». В режимі вводу команди нам це вдалося, а при передаванні даних програмі *cat* «*клавіші вліво*» не по перемістила курсор, а передала цілу послідовність символів: «*^[*», «*[*» і «*D*». Справа в тому, що на клавіатурі терміналу може бути так багато нетекстових клавіш, що на віх їх обмеженої кількості *різних* управляючих символів. Тому більшість нетекстових клавіш повертають так звані **управляючі послідовності**, які починаються управляючим символом (як правило *Escape*, тобто «*^[*»), за розміщується строго визначене число звичайних символів (для *клавіші вліво* – «*[*» і «*D*»).

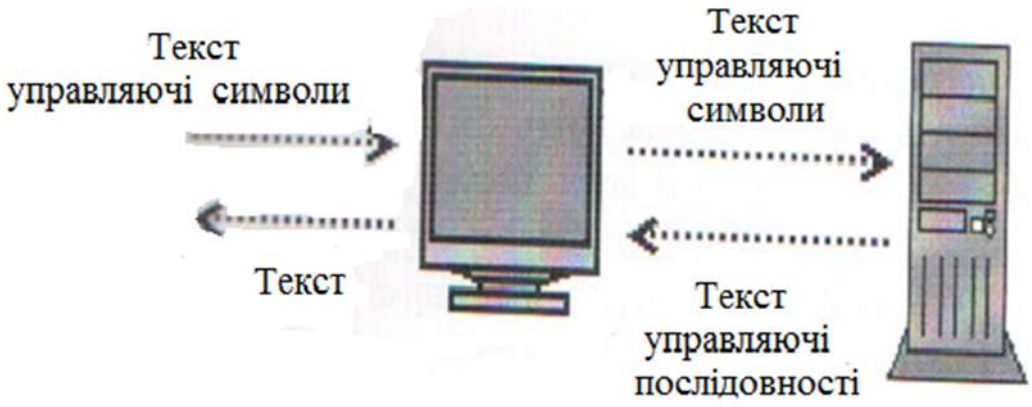

Рис. 2.1 Інтерфейс командного рядка. Взаємодія користувача з комп’ютером за допомогою терміналу

Теж саме можна сказати і про *вивід* управляючих послідовностей на термінал. Сучасний термінал вміє достатньо багато крім простого виводу тексту: переміщувати курсор по всьому екрану (щоби вивести текст там), знищувати і вставляти рядки на екрані, використовувати колір і тому подібне. Всім цим керують **управляючі послідовності**, які при виводі на екран не відображаються як текст, а виконуються раніше заданим засобом. В деяких випадках управляючі послідовності, які повертаються клавішами, співпадають з тими, що управляють поведінкою терміналу. Тому, ми не побачили «*Any Text agaim^ [[D*» у видачі *cat* : «*[[D*» при виводі на термінал переміщує курсор на одну позицію вліво, так щоби було виведено «*Any Text agaim*», потім курсор став прямо над «*m*» і поверх нього було виведено «*n*». Якби термінал мав би замість дисплею друкарський пристрій. І цьому місці виявилося би дещо, яке складалося би з накреслень «*m*» і «*n*».

Вимоги до терміналу як до точки входу користувача в систему виявляються вельми невисокими. Формально кажучи, термінал повинен задовольняти трьом обов'язковим вимогам і одному необов'язковому. Термінал повинен вміти:

– передавати текстові дані від користувача системі;

– передавати від користувача системі небагато чисельні управляючі команди;

– передавати текстові дані від системи користувачу;

– (необов'язково) інтерпретувати деякі дані, які передаються від системи користувачу, як управляючі послідовності і відповідно обробляти їх.

Обмеження на інтерфейс на пряму не позначається на ефективності роботи користувача у системі. Одначе, чим менше вимог до інтерфейсу, тим важніше розумно його організовувати. Люба взаємодія може бути описана з трьох точок зору: по-перше, яку задачу вирішує користувач (що він бажає від системи), по-друге, як він формулює задачу у доступному до розуміння системі вигляді; і третє, якими засобами він користується для взаємодії з системою. Зокрема, текстовий інтерфейс зручно розглядати з точки зору поданої ним *мови* спілкування з машиною: по-перше, описом цієї мови задається діапазон задач, які вирішуються з її допомогою; по-друге, слова цієї комп'ютерної мови (які називаються в програмуванні операторами) являють собою засіб рішення користувацьких задач (у вигляді невеликих програм сценаріїв). Команди, які допомагають швидко і ефективно обмінюватися з машиною пропозиціями на цій мові, і будуть третьою складовою інтерфейсу командного рядка.

2.2 Командний рядок

Основне середовище взаємодії з Linux – **командний рядок**. Суть його в тому. Що кожний рядок, який передається користувачем системі, - це **команда**,

яку та повинна виконати. Поки не натиснуте *Enter*, рядок можна редагувати, потім він відсилається системі.

Приклад 2.3. Команди echo і cal

```
[methody@localhost methody]S cal
Сентября 2DD4
Вс Пн Вт Ср Чт Пт Сб
   1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20  21 22 23 24 25
 26 27 28 29 30
[methody@localhost methody]S echo Hello, world!
Hello, world!
```

Команда *cal* виводить календар на поточний місяць, а команда *echo* просто виводить на термінал все. Що було слід в командному рядку *після неї*. Виходить, що одну і ту ж команду можна використовувати з різними **параметрами** (або *аргументами*), причому, ці параметри змінюють поведінку команди. Тут ми забажали подивитися календар за березень 2005-го року, для чого і передали команді *cal* для параметри *3* і *2005*:

Приклад 2.4. Команда cal з параметрами

```
[methody@localhost methody]$ cal 3 2005
Марта 2005
Вс Пн Вт Ср Чт Пт Сб
      1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

У більшості випадків при розборі командного рядка *перше* слово вважається іменем команди, а інше – її параметрами.

2.3 Підсистема допомоги

Настав момент узнати, чи є якась оперативна допомога в Linux, щоб використати його *головні* команди для подальшого проходження курсу вивчення системи. За частотністю вживання при вивченні системи є дві **підсистеми допомоги**: *man* і *info*.

Працювати з Linux, не заглядаючи в документацію, практично не можливо. Усім наполегливо рекомендується, приступаючи до роботи, а тим більше - до вивчення Linux, користуватися усіма доступними керівництвами. Всі **утиліти**, всі **демони** Linux, всі функції **ядра** і **бібліотек**, структура більшості **конфігураційних файлів**, на кінець багато умоглядних, але важливих понять системи описані або в керівництвах, або в *info*-сторінках, або, на худий кінець, в несистематизованій документації, яка супроводжує систему. Тому, від користувача системи не вимагається завчити всі можливі варіанти взаємодії з нею. Достатньо *розуміти* основні принципи її будово і вміти знаходити довідкову інформацію.

2.3.1 Сторінки керівництва (man)

Більше всього, різної корисної інформації вміщується в **сторінках керівництва (manpages)**. Кожна сторінка керівництва присвячена якому-небудь об'єкту системи. Для того, щоби продивитися сторінку керівництва, треба дати команду системі *man об'єкт*:

Приклад 2.5. Перегляд сторінки допомоги

```
[methody@localhost methody]$ man cal
CAL(1)          BSD General Commands Manual          CAL(1)
NAME
cal - displays a calendar
SYNOPSIS
cal [-smjy13] [[month] year]
DESCRIPTION
```

Cal displays a simple calendar. If arguments are not specified, the current month is displayed. The options are as follows:

Сторінка керівництва займає , як правило, *більше однієї сторінки екрану*, Для того, щоби читати було зручніше, *man*, запускає програму перегляду текстів – *less*. Управляти програмою less просто: сторінки перегортаються пробілом, а коли читати набридне, треба натиснути «*q*» (*Quit*). Перегортати сторінки можна і клавішами *Page Up*/ *Page Down*, для зсуву на один рядок у перед можна застосувати *Enter* або стрілку вниз, а на один рядок назад – стрілку вверх. Перехід на начало і кінець тексту виконується по командах «*g*» і «*G*» відповідно (*Go*). Повний список того, що можна робити з текстом в less, виводиться по команді «*H*» (*Help*).

Сторінка керівництва складається з полів – стандартних розділів, з різних сторін описуючих об'єкт, який нас зацікавив, команду *cal*. В полі *NAME* вміщується короткий опис об'єкту (таке, щоби його призначення було зрозуміло з першого погляду). В полі *SYNOPSIS* дається *формалізований* опис способів використання об'єкту (в даному випадку – того , як і з якими параметрами запускати команду *cal*). Як правило, в квадратні дужки в цьому полі укладені

необов'язкові параметри команди, які можна їй передати, а можна і опустити. Наприклад. рядок «[[*month*] *year*]» означає , що в цьому місці командного рядку у команди може не бути зовсім, може бути вказано рік або – пара місяць і рік. На кінець, текст у полі *DESCRIPTION* - це розгорнутий опис об'єкту, достатній для того, щоби ним скористуватися. Одне з самих важливих полів керівництва знаходиться у кінці тексту. Якщо у процесі читання *NAME* або *DESCRIPTION* користувач розуміє, що не знайшов в керівництві того, що шукав, він може захтіти подивитися, чи є *інші* керівництва або інші джерела інформації *по тій же самій темі*. Список таких джерел вміщується в полі *SEE ALSO:*

Приклад 2.6. Поле SEE ALSO допомоги

```
[methody@localhost methody]$ man man
SEE ALSO
. . .
apropos(l), whatis(1), less(1), groff(1), man.conf(5).
. . .
```

До цього поля ми добралися за допомогою вже відомої команди «*G*». В полі *SEE ALSO* винайшлися посилання на керівництва по *less, goff* (програми форматування сторінки керівництва), структурі конфігураційного файлу для man та іншим командам, які є супутні. Треба пам'ятати, що не знаючи дію команди, користуватися нею заборонено. Тому далі, ми скористалися командою, яка не викликає небажаних реакцій системи.

Приклад 2.7. Виклик whatis

```
[methody@localhost methody]$ whatis apropos
apropos      (1) - search the whatis database for strings
[methody@localhost methody]$ man apropos
apropos(l)                                          apropos(l)
NAME
apropos - search the whatis database for strings
```

Команда *whatis* не робить нічого руйнівного. Як і команда *apropos, whatis* шукає під рядок в деякій базі даних, яка складається з полів *NAME* всіх сторінок допомоги у системі. Різниця між ними в тому, що *whatis* шукає тільки серед імен об'єктів, (в лівих частинах полів *NAME*), а *apropos* – по всій базі. В результаті у *whatis* отримується список коротких описів об'єктів з іменами, які вміщують в собі слово яке ми шукаємо, а у *apropos* – список, в якому це слово *згадується*. Для того, щоби це впізнати, все рівно, прийшлося би один раз прочитати документацію.

В системі може зустрітися декілька об'єктів *різного типу*, але з однаковою назвою. Часто співпадають, наприклад, імена **системних викликів** (функцій

ядра) і програм, які дозволяють користуватися цими функціями із командного рядка (так звані **утиліти**).

Приклад 2.8. Допомога с однаковими іменами [methody@localhost methody]$ whatis passwd

passwd (1) - update a user's authentication tokens(s)
passwd (5) - password file
passwd (8) - manual page for passwd wrapper version 1.0.5

Опис об'єктів, які виводяться *whatis*, відрізняється число в дужках – номером **розділу**. В системі керівництва Linux дев'ять розділів, кожний з яких вмішує сторінки керівництва до об'єктів визначеного типу. Всі розділи вміщують по одному керівництву з іменем «*intro*», якому в загальному вигляді на прикладах розказано, що за об'єкти мають відношення до даного розділу.

Приклад 2.9. Допомога intro

george@localhost:~> whatis intro

intro	(1)	-Introduction to	user commands
intro	(2)	-Introduction to	system calls
intro	(3)	-Introduction to	library functions
intro	(4)	-Introduction to	special files
intro	(5)	-Introduction to	file formats
intro	(6)	-Introduction to	games
intro	(7)	-Introduction to	conventions and miscellany section
intro	(8)	- Introduction to	administration and privileged commands
intro	(9)	-Introduction to	kernel interface

Ось назви розділів в перекладі на українську мову:

1. Команди користувача.
2. Системні виклики (користувацькі функції ядра Linux, керівництва розраховані на програміста, який знає мову «*C*»).
3. Бібліотечні функції (функції, які належать різноманітним бібліотекам підпрограм, керівництва розраховані на програміста, який знає мову «*C*»).
4. Зовнішні пристрої і робота з ними (в Linux вони називаються **спеціальними файлами**.)
5. Формати різних стандартних файлів системи (наприклад, **конфігураційних**).
6. Ігри, дрібнички, та інші речі, які не мають системної цінності.
7. Теоретичні положення, домовленості і все, що не може бути класифіковано.
8. Інструменти адміністратора (часто не доступні загальному користувачеві).

9. Інтерфейс ядра (*внутрішні* функції і структури даних ядра Linux, які необхідні тільки системному програмісту, який виправляє або доповнює ядро).

Зокрема, приклад з *passwd* показує, що в системі «Some Linux», яку ми описуємо є програма *passwd* (саме з її допомогою ми міняли пароль на попередній лекції), файл *passwd*, утримує інформацію про користувачів і адміністраторська програма *passwd*, яка володіє більше широкими можливостями. По замовченню *man* розглядає усі розділи і показує *перше* керівництво, яке знайдено з *заданим іменем*. Щоби подивитися керівництво по об'єкту з визначеного розділу, необхідно в якості першого параметру команди *man* указати номер розділу:

Приклад 2.10. Вибір серед сторінок допомоги з однаковим іменем

```
[methody@localhost methody]$ man 8 passwd
PASSWD(8) System Administration Utilities       PASSWD(8)
. . .
[methody@localhost methody]$ man -a passwd
PASSWD(1)           Some Linux                  PASSWD(1)
. . .
PASSWD(8) System Administration Utilities       PASSWD(8)
. . .
PASSWD(5)    Linux Programmer's Manual          PASSWD(5)
```

Якщо, в якості першого параметру man використати «-a», будуть послідовно виведені *всі* керівництва з заданим іменем. *В середині* сторінок керівництва прийнято безпосередньо після імені об'єкту ставити у круглих дужках номер розділу, в якому вміщується керівництво по цьому об'єкту: *man (1)*, *less (1)*, *passwd (5)* і т.п. (На практиці спробувати: *man -a passwd*, *man -a less*).

2.3.2 Info

Інше джерело інформації про Linux і складаючи його програми – довідкова підсистема *info*. Сторінка керівництва, не дивлячись на велику кількість посилань різного типу, залишається «лінійним» текстом структурованим тільки логічно. Документ *info* структурований перш за все *топологічно*, це – справжній гіпертекст, в якому безліч невеликих сторінок об'єднаних у дерево. В кожному розділі документу *info* завжди є зміст, з якого можна перейти відразу до потрібного підрозділу. Звідки можна повернутися назад. Крім того, info-документ можна читати і як *безперервний* текст, тому у кожному розділі є послання на попередній і наступний підрозділи.

Приклад 2.11. Перегляд info-документа

```
[methody@localhost methody]$ info info
File: info.info, Node: Top, Next: Getting Started, Up: (dir)
```

Info: An Introduction

. . .

* Menu:

* Getting Started:: Getting started using an Info reader.

* Expert Info:: Info commands for experts.

* Creating an Info File:: How to make your own Info file.

Index:: An index of topics, commands, and variables.

. . .

--zz-Info: (info.info.bz2)Top, строк: 24 --All----------

Welcome to Info version 4.6. Type ? for help, m for menu item.

Програма *info* використовує весь екран: на більшій частині його вона показує текст документу, а перша і дві останніх сторінки відведені для орієнтації в його структурі.

Одна або декілька сторінок, які можна перегортати клавішею *Пробіл* або *Page Up/ Page Down* – це **вузол** (**node**). Вузол вміщує звичайний текст і **меню** (**menu**) – список посилань на інші вузли, які лежать на дереві на більш низькому рівні. Посилання в середині документу мають вигляд «*** ім'я_вузла::**» і переміщувати по ним курсор можна клавішею *Tab*, а переходити до перегляду вибраного вузла – клавішею *Enter*. Повернутися до попереднього переглянутого вузла можна клавішею «*l*» (від «*Last*»). І, головне, вийти з програми info можна, натиснувши «*q*» (*Quit*). Більш детальну інформацію про управління програмою info можна в любий момент получити у самої info, натиснувши «*?*» (*info ?*).

Вузли, які складають документ *info*, можна переглядати і підряд, один за іншим, (за допомогою «*n*», *Next* і «*p*», *Previous*), одначе це буває необхідно не часто. У верхньому рядку екрану info показує ім'я поточного вузла, ім'я наступного вузла і ім'я батьківського (або верхнього вузла), в якому знаходиться посилання на поточний. Показані нам імя вузла (*Top*) і ім'я верхнього вузла (*dir*) означають, що переглядається *корінний* вузол документу., вище якого – тільки каталог із списком всіх *info*-дерев. У нижній частині екрану розташований рядок з інформацією про поточний вузол, а за ним – рядок для вводу довгих команд (наприклад, для пошуку тексту за допомогою команди «/»).

Команді, *info* можна вказувати у параметрах весь *ланцюжок* вузлів, який приводить до того або іншого розділу документації, одначе, це буває довільно рідко:

Приклад 2.12. Перегляд визначеного вузла info-документа

[methody@localhost methody]$ info info "Getting Started" Help-Q

File: info.info, Node: Help-Q, Prev: Help-Int, Up: Getting Started

Quitting Info

. . .

Абсолютно вірно було заключити в лапки ім'я вузла «*Getting Started*» - в цьому випадку *info* шукав вузол за адресою: «info - > *Getting Started* - > *Help-Q*».

Якби команда мала вигляд: *info info Getting Started Help-Q*, то адреса вийшла би не правильною: «*info - > Getting - > Started - > Help-Q*».

2.3.3 RTFT

Документація в Linux грає важливішу роль. Рішення *любої* задачі повинно починатися з вивчення керівництв. Не варто жаліти на це час. Навіть якщо рядом є досвідчений користувач Linux, який, можливо, *знає* відповідь, не варто його турбувати *відразу же*. Цілком можливо, що навіть знаючи, *що* треба робити, він не пам'ятає*, як саме* – і тому (а також тому, що він досвідчений користувач) почне вивчати керівництва. Це – закон, у якого є навіть особиста назва: **RTFM**, що означає «Read That Fine Manual».

RTFM

Правило, згідно якого, рішення любої задачі треба починати з вивчення документації.

Керівництво – це зовсім не підручник, це – довідник. В ньому вміщується інформація, *достатня* для засвоєння об'єкту, який описується, але ніяких навчаючих засобів, ніяких визначень, повторень і виділення головного в ньому зазвичай немає. Тим більше не допускається *урізання* керівництва з метою надати не великі по об'єму, але найбільш важливі відомості. Так прийнято у підручниках, причому головні відомості розкриваються і пояснюються дуже детально, а інші присутні у вигляді посилань на документацію для професіоналів. Сторінки керівництв – і є саме документація для професіоналів. Керівництво найчастіше за все читає людина, яка вже знає про що воно.

Це не означає, що з керівництва не можна зрозуміти як, наприклад, користуватися командою у найпростіших випадках. Навпаки, часто зустрічається поле *EXAMPLES*, яке як раз і вміщує приклади використання команди урізних умовах. Одначе, все це переслідує ціль *не навчити*, а розкрити зміст, пояснити сказане в інших цілях.

Система *info* може вміщувати більше, ніж *man*, тому в неї часто розміщують і підручники (прийнято називати підручник терміном «tutorial»), т.з. «howto» (приклади постановки і рішення типових задач), і навіть, статті по темі. Таким чином, *info* документ може стати , на відміну від сторінки керівництва, повним *склепінням* відомостей. Розробка такого документу – справа трудомістка, тому, далеко не всі об'єкти системи ним супроводжуються. Крім того, і прочитати великий *info* документ часто не можливо. Тому, є сенс починати саме з керівництва, а якщо його не достатньо – вивчати *info*. Буває, що деякий об'єкт системи не має документації ні у форматі *man*, ні у форматі *info*. В цьому випадку можна мати надію, що при ньому є *супроводжувальна документація*. Вона немає ні стандартного формату, ні тим більше, посилань на керівництва з інших об'єктів системи. Така документація (рівно як і приклади використання об'єкту) зазвичай розміщуються в каталозі /usr/share/doc/і'мя_об'єкту.

Документація у більшості випадків пишеться на простій англійській мові. Традиція писати на англійській іде від немалого вкладу Америки у розвиток коп'ютерної справи. Постійний бурхливий розвиток Linux призводить до непосильної невиконаної праці переклад документації на інші мови Світу.

2.4 Ключі

Працюючи у системі і вивчаючи керівництва ми помітили, що параметри команд можна віднести до двох різних категорій. Деякі параметри мають *особистий сенс*, це імена файлів, назви розділів і об'єктів в *man* і *info*, числа і інші параметри особистого сенсу не мають, їх значення можна витлумачити, лише знаючи, до якої команди вони відносяться. Наприклад, параметр «-a» можна передавати не тільки команді man, але і команді who, і команді last, причому, значити для них він буде різне. Такого роду параметри називаються *модифікаторами виконання* або **ключами** (options).

Приклад 2.13. Команда date з ключем

```
[methody@localhost methody]$ date
Вск Сен 19 23:01:17 MSD 2004
[methody@localhost methody]$ date -u
Вск Сен 19 19:01:19 UTC 2004
```

Для рішення різних задач одні і ти ж дії необхідно виконувати злегка по різному. Наприклад, для синхронізації робіт у різних точках земної кулі краще використовувати єдиний час для всіх (за Гринвічем), а для організації особистого робочого дня – місцевий час (з урахуванням зсуву за часовим поясом і різницею зимового і літнього часу). І той, і інший час показує команда *date*, тільки для роботи за Гринвічем їй треба додатковий ключ-параметр «*u*» (він же «*--universal*»).

2.4.1 Одно літерні ключі

Для формату ключів немає жорсткого стандарту, одначе, існують домовленості, порушувати які у наш час не пристойно.

По-перше, якщо параметр починається на «-», це – **одно літерний ключ**. За «-», як правило іде один символ, частіше за все – одна літера, яка позначає дії або властивість, яку цей ключ передає команді. Так простіше відрізняти ключі від інших параметрів – і користувачеві при наборі командного рядка і програмісту, автору команди.

По-друге, бажано, щоби значення ключа було *значущим* – як правило, це перша літера назви нашої дії або властивості. Наприклад, ключ «*-a*» в *man* і *who* походить від слова «*All*» (все), і змінює роботу цих команд так, що вони починають показувати інформацію, про яку вони зазвичай замовчують. А в командах *cal* і *who* зміст ключа «*m*» різний:

Приклад 2.14. Використання ключа «-m» у різних командах

```
[methody@localhost methody]$ who -m
methody tty1         Sep 20 13:56 (localhost)
[methody@localhost methody]$ cal -m
Сентября 2018
Пн Вт Ср Чт Пт Сб Вс
  1  2  3  4  5
 6  7  8  9  10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

-«число» для cal вказує, який місяць треба вивести. Для *who* означає «*Me*», тобто «*Я*», і результаті *who* працює схоже на *whoami*. А для *cal* ключ «*-m*» - це команда видавати календар, враховуючи першим днем тижня понеділок.

Властивість ключа бути, з одного боку, гранично коротким, а з іншого – інформативним, називається ***аббревіальністю***. Не тільки ключі, але і імена найбільше поширених команд Linux володіють цією властивістю.

По-третє, іноді ключ змінює поведінку команди таким чином, що змінюється і тлумачення параметру, який іде в командному рядку за цим ключем. Виглядає це так, ніби ключ сам отримує параметр, тому ключі такого виду називаються ***параметричними***. Як правило. Їх параметри - імена файлів різного призначення, числові характеристики і інші значення, які треба передавати команді.

Приклад 2.15. Використання info -o

```
[methody@localhost methody]$ info info "Expert info" Cross-refs Help-Cross
 -o text
info: Запись ноды (info.info.bz2)Help-Cross...
info: Завершено.
[methody@localhost methody]$ cat text -n
1.      File: info.info, Node: Help-Cross, Up: Cross-refs
2.
3.      The node reached by the cross reference in Info
4.      ------------------------------------------------------------
. . .
```

Тут info запустилася в не якості інтерактивної програми, а як обробник info-документу. Результат роботи – текст, вузла *info → Expert info → Cross-refs →Help-Cross*, програма розмістила у файл *text*. А програма cat вивела вміст цього файлу на термінал, пронумерувавши усі рядки (за проханням ключа «*-n*», «*number*»).

Учетверте, є декілька меш жорстких, але популярних домовленостей про *значення* ключів. Ключ «*-h*» («*Help*») зазвичай примушує команди видавати коротку довідку (дещо схоже на *SYNOPSIS*, іноді з короткими поясненнями). Якщо вказати «-» замість імені *вихідного* файлу у відповідному параметричному ключі (не рідко, це ключ «-o»), виведення буде проводитися на термінал. На кінець, буває необхідність передати команді *параметр, а не ключ*, який починається з «-». Для цього треба використовувати ключ «--».

Приклад 2.16. Параметр – не ключ, який починається на «-»

methody@localhost methody]$ info -o -filename-with-

info: Запись ноды (dir)Top...

info: Завершено.

[methody@localhost methody]$ head -1 -filename-with- head: invalid option -- f

Попробуйте 'head --help' для получения более подробного описания.

[methody@localhost methody]$ head -1 -- -filename-with-

File: dir Node: Top This is the top of the INFO tree

Тут ми спочатку утворили файл *-filename-with-*, а потім, спробували продивитися його перші рядки (команда *head –кількість_рядків ім'я, файлу* виводить першу кількість_рядків з вказаного файлу). Ключ «--» (перший «-» - ознака ключа, другий – сам ключ) зазвичай забороняє команді інтерпретувати всі наступні параметри командного рядка як ключі, не залежно від того, чи починаються вони на «-» чи ні. Тільки після «--» head згодилася з тим, що -filename-with- - це ім'я файлу.

2.4.2 Повноцінні ключі.

Аббревіативність ключів важко дотримати, коли їх у команди *дуже* багато. Деякі букви латинського алфавіту використовуються дуже часто, і могли би служити скороченнями відразу декільком командам, а деякі – рідко, під них буває важко придумати осмислену назву. На такий випадок існує інший, **повноцінний** формат: ключ починається на два «-», за якими іде повне ім'я суті, яка ним позначається.

Приклад 2.17. Ключ –help

[methody@localhost methody]$ head --help

Использование: head [КЛЮЧ]... [ФАЙЛ]...

Print the first 10 lines of each FILE to standard output.

With more than one FILE, precede each with a header giving the file name.

With no FILE, or when FILE is -, read standard input.

Аргументы, обязательные для длинных ключей, обязательны и для коротких.

-c, --bytes=[-]N print the first N bytes of each file;
with the leading '-', print all but the last
N bytes of each file
-n --lines=[-]N print the first N lines instead of the first 10;
with the leading '-', print all but the last
N lines of each file
-q --quiet, --silent не печатать заголовки с именами файлов
-v --verbose всегда печатать заголовки с именами файлов
--help показать эту справку и выйти
--version показать информацию о версии и выйти
N may have a multiplier suffix: b 512, k 1024, m 1024*1024.

Об ошибках сообщайте по адресу <bug-coreutils@gnu.org>.

Ми зробили те, що прохала нас утиліта head. Видно, що деякі ключі head мають і одно літерні, і повноцінні формати, а деякі – тільки повноцінні. Так зазвичай і буває: ключі. Які використовуються часто, мають абревіатуру, а рідкі – ні. Значення параметричних повноцінних ключів прийнято передавати не наступним параметром командного рядку, а за допомогою конструкції «=значення» безпосередньо після ключа.

2.5 Інтерпретатор командного рядка (shell)

В Linux немає окремого об’єкту під назвою «система». Головний з системних компонентів – користувач. Він керує машиною. А та виконує команди. В керівництвах другого і третього розділів описані системні виклики (функції ядра і бібліотек). Вони і є безпосередні команди системи. Правда користуватися ними можна тільки написавши програму (частіше всього , на мові **C**).

Нерідко програма – доволі складна. Справа в тому, що функції ядра реалізують низько рівневі операції, і для рішення навіть самої простої задачі користувача необхідно виконати декілька таких операцій, перетворюючи результат роботи однієї для потреб іншої. Виникає необхідність видумати для користувача іншу – більш високо рівневу і більш зручну у використанні – мову управління системою. Всі команди. Які ми використовували роботі, були частиною саме цієї мови.

З цього не важко заключити, що обробляти ці команди, перетворювати їх у послідовність системних і бібліотечних викликів повинна теж якась спеціальна програма, і саме, з нею безперервно веде діалог користувач, відразу після входу у систему. Так воно і виявилося: програма ця називається *інтерпретатором командного рядку або командна оболонка («shell»).*

«Оболонкою» вона названа, якраз тому, що все управління системою іде як би «з середини» неї: користувач спілкується з нею на зручній для нього мові, а вона спілкується з частинами системи на зручній для них мові.

Звичайно. Командних інтерпретаторів Linux декілька. самий простий з них, який з'явився у ранніх версіях *UINIX*, називався *sh*, або «*Bourne Shell*» - по імені автора, Стівена Борна. З часом, цю оболонку, всюди, де тільки можливо, замінили на більше потужний *bash*, «*Bourn Again Shell*». *bash* перевершує *sh* у всьому, особливо у можливостях *редагування* командного рядка. Крім цих оболонок, у системі може бути встановлено «*The Z Shell*», *zsh*, *самий* потужний, на сьогодення, командний інтерпретатор (22 тисячі рядків документації), або *tcsh*, оновлена і теж дуже потужна версія оболонки «C Shell», синтаксис команд якої схожий на мову програмування «**C**».

Ім'я оболонки, яка запускається для користувача відразу після входу у систему – *стартовий командний інтерпретатор* (*login shell*), - це частина користувацького облікового запису, яку користувач може змінити командою *chsh* (**ch**ange **sh**ell). Яка би задача, зв'язана з управлінням системою не стала перед користувачем Linux, вона повинна мати рішення в термінах командного інтерпретатора. Фактично, рішення користувацької задачі – це опис її на мові *shell*. Мова спілкування користувача і командного інтерпретатора – це високорівнева мова програмування, доповнена, з одного боку, засобами організації взаємодії команд і системи, а з іншого, засобами взаємодії з користувачем, полегшуючими і прискорюючи роботу з командним рядком.

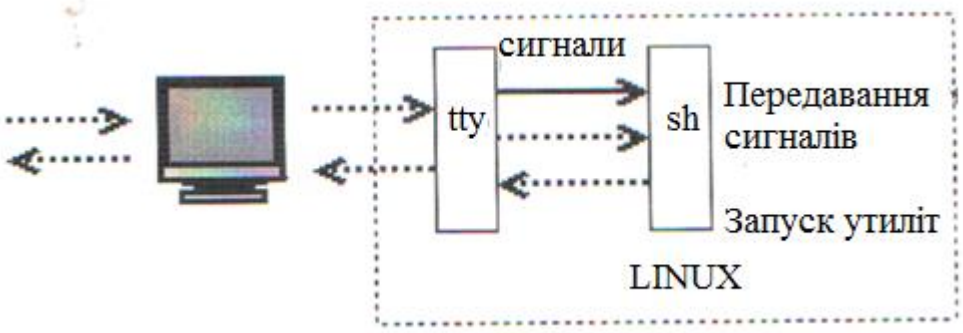

Рис. 2.2 Інтерфейс командного рядку. Взаємодія користувача з комп'ютером за допомогою терміналу і оболонки

2.5.1 Команди і утиліти

Приклад 2.18. Команда без змісту

```
[methody@localhost methody]$ apropos s
. . . (четыре с половиной тысячи строк!)
```

Одного невдалого запуску apropos нам було достатньо для того,щоби зрозуміти: команд в Linux дуже багато. Ніяка програма – хай навіть і оболонка – не може *самостійно* розбиратися у всіх задокументованих командах. Для цього є **утиліти**, тобто, корисні програми. Командний інтерпретатор не зобов'язаний вміти *виконувати* усе, що вводить користувач. Йому достатньо розібрати командний рядок, виділити з нього команду і параметри, а потім запустити утиліту – програму, ім'я якої співпадає з іменем команди.

В дійсності, *особистих* команд у командному інтерпретаторі не багато. В основному, це – оператори мови програмування і інші засоби управління самим інтерпретатором. Усі знайомі нам команди, навіть echo, існують в Linuxy вигляді окремих утиліт. shell займається тільки тим, що підготовлює набір параметрів у командному рядку (наприклад, розкриває **шаблони**), запускає програми і обробляє результати їх роботи.

Приклад 2.19. Визначення типу команди

```
[methody@localhost methody]$ type info
info is /usr/bin/info
[methody@localhost methody]$ type echo
echo is a shell builtin
[methody@localhost methody]$ type -a echo
echo is a shell builtin
echo is /bin/echo
[methody@localhost methody]$ type -a -t echo
builtin
file
[methody@localhost methody]$ type -a -t date
file
[methody@localhost methody]$ type -at cat
File
```

В *bash* типи команд можна визначити за допомогою команди *type*. Особисті команди *bash* називаються *builtin* (вбудовані команди), а для утиліт виводиться шлях, який вміщує назву каталогу, в якому лежить файл з відповідною програмою, і ім'я цієї програми. Деякі самі необхідні команди вбудовані у bash, навіть не дивлячись на те, що вони є у вигляді утиліт (наприклад. *echo*). Працює вбудована команда також, але так як часу на її виконання їде суттєво менше, командний інтерпретатор вибере саме її, якщо буде така можливість. Ключ «*-a*» («*all*», звісно) примушує *type* вивести *усі* можливі варіанти інтерпретації команди, а ключ «*-t*» - вивести тип команди замість шляху.

Ми об'єднали ключі у групу, написавши «*-at*» замість «*-a -t*». Багато утиліт дозволяють так робити, зменшуючи довжину командного рядку. Якщо зустрічається параметричний ключ, він повинен бути у групі останнім, а його

значення – іти за ним, як і вимагається. Групувати можна тільки одно літерні ключі.

2.5.2 Слова і роздільники

При розборі командного рядка shell використовує поняття **роздільник** (*delimiter*). Роздільник – це символ, який розділяє слова, таким чином. Командний рядок – це послідовність *слів* (які мають значення) і *роздільників* (які не мають значення). Для shell роздільниками є: символ пробілу, символ табуляції, і символ переводу рядка. Кількість роздільників між двома сусідніми словами значення не мають.

Перше слово у тройці передається по команді як перший параметр, друге – як другий і т.д. Для того, щоби роздільник попав у *середину* слова (і рядок, який вийшов з роздільником, передався як *один* параметр), весь необхідний під рядок треба оточити одинарними або подвійними лапками.

Приклад 2.20. Лапки в командному рядку

```
[methody@localhost methody]$ echo One   Two   Three
One Two Three
[methody@localhost methody]$ echo One   "Two Three"
One Two     Three
[methody@localhost methody]$ echo 'One
>
> Ой. И что дальше?
> А, кавычки забыл!'
> One
> Ой. И что дальше?
> А, кавычки забыл!
[methody@localhost methody]$
```

У першому випадку команді *echo* було передано *три* параметри «*One*», «*Two*» і «*Three*». Вона їх вивела розділяючи пробілом. У другому випадку параметрів було *два*: «*One*» і «*Two Three*». У результаті ці *два* параметри були також виведені через пробіл. В третьому випадку параметр всього *один* – від відкриваючого апострофа «*'One*» до закриваючого «*забув!'*». Весь час вводу *bash* послужливо подавав підказку «>» на знак того, що набір командного рядка продовжується, але в режимі вводу вмісту лапок.

3 ОРГАНІЗАЦІЯ ФАЙЛОВОЇ СИСТЕМИ

3.1 Понятия файла каталога

Файл – це поняття звичне любому користувачу комп'ютера. Для користувача кожний файл – це окремий предмет, у якого є початок і кінець і який відрізняється від усіх інших файлів іменем і розташуванням. Як і любий предмет, файл можна утворити, перемістити і знищити, одначе, без зовнішнього втручання він буде зберігатися невизначено довгий час. Файл призначений для зберігання даних любого типу – текстових, графічних, звукових, програм, які виконуються та багато іншого. Аналогія файлу з предметом дозволяє користувачеві швидко освоїтися при роботі з даними в операційній системі.

Для операційної системи Linux файл – не менше важливе поняття, чим для її користувача: дані, які зберігаються на любих носіях, обов'язково знаходяться в середині якого-небудь файлу, в протилежному випадку. Вони просто не доступні ні для операційної системи ні для користувача. Більше того, всі пристрої, які підключені до комп'ютера, Linux уявляє як файли (**файли - дирки**). Звісно. Файл, який вміщує звичайні дані, сильно відрізняється від файлу призначеного для звернення до пристрою, тому в Linux визначені декілька різних типів файлів. В загалі, користувач має справу з файлами трьох типів: **звичайні файли**, призначені для зберігання даних, **каталоги** і **файли** – **послання**. Саме про них піде мова в даній лекції.

Файл – окрема область даних на одному з носіїв інформації, у якої є особисте ***ім'я***.

3.1.1 Система файлів: каталоги

Файлова система з точки зору користувача – це «простір», в якому розміщуються файли. Наявність файлової системи дозволяє не тільки визначити «як називається файл», але і «де він знаходиться». Розпізнавати файли за ім'ям, було би дуже не ефективно: про кожний файл приходилося би пам'ятати як він називається і притому, щоби імена ніде не повторювалися. Більше того, необхідно механізм. Який дозволяє працювати з групами тематично пов'язаних між собою файлів. Інакше кажучи, файли треба *систематизувати*.

Файлова система – засіб зберігання і організації доступу до даних на інформаційному носії або його розділі. Класична файлова система має ієрархічну структуру. В якій файл однозначно визначається *повним шляхом* до нього.

Linux може працювати з різними типами файлових систем, які різняться списком можливостей підтримки продуктивності у різних ситуаціях, надійністю і іншими ознаками. Далі ми розглянемо описи можливостей файлової системи *Ext2/Ext3*, на сьогодні de facto стандартної файлової системи для Linux.

Більшість сучасних файлових систем використовують у якості основного організаційного принципу *каталоги*. Каталог – це список посилань на файли і інші каталоги. Прийнято говорити, що каталог *вміщує* в собі файли і інші

каталоги, хоча в дійсності, він тільки *посилається* на них, фізичне розміщення даних на диску зазвичай ніяк не пов'язано з розміщенням каталогу. Каталог, на який є посилання в даному каталозі, називається ***підкаталогом*** або ***вкладеним каталогом***. Каталог у файловій системі більше всього нагадує бібліотечний каталог, який вміщує посилання на об'єднані за якимись ознаками книги і інші розділи каталогу. Посилання на один і той же файл може бути у декількох каталогах одночасно, це може зробити доступ до файлу більше зручним. У файловій системі ***Ext2*** кожний каталог - це окремий файл особливого типу («*d*», від англійського «*directory*»), який відрізняється від ***звичайного файлу*** з даними: в ньому можуть зберігатися тільки послання на інші файли та каталоги.

Увага!

У файловій системі Linux немає *папок* і *документів*. Є ***каталоги*** *і* ***файли***, можливості яких значно ширше.

Досить часто замість терміну ***каталог*** можна зустріти ***пака*** (англійською ***folder***). Цей термін добре вписується в уяву про файли як предмети, які можна розкладувати по папках, одначе, частина можливостей файлової системи, яка суперечить цій уяві, таким чином затемнюється. А саме, з терміном «папка» погано узгоджується те, що посилання на файл може бути присутнім одночасно у декількох каталогах, файл може бути посланням на інший файл і так далі. В Linux ці можливості файлової системи суттєво важливі для ефективної роботи, тому будемо всюди використовувати термін «*каталог*», який найбільше підходить.

У файловій системі, яка організована за допомогою каталогів, на любий файл повинно бути послання, як мінімум, з одного каталогу, у іншому випадку файл не буде доступний в середині цієї файлової системи, інакше кажучи, не буде існувати.

3.1.2 Імена файлів і каталогів

3.1.2.1 Допустимі імена

Головною ознакою, яка відрізняє файли і каталоги – їх імена. В Linux імена файлів і каталогів можуть бути довжиною не більше 256 символів, і можуть вміщувати любі символи, крім «/». Причина такого обмеження очевидна: цей символ використовується як роздільник імен у складі шляху, тому не повинен зустрічатися у самих іменах. Причому, Linux завжди відрізняє прописні і рядкові букви у іменах файлів і каталогів, тому «*method*», «*Method*» і «*METHODY*» будуть трьома *різними* іменами.

Є декілька допустимих символів, в іменах файлів і каталогів, які при цьому, треба використовувати з обережністю. Це – так звані спец символи «*****», «/», «**&**», «<», «>», «**;**», «**(**», «**)**», «|»,а також *пробіли* і *табуляції*. Справа в тому, що ці символи мають *особливе* значення для любої **командної оболонки**, тому треба спеціально подбати про те, щоби командна оболонка сприймала ці символи як частину імені файлу або каталогу. Про спеціальне значення символу «-» для

команд Linux вже йшла мова, коли обговорювалося, як змінити його інтерпретацію.

3.1.2.2 Кодування і імена кирилицею

Як можливо було помітити, поки в усіх іменах файлів і каталогів, які зустрічалися у нашій роботі, використовувалися тільки символи латинського алфавіту і деякі знаки перетину. Це – не випадково, і викликано бажанням забезпечити, щоби приклади, які наводилися, зовсім однаково виглядали на любих системах. В Linux в іменах файлів і каталогів допустимо використовувати любі символи любої мови, одначе, така свобода потребує жертв.

Справа в тому, що з давніх часів кожний символ (літера) кожної мови традиційно представлялася у вигляді *одного* байта. Така уява накладає дуже жорстке обмеження на *кількість* букв у алфавіті: їх може бути не більше 256, а за вирахуванням управляючих символів, цифр, знаків перетину і іншого – і того менше. Великі алфавіти (наприклад, ієрогліфічні японський і китайський) прийшлося би замінювати спрощеною їх уявою. На додаток, перші 128 символів з 256, краще за все залишати незмінними, які відповідають стандарту ASCII, який вміщує латиницю, цифри, знаки перетину, і найбільше популярні символи з тих, що зустрічаються на клавіатурі друкарської машинки. Інтерпретація 128 символів, які залишилися, залежить від того, яке ***кодування*** встановлено в системі. Наприклад, у російському кодуванні KOI8-R 228-й символ такої таблиці відповідає літері «*Д*», а у західноєвропейському кодуванні ISO-8859-1 цей символ відповідає «ä».

Імена файлів, які *записані* на диску одному кодуванні, виглядають безглуздо, якщо при *перегляді* каталогу було встановлено інше кодування. Гірше за те, багато кодувань заповнюють діапазон символів з номерами від 128 до 255 *не повністю*, тому, відповідного символу може взагалі не бути. Це означає, що ввести таке спотворене ім’я файлу з клавіатури (наприклад, для того щоби його перейменувати) напряму не вдасться, прийдеться пускатися на різні хитрощі. На кінець, багато мов, історично мають *декілька* кодувань. На жаль, у наш час, не має стандартного засобу вказати кодування прямо у імені файлу, тому у рамках однієї файлової системи *варто* дотримуватися єдиного кодування при наданні імені файлам.

Існує універсальне кодування, яке вміщує символи усіх письменностей світу – *UNICODE*. Стандарт *UNICODE* в наш час отримує все більше поширення і претендує на статус загального для усіх текстів, які зберігаються у електронному вигляді. Одначе, поки він де досягнув бажаної універсальності, особливо в області імен файлів. *Один* символ *UNICODE* може займати більше одного байту – і в цьому є головний його недолік, так як більшість корисних прикладних програм, які відмінно працюють з *одно байтовими* кодуваннями, необхідно ґрунтовно переробляти для того,щоби навчити їх користуватися *UNICODE*. Можливо. Причина недостатнього поширення цього кодування також і в тому, що *UNICODE* – дуже громіздкий стандарт, і він може виявитися

неефективним при роботі з файловою системою, де швидкість і надійність обробки – дуже суттєві якості.

Це означає, що називаючи файли, не слід використовувати мови, які різняться з англійською. Поки точно відомо, в якому кодуванні задано ім'я файлу – проблем не виникне. Одначе, ми зрозуміли, що гарантії у передаванні названого українською файлу на будь-яку іншу систему можна добитися тільки передаючи разом з ним налаштування кодування, навіть два: у своїй системі і системі адресата. Інший більше легкий засіб, передавати файл, використовуючи у його назві тільки символи *ASCII*.

3.1.2.3 Розширення

Багатьом користувачам відомо поняття ***розширення*** – частина імені файлу після крапки, зазвичай, обмежується кількома символами і вказує на тип даних, які вміщуються у файл. У файловій системі Linux немає таких приписів з приводу розширення: у імені файлу може бути люба кількість точок, а після останньої крапки може бути люба кількість символів. Хоча розширення не обов'язкові і не нав'язуються технологією у Linux, вони широко використовуються: розширення дозволяють людині або програмі, не відкриваючи файлу, тільки за його ім'ям визначити, якого типу дані утримуються у ньому. Одначе, треба враховувати, що розширення – це тільки набір угод з найменування файлів різних типів. Строго кажучи. Дані у файлі можуть не відповідати розширенню, яке заявлено, з тої або іншої причини, тому, повністю покладатися на розширення просто не можна.

Визначити тип вмісту файлу можна і на базі самих даних. Багато форматів передбачають вказівку на початку файлу, як треба інтерпретувати подальшу інформацію: як програму, вхідні дані для текстового редактора. Сторінку *html*, звуковий файл, зображення або щось інше. В розпорядженні користувача Linux завжди є утиліта **file** , яка призначена саме для визначення типу даних, які вміщує файл.

Приклад 3.1 Визначення типу даних в файлі

```
[methody@localhost methody]$ file -- -filename-with-
-filename-with-: ASCII English text
[methody@localhost methody]$ file /home/methody
/home/methody: directory
```

Ми забажали перевірити, що вміщує файл «*-filename with-*», який ми утворили на попередній лекції і забули, яким чином і з якою метою його зробили. Ми повинні знати, що перед тим, як використовувати незнайомий файл, треба з'ясувати, тип даних, які вміщує файл. Не виключно, що це – двійковий файл програми, яка призначена для виконання. В такому файлі можуть зустрічатися послідовності, які випадково співпадають з ***управляючими послідовностями*** терміналу. Поведінка терміналу, після чого, може стати не передбачуваною, а не досвідчений користувач навряд чи зможе з цим впоратися. Ми отримали достатньо точну відповідь від утиліти *file*: у нашому фалі – англійський текст в

кодуванні *ASCII*. *file* вміє розпізнавати дуже велику кількість типів даних і майже напевно видасть вірну інформацію. Ця утиліта ніколи не «*довіряє*» розширенню файлу і аналізує самі дані. *file* розпізнає тип даних, тому, повідомить: звичайний файл, каталог, або інший тип.

3.1.3 Дерево каталогів

Поняття каталогу дозволяє *систематизувати* усі об'єкти, які розміщені на носіях даних. У більшості сучасних файлових систем використовується ієрархічна модель організації даних: існує один каталог, який об'єднує усі дані у файловій системі – це «*корінь*» усієї файлової системи, ***корінний каталог***. Корінний каталог може вміщувати любі об'єкти файлової системи, і зокрема, підкаталоги (другого рівня вкладеності) Ті, у свою чергу, можуть теж утримувати любі об'єкти файлової системи. Таким чином, усе, що записане на диску – файли, каталоги і спеціальні файли – обов'язково «*належать*» корінному каталогу: або безпосередньо, або на певному рівні вкладеності.

Ієрархію вкладених один в один каталогів можна співвіднести з ієрархією даних у системі: об'єднати тематично зв'язані файли у каталог, тематично зв'язані каталоги у один загальний каталог і т.п. Якщо строго слідувати ієрархічному принципу, то чим глибше буде ***рівень вкладеності*** каталогу, тим більше приватною ознакою повинні бути об'єднані дані, які вміщуються у ньому. Якщо цього принципу не дотримуватися, то незабаром виявиться значно простіше складувати *усі* файли в один каталог і шукати потрібний серед них, ніж пробляти такий пошук по всім підкаталогам системи. Одначе, в такому випадку, про яку би то ні було *систематизацію* файлів годі і говорити.

Структуру файлової системи можна представити наглядно у вигляді дерева, «*коренем*» якого є корінний каталог, а у верхівках розташовані всі останні каталоги.

У любій файловій системі Linux завжди є тільки один ***корінний каталог,*** який називається «/». Користувач Linux завжди працює з єдиним деревом каталогів, навіть якщо різні дані розташовані на різних носіях: декількох жорстких або мережевих дисках, CD-ROM і т.п.

Для того, щоби підключати і відключати файлові системи на різних пристроях в одне загальне дерево, використовуються процедури ***монтування і розмонтування,*** про які ми поговоримо пізніше.

Після того, як файлові системи на різних носіях підключені до загального дерева, дані, які вміщуються на них доступні так, як би вони складали єдину файлову систему: може навіть і не знати на якому пристрої які файли зберігаються.

Положення любого каталогу в дереві каталогів точно і однозначно описується за допомогою ***повного шляху***. Повний шлях завжди починається з корінного каталогу і складається з перерахування усіх верхівок, які зустрічаються при русі по ребрам дерева до каталогу, який шукається, включно.

Назви сусідніх верхівок розділяються символом «/» («***слеш***»). В Linux повний шлях, наприклад, до каталогу «*methody*» у файловій системі, записується у наступному вигляді: спочатку символ «/», який позначає корінний каталог, потім до нього додається «home», потім роздільник «/», за яким іде назва каталогу, який шукається. «*methody*». В результаті отримується повний шлях *«/**home/methody***».

Розташування файлу у файловій системі аналогічним чином визначається за допомогою повного шляху, тільки останнім елементом у даному випадку буде не назва каталогу, а назва файлу. Наприклад, повни шлях до утвореного нами файлу «-filename with-» буде виглядати так: *«/**home/methody/-filename with-**»*.

Організація файлової системи у вигляді дерева не допускає появлення циклів: тобто, каталог не може вміщувати в себе каталог, в якому він утримується сам. Завдяки цьому обмеженню повний шлях до любого каталогу або файлу у файловій системі завжди буде *кінченим*.

3.2 Розміщення компонентів системи. Стандарт FHS

Спробуємо більш детально розібратися як налаштовано дерево каталогів Linux і де, і що в ньому можна знайти. Давайте обслідуємо свою файлову систему: починаючи з ***корінного каталогу***. Інструкція порадила нам використовувати для цього команду ***ls каталог***, де «*каталог*» - це повний шлях до каталогу, який нас цікавить. У даному випадку утиліта ***ls*** виведе список всього, що в цьому каталозі є.

Приклад 3.2 Стандартні каталоги в /

```
[methody@localhost methody]$ ls /
bin  dev  home  mnt  root  tmp  var
boot etc  lib   proc sbin  usr
[methody@localhost methody]$
```

Утиліта ***ls*** вивела список підкаталогів корінного каталогу. Цей список буде майже таким або точно таким у любому дистрибутиві Linux. В корінному каталозі Linux- системи зазвичай знаходяться тільки підкаталоги із *стандартними іменами*. Більше того, не тільки імена а і *типи даних*, які можуть попасти у той або інший каталог, теж регламентовані цим стандартом. Цей стандарт називається ***Filesystem Hierarchy Standard*** («*стандартна структура файлових систем*»).

Коротко опишемо, що знаходиться у кожному з підкаталогів корінного каталогу. Ми не будемо наводити повні списки файлів для кожного каталогу, який описується, їх можна продивитися за допомогою команди ***ls ім'я каталогу***.

/bin

Назва цього каталогу походить від слова «***binaries***» («*двійкові*», «*ті що виконуються*»). В цьому каталозі знаходяться файли самих необхідний утиліт, які виконуються. Сюди попадають такі програми. Які можуть знадобитися

системному адміністратору або іншим користувачам для усунення неполадок у системі або відновлення після збоїв.

/boot

«***boot***» - завантаження системи. В цьому каталозі знаходяться файли, необхідні для самого першого етапу завантаження: завантаження **ядра** і, за звичай, само ядро. Користувачу практично ніколи не знадобиться безпосередньо працювати з цими файлами.

/dev

В цьому каталозі знаходяться усі ***файли-дирки***, які є у системі. Це – файли особливого типу, які призначені для звернення до різних системних ресурсів і пристроїв.(англійське «***devices***» - «*пристрій*», звідси і скорочена назва каталогу). Наприклад, файли ***/dev/ttyN*** відповідають віртуальним консолям, де ***N*** – номер віртуальної консолі. Дані, які введені користувачем на першій віртуальній консолі, система зчитує з файлу ***/dev/tty1***, у цей же файл записуються дані, які треба вивести користувачу на цю консоль. У *файлах-дирках*, у дійсності, не зберігаються ніякі дані, за їх допомогою дані *передаються*.

/etc

Каталог для системних ***конфігураційних файлів***. Тут зберігається спеціальна інформація про особисті налаштування даної системи: інформація про зареєстрованих користувачів, доступні ресурси, налаштування різних програм.

/home

Тут розташовані каталоги, які належать користувачам системи – домашні каталоги, звідси і назва «***home***». Виділення усіх файлів, які утворюються користувачами, від інших системних файлів дає очевидні переваги: серйозне пошкодження системи або необхідність оновлення не зачепить найбільше цінної інформації користувацьких файлів. Також треба відмітити, що шкода, яку можуть нанести користувачі системі зведена до мінімуму.

/lib

Назва цього каталогу – скорочення від «***libraries***» (від англійської «*бібліотеки*»). ***Бібліотеки*** – це збірник найбільше стандартних функцій, необхідних багатьом програмам: операції вводу/виводу, малювання елементів графічного інтерфейсу та інше. Щоби не включати ці функції у текст кожної програми, використовуються стандартні функції бібліотек – це значно економить місце на диску у спрощує написання програм. В цьому каталозі вміщуються бібліотеки, необхідні для роботи найбільше важливих системних утиліт. (розташованих у ***/bin/sbin***).

/mnt

Каталог для монтування (від англійської «***mount***») – тимчасового підключення файлових систем, наприклад, на з’ємних носіях (CD-ROM та інші).

/proc

В цьому каталозі усі файли «***віртуальні***» - вони розташовуються не на диску, а в оперативній пам'яті. У цих файлах утримується інформація про програми (***процеси***), які виконуються у даний момент у системі.

/root

Домашній каталог адміністратора системи – користувача ***root***. Сенс розміщувати його окремо від домашніх каталогів інших користувачів складається у тому, що ***/home*** може розташовуватися на окремому пристрої, яке не завжди доступно (наприклад, на мережевому диску), а домашній каталог ***root*** повинен бути присутнім у любій ситуації. Крім того, це підвищує безпеку системи і умови її відновлення.

/sbin

Каталог для найважливіших системних утиліт (назва каталогу – скорочення від «***system binaries***»): на додаток до утиліт ***/bin*** тут знаходяться програми, які необхідні для завантаження, резервного копіювання, відновлення системи. Повноваження на використання цих програм є тільки у системного адміністратора.

/tmp

Цей каталог призначений для ***тимчасових файлів***: в таких файлах програми зберігають проміжні дані , необхідні для роботи. Після завершення роботи програми, тимчасові файли втрачають сенс і повинні бути знищені. Зазвичай, каталог ***/tmp*** очищується при кожному завантаженні системи.

/usr

Каталог ***/usr*** – це «*держава у державі*». Тут можна знайти такі ж підкаталоги ***bin***, ***etc***, ***lib***, ***sbin***, як і у корінному каталозі. Одначе, у корінний каталог попадають тільки утиліти, *необхідні* для завантаження і поновлення системи в аварійній ситуації, усі *останні* програми і дані розташовуються у підкаталогах ***/usr***. Прикладних програм у сучасних системах, зазвичай, встановлено дуже багато, тому цей розділ файлової системи може бути дуже великим.

/var

Назва цього каталогу – скорочення від «***variable***» («*змінні*» дані). Тут розміщуються ті дані, які утворюються у процесі роботи з різними програмами і призначені для передавання іншим програмам і системам (черга друку і електронної пошти та інше) або для відома системного адміністратора (***системні журнали***, які вміщують протоколи роботи системи). На відміну від каталогу ***/tmp*** сюди попадають ті дані, які можуть знадобитися після того, як програма, яка їх утворила, завершила роботу.

Стандарт ***FHS*** регламентує не тільки каталоги, які було перераховано, але і їх підкаталоги, а іноді навіть приводить список конкретних файлів, які повинні бути присутні у визначених каталогах. Цей стандарт послідовно дотримується в усіх Linus-системах, хоча і не без гарячих суперечок між розробниками при виході кожної нової версії.

Стандартне розташування файлів дозволяє людині, навіть програмі передбачити, де знаходиться той чи інший компонент системи. Для людини це означає. Що вона зможе швидко зорієнтуватися у любій системі Linux (де файлова система організована у відповідності до стандарту) і знайти те, що йому треба. Для програм стандартне розташування файлів – це можливість організації автоматичної взаємодії між різними компонентами системи.

Ми вже встигли скористатися деякими перевагами, які дає використання стандартного розташування файлів: на попередніх лекціях ми запускали утиліти, не вказуючи повний шлях до файлу. Який виконується, наприклад, ***cat*** замість ***/bin/cat***. ***Командна оболонка*** «знає», що файли, які виконуються, розташовуються у каталогах ***/bin***, ***/usr/bin*** і т.п.

Саме у цих каталогах вона шукає файл ***cat***, який виконується. Завдяки цьому, кожна заново встановлена у системі програма негайно виявляється доступною користувачеві з командного рядка. Для цього не треба ні перезавантажувати систему ні запускати ніяких процедур – достатньо просто розмістити файл. Який виконується, у один з відповідних каталогів.

Рекомендації стандарту по розташуванню файлів і каталогів базуються на принципі розносити у різні підкаталоги файли, які по різному використовуються у системі.

По ***типу використання у системі***, *файли можна розділити* на наступні групи.

Користувацькі/системні файли

Користувацькі файли – це усі файли, які утворені користувачем і не належать жодному з компонентів системи. Системні файли – файли, які складають систему. Про користь розмежування користувацьких і системних файлів мова йшла вже вище.

Файли, які змінюються/ незмінні файли

До незмінних файлів відносяться усі статичні компоненти програмного забезпечення: бібліотеки, файли, які виконуються, і інше – усе, що не змінюється само без втручання системного адміністратора. Змінні файли – це ті, які змінюються без втручання людини у процесі роботи системи: ***системні журнали***, черги друку і інше.

Виділення незмінних файлів в окрему структуру (наприклад ***/usr***), дозволяє використовувати відповідну частину файлової системи у режимі «*тільки читання*», що зменшує вірогідність випадкового пошкодження даних і дозволяє використовувати для збереження цієї частини файлової системи на CD-ROM і інших носіях, які доступні тільки для читання.

Файли, які розділяються/ нероздільні файли

Це розмежування стає корисним, якщо мова іде про мережу, у якій працює декілька комп’ютерів. Значна частина інформації при цьому може зберігатися на одному з комп’ютерів і використовуватися усіма іншими по мережі (до такої інформації належать, наприклад, багато програм і домашні каталоги

користувачів). Одначе, частину файлів не можна розділяти між системами (наприклад, файли для початкового завантаження системи).

4 РОБОТА З ФАЙЛОВОЮ СИСТЕМОЮ

4.1 Поточний каталог

Файлова система не тільки систематизує дані, а є основою метафори «робочого місця» в Linux. Кожна програма, яка виконується, «працює» у строго визначеному каталозі файлової системи. Такий каталог називається ***поточним каталогом***, можна уявляти, що програма під час роботи «знаходиться» у цьому каталозі, це її «робоче місце». В залежності від поточного каталогу може змінюватися поведінка програми: часто програма буде робити, по замовченню, з файлами, які розташовані саме у поточному каталозі – до них вона «дотягнеться» у першу чергу. Поточний каталог є у любої програми, у тому числі і у командної оболонки (***shell***) користувача. Оскільки взаємодія користувача з системою обов'язково опосередкована командною оболонкою, можна казати проте, що користувач «знаходиться» у тому каталозі, який у даний момент є *поточним каталогом його командної оболонки*.

Усі команди, які даються користувачем за допомогою **shell**, успадковують поточний каталог ***shell***, тобто «працюють» у тому ж каталозі. З цієї причини користувачу важливо знати поточний каталог shell. Для цього служить утиліта **pwd.**

Приклад 4.1 Поточний каталог: pwd

```
[methody@localhost methody]$  pwd /
home/methody
[methody@localhost methody]$
```

pwd (абревіатура від **p**rint **w**orking **d**irectory) повертає повний шлях поточного каталогу командної оболонки, природно, саме тієї командної оболонки, за допомогою якої була виконана команда ***pwd***. В даному випадку ми дізналися, що на цей момент (на цій ***віртуальні консолі***) поточним каталогом є каталог «***/home/methody***».

Майже усі утиліти, з якими ми працювали на попередніх лекціях, по замовченню читають, і створюють файли у поточному каталозі. Так ми зверталися до файлів, не використовуючи ніяких шляхів, просто за іменем. Наприклад, використовували утиліту ***cat***, щоби вивести на екран вміст файлу «***text***».

Приклад 4.2 Повний та відносний шлях до файлу

```
[methody@localhost methody]$  cat text
File: info.info,    Node: Help-Cross,    Up: Cross-refs
The node reached by the cross reference in Info . . .
[methody@localhost methody]$ cat /home/methody/text
File: info.info,      Node: Help-Cross,      Up: Cross-refs
The node reached by the cross reference in Info . . .
```

В дійсності, командна оболонка, перш ніж передати параметр «***text***» (ім'я файлу) утиліті ***cat***, підставляє значення поточного каталогу – отримує повний шлях до цього файлу у файловій системі: «***/home/methody/text***». Вміст саме цього файлу утиліта ***cat***, виведе на екран. Набираючи тільки ім'я файлу без шляху до поточного каталогу. Ми скористалися ***відносним шляхом*** до цього файлу.

Відносний шлях– шл**ях** до об'єкту ***файлової системи***, який не починається ***в корінному каталозі***. Для кожного ***процесу*** Linux визначено ***поточний каталог***, з якого система починає відносний шлях при виконанні файлових операцій.

Відносний шлях будується точно також, як і повний – перерахуванням через «/» усіх назв каталогів, які зустрічаються при русі до каталогу або файлу, який ми шукаємо. Між повним шляхом і відносним є тільки одна суттєва різниця: *відносний шлях* починається *з поточного каталогу*, в той час як *повний шлях* завжди починається *з корінного каталогу*. Відносний шлях любого файлу або каталогу у файловій системі може мати любу конфігурацію: щоби добратися до файлу, який шукається, можна рухатися як в напрямку до корінного каталогу, так і від нього. Linux відрізняє повний і відносний шляхи дуже просто: якщо ім'я об'єкту починається на «/» - це повний шлях, у любому іншому випадку – відносний.

Користувач може звертатися до файлу за допомогою повного або відносного шляху – результат буде абсолютно той же. Так команди ***cat text*** *і* ***cat /home/methody/text,*** які дали ми, мали однаковий результат, оскільки виводився *один і той же файл*. Якщо у відносному шляху зустрічаються символи «/», розглядаються *підкаталоги* поточного каталогу, їх підкаталоги і так далі. Коротше кажучи, ***відносний шлях*** будується за тими же правилами, що і ***повний***, з тією різницею, що відносний шлях починається *не* з символу «/». Сам ***поточний каталог***, який би не був повний шлях до нього, завжди має ще одне позначення «**.**», яке можна використовувати, якщо з якихось причин *вимагається*, щоби навіть у *відносному* шляху до файлу, який знаходиться у поточному каталозі, був присутній елемент «*ім'я каталогу*». Так, шляхи «***text» і «./text»*** також приводять до одного і того ж файлу, одначе, в першому випадку. У рядку шляху не вміщувалося ні чого, крім імені файлу. Відокремити *шлях* до файлу від його *імені* можна за допомогою команд ***dirmane*** *і* ***basename*** відповідно.

Приклад 4.3 Використання dirname та basename

```
[methody@localhost methody]$  basename  /home/methody/text
text
[methody@localhost methody]$ basename text
text
[methody@localhost methody]$ dirname /home/methody/text
/home/methody
[methody@localhost methody]$ dirname ./text .
[methodyOlocalhost methody]$  dirname text
```

Ми помітили, що для «text» і «./text» dirname видало однаковий результат: «.», що зрозуміло: як було сказано вище, ці форми шляху цілковито еквівалентні, а при автоматичній обробці результатів dirname значно краще отримати «.», ніж пустий рядок.

4.2 Домашній каталог

Ми помітили, що на попередніх лекціях і на цій, заходячи з різних віртуальних консолей по черзі і одночасно, ми завжди опинялися в одному і тому ж поточному каталозі: він весь час звертався до своїх файлів за допомогою відносного шляху і завжди знаходив потрібні. Це не випадково – в Linux у кожного користувача обов'язково є свій особистий каталог, який і стає поточним відразу після реєстрації у системі – домашнім каталогом. Для нас домашнім каталогом є *«**/home/methody***».

Домашній каталог– **к**аталог, призначений для зберігання особистих даних користувача Linux. Як правило, є ***поточним*** безпосередньо після реєстрації користувача в системі.

Повний шлях до домашнього каталогу зберігається у ***змінній оточення HOME***.

Оскільки кожний користувач має у розпорядженні свій особистий каталог і по замовченню працює у ньому, вирішується задача розділення файлів різних користувачів. Зазвичай, доступ інших користувачів до чужого домашнього каталогу обмежений: найбільше типова ситуація, коли користувачі можуть читати вміст файлів один одного, але не мають права їх змінювати або знищувати.

4.3 Інформація про каталог

Щоби мати можливість орієнтуватися у файловій системі, треба знати вміст кожного каталогу. Запам'ятати усю структуру файлової системи не можливо і не треба: у любий момент можна продивитися вміст любого каталогу за допомогою утиліти ***ls*** (скорочення від англійської «***list***» - «*список*»):

Приклад 4.4 Команда ls

```
[methody@localhost methody]$ ls
-filename-with- text
 [methody@localhost methody]$
```

Команда ***ls*** без параметрів виводить список і каталогів, які вміщуються у поточному каталозі. За допомогою цієї утиліти ми виявили, що у домашньому каталозі знаходяться два файли, які були утворені на попередній лекції: «***-filename-with-***» і «***text***».

Утиліта ***ls*** приймає один ***параметр***: ім'я каталогу, вміст якого треба вивести. Ім'я може бути задано любим доступним засобом: у вигляді ***повного*** або

відносного шляху. Наприклад, щоби отримати список файлів у своєму домашньому каталозі, могли б використовувати команди: *«**ls /home/methody***» або «**ls .**» - результат був би аналогічним.

Крім параметру утиліта ***ls*** «розуміє» багато ключів, які потрібні, головним чином для того, що би виводити додаткову інформацію про файли у каталозі і виводити список файлів вибірково. Щоби узнати про усі можливості ***ls*** треба, звісно, прочитати ***керівництво*** по цій утиліті («***man ls***»).

Прочитавши керівництво по ***ls***, ми вирішили вивчити вміст своєї файлової системи і почали з початку – *з **корінного каталогу***.

Приклад 4.5 Команда ls -F

```
[methody@localhost methody]$ ls -F /
bin/  dev/  home/  mnt/  root/  swap/  tmp/  var/
boot/  etc/  lib/  proc/  sbin/  sys/  usr/
[methody@localhost methody]$
```

Ми використали ключ ***-F***, щоби відрізняти файли від каталогів. При наявності цього ключа ***ls*** в кінці імені кожного каталогу ставить символ «/», щоби показати, що в ньому може бути іще щось. У списку, який виведено, немає жодного файлу. У корінному каталозі вміщуються тільки підкаталоги.

Крім того, ми вирішили отримати більш детальну інформацію о вмісті свого домашнього каталогу.

Приклад 4.6 Команда ls -aF

```
[methody@localhost methody]$ ls -aF
-filename-with- .bash_history .bashrc .lpoptions .rpmmacros Documents/./
.bash_logout .emacs .mutt/ .xemacs/ text
../ .bash_profile .i18n .pinerc .xsession.d/ tmp/
[methody@localhost methody]$
```

Раптово ми виявили, що файлів у нашому каталозі не два, а значно більше. Справа в тому, що утиліта ***ls***, по замовченню, не виводить інформацію про об’єкти, чиє ім’я починається з «**.**» - в тому числі, з «**.**» і «**..**». Для того, щоби переглянути *повний* список вмісту каталогу, використовується ключ «***-a***» (***all***). Як правило, з «**.**» починаються імена ***конфігураційних файлів*** і ***конфігураційних каталогів***, робота з якими не перетинається з роботою над якою-небудь прикладною задачею. Крім того, подібних файлів у домашньому каталозі активно працюючого користувача з часом заводиться не мало, і їх присутність у видачі ***ls***, сильно захаращує її.

Розберемося детально у списку файлів нашого домашнього каталогу. Почнемо з досить лаконічних імен «**.**» і «**..**». Ми вже знаємо, що «**.**» - це ім’я поточного каталогу. Наступне ім’я у списку «**..**» - *це **батьківський каталог***. Батьківський каталог – це каталог, в *якому* знаходиться даний. Батьківським каталогом для *«**/home/methody***» буде каталог *«**/home***»: він виходить просто відкиданням останнього імені каталогу у повному шляху. Інакше можна сказати, що батьківський каталог – це один крок по дереву каталогів у напрямку до

кореня. «**..**» - це скорочений засіб посилатися на батьківський каталог: поки поточним каталогом є *«**/home/methody***», відносний шлях «**..**» (або теж саме «***/..***») буде еквівалентним «***/home***». З використанням «**..**» можна будувати скільки завгодно довгі шляхи, такі як «***../../usr/../var/log/../run/../../home***».

Увага!

Не зразу зрозуміло, що приводить цей шлях все туди ж в «***/home***».

Одначе, в дійсності вони застосовуються тільки при автоматичній підстановці у програмах, а під час роботи користувача необхідності у такого роду ускладненнях не виникає.

Батьківський каталог – каталог у якому вміщується даний. Для ***корінного каталогу*** батьківським являється він сам.

Посилання на поточний і на батьківський каталог обов'язково присутні у кожному каталозі в Linux. Навіть якщо каталог пустий, тобто, не вміщує жодного файлу чи підкаталогу, команда «***ls -a***» виведе список двох імен «**.**» і «**..**».

За посиланнями на поточний і батьківський каталоги прямують декілька файлів і каталогів, імена яких починаються на «**.**». в них вміщуються налаштування ***командної оболонки*** та інших програм. У домашньому каталозі користувача завжди присутні декілька таких файлів. Використання таких файлів дозволяє користувачам незалежно один від одного налаштовувати поведінку командної оболонки і інших програм – організовувати своє «робоче місце» у системі.

4.4 Переміщення по дереву каталогів

Користувач може працювати з файлами не тільки у своєму домашньому каталозі, але і в інших каталогах. У цьому випадку буде зручно замінити **поточний каталог**, тобто «переміститися» в іншу точку файлової системи. Для зміни поточного каталогу командної оболонки використовується команда ***cd*** (від англійської «***c****hange* ***d****irectory*» - «*змінити каталог*»). Команда ***cd*** приймає один параметр: ім'я каталогу в який треба переміститися – зробити поточним. Зазвичай, в якості імені каталогу можна використати повний і відносний шлях.

Приклад 4.7 Зміна поточного каталогу

```
[methody@localhost methody]$ cd /home
[methody@localhost home]$  ls
methody   shogun
[methody@localhost home]$ cd methody
[methody@localhost methody]$
```

Спочатку ми вирішили переміститися у каталог «***/home***», подивитися що ще є у цьому каталозі, крім нашого домашнього каталогу. Ми винайшли ще один каталог «***shogun***», і здогадалися, що це домашній каталог Шогана, вхідне ім'я якого «***shogun***». Крім того, ми помітили, що змінився вид запрошення **командного рядку** – слово «***methody***» замінилося на «***home***». У запрошенні

командного рядка часто вказується поточний каталог ***shell*** – щоби користувачу легше було користуватися, у якому каталозі він знаходиться у даний момент.

Після того ми вирішили повернутися у свій домашній каталог, але у цьому випадку ми вже використали не повний, а відносний шлях – «***cd methody***». Вводячи цю команду ми не стали повністю набирати ім'я свого домашнього каталогу, а набрали тільки перші дві літери «***me***» і натиснули клавішу «***tab***». Якщо є тільки один варіант завершення імені – оболонка закінчить його сама, користувачеві не прийдеться символи, які залишилися.

Добудова – вельми серйозний засіб економії зусиль і підвищення ефективності при роботі з командним рядком. Сучасні командні оболонки вміють добудовувати імена файлів і каталогів, а також імена команд. Добудова найбільше розвинута у командному інтерпретаторі ***zsh***.

Ті ж самі переміщення – в батьківський каталог і назад – ми могли би зробити набираючи значно менше символів. Для переміщення у батьківський каталог *(«**home**»)* зручно користуватися посиланням «**..**». Необхідність повернутися у домашній каталог з любої точки файлової системи виникає доволі часто, тому командна оболонка підтримує позначення домашнього каталогу за допомогою символу «~». Звідси, щоби перейти у домашній каталог із любого іншого, достатньо виконати команду «***cd ~***». При виконанні команди символ «~» буде замінено командною оболонкою на повний шлях до домашнього каталогу користувача.

Приклад 4.8 Перехід в батьківський та домашній каталог

```
[methody@localhost      methody]$ cd ..
[methody@localhost      home]$ cd ~
[methody@localhost      methody]$ cd ~shogun
[methody@localhost      shogun]$ cd
[methody@localhost      methody]$
```

За допомогою символа «~» можна посилатися на домашні каталоги інших користувачів: «**~ ім'я користувача**». У прикладі 4.8. ми перейшли у домашній каталог Шогуна командою «***cd ~shogun***». Команда ***cd***, подана без параметрів, еквівалентна команді «***cd ~***» і робить поточним каталогом домашній каталог користувача.

4.5 Утворення каталогів

Користувач, звісно, не повинен зберігати усі свої файли в одному каталозі. У домашньому каталозі користувача, як і у любому іншому, можна утворювати скільки завгодно підкаталогів, у них своїх підкаталогів і так далі. Іншими словами, користувачу належить фрагмент (під дерево) файлової системи, корінням якого є домашній каталог користувача.

Щоби організувати таке під дерево, необхідно утворити каталоги у середині домашнього. Для цього використовується утиліта ***mkdir***. Вона

використовується з одним обов'язковим параметром: іменем каталогу, який утворюється. За замовченням, каталог буде утворено у поточному каталозі.

Приклад 4.9 Утворення каталога

```
[methody@localhost methody]$   mkdir examples
[methody@localhost methody]$   ls -F
-filename-with-  Documents/  examples/  text  tmp/
[methody@localhost methody]$
```

Ми вирішили навести деякий порядок у своєму домашньому каталозі і розмістили усі файли з прикладами і вправами у окремий підкаталог – «***examples***». Тепер, утворивши каталог, в нього потрібно перемістити усі файли з прикладами.

4.6 Копіювання і переміщення файлів

Для переміщення файлів і каталогів призначена утиліта ***mv*** (скорочення від англійської «***move***» - «*переміщувати*»). У ***mv*** два обов'язкових параметри: перший – файл або каталог, який переміщується, другий - файл або каталог призначення. Імена файлів або каталогів можуть бути задані у любому допустимому вигляді: за допомогою повного або відносного шляху. Крім того, ***mv*** дозволяє переміщати не тільки один файл або каталог, а відразу декілька. За деталями про допустимі параметри і ключі слід звертатися до керівництва з ***mv***.

Приклад 4.10 Переміщення файлів

```
[methody@localhost methody]$  mv --  -filename-with-  examples/
[methody@localhost methody]$  cd examples
[methody@localhost examples]$ mv ../text  .
[methody@localhost examples]$ ls
 -filename-with-   text
[methody@localhost examples]$
```

Ми спочатку перемістили у каталог «***examples***» файл «***-filename-with-***», оскільки ім'я цього файлу починається з «-», перед ним треба було поставити ключ «--», щоб наступне слово було сприйнято командною оболонкою як параметр.

Потім ми перейшли у каталог «***examples***» і перемістили з батьківського каталогу (***../***) файл «***text***» у поточний каталог («**.**»). Тепер у каталозі «***examples***» два файли з прикладами.

Переміщення файлу у середині однієї файлової системи у дійсності рівнозначно його *перейменуванню*: дані самого файлу залишаються на тих же секторах диску, змінюються *каталоги*, у яких пройшло переміщення. Переміщення передбачає знищення посилання на файл з того каталогу звідки

його переміщено, і додання посилання на той самий каталог, куди він переміщений. У результаті змінюється повне ім'я файлу – ***повний шлях***, тобто, положення файлу у файловій системі.

Іноді вимагається утворити копію файлу: для більшого збереження даних, для того, щоби утворити модифікаційну версію і тому подібне. У Linux для цього призначена утиліта ***cp*** (скорочення від англійської «***copy***» - «*копіювати*»). Утиліта ***cp*** вимагає двох обов'язкових параметрів: перший – файл або каталог, який копіюється, другий – файл або каталог призначення. Зазвичай, у іменах файлів і каталогів можна використовувати повні і відносні шляхи. Є декілька можливостей при комбінації файлів і каталогів у параметрах ***cp*** – про них можна прочитати у ***керівництві***.

Приклад 4.11 Копіювання файлів

```
[methody@localhost examples]$  cp  text  text.bak
 [methody@localhost examples]$ ls
 -filename-with-   text   text.bak
```

Ми вирішили утворити резервну копію файлу «***text***», «***text.bak***», в тому ж каталозі, що і початковий файл. Для цієї простішої операції копіювання достатньо передати ***cp*** у якості двох параметрів ім'я початкового файлу і ім'я копії. За замовченням ***cp***, як і багато інших утиліт, буде робити з файлами у поточному каталозі.

Слід мати на увазі, що у Linux утиліта ***cp*** часто налаштована таким чином, що при спробі скопіювати файл поверх уже існуючого, не виводить ніяких попереджень. У цьому випадку файл буде просто пере записано. Дані, які вміщувалися у старій версії файлу, втрачені безповоротно. Тому, при використанні ***cp***, слід завжди бути обережним і перевіряти імена файлів, які треба скопіювати.

Говорячи про копіювання, своєчасно згадати відомій вислів: «Не слід множити суті поверх необхідного». Утворена за допомогою ***cp*** копія файлу пов'язана в оригіналом тільки у спогадах користувача. У файловій системі попередній файл і його копія – дві, зовсім незалежні і нічим не пов'язані одиниці. Тому, при наявності декількох копій одного і того ж файлу у рамках *однієї файлової системи* підвищується вірогідність заплутатися у копіях або забути про деякі з них. Якщо задача стоїть у тому, щоби забезпечити доступ до одного і того ж файлу з різних точок файлової системи, треба використовувати спеціально призначений для цього механізм файлової системи Linux – посилання.

4.7 Файл та його імена. Посилання

4.7.1 Жорсткі посилання

Кожний файл уявляє собою область даних на жорсткому диску комп'ютера або іншого носія інформації, яку можна знайти по *імені*. У файловій системі

Linux вміст файлу пов'язується з його іменем за допомогою ***жорстких посилань***. Утворення файлу за допомогою любої програми означає, що буде утворене жорстке посилання – ім'я файлу, і відкрита нова область даних на диску. Причому, кількість посилань на одну й ту ж область даних (файл) не обмежено, тобто, у файла може бути декілька імен.

Користувач Linux може добавити файлу ще одне ім'я (утворити ще одне жорстке послання на файл) за допомогою утиліти ***ln*** (скорочення від англійської «***link***» - «*зв'язувати*»). Перший параметр – це ім'я файлу, на який треба утворити посилання, другий – ім'я нового посилання. За замовченням, посилання буде утворене у поточному каталозі.

Приклад 4.12. Утворення жорстких послань

```
[methody@localhost methody]$ ln examples/text text-hardlink
[methodyQlocalhost methody]$ ls –lR
./:
. . .
drwxr-xr-x   3 methody methody 4096 Окт 16 04:45 examples
-rw-r--r--         2 methody methody   653 Окт   6 10:31 text-hardlink
./examples:
итого  92
-rw-r--r--         1 methody methody     84718 Окт   6 10:31   -filename-with-
-rw-r--r--         2 methody methody           653 Окт   6 10:31     text
```

Користувач утворив у своєму домашньому каталозі жорстке посилання з іменем «***text-hardlink***» на файл «***text***», який знаходиться у каталозі «***examples***». Виводячи детальний список файлів поточного каталогу і його підкаталогів («***ls -lR***»), ми звернули увагу що у файлів «***text***» і «***text-hardlink***» співпадають і розміри («***653***») і час утворення. Це зовсім на не здивувало, оскільки ми знаємо, що тепер «***/home/ methody/ text-hardlink***» і ««***/home/ methody/ examples/text***» - це два імені одного і того ж файлу. У детальному описі, виведеному командою «***ls -lR***», нам залишилися не зрозумілими тільки два перших поля. Як потім з'ясується, перше слово із знаків «***-drwx***», – це позначення прав доступу до файлу. А наступне за ним число – кількість жорстких посилань на даний файл або каталог. У «***text***» і «***text-hardlink***» стоїть число «***2***» – у цього файла два імені.

Доступ до одного і того ж файлу за допомогою декількох імен може знадобитися у наступних випадкахю.

1. Одна і та ж програма відома під декількома іменами.
2. Доступ користувачів до деяких каталогів у системі може бути обмеженим з міркувань безпеки. Одначе, якщо все ж таки треба організувати доступ користувачів до файлу, який знаходиться у такому каталозі, можна утворити жорстке послання на цей файл у іншому каталозі.

3. Сучасні файлові системи навіть на домашніх персональних комп'ютерах можуть нараховувати до десятків тисяч файлів і тисячі каталогів. Зазвичай, у таких файлових системах складна багаторівнева ієрархічна організація – у результаті шляхи до багатьох файлів стають дуже довгими. Щоб організувати більше зручний доступ до файлу, який знаходиться «глибоко» у ієрархії каталогів, також можна використовувати жорстке послання у більше доступному каталозі.

4. Повне ім'я програм може бути вельми довгим (наприклад, ***i586-alt-linux-gcc-3.3***), до таких програм зручніше звертатися за допомогою скороченого імені (жорсткого послання) – ***gcc-3.3***.

4.7.2 Індексні дескриптори

Оскільки, завдяки жорстким посиланням у файла може бути декілька імен, розуміло, що уся суттєва інформація про файл у файловій системі прив'язана не до імені. У файловій системі Linux уся інформація, яка необхідна для роботи з файлом, зберігається у ***індексному дескрипторі***. Для кожного файлу існує індексний дескриптор: не тільки для звичайних файлів, але і для каталогів, ***файлів-дирок*** і інших. Кожному файлу відповідає рівно один ***індексний дескриптор***.

Індексний дескриптор – це опис файла, в якому вміщується:

- тип файла (звичайний, каталог, файл-дирка та інше);
- ***права доступу*** до файлу;
- інформація про те, кому належить файл;
- відмітка про час утворення, модифікації, останнього доступу до файлу;
- розмір файла;
- вказівники на фізичні блоки на диску, які належать цьому файлу – у цих блоках зберігається «вміст» файлу.

Усі індексні дескриптори пронумеровані, тому, номер індексного дескриптора – це унікальний ідентифікатор файлу у файловій системі – на відміну від *імені* файлу (жорсткого послання на нього), яких може бути декілька. Дізнатися номер індексного дескриптора любого файлу можна за допомогою все тієї ж утиліти ***ls*** з ключем ***-i***:

Приклад 4.13 Інформація про індексні дескриптори файлів

```
[methody@localhost methody]$ ls -i ./text-hardlink examples/text
127705 examples/text  127705 ./text-hardlink
```

Ми вирішили поцікавитися номерами індексних дескрипторів файлу «***text***» і жорсткого послання на нього «***text-hardlink***» – ми виявили, що ці номери співпадають («***127705***»), тобто, цим двом іменам відповідає один індексний дескриптор, тобто, один і той же файл.

Усі операції з файловою системою – утворення, знищення або переміщення файлів – провадяться, насправді, над індексними дескрипторами,

імена потрібні тільки для того, щоби користувач міг легко орієнтуватися у файловій системі.

Більше того, ім'я (або імена) файлу *не вказані* у його індексному дескрипторі. У файловій системі ***Ext2*** імена файлів зберігаються у *каталогах*: кожний каталог уявляє собою список імен і номерів їхніх індексних дескрипторів. Жорстке послання можна уявляти як каталожну картку, на якій вказано номер індексного дескриптора – ідентифікатор файлу.

Жорстке послання

Запис виду *ім'я файлу + номер індексного дескриптору* у ***каталозі***. Жорсткі послання у Linux – основний засіб звернутися до файлу ***за іменем***.

4.7.2.1 Символьні послання

У жорстких посилань є два суттєвих обмеження.

1. Жорстке посилання може вказувати тільки на файл, але не на каталог том, що у протилежному випадку, у файловій системі можуть виникнути нескінченні цикли.
2. Жорстке послання не може вказувати на файл на іншій файловій системі. Наприклад, не можливо утворити на жорсткому дику жорстке послання на файл, який розташований на флешці.

Щоби уникнути цих обмежень, були розроблені символьні послання. Символьне послання – це просто файл в якому вміщується ім'я іншого файлу. Символьні послання, як і жорсткі, представляють можливість звертатися до одного і того ж файлу за різними іменами. Крім того, символьні послання можуть вказувати і на каталог, чого не дозволяють жорсткі послання. Символьні послання називаються так тому, що вміщують *символи* – шляхи до файлу або каталогу.

Символьне послання

Файл особливого типу («***l***»), в якому вміщується шлях до іншого файлу. Якщо на шляху до файлу зустрічається символьне послання, система виконує *підстановку*: початковий шлях замінюється на той, що вміщується у посланні.

Символьне послання можна утворити за допомогою команди ***ln*** з ключем «***-s***» (скорочення від «***symbolic***»):

Приклад 4.14. Утворення символьних послань

```
[methody@localhost methody]$ ln -s examples/text text-symlink
[methody@localhost methody]$ ls -li
. . .
127699 drwxr-xr-x  2 methody  methody 4096  Окт  4 17:12 examples
127705 -rw-r--r--  2 methody  methody  653  Сен 30 10:04 text-hardlink
 3621 lrwxrwxrwx  1 methody  methody    3  Окт  4 18:05 ext-symlink-> examples/text
[methody@localhost  methody]$
```

Тепер ми вирішили створити у своєму домашньому каталозі і символьне послання на файл ***text*** і назвати його ***text-symlink***. Команда ***ls -li*** відобразила цей файл зовсім не так, як інші: стрілочка («->») вказує, куди направлене посилання. Крім того, ми звернули увагу, що номер індексного дескриптора (перше поле), розмір і час утворення файлу ***text-symlink*** відрізняється від ***text-hardlink***, а також у третьому полі (кількість жорстких посилань на файл) ***text-symlink*** вказано «***1***». Усі ці ознаки чітко засвідчують про те, що ***text-symlink*** і ***text*** – це *різні* файли. Одначе, якщо виконати команду ***cat text-symlink***, то на екран буде виведено вміст файлу ***text***.

Символьне посилання цілком може вміщувати ім'я неіснуючого файлу, у цьому випадку, посилання буде існувати, але «працювати» не буде: наприклад. якщо спробувати вивести вміст такого «битого» послання за допомогою команди ***cat***, буде виведено повідомлення про помилку. Узнати, куди вказує символьне послання, можна за допомогою утиліти ***realpath***.

Приклад 4.15. Розкриття символьних послань

[methody@localhost methody]$ realpath text-symlink /home/methody/examples/text

4.8 Знищення файлів і каталогів

У Linux для знищення файлів призначена утиліта ***rm*** (скорочення від англійської «***remove***» - «*знищувати*»).

Приклад 4.16. Знищення файлів

```
[methody@localhost methody]$ rm   examples/text
[methody@localhost methody]$ ls -l  text-hardlink
-rw-r--r--  1  methody  methody  653  Сен 30 10:04  text-hardlink
[methody@localhost methody]$ rm  text-hardlink
[methody@localhost methody]$ ls -l text-hardlink
ls: text-hardlink: No such file or directory
```

Розібравшись у посланнях, ми вирішили знищити файл ***text*** в каталозі ***examples***. Після цього, файл ***text-hardlink*** у нашому домашньому каталозі, який є жорстке послання на знищений файл продовжує існувати. Єдина різниця, яку ми помітили – кількість жорстких посилань на файл тепер зменшилася з «***2***» до «***1***» – дійсно, ***text-hardlink*** – тепер єдине ім'я цього файлу. Виходить, що ми знищили тільки одне з імен цього файлу. Виходить, що ми знищили тільки одне з імен цього файлу (**жорстке посилання**), сам файл залишився не займаним.

Одначе, якщо ми знищимо і жорстке посилання ***text-hardlink*** – у цього файла не залишиться більше жодного імені, він стане недоступним користувачу файлової системи і буде знищений.

Утиліта ***rm*** призначена саме для знищення жорстких посилань, а не для самих файлів. В Linux, щоби повністю знищити файл, вимагається послідовно

знищити усі жорсткі посилання на нього. При цьому, усі жорсткі посилання на файл (його імена) рівноправні – серед них немає «головного», з зникненням якого зникне файл. Поки є хоч одне посилання, файл продовжує існувати.

Втім, у більшості файлів у Linuxє тільки одне ім'я (одне жорстке посилання на файл), тому команда ***rm ім'я файлу*** успішно знищить цей файл у більшості випадків.

Як вже говорилося, символьні посилання – це окремі файли, тому після того, як ми знищили файл ***text, text-symlink***, який посилається на цей файл, продовжує існувати, одначе тепер це – «бите послання», тому його також можна знищити командою ***rm***.

Ми вирішили утворити каталог для різних вправ – ***test***, а потім прийняли рішення обійтися одним каталогом ***examples***. Одначе, команда ***rm*** не спрацювала, заявивши що ***test*** – це каталог:

Приклад 4.17. Знищення каталогу

```
[methody@localhost methody]$ mkdir test
[methody@localhost methody]$ rm test
 rm: невозможно удалить 'test': Is a directory
[methody@localhost methody]$ rmdir test
 [methody@localhost methody]$
```

Для знищення каталогів призначена інша утиліта ***rmdir*** (від англійської «***remove directory***»). Втім, ***rmdir*** згодиться знищити каталог, якщо він пустий: у ньому немає ніяких файлів і підкаталогів. Знищити каталог разом з його вмістом, можна командою ***rm*** з ключем «***-r***» (***recursive***). Команда ***rm-r*** каталог – дуже зручний засіб загубити відразу *усі* файли: вона рекурсивно обходить весь *каталог*, знищуючи усе що попадеться: файли, підкаталоги символьні посилання … , а ключ «***-f***» (**f**orce) робить її роботу ще не зворотною так, як пригнічує запити виду «знищити захищений від запису файл», так що ***rm*** працює тихо і без зупинки.

Увага!

Пам'ятайте, якщо ви нищили файл, це означає, він вже не потрібен, і не підлягає відновленню.

В Linux не передбачені процедури відновлення знищених файлів і каталогів. Тому, варто бути дуже обережним відаючи команду ***rm*** і тим більше: ***rm –r***, немає ніякої гарантії, що вдасться відновити випадково знищені дані.

5 ДОСТУП ПРОЦЕСІВ ДО ФАЙЛІВ ТА КАТАЛОГІВ

5.1 Процеси

Як вже згадувалося раніше, завантаження Linux завершується тим, що на всіх віртуальних консолях, призначених для роботи користувачів, запускається програма ***getty***. Програма виводить запрошення і очікує активність користувача, який може забажати працювати саме на цьому терміналі. Вхідне ім'я, яке введене, ***getty*** передає програмі ***login***, яка вводить пароль і визначає, чи дозволено працювати у системі з цими вхідним іменем та паролем. Якщо ***login*** приходить до висновку, що працювати можна, він запускає стартовий ***командний інтерпретатор***, за допомогою якого користувач і керує системою.

Програма, яка виконується, називається в Linux ***процесом***. Усі процеси система реєструє у ***таблиці процесів***, надаючи кожному унікальний номер – ***ідентифікатор процесу*** (***p****rocess* ***id****entificator,* ***PID***). Маніпулюючи процесами, система має справу саме з їх ідентифікаторами, іншого засобу відрізнити один процес від іншого, за великим рахунком, не має. Для перегляду своїх процесів можна скористуватися утилітою ***ps*** («***p****rocess* ***s****tatus*»):

Приклад 5.1 Перегляд таблиці особистих процесів.

```
[methody@localhost methody]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
methody   3590  1850  0 13:58 tty3     00:00:00 -bash
methody   3624  3590  0 14:01 tty3     00:00:00 ps -f
```

Тут ми викликали ***ps*** з ключем «***-f***» («***-full***»), щоби добути побільше інформації. Представлені обидва процеси, які належать нам: стартовий командний інтерпретатор, ***bash***, і ***ps***, яка виконується. Обидва процеси запущено з терміналу ***tty3*** (третьої системної консолі), і мають ідентифікатори ***3590*** та ***3624*** відповідно. У полі ***PPID*** («***p****arent* ***p****rocess* ***id****entificator*») вказано ідентифікатор батьківського процесу, який *породив* даний. Для ***ps*** це – ***bash***, для ***bash***, очевидно, ***login***, так як саме він запускає стартовий ***shell***. При виведенні даних не знайшлося рядку для цього ***login,*** рівно як і для більшості інших процесів системи. Оскілки вони не належать користувачеві.

Процес **–** програма, яка виконується у Linux. Кожний процес має ***унікальний ідентифікатор процесу, PID***. Процеси отримують доступ до ресурсів системи (оперативної пам'яті, файлам, зовнішнім пристроям, тощо) і можуть змінювати їх вміст. Доступ до ресурсів регулюється за допомогою ідентифікатора користувача і ідентифікатора групи, які система присвоює кожному процесу.

5.1.1 Запуск дочірніх процесів

Запуск одного процесу замість *іншого* об лаштований в Linux за допомогою ***системного виклику exec()***. Старий процес знищується у пам'яті назавжди, замість нього завантажується новий, при цьому налаштування *оточення* не змінюється, навіть ***PID*** залишається колишнім. Повернутися до виконання старого процесу не можливо, якщо тільки не запустити його по новому за допомогою того ж ***exec()*** (від «***execute***» - «*виконувати*»). До речі, *ім'я файлу* (програми), з якого запускається процес, і *особисте ім'я процесу* (у таблиці процесів) можуть не співпадати. Особисте ім'я процесу – це такий же параметр командного рядку, як і ті що передаються йому користувачем: для ***exec()*** вимагається і шлях до файлу, і *повний* командний рядок, *нульовий* (стартовий) елемент якого – як раз є назвою команди.

Увага!

Нульовий параметр – ***argv[0]*** у термінах мови С і $0 у термінах shell.

Ось звідки «-» на початку імені *стартового* командного інтерпретатора (-***bash***): його «поставила» програма ***login***, щоби була можливість відрізняти його від інших запущених тим же користувачем оболонок.

Для роботи командного інтерпретатора не достатньо одного ***exec()***. Звичайно, ***shell***, не просто запускає утиліту, а чекає її завершення, обробляє результати її роботи і продовжує діалог з користувачем. Для цього в Linux служить системний виклик ***fork()*** («*вилка, розвилка*»), застосування якого призводить до виникнення ще одного *дочірнього* процесу – точної копії *батьківського*, який породив його. ***Дочірній процес*** ні чим не відрізняється від батьківського: має таке ж оточення, ті ж стандартні введення і вивід, однаковий вміст пам'яті і продовжує роботу з тієї ж самої точки (повернення з ***fork ()***). Відміни дві - *по перше*, ці процеси мають *різні* ***PID***, під якими їх зареєстровано у таблиці процесів, а *по друге*, різниться ***значення fork (), яке повертається:*** *батьківський процес* отримує в якості результату ***fork ()*** ідентифікатор процесу-нащадка, а *процес-нащадок* отримує «***0***».

Подальші дії ***shell*** при запуску якої-небудь програми очевидні. Shell-нащадок відразу викликає цю програму за допомогою ***exec()***, а ***shell***-батько дочікується завершення роботи процесу-нащадка. (***PID*** якого йому відомий) за допомогою ще одного системного виклику ***wait()***. Дочекавшись і проаналізувавши результат команди, ***shell*** продовжує роботу.

Приклад 5.2 Створення сценарію, який виконується нескінченно.

```
[methody@localhost methody]$ cat > loop
while true; do true; done
^D
[methody@localhost methody]$ sh loop
 ^C
[methody@localhost methody]$
```

За порадою інструкції, ми утворили сценарій для ***sh*** (або ***bash***, на *такому* рівні їхні команди співпадають), який нічого не робить. Точніше було би сказати, що цей сценарій робить нічого, нескінченно повторюючи у циклі команду, вся робота якої складається у тому, що вона закінчується без помилок. Запустивши цей сценарій за допомогою команди ***sh імя_сценарію,*** ми нічого не побачили, але почули, як загудів вентилятор охолодження центрального процесору: машина працювала. Управляючий символ «***^C***», зазвичай привів до завершення активного процесу, і командний інтерпретатор продовжив роботу.

Як би у ситуації, яка вище описана, батьківський процес не очікував, поки дочірній завершиться, а відразу продовжував працювати, вийшло би, що обидва процеси виконуються «*паралельно*»: поки запущений процес щось робить, користувач продовжує командувати оболонкою. Для того, щоби запустити процес паралельно, в ***shell*** достатньо добавити «***&***» у кінець командного рядку:

Приклад 5.3 Запуск фонового процесу

```
[methody@localhost methody]$ sh loop&
 [1] 3634
[methody@localhost methody]$ ps –f
UID         PID    PPID    C   STIME   TTY      TIME       CMD
methody    3590    1850    0    13:58   tty3   00:00:00   -bash
methody    3634    3590   99    14:03   tty3   00:00:02    sh loop
methody    3635    3590    0    14:03   tty3   00:00:00    ps –f
```

У результаті, стартовий командний інтерпретатор (***PID 3590***) виявився батьком відразу двох процесів: ***sh***, який виконує сценарій ***loop*** і ***ps***. Процес, який запускається паралельно. Називається фоновим («***background***»). Фонові процеси не мають можливості *вводити* дані з того ж терміналу, що і процес, який їх народив, ***shell*** (тільки з файлу), зате *виводити* на цей термінал можуть (правда, коли на одному і тому ж терміналі упереміж в'являються повідомлення від декількох фонових процесів, починається повний безлад). При кожному терміналі у кожний момент часу може бути не більше одного ***активного*** (***foreground***) процесу, якому дозволено вводити з терміналу. На час, поки команда (наприклад, ***cat***) працює у активному режимі, командний інтерпретатор, який породив її, «*іде у фон*», і там, у фоні, виконує свій ***wait()***.

Активний процес – процес, який має можливість вводити дані з терміналу. У кожний момент, у кожного терміналу може бути не більше одного активного процесу.

Фоновий процес – процес, який не має можливості вводити дані з терміналу. Користувач може запустити любу кількість фонових процесів. Ця кількість не повинна перебільшувати раніше заданого системі числа можливих фонових процесів.

Варто помітити, що паралельність роботи процесів в Linux – *дискретна*. Тут і зараз виконатися може стільки процесів, скільки паралельних процесорів є у комп'ютері (наприклад, один). Давши цьому одному процесу трохи поробити, система запам'ятовує усе, що тому для роботи необхідно, призупиняє його, і запускає *наступний* процес, потім наступний і так далі. Виникає *черга* процесів, які очікують виконання. Процес, який тільки що поробив, розміщується у кінці цієї черги, а наступний вибирається з її початку. Коли черга знову підходить до того першого процесу, система згадує необхідні для його виконання дані (вони називаються ***контекстом процесу***), і він продовжує роботу як ні в чому не бувало. Така схема *розділення часу* між процесами носить назву ***псевдо паралелізму***.

У видачі ***ps***, яку ми отримали, можна помітити, що ***PID*** стартової оболонки рівний ***3590***, а ***PID*** запущений із під його команд (одна фонова і одна активна) ***3634*** і ***3635***. Це означає, що за час, який пройшов з моменту нашого входу у систему, до моменту запуску ***sh loop&***, у системи було запущено ще 3634-3590=44 процеси. Ми знаємо, що в Linux можуть одночасно працювати декілька користувачів, але і самій системі треба іноді запустити яку-небудь утиліту (наприклад, виконуючи дії за розкладом). А ось ***sh*** і ***ps*** отримали з'єднання ***PID*** рядом, це означає, що поки натискали ***Enter*** і набирали ps ***-f***, ніяких інших процесів не запускалося.

У дійсності, далеко не усім процесам, зареєстрованим у системі, *на справді*, необхідно давати попрацювати нарівні з іншими. Більшості процесів працювати *прямо зараз* не потрібно: вони очікують якої-небудь події, яку їм треба обробити. Частіше за все. Процеси очікують завершення операції вводу-виводу. Щоби подивитися як використовуються ресурси системи, можна використати утиліту ***top***. Але, спочатку, ми вирішили запустити *ще однин* нескінченний сценарій: цікаво, як два процеси конкурують за ресурси системи між собою:

Приклад 5.4 Розділення часу між процесами.

```
[methody@localhost methody]$ bash loop&
[2] 3639
[methody@localhost methody]$ top
14:06:50  up  3:41,  5 users,  load average:  1,31, 0,76, 0,42
4 processes:  1 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 99,4%% user,  0,5%% system,  0,0%% nice,  0,0%% iowait,  0,0%% idle
Mem:  514604k av,  310620k used,  203984k free,0k shrd,  47996k buff
                   117560k active,               148388k inactive
Swap: 1048280k av,                0k used,  1048280k free  84340k cached
```

```
  PID                                    S     '     /     T     C
USER      RI    I    IZE   SS    HAR TAT  /.CPU  MEM  IME   OMM
                         E                                  AND
  3639                                   F     5     0     0     b
methody   0          260   260   044       0,3   ,2    :12   ash
  3634                                   F     4     0     3     s
methody   8          80    80    44        9,1   ,1    :06   h
  3641                                   F     0     0     0     t
methody              060   060   72        ,1    ,2    :00   op
  3590                                   S     0     0     0     b
methody              652   652   264       ,0    ,3    :00   ash
  ^C
```

Виявилося, що сперечаються навіть не два процеси, а три: sh (перший із запущених інтерпретаторів ***loop***), ***bash*** (другий) і сам ***top***. Правда, за відомостями з поля ***'/.CPU,*** левову частку процесорного часу відібрали ***sh*** і ***bash*** (вони постійно обчислюють), а ***top*** задовольняється десятою часткою відсотка (а то і меншій помилці округлення). Стартовий ***bash*** взагалі не бажає працювати, він *спить* (значення «***S***», ***S****leep*, поля **STAT**,***stat****us*): чекає завершення активного процесу ***top***.

Побачивши таку різноманітність інформації, можна спробувати читати керівництво по ***top***, одначе, без знання *архітектури* Linux більша її частина не має сенсу. Хоча, деяка частина все ж таки зрозуміла: об'єм оперативної пам'яті (усієї, яка використовується, і вільна), час роботи машини, об'єм пам'яті, яка зайнята процесами і тому подібне.

Останній процес, запущений з оболонки у фоні, можна з *цієї* оболонки зробити активним за допомогою команди ***fg*** («***foreground***» - «*передній план*»).

Приклад 5.5. Перевід фонового процесу в активний стан за допомогою команди fg (foreground)

```
[methody@localhost methody]$ fg
bash loop
^C
```

Послужливий bash навіть написав командний рядок, яким було запущено цей процес: «***bash loop***». Ми вирішили «*убити*» його за допомогою управляючого символу «***^C***». Тепер останнім у фоні процесом став ***sh***, який виконує сценарій ***loop***.

5.1.2 Сигнали

Щоби завершити роботу фонового процесу за допомогою «***^C***», нам прийшлося зробити його активним. Це не завжди можливо, і не завжди зручно. Насправді, «***^C***» - це не магічна кнопка-убивець , а попередньо встановлений символ (з ascii-кодом 3), при отриманні якого з терміналу Linux передасть активному процесу ***сигнал 2*** (по «імені,***INT***», від «***interrup***t» - «*перервати*»).

Сигнал – здібність процесів обмінюватися стандартними короткими повідомленнями безпосередньо за допомогою системи. Повідомлення-сигнал не вміщує ніякої інформації, крім номеру сигналу (для зручності замість номеру можна використовувати наперед визначене системою ім'я). Для того, щоби передати сигнал, процесу достатньо системний виклик ***kill()***, а для того, щоби прийняти сигнал, не треба нічого. Якщо процесу треба якось по-особливому реагувати на сигнал, він може зареєструвати *обробник*, а якщо обробника немає, за нього відреагує система. Як правило, це призводити до негайного завершення процесу, який отримав сигнал. Обробник сигналу запускається *асинхронно*, негайно після отримання сигналу, щоби процес у цей час не робив.

Сигнал – коротке повідомлення, яке посилається системою або процесом іншому процесу. Обробляється асинхронно спеціальною програмою-обробником. Якщо процес не обробляє сигнал самостійно, це робить система.

Два сигнали – ***9*** (*KILL*) і ***19*** (*STOP*) – завжди обробляє система. Перший з них потрібен для того, убити процес *напевно* (звідси і назва). Сигнал *STOP* призупиняє процес: у такому стані процес не знищується з таблиці процесів, але й не виконується до тих пір, поки не отримає сигнал ***18*** (*CONT*) – після чого продовжить роботу. У Linux сигнал STOP можна передати активному процесу за допомогою управляючого сигналу «^***Z***».

Приклад 5.6 Переведення процесу в фон за допомогою «^Z» и bg

```
[methody@localhost methody]$ sh loop
^Z
[1]+   Stopped                 sh loop
[methody@localhost methody]$ bg
 [1]+ sh loop &
[methody@localhost methody]$ fg
sh loop
^C
[methody@localhost methody]$
```

Ми вирішили спочатку запустити вічний цикл у якості активного процесу, потім передали йому сигнал *STOP* за допомогою «^***Z***», після чого дали команду ***bg*** (***back ground***), яка запускає у фоні останній зупинений процес. Після цього, він знову перевів цей процес у активний режим, і на кінець, убив його.

Передавати сигнали з командного рядка можна любим процесам за допомогою команди ***kill –сигнал PID*** або просто ***kill PID***, яка передає сигнал ***15 (TERM)***.

Приклад 5.7 Запуск множини фонових процесів

```
[methody@localhost methody]$ sh
$ sh loop & bash loop &
[1] 3652
[2] 3653
```

$ ps -fH

UID	PID	PPID	C	STIME	TTY	TIME	CMD
methody	3590	1850	0	13:58	tty3	00:00:00	-bash
methody	3634	3590	7	14:03	tty3	00:14:18	sh loop
methody	3651	3590	0	14:19	tty3	00:00:00	sh
methody	3652	3651	0	14:19	tty3	00:00:00	sh loop
methody	3653	3651	5	14:19	tty3	00:01:01	bash loop
methody	3654	3651	0	14:19	tty3	00:00:00	ps -fH

Ми вирішили позапускати процеси, а потім вибірково вбити їх. Для цього ми, на додаток до вже висячого у фоні процесу ***sh loop***, запустили у якості активного процесу новий командний інтерпретатор, ***sh*** (при цьому змінилося ***запрошення командного рядку***). Із цього shвін запустив у фоні ще один ***sh loop*** і новий ***bash loop***. Зробили ми це одним командним рядком (при цьому команди *розділяються* символом «***&***», тобто «I»; виходить так, що запускається ***i*** та , ***i*** інша команда). У ***ps*** ми використали новий ключ – «***H***» («***Hierarchy***», «*ієрархія*»), який добавляє у видачу ***ps*** відступи, які показують відношення «*батько* - *нащадок*» між процесами.

Приклад 5.8 Примусове завершення процесу за допомогою kill

$ kill 3634

[1]+ Terminated sh loop

sh-2.05b$ ps -fH

UID	PID	PPID	C	STIME	TTY	TIME	CMD
methody	3590	1850	0	13:58	tty3	00:00:00	-bash
methody	3651	3590	0	14:19	tty3	00:00:00	sh
methody	3652	3651	34	14:19	tty3	00:01:10	sh loop
methody	3653	3651	34	14:19	tty3	00:01:10	bash loop
methody	3658	3651	0	14:23	tty3	00:00:00	ps-fH

Ми взялися завершувати процеси. Спочатку ми зупинили роботу давно запущеного sh, який виконував сценарій з вічним циклом (PID 3634). Як видно з

попереднього прикладу, цей процес за 16 хвилин роботи системи з'їв не менше 14 хвилин процесорного часу, і звісно, нічого корисного не зробив. Сигнал про те, що процес-нащадок вмер, дійшов до обробника у стартовому bash (PID 3590), і на термінал вивелося повідомлення «[1]+ Terminated sh loop», після чого стартовий bash продовжив чекати завершення активного процесу – sh (PID 3651).

Приклад 5.9 Завершення процесу природнім шляхом за допомогою сигналу «.Hang Up».

```
$ exit
[methody@localhost methody]$ ps -fH
UID          PID    PPID  C    STIME TTY          TIME CMD
Methody     3590    1850  0    15:17 tty3     00:00:00 -bash
methody     3663    3590  0    15:23 tty3     00:00:00   ps -fH
methody     3652       1  42   15:22 tty3     00:00:38 bash loop
methody     3653       1  42   15:22 tty3     00:00:40 sh loop
[methody@localhost methody]$ kill  -HUP  3652  3653
[methody@localhost methody]$ ps
PID  TTY          TIME CMD
3590 tty3     00:00:00 bash
3664 tty3     00:00:00 ps
```

Чекати залишилося недовго. Цей ***sh*** завершився природнім шляхом, від команди ***exit***, залишивши після себе двох дітей (***PID 3652*** і ***3653***), які відразу усиновив «*батько усіх процесів*» - ***init*** (***PID 1***). Коли ми розправилися і з ними – за сигналу ***1*** (***HUP***, тобто «***Hang UP***» - «*повісити*») нікому навіть повідомити про їх зникнення (якби процес-батько був живий, на зв'язаний з ним термінал вивелося би що-небудь на шталт «***[1]+ Hangup sh loop***»).

5.2 Доступ до файлу і каталогу

Пора нам подумати і про інший бік роботи з Linux: про права і свободи. Для початку – про свободи. Таблиця процесів вміщує най важливих об'єктів системи – процесів. Одначе, не менше важливі і об'єкти іншого класу, ті, що доступні у файловій системі: файли, каталоги і спеціальні файли (символьні посилання, пристрої і тому подібне). По відношенню до об'єктів файлової системи процеси виступають у ролі *діючих суб'єктів*: саме процеси користуються файлами, утворюють, знищують і змінюють їх. ***Факт*** використання файлу процесом називається ***доступом*** до файлу, а ***засіб*** скористатися файлом (каталогом, посиланням і іншим) – ***видом*** доступу.

5.2.1 Читання, запис і використання

Видів доступу у файловій системі Linux *три*. Доступ на ***читання*** (**read**) дозволяє отримувати інформацію з об'єкту, доступ на ***запис*** (**write**) – змінювати

інформацію в об’єкті, а доступ на ***використання*** (**execute**) – виконати операцію, специфічну для даного типу об’єктів. Доступ до об’єкту можна змінити командою ***chmod*** (***ch****ange* ***mod****e*, змінити режим (доступу)). У простих випадках формат цієї команди такий: ***chmod доступ об’єкт***, де *об’єкт* – це ім’я файлу, каталогу і тому подібне, а *доступ* описує вид доступу, який необхідно дозволити або заборонити. Значення «***+r***» *дозволяє* доступ до об’єкту на читання (**r**ead), «***-r***» – *забороняє*. Аналогічно «***+w***», «***-w***», «***+x***», і «***-x***» дозволяють і забороняють доступ на запис (**w**rite) і використання (e**x**ecute).

5.2.2 Доступ до файлу

Доступ до *файлу* на читання і запис – досить очевидні поняття.

Приклад 5.10 Що можна і що не можна робити з файлом, доступ до якого обмежено

```
[methody@localhost methody]$ date  >  tmpfile
[methody@localhost methody]$ cat  tmpfile
Срд Сен 22 14:52:03 MSD 2004
[methody@localhost methody]$ chmod   -r   tmpfile
[methody@localhost methody]$ cat  tmpfile
cat: tmpfile: Permission denied
[methody@localhost methody]$ date  -u  >  tmpfile
[methody@localhost methody]$ chmod  +r   tmpfile;   chmod   -w   tmpfile
[methody@localhost methody]$ cal  >  tmpfile
-bash: tmpfile: Permission denied
[methody@localhost methody]$ cat  tmpfile
Срд Сен 22 10:52:35 UTC 2004
[methody@localhost methody]$ rm   tmpfile
rm: удалить защищённый от записи обычный файл ‘tmpfile’? y
```

Слід зауважити, що нам відома операція ***пере направлення виводу*** – «>», за допомогою якої ми утворюємо файли у своєму домашньому каталозі.

Додавання «> ***файл***» у командний рядок призводить до того, що усе, що виводилося би на екран терміналу, попаде у *файл*. Ми утворюємо файл, перевіряємо, чи можна з нього читати, командою ***cat***, забороняємо доступ на читання і знову перевіряємо: на цей раз ***cat*** повідомляє про відмову у доступі («*Permission denied*»). Проте, *записати* цей файл, перенаправляючи видачу ***date –u*** виявляється можливим тому, що доступ на запис не закритий. Якщо ж закрити доступ на запис, а доступ на читання відкрити (ми зробили це в одному командному рядку, розділивши команди символом «***;***»), *неможливим* стане *зміна* цього файлу: спроба перенаправити вивід програми ***cal*** буде не успішною, а читання знову запрацює. Спрацює і *знищення* цього файлу, хоча ***rm***, на всяк випадок попередить про те, що файл захищено від запису.

Доступ до файлу на *виконання* означає можливість запускати цей файл у якості програми, виконати його. Наприклад, з каталогу ***/bin*** (у тому числі ***/bin/ls***,

/bin/rm, ***/bin/cat***, ***/bin/echo*** і ***/bin/data***) – **виконуються**, тобто доступні на виконання, і від того їх можна застосовувати у командному рядку у якості команд. У загальному випадку, необхідно вказати шлях до програми, наприклад, ***/bin/ls***, одначе, програми, які знаходяться у каталогах, перерахованих у змінній оточення **РАНТ**, можна викликати просто по імені: ***ls***.

5.2.3 Сценарій

Файли, які виконуються у Linux, бувають двох типів.

Перший - це файли у форматі, який особисто виконується (***executable***). Як правило, такі файли – результат *компіляції* програм, написаних на класичних мовах програмування, на шталт ***Ci***. Спроба прочитати такий файл за допомогою, наприклад, cat не призведе ні до чого корисного: на екран полізуть різноманітні беззмістовні символи, у тому числі, і управляючі. Це – так звані, *машинні коди*, мова зрозуміла тільки комп'ютеру.

У Linux виконуються декілька форматів файлів, які виконуються. Вони складаються із машинних кодів і службової інформації, необхідної операційній системі для запуску програми: згідно цієї інформації, ядро Linux виділяє для програми, що запускається, оперативну пам'ять, завантажує програму з файлу і передає їй управління. Більшість утиліт Linux – програми саме такого «*двійкового*» формату.

Другий вид файлів, що виконуються, – ***сценарії***. ***Сценарій*** – це *текстовий* файл, призначений для обробки якою-небудь утилітою. Частіше за все, така утиліта – це *інтерпретатор* деякої мови програмування, а вміст файлу – програма на цій мові. Ми уже писали один сценарій для ***sh***: програму ***loop***, яка нескінченно виконується. Оскільки, до того часу, ми ще не знали, як користуватися ***chmod***, нам кожний раз приходилося *явно* вказувати інтерпретатор (***sh*** або ***bash***), а сценарій передавати йому у вигляді параметру (див. приклади у розділі 5.1.).

Сценарій – файловий текст, який виконується. Для виконання сценарію необхідна програма-

інтерпретатор, шлях до якої може бути указано на початку сценарію у вигляді

«#! шлях_до інтерпретатора». Якщо інтерпретатор не задано, ним вважається ***/bin/sh***.

Якщо ж зробити файл тим, що виконується, то ту ж саму процедуру – запуск інтерпретатора і передача йому сценарію у якості параметру командного рядку – буде виконувати система:

Приклад 5.11 Утворення простішого сценарію, який виконується.

```
[methody@localhost methody]$ cat > script
echo 'Hello, Methody!'
^D
```

```
[methody@localhost methody]$ ./script
-bash: ./script: Permission denied
[methody@localhost methody]$ sh script
Hello, Methody!
[methody@localhost methody]$ chmod +x script
[methody@localhost methody]$ ./script
Hello, Methody!
```

З першого раз нам не вдалося запустити сценарій ***script***, тому, що за замовченням, файл створюється доступним на запис і на читання, але не на виконання. Після ***chmod +x*** файл став тим, що виконується. Оскільки, наш домашній каталог не входив у ***PATH***, прийшлося використовувати ***шлях*** до сценарію, який було утворено. Добре, що шлях виявився не довгим: «***поточний_каталог/ім'я_сценарію***», тобто ***./script***.

Якщо системі не натякнути спеціально, у якості інтерпретатору вона запускає стандартний – ***/bin/sh***. Одначе, є можливість написати сценарій для *любої* утиліти, у тому числі і написаної самостійно. Для цього, першими двома байтами сценарію повинні бути символи «***#!***», тоді, весь перший рядок, починаючи з третього байту, система сприйматиме як *команду* обробки. Виконання такого сценарію призведе до запуску указаної після «***#!***» команди, останнім параметром якої добавиться ім'я самого файлу сценарію.

Приклад 5.12 Створення sort-сценарію.

```
[methody@localhost methody]$ cat  >  to.sort
#!/bin/sh
some
unsorted
lines
^D
[methody@localhost methody]$ sort  to.sort
#!/bin/sh
lines
some
unsorted
[methody@localhost methody]$ chmod  + x  to.sort
[methody@localhost methody]$ ./to.sort
#!/bin/sort
lines
some
unsorted
```

Утиліта ***sort*** сортує – розставляє компоненти у алфавітному, зворотному алфавітному і іншому порядку – рядки файлу, який їй передається. Те ж саме відбудеться, якщо зробити цей файл таким, що виконується, написавши на початку ***/bin/sh*** у якості інтерпретатора., а потім виконати сценарій, що вийшов: запуститься утиліта ***sort***, а сценарій (файл з не сортованими рядками) передається їй у якості параметру. Оформлювати файли, які необхідно відсортовувати у вигляді sort-сценаріїв досить безглуздо, просто ми хотіли ще раз запевнитися, що сценарій можна писати для чого завгодно.

5.2.4 Доступ до каталогу

На відміну від файлу, новий каталог утворюється (за допомогою ***mkdir***) доступним на читання, запис і на виконання. Суть усіх трьох видів доступу менше очевидна ніж суть доступу до файлу. Коротко вона така: доступ по читанню – це можливість *переглянути вміст* каталогу (список файлів), доступ по запису – це можливість *змінювати вміст* каталогу, а доступ на виконання – можливість *скористатися* цим вмістом: *по-перше*, зробити цей каталог ***поточним***, а *по-друге*, ***звернутися за доступом*** до файлу, який є в ньому.

Приклад 5.13 Доступ до каталогу на читання і виконання.

```
 [methody@localhost methody]$ mkdir  dir
[methody@localhost methody]$ date  >  dir/smallfile
[methody@localhost methody]$ /bin/ls dir
smallfile
[methody@localhost methody]$ cd  dir
[methody@localhost dir]$ pwd
/home/methody/dir
[methody@localhost dir]$ cd
[methody@localhost methody]$ pwd
/home/methody
[methody@localhost methody]$ cat  dir/smallfile
Срд Сен 22 13:15:20 MSD 2018
```

Ми утворили каталог ***dir*** і файл ***smalfile*** у ньому. При заміні поточного каталогу, ***bash*** автоматично змінив рядок-підказку: як було сказано у лекції 4, останнє слово цієї підказки – ім’я поточного каталогу. Команда ***pwd*** (***p****rint* ***w****ork* ***d****irectory*), яка описана у тій же лекції, підтверджує це. Команда ***cd*** *без* параметрів, як це їй і покладається, робить поточним ***домашній каталог*** користувача.

Приклад 5.14 Доступ до каталогу на читання без виконання.

```
[methody@localhost methody]$ chmod  -x  dir
[methody@localhost methody]$ ls  dir
```

```
ls: dir/smallfile: Permission denied
[methody@localhost methody]$ alias  ls
alias ls='ls --color=auto'
[methody@localhost methody]$ /bin/ls dir
smallfile
[methody@localhost methody]$ cd  dir
-bash: cd: dir: Permission denied
[methody@localhost methody]$ cat  dir/smallfile
cat: dir/smallfile: Permission denied
```

Ми заборонили доступ на використання каталогу, очікуючи, що переглянути його вміст за допомогою команди ***ls*** буде можливо, а зробити його поточним і добути вміст файлу, який у ньому знаходиться – не можна. Несподівано, команда ***ls*** видала помилку. Ми помітили, що *ім'я файлу* – ***smallfile***, команда ***ls*** все-таки добула, але їй чогось знадобився і сам файл. Інструкція порадила подивитися, що дає команда ***alias ls***. Виявляється, замість простого ***ls*** розумний ***bash*** підставляє специфічну для Linux команду ***ls --color=auto***, яка розмальовує імена файлів у різні кольори у залежності від їх типу. Для того, щоби визначити тип файлу (наприклад, чи запускається він) необхідно отримати до нього доступ, а цього зробити не можна *використати* каталог. Якщо явно викликати утиліту ***ls*** (***/bin/ls***), поведінка каталогу становиться повністю передбаченою. Детальніше про те, чим займається команда ***alias*** (про **скорочення**), ми обговоримо далі.

Приклад 5.15 Доступ до каталогу на виконання без читання.

```
[methody@localhost methody]$ chmod  +x  dir;   chmod -r dir
[methody@localhost methody]$ /bin/ls dir
/bin/ls: dir: Permission denied
[methody@localhost methody]$ cat  dir/smallfile
Срд Сен 22 13:15:20 MSD 2004
[methody@localhost methody]$ cd  dir
[methody@localhost dir]$ /bin/ls
ls: .: Permission denied
[methody@localhost dir]$ cd
[methody@localhost methody]$
```

Якщо каталог, доступний на читання, але не доступний на використання, мало коли потрібен, то каталог, доступний на використання, але не на читання, може згодитися. Як видно з прикладу, отримати список файлів, які знаходяться у такому каталозі, не вдасться, але отримати доступ до самих файлів, *знаючи* їх імена – можна. Ми могли би спробувати покласти в каталог, який створили, який-небудь потрібний, але дуже секретний файл, щоби ім'я цього файлу, крім

близьких друзів, ніхто не знав. Але відмовимося від цієї затії: по-перше, знаючи, що адміністратор (***супер користувач***) все рівно зможе переглянути вміст такого каталогу, а по-друге, не знайшлося такого секретного файлу.

Приклад 5.16. Рекурсивне видалення каталогу.

```
[methody@localhost methody]$ rm  -rf  dir
rm: невозможно открыть каталог ‘dir’: Permission denied
[methody@localhost methody]$ chmod  -R  +rwX  dir
[methody@localhost methody]$ rm  -rf  dir
```

Ми вирішили знищити каталог ***dir***. Ми можемо це зробити не за допомогою ***rmdir***, а за допомогою ***rm*** з ключем «***-r***» (***recursive***). Але з ходу використати ***rm*** не вдалося: читання з каталогу не можливе, і тому, не відомо, які файли там знищувати. Тому, спочатку треба дозволити усі види доступу до ***dir***. На всякий випадок (а раптом у середині ***dir*** попадеться такий же каталог. Який не читається). Ми виконуємо рекурсивний варіант ***chmod*** – з ключем «***-R***» («***R***» тут велике а не мале, тому що «***-r***» уже зайнято: означає заборону на читання).

Команда «***chmod -R +rwX dir***» робить усі файли і каталоги у dir доступними на читання і запис; і усі каталоги доступними на використання. Параметр «***X***» (великий) встановлює доступ на використання файлу, тільки якщо цей доступ уже був дозволений *хоч кому-небудь* (хазяїну, групі або усім іншим). Він дозволяє не робити усі підряд файли, щоби запускалися. Але це – не важливе, якщо наступна команда буде ***rm***.

6 ПРАВА ДОСТУПУ В ОС LINUX

6.1 Права доступу до файлів і каталогів

Увага!

Між ALT і MINT оточеннями Linux є не принципові різниці, тому ті частини прикладів, які очікуються бути в зміненому вигляді, або будуть відсутні ми підкреслюємо. Теж саме ми робимо з тими об'єктами, які описуємо у посібнику. Слід пам'ятати, що вихід з каналу ^D

Працюючи з Linux, ми помітили, деякі файли і каталоги не доступні нам ні на читання, ні на запис, ні на використання. Навіщо такі файли потрібні? Виявляється, інші користувачі *можуть* звертатися до цих файлів, а у нас просто не *вистачає прав*.

6.1.1 Ідентифікатор користувача

Говорячи про *права* доступу користувача до файлів, слід помітити, що в дійсності маніпулює файлами не сам користувач, а запущений ним процес (наприклад, утиліта ***rm*** або ***cat***). Оскільки, і файл, і процес утворюються і управляються системою, їй не важко яку завгодно політику доступу одних до других, базуючись на любих властивостях процесів як *суб'єктів* і файлів як *об'єктів* системи.

В Linux, одначе, використовуються не які завгодно властивості, результат ***ідентифікації*** користувача – його ***UID***. Кожний процес системи обов'язково належить якому-небудь користувачеві, і ***ідентифікатор користувача*** (***UID***) – обов'язкова властивість любого процесу Linux. Коли програма ***login*** запускає ***стартовий командний інтерпретатор***, вона приписує йому ***UID***, отриманий у результаті діалогу. Звичайний запуск програми (***exec***()) або народження нового процесу (***fork***()) не змінюють ***UID*** процесу, тому усі процеси, які запущені користувачем під час термінальної сесії, будуть мати його ідентифікатор.

Оскільки ***UID*** однозначно визначається вхідним іменем, воно не рідко використовується *замість* ідентифікатора – для наявності. Наприклад, замість виразу «ідентифікатор користувача, відповідний вхідному імені ***methdy***», говорять «***UID methdy***» (у прикладах нижче цей ідентифікатор дорівнює ***503***).

Приклад 6.1 Знаходження ідентифікатори користувача та членство в групах

```
[methody@localhost methody]$ id
uid=503(methody) gid=503(methody) группы=503(methody)
[methody@localhost methody]$ id shogun
uid=400(shogun)                                    gid=400(shogun)
группы=400(shogun),4(adm),10(wheel),19(proc)
```

Утиліта ***id***, якою ми скористалися, виводить наше **вхідне ім'я** і відповідний нам ***UID***, а також **групу за замовченням** і повний список **груп**, членами яких ми є.

6.1.2 Ярлики об'єктів файлової системи

При *утворенні* об'єктів файлової системи – файлів, каталогів і тому подібне, – кожному, обов'язковому порядку приписується **ярлик**. Ярлик включає в себе ***UID*** ідентифікатор користувача-*хозяїна* файлу, ***GID*** – ідентифікатор групи, до якої належить файл, тип об'єкту і набір, так званих, **атрибутів**, а також деяку додаткову інформацію. **Атрибути** визначають, хто і що з файлом має право робити, і описані нижче.

Приклад 6.2 Права доступа до файлів и каталогів

```
[methody@arnor methody]$ ls -l
итого 24
drwx-------  2  methody methody  4096     Сен 12   13:58   Documents
drwxr-xr-x  2  methody methody  4096     Окт 31   15:21   examples
-rw-r--r--     1  methody methody      26      Сен 22   15:21   loop
-rwxr-xr-x   1  methody methody      23      Сен 27   13:27   script
drwx------    2  methody methody  4096     Окт  1   15:07   tmp
-rwxr-xr-x   1  methody methody      32      Сен 22   13:26   to.sort
```

Ключ «***-l***» утиліти ***ls*** визначає «довгий» (***long***) формат видачі (справа на ліво): ім'я файлу, час останньої зміни файлу, розмір файлу у байтах, група,хазяїн, ***кількість жорстких посилань*** і рядок ***атрибутів***. Перший символ (зліва направо) у рядку атрибутів визначає тип файлу. Тип «-»відповідає «звичайному» файлу, а тип «***d***» – каталогу «***d****irectory*». Ім'я користувача і групи, яким належить вміст нашого домашнього каталогу, природно, ***methody.*** Нас зацікавило ось що: не дивлячись на те, що утворення *жорстких посилань* на каталог не можливо, поле «кількість жорстких посилань» для *усіх* каталогів прикладу дорівнює *двом*, а не одному. Насамперед, цього і слід було очікувати, тому, що *любий* каталог файлової системи Linux завжди має не менше двох імен: особисте (наприклад, ***tmp***) і ім'я «**.**» у самому каталозі (**tmp/.**). Якщо у каталозі утворити підкаталог, кількість жорстких посилань на цей каталог збільшиться на 1 за рахунок імені «**..**» у підкаталозі (***tmp/subdir1/..***):

Приклад 6.3. Декілька жорсткіх послань на каталог

```
[methody@arnor methody]$ ls -ld tmp
drwx------ 3 methody methody 4096 Окт 1 15:07 tmp
[methody@arnor methody]$ mkdir tmp/subdir2
[methody@arnor methody]$ ls -ld tmp
drwx------ 4 methody methody 4096 Окт 1 15:07 tmp
[methody@arnor methody]$ rmdir tmp/subdir*
[methody@arnor methody]$ ls -ld tmp
drwx------ 2 methody methody 4096 Окт 1 15:07 tmp
```

Тут ми використали ключ «***-d***» (directory) для того, щоби ***ls*** виводив інформацію про зміст каталогу ***tmp***, а про самий каталог. Приклад від рута е сенс наводити тільки у тому випадку, якщо приклад на щось універсальне, що

обов'язково буде влаштовано точно так, як у любому Лінуксі. Інакше, користувач почне мудрувати і наробить помилок. В Linux визначено декілька ***системних груп***, завдання яких – забезпечити доступ членів цих груп до різноманітних ресурсів системи. Часто такі групи носять назви: «***disk***», «***audio***», «***cdwriter***», тощо. Тоді, звичайним користувачам доступ до деякого файлу, каталогу або файлу-дирці Linux буде закрито, але відкрито членам групи, до якої цей об'єкт належить.

Наприклад в Linux майже завжди використовується ***віртуальна файлова система /proc*** – каталог, у якому у вигляді підкаталогів і файлів подана інформація з ***таблиці процесів***. Ім'я підкаталогу ***/proc*** співпадає з ***PID*** відповідного процесу, а вміст цього каталогу відображає властивості процесу. ***Хазяїном*** такого підкаталогу буде хазяїн процесу (з правами на читання і використання), тому *любий* користувач може продивитися інформацію про свої процеси. Саме каталогом ***/proc*** користується утиліта ***ps***.

Приклад 6.4 Обмеження доступу до повної таблиці процесів

```
[methody@arnor methody]$ ls -l /proc
. . .
dr-xr-x--- 3 methody proc 0 Сен 22 18:17 4529
dr-xr-x--- 3 shogun proc   0 Сен 22 18:17 4558
dr-xr-x--- 3 methody proc 0 Сен 22 18:17 4589
. . .
[methody@localhost methody]$ ps -af
UID         PID  PPID   C  STIME  TTY TIME       CMD
methody  4529   4523   0   13:41     tty1 00:00:00   –bash
methody  4590   4529   0   13:42     tty1 00:00:00    ps –af
```

Виявляється, запущено не мало процесів, у тому числі, один – користувачем ***shogun*** (***PID 4558***). Одначе, не дивлячись на ключ «***-a***» (***all***), ***ps*** видала нам тільки відомості про наші процеси: ***4529*** – це вхідний shell, а ***4589*** – мабуть, сам ***ls***.

Інша справа – адміністратор, він, як видно з прикладу 6.1., входить у групу ***/proc***.

Приклад 6.5 Доступ до повної таблиці процесів: група proc

```
shogun@localhost ~ $ ps -af
UID         PID    PPID  C  STIME  TTY  TIME     CMD
methody  4529   4523   0   13:41    tty1  00:00:00  -bash
shogun    4558   1828  0   13:41    tty3  00:00:00  -zsh
shogun    4598   4558  0   13:41    tty3  00:00:00  ps –af
```

Адміністратор – досвідчений користувач Linux, надає перевагу ***bash-y*** «***The Z Shell***», ***zsh***. Звідси і відміни запрошення у командному рядку. В усіх shell-ах, крім самих старих, символ «**~**» означає домашній каталог. Цим скороченням зручно користуватися, якщо поточний каталог – не домашній, а звідти (або туди)

треба скопіювати файл. Маємо команду на шталт «***ср ~/ необхідний файл .***» або «***ср необхідний файл ~***» відповідно.

Команда ***ps -a*** виводить інформацію про *всі* процеси запущені «живими» (а не системними користувачами). Для перегляду усіх процесів адміністратор запускає ***ps –efH*** (використати у лабораторній роботі).

6.1.4 Каталоги, що розділяються

Проаналізувавши систему прав доступу до каталогів у Linux ми прийшли до висновку, що в ній є суттєвий недолік. Той, хто має право змінювати каталог, може *знищити любий* файл у ньому, навіть такий до якого зовсім немає доступу. Формально усе вірно: знищення файлу з каталогу – всього лише зміна вмісту каталогу. Якщо у файла було більше одного імені (існувало декілька жорстких посилань на цей файл), не якого знищення даних *не буде*, а якщо посилання було останнім – файл дійсно знищиться. Ось саме це, на нашу думку, погано. Щоби доказати нам, що право на знищення любих файлів *корисне*, від імені адміністратора було утворено у нашому домашньому каталозі файл, зовсім не допустимий, та ще до того ж, з підозрілим іменем.

Приклад 6.6 Знищення чужого файлу з незручним ім'ям.

```
[methody@localhost methody]$ ls
4TO-TO Mep3koe Documents examples loop script tmp to.sort
[methody@localhost methody]$ ls -l 4*
-rw------- 1 root root 0 Сен 22 22:20 4TO-TO Mep3koe
[methody@localhost methody]$ rm -i 4*
rm: удалить защищённый от записи пустой обычный файл‘4TO-TOMep3koe’? y
```

Примітка

Для виконання роботи, необхідно мати в каталозі ***/home/user1*** файл ***4TO-TO Mep3koe***, утворений від імені адміністратора.

Підозрюючи, від зломщиків усе можна очікувати ми вирішили не набирати ім'я файлу,який знищується, з клавіатури, а скористатися *шаблоном* і ключем «***-i***» (***i****nteractive*) команди ***rm***, щоби та очікувала підтвердження перед тим, як знищувати черговий файл. Не дивлячись на відсутність доступу до самого файлу, знищити його виявилося можливим.

Приклад 6.7 Робота з файлами в каталозі який розділяється

```
[methody@localhost methody]$ ls -dl /tmp
drwxrwxrwt 4 root root 1024 Сен 22 22:30 /tmp
[methody@localhost methody]$ ls -l /tmp
итого 4
-rw-r--r-- 1 root root          13 Сен  22  17:49  read.all
-rw-r----- 1 root methody    23 Сен  22  17:49  read.methody
-rw------- 1 methody root    25 Сен  22  22:30  read.Methody
-rw-r----- 1 root wheel       21 Сен  22  17:49  read.wheel
```

[methody@localhost methody]$ rm -f /tmp/read.*
rm: невозможно удалить ‘/tmp/read.all’: Operation not permitted
rm: невозможно удалить ‘/tmp/read.methody’: Operation not permitted
rm: невозможно удалить ‘/tmp/read.wheel’: Operation not permitted
[methody@localhost methody]$ ls /tmp
read.all read.methody read.wheel

Примітка

Необхідно знати утиліту:

chmod [**u** | **g** | **a** | **o**] [**+** | **-** | **=**] **file1** **file2 …**

u – хазяїн – користувач
g – хазяїн – група
o – інші користувачі
a – усі класи користувачів
+ - добавити
= - привласнити
- - знищити

Спрацьовує по такому сценарію, якщо у каталогу є атрибут «***t***»

Упевнившись, що *любий* доступ у каталог ***/tmp*** відкрито усім, у нас з’явилося бажання знищити там усі файли. Задумка значно більше небезпечна, ніж утворення файлу, а раптом, вони кому-небудь потрібні. Дивно, але вдалося знищити тільки файл, який належить нам.

Справа в тому, що ми не побачили одну особливість атрибутів каталогу ***/tmp***: замість «***x***» у трійці «для сторонніх» ***ls*** видав «***t***». Це – *ще* один атрибут каталогу, наявність якого якраз і забороняє користувачеві знищувати звідти файли, яким ми не хазяї. Таким чином, права запису у каталог з ярликом ***drwxrwxrwt група хазяїн*** і для членів групи *група*, і для сторонніх обмежені їх особистими файлами, тільки *хазяїн* має право змінювати список файлів у каталозі, як йому заманеться. Такі каталоги називаються тими, що **розділяються**, тому що призначені вони, як правило, для сумісної роботи усіх користувачів у системі, обміну інформацією тощо.

При встановленні атрибута «***t***» доступ для використання для сторонніх («***t***» у рядку атрибутів стоїть на місці останнього «***x***») не відміняється. Просто вони рідко використовуються один без одного, що виводить ***ls*** їх в одному і тому ж місці. Якщо кому-небудь прийде у голову організувати каталог, що розділяється *без* доступу стороннім на використання, ls виведе на місті дев’ятого атрибуту не «***t***», а «***T***».

Приклад 6.8. Робота з чужим файлом в своєму каталозі

[methody@localhost methody]$ ls -l loop
-rw-r--r-- 1 root root 26 Сен 22 22:10 loop
[methody@localhost methody]$ chown methody loop
chown: изменение владельца ‘loop’: Operation not permitted
[methody@localhost methody]$ cp loop loopt
[methody@localhost methody]$ ls -l loop*

-rw-r--r—1 root root 26 Сен 22 22:10 loop
-rw-r--r-- 1 methody methody 26 Сен 22 22:15 loopt
[methody@localhost methody]$ mv -f loopt loop
[methody@localhost methody]$ ls -l loop*
-rw-r--r-- 1 methody methody 26 Сен 22 22:15 loop

Примітка

Спочатку треба від адміністратора утворити файл ***loop*** у каталозі ***/home/user1***

Виявляється, незрозумілості продовжуються. Хтось замінив файлу ***loop*** хазяїна, що тепер ми можемо тільки читати його але на змінювати. Знищувати цей файл дуже просто, але хочеться повернути старі права на нього. Це нескладно: чужий файл можна перейменувати (це дія над каталогом а не файлом), скопіювати перейменований файл у файл з іменем старого (доступ на читання відкрито), на кінець, знищити чужий файл. Ключ «***-f***» (**f***orce*, «силоміць»)дозволяє утиліті ***mv*** робити свою справу, не питаючи підтверджень. Зокрема. Побачивши, що файл з іменем, в яке необхідно перейменувати існує, навіть чужий, навіть недоступний на запис, ***mv*** спокійно знищує його і виконує операцію перейменування.

6.2. Суперкористувач

Ми задумалися, дізнавшись, що дехто може пробляти з нами жарти, які сам користувач ні над ким пробляти не може. Обґрунтована підозра на єдиного адміністратора цієї системи, який має права суперкористувача.

Суперкористувач – єдиний користувач у Linux, на якого не поширюються ***права доступу***. Має ***нульовий ідентифікатор користувача***.

UID суперкористувацьких процесів дорівнює 0: так система їх від процесів інших користувачів. Саме суперкористувач має можливість вільно змінювати хазяїна і групу файлу. Йому відкритий доступ на читання і запис до любого *файлу* системи і доступ на читання і запис до любого *каталогу*. На кінець, суперкористувацьких процес може на деякий час змінити *свій особистий* ***UID*** з нульового на любий інший. Саме так діє програма ***login***, коли, провівши процедуру ідентифікації користувача, запускає ***стартовий командний інтерпретатор***.

Серед облікових записів Linux завжди є запис ***root*** («*корінь*»), який відповідає нульовому ідентифікатору, тому замість «*суперкористувач*» часто говорять «***root***». Велика кількість системних файлів належать ***root-у***, велика множина файлів тільки йому доступна на читання і запис. Пароль цього облікового запису – одна з самих великих цінностей системи. Саме з її допомогою системні адміністратори виконують саму відповідальну роботу. Властивість ***root*** мати доступ до *всіх* ресурсів системи, накладає дуже високі вимоги на *людину*, яка знає пароль ***root***. Суперкористувач моє все – в тому числі

і все зламати, тому *любу роботу* варто вести з правами звичайного користувача, а до прав ***root*** звертатися тільки за необхідністю.

Є *два* різних *засоби* отримати права суперкористувача. *Перший* – це зареєструватися у системі під цим іменем, ввести пароль і отримати стартову оболонку, яка має *нульовий* ***UID***. Це самий правильний засіб, користуватися яким варто, тільки якщо не можна застосовувати інші. Команда last дасть повідомлення, що з такої-то консолі в систему увійшов *невідомо хто* з правами *суперкористувача* і *щось* там *робив*. З точки зору системного адміністратора, це – дуже підозріла подія, особливо, якщо сам він в цей час не підходів до цієї консолі. Це – дуже небезпечний варіант.

Другий засіб – це скористатися спеціальною утилітою ***su*** (***s****hell of* ***u****ser*), яка дозволяє виконати одну або декілька команд від особи іншого користувача. По замовченню, ця утиліта виконує команду ***sh*** від особи користувача ***root***, тобто запускає командний інтерпретатор з нульовим ***UID***. Відміна від попереднього засобу в тому, що зажди відомо, *хто саме* запускав ***su***, а це означає, с кого спита тати за наслідки. У деяких випадках зручніше не користуватися ***su***, а використовувати утиліту ***sudo***, яка дозволяє виконувати *тільки раніше задані* команди.

6.3. Підміна ідентифікатору

Утиліти ***su*** і ***sudo*** мають деяку дивність, пояснити яку ми поки що не в стані. Ця ж дивність поширюється і на давно відому програму ***passwd***, яка дозволяє редагувати особистий ***обліковий запис***. Процес, який запускається, *спадкує* ***UID*** від батьківського, тому, якщо цей ***UID*** не нульовий, він не в стані *поміняти* його. Тоді, як же ***su*** запускає для звичайного користувача *сперкористувацький* ***shell***? *Я*к ***passwd*** отримує доступ до сховища *всіх* облікових записів? Повинен існувати механізм, який дозволяє користувачу запускати процеси з ідентифікаторами *іншого* користувача, причому, механізм повинен строго контролюватися, інакше з його допомогою можна наробити не мало бід.

В Linux цей механізм називається ***підміною ідентифікатора*** і об лаштований дуже просто. Процес може змінити свій ***UID***, якщо запустить замість себе , за допомогою ***exec()*** *іншу* програму з файлу, який має спеціальний атрибут ***SetUID***. В цьому випадку ***UID*** *процесу* стає рівним ***UID*** *файлу*, з якого програма була запущена.

Приклад 6.9. Звичайна программа passwd, яка використовує SetUID

```
[foreigner@somewhere foreigner]$ ls  -l  /usr/bin/passwd   /bin/su
-rws--x--x 1 root root 19400  Фев 9 2018 /bin/su
-rws--x--x 1 root root 5704   Янв 18 2018 /usr/bin/passwd
[foreigner@somewhere foreigner]$ ls -l  /etc/shadow
-r--------    1 root root 5665  Сен 10 02:08 /etc/shadow
```

Як і у випадку з ***t-атрибутом, ls*** виводить букву «***s***» замість «***x***» у трійці «*для хазяїна*». Точно також, якщо відповідного ***x-артрибуту*** немає (що буває дуже рідко), ***ls*** виводить «***S***» замість «***s***». У багатьох дистрибутивах Linux і ***/bin/su***, і ***/usr/bin/passwd*** мають встановлений ***SetUID*** і користувачу ***root***, що дозволяє ***su*** запускати процеси з правами цього користувача (це означає і любого іншого), а ***passwd*** – модифікувати файл ***/etc/shadow***, який вміщує у таких системах відомості про *усі* облікові записи.

Як правило, ***SetUID***-ні файли доступні звичайним користувачам тільки на виконання, щоби не провокувати цих звичайних користувачів розглядати вміст цих файлів і досліджувати їхні *не документовані можливості*. Якщо винайдеться спосіб заставити, припустимо, програму ***passwd*** *виконувати* любу іншу програму, то усі проблеми захисту системи від злому будуть разом вирішені – немає захисту, немає і проблем.

Одначе, ми працюємо з такою системою, де ***/usr/bin/passwd*** взагалі не має атрибута ***SetUID***. Зате ця програма належить до *групи* ***shadow*** і має інший атрибут, ***SetGID***, так що при її запуску процес отримує ідентифікатор групи ***shadow***. Утиліта ***ls*** виводить ***SetGID*** у вигляді «***s***» замість «***x***» другій тройці атрибутів («*для групи*»). Примітки відносно «***s***», «***S***» і «***x***» дійсні для ***SetGID*** також, як для ***SetUID***.

Приклад 6.10. Программа, яка не пядлягає взлому passwd використовує SetGID

```
[root@localhost root]# ls -l /usr/bin/passwd
-rwx--s--x 1 root shadow 5704 Jan 18 2004 /usr/bin/passwd
[root@localhost root]# ls -al /etc/tcb/methody
total 3
drwx--s---  2 methody  auth      1024 Sep 22 12:58 .
drwx--x--- 55 root     shadow    1024 Sep 22 18:41 ..
-rw-r-----  1 methody  auth        81 Sep 22 12:58 shadow
-rw-------  1 methody  auth         0 Sep 12 13:58 shadow-
-rw-------  1 methody  auth         0 Sep 12 13:58 shadow.lock
```

Примітка

В Linux mint, політика безпеки дещо відрізняється від політики безпеки у Alt Linux, тому, каталог **tcb** відсутній.

Далі опишемо це питання безпеки відносно Alt Linux. Каталог **/etc/tcb/** у цій системі вміщує підкаталоги які відповідають іменам усіх користувачів. В кожному підкаталозі зберігається, разом з іншим, *особистий* файл ***shadow*** відповідного користувача. Доступ до каталогу **/etc/tcb/** на *використання* (а отже, і до усіх його підкаталогів) мають, крім ***root***, тільки члени групи ***shadow***. Доступ на *запис* до каталогу ***/etc/tcb/methody*** і до файлу **/etc/tcb/*methody*/*shadow*** має тільки користувач ***methody***. Це означає, щоби змінити що-небудь у цьому файлі процес *зобов'язаний* мати ***UID methody*** і ***GID shadow*** (або нульовий UID, звісно). Саме такий процес запускається з ***/usr/bin/passwd***: він спадкує ***UID*** у

командного інтерпретатора і отримує ***GID shadow*** з атрибуту ***SetGID***. Це означає, що навіть знаходячи у програмі ***passwd*** помилки і примусивши її зробити що завгодно, зловмисник зможе тільки відредагувати *свій особистий* обліковий запис.

Виявляється, ***SetGID*** можна присвоювати *каталогам*. В останньому прикладі каталог ***/etc/tcb/methody*** мав ***SetGID***, тому усі файли, які в ньому утворюються, получають ***GID***, який дорівнює ***GID*** самого каталогу ***auth***. Використовується це різноманітними процесами ідентифікації, які не є суперкористувацькими, але входять в групу ***auth*** і ***shadow***, і можуть прочитати інформацію з файлів ***/etc/tcb/вхідне_ім'я/shadow***.

Цілком очевидно, що підміна ідентифікатора *поширюється* на програми, але не працює для *сценаріїв*. Дійсно, при виконанні сценарію, *процес* запускається не з нього, а з програми-інтерпретатора. Файл із сценарієм передається всього лише як параметр командного рядка, і переважна більшість інтерпретаторів просто не звертають уваги на атрибути цього файлу. Інтерпретатор спадкує ***UID*** у батьківського процесу, так що, якщо він і виявить ***SetUID*** у сценарію, зробити він нічого не зможе.

6.1.7 Вісімкова уява атрибутів

Усі дванадцять атрибутів можна представити у вигляді бітів двійкового числа, рівних ***1***, якщо атрибут встановлено, і ***0***, якщо ні. Порядок бітів у числі наступний: ***sU|sG|t|rU|wU|xU|rG|wG||xG|rO|wO|xO***, де ***sU*** – це ***SetUID***, sG – це ***SetGID***, ***t*** – це ***t*** *–атрибут*, після чого ідуть три трійки атрибутів доступу. Цім форматом можна скористатися у команді ***chmod*** замість конструкції «*ролі=види_доступу*», причому число треба записувати у вісімковій *системі числення*. Так, атрибути каталогу ***/tmp*** будуть рівні ***1777***, атрибути ***/bin/su*** – ***4711***, атрибути ***/bin/ls*** – ***755*** тощо.

Той же самою побітовою уявою атрибутів регулюються і права доступу по замовченню при *утворення* каталогів і файлів. Робиться це за допомогою команди ***umask***. Єдиний параметр ***umask*** – вісімкове число, яке задає атрибути, які *не треба* встановлювати новому файлу або каталогу. Так ***umask 0*** призведе до того, що файли будуть утворюватися з атрибутами «***rw-rw-rw-***», а каталоги «***rwxrwxrwx***». Команда ***umask 022*** забирає з атрибутів по замовченню права доступу на запис для усіх, крім хазяїна (отримуємо «***rw-r--r--***» і «***rwxr-xr-x***» відповідно), а ***umask 077*** нові файли і каталоги стають для них повністю не доступні. («***rw------***» і «***rwx------***»).

7 РОБОТА З ТЕКСТОВИМИ ДАНИМИ

7.1 Введення і виведення даних

Люба програма – це автомат призначений для обробки *даних*: отримуючи на вхід одну інформацію вона в результаті роботи видає деяку іншу. Хоча інформація, яка входить або виходить, може бути нульовою, тобто просто бути відсутньою. Ті дані, які передаються програмі для обробки – це її ***введення*** те, що вона видає у результаті роботи – ***вивід***. Організація введення і виведення для кожної програми – це задача операційної системи.

Кожна програма працює з даними визначеного типу: текстовими, графічними, звуковими і тому подібне. Як мабуть вже стало зрозуміло з попередніх лекцій, основний інтерфейс управління системою в Linux – це ***термінал,*** який призначений для передавання текстової інформації від користувача системі і назад. Оскільки ввести з терміналу і вивести на термінал можна тільки текстову інформацію, то введення і виведення програм, пов'язаних з терміналом, теж повинно бути текстовим.

Одначе, необхідність оперувати з текстовими даними не обмежує можливості управління системою, а навпаки розширює їх. Людина може *прочитати* вивід любої програми і розібратися, що відбувається у системі, а різні програми виявляються сумісними між собою, оскільки використовують один і той же вид подання даних – текстовий. Можливостям, які дає Linux при роботі з даними у тестовому форматі, і присвячена дана лекція.

«Текстова форма» даних всього лише *домовленість* про їхній формат. Ніхто не заважає виводити на екран не текстовий файл, одначе, користі від цього буде мало. По перше, раз файл вміщує не текст, то не передбачається, що людина зможе щось зрозуміти з його вмісту. По друге, якщо у нетекстових даних, які виводяться на термінал, *випадково* зустрінеться управляюча послідовність, термінал її виконає. Наприклад, якщо у скомпільованій програмі записано число ***1528515121***, воно представлено у вигляді чотирьох байтів: ***27***, ***91***, ***49*** і ***74***. Відповідний їм текст складається з чотирьох символів ***ASCII***: «***ESC***», «***[***», «***1***» і «***J***», при виведенні файлу на віртуальну консоль Linux в цьому місці виконається очищення екрану, так як «^***[[1J***» саме така управляюча послідовність для віртуальної консолі. Не усі управляючі послідовності такі нешкідливі, тому використовувати нетекстові дані у якості текстів не рекомендується.

Що ж робити, якщо вміст нетекстового файлу все тати бажано *переглянути* (тобто перетворити на *текст*)? Можна скористатися програмою ***hexdump***, яка дає вміст файлу у вигляді шістнадцятирічних ASCII-кодів, або ***strings***, яка показує тільки ті частини файлу, що можуть бути подані у вигляді тексту:

Приклад 7.1 Використання hexdump и strings

```
[methody@localhost methody]$ hexdump -C /bin/cat  | less
```

```
00000000  f 45 4c 46 01 01 01 00   00 00 00 00 00 00 00 00 | . ELF. . . . . .
. |
00000010  02 00 03 00 01 00 00 00  90 8b 04 08 34 00 00 00| . . . . . .
.4.. |
00000020  e0 3a 00 00 00 00 00 00  34 00 20 00 07 00 28 00 |Ю : . . . . . 4. . .
( . |
. . .
00000100  00 00 00 00 00 00 00 00  00 00 00 00 06 00 00 00| . . . . . .
. . |
00000110  04 00 00 00 2f 6c 69 62   2f  6c 64 2d 6c 69 6e 75| . . . . / l i b/ l d-l i n
u|
00000120  78 2e 73 6f 2e 32 00 00 04 00 00 00 10 00 00 00 | x. so.2 . . . . . .
. |
00000130 01 00 00 00 47 4e 55 00 00 00 00 00 02 00 00 00| . . . .GNU . . . . . .
. . |
. . .
[methody@localhost methody]$ strings /bin/cat  | less
 /lib/ld-linux.so.2
 _Jv_RegisterClasses
_ _ gmon_start_ _
libc.so.6
stdout
. . .
[methody@localhost methody]$ strings  -n3 /bin/cat  | less
/lib/ld-linux.so.2
GNU
_Jv_RegisterClasses
_ _ gmon_start_ _
libc.so.6
stdout
. . .
```

У прикладі ми знаючи наперед, що тексту видано багато, використали ***конвеєр | less***», більш детальніше він описаний далі. З ключем «***-C***» утиліта ***hexdump*** виводить у правому боці екрану текстову уяву даних, замінюючи символи, які не друкуються, крапками (щоби серед тексту, який виводиться, не зустрілося управляючої послідовності). Ми помітили, що ***hexdump*** «*не знайшла*» в файлі ***/bin/cat*** явно текстових під рядків «***ELF***» і «***GNU***»: перший з них – взагалі не текст (перед нею стоїть символ, який не друкується, ***7f***, а після – символ з кодом ***1***), а другий – дуже короткий, хоча і обмежений символами з кодом ***0***, як це і передбачається рядку у скомпільованій програмі. Найменша довжина рядка передається ***strings*** ключем «***-n***».

Примітка

– ***head*** – програма друку «***N***» рядків з файлу у терміналі.

– ***cat (concatenate)*** – зціплює (пов'язує) файли і тому і подібне, може виводити вміст файлу на екран.

7.2 Перенаправлення введення і виведення

Для того, щоби записати дані у *файл* або прочитати їх звідти, процесу необхідно спочатку відкрити цей файл (при відкритті на запис, можливо, попередньо створити його). При цьому процес отримує **дескриптор** (списник) відкритого файлу – унікальне для цього процесу число, яке він і буде використовувати у всіх операціях запису. Перший відкритий файл отримає дескриптор *0*, другий – *1* і так далі. Закінчивши роботу з файлом, процес *закриває* його, при цьому дескриптор звільнюється і може бути використаний вдруге. Якщо процес завершується не закривши файли, за нього це робить система. Строго кажучи, тільки в операції відкриття дескриптора вказується, який саме файл буде використовуватися. У якості «файлу» і звичайні файли і файли-дирки (частіше за все - термінали), і **канали**, які обговоримо далі. Подальші операції – читання, запис і закриття, працюють з дескриптором, як з *потоком даних*, а куди саме веде цей потік, не важливо.

Кожний процес Linux отримує на старті *три* «файли», відкритих для нього системою.

Перший з них (дескриптор 0) *відкрито* на читання, це ***стандартне введення*** процесу. Саме, із стандартним введенням працюють всі операції читання, якщо у них не вказано дескриптор файлу.

Другий (дескриптор 1) – відкритий на запис, це – ***стандартне виведення*** процесу. З ним працюють усі операції запису, якщо дескриптор файлу не вказано у них явно.

На кінець, третій потік (дескриптор 2) передбачається для виводу діагностичних повідомлень, він називається ***стандартне виведення помилок***.

Оскільки ці три дескриптори вже відкрити до моменту запуску процесу, перший файл, відкритий самим процесом, буде, скоріш за все, мати дескриптор 3.

Дескриптор – описувач потоку даних відкритого процесом. Дескриптори нумеруються починаючи з 0. При відкритті нового потоку даних, його дескриптор отримує найменший з номерів, який у даний момент не використовується. Три наперед відкритих дескриптори: **стандартне введення** (0), стандартне виведення (1) і **стандартне виведення помилок** (2) процесу видаються при запуску.

Механізм копіювання оточення , описаний у лекції 5, має на увазі, у числі іншого, копіювання усіх відкритих дескрипторів батьківського процесу дочірньому. У результаті, і батьківський і дочірній процес мають під однаковими

дескрипторами одні і ті ж потоки даних. Коли запускається ***стартовий командний інтерпретатор***, усі три раніше відкритих дескриптори пов'язані у нього з *терміналом* (точніше, з відповідним файлом-диркою типу ***tty***): користувач вводить команди з клавіатури і бачить повідомлення на екрані. Отже, люба команда, яка запускається з командної оболонки, буде виводити на той же термінал, а люба команда, яка запущена *інтерактивно* (не у фоні) – вводить звідти.

7.2.1 Стандартне виведення

Ми вже знаємо, що деякі програми вміють виводити не тільки на термінал, але і у файл, наприклад, ***info*** при вказівці *параметричного ключа* «***-o***» з іменем файлу, виведе текст керівництва у файл, замість того, щоби відображати його на моніторі. Навіть, якщо розробниками програми не передбачено такий ключ, відомий і інший засіб зберегти вивід програми у файлі замість того, щоби виводити його на монітор: поставити знак «>» і вказати після нього ім'я файлу. Таким чином ми вже утворювали короткі текстові файли (сценарії) за допомогою утиліти ***cat*** (Лекція 5).

Приклад 7.2 Пере направлення стандартного виведення у файл

```
[methody@localhost methody]$ cat > textfile
Это файл для примеров.
^D
[methody@localhost methody]$ ls -l textfile
-rw-r--r--    1 methody methody 23 Ноя 15 16:06 textfile
```

Від використання символу «>» можливості самої утиліти **cat**, зазвичай, не розширилися. Більше того, ***cat*** у цьому прикладі не від командної оболонки ніяких параметрів: ні знака «>», наступного за ним імені файлу. У цьому випадку ***cat*** працювала як зазвичай, не знаючи, куди попадуть виведені дані на екран монітору, у файл чи куди ще. Замість того, щоби самій забезпечувати доставку виведення до кінцевого адресату, ***cat*** відправляє усі дані на ***стандартне виведення*** (скорочено - ***stdout***).

Підміна стандартного виведення – завдання командної оболонки (***shell***). У даному прикладі ***shell*** утворює пустий файл, ім'я якого вказано після знаку «>», і дескриптор цього файлу передається програмі ***cat*** під номером ***1*** (***стандартне виведення***). Робиться це просто В *Лекції 5* було розказано, як запускаються команди з командної оболонки. Зокрема, після виконання ***fork()*** з'являються два однакових процеси, один з яких – дочірній – повинен запустити замість себе команду (виконати ***exec()***). Так ось, перед цим він *закриває* стандартне виведення (дескриптор 1 звільнюється) і відкриває файл (з ним зв'язується *перший* вільний дескриптор, тобто 1), команді, яка запускається, нічого знати і не потрібно: її стандартне виведення вже підмінене. Ця операція називається *перенаправленням*

стандартного виведення. У тому випадку, якщо файл уже є, ***shell*** запише його заново, повністю знищивши все, що в ньому вміщувалося до цього. Тому нам, щоби продовжити записувати дані в ***textfile***, знадобиться інша операція «>>».

Приклад 7.3 Недеструктивне пере направлення стандартного виведення

[methody@localhost methody]$ cat >> textfile
Пример 1.
^D
[methody@localhost methody]$ cat textfile
Это файл для примеров.
Пример 2.
[methody@localhost methody]$

Ми отримали саме той результат, який нам треба було: добавили у кінець вже існуючого файлу дані із стандартного виведення чергової команди.

Стандартне виведення – потік даних, який відкривається системою, для кожного процесу у момент його запуску, і призначений для даних, які виводяться процесом.

7.2.2 Стандартне введення

Аналогічним чином для передавання даних програмі може бути використано ***стандартна введення*** (скорочено - ***stdin***). При роботі з командним рядком стандартне введення – це символи, які вводяться користувачем з клавіатури. Стандартне введення можна перенаправити за допомогою командної оболонки, давши на нього дані з деякого файлу. Символ «<» служить для перенаправлення вмісту файлу на стандартне виведення програми. Наприклад, якщо викликати утиліту ***sort*** без параметру, вона буде читати рядки із стандартного виведення. Команда «***sort < ім'я_файлу*** » подасть на введення ***sort*** дані з файлу.

Приклад 7.4 Перенаправлення стандартного виведення з файла

[methody@localhost methody]$ sort < textfile
Пример 1
Это файл для примеров.
[methody@localhost methody]$

Результат дії цієї команди зовсім аналогічний команді ***sort textfile***, різниця в тому, що коли використовується «<», ***sort*** отримує данні із стандартного введення, нічого не знаючи про «***textfile***», звідки вони поступають. Механізм роботи ***shell*** у даному випадку той же, що і при пере направленні виводу: ***shell*** читає дані з файлу «***textfile***», запускає утиліту ***sort*** і передає їй на стандартне введення вміст файлу.

Варто пам'ятати, що операція «>» *деструктивна*: вона завжди утворює файл нульової довжини. Тому для, припустимо, сортування даних у файлі треба застосовувати послідовно ***sort < файл > новий_файл*** і ***mv новий_файл файл***.

Команда вигляду ***команда < файл > той_ же_файл*** просто уріже його до нульової довжини.

Стандартне виведення – потік даних, який відкривається системою для кожного процесу у момент його запуску, і призначений для вводу даних.

7.2.3 Стандартне виведення помилок

В якості першого прикладу і вправи на пере направлення ми вирішили записати керівництво по ***cat*** у свій файл ***cat.info***.

Приклад 7.5 Стандартне виведення помилок

```
[methody@localhost methody]$ info cat > cat.info
info: Запись ноды (coreutils.info.bz2)cat invocation...
info: Завершено.
[methody@localhost methody]$ head -1 cat.info
File: coreutils.info,  Node: cat invocation,  Next: tac invocation,
Up: Output of entire files
[methody@localhost methody]$
```

Ми винайшли, що незважаючи нашим вказівкам відправлятися у файл, два рядки, які виведено командою ***info***, все рівно проникнули на термінал. Очевидно, ці рядки не попали на ***стандартне виведення*** тому, що не відносяться безпосередньо до керівництва, яке повинна вивести програма, вони інформують користувача про хід *виконання* роботи: запис керівництва у файл. Для такого роду діагностичних повідомлень, а також для повідомлень про помилки, які виникнуть у ході виконання програми, в Linux передбачено ***стандартне виведення помилок*** (скорочено - ***stderr***).

Стандартне виведення помилок – потік даних, який відкривається системою, для кожного процесу у момент його запуску, і призначений для ***діагностичних повідомлень***, які виводяться процесом.

Використання стандартного виводу помилок рядом із стандартним введенням дозволяє відділити результат роботи програми від різнорідної інформації, яка супроводжує, наприклад, направивши їх у різні файли. Стандартне виведення помилок може бути пере направлено також, як і стандартні введення/виведення, для цього треба використовувати комбінацію символів «***2>***».

Приклад 7.6. Перенаправлення стандартного виведення помилок

```
[methody@localhost methody]$ info cat > cat.info 2> cat.stderr
[methody@localhost methody]$ cat cat.stderr
info: Запись ноды (coreutils.info.bz2)cat invocation...
info: Завершено.
[methody@localhost methody]$
```

На цей раз на термінал вже нічого не попало, стандартне виведення відправилося у файл ***cat.info***, стандартне виведення помилок у – ***cat.stderr***. Замість «>» і «***2***>» ми могли би написати «***1***>» і «***2***>». Цифри у даному випадку позначають номери дескрипторів файлів, які *відкриваються*. Якщо деяка утиліта очікує отримати відкритий дескриптор з номером, припустимо 4, то для її запуску обов'язково знадобиться використати поєднання «***4***>».

Іноді, одначе, вимагається об'єднати стандартне виведення і стандартне виведення помилок в одному файлі, а не розділяти їх. У командній оболонці bash для цього є спеціальна послідовність «***2>&1***». Це означає – направити стандартне виведення помилок туди, куди і стандартне виведення.

Приклад 7.7 Об'єднання стандартного виведення и стандартного виведення ошибок

```
[methody@localhost methody]$ info cat > cat.info 2>&1
[methody@localhost methody]$ head -3 cat.info
info: Запись ноды (coreutils.info.bz2)cat invocation...
info: Завершено.
File: coreutils.info,    Node: cat invocation,    Next: tac invocation,
Up: Output of entire files
[methody@localhost methody]$
```

В цьому прикладі важливий порядок пере направлень: у командному рядку ми спочатку вказали куди пере направити стандартне виведення («> ***cat.info***») і тільки потім наказали направити туди ж стандартний вивід помилок. Якщо би ми зробили навпаки («***2>&1 > cat.info***») результат вийшов би неочікуваний: у файл попало би тільки стандартне виведення, а діагностичні повідомлення з'явилися би на терміналі. Одначе, усе правильно: на момент виконання операції «***2>&1***» стандартне виведення було би пов'язане з терміналом, це означає, що *після* його виконання стандартне виведення помилок також буде пов'язано з терміналом. А подальше пере направлення стандартного виведення у файл, звісно, ніяк не відобразиться на стандартному виведенні помилок. Номер конструкції «***&номер***» - це номер *відкритого* дескриптору. Якби, згадана вище утиліта, яка записує у четвертий дескриптор, була написана на ***shell***, в ній би використовувалися пере направлення виду «***>&4***». Щоб не набирати громіздку конструкцію «***> файл 2>&1***» в ***bash*** використовується скорочення: «***&> файл***», або, що теж саме, «***>& файл***».

7.2.4 Пере направлення в нікуди

Іноді, наперед відомо, якісь дані, які виведені програмою, не знадобляться. Наприклад, попередження із стандартного виводу помилок. В цьому випадку, можна перенаправити стандартний вивід помилок у ***файл-дирку***, спеціально призначений для знищення даних - ***/dev/null***. Все, що записується у цей файл, просто буде *викинуто* і *ніде не зберігатися*.

Приклад 7.8 Пере направлення в /dev/null

[methody@localhost methody]$ info cat > cat.info 2> /dev/null

[methody@localhost methody]$

Точно таким же чином можна позбутися від стандартного виведення, відправивши його у ***/dev/null***.

7.3 Обробка даних у потоці

7.3.1 Конвеєр

Нерідко виникають ситуації, коли треба обробити виведення одної програми якою-небудь іншою програмою. Користуючись пере направленням введення-виведення, можна зберегти вивід одної програми у файлі, а потім направити його у файл на введення іншій програмі. Одначе, теж саме можна зробити і більше ефективно: пере направляти виведення можна не тільки у файл, а і *безпосередньо* на стандартне введення іншої програми. У цьому випадку замість двох команд буде потрібно тільки одну – програми передають одна одній дані «із рук у руки», В Linux такий засіб передавання даних називається **конвеєр**.

У ***bash*** для пере направлення стандартного виведення на стандартне введення іншій програмі служить символ «|». Самий простий і найбільше поширений випадок, коли вимагається використати конвеєр, виникає, якщо виведення не розміщується на екрані монітору і дуже швидко «пролітає» перед очима, так що людина не встигає його прочитати. У цьому випадку можна направити виведення у програму перегляду (***less***), яка дозволяє не кваплячись, перегорнути увесь текст, повернутися до початку, тощо.

Приклад 7.9 Найпростіший конвеєр

[methody@localhost methody]$ cat cat.info | less

Можна послідовно обробити дані декількома різними програмами, пере направляючи виведення на введення наступній програмі і організуючи скільки завгодно довгий ***конвеєр***, для обробки даних. У результаті отримуються дуже довгі командні рядки виду «***cmd1 | cmd2 | …|cmdN***», які можуть показатися громіздкими і незручними, але виявляються дуже корисними і ефективними при обробці великої кількості інформації, як буде показано далі у цій лекції.

Організація конвеєру об лаштована у shell за тією ж схемою, що і перенаправлення у файл, але з використанням особливого об’єкту системи – ***каналу***. Якщо файл можна уявити у вигляді *Коробки* з *Даними*, забезпеченої *Клапаном* для *Читання* або *Клапаном* для *Запису*, то ***канал*** – обидва *Клапани*, які один до одного взагалі без *Коробки*. Для визначеності, між клапанами можна поставити *Трубу*, яка негайно подає дані від входу до виходу (англійський термін «***pipe***» - базується як раз на цій уяві, у ролі труби виступає, звичайно, сам Linux). Каналом користуються відразу два процеси: один пише туди, другий читає.

Зв’язуючи дві команди конвеєром, shell відкриває ***канал*** (заводяться *два* дескриптори – вхідний і вихідний), підмінюються по уже описаному алгоритму ***стандартне введення*** першого процесу на вхідний дескриптор каналу, а ***стандартне виведення*** другого процесу – на вихідний дескриптор каналу. Після цього, залишається запустити по команді у цих процесах і стандартне виведення першої попадає на стандартне введення другої.

Канал – нероздільна пара дескрипторів (вхідний і вихідний), пов’язані один з одним таким чином , що дані, які записані вхідний дескриптор, будуть негайно доступні на читання з вихідного дескриптора.

7.3.2 Фільтри

Якщо програма і вводить дані і виводить, то її можна уявляти як трубу, у яку щось входить і щось виходить з неї. Зазвичай, зміст роботи таких програм полягає у тому, щоби визначеним чином *обробити* дані, які поступили. В Linux такі програми називаються **фільтрами**: дані проходять через них , причому, щось «застрягає» у фільтрі і не з’являється на виході, щось змінюється, щось проходить через фільтр незмінним. Фільтри в Linux, зазвичай, за замовченням, читають дані із стандартного введення, а виводять на стандартне виведення. Найпростішим фільтром ми вже користувалися багато разів – це програма ***cat***: власне, ніякої «фільтрації» даних вона не призводить, вона просто копіює стандартне введення на стандартне виведення.

Дані, які проходять через фільтр, уявляють собою текст: у стандартних потоках введення-виведення всі дання передаються у вигляді символів, рядок за рядком, як і у терміналі. Тому можуть бути зі стиковані за допомогою конвеєра введення і виведення любих двох програм, які підтримують стандартні потоки вводу-виводу. Це нагадує стандартний конструктор, де усі деталі поєднуються між собою.

У любому дистрибутиві Linux присутній набір стандартних утиліт, призначених для роботи з файловою системою і обробці текстових даних. Багатьма з них ми вже користувалися, це: ***who, cat, ls, pwd, cp, chmod, id, sort*** та інші. Ми вже встигли помітити, що кожна з цих утиліт призначена для використання якою-небудь *однією* операцією над файлами або текстом: виведення списку файлів у каталозі, копіювання, сортування рядків, хоча кожна утиліта може виконувати свою функцію дещо по-різному, в залежності від переданих їй ключів і параметрів. При цьому, вони працюють із стандартними потоками вводу/виводу, тому добре пристосовані для побудови конвеєрів: послідовно виконуючи прості операції над потоком даних, можна вирішувати доволі не тривіальні задачі.

Принцип комбінування елементарних операцій для виконання складних задач Linux взяв у спадщину від операційної системи Unix (як і багато інших принципів). Переважна більшість утиліт Unix не загубили свого значення і в Linux. Всі вони орієнтовані на роботу з даними у текстовій формі, багато є

фільтрами, всі на мають графічного інтерфейсу і викликаються з командного рядка. Цей пакет утиліт називається ***coreutils***.

7.3.3 Структурні одиниці тексту

Роботу у системі Linux майже завжди можна уявити як роботу з текстами. Пошук файлів і інших об'єктів системи – це отримання від системи тексту *особливої* структури – списку імен. Операції над файлами: утворення, перейменування, переміщення, а також, сортування, перекодування, та інше – означає заміну одних *символів* і *рядків* на інші або у каталогах, або у самих файлах. Налаштування системи у Linux призводить безпосередньо до роботи з текстами – редагуванню ***конфігураційних файлів*** і написанню сценаріїв.

Працюючи з текстом, у Linux треба прийняти до уваги, що текстові дані, які передаються системі, структуровані. Більшість утиліт обробляє не безперервний потік тексту, а послідовність *одиниць*. В текстових даних в Linux виділяються наступні структурні одиниці:

Рядок – основна одиниця передавання тексту в Linux. Термінал передає дані від користувача системі рядками (командний рядок), більшість утиліт вводять і виводять дані порядково, при роботі багатьох утиліт одному рядку відповідає один об'єкт системи (ім'я файлу, шлях і тому подібне) sort сортує рядки. Рядки розділяються кінця рядку «***n**» (***n****ewline*).

Поля В одному рядку може згадуватися і більше одного об'єкту. Якщо розуміти об'єкт як послідовність символів з визначеного набору (наприклад, літер), то рядок можна розглядати як утворення із слів і ***роздільників***. В цьому випадку текст від початку рядка до першого роздільника – це *перше* ***поле***, від першого роздільника до другого – *друге* ***поле*** і так далі. У якості роздільника можна розглядати любий символ, який не може використовуватися в об'єкті. Наприклад, якщо у шляху «***/home/methody***» роздільником є символ «/» то перше поле пусте, друге вміщує слово «***home***», третє – «***method***». Деякі утиліти дозволяють вибирати з рядків окремі поля (за номером) і працювати з рядками як з таблицею: вибирати і об'єднувати необхідні колонки і тому подібне.

Мінімальна одиниця тексту – ***символ***. ***Символ*** – це одна буква або другий письменний знак. Стандартні утиліти Linux дозволяють заміняти одні символи іншими (проводити транслітерацію), шукати і замінювати в рядках символи і комбінації символів.

Символи кінця рядку у кодуванні ***ASCII*** співпадає з управляючою послідовністю «^***J***», «переведення рядка», одначе у інших кодуваннях він може бути другим. Крім того, на більшості терміналів (не на всіх) - слід за переведенням рядка виводити ще символ повернення каретки («^***M***»). Це викликало плутанину: деякі символи потребують , щоб у кінці текстового файлу стояло *обидва цих* символи в певному порядку. Щоби плутанини уникнути, в Unix (як наслідок, в Linux), було прийнято єдине вірне рішення: вміст *файлу*

відповідає кодуванню, при *виведенні* на термінал кінці рядка перетворюються в управляючі послідовності згідно налаштуванню термінала.

У розпорядженні користувача Linuxє ряд утиліт, які виконують елементарні операції з одиницями тексту: пошук, заміну, розділення і об'єднання рядків, полів, символів. Ці утиліти мають, як правило, *однакову* уяву про те, як визначаються одиниці тексту: що таке рядок, які символи є роздільниками, і тому подібне. В багатьох випадках їх уяву можна міняти за допомогою налаштувань. Тому, такі утиліти легко взаємодіють одна з одною. Комбінуючи їх, можна автоматизувати доволі складні операції з обробки тексту.

7.4 Приклади задач

Цей підрозділ присвячено декільком прикладам використання стандартних утиліт для рішення різних типових (і не дуже) задач. Ці приклади не слід сприймати як вичерпний список можливостей, вони наведені просто як демонстрації того, як можна організувати обробку даних за допомогою конвеєра. Щоби засвоїти їх, треба читати керівництва і експериментувати.

7.4.1 Підрахунок

У європейській культурі дуже великим авторитетом користуються точні числа і кількісні оцінки. Тому, користувачу часто буває цікаво і навіть необхідно точно підрахувати щось багато чисельне. Комп'ютер як раз зручний для такої процедури. Стандартна утиліта для підрахунку рядків, слів і символів – ***wc*** (від англійської «***word count***» - «*підрахунок слів*»).

Одначе, ми запам'ятали, що в Linux багато чого можна представити як слова і рядки, і вирішили за допомогою цієї утиліти підрахувати свої файли.

Приклад 7.1. Підрахунок файлів за допомогою find и wc

[methody@localhost methody]$ find . | wc -l

42

[methodyOlocalhost methody]$

Ми отримали бажане число – «***42***». Для цього нам знадобилася команда ***find*** – яка рекомендувалася у керівництві, як інструмент потрібний необхідних файлів у системі. Ми выкликали ***find*** з одним параметром – каталогом, з якого починати пошук «**.**».***find*** виводить список знайдених файлів по одному на рядок, а оскільки критерії пошуку в даному випадку не уточнювалися, то ***find*** просто вивела список усіх файлів у всіх підкаталогах поточного каталогу (нашого домашнього каталогу). Цей список ми передали утиліті ***wc***, попрохавши її кількість отриманих рядків «***-l***», ***wc*** видала у відповідь число, яке ми шукали.

Задавши ***find*** критерії пошуку, можна підрахувати і що-небудь менш тривіальне, наприклад, файли, які утворювалися або були змінені у певний проміжок часу, файли з визначеним режимом доступу, з визначеним іменем і

тому подібне. Узнати про всі можливості пошуку за допомогою ***find*** і підрахунку за допомогою ***wc*** можна з керівництва по цим програмам.

7.4.2 Відкидання непотрібного

Іноді користувача цікавить тільки частина з тих даних, які збирається виводити програма. Ми вже користувалися утилітою ***head***, яка потрібна, щоби вивести тільки перші декілька рядків файлу. Не менш корисна утиліта ***tail*** (англійська - «*хвіст*»), яка виводить тільки останні рядки файлу. Якщо користувача цікавить тільки визначена частина кожного рядка файлу – допоможе утиліта ***cut***.

Припустимо, нам знадобилося отримати список всіх файлів і підкаталогів в «***/etc***», які належать групі «***adm***». І при тому, нам треба, щоби файли, які будуть знайдені, у списку були представлені не повним шляхом, а тільки іменем файлу (скоріше за все, це вимагається для наступної автоматичної обробки цього списку).

Приклад 7.11 Вилучення окремого поля

```
[methody<@localhost methody]$ find /etc -maxdepth 1 -group adm 2> /dev/null
\
> |  cut -d / -f 3
syslog.conf
anacrontab
[methody@localhost methody]$
```

Якщо команда виходить такою довгою, що її незручно набирати у однин рядок, можна розбити його на декілька рядків, для цього у тому місті, де треба продовжити набір з наступного рядка, достатньо поставити символ «\» і натиснути ***Enter***. При цьому на початку рядка ***bash*** виведе символ «>», який означає, що команда ще не передана системі і набір продовжується. Для системи байдуже, в скільки рядків набрано команду, можливість набирати в декілька рядків необхідна тільки для зручності користувача.

Ми отримали необхідний результат, задавши параметри ***find*** – каталог, де треб шукати і параметр пошуку – найбільшу допустиму глибину вкладеності і групу, які потрібні належати файли, які знайдені. Діагностичні повідомлення, які не потрібні, про заборонений доступ, ми відправили у ***/dev/null***, а потім вказали утиліті ***cut***, що в отриманих від стандартного введення рядках треба вважати роздільником символ «/» і вивести тільки третє поле. Таким чином, від рядків виду «***/etc/replace{filename}***» залишилося тільки «***replace{filename}***».

7.4.3 Вибір потрібного

7.4.3.1 Пошук

Часто користувачу треба знайти тільки нагадування чого-то конкретного серед даних, які виводяться утилітою, зазвичай ця задача зводиться до пошуку рядків. У яких зустрічається визначене слово або комбінація символів. Для цього

підходить стандартна утиліта ***grep***. ***Grep*** може шукати рядок у файлах, а може працювати і як фільтр: отримавши рядки із стандартного введення, вона виведе на стандартне виведення тільки ті рядки, де зустрілося поєднання символів, які шукаються. Ми вирішили зацікавитися процесами ***bash***, які виконуються у системі.

Приклад 7.12. Пошук рядка у виведенні програми

```
[methody@susanin methody]$ ps aux | grep bash
methody   3459  0.0  3 .0  2524  1636 tty2     S      14:30    0:00 -bash
methody   3734  0.0  1.1   1644   612 tty2     S      14:50    0:00 grep bash
```

Перший аргумент команди ***grep*** – той рядок, який треба шукати у стандартному введенні, даному випадку – це «***bash***», а оскільки ***ps*** по рядку на кожний процес, то ми отримали тільки процеси, в імені яких є ***bash***. Одначе, ми несподівано знайшли більше ніж шукали: у списку процесів, які виконуються, присутні два рядки, в яких зустрілося слово ***bash***, тобто: два процеси – один, який ми шукаємо – командний інтерпретатор ***bash***, а другий – процес пошуку рядка «***grep bash***», запущений нами *після* ***ps***. Це вийшло тому, що після розбору командного рядка **bash** запустила *обидва* дочірніх процеси, щоби організувати конвеєр, і на момент виконання команди ***ps*** процес ***grep bash*** вже був запущений і теж попав у вивід ***ps***. Для того, щоби отримати вірний результат, нам слід було добавити у конвеєр ще один ланцюг: **|** ***grep -v grep***, ця команда виключить з кінцевого виводу усі рядки, у яких зустрічається «***grep***».

7.4.3.2 Пошук за регулярним виразом

Дуже часто точно не відомо, яку саме комбінацію символів необхідно знайти. Точніше, відомо тільки те, як приблизно повинно виглядати слово, яке шукається, що в нього повинно входити і в якому порядку. Так, зазвичай, буває якщо деякі фрагменти тексту мають строго визначений формат. Наприклад. в керівництвах, які виводяться програмою ***info***, прийнятий такий формат посилань «****Note назва_вузла::***». В цьому випадку треба шукати не конкретне словосполучення символів, «Строку ****Note***», за якою іде назва вузла (одне або декілька слів і пробілів), які закінчуються символами «***::***»». Комп'ютер цілком здібний виконати такий запит, якщо його сформулювати на строгій і зрозумілій мові ***регулярних виразів***. Регулярний вираз – це засіб однією формулою задати усі послідовності символів, які підходять користувачу.

Приклад 7.13. Пошук посилань у файлі info

```
[methody@susanin methody]$ info grep > grep.info 2> /dev/null
[methody@susanin methody]$ grep -on "\*Note[":]*::" grep.info
324:*Note Grep Programs::
684:*Note Invoking:: [methody@susanin methody]$
```

Перший параметр ***grep***, який взятий у лапки – це і є регулярний вираз для пошуку посилань у форматі info, другий параметр – ім'я файлу, у якому треба шукати. Ключ «***o***» примушує grep виводити рядок не повністю, а тільки ту частину, яка співпала з регулярним виразом (шаблон пошуку), а «***-n***» - виводить номер рядку, у якому зустрілося дане спів падіння.

У регулярному виразі більшість символів позначають самі себе, так якби ми шукали звичайний текстовий рядок, наприклад «***Note***» і «***::***» в регулярному виразі відповідають відповідним рядкам «***Note***» і «***::***» в тексті. Одначе, деякі символи мають спеціальні значення, самий головний з таких символів є – зірочка «*****», яка поставлена після елемента регулярного виразу позначає, що можуть бути знайдені тексти, де цей елемент повторено любу кількість разів, в тому числі і ні жодного, тобто просто відсутній. В нашому прикладі зірочка зустрічається двічі: перший раз знадобилося включити у регулярний вираз саме символ «зірочка», для цього знадобилося лише його спеціальне значення, поставивши перед ним «\».

Друга зірочка позначає, що елемент, який стоїть перед нею, може бути повторений любу кількість разів від нуля до нескінченності. У нашому випадку зірочка відноситься до виразу у квадратних дужках – «**^*:***», що означає «любий символ крім «***:***»». Весь регулярний вираз можна прочитати так : «Рядок «****Note***», за яким іде нуль або більше символів, крім «***:***», за якими іде рядок «***::***»». Особливість роботи «*****» складається з того, що вона намагається вибрати співпадіння максимальної довжини. Саме тому, елемент, до якого відносилася «*****», був заданий як «***не*** «***:***»». Вираз «нуль або більше *любих* символів» (він записується як « **.***») у випадку, коли в одному рядку зустрічаються *два* посилання, вбирає підрядок від кінця першого *«*****Note***» до початку *останнього* «***::***» (символи «***:***», які розмістилися у середині цього підрядку, відмінно пізнаються як «любі»).

На мові регулярних виразів можна також позначити «любий символ» («**.**»), «одне або більше співпадань» («**+**»), початок і кінець рядку («^» і «***$***» відповідно) і тому подібне. Завдяки регулярним виразам можна автоматизувати дуже багато задач, які у протилежному випадку вимагали би величезної кропіткої роботи людини. Більше детальні відомості про можливості мови регулярних виразів можна отримати в керівництві ***regex (7)***. Одначе, керівництво – це на підручник для використання, тому, щоби навчитися економити час і зусилля за допомогою регулярних виразів, корисно звернутися до відповідних навчальних посібників.

Регулярні вирази в Linux використовуються не тільки для пошуку програмою ***grep***. Дуже багато програм так або інакше працюють з текстом, в першу чергу, текстові редактори, підтримують регулярні вирази. До таких програм відносяться два «головних» текстових редактори Linux – ***vim*** і ***emacs***, про які піде мова далі. Одначе, треба враховувати, що в різних програмах використовуються різні діалекти мови регулярних виразів, де одні і ті ж поняття

мають різні позначення, тому завжди треба звертатися до керівництва по конкретній програмі.

На завершення можна сказати, що регулярні вирази дозволяють різко підвищити ефективність роботи, вони добре інтегровані у робоче середовище в системі Linux, і е сенс витратити час на їх вивчення.

7.4.4 Заміни

Зручність роботи з потоком не в останню чергу складається в тому, що можна не тільки вибірково передавати результати роботи програм, але і автоматично замінювати один текст іншим прямо у потоці.

Для заміни одних символів на інші призначена утиліта ***tr*** (скорочення від англійського «***translate***», «*перетворити, перевести*», яка працює як фільтр. Ми вирішили використати її прямо за призначенням і виконати за її допомогою транслітерацію – заміну латинських символів близькою за звучанням кирилицею.

Приклад 7.14 Заміна символів (транслітерація)

```
[methody<3localhost methody]$ cat cat.info | tr abcdefghijklmnopqrstuvwxyz \
> абцдефгхийклмнопкрстуввсиз | tr ABCDEFGHIJKLMNOPRSTUVWXYZ
\
>АБЦДЕФГХИЙКЛМНОПКРСТУВВСИЗ | head -4
Филе: цореутилс.инфо,  Ноде: цат инвоцатион,  Нест: тац инвоцатион,
Тп: Оутпут оф ентире филес
'цат': Цонцатенате анд врите филес
============================
[methody@localhost methody]$
 (через X-Windows)
```

Ми попрацювали, складаючи два параметри для утиліти ***tr***: відповідності латинських літер кириличним. Перший символ з першого параметру ***tr*** заміняє першим символом другого, другий – другим і так далі. Ми обробили потік фільтром ***tr*** двічі: спочатку щоби замінити рядкові літери, а потім – прописні, ми могли зробити це за один прохід (просто добавивши до параметрів прописні після рядкових), але не захтіли виписувати такі довгі рядки. Тексту, який ми отримали на виході, навряд чи можна знайти практичне застосування, одначе, транслітерацію можна застосувати і з користю. Якщо не вказати ***tr*** другого параметру, то усі символи, які перераховано у першому, будуть замінені на «ніщо», тобто, повністю знищені з потоку. За допомогою ***tr*** також можна знищити дублюючі символи (наприклад, зайві пробіли або переведення рядків), замінити пробіли переведеннями рядків і тому подібне.

Крім простої заміни окремих символів, можлива заміна послідовностей (слів). Спеціально для цього призначений потоковий редактор ***sed*** (скорочення

від англійської «***stream editor***»). Він працює як фільтр і виконує редагування рядків, які поступають: заміну одних послідовностей символів на інші, причому, можна замінювати і ***регулярні вирази***.

Наприклад, ми за допомогою ***sed*** можемо зробити більше для незвичного читача список файлів, якій виводиться ***ls***.

Приклад 7.15. Заміна за регулярним виразом

```
[methody@localhost methody]$ ls -l |  sed s/"-^rwx^/Файл:/ \
>       |  sed s/~d[-rwx]*/Каталог:/
итого 124
Файл:       1 methody methody     2693  Ноя  15  16:09  cat.info
Файл:       1 methody methody       69  Ноя   15 16:08  cat.stderr
Каталог:    2 methody methody     4096   Ноя   15 12:56  Documents
Каталог:    3 methody methody     4096   Ноя   15 13:08  examples
Файл:       1 methody methody    83459    Ноя   15 16:11  grep.info
Файл:       1 methody methody       26   Ноя    15 13:08  loop
Файл:       1 methody methody       23   Ноя    15 13:08  script
Файл:       1 methody methody       33   Ноя    15 16:07 textfile
Каталог:    2 methody methody     4096    Ноя    15 12:56 tmp
Файл:       1 methody methody       32    Ноя    1513:08  to.sort
[methody@oblomov methody]$
```

У ***sed*** дуже широкі можливості, але достатньо незвичний синтаксис, наприклад, виконується командою «***s/що_замінювати /на_що_замінювати/***». Щоби у ньому розібратися, треба обов'язково прочитати керівництво ***sed(1)*** і знати регулярні вирази.

7.4.5 Упорядкування

Для того, щоби розібратися у даних, нерідко вимагається їх упорядкувати: за алфавітом, по номеру, по кількості вживань. Основний інструмент для упорядкування – утиліта ***sort***, вже нам знайома. Одначе, тепер ми вирішили використати її у поєднанні з декількома іншими утилітами:

Приклад 7.16. Отримання упорядкованого списку за частотністю слововживань

```
[methody@localhost methody]$ cat grep.info | tr "[:upper:]" "[:lower:]" \
> | tr "[:space:][:punct:]" "\n" \
> |  sort  | uniq -c |  sort -nr | head -8
15233
720 the
342 of
251 to
244 a
```

213 and
180 or
180 is
[methody@localhost methody]$

Комп'ютер по нашому плану підрахував, скільки разів які слова були вживані у файлі «***grep.info***» і вивів декілька самих частотних з вказівкою кількості вживань у файлі. Для цього знадобилося, спочатку, замінити всі великі літери маленькими, щоби не було різних засобів написання одного слова, потім, замінити всі пробіли і знаки перетину кінцем рядку (символ «***n***»), щоби у кожному рядку було рівно по одному слову. Ми всюди взяли параметри ***tr*** у лапки, щоби ***bash*** не зрозумів їх невірно. Потім список було відсортовано, всі слова, які повторюються, були замінені одним словом з вказівкою кількості повторів («***uniq -c***»), потім рядки знову відсортовано за зменшенням чисел на початку рядку («***sort -nr***») і виведені перші 8 рядків («***head -8***»).

7.4.6 Запуск команд

Дані, які отримані у конвеєрі, можна перетворити на керівництво до дії для комп'ютера. Наприклад, для кожного отриманого із стандартного вводу рядка можна запустити яку-небудь команду, передавши їй цей рядок у якості параметру. Для цієї цілі служить утиліта ***xargs***.

Приклад 7.17. Пошук всіх файлів, які виконуються, і точно є сценаріями

```
[methody@localhost methody]$ find /bin -type f -perm +a=x \
 >  | xargs grep -l -e '"#!/' 2> /dev/null
/bin/egrep
/bin/fgrep
/bin/unicode_start
/bin/bootanim
[methody@localhost methody]$
```

Тут ми вирішили визначити, які з файлів, які виконуються в каталозі «***/bin***», є сценаріями. Для цього ми знайшли всі звичайні файли, які виконуються, (вказати «***type -f***» «*звичайні файли*» знадобилося, щоби у результат не попали каталоги, які всі виконуються), потім для кожного знайденого файлу викликали ***grep***, щоби пошукати в ньому сполучення «***#!/***» на початку рядку. Ключ «***-l***» звелів ***grep*** виводити не виявлений рядок, а ім'я файлу, у якому знайдено співпадіння. Так ми отримали список файлів, які виконуються, в яких є рядок з вказівкою інтерпретатору беззаперечних сценаріїв.

Розширене пояснення попереднього прикладу для практичної роботи

**]$ find /bin -type f -perm +a=x **
> | xargs grep -l -e '^#!/' 2> /dev/null
----------"----------→ *find [шлях] [опції]*

Знайти у каталозі ***/bin***:

- **type f →** (type x – знаходить файли заданого типу , де x – один з визначених типів), f - порівнює файли.
- **perm** – права: знаходить файли с заданими правами доступу; «+» – пошук файлів у яких встановлений із указаних бітів дозволений, інші біти при цьому, ігноруються;
- **+a=x** – тільки файли з атрибутом виконання, інші ігноруються;
- **xargs** – утиліта для формування списку аргументів і виконання команди Linux. Команда ***xargs*** об'єднує зафіксований набір заданих у командному рядку початкових аргументів з аргументами, прочитаними із стандартного вводу, і виконує команду один або декілька разів.
- **(***В даному випадку***!)** Приймає на вхід потік виводу від команди ***find*** і відправляє його як аргументи у команду ***grep***;
- **grep** – пошук файлів або рядків у файлі по шаблону;
- **-l** – видає тільки імена файлів, які вміщують зіставні рядки, по одному в рядку. Якщо зразок файлу знайдено у декількох рядках файлу, ім'я файлу не повторюється;
- **-e** – е-список зразків, тобто: задає один або декілька зразків для пошуку. В даному випадку – однин зразок «**'^#!/'**» – регулярний вираз;
- **2> /dev/null** – знищення повідомлень про вивід.

8 МОЖЛИВОСТІ КОМАНДНОЇ ОБОЛОНКИ LINUX

8.1 Редагування вводу

Вже деякий час попрацювавши в Linux, понабиравши команди у командному рядку, ми прийшли до висновку, що в звернені до оболонки не завадять деякі зручності. Одна з таких зручностей – можливість редагувати рядок, який вводиться, за допомогою клавіші ***Backspace*** (знищення останнього символу), «^***W***» (знищення слова) і «^***U***» (знищення всього рядка) – надає сам термінал Linux. Ці команди працюють для *любого* порядкового введення: наприклад, якщо запустити програму ***cat*** без параметрів, щоби та негайно відображала рядки, які вводяться з терміналу. Якщо з якихось причин в рядок на екрані влізло сміття , можна натиснути «^***R***» (***redraw***) – система виведе у новому рядку вміст вхідного буферу.

Ми не забули, що ***cat*** без параметрів слід завершувати командою «^***D***» (кінець введення). Цю команду, як і попередні, інтерпретує при введенні з терміналу система. Система також перетворює інші управляючі символи (наприклад, «^***C***» або «^***Z***») в ***сигнали***. В дійсності, усі управляючі символи, які інтерпретує система, можна пере налаштовувати за допомогою команди ***stty***. Повний список того, що можна налаштовувати, видає команда ***stty –a***.

Приклад 8.1 Налаштування термінальної лінії

```
[methody@localhost methody]$ stty -a
localhost 38400 baud; rows 30; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eof = <undef>;
eof2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^0; min = 1 ; time = 0 ;
-parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-iuclc -ixany -imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

Побачивши стільки багато можливостей, можна взятися читати керівництво (***man stty***), одначе, знайдено в ньому буде не так уже й багато корисного для нас. Із управляючих символів (рядки з другого по четвертий) цікаві «^***S***» і «^***Q***», за допомогою який можна, відповідно, за допомогою яких можна, відповідно, призупинити і відновити подання на термінал (якщо тексту вивелося вже багато, а прочитати його не встигаєш). Можна помітити, що налаштування ***erase*** (знищення одного символу) відповідає управляючому символу, який повертається клавішею ***Backspace*** саме віртуальної консолі Linux

– «^***?***». На багатьох терміналах клавіша ***Backspace*** повертає інший символ – «^***H***». *Я*кщо необхідно *перевизначити* налаштування ***erase***, можна скористатися командою «***stty erase*** ^***H***» причому, «^***H***» (для зручності) дозволено вводити і як *два* символи: «^» і «***H***».

На кінець, щоби *позбавити* символ, який передається, управляючих функцій (якщо, наприклад, вимагається передати програмі на введення символ з кодом ***3***, тобто «^***C***»), безпосередньо, перед введенням цього символу, треба подати команду «^***V***» (***lnext***).

Приклад 8.2 Екранування управляючих символів

[methody@localhost methody]$ cat | hexdump -C Зараз натиснемо Ctrl+C

[methody@localhost methody]$ cat | hexdump -C Зараз Ctrl+V, Ctrl+C, enter и Ctrl+D'C

00000000 f4 c5 d0 c5 d2 d8 20 43 74 72 6c 2b 56 2c 20 43 |Зараз Ctrl+V, C|
00000010 74 72 6c 2b 43 2c 20 45 6e 74 65 72 20 c9 20 43 | trl+C, enter i C|
00000020 74 72 6c 2b 44 03 0a |trl+D.. |
00000027

Пояснення:

На практиці набирати: ^***V*** , ^***C*** *Enter*
^***V*** , ^***D*** *Enter*
^***V*** , ^***C*** *Enter*
^***D***

Отримаємо: ***03 0a 04 0a 03 0a***

Тут ми перервали, як і збиралися, роботу першого ***cat***. При цьому до ***hexdump***, фільтра, який переводить вхідний потік у шістнадцятерічне уявлення, права навіть не дійшла, тому що ***cat*** не встиг обробити жодного рядку. У другому випадку «^***C***» після «^***V***» втратило управляючий зміст і відобразилося при виведенні. З ключем «-***C***» ***hexdump*** виводить також і текстову уяву вхідного потоку, замінюючи символи, які не друкуються, крапками. Так, на крапки були замінені і «^***C***» (*ASCII-код 03*), і *Enter-символ* кінця рядку (*ASCII-код Оа, в десятковій уяві - 12*). «^***V***» і «^***D***» на вхід ***hexdump*** не попали: їх, як управляючі символи обробила система.

Інші налаштування ***stty*** відносяться до обробки тексту при виведенні на термінал і введенні з нього. Вони цікаві тільки у тому сенсі, що при їх зміненні робота з командною оболонкою стає не зручною. Наприклад, налаштування ***echo*** визначає, чи буде *система* відображати на екрані все, що вводить користувач. При включеному ***echo***, натиск любої алфавітно-цифрової клавіші (введення символу) призводить до того, що система (пристрій типу ***tty***) *виводить* цей символ на термінал. Налаштування відключається, коли з клавіатури вводиться пароль. При цьому важко звільнитися від відчуття, що введення з клавіатури не

відбувається. Ще гірший стан справ з налаштуваннями, які складаються з кусків виду «***i***», «***o***», «***cr***» і «***nl***». Ці налаштування управляють перетворенням при введенні і виведенні, позначення кінця рядку *двома* символами в о*дин*, прийняте в Linux, яке історично склалося. Може бути так, що клавіша ***Enter*** терміналу повертає як раз невірний символ кінця рядку, а перетворення відключено. Тоді, замість ***Enter*** слід використовувати «***^J***» – символ, який насправді відповідає кінцю рядку.

У всіх випадках, коли термінал знаходиться у не зрозумілому стані – не реагує на ***Enter***, не показує введення, не знищує символів, виводить текст «сходинками» і тому подібне, рекомендується «лікувати» налаштування терміналу за допомогою ***stty sane*** – спеціальної форми ***stty***, яка скидає налаштування терміналу у деякий придатний до роботи стан. *Я*кщо незрозумілий стан терміналу виник одноразово, наприклад, після аварійного завершення екранної програми (редактора ***vim*** або оболонки ***mc***), скористатися командою ***reset***. Вона знову налаштовує термінал у повній відповідності з системною конфігурацією (яка вказана у файлі ***/etc/inittad***, лекція «Етапи завантаження системи (10)» та *terminfo*)/

Увага!

*Я*кщо терміна веде себе дивно, послідовність «***^J stty sane - ^J***» може його вилікувати!

Встановлення mc в mint 17.3 в root: ***# apt-get install mc***

8.1 Редагування командного рядку

Навіть не вивчаючи спеціально можливості командної оболонки, ми активно використовували деякі з них, не доступні при введенні при введенні тексту більшості утиліт (зокрема, ні ***cat***, ні ***hexdump***). Мова піде про клавіші *Стрілка вліво* і *Стрілка вправо*, за допомогою яких можна переміщувати курсор по командному рядку, і клавіші *Del*, яка знищує символ *під* курсором, а не після нього. Раніше ми впевнилися, що ці команди працюють у ***bash***, але не працюють для ***cat***. Більше того, для простого командного інтерпретатору – ***sh*** – вони теж не працюють.

Це означає, що можливості редактора командного рядку специфічні для різних командних оболонок. Одначе, самі необхідні команди редагування підтримуються у всіх різновидах ***shell*** подібним чином. В усіх видах Linux обов'язково є ***bash***, а якщо ми достатньо досвідчені, щоби встановлювати і налаштовувати пакети, можна встановити ***zsh***, у нього можливостей більше, чим може знадобитися одній людині. Бажано зайнятися вивченням документації по ***bash***, хоча – це справа не проста (***bash.info*** має більше восьми з половиною тисяч рядків). Навіть про редагування командного рядку, написано стільки, що за один раз прочитати важко.

Спроба зробити огляд матеріалу принесла деяку користь. По-перше, переміщуватися у командному рядку можна не тільки по одному символу вперед і назад, але і по словам: команди ***ESCF/ECSB*** або ***Alt+F/Alt+B*** відповідно (від ***f****orward* і ***b****ackward*), працюють також клавіші ***Home*** і ***End***, або теж саме «^***A***» і «^***E***» . По-друге, крім роботи з *одним* командним рядком, існують ще не мало інших зручностей.

8.1.2 Історія команд

Двома іншими клавішами із стрілками – уверх і вниз – ми теж активно користувалися, не підозрюючи, запускаємо в дію цим досить потужний механізм ***bash*** – роботу з **історією команд**. Усі команди, які набрані користувачем, ***bash*** запам'ятовує і дозволяє звертатися до них у подальшому. По стрільці уверх (можна використовувати і «^***P***», ***p****revious*), список поданих команд «провертається» від останньої до першої, а по стрільці униз («^***N***», ***n****ext*) – назад. Відповідна команда відображається у командного рядка, як така яку тільки що набрали, її можна редагувати і подати оболонці (підганяти курсор до кінця рядку при цьому не обов'язково).

Якщо необхідно добути з історії якусь давню команду, простіше не ганяти список історії стрілками, *пошукати* в ній за допомогою команди «^***R***» (***r****everse search*). При цьому виводиться підказка спеціального виду («(***reverse -i-search***)»), підрядок пошуку (оточений символами ‘ і ’) і остання з команду історії, в якій ця підказка присутня.

Приклад 8.3 Пошук по історії команд

Примітка:

[methody@localhost methody]$ (***Набирається ^R, потім,***
^R | (reverse-i-search)'': ***після: набирають***
i | (reverse-i-search)'i': ls i ***початок команди, яку***
n | (reverse-i-search)'in': info ***шукають в історії***)
f | (reverse-i-search)'inf': info
o | (reverse-i-search)'info': info
^R | (reverse-i-search)'info': man info
^R | (reverse-i-search)'info': info "(bash.info.bz2) Commands For History"

Примітка:

Набирається ^***R***, потім набирається початок команди в історії, яка шукається.

Приклад представляє символи, які вводяться, нами (в лівій частині до «|») і вміст останнього рядку терміналу. Це «кадри» роботи з одним і тим же рядком, які показують, як він змінюється при наборі. Набравши «***info***» ми продовжили пошук цього підрядку, повторюючи «^***R***» до тих пір поки не наштовхнулися на

необхідну нам команду, яка вміщує підрядок «***info***». Залишилося тільки передати його ***bash*** за допомогою *Enter*.

Щоби історія команд могла зберігатися *між* сеансами роботи користувача, ***bash*** записує її у файл ***.bash_history***, який знаходиться у домашньому каталозі користувача. Робиться це у момент *завершення* роботи оболонки: історія, яка накопичена за час роботи, дописується у кінець цього файлу. При наступному запуску ***bash*** зчитує ***.bash_history*** повністю. Історія зберігається не вічно, кількість команд, які запам'ятовуються у ***.bash_history***, обмежена (зазвичай, 500 командами, але можна і пере налаштувати).

Примітка:

На практиці для перегляду історії команд треба: ***cat .bash_history/less***.

8.1.3 Скорочення

Пошук по історії – зручний засіб: довгий командний рядок можна не набирати повністю, знайти і використати. Одначе, *давню* команду прийдеться добувати за допомогою декількох «**^*R***» – а можна і зовсім не дошукатися, якщо вона вже вибула звідти. Для того, щоби оперативно замінювати короткі команди довгими, варто скористатися **скороченнями** (***aliases***). У конфігураційних файлах командного інтерпретатора користувача, зазвичай, *вже* визначено декілька скорочень, список яких можна переглянути за допомогою команди ***aliases*** без параметрів:

Приклад 8.4. Перегляд раніше визначених скорочень

```
[methody@localhost methody]$ alias
alias cd..='cd ..'
alias cp='cp -i'
alias l='ls -lapt'
alias ll='ls -laptc'
alias ls='ls --color=auto'
alias md='mkdir'
alias mv='mv -i'
alias rd='rmdir'
alias rm='rm -i'
```

Примітка:

підкреслений курсив, набирати у командному рядку для приклада 8.5

Примітка

Із скороченнями ми вже стикалися у лекції «права доступу (6)», де команда ***ls*** відмовилася працювати згідно з теорією. Виявилося, що по команді ***ls*** замість утиліти ***/bin/ls*** **bash** запускає особисту команду-скорочення, і перетворюється в команду ***ls -color=auto.*** *Повторно* підрядок «***ls***», який з'явився у команді, інтерпретатор вже не відкидає, щоб уникнути вічного циклу. Наприклад, команда ***ls -a*** перетворюється в результаті в ***ls -color=auto -al***. Також, люба команда, яка починається з ***rm***, перетворюється в ***rm -i*** (***i****nteractive*), що нас

бентежить, тому, що жодне видалення не обходиться без запитів у стилі «***rm***: *удалить обычный файл* **'*файл*'?**».

Примітка до прикладу 8.5

/usr/share/doc – немає таких файлів або каталогів треба: /***home/user1/tmp ->*** отримаємо: ***~/tmp/bin/bin~***

Приклад 8.5. Використання скорочень і pushd/popd

```
[methody@localhost methody]$ unalias cp rm mv
[methody@localhost methody]$ alias pd=pushd
[methody@localhost methody]$ alias pp=popd
[methody@localhost methody]$ pd /bin
/bin ~
[methody@localhost bin]$ pd /usr/share/doc  →див. примітку
/usr/share/doc /bin ~                       →див. примітку
[methody@localhost doc]$ cd /var/tmp
[methody@localhost tmp]$ dirs
/var/tmp /bin ~
[methody@localhost tmp]$ pp
/bin ~
[methody@localhost bin]$ pp
~
[methody@localhost methody]$ pp
-bash: popd: directory stack empty
```

Від «***-i***», яке надоїло, ми позбулися за допомогою команди ***unalias***, а за одне, ввели скорочення для **bash** – ***pushd*** і ***popd***. Ці команди, подібно ***cd***, змінюють каталог. Різниця складається в тому, що ***pushd*** всі каталоги, які користувач робить поточними, запам'ятовує в особливому списку (стеку). Команда ***popd*** знищує останній елемент цього стеку, і робить поточний каталог перед останнім. Обидві команди, на додаток, виводять вміст стеку каталогів (теж саме робить команда ***dirs***). Команда ***cd*** в **bash** також працює із стеком каталогів: вона *замінює* його останній елемент новим.

Команда-скорочення – внутрішня команда ***shell***, яка задається користувачем. Зазвичай, замінює одну більш довгу команду, яка часто використовується при роботі у командного рядка. Скорочення не успадковуються з оточенням.

8.1.4 Добудова

Скорочення дозволяють швидко набирати *команди*, одначе, ніяк не торкаються імен *файлів*, які частіше всього, виявляються параметрами цих команд. Буває, що набраного рядка – шляху до файлу, і декількох перших літер його імені достатньо для *однозначної* вказівки на цей файл, тому що по введеному шляху, більше файлів, чиє ім'я починається на ці літери, просто не має, щоби не дописувати літери, які залишилися (а імена файлів в Linux можуть

бути досить довгими), необхідно нажати клавішу *Tab*. І ***bash*** сам добудує ім'я файлу до цілого (знову скористаємося методом кадрів):

Приклад 8.6. Використання добудови

(Приклад – теоретичний, немає таких об'єктів, див. (*))

[methody@localhost methody]$ ls -al /bin/base

Tab | [methody@localhost methody]$ ls -al /bin/basename

-rwxr-xr-x 1 root root 12520 Июн 3 18:29 /bin/basename

[methody@localhost methody]$ base

Tab | [methody@localhost methody]$ basename

Tab | [methody@localhost methody]$ basename ex

Tab | [methody@localhost methody]$ basename examples/

Tab | [methody@localhost methody]$ basename examples/-filename-with-

-filename-with-

Примітка (*)

Практичний приклад, **вводити символи, які підкреслені** (необхідно мати дерево ***/home/user1/ddd***; і в ***ddd*** мати файли ***textfile*** и ***loop***):

…$ ls –al /**h**ome/**u**ser1/**d**dd/**t**extfile

<tab> <tab> <tab> <tab>

…$ ls –al /**h**ome/**u**ser1/**d**dd/**l**oop

<tab> <tab> <tab> <tab>

Примітка. Даний приклад є суто теоретичним оскільки немає таки об'єктів, до яких звертається програма.

На практиці треба:

Утворити на шляху ***/home/user1/ddd*** файли – ***textfile*** і ***loop***.

В приведених нижче прикладах, після символів, які вводяться, (для виділення, вони підкреслені) натискається «*Tab*»:

…$ *ls -al /home/user1/ddd/textfile*

…$ *ls -al /home/user1/ddd/loop*

Виявляється, і ім'я команди можна вводити не повністю: оболонка здогадується добудувати слово, яке набирається, якщо це слово стоїть з початку командного рядку. Таким чином, команду ***basename examples/-filename-with-*** ми набрали за значно менше натисків на клавіатуру («***base***» і чотири *Tab*).

Нам не прийшлося початок імені файлу в каталозі ***examples***, тому що файл там був всього один.

Виконуючи ***добудову*** (*completion*), ***bash*** може вивести не весь рядок, а тільки ту частину, відносно якої у нього немає сумнівів. Якщо подальша добудова може піти *декількома* шляхами, то одноразовий натиск *Tab*, призведе до того, що ***bash*** розгублено пискне, другий натиск приведе до виведення під командним рядком списку усіх можливих варіантів. В цьому випадку треба підказати командній оболонці продовження: дописати декілька символів, визначаючих, за яким шляхом піде добудова і знову натиснути *Tab*.

Пошук ключового слова «*completion*» по документації ***bash*** видав так багато інформації, що необхідно звернутися за допомогою. Якщо у ***bash*** *декілька* типів добудови (за іменем файлів, за іменем команд і тому подібне), то в ***zsh*** їх *скільки завгодно*: існує засіб запрограмувати любий алгоритм добудови і задати шаблон командного рядку, в якому саме цей засіб буде застосовуватися.

8.2 Генерація імен файлів

Добудова дуже зручна, коли ціль користувача – задати *один* конкретний файл в командному рядку. Якщо треба робити відраз з *декількома* файлами – наприклад, для переміщення їх у другий каталог за допомогою ***mv***, добудова не допомагає. Необхідний засіб – задати одне «*загальне*» ім'я групи файлів, з якими буде працювати команда. В переважній більшості випадків це можна зробити за допомогою **шаблону**.

8.2.1 Шаблони

Шаблони в командному інтерпретаторі використовуються приблизно у тих же цілях, що і **регулярний вираз**, згаданий у лекції «робота з текстовими даними (7)»: для пошуку рядків визначеної структури серед множини різноманітних рядків. На відміну від регулярного виразу, шаблон завжди використовується у рядку цілком, крім того, він об лаштований значно простіше (значить - бідніше).

Символи у шаблоні поділяються на звичайні і **спеціальні**. Звичайні символи відповідають таким же символам у рядку, а спеціальні обробляються особливим засобом:

– шаблону, який складається тільки із звичайних символів, відповідає єдиний рядок, який складається з тих же символів у тому ж порядку. Наприклад, шаблону «***abc***» відповідає рядок ***abc***, але не ***aBc*** або ***ABC***, тому що великі і маленькі літери розрізняються.

– шаблону, який складається з єдиного символу «*******», відповідає *любий* рядок любої довжини (в тому числі і пустий).

– шаблону, який складається з єдиного символу «***?***», відповідає любий рядок довжиною в *один* символ, наприклад, ***a***, **+**, але не ***ab*** або ***8888***.

– шаблону, який складається з любих символів, укладених у квадратні дужки «***[***» і «***]***» відповідає рядок довжиною в *один* символ, при чому, цей символ повинен *зустрічатися* серед вміщених в дужки. Наприклад, шаблону «***[bar]***» відповідають тільки рядки ***a***,***b*** і ***r***, але не ***c***, ***B***, ***bar*** або ***ab***. Символи в середині дужок можна не перераховувати повністю, а задавати *діапазон*, на початку якого стоїть символ з найменшим ACSCII-кодом, після іде «-», а потім – символ з найбільшим ACSCII-кодом. Наприклад, шаблону «***[0-9a –fA-F]***» відповідає одна шістнадцятирічна цифра (скажімо ***5***, ***e*** або ***C***). Якщо, після «***[***» в шаблоні іде «***!***», то йому відповідає рядок з одного символу *не* перерахованому між дужками.

– шаблону, який складається з *декількох* частин, відповідає рядок, який можна розбити на стільки ж підрядків (можливо пустих), причому, перший підрядок буде відповідати першій частині шаблону, другий – другій частині і так далі. Наприклад, шаблону «***a*b?c***» будуть відповідати рядки ***ab@c*** («*******» відповідає пустий підрядок), ***a+b=c*** і ***aaabbc***, але *не* відповідати ***abc*** («***?***» відповідає підрядок ***c***, а для «***c***» відповідності не знаходиться), ***@ab@c*** (немає відповідності для «***a***») або ***aaabbbc*** (з трьох ***b*** перше відповідає «***b***», друге – «***?***», ось третє приходиться на «***c***»).

Щаблон – **р**ядок спеціального формату, який використовується у процедурах текстового пошуку. Кажуть, що рядок ***відповідає*** шаблону, якщо можна, по визначеним правилам, кожному символу рядка поставити у відповідність символ шаблону.

В Linux найбільш популярні шаблони у форматі командного інтерпретатора і ***регулярні вирази***.

8.2.2 Використання шаблонів

Шаблони використовуються у декількох конструкціях ***shell***. Головне місце їх застосування – командний рядок. Якщо оболонка бачить в командному рядку шаблон, вона негайно замінює його на список *файлів*, імена яких йому відповідають. Команда. Яка потім викликається, отримує у якості параметрів список файлі уже без всіляких шаблонів, так якби цей список користувач ввів вручну. Ця здібність командного інтерпретатора називається ***генерацією імен файлів***.

Приклад 8.7 Використання шаблона у командному рядку

```
[methody@localhost methody]$ ls .bash*
.bash_history .bash_logout .bash_profile .bashrc
[methody@localhost methody]$ /bin/e*
/bin/ed /bin/egrep /bin/ex
[methody@localhost methody]$ ls  /home/user1/*a*
-filename-with-
[methody@localhost methody]$ ls -dF /home/user1/*[ao]*
Documents/ examples/ loop to.sort*
```

Ми, як це буває з новаками, негайно наштовхнулися на декілька «підводних каменів», неминучих у ситуації, коли кінцевий результат не відомий. Тільки перша команда спрацювала, як очікувалося: шаблон «***.bash****» було перетворено командною оболонкою у список файлів, які починаються на *.bash*, цей список вона отримала у якості параметрів командного рядку ***1S***, після чого його вивела. З «***/bin/e****» повезло, цей шаблон перетворився у список файлів з каталогу ***/bin***, які починаються на «***e***», і першим файлом у списку виявилася утиліта ***/bin/echo***. Оскільки, у командного рядка нічого крім шаблону не було,

саме цей рядок ***/bin/echo*** було сприйнято оболонкою у якості *команди*, якій у якості *параметрів* були передані *інші* елементи списку ***/bin/ed***, ***/bin/egrep***, ***/bin/ex***.

Що ж стосується ***ls *a****, то по нашим розрахункам, ця команда повинна булу видати список файлів у поточному каталозі, ім'я яких вміщує «***a***». замість цього, на екран вивелося ім'я файлу з підкаталогу ***examples …*** Але, ніякої плутанини тут не має. По-перше, імена файлів виду «***.bash****» хоча і вміщують «***a***», але починаються на крапку, і, це означає, вважаються **прихованими**. Приховані файли подають у результат генерації імен тільки, якщо крапка на початку вказана *явно*, (як у першій команді параметру). Тому по шаблону «****a****» у нашому домашньому каталозі **bash** знайшов тільки підкаталог ***examples***, його він і передав у якості параметру утиліті ***ls***. Що ж вивелося на екран у результаті команди, яка утворилася, ***ls examples***? Звісно ж зміст каталогу. Шаблон в останній команді прикладу, «****[ao]****», було перетворено у список файлів, чиї імена вміщують «***a***» або «***o***» - **Documents/ examples/ loop to.sort**, а ключ «***-d***» зажадав у ***ls*** показувати інформацію про каталоги, а не про їх вміст. У відповідності з ключем «***-F***», ***ls*** розставив «/» після каталогів і «*****» після файлів, які виконуються.

Ще одна відмінність генерації імен від стандартної обробки шаблону – тому, що символ «/», який розділяє елементи шляху, *ніколи* не ставиться у відповідність «*****» або діапазону. відбувається це тому, що спотворений алгоритм, а тому, що при генерації імен шаблон застосовується саме до елементу шляху, в середині якого вже немає «/». Наприклад, отримати список файлів, які знаходяться у каталогах ***/usr/bin*** і ***/user/sbin*** і вміщують підрядок «***ppp***» у імені, можна за допомогою шаблону «***/usr/*bin/*ppp****». Одначе, одного шаблону, який би включав у цей список ще і каталоги ***/bin*** і ***/sbin***, тобто підкаталоги *другого* рівня вкладеності – за стандартними правилами, зробити на можна.

Якщо, перед любим спеціальним символом стоїть «\», цей символ позбавляється спеціального значення, **екранується**:пара «***символ***» замінюється командним інтерпретатором на «***символ***» і передається у командний рядок без усіякої подальшої обробки:

Приклад 8.8. Екранування спеціальних символів і обробка «пустих» шаблонів

```
[methody@localhost methody]$ echo *o*
Documents loop to.sort
[methody@localhost methody]$ echo \*o\*
*o*
[methody@localhost methody]$ echo "*o*"
*o*
[methody@localhost methody]$ echo *y*
*y*
[methody@localhost methody]$ ls *y*
```

ls: *y*: No such file or directory

Примітка.

1. Якщо немає файлу (каталогу з такою буквою в імені).
2. Щоби упевнитися – замінити “**y**” на “**z**”.

Ми помітили, що шаблон, якому не відповідає жодне ім’я файлу, **bash** розкривати не став, як би усі «*****» в ньому були екрановані. Справді, яке з двох зол менше: змінять *інтерпретацію* спецсимволів, у залежності від вмісту каталогу, або зберігати логіку інтерпретації з ризиком перетворити команду з параметрами у команду без *параметрів*? Якщо би, припустимо, шаблон, якому не знайшлося відповідності, просто знищувався, то команда **ls *y*** претворился би на ***ls*** і неочікувано видала би вміст всього каталогу. Автори **bash** вибрали більш непослідовний але більш безпечний перший засіб.

Позбавити спеціальні символи їх спеціального значення можна і іншим засобом. Роздільники (пробіли, символи табуляції, і символи переведення рядку) перестають сприйматися такими, якщо частину командного рядку, яка їх вміщує, оточити подвійними або одинарними лапками. В лапках перестає «працювати» і генерація імен (як це видно з прикладу), і інтерпретація інших спеціальних символів. Подвійні лапки, одначе, допускають виконання **підстановок** змінної оточення і результату роботи команди, описаних у двох наступних розділах.

8.3 Оточення

У розділі «Доступ процесів до файлів і каталогів» вже згадувалося, що системний виклик ***fork()***, утворюючи точну копію батьківського процесу, копіює також і **оточення**. Необхідність в «оточенні» відбувається ось з якої задачі. Акт передання даних від *цього конкретного* батьківського процесу дочірньому, і, ще важливіше, системі, повинен володіти властивістю ***атомарності***. Якщо використати для цієї цілі *файл* (наприклад, конфігураційний файл програми, яка запускається), завжди зберігається вірогідність, що за час між зміною файлу і подальшим читанням програмою, яка запущена, хтось – наприклад, інший процес того ж користувача – знову змінить цей файл. Було би добре, якби *зміна* даних і *передання* робилися би *однією* операцією. Наприклад, завести для кожного процесу такий «файл», вміст якого по-перше, міг би змінювати *тільки* цей процес, і, по-друге, ця зміна автоматично копіювалася би у аналогічний «файл» дочірнього процесу при його народжені.

Ці властивості і реалізовані у понятті ***«оточення»***. Кожний процес, який запускається, система наділяє деяким інформаційним простором, який цей процес має право змінювати як йому заманеться. Правила користування цим простором прості: в ньому можна задавати іменовані сховища даних ***(змінні оточення)***, у якому записувати яку завгодно інформацію (присвоювати значення змінній оточення), а подалі, цю інформацію зчитувати (підставляти значення

змінної). Дочірній процес – точна копія батьківського, тому його оточення – також точна копія батьківського. Якщо про дочірній процес відомо, що він використовує значення деяких змінних з числа тих, які передаються йому з оточенням, батьківський може заздалегідь вказати, яким змінним, які копіюються в оточенні, треба змінювати значення. При цьому, з одного боку, ніхто (крім системи. звісно) не зможе втручатися у процес передавання даних, а з іншого боку, одна і таж утиліта може бути використана одним і тим же засобом, але не в зміненому оточенні – видавати різні результатию.

Приклад 8. . Утиліта date користується змінною оточення LC_TIME

```
[methody@localhost methody]$ date
Птн Ноя 5 16:20:16 MSK 2004
[methody@localhost methody]$ LC_TIME=C date
Fri Nov 5 16:20:23 MSK 2004
```

Оточення – набір даних, які приписані системному процесу. Процес може користуватися інформацією з оточення для налаштування, змінювати і доповнювати його. Оточення подану у вигляді ***змінних оточення*** і їх значень. При породженні процесу оточення батьківського процесу спадкується дочірнім (копіюється).

8.3.1 Робота із змінними у shell

В останньому прикладі ми скористалися одним засобом: привласнили деяке значення змінній оточення у командному рядку *перед* іменем команди. Командний інтерпретатор, побачивши «=» в середині першого слова командного рядку, приходить до висновку, що це – операція присвоєння, а не ім’я команди, і запам’ятовує, як треба змінити оточення команди, яка іде після. Змінна оточення ***LC_TIME*** наказує використовувати визначену мову при виведенні дати і часу, а значення «***C***» відповідає «стандартній системній» мові (частіше за все - англійській).

Якщо розглядати ***shell*** у якості високорівневої мови програмування, її змінні – *звичайні* рядкові змінні. Записати значення у змінну можна за допомогою операції присвоювання, а прочитати його звідти – за допомогою операції підстановки виду ***$змінна***.

Приклад 8.10 Підставлення значень змінних

```
[methody@localhost methody]$ A=dit
[methody@localhost methody]$ C=dah
[methody@localhost methody]$ echo $A $B $C
dit dah
[methody@localhost methody]$ B=" "
[methody@localhost methody]$ echo $A $B $C
```

dit dah
[methody@localhost methody]$ echo "$A $B $C"
dit dah
[methody@localhost methody]$ echo ’$A $B $C’
$A $B $C

Як видно з прикладу, значення *невизначеної* змінної (***B***) в **shell** вважається пустим, при підстановці не виводиться ніяких попереджень. Сама підстановка проходить, як і генерація імен, *перед* розбором командного рядку набраного користувачем. Тому, друга команда ***echo*** в прикладі отримала, як і перша *два* параметри («***dit***» і «***dah***»), не дивлячись на те, що змінна ***B*** була до того часу визначена і вміщувала **роздільник**-пробіл. А от третя і четверта команди ***echo*** отримали по одному параметру. Тут позначилася різниця між одинарними і парними лапками у **shell**: в середині парних лапок діє підстановка значень змінних.

Змінні, які командний інтерпретатор **bash** визначає *після* запуску, не належать оточенню, і, стало бути, не успадковуються дочірніми процесами. Щоби змінна **bash** попала в оточення, її треба *експортувати* командою ***export***

Приклад 8.11. Експорт змінних shell в оточення
[methody@localhost methody]$ echo "$Qwe -- $LANG"
-- ru_RU.KOI8-R -----→*ru_UA.UTF-8*
[methody@localhost methody]$ Qwe="Rty" LANG=C
[methody@localhost methody]$ echo "$Qwe -- $LANG"
Rty -- C
[methody@localhost methody]$ sh
sh-2.05b$ **echo "$Qwe -- $LANG"**
-- C
sh-2.05b$ **exit**
[methody@localhost methody]$ echo "$Qwe -- $LANG"
Rty -- C
[methody@localhost methody]$ export Qwe
[methody@localhost methody]$ sh
sh-2.05b$ **echo "$Qwe -- $LANG"**
Rty -- C
sh-2.05b$ **exit**

Примітка.
На практиці «*sh-2.05b»* не показується замість запрошення *sh-2.05b*$ з’являється запрошення **$**

Тут ми утворили нову змінну ***Qwe*** і змінили значення змінної оточення ***LANG***, яка дісталася стартовому **bash** від програми **login**. В результаті,

запущений дочірній процес ***sh*** отримав *змінене* значення ***LANG*** і ніякої змінної ***Qwe*** в оточенні. Після ***export Qwe***, ця змінна була добавлена в оточення і, відповідно, передалась ***sh***.

8.3.2 Змінні оточення, які використовуються системою і командним інтерпретатором

При час сеансу роботи користувача, стартовий командний інтерпретатор отримує від ***login*** досить багате оточення, до якого додаються і особисті налаштування. Подивитися оточення в ***bash*** можна за допомогою команди ***set.*** Більшість раніше визначених змінних використовується або самою командною оболонкою, або утилітами системи, тому їх зміна призводить до того, що оболонка і утиліти починають працювати злегка інакше.

Вельми примітна змінна оточення ***PATH***. В ній утримується список каталогів, елементи якого розділяються двокрапкою. Якщо команда в командному рядку – не особиста команда ***shell*** (на зразок ***cd***) і не подана у вигляді *шляху* до файлу, який запускається, (як ***/bin/ls*** або ***./script***), то shell буде шукати цю команду серед імен файлів, які запускаються, у всіх каталогах ***PATH***, і тільки у них. Точно так будуть постувати і інші утиліти, які використовують **бібліотечну функцію *execlp()*** або ***execvp()*** (запуск програми).

Приклад 8.1 Використання PATH

[methody@localhost methody]$ echo $PATH

/home/methody/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/usr/games

[methody@localhost methody]$ mkdir /home/*methody***/bin** *->курсив замінити на :user1*

[methody@localhost methody]$ mv *to.sort loop script* **bin/** *-> textfile a b*

[methody@localhost methody]$ *script ->якщо не вийде, обновити екран ^D і повторити виклик файлу*

Hello, Methody!

Примітка.

Якщо файлу script (вміст якого є інструкція до використання) немає, можна вивести на екран текст любого текстового файлу, наприклад: ***cat a*** (в каталозі «***bin***»).

На практиці потрібно:

– зробити заміни: *methody* – ***user1***; *to.sort* – ***textfile***; *loop* – ***a***; *script* – ***b;(*** файли, на які робиться заміна, повинні існувати***);***

– ящо файлу *script* (вміст якого є інструкція до виконання) немає, можна вивести на екран текст любого файлу, наприклад: ***cat a*** (в каталозі «***bin***»);

– якщо не вийде – обновити екран (***^D***) і повторити виклик файлу.

Шляхи, які указані в PATH, не зобов'язані існувати насправді. Винайшовши в $PATH елемент ***/home/methody/bin*** (підкаталог ***bin*** домашнього каталогу), ми вирішили, набагато правильніше буде складувати файли, які виконуються, не куди прийдеться, а саму у цей каталог: по-перше, це упорядкує

структуру домашнього каталогу, а по-друге, дозволить запускати ці файли по імені. Але, спочатку, прийшлося цей каталог *утворити*. Порядок, при якому, особисті програми лежать у спеціальному каталозі, значно *безпечніше* порядку, при якому пошук файлу по імені, який запускається, починається з *поточного* каталогу. Якщо у поточному каталозі виявиться програма ***ls***, і цей каталог – не ***/bin***, а, припустимо ***/home/shogun/dontrunme***, варто чекати підступу.

Змінних оточення, які впливають на роботу *різних* утиліт, доволі багато. Наприклад. змінні сімейства ***LC_*** (повний список видається командою ***locate***), яка визначає мову, на якій виводяться діагностичні повідомлення, стандарти на формати дати, грошових одиниць, чисел, засоби перетворення рядків, і тому подібне. Дуже важлива змінна ***TERM***, яка визначає *тип* терміналу: як відомо з лекції «Термінал і командний рядок (2)», різні термінали мають різні управляючі послідовності, тому програми, які бажають ці послідовності використовувати, обов'язково звіряються із змінною ***TERM***. Якщо така утиліта вимагає редагування файлу, цей файл передається програмі, шлях до якої зберігається в змінній ***EDITOR*** (за звичай це ***/usr/bin/vi***, про який мова піде у лекції «Текстові редактори (9)»). На кінець, деякі змінні як ***UID***, ***USER*** або ***PWD*** просто вміщують корисну інформацію, яку можна було би добути і іншими засобами.

Деякі змінні оточення призначені спеціально для ***bash***: вони задають його властивості і особливості поведінки. Такі змінні сімейства ***PS*** (***P****rompt* ***S****tring*). У цих змінних зберігається рядок-підказка, яку командний інтерпретатор виводить урізних станах. Зокрема, вміст ***PS1*** – це підказка , яку **shell** показує, коли виводить командний рядок, а ***PS2*** – коли користувач натискає *Enter*, а інтерпретатор з якої-небудь причини вважає, що введення командного рядку не завершено (наприклад. не закриті лапки). З ***$PS2*** (символ «>») ми вже стикалися у лекціях **2** і **7**.

Приклад 8.13 Використання змінної оточення PS1

[methody@localhost methody]$ cd examples/ → *каталог examples є, якщо його немає, можна*

[methody@localhost examples]$ echo $PS1 з*астосувати який є, наприклад, «ddd».*

[\u@\h \W]\$

[methody@localhost examples]$ PS1="--> "

-->

--> ***Курсивом дописано:***

PS1="\t \w "*/examples*

22:11:47 ~ */examples*

22:11:48 ~*/examples*

22:11:48 ~ */examples* **PS1="\u@\h:\w \$ "**

methody@localhost:~/examples $

methody@localhost:~/examples $

methody@localhost:~/examples $ cd
methody@localhost:~ $

Примітка (на практиці):

– Якщо немає каталогу **examples**, можна застосувати який у нас є. наприклад, «***ddd***»

– Фрагмент

PS1="\t \w "
22:11:47 ~
22:11:48 ~
22:11:48 ~ PS1="\u@\h:\w \$ "

виглядає на практиці так:

-->PS1="\t \w " **/examples**
22:11:47 ~ **/examples**
22:11:48 ~ **/examples**
22:11:48 ~ **/examples** PS1="\u@\h:\w \$ "

Ми експериментували з ***PS1***, паралельно читаючи документацію по **bash** (≪***(bash.info)Printing a Prompt***≫). Виявляється, в цьому командному інтерпретаторі вміст ***PS1*** не просто підставляється при виводі, він ще і перетворюється, дозволяючи виводити всіляку корисну інформацію: ім'я користувача (відповідає підрядку ≪***u**≫**, user**), ім'я комп'ютеру (≪***h**≫, ***host***), час (≪***t**≫, ***time***), шлях до поточного каталогу (≪***w**≫, ***w**ork directory*) і тому подібне. У прикладі ми замінили, на кінець, ≪***W**≫ (опцію, яка показує *останній* елемент шляху, тобто, особисте ім'я поточного каталогу) на «***w**», *повний* шлях, тому що «***w**» володіє властивістю виділяти у повному шляху домашній каталог і замінювати його на «**~**». Таке перетворення значень сімейства ***PS1*** відбувається тільки коли їх використовує **bash** у якості підказки, а при звичайній підстановці цього не відбувається.

8.4 Мова програмування sh

Колись ми вивчили декілька мов програмування, і вже збиралися написати деякі потрібні програми на **Си** або **Python**, одначе, це – не потрібно. Основна частина того, що починаючому користувачеві Linux, робиться за допомогою однієї правильної команди або викликом декількох команд у конвеєрі. Від користувача тільки вимагається оформити рішення задачі у вигляді **сценарію** на ***shell***. Насправді вже самий перший з командних інтерпретаторів ***sh***, був справжньою високорівневої мовою програмування – якщо, звичайно, вважати усі утиліти системи його операторами. При такому підході, від ***sh*** вимагається зовсім не багато: можливість викликати утиліти, можливість вільно

маніпулювати результатом їх роботи і декілька алгоритмічних конструкцій (умови і цикли).

На жаль, програмування на ***shell***, а також, інших, більше потужних мовах, які інтерпретують у Linux, не розміщується в рамках нашого курсу. Так, що поки ми повинні читати документацію по ***bash*** і робити вправи з написання сценаріїв. Зараз ми тільки поверхнево розглянемо властивості ***shell*** як мови програмування і *інтерпретатора* команд. Зауважимо принагідно, що писати сценарії для ***bash*** – непрактично, так як використовуватися вони можуть тільки за допомогою ***bash***. Якщо обмежити себе рамками ***sh***, сумісність з яким об'явлена і в ***bash***, і в z***sh*** і в ***ash*** (найбільше близькому до ***sh*** по можливостям), і інших командних інтерпретаторах, виконуватися ці сценарії зможуть любим з ***sh***-подібних інтерпретаторів, і не тільки в Linux.

8.4.1 Інтеграція процесів

Кожний процес Linux при завершенні передає батьківському **код повернення** (*exit status*), який дорівнює нулю, якщо процес вважає, що його робота була успішною, або *номеру помилки* у протилежному випадку. Командний інтерпретатор зберігає код повернення останньої команди у спеціальній змінній «***?***». Що більше важливо, код повернення використовується в умовних операторах: якщо він дорівнює нулю,умова вважається виконаною, а якщо ні – не виконаною:

Приклад 8.1 Оператор if використовує код повернення програми grep

```
[methody@localhost methody]$ grep Methody bin/script->курсив замінити на /bin/script
echo 'Hello, Methody!'
[methody@localhost methody]$ grep -q Methody bin/script ; echo $?
0
[methody@localhost methody]$ grep -q Shogun bin/script ; echo $?
1
[methody@localhost methody]$ if grep -q Shogun bin/script ; then echo "Yes"; fi
[methody@localhost methody]$ if grep -q Methody bin/script ; then echo "Yes"; fi
Yes
```

Умовний оператор if запускає команду-умову, ***grep -q***, яка нічого не виводить на екран, зате, повертає ***0***, якщо шаблон знайдено, і ***1***, якщо ні. В залежності від коду повернення цієї команди, ***if*** виконує або не виконує *тіло*: список команд, розміщених між ***then*** і ***fi***. Крапка з комою *розділяє* оператори в ***sh***; або вона, або переведення рядку необхідні перед ***then*** і ***fi***, інакше все, що іде після ***grep*** інтерпретатор вважатиме параметрами цієї утиліти.

Багатьма функціями володіє команда ***test***: вона вміє порівнювати числа і рядки, перевіряти ярлик об'єкту файлової системи і наявність самого цього об'єкту. У «***test***» є друге ім'я: «[»(як правило, ***/usr/bin/[*** – символьне або навіть жорстке послання на ***/usr/bin/test***), яке дозволяє оформляти оператор ***if*** у більше звичному вигляді.

Приклад 8.15. Оператор test

```
[methody@localhost methody]$ if test -f examples ; then ls -ld examples ; fi
[methody@localhost methody]$ if [ -d examples ] ; then ls -ld examples ; fi
drwxr-xr-x 2 methody methody 4096 Окт 31 15:26 examples
[methody@localhost methody]$ A=8; B=6; if [ $A -lt $B ] ; then echo "$A<$B"
; fi
[methody@localhost methody]$ A=5; B=6; if [ $A -lt $B ] ; then echo "$A<$B"
; fi
5<6
```

Примітка (на практиці).

Замість **examples** ставити **ddd.**

Команда ***test -f*** перевіряє, чи не є її аргументом файл; оскільки ***examples*** – це каталог, результат – не успішний. Команда ***[-d*** – теж саме, що і ***test –d*** (чи не каталог перший параметр), тільки останнім параметром цієї команди – виключно для краси – повинен бути символ «***]***». Результат успішний і виконується команда ***ls -ld***. Команда ***test параметр1 -lt параметр3*** перевіряє, чи є *параметр1* числом, меншим, ніж (***l****ess* ***t****hen*) *параметр3.* У більше складних випадках оператор if зручніше записувати «драбинкою», виділяючи переведеннями рядків закінчення умови і команди в середині тіла (їх може бути багато).

Другий тип підстановки, яку робить shell, робиться в середині подвійних лапок – це **підстановка виводу** команди. Підстановка виводу має вигляд «***'команда'***» (інший варіант «***$(команда)***»). Як і підстановка значення змінної, вона відбувається *перед* тим, як почнеться розбір командного рядку: виконавши команду і отримавши від неї якійсь текст, shell береться розбирати його, так як би цей текст користувач набрав вручну. Це дуже зручний засіб, якщо те, що виводить команда, необхідно передати самому інтерпретатору:

Приклад 8.16 Підставлення виводу команди

```
[methody@localhost methody]$ A=8; B=6
[methody@localhost methody]$ expr $A + $B
14
[methody@localhost methody]$ echo  -n "$A + $B ="; expr $A + $B
8 + 6 = 14
[methody@localhost methody]$ A=3.1415; B=2.718
[methody@localhost methody]$ echo "$A + $B ="; expr $A + $B
```

expr: нечисловой аргумент
3.1415 + 2.718 = *non-integer argument*
[methody@localhost methody]$ echo "$A + $B" | bc
5.8595
[methody@localhost methody]$ *C=‘echo "$A + $B" | bc‘ (див. примітку)*
[methody@localhost methody]$ echo "$A + $B = $C"
3.1415 + 2.718 = 5.8595
На практиці приклад виглядає так:
[methody@localhost methody]$ A=8; B=6
[methody@localhost methody]$ expr $A + $B
14
[methody@localhost methody]$ echo -n "$A + $B =" expr $A + $B
8 + 6 = 14
[methody@localhost methody]$ A=3.1415; B=2.718
[methody@localhost methody]$ echo "$A + $B =" expr $A + $B
3.1415 + 2.718 = non – integer argument
[methody@localhost methody]$ echo "$A + $B" | bc
5.8595
[methody@localhost methody]$ C=$(echo ”scale=9; $A+$B” | bc) → *scale - точність*
[methody@localhost methody]$ echo "$A + $B = $C"
3.1415 + 2.718 = 5.8595

Спочатку, для арифметичних обчислень забажали скористатися командою ***expr***, яка працює з параметрами командного рядка. З цілими числами ***expr*** працює не погано, і її результат можна підставити прямо у аргумент ***echo***. З дійсними числами вміє працювати утиліта-***фільтр bc***; арифметичний вираз прийшлося сформувати за допомогою ***echo*** і передати по конвеєру, а результат – помістити у змінну ***C***. У багатьох сучасних командних оболонках є вбудована цілочисельна арифметика виду «***$((вираз))***».

8.4.2 Сценарії

У мові **sh** багато уваги було приділено зручності написання **сценаріїв**. Зокрема, параметри командного рядку, які передані сценарію, доступні в ньому у вигляді змінних, імена яких співпадають з порядковим номером параметру.

Приклад 8.17. Використання позиційних параметрів у сценарії
[methody@localhost methody]$ cat > bin/two
#!/bin/sh
echo "Running $0: $*"
$1 $2
$2 $3

^*D*
[methody@localhost methody]$ chmod +x bin/two
[methody@localhost methody]$ bin/two file "ls -ld" examples
Running bin/two: file ls -ld examples
examples: directory
drwxr-xr-x 2 methody methody 4096 Окт 31 15:26 examples
[methody@localhost methody]$ two "ls -s" wc "bin/two bin/loop" junk
Running /home/methody/bin/two: ls -s wc bin/two bin/loop junk
4 bin/loop 4 bin/two
4 9 44 bin/two
1 5 26 bin/loop
5 14 70 итого
6

*Я*к видно з прикладу, форма «***$номер_параметру***» дозволяє звернутися і до нульового параметру – команді, а *весь* рядок зберігається в змінній «*». Крім того, властивості підстановки виконуються *до* розбору командного рядку, що дозволило нам передати у якості *одного* параметру «***ls -ld***» або «***bin/two bin/loop***», а інтерпретатору – розібрати ці параметри на ім'я команди і ключі і два імені файлу відповідно.

На практиці рекомендовано:

Розібрати сценарій, в якому наведені приклади кодів повернення команд і самого сценарію (…~$ - запрошення терміналу):

…~$ cat > exit.sh
#! /bin/bash
echo hello
echo $? *# код повернення = 0, оскільки команда виконалася успішно*
lskdf *# команда, яка не існує*
echo$? *# не нульовий код повернення, оскільки команду виконати не вдалося*
exit 113 *# явна вказівка коду повернення*
^D
…~$ chmod +x exit/sh
…~$ ./exit.sh
Hello
0
./exit.sh: line 4: lskdf: command not found
127
…~$ echo $?
113

Якщо набрати ***…~$ exit***, то відбудеться миттєвий вихід з терміналу (програма обробляє дуже швидко і змінна ***$?*** отримує нуль).

В ***sh*** є і оператор циклу ***while***, формат якого аналогічний ***if***, більше зручний саме у сценаріях оператор обходу списку ***for*** (список ділиться на слова також як і командний рядок – за допомогою **роздільників**.

Приклад 8.18 Використання for і операції «.» (перший for)

```
[methody@localhost methody]$ for Var in Wuff-Wuff Miaou-Miaou; do \
> echo $Var; done
Wuff-Wuff
Miaou-Miaou
```

У другому ***for*** ми скористалися підстановкою виводу команди ***date***, кожне слово якої вивели за допомогою ***echo -n*** в один рядок, а у кінці команди прийшлося вивести одне переведення рядка вручну.

Приклад 8.18. Використання for і операції «.» (другий for)

```
[methody@localhost methody]$cat >fff.sh
>#/bin/sh
>sum=0
>for i in 1 2 3 4 5 6 7 8 9 10
> do sum=$(($sum+$i))
>done
>echo “summa=$sum”
>^D
[methody@localhost methody]$chmod  +x fff.sh
[methody@localhost methody]$ ./fff.sh
[methody@localhost methody]$ summa= 55
```

Друга половина прикладу ілюструє ситуацію, з якою ми зіткнулися під час своїх експериментів: всі змінні, які визначаються у сценарії, кудись зникають після закінчення його роботи. Воно і зрозуміло: для обробки сценарію, кожний раз запускається *новий* інтерпретатор (**дочірній процес**), і всі його змінні належать саме йому і, с завершенням процесу, знищуються. Таким чином досягається відсутність *побічних ефектів*: запускаючи програму користувач може бути впевнений, що та не змінить оточення командної оболонки. Одначе, у *деяких* випадках вимагається зворотне: запустити сценарій, який необхідним чином налаштує оточення. Єдиний вихід – віддавати такий сценарій на обробку *поточному*, а не новому інтерпретатору. (тобто, тому який розбирає команди користувача). Це і робиться за допомогою спеціальної команди «**.**». Якщо раптом у сценарії, який передається, виявиться команда ***exit***, ***exec*** або яка-небудь інша,

яка приводить до завершення роботи інтерпретатору, завершиться саме *поточна* командна оболонка, ніж сеанс роботи користувача в системі може і закінчитися.

На практиці рекомендовано:

```
…$ cat > fff.sh                    # …$ – це запрошення терміналу
>#/bin/sh
>sum=0
>for i in 1 2 3 4 5 6 7 8 9 10
> do
>sum=$(()$sum +$i)
>done
>echo “Summa=$sum”
>^D
…$ chmod +x fff.sh
…$ ./fff.sh
…$ Summa=55
```

Спочатку ***for*** виводить значення аргументів із введеного списку. Далі, ми пишемо файл-сценарій ***fff.sh***, в якому застосовуємо ***for*** для підрахунку суми перших десяти чисел натурального ряду. Потім запускаємо сценарій і отримуємо результат.

Увага!

1. У розділі 8.5 описані налаштування, які суттєво впливають на змінні оболонки і її режими роботи, що може значно змінити роботу терміналу, навіть унеможливити її.
2. У в’язку з цією небезпекою, практикуватися у комп’ютерному класі заборонено.
3. Дозволяється експериментувати тільки на власних ПК.

8.5 Налаштування командного інтерпретатору

Навчившись (головним чином у результаті читання документації і постійних експериментів) утворювати робочі сценарії можна приступити до налаштування командної оболонки, оскільки для цього використовуються саме сценарії.

8.5.1 Прив’язка до клавіш

Виявляється, що *налаштування управляючих клавіш* у ***bash*** не виглядає як сценарій, і навіть немає відношення не тільки до ***bash***, але і до усіх програм, які використовують ***бібліотеку*** термінального введення ***readline***. ***Конфігураційний файл readline*** називається ***.inputrc*** і складається, в основному, з команд виду ***“управляюча_послідовність”: функція***, де *управляюча_послідовність* – це символи, при отриманні яких ***readline*** виконає *функцію* роботи з рядком, який

вводиться. Список функцій ***readline*** можна узнати у bash за командою ***bind -l***, а список всіх *прив'язок* цих функцій до клавіатурних послідовностей – по команді ***bind –p***. Ми вписали в ***.inputrc*** такі два рядки:

Теорія (писати у vim)

Приклад 8.19 Налаштування .inputrc

"\e[5~": backward-word
"\e[6~": forward-word

Згадані у прикладі функції дозволяють переміщувати курсор у командному рядку *по словам*, а **esc**-послідовності повертаються, відповідно, клавішами *Page Up* і *Page Down* віртуальній консолі Linux (сполучення «***e***» означає в ***.inputrc*** клавішу **esc,** тобто «*^[*», символ з **ASCII**-кодом **27**).

До однієї і тієї ж функції ***readline*** можна прив'язати *скільки завгодно* управляючих послідовностей: наприклад, клавіша *Home* робить те ж, що і «*^A*», *Стрілка уверх* – те що і «*^P*», а *Del* – те, що і «*^D*» (тільки не у пустому рядку). Цим частково вирішується проблема *несумісності* управляючих послідовностей терміналів: якщо в якому-небудь терміналі *іншого* типу *Page Up* і *Page Down* будуть повертати інші послідовності, ми добавимо в ***.inputrc*** ще одну пару команд. Правда інструкція радить зовсім відмовитися від редагування ***.inputrc***, а скористатися утилітою ***tput***, яка звертається до змінної **TERM** і базі даних по терміналам **terminfo** і готова видати вірну для любого терміналу інформацію по ***kpp*** (***k****ey* ***p****revious* ***p****age*) і ***knp*** (***k****ey* ***n****ext* ***p****age*). Видачу ***tput*** можна віддати тій же ***bind*** і тримати команду, яка працює на *любому* терміналі: ***bind 'tput kpp'": backward-word"*** (лапки, екрановані зворотною косою рискою передадуться ***bind*** у незмінному вигляді).

8.5.2 Стартові сценарії

Налаштування оболонки – це у першу чергу, налаштування **оточення**. На початку сеансу роботи (при запуску **стартового командного інтерпретатору**) виконується за допомогою команди «**.**» сценарій з файлу із спеціальним іменем ***/etc/profile***. Це – так званий ***загально системний профіль***, стартовий сценарій, який виконується при вході у систему *любого*, хто виконує командну оболонку подібну ***sh***. Далі, виконується ***персональний профіль*** (або просто **профіль**) користувача, сценарій знаходиться у домашньому каталозі і називається ***.profile***. Цей сценарій користувач може змінювати як йому заманеться.

Що стосується ***bash***, то структура його стартових файлів складніша. Перш за все, ***~/.profile*** виконується тільки якщо у домашньому каталозі немає файлу ***.bash_profile*** або ***.bash_login***, інакше, стартовий сценарій береться звідти. В ці файли можна розташовувати команди, не сумісні з іншими версіями **shell**, наприклад, управління **скороченнями** або прив'язку функцій до клавіш. Крім

того, кожний *інтерактивний* (взаємодіючий з користувачем), але не стартовий ***bash***, виконує системний і персональний конфігураційні сценарії ***/etc/bashrc*** і ***~/.bashrc***. Щоб стартовий ***bash*** теж виконував ***~/.bashrc***, відповідну команду необхідно вписати в ***~/.bash_profile***. Далі. Кожний *інтерактивний* (запущений для виконання сценарій) **bash** звіряється із змінною оточення ***BASH_ENV*** і, якщо у цій змінній записано ім'я існуючого файлу, виконуються команди звідти. На кінець, при завершенні стартового **bash**, виконуються команди із файлу ***~/.bash_logout***.

8.5.3 Приклад налаштувань

Нижче наведені приклади конфігураційних файлів, які ми при допомозі інструкції, за бажанням, можемо розмітити у домашньому каталозі.

Приклад 8.2 . Приклад файлу .bash_profile

```
PS1="\u@\h:\w \$ "
EDITOR="/usr/bin/vim"
export PS1 EDITOR
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
```

В цьому файлі викликається ~/.bashrc (якщо він існує

В цьому прикладі називається ***~/.bashrc*** (якщо він існує).

Приклад 8.21 Приклад файлу .bashrc

```
# User specific aliases and functions
if [ -r ~/.alias ]; then
. ~/.alias
fi
# Source global definitions
if [ -r /etc/bashrc ]; then
. /etc/bashrc
fi
```

Ми вирішили, що скорочення зручніше буде зберігати в окремому файлі – ***~/.alias***. Крім того, викликається сценарій ***bashrc***, який ми виявили у каталозі ***/etc***. Цей файл не входить у число файлів, які автоматично виконує **bash**, тому його виконання треба задати явно.

Приклад 8.22 Приклад файлу .bash_logout

```
alias > ~/.alias
```

Помітивши, що команда ***alias*** видає список скорочень утому ж форматі, в якому вони задаються, ми придумали, як обійтися без редагування файлу ***~/.alias***.

З даного моменту. Усі скорочення, які визначені до моменту завершення сеансу роботи, будуть записуватися назад в ***.alias***. Туди попадуть і ті скорочення, що прочиталися під час виконання ***.bashrc***, і ті, що далі були визначені вручну.

Приклад 8.23 Приклад файлу .alias

```
alias l=’ls -FAC’
alias ls=’ls --color=auto’
alias pd=’pushd’
alias pp=’popd’
alias v=’ls -ali’
alias vi=’/usr/bin/vim’
```

Останній запис у файлі ***.alias*** відноситься до інструменту, яким утворюються всі текстові файли: текстовий редактор ***vim***. Про нього піде мова у наступній лекції.

9 ТЕКСТОВІ РЕДАКТОРИ

9.1 Задача текстових редакторів

Після основних утиліт для роботи з файлами і текстом, перша програма, яка знадобиться любому користувачу Linux – це ***текстовий редактор*** (для скорочення – просто ***редактор***). З попередніх лекцій і особистих експериментів вже стало зрозуміло, яке значне місце займають в системі Linux дані в ***текстовому форматі***, тобто ті, що складаються з символів,які можуть бути відображені на екрані терміналу і які може прочитати людина. Одначе, ми поки що можемо працювати з текстом тільки послідовно: рядок за рядком, навіть якщо маємо справу з файлом. Ми не можемо повернутися і відредагувати вже передані системі рядки. Саме для того, щоби працювати з текстовим файлом, як із сторінкою, по якій можна переміщуватися і редагувати текст в любій точці, і потрібні текстові редактори.

Текстовий редактор необхідний користувачу Linuxв першу чергу для того, щоби змінювати налаштування системи або свого оточення, наприклад. shell – при цьому треба буде редагувати конфігураційні файли, які завжди представлені в текстовому форматі. Але і особисті задачі користувача можуть вимагати редагування текстових файлів: наприклад. сценарії і програми, електронні листи, також нотатки для себе, які ми пишемо – це все дані у текстовому форматі. Текстові дані, які отримані за допомогою утиліт, теж буває зручно зберігати у текстових файлах і редагувати.

Не варто плутати текстові редактори і ***текстові процесори***. Текстові процесори, наприклад, – ***OpenOffice.org Writer*** або ***Microsoft Word*** – призначені для утворення текстових *документів*, в яких, крім власне тексту, розміщується різна *мета інформація* (інформація про оформлення): розміщення шрифту на сторінках, шрифт, і тому подібне. Оскільки. В текстовому форматі не передбачено засобів для зберігання інформації про оформлення (там є тільки символи і рядки), текстові процесори використовують свої особисті формати для зберігання даних. Текст в якому немає ніякої інформації про оформлення називають «***plain text***» (тільки текст «***плоский***», просто текст).

Одначе, за допомогою текстових редакторів можна працювати не тільки з форматом ***plain text***. Різна мета інформація (про оформлення, засобі використання тексту, наприклад, в якості посилання та інше) може бути записана у вигляді звичайних символів (тобто, в текстовому форматі), але із спеціальною домовленістю, що ці символи треба інтерпретувати особливим чином: як інструкції по обробці тексту, а не як текст. Такі інструкції називаються **розміткою**. Таким чином об лаштований, наприклад. формат ***html***. Для того, щоби обробити розмітку ***html*** і у відповідність до неї *відобразити* текст, потрібна спеціальна програма – браузер, але *редагувати* файли ***html*** і інші формати розмітки можна за допомогою тестового редактору. Крім того, програми на

любих мовах програмування і сценарії (програми на **shell**), теж уявляють собою текстові файли. Багато текстових редакторів орієнтовані на роботу не тільки з плоским текстом, але і з текстом у різних форматах. Для цього придумано масу удосконалень, які зменшують кількість символів, котрі потрібно вводити вручну: спеціальні команди, клавіатурні скорочення і авто доповнення ключових слів і конструкцій.

Найважливіша умова для текстового редактору в Linux – можливість працювати у терміналі, тому, що це – основний засіб управління системою. Тому і введення даних і редагування повинні повністю здійснюватися засобами терміналу, тобто, алфавітно-цифровими і декількома функціональними клавішами. Оскільки, функціональних клавіш, на які можна розраховувати на любому терміналі зовсім не багато, а команд, які треба давати редактору, - дуже багато, вимагається засіб вводити любі команди обмеженими засобами терміналу. Ця умова, рівно як і вимоги до зручної роботи з різноманітними структурованими текстами, виконана в «двох» головних текстових редакторах Linux – ***vim*** і ***emacs***, по які, в основному, і буде іти мова в цій лекції.

9.2 Редактор Vi і його модифікації

В любій системі Linux, навіть у самій мінімальній, завжди присутній текстовий редактор, оскільки в любій – навіть самій катастрофічній ситуації у користувача, повинна бути можливість відредагувати конфігураційні файли, щоб привести систему до робочого стану. За традицією, текстовим редактором, який обов'язково запуститься з любого командного рядку Linux, є ***vi*** (visual editor).

Проте, вірно і зворотне: якщо ви працюєте в незнайомій системі Linux або вийшов збій, у результаті якого доступна тільки дуже невелика частина системи, не можна бути впевненим, що знайдеться хоч який-небудь інший текстовий редактор, крім ***vi***. Тому, кожному користувачу Linux необхідні хоча би основні навички роботи в ***vi***.

При ***першому*** знайомстві з ***vi,*** робота, зазвичай, не іде: дуже вже він не звичний, його неможна зручно використовувати, запам'ятавши тільки дві-три найпростіші команди редагування. Варто зрозуміти основні принципи роботи в ***vi*** і затратити деякий час на його засвоєння, тоді в ньому відкриється потужний інструмент, який дозволяє дуже ефективно працювати з текстом.

Під іменем ***vi***, насправді, можуть ховатися декілька різних програм: з моменту появи ***vi*** в операційній системі Unix (а це відбулося більше 40 років тому), цей редактор став на зразок стандарту. В даний час, існує ряд програм, які, або точно повторюють вид і поведінку «класичного» ***vi*** (наприклад, ***nvi***), або дуже схожі на нього, але із значними розширеними можливостями (***vim***, ***elvis***). Найбільшою популярністю користується ***vim***, можливості якого величезні, для їх опису знадобилося би майже сто тисяч рядків документації. Коли користувач Linux набирає в командному рядку ***vi***, скоріше всього, буде запущена

«полегшена» версія ***vim***, яка налаштована таким чином, щоби максимально відтворювати поведінку класичного редактору ***vi***. Природно, в такому режимі, частина можливостей ***vim*** недоступна. Всі властивості, якими ***vim*** відрізняється від ***vi***, обов'язково забезпечені в керівництві по ***vim*** вказівкою «***not in vi***». У подальшому викладені матеріалу, під ***vi*** ми будемо розуміти саме ***vim*** в режимі сумісності, всі можливості, які не доступні в цьому режимі, будуть окремо оговорюватися. Щоби викликати ***vim*** у повнофункціональному режимі, достатньо набрати команду ***vim***.

Популярність ***vim*** не випадкова: цей текстовий редактор дозволяє не тільки виробляти прості операції редагування текстових файлів, але і добре пристосований для максимально швидкого і ефективного ряду суміжних з редагуванням задач. Серед самих важливих його можливостей – інструменти для роботи з текстами на різних мовах програмування і різних форматах розмітки. ***Vim*** вміє підсвічувати різними кольорами синтаксичні конструкції мови програмування або розмітки, автоматично виставляти відступи, щоби облегшити сприйняття структури документу. Крім того, в ***vim*** є спеціальні засоби для організації циклу налагодження програми: компіляція – правка вихідного тексту – компіляція … Детальніше про ці і інші можливості можна узнати з керівництва по ***vim***.

9.2.1. Запуск редактора *vi*

Щоби почати сеанс редагування в ***vi***, достатньо виконати команду ***vi*** на любому терміналі. Щоби відкрити для редагування файл, який вже існує, шлях до цього файлу треба вказати у якості параметру: «***vi шлях_до_файлу***». Як люба програма Unix, яка себе поважає, ***vi*** може бути запущений з багатьма модифікуючи ми його поведінку **ключів**, які детально описані в керівництві. Викликаний без параметрів редактор, відкриє пустий **буфер** – чистий лист утворення нового тексту. В центрі екрану може з'явитися коротке привітальне повідомлення, де вказані версія програми і команди для отримання допомоги і виходу з редактору. Проте, таке повідомлення може і не виникнути – це залежить від версії ***vi***, яка встановлена в системі.

Для відображення тексту і роботи з ним ***vi*** використовує весь екран терміналу, тільки останній рядок призначено для діалогу з користувачем: виводу інформаційних повідомлень і виводу команд. Поки буфер на заповнено текстом, на початок кожного рядку екрану відображається символ «~», який позначає, що в цьому місті буферу нічого не має, навіть пустого рядку. Загальний вигляд екрану на початку роботи буде, приміром, такий:

Приклад 9.1 Початок роботи з vi

```
#
~
~
```

~
~
~

Символ «#» позначає курсор. На екрані терміналу вміщується більше рядків, але в прикладах ми будемо для компактності зображати тільки необхідний мінімум.

9.2.2. Режими

В ***vi*** проблема розділення команд редактору і тексту, який вводиться вирішена за допомогою **режимів**: в *командному* режимі, натиск, на любу клавішу – це команда редактору, в режимі *вставки*, натиск на клавішу призводить до вставки відповідного символу у текст, який редагується. Тому, при роботі редактору ***vi***, користувачу завжди треба звертати увагу, в якому режимі знаходиться редактор.

Режими *vi*

Стан редактору ***vi***, в якому він по-різному обробляє натиск клавіш. Розпізнають три режими ***vi*** *командний* (натиск любої клавіші вважається командою і негайно виконується), *вставки* (натиск клавіші друкарського символу призводить до вставки цього символу в текст) і *командний рядок* (для введення довгих команд, які відображаються на екрані; введення завершується натиском на *Enter*).

Vi завжди починає роботу у командному режимі. В цьому режимі є два засоби відати команду редактору. По-перше, натиск практично на любу клавішу редактор сприймає як команду. В ***vim***, навіть режимі ***vi***-сумісності, командне значення визначено для всіх латинських літер (у верхньому і нижньому регістрах), цифр, знаків перетину і більшості інших друкарських символів. При натиску на ці клавіші, команди, які вводяться. Ніде не відображаються, вони просто *виконуються*.

Увага!

Не треба намагатися вводити текст у командному режимі: оскільки, у кожної літери є командне значення. Результат може бути самим неочікуваним.

По-друге, у ***vi*** є свій командний рядок: щоби його викликати, треба ввести «**:**» у командному режимі. В результаті, на початку останнього рядку екрану з’явиться двокрапка – це запрошення командного рядку. Тут виводяться більш складні команди ***vi***, які включають в себе цілі слова (наприклад. імена файлів), причому текст команди, яка набирається, звісно, відображається. Команди передаються ***vi*** клавішею *Enter*. В сучасних версіях ***vi*** з командним рядком можна робити та як і в **shell**: редагувати його, добудовувати команди клавішею *Tab*, користуватися історією команд.

Головна команда командного рядка ***vim*** – виклик підсистеми допомоги «***:help Enter***». Двокрапка переводить ***vim*** в режим командного рядка, *Enter*

передає команду, **help** можна викликати з аргументом: назвою команди або налаштування ***vim***. ***Vim*** дуже добре документовано, тому по команді «***:help об'єкт***» можна отримати інформацію про любу властивість ***vim***, наприклад, команда «***:help i***» виведе відомості про значення клавіші «***i***» у командному режимі ***vim***.

Команда ≪:***set имя_настройки***≫ дозволяє налаштовувати ***vi*** прямо у процесі роботи з ним. Наприклад, віддавши команду ≪***:set wrap***≫ користувач тим самим включає налаштування «***wrap***», що примушує редактор переносити занадто довгі рядки, які не вміщуються у ширину терміналу. Виключити це налаштування можна командою «***:set nowrap***», так що кінці довгих рядків зникнуть за правим краєм екрану.

На кінець, щоби ввести текст, треба перейти з командного режиму у режим *вставки*, натиснувши клавішу «***i***» (від «***i****nsert*» - «*вставка*»). В цей момент в останньому рядку з'явиться інформаційне повідомлення про те, що редактор знаходиться у режимі вставки: ≪***--INSERT--***≫ або ≪***--ВСТАВКА--***≫, в залежності від встановленої мови системних повідомлень.

В режимі вставки можна вводити текст, завершуючи рядок натиском на *Enter*. Проте, треба пам'ятати, що в деяких ортодоксальних версіях ***vi*** в режимі введення не працюють *ніякі* команди переміщення по тексту, тут можна тільки набирати. Якщо ви помітили, що ви помилилися в наборі – не треба відразу намагатися перемістити курсор і виправляти помилку: значно зручніше буде вносити всі виправлення потім, у командному режимі, багато доступно спеціальних команд швидкого переміщення і заміни тексту. Щоби перейти з режиму вставки назад у командний режим, треба натиснути *ESC*.

Увага!

Якщо ***vi*** прийшов в незрозумілий для нас стан, треба натиснути *ESC*, щоби повернутися у командний режим (іноді вимагається натиснути *ESC* двічі).

Зараз можна перейти до управ з ***vi*** на файлі прикладів.

Приклад 9.2 vi в командному режимі

```
methody@oblomov:~ $ vi textfile
Це файл для прикладів.
Приклад 1
~
~
~
"textfile" 2L, 33C 1,1 Весь
```

Vi почав роботу – як і покладено – в командному режимі. В останньому рядку виникли деякі корисні відомості про відкритий файл: його ім'я, загальна кількість рядків («***2L***»), символів («***33C***»), позиція курсору («***1,1***» - номер рядку,

номер символу). «***Весь***» означає, що весь вміст файлу вмістився на екрані терміналу. Тепер ми натиснемо «***i***» і введемо трохи тексту.

Приклад 9.3 vi в режимі вставки

```
Це файл для прикладів.
Приклад 2
Моя перша сторінка в vi!
~
~
~
-- ВСТАВКА -- 3,24 Весь
```

Тепер ***vi*** в режимі вставки: в останньому рядку з'явилося інформаційне повідомлення про це. Набравши текст, ми можемо повернутися у командний режим, Натиснувши ESC (підказка ≪--***ВСТАВКА***--≫ при цьому зникне з останнього рядку).

Насправді, з командного режиму можна перейти у режим введення декількома командами, різниця між ними полягає в тому, у якій точці почнеться введення символів. Наприклад, по командам «***O***» і «***o***» («***o****pen*») можна вводити текст з нового рядку (до або після поточного), по команді «***I***» – з початку рядку, а команди «***a***» і «***A***» («***a****ppend*») передають *додавання* символів (після курсору або у кінець рядку) і т.п.

9.2.3 Робота з файлами

Редагуючи текст в ***vi***, користувач працює безпосередньо не з файлом, а з **буфером**. Якщо відкривається вже існуючий файл, ***vi*** копіює його вміст у буфер і відображає буфер на екрані. Всі зміни, які робить користувач, відбуваються саме у вмісті буферу – відкритий файл поки що залишається не змінним. Якщо ***vi*** викликано без параметрів, то утворюється пустий буфер, який поки що не пов'язаний не з яким файлом.

Щоби записати зміни, які зроблені, у файл, використовується команда «*:**w** Enter*» (щоби її зробити, треба спочатку перейти у *командний* режим). Про те, що «***w***» – це скорочення від англійського «***w****rite*», «*писати*», можна узнати, натиснувши *Tab* після «*:**w***» – і ***vi*** доповнить цю команду до «***write***». Подібним чином можна поступити з більшістю команд у командному рядку ***vi*** в цьому редакторі дуже послідовно дотримується принцип **аббревіативності**. Ми виконали команду «***write***» і отримали таке інформаційне повідомлення.

Приклад 9.4 Запис файлу

```
Це файл для прикладів.
Приклад 1.
Мій перший рядок у vi!
```

~
~
~
~
"textfile" 3L, 57C записано 3,24 Весь

Ми не вказали, куди саме записати вміст буферу, і по замовченню, він був записаний у той файл, який ми відкрили для редагування: «textfile». Проте команді «write» можна вказати любе ім'я файлу, у якості параметру – і тоді вміст буферу буде записано у той файл, якщо такого файлу немає, то його буде утворено. Параметр «write» обов'язково вимагається, якщо текст у буфері ще не записано ні в якому файлі.

Найбільш важлива для новачка команда виходу із ***vi*** – «:qEnter» (скорочення від «quite»). Користувач, який запустив редактор в перший раз, не рідко стикається з тим, що ніяк не може його закрити: не працює жоден із звичних засобів завершення програми, навіть «^***C***» ***vi*** обробляє по своєму. І ≪***:w***≫ і ≪***:q***≫ – команди режиму *командного рядку*, у цей режим ***vi*** переводиться з *командного режиму* за допомогою «***:***» у їх початку.

Проте, якщо в буфері є зміни, які ще не записані ні в якому файлі, то ***vi*** відмовиться виконувати команду ≪***:q***≫, запропонувавши спочатку зберегти ці зміни. Якщо не треба зберігати зміни, наполягти на своєму бажанні і вийти із ***vi***, додавши до команди виходу знак оклику «***:q!***». У цьому випадку всі зміни, які не збережені, будуть викинуті. Знак оклику можна додавати в кінці любої файлової команди в командному рядку ***vi***, в цьому випадку ***vi*** буде виконувати команди, не пред'являючи користувачу ніяких заперечень.

У ***vi*** зроблено багато зусиль для економії зусиль і часу користувача, який керує редактором. Тому, цілком можливо одним разом записати текст і вийти з редактору: командою ≪***:wq***≫ або аналогічною командою ≪***:x***≫ або натиснувши просто ≪***ZZ***≫ в командному режимі.

9.2.4 Переміщення по тексту

При редагуванні в тексті завжди є точка, в якій ми «знаходимося» у даний момент: символи, які вводяться з клавіатури, з'являться саме тут, знищуватися символи будуть теж звідси, до тієї точки будуть застосовуватися команди редактору і т.п. Зазвичай. Така точка буде позначатися курсором, а в останньому (інформаційному) рядку екрану ***vi*** вказує номери поточного рядку і колонки (номер символу в рядку), в якому знаходиться курсор.

Для того, щоби виконувати редагування тексту, по ньому необхідно переміщуватися, тобто, переміщувати курсор. Самий наглядний засіб це робити – користуватися клавішами із стрілочками. Натискання на одну з цих клавіш, зазвичай, примушує курсор переміщуватися на один символ вліво/вправо або на один рядок вверх/вниз. Важко придумати більш не ефективний і повільний засіб

переміщення, якщо треба попасти на інший кінець об'ємного тексту, або навіть, просте переміщення до початку рядка може зайняти декілька секунд.

Зауважимо, що у процесі редагування тексту, зазвичай, виникає необхідність переміщуватися не в довільну точку, деякі ключові: *початок* і *кінець* рядку, слова, речення, абзацу, виразу, укладений в дужки цілий текст. Особливо це помітно, якщо вимагається редагувати *структурований* текст: програму (наприклад, сценарій), конфігураційний файл і т.п. В ***vi*** для кожного такого переміщення передбачені спеціальні команди, зазичай, вони складаються з натиску *однієї* клавіші у командному режимі. Використовуючи їх, можна не тільки однією кнопкою переміститися на любу відстань у тексті, але і рухатися по структурним елементам переходячи до попереднього/наступного слова, речення абзацу, лапкам і т.д.

Увага!

Перш, ніж починати експериментувати з переміщеннями, треба перейти у командний режим.

Отже, пер двинути курсор на початок поточного рядку можна командою «***O***», на перший не пробільний символ в рядку – «^», в кінець рядку – «***$***». Абзацами ***vi*** вважає фрагменти тексту, які розділені пустим рядком. До початку попереднього/наступного абзацу можна попасти командами «***{***» і «***}***» відповідно. Дуже поширена необхідність – попасти у самий кінець файлу: для цього служить команда «**G**» («***G****o*»), у самий початок «***gg***» («***g****o*-***g****o*»).

Пересунути курсор вперед до початку наступного слова можна командою ***≪w≫*** (від «***w****ord*», слово), на початок попереднього – ***≪b≫*** (від «***b****ackward*», назад). До початку попереднього/наступного речень можна переміститися командами «***(***» і «***)***» відповідно. Треба враховувати, що границі слів і речень ***vi*** знаходить за формальними ознаками (керуючись спеціально визначеними ***регулярними виразами***), тому рішення ***vi*** може іноді не співпадати з уявою користувача про границі слів і речень. Проте, користувач завжди може змінити відповідні регулярні вирази, подробиці – в документації з ***vim***.

В ***vi*** ніколи не слід вручну повторювати по декілька разів одну і ту ж команду: якщо треба перейти на три слова вперед, не слід тричі натискати ***≪w≫***, для повторення команди використовується ***множник***. Множник, це любе число, яке набирається перед командою ***vi***: команда буде повторена таке число разів. Наприклад, ≪3***w***≫ – означає «тричі перемістити курсор на слово вперед», інакше кажучи, перемістити курсор на три слова вперед. Треба звернути увагу, що множники можуть застосовуватися не тільки з командами переміщення, а і з *любими* командами ***vi***. Аналогічно, можна перемістити курсор на 10 абзаців у перед командою «***10}***».

Множник – число, яке стоїть перед командою ***vi***, і означає, що дану команду слід виконати вказане число разів.

Не відразу, очевидно, що пошук шаблону в тексті (рядки або регулярні вирази) – це теж команда переміщення. Як і любе переміщення пошук

відбувається в командному режимі: перш за все, треба натиснути «/»: в останньому рядку виникне символ «/». Далі слід ввести шаблон для пошуку, він буде відображатися в цьому рядку, його можна редагувати. Зазвичай, ***vi*** налаштований таким чином, що шаблон для пошуку інтерпретується як регулярний вираз, де ряд символів має спеціальне значення, це налаштування можна виключити («***:set nomagic***»). Після того, як введено шаблон, слід натиснути *Enter* – курсор переміститься до найближчого (далі по тексту) співпадіння з шаблоном. Пошук в зворотному напрямку (до попереднього співпадіння) слід починати з команди «***?***».

Зовсім просто перейти до наступного вживання в тексті того слова, на якому стоїть курсор: для цього треба просто натиснути «*******» в командному режимі. Аналогічна команда пошуку в зворотному напрямку – «***#***». Можна спеціально відмітити в тексті точку і потім, в любий момент, повернутися до неї як до закладки, одну закладку визначає сам ***vi*** – «***‘‘***», місце в тексті де була зроблена остання зміна. Детально про ці та інші команди переміщення можна прочитати в керівництві ***vim*** з по команді ***≪:help usr_03.txt≫*** .

9.2.5 Зміна тексту

В командному режимі не можна вводити символи у текст з клавіатури, але змінити текст при цьому можна, наприклад, *знищуючи* символи. Щоби знищити окремий символ (той, на якому стоїть курсор), достатньо натиснути «***x***» в командному режимі, а щоби знищити зразу цілий рядок (природно, поточний, тобто той, на якому стоїть курсор), «***dd***». «***d***» – це скорочення від слова «***d****elete*», знищити, а «***dd***» – характерний прийом ***vi***: подвоєння команди означає, що її треба застосувати до поточного рядка.

З командного рядка ***vi*** можна виконати операцію пошуку і заміни: для простого рядка або для регулярного виразу. Причому, синтаксис команди пошуку і заміни повністю відтворює синтаксис потокового редактору ***sed*** , про який вже йшла мова в лекції «Робота з текстовими даними (7)».

Приклад 9.5 Заміна по шаблону в vi

```
Це файл для прикладів.
Приклад 1.
Мій перший рядок у vi...
~
~
~
:s/.$/.../
```

Ми забажали замінити у своєму файлі крапки в кінцях рядків трьома крапками. Для цього, в командному режимі, ми натиснули «***:***» (виклик командного рядку ***vi***), де набрали команду «***s***» (скорочення від «***s****ubstitute*», замінити), за якою іде вже знайоме слово по «***sed***» вираз «***що_замінити/***

на_що_замінятиме». Тільки результат вийшов зовсім не той, якого ми очікували: замінився на три крапки знак оклику в останньому рядку. Ми не врахували наступного: за замовченням, команди-шаблони для пошуку і заміни – це регулярні вирази, тобто «**.**» означає зовсім не крапку, а «любий символ», маючи на увазі крапку, слід написати «**.**» «***$***», як ми і очікували – це кінець рядку. На момент виконання команди пошуку, курсор знаходився в останньому рядку, в першому співпадінні після курсору і була зроблена заміна.

Форматування тексту – це розставлення кінця рядку , пробілів і табуляцій таким чином, щоби текст мав гарний вигляд на екрані терміналу. Робити форматування вручну вкрай не ефективно. В ***vim*** автоматичне форматування тексту (якщо редагується програма на якій-небудь мові програмування, з врахуванням правил цієї мови) може відбуватися прямо в режимі вставки. В режимі вставки можна змінювати *відступ* поточного рядку (по командам «^***T***» і «^***D***»). Для вирівнювання тексту по центру, правому, або лівому краю є команди «***:center***», «***:left***» і «***:right***» відповідно. Ці команди, як і більшість команд командного рядку, можна застосовувати для **діапазону рядків** засобом, який описано нижче.

Режим введення не багатий спеціальними командами зміни тексту, що і зрозуміло: він призначений саме для *введення*, і більше ні для чого. Проте, в ***vim*** (але не в ***vi***) є деякі зручності, управляють і самим процесом набору. Якщо слово, яке треба ввести, вже зустрічалося в тексті, можна набрати тільки перші літери і натиснути «^***P***» («***p****revious*») – ***vim*** спробує сам завершити його. Якщо ***vim*** не вгадав і запропонував не те слово, можна продовжити перебирати варіанти. Команда «^***N***» («***n****ext*») підставляє слова, які зустрічалися нижче по тексту. Детальніше про цю функцію можна узнати з керівництва по команді ≪:***help ins-completion***≫.

Іноді, змінивши текст, відразу хочеться повернути все назад, для цього в ***vi*** передбачена команда відміни останніх змін: «***u***» в командному режимі (від «***u****ndo*», відмінити). Наскільки би складною , масштабною (і руйнівною) не була би зміна, вчинена останньою командою, «***u***» поверне текст в початковий стан. Втім, саму відміну теж можна відмінити. В класичному ***vi*** доступна відміна тільки останньої команди, яка була виконана, в ***vim*** відмінити можна скільки завгодно останніх команд, а також, повторити їх командою «^***R***».

9.2.6 Робота з фрагментами тексту

Люба команда переміщення визначає дві точки в тексті: та в якій був курсор до переміщення, і та, в яку він перемістився у результаті даної команди. Розташований між цими двома точками відрізок тексту однозначно задається командою переміщення. Наприклад, команда «***)***» захоплює текст від поточного положення курсору до початку наступного речення. ***vi*** дозволяє застосувати до цього фрагменту любу команду – так влаштовані **гніздові команди**. Гніздова команда складається з *дії* і наступного за нею переміщення. Переміщення задає

фрагмент тексту, а дія визначає, що з цим фрагментом робити. Наприклад, команда «***d)***» знищить весь текст від поточної позиції курсору до початку наступного речення. Найбільше корисні дії «***d***» (***d****elete*), «***c***» (***c****hange*), «>» і «<» (*зсунути*) , «***y***» (*запам'ятати*) і «***gq***» (*від форматувати*).

Гніздова команда – команда редактора ***vi***, яка дозволяє застосувати указану дію до вказаного відрізку тексту. Відрізок задається стандартною командою переміщення по тексту.

Дуже часто виникає необхідність замінити фрагмент в тексті: слово, речення, рядок і т.п. Це можна зробити у дві дії: спочатку знищити частину тексту, потім перейти у режим вставки і вставити заміну. ***vi*** дає можливість спростити цю операцію, зводячи дві дії до однієї: замінити. Гніздова команда «***c***» призначена саме для цього. Наприклад, команда «***cw***» (буквально, ***c****hange* ***w****ord*) замінить текст від курсору до початку наступного слова (так можна замінити одне слово), «***c)***» замінить текст від курсору до початку наступного речення.

Ми не забули, що команди переміщення можна використовувати з ***множниками***, і спробували замінити відразу три слова в своєму файлі на інші: для цього ми в командному режимі підігнали курсор до початку слова «первая» і набрали «***c3w***» («замінити фрагмент з цього місця до початку третього слова», буквально «***c****hange* ***3*** ***w****ords*»). Результат цієї команди виглядає так

Приклад 9.6 Команда заміни в vi

```
Це файл для прикладів.
Приклад 1.
Мій  #vi...
~
~
~
-- ВСТАВКА --                  3,5       Весь
```

В прикладі знак «***#***» означає позицію курсору: як видно ***vi***, знищив три слова, які попали в зону дії команди і відразу перейшов у режим вставки. Нам залишилося тільки набрати щось взамін.

Перестановка частин – типова задача, яка виникає у процесі редагування. Для переміщення вимагається знищити фрагмент в одному місці тексту і вставити його в іншому. Для рішення першої частини задачі в ***vi*** не знадобиться спеціальних засобів, тому, що люба команда знищення («***c***», «***d***», «***x***» і інші) зберігає знищений текст у спеціальному **регістрі**. Для вставки останнього знищеного фрагменту служить команда «***p***» в командному режимі (від «put», покласти). За допомогою цієї команди ми можемо вставити три слова, які тільки що були знищені командою «***c3w***»:

Приклад 9.7 Вставка знищеного фрагменту

```
Це фацл для прикладів.
```

Приклад 1.
Мій новий рядок у vi...
перший рядок у
~
~

4,1 Весь

Для того, щоби скопіювати фрагмент тексту, служить команда «***y***» (від «***yank***», забрати , зірвати): вона не знищує текст а просто зберігає його в тому ж регістрі, що і команди управління. Команду «***y***» можна використовувати в гніздових командах, наприклад. «***y5w***» збереже в регістрі фрагмент тексту від курсору до початку п'ятого слова. Вставити скопійований фрагмент можна все тією ж командою «***p***». Проте, таким засобом тільки текст, знищений або скопійований *останнім*, для зберігання декількох різних фрагментів тексту слід використовувати іменовані регістри (детальніше у керівництві по ***vim***).

Для того, щоби застосувати команду до кількох *рядків* тексту, не обов'язково підганяти до них курсор. У командному рядку ***vi*** любій команді може передувати вказівка **діапазону** в тексті, до якого слід цю команду застосувати. Команди з вказівкою діапазону виглядають так: «***: початок, кінець команда***», де *початок* і *кінець* – це адреса початкового і кінцевого рядків діапазону (тобто, фрагменту тексту), *команда* – це команда режиму *командного рядку*, така як «***:w***» або «***:r***». Багато команд *командного* режиму (зокрема, «***d***» і «***y***») доступні також і в командному рядку. В якості адреси можна використати номери рядків у файлі (команда «***:1, 5y***» буде означати «скопіювати у регістр рядки з першого по п'ятий»), є спеціальні позначення для поточного рядку («**.**»), останнього рядку («***$***») і усього файлу («***%***»). Вказати границю діапазону можна за допомогою шаблону: граничним буде вважатися той рядок, в якій виявиться шаблон.

Останньою властивістю ми скористувалися, щоби знищити плоди своїх експериментів: ми виконали команду «***: /Пример 1/, $d***» (від рядка «***Пример 1***» до кінця файлу – знищити).

Приклад 9.8 Знищення діапазону по шаблону. Виконана команда ≪:/Приклад 1/,$dEnter≫

Це файл для прикладів.
~
~
~
fewer lines

9.2.7 Налаштування vi і vim

Вид і поведінку ***vi*** і ***vim*** можна суттєво змінювати за допомогою налаштувань, пристосувавши редактор до своїх смаків і звичок. Прямо під час

роботи редактору можна змінювати налаштування з командного рядку ***vi*** за допомогою команди «***:set ім'я_ налаштування***». Крім того, можна зробити налаштування постійним, вписавши всі необхідні значення в конфігураційний файл ***.vimrc*** (або ***.exrc*** – для ***vi***) в домашньому каталозі користувача. При кожному запуску ***vi***/***vim*** читає цей файл і виконує всі команди, які є в ньому. Це – дуже великий об'єм інформації, який виходить за межі даного курсу. Щоби оцінити можливості налаштування, можна виконати в ***vim*** (але не в ***vi***) команду «***:options***», по якій буде виведено список всіх допустимих опцій з коротким описом їх змісту.

9.3. Редактор Emacs

Редактор ***emacs*** є одноліток редактору ***vi***. Обидва цих редактори з'явилися більше сорока років тому, але їхній поважний вік тільки пішов їм на користь: велика кількість користувачів у всьому Світі усі ці роки займалася їх від лагодженням, локалізацією і розширенням.

Vim і ***emacs*** утворюють альтернативну пару не тільки за історичним випадком: обидва редактори претендують на роль універсального засобу для роботи з текстом на любих природних і штучних мовах. І дійсно, важко назвати текстовий редактор, який може порівнятися з ними по можливостям, та ще і настільки не вимогливий до інтерфейсу. ***Vim*** і ***emacs*** будуть працювати на любому терміналі. Проте, обмежені можливості інтерфейсу терміналу вимагають від програм надійного засобу відділення команд редактору від тексту, який вводиться. В ***Vim*** і ***emacs*** ця задача вирішена по-різному – звідси велика частина різниць у стилі роботи з цими редакторами звідси і традиційні суперечки прихильників цих редакторі проте, котрий з них кращий.

9.3.1. Тексти на різних мовах

Головна властивість, яка зробила ***emacs*** таким популярним і багато функціональним редактором – це закладена в ньому з самого початку принципова *розширюваність*. ***Emacs*** майже повністю написаний на спеціально утвореній для нього мові програмування ***Emacs Lips***, і у любого користувача є можливість запрограмувати любі необхідні *саме йому* функції і підключити їх у якості модуля ***emacs***. При цьому сам ***emacs*** ніяк змінювати не вимагається. Сукупність програмістів відразу скористувалася розширюваністю ***emacs***, і в наш час, найважливішим добутком цього редактору є саме у вільно розповсюджуваних пакетах розширень, які вміщують інструменти для рішення самих різнобічних задач, пов'язаних з редагуванням тексту.

Сучасний ***emacs*** – це не просто текстовий редактор, а інтегроване середовище для *роботи в системі*. Головна ідея розробників і користувачів ***emacs*** полягає в тому, що ***emacs*** дозволяє працювати з любими даними, які можуть бути подані як текст (в лекції «Робота з текстовими даними (7)»

обговорювалося, що таким чином подати дуже багато у системі). Природно, список файлів, які є в каталозі, програма на любій мові програмування або електронний лист – це тексти, які дуже відрізняються за структурою і тому, що в них треба користувачу. В ***emacs*** для роботи з текстом різного типу використовуються ***режими***.

Режим *emacs* – комплекс команд і налаштувань ***emacs***, призначений для роботи з текстом визначеної структури , наприклад, вмістом каталогу, програмою на Сі і т.п.

Кожний ***буфер*** в ***emacs*** знаходиться в одному з ***основних режимів***. Основний режим – це набір функцій і налаштувань ***emacs*** пристосованих для редагування тексту визначеного виду. Кожний основний режим по-своєму визначає деякі управляючі символи, так що найбільше доступними стають команди, частіше всього потрібні саме для роботи з текстом даного типу. Команди специфічні для поточного основного режиму, зазвичай починаються з управляючого символу ***C-c***. Деяку уяву про можливості ***emacs*** може дати неповний список тих текстів, для яких існують основні режими:

- список файлів у каталозі;
- програми на самих різних мовах програмування, від Сі до самих екзотичних;
- тексту в різних форматах розмітки: xml, html, tex;
- словники;
- електронна пошта (режим дозволяє не тільки читати і писати листи, але і відправляти і получати їх);
- календарі;
- щоденники і особистий розклад;
- багато іншого.

Коли вимагається багато і швидко працювати з текстом на якій-небудь штучній мові (мова програмування, розмітки і тому подібне), можливо ***emacs*** – найкращій вибір.

Увага!

Звернути увагу, що в ***emacs*** поняття «режим» має зовсім інший зміст ніж у ***vi***.

Не менше добре в ***emacs*** розвинуті засоби роботи з текстами на самих різних природніх мовах з самими екзотичними письменностями. Просто для оцінки можливостей ***emacs*** в цій галузі, можна виконати команду «***C-h h***», по якій буде виведено файл, який зображає вітання на різних мовах.

9.3.2 Команди *emacs*

Якщо в системі встановлено ***emacs***, то можна запустити його, набравши ***emacs*** в командному рядку любого терміналу. Як і ***vi***, ***emacs*** використовує весь екран терміналу, хоча, інтерфейс у нього багатий: зверху екрану знаходиться рядок з пунктами меню, під ним – вікно для відображення і редагування тексту,

яке закінчується **рядком режиму**, в якому відображаються відомості про те, що відбувається у вікні. У самому низу екрану – рядок **міні буферу** , який використовується для відображення і редагування команд, що вводяться.

Vi і за ним ***vim*** – це *багато режимні* редактори, коли команди вводяться в одному режимі, а текст – в іншому, що дозволяє використовувати у якості командних любі клавіші. В ***emacs*** немає спеціального командного режиму, але використано той факт, що з клавіатури можна вводити не тільки друкарські, але і деякі **управляючі символи**. Для цього використовуються декілька управляючих клавіш терміналу (перш за всі *Ctrl* і *Alt*), натиснуті у сукупності з різними текстовими символами. Щоби ввести такий символ, треба натиснути управляючу клавішу (наприклад, *Ctrl*) і, утримуючи її, натиснути клавішу з одним з друкарських символів (наприклад «***x***»). Крім того, в ***emacs*** використовується управляюча клавіша *Meta*, на тих терміналах, де вона відсутня, її функції, зазвичай, передаються клавіші *Alt*. На «справжніх» терміналах зазвичай не буває ні *Meta* ні *Alt*; з **клавіатурних модифікаторів** присутні тільки *Ctrl* і *Shift*. Тоді на допомогу приходить *ESC*: натиск на *ESC*, а *після* неї – на друкарський символ (той же «***x***») еквівалентно «*Meta x*».

Команд в редакторі ***emacs*** дуже багато, доступних управляючих символів на всіх не вистачає, тому щоби викликати команду ***emacs***, зазвичай, треба ввести **ключ**, який починається з управляючого символу, за котрим іде комбінація з управляючих або звичайних символів, або просто, ім’я команди. Послідовність символів, достатня для виклику команди, називається **закінченим ключем**, а якщо введених символів не достатньо для однозначного визначення команди це – **префіксний ключ**.

Загальне правило тут таке: чим частіше потрібна команда, тим коротше ключ, який її викликає, і навпаки. Для лаконічного запису довгих клавіатурних комбінацій у користувачів ***emacs*** склалася особлива традиція скорочених позначень. Клавішу *Ctrl* позначають великою літерою «*C*», *Meta* –«*M*». Сполучення з командною клавішею позначають дефісом, наприклад. запис: *C-h* позначає, що треба, утримуючи *Ctrl*, натиснути «*h*». *C-h* – це префіксний ключ для команд довідкової системи ***emacs***. Користувачу-початківцю варто виконати команди «*C-h ?*» (набрати *C-h* і потім натиснути «*?*») – довідка по командам допомоги, «*C-h t*» - інтерактивний підручник для користувачів-початківців ***emacs***, і «*C-h i*» - повне керівництво по ***emacs*** (в форматі ***info***). З ключа *C-x* починаються основні команди ***emacs***, зокрема, для роботи з файлами і буферами. Щоби завершити роботу ***emacs***, треба ввести «***C-x C-c***».

У любої команди ***emacs*** є особисте ім’я, за цим іменем можна викликати команду навіть, якщо вона не прив’язана не до якого клавіатурного ключа. Для виклику команд по імені використовується префіксний ключ *M-x*. Наприклад, переглянути довідку про допомогу ***emacs*** можна командою «***M-x help-for-help***».

9.3.3 Робота з файлами

В ***emacs*** як і в ***vi***, користувач редагує текст не в самому файлі, а в **буфері**. Відміна ***emacs*** в тому, що неможна написати «безіменний» текст і потім, зберегти його у файлі. При запуску ***emacs*** без параметрів, відкривається спеціальний буфер – «****scratch****», він призначений для тимчасових заміток – його вміст буде викинуто при закритті ***emacs***. Якщо треба утворити новий файл – його слід *відкрити* командою «***C-x C-f***», точно також відкривається для редагування вже існуючий файл.

Після того, як ми натиснули «***C-x C-f***», в **міні буфері** виникло запрошення «***Find file: ~/***». Тепер треба ввести **шлях** до файлу, починаючи з поточного каталогу (***emacs*** люб'язно підказав йог нам). З текстом в міні буфері можна оперувати майже так, як з командним рядком ***shell*** або ***vim***: редагувати, використовувати авто заповнення (клавішею *Tab*), переміщуватися по історії стрілочками вверх/вниз. Ми скористалися цією можливістю і набрали «***te***», натиснули *Tab* і *Enter*, почали редагувати файл «***textfile***». Зберегти зміни, які зроблено, можна командою «***C-x C-s***».

Коли ми забажали відкрити ще один буфер, щоби відредагувати один із своїх сценаріїв, ми запам'ятали точну назву необхідного файлу і набравши «***C-x C-f bin/»*** натиснули *Enter*. При цьому, в вікні виник список файлів в підкаталозі «***~/bin***», схожий на вивід ***ls -l***

Приклад 9.9 Emacs. Режим dired

```
 File Edit Options Buffers Tools Operate Mark Regexp Immediate Subdir Help
/home/methody/bin:
итого 24
drwxr-xr-x 2 methody methody 4096 Дек 2 15:21 .
drwx------ 10 methody methody 4096 Дек 2 15:21 ..
-rwxr-xr-x 1 methody methody 26 Ноя 9 21:34 loop
-rwxr-xr-x 1 methody methody 23 Ноя 9 21:34 script
-rwxr-xr-x 1 methody methody 32 Ноя 9 21:34 to.sort
-rwxr-xr-x 1 methody methody 44 Ноя 9 21:34 two
-RRR:%%-F1 bin (Dired by name)--L5--C51--All--
Reading%%directory%%/home/
methody/bin/...done
```

Як указано в рядку режиму, це ***Dired***, редактор каталогів, **режим *emacs***, призначений для перегляду і зміни каталогів прямо в редакторі. В ***Dired*** можна вибирати окремі файли, або групи файлів і проводити над ними різні дії: відкривати і редагувати, знищувати, копіювати, переміщувати, перейменовувати за визначеною схемою – отже ***Dired***, доволі потужний засіб для наявної роботи з файловою системою, особливо він зручний для роботи з групою файлів. Подробиці про команди, доступні в цьому режимі, можна знайти в керівництві з ***emacs***.

9.3.4 Переміщення по тексту

В ***emacs***, як і в ***vi***, є поняття **точки**. Точка – місце в буфері, де буде проходити вставка або знищення даних. Переміщення по тексту – це переміщення точки. Команди переміщення по структурним елементам тексту розвинуті не менше, ніж у ***vi***, крім звичайних стрілок, діють команди переміщення на початок і кінець рядку (***C-a*** і ***C-e***), буферу (***M-<*** і ***M->***), речення (***M-a*** і ***M-e***); до попереднього і наступного слова (***M-f*** і ***M-b***), абзацу (***M-{*** і ***M-}***). Різні **основні режими** представляють спеціалізовані команди для переміщення по структурним елементам текстів на різних мовах програмування, розмітки і іншого.

В ***emacs*** декілька видів пошуку: існують окремі команди для пошуку рядку і пошуку за регулярним виразом. Якщо вимагається знайти найближче вживання конкретного слова, зручніше за все скористатися ***нарощуваним пошуком*** по команді ***C-s***. Нарощуваний пошук вже зустрічався нам: так був облаштований пошук по історії команд в ***bash***. По мірі набору перших символів рядка, який шукається, ***emacs*** переносить точку до такого найближчого сполучення символів після курсору. Пошук в зворотному напрямку (до початку буферу) здійснюється командою ***C-r***. Нарощуваний пошук можна виконувати за регулярним виразом (***C-M-s***). Всі види нарощуваного пошуку в ***emacs*** не відрізняють великих літер від малих.

9.3.5 Зміна тексту

В ***emacs*** доступно багато команд, які економлять зусилля при редагуванні тексту. Якщо користувач усвідомлює, що набрав щось не вірно, то можна знищити разом (*M-Del*) останнє слово, речення (*C-x del*). Можна знищувати і вперед: до кінця слова (*M-d*) і речення (M-k). ***Emacs*** зберігає не тільки останній знищений фрагмент, а і усі попередні, формуючи список знищень. Тільки що знищений текст можна вставити командою *C-y*. Після цього його можна замінити попереднім знищеним фрагментом – *M-y*. можна рухатися і далі назад по списку знищень, повторюючи *M-y*.

Добре продумані команди для пер становлення части тексту навколо точки:двох знаків (*C-t*), слів (*M-t*), рядків (*C-x C-t*). Команда *M-t* не переміщує знаки перетину між словами, тому «*помiха, дiло*» перетвориться в «*дiло, помiха*».

Прямо з ***emacs*** можна викликати програму перевірки орфографії («***M-xispell-buffer***») або, навіть включити в перевірку «*у польоті*», коли невірно написані слова виділяються іншим кольором. («***M-x flyspell-mode***»). Можна перевірити напис окремого слова, в якому знаходиться точка (***M-x $***) або завершити недописане слово, опираючись на орфографічний словник (***M-x Tab***).

В ***emacs*** так багато спеціальних команд для зміни тексту, що команди пошуку і заміни бувають потрібні не так часто. Провести заміну рядку всюди у буфері можна по команді «***M-x replace-string що замінити Enter***

на_що_замінити Enter», для заміни регулярного виразу – аналогічна команда «***M-x replace-regexp***».

Якщо треба замінити рядок тільки у деяких випадках, стане в нагоді команда ***M-%***, яка запитує підтвердження про заміну при кожному знайденому рядку. Аналогічна команда для регулярних виразів – ***C-M-%***. Любі зміни в тексті можна відмінити командою ***C-_*** (треба натиснути ***Ctrl***, ***Shift*** і ≪-≫).

9.3.6 Робота з фрагментами тексту

Багато команд ***emacs*** працюють з довільним фрагментом тексту буферу. Такі команди завжди застосовуються до поточної **області**. Область – це вирізок тексту між **точкою** (де знаходиться курсор) і **міткою**. Мітка в любий момент присутня в любому буфері, користувач може встановити її в любому місці тексту явно – командою ***M-Пробел***. Мітка може переміщатися і без втручання користувача: команди переміщення і редагування можуть змінювати положення мітки. Таким чином, щоби виділити в буфері фрагмент тексту, можна провести наступні операції:

- перемістити точку (курсор) на один кінець нашого фрагменту;
- натиснути ***M-Пробел*** (встановити мітку);
- неремістити точку до другого кінця нашого фрагменту.

Зараз можна виконати команду редагування – вона буде застосована саме до виділеної **області**. Наприклад, ***C-w*** знищить текст області, ***M-w*** а скопіює його. Вставити знищений або скопійований фрагмент можна командою ***C-y***.

Є група команд, які більш ефективно дозволяють працювати з міткою: встановити мітку після кінця наступного слова (***M-@***), помітити поточний абзац (***M-h***), весь буфер (***«C-x h»***). Різні основні режими представляють команди для помітки структурних елементів тексту, наприклад, розділів документу, визначення функції (в тексті програми) і т.п. всі положення мітки зберігаються в **списку поміток**, перенести точку в любе з попередніх положень мітки можна так: необхідну кількість разів повторити команду «***C-u C-@***».

Область – безперервний відрізок тексту, обмежений точкою з одної сторони і міткою з іншої.

Як і в ***vi*** в ***emacs*** можна використовувати для зберігання інформації **регістри**. В регістрі ***emacs*** можна зберігати позицію в буфері і потім перейти до цієї позиції (***≪C-x r Пробіл x≫*** записує позицію точки у регістр ***≪x≫***, ***а ≪C-x r j x≫*** переходить в цю позицію). В регістрі можна зберігати текст з області (***≪C-x r s x≫*** зберігає область у регістрі ***≪x≫***, ***≪C-x r i x≫*** – вставляє текст з цього регістру). В регістрах також можна зберігати числа, імена файлів, конфігурацію вікон. Подробиці завжди доступні в керівництві.

9.3.7 Налаштування *emacs*

Кажучи коротко, в ***emacs*** можна налаштувати все: зв’язки між **ключами** і командами редактору, визначити макрокоманди, написати особисті розширення.

Є можливість міняти налаштування ***emacs*** як у процесі роботи, так і за допомогою конфігураційного файлу ***.emacsrc***.

9.4 Підсумки аналізу текстових редакторів

І в **vi** і в **emacs** інтегровано багато засобів для автоматизації процесу редагування. Ці редактори стають зручними в тому випадку, якщо, перш за все, перед тим як братися робити щось вручну, користувач звертається до керівництва і находіть в ньому засіб виконати свою задачу максимально швидко з мінімальним втратами ручної праці. Проте, якщо користувача не влаштовує такий принцип роботи (коли треба часто і довго читати документацію і думати, щоб ручну працю виконував комп'ютер), **vi** і **emacs** будуть не найкращим вибором. Для звичайного редагування тексту вручну, краще вибрати однин з текстових редакторів із звичним інтерфейсом: в дистрибутивах Linux можна знайти велику кількість таких текстових редакторів з більшими або меншими можливостями: mcedit, joe, pico (частина поштової програми pine) – всіх не перерахувати. Є редактори, які призначені для роботи не в терміналі, а в графічних середовищах (наприклад, nedit), у тих же **vi** і в **emacs** є графічні варіанти (GVim і Emacs-X11 або XEmacs), в яких доступні додаткові можливості графічного користувацького інтерфейсу: меню, іконки, і так далі.

ВИСНОВОК

В даному навчальному посібнику предметом розгляду є ОС Linux, яка складає основу системного програмного забезпечення

Посібник розкриває зміст сеансу роботи в Linux, виводить основні поняття терміналу і командного рядку, надає структуру файлової системи Linux, показує основні принципи роботи з файловою системою, має інформацію про процеси та доступи до об'єктів обробки, знайомить з основними прийомами роботи з текстовими даними та можливостями командної оболонки і описує базові відомості про текстові редактори vi і emacs.

Ця основа знань і умінь надає можливість утворити базу на якій створюються умови якісно проводити подальші навчання в області операційних середовищ, систем і оболонок, а також, адміністрування системами.

Автори постаралися додати матеріалу практичну спрямованість. Викладення супроводжується великою кількістю практичних прикладів. Даний посібник може розглядатися як посібник для студентів, які починають навчання в області операційного системного забезпечення і ще не знайомі з ОС Linux.

ЛІТЕРАТУРА

1. Брайан Уорд. Внутреннее устройство Linux. С-П., Питер, 2016.
2. Дэниел Барретт «Карманный путеводитель по Linux» 3-е изд. Озон, 2016.
3. Сервер, ядро Linux. [Елетронний ресурс]. – Режим доступу: www.linuxhq.com LinuxHQ
4. News for the Open Source Professional [Елетронний ресурс]. – Режим доступу: http://www.linux.com
5. Welcome to Our Community [Елетронний ресурс]. – Режим доступу: http://www.linux.org
6. Європейський Web-сервер Linux [Елетронний ресурс]. – Режим доступу: http://www.linux.org.uk.
7. Основний сервер з документацією з Linux [Елетронний ресурс]. – Режим доступу: http://www.linuxdoc.org
8. Щоденні повідомлення про всі новинки програмного забезпечення для Linux [Елетронний ресурс]. – Режим доступу: http://freshmeat.net
9. 2018 KDE Connect Development Sprint
10. KDE Connect Development Sprint [Елетронний ресурс]. – Режим доступу: http://www.kde.org