

Отчёт по лабораторной работе №12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Митичкина Екатерина Павловна

Содержание

| | |
|--------------------------------------|---|
| Цель работы | 1 |
| Задача | 1 |
| Теоретическое введение: | 2 |
| Выполнение лабораторной работы | 2 |
| Задание 1 | 2 |
| Задание 2 | 4 |
| Задание 3 | 6 |
| Выводы | 7 |
| Ответы на контрольные вопросы | 7 |

Цель работы

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

Задача

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, 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.

Теоретическое введение:

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 оболочке 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.

Выполнение лабораторной работы

Задание 1

1. Я создала файла `prog1.sh`

```
epmitichkina@fedora:~  
[epmitichkina@fedora ~]$ touch prog1.sh  
[epmitichkina@fedora ~]$ emacs &  
[1] 2374
```

{#fig:001

width=70%}

2. Написала код в редакторе emacs

```
#!/bin/bash  
function waiting  
{  
    s1=$(date +%s)  
    s2=$(date +%s)  
    ((t=s2-s1))  
    while ((t<t1))  
    do  
        echo "Ожидание"  
        sleep 1  
        s2=$(date +%s)  
        ((t=s2-s1))  
    done  
}  
  
function doing  
{  
    s1=$(date +%s)  
    s2=$(date +%s)  
    ((t=s2-s1))  
    while ((t<t1))  
    do  
        echo "Выполнение"  
        sleep 1  
        s2=$(date +%s)  
        ((t=s2-s1))  
    done  
}
```

prog1.sh

```

done
}
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then
        echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then waiting
    fi
    if [ "$command" == "Выполнение" ]
    then doing
    fi
    echo "Следующее действие:"
    read command
done

```

prog1.sh

3. Предоставила право на выполнение и проверка файла

```

[epmitichkina@fedora ~]$ chmod +x prog1.sh
[epmitichkina@fedora ~]$ ./prog1.sh 3 5 Ожидание > /dev/tty1 &
[1] 2664
bash: /dev/tty1: Отказано в доступе
[1]+  Выход 1          ./prog1.sh 3 5 Ожидание > /dev/tty1
[epmitichkina@fedora ~]$ ./prog1.sh 3 5 Ожидание > /dev/tty2 &
[1] 2669
[epmitichkina@fedora ~]$ ./prog1.sh 3 5 Выполнение > /dev/tty2 &
[2] 2687

[1]+  Остановлен      ./prog1.sh 3 5 Ожидание > /dev/tty2
[epmitichkina@fedora ~]$

```

Предоставление прав и проверка

Задание 2

1. Я проверила содержимое `/usr/share/man/man1`

```
epmitichkina@fedora:/usr/share/man/man1
[epmitichkina@fedora man1]$ cd
[epmitichkina@fedora ~]$ cd /usr/share/man/man1
[epmitichkina@fedora man1]$ ls
.:.1.gz
' [.1.gz'
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz
```

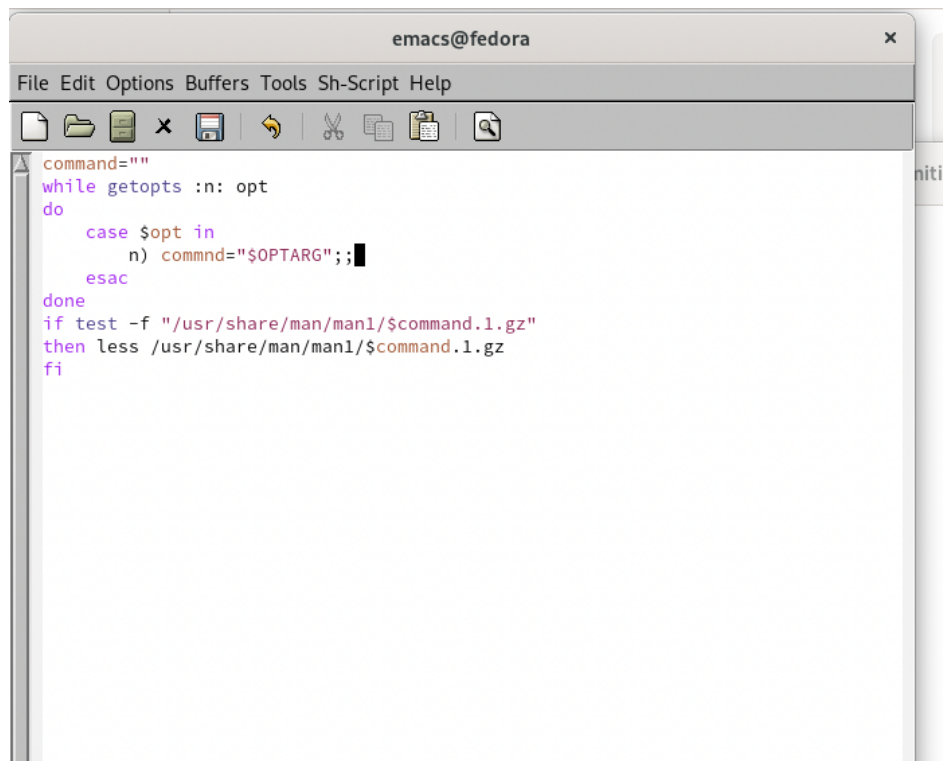
содержимое /usr/share/man/man1

2. Я создала файл prog2.sh

```
[epmitichkina@fedora ~]$ touch prog2.sh
[epmitichkina@fedora ~]$ emacs &
[2] 2876
[epmitichkina@fedora ~]$ touch prog2.sh
[1]-  Завершён      emacs
[2]+  Завершён      emacs
[epmitichkina@fedora ~]$ emacs &
[1] 2914
[epmitichkina@fedora ~]$ chmod +x prog2.sh
[epmitichkina@fedora ~]$ ./prog2.sh -n mc
[epmitichkina@fedora ~]$ ./prog2.sh -n rm
```

создание prog2.sh

2. Написала код в редакторе emacs



```
command=""
while getopts :n: opt
do
    case $opt in
        n) commnd="$OPTARG";;
    esac
done
if test -f "/usr/share/man/man1/$command.1.gz"
then less /usr/share/man/man1/$command.1.gz
fi
```

prog2.sh

3. Предоставила право на выполнение и проверка файла

```
[epmitichkina@fedora ~]$ chmod +x prog2.sh
[epmitichkina@fedora ~]$ ./prog2.sh -n mc
[epmitichkina@fedora ~]$ ./prog2.sh -n rm
[epmitichkina@fedora ~]$ ./prog2.sh -n vo
```

Предоставление прав и проверка

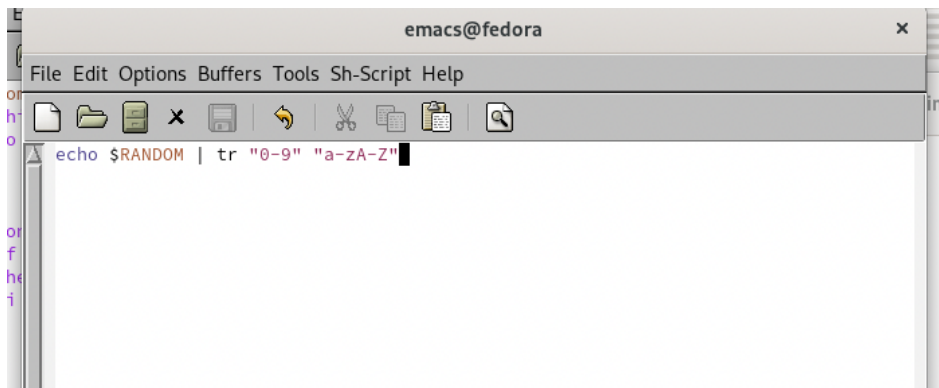
Задание 3

1. Я создала файл prog3.sh

```
[epmitichkina@fedora ~]$ ./prog2.sh -n shshgs
[epmitichkina@fedora ~]$ touch prog3.sh
[epmitichkina@fedora ~]$ emacs &
[2] 2987
```

создание prog3.sh

2. Написала код в редакторе emacs



prog3.sh

3. Предоставила право на выполнение и проверка файла

```
[epmitichkina@fedora ~]$ ./prog3.sh
bcbgi
[epmitichkina@fedora ~]$ ./prog3.sh
ccdig
[epmitichkina@fedora ~]$ ./prog3.sh
cgbbe
[epmitichkina@fedora ~]$
```

Предоставление прав и проверка

Выводы

В результате работы изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Ответы на контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке:
 - 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 (конкатенация) несколько строк в одну?

Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами: - Первый: VAR1="Hello," VAR2=" World" VAR3="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. Скажите 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 основе неполных 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 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 языком общего назначения
 - Утилиты, при выполнении скрипта, запускают свои процессы, которые, в свою очередь, отражаются на скорости выполнения этого скрипта
 - Скрипты, написанные на bash, нельзя запустить на других операционных системах без дополнительных действий