КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«ПОГЛИБЛЕННЯ КУРСУ ПРОГРАМУВАННЯ В UNIX-СИСТЕМАХ. УТИЛІТА AWK»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Павло Середюк _____

Науковий керівник:
доктор технічних наук, професор
Сергій Погорілий _____

Рецензент:
Іван Коломієць

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 51 сторінка, 4 рисунків, 1 додаток, 6 джерел посилань.

AWK, ФІЛЬТРАЦІЯ ТЕКСТОВИХ ДАНИХ, СТРУКТУРА ТА ПРИНЦИП РОБОТИ, ОБРОБКА ТА ОЧИЩЕННЯ, ПОШУК ТА АНАЛІЗ, СИСТЕМНЕ АДМІНІСТРУВАННЯ, АВТОМАТИЗАЦІЯ

Об’єктом роботи є дослідження та застосування утиліти AWK. Предметом роботи є фільтрування текстового потоку даних, його обробка.

Метою роботи є:

— Написання навчально-методичних матеріалів для проведення контролю знань студентів при вивченні курсу «програмування в UNIX-системах» тема: «утиліта фільтрації AWK». Мною було створено 2 посібника: посібник для студентів, що містить 30 завдань для створення сценаріїв різного рівня складності. Перші 5 є початковим рівнем, наступні 5 легкого рівня, 10 середнього та 10 важкого рівня; посібник для викладача, із розв’язками 30 сценаріїв із посібника для студентів

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

AWK — утиліта для обробки тексту

sed — потоковий текстовий редактор

grep — утиліта, яка знаходить на вводі рядки, що відповідають заданому регулярному виразу

JSON — файли, що складаються з пар ключ-значення, де кожен ключ - це рядок, а значення може бути рядком, числом, нулем, масивом або іншим об'єктом

CVS — файли, що зберігають табличні дані в структурованому форматі відокремленими комами або іншим розділювачем

ЗМІСТ

ВСТУП

У сучасному світі постійно генерується величезний обсяг даних, які збираються та зберігаються із неймовірною швидкістю. Однак ці дані часто не структуровані, хаотичні і їх важко аналізувати без належних інструментів та методів зберігання та подання.

Обробка та аналіз даних мають вирішальне значення для розкриття сутності та цінності, прихованих у цих величезних масивах даних. Застосовуючи статистичні та обчислювальні методи, можливо виявити закономірності даних, які можуть стати основою для прийняття рішень. Однак для цього потрібні потужні інструменти, такі як мови програмування, які здатні впоратися з обсягом і складністю сучасних даних.

Утиліта AWK — один з таких інструментів, який набув популярності в останні роки завдяки своїй ефективності фільтрувати та обробляти великі масиви даних. Ця утиліта є найбільш наближеною до мови програмування, вона працює із текстовими даними. Її універсальність і гнучкість зробили її популярним методом для аналізу даних, використовується у UNIX-подібних системах.

Результати роботи мають дослідити використання AWK для фільтрації та аналізу великих наборів даних. Демонструючи функціонал та можливості утиліти, продемонструвати актуальність і важливість обробки та збору даних у сучасному світі, а також показати, як такі потужні інструменти можуть допомогти із розкриттям цінності інформації, яка прихована у величезному наборі даних.

РОЗДІЛ 1. АНАЛІЗ ЛІТЕРАТУРИ

1.1. Задача фільтрації даних

Фільтрація є важливим завданням в аналізі даних, що передбачає вилучення певної інформації з великого набору даних, вона допомагає спростити і зрозуміти зміст.

Без фільтрації, дані будуть важко аналізуватись, що призведе до неточних висновків. Вона гарантує, що обробляються лише необхідні дані, що може заощадити час і ресурси, а також підвищить точність аналізу. Фільтрація даних також корисна для видалення небажаної інформації, адже у наборах даних часто трапляються дубльовані або помилкові дані, які можуть вплинути на результат аналізу. Фільтрація допоможе виявити і видалити ті дані, які ми не бажаємо аналізувати у свої роботі.

Існують різні методи фільтрації даних, кожен з яких має свої переваги та недоліки. Наведу декілька прикладів:

— Відповідність шаблону: цей метод передбачає пошук певних шаблонів у наборі даних для ідентифікації та вилучення релевантних даних. Пошук за шаблоном можна здійснювати за допомогою регулярних виразів або інструментів текстового пошуку.

— Фільтрація на основі правил: Цей метод передбачає використання заздалегідь визначених правил для фільтрації даних. Ці правила можуть базуватися на певних критеріях, таких як діапазони дат, категорії продуктів або сегменти клієнтів.

— Фільтрація на основі машинного навчання: Цей метод передбачає використання алгоритмів машинного навчання для виявлення та вилучення релевантних даних. Моделі машинного навчання навчаються на наборі даних і можуть бути використані для прогнозування того, які дані є релевантними для аналізу.

1.2. Аналіз інструментальних засобів фільтрації

Інструменти фільтрації є важливими в обробці даних, оскільки вони допомагають витягнути релевантну інформацію з великих наборів даних, створюючи при цьому чистий набір даних для аналізу. Використовуючи правильний інструмент фільтрації, аналітики даних можуть покращити якість свого аналізу та отримати точніші висновки з даних.

Існує кілька інструментів фільтрації, кожен з яких має свої переваги та недоліки. Наведу декілька прикладів:

— UNIX редактор sed: є інструментом обробки тексту, який зазвичай використовується для операцій пошуку та заміни. sed надає простий синтаксис для фільтрації та перетворення текстових даних. sed ідеально підходить для роботи з малими та середніми наборами даних і може використовуватися для виконання простих завдань фільтрації.
— UNIX команда grep: це інструмент командного рядка, який зазвичай використовується для пошуку текстових даних на основі певних шаблонів. grep надає простий синтаксис для фільтрації текстових даних на основі певних критеріїв, таких як текст або числа. grep ідеально підходить для роботи з великими наборами даних і може використовуватися для виконання складних завдань фільтрації.

Інструменти фільтрації є важливими для аналізу даних, оскільки вони допомагають виокремлювати релевантну інформацію з великих наборів

даних. Існують різні інструменти фільтрації, кожен з яких має свої переваги та недоліки. Далі ми поговоримо про таку утиліту як Linux AWK-універсальний інструмент, придатний як для обробки тесту, так і для аналізу.

1.3. Фільтрація текстових даних за допомогою AWK (LINUX)

AWK - це інструмент обробки тексту, який використовується в UNIX-подібних операційних системах. AWK є універсальним інструментом, який можна використовувати для різних завдань обробки тексту, включаючи фільтрацію, пошук і перетворення текстових даних. AWK надає простий синтаксис для обробки текстових даних і ідеально підходить для роботи з великими наборами даних.

AWK надає потужний набір функцій фільтрації, які можна використовувати для вилучення потрібної інформації з текстових даних. Наведу декілька прикладів функцій фільтрування AWK:

— Регулярні вирази: AWK надає потужний механізм регулярних виразів, який можна використовувати для фільтрації текстових даних на основі певних шаблонів. Регулярні вирази - це набір правил, які визначають шаблон пошуку, що дозволяє легко фільтрувати текстові дані на основі певних критеріїв, таких як текст або числа.

— Роздільники полів: AWK надає можливість розділяти текстові дані на поля на основі вказаного роздільника. Це дозволяє легко фільтрувати текстові дані на основі певних полів, наприклад, стовпців у наборі даних. Тобто, ми можемо використовувати AWK для фільтрації набору даних, на основі значень у певному стовпчику.

— Умовні оператори: AWK надає можливість фільтрувати текстові дані на основі певних умов. Це полегшує вилучення потрібної інформації з

текстових даних. Наприклад, ми можемо використовувати AWK для фільтрації набору даних на основі певних дат або текстових значень.

— Маніпуляції з рядками: AWK має набір функцій маніпулювання рядками, які можна використовувати для фільтрування текстових даних. Ці функції дозволяють маніпулювати та витягувати певні частини рядка, що полегшує фільтрацію текстових даних на основі певних критеріїв.

AWK надає потужний набір функцій фільтрації, які можна використовувати для вилучення потрібної інформації з текстових даних. AWK - універсальний, швидкий, гнучкий і портативний інструмент, який ідеально підходить для роботи з великими наборами даних.

1.4. Порівняння AWK з іншими інструментами обробки текстів

Порівняємо AWK із вже вище згаданими sed та grep.

AWK надає гнучкий синтаксис, який дозволяє користувачам легко виконувати складні завдання фільтрації. sed і grep мають простіший синтаксис, який ідеально підходить для виконання простих завдань фільтрації. AWK і grep - швидкі інструменти, які можуть швидко обробляти великі набори даних. sed працює повільніше, ніж AWK і grep, і ідеально підходить для роботи з малими і середніми наборами даних. AWK надає універсальний набір інструментів для фільтрації, пошуку та перетворення текстових даних. sed в основному використовується для операцій пошуку і заміни, тоді як grep використовується для пошуку текстових даних на основі певних шаблонів. AWK є складнішим інструментом, ніж sed і grep, що робить його ідеальним для виконання складних завдань фільтрації. sed і grep - простіші інструменти, які ідеально підходять для виконання простих завдань.

Отже, AWK, sed і grep - це потужні текстові редактори, які надають потужний набір інструментів для фільтрації та обробки текстових даних. Хоча

вони мають деякі спільні риси, вони також мають відмінності в синтаксисі, швидкості, функціональності та складності. Розуміючи ці відмінності, аналітики можуть вибрати правильний інструмент для своїх конкретних потреб в обробці тексту і підвищити якість аналізу.

РОЗДІЛ 2. ОСОБЛИВОСТІ ПРОГРАМУВАННЯ AWK

2.1. Структура AWK-програм

Програми AWK зазвичай виконуються з командного рядка, а код програми міститься у файлі. Базова структура AWK-програми складається з низки правил, які застосовуються до кожного рядка вхідних даних. Кожне правило складається з шаблону і дії.

Шаблон визначає умови, які повинні бути виконані для того, щоб дія була виконана. Шаблон може бути регулярним виразом, оператором порівняння або діапазоном значень. Якщо шаблон не вказано, дія виконується для кожного рядка вхідних даних.

Дія визначає інструкції, які будуть виконані, коли шаблон буде знайдено. Дія може бути однією командою або рядом команд, укладених у фігурні дужки. Команди можуть бути будь-якими допустимими командами AWK, включно з друком, присвоєнням змінних і виконанням арифметичних операцій.

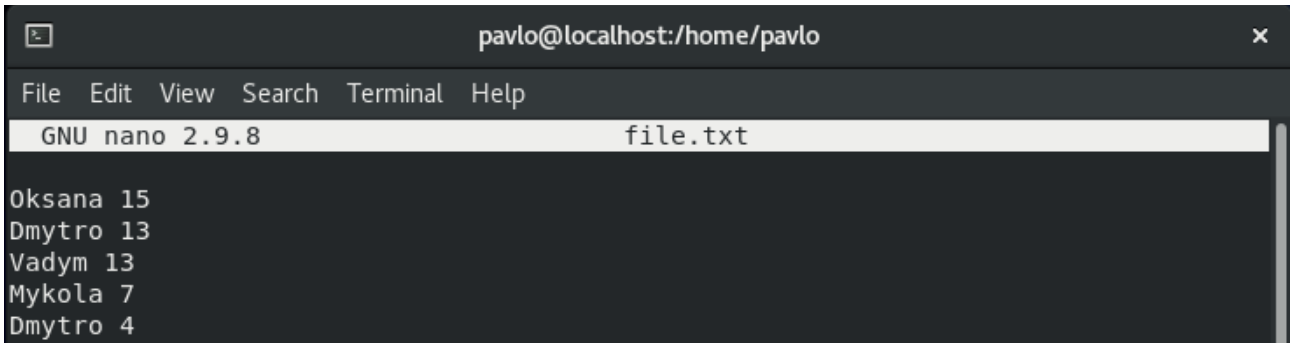
Нижче, наведено приклад простої програми AWK, яка виводить перше поле кожного рядка вхідних даних:

```
[root@localhost pavlo]# awk '{print $1}' file.txt
```

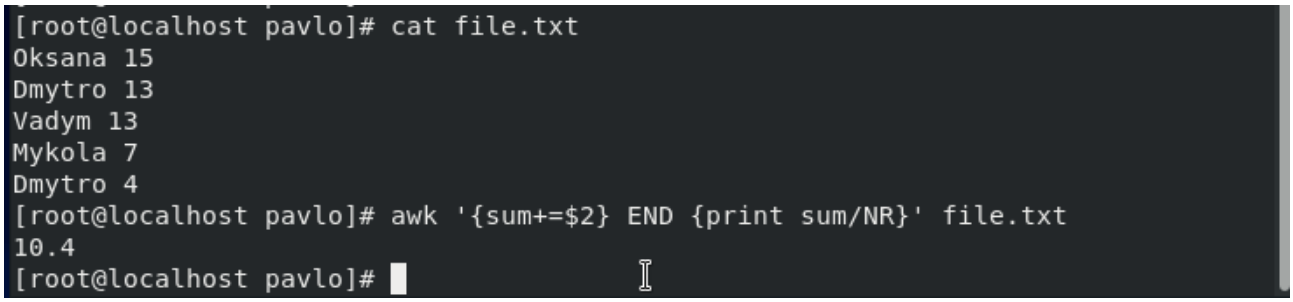
У цьому прикладі шаблон не вказано, тому дія виконується для кожного рядка вхідних даних. Дія складається з однієї команди, яка виводить перше поле кожного рядка.

Структура AWK-програм може бути складнішою, з декількома правилами і діями. У таких випадках правила обчислюються по порядку, і виконується перше правило, шаблон якого збігається з вхідними даними.

Нижче, наведено приклад більш складнішої програми, яка обчислює середнє значення другого поля кожного рядка вхідних даних. Маю ось такий файл із вхідними даними:



Тепер використаємо шаблон для обчислення середнього значення в 2 полі кожного рядку:



У цьому прикладі є дві дії розділені ключовим словом END. Перша дія додає друге поле кожного рядка до змінної з назвою sum. Друга дія, що виконується після обробки всіх вхідних даних, виводить середнє значення другого поля шляхом ділення суми на загальну кількість вхідних рядків. Значення NR в AWK, є загальною кількістю рядків у вхідному файлі. Вбудовані зміні детально розглянемо в наступному розділі.

Як з’ясувалось, програми AWK структуровані за допомогою серії правил, які застосовуються до кожного рядка вхідних даних. Кожне правило складається з шаблону та дії, причому шаблон визначає умови для виконання дії, а дія - інструкції, які потрібно виконати. Програми AWK можуть бути простими або складними, залежно від вимог завдання аналізу текстових даних.

2.2. Змінні та типи даних

Змінні використовуються для зберігання значень даних в AWK. Вони схожі на контейнери, які зберігають значення і можуть бути використані пізніше в програмі. Змінні можна використовувати для зберігання будь-якого типу даних, включаючи числа, рядки і масиви. AWK підтримує три типи змінних: вбудовані, визначені користувачем і поля.

— Вбудовані змінні: AWK має декілька вбудованих змінних, які можна використовувати у програмі без їх оголошення. Ці змінні мають наперед визначені значення і можуть використовуватися для зберігання системної інформації, специфічної для програми інформації та інформації про введення/виведення. Ось деякі поширені вбудовані змінні:

- NR: вказує на поточний номер запису, який обробляється. Іншими словами, змінна відстежує кількість вхідних записів, які вже оброблено програмою AWK.
- NF: вказує на кількість полів у поточному рядку, який обробляється. Поля - це частини запису, розділені роздільником полів, який задається змінною FS.
- FS: позначає символ або рядок, який використовується для розділення полів у кожному вхідному записі. За замовчуванням, роздільником полів є пробіл, але його можна встановити на будь-який символ або рядок за допомогою змінної FS.
- RS: позначає символ або рядок, який використовується для розділення рядків у вхідних даних. За замовчуванням, розділювачем записів є символ нового рядка, але його можна встановити на будь-який символ або рядок за допомогою змінної RS.
- FILENAME: Ім'я поточного вхідного файлу.

— Змінні задані користувачем: змінні, що створюються користувачем під час використання утиліти. Вони призначені для зберігання значень для наступних використань у програмі. Можна присвоїти значення за допомогою оператора «=» і використовувати будь-де у програмі. AWK є динамічно типізованою мовою, тому не потрібно оголошувати тип даних при створенні змінної.
— Поля: Поля - це стовпчики у вхідних даних, розділені роздільником. У AWK поля автоматично розділяються на окремі змінні та викликаються символом «$» та відповідним номером поля. Наприклад, якщо у мене є запис з двома полями, розділеними пробілом, я можу отримати доступ до цих полів за допомогою змінних $1 і $2. Що цікаво, змінна $0 містить в собі весь запис поточного рядку, включаючи всі поля і роздільники полів.

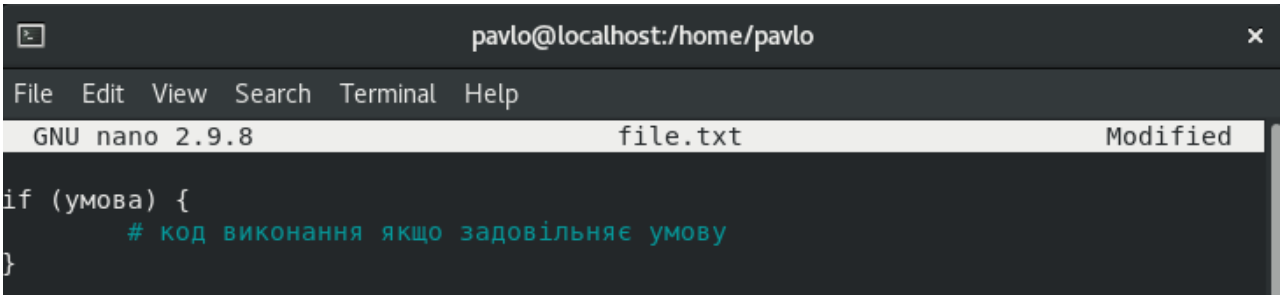
AWK підтримує декілька типів даних, зокрема числові, рядкові та масиви.

— Числовий: числові типи даних використовуються для зберігання чисел. AWK підтримує два числові типи даних: цілі та з плаваючою комою. Цілі типи даних використовуються для зберігання цілих чисел, тоді як типи даних з плаваючою комою використовуються для зберігання десяткових значень.
— Рядковий: рядкові типи даних використовуються для зберігання тексту. Рядки беруться у подвійні лапки.
— Масивний: масиви використовуються для зберігання декількох значень в одній змінній. В AWK масиви створюються шляхом вказівки індексу та значення. AWK підтримує одновимірні та багатовимірні масиви.

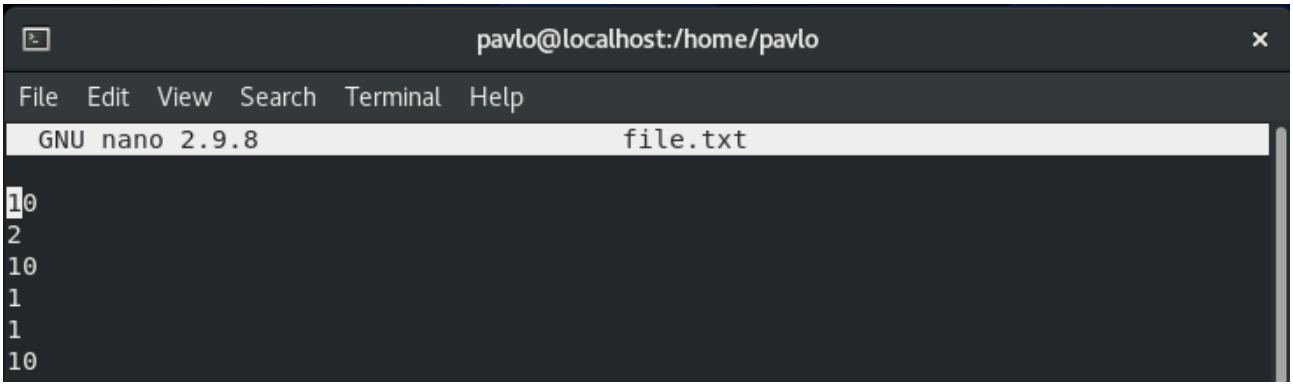
2.3. Умовні оператори та цикли

Однією з ключових особливостей утиліти AWK — можливість використання умовних операторів та циклів для керування потоком програми.

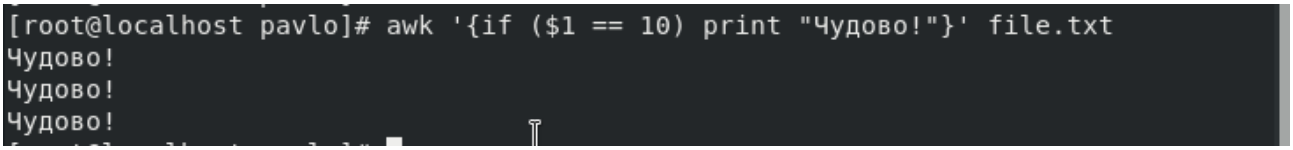
Умовні оператори в AWK дозволяють виконувати певні блоки коду на основі обчислення логічного виразу. Приклад синтаксису оператора if:



Наступна програма виводить повідомлення «Чудово!», якщо зустрічає у вихідному файлі зустрічає значення «10». Вхідний файл:

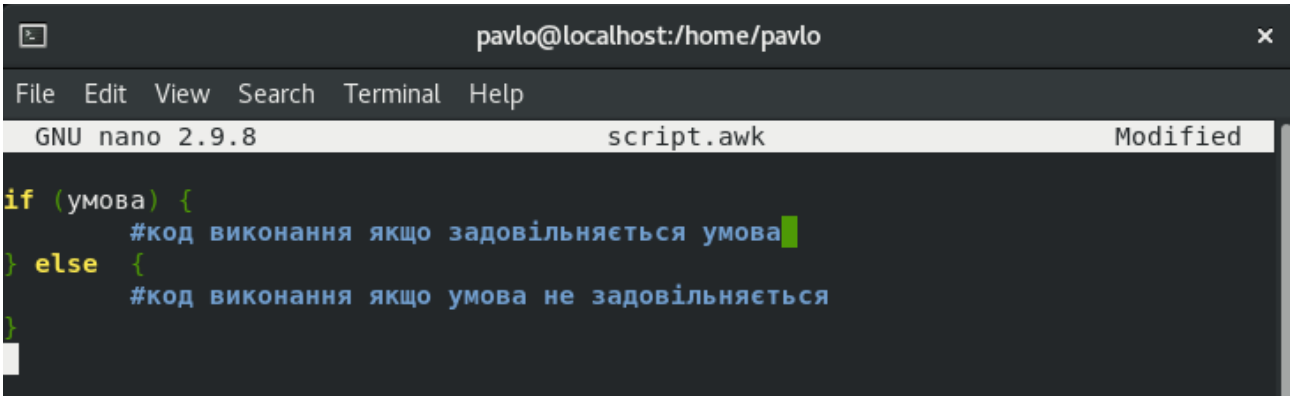


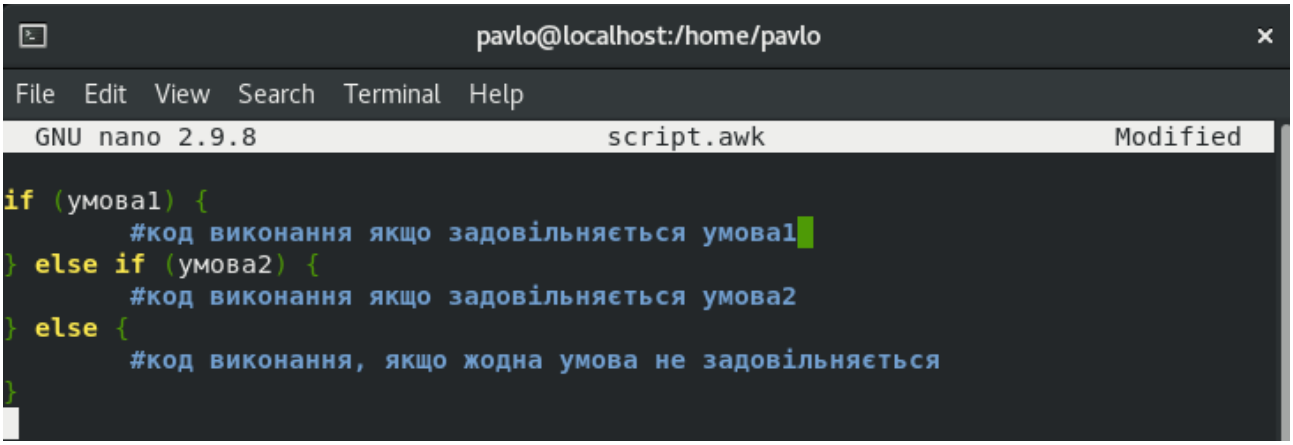
Застосовуємо програму:



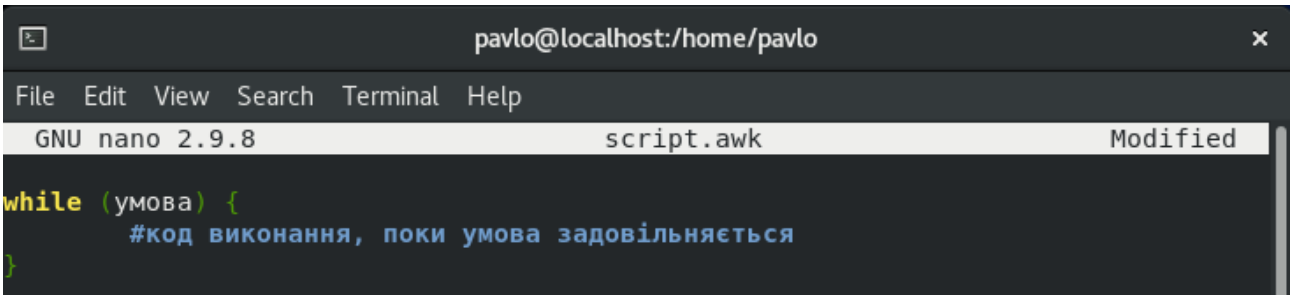
Оператор if перевіряє, чи перше поле $1 дорівнює значенню "10". Якщо це так, то програма виводить повідомлення «Чудово!».

AWK також підтримує використання ключових слів else та else if для створення складніших умовних операторів. Приклад синтаксису:

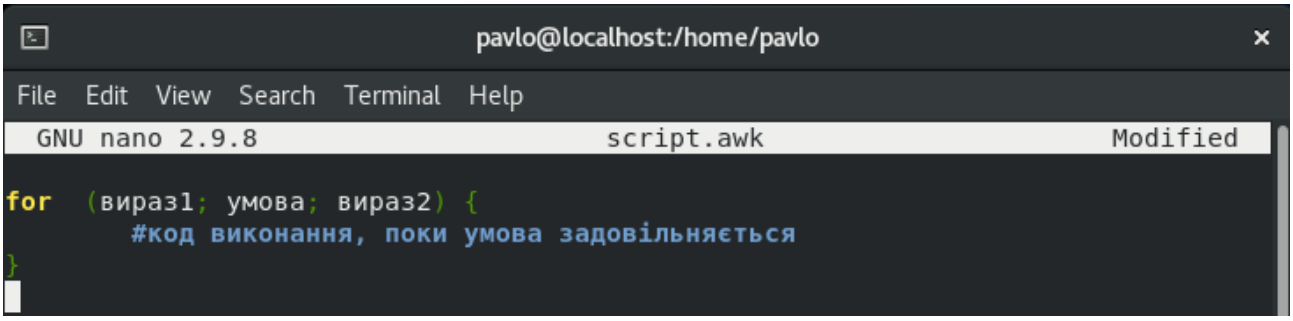




Цикли в AWK дозволяють повторювати блок коду, поки виконується певна умова. В AWK існує два типи циклів: цикли while і for. Синтаксис циклу while:



Синтаксис циклу for:

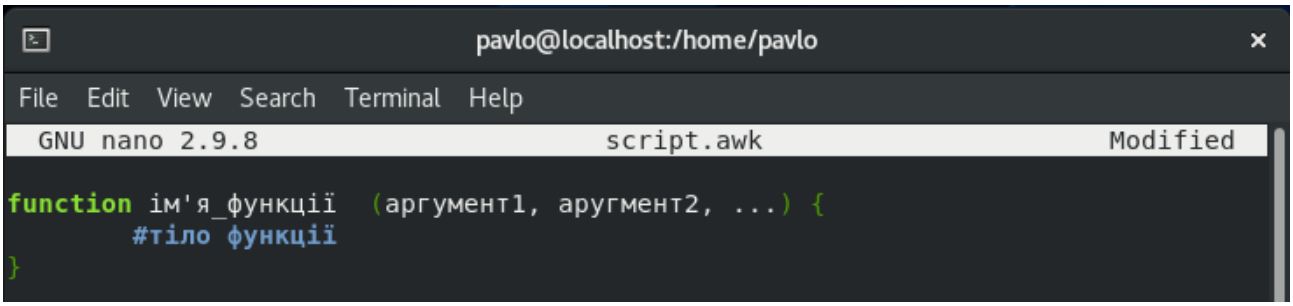


Зміні та типи даних дозволяють керувати програмним потоком і є важливими і корисними для багатьох завдань із обробки тексту.

2.4. Функції та масиви

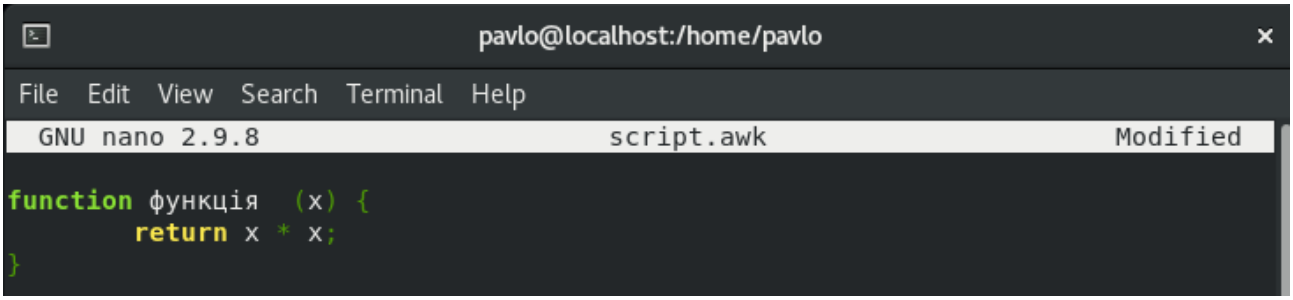
Функції в AWK використовуються для виконання певних завдань або операцій над даними. Вони визначаються за допомогою ключового слова function і можуть бути викликані пізніше у програмі. Функція може приймати нуль або більше аргументів і може повертати значення.

Нижче, наведено приклад синтаксису функції в AWK:



Ім'я функції повинно починатися з літери і може бути комбінацією символів і символів підкреслення. Функції можуть приймати декілька аргументів, розділених комами, але це не є обов'язковим. Тіло функції складається з одного або декількох операторів AWK.

Нижче, наведено приклад функції, що обчислює квадрат числа:

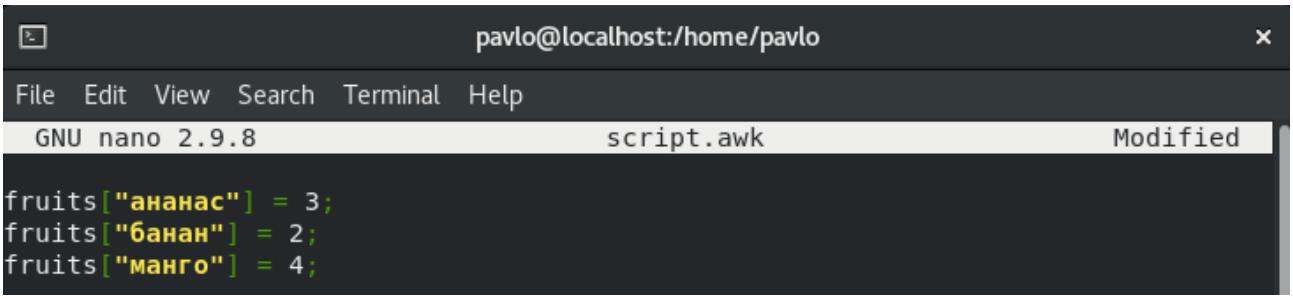


У цьому прикладі функція приймає один аргумент x і повертає квадрат x. Також, цю функцію можна викликати пізніше у програмі.

Масиви в AWK використовуються для зберігання наборів даних. Вони оголошуються за допомогою квадратних дужок ([]) і можуть індексуватися за допомогою числових або рядкових значень.

AWK підтримує єдиний набір назв, які можна використовувати для іменування змінних, масивів та функцій. З цього випливає, що не можливо мати змінну і масив із однаковими іменами в одній програмі.

Зараз продемонструємо приклад масиву та його використання в AWK. Маю масив fruits, що зберігає назви фруктів та їх поточну кількість:



Далі, можна отримати доступ до значення із масиву наступним чином:

```
print fruits["банан"]; # вивід: 2
```

Функції використовуються для виконання певних завдань або операцій над даними, в той час як масиви використовуються для зберігання даних.

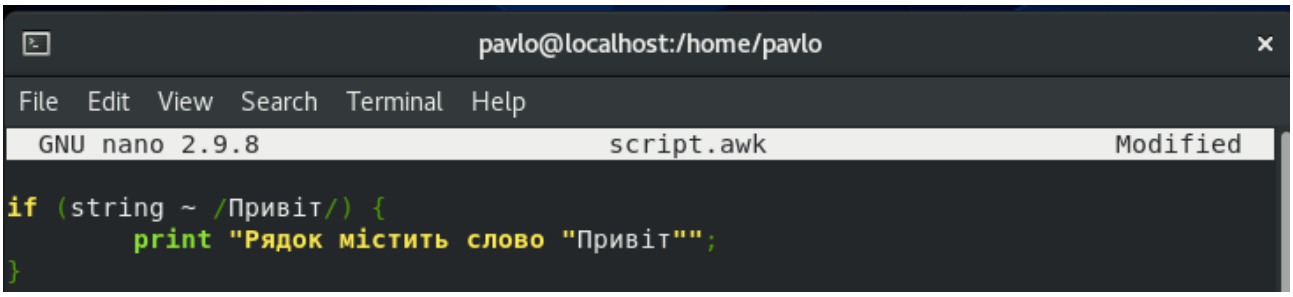
РОЗДІЛ 3. ПОГЛИБЕННІ МЕТОДИ AWK-ПРОГРАМУВАННЯ

3.1. Регулярні вирази та пошук за зразком

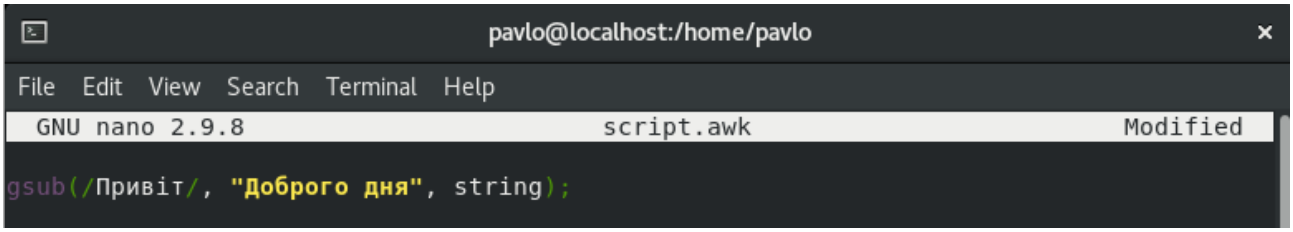
Регулярні вирази використовуються для зіставлення і маніпулювання текстом на основі певних шаблонів і послідовностей. В AWK регулярні вирази укладаються у прямі косі риски «/ /» і можуть використовуватися у різних контекстах, зокрема в операціях пошуку і заміни.

Нижче, наведено декілька прикладів використання регулярних виразів в AWK:

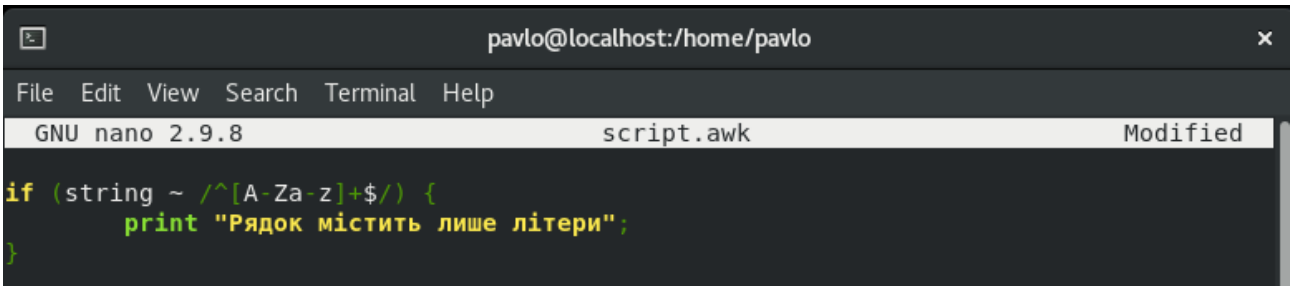
— Пошук певного шаблону в вхідному файлі за допомогою регулярного виразу: Щоб зіставити певний шаблон у рядку, використаємо оператор ~ з регулярним виразом, укладеним у пряму похилу риску. Наприклад, наступний код буде виводити повідомлення «рядок містить слово «Привіт»» у будь-якому рядку, що містить слово "Привіт":



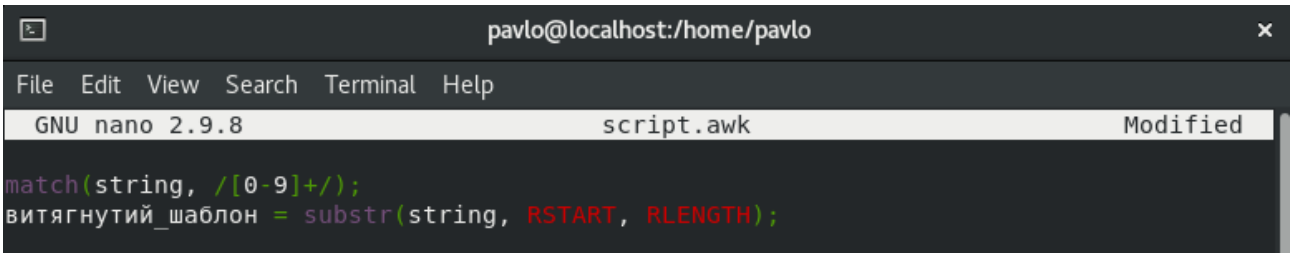
— Використання регулярних виразів в операціях пошуку та заміни: Регулярні вирази можна використовувати для пошуку і заміни тексту в рядку. Функцію gsub() можна використовувати для глобального пошуку і заміни, а функцію sub() - для пошуку і заміни окремих входжень. Наприклад, наступний код замінює всі входження "Привіт" на "Доброго дня" у рядку:



— Використання регулярних виразів в умовних операторах: Регулярні вирази можна використовувати в умовних операторах для перевірки відповідності рядка певному шаблону. Наприклад, наступний код перевіряє, чи містить рядок лише літери:



— Використання регулярних виразів для вилучення певних шаблонів з рядка: Регулярні вирази також можна використовувати для вилучення певних шаблонів з рядка. Функця match() може бути використана для пошуку першого входження регулярного виразу в рядку і повернення позиції збігу. Потім функція substr() може бути використана для вилучення відповідного шаблону. Наприклад, наступний код витягує перше входження числа в рядку:



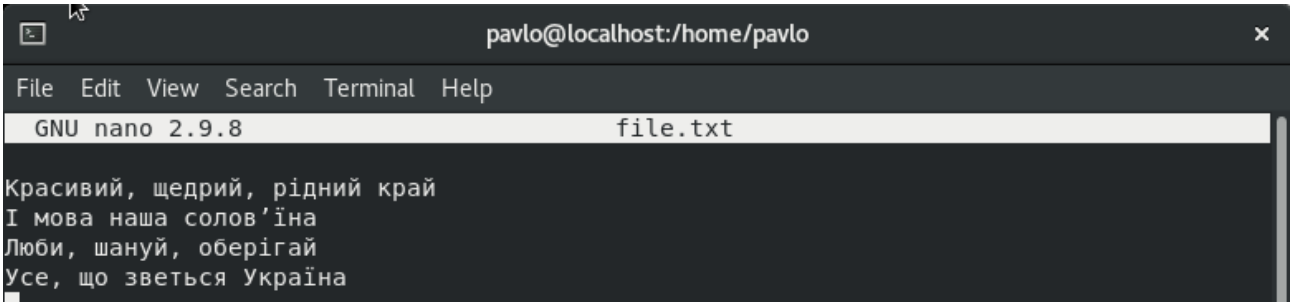
Функція match приймає два аргументи: рядок і шаблон регулярного виразу. Вона шукає у рядку перше входження шаблону і повертає позицію, з якої починається збіг. У цьому випадку шаблоном регулярного виразу є /[0-9]+/, який відповідає одній або декільком цифрам. Функція match використовується для пошуку першого входження однієї або декількох цифр у рядку. Функція

substr використовується для вилучення знайденого шаблону з рядка. Функція substr приймає три аргументи: рядок, з якого потрібно витягти шаблон, початкову позицію підрядка і довжину підрядка. В даному випадку початкова позиція - це значення, повернуте змінною RSTART, яка була встановлена функцією збігу на позицію першого символу знайденого зразка. Довжина підрядка - це значення змінної RLENGTH, яка також була встановлена функцією збігу на довжину знайденого зразка. Результатом роботи цього сценарія є те, що змінна витягнутий_шаблон міститиме знайдений шаблон з однієї або декількох цифр, витягнутих з вихідного рядка за допомогою регулярних виразів і зіставлення шаблонів.

3.2. Попередня обробка то очищення тексту

Завдяки вбудованим функціям і регулярним виразам, AWK дозволяє легко маніпулювати даними і витягувати їх з текстових файлів. Нижче, наведено приклади використання AWK для попередньої обробки та очищення даних:

— Видалення непотрібних символів і пробілів: Одним з найпоширеніших завдань попередньої обробки текстових даних є видалення непотрібних символів і пробілів. Це можна зробити за допомогою функцій sub() і gsub() в AWK. Нагадаю, функція sub() замінює перше входження шаблону у рядку, а функція gsub() замінює всі входження шаблону у рядку. Наприклад, щоб видалити всі коми з рядка, скористаємось функцією gsub(), яка замінить всі входження першого аргументу на другий. Маю ось такий текстовий файл:



Застосування сценарію:

```
[root@localhost pavlo]# awk '{gsub(",", ""); print}' file.txt
Красивий щедрий рідний край
І мова наша солов'їна
Люби шануй оберігай
Усе що зветься Україна
```

Якщо потрібно зберегти результат відразу у вигляді текстового файлу, скористаємось перенаправленням командного інтерпретатора « >» та вкажемо назву нового файлу або вже існуючого. В результаті буде створено новий файл з ім'ям newfile, який містить зміненні рядки без ком:

```
[root@localhost pavlo]# awk '{gsub(",", ""); print}' file.txt  > newfile.txt
[root@localhost pavlo]# ls
awk      Documents  file.txt  newfile.txt  Public     Templates
Desktop  Downloads  Music     Pictures     script.awk  Videos
[root@localhost pavlo]# cat newfile.txt
Красивий щедрий рідний край
І мова наша солов'їна
Люби шануй оберігай
Усе що зветься Україна
```

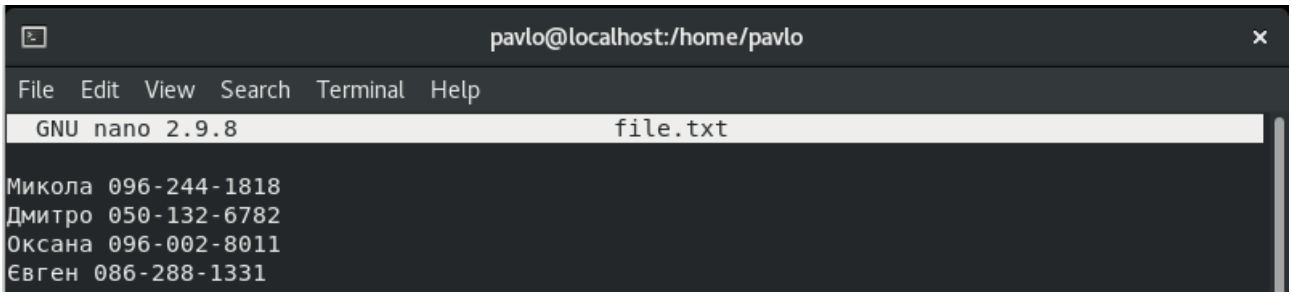
— Перетворення тексту на малі або великі літери: AWK має функції tolower() і toupper() для перетворення тексту у малі або великі літери. Це може бути корисно під час роботи з текстовими даними, не чутливими до регістру. Наприклад, щоб перетворити весь текст у файлі до нижнього регістру, скористаємось функцією tolower() наступним чином. Маю наступний текстовий файл:

Застосування сценарію:

```
[root@localhost pavlo]# awk '{print tolower($0)}' file.txt
красивий, щедрий, рідний край
і мова наша солов'їна
люби, шануй, оберігай
усе, що зветься україна
```

Цей сценарій буде виконаний для всього текстового файлу так як аргументом вказане значення $0. Функція перетворила всі символи в нижній регістр.

— Витягування даних за допомогою регулярних виразів: AWK підтримує регулярні вирази за допомогою вбудованої функції match(), за допомогою якої можна знайти шаблон у рядку. Наприклад, щоб витягти всі телефонні номери з файлу, використаємо функцію match() наступним чином. Маю ось такий текстовий файл, із корситувачами та їх мобільними номерами.



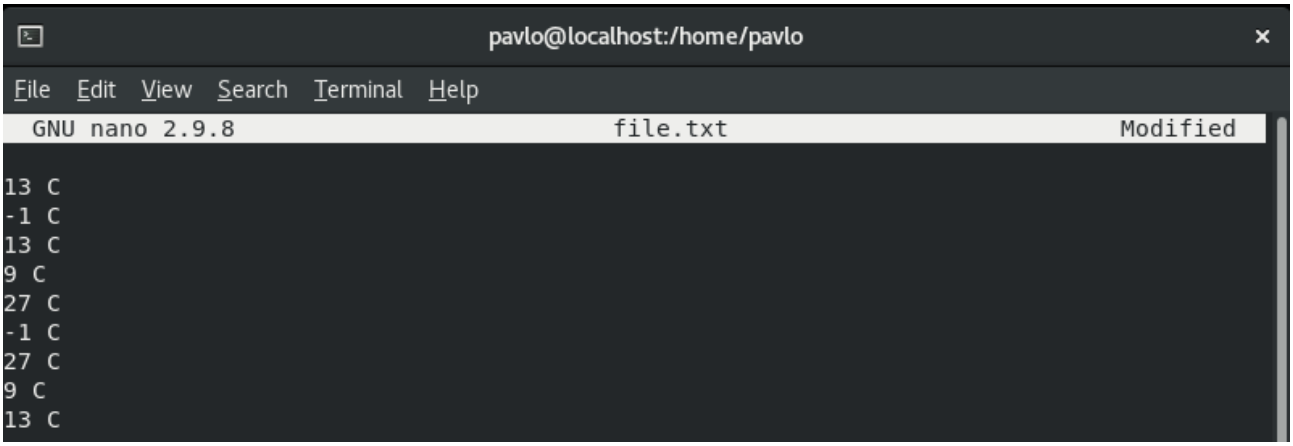
Застосування сценарію:

```
[root@localhost pavlo]# cat file.txt
Микола 096-244-1818
Дмитро 050-132-6782
Оксана 096-002-8011
Євген 086-288-1331

[root@localhost pavlo]# awk '{if (match($0, /[0-9]{3}-[0-9]{3}-[0-9]{4}/)) print substr
($0, RSTART, RLENGTH)}' file.txt
096-244-1818
050-132-6782
096-002-8011
086-288-1331
```

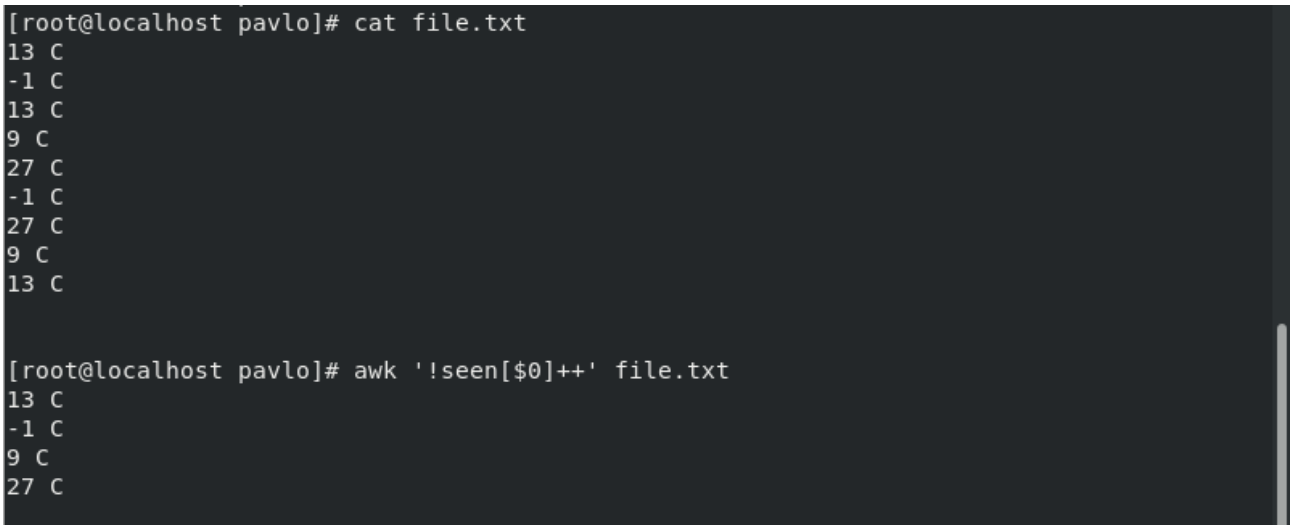
Регулярний вираз задає шаблон для телефонного номера у форматі xxx-xxx-xxxx, де "x" означає будь-яку цифру від 0 до 9. print substr($0, RSTART, RLENGTH) - це дія, яка виконується, коли інструкція if набуває

значення. Вона виводить підрядок поточного рядка, який відповідає регулярному виразу [0-9]{3}-[0-9]{3}-[0-9]{4}.

— Видалення повторюваних рядків: Видалення повторюваних рядків є поширеним завданням при роботі з текстовими даними. Далі розглянемо видалення повторюваних рядків на практиці. Маю ось такий текстовий файл, із вхідними даними по температурі:



Застосування сценарію:

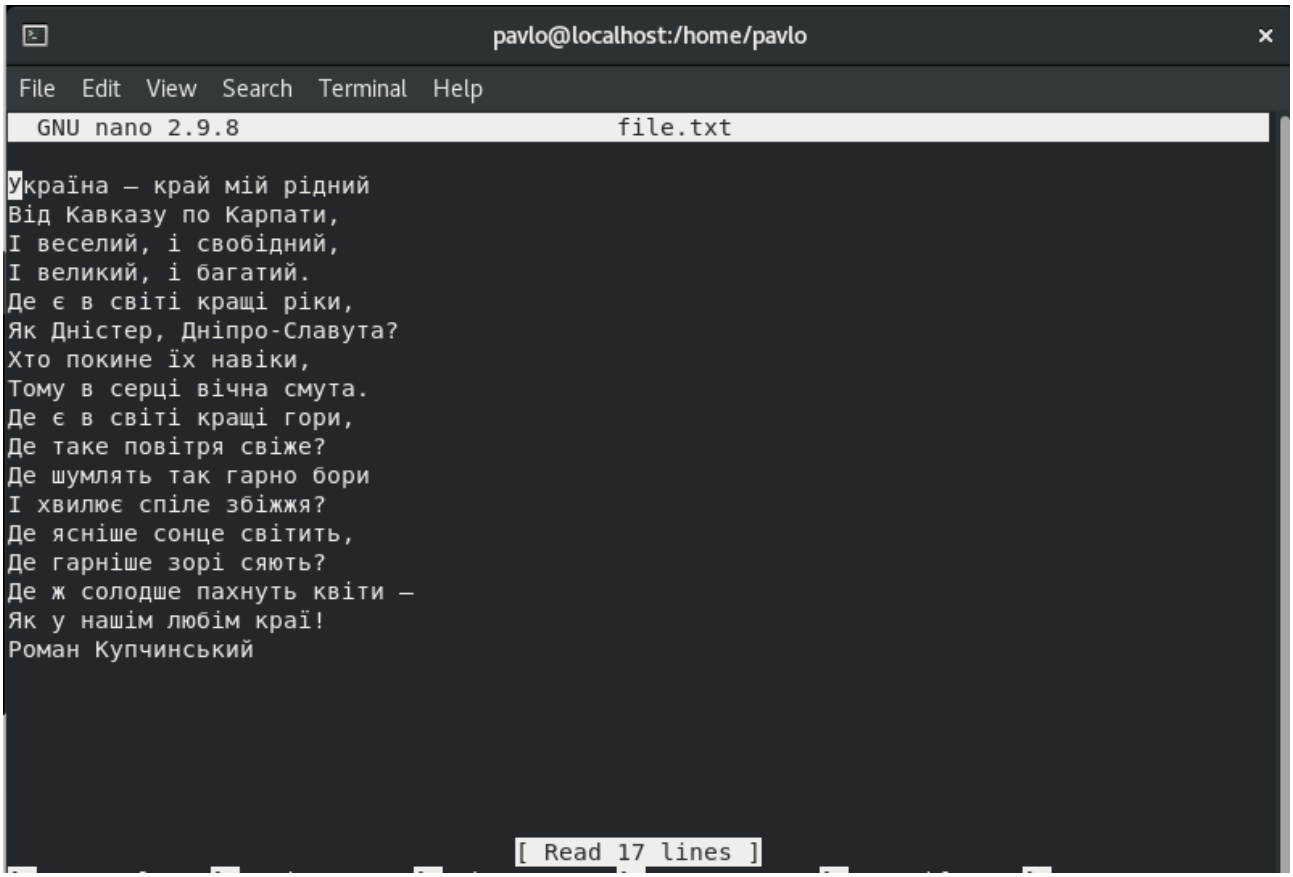


Цей сценарій використовує асоціативний масив seen для відстеження рядків, які вже були переглянуті. Вираз !seen[$0]++ обчислюється як true, якщо поточний рядок не зустрічався раніше, і як false у протилежному випадку. Коли зустрічається новий рядок, значення seen[$0] приростає, а потім оцінюється на істинність. Оператор ! інвертує результат, тобто true стає false, а false стає true.

3.3. Пошук та аналіз тексту

Розглянемо деякі з найпоширеніших потреб AWK для пошуку та аналізу тексту:

— Підрахунок кількості рядків, слів і символів: Як було сказано вище, AWK має декілька вбудованих змінних, які можна використовувати для цієї мети. Змінна NR показує номер поточного рядка, а змінна NF - кількість полів у поточному рядку. Функцію length можна використовувати для підрахунку кількості символів у рядку. Для підрахунку кількості рядків, слів і символів у файлі розглянемо наступний сценарій. Маю ось такий вхідний текстовий файл:



Сценарій:

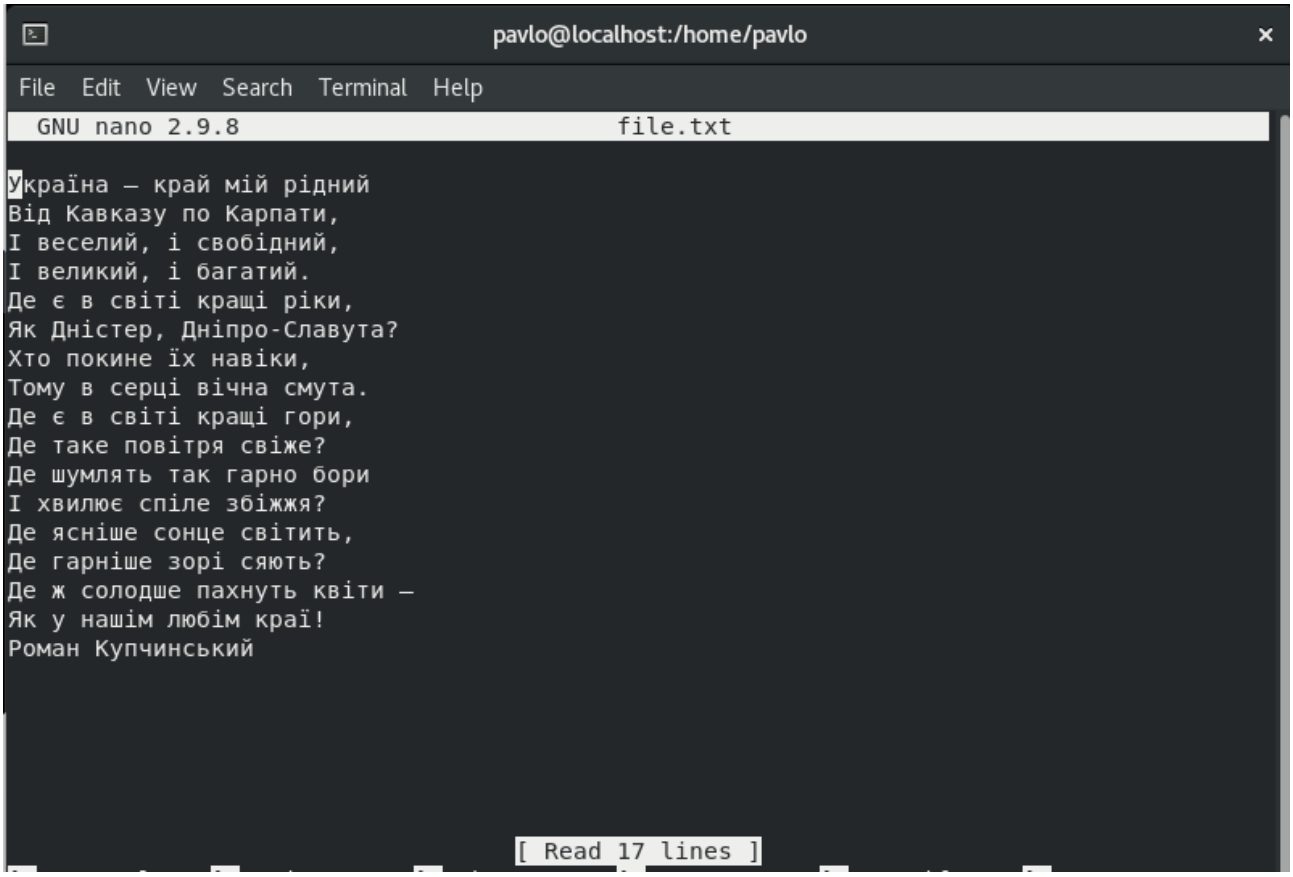
```
{
    lines++
    words += NF
    chars += length + 1
} END {
    print "Рядків: " lines ", Слів: " words ", Символів: " chars
}
```

Застосування сценарію:

```
[root@localhost pavlo]# awk -f script.awk file.txt
Рядків: 17, Слів: 75, Символів: 412
```

Сценарій підраховує змінні lines, words та char для кожного рядка у файлі і виводить кінцевий підрахунок.

— Знаходження найдовших і найкоротших рядків: AWK має декілька вбудованих змінних, які можна використовувати для цієї мети. За допомогою функції length можна визначити довжину рядка, а змінна FNR - номер рядка у поточному файлі. Щоб знайти найдовший і найкоротший рядки у файлі, розглянемо наступний сценарій. Будемо використовувати тектовий файл із минулого прикладу.



Сценарій:

```
{
if (length > max_length) {
      max_length = length
      max_line = $0
}
if (length < min_length || NR == 1) {
      min_length = length
      min_line = $0
}
} END {
      print "Найбільший рядок: " max_line ", Довжина: " max_length
      print "Найменший рядок: " min_line ", Довжина: " min_length
}
```

Застосування сценарію:

```
[root@localhost pavlo]# awk -f script.awk file.txt
Найбільший рядок: Де ж солодше пахнуть квіти –, Довжина: 28
Найменший рядок: Роман Купчинський, Довжина: 17
```

Сценарій порівнює довжину кожного рядка з поточним максимальним і мінімальним значенням і відповідно оновлює змінні. Наприкінці виводимо найдовший і найкоротший рядки разом з їхніми довжинами.

— Підрахунок входжень слова: для підрахування певного слова у файлі розглянемо наступний сценарій. Маю текстовий файл із списком імен чоловіків та приданістю до виконання військових обов’язків:

```
[root@localhost pavlo]# cat file.txt
Іван Придатний
Петро Непридатний
Олексій Придатний
Степан Придатний
Вадим Придатний
Констянтин Придатний
Олександр Непридатний
Валентин Придатний
Володимир Придатний
```

Сценарій:

```
{
    word = "Придатний"
    count += gsub(word, "")
} END {
    print "Кількість входжень слова \"" word "\": "  count
}
```

Застосування сценарію:

```
[root@localhost pavlo]# awk -f script.awk file.txt
Кількість входжень слова "Придатний": 7
```

Для кожного рядка використовуємо функція gsub(), яка знаходить і заміняє всі входження пошукового терміна, в мому випадку «Придатний», порожнім рядком. Функція повертає кількість зроблених замін, яка додається до змінної "count". Після обробки всіх рядків

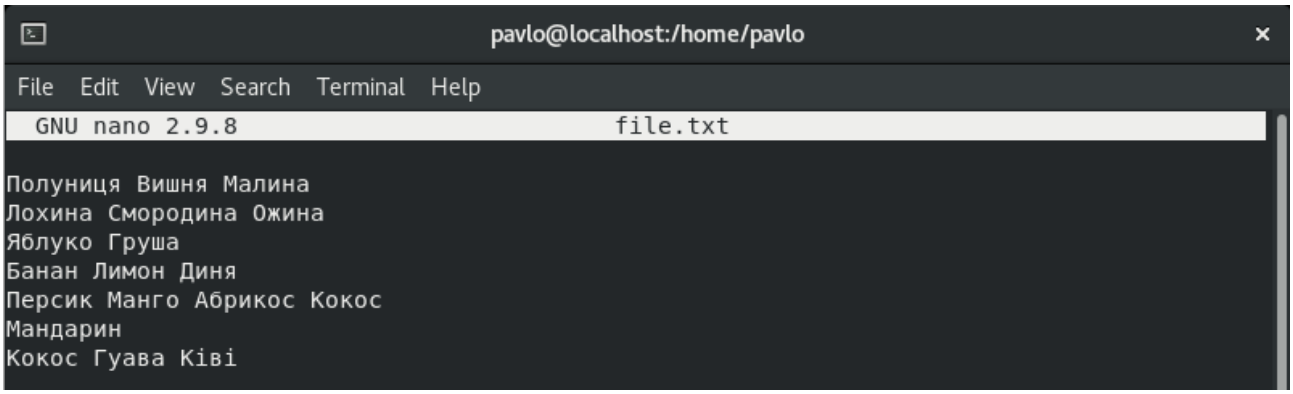
виконується блок END. Цей блок використовується для виконання дій після того, як всі вхідні дані були оброблені. Далі сценарій виводить на консоль повідомлення із зазначенням загальної кількості входжень пошукового терміну, знайденого у файлі. Повідомлення також містить і сам пошуковий термін, який отримується зі змінної "word".

РОЗДІЛ 4. ЗАСТОСУВАННЯ AWK ПРИ НАПИСАННІ СЦЕНАРІЇВ

4.1. Обробка помилок

Використовуючи AWK, можна отримати непоганий спосіб обробки помилок під час виконання програми. Нижче, я наведу декілька прикладів:

— Друк повідомлень про помилки: одним із способів обробки помилок є виведення повідомлень про помилки на консоль або у файл. Для друку повідомлень про помилки у AWK будемо використовувати функцію "print". Маю довільний текстовий файл:



Сценарій:

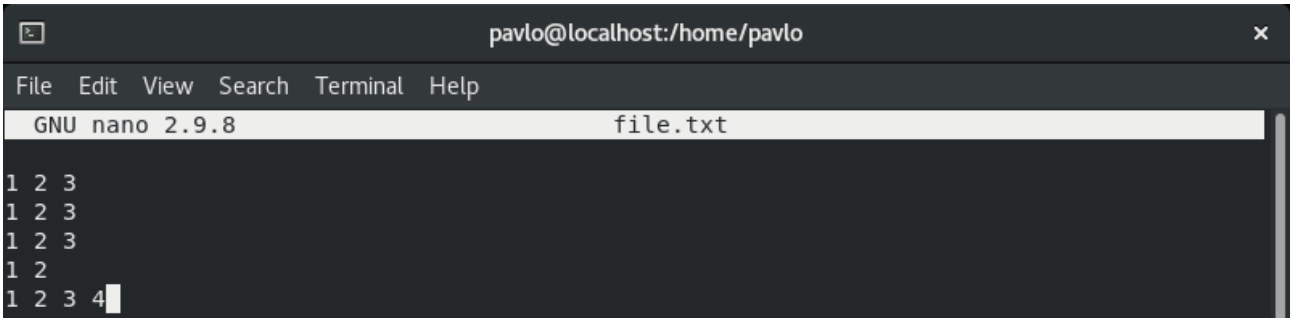
```
{
      if (NF != 3) {
      print "Помилка: недопустима кількість полів у рядку", NR,
      ".Дозволено всього 3$
      }
}
```

Застосування сценарію:

```
[root@localhost pavlo]# awk -f script.awk file.txt
Помилка: недопустима кількість полів у рядку 3 .Дозволено всього 3 поля!
Помилка: недопустима кількість полів у рядку 5 .Дозволено всього 3 поля!
Помилка: недопустима кількість полів у рядку 6 .Дозволено всього 3 поля!
```

Сценарій перевіряє, чи кожен рядок вхідного файлу має три поля. Якщо кількість полів відрізняється, він виводить на консоль повідомлення про помилку з номером рядка та дозволеною кількістю полів.

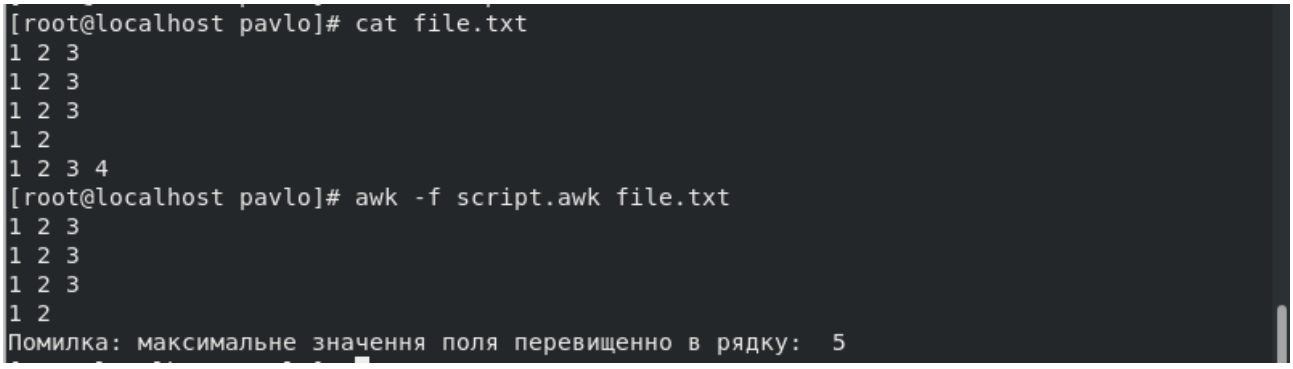
— Завершення програми: у деяких випадках при виникненні помилки може знадобитися опція завершення програми. AWK надає функцію "exit" для завершення роботи програми. Розглянемо наступний сценарій. Маю довільний текстовий файл:



Сценарій:

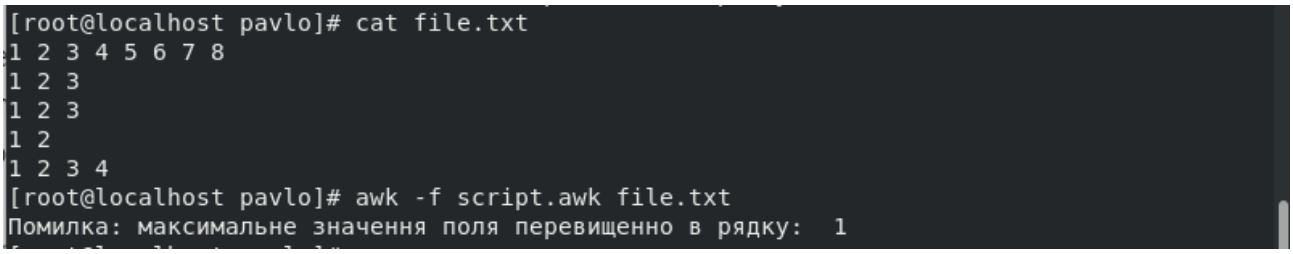
```
{
      if (NF >= 4) {
      print "Помилка: максимальне значення поля перевищенно в рядку: ",
      NR, " "
      exit 1
      }
      print $0
}
```

Застосування сценарію:



Сценарій перевіряє, чи кількість полів у вхідному файлі не перевищує 3. Якщо так, то виводиться повідомлення про помилку і програма завершує роботу. Якщо кількість рядків менша за 10, він виводить кожен рядок на

консоль. В мому прикладі помилковий рядок є останнім, переглянемо, що буде якщо помилковий рядок буде першим:



4.2. Обробка мови та класифікація текстів

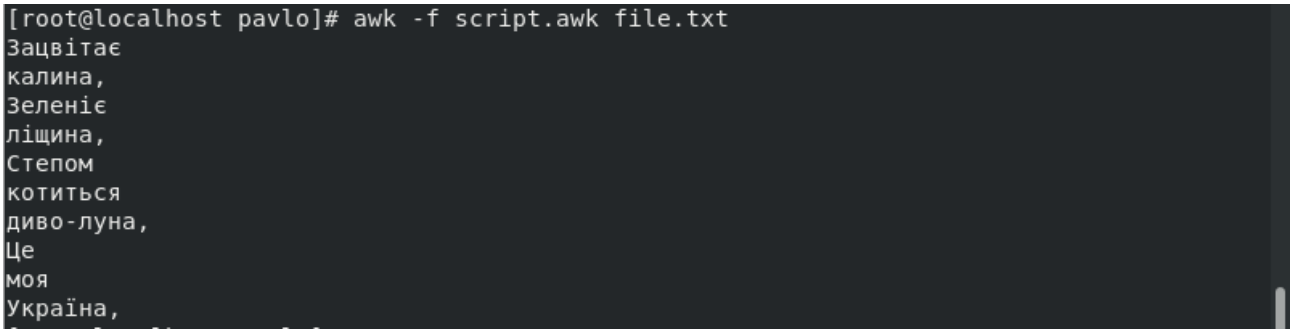
Обробка мови включає в себе такі завдання, як токенізація, розбиття на частини та видалення стоп-слів. Токенізація передбачає розбиття тексту на окремі слова або токени. Стеммінг - це процес скорочення слів до їхньої базової форми або основи. Видалення стоп-слів передбачає видалення загальновживаних слів, які не несуть особливого сенсу в тексті. AWK можна використовувати для всіх цих завдань.

— Токенізація: для токенізації текстового файлу, розглянемо наступний сценарій. Маю довільний текстовий файл який містить довгий рядок:

Сценарій:

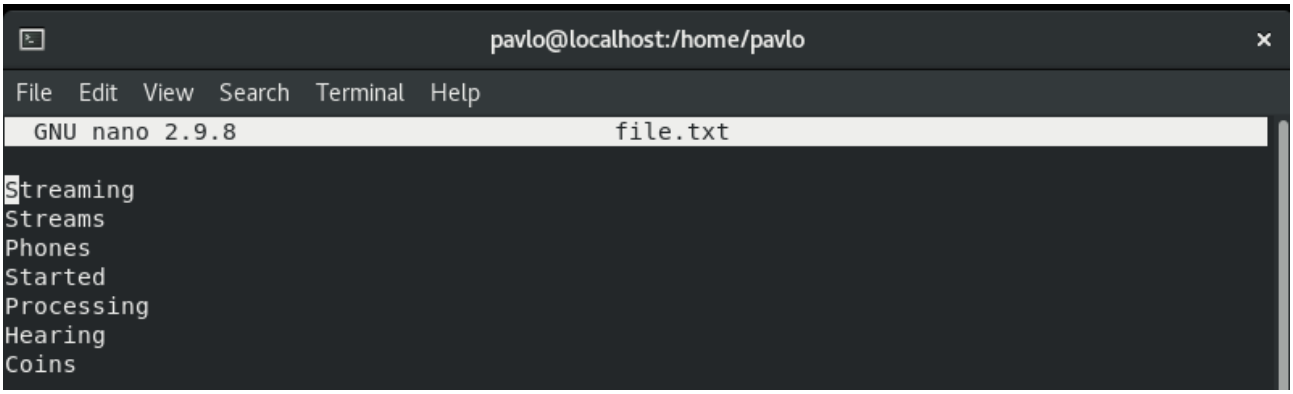
```
{
      for (i=1; i<=NF; i++)
      print $i
}
```

Застосування сценарію:



Сценарій розбиває кожне поле у файлі на окремі слова і виводить їх в окремих рядках.

— Стеммінг: для виконання стеммінгу в AWK, розглянемо наступний сценарій. Маю довільний текстовий файл:



Сценарій:

```
# Правила стеммінгу

function stem(word) {

    if (word ~ /sses$/) {

        return substr(word, 1, length(word) - 4) "ss";

    } else if (word ~ /ies$/) {

        return substr(word, 1, length(word) - 3) "i";

    } else if (word ~ /ss$/) {

        return word;

    } else if (word ~ /s$/) {

        return substr(word, 1, length(word) - 1);
```

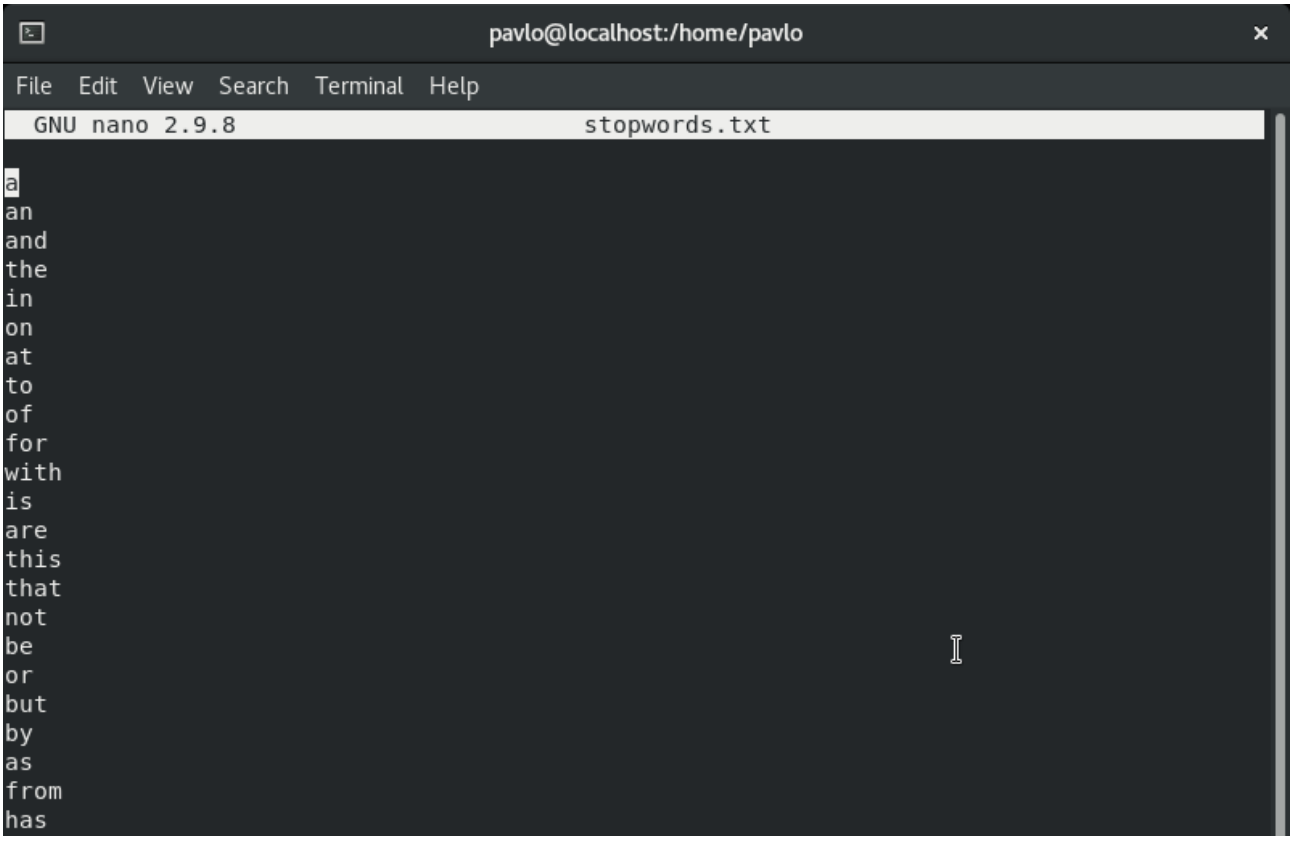
```
    } else if (word ~ /eed$/) {
        if (match(substr(word, 1, length(word) - 3), /[aeiouy]/)) {
            return substr(word, 1, length(word) - 3) "ee";
        } else {
            return word;
        }
    } else if (word ~ /ed$/) {
        if (match(substr(word, 1, length(word) - 2), /[aeiouy]/)) {
            return substr(word, 1, length(word) - 2);
        } else {
            return word;
        }
    } else if (word ~ /ing$/) {
        if (match(substr(word, 1, length(word) - 3), /[aeiouy]/)) {
            return substr(word, 1, length(word) - 3);
        } else {
            return word;
        }
    } else {
        return word;
    }
}

# Частина зчитування вхідного файлу і вивід результату
{
    for (i=1; i<=NF; i++) {
        printf("%s ", stem($i));
    }
```

```
    printf("\n");
}
```

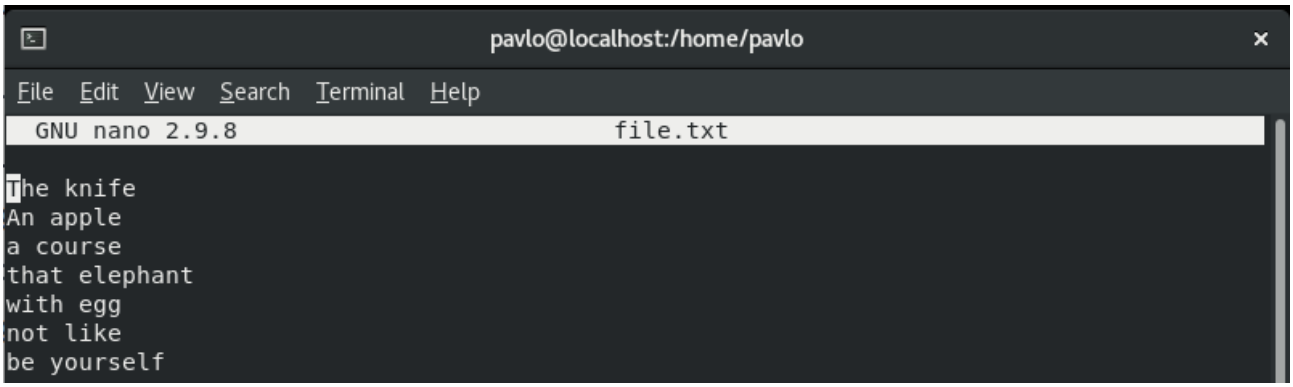
Застосування сценарію:

Сценарій визначає правила розбиття у вигляді функції із іменем steam(), яка видаляє суфікси «ing», «s» та «ed». Функція приймає на вхід слово і застосовує відповідне правило розбиття на частини, щоб створити розбиту версію слова. Потім сценарій читає вхідний файл і циклічно переглядає кожне слово у файлі, застосовуючи функцію stem() до кожного слова і виводячи розбиту на частини версію у стандартний вивід.

— Видалення стоп-слів: розглянемо сценарій, що вилучить із слів у файлі стоп-слова, які я розмістив в окремому файлі. Маю ось такий текстовий файл із стоп-словами:



Текстовий файл для обробки:



Сценарій:

```
# зчитування списку стоп-слів і створення масиву

BEGIN {

    while (getline < «stopwords.txt») {

        stopwords[$1] = 1

    }
```

```
    close(«stopwords.txt»)
}

# обробка текстового файлу та видалення стоп-слів
{
    for (i=1; i<=NF; i++) {
        if (!(tolower($i) in stopwords)) {
            printf(«%s «, $i)
        }
    }
    printf(«\n»)
}
```

Застосування сценарію:

```
[root@localhost pavlo]# awk -f script.awk file.txt
knife
apple
course
elephant
egg
like
yourself
[root@localhost pavlo]#
```

Було створено файл із стоп-словами під назвою «stopwords.txt». Далі в сценарії використали цей файл для списку слів які потрібно вилучити і було створено масив для зберігання стоп-слів. Далі сценарій проходить циклом по кожному слову у файлі і перевіряє, чи є воно стоп-словом. Якщо це стоп-слово, вилучаємо його, якщо ні- друкуємо.

— Класифікація тексту передбачає розподіл тексту на заздалегідь визначені категорії. AWK можна використовувати для простих завдань

класифікації тексту. Наприклад, за допомогою AWK можливо класифікувати імейли на спам і не спам. Розглянемо наступний сценарій. Маю ось такий текстовий файл, в якому перший рядок це повідомлення зі спамом, а другий рядок не містить спаму:

```
[root@localhost pavlo]# cat file.txt
We are excited to offer you an exclusive discount on our latest products. Buy now and g
et up to 50% off on all items. Don't miss this opportunity to save big!
I would like to invite you to a meeting next week to discuss the progress of our projec
t. The meeting will take place on Wednesday at 10am in the conference room.
```

Сценарій:

```
{

if ($0 ~ /buy now/ || $0 ~ /discount/) {
print "SPAM"

} else {

      print "NON-SPAM"

    }

}
```

Застосування сценарію:

```
[root@localhost pavlo]# awk -f script.awk file.txt
SPAM
NON-SPAM
[root@localhost pavlo]#
```

Сценарій класифікує листи, що містять слова "buy now" або "discount", як спам і виводить повідомлення "SPAM". Інші листи будуть класифіковані як не спам і буде виведено повідомлення "NON-SPAM".

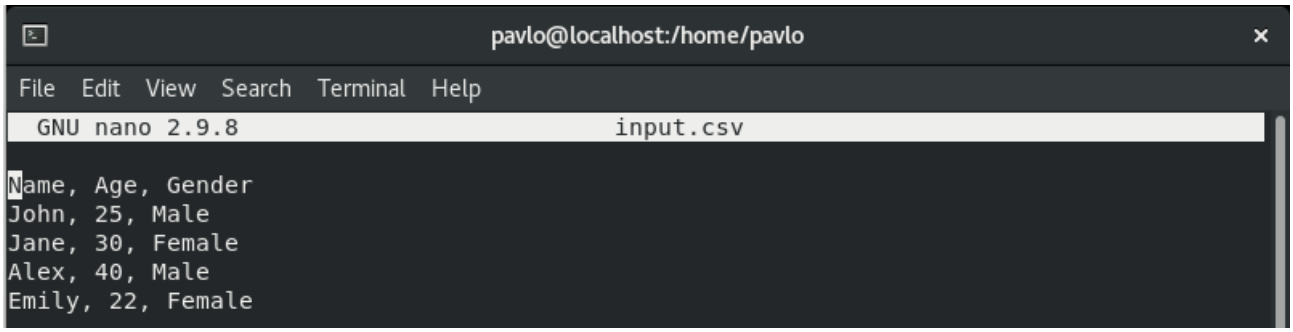
В темі обробки і класифікації мови утиліта AWK не така потужна як спеціальні бібліотеки, але може бути корисним інструментом для швидкого і простого опрацювання тексту.

4.3. Автоматизація сценаріїв

Автоматизація сценаріїв - це процес автоматизації повторюваних завдань за допомогою сценаріїв або програм. Опишемо кілька способів використання AWK для автоматизації:

— Автоматизація обробки файлів: AWK можна використати для автоматизації обробки великих текстових файлів. Як було описано вище із розділу «поглиблені методи AWK-програмування», можна скористатись пошуком за зразками та очищенням тексту, для вилучення певних полів з файлу або для видалення небажаних символів.

— Конвертація файлу: AWK можна використовувати для перетворення даних з одного формату в інший. Наприклад, скористаємось AWK для перетворення CSV-файлу в JSON-файл. У файлі CSV кожен рядок являє собою запис, а кожне поле в рядку відокремлюється комою. JSON використовується для представлення об'єктів даних, що складаються з пар атрибут-значення. У файлі JSON дані представлені парами ключ-значення, де ключі завжди є рядками, а значення можуть бути різних типів, таких як рядок, число, об'єкт або масив. Розглянемо наступний сценарій, маю ось такий csv-файл:

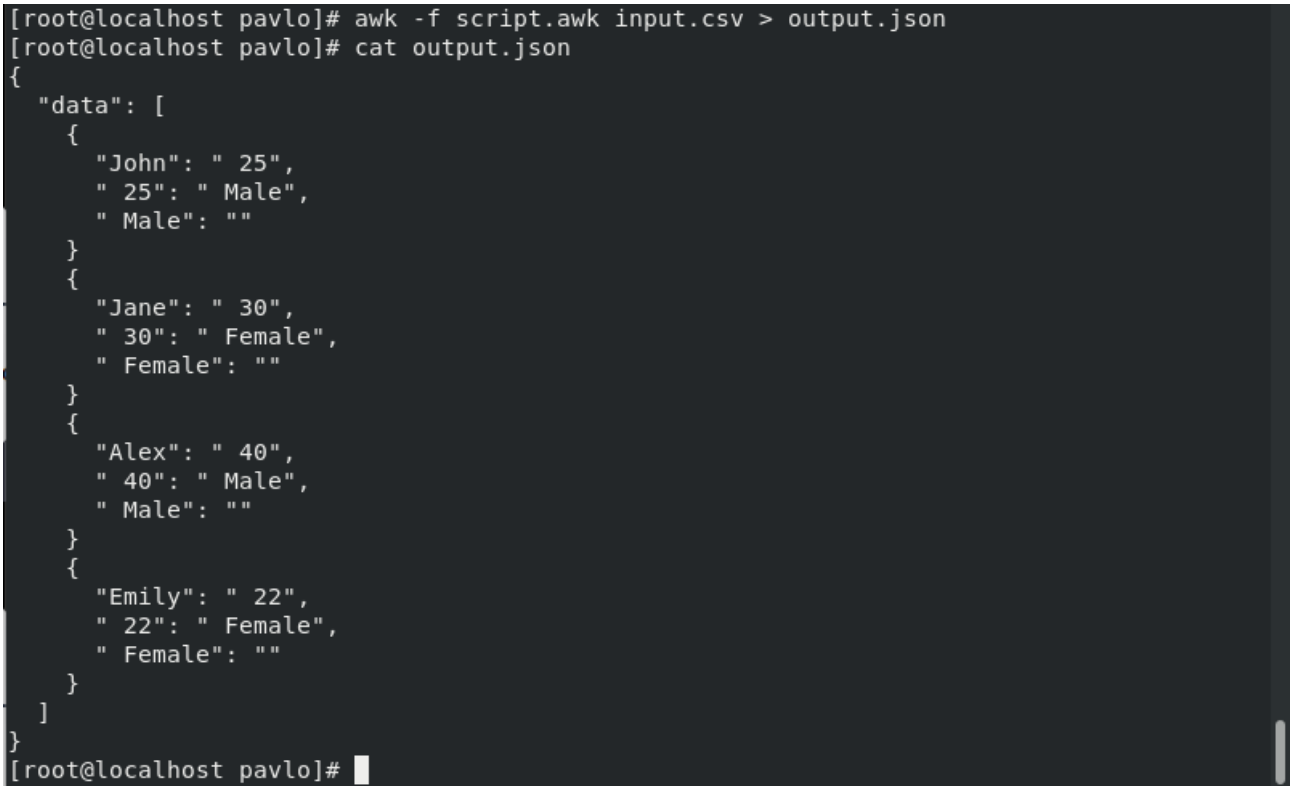


Сценарій:

```
BEGIN {
    FS = ","
    print "{"
```

```
    print "  \"data\": ["
}

NR > 1 {
    print "    {"
    for (i = 1; i <= NF; i++) {
        printf "      \"%s\": \"%s\"", $i, $(i+1)
        if (i != NF) {
            printf ","
        }
        printf "\n"
    }
    print "    }"
    if (NR != FNR) {
        printf ","
    }
}

END {
    print "  ]"
    print "}"
}
```

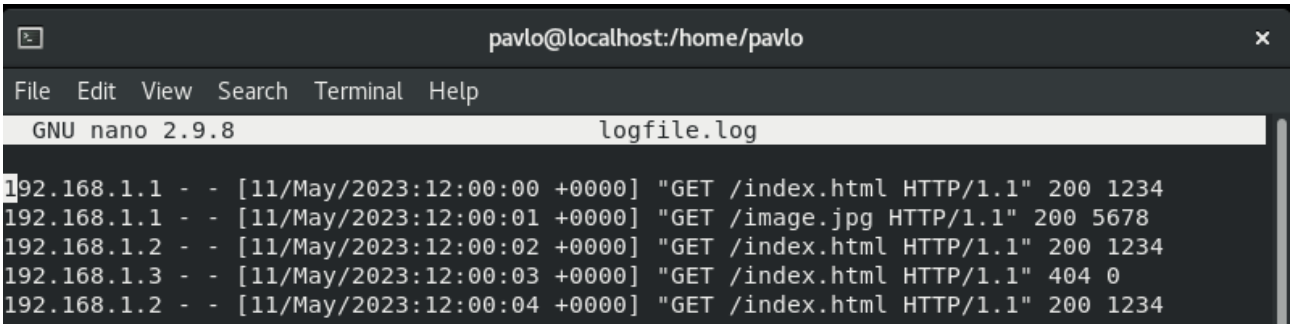
Застосування сценарію та зберігання результат в JSON файл:



4.4. Системне адміністрування

Утиліта AWK корисна не лише для маніпуляцій з даними та обробки тексту, але й для завдань системного адміністрування. Далі наведено приклади, де б AWK було корисним для системного адміністрування:

— Обробка log-файлів: При необхідності обробляти файли журналів для виявлення системних проблем, вузьких місць у роботі та інших подій, що становлять інтерес, AWK можна використовувати для цього аналізу log-файлів і вилучення відповідної інформації, такої як мітки часу, повідомлення про помилки, IP-адреси тощо. За допомогою AWK можна писати сценарії, які можуть автоматизувати обробку log-файлів, заощаджуючи час і зусилля. Розглянемо наступний сценарій, який обробляє log-файл і витягує унікальні IP-адреси. Маю ось такий експериментальний log-файл:



Сценарій:

```
# встановлюю роздільник полів як пробіл або табуляція
BEGIN {
    FS = "[ \t]+"
}

# якщо перше поле це IP адреса, врахувати її у count
{
    if ($1 ~ /^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$/) {
        count[$1]++
    }
}

# вкінці обробки виводжу всі знайдені унікальні IP адреси
END {
    for (ip in count) {
        print ip
    }
}
```

— Керування користувачами та дозволами: AWK можна використовувати для автоматизації управління обліковими записами користувачів і дозволами. Наприклад, можна створювати нові облікові записи

користувачів, змінювати існуючі та видаляти облікові записи. Також, можна використовувати для встановлення дозволів на файли і каталоги, гарантуючи, що лише авторизовані користувачі матимуть доступ до конфіденційних даних. AWK також можна використовувати для автоматизації створення і видалення груп, а також призначення користувачів до груп.

— Моніторинг системних ресурсів: AWK можна використовувати для моніторингу системних ресурсів, таких як використання процесора, пам'яті, диска і пропускної здатності мережі. За допомогою AWK можна писати сценарії, які можуть сповіщати про надмірне використання ресурсів, дозволяючи вжити заходів для виправлення ситуації до того, як це вплине на продуктивність системи. Розглянемо сценарій, який можна використовувати для моніторингу використання ресурсів і сповіщення адміністраторів через емейл, якщо використання перевищує певний поріг. Приклад сценарію:

```
BEGIN {
    # Встановлюю поріг використання ресурсів
    threshold = 90

    # Встановлення емейл адреси для сповіщення
    admin_email = "pavseredyuk1313@gmail.com"

    # Встановлення теми для мейлу
    alert_subject = "High resource usage alert"

    # Встановлення максимальної кількості відправки повідомлень на день
    max_alerts = 3

    # Ініціалізую лічильник кількості відправлених алертів
    alerts_sent = 0
}

{
    # Обробка рядка для вилучення відповідної інформації
    pid = $2
    cpu_usage = $3
    mem_usage = $4
```

```
    # Перевіряю, чи використання процесора або пам'яті не перевищує порогового значення
    if (cpu_usage > threshold || mem_usage > threshold) {
        # Генерування попереджувального повідомлення
        alert_message = sprintf("Process %s is using %s%% CPU and %s%% memory", pid, c$

        # Відправка повідомлення
        if (alerts_sent < max_alerts) {
            command = sprintf("echo '%s' | mail -s '%s' %s", alert_message, alert_subj$
            system(command)
            alerts_sent++
        }
    }
}
```

Цей сценарій зчитує вхідні дані з лог-файлу і перевіряє використання процесора і пам'яті кожним процесом. Якщо використання перевищує заданий поріг, він генерує попередження і надсилає електронного листа адміністратору. Кількість повідомлень, що надсилаються на день, обмежена, щоб уникнути переповнення поштової скриньки адміністратора.

— Автоматизація резервного копіювання та відновлення: AWK можна використовувати для автоматизації резервного копіювання та відновлення системних файлів і даних. Наприклад, AWK можна використовувати для створення сценаріїв резервного копіювання, які копіюють файли на віддалений сервер або пристрій для резервного копіювання. Утиліта також може бути використаний для автоматизації процесу відновлення, дозволяючи адміністраторам швидко відновлювати системні файли і дані в разі збою або катастрофи.
— Керування службами: AWK можна використовувати для автоматизації управління системними службами. Наприклад, AWK можна використовувати для запуску, зупинки і перезапуску служб, а також для моніторингу їх стану. За допомогою AWK можна писати сценарії, які автоматично перезапускатимуть служби, якщо вони вийдуть з ладу або стануть неактивними, гарантуючи, що критичні служби завжди будуть доступні.

Утиліта AWK є дійсно потужним інструментом в системному адмініструванні. Її можна використовувати для автоматизації широкого спектру завдань системного адміністрування, включаючи обробку файлів журналів, керування користувачами та дозволами, моніторинг системних ресурсів, а також керування службами.

ВИСНОВКИ

В ході виконання цієї роботи мною було вивчено та досліджено утиліту AWK в UNIX-системах. Як середовище віртуалізації я використовував ПЗ «Oracle VM VirtualBox», для тестування та створення сценаріїв я створив віртуальну машину на основі ОС Centos. Оскільки основою моєї роботи було саме поглиблення курсу, мною було створено два посібника, для студентів та для викладача. Загалом, мій посібник студента містить 30 завдань різного рівня складності для сценаріїв AWK, з них 5 початкового рівня, 5 легкого рівня, 10 середнього рівня та 10 важкого рівня. Посібник викладача, відповідно, містить 30 сценаріїв-розв'язків для цих завдань.

Потенційне використання моїх сценаріїв полягає в використанні їх для поглиблення навичок студентів із утилітою AWK, та для перевірки викладачем результатів їх робіт

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Aho A. V., Kernighan B. W., & Weinberger P. J. (1988). The AWK Programming Language. Addison-Wesley Professional.

2. Kernighan B. W., & Pike R. (1999). The Practice of Programming. Addison-Wesley Professional.

3. Mendel C., Advanced Bash-Scripting Guide 2.5, переклад Кисилев А., [Електронний ресурс] — Режим доступу до ресурсу:

http://asmodeus.com.ua/library/programing/shell/advanced_bash_scripting.html

4. Посібник з AWK 5.2, Effective AWK Programming: A User's Guide for GNU Awk — Режим доступу до ресурсу:

https://www.gnu.org/software/gawk/manual/gawk.html

5. Гаррелс М., Посібник з Bash для початківців 1.7, перелад Масик М. Розділ 6, 2005 — режим доступу до ресурсу:

http://docs.linux.org.ua/LDP/Bash_beginners_guide/the_gnu_awk_programming_language/

6. Довідкова сторінка Linux для AWK — Режим доступу до ресурсу:

https://linux.die.net/man/1/awk

ДОДАТКИ

Додаток 1. Приклади розділів задачників «Посібник студента» та «Посібник викладача» для написання сценаріїв за допомогою утиліти AWK

ВАЖКИЙ РІВЕНЬ

21. Реалізувати сценарій, що підрахує загальну кількість слів в файлі та виведе найчастіше вживане

Підказка: спочатку, потрібно реалізувати роздільник полів «RS» як регулярний вираз, який буде відповідати будь-якій послідовності символів, що не є літерами, цифрами, підкресленнями, апострофами або дефісами. Далі потрібно буде пройтись по всьому вхідному файлі і записати всі слова у масив з поточним словом та його індексом, та збільшувати індекс кожен раз, коли зустрічаєм подібне слово. Також, потрібно ініціалізувати змінну, для загального запису кількості слів. По закінченню сценарію, потрібно пройтись по масиву, підрахувавши при цьому загальну кількість слів та вивести слово із найбільшим індексом.

Рис. 1. Приклад посібника студента для написання сценаріїв за допомогою утиліти AWK

21. Реалізувати сценарій, що підрахує загальну кількість слів в файлі та виведе найчастіше вживане

Рішення:

Мій сценарій читає текстовий файл і розбиває його на слова за допомогою регулярного виразу. Потім він підраховує частоту кожного слова і обчислює загальну кількість слів. Нарешті, він виводить загальну кількість слів і найчастіше слово, а також кількість разів, коли воно з'являється в текстовому файлі

Сценарій:

#!/usr/bin/awk -f

BEGIN {

RS="[^[:alnum:]_.'-]+"

IGNORECASE=1

}

74

Рис. 2. Приклад посібника викладача для написання сценаріїв за допомогою утиліти AWK

```
{
   words[$0]++
   total++
}

END {
   print "Всього слів:", total

   max = 0
   for (w in words) {
      if (words[w] > max) {
         max = words[w]
         max_word = w
      }
   }
   print "Найчастіше вживане слово:", max_word, "(", max, "разів зустрілось )"
}
```

Створюю довільний текстовий файл:

```
[root@localhost 9]# cat text
complaint_id, date_received, product, issue, sub_issue, consumer_complaint_narrative, c
ompany, state, zip_code, tags
1, 2022-01-01, Credit reporting, Incorrect information on credit report, Account status
, "I found several errors on my credit report which negatively impacted my credit score
", Equifax, NY, 10001, Credit reporting, Debt collection
2, 2022-01-03, Mortgage, Loan modification,collection,foreclosure, "My mortgage company
 has been unresponsive and has not provided clear information about my loan modificatio
n options", Wells Fargo, CA, 90001, Mortgage, Loan servicing
3, 2022-01-04, Credit card, Billing disputes, APR or interest rate, "I was charged an i
ncorrect interest rate on my credit card and have been unable to resolve the issue with
 the company", Capital One, TX, 75001, Credit card, Debt collection
4, 2022-01-06, Student loan, Dealing with my lender or servicer, Problem with a credit
reporting company's investigation into an existing problem, "My student loan servicer h
as been unresponsive and has not provided me with clear information about my repayment
options", Navient, OH, 43001, Student loan, Customer service
5, 2022-01-08, Debt collection, False statements or representation, Attempted to collec
t wrong amount, "A debt collector has been trying to collect a debt that is not mine an
d has made false statements to try to get me to pay", Allied Interstate, IL, 60001, Deb
t collection, Harassment
[root@localhost 9]#
```

75

Рис. 3. Приклад посібника викладача для написання сценаріїв за допомогою утиліти AWK

Застосування сценарію:

```
[root@localhost 9]# awk -f script.awk text
Всього слів: 206
Найчастіше вживане слово: has ( 6 разів зустрілось )
[root@localhost 9]#
```

76

Рис. 4. Приклад посібника викладача для написання сценаріїв за допомогою утиліти AWK