Лекция. ААл

Для понимания полезно вспомнить команды raw\_input(), split(), split(“.”), print()

## 1. Операционные системы и командная строка

**1.1. Операционная система (ОС) – это программа**, которая запускается автоматически, когда включаете компьютер. Она разбита на много-много-много программных модулей, каждый модуль – отдельный файл.

ОС работает все время, реагируя на ваши действия. Во многих ее модулях написана команда вроде команды raw\_input(). И модуль постоянно ждет ответа – вашего действия.

**1.2. ОС Windows XP (2002 год) 45 млн. строк** (поэтому Гейтсу нужны программисты по всему миру ☺ )

Когда щелкаете мышкой по иконке файла protein.txt происходит вот что.

### Выполняется функция Windows из определенного модуля: та функция, в которой выполнена команда вроде raw\_input(), только она ждет “щелчок по иконке”. Эта функция получает на вход текстовую переменную “protein.txt”, записанную в иконке .

1. Функция “щелчок по иконке” находит расширение “txt” (помните split(“.”) !)
2. В своем “словаре ассоциаций” находит ключ “txt” и берет значение. Это значение – имя программы notepad (если пользователь его не поменял).
3. Формируется строковая переменная “notepad.exe protein.txt”
4. Вызывается функция “интерпретатор командной строки” с параметром   
   “notepad.exe protein.txt”
5. Интерпретатор командной строки загружает файл notepad.exe в память компьютера и передает программе, записанной в нем, параметр “protein.txt”.
6. Выполняется программа notepad – содержимое файла protein.txt отображается в окне по байтам в виде символов в кодировке Windows (CP1251).

**Упражнение 1.** Cоздайте файл proteins.txt с содержимым на русском языке и выполните действия, начиная с п.5. *Подсказка.* Надо найти командную строку Windows.

**1.3. Семейство ОС linux.**

С самого начала ОС linux разрабатывалась добровольцами по лицензии Free/Libre and Open-Source Software (Свободное программное обеспечение с общедоступными (открытыми) исходными кодами). Поэтому *большинство ОС семейства* ***linux распространяется бесплатно***.

ОС семейства Linux лидируют почти во всех секторах рынках (в смартфонах, интернет-серверах, суперкомпьютерах, на предприятиях, в центрах данных, встраиваемых системах, нетбуках -1/3), **за исключением рынка домашних компьютеров,** где Linux занимает 3 место - несколько процентов рынка. Делайте выводы сами!

**1.4. История.** Linux является UNIX-подобной ОС. Он создан с нуля в противовес коммерциализации ОС UNIX.

Код 1й версии linux 0.01 был публично открыт Ли́нусом То́рвальдом, 23х летним студентом Хельсинского университета, в 1991г. Он использовал наработки GNU (от англ. GNU’s Not UNIX — «GNU не UNIX») — свободная Unix-подобная операционная система, разрабатываемая Проектом GNU (проект начат 1983 году Ричардом Столлманом) . Проект GNU тормозил.

В 1991 г. Торвальд писал: “Я делаю бесплатную операционную систему. Это всего лишь хобби, и она не будет большой и профессиональной как gnu.”

К разработке ядра Linux присоединились добровольцы, затем, и фирмы, которые стали использовать linux (лицензия позволяет изменять ОС, но не позволяет продавать измененную ОС).

**1.5. 2009: Ядро Linux 2.6.32 состоит из 12 606 910 строк кода.** Затраты оцениваются в 73 000 человеко-лет.

Debian, Ubuntu, RedHat, Fedora и др. - разные ОС - надстройки над ядром Linux.

**1.6. Интерпретатор командной строки linux** **называется bash** (бывают и другие, zsh, tcsh, …).

## 2. Работа на kodomo с удаленного компьютера (напр., компьютера класса)

На kodomo стоит ОС Debian (семейство Linux). На компьютерах в к. 207 и 208 - ОС Windows XP и ОС Debian

**Связь с kodomo по протоколу ssh** позволяет работать в своем эккаунте удаленно. В классе используем клиент (программу) **Putty**. Ей следует сообщить протокол связи (ssh) и адрес (kodomo.fbb.msu.ru).

## 3. Команды linux

В ОС linux имеются тысячи команд. Команда – это отдельный файл с написанной кем-то программой. Список команд можно посмотреть, т.к. файлы лежат в директориях /bin, /usr/bin. Просмотреть содержимое этих файлов бессмысленно, так как они – в бинарном формате.

Пару десятков команд надо будет освоить для получения зачета по linux.

**3.1. Как bash разбирает командную строку**

Набираю (синий шрифт):

**aba@kodomo:~/tmp$ python aba\_demo.py Abrakadabra**

(aba\_demo.py – написанный мной скрипт из пяти строк)

С этой строкой bash поступает так:

1. Превращает введенную строку в список слов, считая разделителями пробелы и табуляторы:

[“python”, “aba\_demo.py”, “Abrakadabra”] (похоже на split(), но есть особенности )

2. Первый элемент – имя программы. Точнее, имя файла, в котором записана программа. Программа python – это файл с именем python в директории /usr/bin/

3. Следующие аргументы – параметры, которые надо передать программе python.

На разбор командной строки влияют спецсимволы: **~ ! @ # $ & % \* ( ) [ ] { } ' " \ : ; > < пробел**

Их не следует использовать в именах файлов (хотя – теоретически - не запрещено). Чтобы не промахнуться, используйте в именах файлов латинские буквы, цифры, подчеркивание “\_” и больше ничего.

**Синтаксис.** “A” != “a”. Для linux две строки “Прекрасная лилия” и “Прекрасная Лилия” не равны.

**3.2. Пример работы в linux**

**Упражнение 2.** Определите сколько файлов (= команд linux) лежит в директории /usr/bin

*Подсказки.*

(1) Команда ls (от list, список) показывает список файлов и поддиректорий в указанной директории:

**…$ ls /usr/bin**

Cписок выводится на монитор. Вывод на монитор называется в linux “выходной поток”, или stdout.

(2) Оператор “>” перенаправляет выходной поток в файл:

**…$ ls /usr/bin > commands**

На этот раз список отправляется в файл с именем all\_bash\_commands в текущей рабочей директории.

(3) Проверим, что команда сработала, откроем файл для просмотра. Сделаем это командой less:

**…$ less commands**

Пробел и стрелка вниз – навигация. “q” (от quit, выйти) – выход

(4) Команда wc (от word count, подсчет слов) подсчитывает число строк, слов и символов в текстовом файле

**…$ wc commands**

На монитор выводится три числа и имя файла.

(5) Используем опцию “-w” для того, чтобы выводилось только число слов, т.к. одно слово – имя одного файла:

**…$ wc -w сommands**

Получаем искомое число.

*Предостережение:* команда ls включает в список также имена поддиректорий.

Выполним все действия ОДНОЙ командой, используя оператор “|” – перенаправить

выходной поток одной команды во входной поток следующей

**…$ ls /usr/bin | wc -w**

Таков стиль работы в linux. Этому надо научиться.

**3.3. Структура команд linux**

**Пример.**

**…$ ls -1 /bin**

/bin – директория с командами bash

“-1” - опция, с ней команда ls выводит список файлов в одну колонку

**…$ ls -1 -s /bin**

Опция “-s” приводит к тому, что указывается размер каждого файла в блоках (1 блок на kodomo равен 4096 байт )

**…$ ls -1 -s -S /bin**

Опция “-S” приводит к сортировке списка по убыванию размера файла

Вместо ls -1 -s -S /bin можно написать

**…$ ls -1sS /bin**

В примерах вы увидели односимвольное задание опции. Некоторые опции могут задаваться не одним символом, а словом как в примере:

**…$ ls --help**

Показать подсказку по команде ls. Опция “--help” применима к большинству программ

Некоторые опции требуют указания данных. См. пример.

**…$ grep “gene" /home/students/y13/daria/term1/block2/credits/NC\_003098.gbk**

Команда grep выдает все строки указанного файла (или файлов), в которых встретилось слово “gene” (слово можно и не брать в кавычки, но лучше так).

Для того, чтобы выдать две строчки перед найденной строкой и десять – после следует использовать опции:

**…$ grep -B 2 -A 10 “gene" /home/students/y13/daria/term1/block2/credits/NC\_003098.gbk**

Опции “-B” ( от before, до) и “-A” (от after, после) как раз и приводят к требуемому результату.

Итак,

* разделителями командной строки являются пробелы символы табляции
* сначала идут опции в произвольном порядке
* односимвольные опции без данных можно группировать за одним спецсимволом “-“
* в конце идут обязательные данные – слово (как в grep), имена файлов и т.п.

**3.4. Где искать информацию о командах и их опциях**

* Список опций данной команды обычно можно получить с помощью опции --help
* Описание нужной команды можно получить командой  
   man <имя команды>   
  Например, man ls
* Можно найти команду поиском по ключевому слову из ее описания:   
  man -k <ключевое слово>
* **Google.** Например, если в google задать поиск “ учебник linux ls”, то в одной из первых находок содержится хорошее описание на русском языке некоторых опций ls .

**3.5. Некоторые спецсимволы**

* $ - применяется для подстановки значения переменных bash. Например, чтобы посмотреть значение переменной PATH, содержащей список путей, открытых пользователю (без текущей директории) можно выполнить команду  
  echo $PATH.
* Пробел – разделитель командной строки
* Знак “\” используется для того, чтобы bash воспринял следующий символ как есть, даже если он входит в список спецсимволов.
* Кавычки “ ” указывают, что строку внутри следует понимать как есть, даже если в ней присутствуют спецсимволы; сами кавычки, естественно, отбрасываются. Однако в строку с двойными кавычками разрешена подстановка значений переменных (аналогично подстановке в текстовую строку с помощью “%” в python ).
* Кавычки ‘ ‘ строже, чем “ ”. Например, внутри них подстановки значения переменной не происходит.

**Упражнение 4.** Для связи с сервером Windows из linux надо указать имя компьютера в формате \\<имя компьютера>, например, \\pig.belozersky.msu.ru\COMMON. Как написать это имя в командной строке linux?

*Указание.* Для связи с сервером pig используйте команду smbclient <имя компьютера>. Признаком того, что bash правильно вас понял, будет запрос пользовательского имени (вы его не знаете, так что убивайте команду с помощью “Ctrl+c”). Иначе появится сообщение об ошибке.

**3.6. Директории и пути к ним**

Разделителем имен директорий в linux служит символ “/”. Пример:

/home/students/y13/daria   
 Некоторые директории имеют специальной обозначение. А именно,

* “/” – корневая директория сервера
* “~” – корневая директория пользователя
* “.” – текущая директория
* “..” – родительская директория по отношению к текущей

Каждый файл и каждая директория имеют абсолютный адрес. Он начинается с корня “/”. Пример:

/home/students/y13/daria/term1/block1/credits/NP\_358618.fasta

Относительный адрес задается относительно текущей директории. Примеры.

Пусть текущая директория /home/students/y13/daria/. Тогда относительный адрес файла, указанного выше, такой:

term1/block1/credits/NP\_358618.fasta

Пусть текущая директория /home/students/y13/daria/term1/block1/pr1. Тогда относительный адрес того же файла такой:

../credits/ NP\_358618.fasta

**3.7. Маски, они же шаблоны**

На вход некоторых команд linux можно подавать много файлов одновременно. Примеры:

* cp <имя файла 1> <имя файла 2> … <имя директории> - в этой форме cp копирует пересиленные файлы в указанную директорию
* ls <имя директории или файла 1> <имя директории или файла 2> …
* grep <текст> <имя файла 1> <имя файла 2> … - выдает строки, содержащие указанный текст, из всех перечисленных файлов
* cat <имя файла 1> <имя файла 2> … - выдает содержимое перечисленных файлов в выходной поток

Список файлов можно задать маской с использованием символов

* “\*” – любой число любых символов
* “?” – один любой символ
* [ ] - любой из перечисленных в скобках символов; допускается использование диапазона символов. Примеры:
  + [0-9]
  + [a-Z]
  + [0-9a\_x-z] – либо цифра, либо a, либо знак подчеркивания, либо x, y или z

**Упражнение 3.** Имена скольких файлов из директории /usr/bin

а. начинаются с цифры

b. начинаются с символов a или s, за которым следует цифра

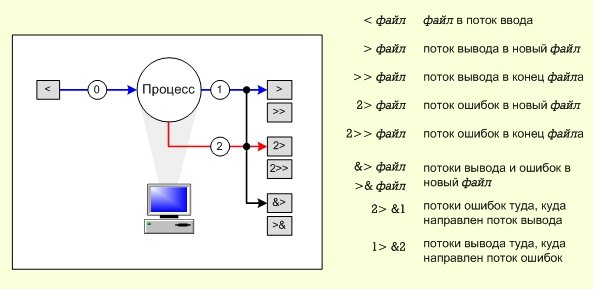
c. начинаются с символа a и состоят ровно из 4х символов

*Подсказка:* используйте команду ls.

**3.8. Потоки**

* Стандартный поток ввода (stdin), номер потока — 0.   
   По нему передаются данные программам, которые используют ввод данных с клавиатуры (*помните raw\_input()* )
* Стандартный поток вывода (stdout) — 1.  
  По нему выводятся на монитор результаты работы программы, которые отправлены на печать (*помните print()* )
* Стандартный поток ошибок (stderr) — 2.  
  По нему передаются на монитор сообщения об ошибках при работе программы (*помните, как ругался python на ваши первые варианты скриптов* ☺ )

На мониторе сообщения по потоку 1 и потоку 2 могут идти вперемешку, и не различишь где какой. В linux реализованы операторы перенаправления потоков – вместо печати выходной поток можно отправить в файл. На картинке изображены варианты перенаправления.



**> файл**

Направить стандартный поток вывода в файл. Если файл не существует, он будет создан; если существует — перезаписан сверху.

**>>файл**

Направить стандартный поток вывода в файл. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.

**2> файл**

Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан; если существует — перезаписан сверху.

**2>>файл**

Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.

**&>файл или >&файл**

Направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >файл 2>&1.

**< файл**

Использовать файл как источник данных для стандартного потока ввода.

Многие программы linux написаны так, что их результат отправляется в поток stdout. Например, команды ls, grep. Запуская программу, пользователь может перенаправить поток stdout в нужный файл, на монитор будут выдаваться только сообщения об ошибках. Это бывает удобно.

Входной поток stdin тоже можно перенаправить, т.е. вместо ввода текста с клавиатуры взять его из файла. Предположим, программа test.py запрашивает строку с клавиатуры с помощью raw\_input(“What is your name? “). Параметров в командной строке у test.py не предусмотрено. Пусть ваше имя написано в файле my\_name.txt. Можно выполнить команду так:

**…$ python test.py <my\_name.txt**

В тот момент, когда исполнится команда raw\_input(“What is your name? “), она сразу получит ответ из файла my\_name.txt, перенаправленного во входной поток, и не останавливаясь продолжит работу.

На практике перенаправление во входной поток используют **в конвейере** с помощью оператора “**|**” – перенаправить stdout одной программы в stdin другой программы

Пример. Команда

**ls | wc -w**

Программа **ls** выводит список всех файлов в текущей директории в stdout. Но опреатор “|” отправляет содержимое stdout (список файлов) во входной поток для команды **wc –w**, которая считает число слов в файле. Однако программа **wc** написана так, что если ей не задан входной параметр – имя файла, то она берет данные из входного потока stdin. Поэтому в нашем примере она выдаст число слов в списке файлов, т.е. число файлов!

Конвейер может быть длинным.

**3.9. Права**

У каждого файла есть атрибуты: дата создания, размер и др. Важным атрибутом являются **права на файл**. Есть три разновидности прав:

- право на чтение **r** (от read, читать)

- право на запись **w** (от write, писать)

- право на исполнение **x** (от eXecute, исполнять)

Право на исполнение присваивается программам и директориям. Программа из файла, которому не присвоено право на исполнение, работать не сможет. Скрипты .py не являются программами – они всего лишь являются входными данными для программы python. Поэтому им не нужно присваивать право на исполнение.

Предусмотрены три категории пользователей: **владелец файла, группа,** **все пользователи**. Для студента группа включает всех студентов того же курса и называется, например, Year-13 .   
Каждой категории права присваиваются независимо.

Посмотреть права на файл или все файлы можно командой

**…$ ls -l** (это буква “эл”)Пример выдачи:

|число линков

| |группа |время модификации

| |владелец |размер |имя файла

| | | | | |

-rw-r--r-- 1 aba Preps 293 Feb 10 16:52 aba\_demo.py

-rw-r--r-- 1 aba Preps 34788 Feb 10 19:15 commands

drwxr-xr-x 2 aba Preps 6 Feb 10 20:01 tmp

|| | |

|| | +----- права всех "прочих" по отношению к файлу

|| +-------- права одной с ним группы

|+----------- права "владельца" файла

+------------ "-" - значит файл, "d" - директория

Права:

r - файл доступен для чтения,

w - файл доступен для записи,

x - файл является выполнимым,

- - данное право доступа отсутствует.

Изменить права на файл commands можно командой chmod. Примеры:

**…$ chmod u+x commands**

u (от user) – владелец файла, ему разрешаем исполнять файл

g (от group) – группа

o (от other) - оставльные пользователи

a (от all) - и владелец, и группа, и остальные

**Упражнение 4.** Права на файл new\_file такие: “- rw- --- ---“, т.е. владелец может читать и записывать, группа и все остальные не имеют прав ни на что. Объясните, что произойдет после каждой из команд

**…$ chmod g+w new\_file**

**…$ chmod a+x new\_file**

**…$ chmod o-x new\_file**

**…$ chmod a-r new\_file**

Из FAR можно менять права на файл. Для этого надо связаться с kodomo по протоколу WinScp, далее Ctrl+a. В меню можно менять права.

**3.10. Удобные возможности**

* Стрелка вверх
* Табулятор
* history и history 10
* копирование
* Ctrl+c

**3.11. bash скрипты**

Команды linux можно записать построчно в файл, сделать этот файл исполняемым, и выполнить. Пример. Пусть файл my\_script содержит команды:

#!/bin/bash

echo “Start..”

ls /usr/bin | wc –w   
echo “..Done”

Выполним команды

**…$ chmod u-x my\_script** # сделали файл исполняемым

**…$ ./my\_script** # выполнили его

**Упражнение 5.** Каков будет результат?

*Замечание.* В bash скриптах предусмотрены операторы, позволяющие программировать. Некоторые линуксоиды на bash пишут сложные программы.