Цель работы

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

Задание

Написать командные файлы для выполнения заданий

Ход работы

Необходимо выполнить следующие задания:

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**.

    - Скрипт на изображении выполняет копирование самого себя в директорию **~/backup** в архив **back.gz** (Рис.1)

    ```
    task1.sh          [----] 36 L:[  1+ 1   2/  2] *(144 / 282b) 0009 0x009
    backup=~/backup><------><------><------>#Переменной присваивается путь в директорию backup
    tar -zcvf ${backup}/back.gz task1.sh<-->#Выполняется копирование с архивированием этого набора команд в архив back.gz
    ```

    Рис.1

    - Результат выполнения скрипта (предварительно добавил разрешение на выполнение) (Рис.2)

    ```
    [dnbabkov@dnbabkov lab11]$ ./task1.sh
    task1.sh
    [dnbabkov@dnbabkov lab11]$ ls -l ~/backup
    total 4
    -rw-rw-r--. 1 dnbabkov dnbabkov 310 May 28 23:35 back.gz
    ```

    Рис.2

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.

    - Командный файл принимает на вход произвольное кол-во аргументов. Пока есть аргументы, цикл for работает и проходит по всем этим аргументам, выводя их на экран (Рис.3)

    ```
    task2.sh          [-M--] 41 L:[  1+ 2   3/  4] *(143 / 148b) 0010 0x00A
    for param in "$@"<----->#Цикл, выполняющийся пока есть параметры
    do
    echo "$param"<-><------>#Вывод параметров
    done
    ```

    Рис.3

- Результат выполнения командного файла (предварительно добавил разрешение на выполнение) (Рис.4)

```
[dnbabkov@dnbabkov lab11]$ ./task2.sh 1 2 3 4 5 6 7 8 9 10 11 12
1
2
3
4
5
6
7
8
9
10
11
12
```

Рис.4

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.

- Командный файл принимает на вход путь к нужной директории. Если путь не задан, то поиск будет проходить по корневому каталогу. В каталоге с заданным именем переменная **f** проходит по всем файлам в этом каталоге. Четырем следующим переменным передается информация о разрешениях файла. Переменные **user**, **group** и **global** принимают значения от 0 до 7, которые соответствуют разрешениям для пользователя, группы и остальных пользователей. После выводится имя файла и разрешения к нему (Рис.5).

```
task3Mod.sh        [----]  0 L:[  1+ 0   1/ 30] *(0   /1298b) 0112 0x070
print_perm() {<><------><------>#Функция для определения разрешений файла
  case "$1" in
    0) printf "NO PERMISSIONS";;
    1) printf "Execute only";;
    2) printf "Write only";;
    3) printf "Write & execute";;
    4) printf "Read only";;
    5) printf "Read & execute";;
    6) printf "Read & write";;
    7) printf "Read & write & execute";;
  esac
}

path=$1><------><------><------>#Переменной path присваивается значение первого оператора -.

for f in $path/*; <----><------>#Проходятся все файлы, рассположенные в директории, в которую ведет путь
do.
    perm=$(stat -c%a "$f")<---->#Передается информация о разрешениях
    user=${perm:0:1}<--><------>#Разрешения пользователя
    group=${perm:1:1}<-><------>#Группы
    global=${perm:2:1}<><------>#Других

    b=$(basename $f)<-->#Переменной b присваивается имя файла (без пути)
    echo $b<---><------>#Выводится имя файла
    printf "\tOwner Access: $(print_perm $user)\n"<---->#Вызывается ранее описанная функция
    printf "\tGroup Access: $(print_perm $group)\n"
    printf "\tOthers Access: $(print_perm $global)\n"
    echo ""
done
```

Рис 5

- Результат работы командного файла (предварительно добавил разрешение на выполнение) (Рис.6)

```
[dnbabkov@dnbabkov lab11]$ ./task3Mod.sh ~
a
        Owner Access: Read & write
        Group Access: Read & write
        Others Access: Read only

abc1
        Owner Access: Read & write
        Group Access: Read & write
        Others Access: Read only

australia
        Owner Access: Read & write & execute
        Group Access: Read only
        Others Access: Read only

backup
        Owner Access: Read & write & execute
        Group Access: Read & write & execute
        Others Access: Read & execute

conf.txt
        Owner Access: Read & write
        Group Access: Read & write
        Others Access: Read only

Desktop
        Owner Access: Read & write & execute
        Group Access: Read & execute
        Others Access: Read & execute

Documents
        Owner Access: Read & write & execute
        Group Access: Read & execute
```

Рис.6

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.

    - Командный файл принимает на вход путь к директории, в которой должен проходить поиск, и расширение, файлы с которым нужно искать. Цикл проходит по всем файлам с нужным расширением, выводит их имя и инкриментирует счетчик (Рис.7).

```
task4.sh          [-M--] 67 L:[  1+ 5   6/ 11] *(469 / 717b) 0010 0x00A
path=$1><------><------>#Присвоение переменной первого параметра (путь)
ext=$2<><------><------>#Присвоение переменной второго параметра (расширение)
count=0><------><------>#Счетчик
for f in $path/*$ext<-->#Цикл, идущий по всем файлам, в директории path с расширением ext
do
    b=$(basename $f)<-->#Присвоение переменной имени файла без пути
    echo $b<--><------>#Вывод файла с заданным расширением
    ((count++))><------>#Инкриментирование счетчика
done
echo $count<--><------>#Вывод кол-ва файлов с заданным расширением
```

Рис.7

    - Результат работы командного файла (предварительно добавил разрешение на выполнение) (Рис.8)

```
[dnbabkov@dnbabkov lab11]$ ./task4.sh ~/work/2020-2021/OperatingSystems/laboratory/intro-os/lab06 .md
prentation6Lab.md
report6Lab.md
2
```

Рис.8

Контрольные вопросы

1. 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;
    - C-оболочка (или csh) − надстройка на оболочкой Борна, использующая Сподобный 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/Linuxподобных 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 set с 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. Оболочка bash поддерживает встроенные арифметические функции. Команда **let** является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение − это единичный терм (term), обычно целочисленный. Команда **let** берет 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 mon и 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 (( )) можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.
7. Стандартные переменные:
    - PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
    - PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу $ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >.
    - 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 (у Вас есть почта).
    - TERM: тип используемого терминала.
    - LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.
8. Такие символы, как **' < > * ? | \ " &**, являются метасимволами и имеют для командного процессора специальный смысл.
9. 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 нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ' , , ". Например,
    - echo * выведет на экран символ *,
    - echo ab'|'cd 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 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 её интерпретацию.

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 unset с флагом -f.
12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами **test -f [путь до файла]** (для проверки, является ли обычным файлом) и **test -d [путь до файла]** (для проверки, является ли каталогом).
13. Команду **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. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов $i, где 0 < i < 10, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i, т. е. аргумента командного файла с порядковым номером i. Использование комбинации символов $0 приводит к подстановке вместо неё имени данного командного файла.
15. Специальные переменные:
    - $* – 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-му элементу массива;
    - ${name[*]} – перечисляет все элементы массива, разделённые 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;

- ${name+value} – это выражение работает противоположно ${name-value}. Если переменная определена, то подставляется value;
- ${name#pattern} – представляет значение переменной name с удалённым самым коротким левым образцом (pattern);
- ${#name[*]} и ${#name[@]} – эти выражения возвращают количество элементов в массиве name.

Вывод

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