

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №10

дисциплина: Операционные системы

Студент: Кармацкий Никита Сергеевич

Группа: НФИбд-01-21

Москва

2022 г.

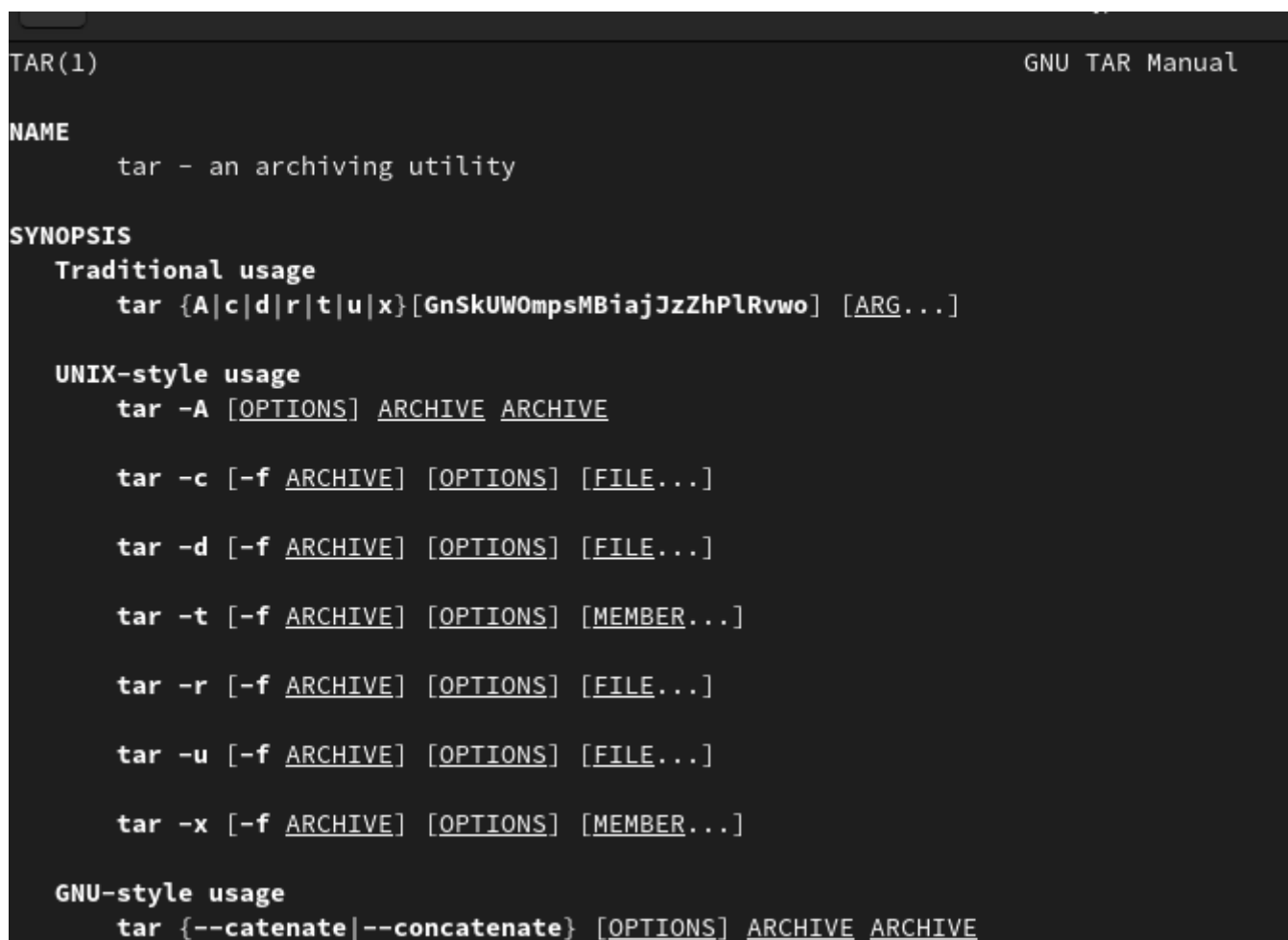
Цель работы:

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

Основные этапы выполнения работы

1. Написали скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в нашем домашнем каталоге. При этом файл будет архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации узнали, изучив справку.

Для начала изучим информацию о архиваторе tar



```
TAR(1) GNU TAR Manual

NAME
    tar - an archiving utility

SYNOPSIS
    Traditional usage
        tar {A|c|d|r|t|u|x}[GnSkUWompsMBiajJzZhPlRvwo] [ARG...]

    UNIX-style usage
        tar -A [OPTIONS] ARCHIVE ARCHIVE

        tar -c [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -d [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]

        tar -r [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -u [-f ARCHIVE] [OPTIONS] [FILE...]

        tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

    GNU-style usage
        tar {--catenate|--concatenate} [OPTIONS] ARCHIVE ARCHIVE
```

Рис.1 Изучали справку о tar

Далее напишем программу

Листинг программы:

```
tar -cf script01.tar script01

mv script01.tar /home/nskarmackiyj/backup/script01.tar
```

Работа программы:

```
[nskarmackiyj@fedora ~]$ mcedit script01
[nskarmackiyj@fedora ~]$ chmod +x script01
[nskarmackiyj@fedora ~]$ ./script01
[nskarmackiyj@fedora ~]$ cd backup/
[nskarmackiyj@fedora backup]$ ls
script01.tar
[nskarmackiyj@fedora backup]$ tar -xf script01.tar
[nskarmackiyj@fedora backup]$ ls
script01 script01.tar
```

Рис.2 Работа программы

2. Написали пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.

Листинг программы:

```
count=0

for parametr in "$@"
do

count=$((count+1))

echo "$count: $parametr"

done
```

Работа программы:

```
[nskarmackiyj@fedora ~]$ mcedit script02

[nskarmackiyj@fedora ~]$ chmod +x script02
[nskarmackiyj@fedora ~]$ ./script02 1 2 3 4
1: 1
2: 2
3: 3
4: 4
[nskarmackiyj@fedora ~]$ ./script02 1 2 3 4 5 6 7 8 9 10 11
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
7: 7
8: 8
9: 9
10: 10
11: 11
```

Рис.3 Работа командного файл

3. Написали 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.

Листинг программы:

```
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 not writeable
        fi
    fi
done
```

Работа программы:

```
[nskarmackiyj@fedora ~]$ chmod +x script03
[nskarmackiyj@fedora ~]$ ./script03
backup: is a directory
bin: is a directory
script01:is a file and writeable
script02:is a file and writeable
script03:is a file and writeable
work: is a directory
Видео: is a directory
Документы: is a directory
Загрузки: is a directory
Изображения: is a directory
Музыка: is a directory
Общедоступные: is a directory
```

Рис.4 Работа командного файла файла

4. Написали 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.

Листинг программы:

```
echo "Insert directory: "

read directory

echo "Insert file format: "

read format

find $direcrory -name "$*$format" -type f | wc -l
```

Работа программы:

```
[nskarmackiyj@fedora ~]$ mcedit script04

[nskarmackiyj@fedora ~]$ chmod +x script04
[nskarmackiyj@fedora ~]$ ./script04
Insert directory:
/home/nskarmackitj
Insert file format:
.jpeg
1
```

Рис.5 Работа командного файла файла

Вывод

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

Ответы на контрольные вопросы

1. 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 называют 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 Сподобный 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).
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), 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.
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 > ${b}ls` приведет к переназначению стандартного вывода команды `ls` с терминала на файл `/tmp/andy-ls`, а использование команд `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. 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. 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 `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

`<=` `var <= expr` 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` Если `expr1` больше, чем `expr2`; иначе 0

`>=` `expr1 >= expr2` Если `expr1` больше, или равно `expr2`; иначе 0

`>=` `var >= expr` 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` Если `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 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, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например, –echo *выведет на экран символ*, –echo ab'|'cd выведет 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 файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноименными именами функций, загружает его и вызывает эти функции.
12. ls -lrt Если есть d, то является файл каталогом
13. 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. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set. Наиболее распространенным является сокращение, избавляющееся от слова let в

программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте `typeset -i` для объявления и 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=y+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 `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 инициализирует 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 `top` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

14. 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. Рассмотрим это на примере. Пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1`. Если Вы введете с терминала команду: `where 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, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал не будет выведено ничего. Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандартный вывод. В этом примере команда `grep` используется как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: `$ where 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 `clist` каталог будет изменен каталог и выведено его содержимое.

15.

– `$*` — отображается вся командная строка или параметры оболочки; `$_` — 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-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 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 удаленным самым коротким левым образцом (pattern); – \${#name[]} и \${#name[@]} — эти выражения возвращают количество элементов в массиве name. – \$# вместо нее будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение