Конспект по теме «Основы запуска скриптов»

Настройка учётной записи в Яндекс.Облаке

1. Зарегистрируйтесь в Яндекс.Облаке: cloud.yandex.ru
2. Создайте платёжный аккаунт. Следуйте инструкции:
   https://cloud.yandex.ru/docs/billing/quickstart/#create_billing_account
3. Сгенерируйте **ssh-ключи** для подключения к вашей виртуальной машине. **SSH** — это технология, позволяющая безопасно управлять операционными системами на виртуальных машинах. Подключение по ssh требует наличия двух ключей: открытый ключ размещают на виртуальной машине, а закрытый хранят у пользователя. Подключение с парой ключей безопаснее, чем подключение по логину и паролю. Следуйте инструкции: https://cloud.yandex.ru/docs/compute/operations/vm-connect/ssh#creating-ssh-keys
4. Создайте виртуальную машину. Следуйте инструкции: https://cloud.yandex.ru/docs/compute/quickstart/quick-create-linux
   Запишите логин в тот же файл, куда вы записали пароль. Логин также понадобится для подключения к виртуальной машине.
   Введите ключ доступа. Перейдите в папку, где он был сохранен в момент генерации (в примере — это `/Users/YOUR_USER/.ssh`). Откройте файл `id_rsa.pub` и скопируйте его содержимое.
5. Подключитесь к виртуальной машине через терминал. Следуйте инструкции:
   https://cloud.yandex.ru/docs/compute/operations/vm-connect/ssh#vm-connect
   Скопируйте публичный ip.

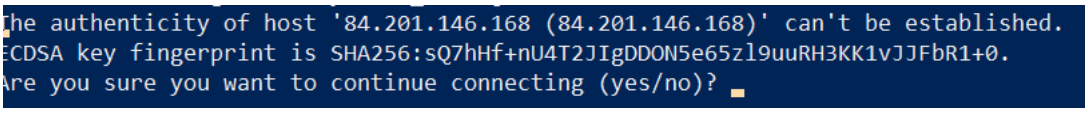
Введите в терминале команду:

```
ssh <ЛОГИН>@<публичный_IP-адрес_виртуальной машины>
```

Например:

```
ssh test_admin@84.201.144.4
```

Система предупредит о том, что это первое подключение к виртуальной машине. Когда она запросит подтверждение, наберите `yes`:

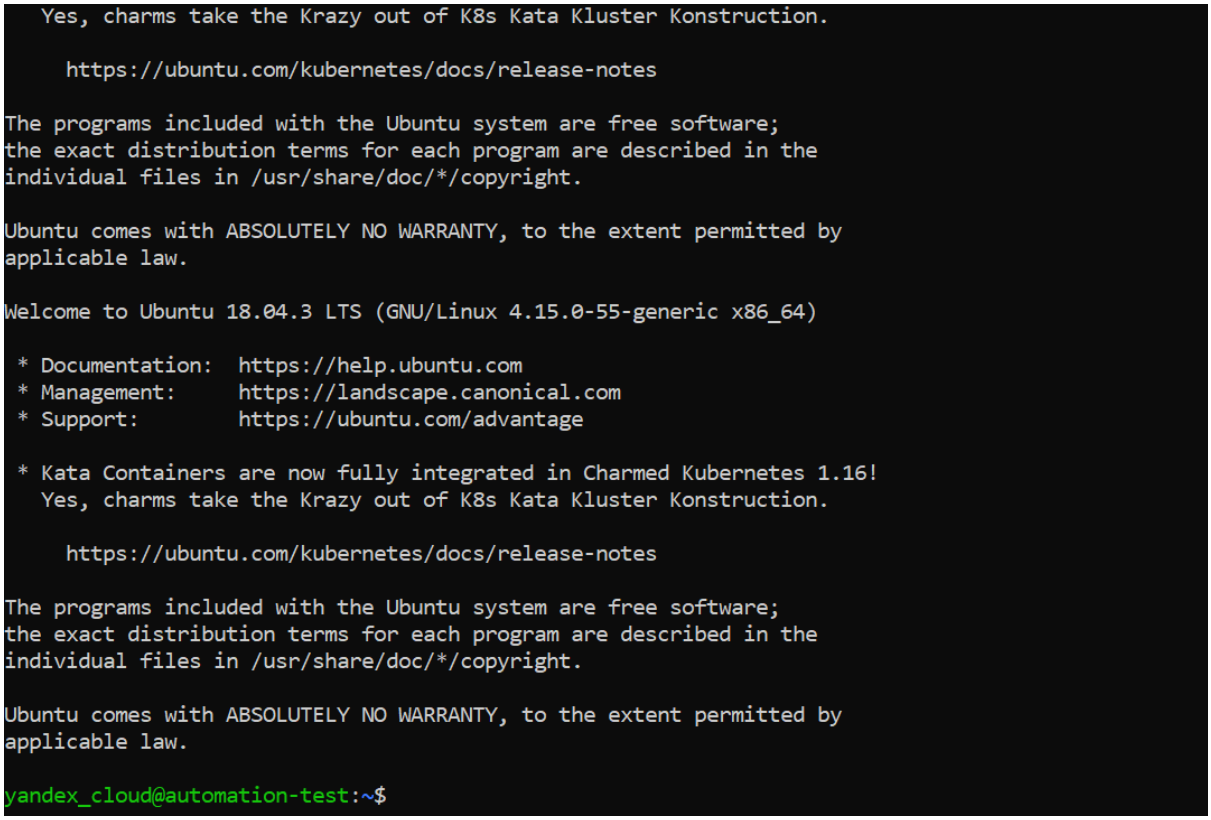


Введите пароль, заданный на этапе создания ssh-ключа. На этом этапе может возникнуть ошибка `permission denied (publickey)`. Возможные причины:

- Неправильный логин: перепроверьте его;
- SSH-ключ скопирован неверно: не целиком или наоборот содержит лишние символы.

Самое простое решение — удалить созданную виртуальную машину и попробовать создать её заново.

В случае успеха появится окно:



Наберите команду `lsb_release -a`, система выведет на экран версию операционной системы Ubuntu.

Доступ к командной строке Linux на вашей локальной машине

Команды можно вводить на вашем компьютере **локально**, без подключения к облаку.

У владельцев Linux доступ к командной строке уже есть.

Mac OS построена на основе Linux и обладает всеми его возможностями. Если вы работаете в Mac OS, перейдите в папку `Applications/Utilities` и найдите программу `Terminal` — это и есть командная строка.

Если вы работаете в Windows 10, можете установить на свою машину «параллельную», полностью функциональную операционную систему **Ubuntu** — разновидность Linux. Для этого:

1. Нажмите кнопку «Пуск» и наберите *Microsoft Store*
2. В *Microsoft Store* найдите Ubuntu
3. Появится окно приложения Ubuntu, нажмите в нем «Установить» или *Install*. Ubuntu установится на ваш компьютер.
4. Перезагрузитесь. После этого нажмите пуск и введите `Ubuntu`.
5. Запустите Ubuntu: подождите, пока система выполняет начальную настройку. Появится окно терминала.

Особенность работы с Ubuntu в Windows 10 — то, что Ubuntu имеет доступ к файловой системе, но организует пути к файлам не так, как Windows. Например, в Windows корневая папка диска C будет доступна по пути `C:\` . В Ubuntu файлы этой папки находятся по адресу: `/mnt/c/`.

Если у вас другая ОС — например, более ранние версии Windows — подключитесь к Ubuntu на виртуальной машине.

Основы работы с командной строкой в Linux

Вы научились присоединяться к виртуальной машине. Ей управляет операционная система Ubuntu — разновидность Linux. В ОС Linux управление и навигация проходит в терминале командной строки, его ещё

называют **bash**.

Когда вы подключаетесь к виртуальной машине, система открывает терминал bash. Вы попадаете в свою основную рабочую папку, или **директорию.**

Строка внизу ожидает ввода bash-команд. Вот самые частотные:

- **whoami** возвращает имя вашего пользователя;
- **cd** позволяет сменить текущую папку;
- **mkdir** создаёт папку;
- **rm** удаляет файл или пустую папку;
- **rm -r** удаляет папку с файлами в ней;
- **cat** выведет на экран содержимое файла;
- **echo** \выводит на экран текст или содержимое переменной.

Команд из списка достаточно для настройки базовой автоматизации. Если хотите погрузиться глубже и стать настоящими гуру, изучите документацию здесь: https://omgubuntu.ru/basic-linux-commands-for-beginners/.

Запуск скрипта из командной строки

Чтобы автоматизация заработала, нужно научить компьютер запускать программы с входными параметрами по расписанию. Для этого пишут Python-скрипты.

Скачайте текстовый редактор (например, Sublime Text или Notepad++), чтобы редактировать ваши Python-скрипты. Создайте в редакторе пустой файл и сохраните его как `test.py`.

Система покажет сообщение: `'Hello world.'`

Ваш Python-скрипт состоит из нескольких основных компонентов:

- `#!/usr/bin/python` говорит ОС, на каком языке программирования написан скрипт;
- `# -*- coding: utf-8 -*-` сообщает, что скрипт использует кодировку **utf-8**. Грубо говоря, эта строка позволяет записывать в скрипте русскоязычные строки и комментарии;
- `if __name__ == "__main__"` — условная конструкция, которая содержит основную исполняемую часть скрипта. Скрипт в Python может быть вызван двумя методами:
  - Как основная программа, если скрипт запускают напрямую из командной строки;
  - Как импортированный модуль: через команду `import` внутри другого файла.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
if __name__ == "__main__":
    # Код
```

Выполнить скрипт из командной строки просто:

```
python /путь_к_скрипту/имя_скрипта.py
```

Импортируем библиотеки **getopt** и **sys**. **getopt** считывает входные параметры, или **опции**. Библиотека **sys** импортирует системные функции. В скрипте пригодится её функция **sys.exit()**: она прервёт выполнение скрипта, если он будет запущен без входных параметров. Код скрипта:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# Импортируем нужные библиотеки
import sys
import getopt
```

```
if __name__ == "__main__":

    # Задаем формат входных параметров
    unixOptions = "s:e:"
    gnuOptions = ["start_dt=", "end_dt="]

    # Получаем строку входных параметров
    fullCmdArguments = sys.argv
    argumentList = fullCmdArguments[1:]

    # Проверяем входные параметры на соответствие формату,
    # заданному в unixOptions и gnuOptions
    try:
        arguments, values = getopt.getopt(argumentList, unixOptions, gnuOptions)
    except getopt.error as err:
        print (str(err))
        sys.exit(2)      # Прерываем выполнение, если входные параметры некорректны

    # Считываем значения из строки входных параметров
    start_dt = ''
    end_dt = ''
    for currentArgument, currentValue in arguments:
        if currentArgument in ("-s", "--start_dt"):
            start_dt = currentValue
        elif currentArgument in ("-e", "--end_dt"):
            end_dt = currentValue

    # Выводим результат
    print(start_dt, end_dt)
```

Разберём код подробнее. Вот импорт нужных библиотек:

```
import sys
import getopt
```

Задаём имена входных параметров:

```
unixOptions = "s:e:"
gnuOptions = ["start_dt=", "end_dt="]
```

- `unixOptions = "s:e:"` задаёт имена параметров в стиле классического **Unix** — семейства операционных систем, разработанных в 1970-х. Хотя этот стиль устарел, его добавляют как дань традиции. Без него скрипт не сработает.
- `gnuOptions = ["start_dt=", "end_dt="]` задаёт имена входных параметров в стиле **GNU** — Unix-подобной операционной системы.

Так, можно вызвать скрипт двумя способами — в разных стилях имён входных параметров:

```
python params_test.py -s '2019-01-01' -e '2019-09-01'
# или
python params_test.py --start_dt='2019-01-01' --end_dt='2019-09-01'
```

Рекомендуем придерживаться второго стиля: он лучше читается и обеспечивает совместимость параметров между программами — так вы не запутаетесь в названиях переменных.

Затем скрипт сохраняет набор входных параметров в переменной `argumentList` .

Входные параметры в нашем примере — это `start_dt` и `end_dt`. Система автоматически считывает их в переменную `sys.argv`, которую мы записываем в переменной `fullCmdArguments`. При сохранении в `argumentList` берём все параметры, кроме самого первого (с индексом 0). Этот параметр — имя самого скрипта, оно не понадобится:

```
 fullCmdArguments = sys.argv
 argumentList = fullCmdArguments[1:]
```

После этого скрипт проверяет, не пуст ли набор входных параметров. Если параметров не оказалось, выполнение программы прерывает команда `sys.exit(2)`. `(2)` означает, что ошибка, вызвавшая остановку

программы, допущена в параметрах командной строки. Например, входные параметры указаны неверно или не указаны совсем:

```
 try:
    arguments, values = getopt.getopt(argumentList, unixOptions, gnuOptions)
 except getopt.error as err:
    print (str(err))
    sys.exit(2)
```

Дальше скрипт перебирает все входные параметры и раскладывает их значения по своим внутренним переменным:

```
start_dt = ''
end_dt = ''
for currentArgument, currentValue in arguments:
   if currentArgument in ("-s", "--start_dt"):
      start_dt = currentValue
   elif currentArgument in ("-e", "--end_dt"):
      end_dt = currentValue
```

И, наконец, скрипт выводит на экран значения переменных `start_dt` и `end_dt`.

Передадим разобранному нами скрипту тестовые параметры:

```
python params_test.py --start_dt='2019-01-01' --end_dt='2019-09-01'
```

Результат:

```
['--start_dt=2019-01-01', '--end_dt=2019-09-01']
('2019-01-01', '2019-09-01')
```

Также в скрипт передают любые входные параметры: числа, логические значения, даты и время, массивы, названия файлов. Учтите, что параметры можно ввести только в виде строк. Их перевод в правильный формат целиком зависит от вас.

Запуск скрипта из командной строки в Яндекс.Облаке

Чтобы скрипт заработал на виртуальной машине, его нужно перенести с машины локальной.

Файлы на виртуальную машину копируют программой **scp**:

```
scp <путь_к_файлу_на_локальной_машине> <ЛОГИН>@<публичный_IP-адрес_виртуальной машины>:
```

Например, если вы работаете в Windows:

1. Создайте файл `test.py` и сохраните его в корневом каталоге диска C.

2. Скопируйте файл на виртуальную машину командой:

```
scp C:\test.py test_admin@84.201.144.4:
```

Двоеточие очень важно: оно означает «домашний каталог пользователя на виртуальной машине». Для других операционных систем команда отличается только обозначением пути к файлу. Изучите самостоятельно, как найти путь к нужному файлу в вашей ОС.

3. Введите пароль. Система покажет, что файл отправлен

4. Подключитесь к виртуальной машине:

```
ssh <ЛОГИН>@<публичный_IP-адрес_виртуальной машины>
```

5. После соединения введите команду **dir**. Система покажет содержимое вашей домашней папки:

6. Запустите тестовый скрипт командой:

```
python3 test.py
```

Запуск скрипта по расписанию

Скрипт — это программа, которая не только принимает данные на вход, но и может запускаться по расписанию. В операционных системах семейства *Unix* — например, Linux и MacOS — расписания создаёт программа **cron**. Она незаметно работает в фоновом режиме и читает специальное расписание **crontab**. В расписание можно добавлять любые команды, которые принимает командная строка.

Пример расписания:

```
5 6 * * 1 python -u -W ignore /home/my_user/script_A.py --start_dt=$(date +\%Y-\%m-\%d\ 00:00:00 -d "1 week ago") >> /home/my_u
#15 7 * * * python -u -W ignore /home/my_user/script_B.py --start_dt=$(date +\%Y-\%m-\%d\ 00:00:00 -d "1 week ago") >> /home/my
```

Разберём, что здесь происходит. Возьмём первую строку:

- `5 6 * * 1` — это определение времени запуска команды. Вот его формат:

`*` — любое значение.

Так, первая строка требует выполнять команду в 6:05 утра каждый понедельник.

- `python -u -W ignore /home/my_user/script_A.py` — сама команда, которая должна выполняться по расписанию. Здесь:
  - Флаг `-u` означает, что результаты выполнения скрипта не будут **буферизоваться**, или накапливаться в памяти компьютера. Они сразу запишутся в log-файл: о нём расскажем чуть позже. Например, если в вашем скрипте есть команды `print()`, их результат не будет аккумулироваться в памяти, а попадёт напрямую в логи.

- `-W ignore` означает, что любые предупреждения, которые появятся в процессе выполнения скрипта, не сохранятся в файле с логами.
- `/home/my_user/script_A.py` — название скрипта, который нужно выполнять по расписанию;

- `--start_dt=$(date +\%Y-\%m-\%d\ 00:00:00 -d "1 week ago")` — входной параметр скрипта. Здесь:
  - `$(date +\%Y-\%m-\%d\ 00:00:00 … )` — выражение командной строки, позволяющее получить текущую дату в формате `'%Y-%m-%d 00:00:00'`. Чтобы понять, как она работает, в командной строке выполните:

На экране появится текущая дата.

- `-d "1 week ago"` определяет временной интервал, который хотим отнять от текущей даты. Видимо, `script_A.py` спроектирован так, что должен получать на вход дату начала прошлой недели. Скорее всего, он собирает какую-то статистику за 7 последних дней. Есть и другие определения временных интервалов. Например:
  - `-d "yesterday"` или `-d "1 day ago"` — вчера;
  - `-d "N days ago"` — N дней назад;
  - `-d "N weeks ago"` — N недель назад;
  - `-d "1 month ago"` — один месяц назад;
  - `-d "N months ago"` — N месяцев назад;
  - `-d "1 year ago"` — один год назад;
  - `-d "N years ago"` — N лет назад;
- `>> /home/my_user/logs/script_A_$(date +\%Y-\%m-\%d).log` означает, что вся информация, которую скрипт выводит на экран, будет сохранена в файле `script_A_датаИсполнения.log` в папке `/home/my_user/logs/`. Сохранение логов при автоматическом выполнении скриптов очень важно: оно позволяет отлавливать ошибки, если они появляются. Без логов можно только гадать, всё ли хорошо с системой автоматизации.
- `2>&1` говорит о том, что все результаты выполнения скрипта (включая ошибки) будут выводиться в одно и то же место — его log-файл.

Текущее время определяется в соответствии с часовым поясом, установленным на машине, где настраивается cron. Большинство серверов работают с часовым поясом `UTC+0`. В этом же часовом поясе обычно сохраняют даты и время выполнения операций и проведения транзакций. Старайтесь привязывать свой анализ и запуск скриптов по расписанию к `UTC+0`, чтобы избежать расхождений результатов отчётов.

Посмотрим, как редактировать таблицу расписания `cron`. Проще всего это сделать на виртуальной машине. Но можно и локально, если у вас Linux или MacOS.

1. Сначала на локальной машине создадим файл, содержащий python-скрипт. С помощью `scp` перебросьте скрипт на виртуальную машину.
2. Подключитесь к виртуальной машине и выполните команду:

```
python cront_test.py
```

3. Создайте директорию `/home/YOUR_USER/logs` для хранения логов:

```
mkdir /home/YOUR_USER/logs
```

4. Выполните команду вызова редактора расписания `cron`:

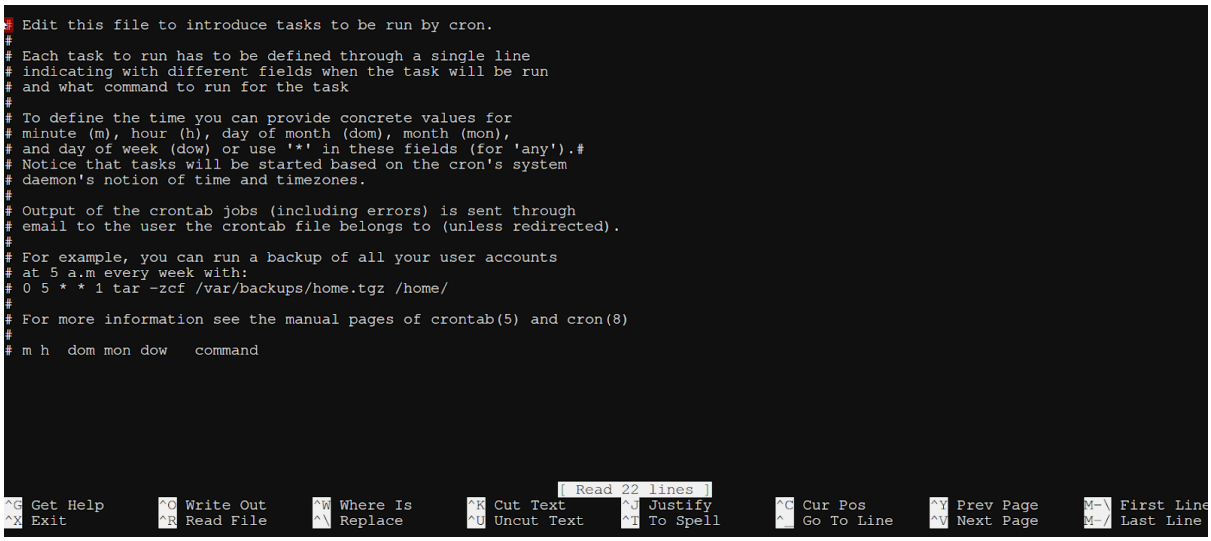
```
crontab -e
```

При первой редакции появится диалог выбора текстового редактора:

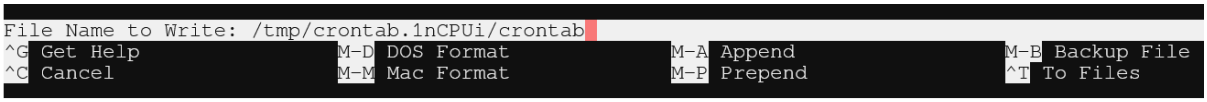
```
yandex_cloud@automation-test:~$ crontab -e
no crontab for yandex_cloud - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/nano        <---- easiest
  2. /usr/bin/vim.basic
  3. /usr/bin/vim.tiny
  4. /bin/ed
```

Нажмите 1 и *Enter* — это выберет *nano* в качестве редактора для *crontab*.

После этого появится текст *crontab*:



В конце файла добавьте строку, описывающую расписание. Нажмите `Ctrl+O` — команда сохранения. Появится системное сообщение:



Нажмите *Enter*. Затем `Ctrl+X`, чтобы выйти из редактора *crontab*. Появится сообщение:

```
15:25 ~ $ crontab -e
crontab: installing new crontab
15:48 ~ $
```

Значит, новые настройки таблицы *cron* вступили в силу. Единственный способ увидеть, что скрипт срабатывает по расписанию — периодически просматривать последний лог исполнения.