

# Задание практикума: командная оболочка операционной системы

Сальников А.Н.

## Содержание

1. Общее описание	1
2. Описание языка команд shell	2
3. Команды и запуск команд	3
4. Режим переднего плана и фоновый режим	4
5. Дополнительные требования	5
6. Рекомендации по написанию кода задания	7

## 1. Общее описание

Требуется написать программу на языке *Cu*, осуществляющую частичную эмуляцию командной оболочки (shell). Примеры программных оболочек: Windows Console (cmd)[1], Windows Power Shell[2], bash[7], tcsh[3], zsh[4].

Командная оболочка представляет собой интерактивную программу. Поток команд берется со стандартного потока ввода. Команды shell-у задаются пользователем через командную строку. Считывать команды нужно в цикле до тех пор, пока на очередной итерации не будет получен конец файла (детектируется при помощи константы EOF или проверкой соответствующих возвращаемых значений функции чтения из файлов/-потоков) либо не будет введена команда **exit**. Перед считыванием нужно вывести на экран приглашение к набору команд. Например приглашение может быть таким: “*vasia*” далее символ \$ и следующий за ним пробел). Длина считываемой строки ограничена только размером виртуальной памяти доступной процессу.

Пример приглашения ввода команд (здесь пользователь уже набрал некоторую команду):

```
vasia$ echo "We found: "; find /home -name \*${USER}\* 2>/dev/null | \
wc -l; echo " files"
```

Завершение shell происходит либо по закрытию стандартного потока ввода, либо в случае выполнения встроенной команды **exit** (Реализованы должны быть оба варианта).

Программа должна корректно обрабатывать все ошибки, выдавая осмысленные информационные сообщения о них в стандартный поток ошибок. В том числе обрабатывать ошибки неправильного синтаксиса командной строки.

## 2. Описание языка команд shell

Командная оболочка shell выполняет группу работ, где каждая работа отделяется от другой символами перевода строки (`'\n'` в языке *Cu*) и символом `','`. Внутри работы могут встречаться команды, их аргументы и перенаправления ввода/вывода, которые могут разделяться символами конвейера `'|'`.

В строке могут встречаться следующие конструкции:

- текст в кавычках (позволяют вставить пробел в аргумент программы). Кавычки бывают двух видов: двойные и одинарные. Отличие текста в двойных кавычках от текста в одинарных заключается в том, что в двойных кавычках производится подстановка переменных, а в одинарных нет.

Например:

```
echo "I am ${USER}" напечатает для пользователя с именем «vasia»
I am vasia
echo 'I am ${USER}' напечатает
I am ${USER}
```

- *эскапирование символа*. Осуществляется при помощи символа `'\'`. Символ после обратного слэша не будет иметь «служебного» смысла. Например обратный слэш позволяет поставить кавычку внутри кавычек. Последовательность `'\\'` позволяет ввести обычный одинарный слэш. Символ `'\'` может так же наоборот означать наличие специального смысла символа в ситуации, когда без этой конструкции особого смысла не было. Если символ `'\'` стоит в конце строки (следующий символ в файле перевод строки), то это означает, что перевод строки «съедается», а строка на новой строчке на самом деле является продолжением текущей.
- *Комментарии*. Если во входной строке встречается символ `'#'`, который находится не внутри кавычек одинарных или двойных и не

экранирован обратным слэшем, то все символы, стоящие после решётки до конца строки игнорируются.

- *Подстановка значений переменных.* Перед тем как интерпретировать команду необходимо подставить в командную строку значения переменных. Могут быть служебные переменные, а могут быть пользовательские переменные. Данное задание предполагает только подстановку служебных переменных. Значение переменной — это некоторый текст, который сопоставлен имени переменной. В случае встречи в строке конструкции вида:

`${ИМЯ_ПЕРЕМЕННОЙ}`

вместо этой конструкции, в строку, будет подставлена строка, являющаяся значением переменной.

### 3. Команды и запуск команд

Команды можно разделить на 2 группы. Первая – встроенные команды shell. Часто они не требуют запуска отдельных процессов (их реализация является частью кода shell<sup>1</sup>), например команда **cd** или **pwd**. Внешние команды shell — это обычные программы, которые запускаются так, что каждая программа оказывается в своём отдельном процессе. На встроенные команды не может быть вызван `exec`.

Команде можно указать список её аргументов. Например:

```
ls -l -a ../.. #конец аргументов
```

передаст программе `ls` после её запуска:

```
в argv[0] "ls",
в argv[1] "-l",
в argv[2] "-a",
в argv[3] "../..".
```

После списка аргументов программы допускается указывать символы `<` и `>`, которые позволяют считать стандартный ввод из файла или вывести стандартный вывод в файл, а также последовательность символов `>>`, что позволяет дописать вывод программы в конец указанного файла. Например:

```
# Перенаправим ввод команде cat
# из файла file.in и допишем
# файл file.out
cat < file.in >> file.out
```

---

<sup>1</sup>Однако они могут присутствовать в конвейере, тогда для них необходимо создавать отдельные процессы, перенаправлять ввод/вывод и т.п., но вместо вызова `exec`, нужно вызвать функцию в коде shell, реализующую встроенную команду.

После перенаправлений ввода/вывода допускается указать символ `&`, который означает запуск команды в фоновом режиме. Подробнее про фоновый режим далее.

При помощи символа `|` организуется конвейер. Смысл символа таков, что команда слева от символа делает свой стандартный поток вывода стандартным потоком ввода для команды справа от символа. И так продолжается дальше по цепочке, пока «вертикальные палки» не закончатся. В случае, если имело место перенаправление ввода/вывода, то приоритет отдаётся именно перенаправлению, а не конвейеру. В результате процесс подключённый к конвейеру справа получит себе конец файла, либо процесс слева будет некому «слушать» из конвейера и он вероятно получит себе *SIGPIPE*.

Символ `&` допускается указывать только в конце определения исполняемой работы `shell`. То есть до символа задающего конвейер `&` указать нельзя.

## 4. Режим переднего плана и фоновый режим

В режиме переднего плана (`foreground`) работы выполняются последовательно одна за другой. Каждая работа на время своего выполнения получает терминал в свой монопольный доступ. После исполнения одной или нескольких работ (в случае, если они разделены точкой с запятой) пользователю выдаётся стандартное приглашение ко вводу следующего набора команд. При этом, нажатие *Ctrl+c* должно приводить к послышке сигнала текущей выполняющейся работе, но не самому процессу реализующему `shell`. По нажатию *Ctrl+z* работа должна приостанавливаться, после чего выдаётся приглашение `shell`, которое позволяет запустить новую порцию команд или продолжить выполнение приостановленной ранее работы путём указания её номера во встроенной команде **fg** или **bg**. Список имеющихся работ можно посмотреть при помощи команды **jobs**.

Работа запущенная в фоновом режиме (`background`) не должна обращаться к терминалу. В случае обращения к терминалу процесс из работы запущенной в фоновом режиме должен быть приостановлен (не завершён). Если работа запускается в фоновом режиме, то процессы не должны получать *SIGINT* в случае нажатия *Ctrl+c* на клавиатуре. В случае запуска работы в фоновом режиме `shell` не вместо ожидания завершения такой работы, сразу либо выдаёт приглашение ко вводу следующего набора команд, либо выполняет следующую работу (например они были разделены точкой с запятой). В случае завершения фоновой работы `shell` информирует об этом пользователя, выдавая соответствующее сообщение в стандартный поток ошибок основным процессом `shell` (тот, который был изначально при запуске приложения).

## 5. Дополнительные требования

Требуется реализовать встроенные команды:

- **cd** – смена текущего каталога
- **pwd** – выдача текущего каталога в стандартный поток вывода
- **exit** – выход из shell
- **export** – передача переменных окружения в запускаемые процессы из текущего shell

Требуется реализовать как встроенные команды следующие внешние программы:

- **cat** – с именем **mcat**, где в качестве параметра либо ничего не указывается, либо указывается имя файла (полный набор параметров реализовывать не нужно).
- **sed** – с именем **msed**. Реализовать минимальный вариант подстановки по образцу, где в качестве шаблона используется просто текст без спецсимволов, которые позволяют задавать синтаксис регулярных выражений [8]. Первый аргумент задаёт строку образца, второй аргумент – строка на которую будет заменён образец. Необходимо осуществлять замену всех вхождений образца в строке. Символ '^' означает, что второй аргумент программы необходимо приписать в начало всех строк. Символ '\$' означает, что второй аргумент программы необходимо приписать в конец всех строк. Внутри строк, на которые заменяем может встречаться перевод строки, который задаётся '\n'.
- **grep** – с именем **mgrep** Реализовать минимальный вариант. Формат шаблона такой же как для предыдущей команды **msed**. Дополнительно возможны конструкции **.\*** – означает произвольное количество любых символов, в том числе пустое; и **.+** – означает произвольное количество любых символов, но не пустое.

Требуется реализовать подстановку любых переменных, которые перед запуском выставлены в переменных окружения, в том числе служебных:

- **\$цифра** – подстановка соответствующего аргумента командной строки самого shell.
- **\$#** – число параметров переданное shell

- `$?` – значение статуса последнего завершившегося процесса в последней выполнившейся работе переднего плана.
- `${USER}` – login пользователя.
- `${HOME}` – домашний каталог пользователя.
- `${SHELL}` – имя shell. Путь до того места в файловой системе, где находится исполняемый файл с ним. (В качестве плохо работающего «костыля» допускается реализация в виде подстановки `argv[0]`).
- `${UID}` – идентификатор пользователя
- `${PWD}` – текущий каталог
- `${PID}` – pid shell

О работе с переменными окружения можно прочитать здесь: [\[9\]](#).

## 6. Рекомендации по написанию кода задания

На начальном этапе при выполнении задания необходимо научиться считывать командные строки шелл и сохранять их в оперативной памяти (не выполняя команды). Рекомендуется данные в памяти хранить как массив структур следующего вида:

```
1 struct program
3 {
4     char* name;
5     int number_of_arguments;
6     char** arguments;
7     char *input_file , *output_file;
8     int output_type; /* 1 - rewrite , 2 - append */
9 };
11 struct job
12 {
13     int background;
14     struct program* programs;
15     int number_of_programs;
16 };
```

После того, как убедились, что разбор команд происходит правильно можно приступить к реализации запуска действий связанных с командами.

Для корректного отслеживания завершающихся процессов полезно определить действие на приход сигнала *SIGCHLD*.

Не стоит надеяться, что служебные переменные, такие как `${HOME}` будут для вас выставлены. Задача Shell как раз заключается в том, что Shell эти переменные определяет сам и выставляет для запускаемых из него программ.

## Список литературы

- [1] Официальная документация на команду `cmd` на сайте Microsoft: <https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/ntcmds.msp?mfr=true>.
- [2] Сайт проекта Microsoft Power Shell: <https://msdn.microsoft.com/en-us/powershell>.
- [3] Сайт tcsh: <http://www.tcsh.org/Welcome>.
- [4] Сайт одного из наиболее полного всем чем только можно шелла: <http://zsh.sourceforge.net/Doc/>.

- [5] Справочная страница с описанием реализации shell /bin/bash: “man bash”.
- [6] Перевод руководства по bash скриптам: [http://www.opennet.ru/docs/RUS/bash\\_scripting\\_guide](http://www.opennet.ru/docs/RUS/bash_scripting_guide).
- [7] Более полное руководство по bash <http://www.gnu.org/software/bash/manual/bashref.html>
- [8] Справочная страница по регулярным выражениям: “man 7 regex”
- [9] Справочная страница по функции getenv “man getenv”