

Цель работы

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

Ход работы

- . Написали скрипт, который при запуске делает резервную копию самого себя в другую директорию под названием `backup`. При этом файл архивируется архиватором `tar` (см. Рис. 1, 2)
- . Написали скрипт, который может последовательно распечатывать все переданные ему аргументы. (см. Рис. 3, 4)
- . Написали командный файл - аналог команды `ls`. Он выдает информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам данного каталога. (см. Рис. 5, 6)
- . Написали командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в данной директории. (см. Рис. 7, 8)

Вывод

Изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы.

Контрольные вопросы

- . Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называются интерпретаторами команд. 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; – C-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд; – оболочка Корна - 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).
- . 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), который определяет различные аспекты построения операционной системы. POSIX включает такие темы, как программный интерфейс, безопасность, работа с сетями и графический интерфейс. POSIX-совместимые оболочки являются будущим поколением оболочек UNIX и других ОС. Windows NT рекламируется как система, удовлетворяющая POSIX-стандартам. POSIX-совместимые оболочки разработаны на базе оболочки Корна; фонд бесплатного программного обеспечения (Free Software Foundation) работает над тем, чтобы и оболочку BASH сделать POSIX-совместимой.

- 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 $mark` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того, чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имяпеременной}` например, использование команды `mv /tmp/andy-ls -l myfile > ${b}ls` приведет к переназначению стандартного вывода команды `ls` с терминала на файл `/tmp/andy-ls`, а использование команды `ls -l > $b ls` 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. Например, `set -A states Delaware Michigan "New Jersey"` Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента.
- 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.
- Какие арифметические операции можно применять в языке программирования `bash`?
 Оператор Синтаксис Результат
`if 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` Вычитает `expr2` из `expr1`
`- = var - = expr` Вычитает `expr` из значения `var` и присваивает результат `var`
`/ expr1 / 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 `va == 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` Исключающая OR

выражений `expr1` и `expr2`
`^= var ^= expr` Присваивает `var` побитовое исключающая OR `var` и `expr`
`|expr1 | expr2` Побитовое OR выражений `expr1` и `expr2`
`|= var |= expr` Присваивает `var` «исключающая OR» переменной `var` и выражения `expr`
`||expr1 ||expr2` 1 если или `expr1` или `expr2` являются ненулевыми значениями; иначе 0
`~expr` Побитовое дополнение до `expr`.

. Условия оболочки `bash`, в двойные скобки — `(())`.

.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 `Sh` имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды `set`.

. Такие символы, как `'< > * ? | \` & являются метасимволами и имеют для командного процессора специальный смысл.

. 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`.

. 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 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 осуществляет интерпретацию.

- Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следуют имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unsetc` 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 файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции.

. Is -l Если есть `d`, то является файл каталогом

- Используется команда `setc` флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, `set-A states Delaw are M ichigan "New Jersey"`. Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды `set`. Наиболее распространенным является сокращение, избавляющее от слова `letv` в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте `typeset-i` для объявления присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово `integer` (псевдоним для `typeset-i`) и объявлять переменные целыми. Таким образом, выражения типа `x=u+z` воспринимаются как арифметические. 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 `unsetc` 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 файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции. В переменные `month` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

- 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 нее 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 нее имени данного

командного файла.Примере:пусть к командному файлу `w here` имеется доступ по выполнению и этот командный файл содержит следующ ий конвейер:`w ho |grep $1` Если Вы введете с терминала команду:`w here andy`,то в случае,если пользователь,зарегистрированный в ОС UNIX под именем `andy`,в данный момент работает в ОС UNIX,на терминал будет выведена строка,содержащ аяномер терминала,используемого указанным пользователем.Если же в данный моментэтотпользовательне работаетв ОС UNIX,то на терминал не будетвыведено ничего.Команда `grep` производит контекстный поиск в тексте,поступающ ем со стандартного ввода,длянахожденияв этом тексте строк,содержащ ихпоследовательности символов, переданные ей в качестве аргументов,и выводитрезультаты своей работы на стандартный вывод.В этом примере команда `grep` используется как фильтр,обеспечивающ ий ввод со стандартного ввода и вывод всех строк,содержащ их последовательность символов `andy`,на стандартный вывод.В ходе интерпретации этого файла командным процессором вместо комбинации символов `$1` осущ ествляетсяподстановка значенияпервого и единственного параметра `andy`.Если предположить,что пользователь,зарегистрированный в ОС UNIX под именем `andy`,в данный момент работает в

ОС UNIX,то на терминале Вы увидите примерно следующ ее:`$ w here andy andy ttyG Jan 14 09 12 $`
Определим функцию ,которая изменяет каталог и печатает список файлов:`$ function clist{ > cd $1 > ls > }`.Теперь при вызове команды `clistкаталог` будет изменен каталог и выведено его содержимое.

- `$*` — отображаетсявсякоманднаястрока или параметры оболочки;
- `$?` — код заверш енияпоследней выполненной команды;
- `$$` — уникальный идентификатор процесса,в рамкахкоторого выполняетсякомандный процессор;
- `!`— номер процесса,в рамкахкоторого выполняетсяпоследняявызваннаяна выполнение в командном режиме команда;
- `$-` — значение флагов командного процессора;
- `${#}`— *возвращ аетцелое число — количество слов,которые были результатом \$;*
- `${#name}` — возвращ ает целое значение длины строки в переменной `name`;
- `${name[n]}` — обращ ение к `n`-ному элементу массива;
- `${name[*]}` — перечисляет все элементы массива,разделенные пробелом;
- `${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 сущ ествования переменной;
- `${name=value}` — если `name` не определено,то ему присваивается значение `value`;
- `${name?value}` — останавливает выполнение,если имя переменной не определено,и выводит `value`, каксообщ ение об ош ибке;

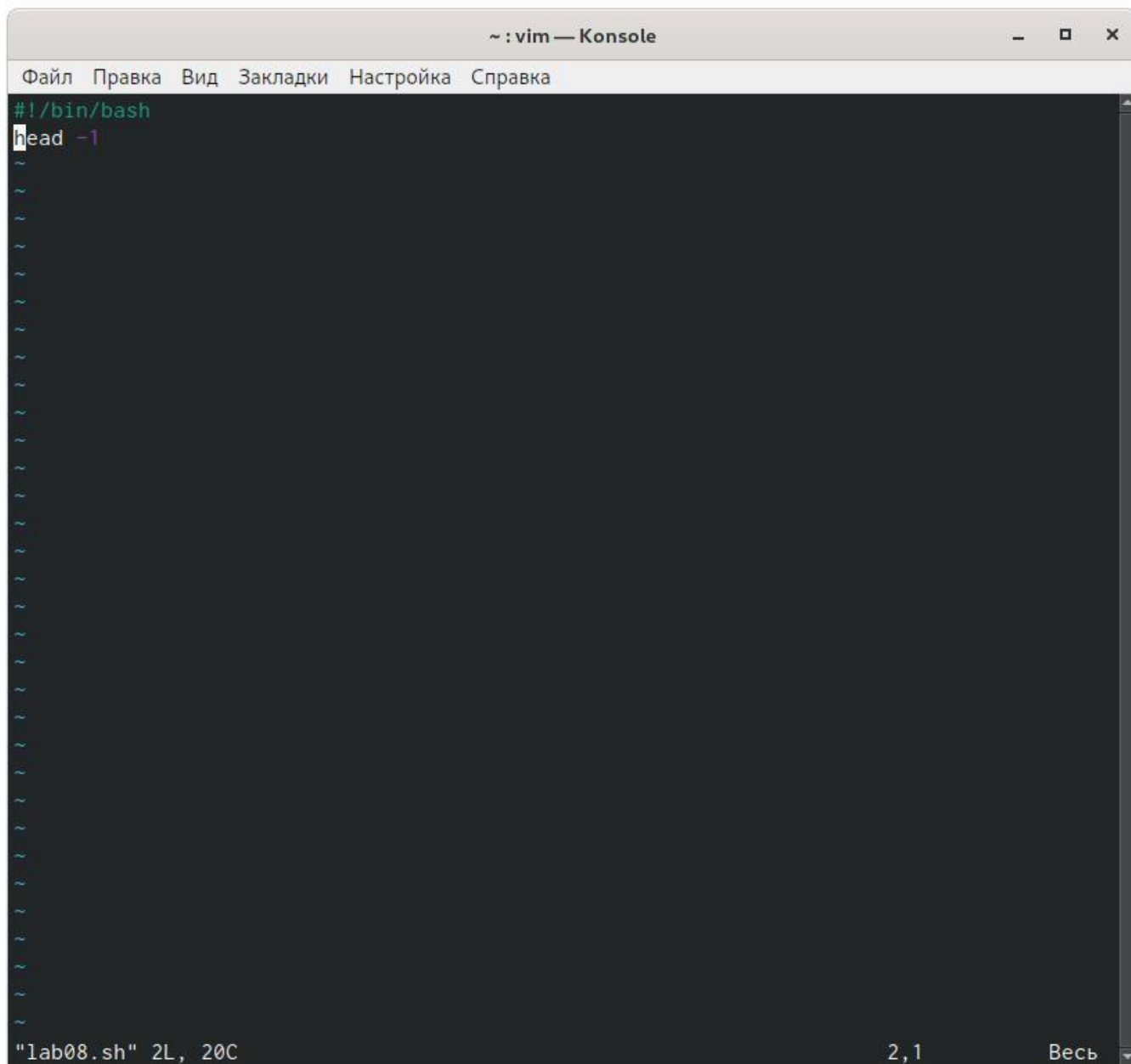


Рис.3


```
~ : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
./lab08.sh: строка 8: test: 2021-04-22: ожидается бинарный оператор
neither readable nor writeable
abcl: is a file andwriteable
backup: is a directory
conf.txt: is a file andwriteable
docs: is a directory
file1: is a file andwriteable
file.txt: is a file andwriteable
fun: is a directory
home: is a directory
image: is a directory
images: is a directory
lab: is a directory
lab02: is a directory
lab_05: is a directory
#lab05.sh#: is a file andwriteable
lab_06: is a directory
#lab06.cpp#: is a file andwriteable
#lab06.sh#: is a file andwriteable
lab_07: is a directory
#lab07.sh#: is a file andwriteable
lab07.sh: is a file andwriteable
lab07.sh~: is a file andwriteable
lab08.sh: is a file andwriteable
lab_09: is a directory
lab_10: is a directory
laboratory: is a directory
may: is a file andwriteable
monthly: is a directory
play: is a directory
public: is a directory
public_html: is a directory
reports: is a directory
ski.plases: is a directory
#test#: is a file andwriteable
```

Рис.6

[illegible]

Рис.7

