Тренинги **Intel Delta Course**

«Дополнительные главы по **Software Engineering**»

Скриптовые языки программирования: Bash, Perl, Python

Виктория Кардакова, **Intel**

«Suppose you went back to Ada Lovelace and asked her the difference between a **script** and a **program**. She'd probably look at you funny, then say something like: Well, a script is what you give the actors, but a program is what you give the audience. That Ada was one sharp lady...

Since her time, we seem to have gotten a bit more confused about what we mean when we say scripting. It confuses even me, and I'm supposed to be one of the experts.»

Larry Wall

Что такое скриптовый язык?

«Скрипт» - это разговорное понятие, точного определения того, что значит «скрипт» или «скриптовый язык» не существует.

Также не существует чёткой границы между тем, что называют «скриптом» и «программой».

Особенности скриптовых языков:

- Скрипт, как правило, исполняется интерпретатором, отдельная стадия компиляции отсутствует
- Наличие сложных типов данных (списки, стеки, хеш-таблицы, …)
- Слабая типизация
  - Тип переменной определяется в зависимости от того, каким образом переменная используется
- Наличие «сборщика мусора»

Преимущества и недостатки скриптовых языков

Преимущества

- Встроенные сложные типы данных и операции над ними
- Удобная работа со строками при помощи регулярных выражений
- Возможность использовать совместно различные программы
- Относительная простота изучения и использования

Недостатки

- Время исполнения скрипта
- Проблемы безопасности
  - Скрипты на web страницах могут использовать уязвимости браузера и ОС
- Результат выполнения скрипта зависит от версии интерпретатора

bash (Bourne Again SHell)

Одна из наиболее популярных командных оболочек для ОС UNIX, Linux.

Соответствует стандарту IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard.

Преимущества:

- Легко запускать другие программы
- Удобное перенаправление ввода/вывода

Недостатки:

- Не является языком программирования общего назначения

bash. Синтаксис.

Первая программа на Bash

```
# This is a comment.
echo Hello world!
```

Оператор условного перехода

```
if test-commands; then
    consequent-commands;
elif more-test-commands; then
    more-consequents;
else
    alternate-consequents;
fi
```

Примеры

```
# file: exists.sh
file=$1
if [ -e $file ]; then
    echo "$file exists"
else
    echo "$file doesn't exist"
fi
```

```
-bash-4.1$ ./exists.sh /proc/cpuinfo
/proc/cpuinfo exists
```

```
# file: strings.sh
first=$1; second=$2;
if [ "$first" == "$second" ]; then
    echo "Strings are equal"
elif [[ $first == *$second* ]]; then
    echo "$first contains $second"
elif [[ $second == *$first* ]]; then
    echo "$second contains $first"
else
    echo "Strings differ"
fi
```

```
-bash-4.1$ ./strings.sh literature era
literature contains era
```

bash. Циклы.

Циклы

```
until test-commands; do
    consequent-commands;
done
```

```
while test-commands; do
    consequent-commands;
done
```

```
for name in list; do
     commands;
done
```

```
for (( expr1; expr2; expr3 ));
do
    commands;
done
```

Примеры

```
# file: reader.sh
file=$1
while IFS= read -r line; do
   echo $line
done < "$file"
```

```
-bash-4.1$ ./reader.sh test.txt
This is a test file.
Last line of the file.
```

```
# file: runner.sh
dir=$1
for file in `ls $dir`; do
     if [[ $file == *.out ]];
    then
    ./$file
    fi
done
```

```
-bash-4.1$ ls ./samples/
hello.c  hello.out  world.c  world.out
-bash-4.1$ ./runner.sh ./samples/
hello
world!
```

bash. Перенаправление ввода/вывода

0 – дескриптор стандартного потока ввода (stdin)
1 – дескриптор стандартного потока вывода (stdout)
2 – дескриптор стандартного потока ошибок (stderr)

Запись вывода программы в файл:

```
ls -lA > ls_result.txt     # перезапишет файл ls_result.txt
ls -lA >> ls_result.txt    # допишет в конец файла ls_result.txt
```

Запись ошибок программы из потока stderr в файл:

```
grep hello * 2> grep-errors.txt
```

Запись вывода программы и её ошибок в файл:

```
grep hello * &> grep-results.txt
grep hello * > grep-results.txt 2>&1
```

Перенаправление ввода

```
grep hello < ./samples/hello.c
```

Конвейерная пересылка данных

```
ls | grep test | xargs cat
```

Регулярные языки

Определение регулярного языка

- Пустое множесто ∅ и множество, состоящее из пустого слова $\{\varepsilon\}$ являются регулярными языками
- Множество {a}, состоящее из одного символа алфавита ∑ является регулярным языком.
- Если языки L_1 и L_2 – регулярные, то:
  - Их объединение $L_1 \cup L_2$ - тоже регулярный язык.
  - Язык, состоящий из всевозможных конкатенаций слов языков L_1 и L_2 – тоже регулярный язык ($L_1\ L_2$).
  - Языки L_1^*, L_2^* - тоже регулярные (L^* - замыкание Клини – минимальное из множеств, содержащее пустое слово ε, а также все слова языка L и замкнутое относительно операции конкатенации).

Регулярные выражения

Регулярные выражения

Регулярные языки вида $\emptyset, \{\varepsilon\}, \{a\}, L_A \cup L_B, L_A\ L_B, {L_A}^*$ часто записывают в виде $\emptyset, \varepsilon$, a, A|B, AB, A*.

Выражения такого вида называются регулярными выражениями.

Если выражение включает несколько операций, то они выполняются со следующими приоритетами:

1. A*
2. AB
3. A|B

Примеры

(a|b)* = {ε, a, b, aa, ab, ba, bb, aaa, aab, ...}

ab*(c| ε) = {a, ab, abb, abc, abbb, abbc, ...}

(ab)*(c| ε) = {ε, c, ab, abc, abab, ababc, ...}

bash. Регулярные выражения.

Используются для поиска строк, соответствующих заданному шаблону.

Когда интерпретатор встречает в строке символы *, ? или [, то он обрабатывает эту строку как шаблон.

Формат записи шаблонов в bash:

- * - спец. символ. соответствует любой строке, в том числе пустой
- ? - спец. символ. соответствует любому единичному символу
- […] – соответствует любому из символов, перечисленных в скобках.
- Можно задавать списки символов, например [0-9] – любая цифра.
- [^…], [!…] – соответствует любому из символов, НЕ перечисленных в скобках.

Пример

```
#file: regex.sh str=$1
if [[ $str ==
    *_[A-Z][a-z][a-z]_[0-9][0-9]
]]
then
    echo $str matches
else
    echo $str doesn\'t match
fi
```

```
-bash-4.1$ ./regex.sh file_Apr_14
file_Apr_14 matches
-bash-4.1$ ./regex.sh file_Apr_2014
file_Apr_2014 doesn't match
-bash-4.1$ ./regex.sh file_APR_2014
file_APR_2014 doesn't match
```

bash. Регулярные выражения.

Расширенная поддержка регулярных выражений доступна при помощи

shopt -s extglob # включение

shopt -u extglob # отключение

- ?(pattern-list) – шаблон встретится не более одного раза
- *(pattern-list) – шаблон встретится любое число раз
- +(pattern-list) – шаблон встретится один или более раз
- @(pattern-list) – шаблон встретится ровно один раз
- !(pattern-list) – соответствует любой строке, за исключением заданного шаблона
- | - разделитель в списке шаблонов

Пример

```
#file: regex.sh str=$1
if [[ $str ==
     *_@(Jan|Feb|Mar|Apr)_[0-9][0-9]
]]
then
     echo $str matches
else
     echo $str doesn\'t match
fi
```

```
-bash-4.1$ ./regex.sh file_Jan_12
file_Jan_12 matches
-bash-4.1$ ./regex.sh file_Apd_14
file_Apd_14 doesn't match
```

Perl

Разработан в 1987 году Ларри Уоллом (Larry Wall) как скриптовый язык общего назначения для ОС Unix.

Преимущества:

- Широкие возможности применения регулярных выражений
- Большое количество доступных модулей (CPAN)
- Переносимость. Существуют интерпретаторы для многих ОС

Недостатки:

- Синтаксис

"Perl - The only language that looks the same before and after RSA encryption."

Keith Bostic, BSD architect

```
"#!/usr/bin/perl
@n=('pwu','grg');sub n{local($n)=eval"get$n[$_[1]]id(\$_[0])";$n&&"($n)";}sub nm {$a-$b;}@gr=split('
',$();$g=shift(@gr);$\="\n";print"uid=$<",&n($<)," gid=",$g, &n($(,1),(" euid=$>".&n($>))x($<!=$>),(" egid=".$)+0
.&n($),1))x($(!=$)), (" groups=".join(',',sort nm grep(($_.=&n($_,1))||1,@gr)))x($#gr>=0);
```

Though I'll admit readability suffers slightly..."

Larry Wall

Perl 5. Синтаксис.

Первая программа на Perl

```
# This is a comment.
print "Hello world!\n";
```

Типы данных

- Скалярные переменные

```
my $str="String";
my $val=3.14;
```

- Массивы

```
my @animals=("dog", "cat",
"owl");
my @numbers=(23, 42, 69, 35, 9);
my @mixed=("camel", 42, 1.23);
```

Индексация начинается с 0.

```
print $animals[1]; # "cat"
```

Индекс последнего элемента:

```
$#array.
```

Прочитать часть массива:

```
@numbers[1,2]; # (42, 69)
@numbers[1..3]; # (42, 69, 35)
```

- Хеш-таблицы

```
my %fruit_color = (
    banana => "yellow",
    apple => "red",
    kiwi => "green"
);
```

Доступ к элементам:

```
print $fruit_color{"apple"};
# "red"
```

Получение списка ключей и значений:

```
my @keys = keys
%fruit_color;
my @vals = values
%fruit_color;
```

Perl 5. Операторы условного перехода

```
$a = TEST ? VAL_IF_TRUE :
VAL_IF_FALSE;

STATEMENT if CONDITION;
STATEMENT unless CONDITION;

if (EXPR) {
    STATEMENTS;
} elsif (EXPR) {
    STATEMENTS;
} else {
    STATEMENTS;
}

unless (EXPR) {
    STATEMENTS;
} elsif (EXPR) {
    STATEMENTS;
} else {
    STATEMENTS;
}
```

Примеры

```
# file: comparison.pl
my $val = $ARGV[0];
print $val." greater than 5\n"
    unless $val <= 5;
```

```
-bash-4.1$ perl ./comparison.pl 7
7 greater than 5
```

```
# file: branches.pl
my $val = $ARGV[0];
my $foo;
if ($val < -4.0) {
    $foo = sqrt(-$val);
} elsif (-4.0 <= $val and
         $val < 1.0) {
    $foo = 2.0;
} else {
    $foo = $val + 1.0;
}
print $foo."\n";
```

Perl 5. Циклы.

```
STATEMENT while CONDITION;
STATEMENT until CONDITION;
STATEMENT for LIST;
STATEMENT foreach LIST;

while (CONDITION) {
    STATEMENTS;
} continue {
    MORE STATEMENTS;
}

for VAR (LIST) {
    STATEMENTS;
}

for ( EXPR1; EXPR2; EXPR3 ) {
    STATEMENTS;
}
```

next; - переход на следующую итерацию

last; - выход из цикла

Примеры

```
# file: loop1.pl
$a = 1;
print ($a++, "\n") until $a > 3;


# file: loop2.pl
my %city_ppl = (
    Moscow => 12.1,
    Mumbai => 12.5,
    Hanoi => 6.8
);

for my $k (keys %city_ppl) {
    my $val = $city_ppl{$k};
    print $k."\t".$val."\n"
        if $val > 10;
}
```

```
-bash-4.1$ perl loop2.pl
Mumbai  12.5
Moscow  12.1
```

Perl 5. **Регулярные выражения.**

Поиск шаблона в строке:

```
if ($a =~ /expr/) {
    print "match\n";
}
```

Замена однократная:

```
$a =~ s/expr1/expr2/;
```

Замена всех expr1 на expr2:

```
$a =~ s/expr1/expr2/g;
```

Специальные символы:

. – любой символ

\s – пробел (‘ ‘, ‘\t’, ‘\n’, …)

\d – любая цифра

\D – любая НЕ цифра

[…] – любой символ из перечисления

[^…] – любой символ НЕ из перечисления

(qqq|www|zzz) - любая строка из списка

^, **$** - символы начала и конца строки

Конструкции, обозначающие количество совпадений:

***** – 0 или более совпадений выражения, стоящего перед *

+ – 1 или более совпадений

? – 0 или 1 совпадение

{n} – ровно n совпадений

{n,} – n или более совпадений

{n,m} – от n до m совпадений

Пример

```
# поиск даты в формате
# YYYY/MM/DD
$str=~/\d{4}\/(0[1-9]|1[0-2]
)\/(30|31|[0-2]\d)/;
```

Python

Возник в конце 1980-х годов как расширяемый скриптовый язык для распределённой ОС Amoeba в институте CWI.

Автор: Гвидо ван Россум (Guido van Rossum)

Преимущества:

- Широкие возможности («богатая» стандартная библиотека)
- Ясный синтаксис
- Переносимый. Существуют интерпретаторы для многих ОС
- Расширяемый (через библиотеки на C/C++)

Недостатки:

- Требователен к памяти

Python 2.x Типы данных

```
# This is a comment.
print "Hello world!\n"
```

- Числовые

```
a = -7
b = 3.14
c = 2 + 1.2j
```

- Последовательности
  - Строки

```
str1 = "String"
str2 = str1[3:5];   # “ing”
str3 = 2*str2 + ">";
```

  - Списки

```
a = [‘aa', ‘ab', 10, 1.23]
b = ['fn', -3]
a.append('b')
a.extend(b)
```

- Коллекции
  - dict - словарь (ассоциативный массив)

```
city_ppl = {
'Moscow' : 12.1,
'Mumbai' : 12.5,
'Hanoi' : 6.8
}
print d['Hanoi'] # 6.8
```

  - set - множество

```
a = set('abracadabra')
# set(['a','r','b','c','d'])
arr = [ 2, 5, 7, 2, 7 ]
uarr = set(arr);
# set([2, 5, 7])
5 in uarr
# True
```

Python 2.x Ветвления. Циклы

Операторы условного перехода

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
else:
    statement(s)
```

Циклы

```
while expression1:
    statement(s)

for target_list in expr_list:
    statement(s)
```

```
x = 5
if x < 0:
    print 'Negative'
elif x == 0:
    print 'Zero'
else:
    print 'Positive'
# Positive
words = ['cat','wind','door']
for w in words:
    print w, len(w)
# cat 3
# wind 4
# door 4
fruits = ['Banana', 'Apple',
'Kiwi']
loud_fruits = \
    [fruit.upper() for fruit
in fruits]
print(loud_fruits)
# ['BANANA', 'APPLE', 'KIWI']
```

Python 2.x **Классы**

- Все поля публичные по умолчанию (однако есть соглашение, что методы начинающиеся с ‘_’ - приватные методы класса)
- Все методы виртуальные
- Синтаксис:

```
class ClassName:
    <выражение-1>
    ...
    <выражение-N>
```

- Наследование:

```
class ClassName(BaseClass):
    <выражение-1>
    ...
    <выражение-N>
```

```
class BaseClass(object):
    def f(self):
        print 'In BaseClass.f()`
class MyClass(BaseClass): # наследование
    """Пример простого класса""" # документация
    i = 12345 # поле класса
    def __init__(self, x): # конструктор
        self.x = x # поле объекта
    def f(self):
         # вызов метода f() класса BaseClass
         super(type(self), self).f()
         print 'In MyClass.f(), x = ', self.x
    @staticmethod
    def f2():
        MyClass.i = 54321
        print 'In MyClass.f2(), i = ', MyClass.i

# использование
inst = MyClass(3.5)
inst.f()
```

Ссылки

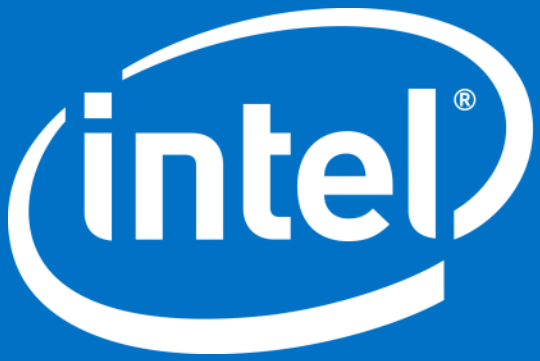
Aho, Alfred V.; Ullman, Jeffrey D. (1992). "Chapter 10. Patterns, Automata, and Regular Expressions". *Foundations of Computer Science*

https://www.gnu.org/software/bash/bash.html

http://www.perl.org/

http://perldoc.perl.org/

https://docs.python.org/2.7/


intel