Компиляция CPython

После того как вы загрузили среду разработки и настроили ее, можно скомпилировать исходный код CPython в исполняемый интерпретатор.

В отличие от файлов Python, исходный код C необходимо заново компилировать при каждом изменении. Вероятно, вам стоит положить закладку на этой главе и запомнить некоторые шаги, потому что они будут неоднократно повторяться.

В предыдущей главе вы видели, как настроить среду разработки с возможностью запуска процесса сборки, который перекомпилирует CPython. Но чтобы операции сборки заработали, вам понадобится компилятор C и некоторые инструменты.

Выбор инструментов зависит от операционной системы, поэтому перейдите к разделу, в котором рассматривается ваша ОС.

ПРИМЕЧАНИЕ

Если вас беспокоит, что какие-либо из этих действий отразятся на уже установленных версиях CPython, не тревожьтесь. Каталог исходного кода CPython ведет себя как виртуальная среда.

При компиляции CPython или изменении исходного кода стандартной библиотеки все остается в изолированной среде («песочнице») каталога с исходным кодом.

Если вы захотите установить нестандартную версию, читайте дальше — этот шаг рассматривается в данной главе.

КОМПИЛЯЦИЯ CPYTHON НА MACOS

Компиляция CPython на macOS потребует дополнительных приложений и библиотек. Прежде всего вам понадобится основной тулкит компилятора C. Command Line Tools — приложение, которое можно обновлять в macOS через App Store. Исходная установка должна выполняться в терминале.

ПРИМЕЧАНИЕ

Чтобы открыть терминал в macOS, выберите команду Applications ▸ Other ▸ Terminal. Приложение лучше сохранить в Dock, поэтому зажмите Ctrl, кликните по иконке и выберите команду Keep in Dock.

В терминале установите компилятор C и тулкит следующей командой:

```
$ xcode-select --install
```

После выполнения команды вам будет предложено загрузить и установить набор инструментов, включая Git, Make и компилятор GNU C.

Кроме того, потребуется рабочая копия OpenSSL для загрузки пакетов с вебсайта PyPI. Если вы планируете использовать эту сборку для установки дополнительных библиотек, потребуется проверка SSL-сертификата.

Чтобы установить OpenSSL в macOS, проще всего воспользоваться менеджером пакетов Homebrew.

ПРИМЕЧАНИЕ

Если вы еще не установили программу Homebrew, загрузите и установите ее прямо с GitHub следующей командой:

```
$ /usr/bin/ruby -e "$(curl -fsSL \
  https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

С помощью Homebrew можно установить зависимости CPython командой `brew install`:

```
$ brew install openssl xz zlib gdbm sqlite
```

Все зависимости готовы, теперь можно запустить скрипт `configure`.

Команда Homebrew `brew --prefix` <*пакет*> выдает каталог, в котором установлен *пакет*. Чтобы включить поддержку SSL, используйте путь, сгенерированный Homebrew.

Флаг `--with-pydebug` активизирует отладочные хуки (hooks). Добавьте этот флаг, если вы хотите отладить процесс разработки или тестирования. Отладка CPython подробно рассматривается в главе «Отладка».

Настроить конфигурации достаточно всего один раз, указав путь до пакета zlib:

```
$ CPPFLAGS="-I$(brew --prefix zlib)/include" \
  LDFLAGS="-L$(brew --prefix zlib)/lib" \
  ./configure --with-openssl=$(brew --prefix openssl) \
  --with-pydebug
```

Команда `./configure` генерирует makefile в корне репозитория. Ее можно использовать для автоматизации процесса сборки.

Теперь можно создать двоичный файл CPython следующей командой:

```
$ make -j2 -s
```

СМ. ТАКЖЕ

За дополнительной информацией о make обращайтесь к разделу «Знакомство с Make».

В процессе сборки могут появиться сообщения об ошибках. В сводной информации make оповестит вас о том, что не все пакеты были собраны. Например, с приведенными инструкциями не соберутся пакеты `ossaudiodev`, `spwd` и `_tkinter`. Это нормально, если вы не планируете их использовать. А если планируете — обращайтесь к руководству Python Developer's Guide[1] за дополнительной информацией.

Сборка займет несколько минут, и в результате будет сгенерирован двоичный файл с именем python.exe. Каждый раз, когда вы вносите изменения в исходный код, вам придется перезапустить `make` с теми же флагами.

[1] https://devguide.python.org/.

Двоичный файл python.exe является отладочной двоичной версией CPython. Запустите python.exe, чтобы увидеть рабочий интерпретатор REPL:

```
$ ./python.exe
Python 3.9 (tags/v3.9:9cf67522, Oct 5 2020, 10:00:00)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

ВАЖНО

Да, все верно, сборка macOS использует расширение .exe. Это расширение появилось вовсе не потому, что это двоичный файл Windows!

Так как в файловой системе macOS учитывается регистр символов, разработчики не хотели, чтобы при работе с двоичным файлом пользователи случайно обращались к каталогу Python/, поэтому они присоединили .exe для предотвращения неоднозначности.

Если позднее вы выполните команду make install или make altinstall, перед установкой в систему файл будет переименован в python.

КОМПИЛЯЦИЯ CPYTHON НА LINUX

Чтобы скомпилировать CPython на Linux, сначала загрузите и установите make, gcc, configure и pkgconfig.

Используйте следующую команду для Fedora Core, RHEL, CentOS или других систем на базе YUM:

```
$ sudo yum install yum-utils
```

Для Debian, Ubuntu или других систем на базе APT команда выглядит так:

```
$ sudo apt install build-essential
```

Затем установите дополнительные необходимые пакеты.

Команда для Fedora Core, RHEL, CentOS или других систем на базе YUM:

```
$ sudo yum-builddep python3
```

Команда для Debian, Ubuntu и других систем на базе APT:

```
$ sudo apt install libssl-dev zlib1g-dev libncurses5-dev \
  libncursesw5-dev libreadline-dev libsqlite3-dev libgdbm-dev \
  libdb5.3-dev libbz2-dev libexpat1-dev liblzma-dev libffi-dev
```

После подготовки зависимостей можно запустить скрипт `configure`, при желании включив отладочные хуки с ключом `--with-pydebug`:

```
$ ./configure --with-pydebug
```

Далее можно собрать двоичный файл CPython, запустив сгенерированный makefile:

```
$ make -j2 -s
```

СМ. ТАКЖЕ

За дополнительной информацией о параметрах make обращайтесь к разделу «Знакомство с Make».

Просмотрите вывод и убедитесь в том, что при компиляции модуля `_ssl` не возникло никаких проблем. Если они возникли, поищите в документации дистрибутива инструкции по установке заголовков для OpenSSL.

В процессе сборки могут появиться сообщения об ошибках. В сводной информации make оповестит вас о том, что не все пакеты были собраны. Это нормально, если вы не планируете их использовать. А если планируете — обращайтесь к описанию пакетов за информацией о необходимых библиотеках.

Сборка займет несколько минут, и в результате будет сгенерирован двоичный файл с именем python. Это отладочная двоичная версия CPython. Запустите ./python.exe, чтобы увидеть рабочий интерпретатор REPL:

```
$ ./python
Python 3.9 (tags/v3.9:9cf67522, Oct 5 2020, 10:00:00)
[Clang 10.0.1 (clang-1001.0.46.4)] on Linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

УСТАНОВКА СПЕЦИАЛИЗИРОВАННОЙ ВЕРСИИ

Если ваши изменения вас устраивают и вы хотите использовать их в своей системе, установите двоичный файл Python из вашего репозитория с исходным кодом как специализированную версию.

Для macOS и Linux используйте команду `altinstall`, которая не создает символические ссылки для `python3` и устанавливает автономную версию:

```
$ make altinstall
```

Для Windows необходимо заменить конфигурацию сборки `Debug` на `Release`, а затем скопировать упакованные двоичные файлы в директорию вашего компьютера, входящую в системный путь.

ЗНАКОМСТВО С MAKE

Возможно, вы как Python-разработчик еще не сталкивались с программой `make`. А может, сталкивались, но ваш опыт работы с ней невелик.

Для C, C++ и других компилируемых языков список команд, которые необходимо выполнить для загрузки, компоновки и компиляции вашего кода в правильном порядке, может оказаться очень длинным. При компиляции приложений из исходного кода необходимо подключить все внешние библиотеки в системе.

Нереально ожидать, что разработчик знает местонахождение всех этих библиотек и сам вставляет их в командную строку, поэтому программы `make` и `configure` часто используются в проектах C/C++ для автоматизации создания скрипта сборки.

Когда вы выполняете `./configure`, утилита `autoconf` ищет в вашей системе библиотеки, необходимые для CPython, и копирует их пути в makefile. Сгенерированный makefile напоминает скрипт командной оболочки и делится на части, называемые **целями** (targets).

Для примера рассмотрим цель `docclean`. Она удаляет сгенерированные файлы документации командой `rm`:

```
docclean:
    -rm -rf Doc/build
    -rm -rf Doc/tools/sphinx Doc/tools/pygments Doc/tools/docutils
```

Чтобы реализовать эту цель, выполните команду `make docclean`. `docclean` — очень простая цель, поскольку запускает всего две команды.

При выполнении make-целей используется следующая конструкция:

```
$ make [параметры] [цель]
```

Если вызвать make без указания цели, команда запустит цель по умолчанию (первая цель, указанная в makefile). В CPython это цель all, компилирующая все части CPython.

make поддерживает большое количество параметров. Вот некоторые параметры, которые могут пригодиться вам по ходу чтения книги.

ПАРАМЕТР	ИСПОЛЬЗОВАНИЕ
-d, --debug[=FLAGS]	Вывод разнообразной отладочной информации
-e, --environment-overrides	Переопределение настроек makefile переменными среды
-i, --ignore-errors	Игнорирование ошибок команд
-j [N], --jobs[=N]	Допуск одновременного выполнения N задач (в противном случае количество задач не ограничивается)
-k, --keep-going	Продолжить выполнение, если некоторые цели не удалось выполнить
-l [N], --load-average[=N], --max-load[=N]	Запуск нескольких задач, только если load < N
-n, --dry-run	Вывод команд вместо их запуска
-s, --silent	Отключение команды echo[1]
-S, --stop	Остановка, если некоторые цели не удалось выполнить

В следующем разделе и далее в книге make будет выполняться со следующими параметрами:

```
$ make -j2 -s [target]
```

Флаг -j2 позволяет выполнять две задачи одновременно. Если на вашем компьютере четыре ядра и более, вы можете задать значение 4 и выше; в этом случае компиляция будет быстрее.

[1] Вывод текста в терминал. — *Примеч. ред.*

Флаг `-s` блокирует вывод на консоль результатов каждой команды из `make`-файла. Если вы хотите увидеть, что происходит при сборке, удалите флаг `-s`.

MAKE-ЦЕЛИ CPYTHON

Как в Linux, так и в macOS периодически возникает необходимость удаления файлов, а также построения или обновления конфигурации. В приведенных ниже таблицах описаны полезные `make`-цели, встроенные в `makefile` CPython.

Цели сборки

Следующие цели используются для сборки двоичных файлов CPython.

ЦЕЛЬ	НАЗНАЧЕНИЕ
all (по умолчанию)	Сборка компилятора, библиотек и модулей
clinic	Запуск Argument Clinic во всех исходных файлах
profile-opt	Компиляция двоичного файла Python с профильной оптимизацией
regen-all	Повторное создание всех сгенерированных файлов
sharedmods	Сборка всех общих модулей

Цели тестирования

Следующие цели используются для тестирования скомпилированного двоичного файла.

ЦЕЛЬ	НАЗНАЧЕНИЕ
coverage	Компиляция и запуск тестов с использованием gcov
coverage-lcov	Создание HTML-отчетов о покрытии кода
quicktest	Выполнение набора более быстрых регрессионных тестов с исключением тестов, занимающих много времени
test	Выполнение основного набора регрессионных тестов

ЦЕЛЬ	НАЗНАЧЕНИЕ
testall	Двукратное выполнение полного набора тестов: один раз без файлов .pyc и один – с ними
testuniversal	Выполнение набора тестов для обеих архитектур в универсальной сборке для OS X

Цели очистки

Основные цели очистки — `clean`, `clobber` и `distclean`. Цель `clean` просто удаляет скомпилированные и кэшированные библиотеки и файлы .pyc.

Если вы обнаружили, что `clean` не делает этого, попробуйте использовать `clobber`. Цель `clobber` удаляет `makefile`, так что вам придется снова запустить `./configure`.

Чтобы полностью очистить среду и перевести ее в состояние до установки дистрибутива, выполните цель `distclean`.

В следующей таблице описаны эти три основные цели очистки, а также ряд дополнительных.

ЦЕЛЬ	НАЗНАЧЕНИЕ
check-clean-src	Проверка чистоты исходного кода при сборке
clean	Удаление файлов .pyc, скомпилированных библиотек и профилей
cleantest	Удаление каталогов test_python_* предыдущих неудачных тестовых заданий
clobber	То же, что clean, но с удалением библиотек, тегов, конфигураций и сборок
distclean	То же, что clobber, но с удалением всего сгенерированного из исходных файлов (например, make-файлов)
docclean	Удаление собранной документации в Doc/
profile-removal	Удаление всех профилей оптимизации
pycremoval	Удаление файлов .pyc

Цели установки

Существуют две разновидности целей установки: стандартные (например, `install`) и альтернативные (например, `altinstall`). Если вы хотите установить скомпилированную версию на ваш компьютер, но не хотите делать ее установкой Python 3 по умолчанию, используйте альтернативную версию команды.

ЦЕЛЬ	НАЗНАЧЕНИЕ
altbininstall	Установка интерпретатора python с присоединенной версией – например, python3.9
altinstall	Установка общих библиотек, двоичных файлов и документации с суффиксом версии
altmaninstall	Установка документации с версией
bininstall	Установка всех двоичных файлов, включая python, idle и 2to3
commoninstall	Установка общих библиотек и модулей
install	Установка общих библиотек, двоичных файлов и документации (с запуском commoninstall, bininstall и maninstall)
libinstall	Установка общих библиотек
maninstall	Установка документации
sharedinstall	Динамическая загрузка модулей

После выполнения установки с помощью `make install` команда `python3` свяжется со скомпилированным двоичным файлом. Однако при использовании `altinstall` установится только `python$(`*версия*`)`, а существующая ссылка на `python3` останется неизменной.

Другие цели

Ниже перечислены некоторые дополнительные `make`-цели, которые могут вам пригодиться.

ЦЕЛЬ	НАЗНАЧЕНИЕ
autoconf	Повторное генерирование configure и pyconfig.h.in
python-config	Генерирование скрипта python-config

ЦЕЛЬ	НАЗНАЧЕНИЕ
recheck	Повторное выполнение configure с теми же параметрами, которые применялись в последний раз
smelly	Проверка того, что имена экспортируемых символических имен начинаются с Py или _Py (см. PEP 7)
tags	Создание файла tags для vi
TAGS	Создание файла tags для Emacs

КОМПИЛЯЦИЯ CPYTHON НА WINDOWS

Существуют два способа компиляции двоичных файлов и библиотек CPython на Windows:

1. Компиляция из командной строки. Для этого варианта потребуется компилятор Microsoft Visual C++, входящий в поставку Visual Studio.
2. Прямая сборка через PCbuild ▸ pcbuild.sln в Visual Studio.

В следующих разделах рассматриваются оба варианта.

Установка зависимостей

Как при компиляции из командной строки, так и через Visual Studio необходимо установить ряд дополнительных инструментов, библиотек и заголовков C.

В папке PCbuild находится файл .bat, автоматизирующий этот процесс. Откройте окно командной строки в PCbuild и выполните команду PCbuild ▸ get_externals.bat:

```
> get_externals.bat
Using py -3.7 (found 3.7 with py.exe)
Fetching external libraries...
Fetching bzip2-1.0.6...
Fetching sqlite-3.28.0.0...
Fetching xz-5.2.2...
Fetching zlib-1.2.11...
Fetching external binaries...
Fetching openssl-bin-1.1.1d...
Fetching tcltk-8.6.9.0...
Finished.
```

Теперь можно выполнить компиляцию из командной строки или Visual Studio.