1. **Установка Python.**

**Установка Python в Windows.**

Cкачать дистрибутив с официального сайта, перейдя по ссылке

<https://www.python.org/downloads/>

Предлагается два варианта Install Now и Customize installation. При выборе Install Now, Python установится в папку по указанному пути. Помимо самого интерпретатора будет установлен IDLE (интегрированная среда разработки), pip (пакетный менеджер) и документация, а также будут созданы соответствующие ярлыки и установлены связи файлов, имеющие расширение .py с интерпретатором Python. Customize installation – это вариант настраиваемой установки. Опция Add python 3.5 to PATH нужна для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке.

Дополнительные опции в процессе установки:

* Install for all users – Установить для всех пользователей. Если не выбрать данную опцию, то будет предложен вариант инсталляции в папку пользователя, устанавливающего интерпретатор.
* Associate files with Python – Связать файлы, имеющие расширение .py, с Python. При выборе данной опции будут внесены изменения в Windows, позволяющие запускать Python скрипты по двойному щелчку мыши.
* Create shortcuts for installed applications – Создать ярлыки для запуска приложений.
* Add Python to environment variables – Добавить пути до интерпретатора Python в переменную PATH.
* Precomple standard library – Провести прекомпиляцию стандартной библиотеки.

**Установка Python в Linux**

Чаще всего интерпретатор Python уже в входит в состав дистрибутива. Это можно проверить набрав в терминале

*> python*

или

*> python3*

В первом случае, вы запустите Python 2 во втором – Python 3. В будущем, скорее всего, во всех дистрибутивах Linux, включающих Python, будет входить только третья версия. Если у вас, при попытке запустить Python, выдается сообщение о том, что он не установлен, или установлен, но не тот, что вы хотите, то у вас есть два пути: а) собрать Python из исходников; б) взять из репозитория.

Для установки из репозитория в Ubuntu воспользуйтесь командой:

*> sudo apt-get install python3*

**Проверка интерпретатора Python**

Для начала протестируем интерпретатор в командном режиме. Если вы работаете в Windows, то нажмите сочетание Win+R и в появившемся окне введите:

*python*

В Linux откройте окно терминала и в нем введите:

*python3* (или *python*).

В результате Python запустится в командном (интерактивном) режиме

Для выхода из интерактивного режима, наберите команду:

*exit()*

1. **Установка виртуального окружения (virtualenv).**

Оно позволяет проектам, которые имеют различные (и часто конфликтующие) зависимости (например, на одном проекте Джанга 1.10, на втором - 1.8 , а на третьем - 1.5, да и в придачу Питон 2-й версии), сосуществовать на одной системе.

Для **установки** достаточно выполнить:

*pip install virtualenv*

**Создаем виртуальное окружение** внутри папки будущего проекта:

Меняем диск, если нужно: *cd /d b:*

Создаем папку проекта, например, так:

*mkdir <name>*

Заходим в папку проекта с помощью:

*cd <name>*

Внутри папки будущего проекта набираем:

*virtualenv < имя виртуального окружения>* или:

*?python virtualenv < имя виртуального окружения>* или:

*python -m venv < имя виртуального окружения>*

После того как вы установили новое окружение, у вас будет следующая структура:

~/**<venv\_name>**

├── bin

│   ├── activate

│   ├── easy\_install

│ ├── pip

│ ├── python

│   └── ...

├── include

│   └── python2.7 -> /usr/include/python2.7

├── lib

│   └── python2.7

│   ├── distutils

│   │   └── ...

│   ├── site-packages

│ │ └── ...

│   └── ...

└── local

├── bin -> /home/username/venv/**<venv\_name>**/bin

├── include -> /home/username/venv/**<venv\_name>**/include

└── lib -> /home/username/venv/**<venv\_name>**/lib

В результате внутри папки /venv\_name/ создастся маленькая рабочая среда с папками bin/, include/, lib/, local/, куда будет перенесен «набор джентльмена» для работы — python, менеджеры пакетов pip и easy\_install. Сюда же могут доставляться все необходимые пакеты, фреймворки (в том числе Django) и утилиты. В пределах каждого виртуального окружения они будут изолированы друг от друга, не оказывая никакого взаимного «паразитного» влияния.

Для **активации** виртуального окружения нужно зайти в его папку (*cd*) и выполнить:

*<имя виртуального окружения>\scripts\activate*

После активации командная строка изменится: перед именем пользователя появится название виртуального окружения в скобках:

*(<имя виртуального окружения>)имя\_пользователя>@имя\_компьютера ~».*

Теперь любые команды по установке пакетов (например, «pip install django») или по их удалению будут выполнятся только в пределах активированного окружения.

Для установки пакетов в окружение надо активировать окружение и запустить:

*pip <something>* или:

*easy\_install <something>*

Для **выхода** из виртуального окружения и перехода в обычный режим достаточно набрать:

*deactivate*

1. **Установка Django.**

Перед этим нужно удостовериться, что установлена последняя версия pip — менеджера пакетов Python, программы, которуа используется для установки Django.

*pip install --upgrade pip*

Затем запусти команду, чтобы установить Django:

*pip install django*

1. **Структура проекта и его создание**

Проекты имеют простую структуру, состоящую из 5 папок:

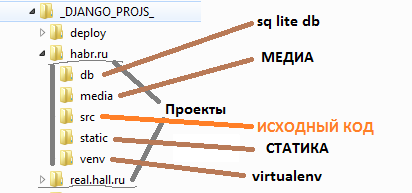
src — для исходников

venv — для виртуального окружения

static — для всей статики (для команды collectstatic при деплое(?))

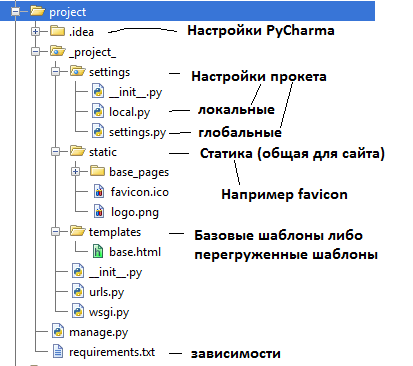
media — для меди файлов

db — для хранения БД



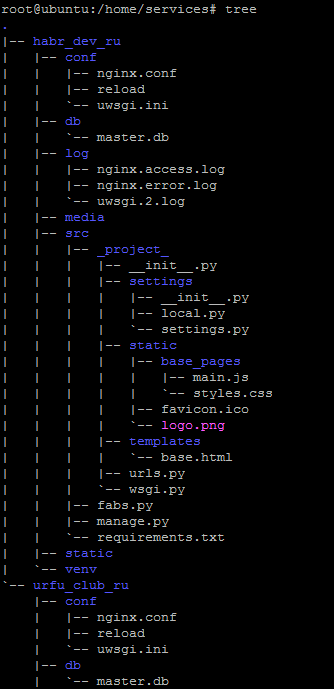
Кроме проектов в каталоге есть папка deploy, в которой хранится fabfile.py скрипт, который и инициализирует новый проект, так же в ней находятся конфиги для деплоя и шаблон нового проекта.

Структура папки**src**, поскольку все исходные коды проекта будут находиться в ней, она же будет git репозиторием.

  
**В корне** лежат: папка с настройками PyCharma и папка **\_project\_**, файлы: **.gitignore** (для git`a) и **requirements.txt** (зависимости проекта).

В \_project\_ находятся настройки проекта и все то, что специфично именно для данного проекта (шаблоны, статика, url.py). ([подробнее о структуре Django проекта и requirements.txt](http://www.deploydjango.com/django_project_structure/index.html)) Каталог static в папке \_project\_ нужен для хранения статических файлов специфичных для данного проекта (например favicon), не путайте его с каталогом static в корне, который нужен для команды collectstatic.

Как это выглядит на сервере:



**Создаём новый проект Django.**

В Windows запустить следующую команду **из** каталога **scr**:

*…\src>django-admin.exe startproject balealhome .*

— это скрипт, который создаст необходимую структуру директорий и файлы.

Точка . в конце говорит скрипту установить Django в текущем каталоге (который и обозначается сокращённо точкой.)

Теперь у проекта должна быть следующая структура:

scr

├───manage.py

└─── balealhome

settings.py

urls.py

wsgi.py

\_\_init\_\_.py

**Изменяем настройки** balealhome/**settings.py** (настройки БД см. в MySQL):

Часовой пояс из поля TZ на сайте https://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones

*TIME\_ZONE = 'Europe/Moscow'*

# информация о расположении статических файлов: в конец файла после переменной STATIC\_URL добавь новую – по умолчанию:

*STATIC\_URL = '/static/'*

*STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')*

# фильтр datе в шаблоне для объектов datetime не понимает пробел после двоеточия и двойные кавычки! (хз, почему), писать в одиночных и без пробела:

{{ dat|date:'j E Y, l, H:i (T)' }}

ALLOWED\_HOSTS = []

**Локализация и интернационализация:**

*LANGUAGE\_CODE = 'en-us'*

*USE\_I18N = True*

*USE\_L10N = True*

*USE\_TZ = True*

**Режим отладки:**

DEBUG = True

1. **Создание рабочей базы данных.**

Если вы работаете над чем-то существенным, рекомендуется выбрать ту же базу данных, которая будет использована на “боевом” сервере.

**Устанавливаем сервер базы данных MySQL.** Заходим на сайт <https://dev.mysql.com/downloads/mysql/> и скачиваем [сервер](http://dev.mysql.com/downloads/mysql/), устанавливаем и запускаем.

Думаю, никаких проблем с созданием локального сервера у вас возникнуть не должно.

**Запускаем** сервер базы данных.

Указываем данные для подключения к локальной БД MySQL: localhost: **root/пароль**.

**Создаем новую базу данных** - schema:`balealhome` , collation: utf8\_general\_ci

**Установка драйверов** для подключения к **Python.**

При использовании MySQL понадобится [DB API драйвер](https://djbook.ru/rel1.8/ref/databases.html#mysql-db-api-drivers), такой как mysqlclient. Для начала, драйвера нужно скачать [отсюда](http://web-proger.ru/mysqlclient-1.3.9-cp34-cp34m-win32.whl): <https://pypi.python.org/pypi/mysqlclient>, важно обратить вниманиена **соответствие разрядности версии ОС** 32 или 64.

**Файл**, например, mysqlclient-1.3.12-cp36-cp36m-win\_amd64.whl **скачать в каталог с Django**.

В консоли Windows, переходите в каталог с Django и вводите команду:

*(tenv) b:\Sitio\jango>pip install mysqlclient-1.3.12-cp36-cp36m-win\_amd64.whl*

Если все пройдет успешно, консоль выдаст оповещение

http://web-proger.ru/wp-content/uploads/2016/12/%D0%A1%D0%BA%D1%80%D0%B8%D0%BD%D1%88%D0%BE%D1%82-26-12-2016-145055.png

Если ошибка "is not a supported wheel on this platform" Проверить разрядность (32, 64) у ОС и у интерпретатора. Должна быть в обоих случаях 64.

Чтоб Django понял, что он должен использовать именно MySQL, а не SQLite, **редактируем mysite/settings.py**:

*DATABASES = {*

*'default': {*

Указываем Django, какую БД использовать MySQL:

*'ENGINE': 'django.db.backends.mysql',*

Указываем имя нашей БД, которую мы указывали при создании:

*'NAME': 'balealhome',*

Указываем имя пользователя:

*'USER': 'root',*

Указываем пароль от нашей БД

*'PASSWORD': 'Disel260879',*

Поля HOST и PORT лучше не изменять:

*'HOST': '127.0.0.1',*

*'PORT': '3306',*

}

}

**Cоздаем базу данных для** проекта:

*python manage.py migrate*

Если вы планируете использовать команду manage.py migrate для автоматической генерации таблиц БД для ваших моделей (после первого запуска Django и создания проекта), вам нужно убедиться, что Django имеет разрешение на создание и изменение таблиц БД; если же вы хотите создавать таблицы вручную, вам надо просто предоставить Django разрешения SELECT, INSERT, UPDATE и DELETE. После создания пользователя БД с указанными привилегиями, вам понадобится изменить настройки вашего проекта, см. [DATABASES](https://djbook.ru/rel1.8/ref/settings.html#std:setting-DATABASES).

Если вы используете [среду тестирования Django](https://djbook.ru/rel1.8/topics/testing/index.html) для проверки запросов к базе данных, Django понадобится разрешение на создание тестовой БД.

1. **Установка и настройка системы VCS: Git**

**Установка git для Windows:**

Рекомендуется официальный дистрибутив, содержащий в себе графическую оболочку и консоль. Скачать git клиент можно  с официального сайта [git-scm.com](https://git-scm.com/). Можно нажимать "дальше, дальше, дальше" на всех этапах установки за исключением одного: на пятом шаге, который называется "Adjusting your PATH environment" (Настройка системной переменной Path), выбери "Use Git and optional Unix tools from the Windows Command Prompt" (Запуск Git и соответствующих Unix утилит через командную строку Windows, нижняя опция). Все остальные настройки можно оставить по умолчанию. Также неплохо будет выбрать опцию "Checkout Windows-style, commit Unix-style line endings".

После окончания установки - перезапустить командную строку или powershell.

Консоль Git Bash – конфигурить Git нужно в ней, т. к. консоль Windows – cmd не поддерживает некоторые команды Git.

**Установка git для Linux:**

Откройте терминал и введите команду sudo apt-get install git

**Создание** **учётной записи на GitHub**. Перейди на [GitHub.com](https://www.github.com/) и зарегистрируй новый бесплатный аккаунт.

Перед началом работы настоятельно рекомендуется произвести инициализацию вас как пользователя.

*git config --global user.name "Ваше имя"*

*git config --global user.email ваш@mail.ru*

После указания имени и email ваши данные станут публичными. Все ваши действия будут помечены именем и почтой.

Для **инициализации репозитория**, в консоли необходимо ввести команду *git init,* предварительно выбрав директорию для работы.

В ответ мы должны получить *Initialized empty Git repository in [путь]* - это означает что git репозиторий был успешно инициализирован и в корне проекта была создана скрытая папка с названием .git в которой хранятся git настройки и прочие служебные файлы.

Создаем файл **.gitignore** в корневом каталоге репозитория. Настройки в файле .gitignore позволяют исключить из списка файлов сканируемых GIT, все то, что отслеживать не надо. Обычно это так называемые юзер-файлы, изображения, архивы, документация и т.п.

Пример файла .gitignore:

*# комментарий — эта строка игнорируется*

*# не обрабатывать файлы, имя которых заканчивается на .a*

*\*.a*

*# НО отслеживать файл lib.a, несмотря на то, что мы игнорируем все .a файлы с помощью предыдущего правила*

*!lib.a*

*# игнорировать только файл TODO находящийся в корневом каталоге, не относится к файлам вида subdir/TODO*

*/TODO*

*# игнорировать все файлы в каталоге build/*

*build/*

*# игнорировать doc/notes.txt, но не doc/server/arch.txt*

*doc/\*.txt*

*# игнорировать все .txt файлы в каталоге doc/*

*doc/\*\*/\*.txt*

**Сommit**

Вся суть использования git в commits (коммиты). Коммит – это так называемый снимок состояния проекта на определенном этапе. После каждого значимого изменения, например, исправление бага в функции, нужно делать комит.

Прежде чем коммитить, надо добавить измененные файлы в индекс, так называемое временное хранилище изменений. **Проиндексировать** все обновленные и новые файлы:

*git add --all .*

*-* именно с точкой в конце и пробелом перед ней.

*git status* – основная команда, отслеживающая **состояние файлов**. Она показывает есть ли изменения в отслеживаемых файлах или наличие не отслеживаемых файлов. Отслеживаемые файлы – это те файлы, которые есть в предыдущем коммите. Если же мы добавили в проект новый файл, то он будет считаться не отслеживаемым.

Если вы случайно проиндексировали не нужный файл, то удалить его из индекса поможет команда git reset, например **удалить файл** README **из индекса**: *git reset README.*

Если вы передумали оставлять изменения внесенные в файл, выполните команду: *git checkout -- имя файла* и он вернется в то состояние, в котором находился во время последнего коммита, только учтите, что все изменения в этом файле исчезнут.

**Создание коммита.**

*git commit –m “ комментарий в кавычках”*

Есть возможность проиндексировать все изменения и сделать коммит одной командой:

*git commit –a –m “комментарий в кавычках”*

-a означает: добавить все изменения в индекс до передачи.

-m: комментарий.

Историю изменений можно посмотреть командой: *git log.*

*git show* покажет только последние изменения. Для выхода из режима просмотра нажмите q.

Если вам нужно **отменить** последний **коммит**, используйте команду:

*git revert HEAD*

Отсчет комитов ведется с 0 начиная с последнего, например, если нужно отменить третий коммит, то следует выполнить:

*git revert HEAD~2*

*git reset --soft HEAD~3* удалит 3 последних коммита и **откатит проект** в состояние четвертого коммита, при этом сохранив в индексе все изменения последних трех коммитов.

*git reset -- hard HEAD~3* **полностью удалит** три последних коммита.

Теперь, когда в локальном репозитории были проведены необходимые изменения, их можно **загрузить в удаленный репозиторий**. Делается это выполнением команды:

*git push origin master*

При этом потребуется ввести логин и пароль от github. Однако загрузки может не произойти. Причина может крыться в том, что в удаленном репозитории появились изменения, которые отсутствуют в локальном. Чтобы выйти из положения, необходимо забрать эти новые изменения в локальный репозиторий командой:

*git pull*

Чтобы избавиться от необходимости каждый раз вводить логин и пароль при отправке измененений в удаленный репозиторий, можно использовать SSH ключи.

Самый удобный визуальный клиент Git под основные платформы – SmartGit:

<http://www.syntevo.com/smartgit>

**Основной список git команд:**

git init - Инициализация репозитория

git status - Просмотр состояния

git clone [ссылка на удаленный git репозиторий] - Скачивание git репозитория

git commit -m 'сообщение' - Фиксация изменений

git add . - Добавить все

git add -A - Добавить все

git add [файл с расширением] - Добавить конкретный файл

git push origin master - Отправка изменений в ветку master

git pull origin master - Принятие изменений из ветки master

git checkout [git ветка] - Перейти на ветку

git branch - Посмотреть все имеющиеся ветки

git branch [git ветка] - Создать ветку

git merge [git ветка] - Слияние веток

git rm [файл] - Удаление файла

git push origin HEAD - Отправить текущую ветку на удаленный git репозиторий, не вводя ее название

git push origin - Отправить все ветки на удаленный git репозиторий

git branch -d [git ветка] - Удалить ветку (после git merge)

git branch -D [git ветка] - Просто удалить ветку (игнорируя git merge)

git push origin :[git ветка] - Удалить ветку в удаленном git репозитории

git reset --hard d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4 - Жесткий откат к конкретному коммиту (хэш смотрим в «git log»)

git reset --soft d8578edf8458ce06fbc5bb76a58c5ca4a58c5ca4 - Мягкий откат к конкретному коммиту (хэш смотрим в «git log»)

git remote add origin [ссылка на удаленный git репозиторий] - Подключится к удаленному git репозиторию

1. **Установка редактора кода**

Atom — это новейший текстовый редактор от [GitHub](https://github.com/" \t "_blank). Он является бесплатным, открытым, легко устанавливается и прост в использовании. Доступен для Windows, OSX и Linux. [Скачать его можно здесь](https://atom.io/)

1. **Запуск локального сервера**

Набрать из каталога проекта:

*python manage.py runserver*

В Windows команда может выдавать ошибку UnicodeDecodeError, тогда вместо неё:

*python manage.py runserver 0:8000*

Теперь нужно проверить работает ли веб-сайт - набери следующий адрес в браузере: http://127.0.0.1:8000/

**Favicon.ico** (сокр. от англ. **FAVorite** **ICON** — «значок для избранного») — значок веб-сайта или веб-страницы. Отображается браузером во вкладке перед названием страницы, а также в качестве картинки рядом с закладкой...

#### Деплой

1. **Установка Apache и mod\_wsgi**

Если вы просто хотите поэкспериментировать с Django, пропустите этот раздел и перейдите к следующему; Django включает в себя легковесный web-сервер, предназначенный для тестирования, поэтому вы вправе не устанавливать Apache до тех пор, пока ваш проект не будет готов для развёртывания на “боевом”-сервере.

Если вам нужно использовать Django на публичном, производственном сайте, используйте [Apache](http://httpd.apache.org/)и [mod\_wsgi](http://modwsgi.readthedocs.org/en/latest/) ; mod\_wsgi может работать в одном из двух режимов: во встроенном режиме и в режиме демона. Во встроенном режиме mod\_wsgi очень похож на mod\_perl – он “встраивает” Python в Apache и загружает код Python в память при запуске сервера. Код остаётся в памяти в течение жизни процесса Apache, что даёт значительный рост производительности по сравнению с другими механизмами. В режиме демона mod\_wsgi порождает независимый процесс-демон, который обрабатывает запросы. Процесс-демон может работать от имени различных пользователей Web-сервера, что может весьма повысить уровень безопасности. Кроме того, процесс-демон может быть перезапущен без останова и перезапуска самого Apache. Обратитесь к [документации mod\_wsgi](http://modwsgi.readthedocs.org/en/latest/), чтобы определить какой режим подходит вам в большей степени. Убедитесь, что Apache установлен и модуль mod\_wsgi активирован. Django будет работать с любой версией Apache, которая имеет поддержку mod\_wsgi.

См. [Использование Django и mod\_wsgi](https://djbook.ru/rel1.8/howto/deployment/wsgi/modwsgi.html) для получения информации о настройке mod\_wsgi сразу после того, как он будет установлен.

Если по какой-либо причине вы не можете использовать mod\_wsgi, не волнуйтесь: Django поддерживает множество других вариантов развёртывания. Один из них – [uWSGI](https://djbook.ru/rel1.8/howto/deployment/wsgi/uwsgi.html); он отлично работает с [nginx](http://nginx.org/ru/) . Дополнительно Django следует WSGI спецификации (PEP 3333), которая позволяет ему работать на различных серверных платформах.

Деплой будет происходить при помощи клонирования git репозитория и настройки всех нужных пакетов и окружения.  
Мне нравится связка **Ubuntu + nginx + uWSGI**.  
Тут уже все намного проще. На эту тему написано много статей, в частности на хабре: [kmike](http://habrahabr.ru/post/113636/), [Nginx + uWSGI](http://habrahabr.ru/post/116894/), [Nginx + uWSGI + virtualenv](http://habrahabr.ru/post/136295/), [nginx + gunicorn](http://habrahabr.ru/post/159575/), [Виртуальный хостинг Django, модель N](http://habrahabr.ru/post/139270/), [DTemplate](http://habrahabr.ru/post/81050/), [Django и Fabric](http://habrahabr.ru/post/76619/), [Fabric](http://habrahabr.ru/post/141271/)…

Можно сказать еще про структуру каталогов с сайтами на deploy сервере, она почти полностью повторяет структуру каталогов для разработки, добавлены лишь каталоги log и conf.  
**log** — каталог для хранения логов, в частности nginx и uWSGI  
**conf** — каталог с настройками для nginx и uWSGI

Команда **cd ..** указывает системе подняться на один уровень выше текущего каталога. Очевидно, команда **cd ../..** перенесёт вас выше на два каталога.