Командная оболочка — это интерпретатор командной строки, который выполняет команды, вводимые пользователем. Мы вводим команду, она интерпретируется, выполняется а затем мы получаем результат ее выполнения. Оболочка обеспечивает традиционный интерфейс ввода команд, к которому мы привыкли. Это, как правило, черный экран и белый текст. Мы вводим команды в виде обычного текста, а также можем создавать скрипты из одной или нескольких команд.

Помимо стандартной оболочки bash существует множество других. Рассмотрим наиболее известные:

**Bourne Shell (sh)**. Свой подъем Unix начал с оболочкой Борна. Она была написана Стефаном Борном в Bell Labs и использовалась как оболочка по умолчанию в Unix версии 7 от 1979 года. Здесь уже было реализовано большое количество возможностей доступных в современных оболочках — дополнение имен файлов, автодополнение команд, стандартные переменные окружения и встроенные управляющие структуры. Bourne Shell называлась sh и размещалась в файловой системе Unix по адресу /bin/sh.

**C shell** (**csh**) — командная оболочка UNIX со встроенным скриптовым языком, разработанная Биллом Джоем. Эта оболочка очень распространена в системах BSD Linux. Здесь есть много интересных особенностей, в том числе контрольные структуры и грамматические выражения. Эта оболочка также впервые представила большое количество интересных функций, таких как история и механизмы редактирования, псевдонимы, CDPATH, управление задачами и хеширование, перенаправление вывода, присоединение, замена переменных, выполнение в фоне и т д.

**Korn shell (ksh)** - Написана Девидом Кроном и основана на исходниках оболочки Борна. KornShell (ksh) это оболочка, разработанная в Bell Labs еще в 1980. Она имеет обратную совместимость с Bourne Shell, а также включает многие черты оболочки С.

**Bourne-Again Shell (Bash)** - Написанная Браеном Фоксом в рамках проекта GNU как бесплатная и свободная замена для оболочки Борна. Bash -наиболее популярная и широко используемая из всех оболочек. Все дистрибутивы Linux поставляются по умолчанию с этой оболочкой. Она расширяет набор функций Bourne Shell. В большинстве систем Unix / Linux эта оболочка может быть найдена в файловой системе по адресу /bin/bash. Она была выпущена в 1989 году.  Благодаря такой популярности она была портирована на Windows и распространяется вместе с набором компиляторов Cygwin и MinGW. Также Bash используется в Android, для доступа к ней можно использовать различные эмуляторы терминала. Здесь поддерживается автодополнение, перенаправление ввода / вывода, дополнение команд, переменные и управляющие структуры для принятия решения (if-then-elese if) и циклы (loop).

**Z shell (zsh)** - Пол Фальстад написал первую версию командой оболочки zsh в 1990 году. Это командная оболочка Linux, которая может быть использована как интерактивная оболочка входа в систему, очень мощный интерпретатор команд. На самом деле Zsh это расширенная оболочка Борна с большим количеством улучшений, которая включает некоторые функции из Bash, KSH и Tcsh. Имя Zsh происходит от имени Йельского профессора Чжун Шао (Zhong Shao) так как Пол был студентом Принстонского университета. Поддерживаются такие интересные функции:

* Автозавершение строк
* Совместная история команд для всех сеансов оболочки
* Улучшена работа с переменными и массивами
* Редактирование нескольких строк в одном буфере
* Коррекция орфографии и много другое.

**Fish (Friendly Interactive Shell)** — это командная оболочка Linux нового поколения. Она разработана чтобы облегчить пользователю выполнение команд, есть подсветка синтаксиса, подсветка правильных адресов файлов, быстрый поиск по истории, веб-конфигуратор, а также особый синтаксис скриптов.

**Xiki shell (xsh)** - новая командная оболочка, созданная Крэйгом Мутом в 2012г. От остальных командных оболочек xsh отличается предоставлением графического интерфейса для навигации по файлам, работы с системами контроля версий, редактирования файлов и т. д.

**Основы скриптов**

Скрипт или как его еще называют — сценарий, это последовательность команд, которые по очереди считывает и выполняет программа-интерпретатор, в нашем случае это программа командной строки — bash.

**Скрипт** — это обычный текстовый файл, в котором перечислены обычные команды, которые мы привыкли вводить вручную, а также указанна программа, которая будет их выполнять. Загрузчик, который будет выполнять скрипт не умеет работать с переменными окружения, поэтому ему нужно передать точный путь к программе, которую нужно запустить. А дальше он уже передаст ваш скрипт этой программе и начнется выполнение.

Простейший пример скрипта для командной оболочки Bash:

***#!/bin/bash  
echo "Hello world"***

Первая строка особая, она задает программу, которая будет выполнять команды. Вообще говоря, мы можем создать скрипт на любом другом языке программирования и указать нужный интерпретатор, например, на python:

***#!/usr/bin/env python  
print("Hello world")***

Чтобы запустить скрипт можно использовать команду bash

***bash script.sh***

или дать скрипту права на исполнение (chmod +x script.sh) и запустить через полный или относительный путь:

***./script.sh***

**Переменные**

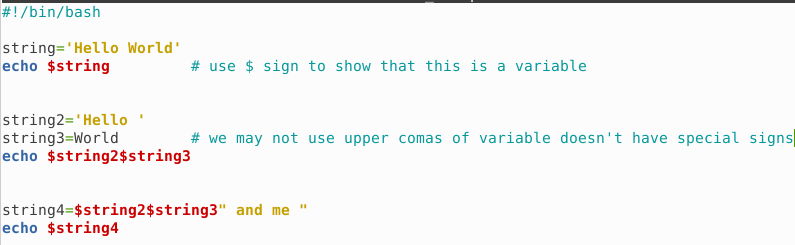
Например, объявим переменную string:

***string="Hello world"***

Значение нашей строки в кавычках. Но на самом деле кавычки не всегда нужны. Здесь сохраняется главный принцип bash — пробел — это специальный символ, разделитель, поэтому если не использовать кавычки world уже будет считаться отдельной командой, по той же причине мы не ставим пробелов перед и после знака равно.

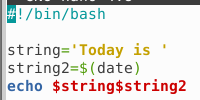
Чтобы вывести значение переменной используется символ $. Например:

***echo $string***



В переменные можно записывать результат выполнения утилит

Используется такой синтаксис **$(команда)**

******

******

**Параметры скрипта**

Не всегда можно создать bash скрипт, который не зависит от ввода пользователя. В большинстве случаев нужно спросить у пользователя какое действие предпринять или какой файл использовать. При вызове скрипта мы можем передавать ему параметры. Все эти параметры доступны в виде переменных с именами в виде номеров.









**Проверка условий if**

Создание bash скрипта было бы не настолько полезным без возможности анализировать определенные факторы, и выполнять в ответ на них нужные действия. Это довольно-таки сложная тема, но она очень важна для того, чтобы создать bash скрипт.

В Bash для проверки условий есть команда Синтаксис ее такой:

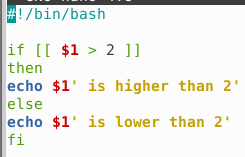
**if команда\_условие**  
**then**  
**команда**  
**else**  
**команда**  
**fi**

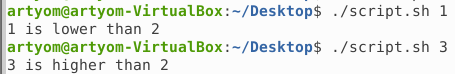
Эта команда проверяет код завершения команды условия, и если 0 (успех) то выполняет команду или несколько команд после слова then, если код завершения 1 выполняется блок else, fi означает завершение блока команд.

Но поскольку нам чаще всего нас интересует не код возврата команды, а сравнение строк и чисел, то была введена команда [[, которая позволяет выполнять различные сравнения и выдавать код возврата зависящий от результата сравнения. Ее синтаксис:

**[[ параметр1 оператор параметр2 ]]**

Для сравнения используются уже привычные нам операторы <,>,=,!= и т д. Если выражение верно, команда вернет 0, если нет — 1.





**Цикл for**

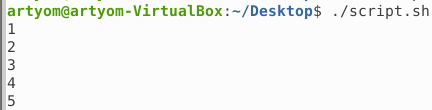
Преимущество программ в том, что мы можем в несколько строчек указать какие действия нужно выполнить несколько раз. Например, возможно написание скриптов на bash, которые состоят всего из нескольких строчек, а выполняются часами, анализируя параметры и выполняя нужные действия.

Первым рассмотрим цикл for. Вот его синтаксис:

**for переменная in список**  
**do**  
**команда**  
**done**

Перебирает весь список, и присваивает по очереди переменной значение из списка, после каждого присваивания выполняет команды, расположенные между do и done.

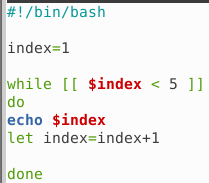


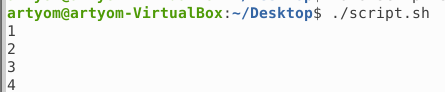


**Цикл while**

Второй цикл — это цикл while, он выполняется пока команда условия возвращает код 0, успех. Рассмотрим синтаксис:

**while команда условие**  
**do**  
**команда**  
**done**





Как видите, все выполняется, команда let просто выполняет указанную математическую операцию, в нашем случае увеличивает значение переменной на единицу.

Хотелось бы отметить еще кое-что. Такие конструкции, как while, for, if рассчитаны на запись в несколько строк, и если вы попытаетесь их записать в одну строку, то получите ошибку. Но тем не менее это возможно, для этого там, где должен быть перевод строки ставьте точку с комой «;». Например, предыдущий цикл можно было выполнить в виде одной строки:

index=1; while [[ $index < 5 ]]; do echo $index; let "index=index+1"; done;

**Переменные окружения**

Переменные окружения в Linux — это специальные переменные, определенные оболочкой и используемые программами во время выполнения. Они могут определяться системой и пользователем. Системные переменные окружения Linux определяются системой и используются программами системного уровня.

Например, команда PWD использует системную переменную, чтобы сохранять прежнюю рабочую директорию. Пользовательские переменные окружения устанавливаются пользователем, для текущей оболочки, временно или постоянно. Вся концепция добавления и удаления переменных оболочки крутится вокруг нескольких файлов, команд и различных оболочек.

Если смотреть более широко, переменная окружения может быть трех типов:

**1. Локальные переменные окружения**

Эти переменные определены только для текущей сессии. Они будут безвозвратно стерты после завершения сессии, будь то удаленный доступ или эмулятор терминала. Они не хранятся ни в каких файлах, а создаются и удаляются с помощью специальных команд.

**2. Пользовательские переменные оболочки**

Эти переменные оболочки в Linux определяются для конкретного пользователя и загружаются каждый раз когда он входит в систему при помощи локального терминала, или же подключается удаленно. Такие переменные, как правило, хранятся в файлах конфигурации: .bashrc, .bash\_profile, .bash\_login, .profile или в других файлах, размещенных в директории пользователя.

**3. Системные переменные окружения**

Эти переменные доступны во всей системе, для всех пользователей. Они загружаются при старте системы из системных файлов конфигурации:  /etc/environment, /etc/profile, /etc/profile.d/ /etc/bash.bashrc.

**Добавляем и удаляем переменные**

**Локальные переменные**

Локальные переменные окружения в Linux можно создавать следующими командами:

|  |
| --- |
| **var = значение** или **export var = значение** |

Для получения значения переменной используется символ $, например вывести на экран значение переменной var можно командой:

|  |
| --- |
| **echo $var** |

*Не забываем, что в Линуксе регистр имеет значение*

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**env**

Сама по себе команда env выводит список всех переменных среды и их значений

С опцией -i она позволяет временно удалить все переменные оболочки и выполнить одну команду с заданными переменными или вообще без них.

|  |
| --- |
| **env –i** [Var=Значение] **команда** |

**Var** — это любая переменная, которую вы хотите передать этой команде.

Такая команда запустит оболочку вообще без переменных окружения:

|  |
| --- |
| **env –i bash** |

После запуска такого окружения, не будет доступно никаких переменных, но после выхода все вернется на свои места.

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**unset**

Это другой способ удаления переменных окружения Linux. Unset удаляет переменную по имени до конца текущей сессии:

|  |
| --- |
| **unset имя\_переменной** |

**Установка и удаление пользовательских и системных переменных окружения**

Если нужно задать переменную на постоянной основе, то для этого придется редактировать файлы (см следующий [раздел](https://wiki.a1qa.com/pages/viewpage.action?pageId=383096085#id-%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%A4%D0%B0%D0%B9%D0%BB%D1%8B%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D1%85%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B))

Открываем файл, например, .bashrc и добавляем в него строку

**export var = значение**

После этого сохраняемся и перечитываем настройки из файла командой

|  |
| --- |
| **source имя\_файла** |

Чтобы удалить пользовательскую или системную переменную, соответственно удаляем определенную строку из нужного файла.

**Файлы переменных окружения**

**~/.bashrc**

Это файл переменных конкретного пользователя. Загружается каждый раз, когда пользователь создает терминальный сеанс, то есть проще говоря, открывает новый терминал. Все переменные окружения, созданные в этом файле вступают в силу каждый раз когда началась новая терминальная сессия.

**~/.bash\_profile**

Эти переменные вступают в силу каждый раз когда пользователь подключается удаленно по SSH. Если этот файл отсутствует система будет искать .bash\_login или .profile.

**/etc/environment**

Этот файл для создания, редактирования и удаления каких-либо переменных окружения на системном уровне. Переменные окружения, созданные в этом файле доступны для всей системы, для каждого пользователя и даже при удаленном подключении.

**/etc/bash.bashrc**

Системный bashrc. Этот файл выполняется для каждого пользователя, каждый раз когда он создает новую терминальную сессию. Это работает только для локальных пользователей, при подключении через интернет, такие переменные не будут видны.

**/etc/profile**

Системный файл profile. Все переменные из этого файла, доступны любому пользователю в системе, только если он вошел удаленно. Но они не будут доступны, при создании локальной терминальной сессии, то есть если вы просто откроете терминал.

Все переменные окружения Linux созданные с помощью этих файлов, могут быть удаленны всего лишь удалением их оттуда. Только после каждого изменения, нужно либо выйти и зайти в систему, либо выполнить эту команду:

|  |
| --- |
| source имя\_файла |

Основные системные переменные

| **Имя** | **Что значит** |
| --- | --- |
| **Имя** | **Что значит** |
| $HOME | Переменная содержит путь к домашнему каталогу текущего пользователя. |
| $USER | Имя текущего пользователя |
| $PATH | Список каталогов для поиска оболочкой исполняемых программ. |
| $PWD | Путь к текущему рабочему каталогу (либо *pwd*). Пример: dir\_path=$(pwd) |
| $OLDPWD | Предыдущий текущий каталог |
| $SHELL | Интерпретатор по умолчанию |
| $RANDOM | Генерирует рандомное число 0..32767 при каждом доступе к переменной. |
| $SECONDS | Время в сек. с момента запуска командной оболочки. |
| $? | Результат выполнения предыдущей команды. |
| $HOSTNAME | Имя компьютера |
| $HISTFILE | Путь к файлу истории интерпретатора |
| $IFS | Список символов-разделителей команд, параметров, элементов массива (по умолчанию: пробел, таб, новая строка) |
| $PS1 | Шаблон строки приглашения интерпретатора. |
| $TERM | тип терминала |
| $MANPATH | место поиска файлов man-страниц (каталоги, разделенные двоеточием) |
| $DISPLAY | текущий дисплей в X11 |
| $LANG | язык и кодировка пользователя (иногда LANGUAGE) |
| $EDITOR | консольный текстовый редактор по умолчанию (например nano, vim) |
| $PAGER | Программа для постраничного просмотра по умолчанию (less, more) |