

# Задача про систему контроля версий

Юрий Литвинов  
yurii.litvinov@gmail.com

02.03.2017г

# Немного про внутреннее устройство Git<sup>1</sup>

Структура папки .git:

- ▶ HEAD
- ▶ index
- ▶ config
- ▶ description
- ▶ hooks/
- ▶ info/
- ▶ objects/
- ▶ refs/
- ▶ ...

---

<sup>1</sup> По гл. 10 <https://git-scm.com/book> и <http://aosabook.org/en/git.html>

## Объекты

Git внутри — хеш-таблица, отображающая SHA-1-хеш файла в содержимое файла. Пример:

```
$ git init test
```

```
Initialized empty Git repository in /tmp/test/.git/
```

```
$ cd test
```

```
$ find .git/objects
```

```
.git/objects
```

```
.git/objects/info
```

```
.git/objects/pack
```

```
$ echo 'test content' | git hash-object -w --stdin  
d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

```
$ find .git/objects -type f
```

```
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

## Объекты (2)

Как получить сохранённый объект:

```
$ git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4  
test content
```

Версионный контроль:

```
$ echo 'version 1' > test.txt  
$ git hash-object -w test.txt  
83baae61804e65cc73a7201a7252750c76066a30  
$ echo 'version 2' > test.txt  
$ git hash-object -w test.txt  
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a  
$ find .git/objects -type f  
.git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a  
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30  
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

## Объекты (3)

Переключение между версиями файла:

```
$ git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30 \  
> test.txt
```

```
$ cat test.txt  
version 1
```

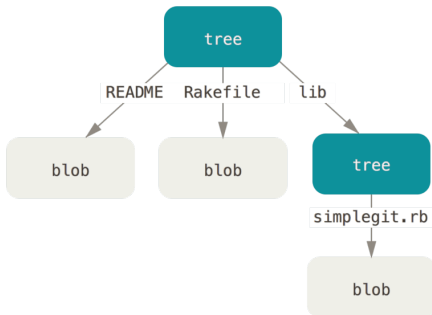
```
$ git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a \  
> test.txt
```

```
$ cat test.txt  
version 2
```

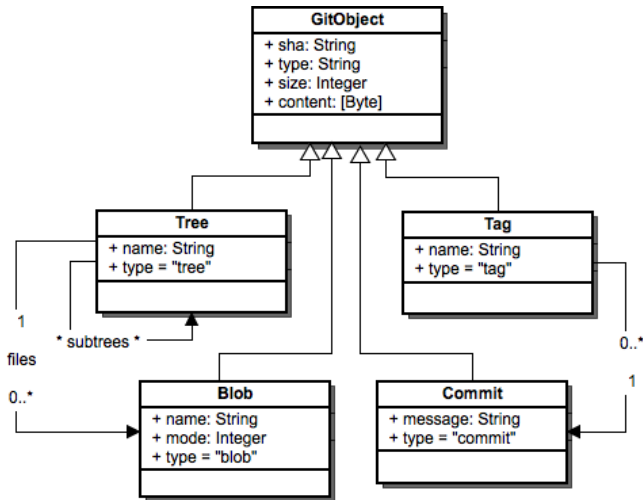
# Деревья

blob (то, что мы видели раньше) хранит только содержимое файла, не хранит даже его имя. Решение проблемы — tree:

```
$ git cat-file -p master^{tree}
100644 blob a906cb2a4a904a152e80877d4088654daad0c859    README
100644 blob 8f94139338f9404f26296befa88755fc2598c289    Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0    lib
```



# Какие ещё виды объектов бывают



## Коммиты

tree-объекты могут хранить структуру файлов (как inode в файловой системе), но не хранят метаданные типа автора файла и даты создания. Это хранится в commit-объектах:

```
$ echo 'first commit' | git commit-tree d8329f  
fdf4fc3344e67ab068f836878b6c4951e3b15f3d
```

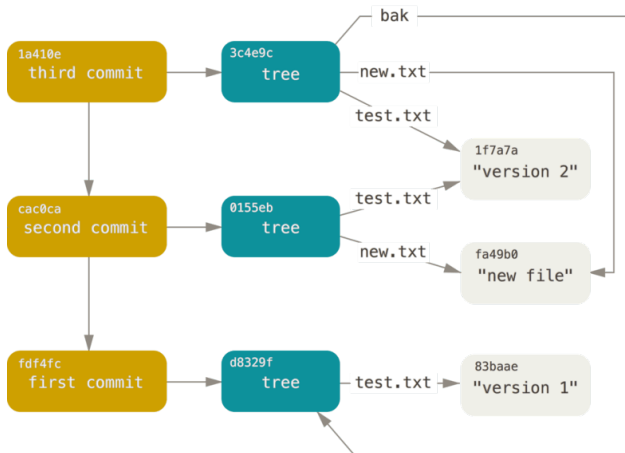
```
$ git cat-file -p fdf4fc3  
tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579  
author Scott Chacon <schacon@gmail.com> 1243040974 -0700  
committer Scott Chacon <schacon@gmail.com> 1243040974 -0700
```

first commit

Ещё коммит хранит список коммитов-родителей



# Коммиты, как это выглядит



## Ссылки

Теперь вся информация хранится на диске, но чтобы ей воспользоваться, нужно помнить SHA-1 хеши. На помощь приходят reference-ы.

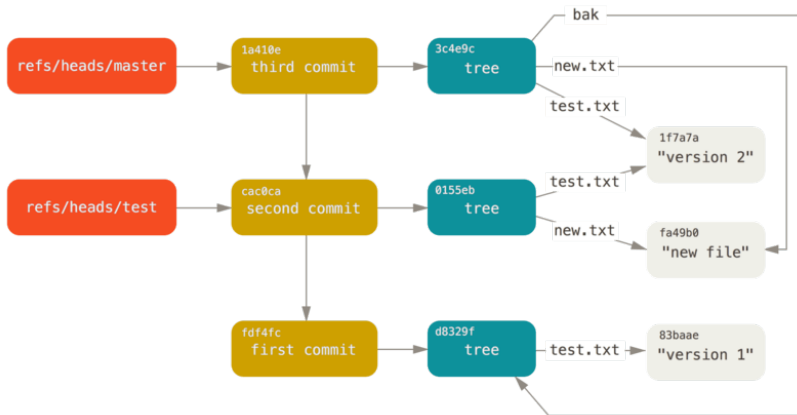
- ▶ .git/refs
- ▶ .git/refs/heads
- ▶ .git/refs/tags

```
$ echo "1a410efbd13591db07496601ebc7a059dd55cfe9" \  
> .git/refs/heads/master
```

```
$ git log --pretty=oneline master  
1a410efbd13591db07496601ebc7a059dd55cfe9 third commit  
cac0cab538b970a37ea1e769cbbde608743bc96d second commit  
fdf4fc3344e67ab068f836878b6c4951e3b15f3d first commit
```

- ▶ Команда git update-ref

# Ссылки, как это выглядит



# HEAD

Теперь не надо помнить хеши, но как переключаться между ветками?

Текущая ветка хранится в HEAD. HEAD — символическая ссылка, то есть ссылка на другую ссылку.

```
$ cat .git/HEAD  
ref: refs/heads/master
```

```
$ git symbolic-ref HEAD refs/heads/test  
$ cat .git/HEAD  
ref: refs/heads/test
```

# Тэги

Последний из объектов в Git — tag. Это просто указатель на коммит.

- ▶ Легковесный тэг:

```
git update-ref refs/tags/v1.0 cac0cab538b970a37ea1e769cbbde608743bc96d
```

Или просто `git tag`

- ▶ Аннотированный тэг:

```
$ git tag -a v1.1 1a410efbd13591db07496601ebc7a059dd55cfe9 -m 'test tag'
```

```
$ git cat-file -p 9585191f37f7b0fb9444f35a9bf50de191beadc2
```

```
object 1a410efbd13591db07496601ebc7a059dd55cfe9
```

```
type commit
```

```
tag v1.1
```

```
tagger Scott Chacon <schacon@gmail.com> Sat May 23 16:48:58 2009 -0700
```

```
test tag
```

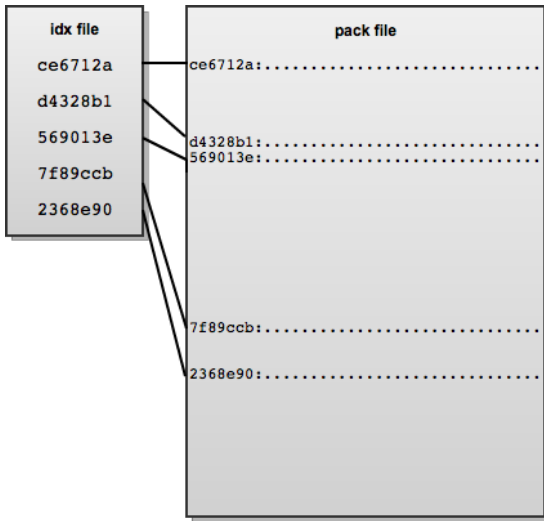
# Packfiles

Пока что получалось, что все версии всех файлов в Git хранятся целиком, как они есть. Все они всегда сжимаются zlib, но в целом, если создать репозиторий, добавлять туда файлы, коммитить и т.д., все версии всех файлов будут в нём целиком. На помощь приходят .pack-файлы:

```
$ git gc
Counting objects: 18, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (18/18), done.
Total 18 (delta 3), reused 0 (delta 0)
```

```
$ find .git/objects -type f
.git/objects/bd/9dbf5aae1a3862dd1526723246b20206e5fc37
.git/objects/d6/70460b4b4aeece5915caf5c68d12f560a9fe3e4
.git/objects/info/packs
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx
.git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.pack
```

# Как оно устроено



# Pack-файлы, подробности

- ▶ Упаковка происходит, когда:
  - ▶ Выполняется `git push`
  - ▶ Слишком много «свободных» объектов (порядка 7000)
  - ▶ Вручную вызвана `git gc`
- ▶ Используется дельта-компрессия
  - ▶ Последняя версия хранится целиком, дельты «идут назад»
- ▶ Можно заглянуть внутрь, `git verify-pack`
- ▶ Git может хитро перепаковать pack-файлы



## Reflog и восстановление коммитов

```
$ git reflog
```

```
1a410ef HEAD@{0}: reset: moving to 1a410ef
```

```
ab1afef HEAD@{1}: commit: modified repo.rb a bit
```

```
484a592 HEAD@{2}: commit: added repo.rb
```

```
$ git log -g
```

```
commit 1a410efbd13591db07496601ebc7a059dd55cfe9
```

```
Reflog: HEAD@{0} (Scott Chacon <schacon@gmail.com>)
```

```
Reflog message: updating HEAD
```

```
Author: Scott Chacon <schacon@gmail.com>
```

```
Date: Fri May 22 18:22:37 2009 -0700
```

third commit

```
$ git branch recover-branch ab1afef
```

# Как более капитально прострелить себе ногу

И что делать

```
$ git branch -D recover-branch
```

```
$ rm -Rf .git/logs/
```

```
$ git fsck --full
```

Checking object directories: 100% (256/256), done.

Checking objects: 100% (18/18), done.

dangling blob d670460b4b4aece5915caf5c68d12f560a9fe3e4

dangling commit ab1afef80fac8e34258ff41fc1b867c702daa24b

dangling tree aea790b9a58f6cf6f2804eeac9f0abbe9631e4c9

dangling blob 7108f7ecb345ee9d0084193f147cdad4d2998293

Git не удалит даже «висячие» объекты несколько месяцев, если его явно не попросить.

# Условие задачи

## Своя локальная система контроля версий

Требуется сделать систему контроля версий, представляющую из себя консольное приложение и умеющую:

- ▶ `commit` с `commit message` (сообщение обязательно и принимается как параметр, система должна сама добавлять ещё дату коммита и автора)
- ▶ работу с ветками: создание и удаление
- ▶ `checkout` по имени ревизии или ветки
- ▶ `log` — список ревизий вместе с `commit message` в текущей ветке
- ▶ `merge` — сливает указанную ветку с текущей
  - ▶ конфликты разрешайте (или не разрешайте) любым разумным способом

# Нефункциональные требования

- ▶ Документация: комментарии, помощь для пользователя, краткое описание внутреннего устройства
- ▶ Тесты: эмуляция окружения, включая файловую систему
- ▶ Логгирование
- ▶ Исключения, обработка ошибок
- ▶ Вывод в консоль — только в клиентском коде типа `main()`, основной код должен позволять себя использовать как библиотеку
- ▶ Развитый программный интерфейс, должно быть можно без проблем потом прикрутить GUI
- ▶ Аннотации `@NotNull`, `@Nullable/Optional`
- ▶ Continuous Integration

# Примечания

- ▶ Не накладывается никаких ограничений на хранимые на диске данные и их формат
  - ▶ Дельта-компрессию делать не надо
- ▶ Не требуется работа с удалёнными репозиториями
- ▶ Дедлайн: до 23:59 30.03