Отчёт по изучению команды find.

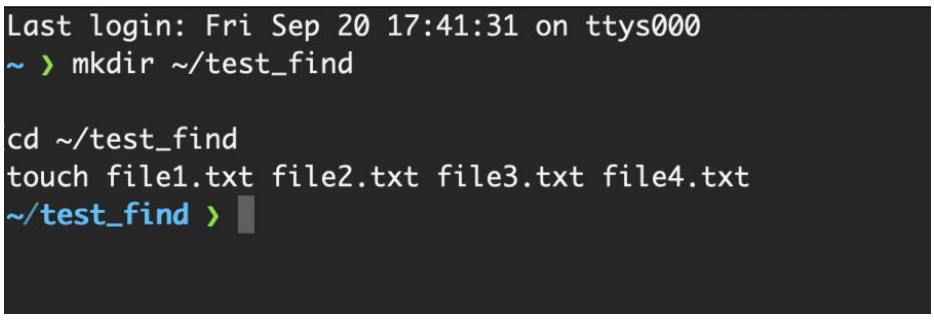
Введение.

В рамках изучения возможностей командной оболочки Linux была поставлена задача ознакомиться с командой find. Данная команда предоставляет мощные инструменты для поиска файлов и каталогов по различным критериям. В данном отчёте описывается процесс создания каталога и файлов, установки времени последнего доступа к файлам, а также использование команды find для поиска и перемещения файлов, не использовавшихся более 14 дней, с исключением определённого каталога из поиска.

Создание каталога и файлов.

Для начала необходимо создать основной каталог, в котором будут находиться файлы для дальнейшей работы. Используется команда mkdir для создания каталога и touch для создания файлов.

```
mkdir ~/test find
cd ~/test find
touch file1.txt file2.txt file3.txt file4.txt
```



В результате выполнения данных команд создаётся каталог test find, содержащий четыре файла: file1.txt, file2.txt, file3.txt, file4.txt

Установка времени последнего доступа с помощью touch**.**

Команда touch позволяет изменять временные метки файлов. Для задания различных времён последнего доступа (atime) к созданным файлам воспользуемся опцией -a и параметром -t указывающим нужную дату и время.

```
# Установка времени последнего доступа более 14 дней назад
touch -a -t 202304010000 file1.txt
touch -a -t 202305010000 file2.txt

# Установка времени последнего доступа менее 14 дней назад
touch -a -t 202409200000 file3.txt
touch -a -t 202409190000 file4.txt
```

В результате файлы file1.txt и file2.txt получают временные метки последнего доступа, которые старше 14 дней, а файлы file3.txt и file4.txt — менее 14 дней.

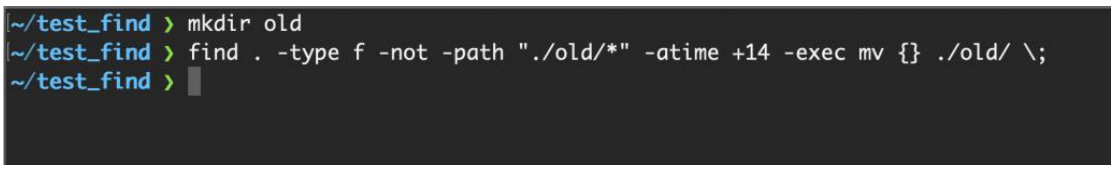
Поиск и перемещение файлов с помощью find**.**

Целью является поиск файлов, к которым не было обращения более 14 дней, и их перемещение в каталог ./old. Для этого сначала создадим каталог old внутри test_find

```
mkdir old
```

Теперь используем команду find для поиска соответствующих файлов и перемещения их в каталог old

```
find . -type f -not -path "./old/*" -atime +14 -exec mv {}
./old/ \;
```



Пояснение команд

find .— начинает поиск в текущем каталоге.

-type f— ищет только файлы.

-not -path "./old/*"— исключает из поиска все файлы внутри каталога ./old

-atime +14 — находит файлы, к которым не обращались более 14 дней.

-exec mv {} ./old/ \; — выполняет команду перемещения найденных файлов в каталог old

Исключение каталога ./old из поиска

Важно исключить каталог ./old из поиска, чтобы избежать бесконечной рекурсии и перемещения уже перенесенных файлов. Это достигается с помощью параметра -not -path "./old/*" который исключает из результатов поиска все файлы, находящиеся внутри каталога old.

Проверка результатов

После выполнения команды find можно проверить, какие файлы были перемещены в каталог old.

ls old

Должны быть видны файлы file1.txt и file2.txt , так как их время последнего доступа старше 14 дней. Файлы file3.txt и file4.txt останутся в основном каталоге.

Заключение

В ходе выполнения задания была изучена команда find и её возможности по поиску файлов по различным критериям. Были созданы каталог и файлы, установлено различное время последнего доступа к ним с помощью команды touch. Командой find удалось эффективно найти файлы, к которым не обращались более 14 дней, и переместить их в указанный каталог, при этом исключив из поиска сам каталог ./old . Данный процесс демонстрирует практическое применение команды find для управления файлами на основе их временных меток.

Отчет по теме "Изучение сценариев bash"

Задание 1: Скрипт ptxt с использованием case и shift

Описание

Скрипт ptxt выводит заданный текст определённое количество раз с заданным интервалом времени. Параметры передаются с помощью опций -n (количество выводов) и -t (таймаут в секундах), после которых следует текст для вывода.

Код скрипта:

Создайте файл ptxt_case_shift.sh и добавьте в него следующий код:

```
#!/bin/bash
# Проверка, что хотя бы один аргумент передан
if [ $# -lt 5 ]; then
        echo "Использование: $0 -n <число выводов> -t
<таймаут> -- <текст для вывода>"
        exit 1
fi
# Инициализация переменных
count=0
timeout=0
text=" "
# Обработка параметров с использованием case и shift
while [ -n "$1" ]; do
        case "$1" in
                -n)
                        shift
                        if [[ "$1" =~ ^[0-9]+$ ]]; then
```

```
                count=$1
            else
                echo "Ошибка: -n требует числового
значения."
                exit 1
            fi
            shift
            ;;
        -t)
            shift
            if [[ "$1" =~ ^[0-9]+$ ]]; then
                timeout=$1
            else
                echo "Ошибка: -t требует числового
значения."
                exit 1
            fi
            shift
            ;;
        --)
            shift
            text="$@"
            break
            ;;
        *)
            echo "Неверный аргумент: $1"
            echo "Использование: $0 -n <число выводов> -t
```

```
<таймаут> -- <текст для вывода>"
                        exit 1
                        ;;
        esac
done
# Проверка обязательных параметров
if [ "$count" -le 0 ] || [ "$timeout" -le 0 ] || [ -z  "$text" ]; then
        echo "Ошибка: Недостаточно аргументов."
        echo "Использование: $0 -n <число выводов> -t
<таймаут> -- <текст для вывода>"
        exit 1
fi
# Вывод текста заданное количество раз с интервалом
for ((i=1; i<=count; i++)); do
        echo "$text"
        if [ "$i" -lt "$count" ]; then
                sleep "$timeout"
        fi
done
```

Пояснения.

Проверка аргументов: Скрипт сначала проверяет, что передано достаточное количество аргументов.

Обработка параметров: Используется case для обработки опций -n , -t и -- . После каждой опции shift сдвигает позицию аргументов.

Валидация: Проверяется, что значения для -n и -t являются числами.

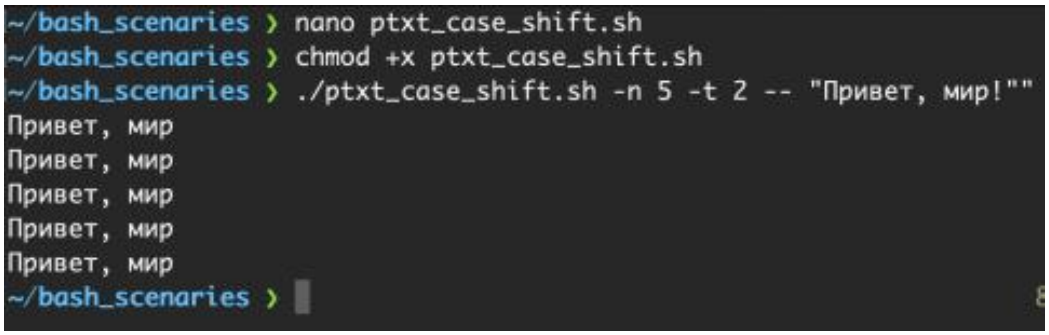
Вывод текста: Используется цикл for для вывода текста с заданным интервалом.

Пример использования

chmod +x ptxt_case_shift.sh

./ptxt_case_shift.sh -n 5 -t 2 -- "Привет, мир!""

Этот пример выведет строку "Привет, мир!" 5 раз с интервалом в 2 секунды между выводами.



Задание 2: Скрипт ptxt с использованием getopts.

Описание.

Аналогичный скрипт ptxt , но обработка параметров выполняется с помощью встроенной утилиты getopts , которая упрощает разбор опций командной строки.

Код скрипта:

Создайте файл ptxt_getopts.sh и добавьте в него следующий код:

```
#!/bin/bash
# Инициализация переменных
count=0
timeout=0
text=""
# Обработка опций с помощью getopts
while getopts "n:t:" opt; do
        case "$opt" in
                n)
                        if [[ "$OPTARG" =~ ^[0-9]+$ ]]; then
                                count=$OPTARG
                        else
                                echo "Ошибка: Опция -n требует числового
значения."
                                exit 1
                        fi
                        ;;
                t)
```

```
                if [[ "$OPTARG" =~ ^[0-9]+$ ]]; then
                        timeout=$OPTARG
                else
                        echo "Ошибка: Опция -t требует числового
значения."
                        exit 1
                fi
                ;;
        \?)
                echo "Неверная опция: -$OPTARG" >&2
                echo "Использование: $0 -n <число выводов> -t
<таймаут> -- <текст для вывода>"
                exit 1
                ;;
        :)
                echo "Опция -$OPTARG требует аргумента." >&2
                echo "Использование: $0 -n <число выводов> -t
<таймаут> -- <текст для вывода>"
                exit 1
                ;;
    esac
done
# Сдвиг обработанных опций
shift $((OPTIND -1))

# Остальные аргументы после --
if [ "$#" -lt 1 ]; then
```

```
        echo "Ошибка: Не указан текст для вывода."
        echo "Использование: $0 -n <число выводов> -t
<таймаут> -- <текст для вывода>"
        exit 1
fi
text="$*"
# Проверка обязательных параметров
if [ "$count" -le 0 ] || [ "$timeout" -le 0 ]; then
        echo "Ошибка: Параметры -n и -t должны быть
положительными числами."
        echo "Использование: $0 -n <число выводов> -t
<таймаут> -- <текст для вывода>"
        exit 1
fi
# Вывод текста заданное количество раз с интервалом
for ((i=1; i<=count; i++)); do
        echo "$text"
        if [ "$i" -lt "$count" ]; then
                sleep "$timeout"
        fi
done
```

Пояснения .

Использование getopts : getopts упрощает разбор опций -n и -t , автоматически обрабатывая ошибки и аргументы.

Валидация: Проверяется, что значения для опций являются числами.

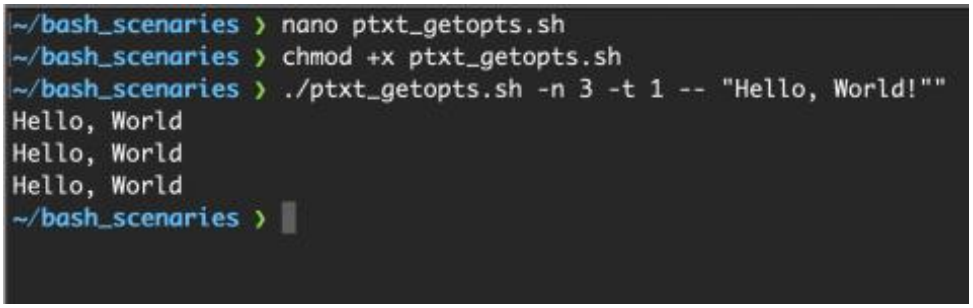
Сдвиг аргументов: После обработки опций выполняется shift $((OPTIND -1)) , чтобы удалить обработанные опции из списка аргументов.

Вывод текста: Аналогично первому скрипту.

Пример использования

```
chmod +x ptxt_getopts.sh
./ptxt_getopts.sh -n 3 -t 1 -- "Hello, World!""
```

Этот пример выведет строку "Hello, World!" 3 раза с интервалом в 1 секунду между выводами.



Задание 3: Скрипт sort_args.sh **для сортировки аргументов.**

Описание.

Скрипт sort_args.sh принимает произвольное количество аргументов, сортирует их в алфавитном порядке с использованием массива и сортировки методом пузырька, и выводит отсортированный список. При этом не используется встроенная команда sort .

Код скрипта .

Создайте файл sort_args.sh и добавьте в него следующий код:

```
#!/bin/bash
# Проверка, что переданы аргументы
if [ $# -eq 0 ]; then
    echo "Использование: $0 <аргументы для сортировки>"
    exit 1
fi
# Инициализация массива с аргументами
arr=("$@")
n=${#arr[@]}
# Сортировка методом пузырька
for ((i = 0; i < n-1; i++)); do
    for ((j = 0; j < n-i-1; j++)); do
        if [[ "${arr[j]}" > "${arr[j+1]}" ]]; then
            # Обмен элементов
            temp="${arr[j]}"
            arr[j]="${arr[j+1]}"
            arr[j+1]="$temp"
        fi
    done
```

```
done
# Вывод отсортированных аргументов
echo "Отсортированные аргументы:"
for item in "${arr[@]}"; do
        echo "$item"
Done
```

Пояснения .

Инициализация массива: Все переданные аргументы сохраняются в массив arr .

Метод пузырька: Два вложенных цикла проходят по массиву, сравнивая и обменивая элементы, если они находятся в неправильном порядке.

Сравнение строк: Используется условие [["${arr[j]}" > "${arr[j+1]}"]] для сравнения строк в алфавитном порядке.

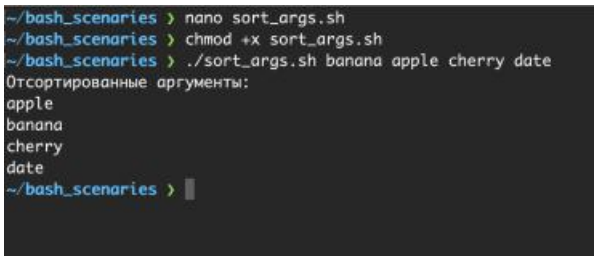
Вывод результата: После сортировки выводятся отсортированные элементы массива.

Пример использования .

```
chmod +x sort_args.sh
./sort_args.sh banana apple cherry date
```

Вывод:

```
Отсортированные аргументы:
apple
banana
cherry
date
```



Заключение

Все три скрипта продемонстрировали различные методы обработки аргументов командной строки и выполнения задач в bash. Скрипты ptxt_case_shift.sh и ptxt_getopts.sh показывают два подхода к разбору опций, тогда как sort_args.sh иллюстрирует работу с массивами и реализацию простого алгоритма сортировки. Вы можете использовать и модифицировать эти скрипты в соответствии с вашими потребностями.

Отчёт по теме "Изучение сценариев bash 2.0"

Цель работы:

1. Изучить основы работы с командной оболочкой Bash, научиться работать с переменными, циклами, функциями, параметрами командной строки, операторами условия и командами.
2. Написать сценарии (скрипты) для решения двух задач:
   a) Построчный вывод всех путей из переменной окружения PATH .
   b) Построчный вывод имён файлов из указанного каталога и его подкаталогов с указанием уровня вложенности.

Описание решения задач:

1. **Сценарий для построчного вывода всех путей из переменной PATH**:

Переменная PATH содержит список директорий, в которых система ищет исполняемые файлы. Каждый путь в переменной PATH разделен символом двоеточия : . Для того чтобы вывести все пути построчно, использован цикл for in , который последовательно обрабатывает каждый путь.

Сценарий:

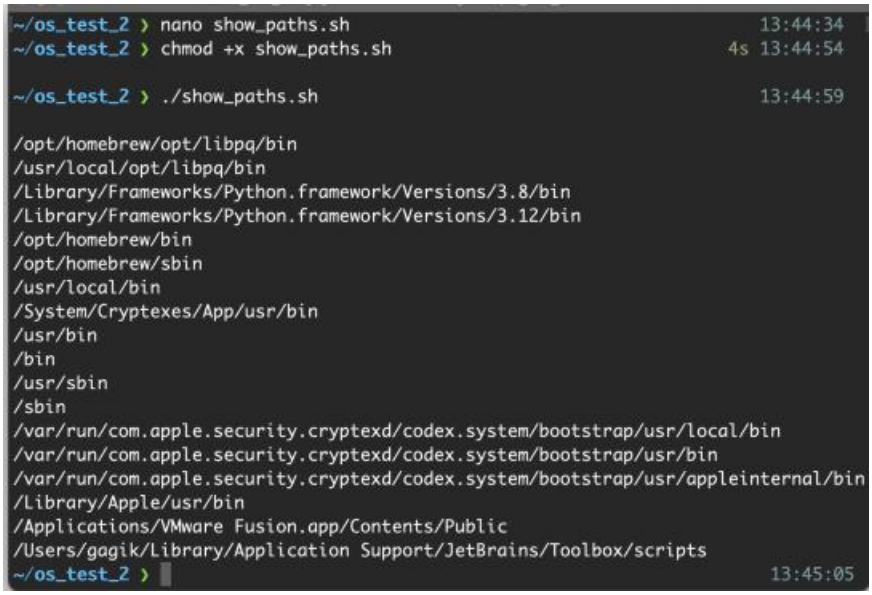
```
#!/bin/bash
# Получение списка директорий из PATH и их построчный
вывод
IFS=':' # Устанавливаем разделитель полей как ':'
for dir in $PATH; do
        echo $dir
done
```

Описание работы сценария:

- Переменной IFS задается значение : для того, чтобы цикл for корректно разбил строку по каждому пути в PATH .
- Используя цикл for in , построчно выводим каждый элемент (директорию) из переменной PATH .

Вывод:

Сценарий корректно выводит все пути из переменной окружения PATH построчно.



2. **Сценарий для вывода имён файлов из указанного каталога с учётом уровня вложенности**:

В этой задаче необходимо построчно выводить имена файлов из указанного каталога и всех его подкаталогов. При этом перед именем файла нужно указывать уровень вложенности подкаталога, начиная с 1. Для решения задачи была использована рекурсия и цикл for .

Сценарий:

```
#!/bin/bash
# Функция для рекурсивного обхода директорий и вывода файлов с указанием уровня вложенности
list_files() {
    local directory="$1" # Каталог для обработки
    local level="$2" # Уровень вложенности
    for file in "$directory"/*; do
        if [ -d "$file" ]; then
            # Если это директория, рекурсивно обрабатываем её
            echo "$level) $file/"
            list_files "$file" $(expr $level + 1)
        else
            # Если это файл, выводим его с указанием уровня вложенности
            echo "$level) $file"
        fi
    done
}
```

```
# Вызов функции с передачей аргументов: директория и начальный уровень
list_files "$1" 1
```

Описание работы сценария:

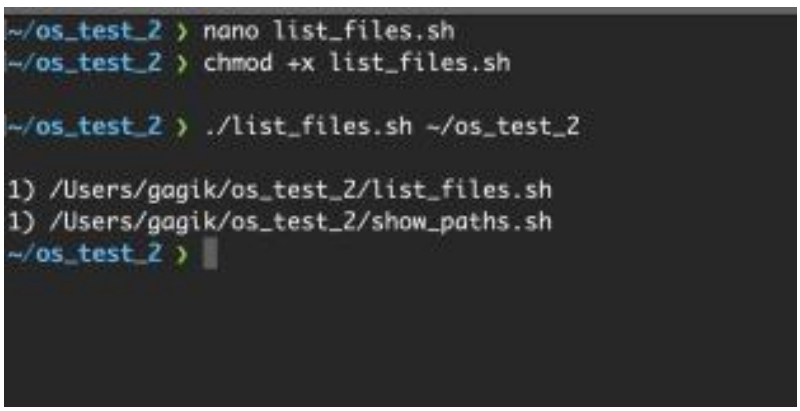
- Определена функция list_files , которая принимает два параметра: путь к директории и текущий уровень вложенности.
- С помощью цикла for in происходит обход всех файлов и директорий в переданной директории.
- Если элемент является директорией (условие if [-d "$file"]), вызывается рекурсивно функция list_files для обработки этой поддиректории, при этом уровень вложенности увеличивается.
- Если элемент является файлом, он выводится на экран с указанием текущего уровня вложенности.

Пример использования:

```
./script.sh /path/to/directory
```

Вывод:

Сценарий корректно отображает имена файлов и директорий с указанием уровня вложенности



Выводы:

В ходе выполнения задания были изучены и применены следующие конструкции и операторы командной оболочки Bash:

- Переменные и подстановка их значений.
- Циклы for in для перебора элементов.
- Условные операторы if .
- Функции с рекурсивным вызовом для решения задачи обхода каталогов.
- Работа с переменной окружения PATH и разбор путей с использованием переменной IFS .

Данный опыт углубил знания в работе с Bash и позволил научиться создавать более сложные сценарии с использованием рекурсии и циклов.