CAOS 3 – asm

Manakov Danila

MIPT

22 сентября 2022 г.

bash

Что вообще такое bash?

- оболочка (shell)
- интерпретатор командной строки
- вы вводите команду - оно её выплняет

bash

Помимо bash существуют другие оболочки

- sh
- bash
- dash
- csh
- fish
- zsh
- ...

но bash стал 'стандартом', поэтому все скрипты пишутся в основном под него

приглашение командной строки

```
1  user@my_pc_name:/path/to/current/directory$ <write command here>
```

приглашение командной строки

```
1  user@my_pc_name:/path/to/current/directory$ <write command here>
```

- user
- @
- my_pc_name
- :
- /path/to/current/directory

приглашение командной строки

```
1 user@my_pc_name:/path/to/current/directory$ <command from user>
2
3 root@my_pc_name:/path/to/current/directory# <command from root>
```

откуда берутся команды

Есть переменная окружения PATH. В ней через двоеточие записаны пути до папок с исполняемыми файлами

```
$ echo $PATH
/home/dmanakov/anaconda3/bin:
/home/dmanakov/anaconda3/condabin:
/usr/local/sbin:
/usr/local/bin:
/usr/sbin:
/usr/bin:
/sbin:/bin:
/usr/games:
/usr/local/games:
/snap/bin
```

откуда берутся команды

```
1 $ ls
2 a.out main.c
3
4 $ /usr/bin/ls
5 a.out main.c
```

откуда берутся команды

```
1 $ whereis ls
2 ls: /usr/bin/ls
3 /usr/share/man/man1/ls.1.gz
4 /usr/share/man/man1/ls.1posix.gz
```

синтаксис команды

```
1  $ command [arg1] [arg2] [arg3] ...
```

У любой команды минимум один аргумент (arg0), в нем прописан путь до исполняемого файла

Самые популярные команды

- cd
  (попробуйте ввести 'whereis cd')
- ls
- echo
- cat
- mkdir
- touch
- rm (rmdir)
- grep
- sed

Навигация по папкам (cd ls)

cd - ChangeDirectory

```
1  $ cd /path/to/new/dir # абсолютный путь
2  $ cd path/to/dir      # относительный путь
3  $ cd .                # cd to current_dir
4  $ cd ..               # cd to parrent_dir
5  $ cd ~                # cd to home_dir
6  $ cd ~/some/path      # example
```

ls - LiSt files & directories

```
1  $ ls              # list files in current directory
2  $ ls /some/path # list files in /some/path
```

Потоки ввода и вывода

```
1 $ some_command < stdin > stdout
```

'<' и '>' это НЕ АРГУМЕНТЫ, это перенаправление потоков

Потоки ввода и вывода

```
1 $ some_command < stdin > stdout
```

'<' и '>' это НЕ АРГУМЕНТЫ, это перенаправление потоков

```
1 $ some_command 0>stdin 1>stdout 1>stderr
```

Пример

```
1  $ ls > ls_output.txt
2  $ cat ls_output.txt
3  a.out main.c
4  $ cat < ls_output.txt
5  a.out main.c
```

grep

```
1  $ cat main.c
2  #include <stdio.h>
3  #include <math.h>
4
5  int main() {
6          printf("Hello\n");
7  }
8
9  $ grep 'include' main.c
10 #include <stdio.h>
11 #include <math.h>
12
```

А что если...

```
$ lorem -s 1
Cupiditate voluptatem tempora debitis ipsam cumque veritatis.
```

The ЦЕЛЬ

ЦЕЛЬ: сгенерируем 100 предложений; выведем те, которые содержат 'ips'

Перенаправление через pipe

```
$ lorem -s 1 | cat
Cupiditate voluptatem tempora debitis ipsam cumque veritatis.

# почти эквивалентно

$ lorem -s 1 > buffer.txt
$ cat buffer.txt
Cupiditate voluptatem tempora debitis ipsam cumque veritatis.
$ rm buffer.txt
```

echo

echo - выводит свои аргументы

```
1  $ echo hello world
2  hello world
3
4  $ echo hello world again
5  hello world again
6
7  $ echo -e 'hello\n css!'
8  hello
9   css!
```

sed

sed - Stream EDitor - потоковый текстовый редактор

```
1 $ echo 'some file' | sed 's/file/data/g'
2 some data
```

За подробностями можно сюда (эта штука кликабельна)

The ЦЕЛЬ

Что нам соответственно нужно сделать?

1. Сгенерировать 100 предложений
2. Заменить '. ' на '
   n'
3. Найти 'ips'

Проверяем

```
1  $ lorem -s 4 | sed -e 's/\. /\.\n/g'
2  Id quo quis dolorem molestias porro.
3  Illum delectus nesciunt temporibus natus ipsa aut autem aspernatur.
4  Et fuga quia voluptatem nisi eos quaerat accusantium.
5  Non fugit architecto eum.
```

Достигаем цель

```
1 $ lorem -s 100 | sed -e 's/\. /\.\n/g' | grep ips
2 Nihil perferendis et blanditiis suscipit ab ipsam.
3 Deleniti ipsa officiis non rem ullam optio.
4 Ratione ipsam eaque consequatur minus fuga vitae magni.
5 Quo explicabo ipsam dolores dolorem ea.
```

Переменные в bash

```
1 $ some_var_1=123
2 $ some_var_2='some var value as string'
3 $ export some_var_3='some value'
```

Переменные в bash

```
1  $ some_var_1=123
2  $ some_var_2='some var value as string'
3  $ export some_var_3='some value'
```

Доступ к переменным осуществляется через знак $

```
1  $ echo $some_var_1
2  123
3  $ echo $some_var_2
4  some var value as string
5  $ echo $some_var_3
6  some value
```

(доллар слева - приглашение командной строки)

Разница между кавычками

```
1  $ a=123
2  $ quates='a = $a'
3  $ dquates="a = $a"
4
5  $ echo $quates
6  a = $a
7
8  $echo $dquates
9  a = 123
```

Сохранение вывода в переменную

```
1 $ ls_result=$(ls)
2 $ another_ls_result=`ls`
```

Математика в bash (команда bc)

(с ней все плохо)

```
$ bc
2 + 2
4
$ echo '(2 + 2) * 2' | bc
8

$ a=123
$ b=456
$ echo "$a + $b" | bc
579
```

Математика в bash (команда bc)

(с ней все плохо)

```
$ bc
2 + 2
4
$ echo '(2 + 2) * 2' | bc
8

$ a=123
$ b=456
$ echo "$a + $b" | bc
579
```

совет: в обычной жизни используйте python / lua / js / ...

ASM

Исходник

```
1  void _start() {
2          int a = 10;
3          int b = 22;
4          int c = a + b;
5  }
```

x86 version

```
1  0000000000001000 <_start>:
2  f3 0f 1e fa                endbr64
3  55                         push   rbp
4  48 89 e5                   mov    rbp,rsp
5  c7 45 fc 0a 00 00 00       mov    DWORD PTR [rbp-0x4],0xa
6  c7 45 f8 16 00 00 00       mov    DWORD PTR [rbp-0x8],0x16
7  8b 55 fc                   mov    edx,DWORD PTR [rbp-0x4]
8  8b 45 f8                   mov    eax,DWORD PTR [rbp-0x8]
9  01 d0                      add    eax,edx
10 89 45 f4                   mov    DWORD PTR [rbp-0xc],eax
11 90                         nop
12 5d                         pop    rbp
13 c3                         ret
```

arm version

```
1   0x000001a1                  unaligned
2   0x000001a2      85b0         sub sp, 0x14
3   0x000001a4      00af         add r7, sp, 0
4   0x000001a6      0a23         movs r3, 0xa
5   0x000001a8      fb60         str r3, [r7, 0xc]
6   0x000001aa      1623         movs r3, 0x16
7   0x000001ac      bb60         str r3, [r7, 8]
8   0x000001ae      fa68         ldr r2, [r7, 0xc]
9   0x000001b0      bb68         ldr r3, [r7, 8]
10  0x000001b2      1344         add r3, r2
11  0x000001b4      7b60         str r3, [r7, 4]
12  0x000001b6      00bf         nop
13  0x000001b8      1437         adds r7, 0x14
14  0x000001ba      bd46         mov sp, r7
15  0x000001bc      5df8047b     ldr r7, [sp], 4
16  0x000001c0      7047         bx lr
```

RISC vs CISC

RISC

- Emphasis on software
- Small number of fixed length instructions
- Simple, standardised instructions
- Single clock cycle instructions
- Heavy use of RAM

CISC

- Emphasis on hardware
- Large number of instructions
- Complex, variable-length instructions
- Instructions can take several clock cycles
- More efficient use of RAM

Как собирать под arm

Т.к. почти у всех компы на x86_64, придется немного заморочиться.
Варианты следующие:

1. Вять raspberry pi / другой одноплатник на arm
2. Через cross-compilation

cross-compilation под arm

Нам нужно скачать arm-компилятор (например от проекта linaro). И использовать его вместо стандартного gcc

1. Скачиваем linaro и запускаем их компилятор
   ссылка_на_старый_ридинг_с_инструкцией
   ссылка_на_linaro
2. Вариант для ленивых (установка из репозитория)
   ссылка_как_поставить_cross-compiler

Независимо от выбранного способа нужно перейти сюда (ссылка_на_linaro) и скачать sysroot (там содержатся динамические библиотеки)

пример

```
1 $ arm-linux-gnueabihf-gcc main.c  # на выходе будет ELF для arm
2 $ gcc main.c # на выходе будет ELF под x86_64
```

Запуск arm ELF

Поскольку исполнять arm код x86_64 процессор не умеет, нам нужна виртуальна машина. В рамках курса мы используем qemu.

```
$ sudo apt-get install qemu-user # вроде этого достаточно чтоб поставит
$ arm-linux-gnueabihf-gcc -static main.c # компилируем
$ qemu-arm -L /home/dmanakov/sysroot a.out # запускаем
Hello world
```

Удаленная отладка

Ставим gdb-multiarch и подключаемся удаленно Запускаем в qemu

```
$ qemu-arm -L /home/dmanakov/sysroot -g 4321 a.out # запускаем (4321 -
```

Подключаемся удаленно через отладчик

```
$ gdm-multiarch
(gdb) target remote localhost:4321
(отлаживаем)
```

Пример с linaro и qemu

ссылочка на репу с примером удаленной отладки