

Лабораторная работа №12

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Демидова Екатерина Алексеевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Контрольные вопросы	13
6	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Скрипт 1	9
4.2	Вывод скрипта 1	10
4.3	Скрипт 2	10
4.4	Вывод скрипта 2	11
4.5	Скрипт 3	11
4.6	Вывод скрипта 3	12

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 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

3 Теоретическое введение

При интерактивной работе с операционной системой пользователь постоянно сталкивается с необходимостью отдавать ей команды. В UNIX эту работу выполняет программа, которая называется командным процессором (shell). Иногда командный процессор называют шеллом или оболочкой, или интерпретатором команд (последнее неточно, потому что круг задач командного процессора шире, чем интерпретация команд).

Shell действует как посредник между вами и ядром операционной системы. Ядро – это часть операционной системы, которая всегда находится в памяти компьютера, это программа. Командный процессор преобразует ваши команды в инструкции для операционной системы, а операционная система превращает их в инструкции для аппаратных средств компьютера. По сути, именно оболочка придает определенную “персонализацию” системам UNIX.

Командный процессор выполняет в системе следующие задачи:

- интерпретация команд пользователя, в том числе разбор командной строки;
- запуск программ;
- организация перенаправлений потоков между процессами;
- интерпретация языка скриптов и их выполнение;
- управление заданиями;
- интерпретация шаблонов имен файлов;

- подстановка имен файлов в командную строку.

Кроме того, shell является мощным языком программирования.

Любая команда, являющаяся отдельной программой, т.е. не встроенной в интерпретатор, будет выполняться одинаково, независимо от shell'a. Например, если вы хотите что-то напечатать, команда печати всегда работает одинаково.

Некоторые команды встроены в shell, т.е. они являются частью программы оболочки и, как следствие, могут выполняться по-разному в зависимости от оболочки, в которой они запускаются. Есть три вида команд, встроенных в shell:

- общие команды запускаются несколько быстрее, так как они являются частью оболочки;
- команды адаптации позволяют адаптировать оболочку.
- команды программирования образуют язык программирования оболочки.

При смене shell'a вы не заметите никакой разницы между общими командами, которые встроены просто для повышения быстродействия. Однако команды адаптации и программирования изменяются.

Примером общих команд, встроенных в оболочку, служат команды `cd`, `echo`, `pwd`, `login`, `umask`.

Адаптация включает в себя создание новых команд или новых имен для старых команд и привлечение новых возможностей. Примером общепринятой адаптации служит изменение стимула (или приглашения системы).[1].

4 Выполнение лабораторной работы

1. Написать командный файл, реализующий упрощённый механизм семфоров. 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. (рис. 4.1, 4.2)



```
1 #!/bin/bash
2
3 let L=5
4 let N=20
5
6
7
8 echo "Процесс заблокирован"
9 while (test -f lockfile) && ((N<=1))
10 do
11     sleep 1
12     echo "${N}" сек."
13 done
14 touch lockfile
15 echo "Процесс блокирует ресурс"
16 while (test -f lockfile) && ((L<=1))
17 do
18     sleep 1
19     echo "До разблокировки еще: "${L}" сек."
20     echo "${L}" >> resurs.txt
21 done
22 rm lockfile
23 echo "Ресурс разблокирован, выход."
```

Рис. 4.1: Скрипт 1



```
eademidova@Demidova: ~  
eademidova@Demidova:~$ pgrep -f maxlma  
54326  
54334  
eademidova@Demidova:~$ bash senaphore.sh 543226  
Процесс заблокирован  
Процесс блокирует ресурс  
До разблокировки еще: 4 сек.  
До разблокировки еще: 3 сек.  
До разблокировки еще: 2 сек.  
До разблокировки еще: 1 сек.  
Ресурс разблокирован, выход.  
eademidova@Demidova:~$
```

Рис. 4.2: Вывод скрипта 1

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`..(рис. 4.3, 4.4)



```
man.sh  
1 #!/bin/bash  
2  
3  
4 if test $1  
5 then  
6 less /usr/share/man/man1/"$1".1.gz  
7 else  
8 echo "Команда не найдена"  
9 fi  
10
```

Рис. 4.3: Скрипт 2

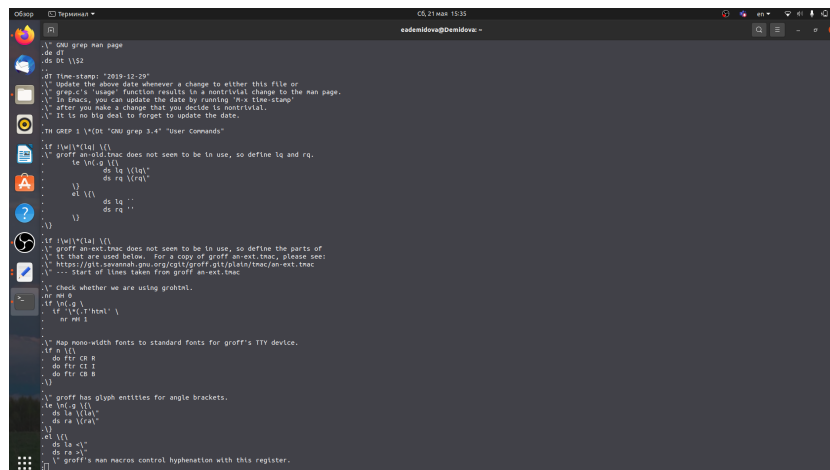


Рис. 4.4: Вывод скрипта 2

- 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 (рис. 4.5, 4.6)

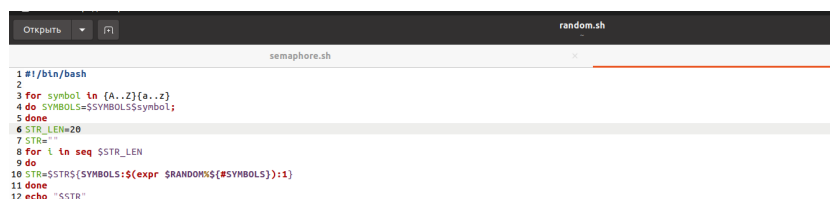
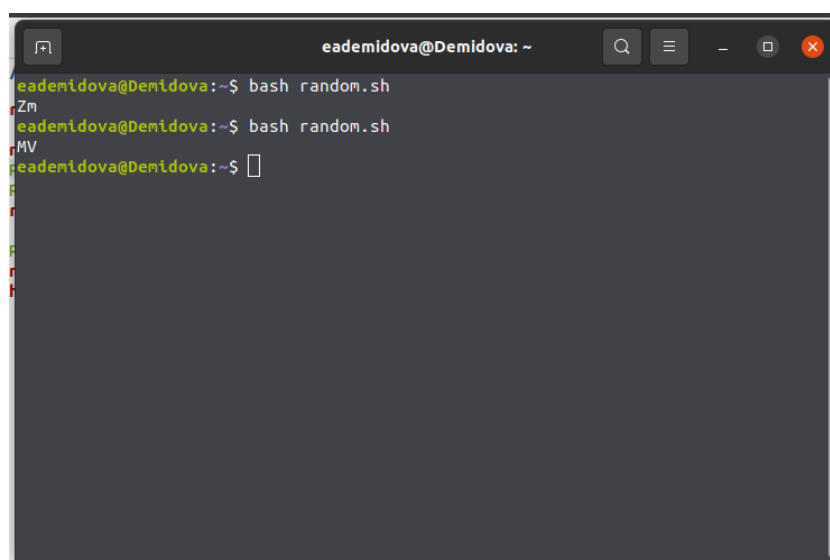


Рис. 4.5: Скрипт 3

A terminal window titled 'eademidova@Demidova: ~' with search, menu, and window control icons. The terminal shows the execution of a script named 'random.sh'. The first run outputs 'Zm' and the second run outputs 'MV'. The prompt 'eademidova@Demidova:~\$' is followed by a cursor.

```
eademidova@Demidova:~$ bash random.sh
Zm
eademidova@Demidova:~$ bash random.sh
MV
eademidova@Demidova:~$
```

Рис. 4.6: Вывод скрипта 3

5 Контрольные вопросы

1. В следующей строке кода синтаксическая ошибка состоит в том, что после первой и перед последней квадратными скобками пропущены пробелы.

```
while [$1 != "exit"]
```

2. Как объединить (конкатенация) несколько строк в одну? Самый простой способ объединить две или более строковые переменные – записать их одну за другой

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

```
Hello, World
```

Вы также можете объединить одну или несколько переменных с литеральными строками:

```
VAR1="Hello,"  
VAR2="${VAR1}World"  
echo "$VAR2"
```

```
Hello, World
```

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

Эта утилита выводит последовательность целых чисел с шагом, заданным пользователем. По-умолчанию, выводимые числа отделяются друг от друга символом перевода строки, однако, с помощью ключа -s может быть задан другой разделитель.

```
bash$ seq 5
```

```
1
2
3
4
5
```

```
bash$ seq -s : 5
```

```
1:2:3:4:5
```

Её функционал можно реализовать с помощью утилиты jot.

4. Вычисление выражения $\$((10/3))$ даст результат 3?
5. Укажите кратко основные отличия командной оболочки zsh от bash.
- Zsh более интерактивный и настраиваемый, чем Bash.
 - У Zsh есть поддержка с плавающей точкой, которой нет у Bash.
 - В Zsh поддерживаются структуры хеш-данных, которых нет в Bash.
 - Функции вызова в Bash лучше по сравнению с Zsh.
 - Внешний вид подсказки можно контролировать в Bash, тогда как Zsh настраивается.

- Конфигурационными файлами являются `.bashrc` в интерактивных оболочках без регистрации и `.profile` или `.bash_profile` в оболочках входа в Bash. В Zsh оболочками, не входящими в систему, являются `.zshrc`, а оболочками для входа - `.zprofile`.
- Массивы Zsh индексируются от 1 до длины, тогда как Bash индексируется от -1 до длины.
- В Zsh, если шаблоны не совпадают ни с одним файлом, выдается ошибка. Находясь в Баше, он остается без изменений.
- Правая часть конвейера запускается как родительская оболочка в Zsh, в то время как в Bash она запускается как подоболочка.
- В Zsh функция `zmv` используется для массового переименования, тогда как в Bash мы должны использовать функцию расширения параметров.
- Bash имеет хорошие возможности написания сценариев в одной строке, в то время как в Zsh мы не смогли найти то же самое.
- По умолчанию выходные данные хранятся во временном файле в Zsh, а в Bash - нет.
- Многие встроенные функции в Bash упрощают сложные программы, тогда как в Zsh встроенных функций для сложных программ меньше.
- Zsh эффективно управляет своими файлами, в то время как Bash плохо умеет работать с файлами.

6. Синтаксис данной конструкции верен.

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

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

Python — популярная альтернатива Bash для написания сценариев настройки среды, сборки и выпуска. Проект Electron использует Python для сценариев нескольких утилит. В Python нельзя выполнять команды напрямую, поскольку это не командный язык. Но запуск команд и перехват вывода реализуются

проще простого с помощью модуля `subprocess`. Используя встроенные функции Python, можно писать современные сложные Shell-сценарии. Но, в отличие от Bash, интерпретатор Python изначально не поддерживает выполнение процесса. Поэтому, если нужно упростить сценарий Python, чтобы он больше походил на Bash, используется такой инструмент, как `Shellpy`.

Стоит делать свой выбор между Bash, Python и JavaScript при написании Shell-сценариев, основываясь на следующих фактах и условиях:

Если вам нужно часто инициировать процессы и писать небольшой переносимый Shell-сценарий для Unix или Unix-подобных операционных систем, Bash, несомненно, будет хорошим выбором. Например, я написал сценарий Bash для сборки двоичных файлов проекта с открытым исходным кодом, основанного на пользовательской архитектуре. Если вам нужно написать кроссплатформенный Shell-скрипт для обработки некоторых данных и выполнения определенных команд, можете выбрать Python. Например, в проекте Electron есть несколько сценариев Python для обработки и загрузки файлов. Однако не стоит ожидать высокой производительности от Shell-сценариев на базе Python. JavaScript отлично подходит для тех же сценариев, что и Python. Однако, в отличие от Python, JavaScript имеет некоторые дополнительные преимущества. JavaScript быстр, изначально поддерживает JSON и имеет впечатляющие встроенные функции. Я нашел несколько сценариев Shell на базе Javascript в каталоге сценариев репозитория React Native.

6 Выводы

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

Список литературы

1. Командные процессоры ОС UNIX [Электронный ресурс]. life-prog.ru, 2014.
URL: https://life-prog.ru/1_54716_glava--komandnie-protssessori-os-UNIX.html.