Лекция 4. Командная строка.

3 апреля 2025 г.

Содержание

1 Запуск программы с передачей аргументов

В Bash можно запускать программы и передавать им аргументы. Если введенная команда не является встроенной командой оболочки, интерпретатор ищет программу в системных путях и, если находит, запускает её, передавая переданные аргументы.

Если аргумент содержит пробелы, его нужно заключить в кавычки:

```
    ./args "a b c"
```

Можно использовать как одинарные, так и двойные кавычки. Внутри двойных кавычек разрешено использовать одинарные, и наоборот:

```
./args a b c (запускаем программу args с аргументами [a], [b], [c])
./args 'a "b c"' (запускаем программу args с аргументом [a "b c"])
./args "a 'b c'" (запускаем программу args с аргументом [a 'b c'])
```

Чтобы передать двойную кавычку внутри двойных кавычек, используют обратный слеш (\).

```
./args "a $\backslash$" b c" (запускаем программу args с аргументом [a "b c]).
```

Аналогичо можно сделать с одинарными кавычками.

Shell разбивает строку по пробелам, поэтому следующие конструкции эквивалентны:

```
./args "a b "c
./ar"g"s "a b c"
./args "a b c"
```

Передать null-терминатор (\0) в аргументе нельзя – строка обрежется по нему.

2 Просмотр содержимого файлов

В директории находится полное собрание сочинений Шекспира. Чтобы посмотреть содержимое файла, можно воспользоваться командой:

```
cat filename
```

Но это неудобный вариант, так как весь текс выводится сразу.

Для постраничного просмотра можно использовать команду:

```
more filename
```

Эта команда выводит текст по страницам, но прокручивать можно только вперед, что не всегда удобно.

Более удобный вариант — команда:

```
less filename
```

С её помощью можно прокручивать текст вверх и вниз.

3 wc

С помощью команды wc (wordcount) можно узнать количсетво строк, слов и байт:

```
wc filename
```

Вывод состоит из трёх чисел:

1. количество строк,

2. количество слов,

3. размер файла в байтах.

Интерпретатор командной строки воспринимает символы, начинающиеся с дефиса, как настройки.
Чтобы вывести только количество строк, используется опция -l:

```
wc -l filename
```

Другие опции команды wc можно посмотреть с помощью:

```
man wc
```

Если имя файла начинается с дефиса, то перед именем надо написать −−:

```
wc -- -filename
```

Если просто ввести wc в терминале без указания файла, программа перейдет в режим ожидания ввода. После ввода текста и нажатия Ctrl+D будет выведено количество строк, слов и байтов введённого текста.

4 Вывод команд в файлы

Чтобы записать результат wc filename1 в filename2, можно использовать такой синтаксис:

```
wc filename1 > filename2
```

Так можно делать со смногими другими командами.
Так же можно перенаправить на стандартный ввод:

```
wc < filename1
```

5 Передача вывода одной программы другой программе

Чтобы найти все строки, содержащие "Romeo используем команду:

```
grep Romeo filename
```

Допустим, мы хотим найти строки, где встречаются и "Romeo и "Juliet". Можно сделать это в два шага:

```
grep Romeo filename > result
grep Juliet result
```

Но можно обойтись без промежуточного файла, используя конвейер (pipe, |):

```
grep Romeo filename | grep Juliet
```

Чтобы узнать количество строк, содержащих оба слова, добавим команду wc -l:

```
grep Romeo filename | grep Juliet | wc -l
```

Рассмотрим команду, которая выдаст ошибку, так как файла filename2 не существует:

```
grep Romeo filename filename2
```

Для начала рассмотрим обозначения стандартных потоков:
Стандартный вход — 0
Стандартный выход — 1
Стандартный поток ошибок — 2
Если мы хотим, чтобы результат поиска записался в один файл, а ошибки — в другой, используем перенаправление:

```
grep Romeo filename filename2 >romeo.txt 2>errors.txt (результат grep направляем в
    romeo.txt, поток ошибок - в errors.txt)
```

Чтобы направить и стандартный вывод, и ошибки в один файл, используем 2> &1:

```
grep Romeo filename filename2 >romeo.txt 2>&1
```

Здесь 2> &1 означает:
"Перенаправить поток ошибок (2) в стандартный вывод (1)
А стандартный вывод уже был направлен в romeo.txt, значит, туда же попадут и ошибки.

6 Bash-скрипты

Чтобы не вводить одни и те же команды в терминале, можно сохранить их в файл с расширением .sh
Файл можно исполнить разными способами:

```
source filename.sh
```

Или:

```
. filename.sh
```

Но мы не можем запустить файл таким образом:

```
./filename.sh
```

так как у нас нет прав на его исполнение.
Но это можно исправить, разрешив выполнение файла следующей командой:

```
chmod +x filename.sh
./filename.sh
```

Так как shell бывает разной, в начале скрипта нужно явно указать, какой интерпретатор использовать. Это делается через #! в первой строке файла:

```
#!/bin/dash
```

В shell можно создавать переменные следующим образом:

```
first_char=Romeo
echo $first_char (смотрим содержимое переменной)
```

Чтобы "$first_char"воспринимался как текст или полное название файла, а не переменная, можно экранировать следующими образами:

```
echo '$first_char'
echo\$first_char
```

В этом варианте мы передаём аргументы внутри скрипта, передать какие-то другие аргументы нельзя:

```
#!/bin/dash
<filename.txt grep Romeo | grep Juliet | wc -l >united.txt
```

Чтобы сделать скрипт гибким и передавать параметры при запуске, нужно использовать специальные переменные:

```
#!/bin/dash
<filename.txt grep $1 | grep $2 | wc -l >united.txt
% $1 означает первый аргумент командной строки, $2 - второй аргумент командной строки
```

Запустить с параметрами можно так:

```
.filename.sh Romeo Juliet
```

Но к раскрытию переменных стоит относиться осторожно, например такой вызов вызовет ошибку:

```
.filename.sh "King Leer" the
```

Если в одну переменную может поступить несколько слов, то стоит писать вот так:

```
#!/bin/dash
<filename.txt grep "$1" | grep "$2" | wc -l >united.txt
```

7 Статусы завершения

Статус завершения последней программы можно посмотреть так:

```
echo $?
```

Напишем скрипт, чтобы определять существует ли персонаж в произведениях Шекспира:

```
#!/bin/bash

character="$1"
complete_works=100.txt.utf-8 % в каком файле смотрим

% написашем /dev/nell, так как нас не инересуют найденные строки
if grep "$character" $complete_works > /dev/null ; then
    echo "$character found"
else
    echo "$character not found"
fi
```