# Лабораторная работа №3

# Написание составных команд в командной оболочке bash

Команды сами по себе являются программами, скомпилированными в бинарные файлы, которые лежат в директории /bin/. Вызывая команду в терминале, вы фактически запускаете отдельный процесс. Каждый запущенный процесс в системе Linux имеет связанные с ним три информационных канала:

* Стандартный поток ввода stdin
* Стандартный поток вывода stdout
* Стандартный поток ошибок stderr

Каждый из этих каналов имеет свой файловый дескриптор. Файловый дескриптор – это индекс (целое неотрицательное число), указывающий на строку в специальной таблице, содержащую информацию об открытых процессом файлах. Стандартные потоки имеют фиксированные значения файловых дескрипторов: STDIN - 0, STDOUT - 1, STDERR – 2.

Кроме этого, каждый процесс связан со своим управляющим терминалом. Стандартные потоки процесса по умолчанию связаны с таким терминалом. Именно поэтому в окне нашего терминала отображается результат вывода команд. А также благодаря этой связи введенный текст при необходимости попадает на вход запущенных команд.

Эти три стандартных потока фактически выступают универсальным интерфейсом, с помощью которого мы можем взаимодействовать с процессом, связывать команды, выстраивать цепочки (пайплайны) или перенаправлять потоки в места отличные от терминала (файлы, сокеты и т.д.).

Кроме ключей, аргументов и стандартного потока ввода процесс может брать входную информацию из переменных окружения. Окружение – это не что иное как строки, содержащие пары «ключ=значение». Чаще всего их используют для конфигурирования, чтобы изменять и настраивать поведение тех или иных команд.

*Важно*

Далее по тексту, где указываются команды, которые необходимо вводить в терминал, знак **$** вводить не нужно. Этим знаком обозначается обычное приглашение консоли.

*Полезные ресурсы*

https://www.gnu.org/software/bash/manual/bash.html#Shell-Parameter-Expansion

https://www.cyberciti.biz/faq/linux-unix-bash-for-loop-one-line-command/

*Требования к отчету*

Сопроводите свою работу несколькими скриншотами. К отчету приложите файл record\_result\_lab\_3. Это текстовый файл с вашими командами. Ответьте на вопрос, удобно ли писать сложные команды на bash на ваш взгляд? В чем достоинства и недостатки?

## Ход работы

Подготовка

1. Запустите виртуальную машину и дождитесь загрузки ОС.
2. Откройте терминал. Это можно сделать с помощью основного меню или сочетанием клавиш **Ctrl+Alt+T**. Для запуска через основное меню нажмите на Menu -> System Tools -> Mate Terminal.
3. **Запустите запись терминала.**  **Это важно!**  Именно файл с записью истории ваших команд будет основной частью отчета. Для этого введите и запустите команду

$ script -a record\_result\_lab\_3

1. Убедитесь, что команда выполнилась без ошибок. Введите нижеуказанную команду и найдите имя файла record\_result\_lab\_3 в ее выводе:

$ ls -l

Перенаправление

1. Поместите на рабочий стол файл lab3.py, прилагаемый к этому руководству ( и все остальные файлы тоже lab3\_2.py, the-matrix-reloaded.csv, variants.txt).
2. Вернитесь в терминал, в котором уже идет запись (см. Подготовка).
3. Перейдите на рабочий стол. Для этого:

$ cd ./Desktop

в случае с русским языком нужно использовать экранирование пробела

$ cd ./Рабочий\ стол/

1. Выведите список всех файлов и каталогов. Убедитесь, что в выводе есть файл lab3.py

$ ls -l

1. Запустите генерацию вашего варианта. Вывод этой команды добавьте в отчет в виде скриншота. Для генерации:

$ python lab3.py

1. Снова запустите команду выше и перенаправьте стандартный поток вывод STDOUT в файл фамилия\_out. Для этого используйте символ перенаправления потока STDOUT «>»:

$ python lab3.py > фамилия\_out

1. Снова запустите команду выше и перенаправьте стандартный поток ошибок STDERR в файл фамилия\_err. Для этого используйте символ перенаправления потока «>» и номером файлового дескриптора, соответствующего этому потоку. Обратите внимание, что «>» не должен обрамляться пробелами:

$ python lab3.py 2>фамилия\_err

1. Откройте файлы и убедитесь, что там появился текст:

$ nano фамилия\_out

$ nano фамилия\_err

1. Две предыдущие команды можно записать в одну строку. Созданные файлы предварительно удалите командой rm.

$ python lab3.py>фамилия\_out 2>фамилия\_err

1. Проверьте, что файлы снова создались и опять наполнены текстом.
2. Как вы могли заметить, текст в этих файлах представляет собой диалог, но часть фраз находится в одном файле, а часть в другом. Давайте восстановим диалог. Для этого нам требуется перенаправить STDOUT и STDIN в один файл. Символ & позволяет понять командой оболочке, что 1 – это номер файлового дескриптора (STDOUT), а не имя файла. Для этого:

$ python lab3.py>фамилия\_final 2>&1

1. Теперь давайте посчитаем количество слов в каждом созданном файле. Для этого воспользуемся утилитой wc:

$ wc -w < имя\_файла

1. В отчет добавьте содержание полученных файлов и подсчитанное количество слов.

Пайплайн

1. Теперь научимся связывать стандартные потоки команд. Такая связь называется пайплайн. Вместе с этим файлом лабораторной работы для вас прикладываются файлы the-matrix-reloaded.csv, variants.txt, lab3\_2.py. Попробуем поискать в the-matrix-reloaded.csv некоторые строки по составным запросам. Для этого будем использовать утилиту grep. Изучите мануал по этой утилите.

$ man grep

$ grep pattern\_for\_search file.txt # Пример использования

1. Сгенерируйте свой вариант запустив скрипт lab3\_2.py. Рядом со скриптом должен лежать variants.txt.

$ python lab3\_2.py

1. В терминале вы увидите следующее, пример приведен на Рисунке 1:
2. Чьи фразы нужно найти  
   б) Кто должен упоминаться в этих фразах

в) Какой знак препинания должен содержаться

г) Какой объект, явление, событие, действие упоминается

От вас потребуется найти все строки для запроса «a», затем для «а и б», затем для «а, б и в» и так далее. Далее работа будет рассматриваться на примере задания с рисунка 1. Вам же потребуется сделать на примере своего варианта.

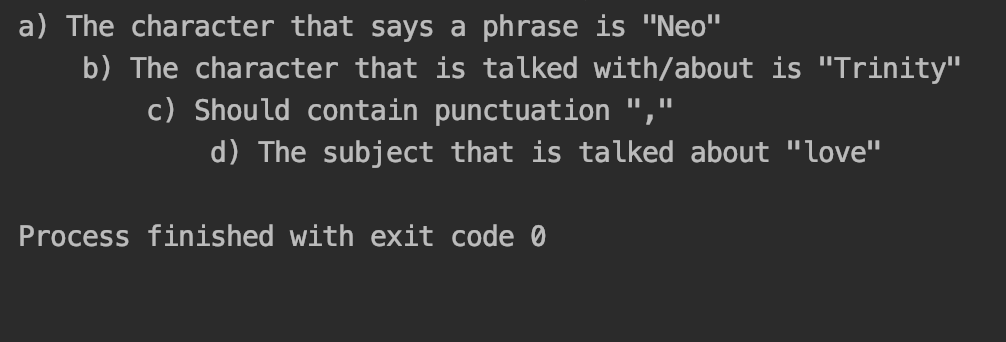


Рисунок 1 – Пример результата генерации задания для пайплайна

1. Используйте утилиту cat, чтобы вывести содержимое файла в консоль. Но вывод перенаправьте с терминала на утилиту grep. Для этого используется символ «|». Найдите строки, в которых говорит «Neo» (запрос «а» из задания), для этого выполните команду:

$ cat the-matrix-reloaded.csv | grep Neo

Можно заметить, что в выводе появляются фразы, где говорит не сам Neo, а где говорят о нём. Чтобы их убрать, давайте скажем grep, чтобы строка начиналась c Neo. Для этого используем символ «^»:

$ cat the-matrix-reloaded.csv | grep ^Neo

1. Теперь добавьте к этой команде еще поиск строк, в которых Neo упоминает Trinity (запрос «б» из задания):

$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity

Как можно увидеть из команды выше, мы можем использовать утилиту повторно и соединять результат первого использования утилиты grep с входом второго вызова утилиты grep. Это работает для всех утилит.

1. Теперь добавьте к этой команде еще поиск строк, в которых присутствует запятая (запрос «в» из задания).

$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity | grep ,

1. Теперь добавьте к этой команде еще поиск строк, в которых используется слово «love»:

$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity | grep , | grep love

1. В конце посчитаем количество полученных строк. Для этого воспользуемся утилитой wc (известной нам по предыдущей лабораторной работе).

$ cat the-matrix-reloaded.csv | grep ^Neo | grep Trinity | grep , | grep love | wc -l

1. А теперь давайте выведем количество строк без использования wc. Вы же прочитали man для grep? Найдите там необходимый ключ.

Переменные

Создание переменных может пригодиться при составлении сложных команд в терминале. Чтобы создать переменную достаточно сделать следующее:

$ MY\_VAR=3 # создали переменную равную 3

$ echo $MY\_VAR

Обратите внимание, что знак «=» не обрамляется пробельными символами. Далее по тексту $ в начале строки означает приглашение к вводу и не набирается, однако **$** **в середине команды** имеет значение «раскрытие значения переменной» и должен набираться в терминале.

1. Возьмите персонажа из предыдущего задания по пайплайну (в примере это Neo). Создадим переменную с именем актера.

$ name='Keanu Reeves'

1. Прочитайте мануал для команды sed. Это утилита является потоковым редактором текста.

$ man sed

1. Выведите все фразы заданного персонажа так, чтобы вначале вывелось «Text for Keanu Reeves». Для этого напишем следующую команду:

cat the-matrix-reloaded.csv | grep ^Neo | **sed "1i\ --Text for $name "**

1. Чтобы не листать вывод, выведем только первые 5 строк

$ cat the-matrix-reloaded.csv | grep ^Neo | sed "1i\ Text for $name" | **head -5**

1. Теперь заменим во всех фразах имя персонажа на имя актера и запишем в файл.

$ cat the-matrix-reloaded.csv | grep ^Neo | **sed "s/Neo,/$name:/"** > keanu\_text

1. Теперь пронумеруем строки в файле, для этого воспользуемся циклом while. Для этого напишем следующую команду

counter=1; while read line; do echo $counter $line; ((counter++)); done < keanu\_text

1. Рассмотрим команду подробнее. Вначале создается переменная счетчика

counter=1

1. Затем создается шапка цикла while. Цикл while имеет следующий синтаксис:

while [ condition ]; do <commands>; done

или

while <control-command>; do <COMMANDS>; done

1. В шапке цикла while используем команду read. Команда read считывает одну строку из стандартного потока ввода.

while read line;

1. Затем переходим к телу цикла. В теле цикла командой echo выводим значение переменных $counter $line.

echo $counter $line

1. Затем в теле цикла инкрементируем счетчик. Это арифметическое выражение. Для подстановки результата арифметического выражения используются двойные круглые скобки:   
     
   ((counter++))
2. Тело цикла закрывается ключевым словом done.
3. На вход всей этой сложной конструкции подаем вывод из файла keanu\_text с помощью конструкции:

(ваша длинная команда с циклом) <keanu\_text

1. Завершите запись команд, набрав в терминале:

$ exit