ОГЛАВЛЕНИЕ

1.BASH-SCRIPT

1. writer_zero-six.sh

Входные значения:

$1 - дата нижней парсинга;

$2 - дата верхней границы парсинга;

time – текущее время.

Данный скрипт осуществляет запись логов по отдельным файлам.

Используемые bash команды:

1. touch предназначен для создания пустого файла;

2. sh - Запись скрипта;

3. journalctl - команда терминала Astra Linux предназначенная для прочтения лог файлов, и их уровней приоритета полученных ядром.

Флаг -p для прочтения конкретного типа логов с определенным уровнем приоритета.

7 - все логи; 0 - неработоспособность системы; 1 - alerts; 2 - критическая ошибка; 3 - ошибки; 4 - предупреждения; 5 - уведомления; 6 - информационные сообщения;

4. grep - перехват и запись определенных логов;

5. cat - утилита Linux, позволяющая взаимодействовать с файлами - в данном коде применяется для записи файлов из буффера;

6. sort - предназначена для ортировки значений, необходимых команде uniq;

7. uniq - игнорирует повторяющиеся строки;

8. echo - n > "тип файла" - предназначен для очистки логов из переменной;

9. rm "путь файла" – удаление файла-буффера;

10. if ["$endDate" -ne "$asloDate"] – проверка двух дат на индентичность;

Пример использования:

1. touch src/main/java/logFiles/buffer - touch предназначен для создания пустого файла buffer, для записи всех логов в буффер;

2. sh src/main/java/scrypt/clear.sh - #Перед записью логов - необходима очистка файлов, для отслеживания логов в конкретный промежуток времени;

3. journalctl -p 1 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4}\ [0-9]{1,2}\ [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer - Предназначен для записи строк логов в определенный промежуток времени в буффер;

4. cat src/main/java/logFiles/buffer src/main/java/logFiles/emergencyLogs/emergency |sort |uniq -u >> src/main/java/logFiles/alertsLogs/alerts

- Предназачен для записи уникальных строк кода из буффера в ошибки;

5. echo -n > src/main/java/logFiles/buffer - предназачен для очистки буффера для последущих уровней приоритетов лога;

6.

```
if [ "$endDate" -ne "$asloDate" ]
then
  time="23:59:59"
fi
```

Листинг 1.1

Данный фрагмент кода предназначен для проверки дат. В случае если сегодня не крайний день парсинга, то условие будет выполнено и значение времени будет установлено крайним за прошедшие сутки. Создано это для того, чтобы можно было отследить все операции за предыдущий день.

Если же сегодня крайний день, то проверка логов будет проходить до момента активации всей программы.

Подробные шаги предоставлены в листинге А.1:

2. clear.sh

Входные данные - отсутствуют

Данный скрипт осуществляет очистку всех файлов логов за сессию работы программы

Используемые bash команды:

1. echo - это команда для вывода команды в терминал или в файл;

Флаг -n Предназначен для опускания повторяющегося перевода последней строки;

Опция > предназначена для перезаписи файла, поскольку передается пустой аргумент командой строки - все файлы перезаписываются пустыми.

src/main/java/logfiles - места для записи логов, предназначеные для разделения логов по типам

Пример использования:

echo -n > src/main/java/logFiles/emergencyLogs/emergency - данная команда перезаписывает файл с пустым значением - совершая очистку для будущей обработки файлов, что позволяет вести работу с буффером. Подробные шаги предоставлены в листинге А.2:

3. log_writer_by_date.sh

Входные параметры: $1, $2

Где: $1 – начальное значение парсинга,

$2 – конечное значение парсинга

Результат работы: записывать информацию из логов в файл содержащий в себе информацию об всех типов логов.

Используемые bash команды:

1. touch предназначен для создания пустого файла;

2. sh - Запись скрипта;

3. journalctl - команда терминала Astra Linux предназначенная для прочтения лог файлов, и их уровней приоритета полученных ядром.

Флаг -p для прочтения конкретного типа логов с определенным уровнем приоритета.

7 - все логи; 0 - неработоспособность системы; 1 - alerts; 2 - критическая ошибка; 3 - ошибки; 4 - предупреждения; 5 - уведомления; 6 - информационные сообщения;

4. grep - перехват и запись определенных логов;

5. cat - утилита Linux, позволяющая взаимодействовать с файлами - в данном коде применяется для записи файлов из буффера;

6. sort - предназначена для ортировки значений, необходимых команде uniq;

7. uniq - игнорирует повторяющиеся строки;

8. echo - n > "тип файла" - предназначен для очистки логов из переменной;

9. rm "путь файла" – удаление файла-буффера;

10. if ["$endDate" -ne "$asloDate"] – проверка двух дат на индентичность;

Пример использования:

1. sh src/main/java/scrypt/clear.sh - Перед записью логов - необходима очистка файлов, для отслеживания логов в конкретный промежуток времени;

2. journalctl -p 7 --since=$startDate --until=$endDate | grep -E '^[а-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/allTypesLogs/all_types - Предназначен для записи строк логов в определенный промежуток времени в лог файл содержащий все виды логов;

3.

```
if [ "$endDate" -ne "$asloDate" ]
then
  time="23:59:59"
fi
```

Листинг 3.1

Данный фрагмент кода предназначен для проверки дат. В случае если сегодня не крайний день парсинга, то условие будет выполнено и значение времени будет установлено крайним за прошедшие сутки. Создано это для того, чтобы можно было отследить все операции за предыдущий день. Если же сегодня крайний день, то проверка логов будет проходить до момента активации всей программы.

Подробные шаги предоставлены в листинге А.3:

4. log_updater.sh

Входные значения:

$1 - дата нижней парсинга;

$2 - дата верхней границы парсинга;

time – текущее время.

Данный скрипт осуществляет запись логов по отдельным файлам.

Используемые bash команды:

1. touch предназначен для создания пустого файла;

2. sh - Запись скрипта;

3. journalctl - команда терминала Astra Linux предназначенная для прочтения лог файлов, и их уровней приоритета полученных ядром.

Флаг -p для прочтения конкретного типа логов с определенным уровнем приоритета.

7 - все логи; 0 - неработоспособность системы; 1 - alerts; 2 - критическая ошибка; 3 - ошибки; 4 - предупреждения; 5 - уведомления; 6 - информационные сообщения;

4. grep - перехват и запись определенных логов;

5. cat - утилита Linux, позволяющая взаимодействовать с файлами - в данном коде применяется для записи файлов из буффера;

6. sort - предназначена для ортировки значений, необходимых команде uniq;

7. uniq - игнорирует повторяющиеся строки;

8. echo - n > "тип файла" - предназначен для очистки логов из переменной;

9. rm "путь файла" – удаление файла-буффера;

10. startDate+="$startTime" – установка времени начала парсинга;

11. endDate+="$endTime" – установка времени конца парсинга.

Пример использования:

1. touch src/main/java/logFiles/buffer - touch предназначен для создания пустого файла buffer, для записи всех логов в буффер;

2. sh src/main/java/scrypt/clear.sh - #Перед записью логов - необходима очистка файлов, для отслеживания логов в конкретный промежуток времени;

3. journalctl -p 1 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4}\ [0-9]{1,2}\ [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer - Предназначен для записи строк логов в определенный промежуток времени в буффер;

4. cat src/main/java/logFiles/buffer src/main/java/logFiles/emergencyLogs/emergency |sort |uniq -u >> src/main/java/logFiles/alertsLogs/alerts

- Предназачен для записи уникальных строк кода из буффера в ошибки;

5. echo -n > src/main/java/logFiles/buffer - предназачен для очистки буффера для последущих уровней приоритетов лога;

Данный фрагмент кода предназначен для проверки дат. В случае если сегодня не крайний день парсинга, то условие будет выполнено и значение времени будет установлено крайним за прошедшие сутки. Создано это для того, чтобы можно было отследить все операции за предыдущий день. Если же сегодня крайний день, то проверка логов будет проходить до момента активации всей программы.

Подробные шаги предоставлены в листинге А.4:

5. csv_conventor.sh

Входные значения:

$1 – документ откуда идёт обработка данных;

$2 – документ куда записываются данные;

Данный bash-скрипт выполняет обработку данных из файла, указанного в аргументе $1, и записывает результаты в файл, указанный в аргументе $2. Однако важно отметить, что код не содержит обработки ошибок, таких как проверка существования файлов, обработка некорректных данных и других возможных проблем. Рекомендуется добавить соответствующую обработку ошибок для обеспечения надежности и безопасности скрипта.

Используемые bash команды:

1. awk 'BEGIN{print "ID,Month,Day,Time,Type"}' > ${to} - Эта команда использует утилиту awk для создания заголовка в файле, указанном в переменной ${to}, содержащем поля "ID,Month,Day,Time,Type".

2. Этот цикл while выполняет чтение строк из файла, указанного в переменной ${from}. Если строка пустая, то происходит переход к следующей итерации цикла. Листинг 5.1.

```
while read -r p;
  do
    if [[ -z ${p} ]]
    then
      continue
    fi
```

Листинг 5.1

3. echo $ind ${p} | awk '{print $1"," $2"," $3"," $4"," $6}' ((ind++)) - Эта команда выводит строку с индексом $ind и содержимым строки ${p} в утилиту awk, которая затем форматирует вывод и добавляет его

в файл, указанный в переменной ${to}. После этого индекс $ind увеличивается на 1.

4. done < ${from} >> ${to} - Эта строка завершает цикл while и перенаправляет его вывод в файл, указанный в переменной ${to}, с помощью оператора >>, чтобы добавить результаты в конец файла.

Подробные шаги предоставлены в листинге А.5:

6. json_convertor.sh

Входные значения:

$1 – документ откуда идёт обработка данных;

$2 – документ куда записываются данные;

Данный bash-скрипт выполняет обработку данных из файла, создает JSON-объекты и записывает их в файл JSON. Однако, аналогично предыдущему коду, отсутствует обработка ошибок, таких как проверка существования файлов, обработка некорректных данных и других возможных проблем. Рекомендуется добавить соответствующую обработку ошибок для обеспечения надежности и безопасности скрипта.

Используемые bash команды:

1. Основная функция createJsonFromLine() использует утилиту jq для создания JSON-объекта на основе входных данных. Она извлекает данные из входных параметров, формирует JSON-объект с указанными полями и возвращает его. Листинг 6.1

```
function createJsonFromLine() {
pl=${6:0:(${#6} - 1)}
payload=${@:7}

Эта функция createJsonFromLine() создает JSON-объект, используя данные из входных параметров.

jq -n \
--argjson date "$(jq -n \
--arg month "$2" \
--arg day "$3" \
--arg time "$4" \
'{$day, $month, $time}')" \
```

```
 --arg                type                "$pl"                \
 --arg              message             "$payload"             \
--arg                 key                "log-$1"              \
'{($key):          {$date,          $type,          $message}}'
}
```

Листинг 6.1

2. Основная функция main() итерирует через строки во входном файле, вызывает функцию createJsonFromLine() для создания JSON-объектов из строк логов и записывает их в выходной JSON-файл. Листинг 6.2.

```
    function                    main()                    {
ind=1

touch                                                   ${2}

while                read                -r                p;
do
  if              [[              -z              ${p}              ]]
  then
    continue
  fi
  createJsonFromLine                ${ind}                ${p}
  ((ind++))
done      <      ${1}      |      jq      -s      add      >      $2
}
```

Листинг 6.2

Подробные шаги предоставлены в листинге А.6:

7. anlog-installer.sh

Этот Bash-Script предназначен для создания инструмента анализа журналов. Скрипт предназначен для помощи пользователям в анализе и преобразовании файлов журналов в форматы JSON или CSV.

Скрипту требуются следующие входные значения:

"from": файл, из которого будут считываться логи.

"to": файл, в который будут записываться выходные данные.

"тип": тип журнала для фильтрации (например, ошибка, предупреждение и т.д.).

"флаги": Необязательные флаги для фильтрации журналов (

`-h` or `--help`: Отображает подсказки.
`-v` or `--version`: Отображает версию Anlog.
`-a` or `--all`: Фильтрует файлы по общему типу.
`-t` or `--type`: Специфирует тип лог файлов по конкретному типу.
`-n` or `--name`: Определеяет конкретное имя конкретного лог-файла.

Используемые команды Bash:

В скрипте используются следующие команды bash:

Инструкция `if` для условного выполнения

Команда "hash" для проверки наличия команды

Команда "sudo" для повышения привилегий

Команда "apt-get" для установки пакетов

Команда `touch` для создания файлов

Команда "chmod" для изменения прав доступа к файлам

Команда "echo" для печати текста

Команда "jq" для преобразования JSON

Команда "awk" для обработки данных

Команда "getopts" для анализа параметров

Команда `grep` для сопоставления с шаблоном

Команда `journalctl` для поиска системного журнала

Скрипт определяет следующие функции:

`createJsonFromLine`: создает объект JSON из одной строки журнала.

`jsonc`: создает файл JSON из файла журнала с необязательной фильтрацией по типу.

`csvc`: Преобразует журналы в формат CSV.

`getlogs`: Извлекает журналы из системного журнала и сохраняет их в файл.

`func`: функция-оболочка для выполнения пользовательской функции.

2.JAVA

1. HelloApplication.java

Класс 'HelloApplication', который наследуется от класса 'Application' из библиотеки JavaFX. Он используется для создания графического интерфейса пользователя (GUI) в приложении.

Используемые методы:

1. 'getPrimaryStage()': метод возвращает основное окно приложения. Предназначенный для повторной загрузки окна без открытия нового.
2. 'start(Stage stage)': метод, который запускает начальное окно, в котором пользователь выбирает дату. Он загружает FXML-файл для отображения начального окна, устанавливает иконку приложения, заголовок окна, устанавливает сцену в основное окно и отображает его.
3. 'main(String[] args)': метод для запуска приложения.

Подробные шаги предоставлены
в src/main/java/com/example/astrahakaton/HelloApplication.java:

2. HelloController.java

Класс HelloController представляет контроллер для управления пользовательским интерфейсом и обработки событий в приложении.

Используемые методы:

Обработчик события нажатия кнопки "Hello".

Метод выполняет сохранение выбранной конечной даты, запуск процесса записи логов и обновление пользовательского интерфейса.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

@throws InterruptedException если поток исполнения был прерван во время ожидания

Подробные шаги предоставлены

в src/main/java/com/example/astrahakaton/HelloController.java:

3. Logs.java

Класс Logs представляет модель данных для представления журнала логов.

Используемые методы:

Использует традиционные геттеры и сеттеры.

public Logs(String date, String user, String type, String comment) -

Создает новый объект Logs с указанными значениями даты, пользователя, типа и комментария.

@param date значение даты

@param user значение пользователя

@param type тип лога

@param comment комментарий лога

4. LogsController.java

Класс LogsController представляет контроллер для управления представлением журнала логов и обработки событий пользовательского интерфейса.

Используемые методы:

public void getAllLogsView(TableView<Logs> logsTableView) - Отображает таблицу логов в указанном месте представления.

@param logsTableView таблица логов

public void getCharts(BarChart barChart, int i, int i1, int i2, int i3) – Отображает диаграмму в указанном месте представления.

@param barChart диаграмма

@param i координата x

@param i1 координата y

@param i2 ширина

@param i3 высота

public ObservableList<Logs> getDataFromTable() - Получает данные из таблицы логов.

@return данные из таблицы логов

public void setPieData(Map<String, Long> data) - Устанавливает данные для PieChart на основе предоставленной карты данных.

@param data карта данных для PieChart

public static void setTable(ObservableList<Logs> s) - Устанавливает данные для таблицы логов на основе предоставленной ObservableList.

@param s данные для таблицы логов

public ObservableList<Menu> getMenuBar() - Получает элементы меню из MenuBar.

@return элементы меню из MenuBar

public void setMenuBar(Map<String, Long> data) - Устанавливает элементы меню в MenuBar на основе предоставленной карты данных.

@param data карта данных для элементов меню

8.private void resetFilter(int index) - Сбрасывает фильтры и очищает таблицу логов на основе указанного индекса.

@param index индекс

public void setUser(Set<String> name, int q) - Устанавливает фильтр для пользователей и обновляет таблицу логов на основе выбранных пользователей.

@param name множество пользователей

@param q индекс меню

9.protected void onClickBackButton() throws IOException - Обрабатывает событие нажатия кнопки "назад" и осуществляет переход к предыдущему представлению.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

10. Далее идут обработчики событий, при нажатии на одну из кнопок – будет рассмотрен пример одной из, чтобы сократить документацию.

@FXML protected void onClickMenuAlerts() throws IOException - брабатывает выбор пункта меню "alerts" и осуществляет переключение на представление журнала alertsLogs.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

11. private void onClickMenu(String fxml, String path) throws IOException - Обрабатывает выбор элемента меню, осуществляя переключение на соответствующее представление журнала логов.

@param fxml путь к FXML-файлу представления

@param path путь к файлу журнала логов

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

12. protected void onClickUpdate() throws IOException - Обрабатывает нажатие кнопки "обновить" и осуществляет обновление логов с сервера.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

13. protected void onClickAnalysis() throws IOException - Обрабатывает нажатие кнопки "анализ" и осуществляет переход к представлению анализа с созданием графиков для различных типов логов.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

14. protected void onClickJSON() throws IOException - Обрабатывает нажатие кнопки "JSON" и осуществляет переход к представлению для отображения пути.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

15. protected void onClickCSV() throws IOException - Обрабатывает нажатие кнопки "CSV" и осуществляет переход к представлению для отображения пути в формате CSV.

@throws IOException если возникает ошибка ввода-вывода при работе с файлами

16. public static FXMLLoader getCurrentFXMLLoader() –

Получает текущий загрузчик FXML-файла.

@return текущий загрузчик FXML-файла

5.PathController.java

Класс PathController отвечает за обработку пути при процессе конвертации логов в форматы JSON и CSV.

Используемые методы:

1. onClickSelect() - Обрабатывает событие клика по элементу списка "Конвертация". В зависимости от выбранного формата конвертации, вызывает соответствующие методы `onClickJSON()` или `onClickCSV()`.
2. onClickJSON(String path) - Конвертирует логи в формат JSON с помощью скрипта `json_convertor.sh` и сохраняет результат в файле, указанном пользователем.
3. onClickCSV(String path) - Конвертирует логи в формат CSV с помощью скрипта `csv_convertor.sh` и сохраняет результат в файле, указанном пользователем.

6.Util.java

Класс Util - это утилитарный класс, который предназначен для ускорения процесса разработки. Он включает в себя методы для фильтрации журналов, создания буферов и создания графиков.

Используемые методы:

1.selectFilter(String currentPath, ActionEvent e, List<String> listFilter) - Фильтрует журналы в зависимости от выбранных параметров фильтрации. 2. createBuffer(String currentPath, ObservableList< Logs> logs) - Создает буферный файл из предоставленных журналов. 3.getEndDate() - Возвращает дату окончания. 4.createFileForConvertor(String path) - Создает файл для конвертации журналов.

5.saveEndDate(LocalDate endDate) - Устанавливает дату окончания.
6.getTime() - Возвращает время.
7.saveTime(String time) - Устанавливает время.
8.createCharts(String path, FXMLLoader fxmlLoader, int i, int i1, int i2, int i3, String type) - Создает графики на основе предоставленной данных.

ЛИСТИНГ

Листинг А.1

```
#!/bin/bash

startDate=$1

# Назначение входного параметра переменной endDate
endDate=$2
# '5-16-2024 23:49:59'
# if endDate = curDate
# time = cur_time

alsoDate=$(date +'%m-%e-%Y')
# shellcheck disable=SC1073
time=$(date | awk '{print $5}')

if [ "$endDate" -ne "$asloDate" ]
then
  time="23:59:59"
fi

# Прибавляем переменной endDate максимальное время в дне
# для того чтобы охватывать записи по всему крайнему дню
endDate+="$time"

touch src/main/java/logFiles/buffer
```

```
journalctl -p 0 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/emergencyLogs/emergency

 # Запись alerts
journalctl -p 1 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/emergencyLogs/emergency |sort |uniq -u >> src/main/java/logFiles/alertsLogs/alerts
echo -n > src/main/java/logFiles/buffer

 # Запись critical
journalctl -p 2 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/alertsLogs/alerts |sort |uniq -u >> src/main/java/logFiles/criticalLogs/critical
echo -n > src/main/java/logFiles/buffer

 # Запись errors
journalctl -p 3 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/criticalLogs/critical |sort |uniq -u >> src/main/java/logFiles/errorLogs/errors
echo -n > src/main/java/logFiles/buffer

 # Запись warning
journalctl -p 4 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
```

```
cat src/main/java/logFiles/buffer src/main/java/logFiles/errorLogs/errors |sort |uniq -u >> src/main/java/logFiles/warningLogs/warning
echo -n > src/main/java/logFiles/buffer

  # Запись notice
journalctl -p 5 --since=$startDate --until=$endDate |grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/warningLogs/warning |sort |uniq -u >> src/main/java/logFiles/noticeLogs/notice
echo -n > src/main/java/logFiles/buffer

  # Запись info
journalctl -p 6 --since=$startDate --until=$endDate |grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/noticeLogs/notice |sort |uniq -u >> src/main/java/logFiles/infoLogs/info
echo -n > src/main/java/logFiles/buffer

  # Запись debug
journalctl -p 7 --since=$startDate --until=$endDate |grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/infoLogs/info |sort |uniq -u >> src/main/java/logFiles/debugLogs/debug

rm src/main/java/logFiles/buffer
```

Листинг А.2

```
#!/bin/bash


echo -n > src/main/java/logFiles/emergencyLogs/emergency
echo -n > src/main/java/logFiles/alertsLogs/alerts
echo -n > src/main/java/logFiles/criticalLogs/critical
echo -n > src/main/java/logFiles/errorLogs/errors
echo -n > src/main/java/logFiles/warningLogs/warning
echo -n > src/main/java/logFiles/noticeLogs/notice
echo -n > src/main/java/logFiles/infoLogs/info
echo -n > src/main/java/logFiles/debugLogs/debug
echo -n > src/main/java/logFiles/allTypesLogs/all_types
```

Листинг А.3

```
#!/bin/bash

# Назначение входных параметров переменным
# для более удобной работы с параметрами

# Назначение входного параметра переменной startDate
startDate=$1

# Назначение входного параметра переменной endDate
endDate=$2
# '5-16-2024 23:49:59'
#

alsoDate=$(date +'%m-%e-%Y')
# shellcheck disable=SC1073
time=$(date | awk '{print $5}')

if [ "$endDate" -ne "$asloDate" ]
then
  time="23:59:59"
fi

# Прибавляем переменной endDate максимальное время в дне
# для того чтобы охватывать записи по всему крайнему дню
endDate+="$time"

echo $endDate
```

```
# Очищение файлов
sh src/main/java/scrypt/clear.sh

# Запись всех типов
journalctl -p 7 --since=$startDate --until=$endDate | grep -E '^[а-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/allTypesLogs/all_types
```

Листинг А.4

```
#!/bin/bash

# if $1 = curDate
# last string of all_types date
# lastTime -> curTime

startDate=$(date +'%Y-%m-%e')
startTime=$1
endTime=$(date|awk '{print $5}')
endDate=$(date +'%Y-%m-%e')
startDate+="$startTime"
endDate+="$endTime"

echo $startDate
echo $endDate

journalctl -p 7 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}' >> src/main/java/logFiles/allTypesLogs/all_types

touch src/main/java/logFiles/buffer

journalctl -p 0 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/emergencyLogs/emergency

 # Запись alerts
```

```
journalctl -p 1 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/emergencyLogs/emergency |sort |uniq -u >> src/main/java/logFiles/alertsLogs/alerts
echo -n > src/main/java/logFiles/buffer

  # Запись critical
journalctl -p 2 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/alertsLogs/alerts |sort |uniq -u >> src/main/java/logFiles/criticalLogs/critical
echo -n > src/main/java/logFiles/buffer

  # Запись errors
journalctl -p 3 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/criticalLogs/critical |sort |uniq -u >> src/main/java/logFiles/errorLogs/errors
echo -n > src/main/java/logFiles/buffer

  # Запись warning
journalctl -p 4 --since=$startDate --until=$endDate | grep -E '^[a-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/errorLogs/errors |sort |uniq -u >> src/main/java/logFiles/warningLogs/warning
echo -n > src/main/java/logFiles/buffer

  # Запись notice
```

```
journalctl -p 5 --since=$startDate --until=$endDate |grep -E '^[а-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/warningLogs/warning |sort |uniq -u >> src/main/java/logFiles/noticeLogs/notice
echo -n > src/main/java/logFiles/buffer

  # Запись info
journalctl -p 6 --since=$startDate --until=$endDate |grep -E '^[а-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/noticeLogs/notice |sort |uniq -u >> src/main/java/logFiles/infoLogs/info
echo -n > src/main/java/logFiles/buffer

  # Запись debug
journalctl -p 7 --since=$startDate --until=$endDate |grep -E '^[а-я]{3,4} [0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}'> src/main/java/logFiles/buffer
cat src/main/java/logFiles/buffer src/main/java/logFiles/infoLogs/info |sort |uniq -u >> src/main/java/logFiles/debugLogs/debug

rm src/main/java/logFiles/buffer
```

Листинг А.5

```
#!/bin/bash

# TODO: check existing file
function main() {
  from=$1
  to=$2

  awk 'BEGIN{print "ID,Month,Day,Time,Type"}' > ${to}
  ind=1
  while read -r p;
  do
    if [[ -z ${p} ]]
    then
      continue
    fi
    echo $ind ${p} | awk '{print $1"," $2"," $3"," $4"," $6}'
    ((ind++))
  done < ${from} >> ${to}
}

main $1 $2
```

Листинг А.6

```
#!/bin/bash

# TODO: check existing file

# function creating json-object of single log
function createJsonFromLine() {
  pl=${6:0:(${#6} - 1)}
  payload=${@:7}

  jq -n \
  --argjson date "$(jq -n \
                    --arg month "$2" \
                    --arg day "$3" \
                    --arg time "$4" \
                    '{$day, $month, $time}')" \
  --arg type "$pl" \
  --arg message "$payload" \
 --arg key "log-$1" \
 '{($key): {$date, $type, $message}}'
}



# main function (read logs from file and create json file)
# parameter ${1} - from what file get data
# parameter ${2} - where write json
function main() {
  ind=1
```

```
  touch ${2}

  while read -r p;
  do
     if [[ -z ${p} ]]
     then
        continue
     fi
     createJsonFromLine ${ind} ${p}
     ((ind++))
  done < ${1} | jq -s add > $2
}



main $1 $2
```