ООП

Лекция 1 | Git и ведение проекта

Лектор: Киселев Станислав Дмитриевич

Общая информация

Посещаемость

- Обязательное посещение лекций
- Очная сдача лабораторных

Экзамен

- 80% сданных отчетов – допуск
- Билеты будут по темам лекций

Связь, вопросы и объявления – через ТГ
(группа и личка)

Запись этой и последующих лекций постараюсь выкладывать в Teams

QR на Teams:

QR на группу для связи:

Порядок работ

Лабораторные

Количество в размере 7

Задачи ориентированны на закрепление знаний по C++

Сдача каждые две недели

Курсовая

Индивидуальная работа по реализации любого рода задачи

За основу курсовой работы разрешается любая тема

Язык программирования может быть выбран из списка:
Go, Rust, C++, Python - остальные по личному согласованию

Состав проекта:

1. Документ, описывающий цель и сам проект
2. Базовая документация кода
   Doxygen достаточно
3. Код на Gitlab / Github

В отдельном файле будут примеры тем – можно выбрать одну из них, или придумать свою любую (по согласованию)

Автоматы

Есть проект:

Оценка 4, 5

Выполнение всех лабораторных в срок на C++

Оценивается проделанная работа за семестр в целом, какая **сложность и объем курсовой работы**

В любом случае можно прийти на экзамен и пересдать на оценку получше (допуск – посещаемость и 80% отчетов)

Ссылка на курсы и таблицу

Будет 2 таблицы со списками

- Для оценок и отметок о сдачи работ
- Для тем курсовых работ

QR на гугл-таблицу с оценками:

QR на гугл-таблицу с темами:

Что такое проект

Виды ПО, проектов и целеполагание

Мы здесь

Релизы и версионирование

Базовые определения и команды Git

Наполнение репозитория и сборка

Файлы проекта, CMake, CPack, Bazel, Build2, Poetry

Заключение

Ваши вопросы

Программное обеспечение (или ПО) –

это совокупность программ на компьютере или другом устройстве. Еще так называют сами программы.
По-английски программное обеспечение – software, поэтому используется еще и термин «софт».

Цифровая система состоит из трех компонентов

Software

Программная «начинка» устройства

Middleware

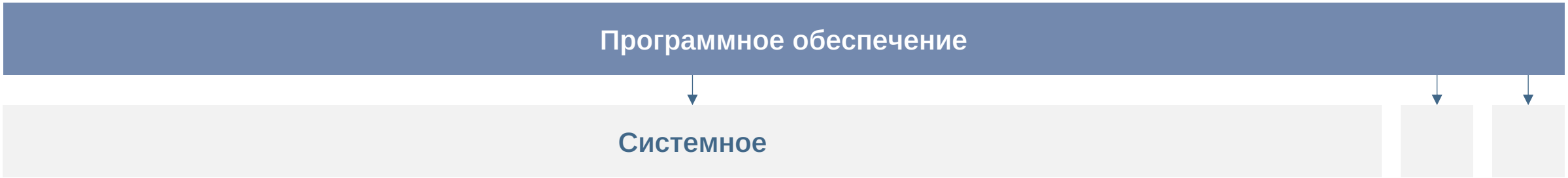
«Прослойка» между железом и программами, инфраструктура для связи компонентов друг с другом

Hardware

Или «железо» – аппаратные составляющие

Виды программного обеспечения

Виды программного обеспечения



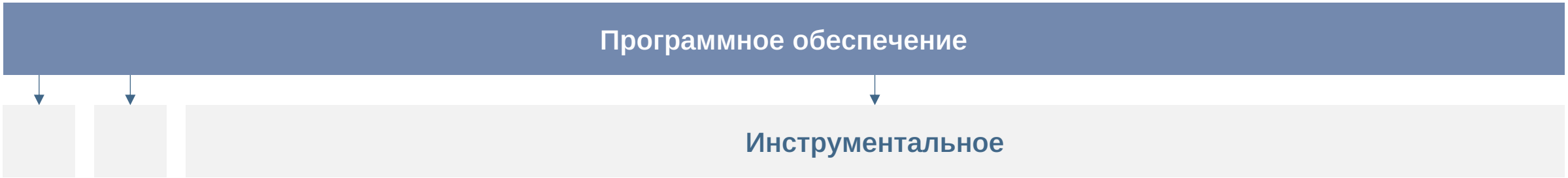
Это программное обеспечение, которое нужно для работы компьютерной системы

Как пример такого софта – операционная система Windows или macOS, ее службы и процессы

Виды программного обеспечения

Это самая знакомая обычному пользователю группа – программы, которыми мы пользуемся в повседневной жизни, от «Блокнота» до 1С. Сюда же относятся приложения на телефон, разные плагины и надстройки, браузеры и многое другое

Виды программного обеспечения



Так называют софт, который нужен для создания других программ

Это профессиональные инструменты айтишников

- Компиляторы и интерпретаторы языков программирования
- Назные библиотеки и фреймворки
- Среды программирования и редакторы кода

Цель проекта

Проект может быть ориентирован на:

- Изучение:
  - Тем
  - Языков программирования
  - Архитектурных стилей
- Понимание правильного проектирования
- Решение личных проблем или финансовая выгода
- Проект ради веселья или интереса

У каждого проекта должна быть четкая цель

SMART целеполагание

S	Specific (конкретная)
M	Measurable (измеримая)
A	Achievable (достижимая)
R	Relevant (релевантная)
T	Time-bound (ограниченная по времени)

Ограниченная по времени

Установленный срок для завершения цели

Релевантная

Соответствующая более широким целям или задачам

Достижимая

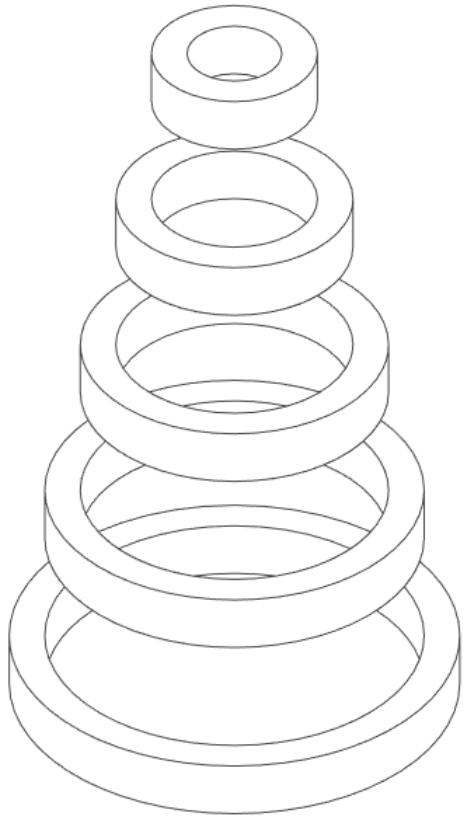
Реалистичная и выполнимая цель

Измеримая

Количественные показатели для отслеживания прогресса

Конкретная

Четко определенная и детализированная цель



Пример качественной цели – проект по созданию дрона

- ✗ Создать прототип дрона до конца года
- ✓ Настроить модуль коммуникации на плате модема к концу недели
- ✗ Продать 500 экземпляров дрона собственной разработки
- ✓ Выступить с питчингом проекта перед инвесторами на конференции «Х» 16 мая

Хорошие цели лучше ставить в позитивном ключе (без частички «НЕ»)

Вместо «Я не буду опаздывать» ставить «Я буду приходить вовремя»

Виды проектов

Pet-проект

Учебный, без строгих сроков

Самоличная инициатива по решению какой-то задачи

Работа может быть направлена на:

- Изучение новых тем, языков, стилей
- Улучшение портфолио и резюме
- Решение личных задач для интереса

Могут быть опущены некоторые стадии разработки (например, отсутствие точного плана, документации)

«Рабочий» проект

Структурированная работа с дедлайнами

Решение конкретной проблемы ради получения выгоды

Подобная работа, как правило, проходит через все циклы разработки:

- Планирование
- Анализ
- Проектирование
- Разработка
- Тестирование
- Интегрирование
- Обслуживание

Личные-проекты: свобода и самодисциплина

Pet-проекты – это личные инициативы без жестких сроков и внешнего давления

Здесь цель – не только конечный результат, но и сам **процесс** обучения, экспериментов и творчества

Как ставить цели:

- **Фокус на обучение**:
  определить, какие технологии или подходы хочется изучить
- **Гибкость**:
  оставить место для изменений в процессе работы
- **Разбивка на этапы**:
  чтобы не забросить проект, полезно ставить короткие и измеримые цели
  (например, «написать MVP за 2 недели»)
- **Баланс между «интересно» и «полезно»**:
  если проект теряет актуальность, можно пересмотреть цели,
  но лучше довести до логического завершения

Коммерческие проекты: структура и результат

Цели жестко привязаны к срокам, клиентским требованиям и работе команды

Здесь важны четкое планирование и контроль сроков

Как ставить цели:

- **SMART-цели**:
  каждая задача должна быть конкретной, измеримой, достижимой, релевантной и ограниченной во времени
- **Определение приоритетов**:
  важные фичи в первую очередь, второстепенные – позже
- **Декомпозиция**:
  разбить задачи на небольшие и четкие этапы, распределить по команде
- **Управление ресурсами**:
  учитывать доступные ресурсы (человеческие, временные, финансовые)
- **Мониторинг и контроль**:
  регулярные митинги и отчеты помогают держать проект в рамках дедлайнов

Отличия

Фактор	Pet-проект	Коммерческий проект
Цель	Обучение, эксперименты, интерес	Выпуск продукта, выполнение требований
Сроки	Гибкие, зависят от мотивации	Жесткие дедлайны, SLA
Команда	Обычно один человек	Команда, разделение ролей
Гибкость	Высокая, можно менять направления	Ограниченная, изменения требуют согласований
Финансы	Без вложений или на личные деньги	Бюджет клиента, инвестиции

Как выбрать язык?

1. Задача проекта
2. Ваш опыт
3. Сообщество
4. Доступность ресурсов
5. Производительность

Исходя из задачи проекта, при выборе языка стоит принимать во внимание:

- Целевую платформу системы –
  Linux, Windows, MacOS, WebApp, Android, IOS
- Интерфейс взаимодействия с клиентом –
  терминал, GUI, Web приложение, API
- Сроки выполнения

Как выбрать язык?

1. Задача проекта
2. Ваш опыт
3. Сообщество
4. Доступность ресурсов
5. Производительность

Имея определенный уровень знаний в основах Computer Science можно выучить основы нового языка в сжатые сроки и тогда нет ограничений при выборе между языка, ориентированного на решение схожих задач или языком общего назначения

Примером может послужить Python для ML

На C++ это тоже реализуемо, но в гораздо большие сроки и с большим количеством проблем

Как выбрать язык?

1. Задача проекта
2. Ваш опыт
3. Сообщество
4. Доступность ресурсов
5. Производительность

Чем больше сообщество языка и выбранных библиотек, тем проще найти решение на Вашу или схожую проблему

Это сильно сокращает время разработки

Как выбрать язык?

1. Задача проекта
2. Ваш опыт
3. Сообщество
4. Доступность ресурсов
5. Производительность

При необходимости в разработки при ограниченных ресурсах важно понимать потребление каждого из компонент и брать из железа всё доступное

Для этого подходят языки низкого уровня, по типу C++, Rust, Zig, Assembly и т.д.

Некоторые библиотеки для языков высокого порядка написаны с большим количеством оптимизаций, упрощая написание кода и убирая необходимость в использовании сложных языков

Как выбрать тему? | Простые проекты

CLI-менеджер задач (C++, Python, Rust, Go)

Консольное приложение для ведения списка задач с возможностью добавления, удаления и отметки выполненных

Парсер веб-страниц (Python, Rust, Go)

Программа, которая загружает веб-страницу и извлекает нужную информацию (например, заголовки новостей)

Генератор паролей (C++, Python)

Простой инструмент для генерации случайных паролей разной сложности

Как выбрать тему? | Средний уровень

Телеграм-бот (Python, Go, Rust)

Бот, который выполняет полезную функцию: показывает курс валют, погоду или подсказывает случайную цитату

Калькулятор расходов (C++, Python, Rust)

Приложение для учета личных финансов с базой данных SQLite

HTTP-сервер для заметок (Go, Rust, Python)

Мини-сервер с REST API, где можно сохранять и получать заметки

Как выбрать тему? | Продвинутые задачи

P2P-чат (C++, Rust, Go)

Программа для обмена сообщениями между пользователями без центрального сервера
+ Добавить шифрование сообщений

Система рекомендаций фильмов (Python, Rust)

Приложение, которое анализирует предпочтения пользователей и предлагает фильмы
+ Можно использовать алгоритмы машинного обучения

Игра с многопоточной логикой (C++, Rust, Go)

Простая 2D-игра с использованием многопоточности (например, симуляция движения объектов)
+ Можно добавить сетевую игру

Мини-брокер для биржевых данных (Go, Rust, Python)

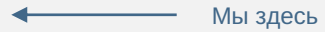
Приложение, которое получает биржевые данные через API и позволяет тестировать простые стратегии
+ Дополнение: визуализация графиков

Что такое проект

Виды ПО, проектов и целеполагание

Релизы и версионирование

Базовые определения и команды Git



Наполнение репозитория и сборка

Файлы проекта, CMake, CPack, Bazel, Build2, Poetry

Заключение

Ваши вопросы

Базовые определения в Git

Git – консольная утилита, для отслеживания и ведения истории изменения файлов, в вашем проекте
Чаще всего его используют для кода, но можно и для других файлов (например, для картинок – полезно для дизайнеров)

С помощью Git-а вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий

Репозиторием называют хранилище вашего кода и историю его изменений
Git работает локально и все ваши репозитории хранятся в определенных папках на жестком диске

Каждая точка сохранения вашего проекта носит название **коммит (commit)**
У каждого commit-а есть hash (уникальный id) и комментарий – Из таких commit-ов собирается ветка

Ветка – история изменений, у каждой ветки есть свое название
Репозиторий может содержать в себе несколько веток, которые создаются из других веток или вливаются в них

Релиз – официальная версия продукта или программы, которая проходит финальную проверку и становится доступной для пользователей

Версионирование

Управление версиями – важная часть разработки программного обеспечения, позволяющая отслеживать изменения в проекте, упрощать релизы и обеспечивать совместимость

Одним из популярных инструментов для автоматизации управления версиями является **bump2version**

Семантическое версионирование предполагает использование формата версий в виде MAJOR.MINOR.PATCH, где

MAJOR – изменение, которое не совместимо с предыдущими версиями

MINOR – добавление нового функционала, совместимого с предыдущими версиями

PATCH – исправления ошибок, не влияющие на функциональность

Bump2version

Создаем файл конфигурации .bumpversion.cfg в корне вашего проекта:

```
[bumpversion]
current_version = 0.1.0   – начальная версия вашего проекта
commit = True             – если True, bump2version будет автоматически коммитить изменения
tag = True                – если True, будет создана метка (tag) в Git

[metadata]
email = your_email@example.com

Major - bump2version major
Minor - bump2version minor
Patch - bump2version patch
```

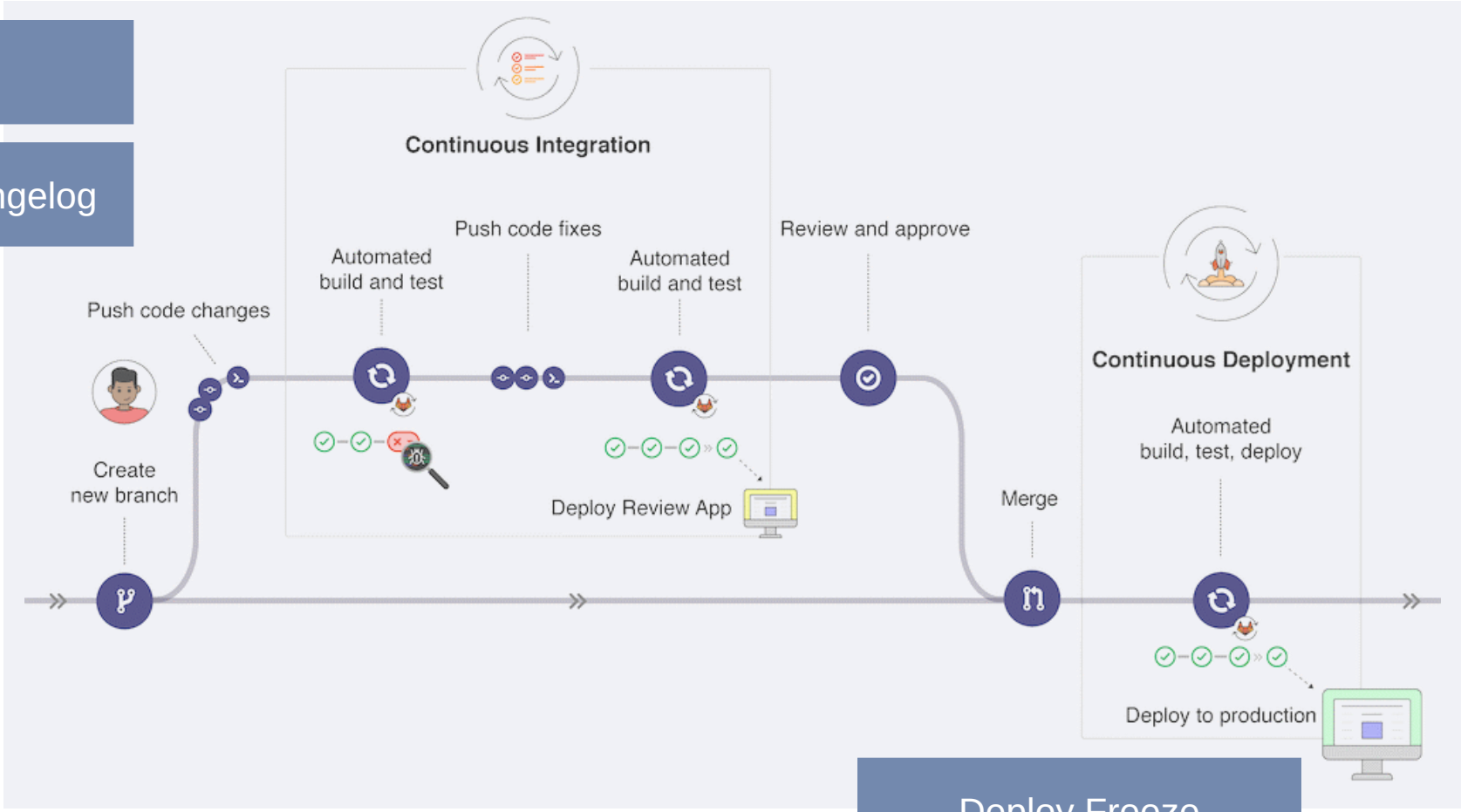
Стандартная структура коммерческого проекта

1. **.gitignore:** Указывает Git, какие файлы и директории игнорировать при коммите
2. **Dockerfile:** Скрипт для сборки Docker-образа с инструкциями по созданию контейнера
3. **.gitlab-ci.yaml:** Конфигурация для автоматизации процессов сборки и развертывания в GitLab CI/CD
4. **.bump2version:** Настройки для автоматического увеличения версии проекта с помощью bump2version
5. **VERSION:** Файл, содержащий текущую версию проекта
6. **LICENSE:** Лицензионное соглашение, определяющее условия использования и распространения кода
7. **README.md:** Документация проекта с описанием, инструкциями и примерами использования

Git Flow | Управление разработкой в Git

fix/Bug

feature/AddChangelog



Deploy Freeze

Создание веток от Develop

Фича

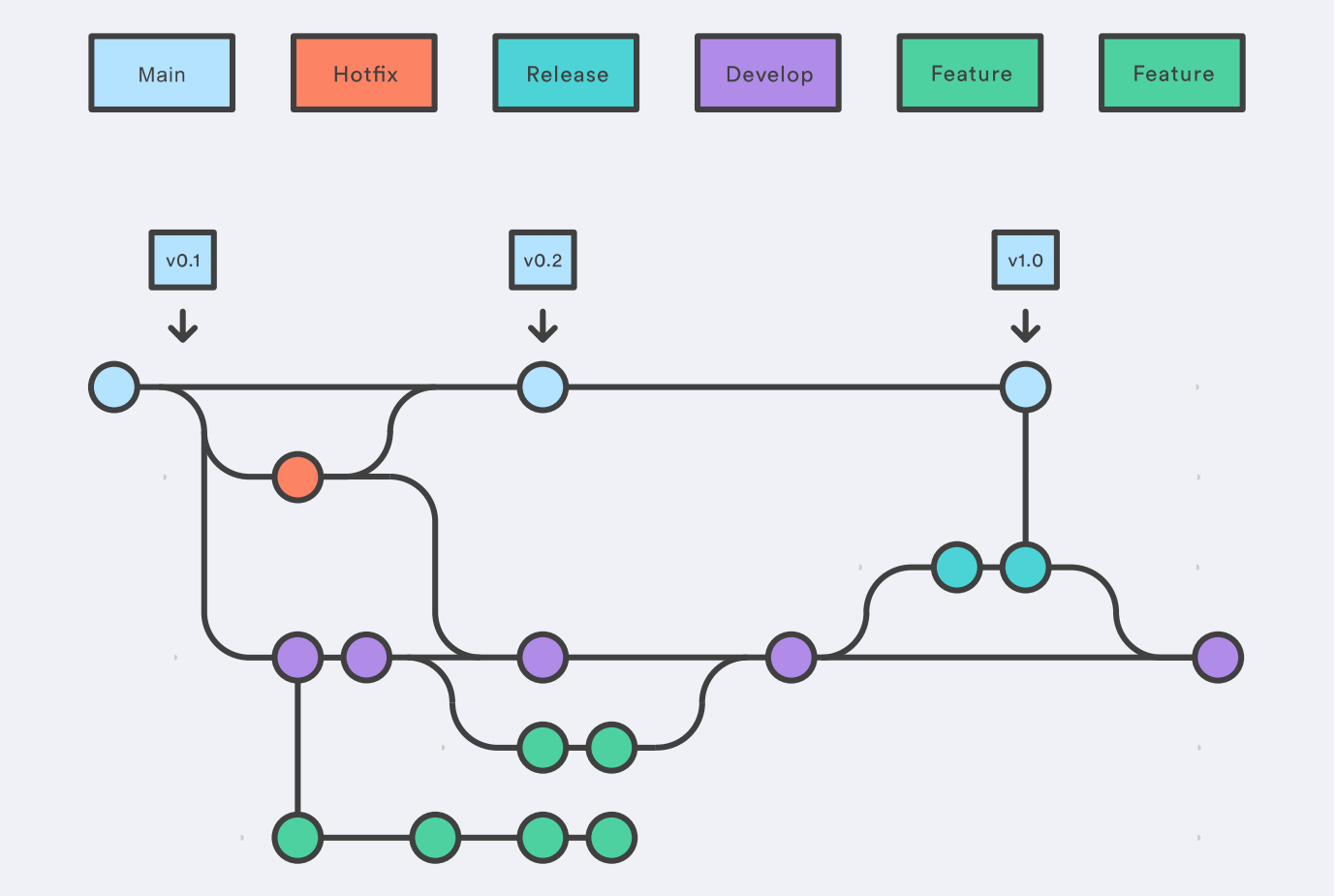
feature/NAME_OF_THE_FEATURE

Исправление бага

fix/NAME_OF_THE_BAG

Баг на продакшене

hotfix/NAME_OF_THE_BAG



Создание веток

Создание новой ветки

git checkout develop
git checkout -b f/feature_branch

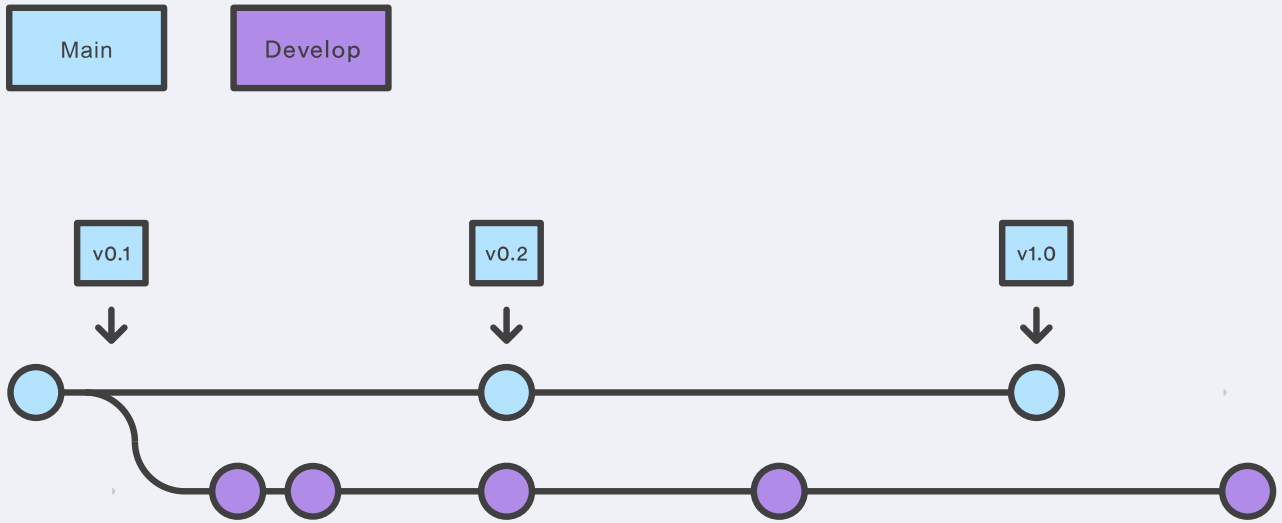
Слияние веток

git checkout develop
git merge feature_branch

Релизы|Теги

git tag TAG_NAME

git push —tags



Создание веток

Скачать ветки с репозитория и удалить устаревшие

git fetch -p

Объединение нужной версии ветки с текущей

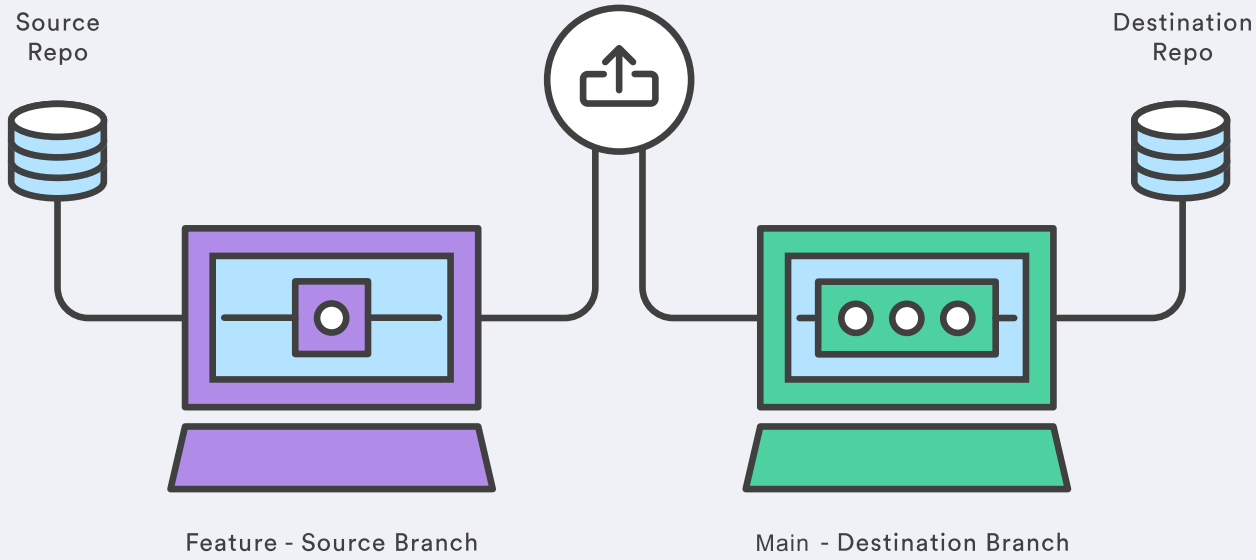
git pull origin BRANCH_NAME

- Например, текущая ветка develop – мы хотим её обновить – достаточно просто git pull origin develop
- Если есть локальные изменения, которых нет в репозитории – скорее всего, придется решать конфликты

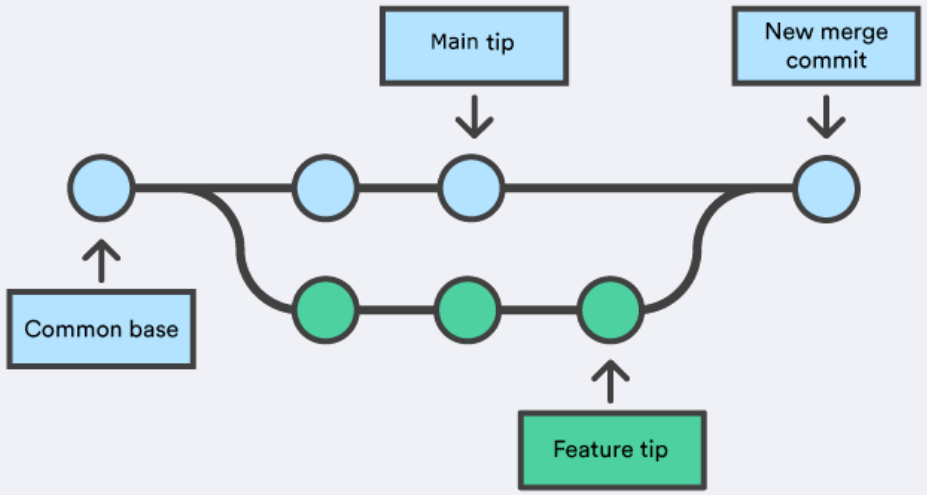
Pull Request – запрос на объединение веток

Merge Request (Pull Request) –
это запрос на слияние изменений, внесенных в ветку разработки, с основной веткой (например, main или develop)

Используется в системах контроля версий, таких как Git, для организации процесса внесения изменений в проект



Pull Request – запрос на объединение веток



1. Разработчик создает функцию в специальной ветке своего локального репозитория
2. Разработчик отправляет ветку в публичный репозиторий
3. Разработчик отправляет запрос на извлечение через Git платформу
4. Остальная часть команды просматривает код, обсуждает его и вносит в него изменения
5. Ответственный за проект объединяет функцию с официальным репозиторием и закрывает запрос на извлечение

Что такое проект

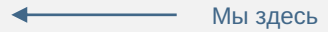
Виды ПО, проектов и целеполагание

Релизы и версионирование

Базовые определения и команды Git

Наполнение репозитория и сборка

Файлы проекта, CMake, CPack, Bazel, Build2, Poetry



Заключение

Ваши вопросы

Обязательные файлы проекта

1

README.md: главный файл документации проекта

Содержит описание проекта, инструкции по установке и использованию

2

LICENSE.md: файл лицензии

Определяет условия использования кода (MIT, Apache, GPL и т. д.)

3

.gitignore: список файлов и папок, которые Git должен игнорировать

Часто содержит временные файлы, логи, сборочные артефакты

4

.pre-commit-config.yaml: конфигурация хуков pre-commit

Запускает линтеры и форматирование перед коммитом

Дополнительные файлы | Настройка автоматизации

- **scripts:** подготовленные файлы автоматизации

  Вспомогательные файлы при работе с проектом

- **.gitlab-ci.yml или .github/workflows/:** конфигурация развертки кода

  Определяет, как и в каких условиях запускать сборку, тесты и отправку кода клиенту

Дополнительные файлы | Для разных языков

- **CMakeLists.txt:** файл сборки для C++ проектов

  Определяет зависимости, настройки компиляции и таргеты

- **requirements.txt или pyproject.toml:** зависимости Python-проекта

  Список необходимых библиотек

- **Cargo.toml:** файл конфигурации Rust-проекта

  Содержит метаинформацию, зависимости и настройки компиляции

- **go.mod:** файл управления зависимостями в Go

  Определяет версии используемых модулей

Конечный вид проекта зависит от инструмента сборки

В первую очередь инструменты сборки – это некоторое программное обеспечение, которое автоматизирует и структурирует процесс подготовки кода к продакшену

А также они автоматизируют и упрощают множество задач, связанных с компиляцией, тестированием, упаковкой и развертыванием приложений:

Автоматизация процессов
избавление от рутинных задач компиляции кода, запуска тестов и создания пакетов

Управление зависимостями
Обеспечение эффективного управления зависимостями, библиотеками и модулями

Кросс-платформенная разработка
Поддержка под разные архитектуры и процессоры
Компиляции кода на более сильном железе

Упрощение развертывания
Доставка программного обеспечения конечным пользователям или в производственные среды

Зачем выбирать?

Существует сотни вариантов инструментов – и у каждого есть свои, отличные, базовые характеристики:

1. **Производительность**
   Имеют разные уровни производительности – некоторые обеспечивают более быструю сборку и тестирование
2. **Сложность настройки**
   Инструмент может требовать более сложной конфигурации, что увеличит время на начальную настройку
3. **Совместимость**
   Разные инструменты могут лучше или хуже интегрироваться с определенными ЯП или фреймворками
4. **Поддержка сообщества**
   Активное сообщество и хорошая документация может облегчить процесс обучения и решения проблем
5. **Гибкость и расширяемость**
   Для сложных проектов с уникальными требованиями полезно наличие функционала настройки и расширения

Инструменты, которые мы рассмотрим в данной лекции

CMake

Bazel

Build2

Poetry

CMake

Кросс-платформенный инструмент для автоматизации сборки проектов

Используется для генерации файлов сборки на различных языках программирования

Преимущества

Кросс-платформенность
Поддерживает все операционные системы и IDE

Гибкость
Работает с различными компиляторами, можно настроить скрипты и функции для работы с нужным языком

Большая экосистема
Обладает широкой документацией, множеством примеров и комьюнити

Совместимость с другими инструментами
Интегрируется с другими инструментами сборки и поставки кода – такими как Docker, GitLab CI, Jenkins

CMake

Кросс-платформенный инструмент для автоматизации сборки проектов

Используется для генерации файлов сборки на различных языках программирования

Недостатки

Сложность конфигурации
Долгая настройка под нестандартные проекты

Производительность
Процесс генерации файлов сборки увеличивается непропорционально росту проекта

Совместимость
Отсутствие полной обратной совместимости версий CMake

CMake

```
cmake_minimum_required(VERSION 3.10)

# Set the project name and version
project(MyApp VERSION 1.0)

# Specify the C++ standard
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# Include directories
include_directories(include)

# Add the executable
add_executable(my_app src/main.cpp src/hello.cpp)

# Install the executable
install(TARGETS my_app DESTINATION bin)

# Install the header files
install(DIRECTORY include/ DESTINATION include)
```

```
cmake -B build -DDEBUB=ON
cmake --build build
cmake --install build
```

CPack (используется только с CMake) –

инструмент упаковки скомпилированного проекта в платформозависимые пакеты и установщики

```
cmake_minimum_required(VERSION 3.10)
project(MyProject)
...
# Настройки для CPack (DEB)
set(CPACK_GENERATOR "DEB")
set(CPACK_PACKAGE_NAME "my_project")
set(CPACK_PACKAGE_VERSION "1.0.0")
set(CPACK_PACKAGE_DESCRIPTION "My simple C++ project")
set(CPACK_PACKAGE_MAINTAINER "Your Name <youremail@example.com>")
set(CPACK_DEB_PACKAGE_DEPENDS "libc6 (>= 2.27)")
set(CPACK_DEB_PACKAGE_ARCHITECTURE "amd64")

# Установка исполняемого файла
install(TARGETS my_project DESTINATION bin)

include(CPack)
```

```
mkdir build && cd build
cmake ..
make
cpack -G DEB
```

CMakeLists

Файл конфигуратор CMake:

```
add_subdirectory(src)
add_subdirectory(apps)
add_subdirectory(docs)

include(cmake/cpack.cmake)

if (BUILD_TESTING)
   add_subdirectory(tests)
endif()
```

- **project**
  - .gitignore
  - README.md
  - LICENSE.md
  - CMakeLists.txt
  - **cmake**
    - FindSomeLib.cmake
    - something_else.cmake
    - cpack.cmake
  - **include**
    - project
      - lib.hpp
  - **src**
    - CMakeLists.txt
  - **apps**
    - CMakeLists.txt
  - **tests**
    - CMakeLists.txt
  - **docs**
    - CMakeLists.txt

Bazel

Разработано Google

Инструмент сборки проектов, ориентирован на масштабируемость и высокую производительность

Подходит **для крупных и сложных проектов** с многими зависимостями

Преимущества

Производительность
Используется инкрементальная сборка, применяя уже собранные компоненты

Масштабируемость и простая конфигурация
Простой синтаксис Build файлов, но при этом хорошая работа как с малыми, так и крупными проектами

Кросс-платформенность
Поддерживает множество языков и компиляторов для сборки

Удобная поддержка тестирования
Встроенные механизмы для тестирования на разных уровнях

Bazel

Разработано Google

Инструмент сборки проектов, ориентирован на масштабируемость и высокую производительность

Подходит для крупных и сложных проектов с многими зависимостями

Недостатки

Сложность освоения
Новичкам сложнее освоить Bazel по сравнению с более традиционными инструментами из-за неочевидных настроек

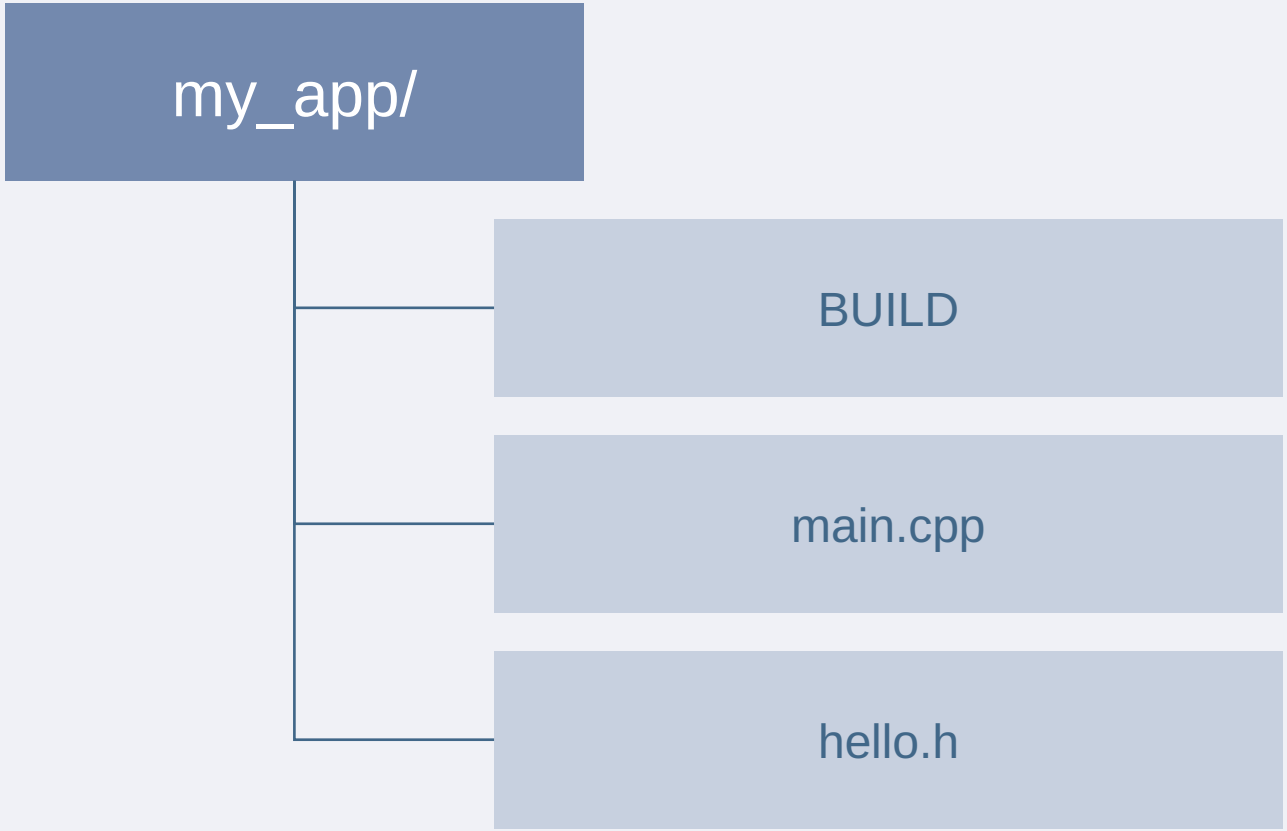
Меньшая гибкость
Жестко привязывается к файлам, делая сложнее использование одних и тех же Build файлов для 2-ух проектов

Зависимости от версий
Отсутствие полной обратной совместимости версий Bazel

Меньшая экосистема
Не такая широкая распространенность среди коммьюнити

Bazel

Разработано Google




```
cc_binary(
    name = "my_app",
    srcs = ["main.cpp"],
    hdrs = ["hello.h"],
)
```

```
bazel build //:my_app
./bazel-bin/my_app
```

Build2

Современный инструмент сборки **для C и C++ проектов**

Предлагает простую и эффективную систему управления зависимостями, сборкой и тестированием

Преимущества

Управление зависимостями (как локальными, так и удаленными)
Есть встроенная систему, которая позволяет легко добавлять и обновлять библиотеки и модули

Простота использования
Простой синтаксис и ясная структура, позволяет быстро начать работу и изучение

Эффективность
Инструмент оптимизирован для быстрого выполнения сборок и управления зависимостями *(быстрее всех)*

Современные возможности
Поддержка последних стандартов C++ и современных практик разработки

Build2

Современный инструмент сборки **для C и C++ проектов**

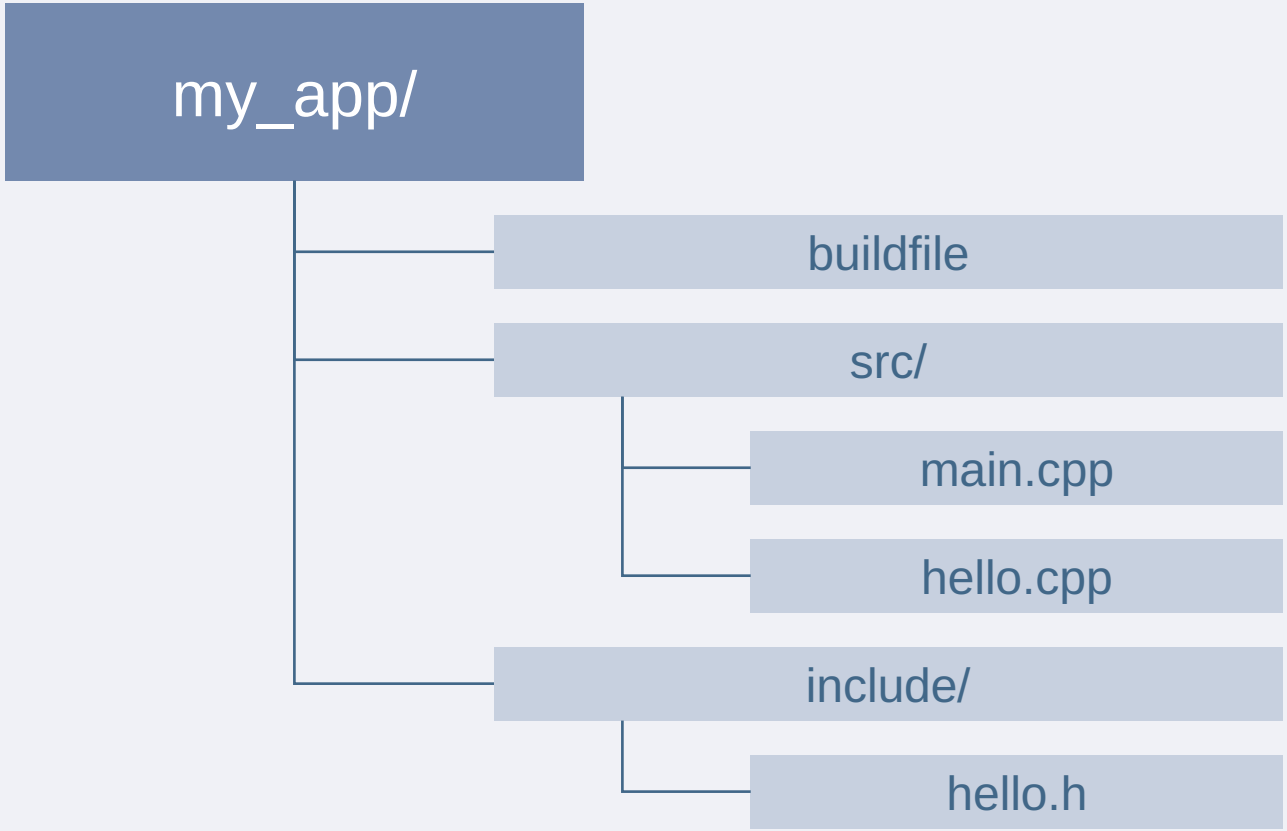
Предлагает простую и эффективную систему управления зависимостями, сборкой и тестированием

Недостатки

Меньшая популярность
Что может ограничивать доступность ресурсов и примеров

Ограниченная поддержка языков
В основном ориентирован на C и C++

Build2




```
# Build2 buildfile

# Define the project
project my_app

# Define the executable
exe my_app : src/main.cpp
src/hello.cpp

# Specify include directories
include include
```

```
b2
./bin/my_app
```

Дальнейшее изучение – список инструментов

Python

1. Poetry
управление зависимостями и инструмент сборки

2. Setuptools
стандартный инструмент для упаковки Python-приложений

3. PyBuilder
интеграция сборки, тестирования и проверки кода

Java

1. Maven
инструмент для управления проектами и зависимостями

2. Gradle
гибкий инструмент с поддержкой Groovy и Kotlin DSL

3. Ant
более старый инструмент, который всё ещё популярен

JavaScript / TypeScript

1. Webpack
сборщик модулей и инструмент для управления зависимостями

2. Parcel
простой и быстрый бандлер для JavaScript

3. Vite
современный инструмент для разработки и сборки

Go

1. Go Modules
встроенное управление зависимостями и сборка

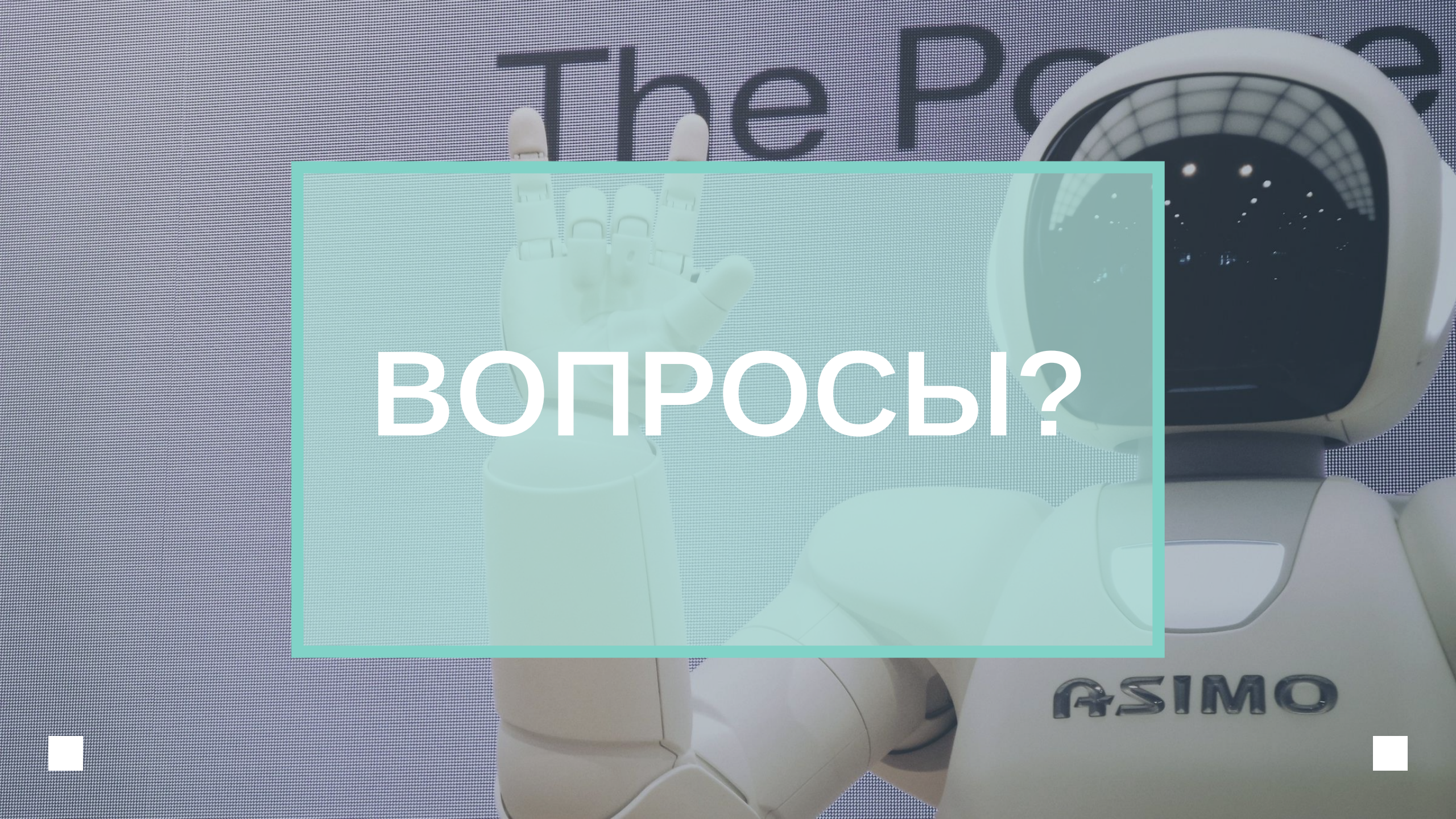
2. Mage
Make для Go, использующий сам язык Go

Rust

1. Fleet
ускоренный Cargo

Haskell

1. Stack
инструмент управления проектами и зависимостями



ВОПРОСЫ?

Спасибо за внимание!
Ваши вопросы?

Оставьте, пожалуйста, обратную связь