Дисципліна «Операційні системи»

Лабораторна робота №4

Тема: «Складна обробка текстових даних засобами оболонки Unix-подібних ОС інтерпретатора команд ОС»

Викладач: Олександр А. Блажко,

доцент кафедри ІС Одеської політехніки, blazhko@ieee.org

Мета роботи: придбання навичок складної обробки текстових даних роботи засобами оболонки *Unix*-подібних *ОС* інтерфейсу командного рядку на прикладі утиліт використання регулярних виразів.

1 Теоретичні відомості

1.1 Використання регулярних виразів шаблонів складного пошуку текстових даних в утиліті *GREP*

1.1.1 Обробка текстових даних за шаблоном

В основі задоволення інформаційної потреби сучасного споживача є швидкий пошук відповідей на запитання «Як швидко та дешево отримати матеріальну потребу?»

В процесі такого пошуку користувач має швидко та дешево обробити (переосмислити) надмірно великий потік даних, який він не може одночасно охопити наявними *INPUT*-засобами, щоб ці дані стали інформацією.

Якщо розглядати дані у формі тексту як набору символів (літер, цифр, спеціальних знаків), тоді перед споживачем для досягнення своєї інформаційної мети як користувачем якоїсь інформаційної системи постають такі завдання:

– пошук (отримання) у множині рядків символів тексту множини підрядків (частини) символів, які задовольняють якомусь шаблону (зразку) символів тексту;

– видалення з множини рядків символів тексту множини підрядка (частини), які задовольняє якомусь шаблону (зразку) символів тексту;

– пошук і заміна у множині рядків символів тексту множини підрядка (частини) символів, які задовольняє якомусь шаблону (зразку) символів тексту.

В більшість сучасних мов програмування включено спеціальну програмну бібліотеку, яка вирішує завдання обробки рядків символів тексту за шаблоном (зразком) символів тексту, з використанням мови регулярних виразів (*Regular Expressions*).

Одним із перших прикладів запроваджень програмної бібліотеки підтримки мови регулярних виразів став текстовий редактор *ed* як перший стандартний текстовий редактор для оболонки комадного рядку ОС *Unix*, створений *Ken Thompson*, співробітником компанії *Bell Labs*, у 1971 році.

1.1.2 Утиліта *GREP* як перший програмний інструмент з автоматизованого пошуку тексту за шаблоном регулярних виразів

Багато команд оболонки *Bash* та зовнішніх програм командного рядку (*Utility program, utility*) працюють з текстовими даними зі стандартних потоків *stdin, stdout, stderr*. Тому пошукові запити на мові регулярних виразів швидко стали популярними та в подальшому на основі програмних бібліотек було створено окрему утиліту *GREP – Search **G**lobally for lines matching the **R**egular **E**xpression, and **P**rint them*.

Утиліта *grep* має наступну синтаксичну конструкцію:

grep 'регулярний_вираз_пошуку' [назви_файлів] [опції]

Рядок з шаблоном пошуку може бути в одинарніх або подвійних лапках.

Популярні опції команди:

– *n* – вивести номер рядка;

– *c* – вивести кількість унікальних слів (підрядків, які виділяються від іншої частини тексту знаками пунктуації);

– *v* – вивести рядки, які не відповідають шаблону;

– *E* – використання *Extended Regular Expressions (ERE)* або нової версії синтаксису.

RegExp-мова при описі шаблону тексту використовує так звані мета-символи, «символи-джокери» (*wildcard characters*) або «байдужі» символи, які можуть замінити («покрити») будь-яку послідовність символів текстового рядка.

Нижче буде наведено основні приклади мета-символів та приклади їх використання.

У попередній лабораторній роботі було використано конвеєр команд для виведення на екран переліку назва каталогів зі змінної *$PATH*:

```
echo $PATH | tr ':/' '\n' | sort -u
```

Якщо необхідно вивести лише назви каталогів, які містять символ а, можна до ланцюжку команд додати команду *grep*:

```
echo $PATH | tr ':/' '\n' | sort -u | grep 'a'
```

Але як вказати команді *grep*, що символ *a* повинен знаходитися нке в будь-якомі місці текстового рядку, а лише на початку або наприкінці рядку?

Для цього можна викристати мета-символи прив'язки шаблону або якоря шаблону (*anchor*) до розташування у рядку:

– символ `^` – прив'язати шаблон до початку рядку;

– символ `$` – прив'язати шаблон до кінця рядку$

Якщо необхідно перевірити наявність порожніх рядків, тоді використовується шаблон, в якому із правої частини символу ^ та з лівої частини символу *$* не буде будь-яких символів, тобто – *^$*

Приклад отримання рядків, які починаються з букви *a*:

```
echo $PATH | tr ':/' '\n' | sort -u | grep '^a'
```

Але як вказати утиліті, що на початку або наприкінці рядку може знаходитися один із декількох символів?

Для цього є мета-символи альтернативи / угруповання і символьні класи:

– символ | розділяє альтернативні варіанти;

– символи круглих дужок (...) групують елементи в один логічний блок;

– символи квадратних дужок [...] визначають клас символів через перерахування допустимих символів або з використанням символу діапазону -

– символи [^ ...] визначають інвертований клас символів через перерахування неприпустимих символів;

Для зручного використання вказаних мета-символів в утиліті *grep* рекомендується вказати опцію *-E*, інакше перед цими мета-символами необхідно вказувати символ слеш \

Для проведення універсальних експериментів з регулярними виразами, які не потребують використовувати будь-які мови програмування та утиліти, можна скористатися *Online*-інструментарієм, наприклад, за адресою https://regex101.com/

На рисунку 1 наведено команди перевірки вмісту рядків за шаблоном регулярних виразів утилітою *GREP*. У вказаному прикладі ланцюжок конвеєру команд *echo* та *tr* дозволяє швидко створити тестові (експериментальні) дані для їх подальшої оброки регулярними виразами різних утиліт.

```
$ echo "cat;dog;mice" | tr ';' '\n' | grep -E '^c|e$'
cat
mice
```

(*a*) `echo "cat;dog;mice" | tr ';' '\n' | grep -E '^c|e$'`
– перевірити наявність рядків, які починаються з літери *c* або закінчуються на літеру *e*

```
$ echo "cat;dog;mice" | tr ';' '\n' | grep -E '^(c|d)'
cat
dog
```

(*b*) `echo "cat;dog;mice" | tr ';' '\n' | grep -E '^(c|d)'`
– перевірити наявність рядку, який починається з літери *c* або *d*

```
$ echo "cat;dog;mice" | tr ';' '\n' | grep -E '(g|e)$'
dog
mice
```

(*c*) `echo "cat;dog;mice" | tr ';' '\n' | grep -E '(g|e)$'`
– перевірити наявність рядку, який закінчується на літеру *g* або *e*

Рис. 1 – Приклади команд перевірки вмісту рядків за шаблоном регулярних виразів *GREP*

```
$ echo "cat;dog;mice" | tr ';' '\n' | grep -E '^c[a-z]t$|^d[a-z]g$'
cat
dog
```

(*d*) `echo "cat;dog;mice" | tr ';' '\n' | grep -E '^c[a-z]t$|^d[a-z]g$'`

– перевірити наявність рядків, які починаються на літеру *c* та закінчуються на літеру *t* або починаються на літеру *d* та закінчуються на літеру *g*

```
$ echo "cat;dog;mice" | tr ';' '\n' | grep -E '^[^a-e]'
mice
```

(e) `echo "cat;dog;mice" | tr ';' '\n' | grep -E '^[^a-e]'`

– перевірити наявність рядків, які не починаються на літери від *a* до *e*

Рис. 1 – Приклади команд перевірки вмісту рядків за шаблоном регулярних виразів *GREP*

(продовження)

Але як вказати утиліті, що напочатку або наприкінці рядку може знаходитися декілька повторів символу або групи із символів?

Для цього можна використати мета-символи повтору або так звані «квантифікатори»:

– символ? позначає повторення будь-кого символу 0 або 1 раз;

– символ * позначає повторення будь-якого символу 0 або більше разів;

– символ + позначає повторення будь-якого символу 1 або більше разів;

– символи `{m,}` позначають повторення попереднього елемента *m* або більше разів;

– символи `{m}` позначають повторення попереднього елемента рівно m разів;

– символи `{m, n}` позначають повторення попереднього елемента від *m* до *n* раз.

Нижче наведено приклад отримання рядків, які починаються з літери *a*, але з повтором два рази:

`echo "a11;aa2;aaa3;bbb;ccc" | tr ';' '\n' | grep -E '^a{2}'`

Додаткові мета-символи, які спрощують процес опису шаблону символів:

d - будь-яка цифра;

D - будь-який символ окрім цифр;

w - будь-який алфавітно-цифровий символ, включаючи символ _;

s - символ прогалини (*space*);

S - будь-який символ окрім символу прогалини;

\b – обмежити символи між вказаним останнім символом двох слів;

\ - символ прямого слешу як знак перемикання коду (*escape*) для використання спеціальних мета-символів, як звичайних;

. (крапка) - будь-який символ.

В таблиці 1 наведемо приклади створення шаблонів для пошуку підрядків в рядках.

Таблиця 1 – Приклади шаблонів регулярного виразу для перевірки присутності підрядка у рядку

Опис завдання на перевірку присутності підрядка у рядку	Шаблон виразу
Перевірити присутність в рядку символів a або b в кінці рядка	*(a|b)$'*
Перевірити присутність в рядку будь-яких цифр з множини 2-9	*[2-9]*
Перевірити присутність в рядку символу 1 не більше одного разу (або його відсутність)	*1?*
Перевірити присутність в рядку символу 1 будь-яку кількість разів (або його відсутність)	*1**
Перевірити присутність в рядку символу 1 не менше одного разу	*1+*
Перевірити присутність в рядку символів 1111	*1{4}*
Перевірити присутність в рядку символів 1111, 11111 або 111111 та більше цифри 1	*1{4,}*
Перевірити присутність в рядку символів 1111 або 111111	*1{4,5}*
Перевірити присутність в рядку символу \	\\
Перевірити присутність в рядку підрядку типу 1X2, де X - будь-яка цифра	*1\d2*
Перевірити присутність в рядку послідовності з будь-яких трьох чисел	*\d{3}*
Перевірити присутність в рядку підрядку типу 1X2, де X - будь-який символ окрім цифр	*1\D2*
Перевірити присутність в рядку підрядку типу 1X2, де X - будь-який символ	*1.2*
Перевірити присутність в рядку підрядку з трьох будь-яких символів крім цифр	*\D{3}*
Перевірити присутність в рядку підрядку з 3-ма або 4-ма цифрами, після яких через пропуск йде 5 заголовних латинських букв	*[0-9]{3,4}[A-Z]{5}*
Перевірити присутність в рядку підрядку з дійсними числами	*([0-9]?\.[0-9]+)*
Перевірити присутність в рядку підрядку з описом часу у форматах типу «1:10» або «23:25»	*([01]?[0-9]|2[0-3]):[0-5][0-9]*
Перевірити присутність в рядку підрядку з описом часу у форматах типу «9:15 am» або «12:25 pm»	*(1[012]|[1-9]):[0-5][0-9]? (am|pm))*

На рисунку 2 представлено приклади виконання утиліти *grep* для підтвердження або спростування присутності підрядка у рядку у відповідності з описом шаблону регулярного виразу.

В прикладах вимагається, щоб рядок повністю відповідав шаблону, для чого шаблон обмежено з обох боків мета-символами прив'язки до початку та кінця рядку. При успішності перевірки на відповідність вмісту рядку шаблону регулярного виразу утиліта grep повертає значення цього рядку.

```
$ echo "23:50" | grep -E '^([01]?[0-9]|2[0-3]):[0-5][0-9]$'
23:50
```

(a) `echo "23:50" | grep -E '^([01]?[0-9]|2[0-3]):[0-5][0-9]$'` - рядок *23:50* відповідає шаблону

```
$ echo "24:00" | grep -E '^([01]?[0-9]|2[0-3]):[0-5][0-9]$'
```

(b) `echo "24:00" | grep -E '^([01]?[0-9]|2[0-3]):[0-5][0-9]$'` - рядок *24:00* не відповідає шаблону

Рис. 2 – Приклади команд перевірки вмісту рядків за шаблоном регулярних виразів *GREP*

1.1.3 *POSIX*-рекомендації з визначення мета-символів

Для забезпечення сумісності різних *Unix*-подібних ОС та переносності прикладних програм на рівні вихідного коду наприкінці 80-х років було запропоновано *POSIX (Portable Operating System Interface* - інтерфейс операційних систем) – набір стандартів, що описують інтерфейси між ОС та прикладною програмою (системний *API – Application Program Interface*), бібліотеку мови С, набір застосунків та їх інтерфейсів.

У регулярних виразах також є *POSIX*-рекомендації з визначення мета-символів, які вказуються безпосередньо у додаткових дужках опису класу – `[[:POSIX:]]`, наприклад:

- *[:alpha:]* – літери
- *[:lower:]* – літери в нижньому регістрі
- *[:upper:]* – літери у верхньому регістрі
- *[:digit:]* – цифри
- *[:alnum:]* – літери та цифри
- *[:space:]* – пробіли (не друковані символи), переклад каретки, новий рядок …
- *[:punct:]* – роздільні знаки пунктуації
- *[:cntrl:]* – керуючі символи
- *[:print:]* – друковані символи

В таблиці 2 наведено декілька прикладів шаблонів регулярного виразу з *POSIX* мета-символами для перевірки присутності підрядка у рядку. Регулярні вирази утиліти *GREP* використовують *POSIX*-рекомендації з визначення мета-символів.

У форматі телефонного номеру є символи круглих дужок та дефісу, які вже є мета-символами регулярного виразу і автоматично будуть розглядатися системою такими у шаблоні з відповідними помилками. Для того, щоб система не помітила такі символи, їх необхідно екранувати символом слеш \

Таблиця 2 – Приклади шаблонів регулярного виразу з *POSIX*-рекомендаціями визначення мета-символів для перевірки присутності підрядка у рядку

Опис завдання на перевірку присутності підрядка у рядку	Шаблон регулярного виразу з *POSIX*-рекомендаціями визначення мета-символів та їх еквівалент спрощеного опису

Перевірити присутність в рядку підрядку з телефонним номером, наприклад, такого формату (777) 77-77	`\([[:digit:]]{3}\)[[:space:]]*[[:digit:]]{2}\-[[:digit:]]{2}`
	або варіант спрощеного опису
	`\(\d{3}\)\s*\d{2}\-\d{2}`

Перевірити присутність в рядку підрядку з описом адреси електронної поштової скриньки, наприклад, *mail@ukr.net*	`[[:alpha:]][[:alnum:]]+@[[:alpha:]][[:alnum:]]+\.[:alpha:]{2,4}`
	або варіант спрощеного опису
	`[a-z]\w+@[a-z]\w+\.[a-z]{2,4}`

На рисунку 3 представлено приклади виконання утиліти *GREP* для підтвердження або спростування присутності підрядка у рядку у відповідності з описом шаблону регулярного виразу з урахуванням *POSIX*-рекомендацій визначення мета-символів.

```
$ echo "(777) 77-77" | grep -E '^\([[:digit:]]{3}\)[[:space:]]*[[:digit:]]{2}-[[:digit:]]{2}$'
(777) 77-77
```

(a) `echo "(777) 77-77" | grep -E '^\([[:digit:]]{3}\)[[:space:]]*[[:digit:]]{2}-[[:digit:]]{2}$'`, - рядок відповідає шаблону

```
$ echo "mail@ukr.net" | grep -E '^[[:alpha:]][[:alnum:]]+@[[:alpha:]][[:alnum:]]+\.[[:alpha:]]{2,4}$'
mail@ukr.net
```

(b) `echo "1mail@ukr.net" | grep -E '^[[:alpha:]][[:alnum:]]+@[[:alpha:]][[:alnum:]]+\.[[:alpha:]]{2,4}$'`, - рядок відповідає шаблону

Рис. 3 – Приклади команд перевірки вмісту рядків за шаблоном регулярних виразів *GREP з* урахуванням *POSIX*-рекомендацій визначення мета-символів

В шаблоні регулярного виразу із однієї частини виразу можна посилатися на іншу раніше визначено частину виразу. Для цього використовується мета-символ *n*, де *n* – порядковий номер логічної групи регулярного виразу, яку обмежено круглими дужками.

На рисунку 4 представлено приклади таких регулярних виразів.

```
$ echo "UNIXX - це ОС" | grep -E '([[:alpha:]])\1'
UNIXX - це ОС
```

(*a*) `echo "UNIXX - це ОС" | grep -E '([[:alpha:]])\1'` – перевіряється наявність у рядку двох символів-дублікатів

```
$ echo "UNIX це це ОС" | grep -E '([[:alpha:]]+)([[:space:]]+)\1'
UNIX це це ОС
```

(*b*) `echo "UNIX це це ОС" | grep -E '([[:alpha:]]+)([[:space:]]+)\1'` – перевіряється наявність у рядку двох слів-дублікатів

```
$ echo "UNIX це це це ОС" | grep -E '([[:alpha:]]+)([[:space:]]+)\1\2\1'
UNIX це це це ОС
```

(*c*) `echo "UNIX це це це ОС" | grep -E '([[:alpha:]]+)([[:space:]]+)\1\2\1'` – перевіряється наявність у рядку трьох слів-дублікатів

Рис. 4 – Приклади команд перевірки вмісту рядків за шаблоном регулярних виразів *GREP з* урахуванням логічних посилань

1.1.4 Пошук назв каталогів та файлів

Після успіху використання утиліти *grep* інші утиліт *Unix*-подібних ОС почали включати регулярні вирази. Утиліта *find* дозволяє шукати каталоги та файли за вказаних шляхом, який може бути описано регулярним виразом:

```
find шлях_пошуку опції регуляр_вираз_назви_файлу
```

Основна опція *-name* – шукати за назвою.

Найчастіше шляхи пошуку визначають через символи:

1) слеш `/` - визначає пошук, починаючи з кореневого каталогу файлової системи
2) крапка слеш `./` - визначає пошук, починаючи з поточного каталогу.

Наприклад, знайти всі каталоги та файли, починаючи з кореневого каталогу `/`, у назві яких першою буквою є буква *b*:

```
find / -name "b*"
```

Наприклад, знайти у поточному каталозі всі каталоги та файли, які починаються з послідовності літер *abc* з перенаправленням всіх повідомлень про помилки (файловий дескриптор=2 потоку *stderr*) у файл невизначеного пристрою */dev/null*

```
find ./ -name "evc*" 2> /dev/null
```

На рисунку 5a наведено приклад результату роботи команди *find* для отримання переліку всіх каталогів та файлів, починаючи з поточного каталогу, який було раніше створено як *Git*-репозиторій. А на рисунку 5b наведено приклад результат цієї ж команди, але вже після створення файлу *README.md* та його фіксації у *Git*-репозиторії. Як видно з рисунку, *Git* додатково створив декілька системних каталогів в каталозі *objects*.

```
$ find ./
./
./.git
./.git/config
./.git/description
./.git/HEAD
./.git/hooks
./.git/hooks/applypatch-msg.sample
./.git/hooks/commit-msg.sample
./.git/hooks/fsmonitor-watchman.sample
./.git/hooks/post-update.sample
./.git/hooks/pre-applypatch.sample
./.git/hooks/pre-commit.sample
./.git/hooks/pre-merge-commit.sample
./.git/hooks/pre-push.sample
./.git/hooks/pre-rebase.sample
./.git/hooks/pre-receive.sample
./.git/hooks/prepare-commit-msg.sample
./.git/hooks/push-to-checkout.sample
./.git/hooks/update.sample
./.git/info
./.git/info/exclude
./.git/objects
./.git/objects/info
./.git/objects/pack
./.git/refs
./.git/refs/heads
./.git/refs/tags
```

(*a*) вміст поточного каталогу після ініціалізації *Git*-репозиторію

```
./.git/hooks/post-update.sample
./.git/hooks/pre-applypatch.sample
./.git/hooks/pre-commit.sample
./.git/hooks/pre-merge-commit.sample
./.git/hooks/pre-push.sample
./.git/hooks/pre-rebase.sample
./.git/hooks/pre-receive.sample
./.git/hooks/prepare-commit-msg.sample
./.git/hooks/push-to-checkout.sample
./.git/hooks/update.sample
./.git/index
./.git/info
./.git/info/exclude
./.git/logs
./.git/logs/HEAD
./.git/logs/refs
./.git/logs/refs/heads
./.git/logs/refs/heads/main
./.git/objects
./.git/objects/05
./.git/objects/05/192b269e4ab3bbbdc9f534daafa4330e2c2383
./.git/objects/17
./.git/objects/17/7201adc6582860e4938104b9fe5cbf1e3f8012
./.git/objects/b4
./.git/objects/b4/3bf86b50fd8d3529a0dc062c30006ed38f309e
./.git/objects/info
./.git/objects/pack
./.git/refs
./.git/refs/heads
./.git/refs/heads/main
./.git/refs/tags
./README.md
```

(*b*) вміст поточного каталогу з *Git*-репозиторієм після створення файлу *README.md*

```
$ find ./ -name '*.sample'
./.git/hooks/applypatch-msg.sample
./.git/hooks/commit-msg.sample
./.git/hooks/fsmonitor-watchman.sample
./.git/hooks/post-update.sample
./.git/hooks/pre-applypatch.sample
./.git/hooks/pre-commit.sample
./.git/hooks/pre-merge-commit.sample
./.git/hooks/pre-push.sample
./.git/hooks/pre-rebase.sample
./.git/hooks/pre-receive.sample
./.git/hooks/prepare-commit-msg.sample
./.git/hooks/push-to-checkout.sample
./.git/hooks/update.sample
```

(c) `find ./ -name '*.sample'`

```
$ find ./ -name 'pre*'
./.git/hooks/pre-applypatch.sample
./.git/hooks/pre-commit.sample
./.git/hooks/pre-merge-commit.sample
./.git/hooks/pre-push.sample
./.git/hooks/pre-rebase.sample
./.git/hooks/pre-receive.sample
./.git/hooks/prepare-commit-msg.sample
```

(d) `find ./ -name 'pre*'`

```
$ find ./ -name '[0-9]*'
./.git/objects/05
./.git/objects/05/192b269e4ab3bbbdc9f5
./.git/objects/17
./.git/objects/17/7201adc6582860e49381
./.git/objects/b4/3bf86b50fd8d3529a0dc
```

(e) `find ./ -name '[0-9]*'`

```
$ find ./ -name '[[:digit:]]*'
./.git/objects/05
./.git/objects/05/192b269e4ab3bbbdc9f5
./.git/objects/17
./.git/objects/17/7201adc6582860e49381
./.git/objects/b4/3bf86b50fd8d3529a0dc
```

(f) `find ./ -name [[:digit:]]*'`

Рис. 5 –приклади роботи команди *find ./*

На жаль, ефективність використання регулярних виразів для утиліти *find* суттєво залежить від особливостей бібліотек *Unix*-подібних ОС. Наприклад, можуть не працювати символи-прив'язки, додаткові мета-символи, квантифікатори. Тому для складних пошукових запитів рекомендується утиліту *find* використовувати у конвеєрі з командою *grep*.

Наведемо приклади виконання попередніх команд з урахуванням конвеєру:

```
find / | grep -E "b*"
find ./ 2> /dev/null | grep -E "evc*"
```

Для пошуку назв каталогів або файлів, які містять не менше 2-х шістнадцятирічних цифр (*0-9abcdef*) необхідно виконати ланцюжок конвеєру команд:

```
find ./ | grep -E "/[0-9abcdef]{2}/"
```

На рисунку 6 представлено приклади пошуку рядку «*Changed*» у файлах прихованого *.git*-каталогу. Використовується дворівневий пошук в каталогах за регулярним виразом */*. Як видно за результатами пошуку, вказаний рядок було використано як частину коментаря *Git*-команди *commit*. В процесі пошуку файлів з'являються повідомлення про помилку, коли утиліта намагається переглянути вміст не файлу, каталогу. Для того, щоб ці повідомлення не заважали, їх можна перенаправити зі стандартного потоку помилок (файловий дескриптор=2 потоку *stderr*) до спеціального фіктивного пристрою */dev/null*

```
$ grep "Changed" */*
logs/HEAD:0000000000000000000000000000000000000000 05192b269e4ab3bbbdc9f534daafa
4330e2c2383 infosystemdepartment <infosystemdepartment@gmail.com> 1649129115 +03
00      commit (initial): Changed by Local Git
grep: logs/refs: Is a directory
grep: objects/05: Is a directory
grep: objects/17: Is a directory
grep: objects/b4: Is a directory
grep: objects/info: Is a directory
grep: objects/pack: Is a directory
grep: refs/heads: Is a directory
grep: refs/tags: Is a directory
```

(a) *grep "Changed" */**

```
$ grep "Changed" */* 2> /dev/null
logs/HEAD:0000000000000000000000000000000000000000 05192b269e4ab3bbbdc9f534daafa
4330e2c2383 infosystemdepartment <infosystemdepartment@gmail.com> 1649129115 +03
00      commit (initial): Changed by Local Git
```

(b) *grep "Changed" */* 2> /dev/null*

Рис. 6 – Приклади пошуку рядку «*Changed*» у файлах прихованого *.git*-каталогу

1.2 Автоматизована модифікація текстових даних потоковим редактором *SED*

Утиліта *GREP* дозволяє автоматизувати пошук підрядків у рядках великого текстового файлу. А що далі з цими рядками робити? Як автоматично, тобто швидко:

– отримати саме значення підрядків, а не весь, можливо, рядок великого розміру;

– видалити увесь рядок або лише підрядок;

– замінити значення підрядків на якесь нове константе значення або за шаблоном.

Утиліта *SED* (***S****tream* ***ED****itor*) була створена у 1974 році *Lee McMahon,* співробітником компанії *Bell Labs,* як потоковий текстовий редактор, що застосовує текстові перетворення до послідовного потоку текстових даних.

SED відрізняється від звичайних текстових редакторів: звичайні текстові редактори спочатку завантажують весь текст документу, а потім застосовують до нього команди по одній, тоді як *SED* спочатку завантажує в себе набір команд, а потім застосовує весь набір команд до кожного рядка тексту. Оскільки одночасно в пам'яті знаходиться тільки один рядок з вхідного потоку, *SED* може обробити великі текстові файли.

Утиліта *AWK* (перші літери прізвищ авторів *Alfred* ***A****ho, Peter* ***W****einberger, Brian* ***K****ernighan*) була створена у 1977 як *C*-подібна скриптова мова рядкового аналізу та обробки вхідного потоку (наприклад, текстового файлу), тому має більш потужні можливості ніж *SED*.

Утиліта *SED* є простішою за функціоналом та меншою за розміром ніж утиліта *AWK*, тому, ймовірно, що для простих завдань, які будуть одночасно виконуватися багатьма процесами, використання утиліти *SED* забезпечить менші витрати ресурсів ОС ніж при використання утиліти *AWK*. Але все треба перевіряти через серію натурних експериментів, про що докладніше дізнаємося у наступній лабораторній роботі.

Основний синтаксис утиліти *SED*:

sed 'опис команд з обробки рядків' вхідний_файл [опції]

Популярні опції утиліти:

– *n* – не виводити рядки, які було оброблено командами, на стандартний потік *stdout*;

– *f файл* – завантаження команд з окремого файлу (файл-скрипт), де кожна команда знаходиться в окремому рядку;

– *E* – використання *Extended Regular Expressions (ERE)* або нової версії синтаксису.

В таблиці 3 наведено приклади команд *SED*:

– *p* – друк рядків (направлення на *stdout*-потік);

– *i* або *a* – додавання нових рядків;

– *d* – видалення рядків за шаблоном;

– *s* – заміна підрядків за шаблоном;

– *y* – заміна символів за шаблоном (еквівалент утиліті *tr*).

Таблиця 3 – Приклади операцій *sed*

Операція	Опис
[Діапазон рядків]p	Друкувати зазначений діапазон рядків або усі рядки, якщо діапазон не вказано
[Номер рядка]i\Рядок	Додати новий рядок *Рядок* перед вказаним номером рядка або перед кожним рядком, якщо номер не вказано
[Номер рядка]a\Рядок	Додати новий рядок *Рядок* після вказаного номеру рядка або після кожного рядка, якщо номер не вказано
[Діапазон рядків]d	Видалити вказаний діапазон рядків або усі рядки, якщо діапазон не вказано
/Шаблон/d	Видалити рядки, які відповідають шаблону *Шаблон*
/Шаблон1/,/Шаблон2/d	Видалити рядки, починаючи з рядка, який відповідає шаблону *Шаблон1* та закінчуючи рядком, який відповідає шаблону *Шаблон2*
[Діапазон рядків] s/Шаблон/Підрядок/ прапорець	Замінити у кожному рядку першу появу підрядку, який відповідає шаблону *Шаблон*, на підрядок *Підрядок* у діапазоні або у всіх рядках, якщо діапазон не вказано Прапорці: – порядковий номер заміни (за замовчуванням = 1), – *g* – заміна усіх знайдених підрядків (за замовчуванням – 1-й підрядок), – *p* – додатково вивести вміст початкового рядка
[Діапазон рядків] y/Символи1/Символи2/	Замінити у кожному рядку символи з впорядкованого списку *Символи1* на відповідні їм за порядковими номерами символи зі списку *Символи2*

Діапазон рядків можна визначати як:

1) *r1* – один зазначений номер рядку;

2) *r1,r2* – множина номерів рядків від *r1* до *r2*;

3) *r1,$* – від вказаного початкового номеру рядку до кінця файлу

Команда друкування рядків найчастіше використовується для тестування (перевірки) правильності виконання інших рядків.

Можна вказувати декілька команд, опис яких розділяється символом двокрапка.

На рисунку 7 наведено приклади команд друкування рядків утилітою *SED (*працює з *stdout*-потоком, отриманим від конвеєру ланцюжка команд *echo* та *tr*.

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed ''
Рядок1
Рядок2
```

(*a*) виконати утиліту без команд, результат – вміст всіх рядків

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed 'p'
Рядок1
Рядок1
Рядок2
Рядок2
```

(*b*) друкувати кожний рядок, результат – дублювання рядків

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed '1,2p'
Рядок1
Рядок1
Рядок2
Рядок2
```

(*c*) виконати утиліту з друком рядків за номерами від 1-го до 2-го

Рис. 7 – Приклади команд друку рядків утилітою *SED* (працює з *stdout*-потоком, отриманим від конвеєру ланцюжка команд *echo* та *tr*)

На рисунку 8 наведено приклади команд додавання нових рядків утилітою *SED*.

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed 'i\Новий'
Новий
Рядок1
Новий
Рядок2
```

(*a*) додати новий рядок перед кожним існуючим рядком

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed 'a\Новий'
Рядок1
Новий
Рядок2
Новий
```

(*b*) додати новий рядок після кожного існуючого рядка

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed '1a\Новий'
Рядок1
Новий
Рядок2
```

(*c*) додати новий рядок після 1-го існуючого рядка

Рис. 8 – Приклади команд додавання нових рядків утилітою *SED*

Утиліта *SED* може читати команди, попередньо розміщені в окремому файлі, що дозволяє створити пакет з великою кількістю команд. На рисунку 9 показано приклад файлу *cmd.sed* з командами та приклад роботи утиліти *SED*, яка читає команди з цього файлу.

```
$ cat cmd.sed
1i\Новий1
1a\Новий2
```

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed -f cmd.sed
Новий1
Рядок1
Новий2
Рядок2
```

Рис. 9 – Приклад використання утиліти *SED*, яка читає команди з файлу *cmd.sed*

На рисунку 10 наведено приклади команд видалення рядків утилітою.

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed '1d'
Рядок2
```

(a) видалити 1-й за номером рядок

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed '1,$d'
```

(b) видалити рядки з першого до останнього за номерами

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed '/Рядок1/d'
Рядок2
```

(c) видалити рядок за шаблоном значення, результат – видалено Рядок1

Рис. 10 – Приклади команд видалення рядків утилітою *SED*

Для визначення номеру рядків, які відповідають шаблону пошуку, використовується команда =, яка ці номери виводить перед знайденими рядками на стандартний потік *stdout*. Самі рядки можна приховати опцією *–n*, що дозволяє спростити процес подальшого використання цих номерів, наприклад:

```
echo "Рядок1;Рядок2" | tr ';' '\n' | sed -n '/Рядок1/='
```

На рисунку 11 наведено приклади команд зміни значення рядків утилітою *SED* (працюють з *stdout*-потоком, отриманим від конвеєру ланцюжка команд *echo* та *tr*).

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed 's/Рядок1/Новий/'
Новий
Рядок2
```

(a) замінити шаблон підрядку *Рядок1* на значення *Новий*

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed 's/Рядок/Новий/'
Новий1
Новий2
```

(b) замінити шаблон підрядку *Рядок* на значення *Новий*

```
$ echo "Рядок1;Рядок2" | tr ';' '\n' | sed 's/Рядок[1-2]/Новий/'
Новий
Новий
```

(c) замінити шаблон підрядку *Рядок[1-2]* на значення *Новий*, результат – змінено значення *Рядок1*, *Рядок2* на *Новий*

Рис. 11 – Приклади команд зміни значення рядків утилітою *SED*

На рисунку 12 наведено приклади команд зміни значення рядків утилітою *SED* з урахування шаблону заміни.

```
$ echo "12+2=14" | sed -E 's/[[:digit:]]/цифра/'
цифра2+2=14
```

(a) echo "12+2=14" | sed -E 's/[[:digit:]]/цифра/'

– замінити перше співпадіння шаблону на слово «цифра»

```
$ echo "12+2=14" | sed -E 's/[[:digit:]]/цифра/g'
цифрацифра+цифра=цифрацифра
```

(b) echo "12+2=14" | sed -E 's/[[:digit:]]+/число/g'

– замінити всі співпадіння шаблону на слово «цифра»

```
$ echo "12+2=14" | sed -E 's/[[:digit:]]+/число/g'
число+число=число
```

(b) echo "12+2=14" | sed -E 's/[[:digit:]]+/число/g'

– замінити всі співпадіння шаблону на слово «число»

Рис. 12 – Приклади команд зміни значення рядків утилітою *SED* з урахування шаблону

На рисунку 13 приклади зміни значення рядків *з* урахуванням логічних посилань \

```
$ echo "значення1:значення2" | sed -E 's/([^:]+)(.*)/\1/'
значення1
```

(*a*) *echo "значення1:значення2" | sed -E 's/([^:]+)(.*)/\1/'* – отримати з рядку підрядок, який знаходиться перед символом : (двокрапка)

```
$ echo "UNIXX - це ОС" | sed -E 's/([[:alpha:]])\1/\1/'
UNIX - це ОС
```

(*b*) *echo "UNIXX - це ОС" | sed -E 's/([[:alpha:]])\1/\1/'* – видалити символ-дублікат

```
$ echo "UNIX - це це ОС" | sed -E 's/([[:alpha:]]+)([[:space:]]+)\1/\1/'
UNIX - це ОС
```

(*c*) *echo "UNIX - це це ОС" | sed -E 's/([[:alpha:]]+)([[:space:]]+)\1/\1/'* – видалити одне слово-дублікат

```
$ echo "UNIX - це це це ОС" | sed -E 's/([[:alpha:]]+)([[:space:]]+)\1\2\1/\1/'
UNIX - це ОС
```

(*d*) *echo "UNIX - це це це ОС" | sed -E 's/([[:alpha:]]+)([[:space:]]+)\1\2\1/\1/'* – видалити декілька слів-дублікатів

```
$ echo "UNIX - старовинна ОС" | sed -E 's/(.*)\s(.*)\s(.*)\s(.*)/\3 \4 \2 \1/'
старовинна ОС - UNIX
```

(*e*) *echo "UNIX - старовинна ОС" | sed -E 's/(.*)\s(.*)\s(.*)\s(.*)/\3 \4 \2 \1/'* – змінити порядок слідування слів

```
$ echo "<html>test</html>" | sed -E 's/(<html>)([a-z]+)(<\/html>)/\2/'
test
```

(*f*) *echo "<html>test</html>" | sed -E 's/(<html>)([a-z]+)(<\/html>)/\2/'* – отримати з рядку значення підрядка, наприклад, значення з *html*-тегу

```
$ echo "<td>комп'ютер</td>" | sed -E 's/(<td>)([[:alpha:]]|[[:print:]]+)(<\/td>)/\2/'
комп'ютер
```

(*g*) *echo "<td>комп'ютер</td>" | sed -E 's/(<td>)([[:alpha:]]|[[:print:]]+)(<\/td>)/\2/'* – отримати з рядку значення підрядка як 2-ї логічної групи, який може мати символи кирилиці та спеціальні символи, наприклад, апостроф

Рис. 13 – Приклади команд зміни значення рядків *з* урахуванням логічних посилань

На рисунку 14 представлено приклади зміни регістру символів на основі додаткових мета-символів \U – перевести у верхній регістр (*Upper*), або \L – перевести у нижній (*Lower*)

```
$ echo "<td>мова</td>" | sed -E 's/(<td>)([[:alpha:]]+)(<\/td>)/\U\2/'
МОВА
```

(*a*) *echo "<td>мова</td>" | sed -E 's/(<td>)([[:alpha:]]+)(<\/td>)/\U\2/'* – отримати з рядку значення підрядка як 2-ї логічної групи та перевести всі символи у верхній регістр

```
$ echo "<td>мОВА</td>" | sed -E 's/(<td>)([[:alpha:]])([[:alpha:]]+)(<\/td>)/\U\2\L\3/'
Мова
```

(*b*) *echo "<td>мОВА</td>" | sed -E 's/(<td>)([[:alpha:]])([[:alpha:]]+)(<\/td>)/\U\2\L\3/'* – отримати з рядку значення 1-го символу підрядка як 2-ї логічної групи, перевести його у верхній регістр, інші символи з 3-ї групи перевести у нижній регістр

Рис. 14 – Приклади команд зміни значення рядків *з* урахуванням логічних посилань та мета-символів зміни регістру символів на верхній або нижній

1.3 Особливості роботи з *Online*-інструментом *regex101*

Для проведення універсальних експериментів з регулярними виразами, які не потребують використовувати будь-які мови програмування та утиліти, можна скористатися *Online*-інструментарієм, наприклад, за адресою https://regex101.com/

Як видно з рисунку 15, *regex101* дозволяє перевіряти результат виконання регулярних виразів з урахуванням різних мов програмування (*PHP, JavaSctipt, Python, Golang, Java, C#*).

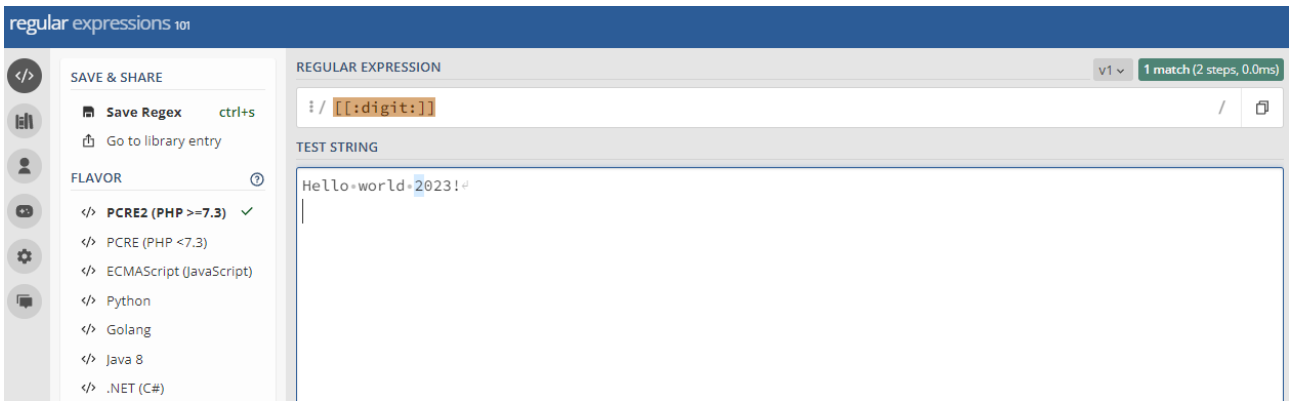

Рис. 15 – Фрагмент екрану *Online*-інструменту *regex101*

Починаючи роботу з *regex101*, треба виконати попередні налаштування, інакше можуть виникнути проблеми у використанні. Наприклад, з рисунку 15 видно, що аналіз рядка регулярним виразом надав підсвітку лише однієї цифри. Це пов'язано зі станом спеціальних прапорців (*Regexp Flags*), які керують процесом обробки рядків. На рисунку 16 наведено приклад таких прапорців та показано результат виставлення прапорця *global*, який примушує продовжувати пошук після успішно знайденого першого підрядка з цифрами та підсвічувати у рядку вже всі знайдені підрядки з цифрами.

Додаткові користні прапорці:

– прапорець *multi line* – для обробки декількох окремих рядків;

– прапорець *unicode* – для обробки символів Кирилиці.

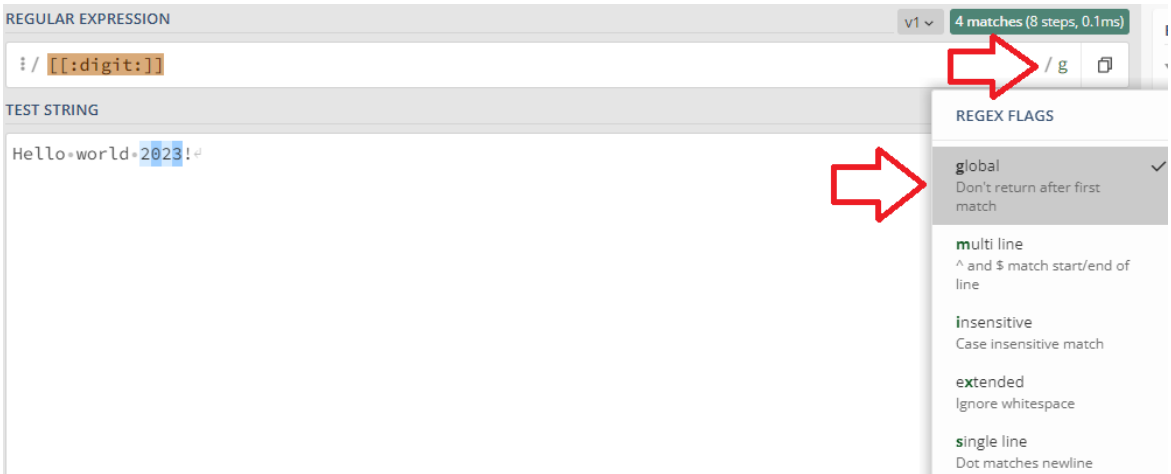

Рис. 16 – Фрагмент екрану з налаштування прапорців

2 Завдання до виконання

2.0 Підготовка до виконання завдань

Використовуючи команди *Bash*, виконати наступні завдання.

2.0.1 Виконати безпечне клонування *GitHub*-репозиторію, який був наданий вам викладачем, створивши на локальному комп'ютері *Git*-репозиторій, або оновити зміст *Git*-репозиторію, який раніше вже було клоновано.

Перейти до катологу із *Git*-репозиторієм.

2.0.2 Створити нову *Git*-гілку з назвою «*Laboratory-work-4*».

Перейти до роботи зі створеною гілкою.

2.0.3 Створити каталог з назвою «*Laboratory-work-4*». Перейти до каталогу.

2.0.4 В каталозі «*Laboratory-work-4*» створити порожній файл *README.md*, використовуючи команду інтерпретатору командного рядку *Bash*.

2.0.5 Використовуючи текстові редактори, які пропонуються оболонкою *Git Bash*, наприклад, текстовий редактор *nano*, додати до файлу *README.md* рядок тексту із темою лабораторної роботи: «*Складна обробка текстових даних засобами оболонки Unix-подібних ОС інтерпретора команд ОС*».

Для рядка визначити *Markdown*-форматування як заголовок 2-го рівня.

Для збереження змін та завершення роботи в редакторі *nano* можна вказати комбінацію клавіш *Ctrl+O* та *Ctrl+X*, відповідно.

2.0.6 Виконати операції з оновлення *GitHub*-репозиторію змінами *Git*-репозиторія через послідовність *Git*-команд *add, commit* із коментарем «*Changed by Local Git*» та *push*.

2.0.7 На веб-сервісі *GitHub* перейти до створеної гілки «*Laboratory-work-4*».

Перейти до каталогу «*Laboratory-work-4*» та розпочати процес редагування файлу *README.md*

2.1 Пошук у системних файлах *Git*-репозиторію

Документуючі рішення цього розділу, у файлі *README.md* необхідно вказати наступний текстовий рядок у вигляді заголовку 3-го рівня *Markdown*-форматування:
«*1 Пошук у системних файлах Git-репозиторію*». В подальшому за результатами рішень кожного пункту завдання до файлу *README.md* додати знімки екрану та підписи до знімків екранів, які містять опис завдань.

Знаходячись у кореневому каталозі *Git*-репозиторія, перейти до прихованого каталогу *.git* та виконати наступні завдання.

2.1.1 Вивести на екран перелік каталогів та файлів поточного каталогу, а також всіх файлів з підкаталогів (рекомендується використати команду `find`).

2.1.2 Вивести на екран перелік каталогів або файлів поточного каталогу, назви яких починаються з цифри та завершуються цифрою, а в середині – будь-які символи (рекомендується використати команду `find`).

2.1.3 Вивести на екран перелік каталогів або файлів, назви яких мають підрядок з не менше ніж трьох цифр (рекомендується використати конвеєр команд `find` та `grep`).

2.1.4 Повторити виконання попереднього завдання, але лише для назв файлів або назв каталогів, які відокремлено один від одного (рекомендується використати конвеєр команди `find` та ланцюжка команд «`tr, sort, grep`» для трансформації назв каталогів та файлів як це було у попередній лабораторній роботі).

2.1.5 Вивести на екран перелік файлів, назви яких мають лише цифри (рекомендується використати конвеєр команд `find, tr, sort, grep`).

2.1.6 Вивести на екран перелік файлів, назви яких мають лише символи, які пов'язані з шістнадцятирічними цифрами, наприклад, *0-9* та *a-f* (рекомендується використати конвеєр команд `find, tr, sort, grep`).

2.1.7 Вивести на екран рядок, в якому може зберігатися підрядок з коментарем для для команди *commit*, який починається зі слова «*Changed*», при цьому потік помилок необхідно перенаправити до спеціального фіктивного пристрою `/dev/null` (рекомендується використати лише команду `grep */*` з шаблоном пошуку файлів у каталогах дворівневої глибини).

2.1.8 Вивести на екран рядок, в якому може зберігатися підрядок з електронною поштовою скринькою облікового запису *Git*-користувача, який виконував команду *commit*, при цьому необхідно використати шаблон регулярного виразу, який забезпечить пошук будь-яких інших подібних поштових скриньок (рекомендується використати лише команду *grep* `*/*` з шаблоном пошуку файлів у каталогах дворівневої глибини).

Примітка: під час виконання команди grep не забувайте про опцію –E, без якої не працює багато мета-символів мови регулярних виразів.

2.2 Складний пошук та заміна текстових даних

Документуючі рішення цього розділу, у файлі *README.md* необхідно вказати наступний текстовий рядок у вигляді заголовку 3-го рівня *Markdown*-форматування:
«2 Складний пошук та заміна текстових даних». В подальшому за результатами рішень кожного пункту завдання до файлу *README.md* додати знімки екрану та підписи до знімків екранів, які містять опис завдань.

GitHub-репозиторій https://github.com/oleksandrblazhko/OS-LabWork4-Examples

містить файли *HTML*-формату, які визначено у відповідності із вашим варіантом попередньої лабораторної роботи.

Перед виконанням завдань перейти до домашнього каталогу вашого користувача ОС, наприклад, виконавши команду `cd`

Клонувати представлений *GitHub*-репозиторій на ваш локальний комп'ютер.

Перейти до каталогу *Git*-репозиторію та виконати наступні завдання.

2.2.1 З отриманого каталогу вивести на екран рядки файлів, в яких є рядок з підрядком з номером вашого варіанту завдання, наприклад, *Варіант 0* (рекомендується використати команду `grep`);

2.2.2 Модифікувати рішення попереднього завдання, вивівши на екран лише назву файлу без зайвої інформації про рядок файлу (рекомендується використати конвеєр команд `grep, sed`).

2.2.3 Скопіювати знайдений файл у каталог «*Laboratory-work-4*» *Git*-репозиторія, використовуючи команду *cp*. Перейти до каталогу «*Laboratory-work-4*».

2.2.4 Вивести на екран вміст знайденого файлу без порожніх рядків (рекомендується використати в подальшому команду `sed`).

2.2.5 Вивести на екран рядки із словами-дублікатами (однакові слова, які йдуть один за одним).

2.2.6 У файлі *HTML*-формату є рядки з *html*-тегами *<td>* та цілими числами. Вивести на екран ці рядки, перетворюючи цілі числа у числа з плаваючою комою, додавши до підрядка з числом підрядок .00, наприклад, підрядок «10» буде перетворено на «10.00»

2.2.7 У файлі є рядки з *html*-тегами *<td>*, які замість чисел містять символи-роздільники (дефіс, відсоток, три крапки). Вивести на екран ці рядки, перетворюючи символи-роздільники на символи прогалини (*space*).

2.2.8 У файлі є підрядки зі словом "Об'єкт". Вивести на екран це слово, замінивши його перший символ, який є у верхньому регістрі, на символ у нижньому регістрі.

2.3 Автоматизована модифікація файлів з текстовими даними

Документуючі рішення цього розділу, у файлі *README.md* необхідно вказати наступний текстовий рядок у вигляді заголовку 3-го рівня *Markdown*-форматування:
«3 Автоматизована модифікація файлів з текстовими даними». В подальшому за результатами рішень кожного пункту завдання до файлу *README.md* додати знімки екрану та підписи до знімків екранів, які містять опис завдань.

Використовуючи знайдений у попередньому завданні *HTML*-файл у відповідності з вашим варіантом, виконати утилітою *SED* дії зі зміни вмісту цього файлу, створивши новий

файл, який починається з вашого прізвища латинськими літерами, наприклад, *blazhko_example.html*:

2.3.1 У файлі є рядок з *html*-тегом <*title*>. Видалити з цього рядка цифри, які розміщено наприкінці рядка.

2.3.2 У файлі є рядок з *html*-тегом <*title*>. Додати після цього рядка новий рядок, який містить наступне: "<*h1*>Таблиця оновлено автоматично. Автор - ПІБ, група</*h1*>" (рекомендується додати за номером, який заздалегіть визначено попердньою командою *sed* наприклад, після 4-го рядку).

2.3.3 Видалити з файлу всі порожні рядки.

2.3.4 Видалити з файлу слова-дублікати.

2.3.5 Об’єднати команди *SED*, створені у попередніх завданнях, в окремий текстовий файл з назвою за шаблоном *surname.sed*, де *surname* – ваше прізвище латинськими літерами. Виконати утиліту *SED* з читанням команд зі створенного файлу.

2.4 Підготовка процесу *Code Review* для надання рішень завдань лабораторної роботи на перевірку викладачем

2.4.1 На веб-сервісі *GitHub* зафіксувати зміни у файлі *README.md*

2.4.2 Скопіювати файли, які було створено у попередньому розділі, в каталог «*Laboratory-work-4*» *Git*-репозиторію.

2.4.3 Оновити *Git*-репозиторій змінами нової гілки «*Laboratory-work-4*» з *GitHub*-репозиторію.

2.4.4 Оновити *GitHub*-репозиторій змінами нової гілки «*Laboratory-work-4*» *Git*-репозиторію.

2.4.5 Виконати запит *Pull Request*.

Примітка: Увага! Не натискайте кнопку «Merge pull request»!

Це повинен зробити лише рецензент, який є вашим викладачем!

Коли рецензент-викладач перегляне ваше рішення він виконає злиття нової гілки та основної гілки, натиснувши кнопку «*Merge pull request*». Якщо рецензент знайде помилки, він повідомить про це у коментарях, які з’являться на сторінці *Pull request.*

2.5 Оцінка результатів виконання завдань лабораторної роботи

Оцінка	Умови
+3 бали	1) всі рішення відповідають завданням 2) *Pull Request* представлено не пізніше найближчої консультації після офіційного заняття із захисту лабораторної роботи
-0.5 балів за кожну помилку	в рішенні є помилка, про яку вказано в *Code Review*
-1 бал	*Pull Request* представлено пізніше дати найближчої консультації після офіційного заняття із захисту лабораторної роботи за кожний тиждень запізнення
+2 бали	Отримано правильну відповідь на два запитання, які стосуються призначення команд, представлених на знімках екранів