

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»

Факультет Физико-математических и естественных наук
Кафедра информационных технологий

Отчет по лабораторной работе №10

Тема "Программирование в командном процессоре ОС UNIX. Командные файлы".

Выполнила:

Студентка группы НПИбд-02-21

Студенческий билет №1032211220

Шаповалова Диана Дмитриевна

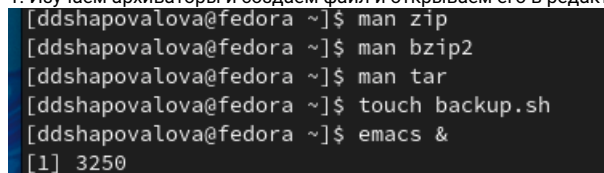
21 мая 2022г

Москва

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

Ход работы

1. Изучаем архиваторы и создаем файл и открываем его в редакторе



```
[ddshapovalova@fedora ~]$ man zip
[ddshapovalova@fedora ~]$ man bzip2
[ddshapovalova@fedora ~]$ man tar
[ddshapovalova@fedora ~]$ touch backup.sh
[ddshapovalova@fedora ~]$ emacs &
[1] 3250
```

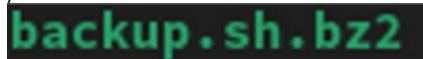
рис. 1

2. Пишем первую программу и проверяем ее работу



```
#!/bin/bash
name = 'backup.sh'
bzip2 ${name}
mv ${name}.bz2 /home/ddshapovalova
echo "done"
```

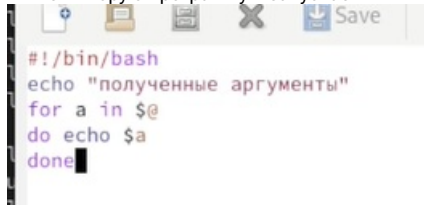
рис. 2



backup.sh.bz2

рис. 3

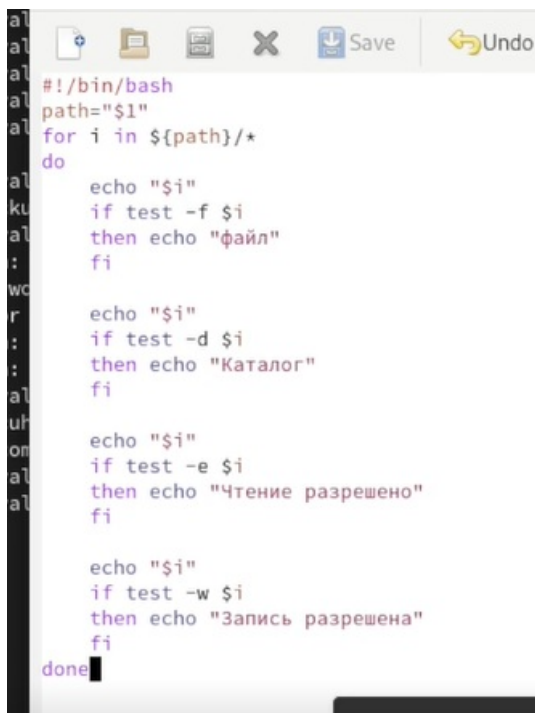
3. Пишем вторую программу и запускаем



```
#!/bin/bash
echo "полученные аргументы"
for a in $@
do echo $a
done
```

рис. 4

4. Пишем третью программу и запускаем



```
#!/bin/bash
path="$1"
for i in ${path}/*
do
    echo "$i"
    if test -f $i
    then echo "файл"
    fi

    echo "$i"
    if test -d $i
    then echo "Каталог"
    fi

    echo "$i"
    if test -e $i
    then echo "Чтение разрешено"
    fi

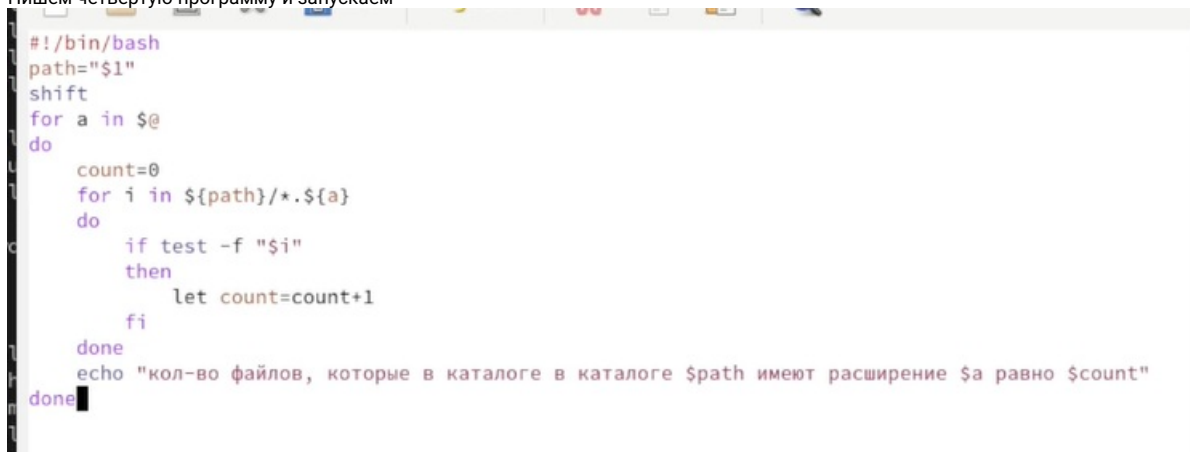
    echo "$i"
    if test -w $i
    then echo "Запись разрешена"
    fi
done
```

рис.5

Чтение разрешено
выполнение разрешено
запись разрешена

рис.6

5. Пишем четвертую программу и запускаем



```
#!/bin/bash
path="$1"
shift
for a in $@
do
    count=0
    for i in ${path}/*.$a
    do
        if test -f "$i"
        then
            let count=count+1
        fi
    done
    echo "кол-во файлов, которые в каталоге $path имеют расширение $a равно $count"
done
```

рис.7

имеют расширение txt равно 2
имеют расширение pdf равно 0
имеют расширение sh равно 4

рис.8

Вывод Я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?
2. Что такое POSIX?
3. Как определяются переменные и массивы в языке программирования bash?
4. Каково назначение операторов let и read?
5. Какие арифметические операции можно применять в языке программирования bash?
6. Что означает операция (())?
7. Какие стандартные имена переменных Вам известны?
8. Что такое метасимволы?
9. Как экранировать метасимволы?
10. Как создавать и запускать командные файлы?
11. Как определяются функции в языке программирования bash?
12. Каким образом можно выяснить, является файл каталогом или обычным файлом?
13. Каково назначение команд set, typeset и unset?
14. Как передаются параметры в командные файлы?
15. Назовите специальные переменные языка bash и их назначение.

Ответы

1. 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;
 - C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
 - оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
 - BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).
2. 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
3. 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 метасимвол. Например, команда `mv afile(mark)` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Оболочка bash позволяет работать с массивами. Для создания массива используется команда `setc` флагом `-A`. За флагом следует имя переменной, а затем список значений, разделённых пробелами. 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. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (`term`), обычно целочисленный. Команда `let` берёт два операнда и присваивает их переменной. Команда `read` позволяет читать значения переменных со стандартного ввода: `echo "Please enter Month and Day of Birth ?"` «read mon day trash». В переменные `moni` `day` будут считаны соответствующие значения, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.
5. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. В (()) записывают условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.
7. `~` — 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, например, 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 (y Вас есть почта)`. `TERM` — тип используемого терминала. `LOGNAME` — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.
8. `'< > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл
9. Снятие специального смысла с метасимвола называется экранированием мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа, который, в свою очередь, является мета символом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$`, `'`, `,`, `"`. Например, `-echo*` выведет на экран символ `,` `-echoab` выведет на экран строку `ab` *`cd`.
10. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `bash командныйфайл [аргументы]`. Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `chmod +x имяфайла`. Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой.
11. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unsetc` флагом `-f`
12. `test -f [путь до файла]` (для проверки, является ли обычным файлом)
`test -d [путь до файла]` (для проверки, является ли каталогом)
13. «set» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «set» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «set| more». Команда «typeset» предназначена для наложения ограничений на переменные. Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.
14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ `$` является метасимволом командного процессора.
15. Специальные переменные:
 1. `$*` — отображается вся командная строка или параметры оболочки;
 2. `$?` — код завершения последней выполненной команды;
 3. `$` (2 Таких знака) — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
 4. `#!` — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
 5. `$` значение флагов командного процессора;
 6. `${#}` — возвращает целое число — количество слов, которые были результатом `$`;
 7. `${#name}` — возвращает целое значение длины строки в переменной `name`;
 8. `${name[n]}` — обращение к n-му элементу массива;
 9. `${name[@]}` — перечисляет все элементы массива, разделённые пробелом;
 10. `${name[@]}` — то же самое, но позволяет учитывать символы пробелы в самих переменных;
 11. `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
 12. `${name:~value}` — проверяется факт существования переменной;
 13. `${name:=value}` — если `name` не определено, то ему присваивается значение `value`;
 14. `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
 15. `${name+value}` — это выражение работает противоположно `${name:-value}`. Если переменная определена, то подставляется `value`;
 16. `${name#pattern}` — представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);
 17. `${#name}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.