

Современный C++

Максим Федоренко

ИАТЭ НИЯУ МИФИ
Кафедра ПМ

3 декабря 2016 г.

Системы контроля версий



- ▶ Проект – множество файлов с исходным кодом
- ▶ Код изменяется
- ▶ За изменениями нужно следить
- ▶ Необходимо поддерживать несколько *версий* проекта

Наивное решение



- ▶ Хранить копии файлов
- ▶ Копии файлов пронумерованы

Недостатки:

- ▶ Приходится хранить несколько практически идентичных копий
- ▶ Требуется повышенное внимание и дисциплина
- ▶ Возникают ошибки
- ▶ Это неудобно



- ▶ Хранилище (репозиторий) находится на сервере
- ▶ Сервер совершает операции над репозиторием
- ▶ Пользователь работает с *рабочей копией*
- ▶ Изменения отправляются на сервер, формируется новая *ревизия*

Примеры:

- ▶ CVS
- ▶ Subversion (SVN)
- ▶ Perforce



- ▶ Клиент-серверная архитектура не обладает гибкостью
- ▶ Слабая поддержка ветвления
- ▶ Линейный подход к разработке



- ▶ Каждая рабочая копия – полноценный репозиторий
- ▶ Для работы не нужен сервер
- ▶ Хранилища синхронизируются между собой

Примеры:

- ▶ Git
- ▶ Mercurial
- ▶ Bazaar



- ▶ Линус Торвальдс, 2005 год
- ▶ Ядро Linux
- ▶ Около 10 миллионов строк кода





- ▶ Скорость
- ▶ Простая архитектура
- ▶ Хорошая поддержка нелинейной разработки (тысячи параллельных веток)
- ▶ Полная децентрализация
- ▶ Возможность эффективного управления большими проектами

Инструменты для работы с Git



- ▶ Командная строка
- ▶ SourceTree
- ▶ GitKraken



- ▶ Создать новый репозиторий:

```
$ git init .
```

- ▶ Клонировать существующий:

```
$ git clone https://github.com/VarLog/cpp_lessons
```

Принципы работы



- ▶ Рабочая копия

```
$ git status
```

- ▶ Добавление изменений в индекс

```
$ git add file1 file2
```



- ▶ Фиксация изменений

```
$ git commit
```

- ▶ Синхронизация между репозиториями

```
$ git fetch
```

```
$ git pull
```

```
$ git push
```

Удалённые репозитории



- ▶ Командная работа над проектом
- ▶ Работа на разных устройствах
- ▶ *Голые* репозитории
- ▶ **GitHub** – хостинг удалённых репозиторий





- ▶ Одно из главных достоинств Git
- ▶ *Коммиты* образуют цепочки
- ▶ Ветка – указатель на конкретный коммит
- ▶ Главная ветка – **master**
- ▶ Слияние веток

```
$ git branch new_branch
```

```
$ git checkout new_branch
```

```
$ git merge new_branch
```



- ▶ Стандартный подход
- ▶ Несколько ключевых веток:
 - `master` указывает на *release* версию
 - `develop` разработка следующей версии
- ▶ Вспомогательные ветки:
 - `feature/*` отдельные ветки для разработки
 - `release/*` стабилизационные ветки перед *релизом*
 - `hotfix/*` критические исправления

Git Flow



Сборка проекта



- ▶ Весь код в одном файле
- ▶ Компилятор C++
 - ▶ GNU Compiler Collection (gcc)
 - ▶ Clang/LLVM
 - ▶ MinGW (gcc на Windows)
- ▶ Создание исполняемого файла:

```
$ c++ main.cpp -o app  
$ g++ main.cpp -o app  
$ clang++ main.cpp -o app
```



- ▶ Заголовочные файлы
- ▶ Единицы трансляции
- ▶ Компиляция
- ▶ Компоновка

```
$ g++ -c main.cpp
```

```
$ g++ -c widget.cpp
```

```
$ g++ main.o widget.o -o app
```



- ▶ Предупреждения
- ▶ Уровни оптимизации
- ▶ Особенности платформы
- ▶ Режим отладки

```
$ g++ -g -Wall -pedantic -c main.cpp
```

```
$ g++ -O2 -flto -march=corei7 -c main.cpp
```



- ▶ Утилита make
- ▶ Специальные файлы Makefile
- ▶ Набор правил

```
$ make
```

```
...
```



- ▶ Набор утилит **Autotools**
- ▶ Автоматическая генерация `Makefile`
- ▶ Поиск зависимостей с помощью **pkg-config**
- ▶ Переносимость ПО



- ▶ Зависимые от *IDE*:
 - ▶ MS Visual Studio
 - ▶ Xcode
- ▶ Независимые:
 - ▶ SCons
 - ▶ CMake



- ▶ Кроссплатформенная система автоматизации сборки
- ▶ Генерирует файлы управления сборкой
- ▶ Правила генерации описаны в файле `CMakeLists.txt`





- ▶ Быстродействие
- ▶ Простой макроязык
- ▶ Подключаемые модули

Пример файла CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
PROJECT("test" CXX)
ADD_EXECUTABLE(app widget.cpp main.cpp)
```



- ▶ CMake умеет генерировать:
 - ▶ Makefile
 - ▶ Visual Studio проект
 - ▶ Xcode проект
 - ▶ и другие
- ▶ Многие IDE поддерживают CMake:
 - ▶ QtCreator
 - ▶ JetBrains CLion



► make-файлы

```
$ mkdir ./build
$ cd ./build
$ cmake ..
...
$ make
...
```



- ▶ Указать генератор явно

```
$ cmake -G "Xcode" ..
```

```
$ cmake -G "Visual Studio 10" ..
```

- ▶ Список генераторов

```
$ cmake -G list
```



► Переменные

```
project("game" CXX)

include_directories("include")

set(HEADERS "include/Game.h"
        "include/GameObject.h")

set(SOURCES "src/main.cpp"
        "src/Game.cpp")

add_executable(${PROJECT_NAME}
               ${SOURCES}
               ${HEADERS})
```



- Поиск файлов по шаблону

```
file(GLOB HEADERS "include/*.h")
file(GLOB SOURCES "src/*.cpp")

add_executable(${PROJECT_NAME}
               ${SOURCES}
               ${HEADERS})
```



► Поиск зависимостей

```
find_package(OpenMP)
if(OPENMP_FOUND)
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS}
        ${OpenMP_C_FLAGS}")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
        ${OpenMP_CXX_FLAGS}")
endif()
```




- ▶ Автоматическая сборка пакетов

Linux: DEB, RPM

Mac OS X: DMG

Windows: NSIS

Архивы: Tar, ZIP

Заключение



Git

<https://git-scm.com/book/ru/v2>

<https://habrahabr.ru/post/106912/>

CMake

<https://habrahabr.ru/post/155467/>

<https://habrahabr.ru/post/155397/>

<https://github.com/kaizouman/gtest-cmake-example>