

Непрерывная интеграция

Юрий Литвинов
y.litvinov@spbu.ru

11.07.2024

Системы сборки

- ▶ Среда разработки не всегда доступна
 - ▶ Continuous Integration-сервера автоматически выполняют сборку после каждого коммита, там некому открыть IDE и нажать на кнопку «запустить»
- ▶ Воспроизводимость сборки
 - ▶ Если чтобы собрать программу надо открыть проект, скопировать пару десятков файлов, поправить кое-какие пути и делать это в полнолуние, то возможны ошибки
- ▶ Автоматизация сборки
 - ▶ git clone
 - ▶ одна консольная команда, которая всё делает за нас
 - ▶ ...
 - ▶ готовое к работе приложение

Сборка вручную без IDE

- ▶ `gcc`
`g++ <имя .c-файла>`
или, например,
`g++ -Wall -o helloworld helloworld.c`
- ▶ Если проект большой, это быстро становится грустно
 - ▶ Десятки тысяч файлов — не редкость

make

- ▶ Стандарт де-факто по «низкоуровневым» правилам сборки
- ▶ Сама ничего не знает про языки программирования, компиляторы и прочие подобные штуки
- ▶ Знает про цели, зависимости, временные штампы и правила
 - ▶ Смотрит на зависимости цели, если у хоть одной временной штамп свежее цели, запускается правило для цели
 - ▶ В процессе цель может обновить свой временной штамп, что приведёт к исполнению правил для зависящих от неё целей
 - ▶ Цели и зависимости образуют направленный ациклический граф (DAG)
 - ▶ make выполняет топологическую сортировку графа зависимостей
 - ▶ Правила применяются в порядке от листьев к корню
- ▶ Правила сборки описываются в Makefile

Пример

```
target [target ...]: [component ...]  
  [command 1]  
  .  
  .  
  .  
  [command n]
```

Пример:

```
hello: ; @echo "hello"
```

Высокоуровневые системы сборки

- ▶ C/C++:
 - ▶ CMake — кроссплатформенная система сборки, очень популярна в C++ open source-сообществе
 - ▶ MSBuild — система сборки Visual Studio
 - ▶ qmake, qbs — системы сборки от фреймворка Qt
- ▶ JVM:
 - ▶ Maven — старая, но популярная и «архитектурно правильная»
 - ▶ Gradle — несколько более «императивна», нынче более популярна, поддерживает Kotlin
- ▶ .NET — dotnet CLI, FAKE

Написание скриптов сборки для большого проекта — отдельная и довольно трудоёмкая задача

Continuous Integration

Непрерывная интеграция — практика слияния всех изменений по несколько раз в день, сборки их в известном окружении и запуска модульных тестов.

- ▶ Автоматическая сборка
 - ▶ Всё, что нужно для сборки, есть в репозитории, может быть получено на чистую (ну, практически) машину и собрано одной консольной командой
- ▶ Большое количество юнит-тестов, запускаемых автоматически
- ▶ Выделенная машина, слушающая репозиторий и выполняющая сборку
 - ▶ Чаще всего каждая сборка запускается на заранее настроенной виртуальной машине или в Docker-контейнере


Continuous Integration

- ▶ Извещение всех разработчиков о статусе
 - ▶ Если сборка не прошла, разработка приостанавливается до её починки
- ▶ Автоматическое выкладывание
- ▶ Пока сборка не прошла, задача не считается сделанной
 - ▶ Короткие сборки (<10 мин.)
 - ▶ deployment pipeline
 - ▶ Отдельная машина для сборки, для коротких тестов, для длинных тестов, для выкладывания


GitHub Actions


- ▶ Бесплатная система облачной сборки для проектов на GitHub
- ▶ <https://docs.github.com/en/actions>
- ▶ Как настроить:
 - ▶ В репозитории на GitHub Settings -> Actions -> Allow all actions
 - ▶ Создаём в корне репозитория папку .github/workflows/
 - ▶ В нём создаём файл <имя действия>.yml (например, ci.yml)
 - ▶ Описываем процесс сборки согласно <https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions>
 - ▶ Пример и описание линуксовой сборки: <https://www.incredibuild.com/blog/using-github-actions-with-your-c-project>
 - ▶ Коммитим-пушим
 - ▶ Смотрим статус коммита и пуллреквеста

Что получится


 [yurii-litvinov / DocUtils](#) Public



[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

 - Made ReadTable return rectangular table CI #9

 Summary


Jobs

 build

Triggered via push 8 months ago	Status	Total duration	Artifacts
 yurii-litvinov pushed  453c0cc release	Success	2m 26s	—

ci.yml

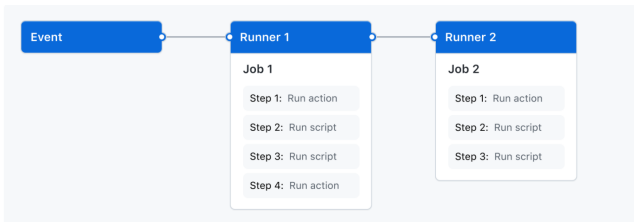
on: push

 build

1m 49s

И появятся иконки статуса рядом с коммитами и пуллреквестами

GitHub Actions, Workflow и Job



- ▶ Step — это либо скрипт, либо Action
- ▶ Action — произвольный код (по сути, отдельное приложение), выполняющийся как шаг Job-а
 - ▶ Переиспользуемый строительный блок
 - ▶ Можно переиспользовать Workflow-ы

Типичный Workflow для сборки

name: Build

on: [push, pull_request]

jobs:

build-Ubuntu:

runs-on: ubuntu-latest

steps:

- **uses:** actions/checkout@v4
- **uses:** actions/setup-dotnet@v4

with:

dotnet-version: '7.x'

- **name:** Build
- run:** for f in \$(find . -name "*.sln"); do dotnet build \$f; done
- **name:** Run tests
- run:** for f in \$(find . -name "*.sln"); do dotnet test \$f; done

build-Windows:

runs-on: windows-latest

steps:

...

- **name:** Build
- run:** For /R %%I in (*.sln) do dotnet build %%I
- **name:** Run tests
- run:** For /R %%I in (*.sln) do dotnet test %%I

Переменные окружения

env:

DAY_OF_WEEK: Monday

jobs:

greeting_job:

runs-on: ubuntu-latest

env:

Greeting: Hello

steps:

- **name:** "Say Hello Mona it's Monday"

if: \${ env.DAY_OF_WEEK == 'Monday' }}

run: echo "\$Greeting \$First_Name. Today is \$DAY_OF_WEEK!"

env:

First_Name: Mona

Матрица сборки

```
runs-on: ${{ matrix.os }}
strategy:
  matrix:
    os: [ubuntu-18.04, ubuntu-20.04]
    node: [10, 12, 14]
steps:
  - uses: actions/setup-node@v2
    with:
      node-version: ${{ matrix.node }}
```

Что ещё?

- ▶ Секреты
 - ▶ **super_secret**: `${{ secrets.SUPERSECRET }}`
- ▶ Кеширование промежуточных результатов
- ▶ Автоматическое развёртывание
 - ▶ В том числе, автодеплой документации на github-pages
- ▶ Проверка стиля кодирования, статический анализ кода и т.п.
 - ▶ Может быть интересно для Python-разработчиков
- ▶ Можно иметь несколько Workflow-ов в одном репозитории

Оформление репозитория

- ▶ README.md — самая важная часть любого репозитория
 - ▶ Плашки CI и анализаторов
 - ▶ Общее описание проекта
 - ▶ Пример использования (с картинками, если уместно)
 - ▶ Как собрать и запустить
 - ▶ Если проект большой, то куда писать баги и как поучаствовать в разработке
- ▶ CI обязательно
- ▶ Внешние анализаторы типа Codacy или CodeCov — опционально, но чем больше — тем лучше
- ▶ .gitignore (и .gitattributes, если используете кириллицу и не хотите эльфийские руны в диффах)
- ▶ Лицензия — обязательно

Лицензия

- ▶ Open source-кодом можно пользоваться, только если автор явно это разрешил, так что просто код на GitHub — не совсем open source
- ▶ Бывают исключительные и личные неимущественные права
 - ▶ Личные неимущественные права неотчуждаемы
 - ▶ Исключительные права можно передать
 - ▶ Права появляются в момент создания произведения и принадлежат автору
 - ▶ Если произведение создано по служебному заданию — работодателю
 - ▶ Знак копирайта служит только для информирования, регистрация прав не требуется
 - ▶ Соавторы владеют произведением в равной степени
- ▶ Идея не охраняется, охраняется её физическое выражение

Open source-лицензии

- ▶ Лицензия — способ передачи части прав на произведение
- ▶ Пример — “Do what the **** you want to public license”
 - ▶ “Want to” может включать в себя патентование произведения и подачу в суд на автора за нарушение патента, поэтому обычно лицензии более длинны и унылы
 - ▶ В России и Европе программы не патентуют, в США — да
- ▶ Каждый нормальный open source-проект должен иметь лицензию

Open source-лицензии

- ▶ Часто используемые open source-лицензии:
 - ▶ GPL, LGPL (GPL вирусная, поэтому использовать её, внезапно, плохая практика)
 - ▶ MIT License
 - ▶ Apache License 2.0 (может применяться пофайлово)
 - ▶ BSD License (в разных вариантах)
 - ▶ The Unlicense — явная передача произведения в Public Domain
 - ▶ Семейство лицензий Creative Commons — не для софта, но хорошо подходит для ресурсов (картинок, текстов и т.д.)