Отчет по лабораторной работе № 13

Тема:

Программирование в командном процессоре ОС UNIX. Расширенное программирование.

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Довлетмурат Байрамгельдыев

Группа: НФИбд-03-20

Москва, 2021г.

Цель работы

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

Введение

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. • 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. Группу команд можно объединить в функцию.

Ход работы.

1. Создал текстовый файл с расширением .sh, после командой chmod разрешил выполнения файла.

```
[dowlet@dowlet ~]$ touch laba13.sh
[dowlet@dowlet ~]$ chmod +x laba13.sh
```

• Написал командный файл, реализующий упрощённый механизм семафоров. Командный файл ждет в течение некоторого времени t1, до освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовал его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом).

dowlet@dowlet:~

Файл Правка Вид Поиск Терминал Справка

```
#!/bin/bash
x="./x"
exec {fn}>$x
echo "блокировка"
until flock -n ${fn}
do
echo "не блокировать"
sleep 1
flock -n ${fn}
done
for((i = 0; i<=5;i++))
do
echo "работает"
sleep 1
done
```

• Запустил 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 в фоновом, а в привилегированном режиме. Доработал программу так, чтобы имелась возможность взаимодействия трёх и более процессов.

```
[dowlet@dowlet ~]$ bash laba13.sh
блокировка
работает
работает
работает
работает
работает
работает
```

2. Создал текстовый файл с расширением .sh, после командой chmod разрешил выполнения файла.

```
[dowlet@dowlet ~]$ touch laba13_2.sh
[dowlet@dowlet ~]$ chmod +x laba13_2.sh
```

• Реализовал команду man с помощью командного файла. Изучил содержимое каталога /usr/share/man/man1. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд.

dowlet@dowlet:~

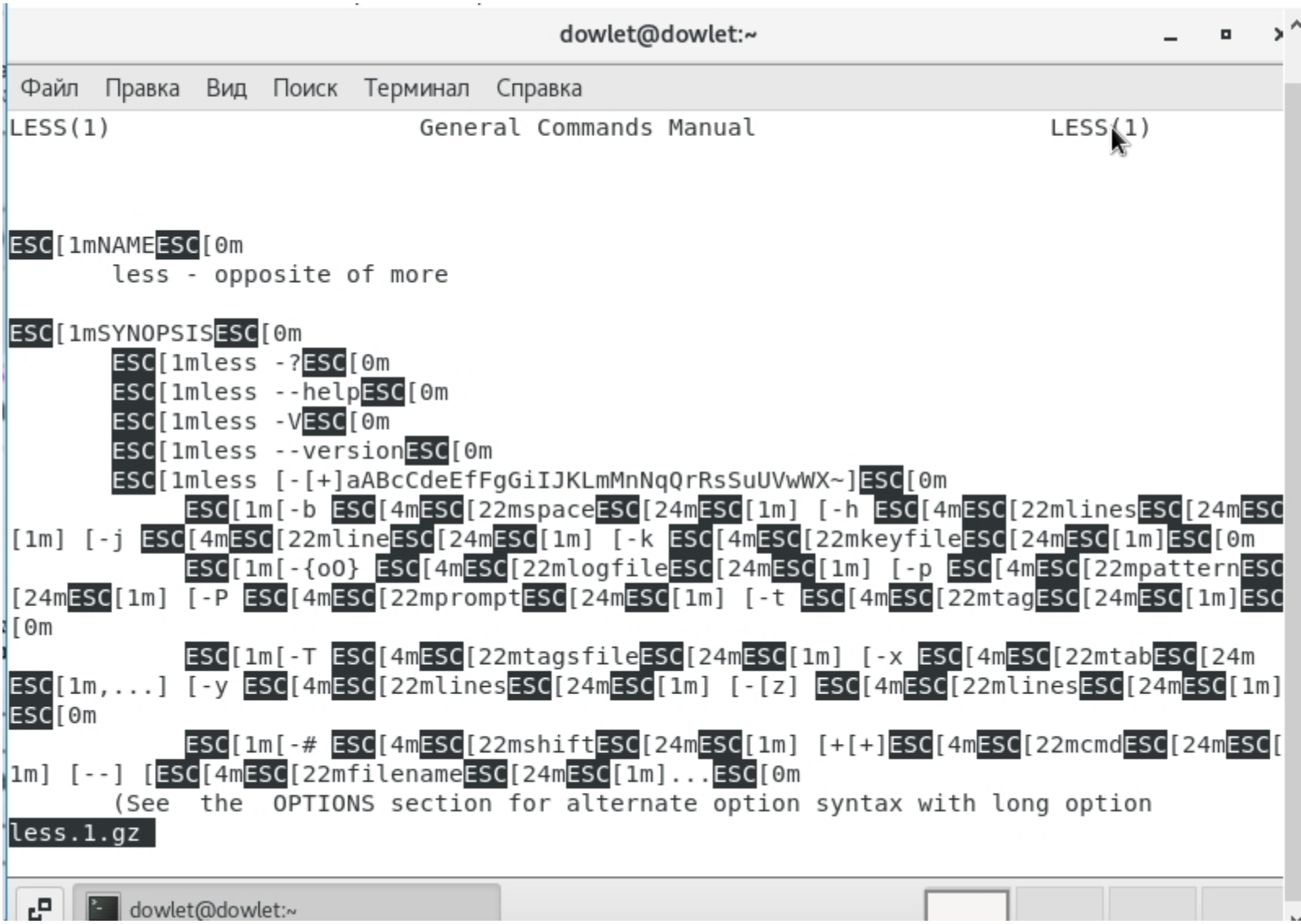
Файл Правка Вид Поиск Терминал Справка

```
#!/bin/bash
cd /usr/share/man/man1
if (test -f $1.1.gz)
then less $1.1.gz
else echo "Данной справки не существует"
fi
```

• Запустил командный файл. Командный файл получает в виде аргумента командной строки название команды и в виде результата выдавает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

```
[dowlet@dowlet ~]$ bash laba13_2.sh cd
[dowlet@dowlet ~]$ bash lab13_2.sh cpd
bash: lab13_2.sh: Нет такого файла или каталога
[dowlet@dowlet ~]$ bash laba13_2.sh cpd
Данной справки не существует
[dowlet@dowlet ~]$ bash laba13_2.sh less
[dowlet@dowlet ~]$
```

• Каждый архив можно открыть командой less сразу же просмотрев содержимое справки.



3. Создал текстовый файл с расширением .sh, после командой chmod разрешил выполнения файла.

```
[dowlet@dowlet ~]$ touch laba13_3.sh
[dowlet@dowlet ~]$ chmod +x laba13_3.sh
```

• Используя встроенную переменную $RANDOM, написал командный файл, генерирующий случайную последовательность букв латинского алфавита. Учел, что RANDOM выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

• Запустил командный файл. Как видим, вывел рандомные 10 слов, состоящих из рандомных букв латинского алфавита.

```
[dowlet@dowlet ~]$ bash laba13_3.sh
```

```
10 рандомных слов:
cbbeicbbbb
1
bbcicdbcdc
2
cgcccbchcc
3
dcebcchcic
4
bjgbbccchc
5
dbcccbbbcd
6
cbgjibbbbf
7
cedbchbdbf
8
bhccjihcdc
9
cccdcbbbbc
10
```

Вывод:

Изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Библиография.

https://dic.academic.ru/dic.nsf/eng_rus_technic/214092/%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D1%8B%D0%B9#:~:text=%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D1%8B%D0%B9%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%20%E2%80%94%20%D0%A7%D0%B0%D1%81%D1%82%D1%8C%20%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B,%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA%D0%B0%D1%8E%D1%89%D0%B0%D1%8F%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8%20%D0%B4%D0%BB%D1%8F%20%D0%B8%D1%85%20%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F.

http://bourabai.kz/os/#:~:text=%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0%2C%20%D0%9E%D0%A1%20%2D%20%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D1%82%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F%20%D1%81%D1%80%D0%B5%D0%B4%D0%B0,%D0%B8%20%D1%83%D0%B4%D0%BE%D0%B1%D1%81%D1%82%D0%B2%D0%B0%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B%20%D1%81%20%D0%BD%D0%B5%D0%B9.

Ответы на контрольные вопросы:

1. В строке while [$1 != "exit"] квадратные скобки надо заменить на круглые.
2. Есть несколько видов конкатенации строк. Например, VAR1="Hello," VAR2=" World" VAR3="$VAR1$VAR2" echo "$VAR3"
3. Команда seq выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. В bash можно использовать seq с циклом for, используя подстановку команд. Например, $ for i in $(seq 1 0.5 4) do echo "The number is $i" done
4. Результатом вычисления выражения $((10/3)) будет число 3.
5. Список того, что можно получить, используя Z Shell вместо Bash: Встроенная команда zmv поможет массово переименовать файлы/директории, например, чтобы добавить ‘.txt’ к имени каждого файла, запустите zmv –C '(*)(#q.)' '$1.txt'. Утилита zcalc — это замечательный калькулятор командной строки, удобный способ считать быстро, не покидая терминал. Команда zparseopts — это однострочник, который поможет разобрать сложные варианты, которые предоставляются скрипту. Команда autopushd позволяет делать popd после того, как с помощью cd, чтобы вернуться в предыдущую директорию. Поддержка чисел с плавающей точкой (коей Bash не содержит). Поддержка для структур данных «хэш». Есть также ряд особенностей, которые присутствуют только в Bash: Опция командной строки –norc, которая позволяет пользователю иметь дело с инициализацией командной строки, не читая файл .bashrc Использование опции –rcfile с bash позволяет исполнять команды из определённого файла. Отличные возможности вызова (набор опций для командной строки) Может быть вызвана командой sh Bash можно запустить в определённом режиме POSIX. Примените set –o posix, чтобы включить режим, или —posix при запуске. Можно управлять видом командной строки в Bash. Настройка переменной PROMPT_COMMAND с одним или более специальными символами настроит её за вас. Bash также можно включить в режиме ограниченной оболочки (с rbash или -restricted), это означает, что некоторые команды/действия больше не будут доступны: Настройка и удаление значений служебных переменных SHELL, PATH, ENV, BASH_ENV Перенаправление вывода с использованием операторов ‘>’, ‘>|’, ‘<>’, ‘>&’, ‘&>’, ‘>>’ Разбор значений SHELLOPTS из окружения оболочки при запуске Использование встроенного оператора exec, чтобы заменить оболочку другой командой
6. Синтаксис конструкции for ((a=1; a <= LIMIT; a++)) верен.
7. Язык bash и другие языки программирования: • Скорость работы программ на ассемблере может быть более 50% медленнее, чем программ на си/си++, скомпилированных с максимальной оптимизацией; • Скорость работы виртуальной ява-машины с байт-кодом часто превосходит скорость аппаратуры с кодами, получаемыми трансляторами с языков высокого уровня. Явамашина уступает по скорости только ассемблеру и лучшим оптимизирующим трансляторам; • Скорость компиляции и исполнения программ на яваскрипт в популярных браузерах лишь в 2-3 раза уступает лучшим трансляторам и превосходит даже некоторые качественные компиляторы, безусловно намного (более чем в 10 раз) обгоняя большинство трансляторов других языков сценариев и подобных им по скорости исполнения программ; • Скорость кодов, генерируемых компилятором языка си фирмы Intel, оказалась заметно меньшей, чем компилятора GNU и иногда LLVM; • Скорость ассемблерных кодов x86-64 может меньше, чем аналогичных кодов x86, примерно на 10%; •

Оптимизация кодов лучше работает на процессоре Intel; • Скорость исполнения на процессоре Intel была почти всегда выше, за исключением языков лисп, эрланг, аук (gawk, mawk) и бэш. Разница в скорости по бэш скорее всего вызвана разными настройками окружения на тестируемых системах, а не собственно транслятором или железом. Преимущество Intel особенно заметно на 32разрядных кодах; • Стек большинства тестируемых языков, в частности, ява и яваскрипт, поддерживают только очень ограниченное число рекурсивных вызовов. Некоторые трансляторы (gcc, icc, ...) позволяют увеличить размер стека изменением переменных среды исполнения или параметром; • В рассматриваемых версиях gawk, php, perl, bash реализован динамический стек, позволяющий использовать всю память компьютера. Но perl и, особенно, bash используют стек настолько экстенсивно, что 8-16 ГБ не хватает для расчета ack(5,2,3)