Отчёт по лабораторной работе №12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Дарья Эдуардовна Ибатулина

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (> /dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

2. Реализовать команду man с помощью командного файла. Изучите содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

3. Используя встроенную переменную $RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита.

Учтите, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

3 Теоретическое введение

3.1 Командные процессы (оболочки)

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;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — 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) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения

совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

3.2 Переменные в языке программирования bash

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
```

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}
```

Оболочка bash позволяет работать с массивами. Для создания массива используется 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. Индексация массивов начинается с нулевого элемента.

3.3 Использование арифметических вычислений. Операторы let и read

Команда let берет два операнда и присваивает их переменной. Положительным моментом команды let можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа let sum=x+7, и let будет искать переменную x и добавлять к ней 7.

Команда let также расширяет другие выражения let, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда let не ограничена простыми арифметическими выражениями.

Команда read позволяет читать значения переменных со стандартного ввода:

```
echo "Please enter Month and Day of Birth ?"
read mon day trash
```

3.4 Командные файлы и функции

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
```

4 Выполнение лабораторной работы

1. Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (> /dev/tty#, где # — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.(рис. 4.1).

Открыть

12_1.sh
~/

```
#!/bin/bash

lockfile="./lock.file"
exec {fn}>$lockfile

while test -f "$lockfile"
do
if flock -n ${fn}
then
    echo "File is blocked"
    sleep 5
    echo "File is unlocked"
    flock -u ${fn}
else
    echo "File is blocked"
    sleep 5
fi
done
```

Рис. 4.1: Код для первой программы

Проверим, что код работает верно (рис. 4.2).

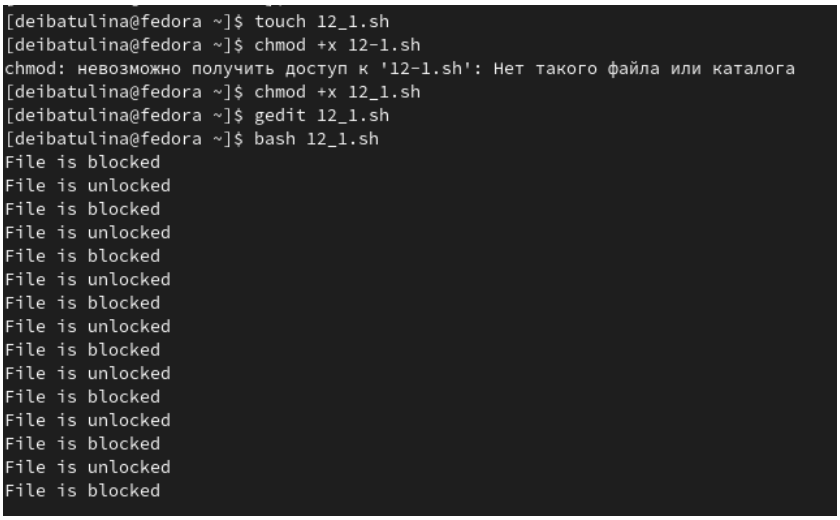

Рис. 4.2: Проверка работы первой программы

2. Реализуем команду man с помощью командного файла. Изучим содержимое каталога /usr/share/man/man1 (рис. 4.3). В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой less сразу же просмотрев содержимое справки. Командный файл (рис. 4.4) получает в виде аргумента командной строки название команды и в виде результата выдаёт справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

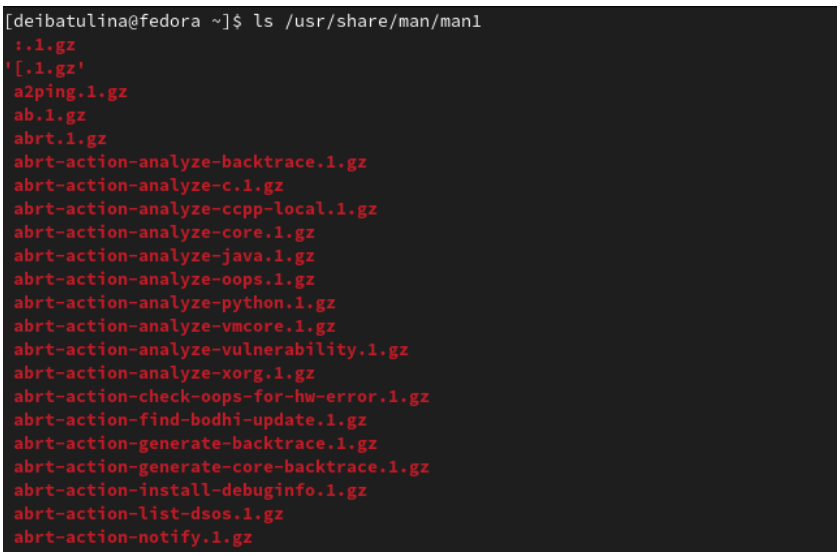

Рис. 4.3: Содержимое каталога /usr/share/man/man1

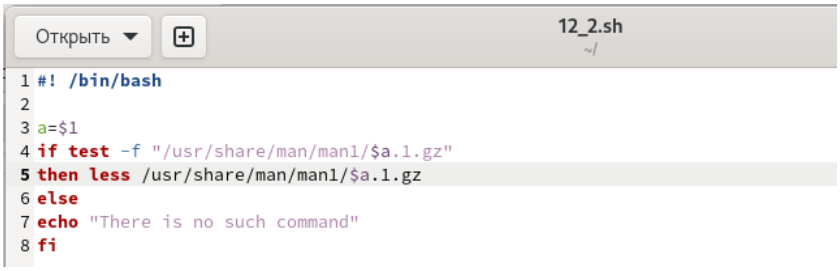

Рис. 4.4: Код для второй программы

Проверим работу данной программы (рис. 4.5).

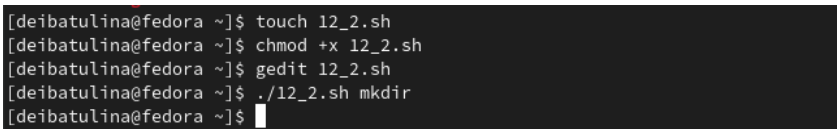

Рис. 4.5: Проверка работы второй программы

В результате мы получаем справку по команде mkdir, которую и запрашивали (рис. 4.6).

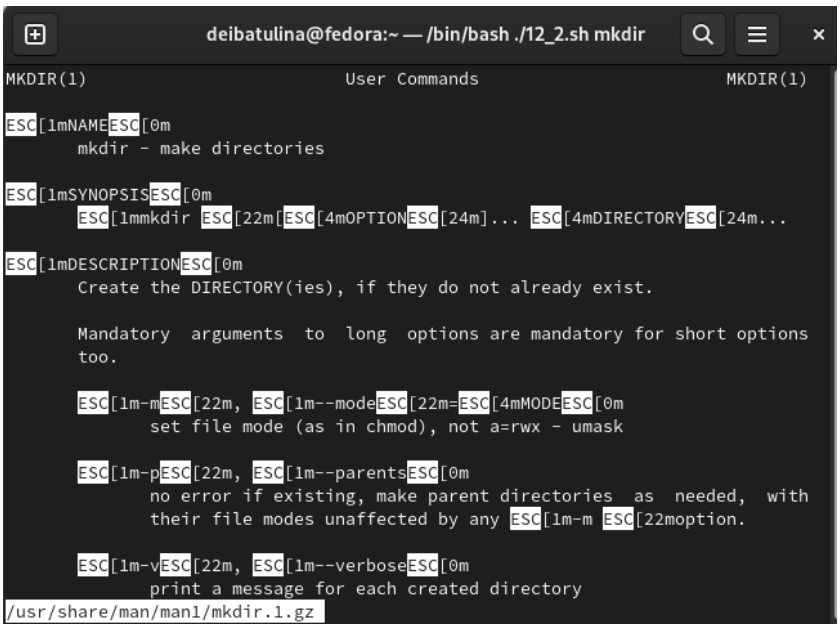

Рис. 4.6: Вывод программы - справка по заданной команде

3. Используя встроенную переменную $RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита (рис. 4.7). Учтём, что $RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

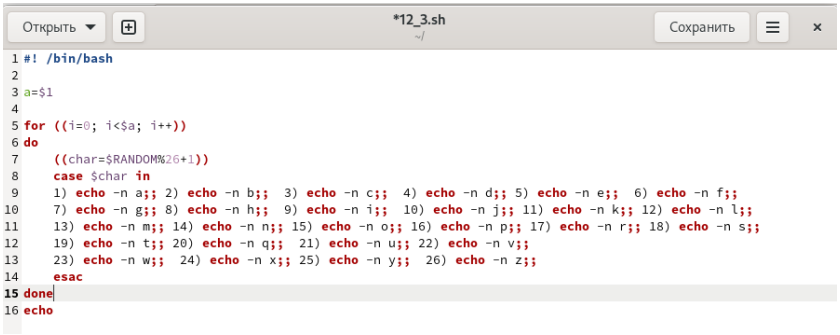

Рис. 4.7: Код для третьей программы

Проверим работу программы (рис. 4.8).

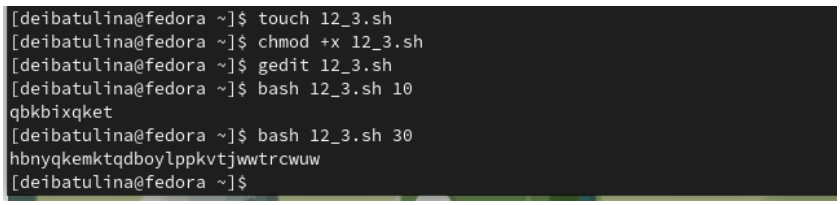

Рис. 4.8: Проверка работы третьей программы

5 Выводы

В ходе выполнения лабораторной работы я научилась писать более сложные командные файлы, используя логические управляющие конструкции и циклы.

6 Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: 1 while [$1 != “exit”]

В данной строчке допущены следующие ошибки: не хватает пробелов после первой скобки [и перед второй скобкой] выражение $1 необходимо взять в “”, потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так:

```
while [ “$1” != "exit" ]
```

2. Как объединить (конкатенацией) несколько строк в одну?

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: Первый:

```
VAR1="Hello,"
VAR2=" World" VAR3="$VAR1$VAR2" echo "$VAR3"
```

Результат:

```
Hello, World.
```

Второй:

```
VAR1="Hello, " VAR1+=" World" echo "$VAR1"
```

Результат:

```
Hello, World.
```

3. Найдите информацию об утилите seq. Какими иными способами можно реализовать её функционал при программировании на bash?

Команда seq в Linux используется для генерации чисел от ПЕРВОГО до ПОСЛЕДНЕГО шага INCREMENT. Параметры: seq LAST: если задан только один аргумент, он создает числа от 1 до LAST с шагом шага, равным 1. Если LAST меньше 1, значение is не выдает. seq FIRST LAST: когда заданы два аргумента, он генерирует числа от FIRST до LAST с шагом 1, равным 1. Если LAST меньше FIRST, он не выдает никаких выходных данных. seq FIRST INCREMENT LAST: когда заданы три аргумента, он генерирует числа от FIRST до LAST на шаге INCREMENT. Если LAST меньше, чем FIRST, он не производит вывод. seq -f «FORMAT» FIRST INCREMENT LAST: эта команда используется для генерации последовательности в форматированном виде. FIRST и INCREMENT являются необязательными. seq -s «STRING» ПЕРВЫЙ ВКЛЮЧЕНО: Эта команда используется для STRING для разделения чисел. По умолчанию это значение равно /n. FIRST и INCREMENT являются необязательными. seq -w FIRST INCREMENT LAST: эта команда используется для выравнивания ширины путем заполнения начальными нулями. FIRST и INCREMENT являются необязательными.

4. Какой результат даст вычисление выражения $((10/3))?

Результатом данного выражения $((10/3)) будет 3, потому что это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки zsh от bash.

Отличия командной оболочки zsh от bash: В zsh более быстрое автодополнение для cd с помощью Tab. В zsh существует калькулятор zcalc, способный выполнять вычисления внутри терминала. В zsh поддерживаются числа с плавающей запятой. В zsh поддерживаются структуры данных «хэш». В zsh поддерживается

раскрытие полного пути на основенеполных данных. В zsh поддерживается замена части пути. В zsh есть возможность отображать разделенный экран, такой же как разделенный экран vim.

6. Проверьте, верен ли синтаксис данной конструкции:

```
for ((a=1; a <= LIMIT; a++))
```

Синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать $ перед переменными ().

7. Сравните язык bash с какими-либо языками программирования. Какие преимущества у bash по сравнению с ними? Какие недостатки?

Преимущества и недостатки скриптового языка bash:

- Один из самых распространенных и ставится по умолчанию в большинстве дистрибутивах Linux, MacOS;
- Удобное перенаправление ввода/вывода;
- Большое количество команд для работы с файловыми системами Linux;
- Можно писать собственные скрипты, упрощающие работу в Linux недостатки скриптового языка bash;
- Дополнительные библиотеки других языков позволяют выполнить больше действий;
- Bash не является языков общего назначения;
- Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на быстроте выполнения этого скрипта;
- Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий.

Список литературы