**Лабораторная работа №1.**

**Основы использования**

**консольного интерфейса ОС GNU/Linux.**

**Рассматриваемые вопросы:**

1. Работа с документацией по командам интерпретатора

2. Использование консольного текстового редактора

3. Создание скриптов для интерпретатора bash

**Методические рекомендации:**

Для получения подробного **справочного руководства** по любой команде

можно набрать в консоли «man название команды», для кратной справки –

название\_команды **-h** или название\_команды **--help**. Примеры: **man man** –

справочное руководство по команде man; **man bash** – справочное руководство

по интерпретатору **bash**.

**Shell-скрипт** – это обычный текстовый файл, в который последовательно

записаны команды, которые пользователь может обычно вводить в командной

строке. Файл выполняется командным интерпретатором – шеллом (shell). В

Linux- и Unix-системах для того, чтобы бинарный файл или скрипт смогли быть

запущены на выполнение, для пользователя, который запускает файл, должны

быть установлены соответствующие права на выполнение. Это можно сделать с

помощью команды **chmod u+x имя\_скрипта**. В первой строке скрипта

указывается путь к интерпретатору **#!/bin/bash**.

Для создания скрипта можно воспользоваться текстовым редактором

**nano** или **vi**, набрав имя редактора в командной строке.

Ниже приводятся основные правила программирования на языке bash.

**Комментарии**. Строки, начинающиеся с символа # (за исключением

комбинации #!), являются комментариями. Комментарии могут также

располагаться и в конце строки с исполняемым кодом.

**Особенности работы со строками**. Одиночные кавычки (' '),

ограничивающие строки с обеих сторон, служат для предотвращения

интерпретации специальных символов, которые могут находиться в строке.

Двойные кавычки (" ") предотвращают интерпретацию специальных символов,

за исключением $, ` (обратная кавычка) и \ (escape – обратный слэш).

Желательно использовать двойные кавычки при обращении к переменным. При

необходимости вывести специальный символ можно также использовать

экранирование: символ \ предотвращает интерпретацию следующего за ним

символа.

**Пробелы и переводы строк.** Интерпретаторы sh и bash чувствительны к

пробелам и переводам строк. Отдельные команды должны начинаться с новой

строки. Если есть необходимость написать еще одну команду в той же строке,

что и предыдущая – можно поставить точку с запятой в конце предыдущей

команды. Пробел, как правило, разделяет название команды и параметры,

которые ей передаются, а также параметры между собой.

**Переменные.** Имя переменной аналогично традиционному

представлению об идентификаторе, т.е. именем может быть

последовательность букв, цифр и подчеркиваний, начинающаяся с буквы или

подчеркивания. Когда интерпретатор встречает в тексте сценария имя

переменной, то он вместо него подставляет значение этой переменной. Поэтому

ссылки на переменные называются подстановкой переменных. Если

variable1 – это имя переменной, то $variable1 – это ссылка на ее

значение. "Чистые" имена переменных, без префикса $, могут использоваться

только при объявлении переменной или при присваивании переменной

некоторого значения. В отличие от большинства других языков

программирования, Bash не производит разделения переменных по типам. По

сути, переменные Bash являются строковыми переменными, но, в зависимости

от контекста, Bash допускает целочисленную арифметику с переменными.

Определяющим фактором здесь служит содержимое переменных.

**Оператор присваивания** "=". При использовании оператора

присваивания нельзя ставить пробелы слева и справа от знака равенства. Если в

процессе присваивания требуется выполнить арифметические операции, то

перед записью арифметического выражения используют оператор let,

например:

Let a=2\\*2

(оператор умножения является специальным символом и должен быть

экранирован).

**Арифметические операторы:**

"+" сложение

"–" вычитание

"\*" умножение

"/" деление (целочисленное)

"\*\*" возведение в степень

"%" остаток от деления

**Специальные переменные.** Для Bash существует ряд

зарезервированных имен переменных, которые хранят определенные значения.

 Позиционные параметры. Аргументы, передаваемые скрипту из командной

строки, хранятся в зарезервированных переменных $0, $1, $2, $3...,

где $0 – это название файла сценария, $1 – это первый аргумент, $2 –

второй, $3 – третий и так далее. Аргументы, следующие за $9, должны

заключаться в фигурные скобки, например: ${10}, ${11}, ${12}.

Передача параметров скрипту происходит в виде перечисления этих

параметров после имени скрипта через пробел в момент его запуска.

 Другие зарезервированные переменные:

$DIRSTACK – содержимое вершины стека каталогов

$EUID – эффективный UID.

$UID – ... содержит реальный идентификатор, который устанавливается

только при логине.

$GROUPS – массив групп к которым принадлежит текущий пользователь

$HOME – домашний каталог пользователя

$HOSTNAME – hostname компьютера

$HOSTTYPE – архитектура машины.

$PWD – рабочий каталог

$OSTYPE – тип ОС

$PATH – путь поиска программ

$PPID – идентификатор родительского процесса

$SECONDS – время работы скрипта (в секундах)

$# – общее количество параметров, переданных скрипту

$\* – все аргументы, переданные скрипту (выводятся в строку)

$@ – то же самое, что и предыдущий, но параметры выводятся в столбик

$! – PID последнего запущенного в фоне процесса

$$ – PID самого скрипта

**Код завершения.** Команда exit может использоваться для завершения

работы сценария, точно так же как и в программах на языке C. Кроме того, она

может возвращать некоторое значение, которое может быть проанализировано

вызывающим процессом. Команде exit можно явно указать код возврата, в

виде exit nnn, где nnn – это код возврата (число в диапазоне 0– 255).

**Оператор вывода.** Echo переменные\_или\_строки

**Оператор ввода.** Read имя\_переменной. Одна команда read может

прочитать (присвоить) значения сразу для нескольких переменных. Если

переменных в read больше, чем их введено (через пробелы), оставшимся

присваивается пустая строка. Если передаваемых значений больше, чем

переменных в команде read, то лишние игнорируются.

**Условный оператор.**

If команда1

then команда2

[else

команда3]

fi.

Если команда1 вернула \_\_\_\_\_\_\_\_\_\_после выполнения значение "истина", то выполняется

команда2 после then. Если есть необходимость сравнивать значения

переменных и/или констант, после if используется специальная команда

[[ выражение ]]. Обязательно ставить пробелы между выражением и

скобками, например:

if [[ "$a" –eq "$b" ]]

then echo "a = b"

fi

**Операции сравнения**:

Операции сравнения целых чисел:

-eq # равно

-ne # не равно

-lt # меньше

-le # меньше или равно

-gt # больше

-ge # больше или равно

Операции сравнения строк:

-z # строка пуста

-n # строка не пуста

= или == # строки равны

!= # строки не равны

< # меньше (сравниваются коды символов)

<= # меньше или равно (сравниваются коды символов)

> # больше (сравниваются коды символов)

>= # больше или равно (сравниваются коды символов)

! # отрицание логического выражения

-a,(&&) # логическое «И»

-o,(||) # логическое «ИЛИ»

**Множественный выбор.** Для множественного выбора может

применяться оператор case.

case переменная in

значение1 )

команда 1

;;

значение2 )

команда 2

;;

esac

Выбираемые значения обозначаются правой скобкой в конце значения.

Разделитель ситуаций – ;;

**Цикл for.** Существует два способа задания цикла for.

1. Стандартный – for переменная in список\_значений; do;

команды; done. Например:

for i in 0 1 2 3

do

echo $i

done

2. С-подобный

for ((i=0; c <=3; i++))

do

echo $i

done

**Цикл while:** while условие; do; команда; done. Синтаксис

записи условия такой же, как и в условном операторе, например:

i=0

while [ i –le 3 ]

do

echo $i

let i+=1

done

**Управление циклами.** Для управления ходом выполнения цикла служат

команды break и continue. Они точно соответствуют своим аналогам в

других языках программирования. Команда break прерывает исполнение

цикла, в то время как continue передает управление в начало цикла, минуя

все последующие команды в теле цикла.

**Задание на лабораторную работу №1**

1. Создайте свой каталог в директории /home/user/ Все скрипты создавайте

внутри этого каталога или его подкаталогов. (mkdir lab1)

2. Напишите скрипты, решающие следующие задачи:

i) В параметрах скрипта передаются две строки. Вывести сообщение о

равенстве или неравенстве переданных строк.

ii) В параметрах при запуске скрипта передаются три целых числа.

Вывести максимальное из них.

iii) Считывать строки с клавиатуры, пока не будет введена строка "q".

После этого вывести последовательность считанных строк в виде

одной строки.

iv) Считывать с клавиатуры целые числа, пока не будет введено четное

число. После этого вывести количество считанных чисел.

v) Создать текстовое меню с четырьмя пунктами. При вводе

пользователем номера пункта меню происходит запуск редактора

nano, редактора vi, браузера links или выход из меню.

vi) Если скрипт запущен из домашнего директория, вывести на экран путь

к домашнему директорию и выйти с кодом 0. В противном случае

вывести сообщение об ошибке и выйти с кодом 1.

**Лабораторная работа №2.**

Обработка текстовых потоков в ОС GNU/Linux Рассматриваемые вопросы

1. Понятие стандартного ввода и стандартного вывода процесса

2. Перенаправление стандартного вывода в файл

3. Связь процессов по вводу/выводу

4. Использование вывода процесса как параметра другого процесса

5. Регулярные выражения и фильтрация текстовых потоков

Методические рекомендации:

Основным интерфейсом в операционных системах GNU/Linux является консольный интерфейс с текстовым вводом и выводом данных. Это определяет подход к управлению объектами операционной системы в их текстовом отображении. Например, состояние процессов отображается в виде набора текстовых файлов в псевдофайловой системе /proc, сведения о событиях в системе хранятся в текстовых файлах журналов, настройки отдельных пакетов в текстовых конфигурационных файлах. Это делает необходимым для решения дальнейших задач управления операционной системы освоение инструментария работы с текстовыми потоками.

**Управление вводом-выводом команд (процессов)**

У любого процесса по умолчанию всегда открыты три файла – stdin (стандартный ввод, клавиатура), stdout (стандартный вывод, экран) и stderr (стандартный вывод сообщений об ошибках на экран). Эти и любые другие открытые файлы могут быть перенаправлены. В данном случае термин "перенаправление" означает: получить вывод из файла (команды, программы, сценария) и передать его на вход в другой файл (команду, программу, сценарий). Дескрипторы файлов открытых по умолчанию: 0 = stdin 1 = stdout 2 = stderr

команда > файл – перенаправление стандартного вывода в файл, содержимое существующего файла удаляется.

команда >> файл – перенаправление стандартного вывода в файл, поток дописывается в конец файла.

команда1 | команда2 – перенаправление стандартного вывода первой команды на стандартный ввод второй команды = образование конвейера команд.

команда1 $(команда2) – передача вывода команды 2 в качестве параметров при запуске команды 1. Внутри скрипта конструкция $(команда2) может использоваться, например, для передачи результатов работы команды 2 в параметры цикла for … in.

**Работа со строками (внутренние команды bash)**

${#string} – выводит длину строки (string – имя переменной);

${string:position:length} – извлекает $length символов из $string, начиная с позиции $position. Частный случай: ${string:position} извлекает подстроку из $string, начиная с позиции $position.

${string#substring} – удаляет самую короткую из найденных подстрок $substring в строке $string. Поиск ведется с начала строки. $substring – регулярное выражение (см. ниже).

${string##substring} – удаляет самую длинную из найденных подстрок $substring в строке $string. Поиск ведется с начала строки. $substring – регулярное выражение.

${string/substring/replacement} – замещает первое вхождение $substring строкой $replacement. $substring – регулярное выражение. ${string//substring/replacement} – замещает все вхождения $substring строкой $replacement. $substring – регулярное выражение.

**Работа со строками (внешние команды)**

Для каждой команды доступно управление с помощью передаваемых команде параметров. Рекомендуем ознакомиться с документацией по этим командам с помощью команды man.

sort – сортирует поток текста в порядке убывания или возрастания, в зависимости от заданных опций.

uniq – удаляет повторяющиеся строки из отсортированного файла.

cut – извлекает отдельные поля из текстовых файлов (поле – последовательность символов в строке до разделителя).

head – выводит начальные строки из файла на stdout.

tail – выводит последние строки из файла на stdout.

wc – подсчитывает количество слов/строк/символов в файле или в потоке

tr – заменяет одни символы на другие.

Полнофункциональные многоцелевые утилиты:

grep – многоцелевая поисковая утилита, использующая регулярные выражения.

grep pattern [file...] – утилита поиска участков текста в файле(ах), соответствующих шаблону pattern, где pattern может быть как обычной строкой, так и регулярным выражением.

Sed – неинтерактивный "потоковый редактор". Принимает текст либо с устройства stdin, либо из текстового файла, выполняет некоторые операции над строками и затем выводит результат на устройство stdout или в файл.

Sed определяет, по заданному адресному пространству, над какими строками следует выполнить операции. Адресное пространство строк задается либо их порядковыми номерами, либо шаблоном. Например, команда 3d заставит sed удалить третью строку, а команда /windows/d означает, что все строки, содержащие "windows", должны быть удалены. Наиболее часто используются команды p – печать (на stdout), d – удаление и s – замена.

awk – утилита контекстного поиска и преобразования текста, инструмент для извлечения и/или обработки полей (колонок) в структурированных текстовых файлах. Awk разбивает каждую строку на отдельные поля. По умолчанию поля – это последовательности символов, отделенные друг от друга пробелами, однако имеется возможность назначения других символов в качестве разделителя полей. Awk анализирует и обрабатывает каждое поле в отдельности. Регулярные выражения – это набор символов и/или метасимволов, которые наделены особыми свойствами. Их основное назначение – поиск текста по шаблону и работа со строками. При построении регулярных выражений используются нижеследующие конструкции (в порядке убывания приоритета), некоторые из которых могут быть использованы только в расширенных версиях соответствующих команд (например, при запуске grep с ключом -E).

с - любой неспециальный символ с соответствует самому себе

\с Указание убрать любое специальное значение символа с (экранирование)

^ Начало строки $ Конец строки; выражение "^$" соответствует пустой строке.

. Любой одиночный символ, за исключением символа перевода строки

[...] Любой символ из ...; допустимы диапазоны типа а-z; возможно объединение диапазонов, например [a-z0-9]

[^...] Любой символ не из ...; допустимы диапазоны

\n Строка, соответствующая n-му выражению \(...\)

r\* Ноль или более вхождений символа r

r+ Одно или более вхождений символа r

r? Нуль или одно вхождение символа r

\<...\> Границы слова

\{ \} Число вхождений предыдущего выражения. Например, выражение "[0-9]\{5\}" соответствует подстроке из пяти десятичных цифр

r1r2 За r1 следует r2

r1|r2 r1 или r2

(r) Регулярное выражение r; может быть вложенным

**Классы символов POSIX**

[:class:] альтернативный способ указания диапазона символов.

[:alnum:] соответствует алфавитным символам и цифрам. Эквивалентно выражению [A-Za-z0-9].

[:alpha:] соответствует символам алфавита. Эквивалентно выражению [A-Za-z].

[:blank:] соответствует символу пробела или символу табуляции.

[:cntrl:] соответствует управляющим символам

[:digit:] соответствует набору десятичных цифр. Эквивалентно выражению [0-9].

[:lower:] соответствует набору алфавитных символов в нижнем регистре. Эквивалентно выражению [a-z].

[:space:] соответствует пробельным символам (пробел и горизонтальная табуляция).

[:upper:] соответствует набору символов алфавита в верхнем регистре. Эквивалентно выражению [A-Z].

[:xdigit:] соответствует набору шестнадцатиричных цифр. Эквивалентно выражению [0-9A-Fa-f].

**Задание на лабораторную работу №2.**

1. Создайте свой каталог в директории /home/user/ Все скрипты и файлы для вывода результатов создавайте внутри этого каталога или его подкаталогов. (mkdir lab2)

2. Напишите скрипты, решающие следующие задачи:

i) Создать файл errors.log, в который поместить все строки из всех доступных для чтения файлов директории /var/log/, начинающиеся c последовательности символов ACPI, без указания имени файла, в котором встретилась строка. Вывести на экран те строчки из получившегося файла, которые содержат полные имена каких-либо файлов.

ii) Создать full.log, в который вывести строки файла /var/log/Xorg.0.log, содержащие предупреждения и информационные сообщения, заменив маркеры предупреждений и информационных сообщений на слова Warning: и Information:, чтобы в получившемся файле сначала шли все информационные сообщения, а потом все предупреждения. Вывести этот файл на экран.

iii) Создать файл emails.lst, в который вывести через запятую все адреса электронной почты, встречающиеся во всех файлах директории /etc.

iv) Найти в директории /bin все файлы, которые являются сценариями, и вывести на экран полное имя файла с интерпретатором, наиболее часто используемым в этих сценариях (только полное имя файла).

v) Вывести список пользователей системы с указанием их UID, отсортировав по UID. Сведения о пользователей хранятся в файле /etc/passwd. В каждой строке этого файла первое поле – имя пользователя, третье поле – UID. Разделитель – двоеточие.

vi) Подсчитать общее количество строк в файлах, находящихся в директории /var/log/ и имеющих расширение log.

vii) Вывести три наиболее часто встречающихся слова из man по команде bash длиной не менее четырех символов.