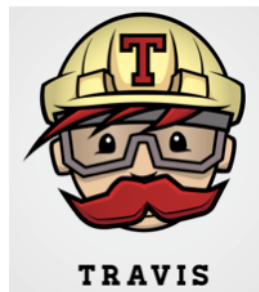


# Инструменты разработки ПО



Кирилл Корняков  
Директор по исследованиям и разработке, Itseez  
Февраль 2016

## Скелетонизация

Занятия
Регистрация (ННГУ, 2 корпус)
Открытие школы (ННГУ, 2 корпус)
Инструменты разработки ПО Корняков Кирилл (ННГУ, 2 корпус)
Кофе-брейк (ННГУ, 2 корпус)
Инструменты разработки ПО Корняков Кирилл (ННГУ, 2 корпус)
Обед (Комбинат питания)
Инструменты разработки ПО Алексей, Лебедев Илья (ННГУ, 6 корпус)
Кофе-брейк (ННГУ, 6 корпус)
Инструменты разработки ПО



Занятия
Регистрация (ННГУ, 2 корпус, а)
Открытие школы (ННГУ, 2 корпус)
Инструменты разработки ПО (
Корняков Кирилл (ННГУ, 2 корпус)
Кофе-брейк (ННГУ, 2 корпус, а)
Инструменты разработки ПО (
Корняков Кирилл (ННГУ, 2 корпус)
Обед (Комбинат питания Н
Инструменты разработки ПО (п
Алексей, Лебедев Илья (ННГУ, 6 корпус)
Кофе-брейк (ННГУ, 6 корпус, а)
Инструменты разработки ПО (п

# Используемые инструменты

Занятия
Регистрация (ИНГУ, 2 корпус)
Открытие школы (ИНГУ, 2 корпус)
Инструменты разработки ПО Корникова Кирилл (ИНГУ, 2 корпус)
Кофе-брейки (ИНГУ, 2 корпус)
Инструменты разработки ПО Корникова Кирилл (ИНГУ, 2 корпус)
Обед (Комбинат питания)
Инструменты разработки ПО Алексей, Лебедев Илья (ИНГУ, 6 корпус)
Кофе-брейки (ИНГУ, 6 корпус)
Инструменты разработки ПО



Занятия
Регистрация (ИНГУ, 2 корпус)
Открытие школы (ИНГУ, 2 корпус)
Инструменты разработки ПО Корникова Кирилл (ИНГУ, 2 корпус)
Кофе-брейки (ИНГУ, 2 корпус)
Инструменты разработки ПО Корникова Кирилл (ИНГУ, 2 корпус)
Обед (Комбинат питания)
Инструменты разработки ПО Алексей, Лебедев Илья (ИНГУ, 6 корпус)
Кофе-брейки (ИНГУ, 6 корпус)
Инструменты разработки ПО



# Содержание

## 1. Кросс-платформенная разработка

- *[C++]*, *CMake*

## 2. Коллективная работа с кодом

- *Git*, *GitHub*

## 3. Автоматическое тестирование

- *Google Test*, *Travis-CI*

# Программная реализация

 3rdparty

 docs

 include

 perf

 sample

 src

 test

 testdata

 .gitignore

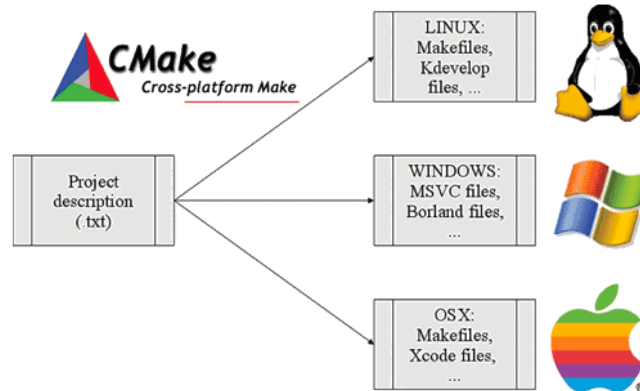
 .travis.yml

 CMakeLists.txt

 README.md



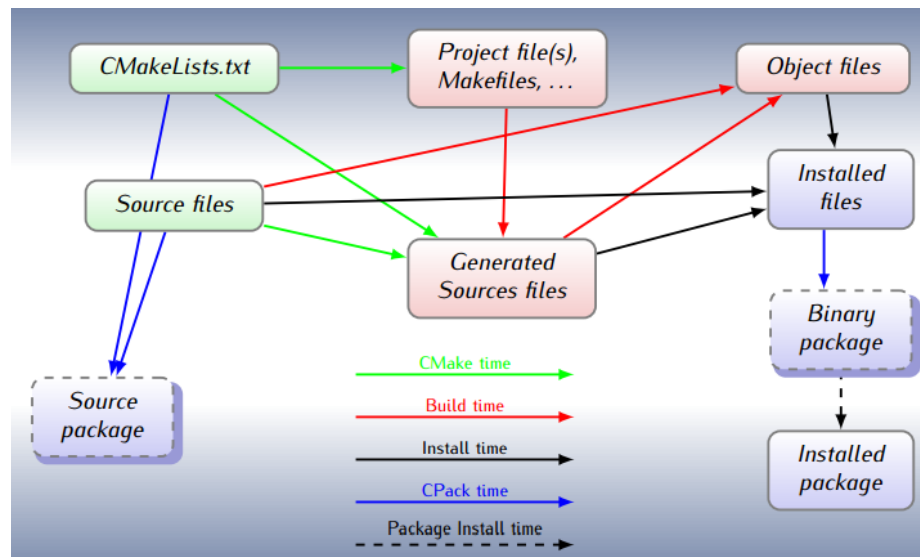
# CMake



- Широкая поддержка разнообразных целевых платформ и IDE
- Максимальная свобода в выборе окружения разработки (в рамках одной команды!)
- В настоящий момент является стандартом де-факто для C++ проектов

# CMake Workflow

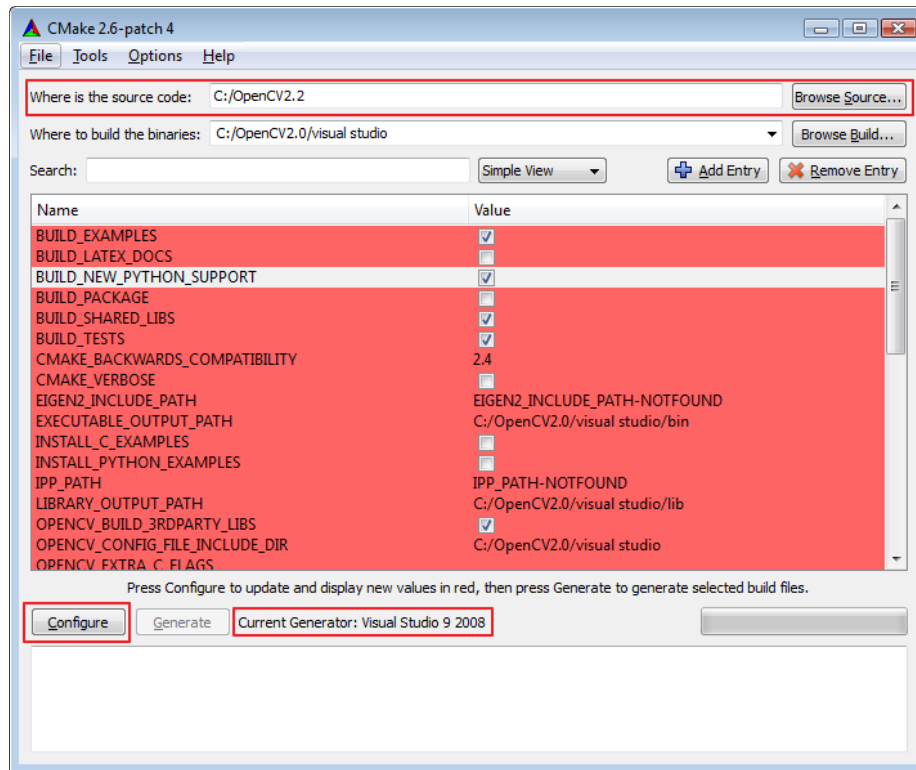
CMakeLists.txt – файл, описывающий порядок сборки приложения



- **Шаг 0.** Генерация *проектных файлов* при помощи cmake или CMakeGui
  - *.vcproj, Makefile, etc*
- **Шаг 1.** Компиляция исходников при помощи компиляторов из Visual Studio, Qt Creator, Eclipse, XCode...
  - *.obj, .o*
- **Шаг 2.** Линковка финальных бинарных файлов компоновщиком (link.exe, ld, ...)
  - *.exe, .dll, .lib, .a, .so, .dylib*



# CMake GUI



# Пример сборки приложения (add\_executable)

Содержимое каталога:

```
code
├── CMakeLists.txt
├── lib.h
├── lib.c
└── main.c
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(first_sample)

set(SOURCES main.c lib.c)
add_executable(sample_app ${SOURCES}) # Объявляет исполняемый модуль с именем sample_app
```

# Out of source build

Плохо: в директории с исходным кодом

```
code
├─ hello.hpp
├─ hello.cpp
└─ hello.exe # Этот файл может случайно попасть в историю Git
```

Хорошо: вне директории (чистый репозиторий, несколько build-директорий)

```
code
├─ hello.hpp
└─ hello.cpp
build
└─ hello.exe
```

Соответствующие команды:

```
$ cd <code>
$ mkdir ../build
$ cd ../build
$ cmake ../code
$ make
```

# Пример сборки библиотеки (add\_library)

Содержимое каталога:

```
code
├── CMakeLists.txt
├── lib.h
├── lib.c
└── main.c
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(second_sample)

set(SOURCE_LIB lib.c)
add_library(library STATIC ${SOURCE_LIB}) # Объявляет библиотеку с именем library

set(SOURCES main.c)
add_executable(main ${SOURCES}) # Объявляет исполняемый модуль с именем sample_app
target_link_libraries(sample_app library) # Указывает зависимость от библиотеки
```

# Добавление подпроекта

Содержимое каталога:

```
code
├── CMakeLists.txt
├── library
│   ├── CMakeLists.txt
│   ├── lib.c
│   └── lib.h
└── main.c
```

Корневой CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8)
project(third_sample)

add_subdirectory(library) # Указывает, что в директории library есть свой CMakeLists.txt

include_directories(library)
set(SOURCES main.c)
add_executable(sample_app ${SOURCES})

target_link_libraries(sample_app library)
```

library/CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(library)

set(SOURCE_LIB lib.c)
add_library(library STATIC ${SOURCE_LIB})
```



# Поиск зависимостей

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(sample)

# Поиск OpenCV
find_package(OPENCV REQUIRED)
if(NOT OPENCV_FOUND)
    message(SEND_ERROR "Failed to find OpenCV")
    return()
else()
    include_directories(${OPENCV_INCLUDE_DIR})
endif()

add_executable(sample_app main.c)
target_link_libraries(sample_app ${OPENCV_LIBRARIES})
```

# Debug / Release

В CMakeLists.txt:

```
SET(CMAKE_BUILD_TYPE Debug)
```

В командной строке:

```
$ cmake -DCMAKE_BUILD_TYPE=Debug ../code # Запомните эту команду!
```

Для библиотек:

```
TARGET_LINK_LIBRARIES(lib RELEASE ${lib_SRCS})  
TARGET_LINK_LIBRARIES(libd DEBUG ${lib_SRCS})
```



# СMake: Резюме

- Основной "недостаток" — собственный язык
- Поначалу инструмент кажется нетривиальным, но очень удобен впоследствии
- Дает членам команды максимальную свободу в выборе инструментов (ОС, IDE или простой текстовый редактор)
- Обеспечивает переносимость и является стандартом де-факто для кросс-платформенных C++ проектов



# Коллективная работа с кодом

## 1. История изменений

- *Откат дефектных изменений*
- *Извлечение кода "из прошлого" (как оно раньше работало?)*
- *Поиск ошибок сравнением (кто это сделал?)*

## 2. Централизованное хранение

- *Актуальное и используемое всеми участниками (где последняя версия?!)*
- *Защищенное, с разграничением прав доступа*

**Машина времени и сетевое хранилище в одном флаконе!**

Нужны ли специальные инструменты? Вспоминаем Sharepoint, tarballs.

## Системы контроля версий

*Системы контроля версий – это программные системы, хранящие несколько версий одного документа, и позволяющие вернуться к более ранним версиям. Как правило, для каждого изменения запоминается дата модификации и автор.*

# Патчи

*Патч (англ. patch — заплатка) — информация, предназначенная для автоматизированного внесения определённых изменений в компьютерные файлы.*

Unified diff format: @@ -l,s +l,s @@ optional section heading

```

13
14 diff --git a/README.md b/README.md
15 index afadff2..bde857e 100644
16 --- a/README.md
17 +++ b/README.md
18 @@ -40,10 +40,10 @@ __Цель данной работы__ - реализовать набор пр
19     содержащий простейшую реализацию класса матриц. Предполагается, что он не
20     редактируется при реализации фильтров.
21     - Модуль `filters` ( `./include/filters.hpp`, `./src/filters_opencv.cpp`,
22       `./src/filters_fabrics.cpp`), содержащий объявление абстрактного класса
23 +   `./src/filters_factory.cpp`), содержащий объявление абстрактного класса
24     фильтров (`filters.hpp`) и его наследника, который реализует перечисленные
25     фильтры средствами библиотеки OpenCV (`filters_opencv.cpp`), а также метод
26 -   создания конкретной реализации класса фильтров (`filters_fabrics.cpp`).
27 +   создания конкретной реализации класса фильтров (`filters_factory.cpp`).
28     - Тесты для класса матриц и фильтров (`matrix_test.cpp`, `filters_test.cpp`).
29     - Пример использования фильтра (`matrix_sample.cpp`).
30
31 @@ -489,11 +489,11 @@ __Примечание:__ генератор проекта должен сов
32     значение, соответствующее вашей реализации фильтра. Назовите его
33     согласно вашей фамилии `YOUR_NAME`. Указанное перечисление используется
34     при прогоне одних и тех же тестов на всех реализациях класса фильтров.
35 -   1. В файле `filters_fabrics.cpp` объявите функцию
36 +   1. В файле `filters_factory.cpp` объявите функцию
37     `Filters* createFiltersYourName()`. Данная функция будет использована
38     при создании объекта класса с вашей реализации фильтров.
39     1. В функции `Filters* createFilters(FILTERS_IMPLEMENTATIONS impl)` (файл
40 -   `filters_fabrics.cpp`) необходимо добавить еще одну ветку у оператора-
41 +   `filters_factory.cpp`) необходимо добавить еще одну ветку у оператора-
42     переключателя `switch`, по которой будет проходить исполнение программы,
43     если создан объект класса фильтров `YOUR_NAME`.
44     1. В файл `filters_YOUR_NAME.cpp` необходимо поместить реализацию функции

```

# Отображение на GitHub

8	README.md	<> View
✚	@@ -40,10 +40,10 @@ __Цель данной работы__ - реализовать набор пр	
40	40	содержащий простейшую реализацию класса матриц. Предполагается, что он не
41	41	редактируется при реализации фильтров.
42	42	- Модуль <code>filters</code> ( <code>./include/filters.hpp</code> , <code>./src/filters_opencv.cpp</code> ,
43	-	<code>./src/filters_fabrics.cpp</code> ), содержащий объявление абстрактного класса
43	+	<code>./src/filters_factory.cpp</code> ), содержащий объявление абстрактного класса
44	44	фильтров ( <code>filters.hpp</code> ) и его наследника, который реализует перечисленные
45	45	фильтры средствами библиотеки OpenCV ( <code>filters_opencv.cpp</code> ), а также метод
46	-	создания конкретной реализации класса фильтров ( <code>filters_fabrics.cpp</code> ).
46	+	создания конкретной реализации класса фильтров ( <code>filters_factory.cpp</code> ).
47	47	- Тесты для класса матриц и фильтров ( <code>matrix_test.cpp</code> , <code>filters_test.cpp</code> ).
48	48	- Пример использования фильтра ( <code>matrix_sample.cpp</code> ).
49	49	
✚	@@ -489,11 +489,11 @@ __Примечание:__ генератор проекта должен сов	
489	489	значение, соответствующее вашей реализации фильтра. Назовите его
490	490	согласно вашей фамилии <code>YOUR_NAME</code> . Указанное перечисление используется
491	491	при прогоне одних и тех же тестов на всех реализациях класса фильтров.
492	-	1. В файле <code>filters_fabrics.cpp</code> объявите функцию
492	+	1. В файле <code>filters_factory.cpp</code> объявите функцию
493	493	<code>Filters* createFiltersYourName()</code> . Данная функция будет использована
494	494	при создании объекта класса с вашей реализации фильтров.
495	495	1. В функции <code>Filters* createFilters(FILTERS_IMPLEMENTATIONS impl)</code> (файл
496	-	<code>filters_fabrics.cpp</code> ) необходимо добавить еще одну ветку у оператора-
496	+	<code>filters_factory.cpp</code> ) необходимо добавить еще одну ветку у оператора-
497	497	переключателя <code>switch</code> , по которой будет проходить исполнение программы,
498	498	если создан объект класса фильтров <code>YOUR_NAME</code> .
499	499	1. В файл <code>filters_YOUR_NAME.cpp</code> необходимо поместить реализацию функции
✚		

## Отображение в командной строке



```
~/Work/summer-school-2015/practice1-devtools (master*)> git show dc2ca9d95c
commit dc2ca9d95cbd5586e9e5ef0felce7db91ea7d3d1
Author: Daniil Osokin <daniil.osokin@itseez.com>
Date: Sun Aug 16 14:35:44 2015 +0300
```

Switched to factory

```
diff --git a/README.md b/README.md
index afadff2..bde857e 100644
```

```
--- a/README.md
```

```
+++ b/README.md
```

```
@@ -40,10 +40,10 @@ __Цель данной работы__ - реализовать набор пр
    содержащий простейшую реализацию класса матриц. Предполагается, что он не
    редактируется при реализации фильтров.
- Модуль `filters` ( `./include/filters.hpp`, `./src/filters_opencv.cpp`,
-   `./src/filters_fabrics.cpp` ), содержащий объявление абстрактного класса
+   `./src/filters_factory.cpp` ), содержащий объявление абстрактного класса
    фильтров ( `filters.hpp` ) и его наследника, который реализует перечисленные
    фильтры средствами библиотеки OpenCV ( `filters_opencv.cpp` ), а также метод
- создания конкретной реализации класса фильтров ( `filters_fabrics.cpp` ).
+ создания конкретной реализации класса фильтров ( `filters_factory.cpp` ).
- Тесты для класса матриц и фильтров ( `matrix_test.cpp`, `filters_test.cpp` ).
- Пример использования фильтра ( `matrix_sample.cpp` ).
```

```
@@ -489,11 +489,11 @@ __Примечание:__ генератор проекта должен сов
    значение, соответствующее вашей реализации фильтра. Назовите его
    согласно вашей фамилии `YOUR_NAME`. Указанное перечисление используется
    при прогоне одних и тех же тестов на всех реализациях класса фильтров.
```

- 1. В файле `filters\_fabrics.cpp` объявите функцию
- + 1. В файле `filters\_factory.cpp` объявите функцию  
`Filters\* createFiltersYourName()`. Данная функция будет использоваться  
при создании объекта класса с вашей реализации фильтров.
- 1. В функции `Filters\* createFilters(FILTERS\_IMPLEMENTATIONS impl)` (файл  
`filters\_fabrics.cpp`) необходимо добавить еще одну ветку у оператора-  
переключателя `switch`, по которой будет проходить исполнение программы,  
если создан объект класса фильтров `YOUR\_NAME`.
- 1. В файл `filters\_YOUR\_NAME.cpp` необходимо поместить реализацию функции

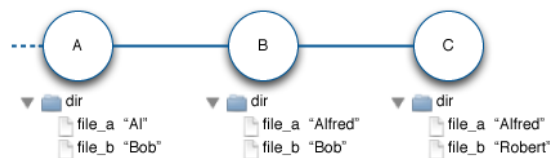
# Терминология

## Патчи

- Патч — это простой текстовый файл, его можно наложить при помощи инструментов (patch).
- Один патч может содержать изменения сразу нескольких файлов в разных директориях.
- Люди могут обмениваться изменениями, посылая друг другу патчи.
- Патч — это атомарное изменение проекта!

## Патчи и СКВ

- СКВ — это своего рода БД патчей, ее называют **репозиторием**.
- Патчи, помещенные в СКВ называются **commit**.
- Последовательности **commit** называются **changeset**.



# История изменений

ddc4a1d — Readme bug fixes.

- README.md

f9e76e6 — Remove dummy implementation

- include/filters.hpp
- samples/matrix\_sample.cpp
- src/filters\_dummy.cpp
- src/filters\_fabrics.cpp
- test/filters\_test.cpp

























aa1611b — Switch from strings to enums

- include/filters.hpp
- samples/matrix\_sample.cpp
- src/filters\_fabrics.cpp
- test/filters\_test.cpp

8e8b21b — Add some error checking

- include/filters.hpp

Commits on Aug 11, 2015

	<b>Readme bug fixes.</b> valentina-kustikova authored 24 days ago	 ddc4a1d	
	<b>Remove dummy implementation</b> kirill-kornyakov authored 24 days ago	 f9e76e6	
	<b>Switch from strings to enums</b> kirill-kornyakov authored 24 days ago	 aa1611b	
	<b>Add some error checking</b> kirill-kornyakov authored 24 days ago	 8e8b21b	
	<b>Run polymorphic tests</b> kirill-kornyakov authored 24 days ago	 a2ab7d7	
	<b>Description bug fixes.</b> valentina-kustikova authored 24 days ago	 6591fb4	
	<b>Description bug fixes.</b> valentina-kustikova authored 24 days ago	 b30fba0	
	<b>Description bug fixes.</b> valentina-kustikova authored 24 days ago	 71db5e6	

Последовательность патчей — это полная история проекта.

- `src/filters_fabrics.cpp`
- `test/filters_test.cpp`

# Визуализация истории изменений

OpenCV 2.4.0



- <http://www.youtube.com/watch?v=ToD91PYaQOU>

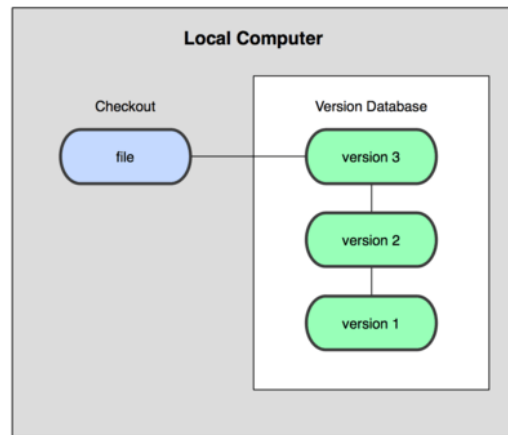
- Сделано при помощи [gource](#)

## Три поколения СКВ

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	CVS, Subversion, SourceSafe, Team Foundation Server
Third	Distributed	Changesets	Commit before merge	Git, Mercurial, Bazaar

Eric Sink ["A History of Version Control"](#)

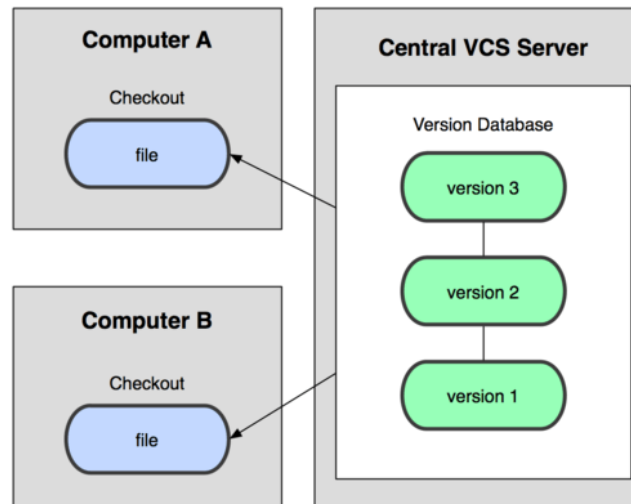
## Три поколения СКВ: Локальные



Примеры: RCS, SCCS

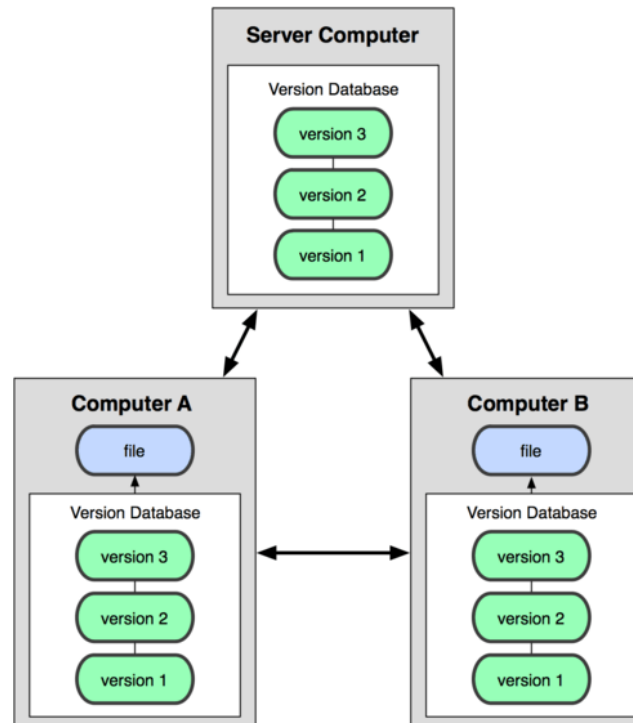


## Три поколения СКВ: Централизованные



Примеры: Subversion, CVS

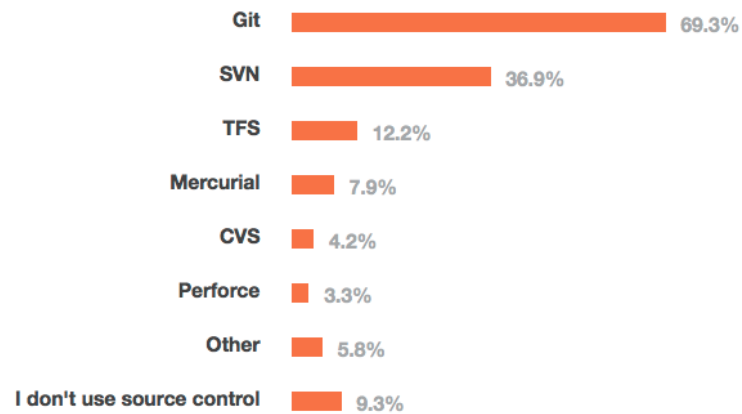
## Три поколения СКВ: Распределенные



- Примеры: Git, Mercurial
- Фактически стали стандартом де-факто
- Сильные стороны:
  - *Допускают локальную работу (коммиты без наличия интернет)*

- Упрощают слияние (а значит параллельную разработку)
- Дают максимальную свободу по организации рабочего процесса (workflow)

# Популярные СКВ



16,694 responses

## Stack Overflow Developer Survey 2015

### ■ Использование в ИТ-проектах:

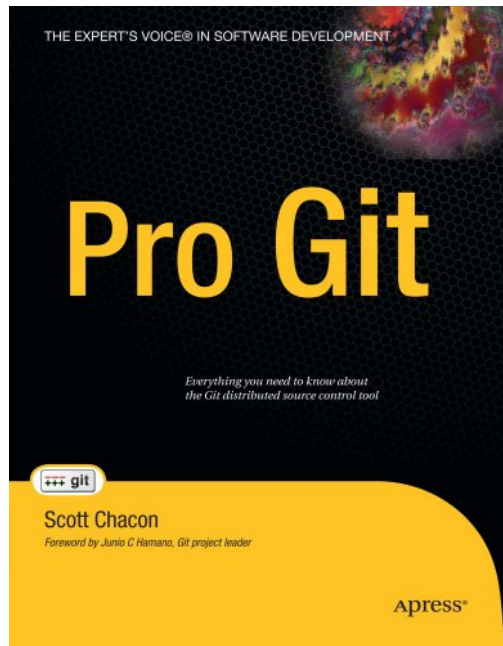
- *Фундаментальный инструмент разработки*
- *Также используется для: файлы конфигурации, документация, тестовые данные и пр.*

# Git



- Разработан Линусом Торвальдсом для работы над ядром Linux в 2005 году.
- В настоящее время поддерживается Джунио Хамано, сотрудником Google.
- Не очень прост в освоении, однако очень быстрый и функциональный.
- Имеет наиболее "сильное" сообщество, инструментальную поддержку.
- Огромное количество информации в интернет: инструкции, уроки, статьи
- Официальный сайт проекта: <http://www.git-scm.org>.

# Pro Git

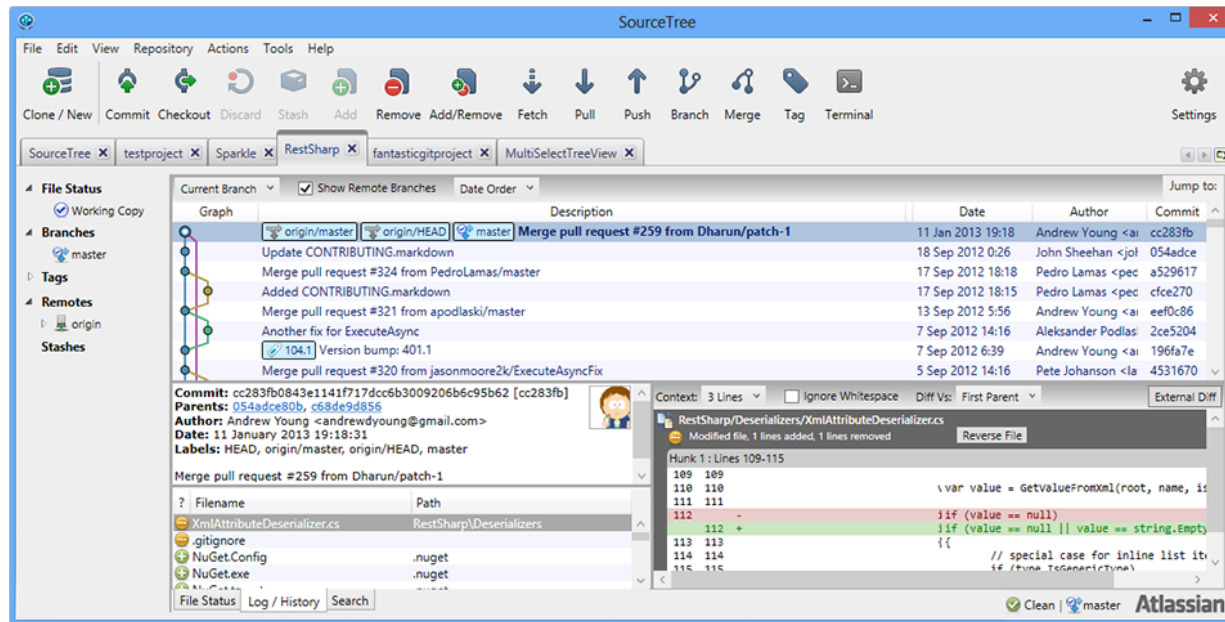


- Лучшая книга про Git
- Доступна бесплатно
- Переведена на [русский язык](#)
- Единственный способ по-настоящему понять Git — это узнать как он работает
- Нужно прочесть хотя бы первые 100 страниц

Как сказал Евклид египетскому царю Птолемею:

*«Царской дороги в геометрии нет!»*

# Atlassian SourceTree



# Tortoise Git

E:\dev\libs-nosync\1\Cinder - Log Messages - TortoiseGit

dev From: 21/04/2010 To: 12/03/2013 Filter by Subject, Messages, Paths, Authors, Emails, SHA-1, Refname, Bug-IDs Author Email

Graph	Actions	Message	Author	Date
		Working dir changes		
		<b>dev  origin/dev Merge branch 'appRewrite' into dev</b>	<b>Andrew Bell</b>	<b>12/03/2013 00:43:58</b>
		Fixing reference to App in QuickTime.cpp	Andrew Bell	12/03/2013 00:42:46
		Changing VC11 foundation template to use v110	Andrew Bell	12/03/2013 00:30:00
		Upgrading VC11 to default to v110 compiler	Andrew Bell	12/03/2013 00:23:57
		Fixing missing sstream	Andrew Bell	11/03/2013 23:04:13
		Adding 1.53 VC10 and VC11 binaries	Andrew Bell	11/03/2013 23:03:46
		Upgrading MSW Boost binaries to 1.53	Andrew Bell	11/03/2013 23:02:43
		QuickTime.cpp change to build MSW qtime against Boost 1.53	Andrew Bell	11/03/2013 22:18:44
		Fixing Cinder-Boost submodule path	Andrew Bell	11/03/2013 20:57:23
		Adding boost to .gitmodules	Andrew Bell	11/03/2013 20:51:26
		Upgrading Cinder to Boost 1.53, submodule, Mac OS/IOS binaries	Andrew Bell	11/03/2013 20:43:57
		Finalized Mac screensaver refactor merge	Andrew Bell	10/03/2013 04:53:13
		Merge pull request #308 from calebjohnston/Vector-Color-additions	Andrew Bell	07/03/2013 17:28:21

SHA-1: 746dcefbeddd8a7aa8f339465e2e1c415dada728

\* Merge branch 'appRewrite' into dev

Path	Extension	Status	L..	L..
<b>Diff with parent 1</b>				
.gitignore	.gitignore	Modified	2	2
.gitmodules	.gitmo...	Added	9	0
README.md	.md	Modified	4	6
blocks/Box2D/cinderblock.png	.png	Added	-	-
blocks/Box2D/cinderblock.xml	.xml	Added	27	0
blocks/Box2D/src/Box2D/Box2D.h	.h	Added	67	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Chai...	.cpp	Added	1..	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Chai...	.h	Added	1..	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Circ...	.cpp	Added	1..	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Circ...	.h	Added	91	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Edge...	.cpp	Added	1..	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Edge...	.h	Added	74	0

Showing 1847 revision(s), from revision 1fdb3ba to revision 746dcef - 1 revision(s) selected

☒ Hide Unrelated Changed Paths
 ☐ Show Whole Project

☐ All Branches
 ☐ Follow renames

☐ First Parent
 ☒ Show tags

Refresh

Statistics

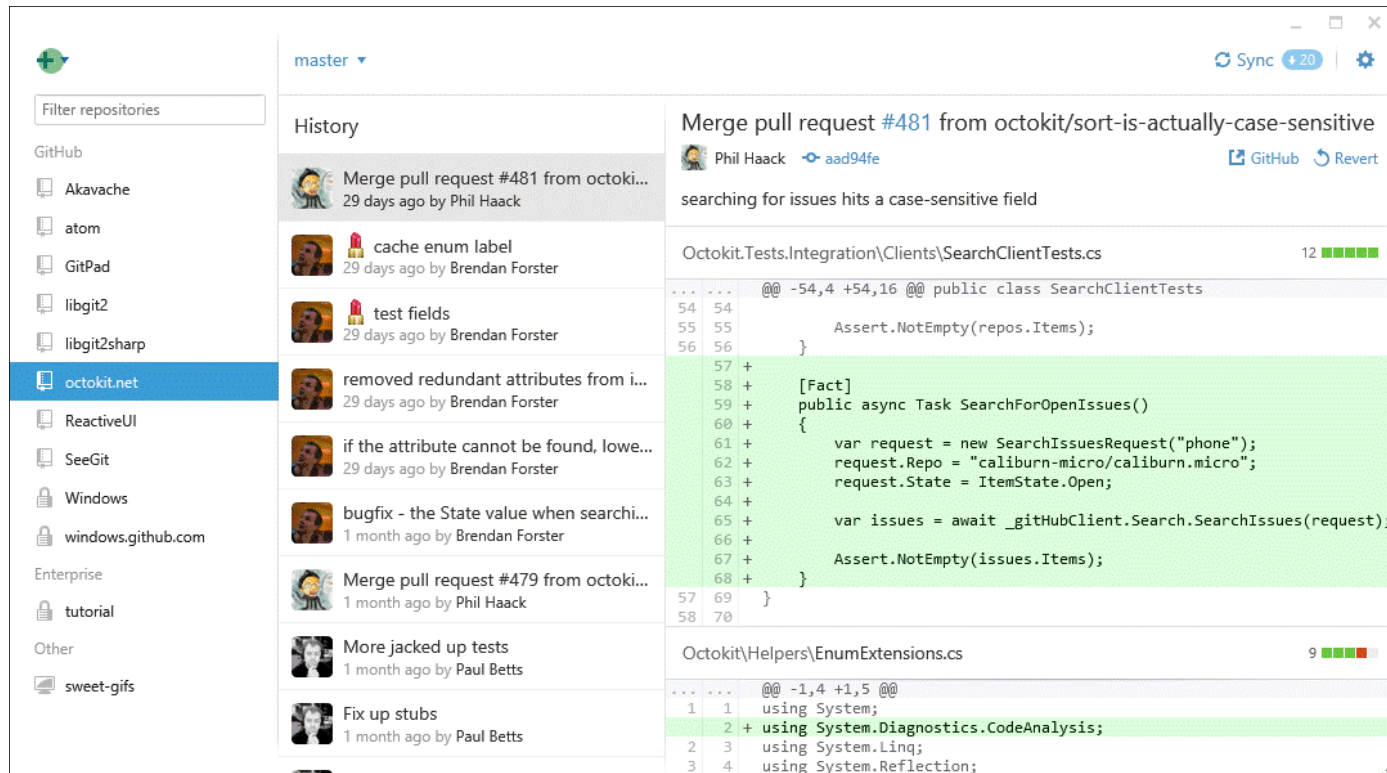
Help

OK





# GitHub Desktop



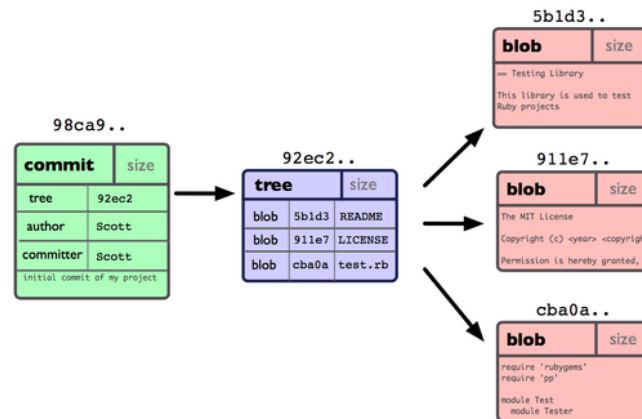
# Command Line Interface!

```
-> git help
usage: git [--version] [--help] [-C <path>] [-c name=value]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]

The most commonly used git commands are:
add      Add file contents to the index
bisect   Find by binary search the change that introduced a bug
branch   List, create, or delete branches
checkout Checkout a branch or paths to the working tree
clone    Clone a repository into a new directory
commit   Record changes to the repository
diff     Show changes between commits, commit and working tree, etc
fetch    Download objects and refs from another repository
grep     Print lines matching a pattern
init     Create an empty Git repository or reinitialize an existing one
log      Show commit logs
merge    Join two or more development histories together
mv       Move or rename a file, a directory, or a symlink
pull     Fetch from and integrate with another repository or a local branch
push     Update remote refs along with associated objects
rebase   Forward-port local commits to the updated upstream head
reset    Reset current HEAD to the specified state
rm       Remove files from the working tree and from the index
show     Show various types of objects
status   Show the working tree status
tag      Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

# Git objects



- По сути это *внутреннее* представление патча
- Пользователю приходится работать только с коммитами (слава богу!)

Показать содержимое коммита:

```
$ git show --raw dc2ca9d95c

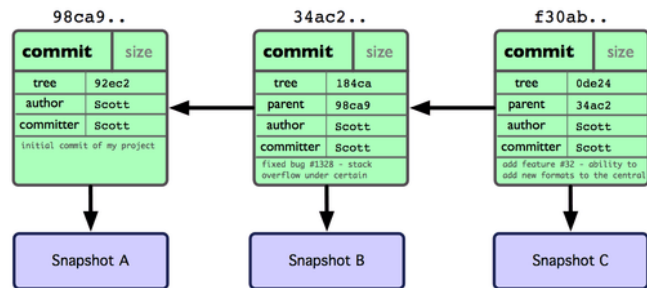
commit dc2ca9d95cbd5586e9e5ef0felce7db91ea7d3d1
Author: Daniil Osokin <daniil.osokin@itseez.com>
Date: Sun Aug 16 14:35:44 2015 +0300

    Switched to factory

:100644 100644 afadff2... bde857e... M README.md
:100644 000000 c977bf3... 0000000... D src/filters_fabrics.cpp
:000000 100644 0000000... c977bf3... A src/filters_factory.cpp
```



# Git commits



Вывести историю изменений:

```
$ git log

commit aaa321be9191da60ad52c2bc41bd749ed546b409
Merge: 98fce98 3c1d15a
Author: Valentina <valentina-kustikova@users.noreply.github.com>
Date: Thu Aug 13 10:14:47 2015 +0300

    Merge pull request #11 from valentina-kustikova/master

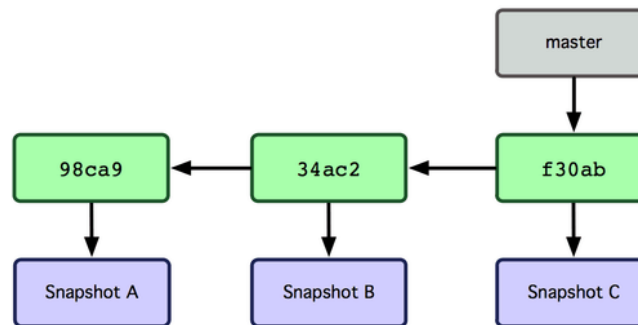
    Practice description (bug fixes).

commit 3c1d15a1bf366864593f2320fa9a0e6cf3586f52
Author: valentina-kustikova <valentina.kustikova@gmail.com>
Date: Thu Aug 13 10:08:59 2015 +0300

    Practice description (bug fixes).
```

# Понятие ветки (branch)

- Ветка в Git'e — это просто **указатель** на один из коммитов.
- Есть соглашение, что имя **master** используется для ветки, указывающей на последнее актуальное состояние проекта.



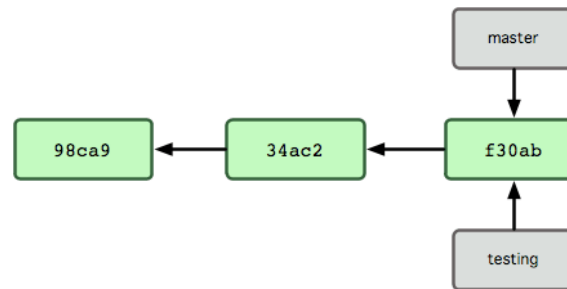
Вывести список существующих веток:

```
$ git branch
* master
```

# Git branch

Создать новую ветку с именем `testing` (указатель на коммит!):

```
$ git branch testing
```



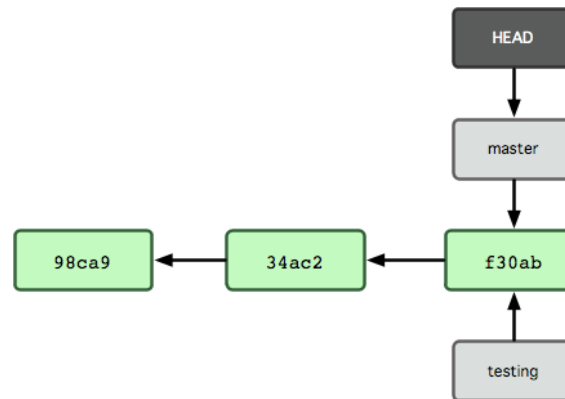
Текущий список веток:

```
$ git branch
* master
testing
```



# HEAD

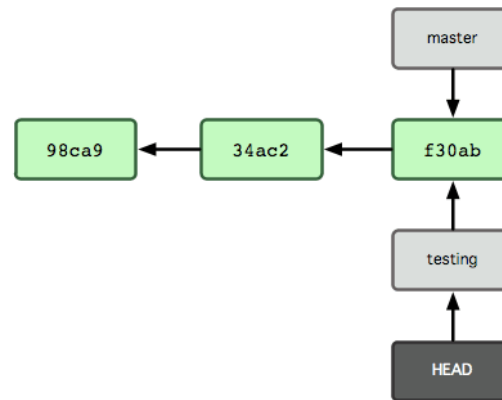
- HEAD — специальный указатель, ссылающийся на локальную ветку, на которой вы находитесь.
- Это просто алиас для текущей ветки, введенный для удобства.



# Git checkout

Извлечь состояние репозитория, соответствующее ветке testing:

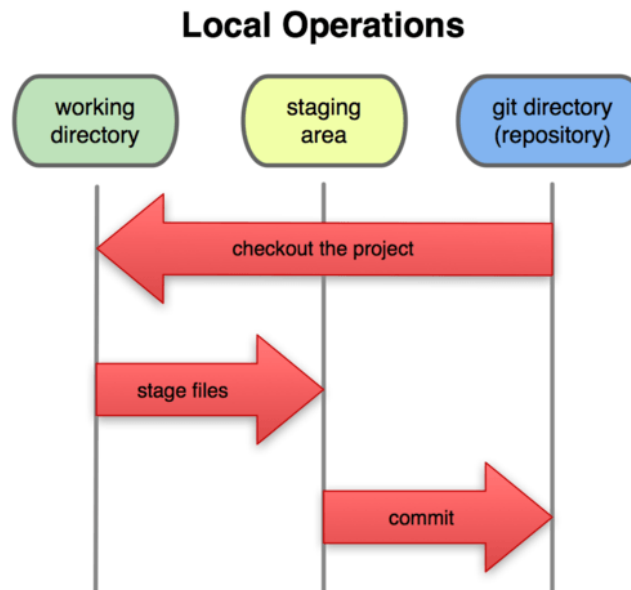
```
$ git checkout testing
```



Вывести список существующих веток:

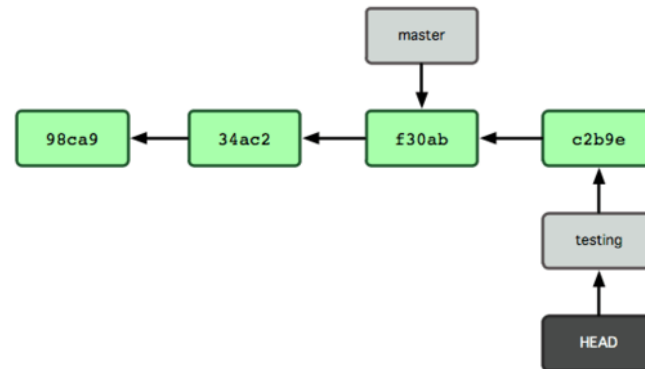
```
$ git branch
master
* testing
```

# Три состояния файлов



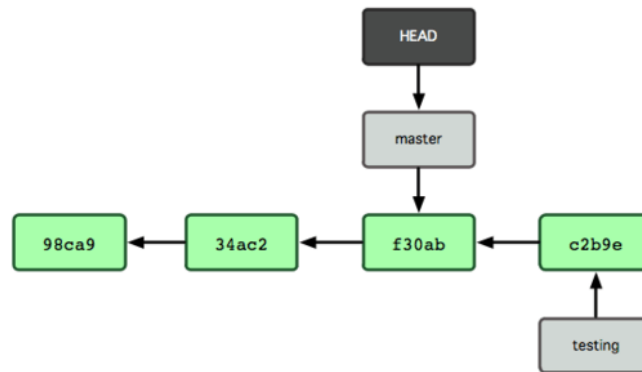
# Git commit

```
$ vim README.md  
$ git add README.md  
$ git commit -m 'Made a change'
```



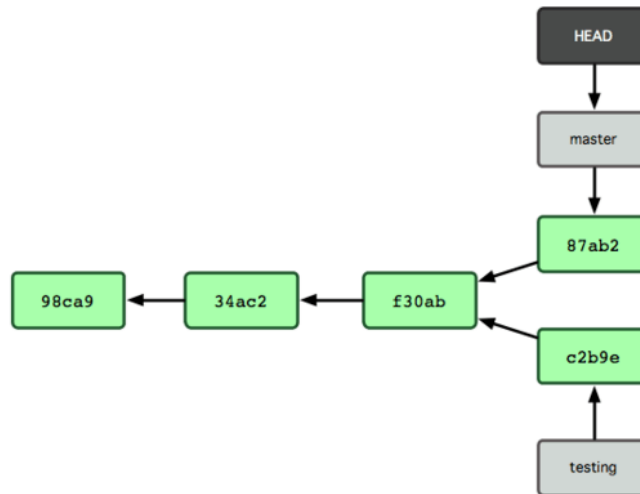
# Go back to master

```
$ git checkout master
```

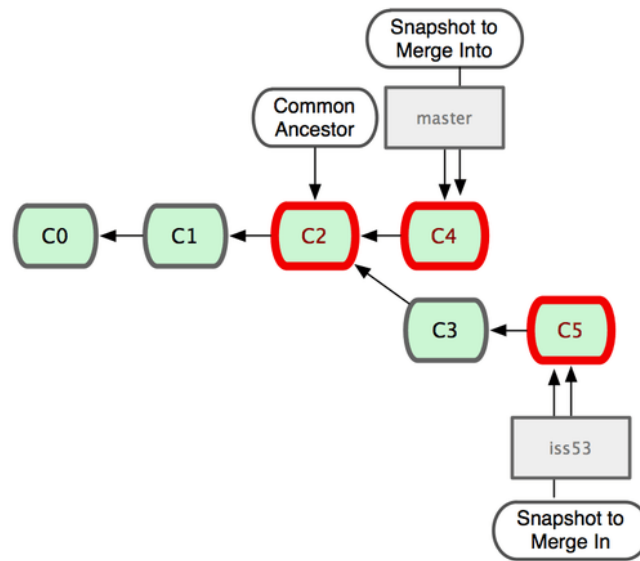


# Make a commit to master

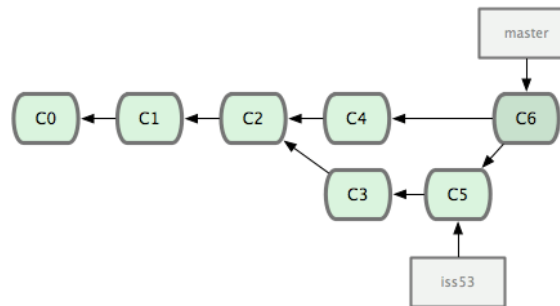
```
$ vim main.cpp  
  
$ git add main.cpp  
$ git commit -m 'Made other changes'  
  
# Или можно сделать так  
$ git status  
$ git commit -a -m 'Made other changes'
```



# Merging



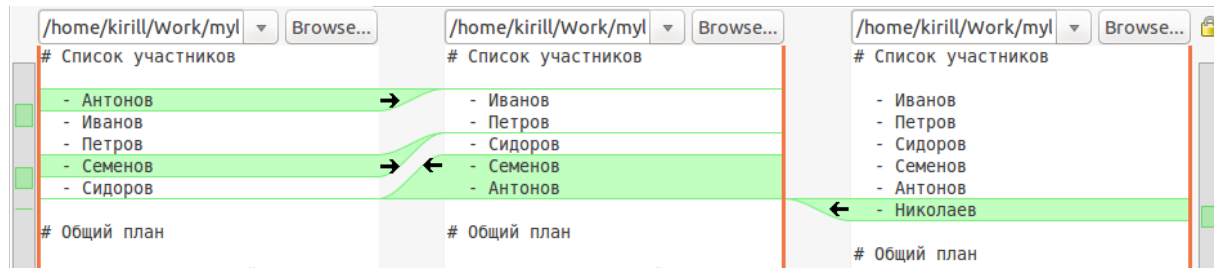
# Merging



- C6 — это так называемый *merge commit*
- Он основан не на каком-то патче, он указывает на состояние проекта, в котором наложены патчи обеих ветвей (master и testing).

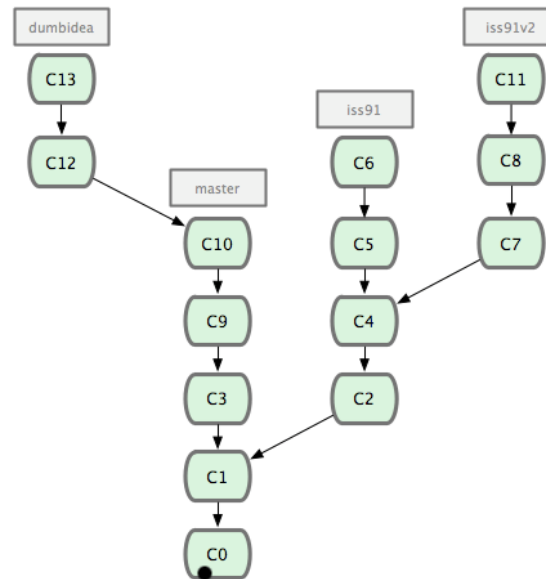


# Merge Conflicts



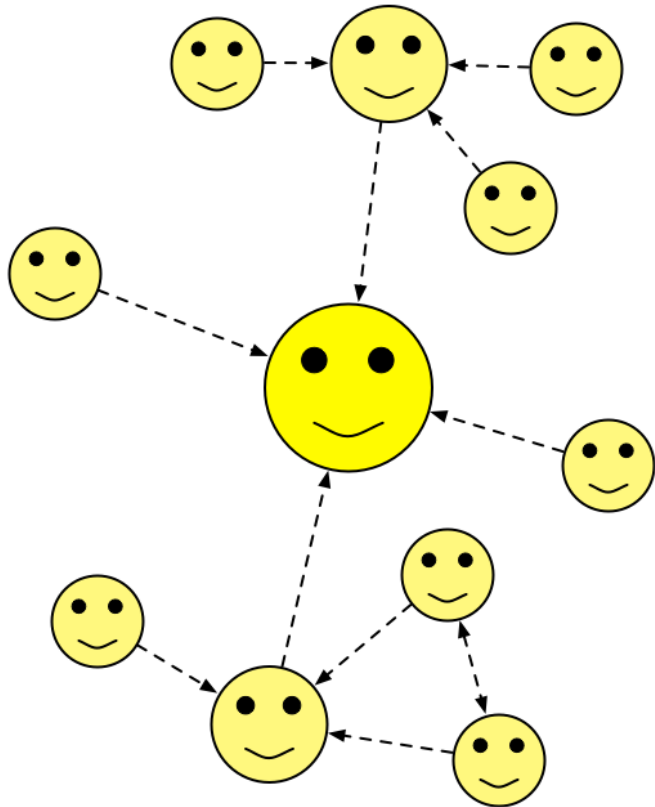
- Возникают когда несколько участников отредактировали одинаковые строки, или когда это произошло в разных ветках.
- Разрешаются человеком при помощи инструментов (`git mergetool`).
- В реальности довольно редкая ситуация, если соблюдать практики:
  - Грамотное распределение задач
  - Частые коммиты, много маленьких веток, частая интеграция

# Multiple branches

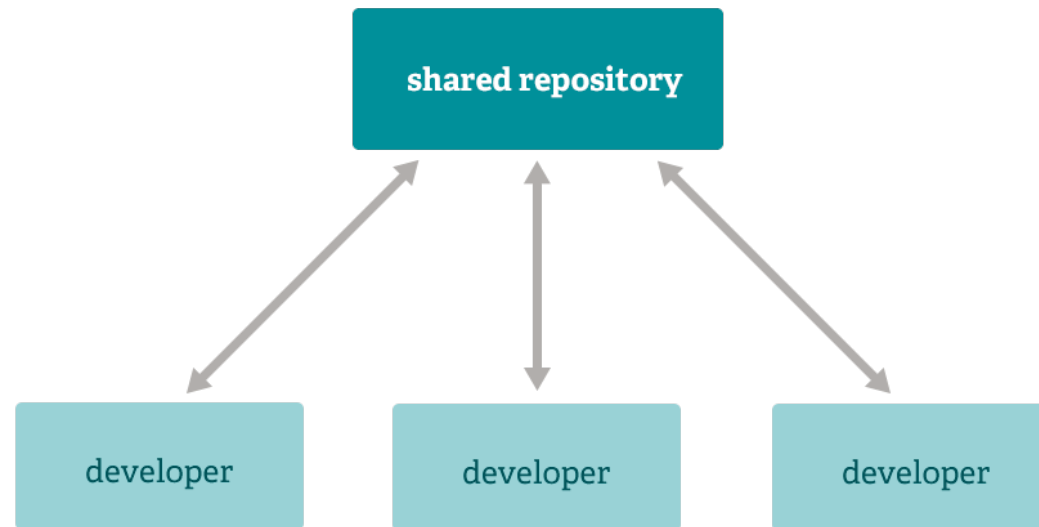


- Даже у одного разработчика может быть несколько активных веток.
- Правильно создавать отдельную ветку на каждую логически независимую задачу.
- Долгоживущие ветки — это неправильно, они быстро устаревают.

## Распределенная работа

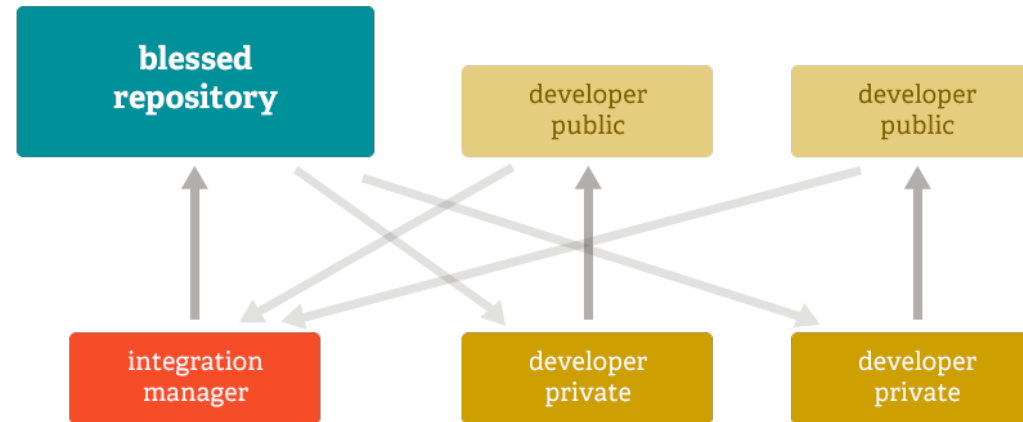


# Centralized Workflow



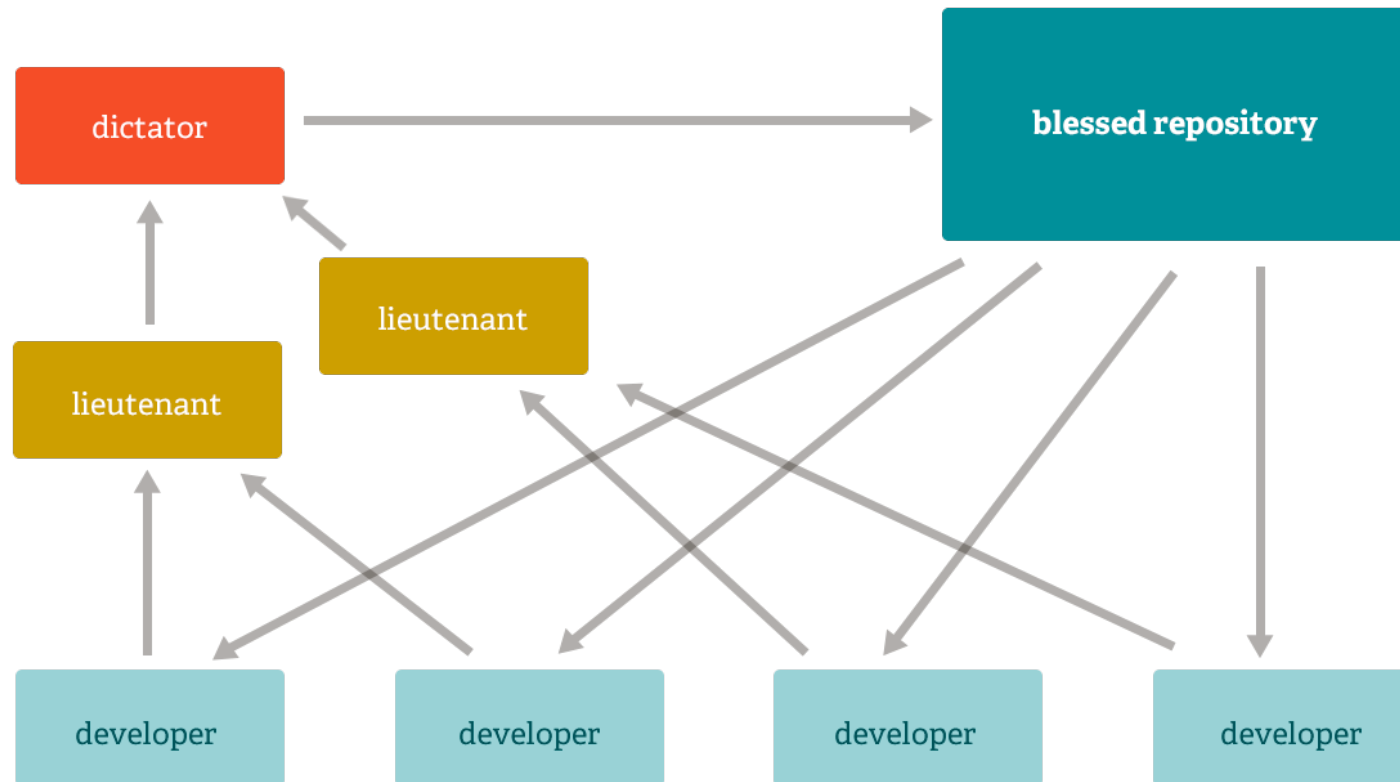
Плюсы и минусы данного подхода?

# Integration Manager Workflow

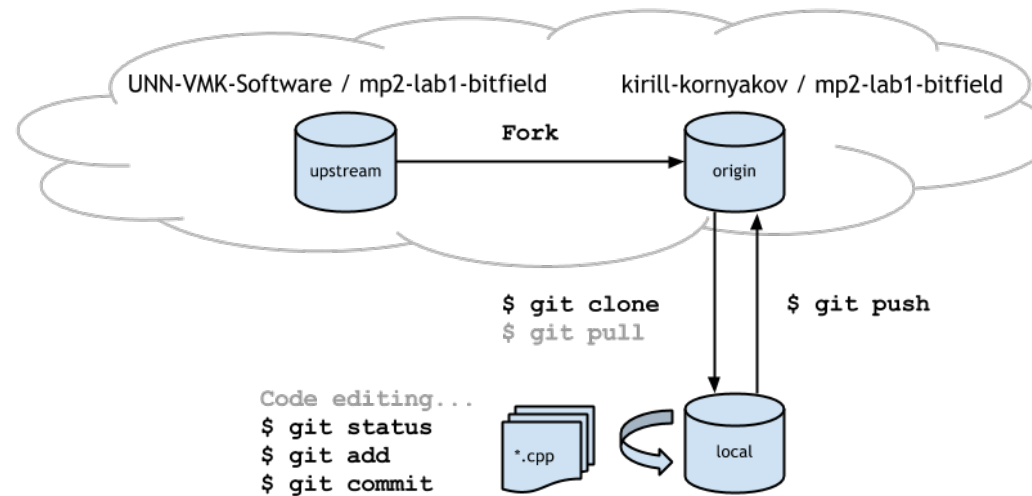


Плюсы и минусы данного подхода?

## Dictator and Lieutenants Workflow



# Triangular Workflow (GitHub)



```
$ cd mp2-lab1-bitfield
$ git remote -v
origin https://github.com/kirill-kornyakov/mp2-lab1-bitfield.git (fetch)
origin https://github.com/kirill-kornyakov/mp2-lab1-bitfield.git (push)
upstream https://github.com/UNN-VMK-Software/mp2-lab1-bitfield.git (fetch)
upstream https://github.com/UNN-VMK-Software/mp2-lab1-bitfield.git (push)
```

# GitHub Flow

github / github

Admin Unwatch Fork Your Fork Pull Request 16 12

Source Commits Network Pull Requests (23) Fork Queue Issues (290) Wiki (98) Graphs Branch: master

Switch Branches (139) Switch Tags (0) Branch List Search source code...

### Branches

Showing 30 of 139 branches

Recently Active Stale

Branch	Status	Compare
<b>master</b>	Base branch	
fi-signup	3 ahead 0 behind	Compare
charlock-linguist	11 ahead 7 behind	Compare
git-http-server	11 ahead 7 behind	Compare
wild-renaming	3 ahead 25 behind	Compare
no-inline-js-config	108 ahead 37 behind	Compare
svg-tests	2 ahead 45 behind	Compare
knyle-style-commits	40 ahead 73 behind	Compare
enterprise-non-config	7 ahead 64 behind	Compare
menu-behavior-act-i	5 ahead 150 behind	Compare
view-modes	36 ahead 209 behind	Compare

GitHub Flow



# GitHub Flow

Anything in the master branch is deployable.

## 1. Create branch

- *To work on something new, create a descriptively named branch off of master (ie: new-oauth2-scopes).*

## 2. Develop in branch

- *Commit to that branch locally and regularly push your work to the same named branch on the server.*

## 3. Open a pull request (ask for review)

- *When you need feedback or help, or you think the branch is ready for merging, open a pull request.*

## 4. Merge after review

- *After someone else has reviewed and signed off on the feature, you can merge it into master.*

## 5. Deploy

- *Once it is merged and pushed to master, you can and should deploy immediately.*

# GitHub Flow

```
# Check that origin and upstream repositories are correctly defined
$ git remote -v

# Get the latest sources from the upstream repository
$ git remote update

# Checkout a new topic branch for development
$ git checkout -b adding-new-feature upstream/master

#
# Do some development...
#

# Check your changes
$ git status

# Commit your changes
$ git commit -a -m "Added a new feature"

# Push your changes to the origin
$ git push origin HEAD
```

# VCS: Резюме

1. Системы контроля версий — центральный инструмент разработки

- *Навигация по истории изменений*
- *Централизованный доступ*

2. Распределенные СКВ фактически стали стандартом. Их сильные стороны:

- *Допускают локальные коммиты (без наличия интернет или доступа к серверу)*
- *Упрощают слияние (а значит параллельную разработку)*
- *Дают максимальную свободу по организации рабочего процесса (workflow)*

3. Git не самая простая в освоении СКВ, однако очень функциональная, к тому же дает максимальную свободу по организации процесса разработки.



# googletest

Google C++ Testing Framework

# Вся правда о ручном тестировании

## ■ Ключевые термины

- *Тест* — проверка, осуществляемая "руками"
- *Тест-план* — документ со списком проверок
- *Отдел тестирования*

## ■ Профессия ручного тестировщика умирает!

- *Программисты несут ответственность за качество (пишут тесты!)*
- *Google: Software Engineer in Test*

## ■ Ручное тестирование все еще используется для:

- *Тестирования GUI и UX (удобства использования)*
- *Бета-тестирование с реальными пользователями*

# Автоматические тесты

- Тест — это "обычная" функция, реализующая некоторый сценарий использования программных сущностей.

```
#include <gtest/gtest.h>

TEST(TBitField, can_set_bit)
{
    TBitField bf(10);
    EXPECT_EQ(0, bf.GetBit(3));

    bf.SetBit(3);
    EXPECT_NE(0, bf.GetBit(3));
}
```

- Тестовая сборка (test suite) — приложение с тестами (обычно консольное).

Тест пишется один раз, а запускается десятки тысяч раз!

# Пример тестов на Java с использованием JUnit

```
@Test
public void canAddNumbers()
{
    // Arrange
    ComplexNumber z1 = new ComplexNumber(1, 2);
    ComplexNumber z2 = new ComplexNumber(3, 4);

    // Act
    ComplexNumber sum = z1.add(z2);

    // Assert
    assertEquals(new ComplexNumber(4, 6), sum);
}

@Test
public void canMultiplyNumbers()
{
    // Arrange
    ComplexNumber z1 = new ComplexNumber(1, 2);
    ComplexNumber z2 = new ComplexNumber(3, 4);

    // Act
    ComplexNumber mult = z1.multiply(z2);

    // Assert
    assertEquals(new ComplexNumber(-5, 10), mult);
}
```

# Фреймворки для Unit-тестирования

Значительно упрощают создание и запуск unit-тестов, позволяют придерживаться единого стиля.

1. xUnit — общее обозначение для подобных фреймворков.

2. Бесплатно доступны для большинства языков:

- *C/C++: CUnit, CPPUnit, GoogleTest*
- *Java: JUnit*
- *.NET: NUnit*

3. Встроены в современные языки:

- *D, Python, Go*



# Типичные возможности

## 1. Удобное добавление тестов

- *Простая регистрация новых тестов*
- *Набор функций-проверок (assert)*
- *Общие инициализации и деинициализации*

## 2. Удобный запуск тестов

- *Пакетный режим*
- *Возможность фильтрации тестов по именам*

## 3. Часто допускают интеграцию с IDE

## 4. Генерация отчета в стандартном XML-формате

- *Возможность последующего автоматического анализа*
- *Публикация на web-страницах проекта*

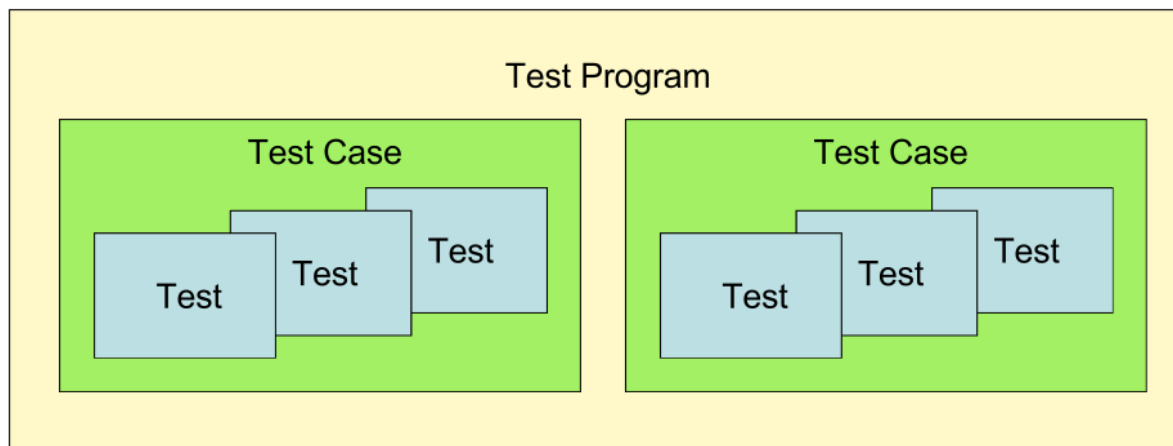
# Google Test

1. Популярный фреймворк для написания модульных тестов на C++, разработанный Google.
2. [Open-source](#) проект с BSD-лицензией  
(допускает использование в закрытых коммерческих проектах).
3. Используется в целом ряде крупных проектов
  - *Chromium, LLVM компилятор, OpenCV*
4. Написан на C++, строится при помощи CMake
  - *Поддерживает: Linux, Mac OS X, Windows, Cygwin, Windows CE, и Symbian*
5. Как правило используется в консольном режиме, но существует вспомогательное GUI [приложение](#).

## Возможности Google Test

- Automatic test discovery
- Rich set of assertions, user-defined assertions
- Death tests
- Fatal and non-fatal failures
- Value- and type-parameterized tests
- Various options for running the tests
- XML test report generation

## Базовые концепции



- Каждый тест реализован как функция, с использованием макроса `TEST()` или `TEST_F()`.
- `TEST()` не только определяет, но и "регистрирует" тест.

# Пример 1

```
#include <gtest/gtest.h>

TEST(MathTest, TwoPlusTwoEqualsFour) {
    EXPECT_EQ(2 + 2, 4);
}
```

## Пример 2

### Функция

```
int Factorial(int n); // Returns the factorial of n
```

### Тесты

```
// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
    EXPECT_EQ(1, Factorial(0));
}

// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(2, Factorial(2));
    EXPECT_EQ(6, Factorial(3));
    EXPECT_EQ(40320, Factorial(8));
}
```

## Пример 3

```
#include <gtest/gtest.h>
#include <vector>

using namespace std;

// A new one of these is created for each test
class VectorTest : public testing::Test {
public:
    vector<int> m_vector;

    virtual void SetUp() {
        m_vector.push_back(1);
        m_vector.push_back(2);
    }

    virtual void TearDown() {}
};

TEST_F(VectorTest, testElementZeroIsOne) {
    EXPECT_EQ(m_vector[0], 1);
}

TEST_F(VectorTest, testElementOneIsTwo) {
    EXPECT_EQ(m_vector[1], 2);
}

TEST_F(VectorTest, testSizeIsTwo) {
    EXPECT_EQ(m_vector.size(), (unsigned int)2);
}
```

# Консольный лог Google Test

```
[mlong@n6-ws2 x86]$ bin/hellotest
Running main() from gtest_main.cc
[=====] Running 4 tests from 2 test cases.
[-----] Global test environment set-up.
[-----] 3 tests from VectorTest
[ RUN      ] VectorTest.testElementZeroIsOne
[       OK ] VectorTest.testElementZeroIsOne (0 ms)
[ RUN      ] VectorTest.testElementOneIsTwo
[       OK ] VectorTest.testElementOneIsTwo (0 ms)
[ RUN      ] VectorTest.testSizeIsTwo
[       OK ] VectorTest.testSizeIsTwo (0 ms)
[-----] 3 tests from VectorTest (0 ms total)

[-----] 1 test from MathTest
[ RUN      ] MathTest.Zero
[       OK ] MathTest.Zero (0 ms)
[-----] 1 test from MathTest (0 ms total)

[-----] Global test environment tear-down
[=====] 4 tests from 2 test cases ran. (0 ms total)
[ PASSED  ] 4 tests.
```



# Полезные советы

Тесты можно временно выключать

```
TEST(MathTest, DISABLED_two_plus_two_equals_four)
{
    int x = 2 + 2;
    EXPECT_EQ(4, x);
}
```

Тесты можно фильтровать по имени при запуске

```
$ ./bin/hellotest --gtest_filter=*Vector*
```

У Google Test есть ряд других полезных опций

```
$ ./bin/hellotest --help
```

## Юнит-тест курильщика

```
[Test]
public void TestMethod1()
{
    var calc = new Calculator();
    calc.ValidOperation = Calculator.Operation.Multiply;
    calc.ValidType = typeof(int);
    var result = calc.Multiply(-1, 3);
    Assert.AreEqual(result, -3);
    calc.ValidOperation = Calculator.Operation.Multiply;
    calc.ValidType = typeof(int);
    result = calc.Multiply(1, 3);
    Assert.IsTrue(result == 3);
    if (calc.ValidOperation == Calculator.Operation.Invalid)
    {
        throw new Exception("Operation should be valid");
    }
    calc.ValidOperation = Calculator.Operation.Multiply;
    calc.ValidType = typeof(int);
    result = calc.Multiply(10, 3);
    Assert.AreEqual(result, 30);
}
```

Авторство: Антон Бевзюк, SmartStepGroup.

## Юнит-тест здорового человека

```
[TestMethod]
public void CanAuthenticateUser() {
    var page = new TestableLoginPage();

    page.AuthenticateUser("user", "user");

    Assert.AreEqual("user", page.AuthenticatedUser);
}
```



## Критерии хорошего теста

1. Короткий (имеет чистый код)
2. Сфокусированный (только один assert)
3. Быстрый
4. Автоматический
5. Независим от порядка исполнения и окружения

Паттерн AAA: Arrange, Act, Assert

# Необходимость автоматических тестов

- Оптимизация производительности!
- Внесение изменений в уже отлаженные компоненты
- Коллективное владение
- Работа с унаследованным и сторонним кодом
- Портирование ПО на новые платформы
- Тестирование новых платформ

# Современная стратегия тестирования

- Без "зеленых" тестов нет уверенности в работоспособности кода
- Фокус на максимальную автоматизацию
  - *Полное тестирование требуется несколько раз в день, каждому члену команды*
- Тесты пишутся самими разработчиками, одновременно с реализацией
  - *Тесты это лучшая документация, которая всегда актуальна (компилятор!)*
  - *Тесты это первые сэмплы, показывающие простые примеры использования*
  - *Test-Driven Development*
- Код тестируется непрерывно
  - *Это делается локально на машине разработчика*
  - *Это делается на сервере до того, как добавить его в репозиторий*

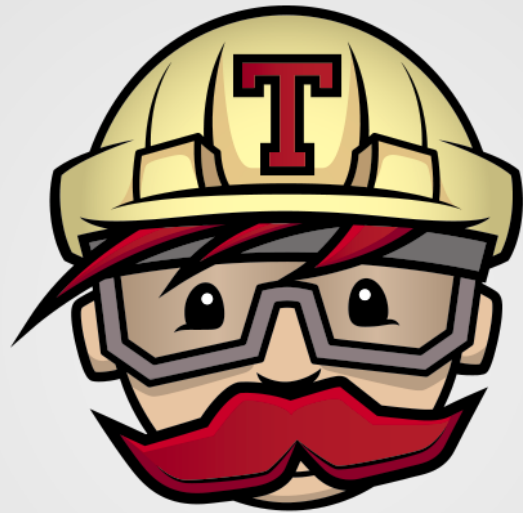
## Современная стратегия тестирования (2)

- Автоматические тесты замещают отладку
  - *Предсказуемость времени разработки*
  - *Пойманный баг документируется в виде теста*
- Тесты — это "first-class citizens"
  - *Стоит отдавать код вместе с тестами*
  - *Нужно заботиться о качестве кода тестов*
  - *Метафора тестов: скелет, позволяющий организму двигаться*

# Google Test: Резюме

1. Пишите "жесткие" тесты, старайтесь сломать свой код.
2. Создание тестов — это составляющая самого процесса программирования.
  - *Почитайте про Test-Driven Development*
3. Без тестов нет уверенности в работоспособности кода.
4. Весь продуктовый код должен быть покрыт автоматическими тестами.
5. Автоматические тесты должны прогоняться при каждом изменении кода.
6. Ежедневно должно проводиться полное тестирование проекта, желательно с публикацией тестовых дистрибутивов (nightly builds).



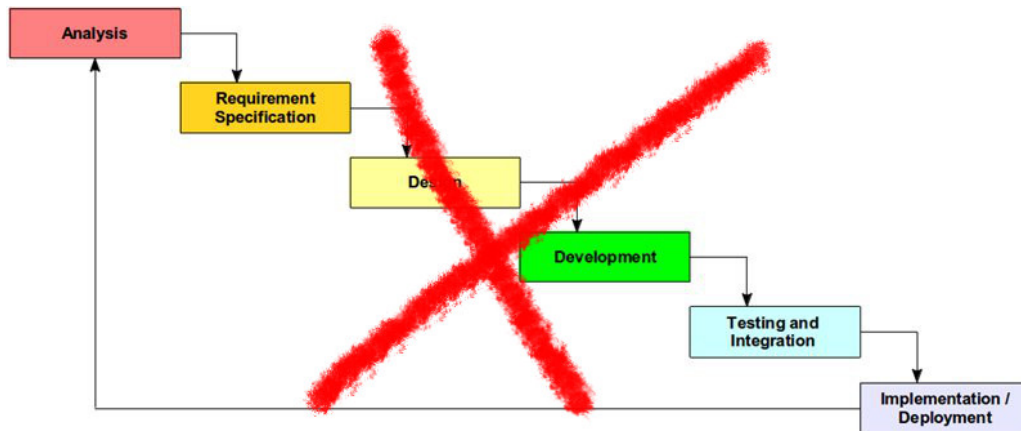


**TRAVIS**

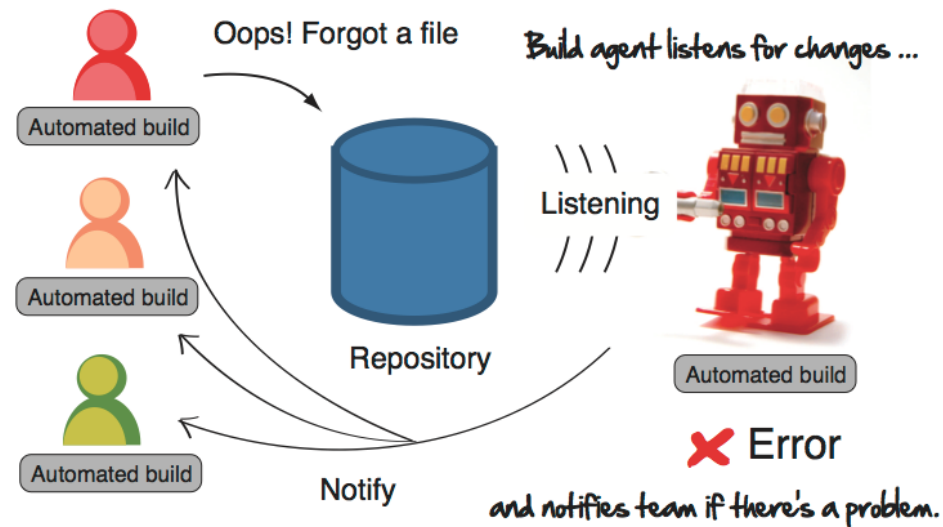
# Непрерывная интеграция

*Непрерывная интеграция (англ. Continuous Integration) – это практика разработки ПО, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем.*

Непрерывная сборка — это сердцебиение вашего проекта.



# Практика непрерывной интеграции



1. Все хранится в **централизованном репозитории**:  
исходный код, конфигурационные файлы, тестовые данные, сборочные и тестовые скрипты
2. **Полная автоматизация** операций с кодом:  
выкачивание из репозитория, сборка, тестирование и так далее.

# Travis CI



- Официальный сайт проекта: <http://travis-ci.org>
- Веб-сервис для сборки и тестирования ПО ([open-source](#))
- Важными особенностями являются интеграция с GitHub и возможность бесплатного использования
- Поддерживает большое количество языков: C, C++, Clojure, Erlang, Go, Groovy, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby и Scala
- Тестирование происходит на виртуальных Linux-машинах, запускаемых в облаке Amazon

# GitHub + Travis CI

Conversation 5 | Commits 7 | Files changed 20

Commits on Mar 7, 2015

- Added lab3**  
Nikolay-Borisov authored on 7 Mar ✖
- copy/paste fixes**  
Nikolay-Borisov authored on 7 Mar ✖
- copy/paste fixes 2**  
Nikolay-Borisov authored on 7 Mar ✖
- copy/paste fixes 3**  
Nikolay-Borisov authored on 7 Mar ✔
- added ViewModelWithTxtLoggerTests.java**  
Nikolay-Borisov authored on 7 Mar ✖
- super commit**  
Nikolay-Borisov authored on 7 Mar ✔

Commits on Mar 9, 2015

- reformat code**  
Nikolay-Borisov authored on 9 Mar ✔

## UNN-VMK-Software/agile-course-practice build passing

Current | Branches | Build History | Pull Requests

✔ 🔗	PR #176 Борисов - Лабораторная работа #3 Borisov-Nikolay committed	# 1477 passed 28f5291
✔ 🔗	PR #176 Борисов - Лабораторная работа #3 Borisov-Nikolay committed	# 1476 passed 8a5db86
✖ 🔗	PR #176 Борисов - Лабораторная работа #3 Borisov-Nikolay committed	# 1475 failed 928f34d
✔ 🔗	PR #176 Борисов - Лабораторная работа #3 Borisov-Nikolay committed	# 1474 passed 6775790
✖ 🔗	PR #176 Борисов - Лабораторная работа #3 Borisov-Nikolay committed	# 1473 failed 3356825
✖ 🔗	PR #176 Борисов - Лабораторная работа #3 Borisov-Nikolay committed	# 1472 failed 917a45d

# Современная стратегия тестирования

- Без "зеленых" тестов нет уверенности в работоспособности кода
- Фокус на максимальную автоматизацию
  - *Полное тестирование требуется несколько раз в день, каждому члену команды*
- Тесты пишутся самими разработчиками, одновременно с реализацией
  - *Тесты это лучшая документация, которая всегда актуальна (компилятор!)*
  - *Тесты это первые сэмплы, показывающие простые примеры использования*
  - *Test-Driven Development*
- Код тестируется непрерывно
  - *Это делается локально на машине разработчика*
  - *Это делается на сервере до того, как добавить его в репозиторий*

## Современная стратегия тестирования (2)

- Автоматические тесты замещают отладку
  - *Предсказуемость времени разработки*
  - *Пойманный баг документируется в виде теста*
- Тесты — это "first-class citizens"
  - *Стоит отдавать код вместе с тестами*
  - *Нужно заботиться о качестве кода тестов*
  - *Метафора тестов: скелет, позволяющий организму двигаться*

# Резюме

В рамках нашей школы будут выполняться *учебные* задания, однако они будут носить все черты *промышленной* разработки ПО.

## ■ Кросс-платформенность

- Построение с использованием *CMake*
- Тестирование на *Linux* (*gcc*, *clang*)
- Потенциальная переносимость на *Android* и *iOS*

## ■ Коллективная разработка

- Использование *Git*
- *Peer code review* от преподавателей и коллег средствами *GitHub*

## ■ Тестирование

- Автоматические тесты на базе *Google Test*
- Непрерывная интеграция на основе *Travis-CI*



## Ссылки

1. Wikipedia "[Системы контроля версий](#)".
2. [Pro Git](#) by Scott Chacon.
3. "[Mercurial tutorial](#)" by Joel Spolsky.
4. [GTest](#)
5. [Google Test Talk](#)

# Спасибо!

Вопросы?