Отчёта по лабораторной работе №10

Программирование в командном процессоре ОС UNIX. Командные файлы

Ортега Вероника

Содержание

# 1 Цель работы

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

# 2 Задание

1. Написать скрипт, который при запуске будет делать резервную копию са- мого 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 произволь- ное 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 команд- ной 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.

# 3 Выполнение лабораторной работы

Первым шагом я ознакомилась с теоретическим материалом, представленным в файле Лабораторной работы и приступила к первому скрипту.В нем было необходимо использовать архив zip, поэтому я с помощью команды man zip прочла справку о нем. (рис. [1](#fig:001)).

Figure 1: Справка о zip

Писать скрипты я решила в редакторе emacs, поэтому я вызвала его через консоль одноименной командой. Далее я создала новый файл script1. После при- ступила непосредственну к коду. Заархивировать файл можно с помощью zip, а перенести его с помощью mv. (рис. [2](#fig:002)).

Figure 2: Первый скрипт

Листинг первого скрипта: zip script1.zip script1 mv script1.zip /home/eeparfenova/backup После я вернулась в консоль, создала каталог backup с помощью mkdir (у меня его не было) и сделала файл исполняемым с помощью команды chmod +x script1. Вызвала скрипт с помощью ./script1. Он сработал успешно. (рис. [3](#fig:003)).(рис. [4](#fig:004)).

Figure 3: Работа первого скрипта

Figure 4: Работа первого скрипта

Переходим ко второму заданию. В нем было необходимо написать командный файл, который бы обрабатывал значения аргументов и печтал их. Я создала файл script2 и осуществила это с помошью цикла for. (рис. [5](#fig:005)).

Figure 5: Второй скрипт

Листинг второго скрипта: count=1 for param in “$@" do echo "$count: $param" count=$((count + 1)) done Снова вернулась в терминал и сделала файл исполняемым той же командой, заменив только название. После проверила скрипт (./script2), вписав вначале три аргумента, а затем свыше 10 (это требовалось проверить в задании)(рис. [6](#fig:006)).

Figure 6: Работа второго скрипта

Переходим к третьему заданию. Еще раз внимательно прочитав теорию, я поняла, что код этого скрипта представлен в файле, поэтому я написала команд- ный файл по подобию примера из Лабораторной работы. Создала файл script3. В первой части кода мы проверяем директория это или файл, а во втором выводим права доступа c помощью циклов.(рис. [7](#fig:007)).

Figure 7: Третий скрипт

Листинг третьего скрипта: for A in \* do if test -d $A then echo $A: is dir else echo -n $A: “is a file and” if test -w $A then echo writeable elif test -r $A then echo readable else echo unwriteable and unredeable fi fi done Сделав файл исполняемым, я проверила его, вызвав с помощью ./script3. Скрипт сработал успешно и вывел информацию по домашнему каталогу(рис. [8](#fig:008)).

Figure 8: Работа третьего скрипта

Посследним заданием было вычислить количество файлов определенного формата в определенной директории. Вначале я создала файл script4. В коде я вначале попросила пользователя ввести директорию и формат, а затем нашла нужное с помощью команды find(рис. [9](#fig:009)).

Figure 9: Четвертый скрипт

Листинг четвертого скрипта: echo “Insert dir:” read dirr echo “Insert file format:” read form find $dirr -name "\*$form” -type f | wc -l Далее я сделала файл исполняемым и вызвала его, используя ./script4. Прове- рила файл, поискав в “Загрузках” файлы формата pdf. Скрипт сработал успешно.(рис. [10](#fig:010)).

Figure 10: Работа четвертого скрипта

(рис. ??). bash prog02.sh

(рис. ??). emacs prog02

(рис. ??). prog02

# 4 Выводы

Мы изучили основы программирования в оболочке ОС UNIX/Linux и научились писать небольшие командные файлы.

# 5 Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

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 csh) — 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 осно- ве своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation)

1. Что такое POSIX?

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 различ- ных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

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

Командный процессор bash обеспечивает возможность использования пере- менных типа строка символов. Имена переменных могут быть выбраны поль- зователем. 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 $. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, ко- торые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: ${имя переменной}. Например, использование команд b=/tmp/andy2 ls -l myfile > bls приведёт к подстановке в командную строку значения переменной bls. Если переменной bls не было предварительно присвоено никакого значения, то её значением будет символ пробела. Оболочка bash позволяет работать с массивами. Для создания массива исполь- зуется 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. Индек- сация массивов начинается с нулевого элемента.

1. Каково назначение операторов let и read?

let - команда, после которой аргументы представляют собой выражение, под- лежащее вычислению.

read - команда, позволяющая читать перемнные, вводящиеся с компьютера. 5. Какие арифметические операции можно применять в языке программиро- вания bash?

Простейшими математическими выражениями являются сложение (+), вычи- тание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%). Однако их намного больше

1. Что означает операция (( ))?

Условия оболочки bush.

1. Какие стандартные имена переменных Вам известны?

PATH; PS1; PS2; HOME; IFC; MAIL; TERM; LOGNAME

1. Что такое метасимволы?

Такие символы, как ’ < > \* ? | ” &, являются метасимволами и имеют для ко- мандного процессора специальный смысл.

1. Как экранировать метасимволы?

Снятие специального смысла с метасимвола называется экранированием мета- символа. Экранирование может быть осуществлено с помощью предшествующе- го 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 $, ’ , , ”

1. Как создавать и запускать командные файлы?

Нужно просто создать файл через touch,а затем сделать его исполняемым через chmod +x/название

1. Как определяются функции в языке программирования bash?

Указать ключевое слово function, затем написать еее название и открыть фи- гурную скобку.

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

Командой ls -lrt. Есть d, то каталог.

1. Каково назначение команд set, typeset и unset?

set - установка перемнных оболочек и сред unset - удаление переменной из командной оболочки typeset - наложение ограничений на переменные

1. Как передаются параметры в командные файлы?

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 < � < 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

1. Назовите специальные переменные языка bash и их назначение.

При использовании в командном файле комбинации символов $# вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение. Вот ещё несколько специальных переменных, используемых в командных файлах: – $\* — 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; – ${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 опре- делено, 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);

# Список литературы