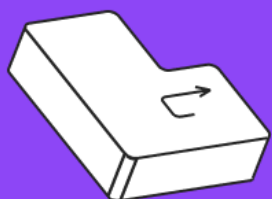


Автоматизация тестирования консольных приложений Linux на Python

Лекция 4
Библиотека Paramiko, деплой
и тестирование приложения
на удаленной машине по ssh



Оглавление

Введение

Термины

Включение и настройка SSH и SFTP

Библиотека paramiko

Управление пакетами Ubuntu

Автодеплой

 Пример. Автодеплой для архиватора 7z

Проверка журналов (логов)

Интеграция с Jenkins

Итоги

Что можно почитать еще?

Используемая литература

Введение

Ранее в этом курсе вы узнали:

- Зачем писать автотесты на Python.
- Какие виды тестов можно автоматизировать.
- Как вызывать из Python команды операционной системы.
- Как писать автотесты с использованием Pytest.
- Как пользоваться фикстурами в Pytest.
- Как формировать отчеты в Pytest.

На этом уроке мы рассмотрим автодеплой (автоматическую установку), работу с логами в Linux и встраивание автотестов в системы непрерывной интеграции.

Это заключительная лекция курса. На ней мы разберем инструменты для организации автотестирования консольных приложений Linux в комплексе.

В реальной работе эта тема очень важна: без знания этих инструментов не получится развернуть законченную систему автотестов для бизнеса.

На этой лекции вы узнаете:

- Как настроить ssh-сервер в Linux.
- Как работать с библиотекой Paramiko.
- Как устанавливать и удалять программы в Ubuntu Linux.
- Как реализовать автодеплой.
- Как работать с журналами Linux.
- Как настроить автозапуск тестов в Jenkins.



Для более детального понимания последовательности работы с примерами посмотрите видеолекцию.

Термины

SSH (Secure Shell, безопасная оболочка) — сетевой протокол прикладного уровня, позволяющий производить удаленное управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов).

SFTP (SSH File Transfer Protocol) — протокол прикладного уровня передачи файлов, работающий поверх безопасного канала. Предназначен для копирования и

выполнения других операций с файлами поверх надежного и безопасного соединения.

Непрерывная интеграция (Continuous Integration, CI) — практика разработки ПО, при которой происходит постоянное (до нескольких раз в день) слияние рабочих копий в основную ветвь разработки и выполнение частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

Автодеплой — развертывание при помощи автоматизированных решений.

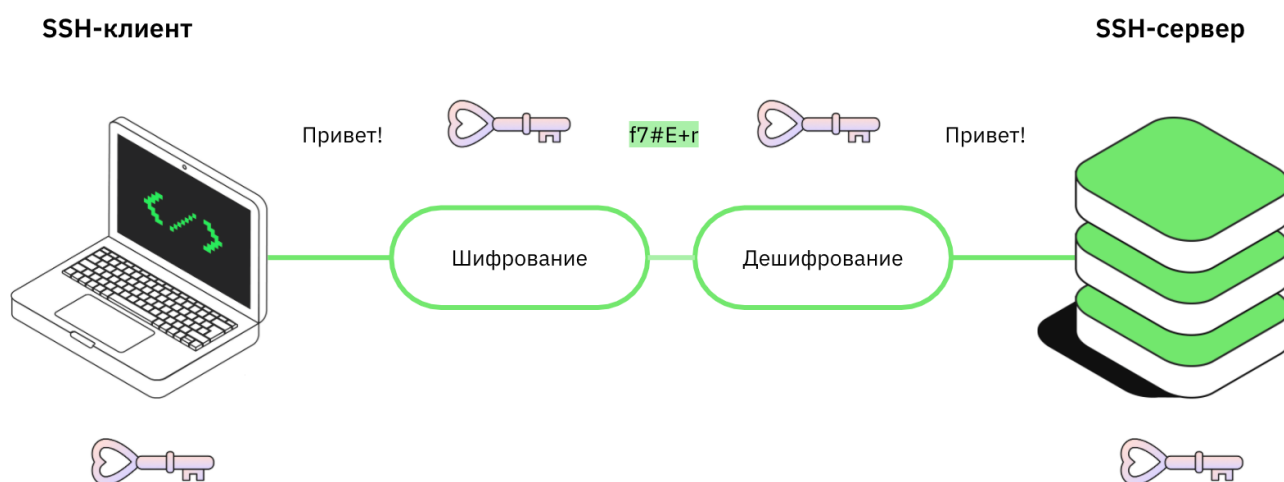
Включение и настройка SSH и SFTP

SSH — защищенный протокол для удаленного доступа к компьютерам. Через SSH можно выполнять операции в командной строке компьютера, который физически находится в другом месте.

Иными словами, SSH — это дистанционная командная строка. Визуально вы работаете на своем компьютере, а в реальности — на другом.

Для подключения к удаленной машине по SSH нужны клиент и сервер — специальная программа. В *nix-подобных системах (Linux, macOS) клиент обычно установлен в системе по умолчанию, поэтому достаточно просто открыть терминал.

Для работы по SSH нужен SSH-сервер и SSH-клиент. Сервер прослушивает соединения от клиентских машин, а при установлении связи производит аутентификацию, после чего начинается обслуживание клиента. Клиент используется для входа на удаленную машину и для выполнения команд.



Чтобы подключиться, нужно указать адрес сервера и, опционально, имя пользователя и порт. Вот как выглядит команда при использовании консольного клиента (в терминале):

```
💡 ssh username@remote_host -p port
```

Например, для подключения к серверу 52.307.169.244 в аккаунт sergey нужно ввести:

```
💡 ssh sergey@52.307.169.244
```

Используемый порт задается при настройке SSH-сервера. Если не указать порт, используется порт SSH по умолчанию — 22.

Простейший вариант — подключение по паролю. После ввода команды ssh система запросит пароль:

```
💡 sergey@52.307.169.244's password:
```

Пароль придется вводить каждый раз.

Чтобы подключаться по SSH (и многим другим сервисам) без пароля, можно использовать ключи. Нужно создать пару ключей: приватный (закрытый) и публичный (открытый).

- **Приватный ключ** нужно хранить и никому не показывать.
- **Публичный ключ** можно показывать всем и распространять свободно.


Эти ключи связаны друг с другом таким образом, что, зашифровав информацию одним ключом, расшифровать ее можно только другим.

SFTP, протокол передачи файлов по SSH или безопасный протокол передачи файлов, — это отдельный протокол, поддерживающий SSH. Его преимущество — возможность использовать защищенное подключение для передачи файлов и просмотра файловой системы как на локальной, так и на удаленной системе.

Если нужно загрузить файлы с удаленного хоста, можно воспользоваться командой:


```
💡 get remoteFile
```

Передать файлы в удаленную систему поможет команда put:


 `put localFile`

Установить SSH на Ubuntu просто. Программа считается стандартной и используется почти везде. Хотя по умолчанию в дистрибутиве ее нет, зато она есть в официальных репозиториях.


Для установки откройте терминал с помощью сочетания клавиш **Ctrl+Alt+T** и выполните команду:

 `sudo apt install openssh-server`


Загрузится несколько пакетов. Когда установка SSH сервера Ubuntu завершится, программа будет готова к работе. Если вы хотите, чтобы служба запускалась автоматически, нужно добавить ее в автозагрузку. Поэтому, чтобы включить SSH в Ubuntu, выполните:

 `sudo systemctl enable sshd`


Клиент SSH уже установлен в системе по умолчанию. Сейчас вы можете попробовать подключиться к локальному SSH серверу:

 `ssh localhost`


Все настройки сервера SSH хранятся в конфигурационном файле `sshd_config`, который находится в папке `/etc/ssh`.

 `sudo vi /etc/ssh/sshd_config`

Чтобы не разворачивать вторую тестовую машину, выполним подключение к своей же машине по адресу `0.0.0.0`, только от имени другого пользователя. Для этого создадим пользователя `user2`:

 `sudo useradd user2 -m`

И зададим ему пароль:

 `sudo passwd user2`

Библиотека paramiko

Paramiko — это модуль для Python, который реализует SSH-протокол для защищенного соединения с удаленным компьютером. При подключении предоставляется высокоуровневое API для работы с SSH — создается объект `SSHClient`. Для большего контроля можно передать сокет (или подобный объект) классу `Transport` и работать с удаленным хостом в качестве сервера или клиента. Клиенту для аутентификации можно использовать пароль или приватный ключ и проверку ключа сервера.

Основной класс для подключения и удаленной работы — `SSHClient`. Он предоставляет нам «сессию», с которой мы можем работать далее.

Для передачи файлов по SFTP можно использовать класс `Transport`. Для непосредственной передачи файлов используются команды `put` и `get`.

Реализуем методы работы с SSH.

Скринкаст. Реализация `ssh_checkout` и `ssh_put`

```
1 import paramiko
2
3 def ssh_checkout(host, user, passwd, cmd, text, port=22):
4     client = paramiko.SSHClient()
5     client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
6     client.connect(hostname=host, username=user, password=passwd,
7                     port=port)
8     stdin, stdout, stderr = client.exec_command(cmd)
9     exit_code = stdout.channel.recv_exit_status()
10    out = (stdout.read() + stderr.read()).decode("utf-8")
11    client.close()
12    if text in out and exit_code == 0:
13        return True
14    else:
15        return False
```

```

1 def upload_files(host, user, passwd, local_path, remote_path,
  port=22):
2     print(f"Загружаем файл {local_path} в каталог {remote_path}")
3     transport = paramiko.Transport((host, port))
4     transport.connect(None, username=user, password=passwd)
5     sftp = paramiko.SFTPClient.from_transport(transport)
6     sftp.put(local_path, remote_path)
7     if sftp:
8         sftp.close()
9     if transport:
10         transport.close()

```

```

1 def download_files(host, user, passwd, remote_path, local_path,
  port=22):
2     print(f"Скачиваем файл {remote_path} в каталог {local_path}")
3     transport = paramiko.Transport((host, port))
4     transport.connect(None, username=user, password=passwd)
5     sftp = paramiko.SFTPClient.from_transport(transport)
6     sftp.get(remote_path, local_path)
7     if sftp:
8         sftp.close()
9     if transport:
10         transport.close()

```

Управление пакетами Ubuntu

Каждый дистрибутив Linux поставляется с определенным менеджером пакетов. Для дистрибутива Ubuntu менеджер пакетов по умолчанию **apt**, **apt-get** или графический **Software Center**. В свою очередь, эти менеджеры пакетов полагаются на инструмент низкого уровня для управления пакетами программного обеспечения **dpkg**.


Команда **dpkg** позволяет устанавливать и удалять программное обеспечение, а также выполнять некоторые другие действия с пакетами.

Вы можете использовать **dpkg** для установки программного обеспечения:

💡 `dpkg -i package_name.deb`

Команда сработает, если на вашем компьютере будет заранее скачанный файл **.deb**.

Чтобы перечислить или отобразить установленные пакеты в дистрибутиве Ubuntu, можно использовать команду:


 `dpkg -l search_pattern`

Вывод выглядит так:

```
ii 0traced 0.01-3 amd64 A traceroute tool that can run wi
ii aapt 1:7.0.0+r33- amd64 Android Asset Packaging Tool
ii acccheck 0.2.1-3 all Password dictionary attack tool f
ii accountsservic 0.6.45-1 amd64 query and manipulate user account
ii ace-voip 1.10-1kali5 amd64 A simple VoIP corporate directory
ii acl 2.2.52-3+b1 amd64 Access control list utilities
ii adduser 3.116 all add and remove users and groups
ii adwaita-icon-t 3.26.1-3 all default icon theme of GNOME
ii afflib-tools 3.7.16-2 amd64 Advanced Forensics Format Library
ii aglfn 1.7-3 all Adobe Glyph List For New Fonts
ii aha 0.4.10.6-4 amd64 ANSI color to HTML converter
ii aircrack-ng 1:1.2-0~rc4- amd64 wireless WEP/WPA cracking utiliti
ii albatross-gtk- 1.7.4-1 all dark and light GTK+ theme from th
ii alsa-tools 1.1.3-1 amd64 Console based ALSA utilities for
ii amap 5.4-4 amd64 next-generation scanning tool for
ii amd64-microcod 3.20171205.1 amd64 Processor microcode firmware for
ii anacron 2.3-24 amd64 cron-like program that doesn't go
```


Если вы хотите увидеть все установленные пакеты, опустите [search_pattern].

Если вы установили пакет, но больше не используете, его можно удалить:


 `dpkg -r package_name.deb`

Эта команда удаляет весь пакет, кроме файлов конфигурации.


Чтобы отобразить содержимое пакета, используйте команду:

 `dpkg --contents package_name.deb`


Если нужно проверить, установлен ли на компьютере определенный пакет, используйте команду:

 `dpkg -s package_name.deb`

Если нужно узнать, где будет установлен пакет, используйте ключ -L:

 `dpkg -L package_name.deb`

Чтобы посмотреть подробную информацию о пакете, введите:

 `dpkg -p package_name.deb`

Если у вас несколько файлов deb, можно установить их все за один раз с помощью команды:

```
💡 dpkg -R --install /deb-files-location/
```

Обратите внимание: для этого нужно поместить все файлы .deb в одну папку.

Вам может понадобится распаковать пакет deb, чтобы вы могли вносить изменения в свои файлы. Для распаковки используйте команду:

```
💡 dpkg --unpack package_name.deb
```

После внесения изменений в файлы используйте эту команду, чтобы настроить и переупаковать его в файл deb для установки:

```
💡 dpkg --configure package_name
```

Автодеплой

Развертывание при помощи автоматизированных решений называется автодеплойем.

Преимущества автодеплоя:

- развертывание сразу в нескольких средах;
- повышение надежности и предсказуемости развертываемого ПО;
- минимизация ошибок, связанных с человеческим фактором;
- снижение затрат на развертывание;
- упрощение рабочего процесса.



Вопрос

Подумайте, что должен проверять автодеплой?



Ответ:

- код возврата и сообщение установщика;
- наличие в списке пакетов.

Пример. Автодеплой для архиватора 7z

Реализуем автодеплой для архиватора 7z. Чтобы не разворачивать вторую тестовую машину, подключаться будем к своей машине по адресу 0.0.0.0, но от имени другого пользователя — user2. Будем считать, что необходимый пакет скачан заранее. Нам нужно выполнить следующие операции:

1. Загрузить пакет в домашний каталог пользователя user2.

2. Выполнить `dpkg -i`, чтобы установить пакет.
3. Выполнить `dpkg -s`, чтобы проверить, установлен ли пакет.

В последнем случае мы будем искать в выводе команды фразу **Status: install ok installed**.

Для установки пользователь `user2` должен обладать правами администратора. Добавим его в группу `sudo`:

💡 `sudo usermod -aG sudo user2`

Осталась проблема: как передать пароль при выполнении нашей команды. Сделать это можно с помощью конструкции:

💡 `echo 'пароль' | sudo -S команда`


```
1 from sshcheckers import ssh_checkout, upload_files
2
3 def deploy():
4     res = []
5     upload_files("0.0.0.0", "user2", "11", "p7zip-full.deb", "/home/
    /user2/p7zip-full.deb")
6     res.append(ssh_checkout("0.0.0.0", "user2", "11", "echo '11' |
    sudo -S dpkg -i /home/user2/p7zip-full.deb",
7                             "Настраивается пакет"))
8     res.append(ssh_checkout("0.0.0.0", "user2", "11", "echo '11' |
    sudo -S dpkg -s p7zip-full",
9                             "Status: install ok installed"))
10    return all(res)
11
12 if deploy():
13     print("Деплой успешен")
14 else:
15     print("Ошибка деплоя")
```

Проверка журналов (логов)

Журналы — один из самых важных источников информации при возникновении ошибок в Linux.


В современных дистрибутивах Linux для просмотра логов определенного сервиса или загрузки системы нужно использовать утилиту **journalctl**.

Основная команда для просмотра:

 `journalctl`

Она выведет все записи из всех журналов, включая ошибки и предупреждения, начиная с того момента, когда система начала загрузаться.

Система записывает события с разными уровнями важности: какие-то из них могут быть предупреждением, которое можно проигнорировать, какие-то могут быть критическими ошибками. Часто бывает нужно выполнить фильтрацию событий в журнале по важности. Если мы хотим просмотреть только ошибки, игнорируя другие сообщения, введем команду с указанием кода важности:


 `journalctl -p 0`

Для уровней важности, приняты следующие обозначения:


- **0: emergency** — неработоспособность системы
- **1: alerts** — предупреждения, требующие немедленного вмешательства
- **2: critical** — критическое состояние
- **3: errors** — ошибки
- **4: warning** — предупреждения
- **5: notice** — уведомления
- **6: info** — информационные сообщения
- **7: debug** — отладочные сообщения

Если указать код важности, `journalctl` выведет все сообщения с этим кодом и выше. Например, если мы укажем опцию `-p 2`, `journalctl` покажет все сообщения с уровнями 2, 1 и 0.

С помощью опции **--since** можно указать дату и время, начиная с которого нужно отображать логи:

 `sudo journalctl --since "2023-01-20 15:10:10"`

С помощью опции **--until** можно указать, до какой даты вы хотите получить информацию:

 `sudo journalctl -e --until "2023-01-20 15:15:50"`

Можно скомбинировать эти опции, чтобы получить логи за нужный период:



```
sudo journalctl --since "2023-01-20 15:10:10" --until "2023-01-20 15:15:50"
```

Интеграция с Jenkins

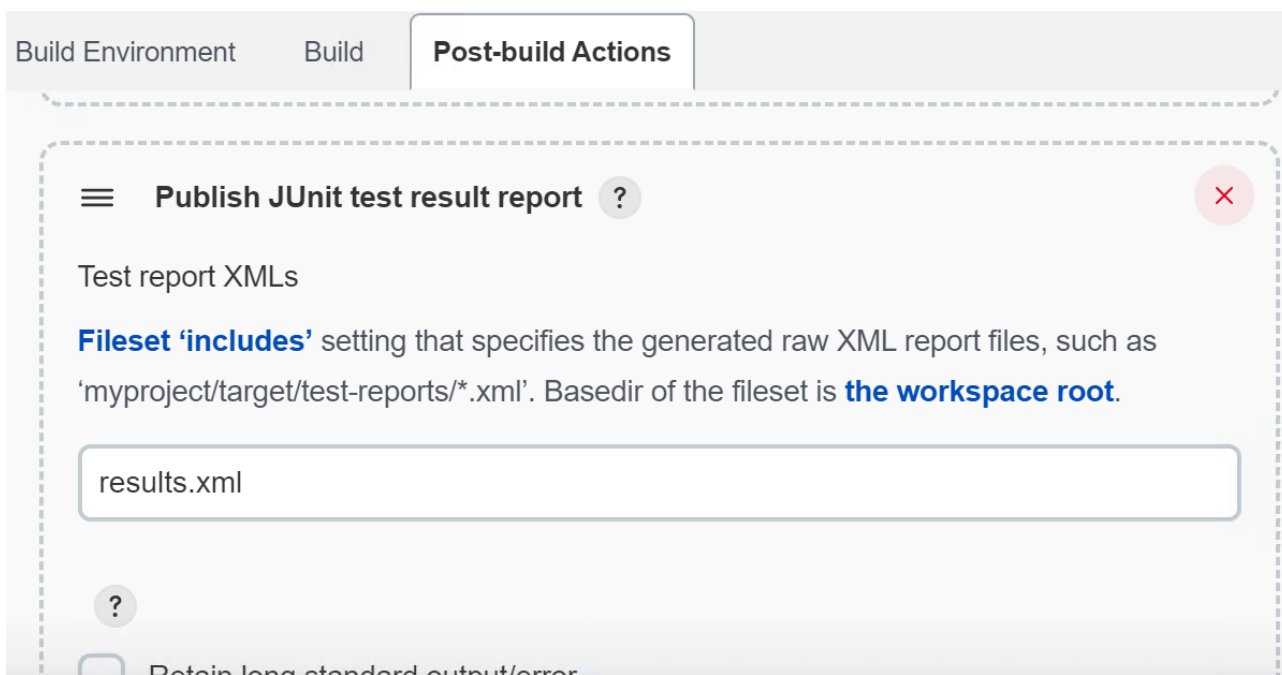
Системы непрерывной интеграции (CI), такие как Jenkins, часто используются для запуска тестовых прогонов после каждой фиксации кода. В Pytest есть опции для генерации файлов в формате junit.xml, которые требуются Jenkins и другим системам CI для отображения результатов тестирования.

Jenkins — это сервер автоматизации с открытым исходным кодом, который часто используется для непрерывной интеграции. Распространенная практика — использование Jenkins или других систем непрерывной интеграции для автоматизации запуска и создания отчетов по проектам Python.

Настроить запуск тестов в Jenkins можно несколькими способами. Самый простой — добавление запуска тестов в разделе Post-build Actions сборки.

Можно сохранить команды перехода в рабочий каталог и запуска Pytest в скрипт и запустить его, вызвав команду **bash -e** в шаге с типом **Execute shell**.

А затем добавить шаг публикации отчета формата **junit**, указав имя файла, который будет сформирован.



Итоги

На этой лекции вы рассмотрели, как работать с инструментами для организации процесса автотестирования консольных приложений Linux в комплексе.

Вы узнали:

- Как настроить SSH-сервер в Linux.
- Как работать с библиотекой Paramiko.
- Как устанавливать и удалять программы в Ubuntu Linux.
- Как реализовать автодеплой.
- Как работать с журналами Linux.
- Как настроить автозапуск тестов в Jenkins.

Этих знаний достаточно, чтобы с нуля развернуть тестовый проект.

Что можно почитать еще?

1. [Обзор пакетных менеджеров Linux](#)
2. [Как установить Jenkins и настроить автоматическую сборку maven-проекта на Ubuntu 20.04](#) — в статье описана работа с Jenkins
3. [Использование journalctl для просмотра и анализа логов: подробный гайд](#) — статья про работу с journalctl
4. [Python Testing with pytest, Second Edition](#) — хорошая книга про Pytest, достаточно нескольких первых глав (на английском).

Используемая литература

1. [Python Testing with pytest, Second Edition by Brian Okken](#)
2. [Документация paramiko](#)
3. [Документация Jenkins](#)