Лабораторная работа № 14. Программирование в командном процессоре ОС UNIX. Расширенное программирование

Отчёт

Сергеев Даниил Олегович

Содержание

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов. [1]

# 2 Задание

Написать командные файлы для задач лабораторной работы

# 3 Ход выполнения лабораторной работы

## 3.1 Выполнение упражнений с основными командами emacs

Создадим каталог lab14 с дополнительными директориями для каждого задания. Приступим к выполнению первой задачи.

Напишем командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, а дождавшись его освобождения использовать его в течение некоторого времени t2<>t1. Каждая смена состояния должна сопровождаться сообщением. Необходимо запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой, в котором также запущен этот файл, только в привилегированном режиме. Доработаем программу для взаимодействия трёх и более процессов. (рис. 1-2)

В качестве ресурса будем использовать файл ./tmp/resource. В него будет записываться номер PID. Этот файл будет создаваться после начала использования некоторым процессом. В случае освобождения ресурса файл будет удаляться.

**Листинг 3.1. – код программы командного файла первого задания**

semaphore=./tmp/resource  
wait\_time="$1"  
use\_time="$2"  
result=0  
function wait\_for\_resource {  
 if ((${wait\_time:-0}>0))  
 then   
 echo [PID $$] Waiting for response for $wait\_time seconds...  
 let local current\_time=0  
 while (($current\_time<$wait\_time))  
 do  
 echo [PID $$] $current\_time  
 if [ ! -f $semaphore ]  
 then  
 echo [PID $$} Resource found.  
 echo "$$" > $semaphore  
 result=1  
 return  
 fi  
 let current\_time+=1  
 sleep 1  
 done  
 echo [PID $$] Couldn\'t get resource for $wait\_time seconds...  
 return  
 fi  
 return  
}  
wait\_for\_resource  
if (($result))  
then  
 if ((${use\_time:-0}>0))  
 then  
 echo [PID $$] Using resource for $use\_time seconds...  
 sleep $use\_time  
 fi  
 if [ -f $semaphore ] && [ "$(cat $semaphore)" == "$$" ]  
 then  
 echo [PID $$] Releasing resource.  
 rm $semaphore  
 fi  
fi

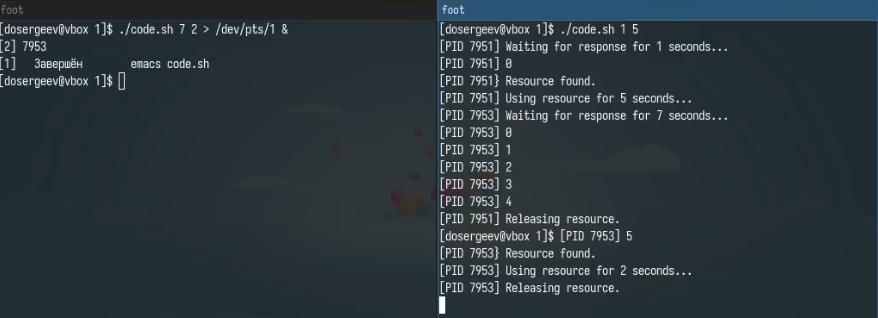


Рис. 1: Работа первого скрипта с двумя терминалами.

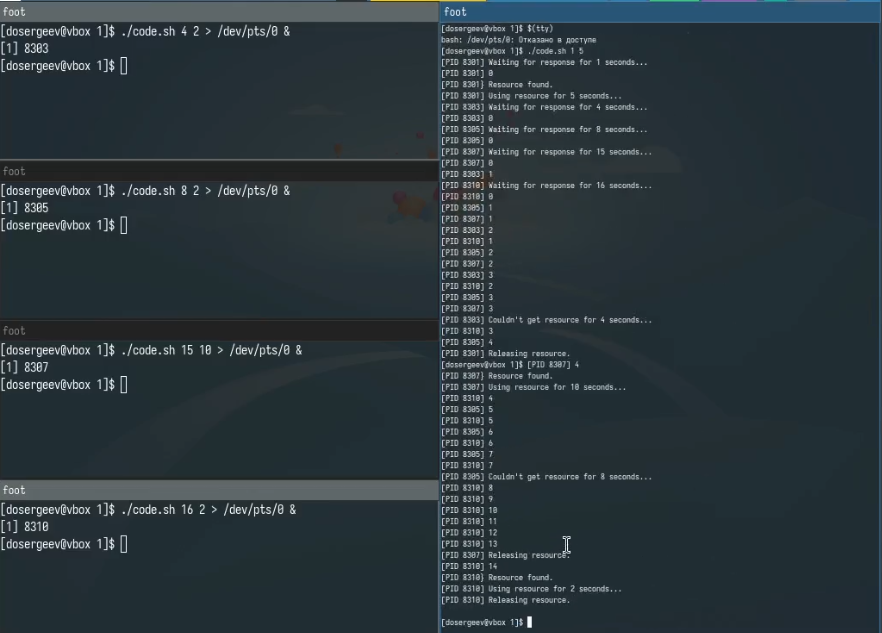


Рис. 2: Работа первого скрипта с пятью терминалами.

Теперь реализуем команду man с помощью командного файла. Используем команду less для чтения текстовых файлов, лежащих в архивах каталога /usr/share/man/man1. Название команды будет приниматься в качестве аргумента командной строки, а если команды нет, то будет выводиться сообщение об отсутствии справки. (рис. 3-4)

**Листинг 3.2. – код программы командного файла второго задания**

if (($#>0))  
then  
 manual=$(find /usr/share/man/man1/\* -name $1.\*)  
 if [ ${manual:-null} == "null" ]  
 then  
 echo Command not found  
 else  
 less -R /usr/share/man/man1/$1.\*  
 fi  
fi

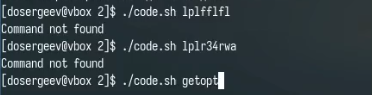


Рис. 3: Результат второго скрипта с неизвестной и известной командой.

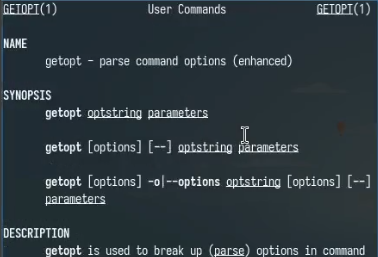


Рис. 4: Открытая страница справки.

Используя встроенную переменную $RANDOM, напишем командный файл, генерирующий случайную последовательность букв латинского алфавита. Создадим массив с всеми 52 буквами латинского алфавита (заглавными и строчными). С помощью переменной $RANDOM будем генерировать размерность последовательности и номер одной из 52 букв. (рис. 5)

**Листинг 3.3. – код программы командного файла третьего задания**

size=$((1 + RANDOM % 200))  
set -a sequence  
alphabet=({a..z} {A..Z})  
for ((i=0; i<size; i++))  
do  
 sym=$((0 + RANDOM % 52))  
 sequence[$i]=${alphabet[$sym]}  
done  
echo ${sequence[\*]}

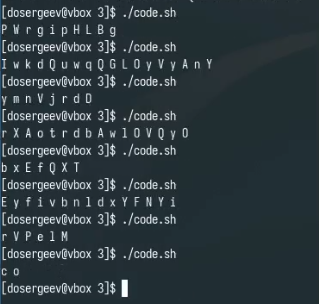


Рис. 5: Результат третьего скрипта с последовательностью до 20 символов.

## 3.2 Ответы на контрольные вопросы

1. Значения переменной $1 и строки “exit” написаны слитно с квадратными скобками, из-за чего программа неправильно воспринимает команды.
2. Объединить нескольско строк в одну можно с помощью оператора ‘+=’ или с помощью подстановки переменной ${}.

Например:

hello="Hello"  
world=" World!"  
hello+=$world  
  
#ИЛИ  
  
echo "${hello}${world}"

1. Утилита seq позволяет генерировать последовательности чисел. Её функционал можно реализовать с помощью фигурных скобок или оператора for.

Например:

echo {1..10}  
for ((i=1; i<=10; i++)); do echo \$i; done

1. Вычисление выражения $((10/3)) даст нам целую часть от деления 10 на 3.
2. Основные отличия командной оболочки Zsh от Bash:

* Существует возможность кастомизации;
* Имеет большое количество плагинов и тем;
* Имеет подсветку синтаксисов и авто-коррекцию;
* Имеет более удобную историю команд;

1. for ((a=1; a <= LIMIT; a++)) – синтаксис верный.
2. По сравнению с другими языками программирования bash имеет универсальный способ объявления переменных без указания типов данных (аналогично python). Он удобно читается и прост к освоению. В качестве минусов можно выделить нестандартный способ подстановки переменных и высокую чувствительность синтаксиса, как в случае с первым вопросом.

# 4 Вывод

В результате выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать более сложные командные файлы.

# Список литературы

1. Kulyabov. Лабораторная работа № 14. Программирование в командном процессоре ОС UNIX. Расширенное программирование. https://esystem.rudn.ru/pluginfile.php/2586593/mod\_resource/content/4/012-lab\_shell\_prog\_3.pdf; RUDN.