

Отчёт по лабораторной работе №9

Программирование в командном процессоре ОС UNIX. Командные файлы

Куликов Александр Андреевич

Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл - аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

Теоретическое введение

- Командный процессор (командная оболочка, интерпретатор команд shell) - это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:
 - оболочка Борна (Bourne shell или sh) - стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;

- С-оболочка (или `csch`) - надстройка на оболочкой Борна, использующая С-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или `ksh`) - напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH - сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).
- POSIX (Portable Operating System Interface for Computer Environments) - набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

Выполнение лабораторной работы

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию `backup` в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор `zip`, `bzip2` или `tar`. Способ использования команд архивации необходимо узнать, изучив справку (см. рис. @fig:001, @fig:002, @fig:003).

```
oskulikov@oskulikov-VirtualBox:~$ nano backup_script.sh
oskulikov@oskulikov-VirtualBox:~$ chmod +x backup_script.sh
oskulikov@oskulikov-VirtualBox:~$ ls
abc1          dle.old      mounthly     snap         Загрузки     Шабло
australia     feathers    my_os        tutorial     Изображения
backup_script.sh file        play         work         Музыка
conf.txt      file.txt    reports      Видео        Общедоступные
ddd           may         ski.plases   Документы    'Рабочий стол'
oskulikov@oskulikov-VirtualBox:~$ ./backup_script.sh
backup_script.sh
Резервная копия создана: /home/oskulikov/backup/backup_script.sh.tar.gz
oskulikov@oskulikov-VirtualBox:~$
```

```
GNU nano 7.2                                backup_script.sh
#!/bin/bash

# Путь к текущему скрипту
SCRIPT_PATH=$(realpath "$0")

# Директория для резервных копий
BACKUP_DIR="$HOME/backup"

# Создание директории для резервных копий, если она не существует
mkdir -p "$BACKUP_DIR"

# Имя файла без пути
SCRIPT_NAME=$(basename "$SCRIPT_PATH")

# Имя и путь резервного архива
BACKUP_FILE="$BACKUP_DIR/${SCRIPT_NAME}.tar.gz"

# Создание архива с резервной копией
tar -czvf "$BACKUP_FILE" -C "$(dirname "$SCRIPT_PATH")" "$SCRIPT_NAME"

[ Прочитана 21 строка ]
^G Справка      ^O Записать     ^W Поиск       ^K Вырезать    ^T Выполнить   ^C Позиция
^X Выход        ^R ЧитФайл    ^\ Замена     ^U Вставить   ^J Выровнять   ^/_ К строке
```

- Результат
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов (см. рис. @fig:004, @fig:005).

```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 print_args.sh
#!/bin/bash

# Проверяем, переданы ли аргументы
if [ $# -eq 0 ]; then
    echo "Не передано ни одного аргумента."
    exit 1
fi

# Цикл по всем переданным аргументам
for arg in "$@"; then
    echo "Аргумент: $arg"
done

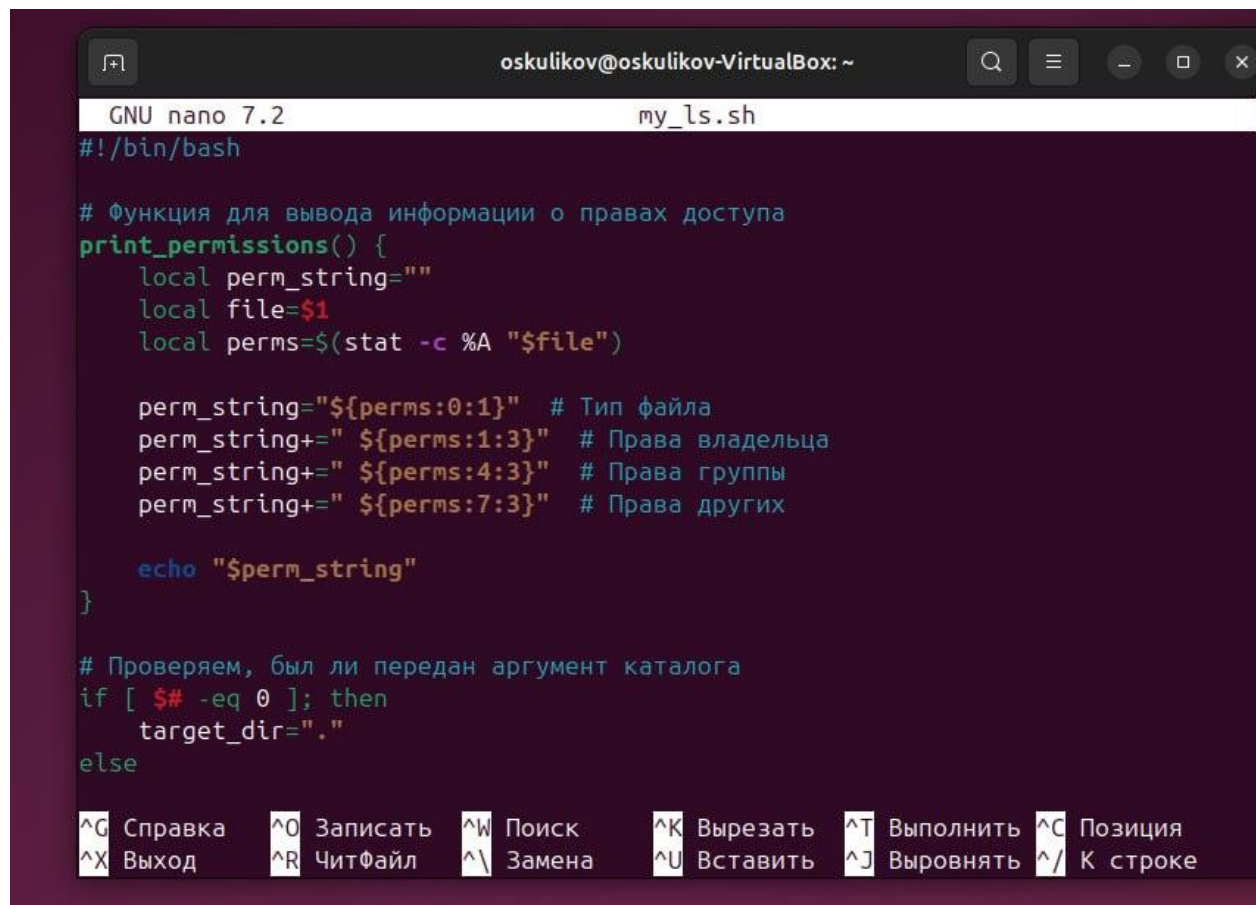
[ Прочитано 12 строк ]
^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/_ К строке

oskulikov@oskulikov-VirtualBox:~$ nano print_args.sh
oskulikov@oskulikov-VirtualBox:~$ chmod +x print_args.sh
oskulikov@oskulikov-VirtualBox:~$ ./print_args.sh arg1 arg2 arg3 arg4 arg5 arg6
arg7 arg8 arg9 arg10 arg11
Аргумент: arg1
Аргумент: arg2
Аргумент: arg3
Аргумент: arg4
Аргумент: arg5
Аргумент: arg6
Аргумент: arg7
Аргумент: arg8
Аргумент: arg9
Аргумент: arg10
Аргумент: arg11
oskulikov@oskulikov-VirtualBox:~$ ./print_args.sh 1 2 3 4 5
Аргумент: 1
Аргумент: 2
Аргумент: 3
Аргумент: 4
Аргумент: 5
oskulikov@oskulikov-VirtualBox:~$
```

-
- Результат

3. Написать командный файл - аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога (см. рис. @fig:006, @fig:007).

```
oskulikov@oskulikov-VirtualBox:~$ ./my_ls.sh
- rw- rw- r-- 0 2024-06-09 21:37:15.822457864 +0300 abc1
d rwx rw- r-- 4096 2024-06-09 21:40:30.131583831 +0300 australia
d rwx rwx r-x 4096 2024-06-12 20:39:48.762861131 +0300 backup
- rwx rwx r-x 705 2024-06-12 20:39:29.549780182 +0300 backup_script.sh
- rw- rw- r-- 451 2024-06-09 21:55:59.164336387 +0300 conf.txt
d rwx rwx r-x 4096 2024-06-09 22:43:22.247322951 +0300 ddd
- rw- rw- r-- 0 2024-06-09 21:46:44.933117862 +0300 dile.old
- rw- rw- r-- 0 2024-06-09 21:44:39.246203265 +0300 feathers
- rw- rw- r-- 2359 2024-06-09 21:52:47.958804074 +0300 file
- rw- rw- r-- 2352 2024-06-09 21:53:33.497945091 +0300 file.txt
- rw- rw- r-- 0 2024-06-09 21:27:55.807875349 +0300 may
d rwx rwx r-x 4096 2024-06-09 21:19:18.358558614 +0300 mounthly
- rwx rwx r-x 1168 2024-06-12 20:48:06.122781423 +0300 my_ls.sh
- r-x rw- r-- 0 2024-06-09 21:43:23.422534569 +0300 my_os
d rwx -wx --x 4096 2024-06-09 21:49:04.726907241 +0300 play
- rwx rwx r-x 299 2024-06-12 20:46:04.518183365 +0300 print_args.sh
d rwx rwx r-x 4096 2024-06-09 21:27:25.165061252 +0300 reports
d rwx rwx r-x 4096 2024-06-09 21:40:00.290176846 +0300 ski.plases
d rwx --- --- 4096 2024-06-09 21:00:35.688235626 +0300 snap
d rwx rwx r-x 4096 2024-06-09 20:36:40.456971955 +0300 tutorial
d rwx rwx r-x 4096 2024-06-09 22:50:46.978862602 +0300 work
```

```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 my_ls.sh
#!/bin/bash

# Функция для вывода информации о правах доступа
print_permissions() {
    local perm_string=""
    local file=$1
    local perms=$(stat -c %A "$file")

    perm_string="${perms:0:1}" # Тип файла
    perm_string+=" ${perms:1:3}" # Права владельца
    perm_string+=" ${perms:4:3}" # Права группы
    perm_string+=" ${perms:7:3}" # Права других

    echo "$perm_string"
}

# Проверяем, был ли передан аргумент каталога
if [ $# -eq 0 ]; then
    target_dir="."
else
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выводить ^/_ К строке

- - Результат
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (см. рис. @fig:008, @fig:009).

```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 count_files.sh *
#!/bin/bash

# Проверка наличия двух аргументов
if [ $# -ne 2 ]; then
    echo "Usage: $0 <file_extension> <directory>"
    exit 1
fi

file_extension=$1
directory=$2

# Проверка существования директории
if [ ! -d "$directory" ]; then
    echo "Ошибка: Директория '$directory' не существует."
    exit 1
fi

# Подсчет количества файлов с указанным расширением
file_count=$(find "$directory" -type f -name "$file_extension" | wc -l)

^G Справка      ^O Записать     ^W Поиск        ^K Вырезать     ^T Выполнить    ^C Позиция
^X Выход         ^R ЧитФайл     ^\ Замена       ^U Вставить     ^J Выводить     ^/ К строке
```

```
oskulikov@oskulikov-VirtualBox:~$ sudo ./count_files.sh .txt /
[sudo] пароль для oskulikov:
find: '/run/user/1000/doc': Отказано в доступе
find: '/run/user/1000/gvfs': Отказано в доступе
Количество файлов с расширением '.txt' в директории '/': 1192
oskulikov@oskulikov-VirtualBox:~$
```

```
find: '/etc/credstore.encrypted': Отказано в доступе
find: '/etc/cups/ssl': Отказано в доступе
Количество файлов с расширением '.txt .doc .jpg .pdf' в директории '/': 0
oskulikov@oskulikov-VirtualBox:~$ sudo ./count_files.sh ".txt .doc .jpg .pdf" /
find: '/run/user/1000/doc': Отказано в доступе
find: '/run/user/1000/gvfs': Отказано в доступе
Количество файлов с расширением '.txt .doc .jpg .pdf' в директории '/': 0
oskulikov@oskulikov-VirtualBox:~$ ./count_files.sh ".txt .doc .jpg .pdf" /home/oskulikov/Изображения/
Количество файлов с расширением '.txt .doc .jpg .pdf' в директории '/home/oskulikov/Изображения/': 0
oskulikov@oskulikov-VirtualBox:~$
```

Результат

Листинги

1. script 1

```
#!/bin/bash
```

```
# Путь к текущему скрипту
```

```
SCRIPT_PATH=$(realpath "$0")
```

```
# Директория для резервных копий
```

```
BACKUP_DIR="$HOME/backup"
```

```
# Создание директории для резервных копий, если она не существует
```

```
mkdir -p "$BACKUP_DIR"
```

```
# Имя файла без пути
```

```
SCRIPT_NAME=$(basename "$SCRIPT_PATH")
```

```
# Имя и путь резервного архива
```

```
BACKUP_FILE="$BACKUP_DIR/${SCRIPT_NAME}.tar.gz"
```

```
# Создание архива с резервной копией
```

```
tar -czvf "$BACKUP_FILE" -C "$(dirname "$SCRIPT_PATH")" "$SCRIPT_NAME"
```

```
echo "Резервная копия создана: $BACKUP_FILE"
```

2. script 2

```
#!/bin/bash
```

```
# Проверяем, были ли переданы аргументы
```

```
if [ $# -eq 0 ]; then
```

```
    echo "Не переданы аргументы."
```

```
    exit 1
```

```
fi
```

```
# Цикл для перебора всех аргументов
```

```
for arg in "$@"
```

```
do
```

```
    echo "Аргумент: $arg"
```

```
done
```

3. script 3

```
#!/bin/bash
```



```
# Функция для вывода информации о правах доступа
```

```
print_permissions() {
```

```
    local perm_string=""
```

```
    local file=$1
```

```
    local perms=$(stat -c %A "$file")
```

```
    perm_string="${perms:0:1}" # Тип файла
```

```
    perm_string+=" ${perms:1:3}" # Права владельца
```

```
    perm_string+=" ${perms:4:3}" # Права группы
```

```
    perm_string+=" ${perms:7:3}" # Права других
```

```
    echo "$perm_string"
```

```
}
```

```
# Проверяем, был ли передан аргумент каталога
```

```
if [ $# -eq 0 ]; then
```

```
    target_dir="."
```

```
else
```

```
    target_dir="$1"
```

```
fi
```

```
# Проверяем, существует ли каталог
```

```
if [ ! -d "$target_dir" ]; then
```

```
    echo "Ошибка: Каталог '$target_dir' не существует."
```

```
    exit 1
```

```
fi
```

```
# Перебираем файлы в каталоге
for file in "$target_dir"/*; do
    if [ -e "$file" ]; then
        permissions=$(print_permissions "$file")
        file_size=$(stat -c %s "$file")
        mod_time=$(stat -c %y "$file")
        file_name=$(basename "$file")
        echo "$permissions $file_size $mod_time $file_name"
    fi
done
```

4. script 4

```
#!/bin/bash
```

```
# Проверяем, были ли переданы два аргумента
```

```
if [ $# -ne 2 ]; then
```

```
    echo "Использование: $0 <формат файла> <путь к директории>"
```

```
    exit 1
```

```
fi
```

```
# Присваиваем аргументы переменным
```

```
file_format=$1
```

```
target_dir=$2
```

```
# Проверяем, существует ли каталог
```

```
if [ ! -d "$target_dir" ]; then
```

```
    echo "Ошибка: Каталог '$target_dir' не существует."
```

```
    exit 1
```

```
fi
```

```
# Инициализируем счетчик
```

```
file_count=0
```

```
# Подсчитываем количество файлов указанного формата
```

```
for format in $file_format; do
```

```
    count=$(find "$target_dir" -type f -name "$format" | wc -l)
```

```
    file_count=$((file_count + count))
```

```
done
```

```
# Выводим результат
```

```
echo "Количество файлов с форматами '$file_format' в каталоге '$target_dir': $file_count"
```

Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?
 - Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам. UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки:
 - оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций;
 - С-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя С-подобный синтаксис команд, и сохраняет историю выполненных команд;
 - оболочка Корна - напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна; -BASH - сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

2. Что такое POSIX?

- POSIX (Portable Operating System Interface for Computer Environments) - интерфейс переносимой операционной системы для компьютерных сред. Представляет собой набор стандартов, подготовленных институтом инженеров по электронике и радиотехнике (IEEE), который определяет различные аспекты построения операционной системы. POSIX включает такие темы, как программный интерфейс, безопасность, работа с сетями и графический интерфейс. POSIX-совместимые оболочки являются будущим поколением оболочек UNIX и других ОС. Windows NT рекламируется как система, удовлетворяющая POSIX-стандартам. POSIX-совместимые оболочки разработаны на базе оболочки Корна; фонд бесплатного программного обеспечения (Free Software Foundation) работает над тем, чтобы и оболочку BASH сделать POSIX-совместимой.

3. Как определяются переменные и массивы в языке программирования bash?

- Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов.
- Например, команда `mark=/usr/andy/bin` присваивает значение строки символов `/usr/andy/bin` переменной `mark` типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует мета символ.
- Использование команд `b=/tmp/andy-ls -l myfile >pblsls/tmp/andy - ls, ls -l >bls` приведет к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то ее значением является символ пробел. Оболочка bash позволяет создание массивов. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, `set -A states Delaware Michigan "New Jersey"` Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.

4. Каково назначение операторов `let` и `read`?

- Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (`term`), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате.
- Этот формат - `radix#number`, где `radix` (основание системы счисления) - любое число не более 26. Для большинства команд основания систем счисления это - 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток (%). Команда `let` берет два операнда и присваивает их переменной.

5. Какие арифметические операции можно применять в языке программирования bash?

- Оператор Синтаксис Результат ! !expr Если expr равно 0, возвращает 1; иначе 0
!= expr1 !=expr2 Если expr1 не равно expr2, возвращает 1; иначе 0 % expr1%expr2
Возвращает остаток от деления expr1 на expr2 %= var=%expr Присваивает
остаток от деления var на expr переменной var & expr1&expr2 Возвращает
побитовое AND выражений expr1 и expr2 && expr1&&expr2 Если и expr1 и expr2 не
равны нулю, воз- вращает 1; иначе 0 &= var &= expr Присваивает var
побитовое AND перемен- ных var и выражения expr * expr1 * expr2 Умножает
expr1 на expr2 = var = expr Умножает expr на значение var и присваивает
результат переменной var + expr1 + expr2 Скла- дывает expr1 и expr2 += var +=
expr Складывает expr со значением var и результат присваивает var - -expr
Операция отрицания expr (называется унарный минус) - expr1 - expr2 Вычитает
expr2 из expr1 -= var -= expr Вычитает expr из значения var и присваи- вает
результат var / expr / expr2 Делит expr1 на expr2 /= var /= expr Делит var на expr и
присваивает результат var < expr1 < expr2 Если expr1 меньше, чем expr2,
возвращает 1, иначе возвращает 0 « expr1« expr2 Сдвигает expr1 влево на expr2
бит «= var «= expr Побитовый сдвиг влево значения var на expr <= expr1 <= expr2
Если expr1 меньше, или равно expr2, возвра- щает 1; иначе возвращает 0 = var
= expr Присваивает значение expr переменной va == expr1==expr2 Если expr1
равно expr2. Возвращает 1; иначе возвращает 0 > expr1 > expr2 1 если expr1
больше, чем expr2; иначе 0 >= expr1 >= expr2 1 если expr1 больше, или равно
expr2; иначе 0 » expr » expr2 Сдвигает expr1 вправо на expr2 бит »= var »=expr
Побитовый сдвиг вправо значения var на expr ^ expr1 ^ expr2 Исключающее OR
выражений expr1 и expr2 ^= var ^= expr Присваивает var побитовое
исключающее OR var и expr | expr1 | expr2 Побитовое OR выражений expr1 и expr2
|= var |= expr Присваивает var «исключающее OR» пе- ременной var и
выражения expr || expr1 || expr2 1 если или expr1 или expr2 являются нену-
левыми значениями; иначе 0 ~ ~expr Побитовое дополнение до expr.
- 6. Что означает операция (())?
 - Условия оболочки bash.
- 7. Какие стандартные имена переменных Вам известны?
 - Имя переменной (идентификатор) - это строка символов, которая отличает
эту переменную от других объектов программы (идентифицирует
переменную в программе). При задании имен переменным нужно соблюдать
следующие правила: § первым символом имени должна быть буква.
Остальные символы - буквы и цифры (прописные и строчные буквы
различаются). Можно использовать символ «_»; § в имени нельзя
использовать символ «.»; § число символов в имени не должно превышать
255; § имя переменной не должно совпадать с зарезервированными
(служебными) словами языка. Var1, PATH, trash, mon, day, PS1, PS2 Другие
стандартные переменные: -HOME - имя домашнего каталога пользователя.
Если команда cd вводится без аргументов, то происходит переход в каталог,
указанный в этой переменной. -IFS - последовательность символов,
являющихся разделителями в командной строке. Это символы пробел,
табуляция и перевод строки(new line). -MAIL - командный процессор каждый
раз перед выводом на экран промптера проверяет содержимое файла, имя
которого указано в этой переменной, и если содержимое этого файла
изменилось с момента последнего ввода из него, то перед тем как вывести

на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта). -TERM - тип используемого терминала. -LOGNAME - содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

8. Что такое метасимволы?

- Такие символы, как ' < > * ? | " & являются метасимволами и имеют для командного процессора специальный смысл.

9. Как экранировать метасимволы?

- Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например, -echo выведет на экран символ, -echo ab'|'cd выведет строку ab|cd.

10. Как создавать и запускать командные файлы?

- Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде bash командный_файл [аргументы] Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды chmod +x имя_файла Теперь можно вызывать свой командный файл на выполнение просто, вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит ее интерпретацию.

11. Как определяются функции в языке программирования bash?

- Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом -f. Команда typeset имеет четыре опции для работы с функциями:
 - -f - перечисляет определенные на текущий момент функции;
 - -ft - при последующем вызове функции иницирует ее трассировку;
 - -fx - экспортирует все перечисленные функции в любые дочерние программы оболочек;
 - -fu - обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом?

- ls -lrt Если есть d, то является файл каталогом

13. Каково назначение команд set, typeset и unset?

- Используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, `set -A states Delaware Michigan "New Jersey"`. Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды `set`. Наиболее распространенным является сокращение, избавляющееся от слова `let` в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте `typeset -i` для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово `integer` (псевдоним для `typeset -i`) и объявлять переменные целыми. Таким образом, выражения типа `x=y+z` воспринимаются как арифметические. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` - перечисляет определенные на текущий момент функции; `-ft` - при последующем вызове функции иницирует ее трассировку; `-fx` - экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` - обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции. В переменные `top` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

14. Как передаются параметры в командные файлы?

- Символ `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров.
- Использование комбинации символов `$0` приводит к подстановке вместо нее имени данного командного файла. Примере: пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1` Если Вы введете с терминала команду: `where andy`, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал не будет выведено ничего. Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты

своей работы на стандартный вывод. В этом примере команда `grep` используется как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: `$ where andy andy ttyG Jan 14 09:12 $` Определим функцию, которая изменяет каталог и печатает список файлов: `$ function clist { > cd $1 > ls > }`. Теперь при вызове команды `clist` каталог будет изменен каталог и выведено его содержимое.

15. Назовите специальные переменные языка `bash` и их назначение.

- `$*` - отображается вся командная строка или параметры оболочки;
- `$?` - код завершения последней выполненной команды;
- `$$` - уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- `#!` - номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- `$-` - значение флагов командного процессора;
- `${#}` - возвращает целое число — количество слов, которые были результатом `$`;
- `${#name}` - возвращает целое значение длины строки в переменной `name`;
- `${name[n]}` - обращение к `n`-ному элементу массива;
- `${name[*]}` - перечисляет все элементы массива, разделенные пробелом;
- `${name[@]}` - то же самое, но позволяет учитывать символы пробелы в самих переменных;
- `${name:-value}` - если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
- `${name:value}` - проверяется факт существования переменной;
- `${name=value}` - если `name` не определено, то ему присваивается значение `value`;
- `${name?value}` - останавливает выполнение, если имя переменной не определено, и выводит `value`, как сообщение об ошибке; это выражение работает противоположно `{name-value}`. Если переменная определена, то подставляется `value`;
- `${name#pattern}` - представляет значение переменной `name` с удаленным самым коротким левым образцом (`pattern`);
- `${#name[*]}` и `${#name[@]}` - эти выражения возвращают количество элементов в массиве `name`.
- `##` - вместо нее будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение

Выводы

В ходе выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux. Научился писать небольшие командные файлы.