**Лабораторная работа №6. Программирование в Shell**

# ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью лабораторной работы является получение практических навыков по написанию Shell-сценариев для ОС Linux.

Задачи:

1. Самостоятельно изучить синтаксис и важнейшие структуры Shell-сценариев.

2. Научиться применять Shell-сценарии для администрирования системы.

3. Закрепить полученные в ходе выполнения лабораторной работы навыки.

Результатами работы являются:

* исполняемый файл, содержащий Shell-сценарии, обеспечивающие заданную согласно индивидуальному заданию функциональность при запуске его в ОС Linux;
* отчет о выполнении домашней работы, содержащий материалы, перечасленные в методических указаниях и оформленный согласно требованиям.

# КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Знание языка командной оболочки является залогом успешного решения задач администрирования системы. Во время загрузки Linux выполняется целый ряд сценариев из файла /etc/rc.d, которые настраивают конфигурацию операционной системы и запускают различные сервисы, поэтому очень важно четко понимать эти скрипты и иметь достаточно знаний, чтобы вносить в них какие-либо изменения.

Язык сценариев легок в изучении, в нем не так много специфических операторов и конструкций. Синтаксис языка достаточно прост и прямолинеен, он очень напоминает команды, которые приходится вводить в командной строке. Короткие скрипты практически не нуждаются в отладке, а отладка даже больших скриптов отнимает весьма незначительное время.

Используемые в сценариях служебные символы и конструкции:

Комментарии.

Строки, начинающиеся с символа # (за исключением комбинации #!) - являются комментариями.

# Эта строка - комментарий.

Комментарии могут располагаться и в конце строки с исполняемым кодом.

echo "Далее следует комментарий." # Это комментарий.

Комментариям могут предшествовать пробелы (пробел, табуляция).

Разделитель команд.

[Точка-с-запятой]

Позволяет записывать две и более команд в одной строке.

echo hello; echo there

Ограничитель в операторе выбора case.

[Двойная-точка-с-запятой]

case "$variable" in

abc) echo "$variable = abc" ;;

xyz) echo "$variable = xyz" ;;

esac

Двойные кавычки.

В строке "STRING", ограниченной двойными кавычками не выполняется интерпретация большинства служебных символов, которые могут находиться в строке.

Одинарные кавычки.

'STRING' экранирует все служебные символы в строке STRING. Это более строгая форма экранирования.

Запятая.

Оператор запятая используется для вычисления серии арифметических выражений. Вычисляются все выражения, но возвращается результат последнего выражения.

let "t2 = ((a = 9, 15 / 3))" # Присваивает значение переменной "a" и вычисляет "t2".

Переменные.

Переменные - это одна из основ любого языка программирования. Они участвуют в арифметических операциях, в синтаксическом анализе строк и совершенно необходимы для абстрагирования каких либо величин с помощью символических имен. Физически переменные представляют собой ни что иное как участки памяти, в которые записана некоторая информация.

Когда интерпретатор встречает в тексте сценария имя переменной, то он вместо него подставляет значение этой переменной. Поэтому ссылки на переменные называются подстановкой переменных.

Необходимо всегда помнить о различиях между именем переменной и ее значением. Если variable1 - это имя переменной, то $variable1 - это ссылка на ее значение. "Чистые" имена переменных, без префикса $, могут использоваться только при объявлении переменный, при присваивании переменной некоторого значения, при удалении (сбросе), при экспорте и в особых случаях - когда переменная представляет собой название сигнала (см. Пример 29-5). Присваивание может производиться с помощью символа = (например: var1=27), инструкцией read и в заголовке цикла (for var2 in 1 2 3).

Заключение ссылки на переменную в двойные кавычки (" ") никак не сказывается на работе механизма подстановки. Этот случай называется "частичные кавычки", иногда можно встретить название "нестрогие кавычки". Одиночные кавычки (' ') заставляют интерпретатор воспринимать ссылку на переменную как простой набор символов, потому в одинарных кавычках операции подстановки не производятся. Этот случай называется "полные", или "строгие" кавычки.

Примечательно, что написание $variable фактически является упрощенной формой написания ${variable}. Более строгая форма записи ${variable} может с успехом использоваться в тех случаях, когда применение упрощенной формы записи порождает сообщения о синтаксических ошибках.

Присваивание значений переменным.

= - оператор присваивания (пробельные символы до и после оператора - недопустимы)

Проверка условий.

Практически любой язык программирования включает в себя условные операторы, предназначенные для проверки условий, чтобы выбрать тот или иной путь развития событий в зависимости от этих условий. В Bash, для проверки условий, имеется команда test, различного вида скобочные операторы и условный оператор if/then.

Оператор if/then проверяет, - является ли код завершения списка команд 0 (поскольку 0 означает "успех"), и если это так, то выполняет одну, или более, команд, следующие за словом then.

Существует специальная команда - [ (левая квадратная скобка). Она является синонимом команды test, и является встроенной командой (т.е. более эффективной, в смысле производительности). Эта команда воспринимает свои аргументы как выражение сравнения или как файловую проверку и возвращает код завершения в соответствии с результатами проверки (0 - истина, 1 - ложь).

Начиная с версии 2.02, Bash предоставляет в распоряжение программиста конструкцию [[ ... ]] расширенный вариант команды test, которая выполняет сравнение способом более знакомым программистам, пишущим на других языках программирования. Обратите внимание: [[ - это зарезервированное слово, а не команда.

Bash исполняет [[ $a -lt $b ]] как один элемент, который имеет код возврата.

Круглые скобки (( ... )) и предложение let ... так же возвращают код 0, если результатом арифметического выражения является ненулевое значение. Таким образом, арифметические выражения могут учавствовать в операциях сравнения.

if echo "Следующий \*if\* находится внутри первого \*if\*."

if [[ $comparison = "integer" ]]

then (( a < b ))

else

[[ $a < $b ]]

fi

then

echo '$a меньше $b'

fi

Циклы и ветвления.

Управление ходом исполнения - один из ключевых моментов структурной организации сценариев на языке командной оболочки. Циклы и преходы являются теми инструментальными средствами, которые обеспечивают управление порядком исполнения команд.

Цикл - это блок команд, который исполняется многократно до тех пор, пока не будет выполнено условие выхода из цикла.

**for (in)**

Это одна из основных разновидностей циклов. И она значительно отличается от аналога в языке C.

for arg in [list]

do

команда(ы)...

done

**while**

Оператор while проверяет условие перед началом каждой итерации и если условие истинно (если код возврата равен 0), то управление передается в тело цикла. В отличие от циклов for, циклы while используются в тех случаях, когда количество итераций заранее не известно.

while [condition]

do

command...

done

Как и в случае с циклами for/in, при размещении ключевого слова do в одной строке с объявлением цикла, необходимо вставлять символ ";" перед do.

while [condition] ; do

**until**

Оператор цикла until проверяет условие в начале каждой итерации, но в отличие от while итерация возможна только в том случае, если условие ложно.

until [condition-is-true]

do

command...

done

Обратите внимание: оператор until проверяет условие завершения цикла ПЕРЕД очередной итерацией, а не после, как это принято в некоторых языках программирования.

Как и в случае с циклами for/in, при размещении ключевого слова do в одной строке с объявлением цикла, необходимо вставлять символ ";" перед do.

until [condition-is-true] ; do

Операторы выбора

Инструкции case и select технически не являются циклами, поскольку не предусматривают многократное исполнение блока кода. Однако, они, как и циклы, управляют ходом исполнения программы, в зависимости от начальных или конечных условий.

**case (in) / esac**

Конструкция case эквивалентна конструкции switch в языке C/C++. Она позволяет выполнять тот или иной участок кода, в зависимости от результатов проверки условий. Она является, своего рода, краткой формой записи большого количества операторов if/then/else и может быть неплохим инструментом при создании разного рода меню.

case "$variable" in

"$condition1" )

command...

;;

"$condition2" )

command...

;;

esac

# ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

*Порядок выполнения:*

1. Изучить краткий [теоретический материал](#z1) и составить алгоритм работы сценария, согласно варианту индивидуального задания, полученному у преподавателя.

2. Создать текстовый файл и разместить в нем сценарий, обеспечивающий требуемый функционал. При необходимости воспользоваться [литературой](#z2), перечисленной в данных указаниях и материалами лабораторного практикума.

3. Разрешить исполнение созданного файла, изменив его атрибуты. Протестировать работоспособность и наличие требуемой функциональности сценария.

4. Оформить отчет.

5. Защитить выполненную работу у преподавателя.

# ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Отчет должен содержать:

* [титульный лист](#z3);
* цели и задачи выполняемой работы;
* текст задания;
* алгоритмическую схему работы сценария;
* листинг разработанного Shell-сценария;
* описание используемых операторов, команд и их параметров;
* результаты тестирования его работы, демонстрирующие соответствие требуемому в задинии функционалу;
* выводы.

**Типовые варианты заданий**

1. Написать скрипт для изменения имен файлов в текущем или выбранном пользователем каталоге с верхнего регистра на нижний и наоборот.
2. Написать скрипт для преобразования десятичных чисел в римскую систему счисления. Результат выдать на экран и сохранить в файл.
3. Написать скрипт для поиска файла программы по идентификатору процесса. Входные и выходные данные программы послать письмом администратору.
4. Написать скрипт для вывода содержимого файла. Имя файла пользователь вводит с клавиатуры.
5. Написать скрипт для удаления из текущего каталога всех файлов размером меньше 1К и послать письмо администратору с именами удаленных файлов.

# ЛИТЕРАТУРА

1. Гончарук С.В. Администрирование ОС Linux [Электронный ресурс]/ Гончарук С.В.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 164 c.— Режим доступа: <http://www.iprbookshop.ru/52142>.— ЭБС «IPRbooks», по паролю.
2. Мамойленко С.Н. Операционные системы. Часть 1. Операционная система Linux [Электронный ресурс]: учебное пособие/ Мамойленко С.Н., Молдованова О.В.— Электрон. текстовые данные.— Новосибирск: Сибирский государственный университет телекоммуникаций и информатики, 2012.— 128 c.— Режим доступа: <http://www.iprbookshop.ru/40540>.— ЭБС «IPRbooks», по паролю.
3. Войтов Н.М. Курс RH-133. Администрирование ОС Red Hat Enterprise Linux : Конспект лекций и практические работы ver. 1.10: учебный курс. - М.:ДМК-Пресс, 2011. - 193 с. - Режим доступа: <http://biblioclub.ru/index.php?page=book_red&id=129920>.— ЭБС «Университетская библиотека ONLINE», по паролю.
4. Романов С.Л. Утилиты обработки текста в операционной системе Linux: учебное пособие для вузов. - СПб: Балтийский государственный технический университет «Военмех» имени Д.Ф. Устинова, 2013. - 21 с. - Режим доступа: <https://e.lanbook.com/book/63721#book_name>.— ЭБС издательства "Лань", по паролю.

**Евгений Владимирович Вершинин**

**АнтонКонстантиновичЖуков**

**ПРОГРАММИРОВАНИЕ В SHELL**

Методические указания по выполнению

домашней работы по курсу "Операционные системы"