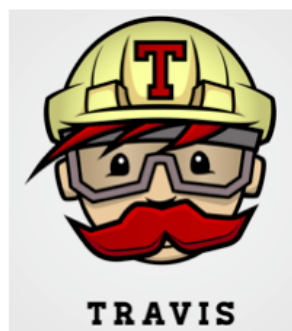


# Инструменты разработки ПО



Кирилл Корняков  
Директор по исследованиям и разработке, Itseez  
Июль 2016

# Содержание

1. CMake: кросс-платформенная разработка
2. Git: системы контроля версий
3. GitHub: коллективная разработка



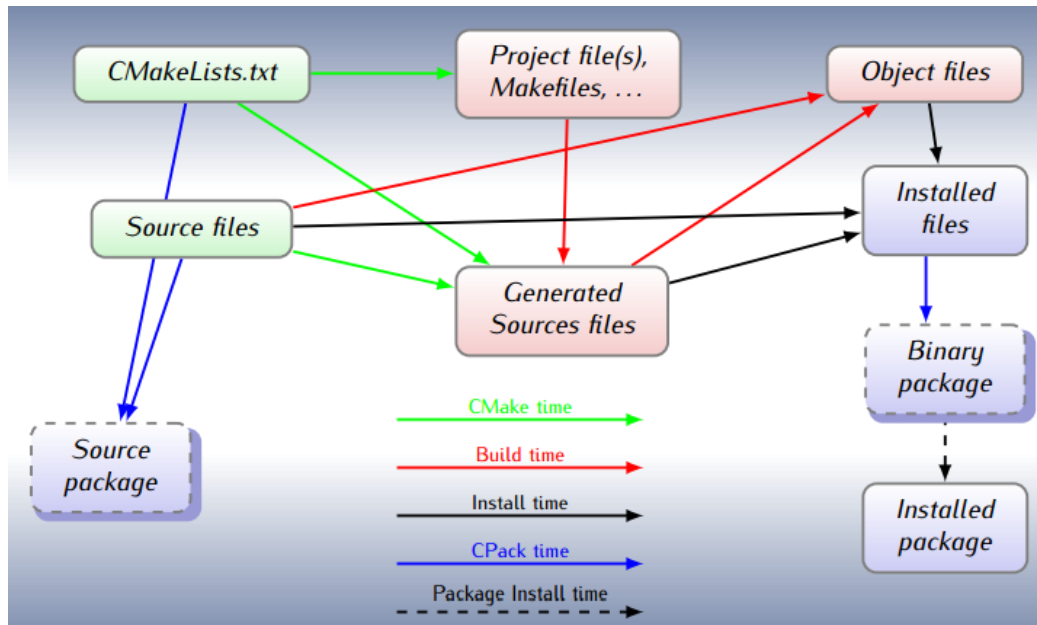
***CMake***  

---

*Cross-platform Make*

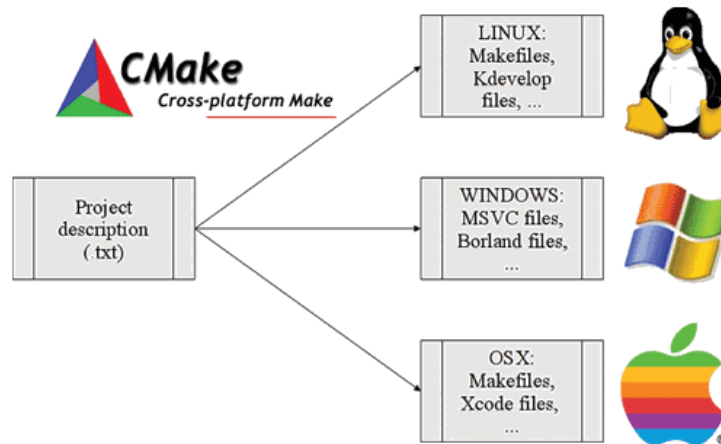
# CMake Workflow

CMakeLists.txt — файл, описывающий порядок сборки приложения



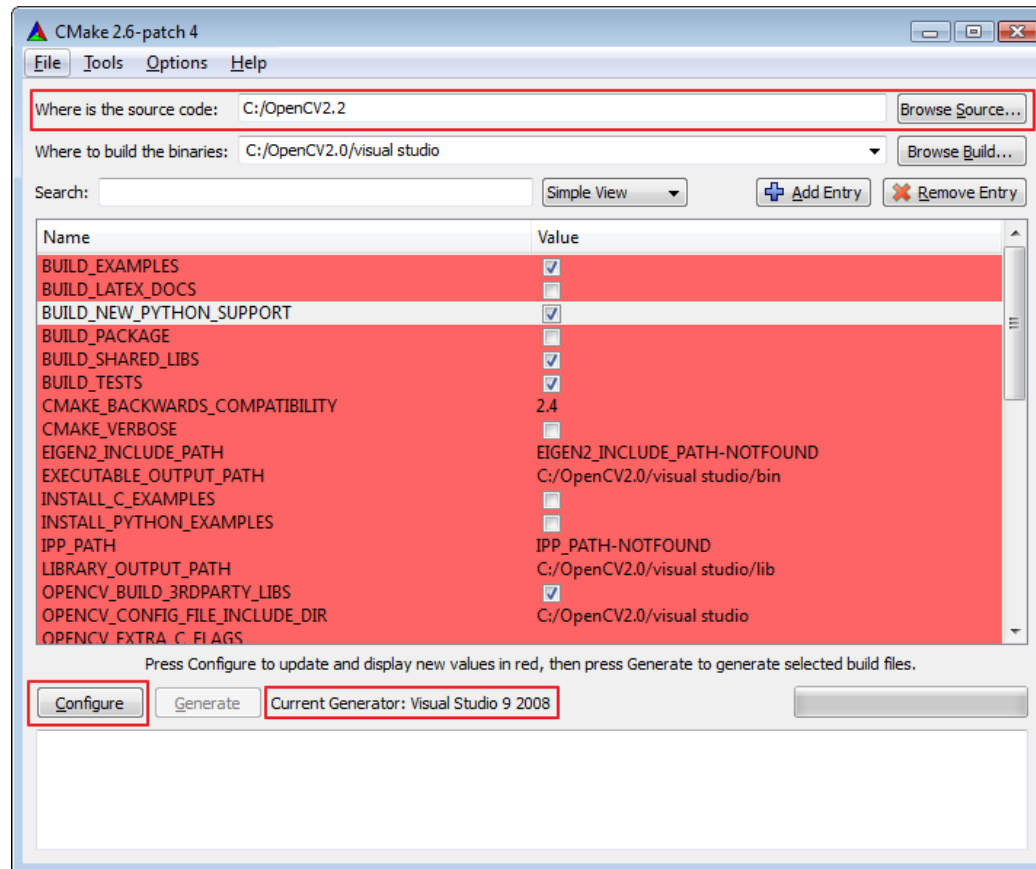
- Шаг 0. Генерация *проектных файлов* при помощи cmake или CMakeGui
  - .vcproj, Makefile, etc
- Шаг 1. Компиляция исходников при помощи компиляторов из Visual Studio, Qt Creator, Eclipse, XCode...
  - .obj, .o
- Шаг 2. Линковка финальных бинарных файлов компоновщиком (link.exe, ld, ...)
  - .exe, .dll,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

# CMake



- Функционирует на большинстве популярных ОС
- Генерирует проекты сборки для большого числа IDE
- Максимальная свобода в выборе окружения разработки (в рамках одной команды!)
- В настоящий момент является стандартом де-факто для C++ проектов

# 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)

# Объявляет исполняемый модуль с именем sample_app
add_executable(sample_app main.c lib.c)
```

# 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-release
└─ hello.exe
build-debug
└─ hello.exe
```

Соответствующие команды:

```
$ mkdir ../build-release && cd ../build-release
$ 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)

# Объявляет статическую библиотеку с именем library
add_library(library STATIC lib.c)

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

# Добавление подпроекта

Содержимое каталога:

```
code
├─ CMakeLists.txt
├─ library
│   └─ CMakeLists.txt
│       ├── lib1.c
│       ├── lib2.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)
add_executable(sample_app main.c)

target_link_libraries(sample_app library)
```

library/CMakeLists.txt

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

set(LIB_SOURCES lib1.c lib2.c)
add_library(library STATIC ${LIB_SOURCES})
```

# Поиск зависимостей

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(fourth_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_SOURCES})  
target_link_libraries(libd DEBUG ${LIB_SOURCES})
```

# CMake: 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



**git**

# 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.*

OpenCV 2.4.0



- <http://www.youtube.com/watch?v=ToD91PYaQOU>
- Сделано при помощи [gource](#)



# Коллективная работа с кодом

## 1. Хранение истории изменений

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

## 2. Организация распределенной работы

- *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.

# Патчи

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

**Unified diff format:** @@ -1,s +1,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

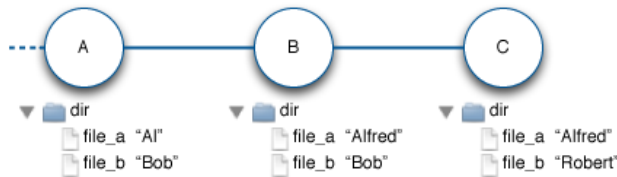
    Switched to factory

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 использована
    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-
+   `filters_factory.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
```



# Патчи



- Патч — это **атомарное изменение** проекта!
- Последовательность патчей — это полная история проекта.
- 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

# 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

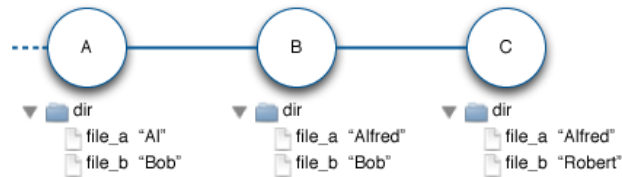
- include/filters.hpp
- samples/matrix\_sample.cpp
- src/filters\_fabrics.cpp
- test/filters\_test.cpp

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	 b39fba0	
	<b>Description bug fixes.</b> valentina-kustikova authored 24 days ago	 71db5e6	



# Патчи и СКВ



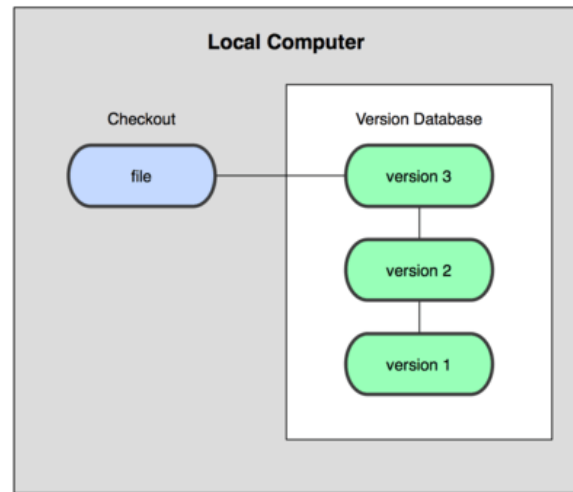
- СКВ — это своего рода БД патчей, ее называют **репозиторием**
- Патчи, помещенные в СКВ называются **commit**
- Последовательности **commit** называются **changeset**

# 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

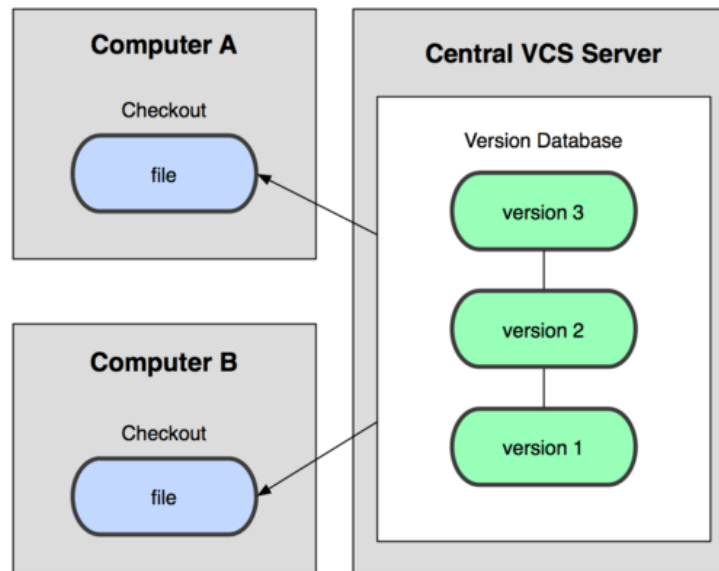
Eric Sink ["A History of Version Control"](#)

# Три поколения СКВ: Локальные



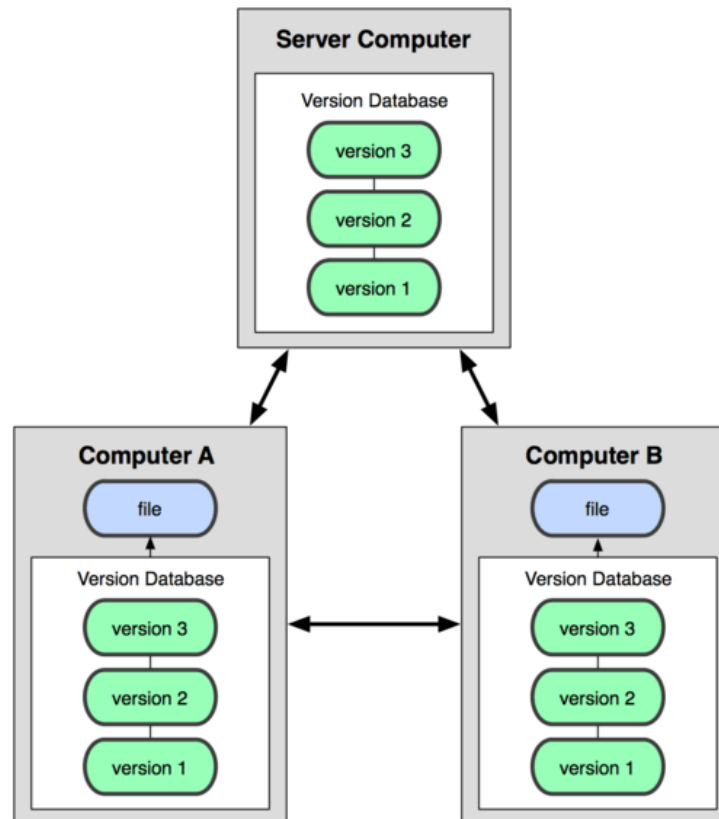
Примеры: RCS, SCCS

## Три поколения СКВ: Централизованные



Примеры: Subversion, CVS

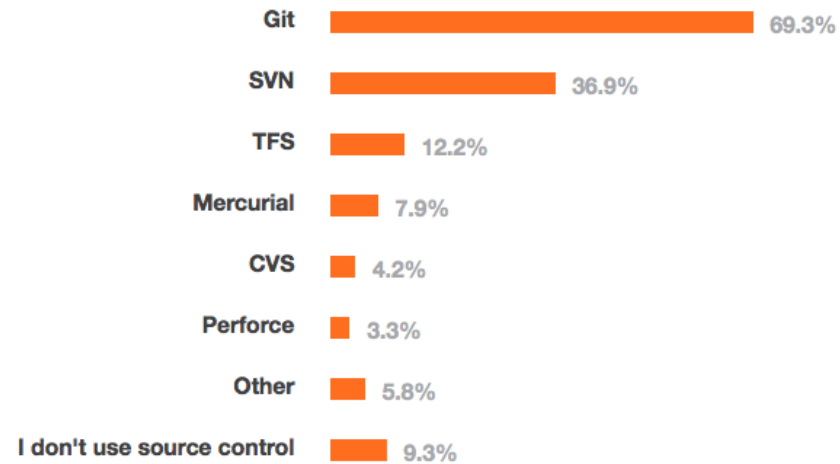
# Три поколения СКВ: Распределенные



- 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.

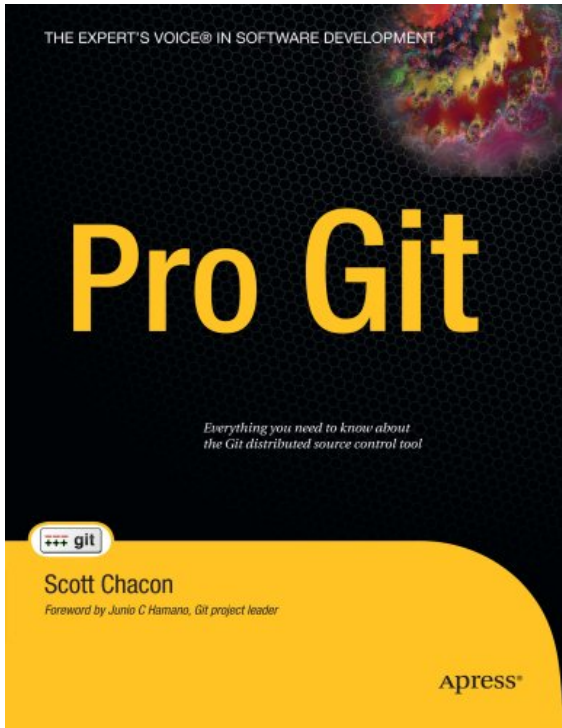
# Git



- 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>.



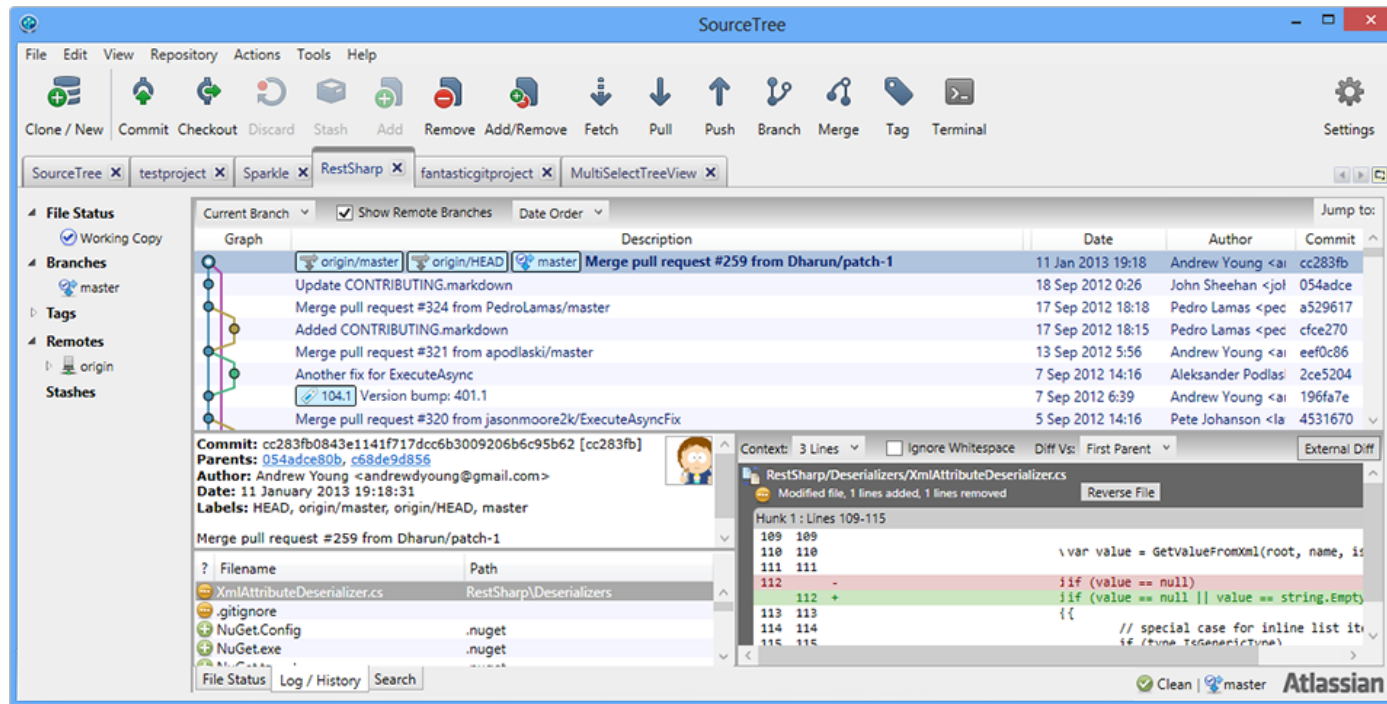
# Pro Git



- 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

The screenshot shows the TortoiseGit Log Messages window for the repository located at `E:\dev\libs-nosync\1\Cinder`. The window displays a list of commit messages, their authors, and dates. The selected commit is `dev origin/dev Merge branch 'appRewrite' into dev` by Andrew Bell, dated 12/03/2013 00:43:58. Below the log, the SHA-1 hash `746dcefbeddd8a7aa8f339465e2e1c415dada728` is shown, along with the command `* Merge branch 'appRewrite' into dev`. The bottom section displays a diff view comparing the current revision (1847) with its parent (1fdb3ba). The diff shows changes to various files, including `.gitignore`, `.gitmodules`, `README.md`, and several files in the `blocks/Box2D/` directory. The window also includes a search bar, a filter dropdown, and a 'Refresh' button.

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/b2Cirde...	.cpp	Added	1..	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Cirde...	.h	Added	91	0
blocks/Box2D/src/Box2D/Collision/Shapes/b2Edge...	.cpp	Added	1..	0
blocks/Box2D/src/Box2D/Collision/Shapes/h2Edge...	.h	Added	74	0

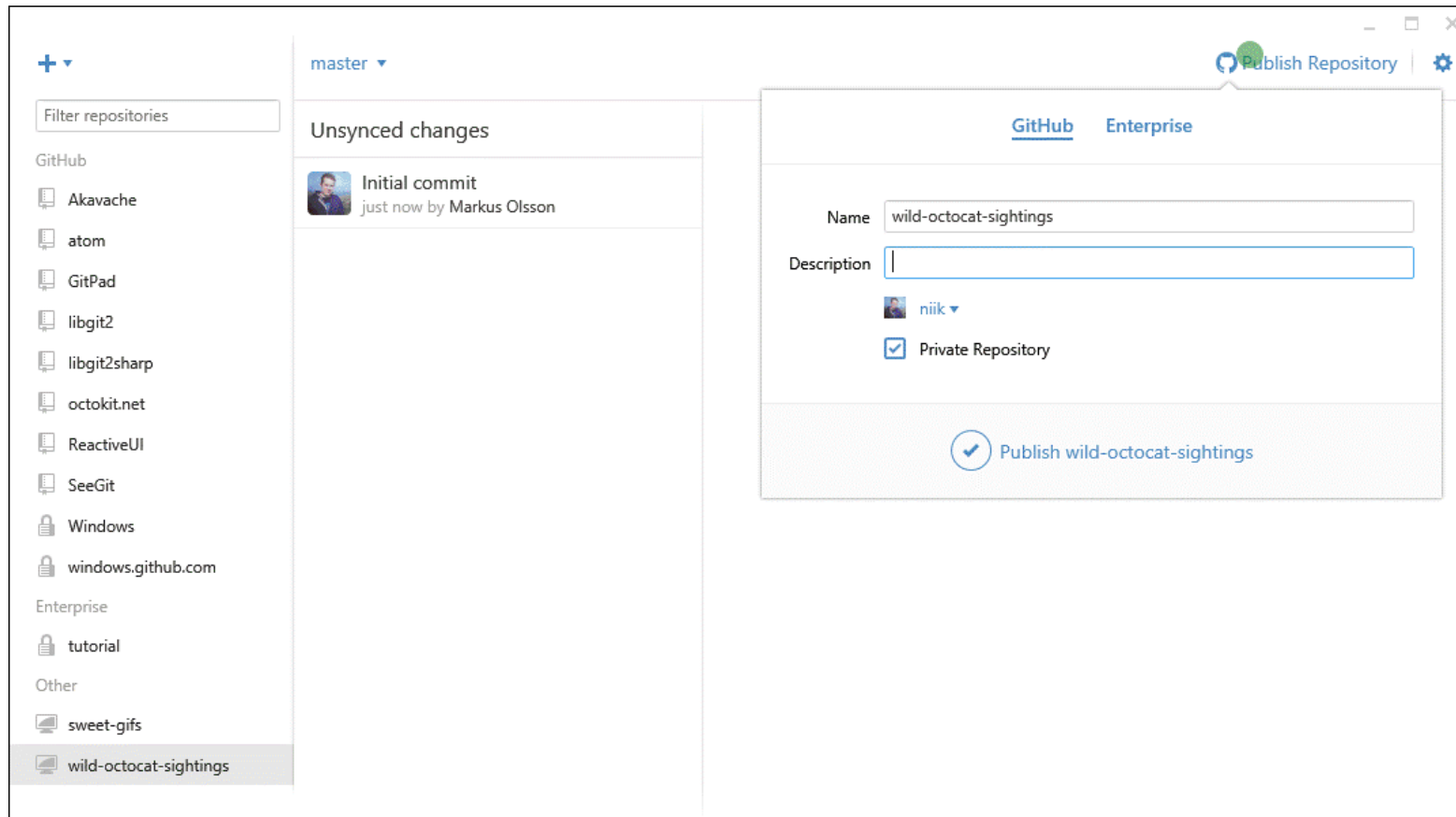
Showing 1847 revision(s), from revision 1fdb3ba to revision 746dcef - 1 revision(s) selected

☒ Hide Unrelated Changed Paths ☐ Show Whole Project  
☐ All Branches ☐ Follow renames  
☐ First Parent ☒ Show tags

Refresh Statistics Help OK



# GitHub Desktop



# Command Line Interface!

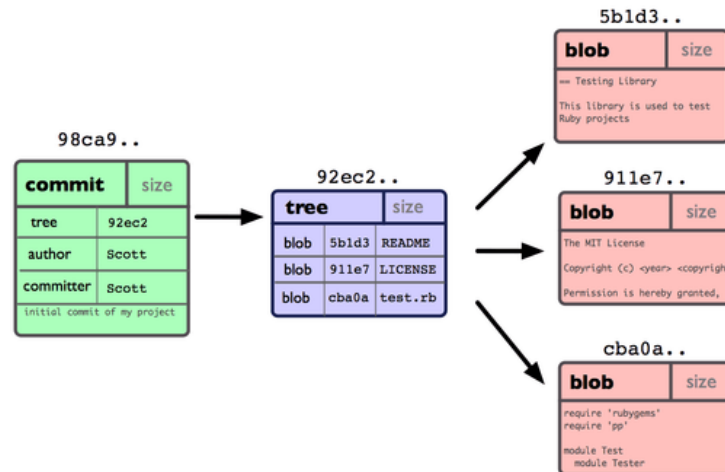
```
~> git help
usage: git [--version] [--help] [-C <path>] [-c name=value]
         [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
         [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
         [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
         <command> [<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 представление патча внутри Git
- Пользователю приходится работать только с коммитами (слава богу!)

Показать содержимое коммита:

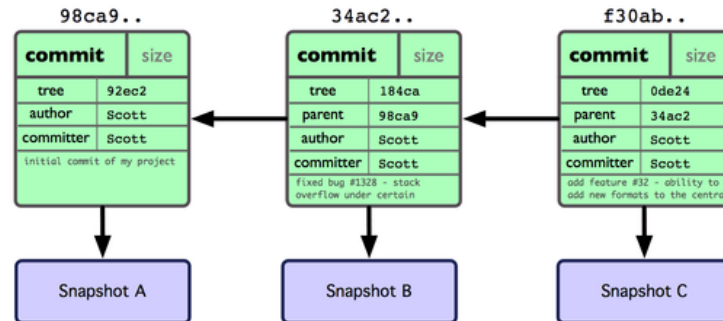
```
$ git show --raw dc2ca9d95c

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_fabricts.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).

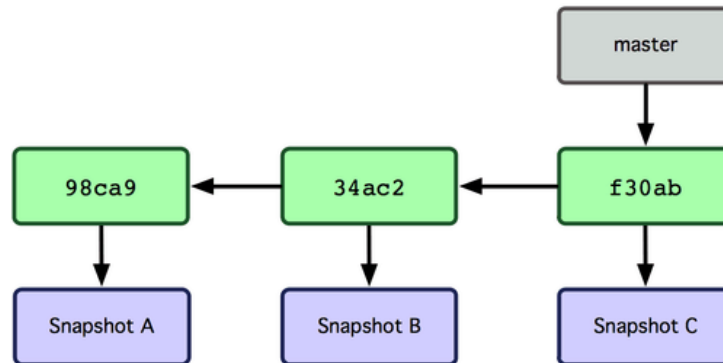
commit 3c1d15a1bf366864593f2320fa9a0e6cf3586f52
Author: valentina-kustikova <valentina.kustikova@gmail.com>
Date: Thu Aug 13 10:08:59 2015 +0300

    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.



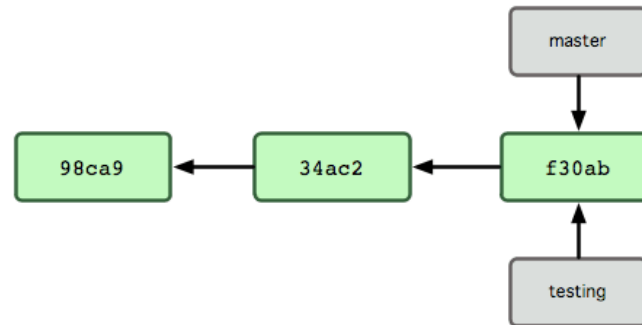
Вывести список существующих веток:

```
$ git branch
* master
```

# Git branch

Создать новую ветку с именем testing:

```
$ git branch testing
```

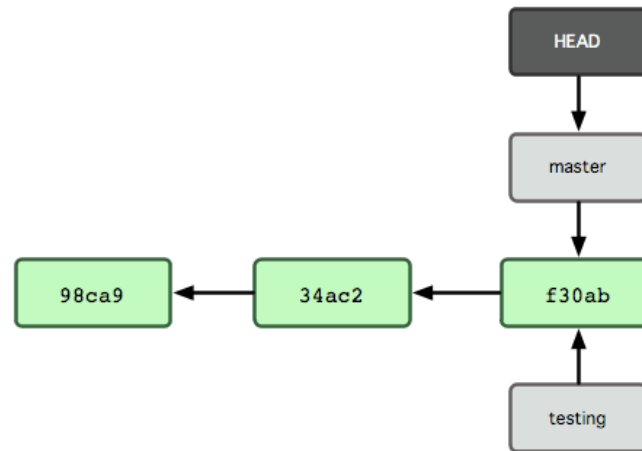


Текущий список веток:

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

# HEAD

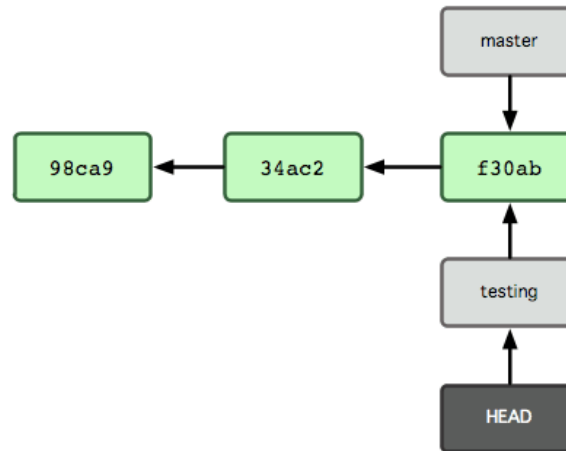
- HEAD — специальный указатель, ссылающийся на локальную ветку, на которой вы находитесь.
- Это просто алиас для текущей ветки, введенный для удобства.



# Git checkout

Извлечь состояние репозитория, соответствующее ветке `testing`:

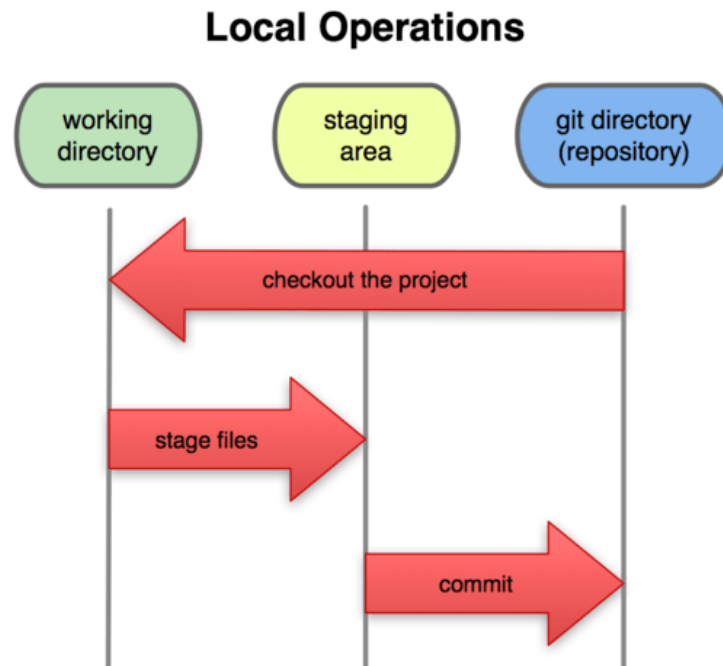
```
$ git checkout testing
```



Вывести список существующих веток:

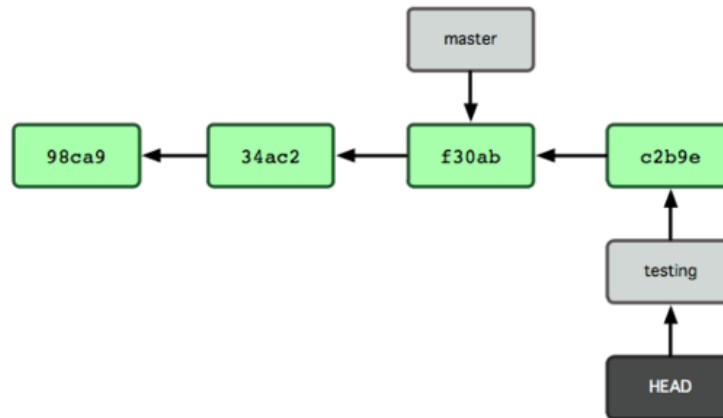
```
$ git branch
master
* testing
```

# Три состояния файлов



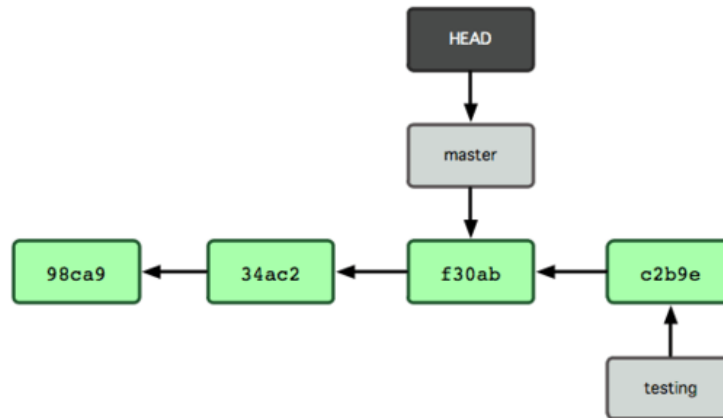
# Git commit

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



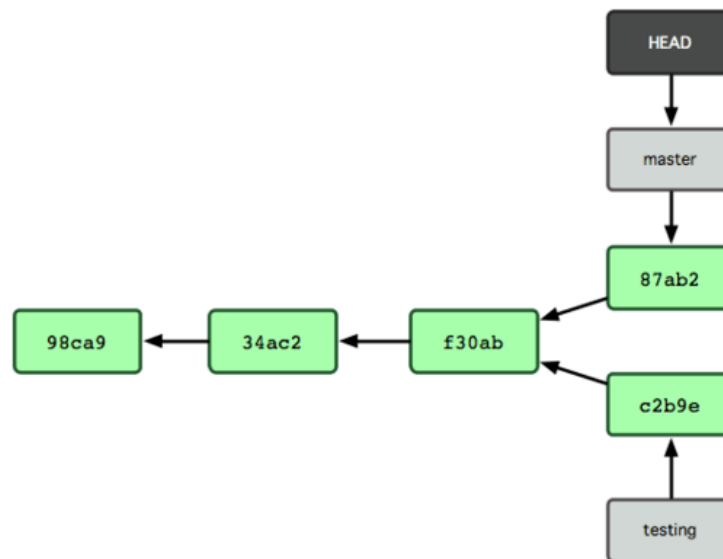
# Go back to master

```
$ git checkout master
```



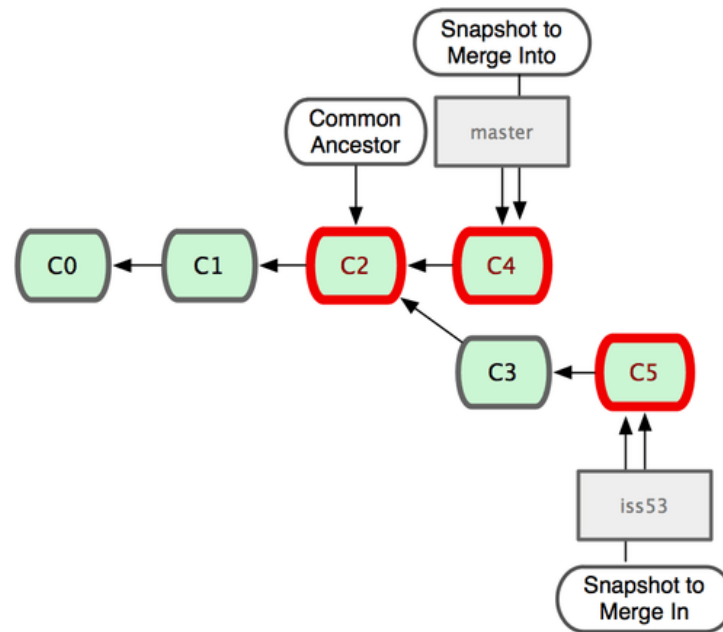
# Make a commit to master

```
$ 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'
```

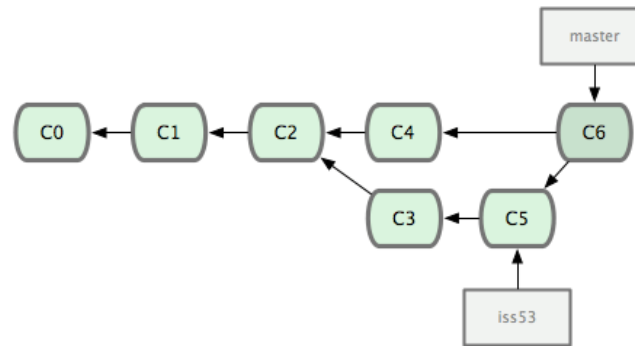




# Merging

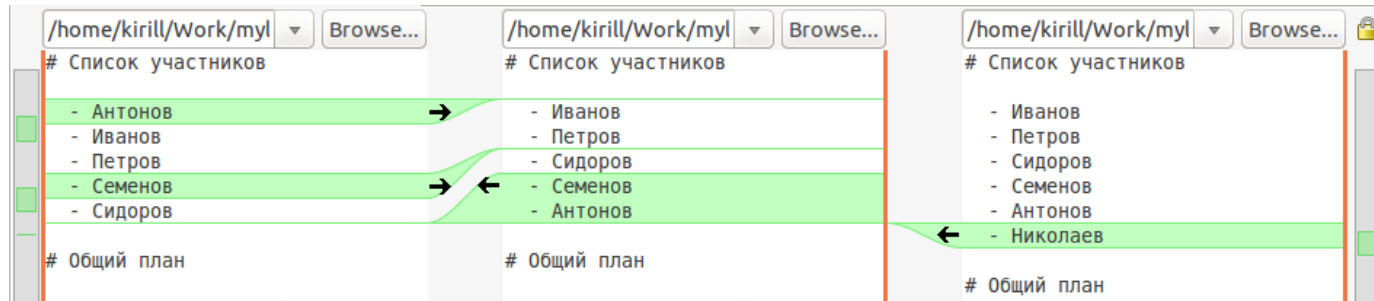


# Merging



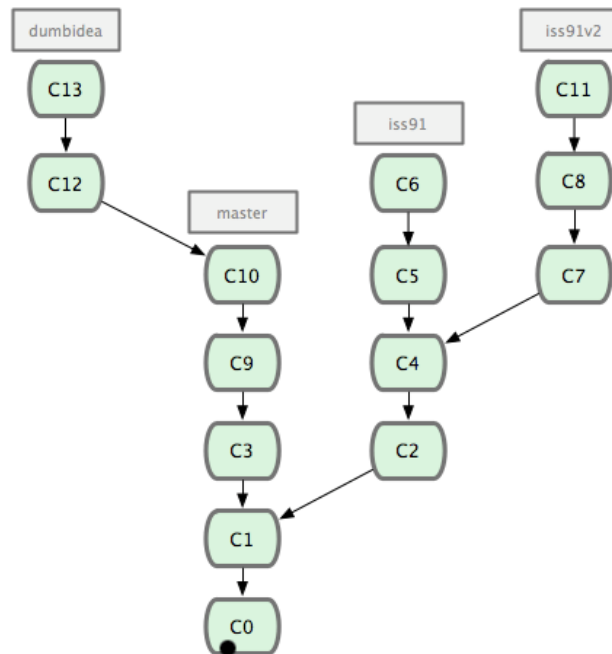
- C6 — это так называемый *merge commit*
- Он основан не на каком-то патче, он указывает на состояние проекта, в котором наложены патчи обеих ветвей (*master* и *testing*).

# 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.

# VCS: Резюме

1. Системы контроля версий — центральный инструмент разработки

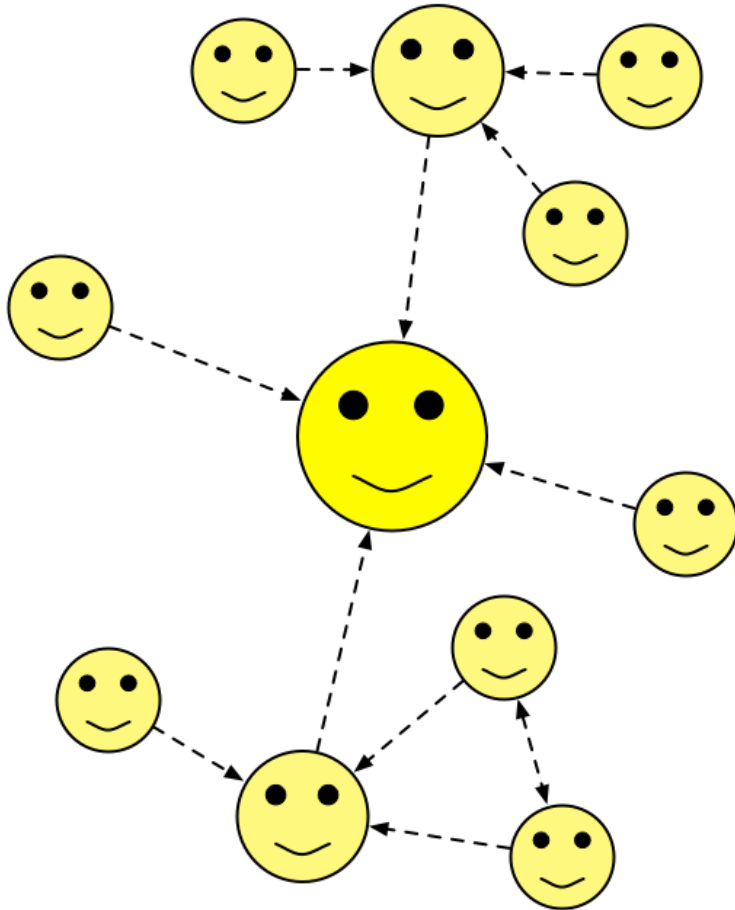
- *Навигация по истории изменений*
- *Централизованный доступ*

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 в плане организации процесса разработки.

# Распределенная работа



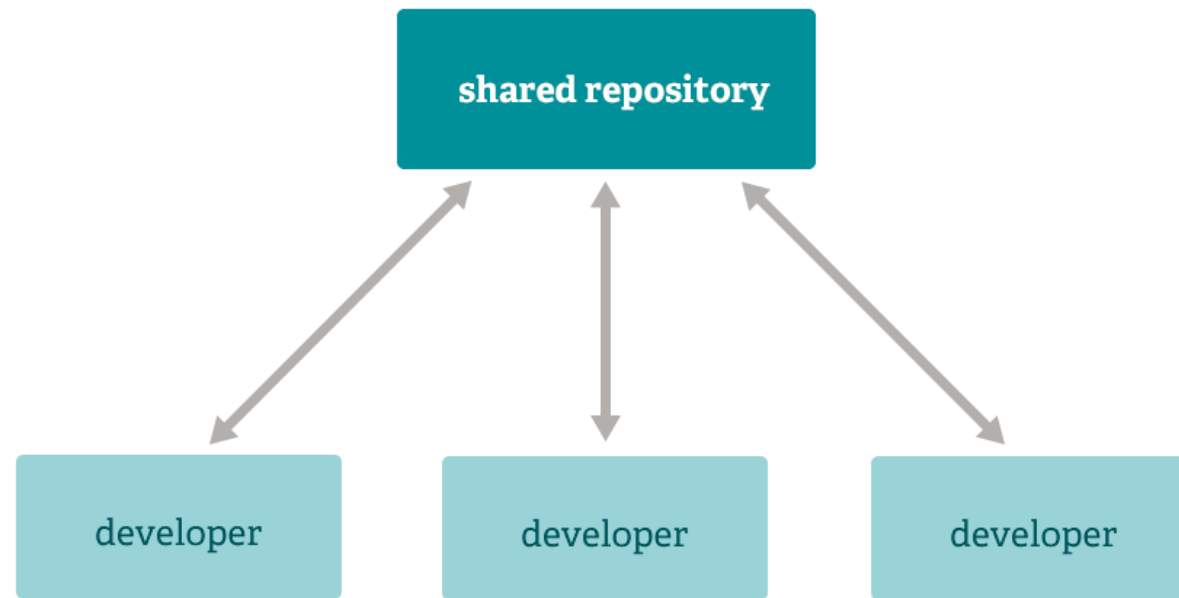
## 1. Распределенные рабочие процессы (workflow)

- *Centralized*
- *Integration Manager*
- *Dictator and Lieutenants*

## 2. Модели ветвления (branching model)

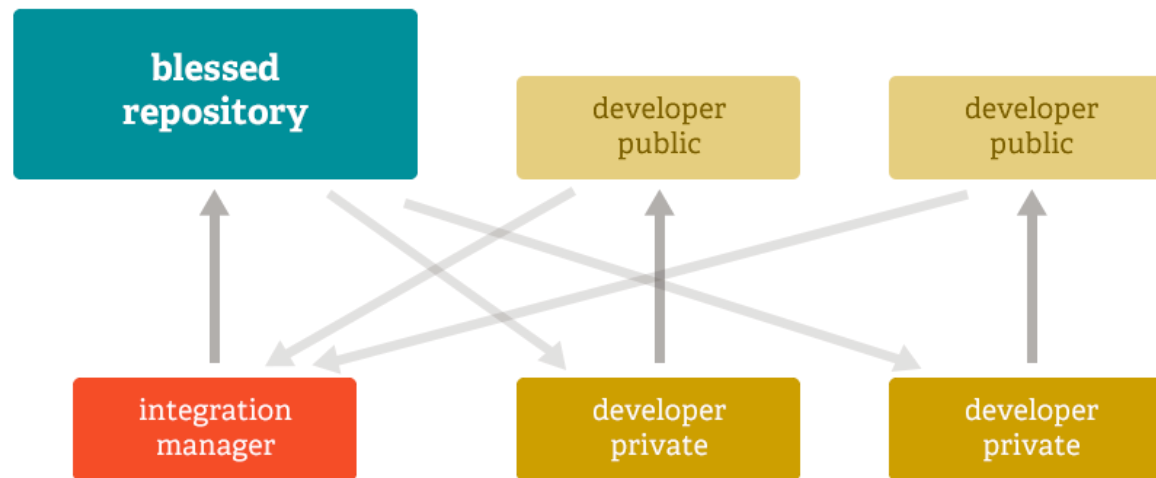
- *GitFlow*
- *GitHub Flow*

# Centralized Workflow



Плюсы и минусы данного подхода?

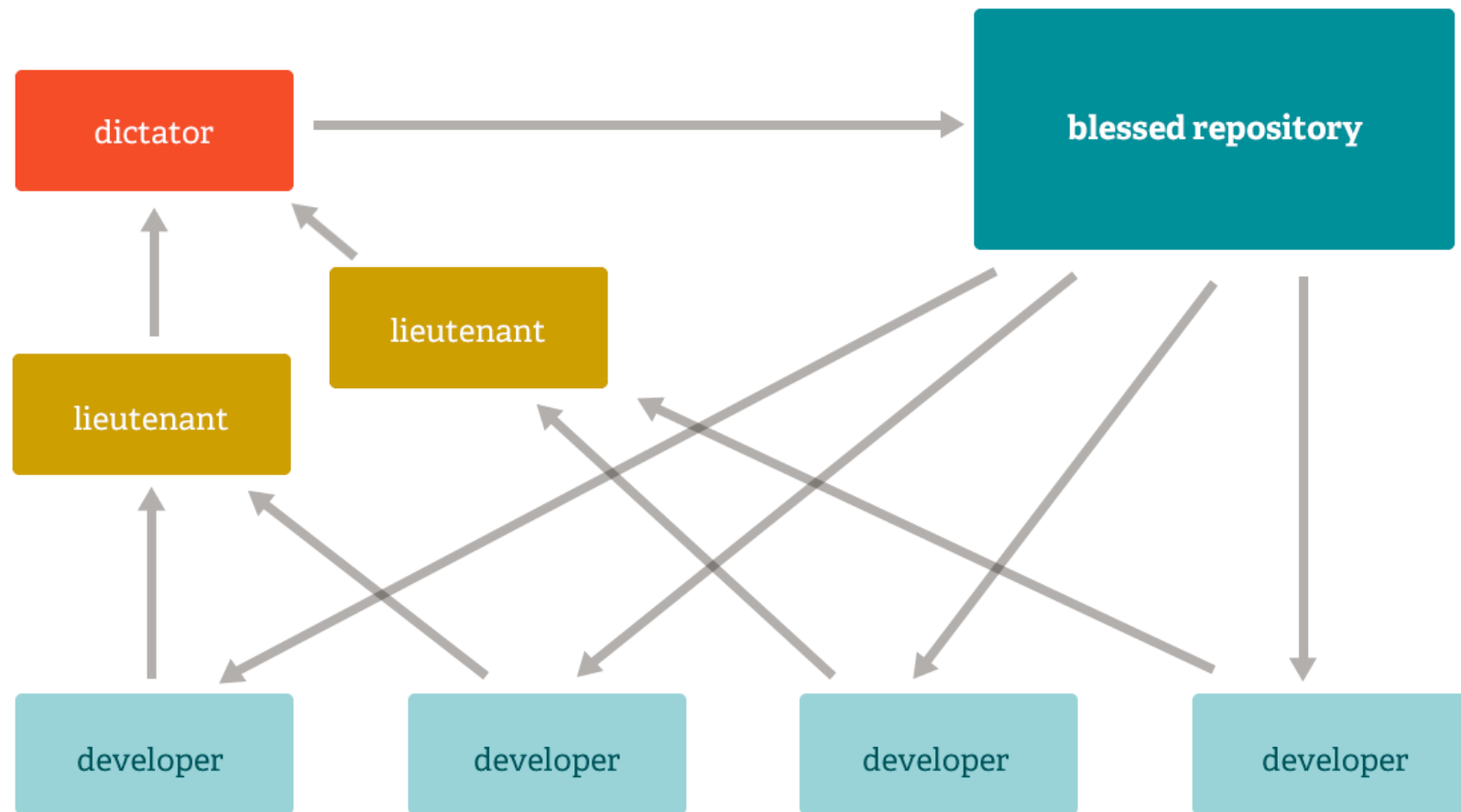
# Integration Manager Workflow



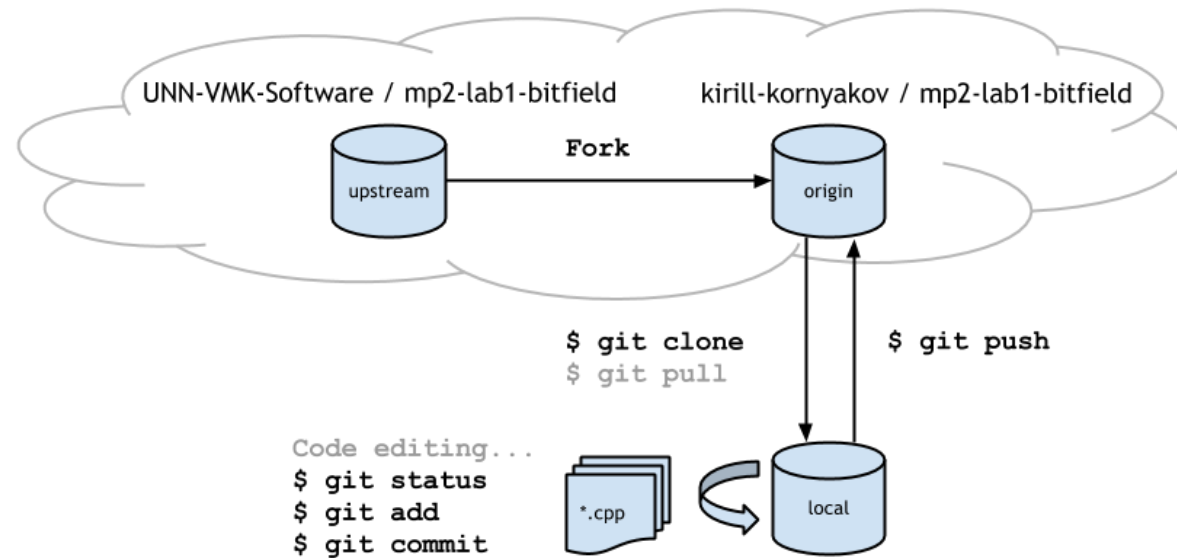
Плюсы и минусы данного подхода?



# 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

master	Base branch
fi-signup Last updated 36 minutes ago by sr	3 ahead 0 behind Compare
charlock-linguist Last updated about 13 hours ago by josh	11 ahead 7 behind Compare
git-http-server Last updated about 14 hours ago by rromayko	11 ahead 7 behind Compare
wild-renaming Last updated about 20 hours ago by defunkt	3 ahead 25 behind Compare
no-inline-js-config Last updated 1 day ago by josh	108 ahead 37 behind Compare
svg-tests Last updated 1 day ago by jsncostello	2 ahead 45 behind Compare
knyle-style-commits Last updated 1 day ago by kneath	40 ahead 73 behind Compare
enterprise-non-config Last updated 2 days ago by rromayko	7 ahead 64 behind Compare
menu-behavior-act-i Last updated 4 days ago by josh	5 ahead 150 behind Compare
view-modes Last updated 5 days ago by kneath	36 ahead 259 behind Compare

[GitHub Flow][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 в командах Git

```
# 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

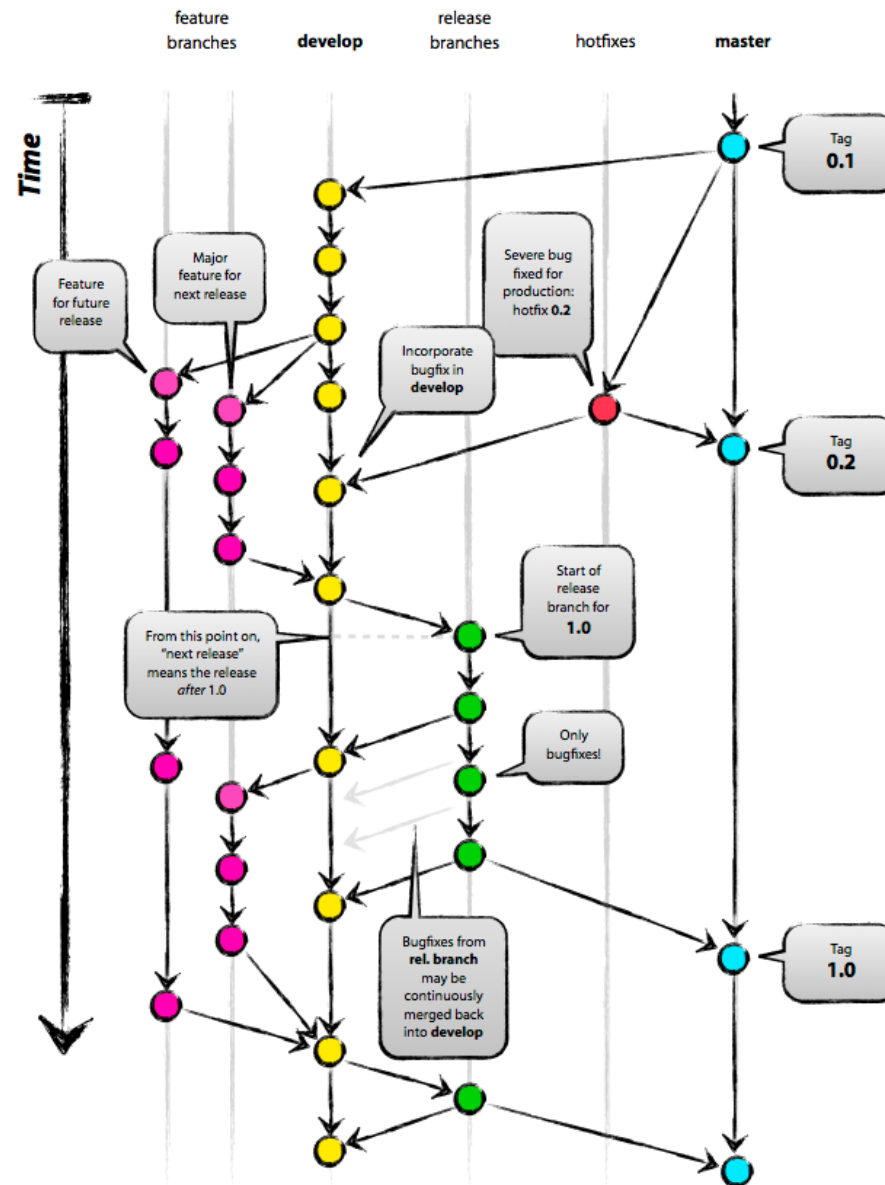
#
# Do some development... Return here if some changes are needed.
#

# 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
```

# Git Flow



A successful Git branching model ([\[link\]](#)[\[gitflow\]](#))

# Базовые принципы

1. Каждому проекту следует выработать свой рабочий процесс и правила именования веток.  
Желательно основываться на популярных подходах.
2. Приложение строится только на основе известного состояния репозитория:
  - *Не только релизы, но и экспериментальные и тестовые сборки (builds).*
  - *В идеале приложение умеет сообщать свою ревизию и параметры сборки.*
3. Стабильность общих (публичных) веток:
  - *Они обязаны компилироваться и проходить все тесты в любой момент времени.*
  - *Изменения тестируются до попадания в репозиторий.*
  - *Если дефектные изменения прошли, они исправляются в срочном порядке.*
4. Абсолютно вся разработка фиксируется в истории:
  - *Это делается в виде отдельных веток локального или глобального репозитория.*
  - *"Удачные" изменения добавляются в основную ветвь.*
5. Публичная история проекта не "переписывается":
  - *Однажды помеченные тэгами и выпущенные релизы модификации не подлежат.*
  - *Промежуточная история не переписывается, потому что будут конфликты.*



# Ссылки

1. Wikipedia ["Системы контроля версий"](#).

**Backup**

# Программная реализация

 3rdparty

---

 docs

---

 include

---

 perf

---

 sample

---

 src

---

 test

---

 testdata

---

 .gitignore


---

 .travis.yml

---

 CMakeLists.txt

---

 README.md

# Используемые инструменты

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

