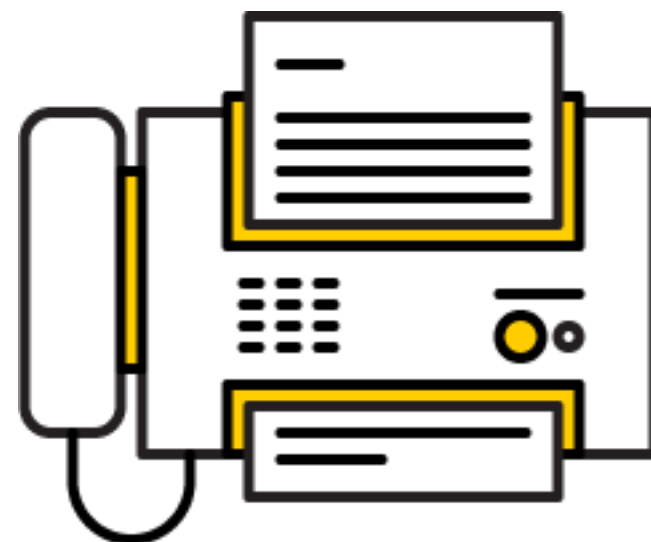


Яндекс



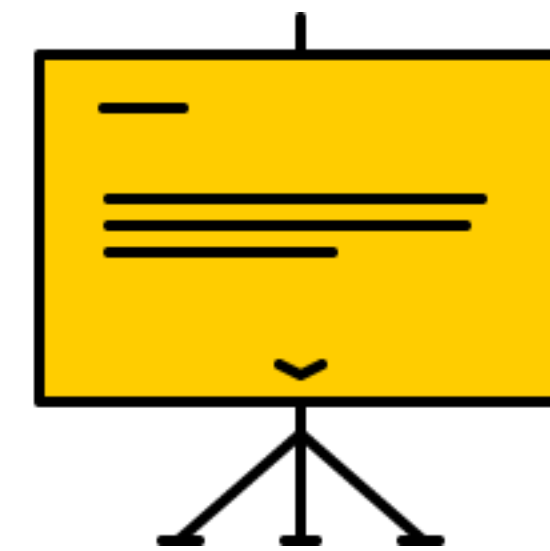
# Gradle для кроссплатформенной разработки на C++



Удалов Илья, C++ разработчик  
[IUdalov@yandex-team.ru](mailto:IUdalov@yandex-team.ru)

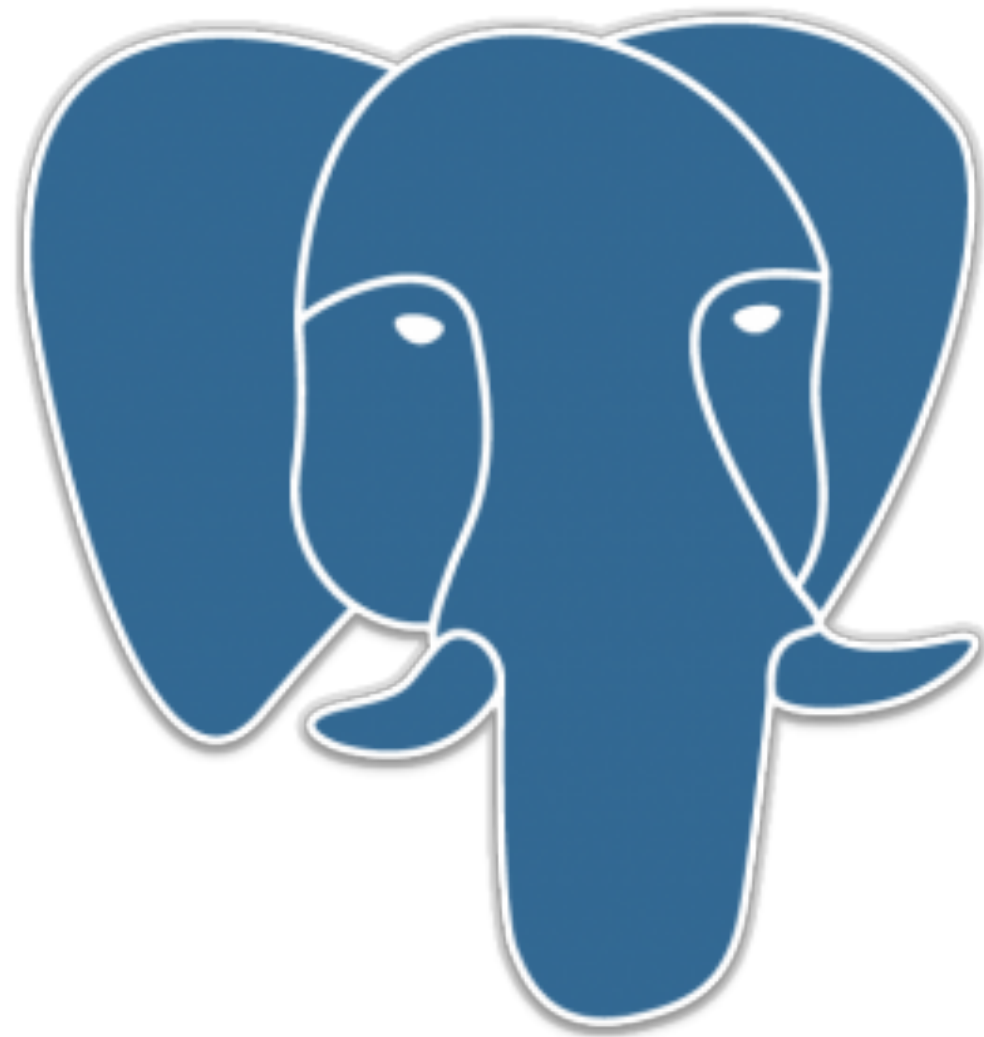
# План выступления

- › Немного о gradle.
- › Декларативное и императивное описание сборки.
- › Организация кода, плагины.
- › Tips & Tricks



[git.io/v1CEI](https://git.io/v1CEI)

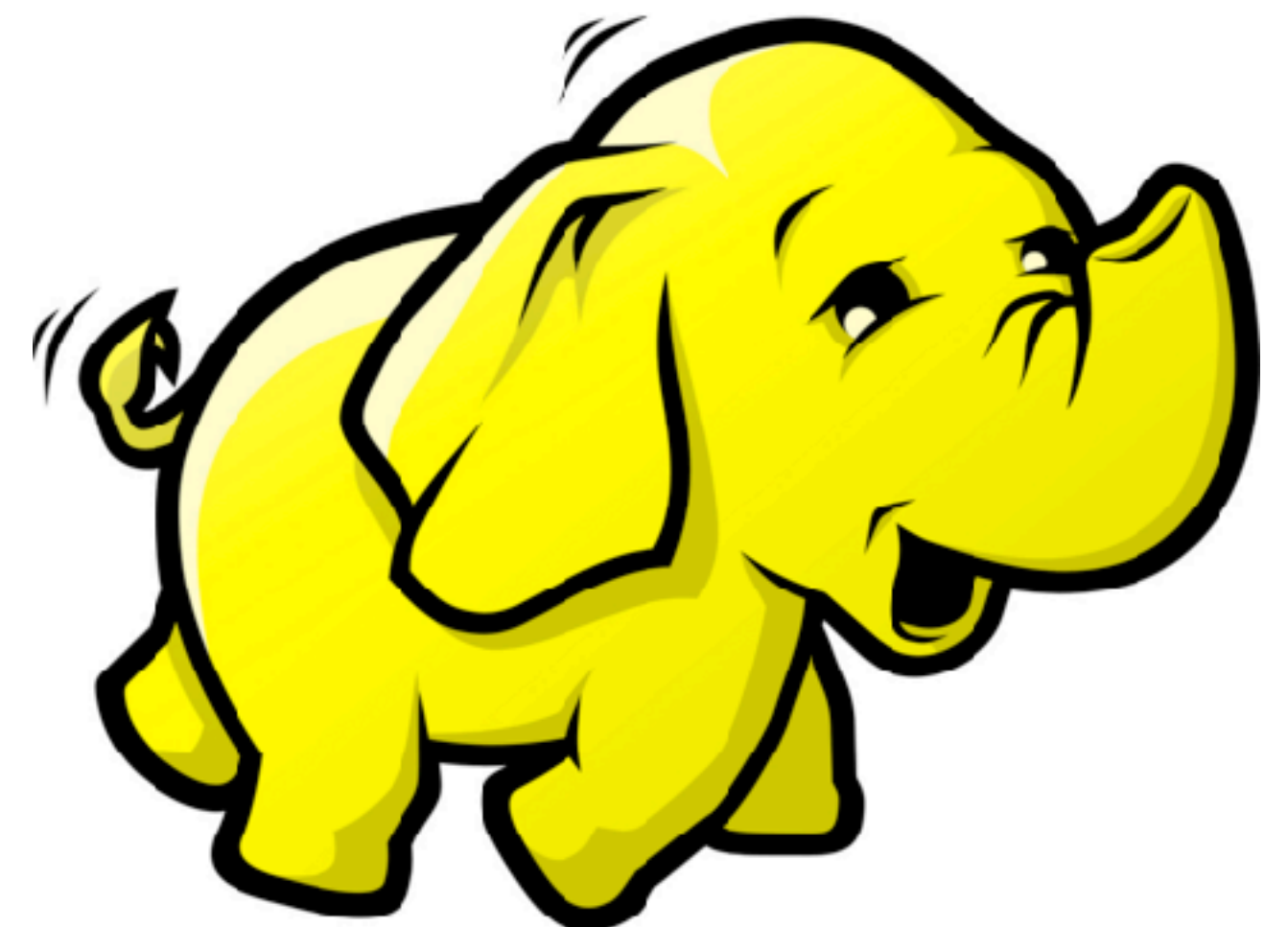
# Угадай слона



???

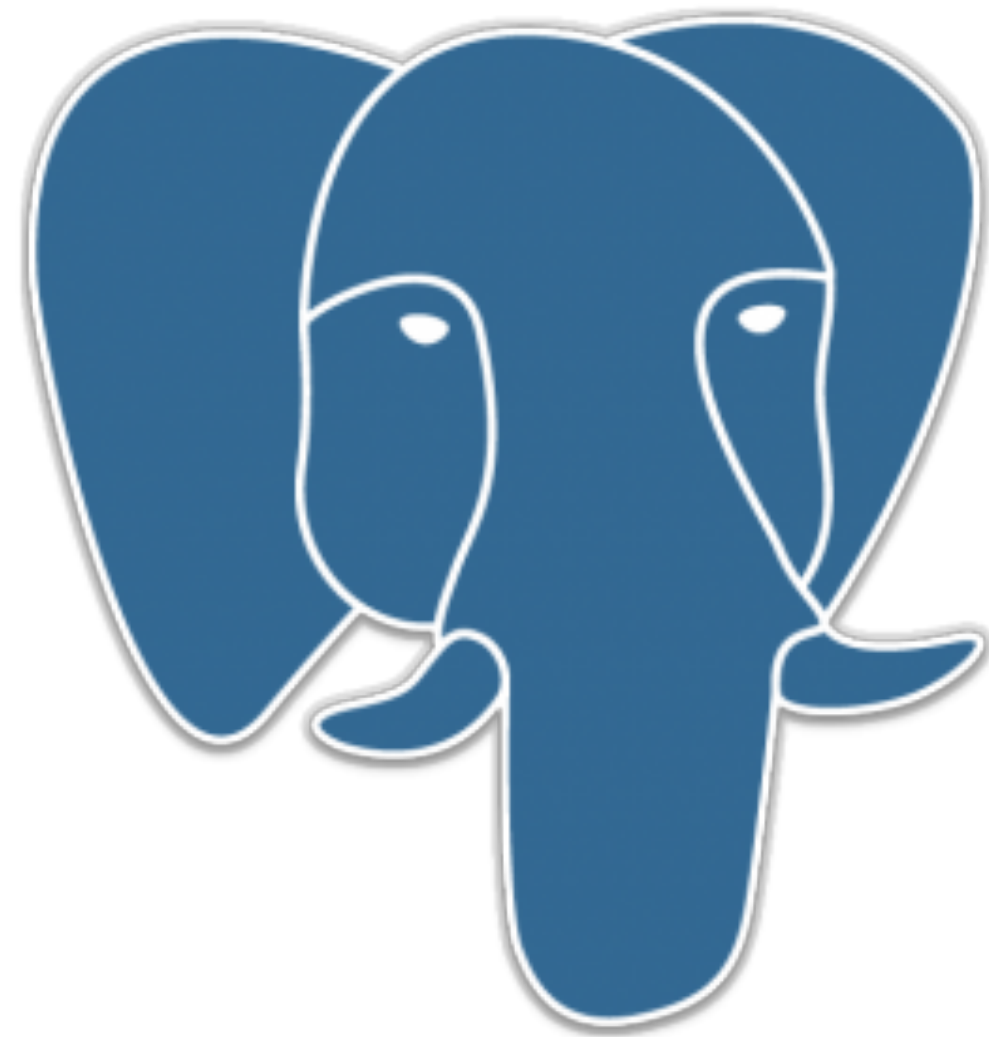


???



???

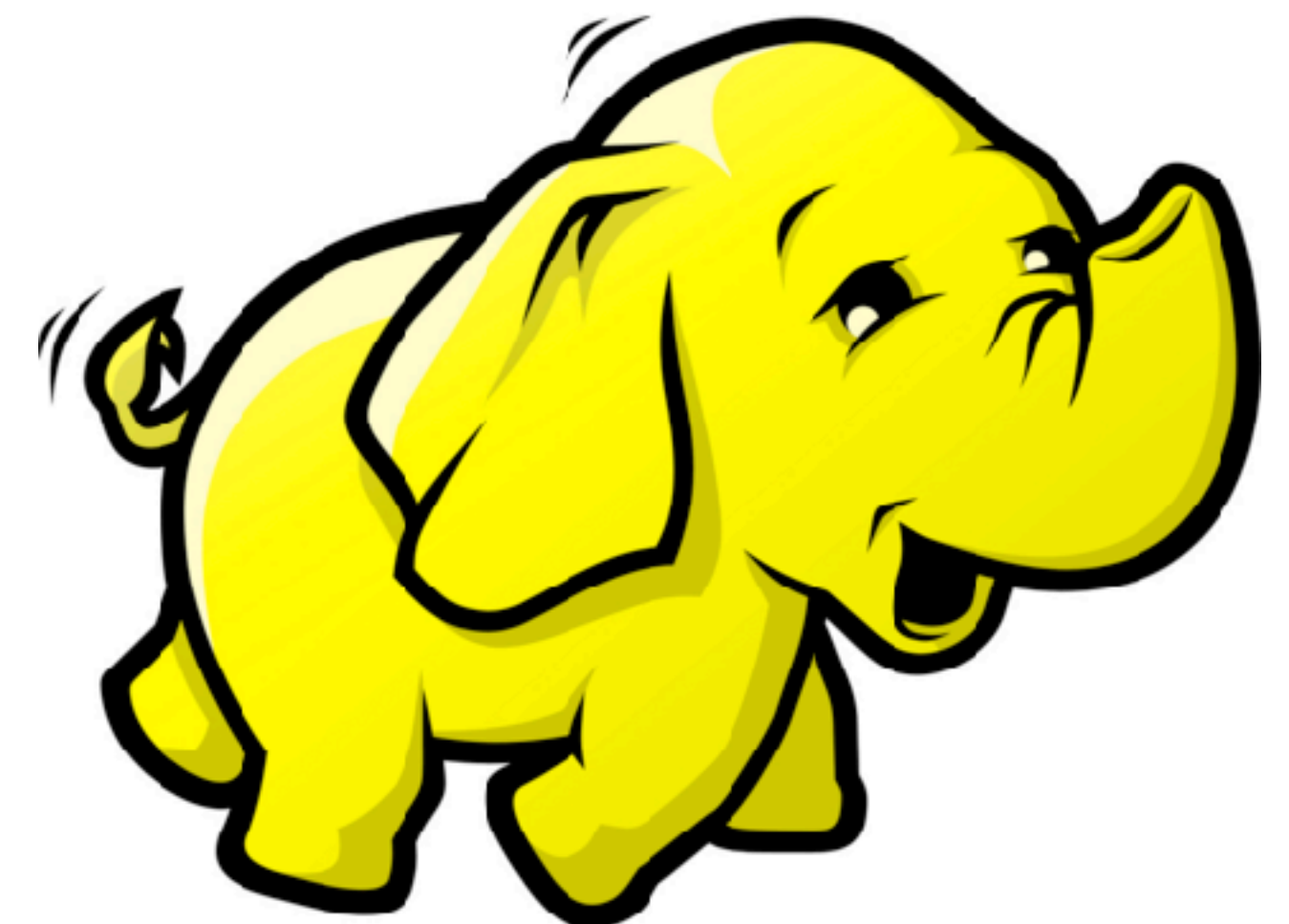
# Угадай слона



PostgreSQL



Gradle



Hadoop

# Исходные условия

- › Несколько библиотек с модульной структурой;
- › Собираются для iOS/Android и \*nix;
- › Есть платформенный код на Objective-C/C++ и Java;
- › Часть кода автоматически генерируется;
- › Инфраструктура (deb/rpm пакеты, CI, CD и т.д.);
- › У команды не было опыта с gradle/groovy.

# Gradle



# Gradle

Система автоматической сборки, использующая DSL на groovy, порядок сборки определяется ациклическим графом зависимостей.

- › рассчитан на большие проекты;
- › модульный и легко расширяемый;
- › прост в использовании.





# Новый проект

```
>> cat src/main.cpp
#include <iostream>

using namespace std;

int main(int, char**) {
    cout << "Hello world!" << endl;
    return 0;
}
```

```
>> gradle init && tree
.
├── build.gradle
├── gradle
│   └── wrapper
├── gradle-app.setting
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    └── main.cpp
```

# Gradle build lifecycle



**Initialization** - выполняется settings.gradle, определяются проекты участвующие в сборке.



**Configuration** - выполняется build.gradle файл, определяется логика сборки.



**Execution** - сборка.

# build.gradle

```
>> cat build.gradle
apply plugin: "cpp"
model {
    components {
        app(NativeExecutableSpec) {
            sources.cpp.source {
                srcDir "src"
                include "**/*.cpp"
            }
        }
    }
}
```

## Поддержка языков:

- › C
- › C++
- › Objective-C
- › Objective-C++
- › Assembler
- › Windows resources
- › Precompiled headers

# gradle tasks

```
>> ./gradlew tasks
```

```
-----
```

**appExecutable** – Assembles executable 'app:executable'.

**assemble** – Assembles the outputs of this project.

**build** – Assembles and tests this project.

**clean** – Deletes the build directory.

**installAppExecutable** – Installs a development image of appExecutable

```
>> ./gradlew appExecutable
```

```
:compileAppExecutableAppCpp
```

```
:linkAppExecutable
```

```
:appExecutable
```

```
BUILD SUCCESSFUL
```

```
model {  
  buildTypes {  
    debug  
    release  
  }  
  components {  
    app {  
      binaries.each {  
        if (buildType == buildType.debug)  
          cppCompiler.args << "-O0" << "-g"  
        else  
          cppCompiler.args << "-O3"  
      }  
    }  
  }  
}
```

# Build variants

```
>> ./gradlew tasks
Build tasks
-----
appDebugExecutable - ...
appReleaseExecutable - ...
assemble - ...
build - ...
clean - ...
installAppDebugExecutable - ...
installAppReleaseExecutable -
```

```
>> ./gradlew appReleaseExecutable
:compileAppReleaseExecutableAppCpp
:linkAppReleaseExecutable
:appReleaseExecutable

BUILD SUCCESSFUL
```

# Toolchains

```
model {  
  platforms {  
    linux  
    mac  
  }  
  components {  
    app {  
      targetPlatform "linux"  
      targetPlatform "mac"  
    }  
  }  
}
```

```
model {  
  toolChains {  
    gcc(Gcc) {  
      target("linux")  
    }  
    clang(Clang) {  
      target("mac")  
    }  
  }  
}
```

# Toolchains tasks

```
>> ./gradlew tasks
```

```
Build tasks
```

```
-----
```

```
appLinuxDebugExecutable
```

```
appLinuxReleaseExecutable
```

```
appMacDebugExecutable
```

```
appMacReleaseExecutable
```

```
assemble
```

```
installAppLinuxDebugExecutable
```

```
installAppLinuxReleaseExecutable
```

```
installAppMacDebugExecutable
```

```
installAppMacReleaseExecutable
```

```
>> ./gradlew appLinuxReleaseExecutable  
:compileAppLinuxReleaseExecutableAppCpp  
:linkAppLinuxReleaseExecutable  
:appLinuxReleaseExecutable
```

```
BUILD SUCCESSFUL
```



# Задачи

```
task run {  
    dependsOn "installAppLinuxReleaseExecutable"  
    doLast {  
        def installTask = tasks["installAppLinuxReleaseExecutable"]  
        exec {  
            executable installTask.executable  
        }  
    }  
}
```

# Задачи

```
task run {
    dependsOn "installAppLinuxReleaseExecutable"
    doLast {
        def installTask = tasks["installAppLinuxReleaseExecutable"]
        exec {
            executable installTask.executable
        }
    }
}
```

# Задачи

```
task run {  
    dependsOn "installAppLinuxReleaseExecutable"  
    doLast {  
        def installTask = tasks["installAppLinuxReleaseExecutable"]  
        exec {  
            executable installTask.executable  
        }  
    }  
}
```

# Задачи

```
>> ./gradlew tasks
```

```
Build tasks
```

```
-----
```

```
appLinuxDebugExecutable - ...
```

```
appLinuxReleaseExecutable - ...
```

```
...
```

```
Other tasks
```

```
-----
```

```
run
```

```
>> ./gradlew run
```

```
:compileAppLinuxReleaseExecutableAppCpp
```

```
:linkAppLinuxReleaseExecutable
```

```
:appLinuxReleaseExecutable
```

```
:installAppLinuxReleaseExecutable
```

```
:run
```

```
Hello world!
```

```
BUILD SUCCESSFUL
```

# Поддержка языков/инструментов

Работает «из коробки»

Linux	GCC/Clang
MacOS X	Xcode
Windows	Visual studio MinGW 32

**iOS/Android** - если указать пути до компиляторов.

**На любой ос** - если там есть JVM.

**С любым языком** - если написать свой плагин.

# Plugins



# Встроенные плагины

В стандартную поставку Gradle входит необходимый минимум плагинов:

- Компиляции (C, C++, Java, Groovy, Scala ...).
- Интеграция с IDE (IntelliJ Idea, Eclipse, Visual Studio).
- Тестирование (JUnit, GoogleTest и java),
- Интеграция с репозиториями.



# Внешние плагины

Много качественных внешних плагинов которые легко подключить и использовать;

```
plugins { id "nebula.ospackage" version "4.1.0" }
task pack(type: Deb) {
    packageName = "app"
    version = "1.2.3"
    dependsOn "installAppLinuxReleaseExecutable"
    from("$buildDir/install/app/linux/release") {
        into "/bin"
    }
}
```



# Создать свой плагин

## Простой

1. Включить gradle файл файл (почти как source в bash).

```
apply from: "../scripts/compile_util.gradle"
```

## Сложный

Написать реализацию интерфейса для класса Plugin (не только на Groovy на Java, Kotlin и Scala).

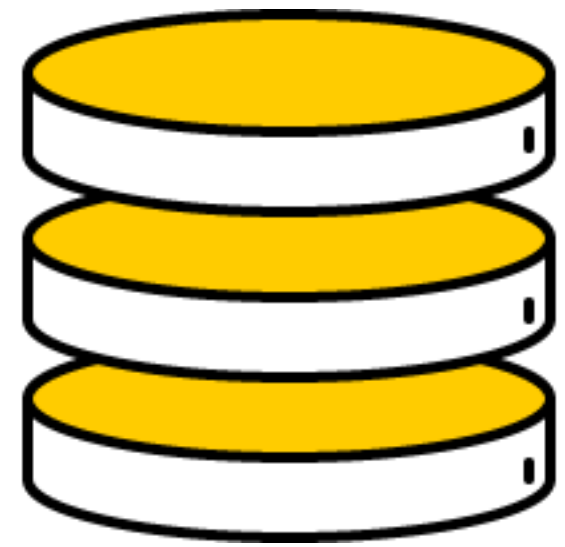
2. Положить исходнодники плагина в папку <rootProject>/buildSrc/src/main/groovy, при первом запуске gradle скомпилирует все сам.
3. Сделать отдельный gradle проект и загрузить его в репозиторий.

# Custom plugin

```
class DumpEnv implements Plugin<Project> {  
  void apply(Project project) {  
    project.tasks.create("systemInfo")  
    project.tasks.systemInfo.doLast {  
      println System.env.collect { k, v ->  
        "$k = $v"  
      }.join("\n")  
    }  
  }  
}
```

```
apply plugin: DumpEnv
```

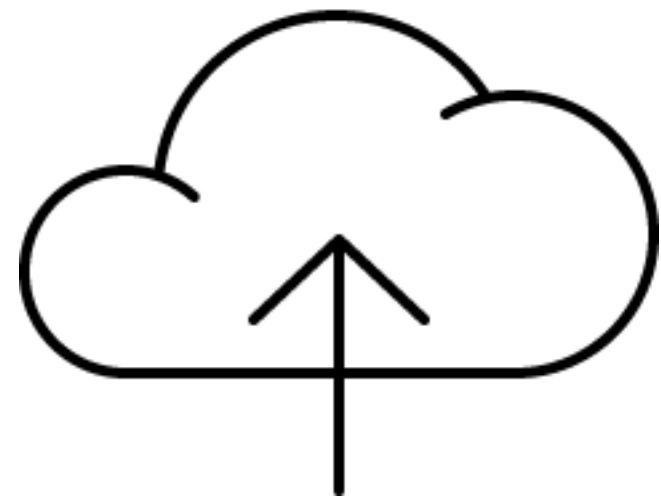
# Делиться плагинами



## Простой

1. Загрузить в репозиторий вместе с кодом

## Сложный



2. Скомпилировать и загрузить в Maven репозиторий

3. Скомпилировать и встроить в gradle.

# Tips & Tricks



# Для разработчика

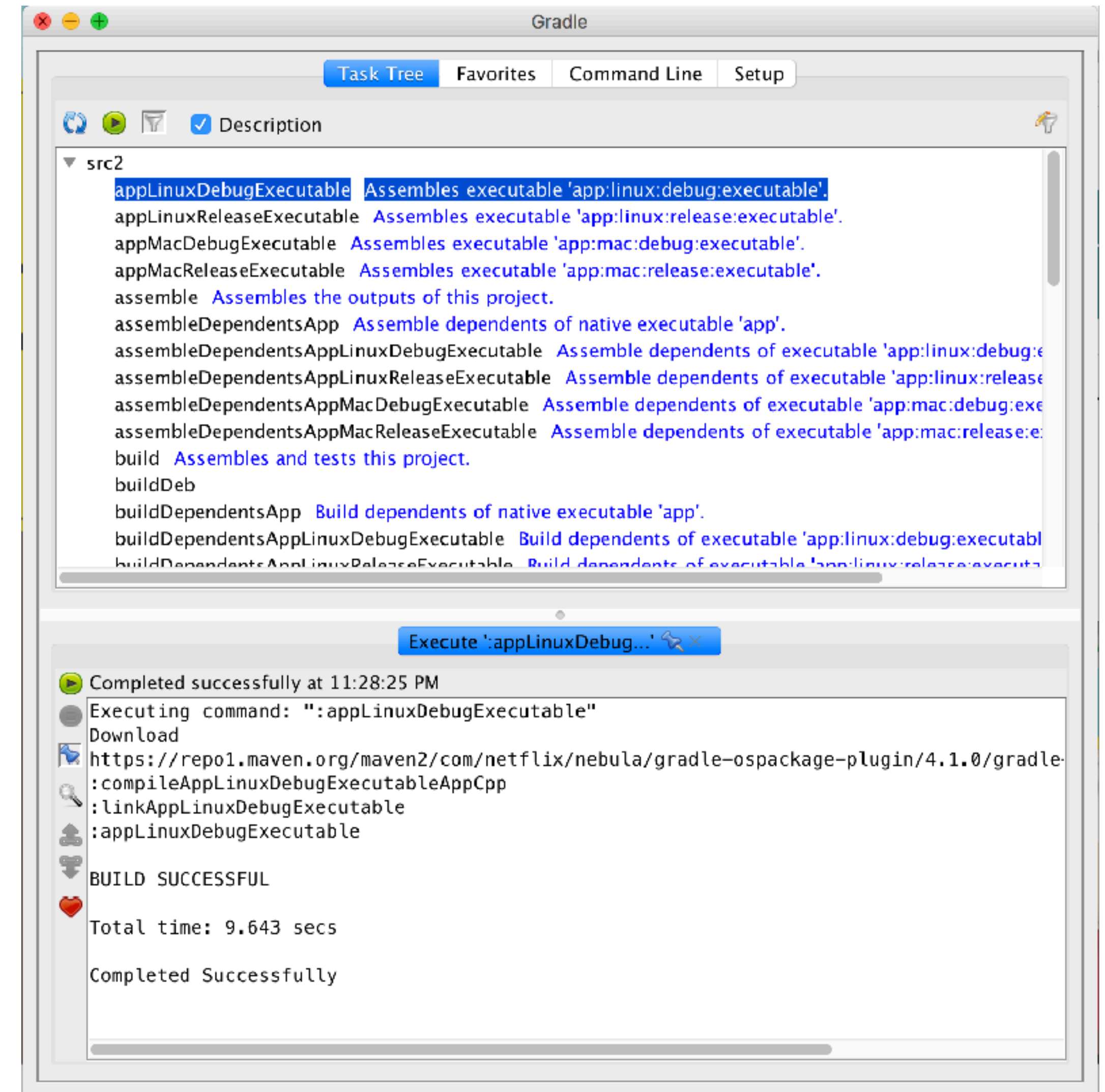
**Gradle daemon** - кэширует результаты, ускоряет сборку.

**--configure-on-demand** - ускорение конфигурации.

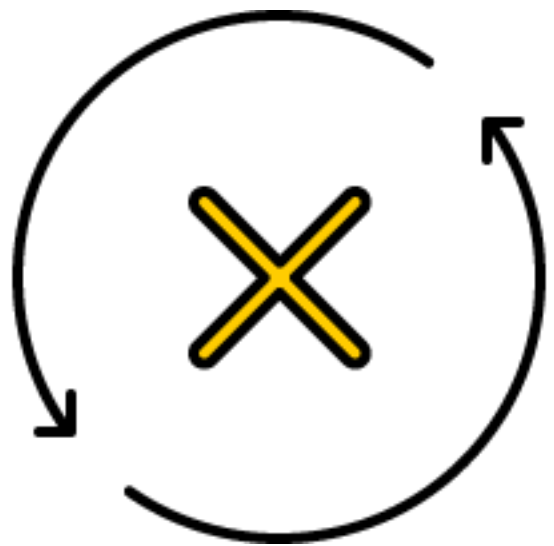
**--gui** - простой UI из которого можно запускать задачи.

**--offline** - gradle не будет скачивать зависимости из сети.

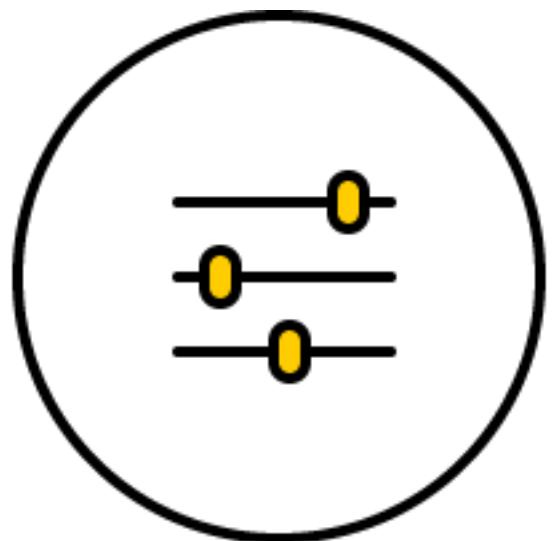
**groovysh** - интерактивная консоль.



# Для непрерывной интеграции



**--continuous**, gradle запущен всегда, при изменении файлов пересборка происходит автоматически.



**--parallel** - проекты начинают компилироваться параллельно.



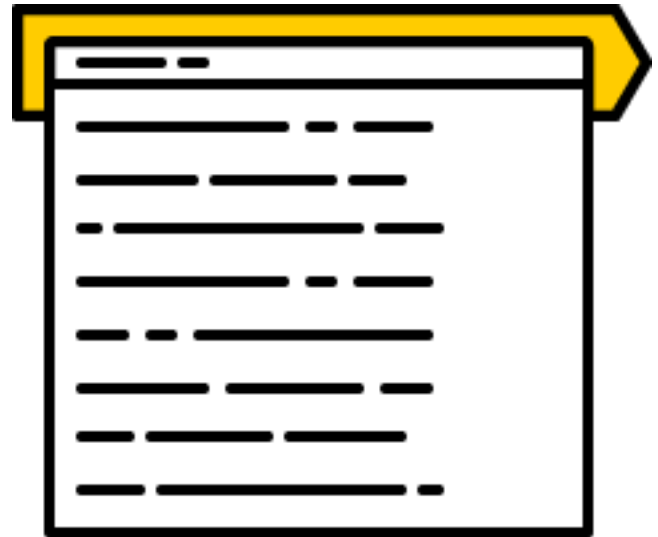
**--profile** - html отчет о времени сборки.

Встроить всю логику CI в gradle скрипты.

# Где искать помощь



Я хочу сделать новую фичу!



1. Есть плагин для gradle?

2. Как сделать это в groovy?

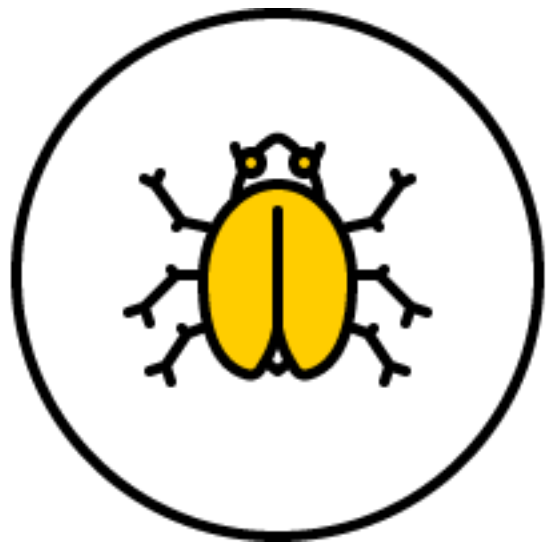
3. Как сделать это в java?



4. Спросить на [discuss.gradle.org](https://discuss.gradle.org) и [stackoverflow](https://stackoverflow.com).

5. Придумать самому.

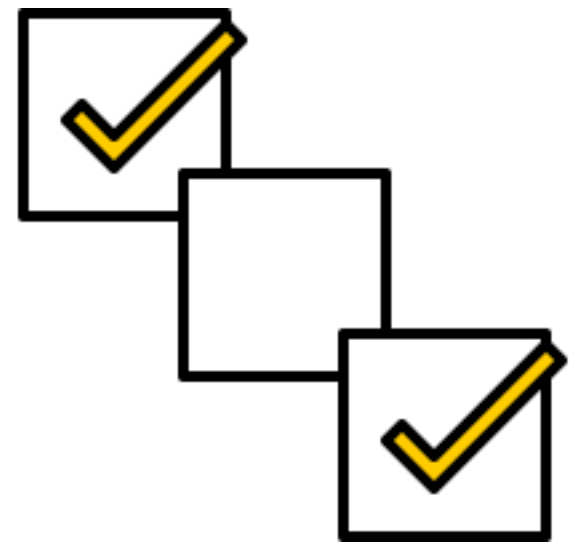
# Проблемы



- › Интеграция с C++ IDE;
- › Инкрементальная сборка, но не компиляция;
- › Долгая конфигурация при первом запуске;
- › API может неожиданно поменяться.



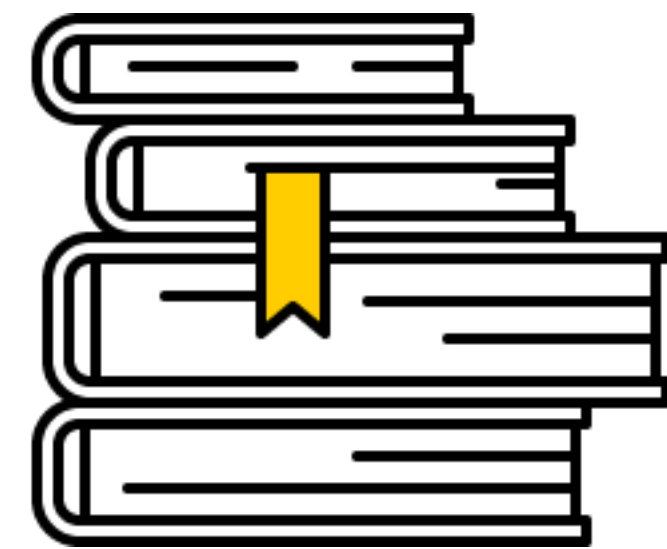
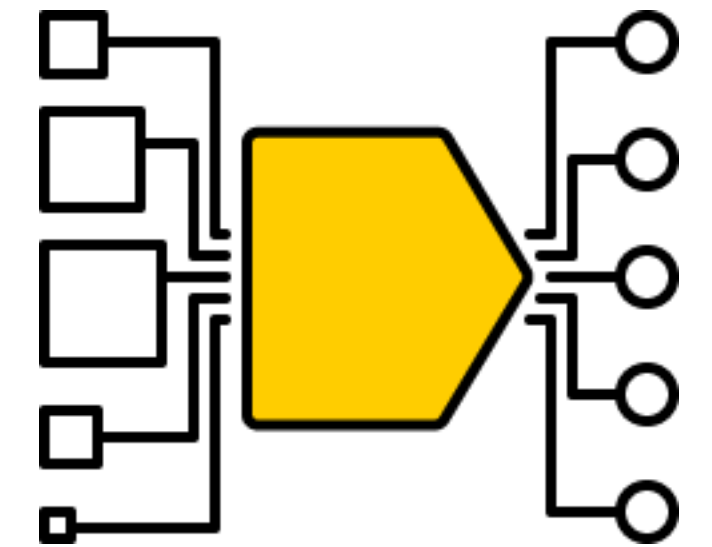
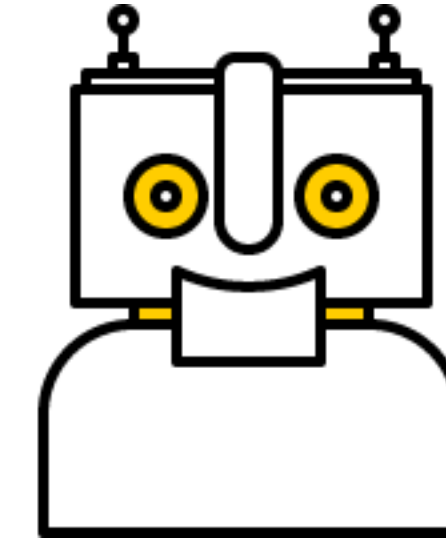
# А нужен ли gradle?



- › Ваше приложение кроссплатформенно.
- › Используется несколько языков.
- › Необходимо иметь полный контроль над процессом.
- › Нужна специфическая логика сборки.

# Чем ещё хорош gradle

- › Генерация кода (Qt MOC, google protobuf ...);
- › Мультипроектные/композиционные сборки;
- › Подключения бинарных библиотек;
- › Управление зависимостями.



# Ключевые идеи



- › Gradle - не только для java;
- › Gradle - просто использовать;
- › Много готовых плагинов;
- › Groovy - простой и гибкий язык;
- › Активно развивается.

# Вопросы?



**#S@&0%\*!**

