

Лабораторная работа 11

Покрас И.М. НФИбд-02-20

Цель работы

изучить основы программирования в оболочке ОС UNIX. Научится писать небольшие командные файлы.

Ход работы:

Сначала я ввёл команду `man tar`, чтобы изучить справку и ознакомиться со способами использования команд. Я написал скрипт, который при запуске будет делать резервную копию самого себя в другую директорию `backup` в моём домашнем каталоге.

```
#!/bin/bash
touch backup.sh
cp script1.sh backup.sh
tar -cf lab.tar backup.sh
mv lab.tar ~/Backup
rm -r backup.sh
```

Script

При этом файл архивируется одним из архиваторов на выбор `zip`, `bzip2` или `tar`.



Script result

Я написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт

может последовательно распечатывать значения всех переданных аргументов.

```
#!/bin/bash
echo "Enter your numbers"
head -1
```

Результат терминала:

```
Impokras@Impokras-VirtualBox:~$ cd ~/Documents
Impokras@Impokras-VirtualBox:~/Documents$ chmod u+x ./script2.sh
Impokras@Impokras-VirtualBox:~/Documents$ ./script2.sh
Enter your numbers
2547 2909 100 5830
2547 2909 100 5830
Impokras@Impokras-VirtualBox:~/Documents$
```

Script result

Я написал командный файл — аналог команды ls.

```
#!/bin/bash
for A in *
do if test -d $A
then echo $A: "is a directory"
else echo -n $A: "is a file and "
if test -w $A
then echo "writeable"
elif test -r $A
then echo "readable"
else echo "neither readable nor writeable"
fi
fi
done
```

Он выдает информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога.

```
#!/bin/bash
format=**
direct=**
echo "enter format"
read format
echo "enter direct"
read direct
find "$direct" -name ".$format" -type f | wc -l
```

Script result

Я написал 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.

```
#!/bin/bash
format=**
direct=**
echo "enter format"
read format
echo "enter direct"
read direct
find "$direct" -name ".$format" -type f | wc -l
```

Script

Путь к директории также передаётся в виде аргумента командной строки.



```
lnpokras@lnpokras-VirtualBox: $ ./script4.sh
enter format
-jpg
enter direct
/home/lnpokras/Downloads/
0
lnpokras@lnpokras-VirtualBox: $
```

Script result

Вывод

Я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.

Контрольные вопросы:

1. Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. 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 Сподобный 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. 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),

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 $mark` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того, чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}` например, использование команд `b=/tmp/andy-ls -l myfile`
>
`bls` приведет к переназначению стандартного вывода команды `ls` терминала на файл `/tmp/andy - ls`, а использование команды `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.
4. 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, возвращает 1; иначе 0
&= `var &= expr` Присваивает `var` побитовое AND переменных `var` и выражения `expr`
* `expr1 * expr2` Умножает `expr1` на `expr2`
= `var = expr` Умножает `expr` на значение `var` и присваивает результат переменной `var`
+ `expr1 + expr2` Складывает `expr1` и `expr2`

`+= var += expr` Складывает `expr` со значением `var` и результат присваивает `var` - `- expr` Операция отрицания `expr` (называется унарный минус) - `expr1 - expr2`
`expr1 -= expr` Вычитает `expr` из значения `var` и присваивает 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`, возвращает 1; иначе возвращает 0 `= var = expr` Присваивает значение `expr` переменной `var` `== 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» переменной `var` и выражения `expr` `|| expr1 || expr2` 1 если или `expr1` или `expr2` являются ненулевыми значениями; иначе 0 `~ ~expr` Побитовое дополнение до `expr`

6. Условия оболочки `bash`, в двойные скобки `--(())`.
7. 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. Такие символы, как ' < > * ? | " & являются метасимволами и имеют для командного
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, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например, `-echo *`выведет на экран символ, `-echo ab'|'cd`выдаст 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 ее интерпретацию.
11. 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