**105.2 Создание и настройка простых скриптов**

Студент должен уметь редактировать существующие, и писать новые скрипты bash.

**Изучаем**:

* стандартный синтаксис sh (циклы, тест);
* подстановку команд;
* вывод команды test;
* условную пересылку почты суперпользователю;
* выбор используемого интерпретатора скриптов;
* управление расположением, владельцами и правами доступа к скриптам.

Linux – операционная система, в которой можно сделать все что угодно, нужно только знать как. Часто для выполнения рутинных задач администраторы Linux пишут скрипты (последовательность команд, сохраненных в виде единого файла).

Каждый скрипт должен начинаться с символов #! после которых указывается путь к интерпретатору команд скрипта. Чаще всего используется bash (хотя может быть любой shell, или даже perl и т.д.) , например: ***#!/bin/bash***

Для того чтобы текстовый файл с набором команд превратить в исполняемый, нужно добавить кому-нибудь права на запуск этого файла, например:

***Chmod u+X ifscript*** *(добавляет владельцу права на запуск файла ifscript в текущем каталоге).*

Каталоги в которых могут находиться исполняемые файлы, видимые системой, определяются переменной **PATH**. Так как туда по умолчанию не входит домашний каталог пользователя, то чтобы запустить скрипт из него нужно запустить его командой ***./ifscript***

Если на файл установлены разрешения запуска (кому-нибудь: владельцу, группе или остальным), то установка suid-бита сделает файл запускаемым от имени владельца. Сделать это можно командой:

***chmod u+s ifscript***

\_\_\_

Команда **test** проверяет файл на соответствие определенным условиям. Например, для использования доступны следующие ключи при проверке файла:

* **-x**  исполняемый файл;
* **-e**  файл существует;
* **-eq** значения равны;
* **-ne**  значения не равны;
* **-z**  существует ли значение.

На выходе команда **test** дает значение переменной **$?**:

* **0** – если результат положительный;
* **не 0**  – если отрицательный.

Например:

***Chmod ugo-X ifscript*** *(отнять у всех права исполнения файла ifscript);*

***Test –x ifscript*** *(проверить является ли файл ifscript исполняемым);*

***Echo $?*** *(вывести значение переменной $?).*

или

***test 100 -gt 99 && echo "Yes, that's true." || echo "No, that's false."*** *(проверить если 100 больше 99 то вывести "Yes, that's true.", в противном случае вывести "No, that's false.")*

Дополнительные символы, рассматриваемые на занятии:

* **&&** - все что идет после него выполнится только в случае успеха того что идет до него;
* **||** - все что идет после него выполняется только если все что шло до него не выполнилось.

**\_\_\_**

Часто в скриптах используются условные выражения, имеющие следующий формат:

**if** [ условие ]

**then**

***набор команд***

**else**

***набор команд***

**fi**

Пример скрипта:

***#!/bin/bash*** *(интерпретатор команд /bin/bash)*

***#My favorite script*** *(комментарий)*

***Echo “Are you hungry?”*** *(вывести в консоль строку “Are you hungry?”)*

***Read VALUE*** *(считать введенные данные)*

***If [ $VALUE = “Yes” ];*** *(если было введено ”Yes”)*

***Then*** *(то)*

***Echo “Make some pancakes”*** *(вывести в консоль строку “Make some pancakes”)*

***Else*** *(во всех других случаях)*

***Echo “Go on with your work then!”*** *(вывести в консоль строку “Go on with your work then!”)*

***Fi*** *(конец условия)*

Для того чтобы добавить отправку почтового сообщения в скрипт, можно в качестве действия указать:

***echo ”He is not hungry!” | mail –s “Info” root*** *(отправить текст «He is not hungry!» в качестве текста почтового сообщения пользователю root с темой “Info”)*

В скриптах можно использовать подстановку команд, например:

***#!/bin/bash*** *(интерпретатор команд /bin/bash)*

***X=$( date )*** *(X принимает значение у вывода команды date)*

***Y=`uptime –p`*** *(Y принимает значение у вывода команды uptime -p)*

***Echo “Today is $X and we are $Y”*** *(Вывести текст «Today is $X and we are $Y», где $X является значением переменной X, а $Y – значением переменной Y)*

\_\_\_

Также в скриптах часто встречаются различные циклические конструкции, в частности циклы **FOR**, например:

***for x in 5 6 7*** *(для x равно 5, 6 и 7)*

***do*** *(выполни)*

***echo “number $x”*** *(вывод «number $x», где $x – значение переменной x)*

***done*** *(конец цикла)*

и циклы **WHILE**, например:

***x=10*** *(для x равно 10)*

***while [$x –ne 20]*** *(до тех пор пока x не равно 20)*

***do*** *(выполни)*

***echo “number $x”*** *(вывод «number $x», где $x – значение переменной x)*

***x=$(($x+1)*** *(увеличь значение переменной x на единицу)*

***done*** *(конец цикла)*

Также последовательность команд можно задавать при помощи команды seq, например:

***For x in `seq 3 8`*** *(для x в диапазоне от трех до восьми)*

В скриптах также часто используется команда **exec**, которая позволяет:

· выполнить определенную программу, не создавая отдельного процесса и заменяя собой оболочку, например:

***exec some\_program*** *(запустит программу, в рамке текущего сеанса пользователя)*

*Тут много преимуществ:*

*1)* *можно бросить это в настройки входа пользователя, и он сможет работать только с этим приложением, не имея доступа к консоли;*

*2)* *можно запускать внутри одной оболочки свою оболочку;*

*3)* *можно переходить от процесса к процессу, не сохраняя предыдущие в ОЗУ, для экономии ресурсов.*

· изменить вывод для текущей оболочки или программы, например:

***exec >/var/adm/my.log 2>&1*** *(отправляй весь вывод в файл /var/adm/my.log, а ошибки шли в консоль)*