

# Git: распределённая система контроля версий

Н. Д. Кудасов

МГУ им. Ломоносова

Москва, 2013

# Содержание

- 1 Введение
- 2 Git: модель репозитория
- 3 Git: ветвление
- 4 Git: удалённые репозитории
- 5 GitHub: открытое взаимодействие

# Системы контроля версий

Системы контроля версий позволяют отслеживать изменения в коде, документах и пр.

## Централизованные системы

Единый сервер для хранения документов и истории.  
Централизованный контроль над хранилищем.

## Распределенные системы

Каждый пользователь имеет собственную версию репозитория.  
Изменения могут передаваться между репозиториями.

# Зачем нужны системы контроля версий?

Случалось ли, что Вы

- изменили код, обнаружили ошибку и захотели откатиться?
- потеряли код, а имеющийся бекап оказался устаревшим?
- должны были поддерживать несколько версий программы?
- хотели посмотреть, чем отличаются две версии программы?
- хотели обнаружить, какое изменение внесло ошибку в программу?
- хотели просмотреть историю исходного кода?
- хотели отправить изменение для чужого кода?
- хотели выложить код, чтобы другие могли работать над ним?
- хотели посмотреть, сколько работы проделано? Когда? Кем?
- хотели попробовать что-то новое, не ломая основного кода?
- и т.д.

# Системы контроля версий: примеры

## Централизованные системы

- CVS
- Subversion (SVN)
- Visual SourceSafe, Vault и т.д.

## Распределенные системы

- Git,
- Darcs,
- Mercurial, Bazaar,
- Bitkeeper и т.д.



Одна из наиболее популярных систем контроля версий.

## Основные преимущества

- пример хорошо написанной и производительной программы на C;
- компактный репозиторий (эффективное сжатие и хранение истории);
- множество копий/бекапов;
- произвольная организация работы с несколькими репозиториями;
- промежуточная область для коммитов;
- простые в использовании ветки.

Git подходит для использования в любом проекте (как и любая распределенная система контроля версий).

## Примеры использований

- ядро ОС Linux и смежные проекты;
- gcc и смежные проекты;
- ОС Android,
- Qt, Cairo,
- Ruby on Rails, jQuery,
- GitHub.com,
- огромное количество других проектов.

- Pro Git: <http://progit.org/book/>
- перевод Pro Git: <http://progit.org/book/ru>
- Git Immersion: <http://gitimmersion.com/>
- Официальный сайт: <http://git-scm.com/>
- `git help [command]`

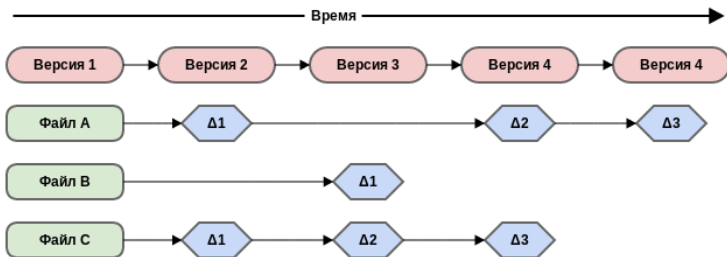


# Содержание

- 1 Введение
- 2 Git: модель репозитория**
- 3 Git: ветвление
- 4 Git: удалённые репозитории
- 5 GitHub: открытое взаимодействие

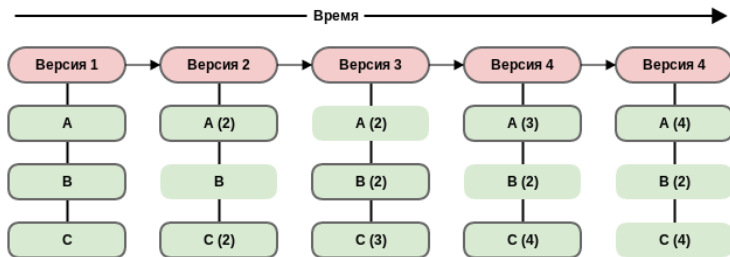
# История: хранение изменений

Большинство систем контроля версий хранят историю в виде списка изменений для каждого файла. В таких системах информация о проекте представляется в виде множества файлов и изменений для каждого файла:



# История: хранение снимков

Git смотрит на дело по-другому: он хранит историю как ряд снимков проекта. При каждом коммите создаётся очередной снимок проекта. Если файл не менялся, в новом снимке будет ссылка на файл из предыдущего снимка.



# Три состояния

Файлы (документы) в Git могут находиться в одном из трёх состояний: зафиксированном, изменённом и подготовленном. *Зафиксирован* означает, что файл сохранён в истории. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит. Таким образом, в Git существуют три части:

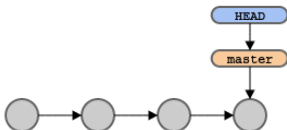


## Коммит

- набор файлов (снимок);
- ссылки на родительские коммиты;
- уникальный SHA1-код (хеш).

## Ссылка

- просто ссылка на коммит;
- имеют имена, по умолчанию создается ссылка master;
- ссылка, активная на текущий момент, имеет синоним HEAD.



# Первоначальная настройка

Прежде, чем начать работать с Git, ему необходимо сообщить немного о себе:

```
fizruk@example[~]$ git config --global user.name "Nickolay Kudasov"  
fizruk@example[~]$ git config --global user.email "nickolay.kudasov@gmail.com"  
fizruk@example[~]$
```

Эта информация будет использована в истории, чтобы можно было проследить, кто ответственен за какие изменения.

Опция `-global` вносит изменения в файл `.gitconfig` в домашней директории. Этот файл используется по умолчанию. Если опция не указана, настройки применяются только к текущему репозиторию.

# Создание репозитория

При создании репозитория, Git создаёт служебную директорию `.git/` в корне проекта:

```
fizruk@example[~]$ mkdir proj
fizruk@example[~]$ cd proj
fizruk@example[~/proj]$ git init
Initialized empty Git repository in /home/fizruk/demo/proj/.git/
fizruk@example[~/proj](master)$ ls -a
./ ../ .git/
fizruk@example[~/proj](master)$
```

Команда `git add` регистрирует изменения, которые попадут в очередной коммит. Команда `git commit` создаёт коммит.

```
fizruk@example[~/proj](master)$ vim README
fizruk@example[~/proj](master)$ git add README
fizruk@example[~/proj](master)$ git commit -m "added README"
[master (root-commit) 946bd2c] added README
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README
fizruk@example[~/proj](master)$
```

Этих команд уже хватает для работы над проектом с Git!

# Просмотр истории

Команда `git log` позволяет показать историю изменений:

```
fizruk@example[~/proj](master)$ git log
commit 034acd567fa854fba09ec892385625358b34a301
Author: Nickolay Kudasov <nickolay.kudasov@gmail.com>
Date:   Mon Nov 25 19:12:33 2013 +0400

    moved prog -> hello

commit bb0cf59975837a895eaaa012dc10fd13cd75a6ee
Author: Nickolay Kudasov <nickolay.kudasov@gmail.com>
Date:   Mon Nov 25 19:12:06 2013 +0400

    added simple program

commit a597a3427b7eef3f77c7f93fc5d371f54785a1b8
Author: Nickolay Kudasov <nickolay.kudasov@gmail.com>
Date:   Mon Nov 25 19:07:58 2013 +0400

    added README
fizruk@example[~/proj](master)$
```

```
fizruk@example[~/proj](master)$ git log --decorate --oneline
034acd5 (HEAD, master) moved prog -> hello
bb0cf59 added simple program
a597a34 added README
fizruk@example[~/proj](master)$
```



# Состояние проекта

Команда `git status` позволяет посмотреть список изменений (зарегистрированных, незарегистрированных, новых файлов, и т.д.):

```
fizruk@example[~/proj](master)$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.cpp
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       LICENSE
no changes added to commit (use "git add" and/or "git commit -a")
fizruk@example[~/proj](master)$
```

# Просмотр изменений

Команда `git diff` позволяет посмотреть сами изменения:

```
fizruk@example[~/proj](master)$ git diff
diff --git a/hello.cpp b/hello.cpp
index 6c5e020..ad4d84b 100644
--- a/hello.cpp
+++ b/hello.cpp
@@ -3,6 +3,8 @@
 using namespace std;

 int main(void) {
- cout << "Hello, username!" << endl;
+ string username;
+ cin >> username;
+ cout << "Hello, " << username << "!" << endl;
   return 0;
 }
fizruk@example[~/proj](master)$
```

# Содержание

- 1 Введение
- 2 Git: модель репозитория
- 3 Git: ветвление**
- 4 Git: удалённые репозитории
- 5 GitHub: открытое взаимодействие

# Создание ветки

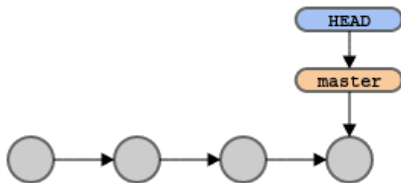
Ветки в системах контроля версий позволяют отклониться от основной линии разработки. В Git (в отличие от многих других систем) ветки легковесны, что позволяет работать одновременно с тысячами веток в больших проектах. Команда `git branch` создаёт новую ветку, а `git checkout` меняет текущую ветку:

```
fizruk@example[~/proj](master)$ git branch test
fizruk@example[~/proj](master)$ git branch -v
* master 034acd5 moved prog -> hello
  test   034acd5 moved prog -> hello
fizruk@example[~/proj](master)$ git checkout test
Switched to branch 'test'
fizruk@example[~/proj](test)$
```

# Что происходит?

Ветки — это просто ссылки!

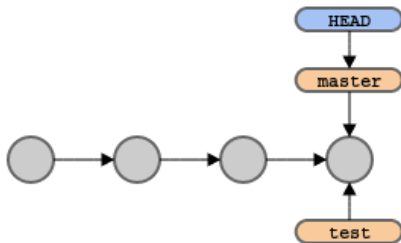
Исходный репозиторий



# Что происходит?

Ветки — это просто ссылки!

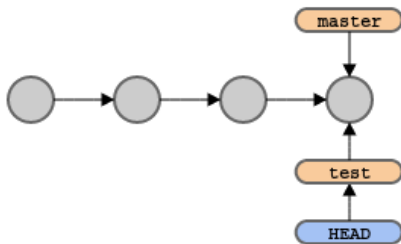
`git branch test`



# Что происходит?

Ветки — это просто ссылки!

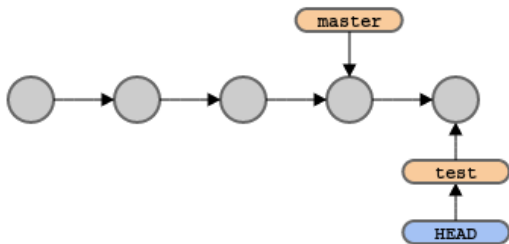
```
git checkout test
```



# Что происходит?

Ветки — это просто ссылки!

`git commit`

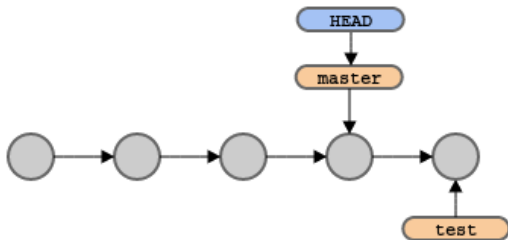




# Что происходит?

Ветки — это просто ссылки!

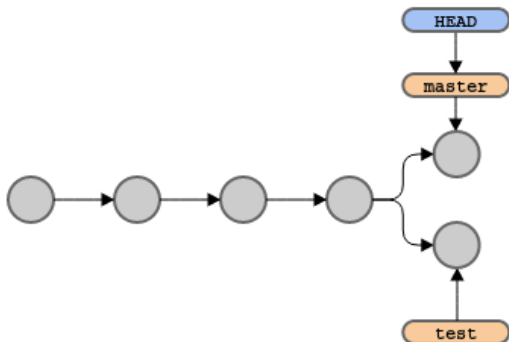
`git checkout master`



# Что происходит?

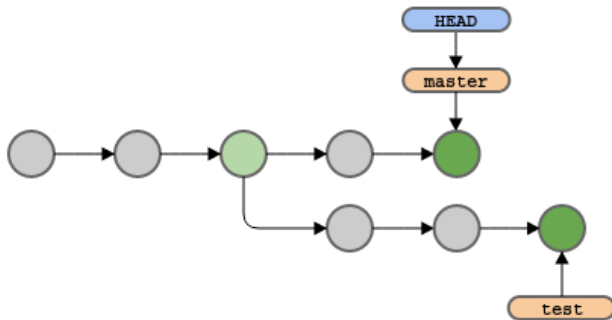
Ветки — это просто ссылки!

`git commit`



## Слияние веток

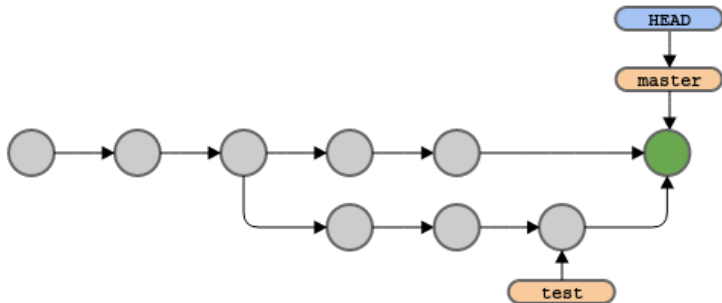
При слиянии веток Git находит наилучшего общего предка двух коммитов, на которые ссылаются ветки. Если в ветке, в которую происходит слияние, нет новых коммитов, то указатель просто передвигается. Иначе создаётся новый коммит с *двумя предками*:



## Исходный репозиторий

# Слияние веток

При слиянии веток Git находит наилучшего общего предка двух коммитов, на которые ссылаются ветки. Если в ветке, в которую происходит слияние, нет новых коммитов, то указатель просто передвигается. Иначе создаётся новый коммит с *двумя предками*:



`git merge test`

# Содержание

- 1 Введение
- 2 Git: модель репозитория
- 3 Git: ветвление
- 4 Git: удалённые репозитории
- 5 GitHub: открытое взаимодействие

# Клонирование репозитория

Команда `git clone` полностью копирует репозиторий.

```
fizruk@example[~]$ git clone https://github.com/fizruk/git-demo.git
Cloning into 'git-demo'...
remote: Counting objects: 30, done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 30 (delta 15), reused 20 (delta 8)
Unpacking objects: 100% (30/30), done.
Checking connectivity... done
fizruk@example[~]$ cd git-demo
fizruk@example[~/git-demo](master)$
```

Команда `git remote` перечисляет список удалённых репозиториев:

```
fizruk@example[~/git-demo](master)$ git remote -v
origin https://github.com/fizruk/git-demo.git (fetch)
origin https://github.com/fizruk/git-demo.git (push)
fizruk@example[~/git-demo](master)$
```

# Удалённые ветки

При клонировании происходит копирование всех веток, но только для основной ветки создаётся локальная версия.

```
fizruk@example[~/git-demo](master)$ git log --all --graph --decorate --oneline
* 32a6825 (HEAD, origin/master, origin/HEAD, master) dummy message in configure
* 53f2aea added Travis CI icon
* 3ea965f added Travis CI config
* 68942ec dummy configure
* 97ab09b add phony test
* 227e690 finish the conversation
| * 1fbd774 (origin/make) use fancy Makefile vars
|/
| * ba562e0 (origin/interact) Merge branch 'master' into interact
| \
|  /
|/ /
* | 8d2695f Merge branch 'master' into make
| \ \
| * | 2b6d101 ignore hello executable
* | | 7f38a6b added Makefile
|/ /
| * e96fd11 really ask now
| * bd6694e ask user for a name
|/
* 2043558 simple hello program
* f583201 Initial commit
fizruk@example[~/git-demo](master)$
```

# Обновление локального репозитория

Команда `git pull` [репозиторий] [:ветка] забирает изменения из удалённого репозитория и сливает их в текущую ветку:

```
fizruk@example[~/git-demo](master ↓)$ git pull
Updating 227e690..53f2aea
Fast-forward
 .travis.yml | 1 +
 Makefile    | 4 ++++
 README.md   | 2 ++
 configure   | 2 ++
 4 files changed, 9 insertions(+)
 create mode 100644 .travis.yml
 create mode 100755 configure
fizruk@example[~/git-demo](master)$
```

По умолчанию, это эквивалентно командам `git fetch` и `git merge`:

```
fizruk@example[~/git-demo](master ↓)$ git fetch
fizruk@example[~/git-demo](master ↓)$ git merge origin/master
Updating 227e690..53f2aea
Fast-forward
 .travis.yml | 1 +
 Makefile    | 4 ++++
 README.md   | 2 ++
 configure   | 2 ++
 4 files changed, 9 insertions(+)
 create mode 100644 .travis.yml
 create mode 100755 configure
fizruk@example[~/git-demo](master)$
```



# Обновление удалённого репозитория

Команда `git push` [репозиторий] [:ветка] отправляет изменения в удалённый репозиторий:

```
fizruk@example[~/git-demo](master)$ git push
Username for 'https://github.com': fizruk
Password for 'https://fizruk@github.com':
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/fizruk/git-demo.git
   53f2aea..32a6825  master -> master
fizruk@example[~/git-demo](master)$
```

Для успешной отправки необходимо, чтобы у Вас были *права на запись* в удалённый репозиторий и в удалённом репозитории не было *новых коммитов*. При нарушении последнего условия следует обновить локальный репозиторий (`git pull`), разрешить конфликты и повторить попытку.

# Содержание

- 1 Введение
- 2 Git: модель репозитория
- 3 Git: ветвление
- 4 Git: удалённые репозитории
- 5 **GitHub: открытое взаимодействие**

# Хостинг репозиториев

Поскольку с распределенными системами контроля версий создание репозиториев становится простым, появляются сервисы, предлагающие хостинг репозиториев для различных систем.

Для Git два основных хостера — это Bitbucket и GitHub. Они оба достаточно хороши и во многом схожи, однако у GitHub'а в несколько раз больше пользователей, что положительно сказывается на открытых проектах.



Помимо хранения репозиторий, GitHub предлагает небольшой набор удобств для работы над проектами:

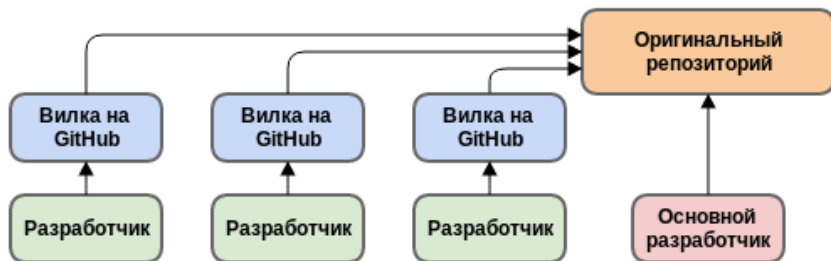
- простая система отслеживания ошибок (issue tracker);
- совместный просмотр кода;
- просмотр коммитов и сравнение веток;
- комментирование и обсуждение проблем, коммитов или отдельных участков кода;
- страницы вики (с настраиваемым доступом) для репозитория;
- хостинг статических сайтов для репозиторий/пользователей;
- возможность подключения сторонних сервисов (Travis CI и пр.).

# Вилки


GitHub представляет возможность копировать репозитории других пользователей (создать вилку/fork).

## Вилка

Клон репозитория *со ссылкой на исходный репозиторий*. Таким образом, существует может существовать один исходный репозиторий и сколько угодно вилок.








# Как это выглядит



This repository ▾ Search or type a command ⓘ

Explore Gist Blog Help



 fizruk   

PUBLIC  fizruk / git-demo 



Unwatch ▾ 1 ★ Star 0 Fork 0








Git demo repository. — Edit


11 commits 3 branches 0 releases 1 contributor

 branch: master ▾ git-demo / 

dummy message in configure

 fizruk authored an hour ago latest commit 32a68255f8 

 .gitignore	ignore hello executable	4 hours ago
 .travis.yml	added Travis CI config	an hour ago
 LICENSE	Initial commit	4 hours ago
 Makefile	add phony test	an hour ago
 README.md	added Travis CI icon	an hour ago
 configure	dummy message in configure	an hour ago
 hello.cpp	finish the conversation	4 hours ago

 README.md

## git-demo

build passing

<> Code

Issues 0

Pull Requests 0

Wiki


Pulse

Graphs

Network

Settings

HTTPS clone URL

https://github.com 

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#). ⓘ

Download ZIP

