# Системы контроля версий. Основы Git

# Не забудь включить запись!

# План

- Version Control и его история
- Принципы работы Git
- Основы работы с Git. Основные команды
- Pull Request и Peer Review

# Version Control

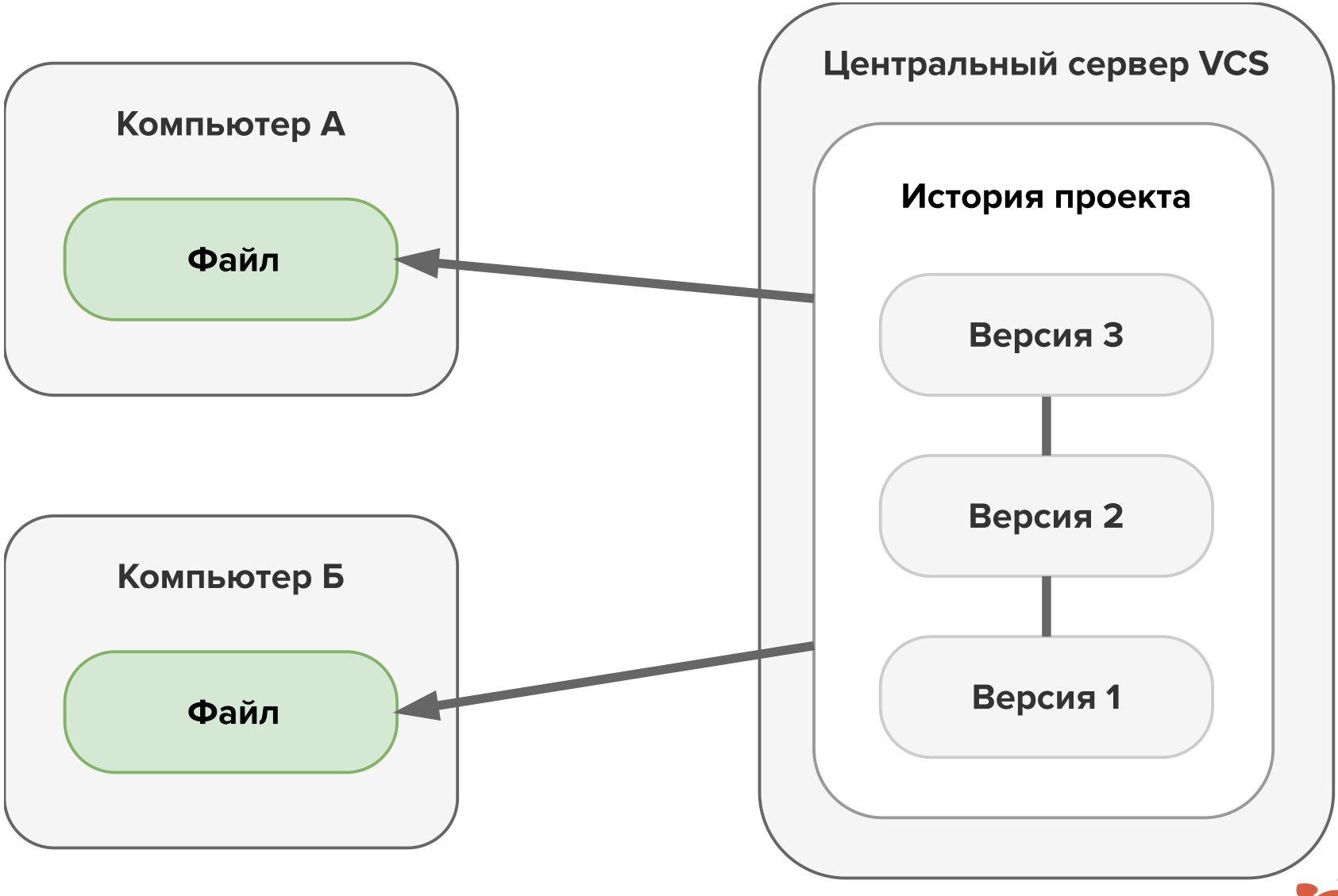
- Единый "источник правды"
- Основа для коллективной работы
- Основа процесса разработки ПО и всего процесса непрерывной поставки в целом
- Основа для DevOps-практик

> ☝ Version Control - это инструмент не только для разработчиков

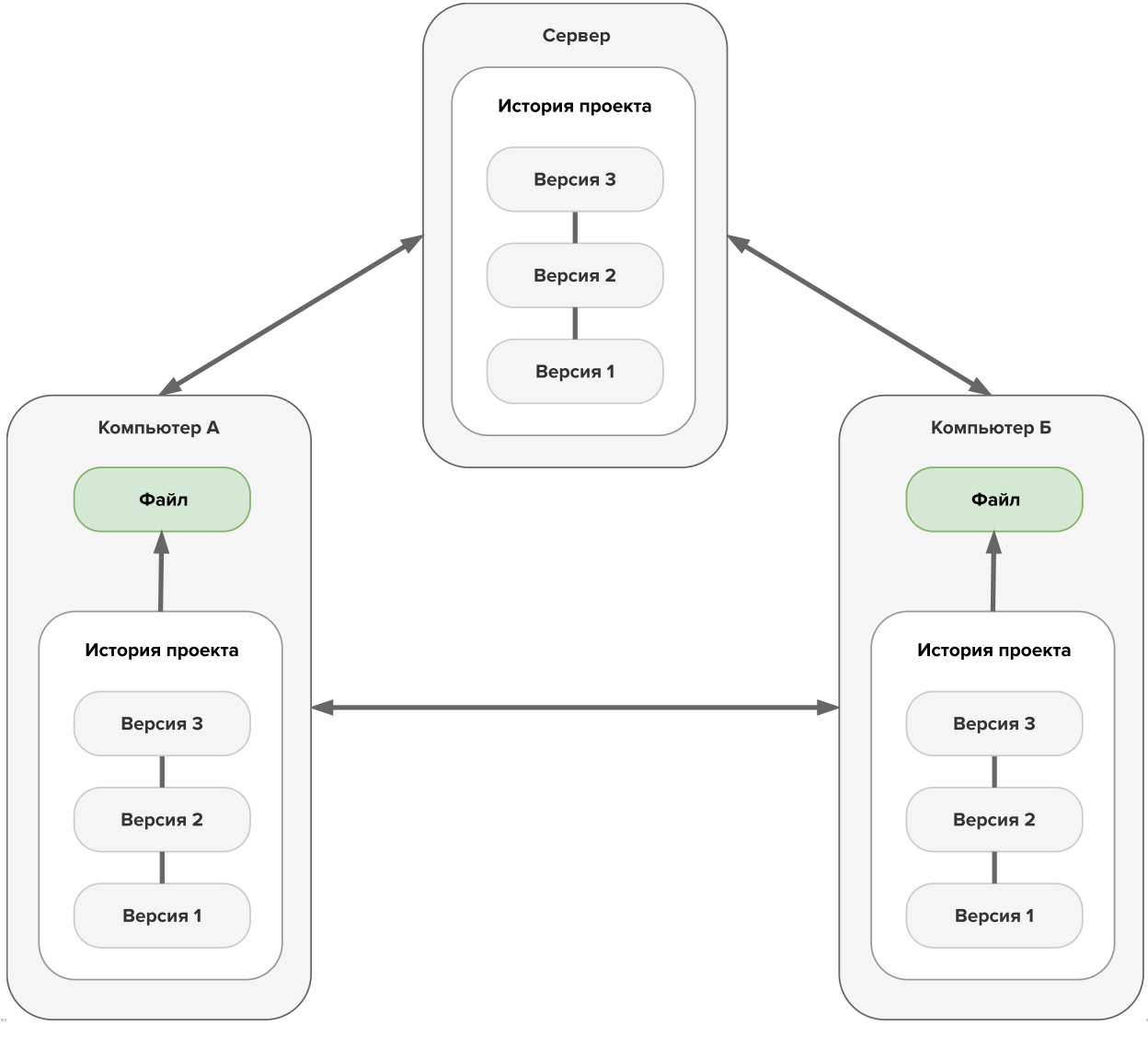
# Зачем нужен Version Control?

- Возврат к старым версиям (safety net)
- Отслеживание изменений
- Хранение истории проекта
- Совместная работа
- Кто, когда и зачем?

# Централизованные VCS



# Распределенные VCS

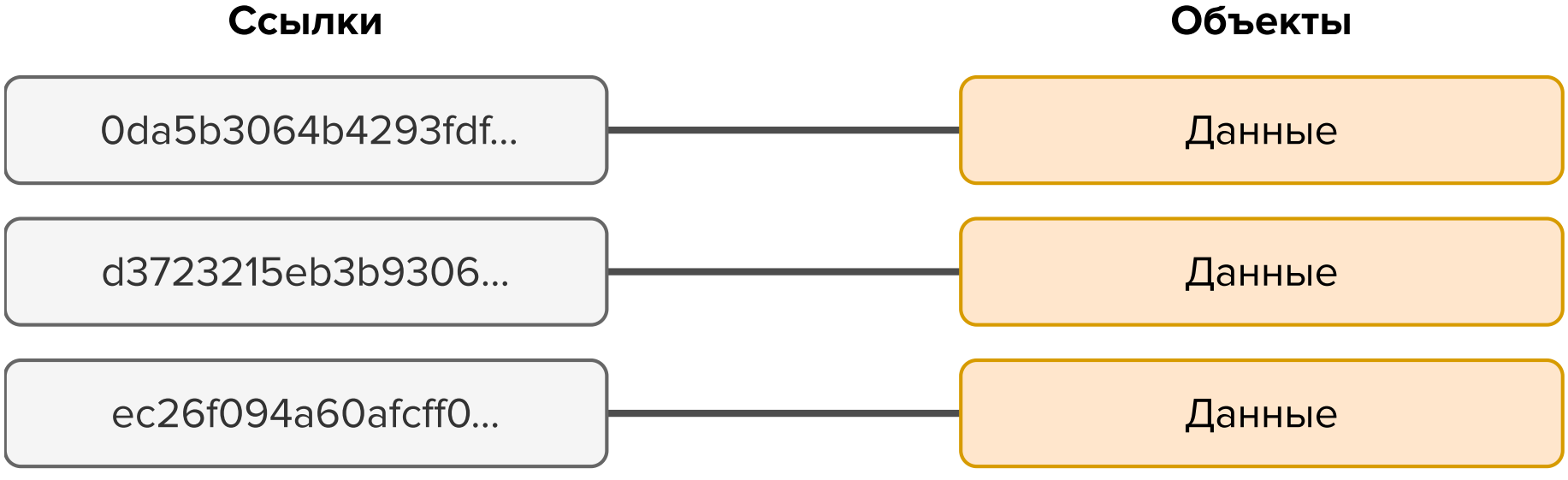


# DVCS vs CVCS

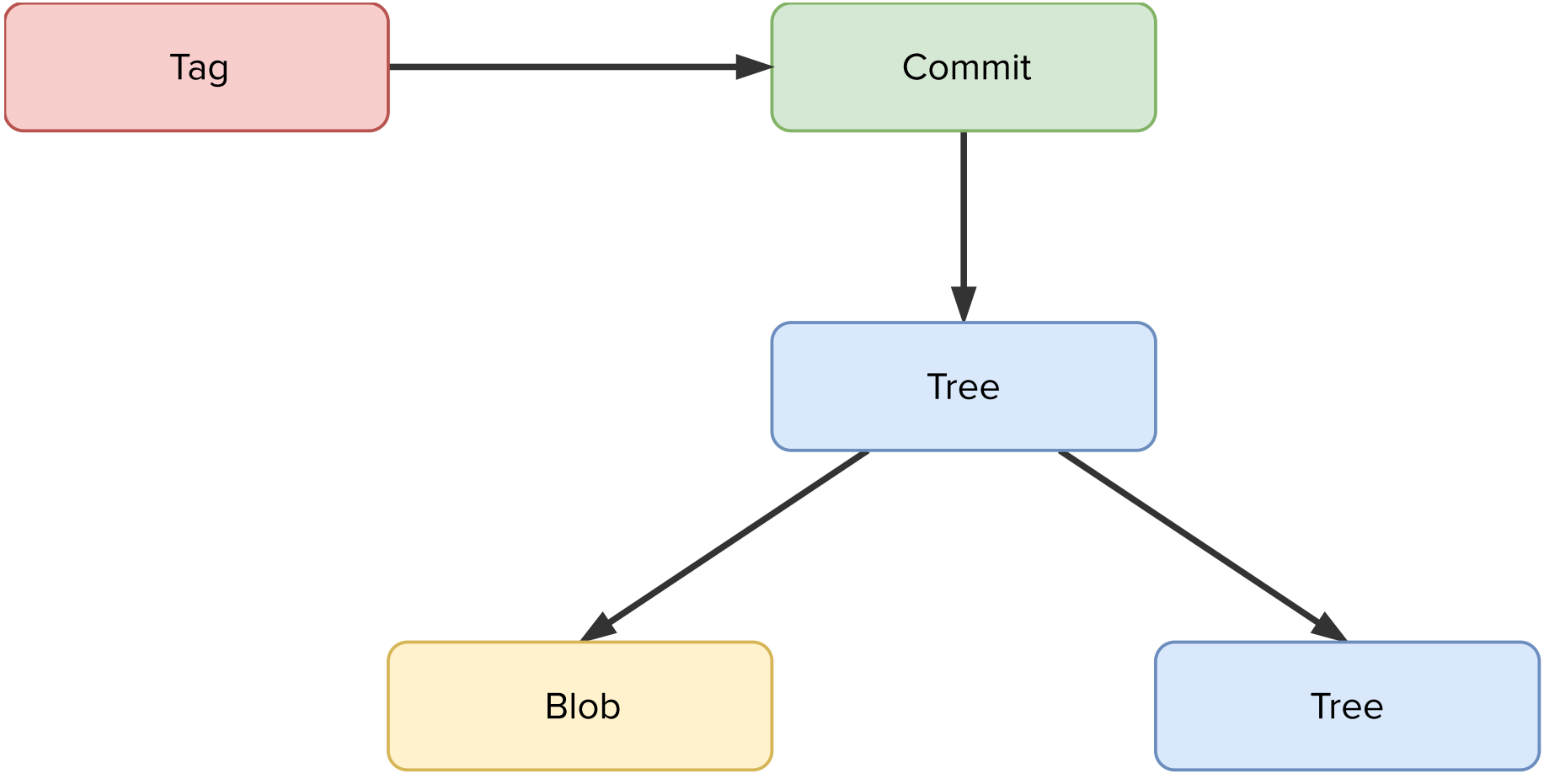
- Полноценная локальная копия проекта
- Локальные репозиторий ~ Резервная копия
- Можно работать оффлайн
- Скорость

# Принципы работы Git

# Хранилище с адресацией по содержимому

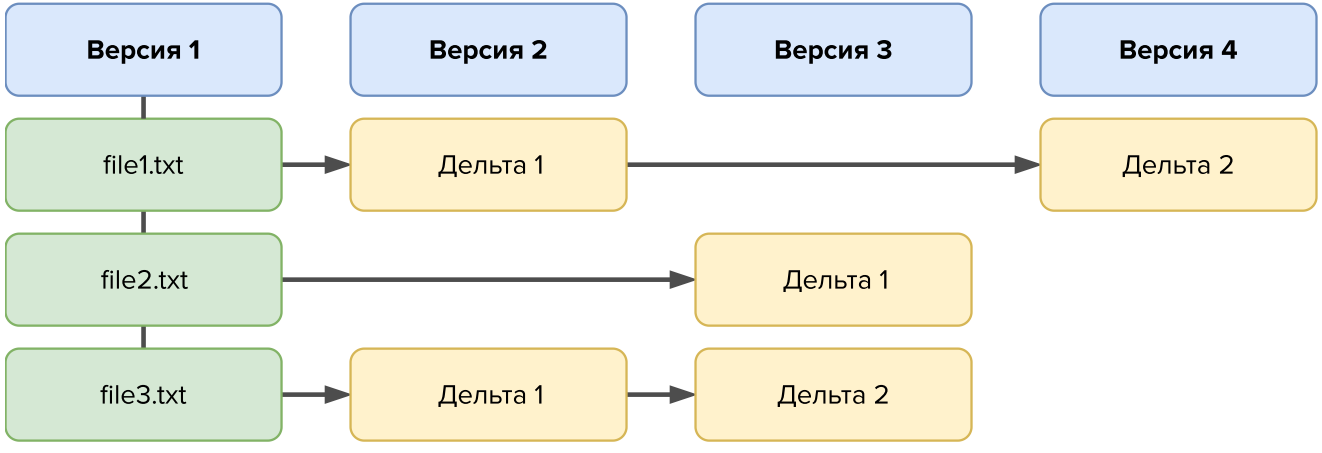


# Структура хранимых данных

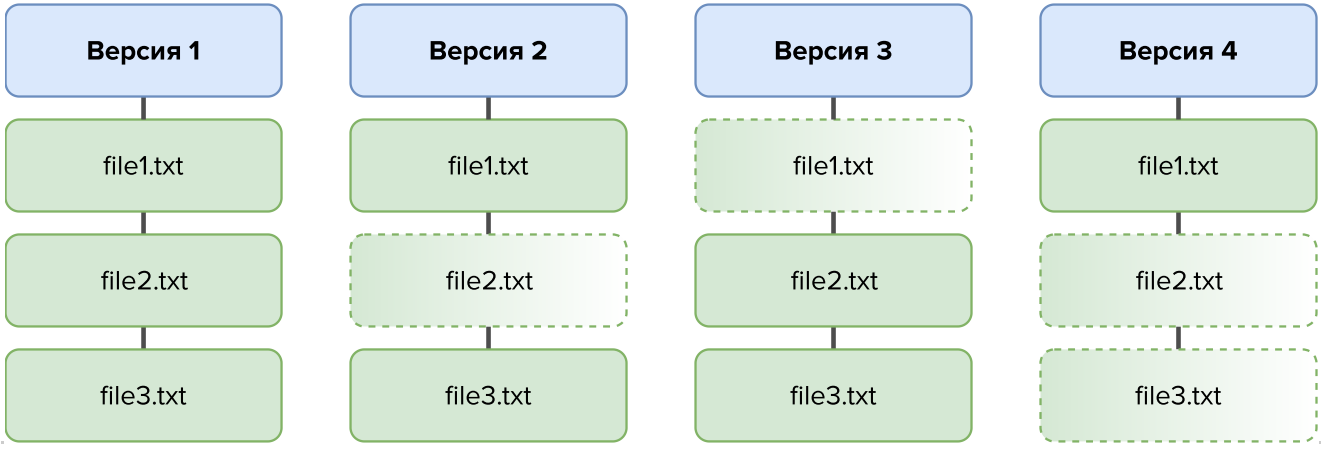


# Хранение изменений

**Формат дельт (хранение различий): SVN**




**Формат срезов (snapshots): Git**



# Где хранится история?

Git-репозиторий хранится в директории `.git` в корне проекта.

В репозитории содержатся:

- объекты коммитов
- ссылки на коммиты и ветки
- конфигурация
- скрипты-хуки

```
.git
├── COMMIT_EDITMSG
├── FETCH_HEAD
├── HEAD
├── config
├── description
├── hooks
├── index
├── info
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
├── objects
│   ├── 01
│   │   └── 2dc51bccc1
│   ├── 5a
│   │   └── 2669f47b80
│   ├── 7b
│   │   ├── bcb3154035
│   │   └── cf9b7d210c
│   ├── 9d
│   │   └── 4fc2f6782e
│   ├── f7
│   │   └── a9d2e60188
│   ├── fd
│   │   └── 96eb24c77b
│   ├── info
│   └── pack
└── refs
    ├── heads
    │   └── master
    └── tags
```

# Как хранятся изменения?

В репозитории есть 4 типа хранимых объектов:

- Blob
- Tree
- Commit
- Tag

Посмотреть тип объекта можно командой:

```
git cat-file -t ${ХЭШ_ОБЪЕКТА}
```

# Blob (он же блоб, он же данные)

Blob - базовая единица хранения данных в Git.

Хранит snapshot (снимок) содержимого файла.

В качестве имени объекта берется SHA1 хеш, содержимого файла и заголовка.

```
~/src/repo [master]$ git cat-file -t 8c7a1cf
blob
~/src/repo [master]$ git cat-file -p 8c7a1cf
this is file.txt
second commit did this
~/src/repo [master]$ cat file.txt
this is file.txt
second commit did this
```

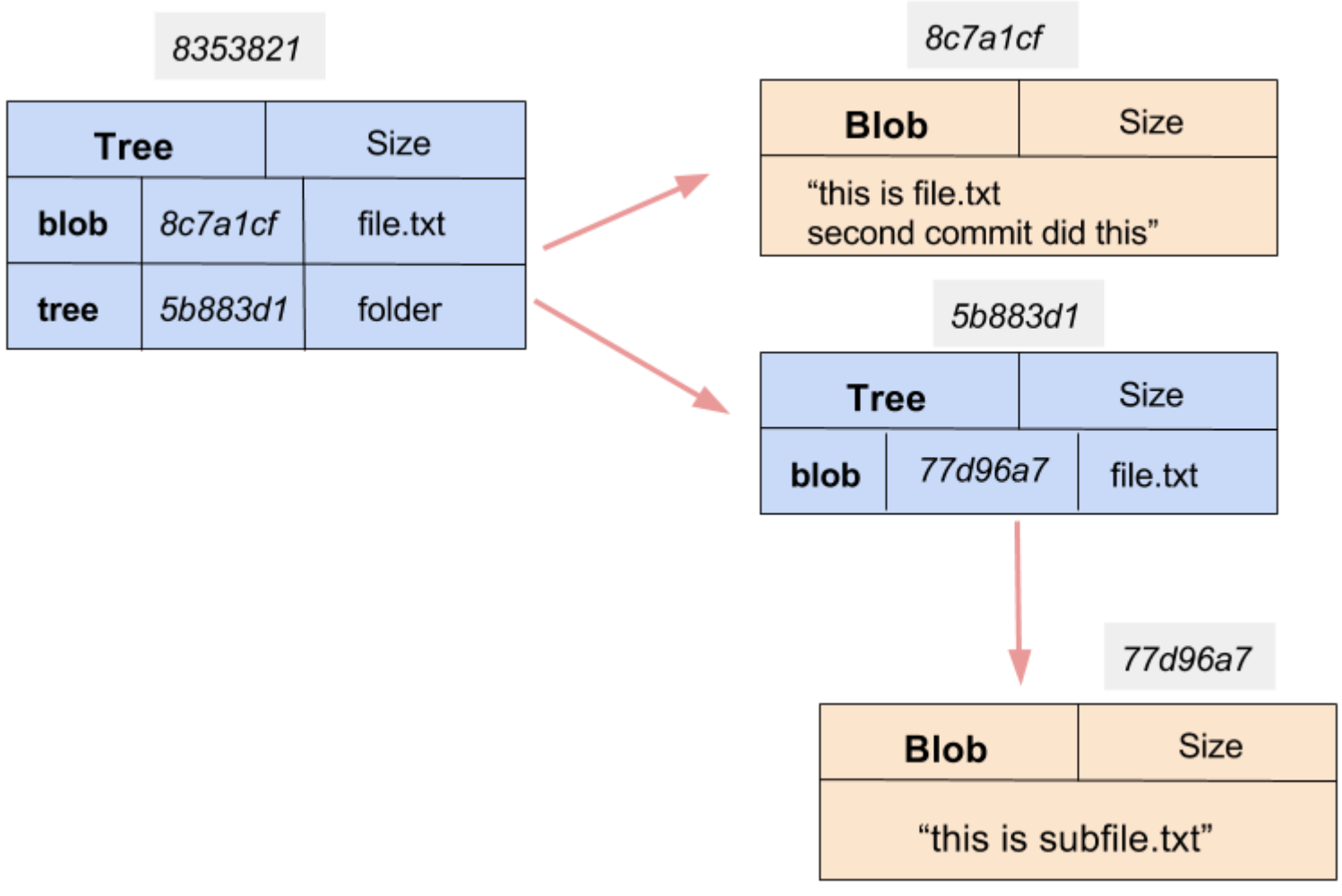
# tree (он же дерево, он же 🌳)

Деревья содержат информацию о блобах, а также других поддеревьях. Решают проблему хранения имен файлов и их группировки по директориям.

```
.
├── file.txt
└── folder
    └── subfile.txt
```

```
~/src/repo [master]$ git cat-file -t 8353821
tree
~/src/repo [master]$ git ls-tree 8353821
100644 blob 8c7a1cf844966a95bf68faf3cdd9da7f3d7662be    file.txt
040000 tree 5b883d12e60cfc52016d9dc9515e28b9e43850b7    folder

~/src/repo [master]$ git ls-tree 5b883d1
100644 blob 77d96a7ca3be9b02eb1d3624d1526d2a0a1ce4d5    subfile.txt
```

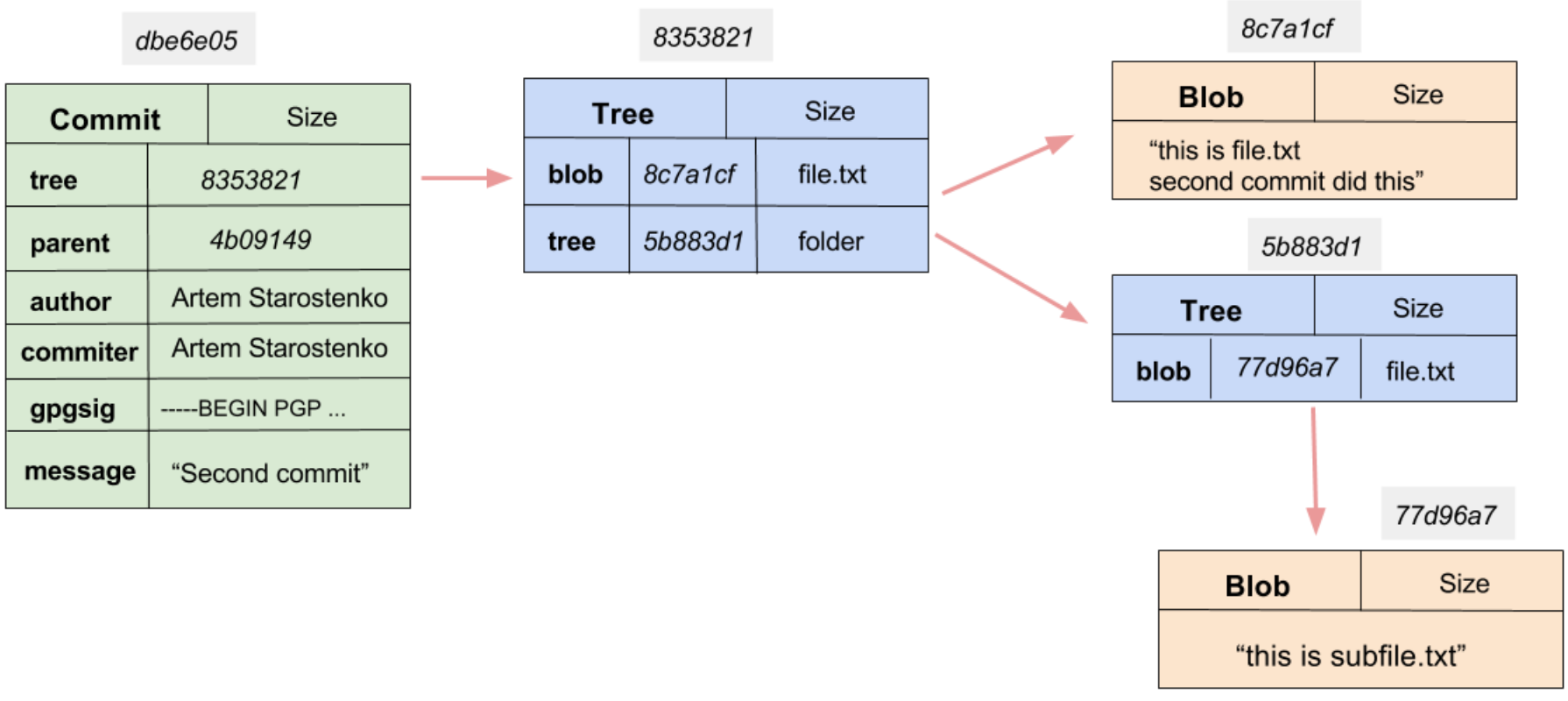
# Blobs и Trees



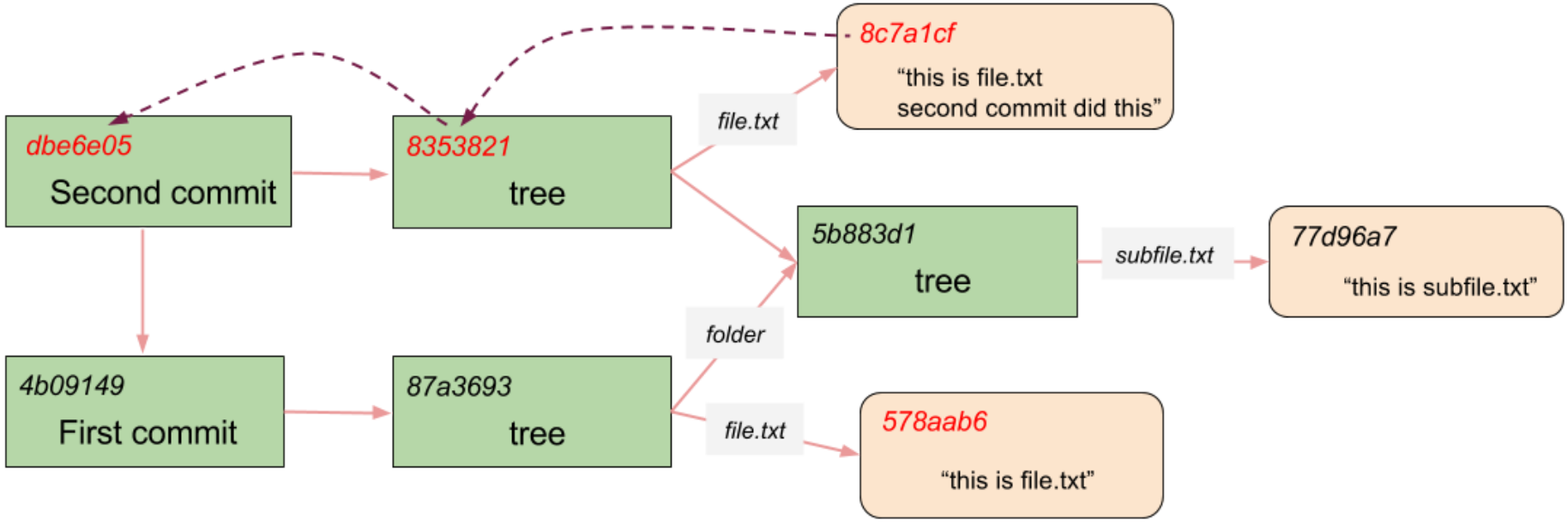
# Commit

- Автор, Коммитер, Дата
- Цифровая подпись
- Сообщение
- Дерево
- Родитель (или родители)

# Commit


dbe6e05
Commit
Size
tree
8353821
parent
4b09149
author
Artem Starostenko
commiter
Artem Starostenko
gpgsig
-----BEGIN PGP ...
message
“Second commit”
8353821
Tree
Size
blob
8c7a1cf
file.txt
tree
5b883d1
folder
8c7a1cf
Blob
Size
“this is file.txt
second commit did this”
5b883d1
Tree
Size
blob
77d96a7
file.txt
77d96a7
Blob
Size
“this is subfile.txt”

# Изменили содержимое file.txt



# Tags aka Тэги

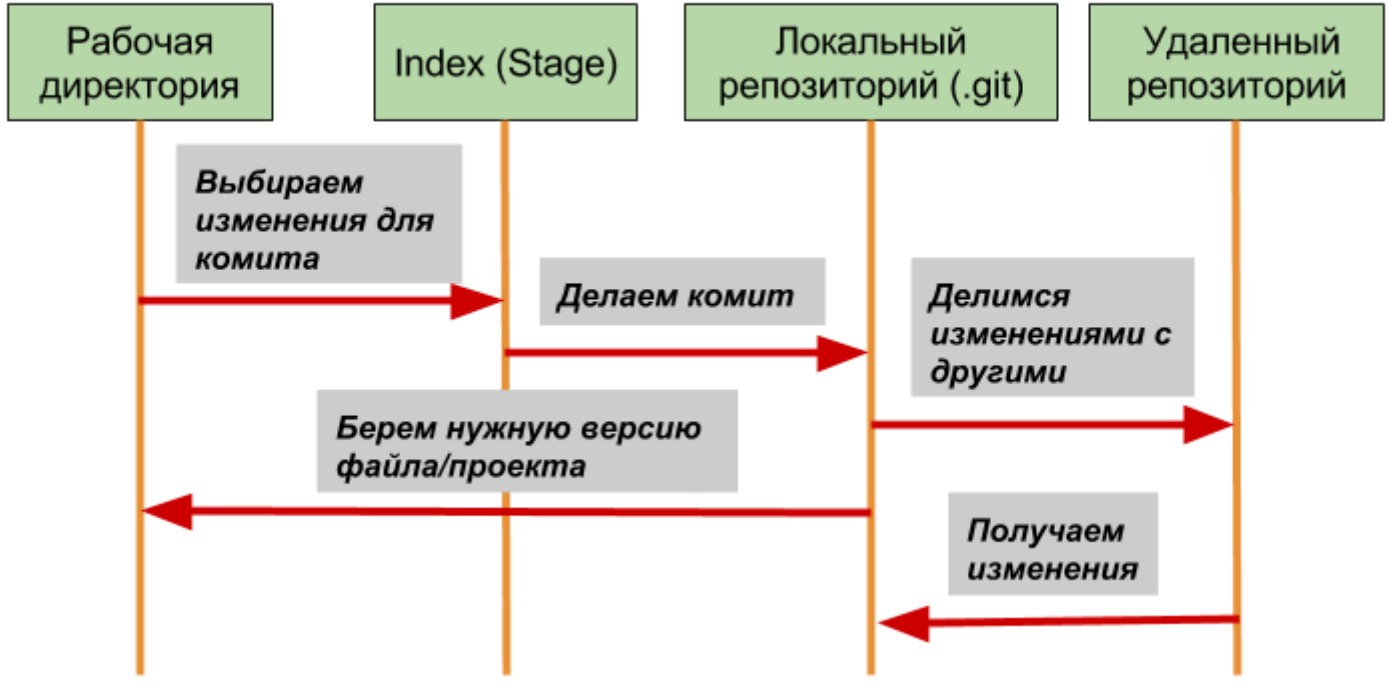
Два типа:

- Легковесные теги - указатели на определенный коммит
- Аннотированные - содержат имя, email поставившего тег, дату, комментарий и могут иметь цифровую подпись. Хранятся как полноценные объекты.

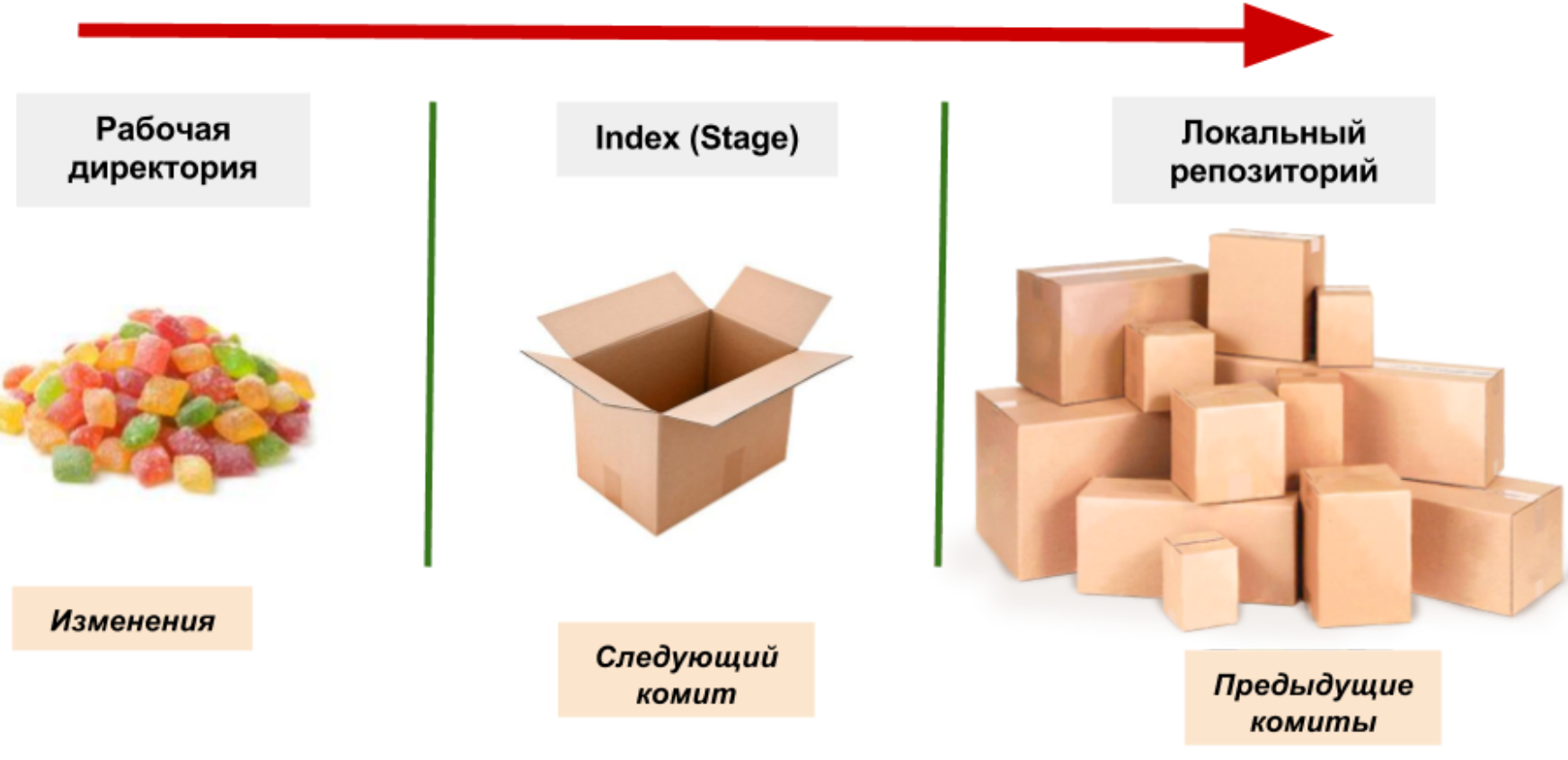
# Tags aka Тэги

```
~/src/repo [master]$ git tag -v v0.1

object dbe6e058e9b173c422234a4ae42e4add3c1de0e8
type commit
tag v0.1
tagger Artem Starostenko <artemstarostenko65@gmail.com> 1493145599 +0300

First working version
gpg: Signature made Tue Apr 25 21:39:59 2017 MSK
gpg:                using RSA key 255025923EC5215D80207A622487ACEADFE90FCC
gpg:                issuer "artemstarostenko65@gmail.com"
gpg: Good signature from "Artem Starostenko <artemstarostenko65@gmail.com>" [ultimate]
```

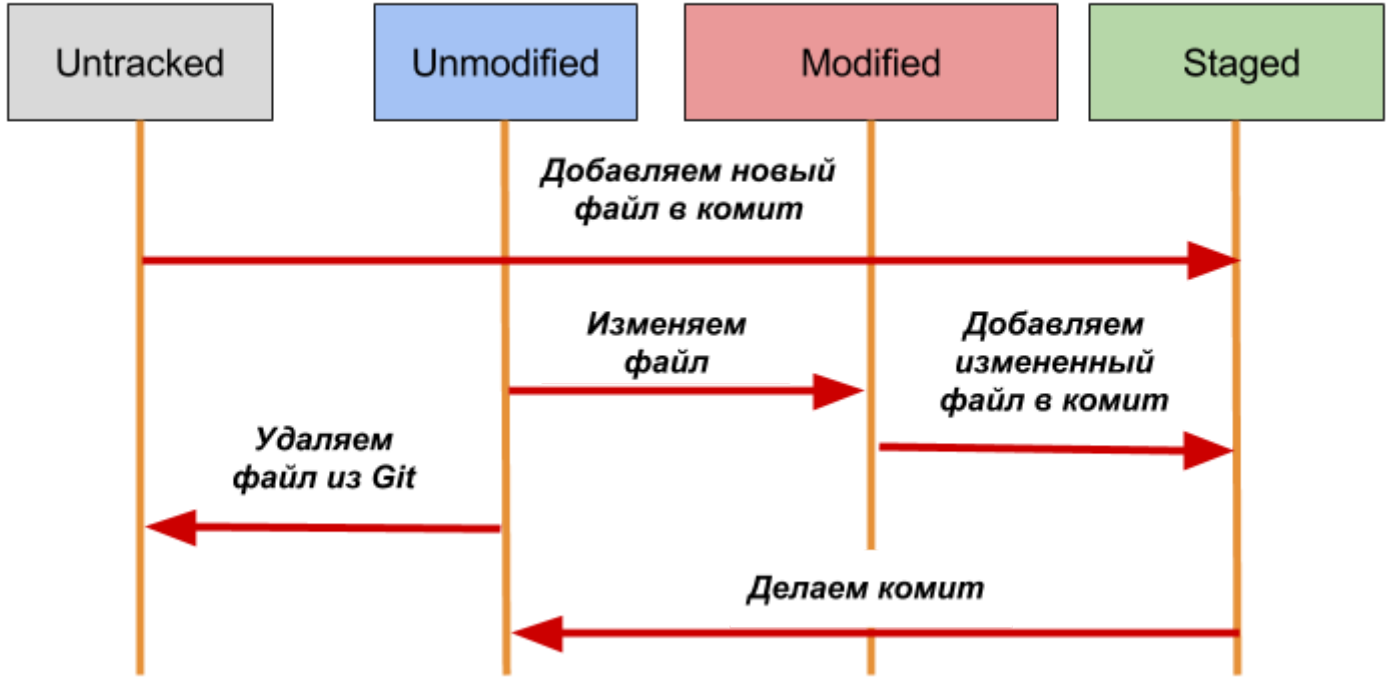
# Работа с Git


Рабочая директория
Index (Stage)
Локальный репозиторий (.git)
Удаленный репозиторий
Выбираем изменения для комита
Делаем комит
Делимся изменениями с другими
Берем нужную версию файла/проекта
Получаем изменения

# Работа с Git


Рабочая директория
Index (Stage)
Локальный репозиторий
Изменения
Следующий комит
Предыдущие комиты

# Состояния файлов



# Основные команды для работы с Git

# Конфигурация пользователя

Конфигурацию Git для конкретного пользователя можно найти так (еще есть глобальные и системные настройки):

- В файле `~/.gitconfig` в Linux
- В файле `$HOME\.gitconfig` в Windows
- Или через команду `git config --help`

# Основные секции

Самое важное, это настроить здесь имя и e-mail, так как данная информация будет использоваться при создании коммитов.

```
[user]
  name = Artem Starostenko
  email = artemstarostenko65@gmail.com
[color]
  ui = auto
[alias]
  co = checkout
  st = status
```

*Aliases* - сокращения для команд, они немного экономят время:

- `st` - `status`
- `co` - `checkout`
- `ci` - `commit`

# git init

Команда `git init` создает репозиторий в папке с проектом:

```
$ git init /path/to/project/folder
$ git init . # создать репозиторий в текущей папке
```

В результате будет создана папка `.git` и теперь можно начать отслеживать историю нашего проекта.

# git status

Команда `git status` отображает список всех изменений, которые имеются на данный момент и не закоммичены:

```
~/src/repo [master*]$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   file.txt

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:   folder/subfile.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    newfile.txt
```

# .gitignore

Позволяет явно объявить какие имена файлов не нужно отслеживать. Можно использовать маски.

```
~/src/repo [master]$ touch unnoticed-file.txt

~/src/repo [master*]$ git status
On branch master
Untracked files:
    unnoticed-file.txt

nothing added to commit but untracked files present (use "git add" to track)

~/src/repo [master*]$ echo "unnoticed-file.txt" >> .gitignore

~/src/repo [master*]$ git status
On branch master
Changes not staged for commit:
  modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
```

# Как сделать коммит?

Добавить только нужные нам файлы (или все подряд через `git add .`) и потом сделать коммит:

```
git add <filename/wildcard>
git commit -m "Ой, я, кажется, накоммитил"
```

Добавить в *staging* **все** отслеживаемые изменения файлов и сразу сделать коммит:

```
git commit -am "My short commit message"
```

# Best Practices при работе с сообщениями

Идеально, если сообщение в каждом коммите сделано по определенным правилам:

- Должно содержать описание целей или причин внесения изменений
- Должно содержать номер задачи в трекере (Jira, GitHub Issues)
- Должно быть содержательным и кратким. "Bugfix", "Ooops", "Commit" - не пояснение к коммиту

Пример таких правил - это Conventional Commits. Внутри команды можно договориться о правилах и использовать commitlint или conform, чтобы обеспечить соблюдение договоренностей.

# Примеры

```
fb49d2a   Set the border for children of root projects only.
6ff5d42   Move the closed project checkbox out of the sidebar.
124a459   Use the main menu for project related actions that support
          cross-project display.
33d78d6   View progress bar of related issues (#3425).
17233e8   Extracts the rendering of related issues to an helper (#3425).
aff93d7   Fix icon-news class (#24313).
745cb7b   Small improvement (#24313).
```

## Пример Conventional Commits:

```
67eadb8   docs: update README.md with prefix tag docs (#284)
689aaf8   chore(release): 5.0.1
bc43bca   docs(readme): fix markdown formatting typo (#289)
cc345ad   feat: add prerelease lifecycle script hook (closes #217) (#234)
```

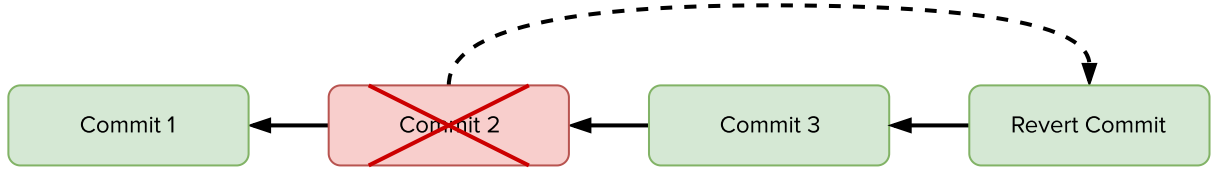
# Как исправить в коммит

Команда `git commit --amend` позволяет исправить последний коммит (изменить сообщение или добавить файлы):

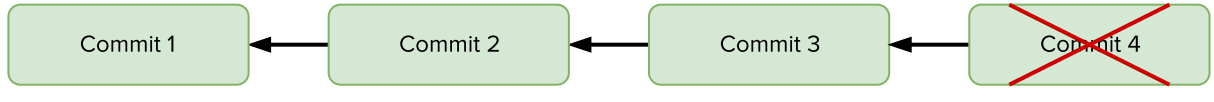
```
~/src/repo [master*]$ git commit -m "bad commit message"

[master 3122367] bad commit message
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 n.txt

~/src/repo [master]$ git commit --amend -m "good commmit message"

[master 6ec9633] good commmit message
 Date: Wed Apr 26 02:29:13 2017 +0300
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 n.txt
```

# Как отменить коммит

- Команда `git revert` делает новый (обратный) коммит. Не меняет истории - безопасный способ откатить изменения. Стоит использовать для публичных коммитов.

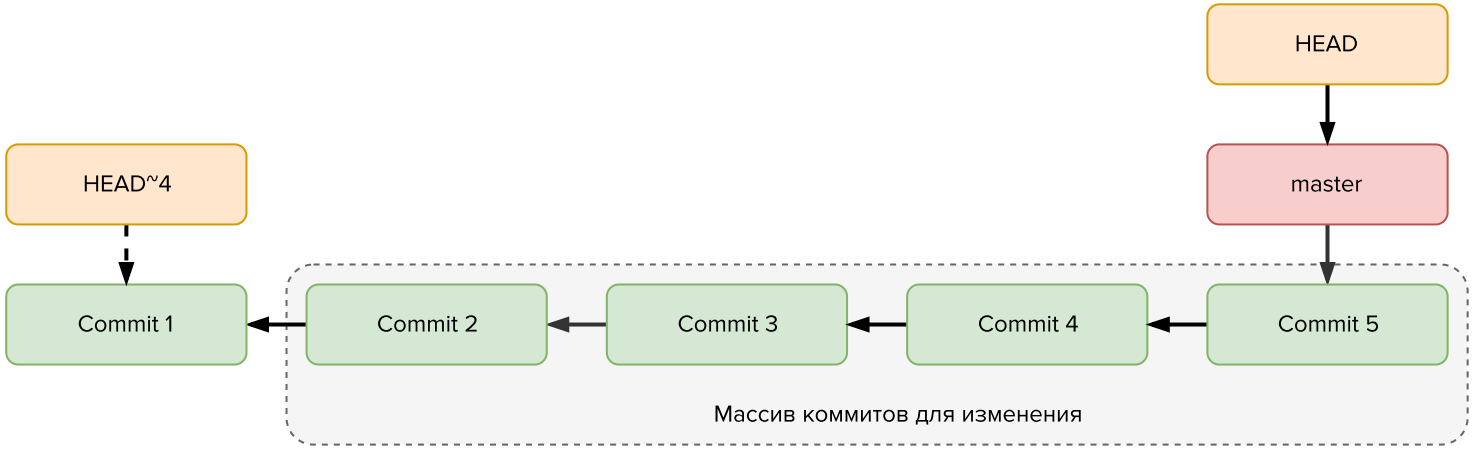



- Команда `git reset HEAD~1` удаляет коммиты после указанного. Меняет историю - небезопасный способ откатить изменения. Можно использовать для локальных коммитов.



# Работа с несколькими коммитами

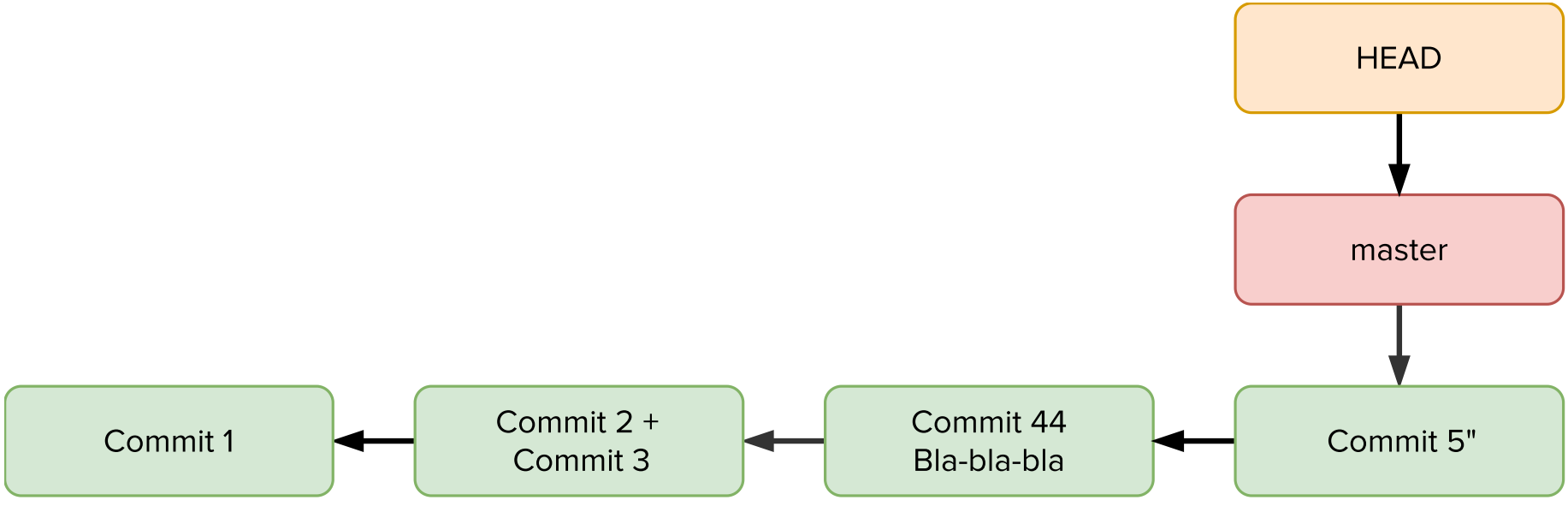
Пока мы не опубликовали наши изменения, можно "переписывать историю" с помощью **rebase**, например так: `git rebase -i HEAD~4`



# Работа с несколькими коммитами

```
git rebase -i HEAD~4
```

```
pick ff80b7e Commit2
squash c16543f Commit3
reword f54fabc Commit4
pick f527ec6 Commit5

# Rebase 316a5e1..f527ec6 onto 316a5e1 (4 commands)
```



# Поиск по сообщению

Команда `git log --grep=XXX` позволяет выполнять поиск по сообщениям в коммитах:

```
$ git log --grep="#24283"

commit 9771e4626255b9dd92ed7a9e60d4dd5ca332d855
Author: Artem Starostenko <artemstarostenko65@gmail.com>
Date:   Wed Apr 26 01:55:58 2017 +0300

    Wrong validation introduced in r15989 (#24283)

commit bd1fab27578a616587c70935740b2a54f7385f9b
Author: Artem Starostenko <artemstarostenko65@gmail.com>
Date:   Wed Apr 26 01:54:36 2017 +0300

    Add length validations for string fields (#24283)
```

# Ветки

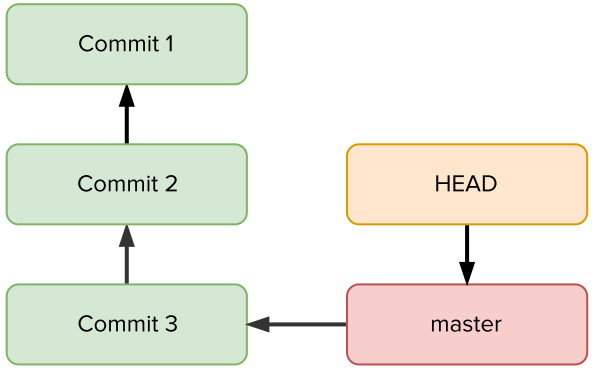

Сокольническая
Замоскворецкая
Арбатско-Покровская
Филёвская
Кольцевая
Калужско-Рижская
Таганско-Краснопресненская
Калининская
Солнцевская
Серпуховско-Тимирязевская
Люблинско-Дмитровская
Большая кольцевая
Каховская
Бутовская
Монорельс (работает в экскурсионном режиме)
Московское центральное кольцо
Станции пересадок
Кросс-платформенные пересадки
Наземные пересадки
Перехватывающие парковки метро

# Ветки

- Подвижный указатель на коммит
- Переключение по веткам осуществляется за счет другого подвижного указателя: **HEAD** (указывает на текущий коммит)
- Текущий указатель автоматом сдвигается после каждого нового коммита

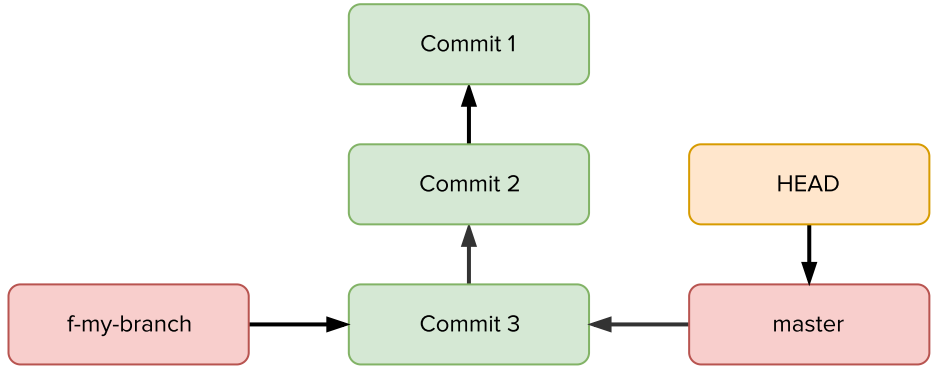
# Ветка master

По умолчанию создается ветка `master`:



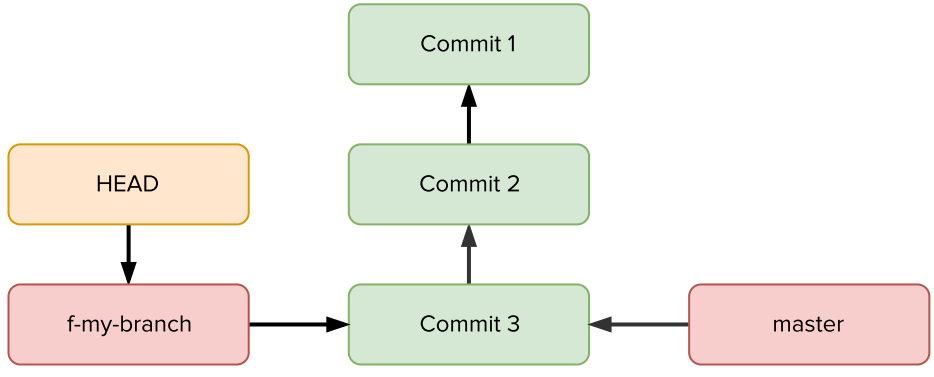
# Ветка feature

Результат создания ветки командой `git branch f-my-branch`:



# Git Checkout

Указатель **HEAD** можно переместить командой `git checkout f-my-branch`:



# Работа с ветками

- Создать ветку:

```
git branch <branch_name>
```

- Переключиться на ветку

```
git checkout <branch_name>
```

- Создать ветку и сразу перейти в нее:

```
git checkout -b <branch_name>
```

# Текущая ветка

Текущая положение указателя **HEAD** можно посмотреть в файле `.git/HEAD`:

```
~/src/repo [new-feature]$ cat .git/HEAD
ref: refs/heads/new-feature
```

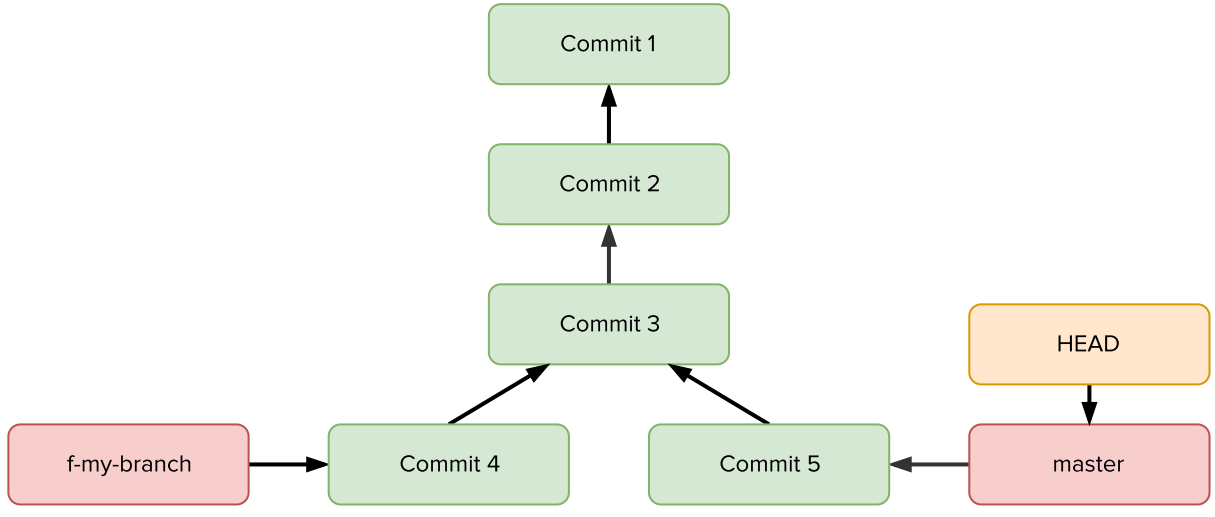
# Ссылки на локальные ветки

В каталоге `.git/refs/heads` перечислены локальные ветки и ссылки на коммиты в них:

```
~/src/repo [new-feature]$ ls .git/refs/heads
master      new-feature

~/src/repo [new-feature]$ cat .git/refs/heads/new-feature
6ec96337ebf165511636f42643234084535748ce

~/src/repo [new-feature]$ git branch
  master
* new-feature
```

Что скрывает от нас директория `.git`

# Добавим коммиты в обе ветки

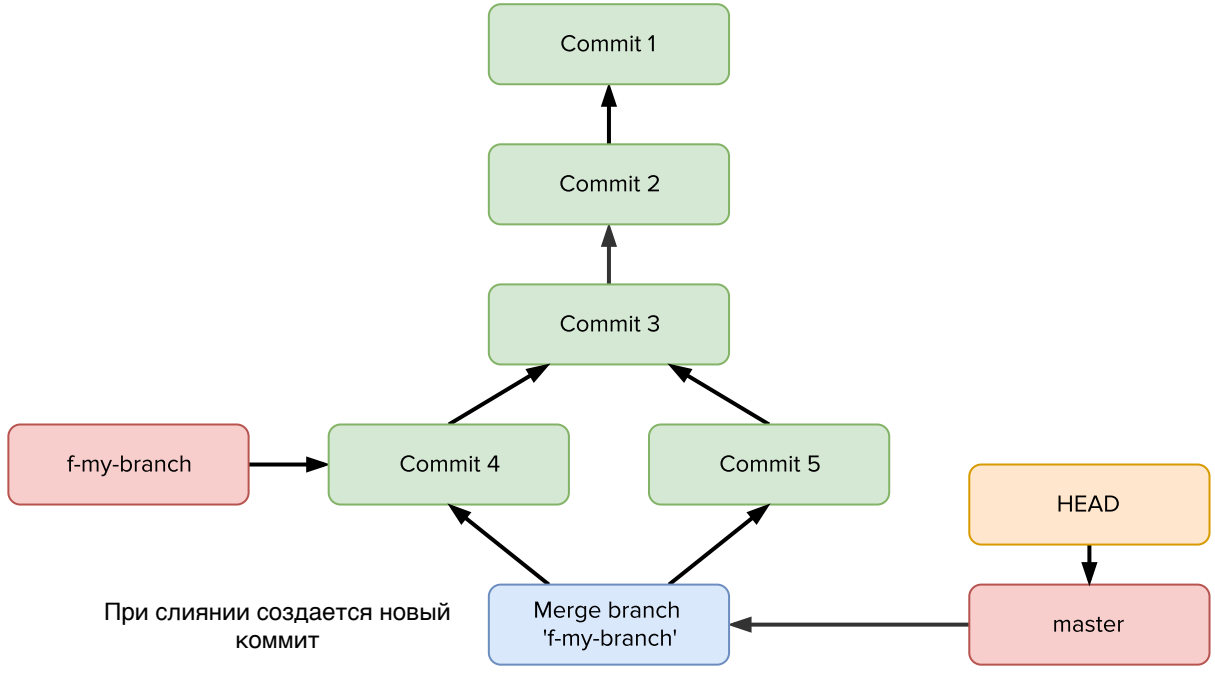



То же самое, но в ASCII:

```
~/src/repo [new-feature]$ git log --graph --abbrev-commit --decorate --all --oneline
* ea16714 (HEAD -> master) Commit5
| * f231481 (feature) Commit4
|/
* 0f72b9b Commit3
* ed0a3de Commit2
* 64a9ac9 Commit1
```

# Git Merge | Перенос изменений

Добавляем изменения из одной ветки в другую:



# Git Merge | CLI

- Переходим на ветку `master`:

```
$ git checkout master
```

- Сливаем ветку `f-my-branch` с веткой `master`:

```
$ git merge f-my-branch
```

```
*   7783eea (HEAD -> master) Merge branch 'f-my-branch'
|\
| * f231481 (f-my-branch) Commit4
* | ea16714 Commit5
|/
* 0f72b9b Commit3
* ed0a3de Commit2
* 64a9ac9 Commit1
```

# Git Merge | Конфликты

- В разных ветках поменяли один и тот же файл в одном и том же месте
- При слиянии неясно, какие изменения применять, и Git не может автоматически объединить изменения
- Git помечает в файлах "спорные места" и эти конфликты необходимо устранить вручную

# Git Merge | Конфликты

## `git merge f-my-branch`:

```
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

## `git status`

```
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

# Git Merge | Конфликты

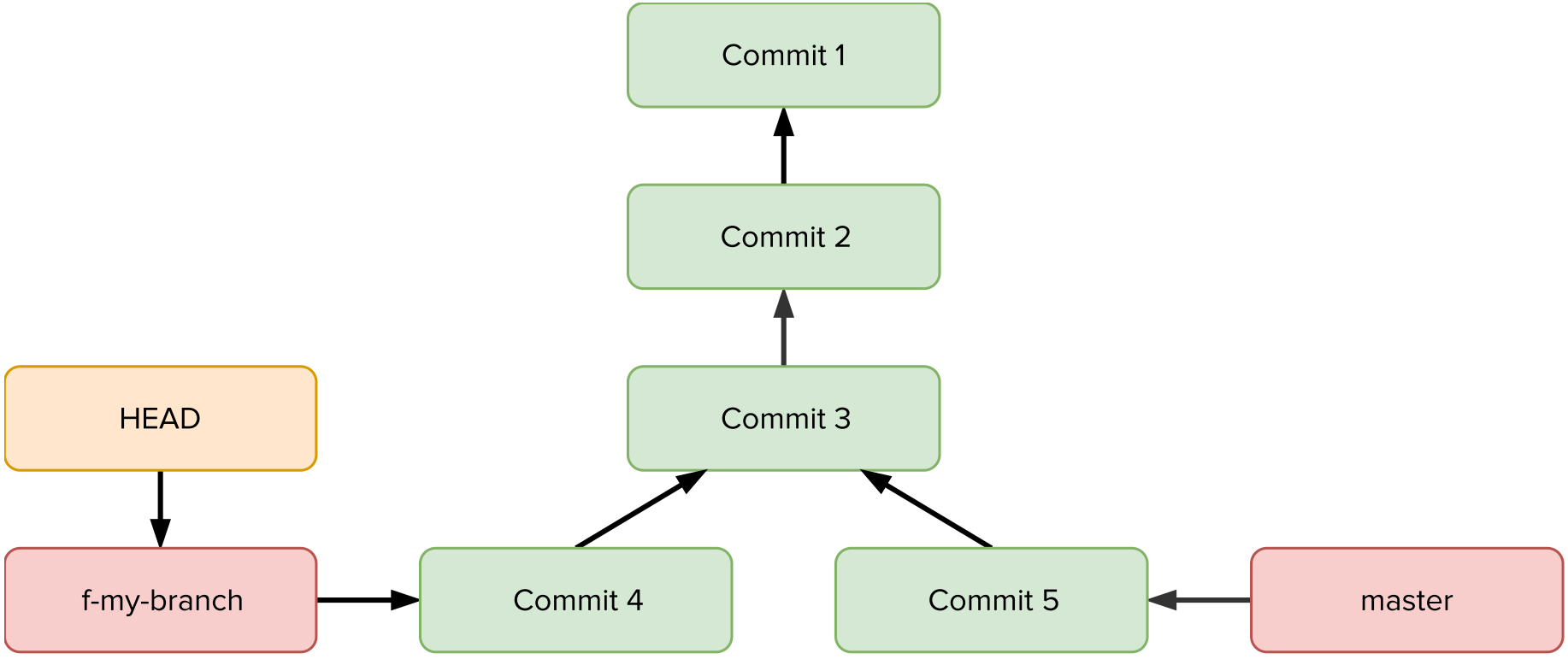
```
$ cat file.txt

коммит 1
коммит 2
<<<<<<< HEAD
коммит 4 (Я - коммит из ветки 'master')
=======
коммит 3 (А я - коммит из ветки 'f-my-branch')
>>>>>>> f-my-branch

$ vim file.txt

коммит 1
коммит 2
коммит 3
коммит 4

$ git add file.txt

$ git commit -m “Merge branch feature”
```

# git rebase

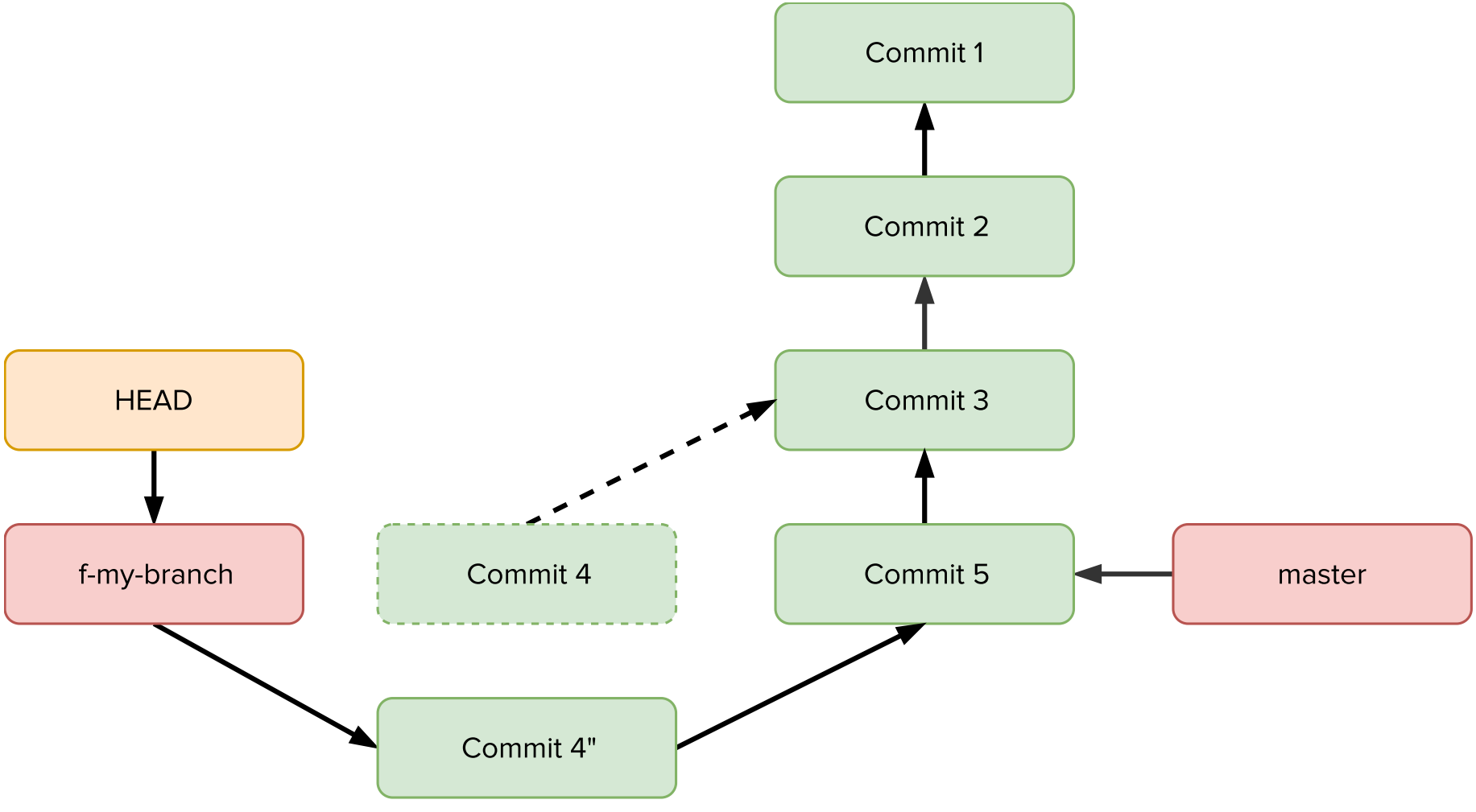
Команда `git rebase` меняет основание текущей ветки и позволяет "исправлять историю".

- Переходим на ветку **f-my-branch**: `git checkout f-my-branch`
- Меняем основание ветки **f-my-branch**: `git rebase master`

# До rebase


Commit 1
Commit 2
Commit 3
HEAD
f-my-branch
Commit 4
Commit 5
master

# После git rebase master



# Git Rebase CLI

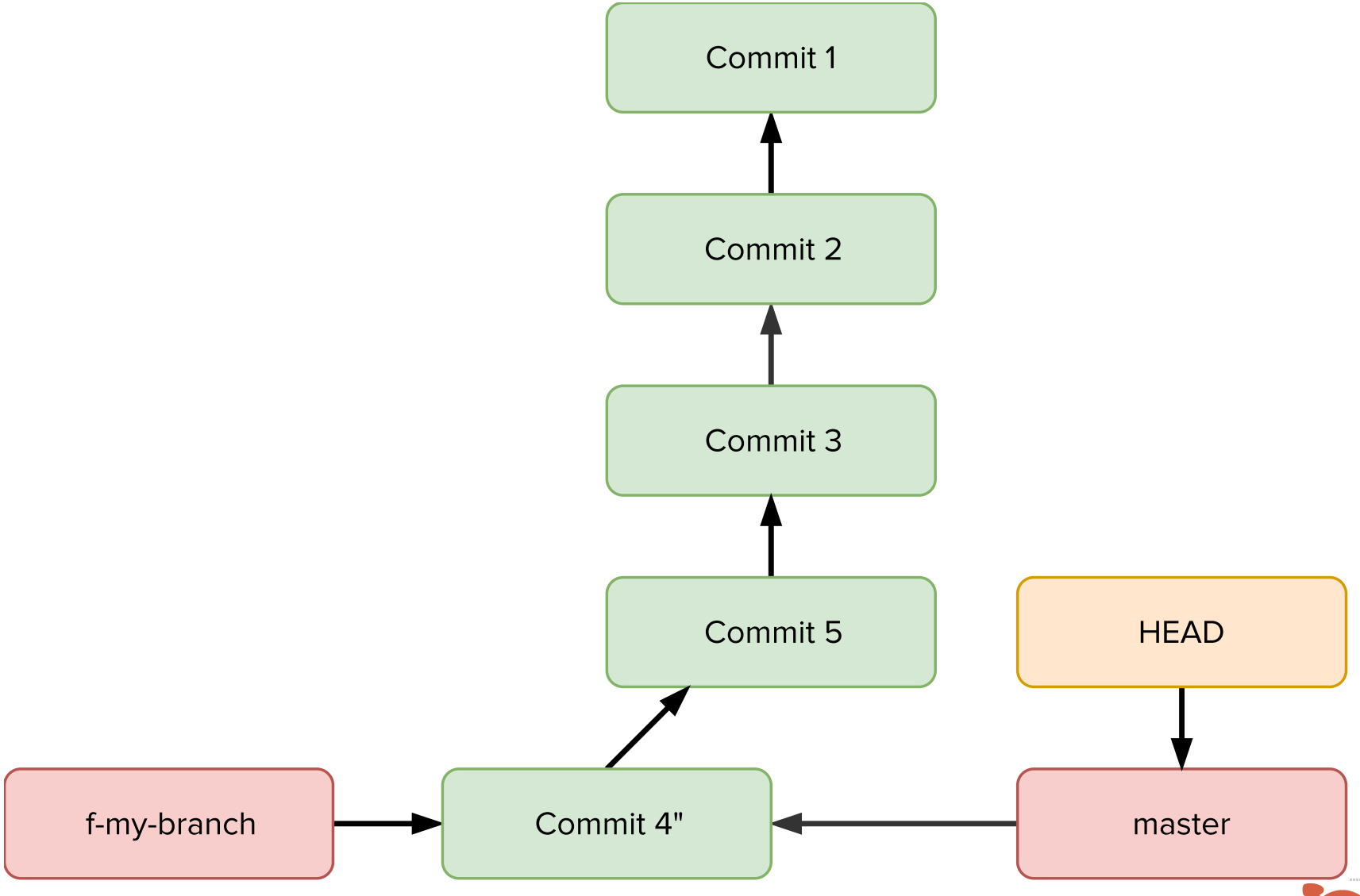
- До `rebase`:

```
* ea16714 (master) Commit5
| * f231481 (HEAD -> f-my-branch) Commit4
|/
* 0f72b9b Commit3
* ed0a3de Commit2
* 64a9ac9 Commit1
```

- После `rebase`:

```
* 34bf60f (HEAD -> f-my-branch) Commit4
* ea16714 (master) Commit5
* 0f72b9b Commit3
* ed0a3de Commit2
* 64a9ac9 Commit1
```

# Fast-forward merge



# Fast-forward merge

- До `git merge f-my-branch`

```
* 34bf60f (f-my-branch) Commit4
* ea16714 (HEAD -> master) Commit5
* 0f72b9b Commit3
* ed0a3de Commit2
* 64a9ac9 Commit1
```

- После `git merge f-my-branch`

```
* 34bf60f (HEAD -> master, f-my-branch) Commit4
* ea16714 Commit5
* 0f72b9b Commit3
* ed0a3de Commit2
* 64a9ac9 Commit1
```

# Опасность ребейза

- Если ветка опубликована - вы ломаете историю
- Если вы разрешили конфликты, то они могут появиться снова

# git show

Команда `git show` показывает определенный коммит:

- Коммит, на который указывает *master*:

```
git show master
```

- Родитель того коммита:

```
git show master^
```

- Дедушка того коммита :)

```
git show master~2
```

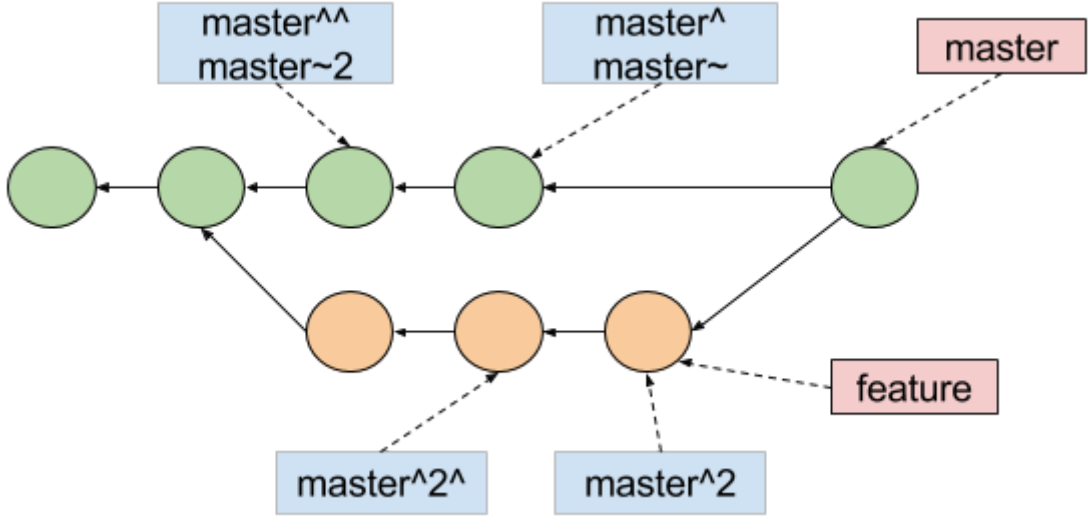
# git show

```
$ git show

commit 99380bf7f49698b87a2be8276f42ef0fc7c30b10
Author: Ivan Evtukhovich <evtuhovich@gmail.com>
Date:   Mon Nov 21 15:31:03 2016 +0300

    Commit 5

diff --git a/file.txt b/file.txt
index 39a0a54..601a67f 100644
--- a/file.txt
+++ b/file.txt
@@ -1,3 +1,3 @@
-Ваня шел гулять на речку
+Знайка шел гулять на речку
 Перепрыгнул через овечку
 А овечка была вредна
```

# git show branch^

```
$ git show branch^

commit 10b00d091e44f2c42fcdfc11efa1a58b26628729
Author: Ivan Evtukhovich <evtuhovich@gmail.com>
Date:   Mon Nov 21 15:29:28 2016 +0300

    Commit 3

diff --git a/file.txt b/file.txt
index 2608c04..39a0a54 100644
--- a/file.txt
+++ b/file.txt
@@ -1,2 +1,3 @@
 Ваня шел гулять на речку
 Перепрыгнул через овечку
+А овечка была вредна
```

# Спецификаторы коммитов



# Смотрим и сравниваем изменения

Без него, как без рук:

- Непроиндексированные изменения - `git diff`
- Изменения, которые добавили в индекс - `git diff --cached`
- Сравнить два коммита - `git diff branch..master`
- Сравнить с текущим коммитом - `git diff branch^..`

# Смотрим историю коммитов

- Показывает коммиты одной ветки - `git log`
- Коммиты всех веток - `git log --all`
- Показывает дельту изменений - `git log -p`
- Рисует граф ветвлений - `git log --graph --all`
- Выводит коммиты в одну строчку - `git log --oneline`

# Аннотация файла

Команда `git blame` показывает, какие коммиты меняли строки файла:

```
$ git blame file.txt

99380bf7 (Ivan Evtukhovich  2016-11-21 15:31:03 +0300 1) Знайка шел гулять на речку
4a53e7f0 (Artem Starostenko 2016-11-21 15:29:16 +0300 2) Перепрыгнул через овечку
10b00d09 (Yuri Ignatov      2016-11-21 15:29:28 +0300 3) А овечка была вредна
1353ea98 (Ivan Evtukhovich  2016-11-21 17:20:51 +0300 4) И купила себе вина.
```

# Удаленные репо

Склонировать удаленный репозиторий на локальную машину:

```
$ git clone https://github.com/express42/zabbixapi.git
```

Посмотреть информацию об удаленных репозиториях

```
$ git remote -v

origin       https://github.com/express42/zabbixapi.git (fetch)
origin        https://github.com/express42/zabbixapi.git (push)
```

# Git Remotes

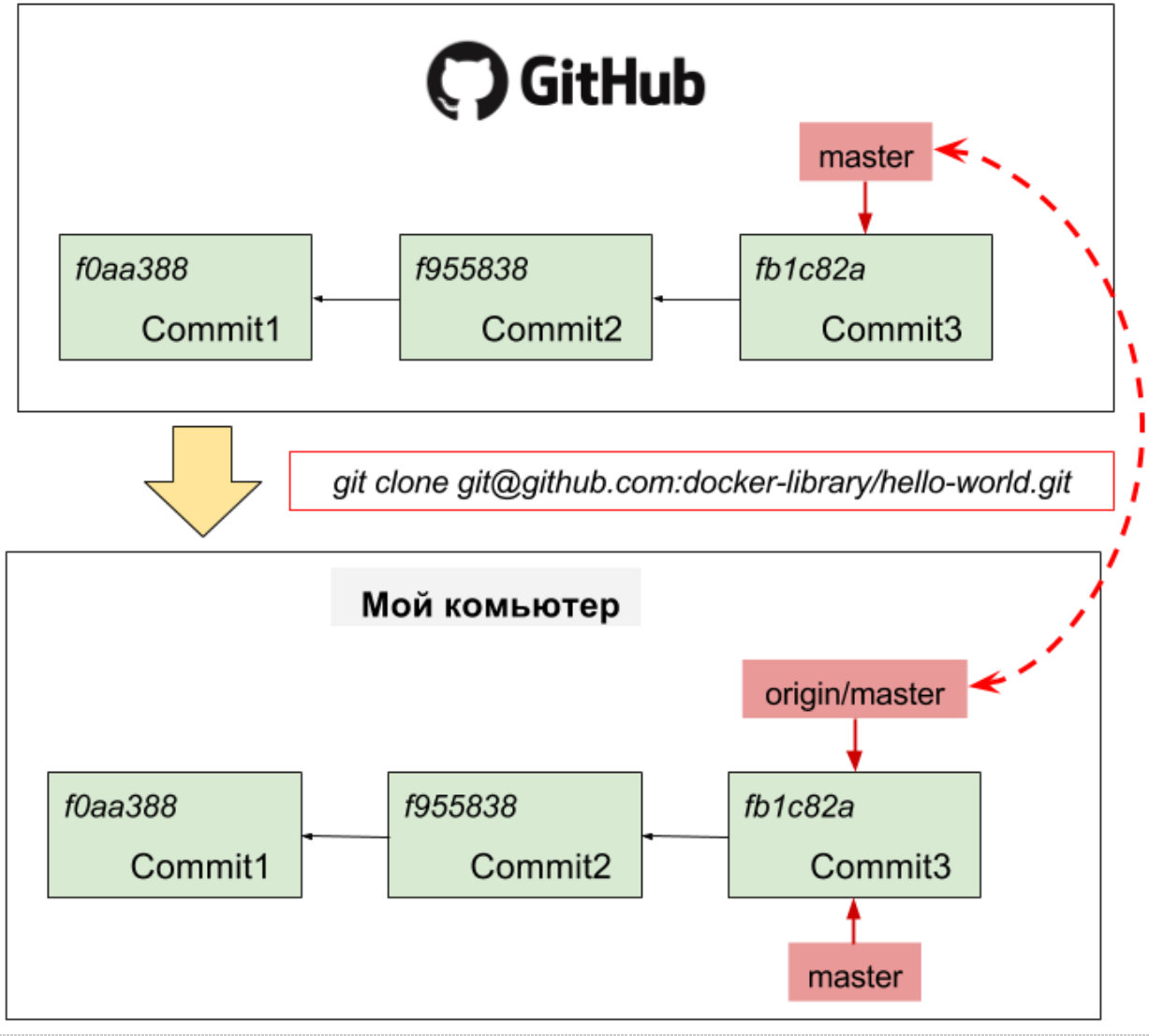
```
user@laptop:~/zabbixapi [master]$ git remote show origin

* remote origin
  Fetch URL: https://github.com/express42/zabbixapi.git
  Push  URL: https://github.com/express42/zabbixapi.git
  HEAD branch: master
  Remote branches:
    master    tracked
    zabbix1.8 tracked
    zabbix2.0 tracked
    zabbix2.2 tracked
    zabbix2.4 tracked
    zabbix3.0 tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

# git fetch

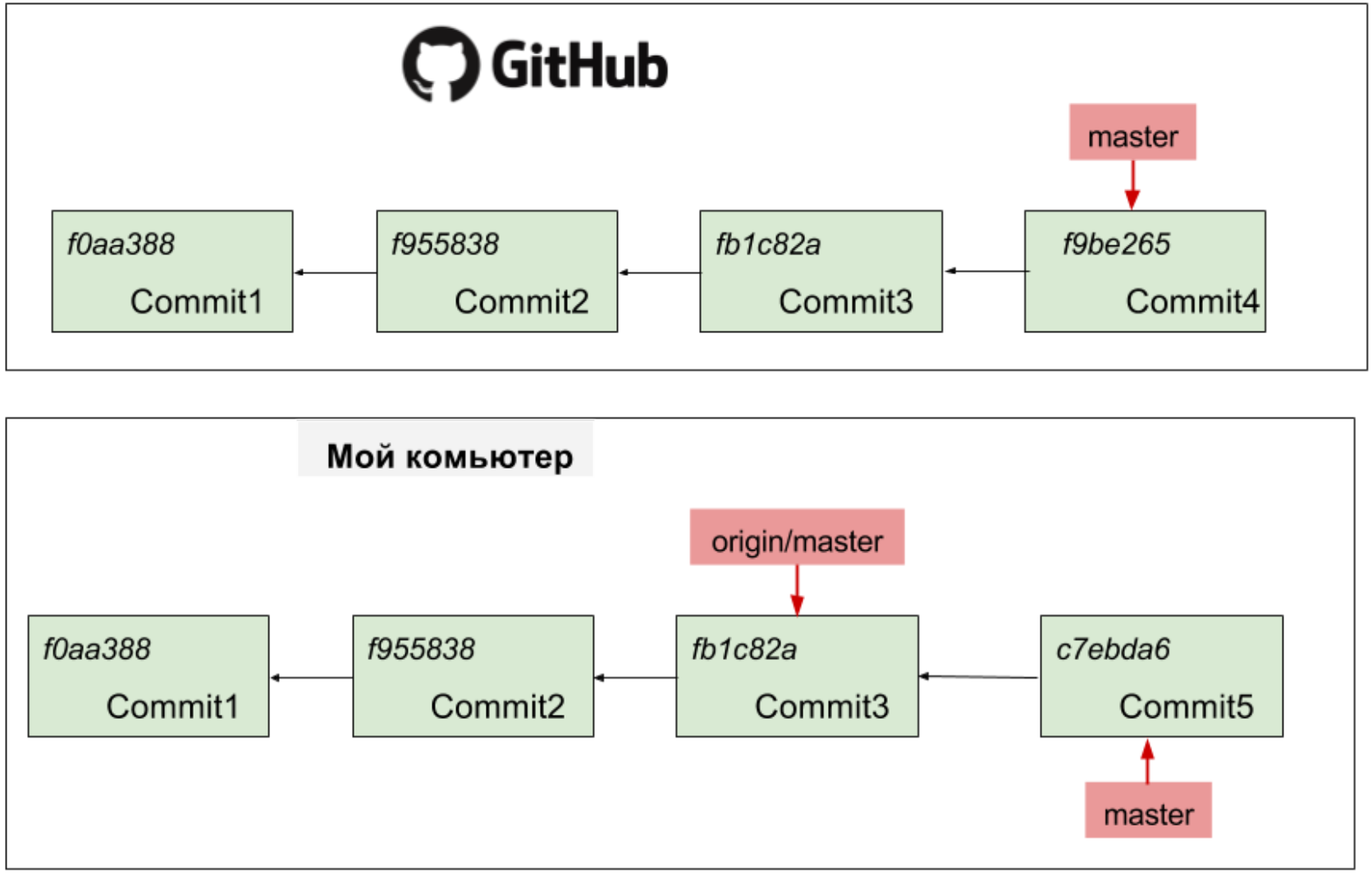
Забирает данные с удаленного репо, но не применяет их:

```
$ git fetch -a
$ git fetch <remotename>
```

# git fetch | Clone



# git fetch | Local changes



# Состояние локального репо

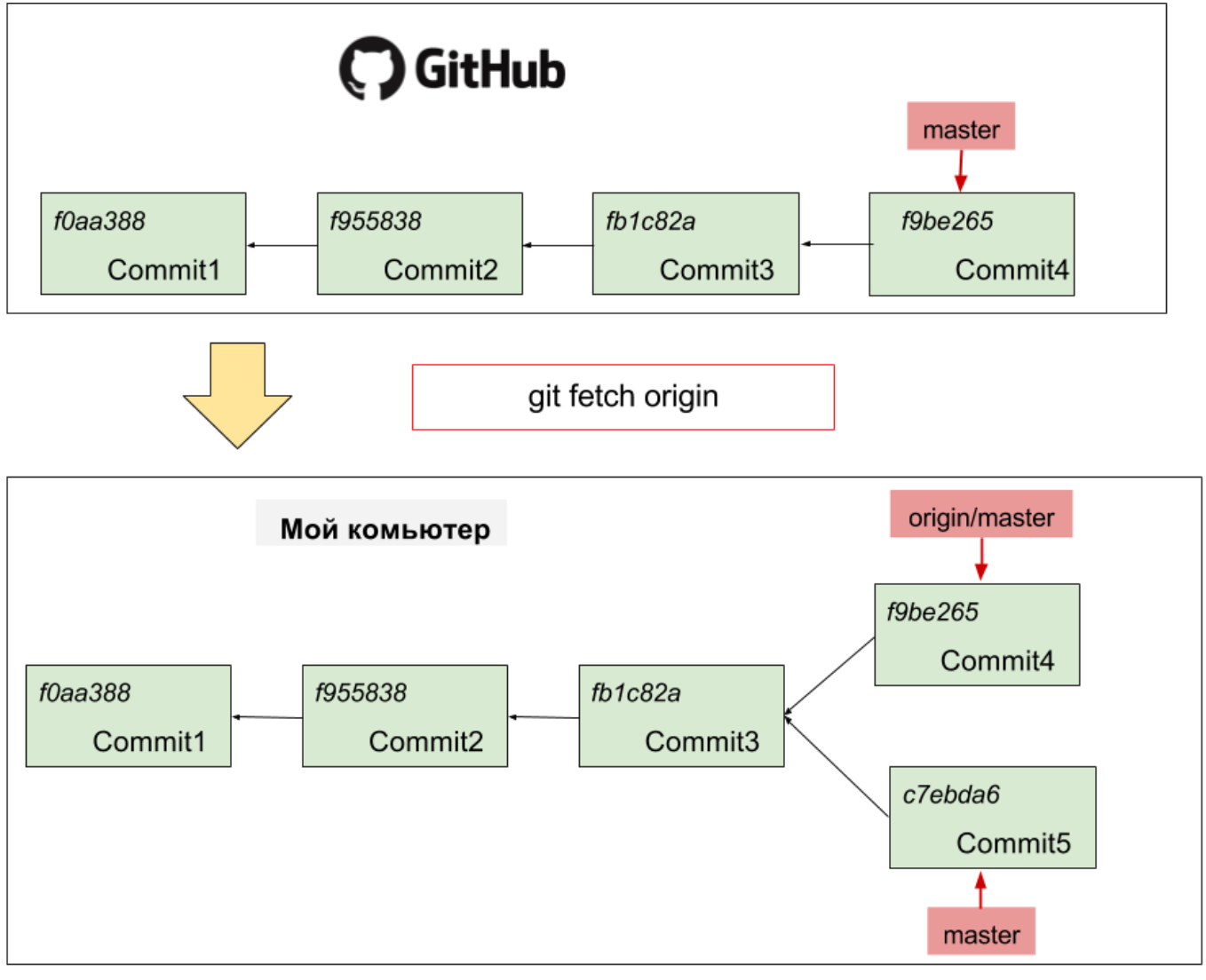
- `git log –graph –abbrev–commit –decorate –all –oneline`

```
* c7ebda6 (HEAD -> master) Commit5
* fb1c82a (origin/master) Commit3
* f955838 Commit2
* f0aa388 Commit1
```

- `git status`

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean
```

# git fetch | Fetch



# git fetch

- `git log –graph –abbrev–commit –decorate –all –oneline`

```
* c7ebda6 (HEAD -> master) Commit5
| * f9be265 (origin/master) Commit4
|/
* fb1c82a Commit3
* f955838 Commit2
* f0aa388 Commit1
```

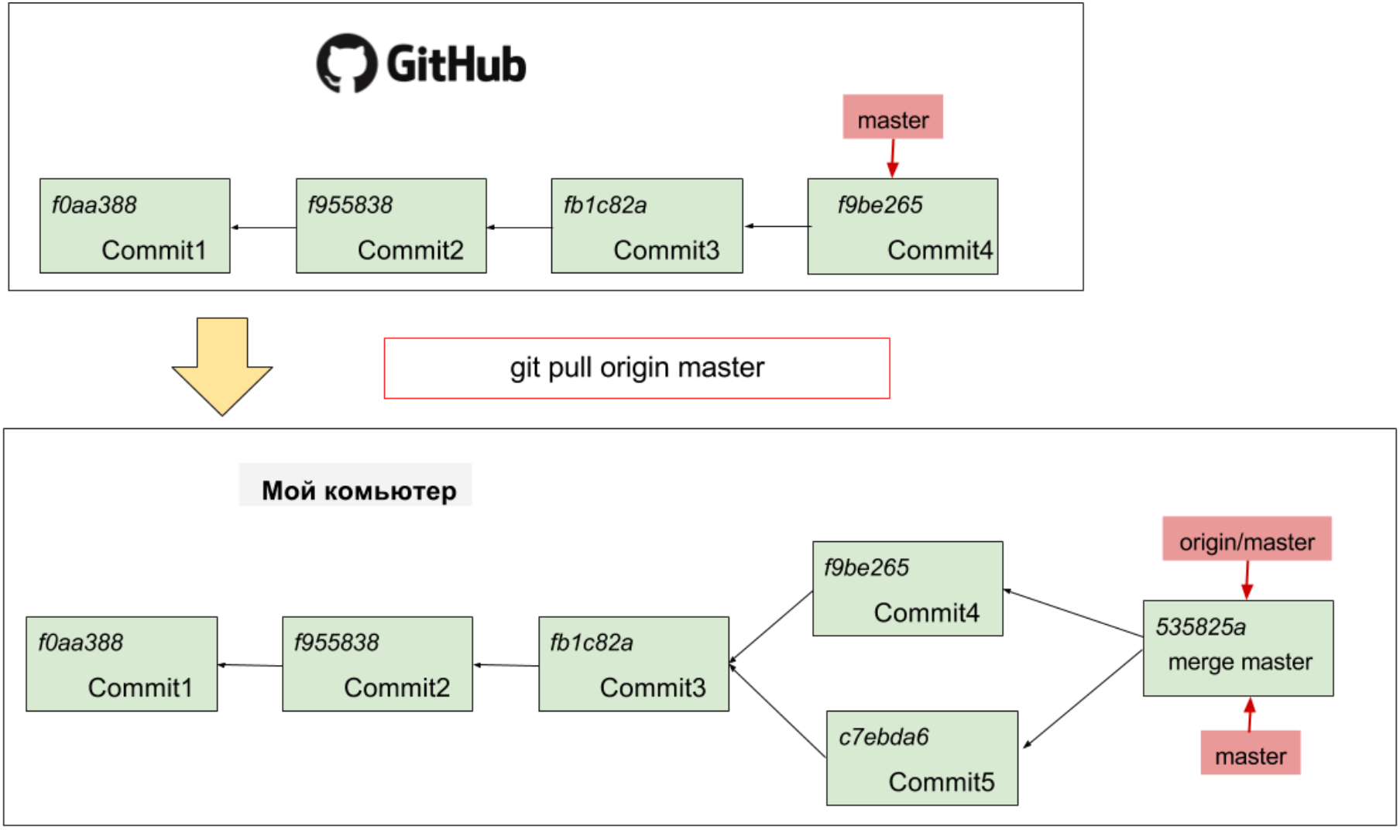
- `git status`

```
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
nothing to commit, working tree clean
```

# git pull

- `git fetch` + `git merge`
- Забирает данные с удаленного репо
- `git pull`
- `git pull origin master`
- `git pull --rebase` = `git fetch` + `git rebase`

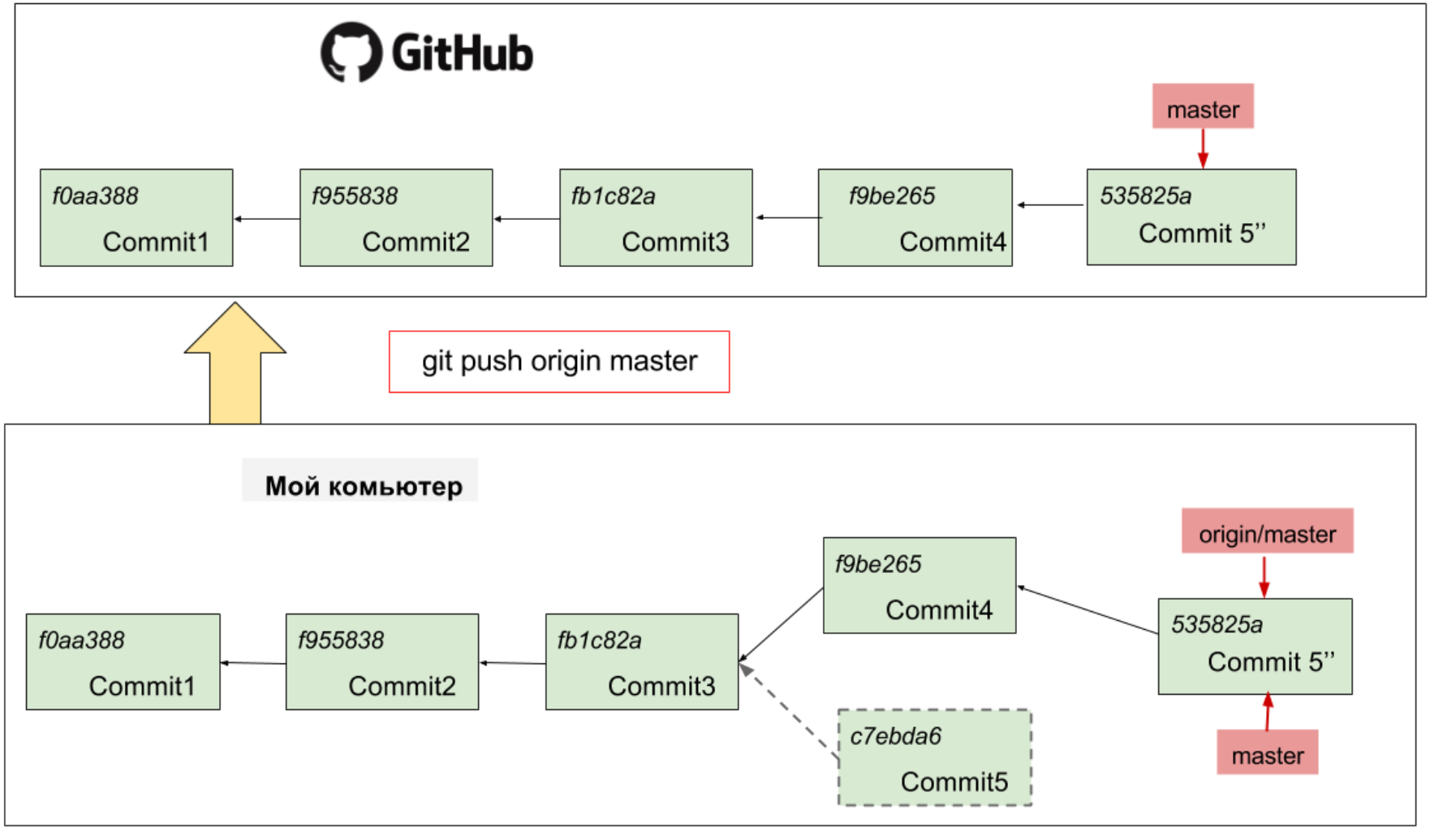
# git pull


GitHub
master
f0aa388
Commit1
f955838
Commit2
fb1c82a
Commit3
f9be265
Commit4
git pull origin master
Мой комьютер
origin/master
f9be265
Commit4
f0aa388
Commit1
f955838
Commit2
fb1c82a
Commit3
535825a
merge master
c7ebda6
Commit5
master

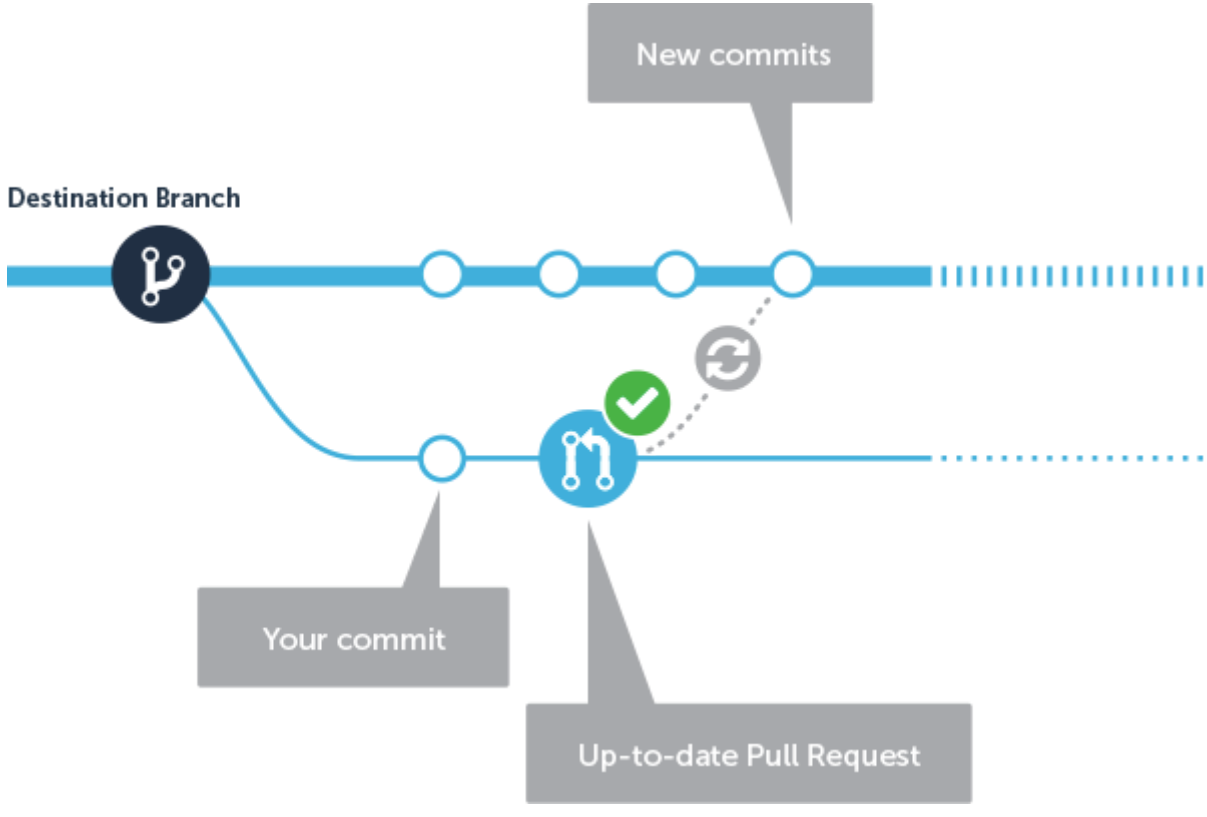
# git push

- Делимся изменениями с другими
- Обновляет удаленную ветку в соответствии в локальной
- `git push`
- `git push origin master`

# git push


GitHub
master
f0aa388
Commit1
f955838
Commit2
fb1c82a
Commit3
f9be265
Commit4
535825a
Commit 5''
git push origin master
Мой комьютер
origin/master
f9be265
Commit4
f0aa388
Commit1
f955838
Commit2
fb1c82a
Commit3
535825a
Commit 5''
c7ebda6
Commit5
master

# Pull request



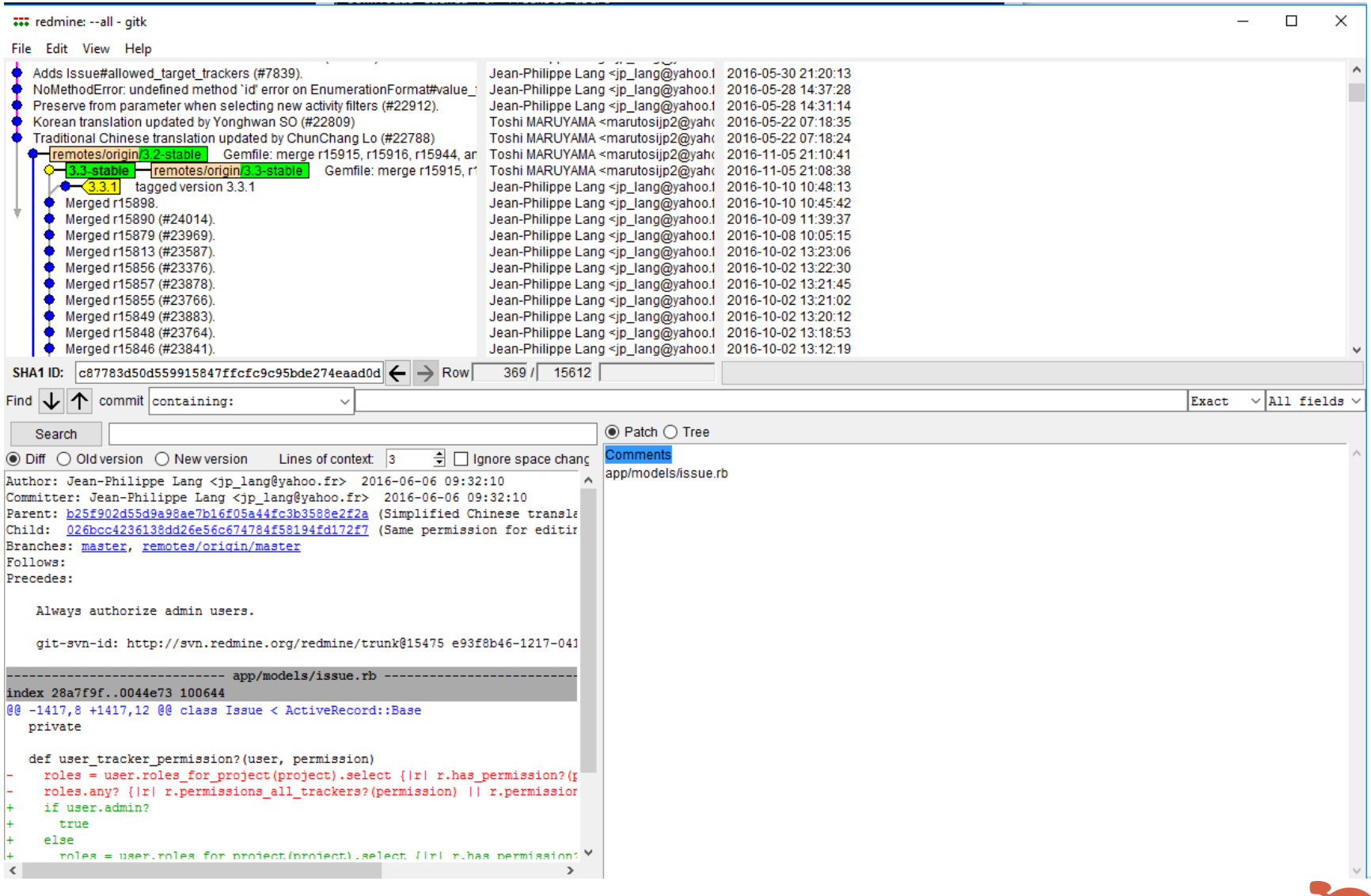
# Peer review

- Хотя бы две пары глаз на каждое изменение
- Коллективное владение кодом
- Удобно делать через pull-requests
- Сложна не технически, а эмоционально
- Имеет смысл обозначить правила

# git help -a

```
add                         credential-store    index-pack          patch-id            shortlog
add--interactive            cvsexportcommit     init                prune               show
am                          cvsimport           init-db             prune-packed        show-branch
annotate                    cvsserver           instaweb            pull                show-index
apply                       daemon              interpret-trailers  push                show-ref
archimport                  describe            log                 quiltimport         stage
archive                     diff                ls-files            read-tree           stash
bisect                      diff-files          ls-remote           rebase              status
bisect--helper              diff-index          ls-tree             receive-pack        stripspace
blame                       diff-tree           mailinfo            reflog              submodule
branch                      difftool            mailsplit           relink              svn
bundle                      difftool--helper    merge               remote              symbolic-ref
cat-file                    fast-export         merge-base          remote-ext          tag
check-attr                  fast-import         merge-file          remote-fd           unpack-file
check-ignore                fetch               merge-index         remote-ftp          unpack-objects
check-mailmap               fetch-pack          merge-octopus       remote-ftps         update-index
check-ref-format            filter-branch       merge-one-file      remote-http         update-ref
checkout                    fmt-merge-msg       merge-ours          remote-https        update-server-info
checkout-index              for-each-ref        merge-recursive     remote-testsvn      upload-archive
cherry                      format-patch        merge-resolve       repack              upload-pack
cherry-pick                 fsck                merge-subtree       replace             var
citool                      fsck-objects        merge-tree          request-pull        verify-commit
clean                       gc                  mergetool           rerere              verify-pack
clone                       get-tar-commit-id   mktag               reset               verify-tag
column                      grep                mktree              rev-list            web--browse
commit                      gui                 mv                  rev-parse           whatchanged
commit-tree                 gui--askpass        name-rev            revert              worktree
config                      hash-object         notes               rm                  write-tree
count-objects               help                p4                  send-email
credential                  http-backend        pack-objects        send-pack
credential-cache            http-fetch          pack-redundant      sh-i18n--envsubst
credential-cache--daemon    http-push           pack-refs           shell
```

# gitk


redmine: --all - gitk
File Edit View Help
Adds Issue#allowed_target_trackers (#7839).
NoMethodError: undefined method `id' error on EnumerationFormat#value_
Preserve from parameter when selecting new activity filters (#22912).
Korean translation updated by Yonghwan SO (#22809)
Traditional Chinese translation updated by ChunChang Lo (#22788)
remotes/origin/3.2-stable
Gemfile: merge r15915, r15916, r15944, ar
3.3-stable
remotes/origin/3.3-stable
Gemfile: merge r15915, r1
3.3.1
tagged version 3.3.1
Merged r15898.
Merged r15890 (#24014).
Merged r15879 (#23969).
Merged r15813 (#23587).
Merged r15856 (#23376).
Merged r15857 (#23878).
Merged r15855 (#23766).
Merged r15849 (#23883).
Merged r15848 (#23764).
Merged r15846 (#23841).
Jean-Philippe Lang <jp_lang@yahoo.f 2016-05-30 21:20:13
Jean-Philippe Lang <jp_lang@yahoo.f 2016-05-28 14:37:28
Jean-Philippe Lang <jp_lang@yahoo.f 2016-05-28 14:31:14
Toshi MARUYAMA <marutosijp2@yah 2016-05-22 07:18:35
Toshi MARUYAMA <marutosijp2@yah 2016-05-22 07:18:24
Toshi MARUYAMA <marutosijp2@yah 2016-11-05 21:10:41
Toshi MARUYAMA <marutosijp2@yah 2016-11-05 21:08:38
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-10 10:48:13
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-10 10:45:42
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-09 11:39:37
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-08 10:05:15
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:23:06
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:22:30
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:21:45
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:21:02
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:20:12
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:18:53
Jean-Philippe Lang <jp_lang@yahoo.f 2016-10-02 13:12:19
SHA1 ID: c87783d50d559915847ffcfc9c95bde274eaad0d
Row 369 / 15612
Find commit containing:
Exact
All fields
Search
Patch Tree
Diff Old version New version Lines of context: 3 Ignore space chang
Comments
app/models/issue.rb
Author: Jean-Philippe Lang <jp_lang@yahoo.fr> 2016-06-06 09:32:10
Committer: Jean-Philippe Lang <jp_lang@yahoo.fr> 2016-06-06 09:32:10
Parent: b25f902d55d9a98ae7b16f05a44fc3b3588e2f2a (Simplified Chinese transla
Child: 026bcc4236138dd26e56c674784f58194fd172f7 (Same permission for editir
Branches: master, remotes/origin/master
Follows:
Precedes:
Always authorize admin users.
git-svn-id: http://svn.redmine.org/redmine/trunk@15475 e93f8b46-1217-041
---------- app/models/issue.rb ----------
index 28a7f9f..0044e73 100644
@@ -1417,8 +1417,12 @@ class Issue < ActiveRecord::Base
private
def user_tracker_permission?(user, permission)
- roles = user.roles_for_project(project).select {|r| r.has_permission?(p
- roles.any? {|r| r.permissions_all_trackers?(permission) || r.permissior
+ if user.admin?
+ true
+ else

# Что почитать

- Официальный сайт
- Книга Pro Git
- Серия лекций от Андрея Куманяева
  - Часть 1, Часть 2, Часть 3, Часть 4, Часть 5
- https://learngitbranching.js.org/