Рафи Кази ар

НКАбд-03-24

1032238132@pfur.ru

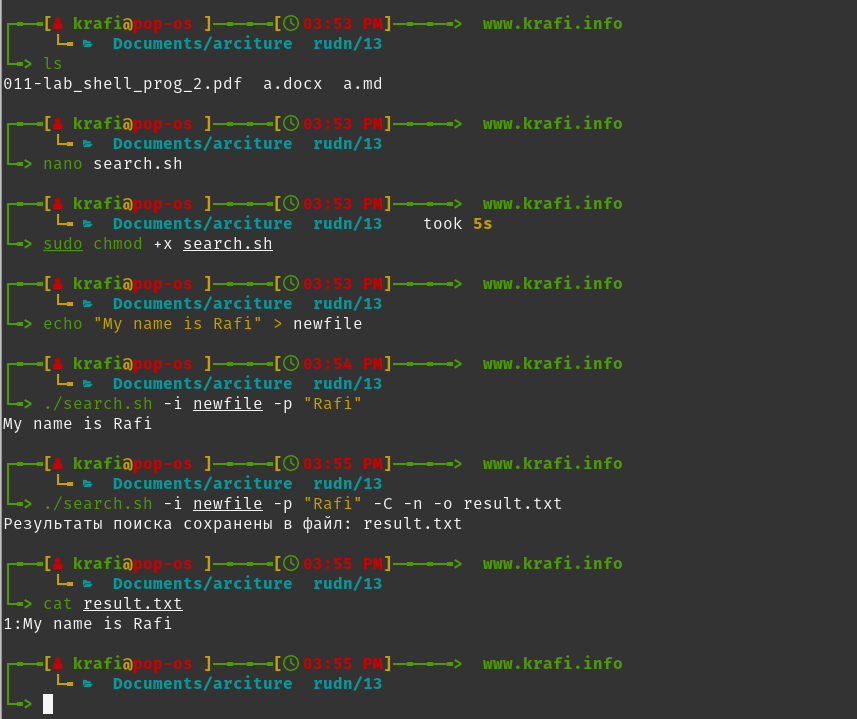
#### **Задание 1: Командный файл с getopts и grep**

Это задание учит создавать скрипты с обработкой ключей командной строки, как это делают стандартные утилиты.

**Листинг скрипта (search.sh):**

#!/bin/bash  
  
# Значения по умолчанию  
inputfile=""  
outputfile=""  
pattern=""  
case\_sensitive=0  
line\_numbers=0  
  
# Обработка ключей с помощью getopts  
# Опция 'C' и 'n' не требуют аргумента, поэтому после них нет двоеточия  
# Для длинных опций мы используем флаги. Стандартный getopts не поддерживает  
# длинные опции (--inputfile), поэтому эмулируем их через короткие (-i, -o, -p).  
while getopts "i:o:p:Cn" opt; do  
 case $opt in  
 i) inputfile="$OPTARG" ;; # Ключ -i соответствует --inputfile  
 o) outputfile="$OPTARG" ;; # Ключ -o соответствует --outputfile  
 p) pattern="$OPTARG" ;; # Ключ -p соответствует --pattern  
 C) case\_sensitive=1 ;; # Включить чувствительность к регистру  
 n) line\_numbers=1 ;; # Включить вывод номеров строк  
 \?) echo "Неверный ключ: -$OPTARG" >&2  
 exit 1 ;;  
 esac  
done  
  
# Проверка на обязательные параметры  
if [[ -z "$inputfile" || -z "$pattern" ]]; then  
 echo "Использование: $0 -i inputfile -p pattern [-o outputfile] [-C] [-n]"  
 exit 1  
fi  
  
# Проверка существования входного файла  
if [[ ! -f "$inputfile" ]]; then  
 echo "Файл '$inputfile' не найден."  
 exit 1  
fi  
  
# Построение команды grep  
grep\_cmd="grep"  
if [[ $case\_sensitive -eq 1 ]]; then  
 grep\_cmd="$grep\_cmd" # По умолчанию grep чувствителен к регистру, ключ -i отключает это.  
 # В задании сказано: -C — различать большие и малые буквы.  
 # Это поведение grep по умолчанию, поэтому ничего делать не нужно.  
else  
 grep\_cmd="$grep\_cmd -i" # Ключ -i: игнорировать регистр  
fi  
if [[ $line\_numbers -eq 1 ]]; then  
 grep\_cmd="$grep\_cmd -n"  
fi  
  
# Выполнение поиска  
# Если указан выходной файл, перенаправляем вывод в него, иначе выводим на экран  
if [[ -n "$outputfile" ]]; then  
 $grep\_cmd "$pattern" "$inputfile" > "$outputfile"  
 echo "Результаты поиска сохранены в файл: $outputfile"  
else  
 $grep\_cmd "$pattern" "$inputfile"  
fi

**Пример использования:**

# Поиск без учета регистра (по умолчанию), вывод на экран  
./search.sh -i myfile.txt -p "hello world"  
  
# Поиск с учетом регистра (-C), с номерами строк (-n) и сохранением в файл  
./search.sh -i myfile.txt -p "Hello World" -C -n -o result.txt

#### **Задание 2: Программа на Си и анализ кода возврата**

Это задание демонстрирует, как оболочка и программы взаимодействуют через коды завершения.

**Листинг программы на Си (check\_number.c):**

#include <stdio.h>  
#include <stdlib.h> // Для функции exit()  
  
int main() {  
 int number;  
 printf("Введите число: ");  
 scanf("%d", &number);  
  
 if (number > 0) {  
 printf("Число положительное.\n");  
 exit(1); // Код возврата 1 для положительного числа  
 } else if (number < 0) {  
 printf("Число отрицательное.\n");  
 exit(2); // Код возврата 2 для отрицательного числа  
 } else {  
 printf("Число равно нулю.\n");  
 exit(0); // Код возврата 0 обычно означает успешное завершение  
 }  
}

**Компиляция:**

gcc -o check\_number check\_number.c

**Листинг командного файла (number\_script.sh):**

#!/bin/bash  
  
# Запускаем скомпилированную программу  
./check\_number  
  
# Анализируем код возврата последней команды через специальную переменную $?  
case $? in  
 0)  
 echo "Было введено число 0." ;;  
 1)  
 echo "Было введено положительное число." ;;  
 2)  
 echo "Было введено отрицательное число." ;;  
 \*)  
 echo "Программа завершилась с неизвестным кодом." ;;  
Esac



#### **Задание 3: Создание и удаление файлов**

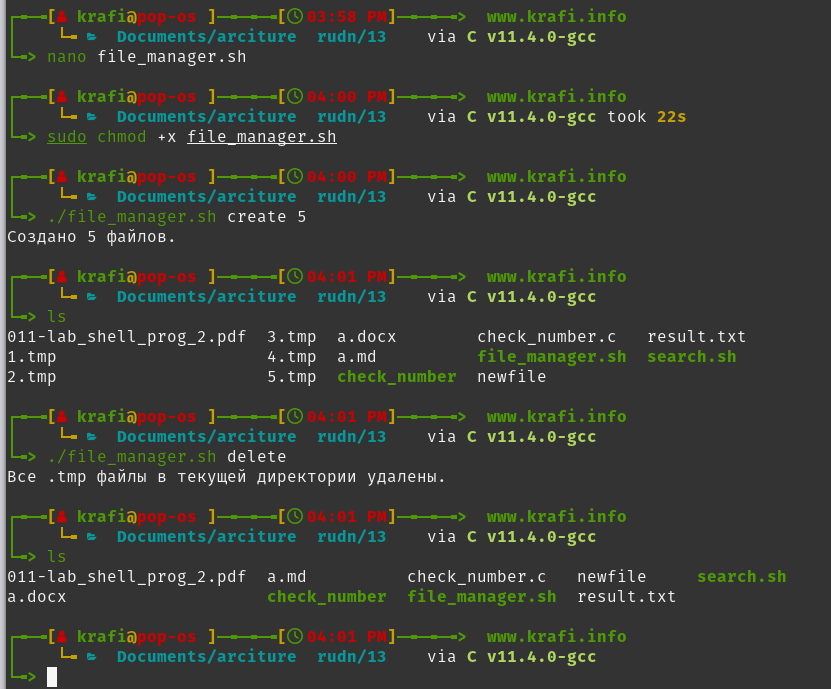
Это задание учит работе с аргументами командной строки и циклами.

**Листинг скрипта (file\_manager.sh):**

#!/bin/bash  
  
# Функция для создания файлов  
create\_files() {  
 local num\_files=$1  
 for ((i=1; i<=num\_files; i++)); do  
 touch "${i}.tmp"  
 done  
 echo "Создано $num\_files файлов."  
}  
  
# Функция для удаления файлов  
delete\_files() {  
 # Удаляем все файлы, подходящие под шаблон '\*.tmp'  
 # Использование команды rm с шаблоном может быть опасным,  
 # но в рамках этого задания допустимо.  
 rm -f \*.tmp  
 echo "Все .tmp файлы в текущей директории удалены."  
}  
  
# Анализ аргументов командной строки  
if [[ $# -eq 0 ]]; then  
 echo "Использование:"  
 echo " $0 create <N> - создать N файлов"  
 echo " $0 delete - удалить все .tmp файлы"  
 exit 1  
fi  
  
case $1 in  
 create)  
 if [[ $# -ne 2 ]]; then  
 echo "Укажите число файлов для создания."  
 exit 1  
 fi  
 # Проверяем, что второй аргумент - число  
 if [[ ! $2 =~ ^[0-9]+$ ]]; then  
 echo "Второй аргумент должен быть целым числом."  
 exit 1  
 fi  
 create\_files "$2"  
 ;;  
 delete)  
 delete\_files  
 ;;  
 \*)  
 echo "Неизвестная команда: $1"  
 exit 1  
 ;;  
esac

**Пример использования:**

./file\_manager.sh create 5 # Создаст файлы 1.tmp, 2.tmp, 3.tmp, 4.tmp, 5.tmp  
./file\_manager.sh delete # Удалит все .tmp файлы



#### **Задание 4: Архивация недавно измененных файлов**

Это задание учит комбинировать команды (find, tar) для решения практических задач.

**Листинг скрипта (archive\_recent.sh):**

#!/bin/bash  
  
# Проверяем, указана ли директория  
if [[ $# -eq 0 ]]; then  
 echo "Использование: $0 <директория>"  
 exit 1  
fi  
  
target\_dir="$1"  
archive\_name="backup\_$(date +%Y-%m-%d).tar.gz"  
  
# Проверяем существование директории  
if [[ ! -d "$target\_dir" ]]; then  
 echo "Ошибка: Директория '$target\_dir' не найдена."  
 exit 1  
fi  
  
# Переходим в целевую директорию, чтобы пути в архиве были относительными  
cd "$target\_dir" || exit  
  
# Находим все файлы, измененные за последние 7 дней, и архивируем их.  
# Ключ -print0 и -0 для tar позволяют работать с файлами, содержащими пробелы.  
find . -type f -mtime -7 -print0 | tar -czf "../$archive\_name" --null -T -  
  
echo "Архив создан: $(pwd)/../$archive\_name"

**Пример использования:**

./archive\_recent.sh /path/to/your/directory



### **Ответы на контрольные вопросы**

1. **Каково предназначение команды getopts?** getopts — это встроенная команда оболочки (shell) для синтаксического разбора (парсинга) аргументов и ключей командной строки. Она упрощает обработку флагов (например, -a, -f filename), обеспечивает стандартное поведение и избегает ошибок при ручном разборе через $1, $2, shift.
2. **Какое отношение метасимволы имеют к генерации имён файлов?** Метасимволы (или шаблоны, wildcards) \*, ?, [ ] используются для генерации списка имен файлов, которые соответствуют заданному шаблону. Этот процесс называется «глоббинг» (globbing). Например, команда rm \*.tmp сначала генерирует список всех файлов, оканчивающихся на .tmp, а затем передает этот список команде rm для удаления.
3. **Какие операторы управления действиями вы знаете?**
   * **Условные операторы:** if-then-elif-else-fi, case-in-esac
   * **Циклы:** for-in-do-done, while-do-done, until-do-done
   * **Логические операторы для управления выполнением команд:**
     + && (И): Выполняет следующую команду, только если предыдущая завершилась успешно (код возврата 0). cmd1 && cmd2
     + || (ИЛИ): Выполняет следующую команду, только если предыдущая завершилась неудачно (код возврата не 0). cmd1 || cmd2
4. **Какие операторы используются для прерывания цикла?**
   * break — немедленно прерывает выполнение цикла и передает управление командам после done.
   * break N — прерывает выполнение N вложенных циклов.
   * continue — прерывает текущую итерацию цикла и переходит к следующей.
   * exit — завершает весь скрипт полностью.
5. **Для чего нужны команды false и true?** Это команды, которые всегда возвращают определенный код завершения.
   * true — всегда возвращает код успеха (0). Часто используется для создания бесконечных циклов (while true; do ...).
   * false — всегда возвращает код ошибки (не 0). Может использоваться для принудительного перехода к ветке || или для заглушки.
6. **Что означает строка if test -f man$s/$i.$s?** Это проверка условия с помощью команды test (синоним [ ]).
   * -f — проверяет, существует ли файл и является ли он обычным файлом (regular file), а не каталогом или устройством.
   * man$s/$i.$s — это путь к файлу. Здесь $s и $i — переменные. Например, если $s=1 и $i=printf, то проверяется существование файла man1/printf.1.
   * Эквивалентная запись в современном синтаксисе: if [[ -f "man${s}/${i}.${s}" ]]
7. **Объясните различия между конструкциями while и until.**
   * **while** — выполняет блок команд **пока** условие (код завершения последней команды в условии) истинно (равно 0). while condition; do commands; done
   * **until** — выполняет блок команд **до тех пор, пока** условие не станет истинным (т.е. пока условие ложно — возвращает не 0). until condition; do commands; done **Проще говоря:** while работает "пока есть успех", а until работает "пока есть неудача".