

Git

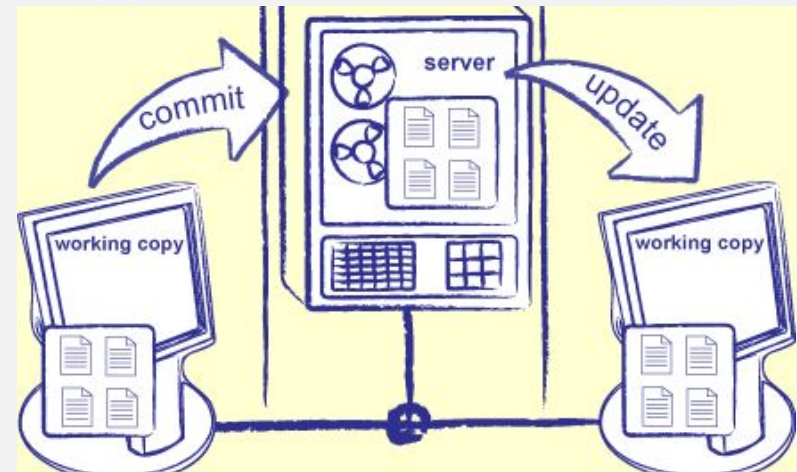
<https://github.com/kontur-courses/git>



Мотивация

Зачем разработчикам система контроля версий?

- Единое место **хранения кода**
- Удобное **объединение изменений** от разных разработчиков
- История изменений с **описанием и авторством**
- **Откат** неудачных изменений
- **Ревью** изменений



А зачем система
контроля версий вам?

Систем контроля версий море...

CVS

SVN

Fossil

TFS

Bazaar

Git

Perforce

Mercurial

Veracity

Git – популярная система контроля версий

Распределенная

- Каждому по репозиторию

Консольная

- Состоит из утилит командной строки
- Кроссплатформенная
- Много разных GUI

Поддерживается

- хостингами репозиториев: *GitHub, GitLab, BitBucket*
- популярными IDE: *Visual Studio, WebStorm, VS Code*





А кто уже пользовался Git?
А другой системой контроля версий?

Как будем изучать?

Как будем изучать?

GUIов много, на любой вкус и цвет,
каждый со своими особенностями,
а придется что-то выбрать



Как будем изучать?

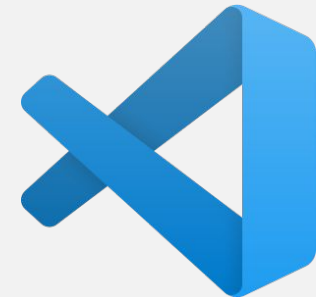
Примеры GUI на **Git Extensions**

- Тонкая настройка над консолью с минимумом магии
- Удобные команды, управление с клавиатуры
- Распространен в Контуре



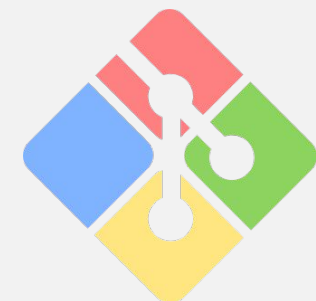
Примеры GUI на **Git Graph**

- Удобное расширение VS Code
- Можно просматривать историю не выходя из редактора



Примеры CLI на **Git Bash**

- Может все!



Как будем изучать?

Задания на **Git Bash + Git Graph**
для Linux, Mac или Windows

Задания на **Git Extensions**
для Windows



Как будем изучать?

VS Code для разрешения конфликтов

- Показывает конфликт он как есть
- Подсветка конфликта, кнопки быстрых действий
- Подсвечивает код



Как будем изучать?

VS Code для редактирования

- Подсвечивает Markdown
- Умеет открывать папку



Как будем изучать?

Особенностей и **нюансов много**, а **времени мало**

Если **освоить правила**,
в нюансах легко разобраться



Сформулируем **10 правил Git**
и связанные с ними команды

Как будем изучать?

Формат

1. Правило и теория к нему
2. Практические задания
3. Синхронизация



А потом много практики в реальной жизни,
чтобы довести до автоматизма 😊

Structure

Actions

Remote

S1. Все локально

A1. Трехсторонний merge
в три шага

R1. Доступен fetch коммитов
любого репозитория
в любой момент

S2. Хранятся
состояния директории,
постепенная сборка коммита

A2. rebase, cherry-pick и amend,
чтобы пересоздать историю

R2. Удаленное изменение
— это push

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

A3. stash, reset, revert
для управления изменениями

R3. Явное сопоставление
локальных веток
с upstream

H1. Гибкое конфигурирование и качественная документация

Н1. Гибкое конфигурирование и качественная документация

Гибкая настройка под любой процесс

Документация ко всем командам

Алиасы для краткости команд

Надо игнорировать все, кроме исходников

Конфигурируется под команду и продукт

У команд Git множество опций

Поведение команд по умолчанию можно настраивать

Настройки репозитория: `git config --local -e`

Настройки пользователя: `git config --global -e`

Настройки системы: `git config --system -e`

Минимальная Жизнеспособная Конфигурация

Git должен знать, кто вносит изменения:

```
git config --global user.name "John Doe"
```

```
git config --global user.email "johndoe@example.com"
```

Стоит задать текстовый редактор:

Windows: `git config --global core.editor notepad`

Linux, Mac: `git config --global core.editor nano`

Хотя бы для того, чтобы отредактировать остальные настройки!

По умолчанию — **vim** 😬

Git спешит на помощь!

```
git help commit
```

```
git commit --help
```

```
git commit -h
```

Чтобы не запоминать все опции команд!



Подсказки при неверных командах

```
$ git comit
git: 'comit' is not a git command. See 'git --help'.
The most similar command is
    commit
```

*Полезно читать вывод
хотя бы при освоении Git!*

```
$ git commit
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

```
$ git push
fatal: The current branch awesome-feature has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin awesome-feature
```

Стандарты опций

Полная запись:

```
git branch --delete awesome-feature
```

Короткая запись:

```
git branch -d awesome-feature
```

Полная усиленная запись:

```
git branch --delete --force awesome-feature
```

Короткая усиленная запись:

```
git branch -D awesome-feature
```

Алиасы для краткости команд

Через .gitconfig

```
[alias]
```

```
it = !git init && git commit -m 'Initial commit' --allow-empty
```

```
st = status -sb
```

```
commend = commit --amend --no-edit
```

```
graph = log --oneline --decorate --graph --all
```

Использование: git graph

Настройка для Windows

Через `.gitconfig`

```
[core]
```

```
autocrlf = true
```

```
safecrlf = true
```

Через команды

```
git config --global core.autocrlf true
```

```
git config --global core.safecrlf true
```

`autocrlf` – преобразование `\r\n` в `\n`

`safecrlf` – проверка обратимости преобразования `\r\n` в `\n`

Настройка для Linux и Mac

Через .gitconfig

```
[core]
```

```
autocrlf = input
```

Через команды

```
git config --global core.autocrlf input
```

Даже если получено `\r\n`, то преобразуется в `\n`

Сохранять в репозиторий нужно
только исходные файлы!
Остальное — в игнор!

Игнорирование файлов

В любой папке и ее подпапках
`.gitignore`

Через .gitconfig
[core]
 excludesFile

Дополнительно в репозитории
<repo>/.git/info/exclude

Коллекция .gitignore от GitHub: <https://github.com/github/gitignore>

Structure

Actions

Remote

H1. Гибкое конфигурирование и качественная документация

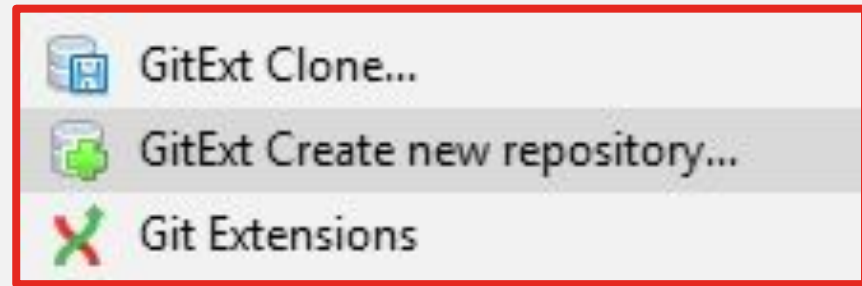
S1. Все локально

Все данные хранятся в локальных репозиториях,
изменения между ними можно синхронизировать

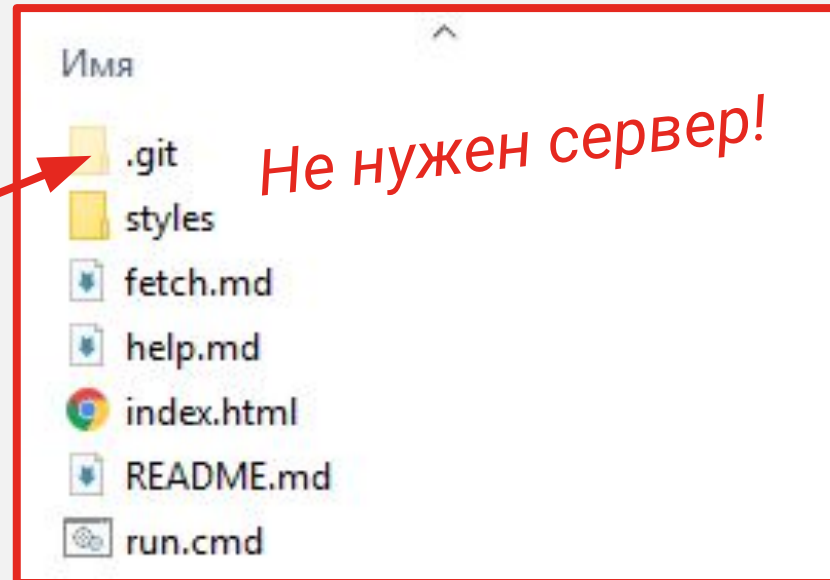
Репозиторий

Репозиторий – хранилище кода со всей историей изменений

`git init` – создать репозиторий для папки



Репозиторий



Не нужен сервер!

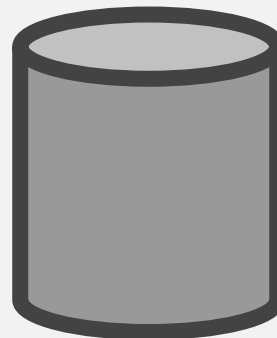
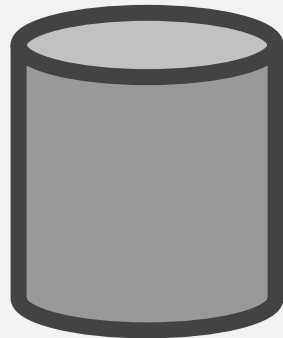
Рабочая директория

Клонирование

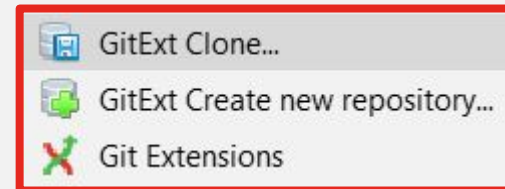
Чтобы работать над существующим проектом надо скопировать репозиторий локально – **склонировать**

`git clone <url>` – склонировать репозиторий

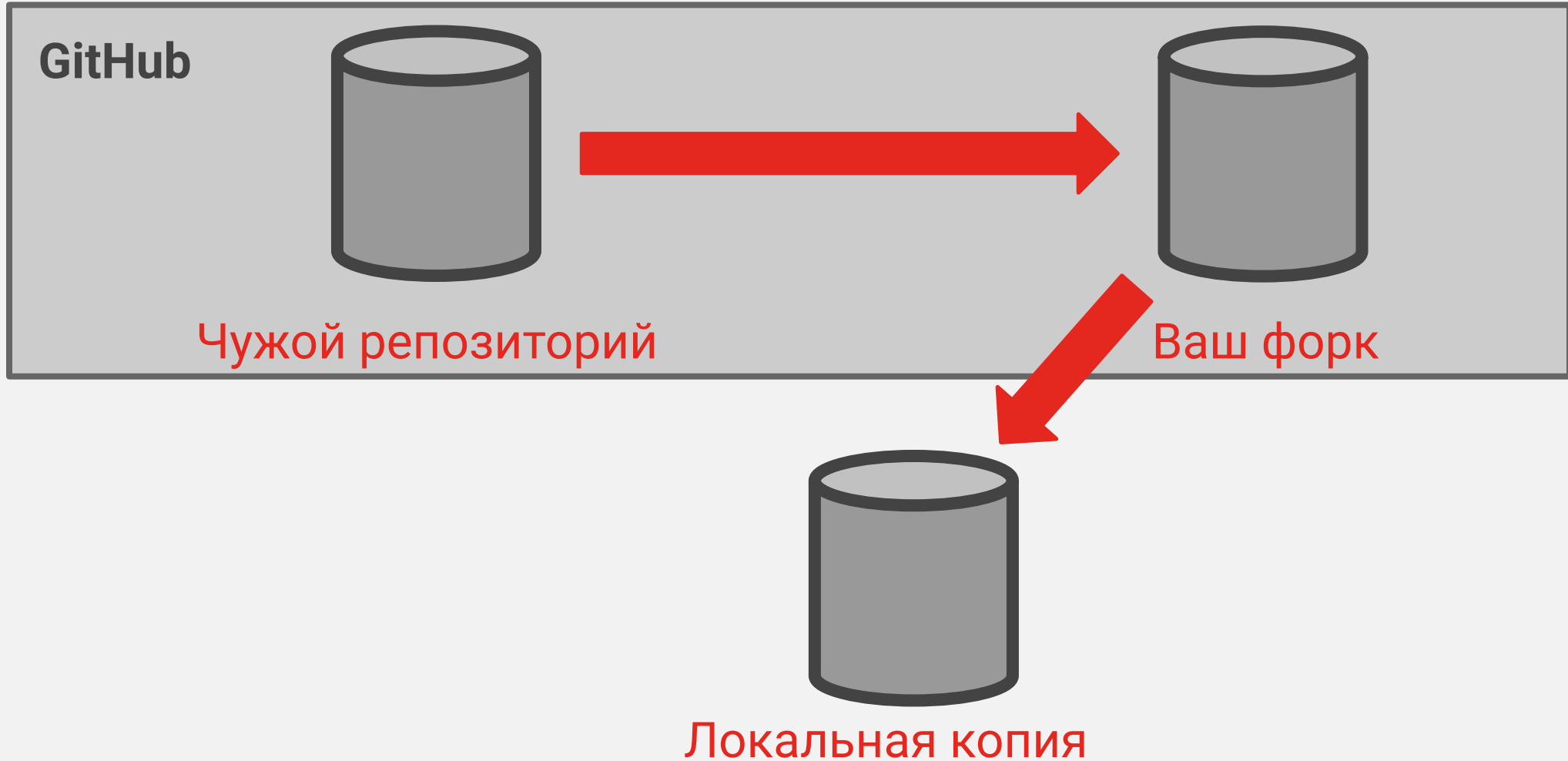
На сервере



Локальная копия

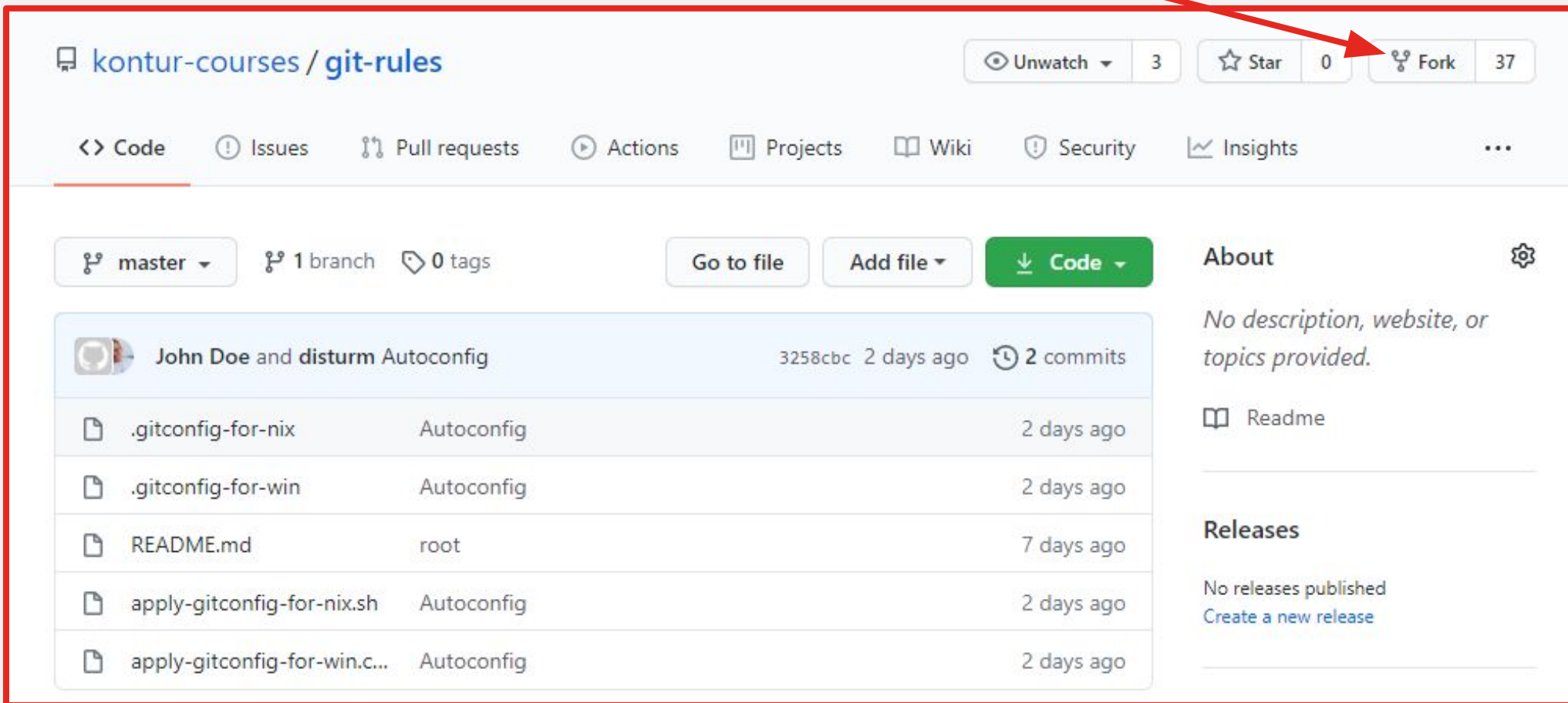


Fork на GitHub



Fork на GitHub

Сделать форк



The screenshot shows the GitHub interface for the repository 'kontur-courses / git-rules'. At the top right, there are buttons for 'Unwatch' (3), 'Star' (0), and 'Fork' (37). A red arrow points from the text 'Сделать форк' to the 'Fork' button. Below the repository name, there are navigation tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. The main content area shows the repository's structure with a table of files and a commit history. The 'About' section on the right indicates no description or topics are provided, and the 'Releases' section shows no releases published.

kontur-courses / git-rules

Unwatch 3 Star 0 Fork 37

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 0 tags

Go to file Add file Code

John Doe and disturm Autoconfig 3258cbc 2 days ago 2 commits

.gitconfig-for-nix	Autoconfig	2 days ago
.gitconfig-for-win	Autoconfig	2 days ago
README.md	root	7 days ago
apply-gitconfig-for-nix.sh	Autoconfig	2 days ago
apply-gitconfig-for-win.c...	Autoconfig	2 days ago

About

No description, website, or topics provided.

Readme

Releases

No releases published
[Create a new release](#)

Vare-репозиторий

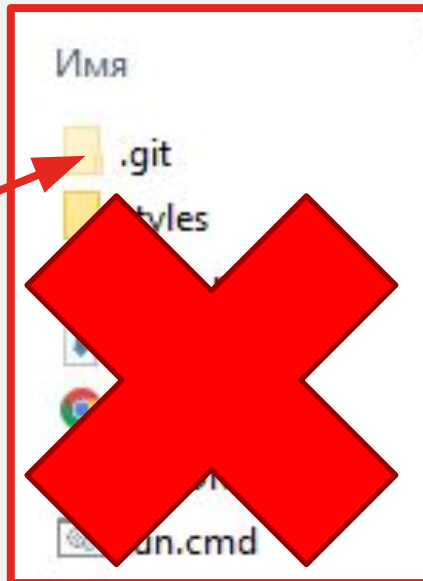
Vare-репозиторий — репозиторий без рабочей директории

`git init --bare` — создать bare-репозиторий

Обычно называется `reponame.git`, например `git@github.com:kontur-courses/git.git`

Обычный репозиторий — для разработки, bare-репозиторий — для облаков

Репозиторий



Vare — англ. Голый, Пустой

Задание 1. Init Repo

Задание 2. Fork and Config

Structure

Actions

Remote

S1. Все локально

H1. Гибкое конфигурирование и качественная документация

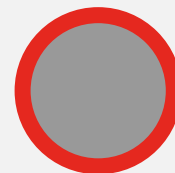
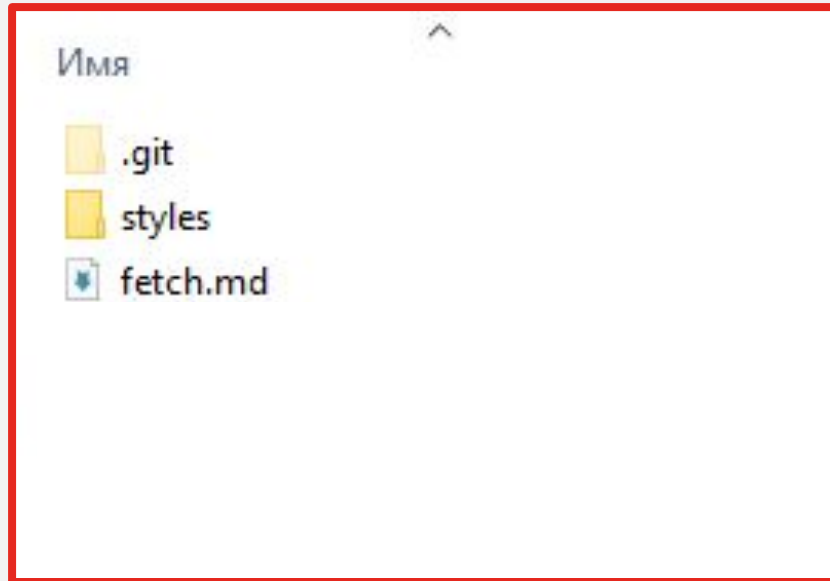
S2. Хранятся состояния директории, постепенная сборка коммита

Хранятся файлы, разница вычисляется на лету

Commit index для постепенной сборки коммита

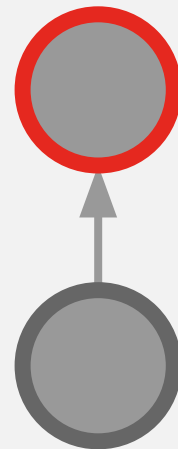
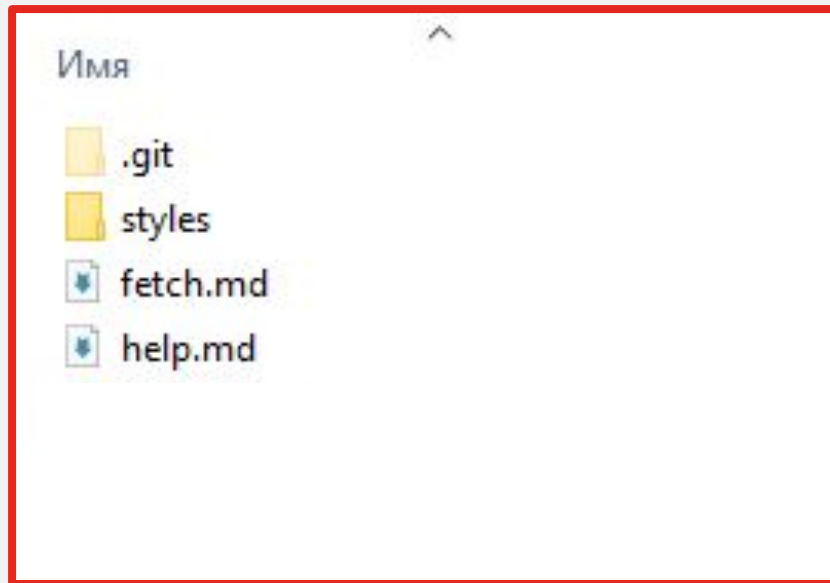
«Снимки» директории

Working directory



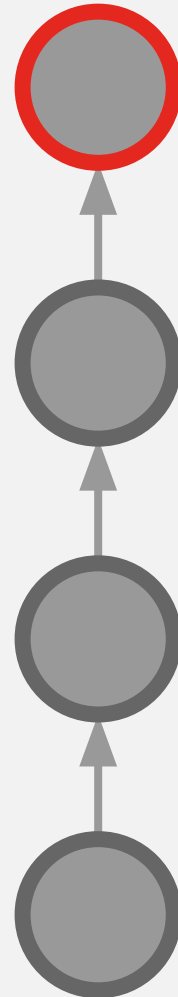
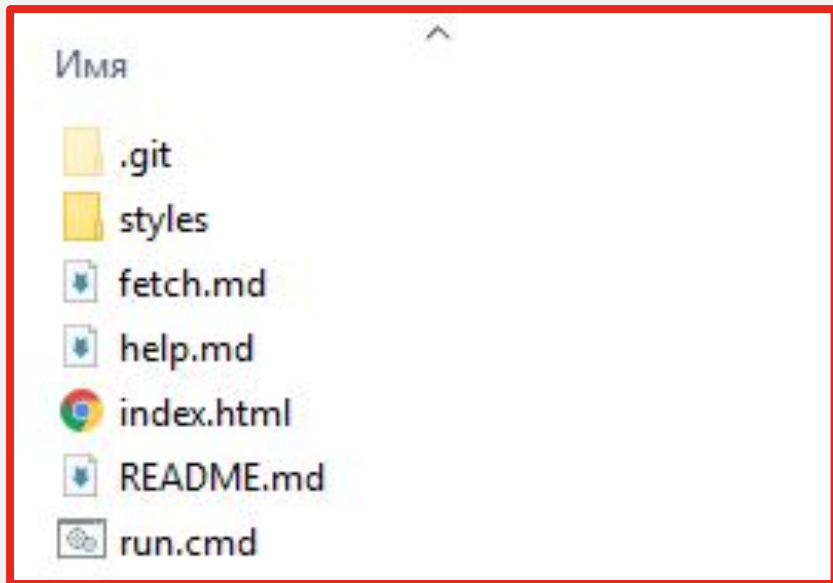
Сохранение состояния

Working directory



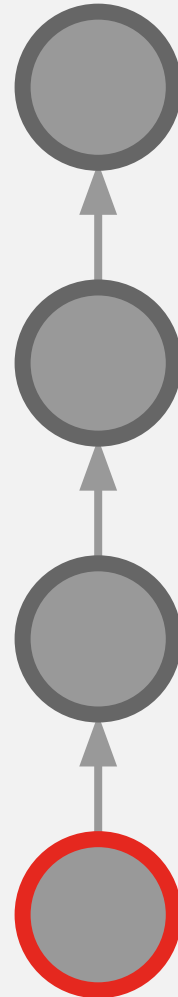
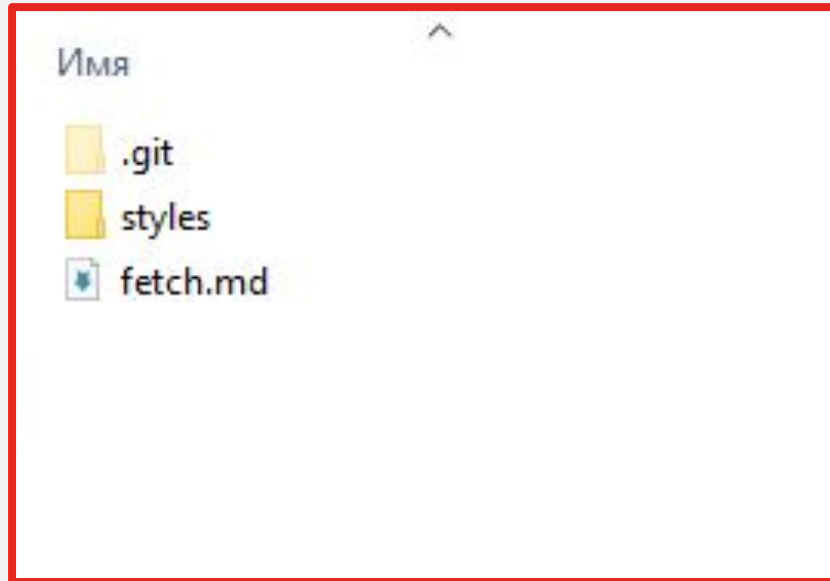
Еще сохранения

Working directory



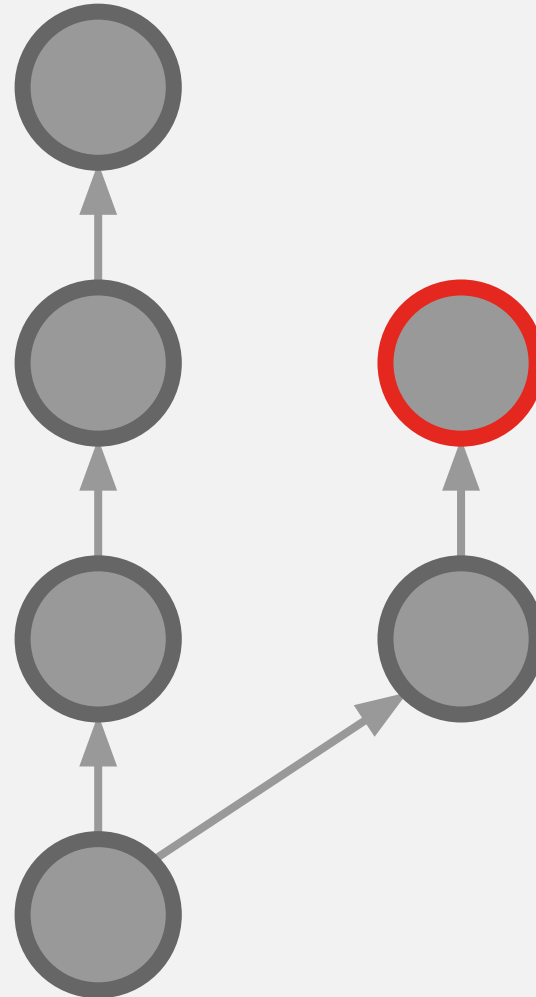
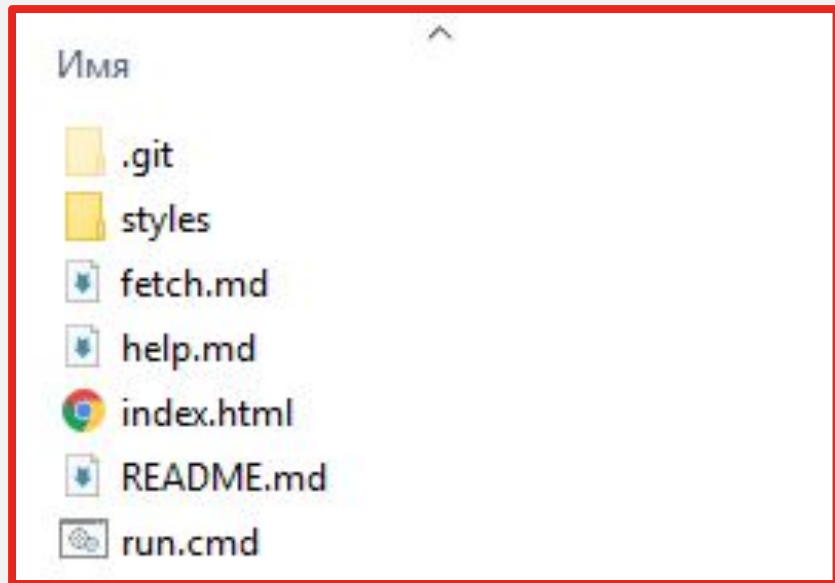
Загрузка состояния

Working directory

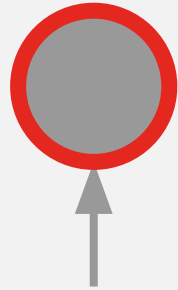


Альтернативная ветка истории

Working directory



Что содержит коммит?



Метаинформация

- Хэш-коммита
- Сообщение
- Информацию об авторе
- Время
- Родитель

Данные

- Полное состояние директории
- Изменения по сравнению с родителем

Метаинформация коммита

Commit	Author	Date	Hash
sheet-feature	digi	1 hour ago	6cba717
Add fetch.md	digi	1 hour ago	2f570da
Add runner	digi	1 hour ago	3687394
sheet markup	digi	1 hour ago	a36dbce
master	digi	3 days ago	07cff6f
Initial commit	Ivan Domashnikh	3 days ago	3d9c360

Commit details for 'Add runner':

- Author: [digi <digi@skbkontur.ru>](#)
- Date: 1 hour ago (24.06.2019 16:28:39)
- Commit hash: [36873949d73816fa1291210b092912173b410c1c](#)
- Child: [2f570da431](#)
- Parent: [a36dbce399](#)

Commit message: Add runner

Сокращенный хэш коммита

Хэш коммита

Полное состояние директории

The screenshot displays a Git GUI interface. The top section shows a commit history with the following entries:

Commit	Author	Time	Hash
sheet-feature	digi	1 hour ago	6cba717
Add fetch.md	digi	1 hour ago	2f570da
Add runner	digi	1 hour ago	3687394
sheet markup	digi	1 hour ago	a36dbce
master	digi	3 days ago	07cff6f
Initial commit	Ivan Domashnikh	3 days ago	3d9c360

The bottom section shows a file tree with the following structure:

- styles
 - marked.css
 - prism.css
- index.html
- README.md
- run.cmd

Изменения по сравнению с родителем

The screenshot displays a Git GUI interface. At the top, the commit history is shown as a vertical list of nodes. The current commit, 'sheet-feature', is selected and highlighted in blue. Below the history, the 'Diff' view is active, showing a comparison between the current commit and its parent, 'sheet markup'. The file 'run.cmd' is selected in the file tree, and the diff content is displayed in the console area.

Commit	Author	Time	Hash
sheet-feature	digi	1 hour ago	6cba717
Add fetch.md	digi	1 hour ago	2f570da
Add runner	digi	1 hour ago	3687394
sheet markup	digi	1 hour ago	a36dbce
master	digi	3 days ago	07cff6f
Initial commit	Ivan Domashnikh	3 days ago	3d9c360

```
diff --git a/run.cmd b/run.cmd
new file mode 100644
index 0000000..6caa56b
--- /dev/null
+++ b/run.cmd
@@ -0,0 +1 @@
1 +npm install --global http-server & cmd /c start htt
```

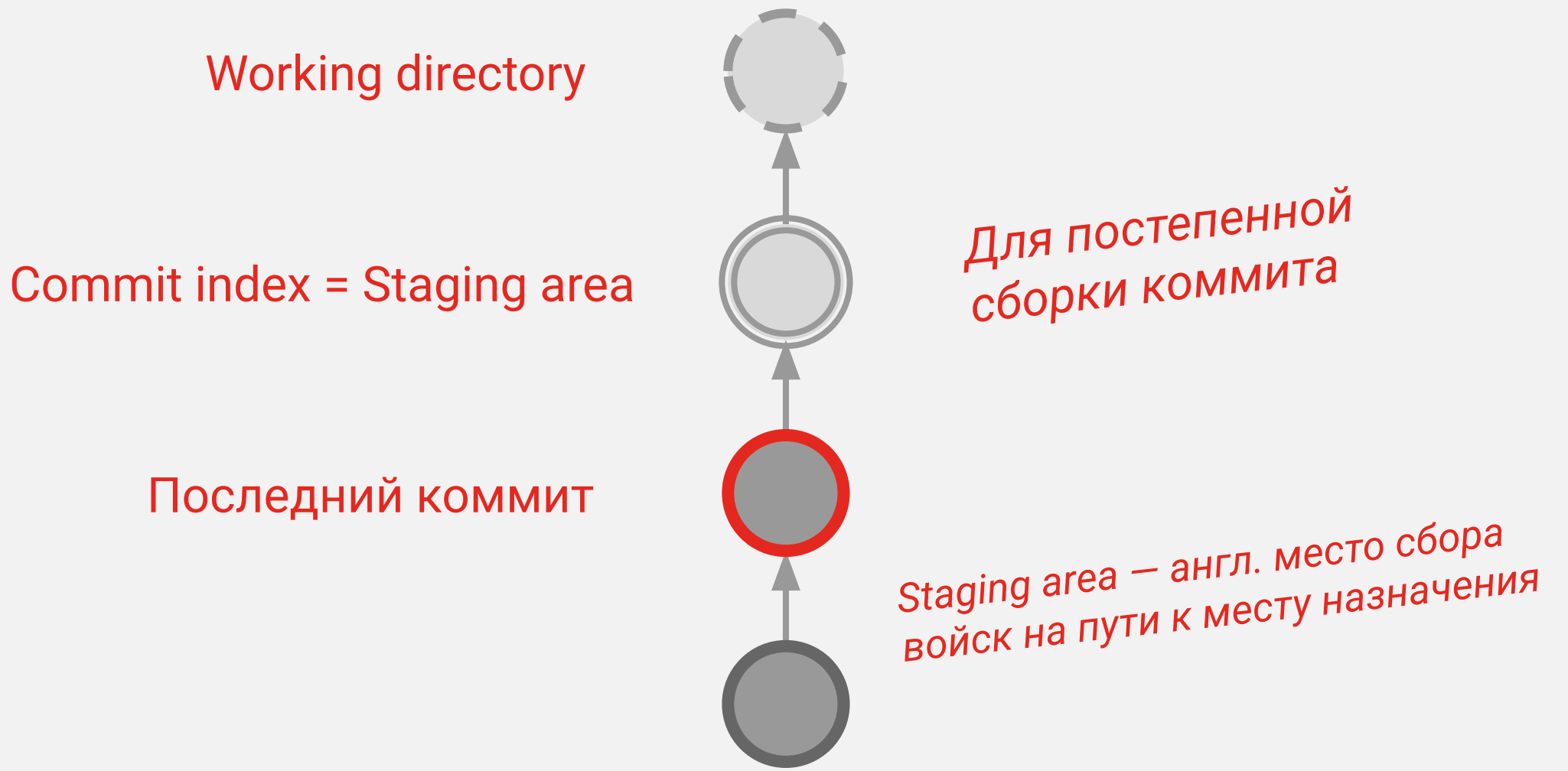
Хранятся файлы, разница вычисляется на лету

1. Каждый коммит хранит структуру каталога и все файлы состояния директории
2. Хранение файлов оптимизировано: **файлы не хранятся повторно**, потому что в структуре каталога хранятся не сами файлы, а ссылки по хэшу на них
3. **Используется сжатие**, чтобы текстовые данные занимали меньше места
4. Разница между коммитами вычисляется на лету и с родителем и с любым другим коммитом

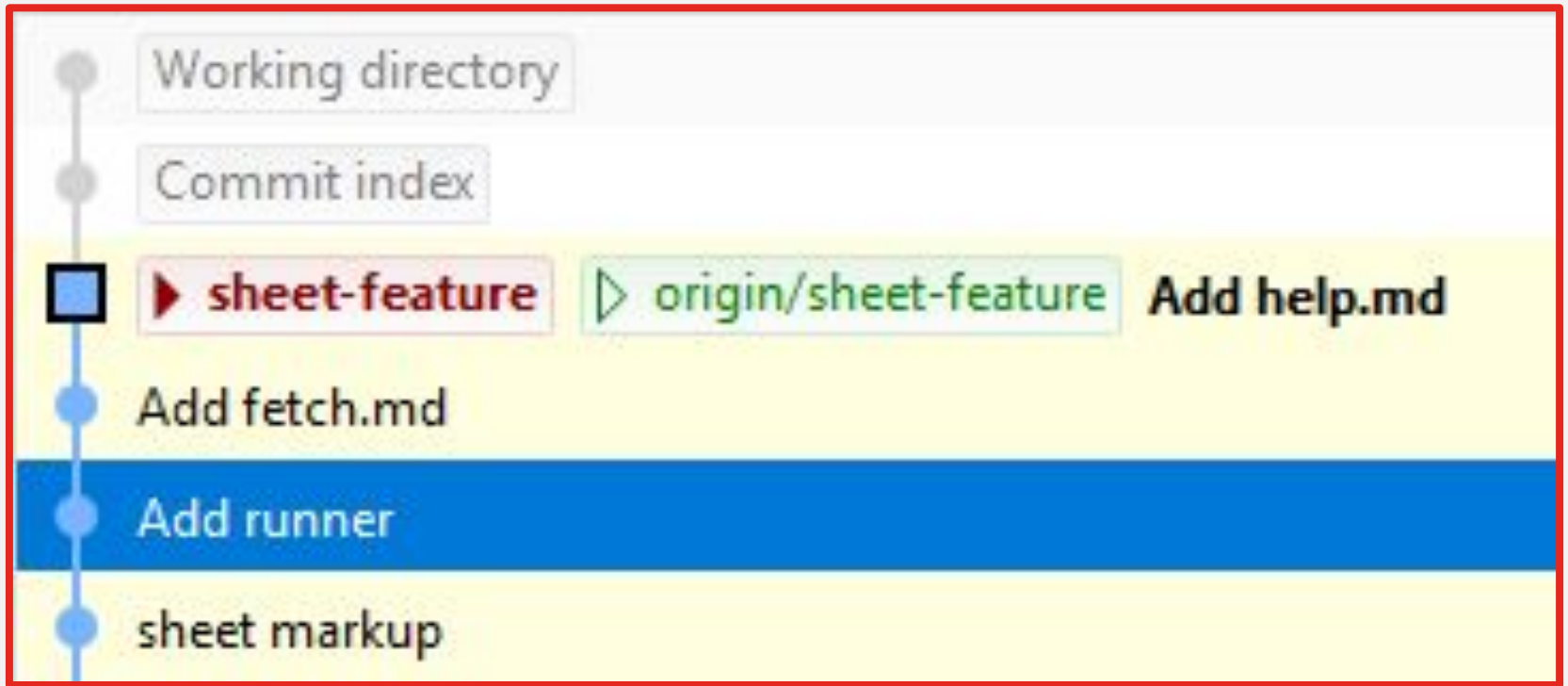
Working directory & Commit index



Working directory & Commit index

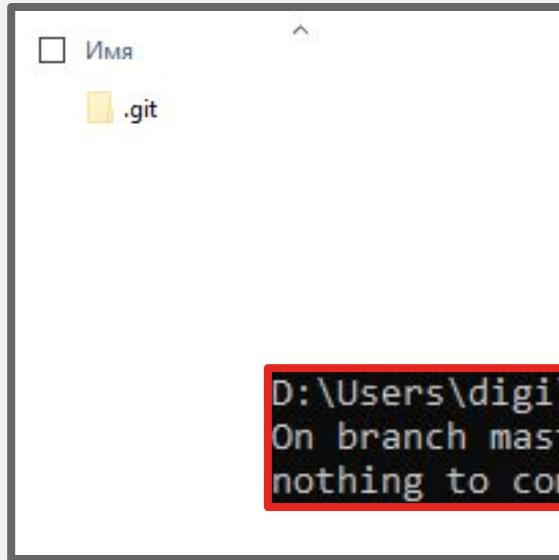


Working directory & Commit index

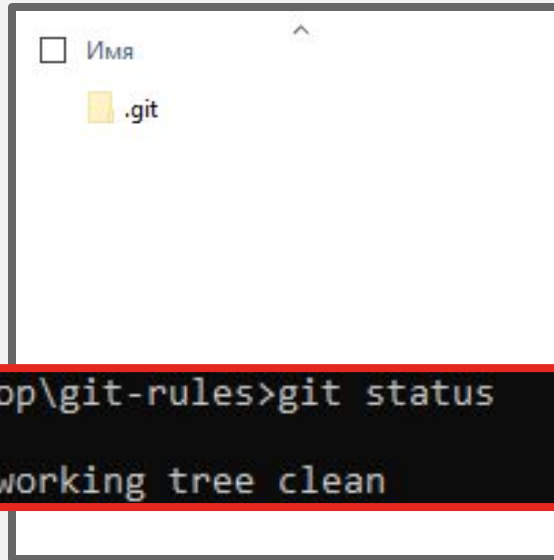


Внесение изменений

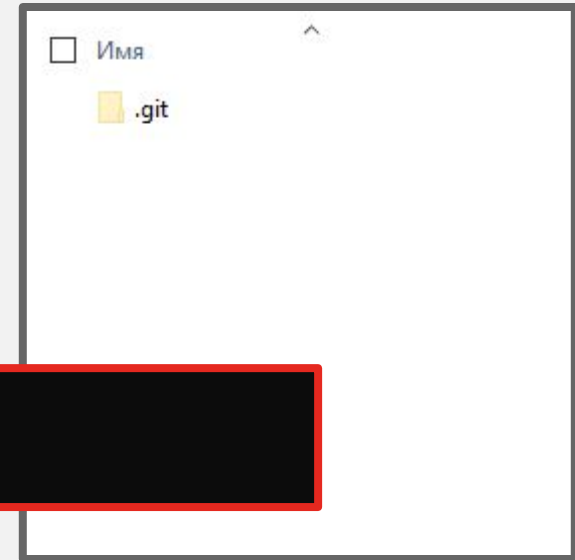
Working directory



Commit index



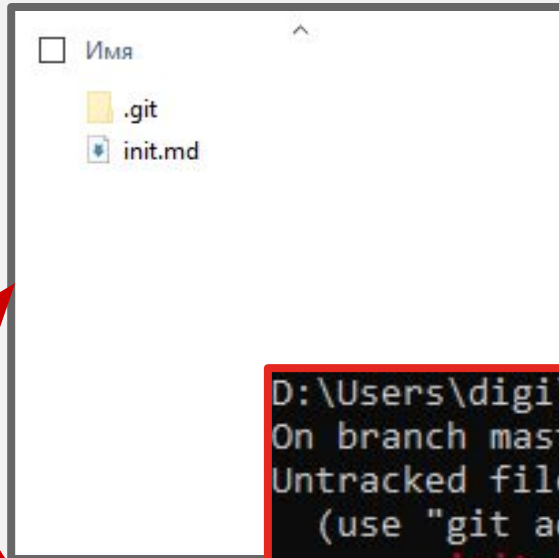
Последний коммит



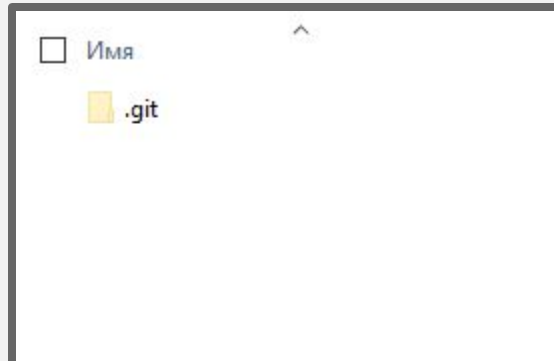
```
D:\Users\digi\Desktop\git-rules>git status  
On branch master  
nothing to commit, working tree clean
```

Внесение изменений

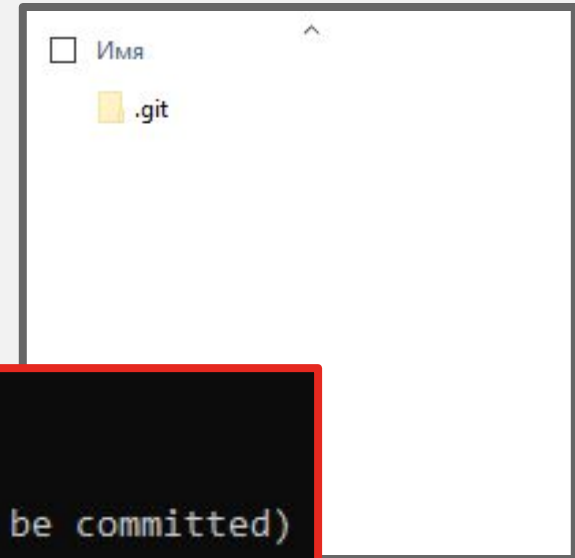
Working directory



Commit index



Последний коммит



```
D:\Users\digi\Desktop\git-rules>git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
   init.md
```

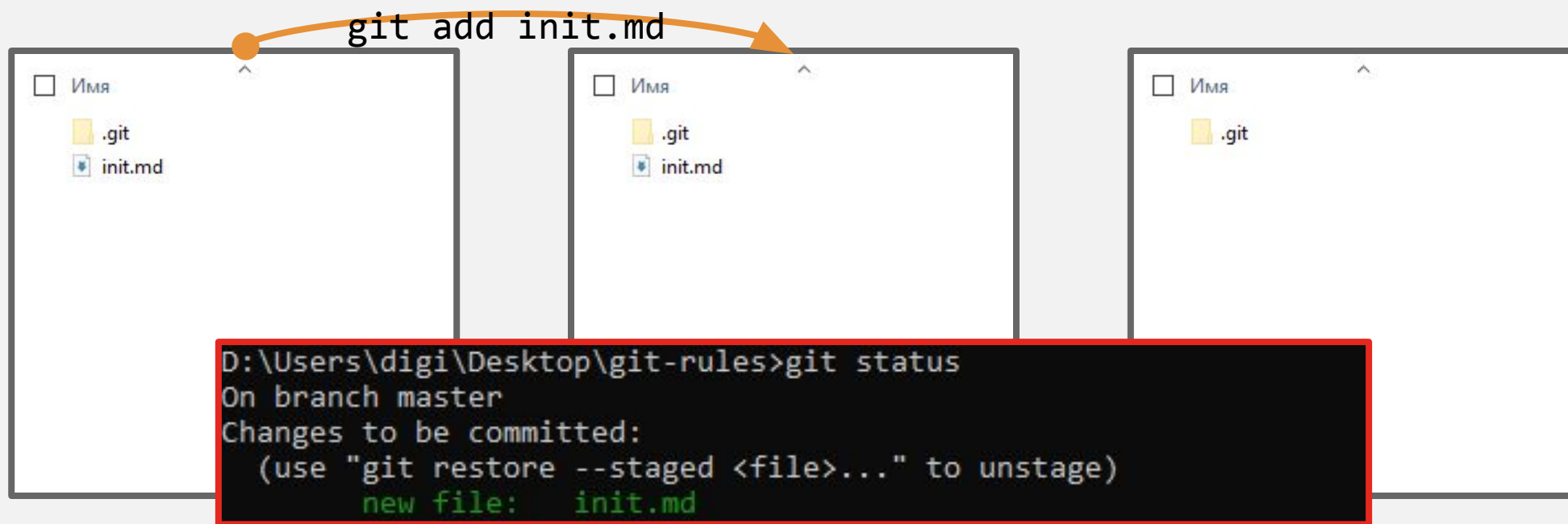
Редактирование

Внесение изменений

Working directory

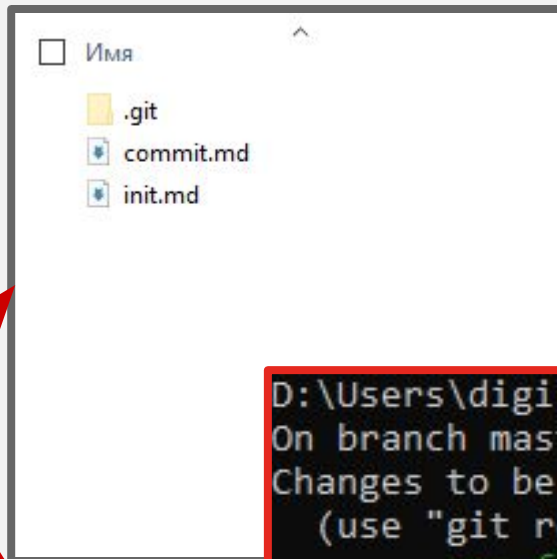
Commit index

Последний коммит

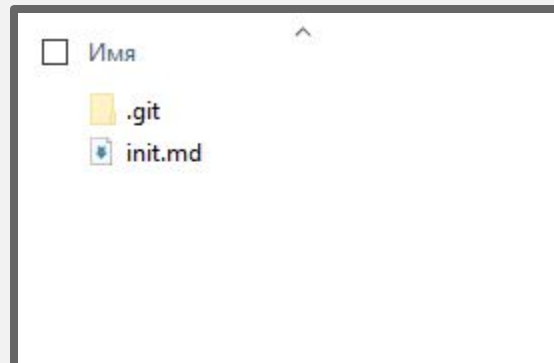


Внесение изменений

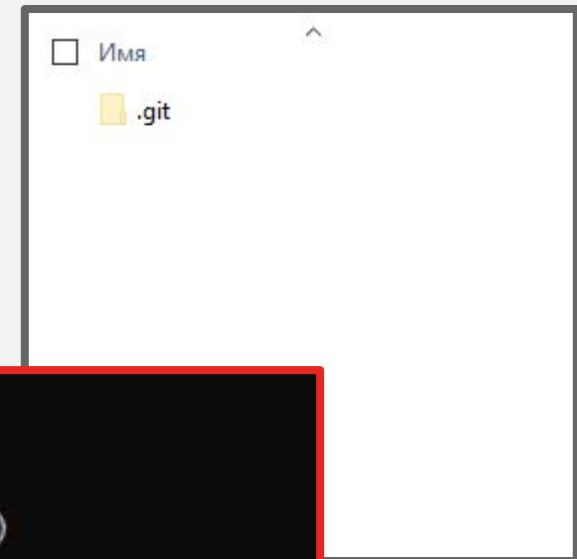
Working directory



Commit index



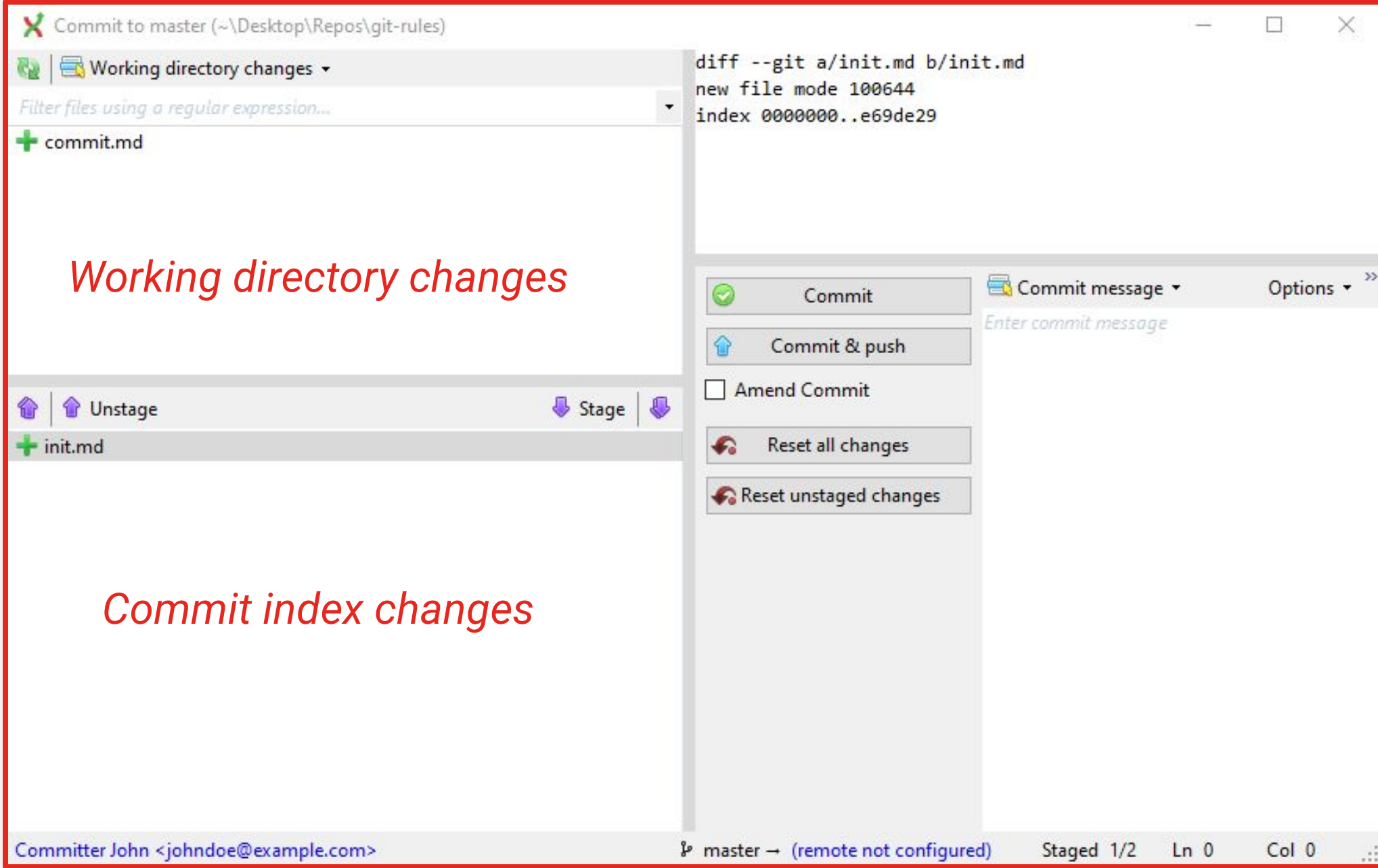
Последний коммит



```
D:\Users\digi\Desktop\git-rules>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
       new file:   init.md

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

Редактирование



Working directory changes

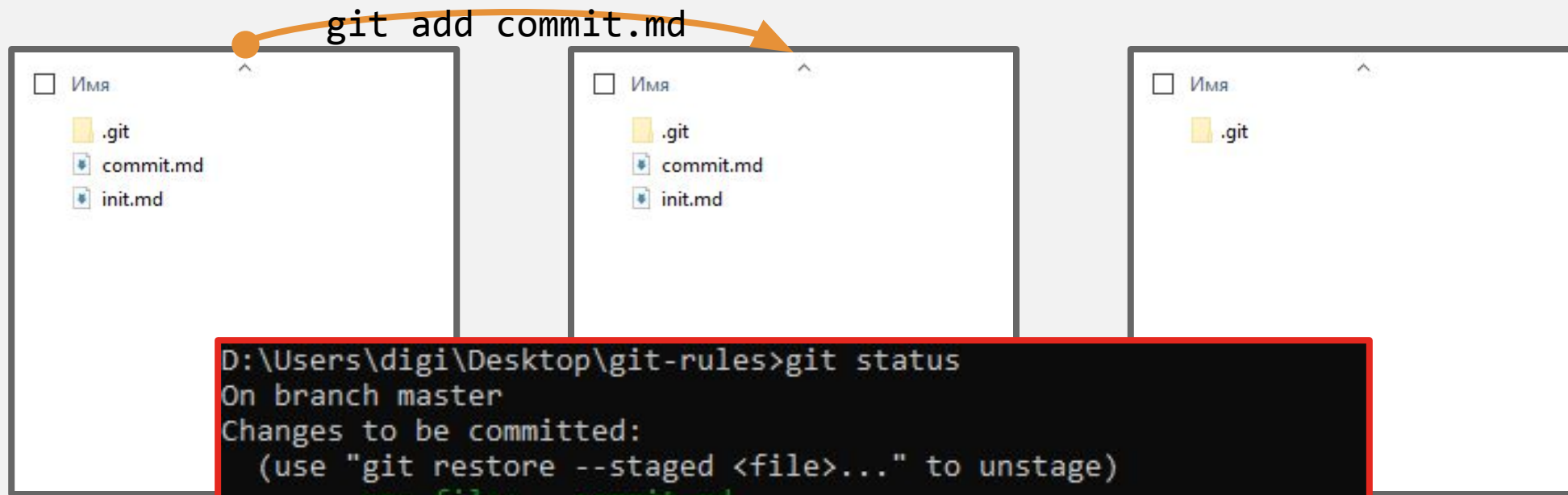
Commit index changes

Внесение изменений

Working directory

Commit index

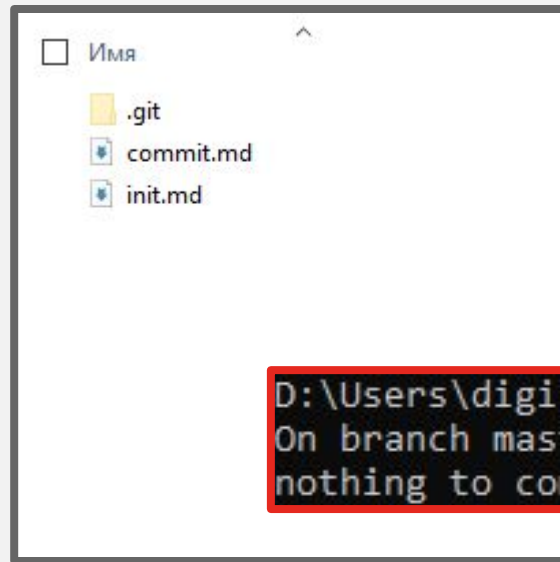
Последний коммит



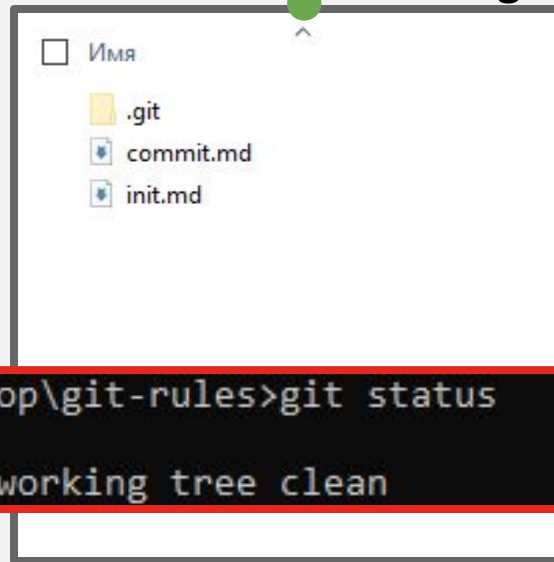
```
D:\Users\digi\Desktop\git-rules>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
   new file:   commit.md
   new file:   init.md
```


Внесение изменений

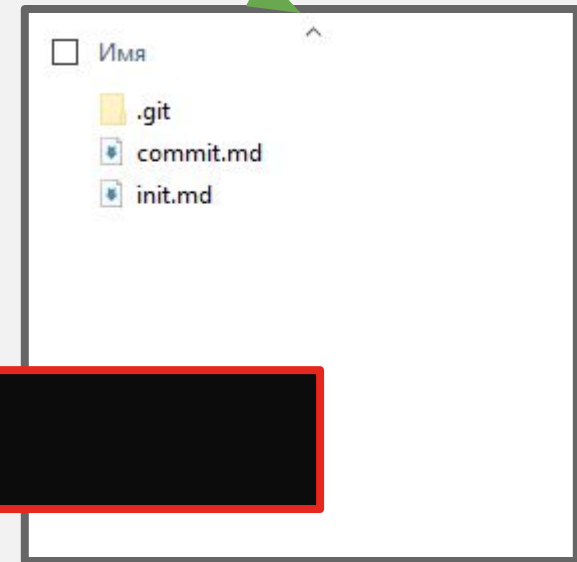
Working directory



Commit index

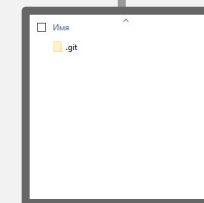


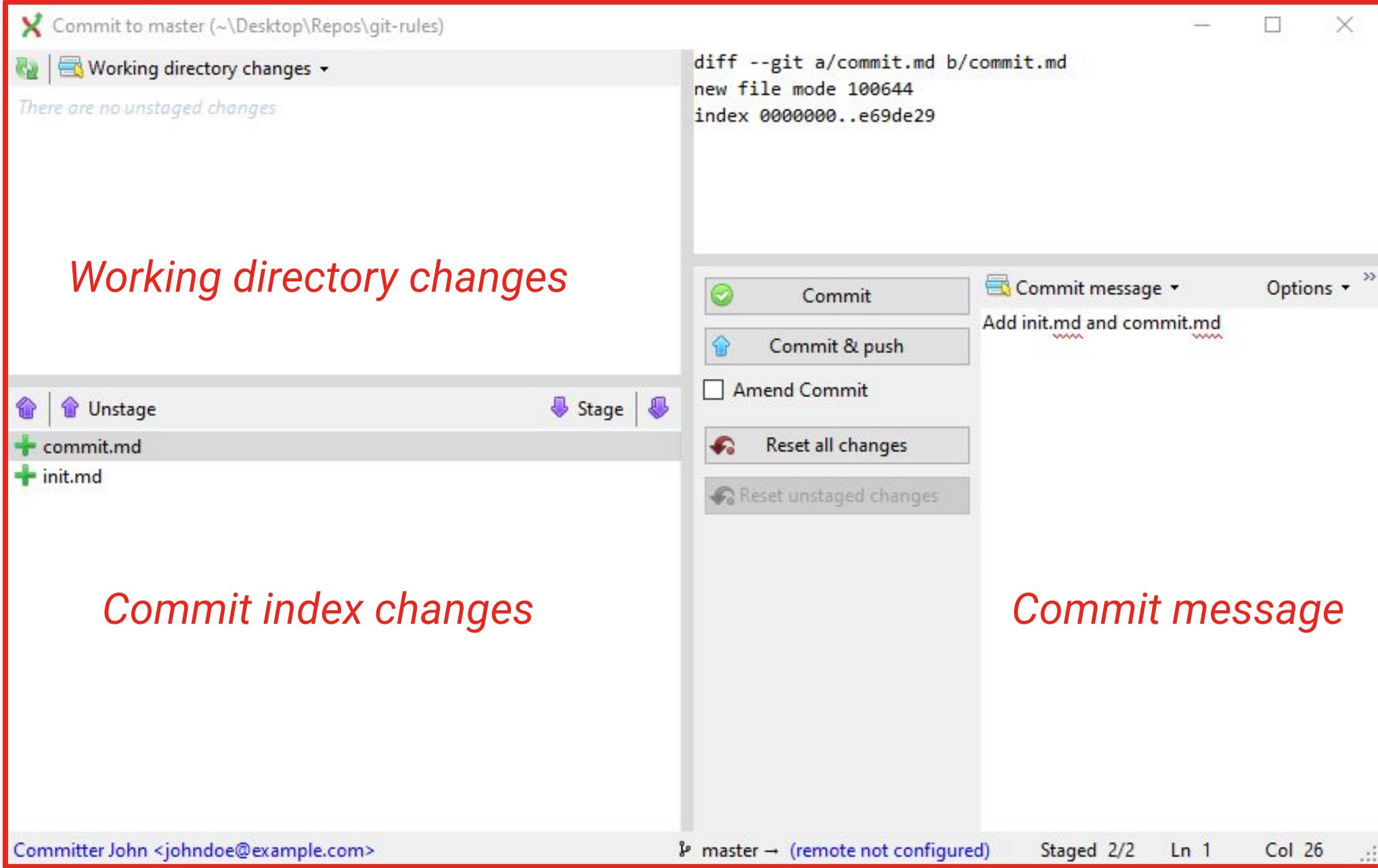
Последний коммит



git commit

```
D:\Users\digi\Desktop\git-rules>git status  
On branch master  
nothing to commit, working tree clean
```



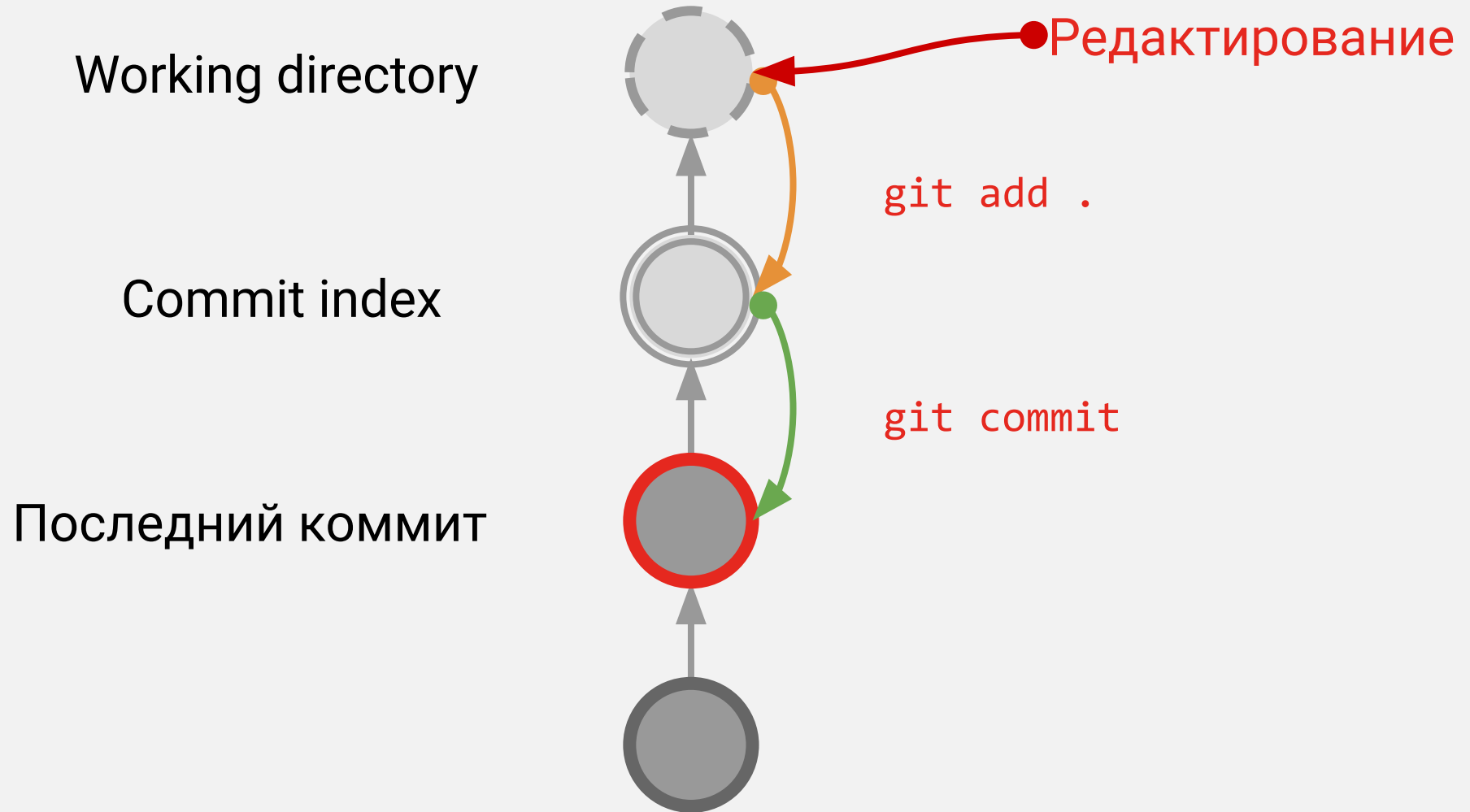


Working directory changes

Commit index changes

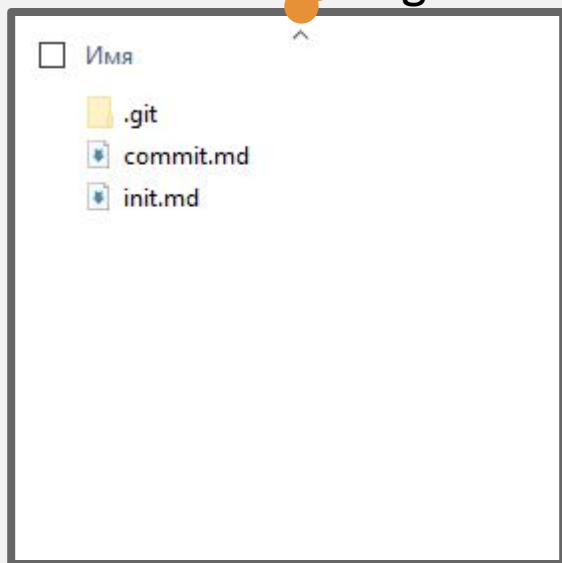
Commit message

Внесение изменений

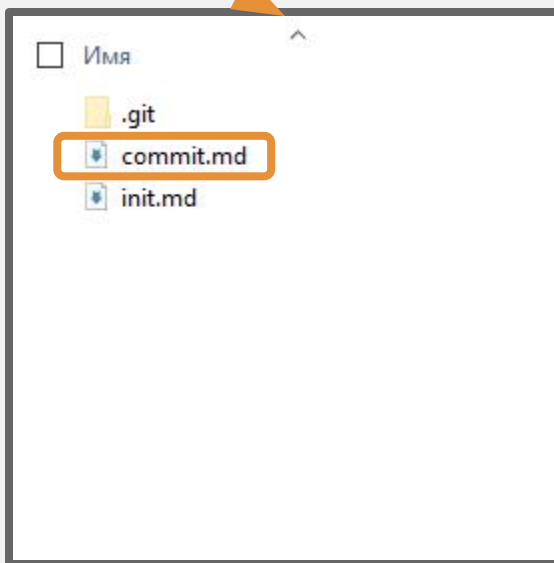


Отмена изменений

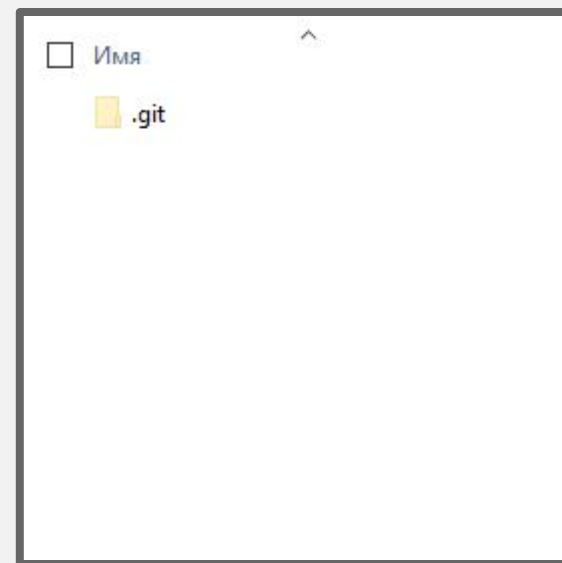
Working directory



Commit index



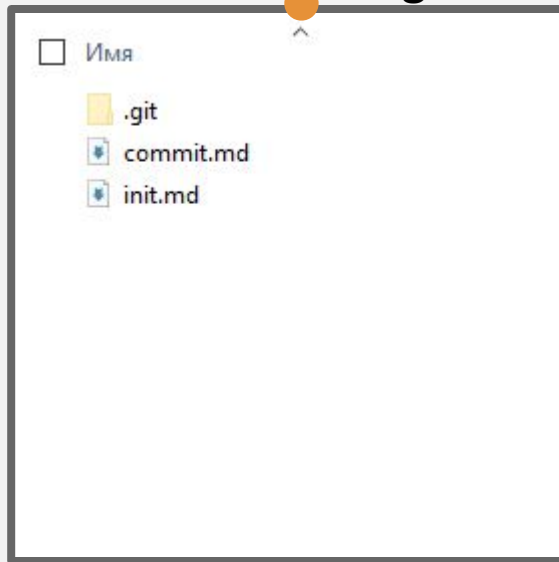
Последний коммит



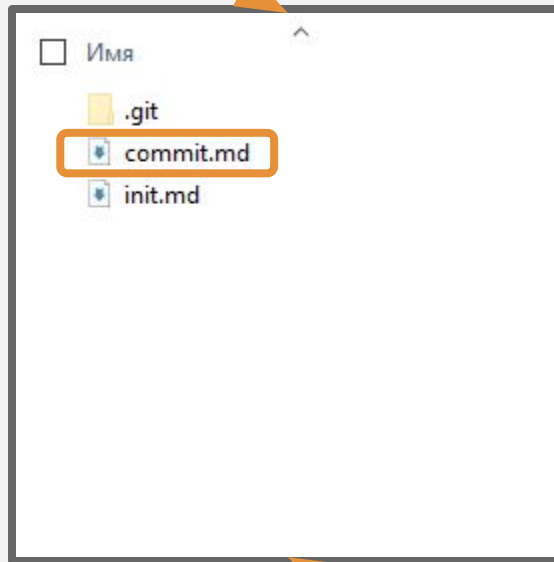
`git add commit.md`

Отмена изменений

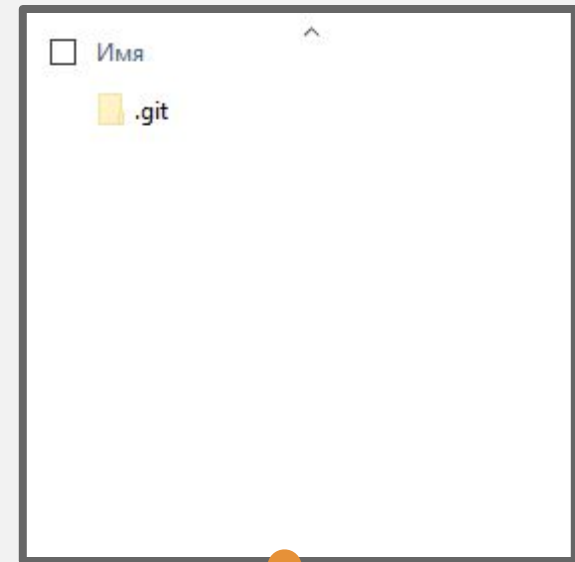
Working directory



Commit index



Последний коммит

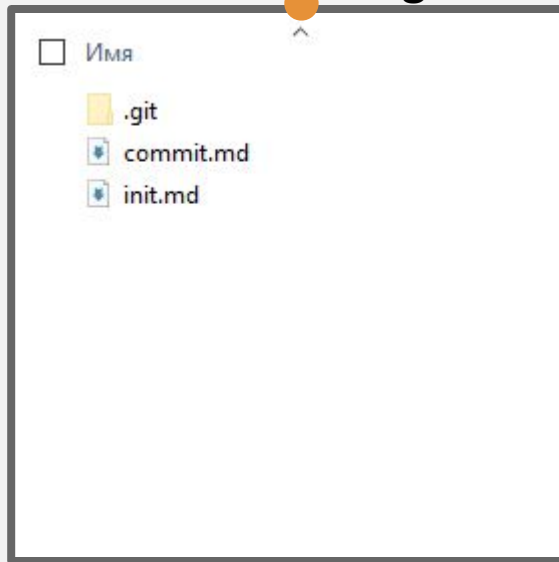


`git add commit.md`

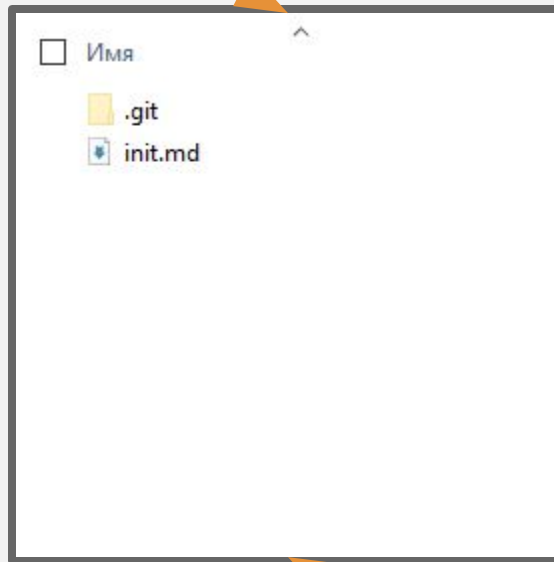
`git restore --staged commit.md`

Отмена изменений

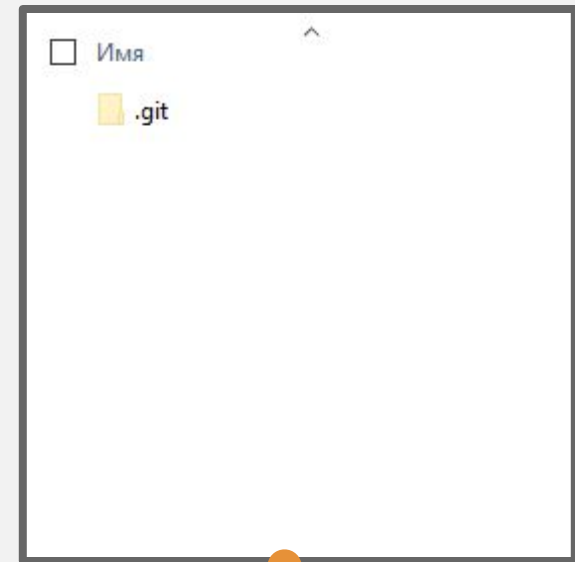
Working directory



Commit index



Последний коммит

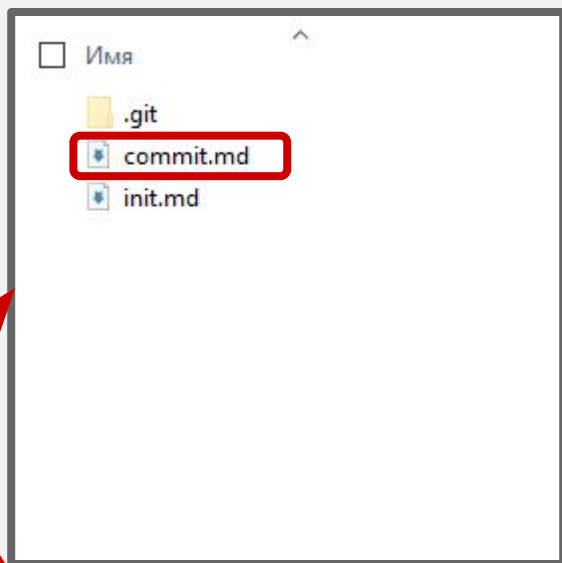


`git add commit.md`

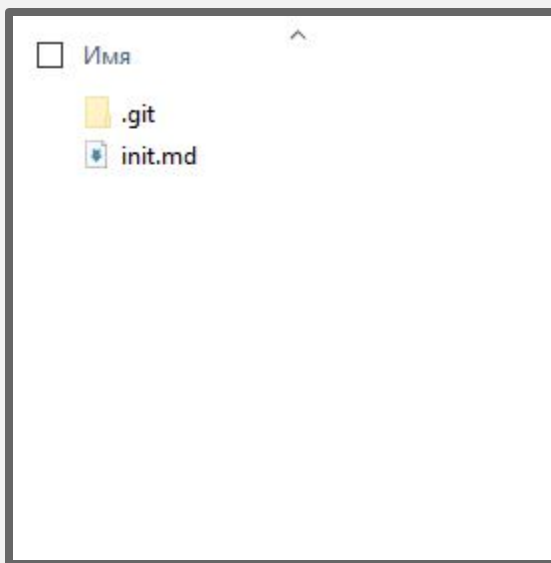
`git restore --staged commit.md`

Отмена изменений

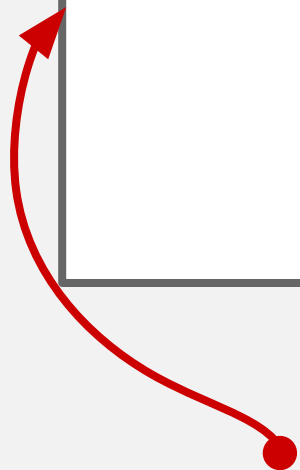
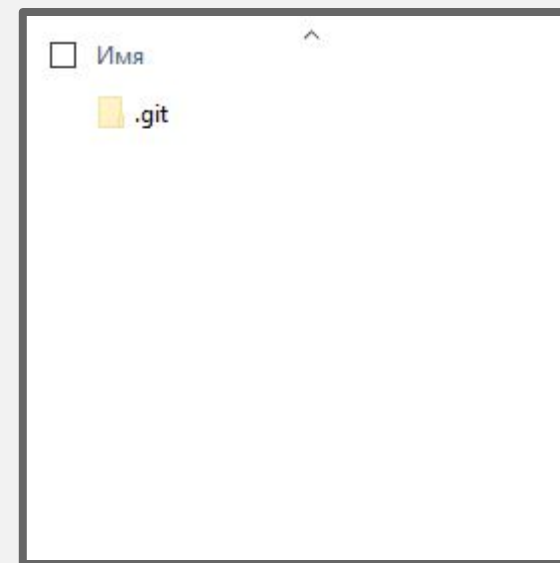
Working directory



Commit index



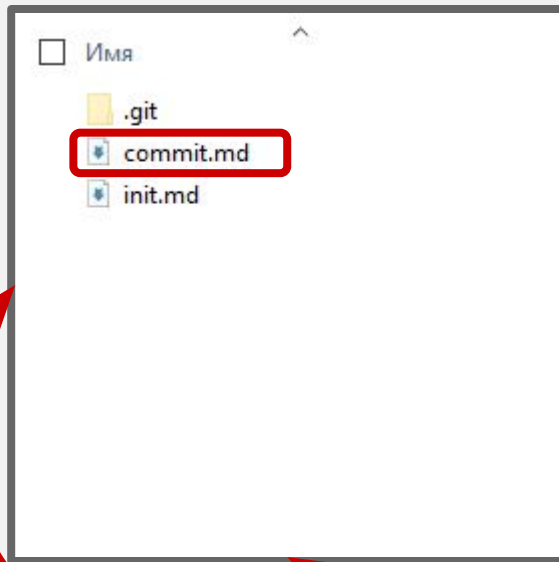
Последний коммит



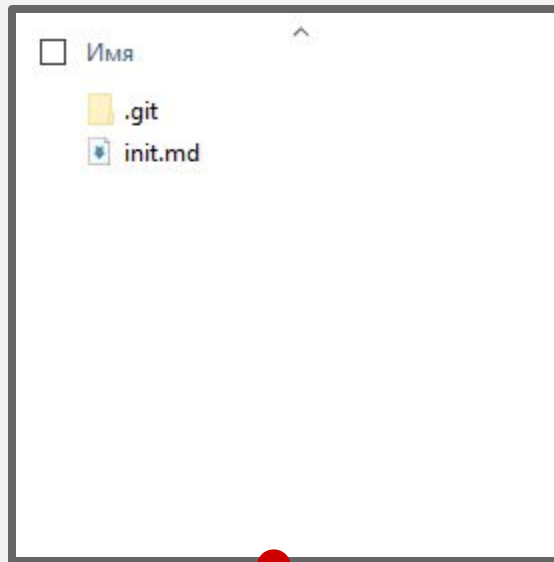
Редактирование

Отмена изменений

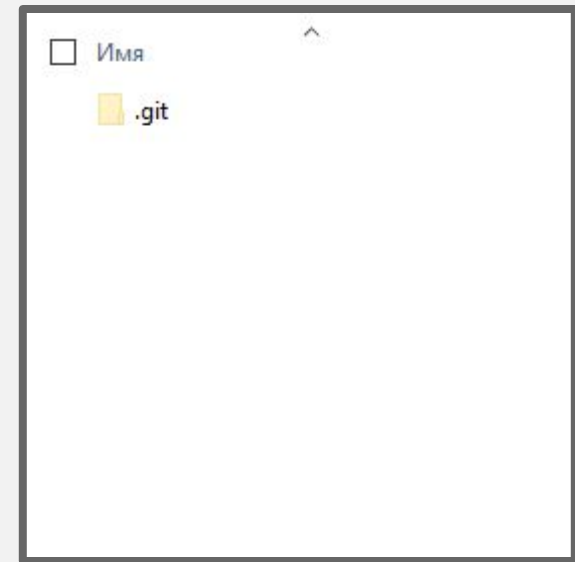
Working directory



Commit index



Последний коммит

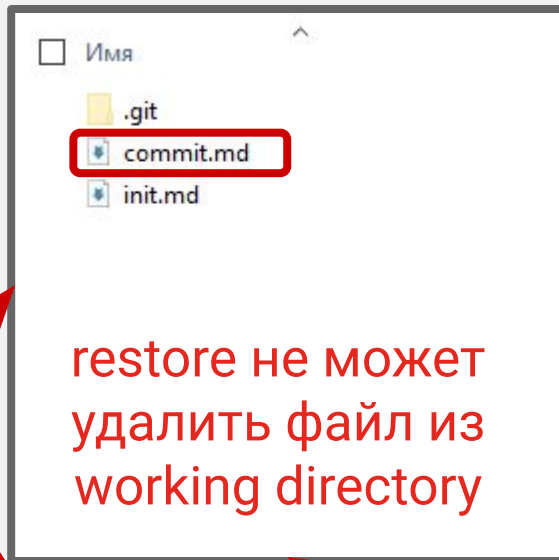


`git restore --worktree commit.md`

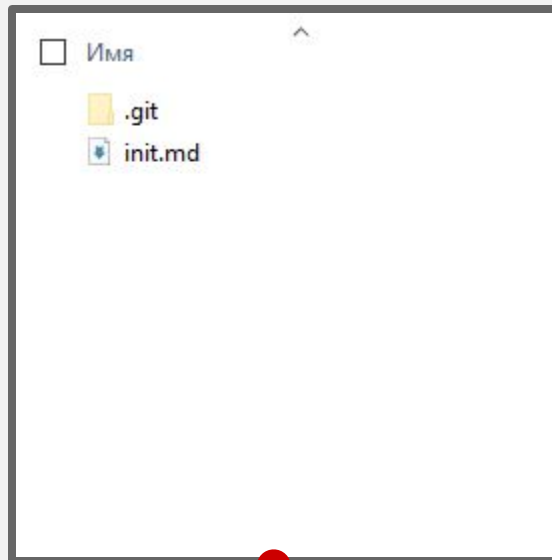
Редактирование

Отмена изменений

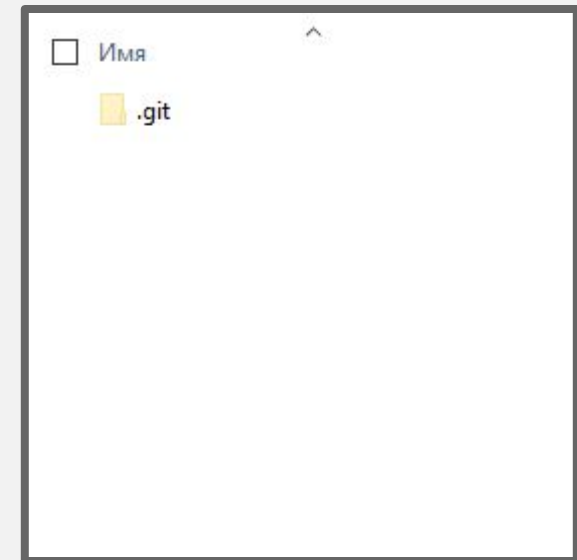
Working directory



Commit index



Последний коммит

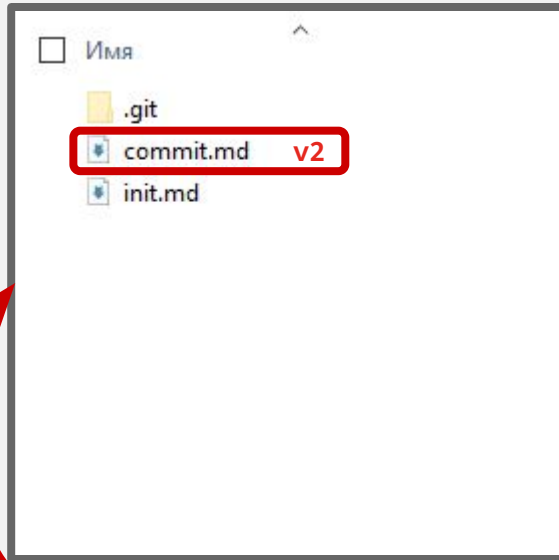


`git restore --worktree commit.md`

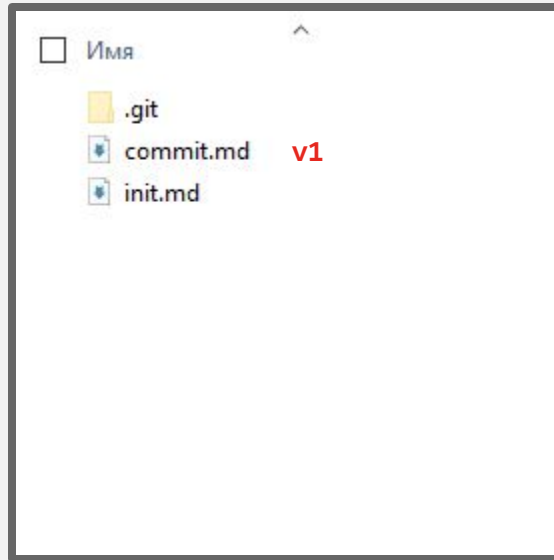
Редактирование

Отмена изменений

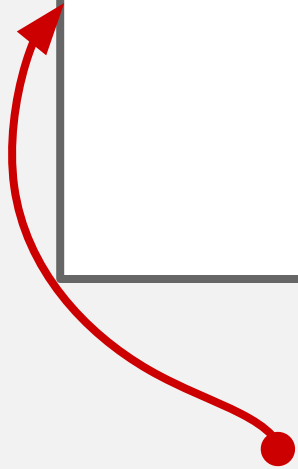
Working directory



Commit index



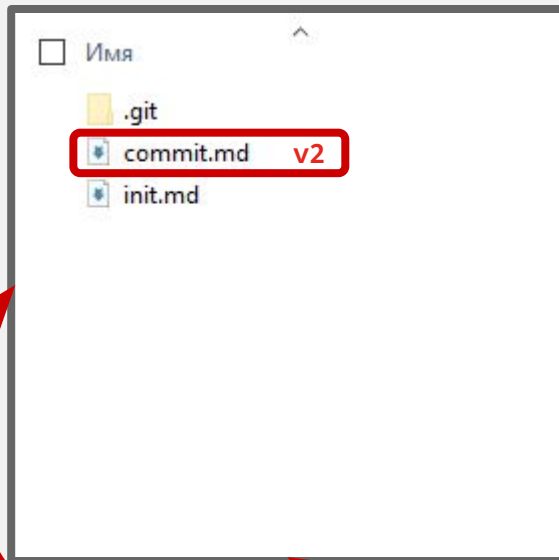
Последний коммит



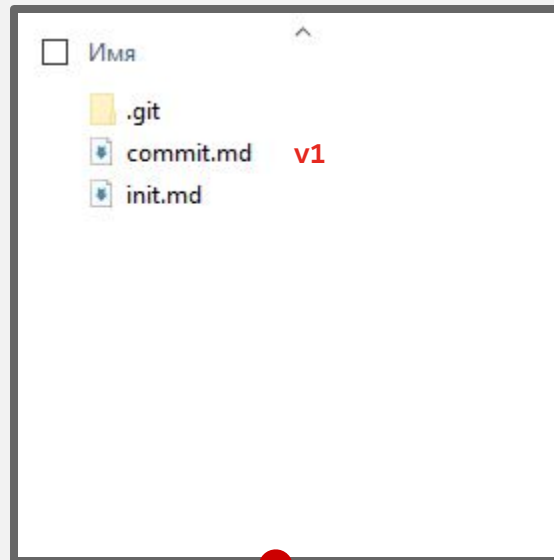
Редактирование

Отмена изменений

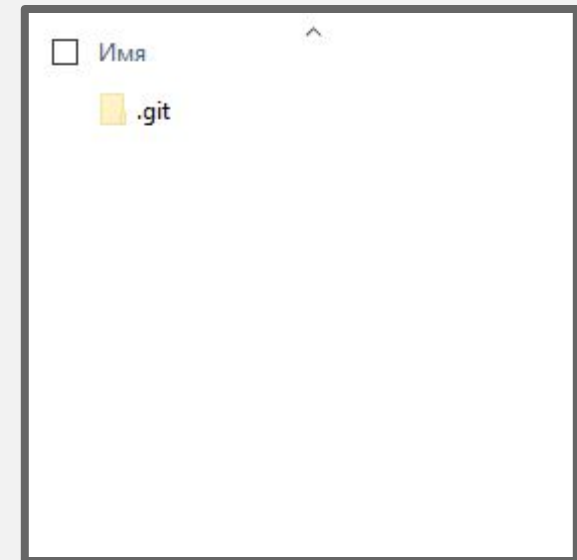
Working directory



Commit index

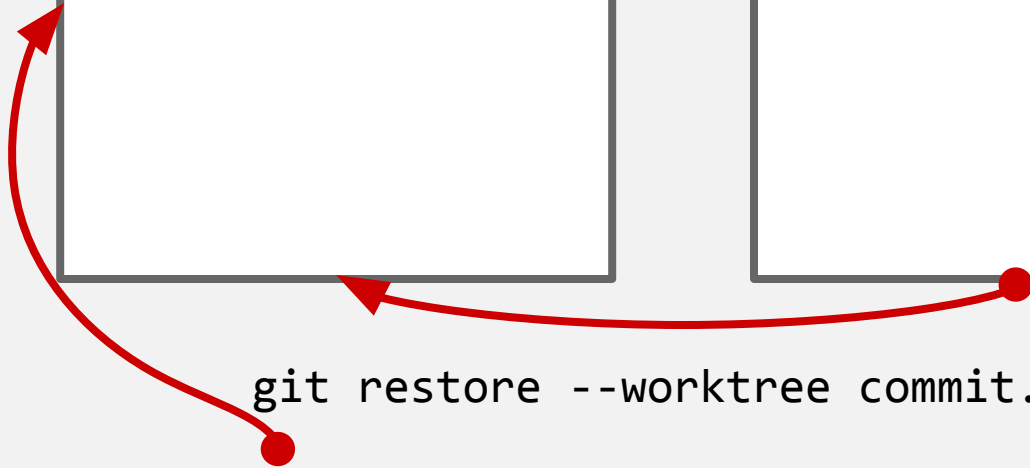


Последний коммит



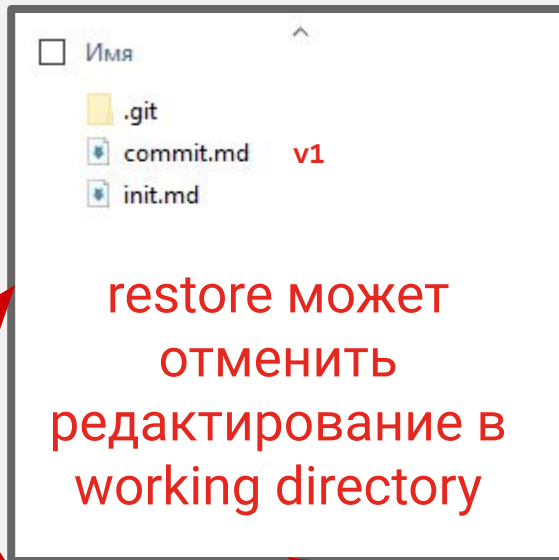
`git restore --worktree commit.md`

Редактирование

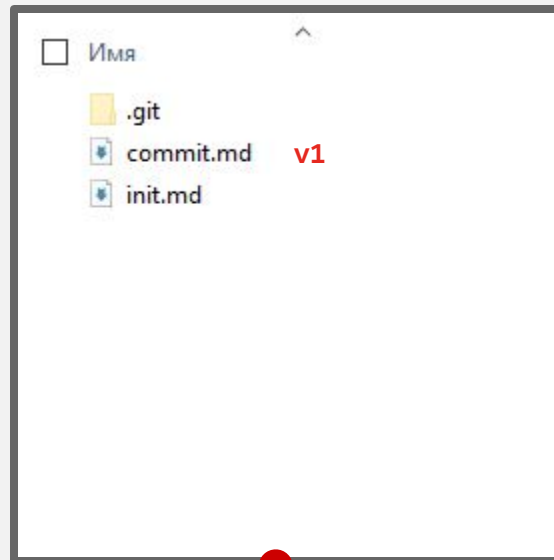


Отмена изменений

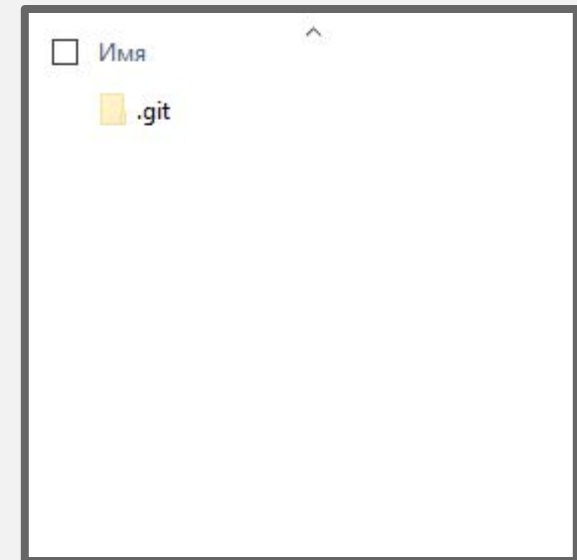
Working directory



Commit index



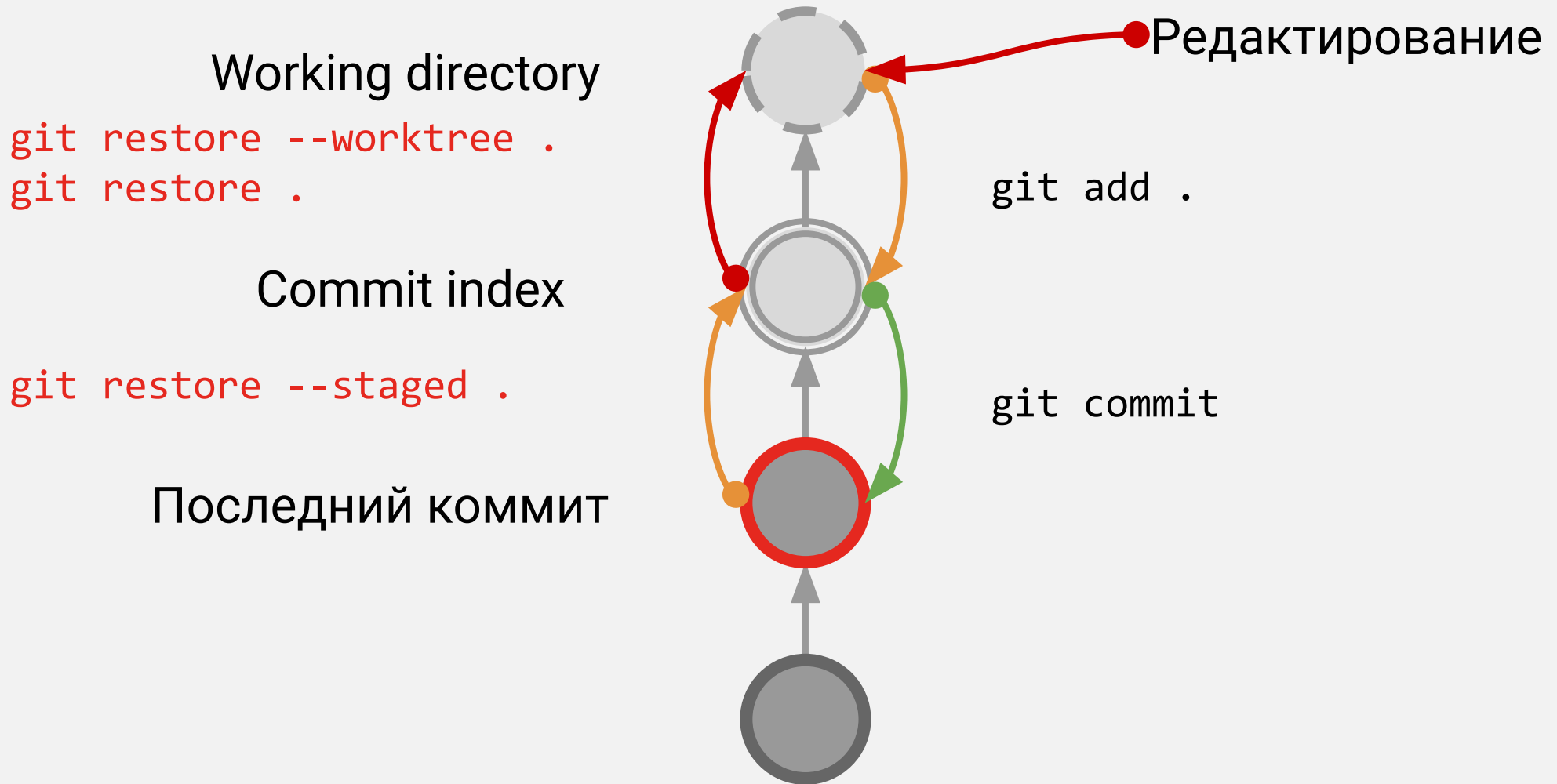
Последний коммит



`git restore --worktree commit.md`

Редактирование

Отмена изменений



Отмена изменений по-старому

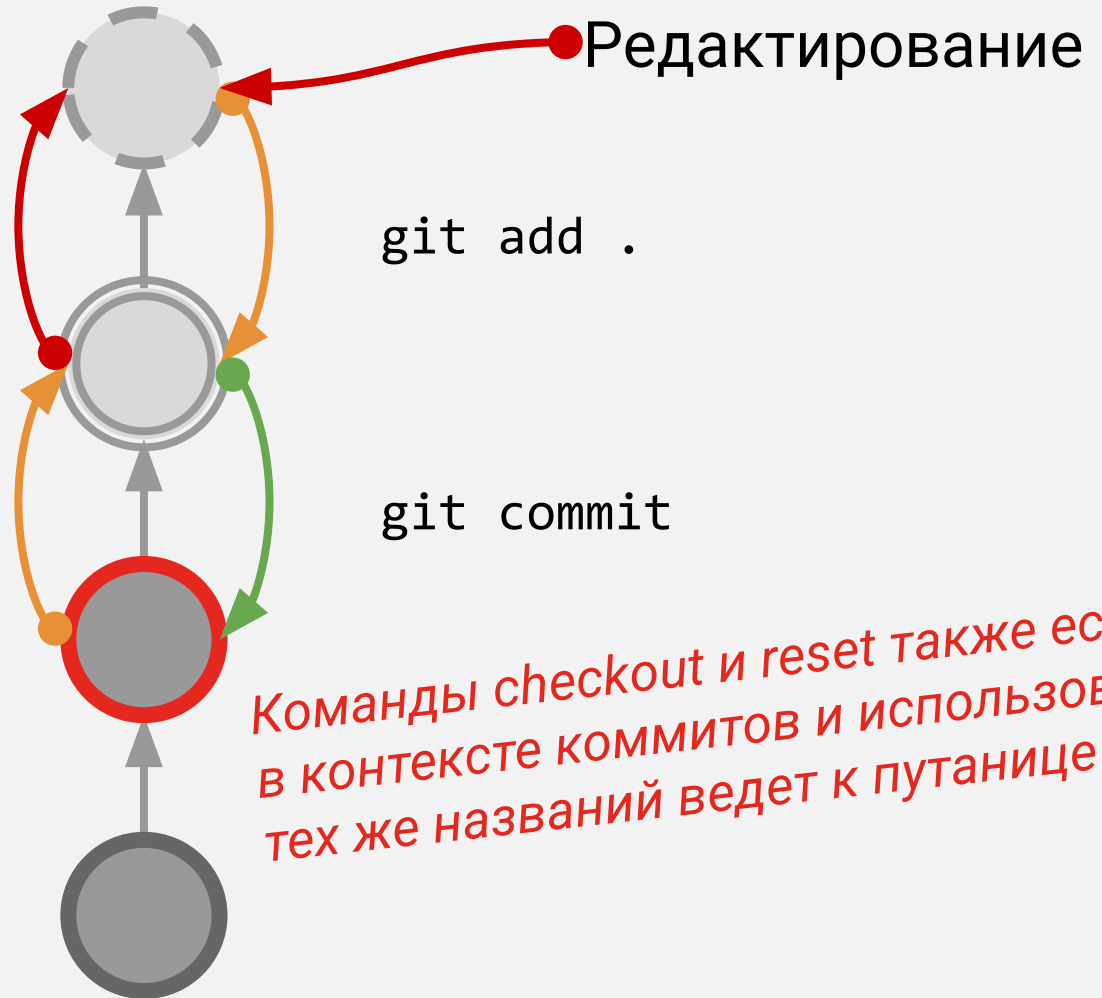
Working directory

```
git restore --worktree .  
git restore .  
git checkout .
```

Commit index

```
git restore --staged .  
git reset .
```

Последний коммит



Команды checkout и reset также есть в контексте коммитов и использование тех же названий ведет к путанице

Отмена изменений

Working directory

```
git restore --worktree .
```

```
git restore .
```

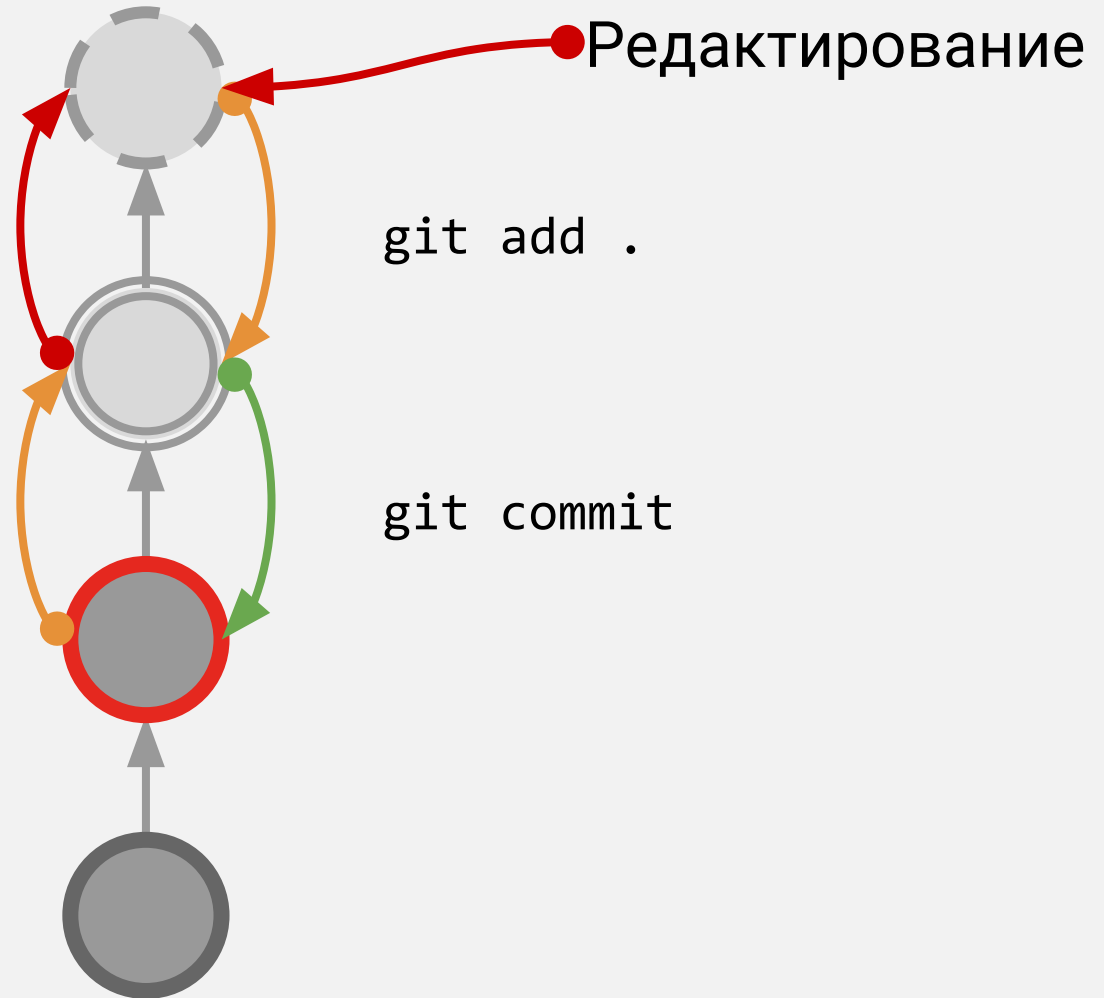
```
git checkout .
```

Commit index

```
git restore --staged .
```

```
git reset .
```

Последний коммит



Один файл с разными версиями

```
D:\Users\digi\Desktop\Repos\git-rules>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   commit.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   commit.md
```


The screenshot shows a Git commit window titled "Commit to master (~\Desktop\Repos\git-rules)". The window is divided into several sections:

- Working directory changes:** A list of files to be committed, currently showing "commit.md".
- Commit index changes:** A diff view showing the changes to "commit.md". The diff includes a header with commit hashes and a list of changes. The changes are:
 - Line 1: "## S2. Хранятся состояния директории, постепенная сборка коммита"
 - Line 2: "### Хранятся файлы, разница вычисляется на лету"
 - Line 3: "-### Commit index для сборки коммита" (removed)
 - Line 4: "+### Commit index для сборки коммита" (added)
 - Line 5: "+1. `git add .` – добавить все измененные файлы в индекс"
 - Line 6: "+2. `git commit -m <msg>` – записать изменения из индекса в репозиторий"
 - Line 7: "+3. `git status -sh` – вывести состояние директории и индекса кратко с указателем на изменения"
- Commit message:** A text area for entering the commit message, currently containing "Add commit.md header".
- Commit options:** A set of buttons for committing, including "Commit", "Commit & push", "Amend Commit", "Reset all changes", and "Reset unstaged changes".

At the bottom of the window, the status bar shows "Committer John <johndoe@example.com>", "master → (remote not configured)", "Staged 1/2", "Ln 1", and "Col 21".

Working directory changes

Commit index changes

Commit message

Удобный алиас

```
git config --global alias.st "status -sb"
```

```
D:\Users\digi\Desktop\Repos\git-rules>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   commit.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   commit.md
```



```
D:\Users\digi\Desktop\Repos\git-rules>git status -sb
## master
AM commit.md
```

Задание 3. Commits

Structure

Actions

Remote

S1. Все локально

S2. Хранятся
состояния директории,
постепенная сборка коммита

H1. Гибкое конфигурирование и качественная документация

S3. Манипуляции через ссылки, нет ссылки — в мусор

HEAD — текущая ссылка,
tag — фиксированная ссылка,
branch — движущаяся за HEAD ссылка

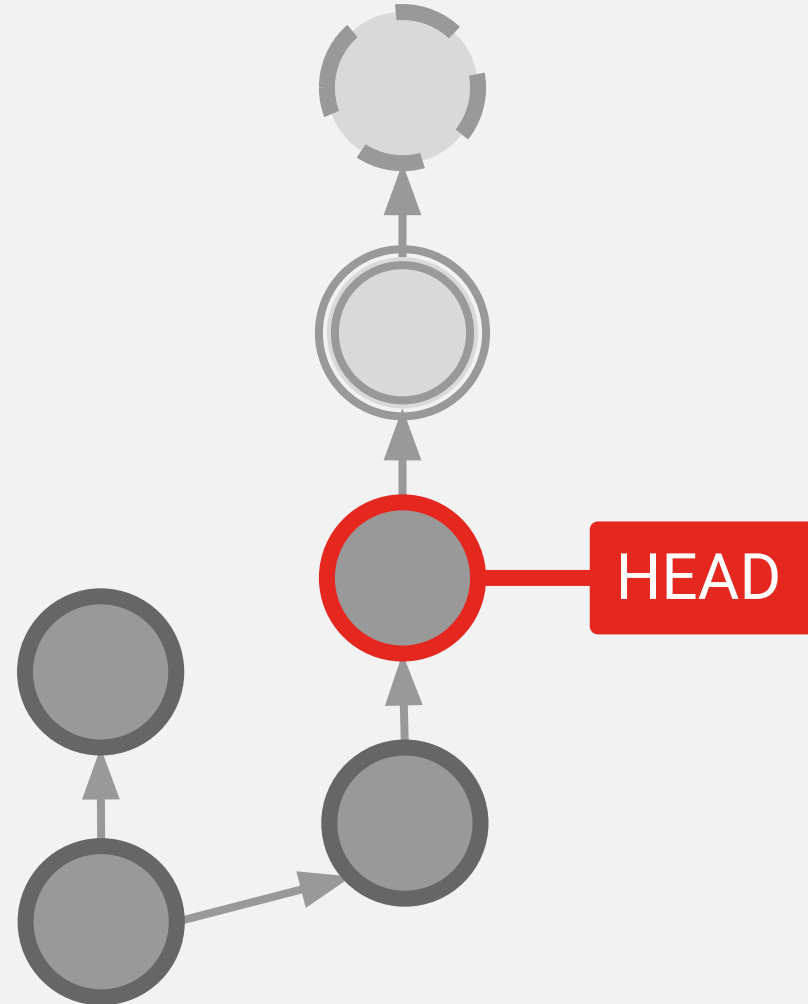
checkout — перемещение на ветку или коммит,
reset — перемещение с веткой на коммит

Видно то, на что есть ссылки, остальное — мусор

HEAD – точка приложения усилий

Указывает на коммит, относительно которого выполняются операции

- с ним связан Commit index и Working directory
- в него будет сделан следующий коммит, смещается при коммите
- перемещается при checkout, switch или reset
- относительно него работают merge, rebase и т.д.



Tag

Именованная ссылка, **привязанная к конкретному коммиту**

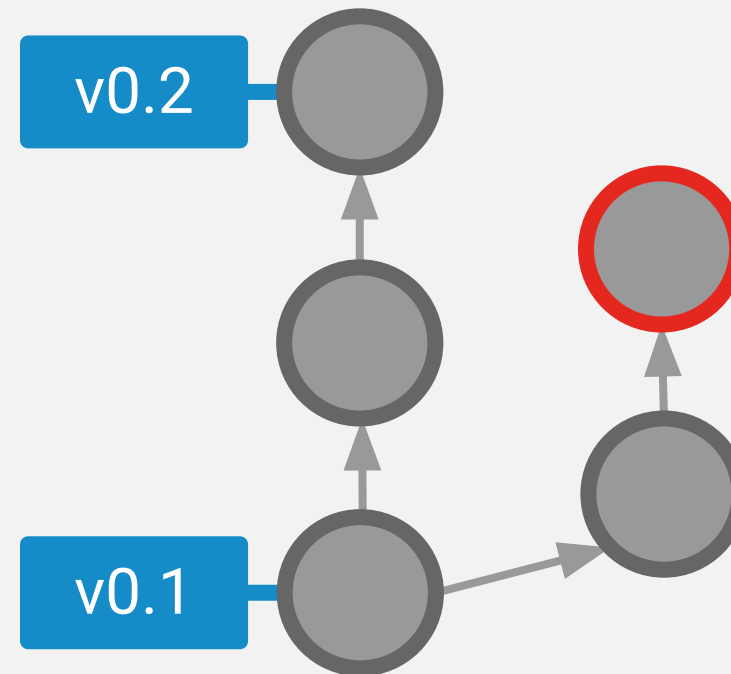
Псевдоним коммита

Применяется **для обозначения версий**

Полезен при манипуляциях над историей

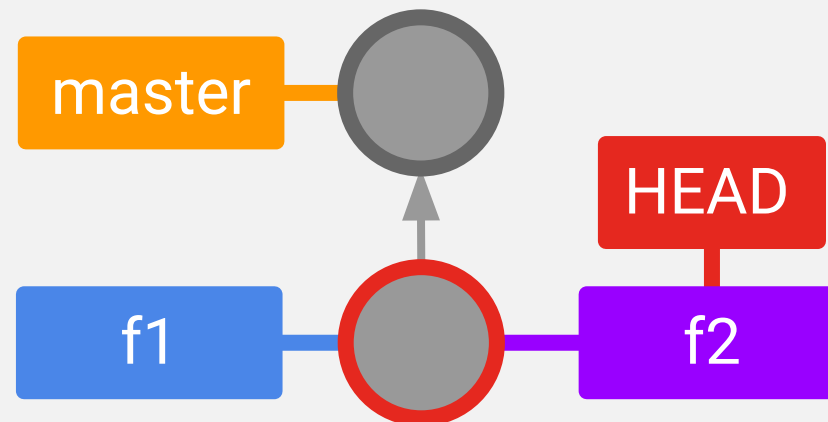
Типы тегов:

- **lightweight** — просто ссылка
- **annotated** — с доп. информацией, в том числе об авторе



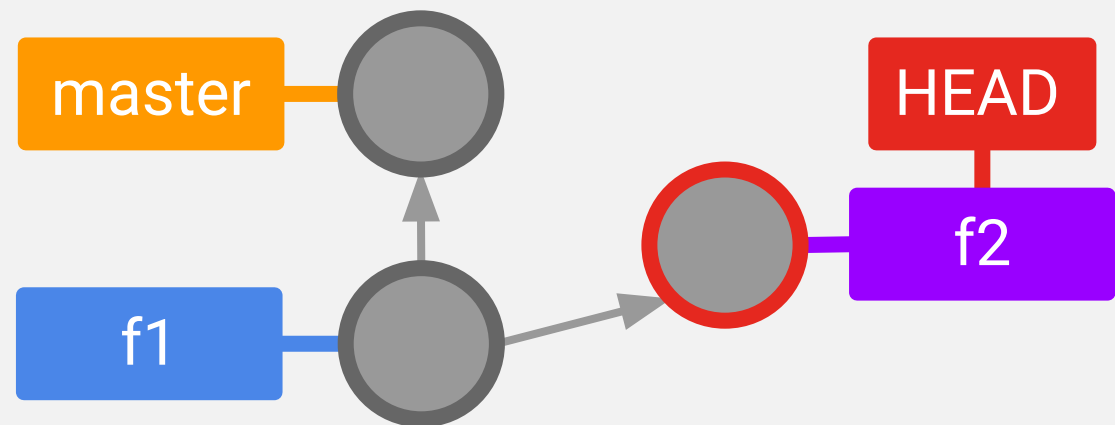
Branch

Движущаяся ссылка, которая сдвигается вместе с HEAD, если тот на нее указывает



Branch

Движущаяся ссылка, которая сдвигается вместе с HEAD, если тот на нее указывает



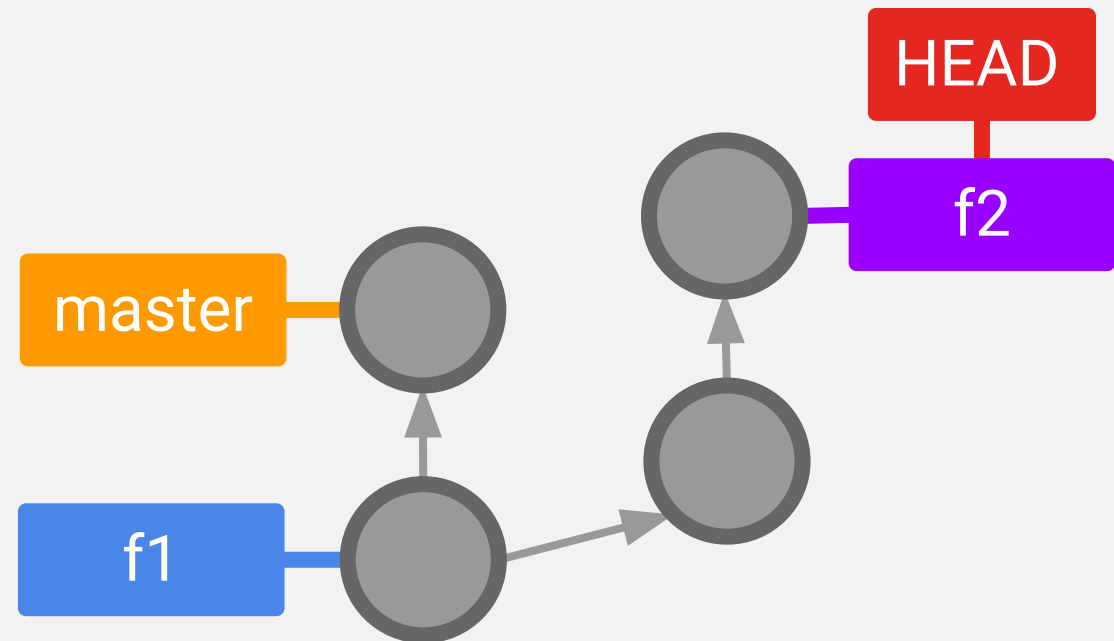
Branch

Движущаяся ссылка, которая сдвигается вместе с HEAD, если тот на нее указывает

Получаемая за этой ссылкой последовательность коммитов **похожа на ветку**

Ветки используются для разработки нового функционала

Главная ветка по соглашению называется **master**

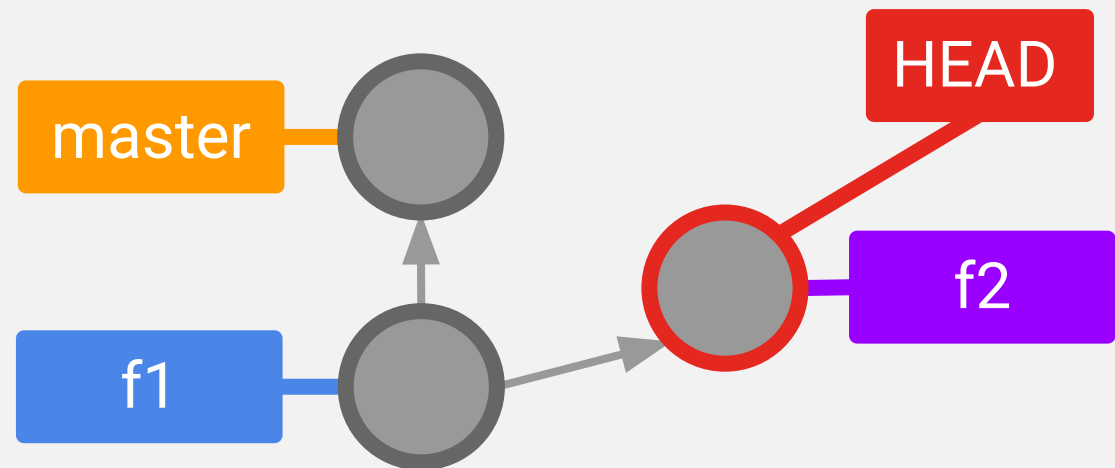


Каждой фиче – отдельная ветка

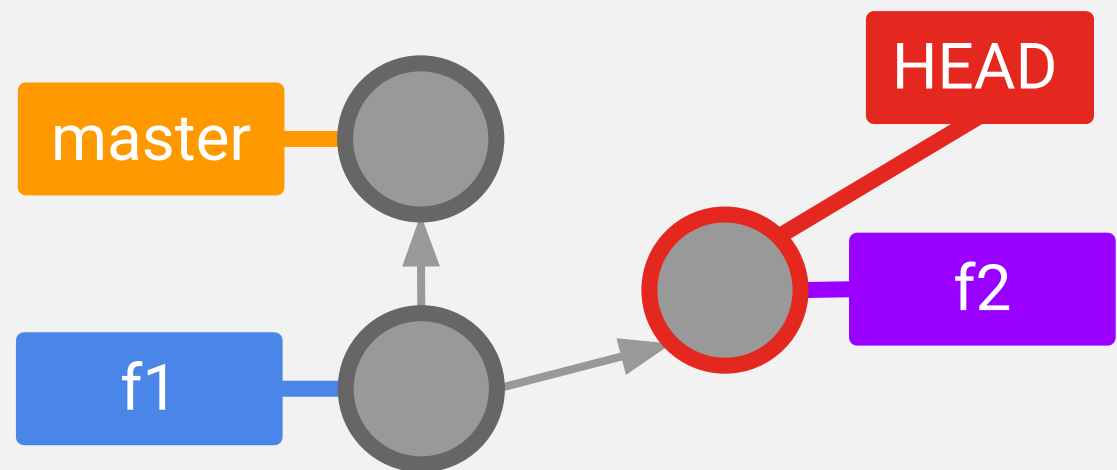
Detached HEAD

HEAD может указывать не только на ветку, но и непосредственно на коммит

В этом случае говорят, что это **detached HEAD**

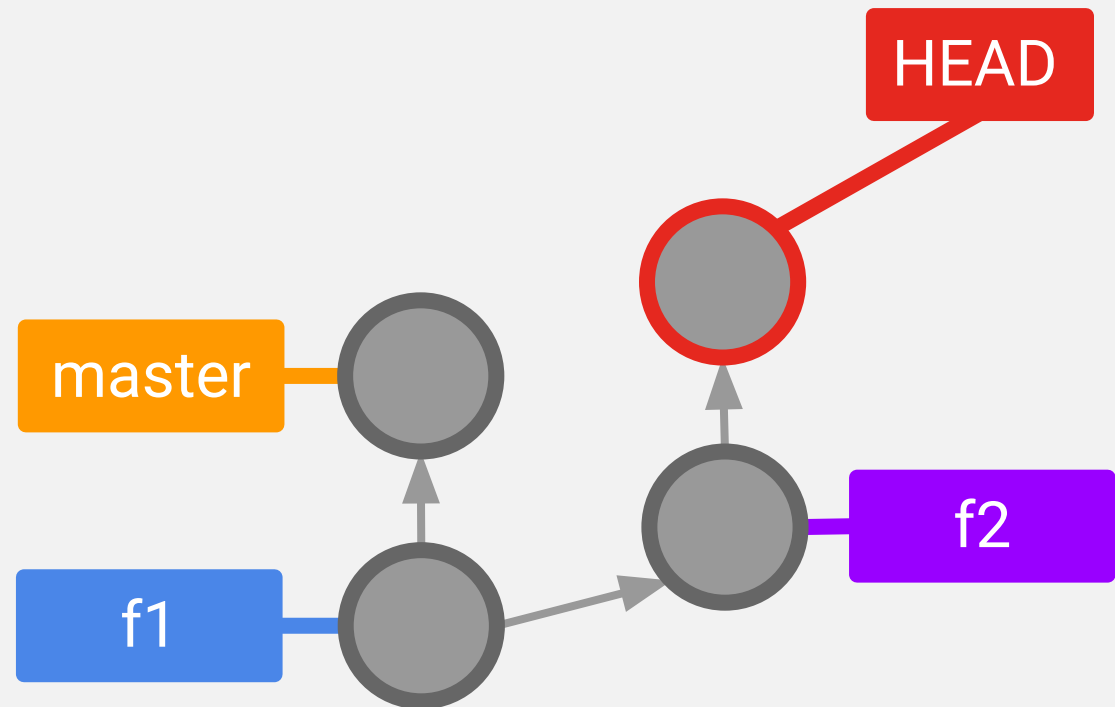


Что произойдет после коммита?

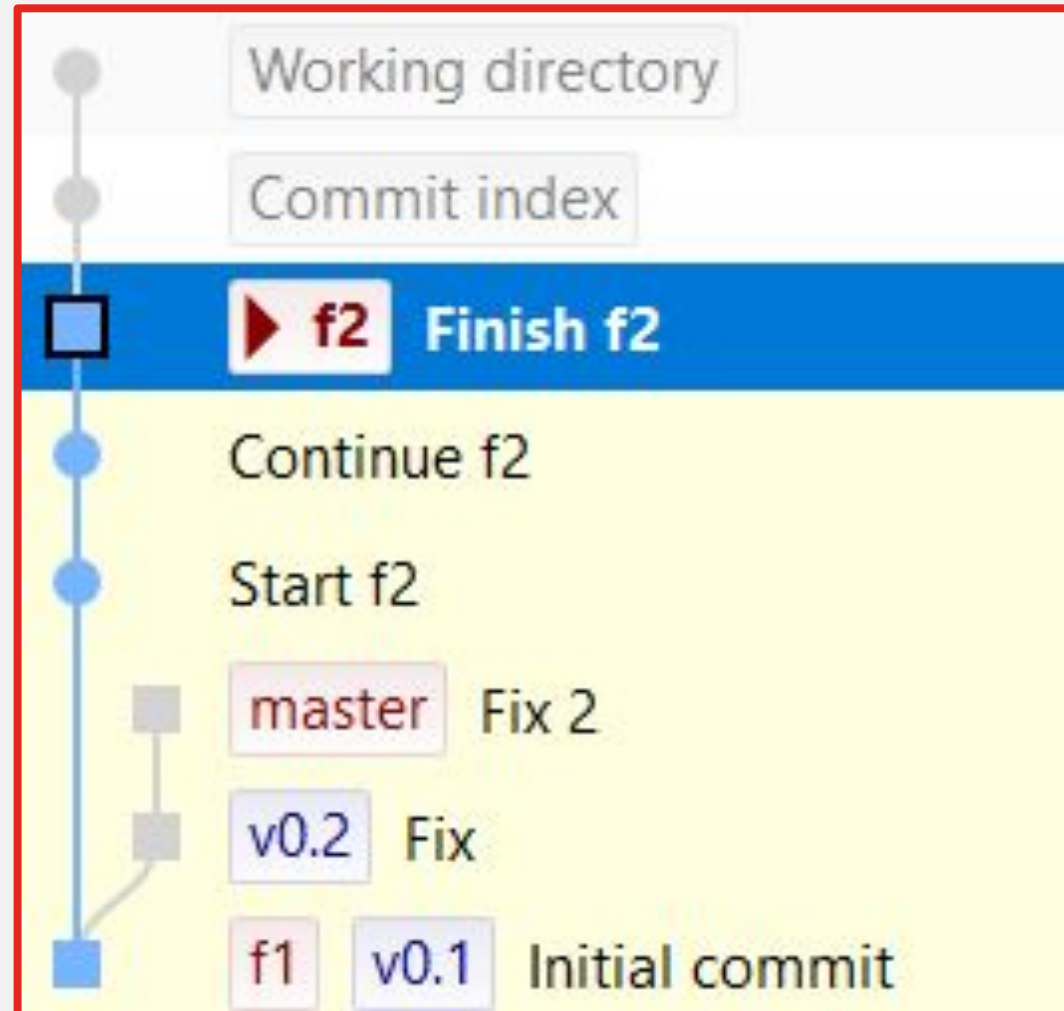


Коммит без движения ветки

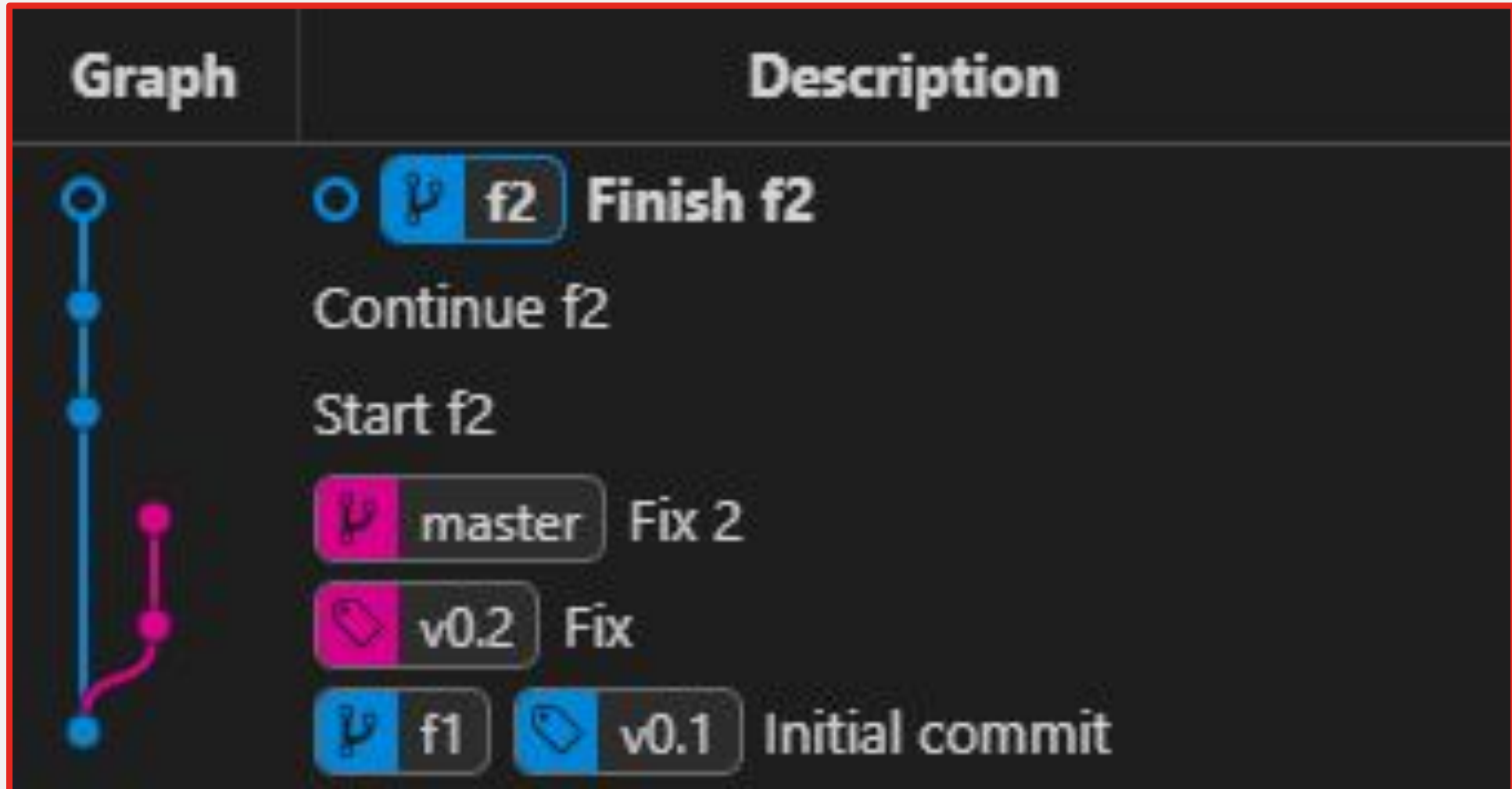
Коммит всегда происходит в HEAD, но если HEAD не указывает ни на одну ветку, то ветки не двигаются



Ветки и теги в Git Extensions



Ветки и теги в Git Graph



Ветки и теги в Git Bash

```
D:\Users\digi\Desktop\Repos\git-rules>git log --oneline --decorate --graph --all
* e1b0448 (HEAD -> f2) Finish f2
* f966ff1 Continue f2
* 00f798b Start f2
| * 2136ddc (master) Fix 2
| * 16f2deb (tag: v0.2) Fix
|/
* 7bdd3ae (tag: v0.1, f1) Initial commit
```

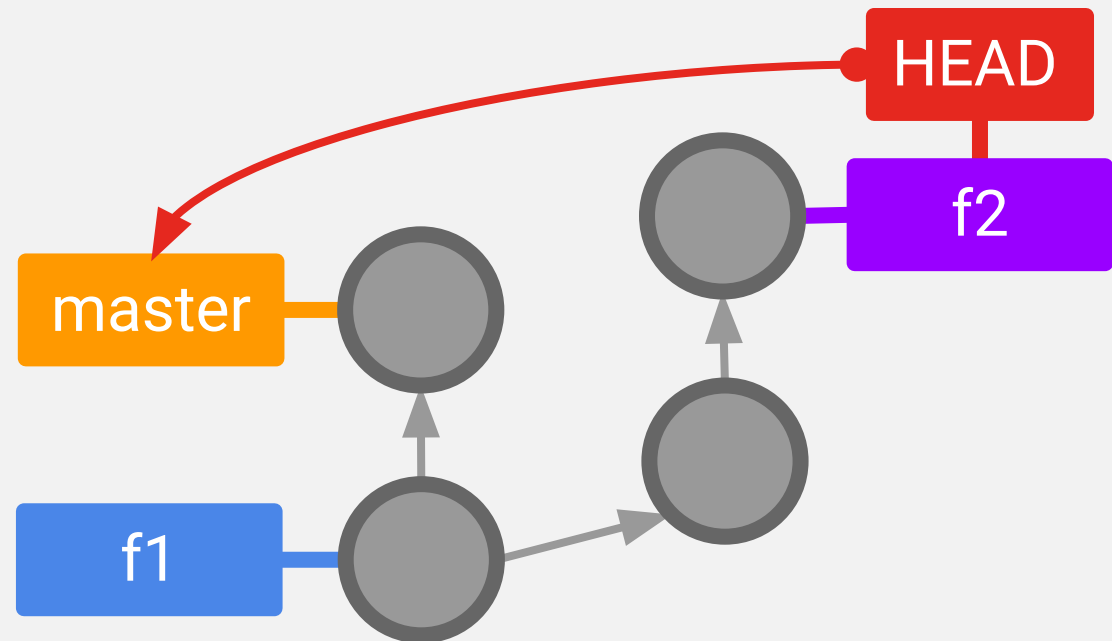
Удобный алиас

```
git config --global alias.graph "log --oneline --decorate --graph --all"
```



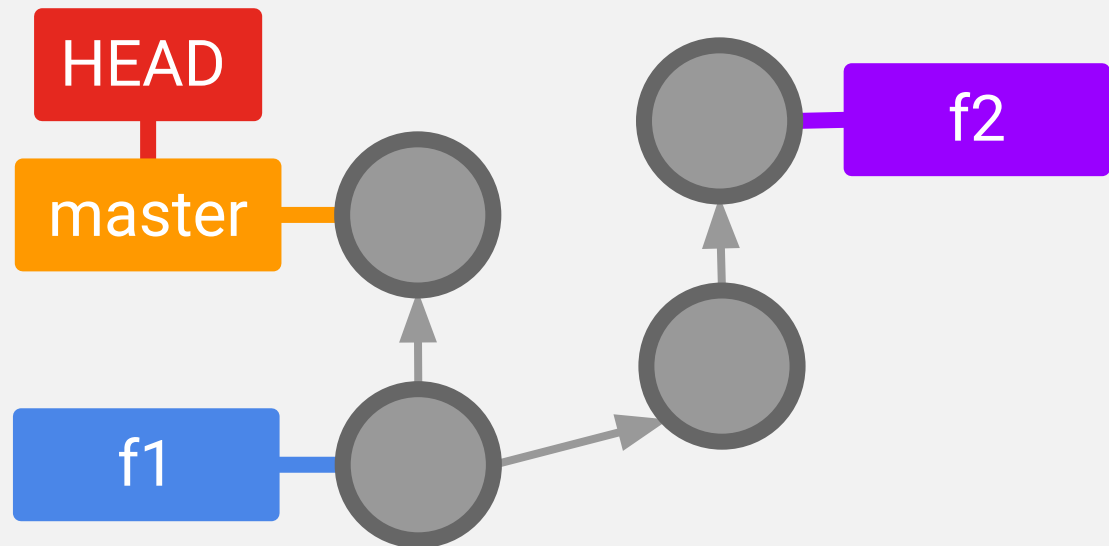
Checkout на ветку

Команда checkout может перенести HEAD на какую-нибудь ветку



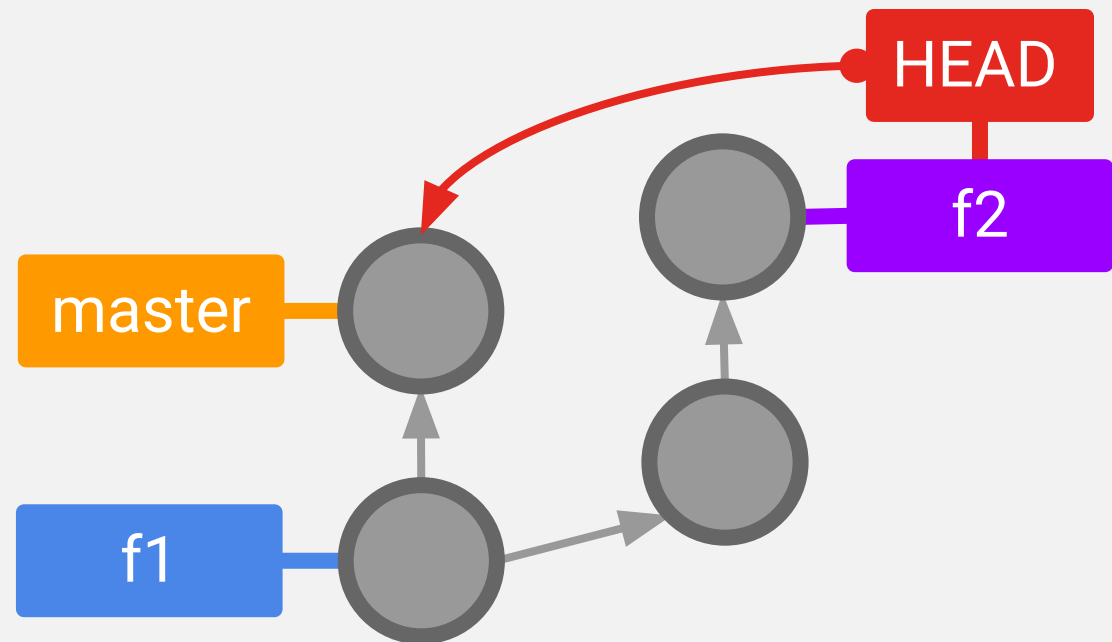
Checkout на ветку

Команда checkout может перенести HEAD на какую-нибудь ветку



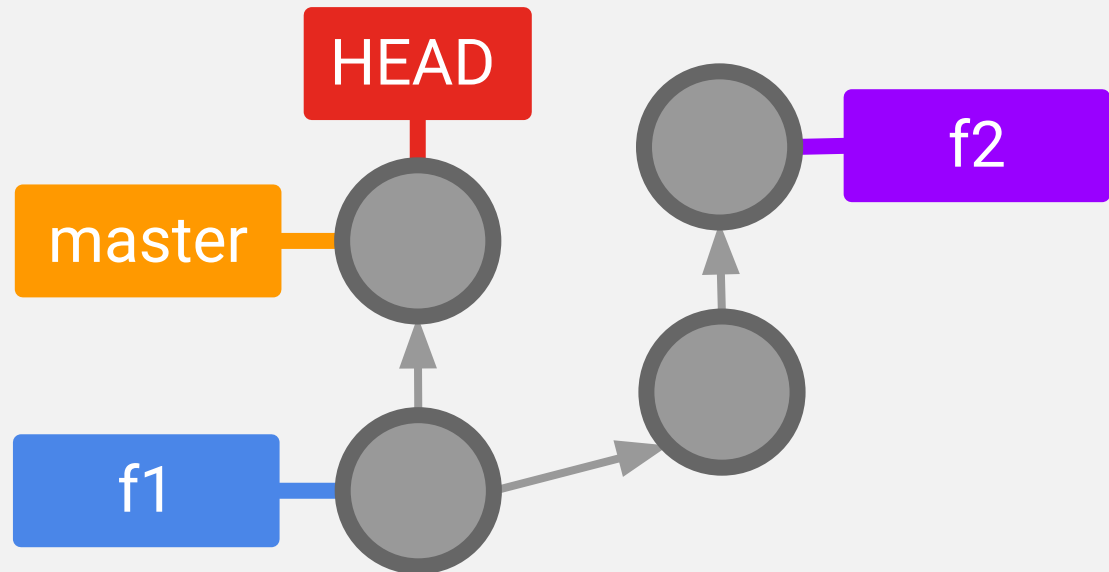
Checkout на коммит

Команда checkout может перенести HEAD на любой коммит, причем получится detached HEAD



Checkout на коммит

Команда checkout может перенести HEAD на любой коммит, причем получится detached HEAD

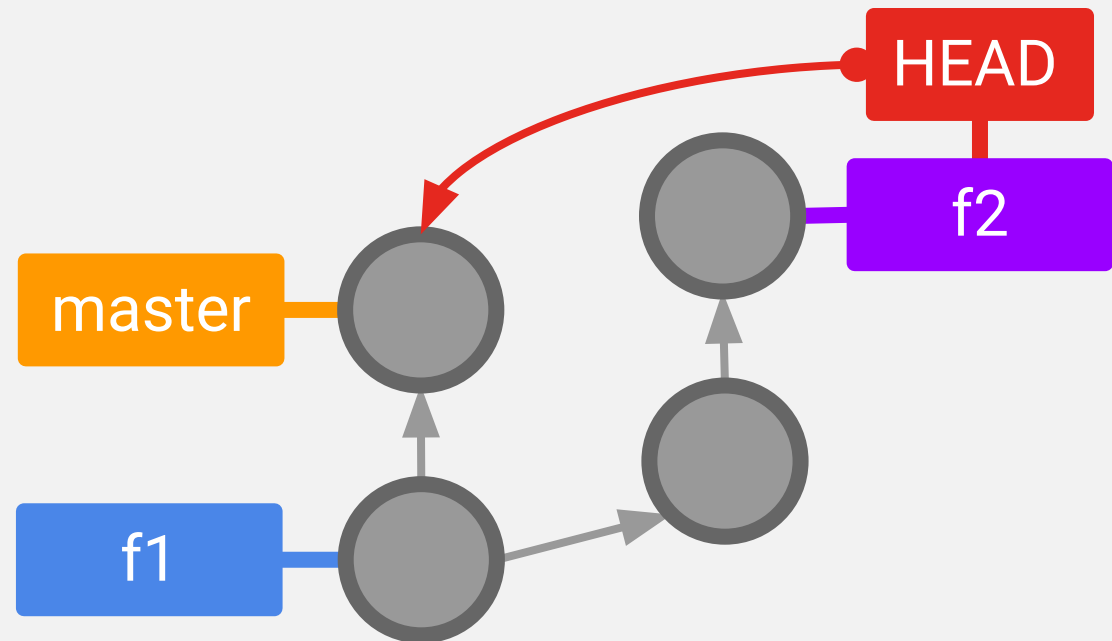


Checkout старается перенести
локальные изменения,
либо не выполняется

Switch — новая команда,
задуманная как замена для checkout

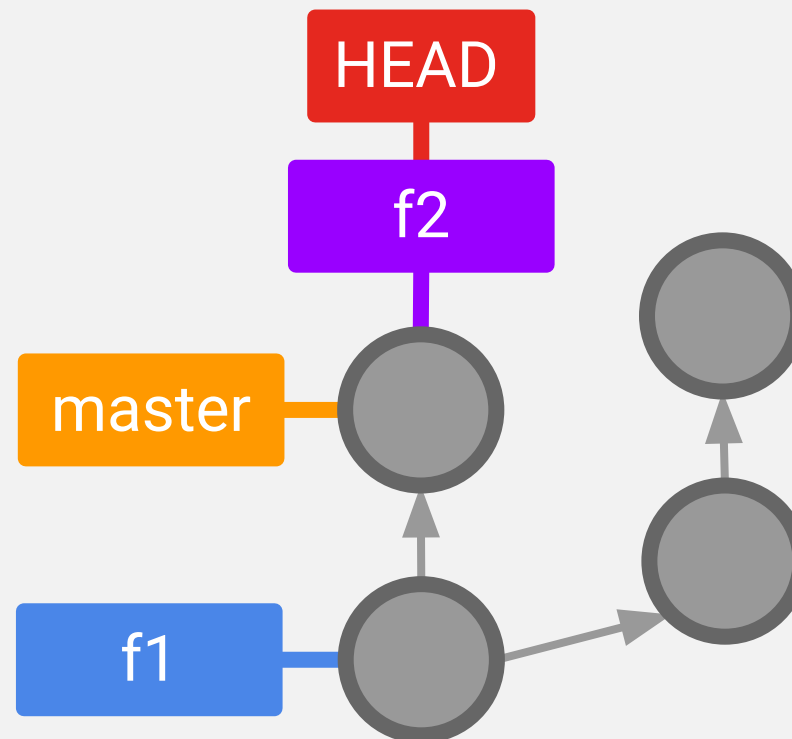
Перенос ветки с reset --hard

Команда reset может перенести HEAD на любой коммит, причем захватывает с собой ветку, на которую указывает HEAD



Перенос ветки с reset --hard

Команда reset может перенести HEAD на любой коммит, причем захватывает с собой ветку, на которую указывает HEAD



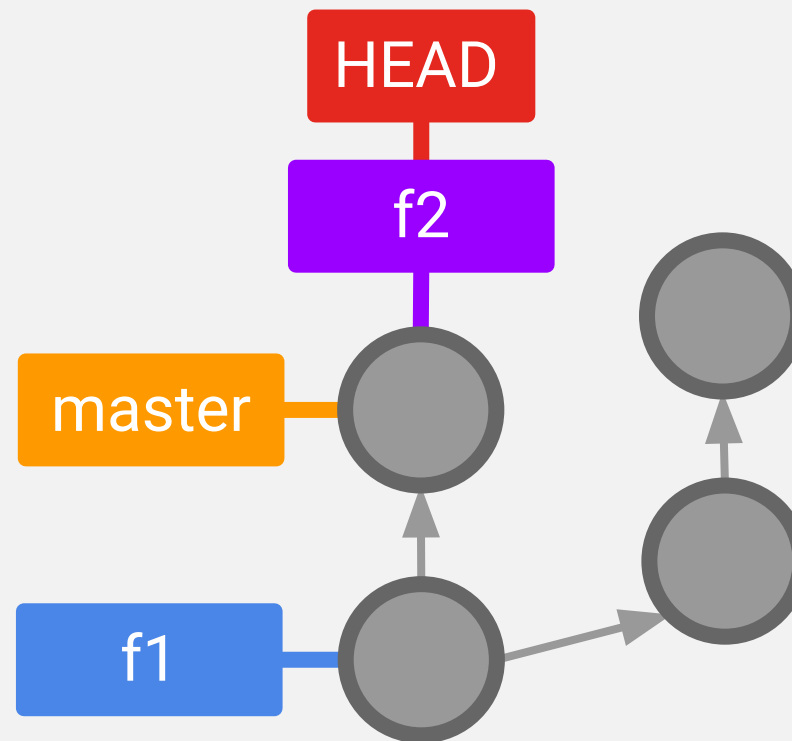
Reset в режиме hard затирает
локальные изменения

Пока этого достаточно!

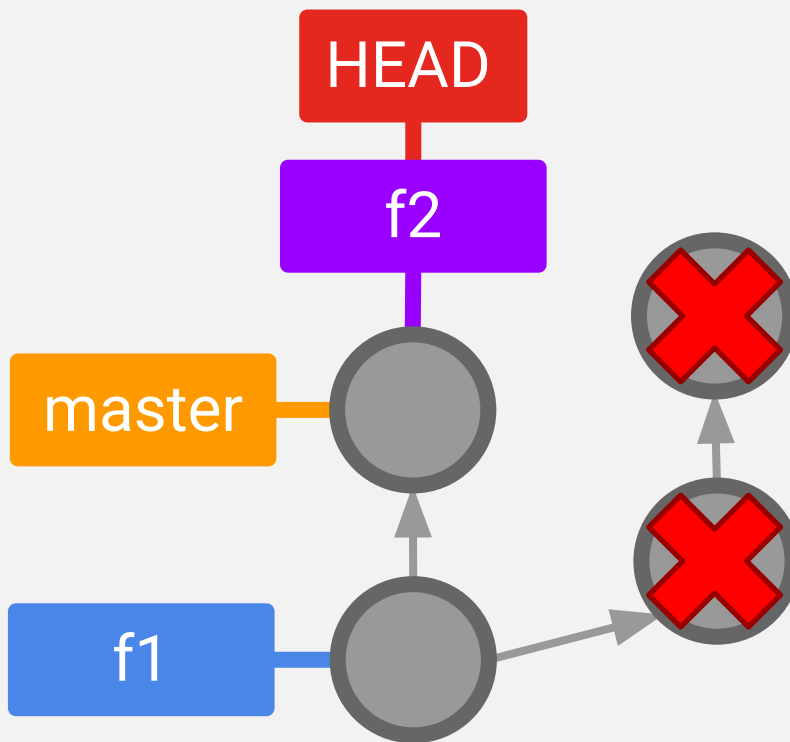
Видно то, на что есть ссылки

- Если на коммит есть ссылка: HEAD, tag, branch – то он показывается, а иначе скрывается
- Если нет ссылок, то коммит будет удален через 30 суток
- `git gc` вызывает ручную очистку ненужного

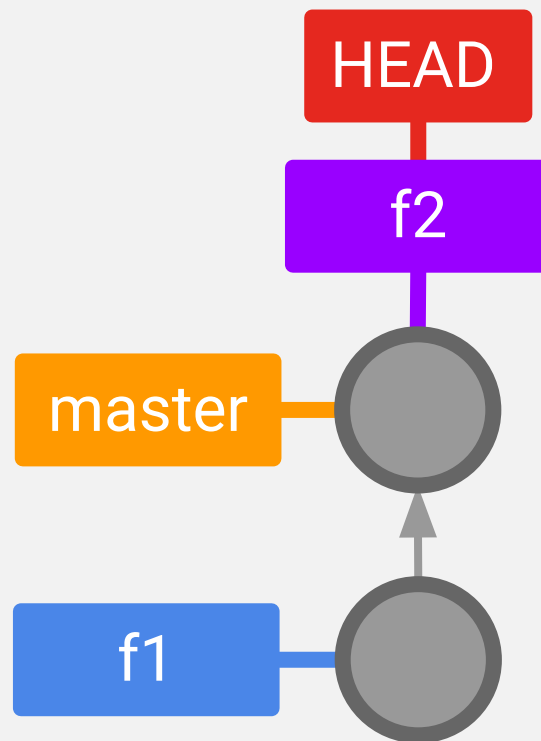
Помните выполнение `reset --hard`?



Вот это уже мусор



Выглядеть будет так



Но все же...

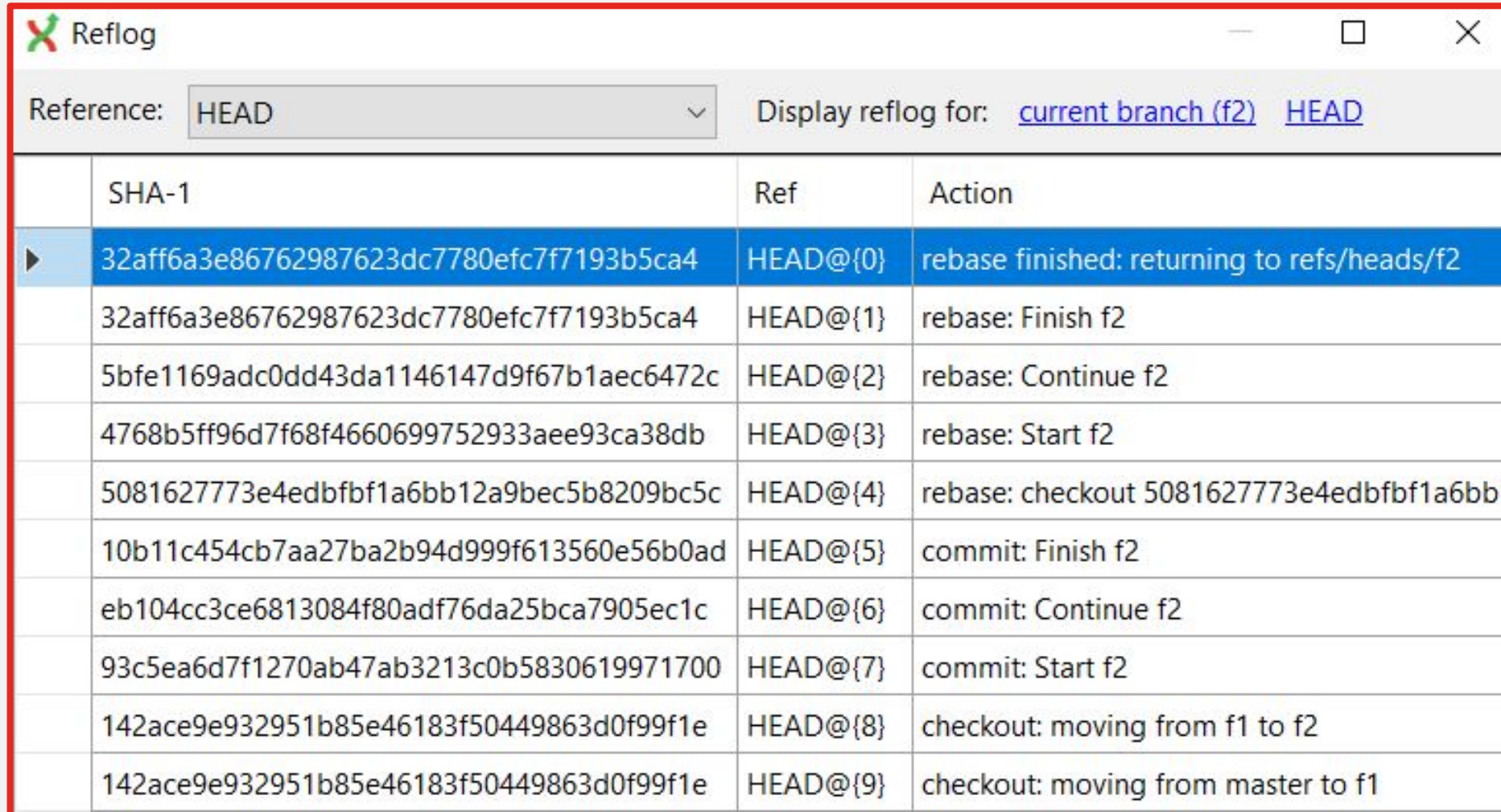
Если **потерялся коммит** в ходе манипуляций,
то он не удален и **его можно найти**

Если **закоммичено** – не потеряете



Все перемещения ссылок логируются
и хэши всех видимых когда-либо
ревизий оседают в этих логах

Reflog в Git Extensions



The screenshot shows the 'Reflog' window in Git Extensions. The window title is 'Reflog'. At the top, there is a 'Reference:' dropdown menu set to 'HEAD' and a 'Display reflog for:' section with two options: 'current branch (f2)' and 'HEAD', both of which are underlined. Below this is a table with four columns: 'SHA-1', 'Ref', and 'Action'. The first column is labeled 'SHA-1' and has a small triangle icon to its left. The table contains 10 rows of data, with the top row highlighted in blue.

SHA-1	Ref	Action
32aff6a3e86762987623dc7780efc7f7193b5ca4	HEAD@{0}	rebase finished: returning to refs/heads/f2
32aff6a3e86762987623dc7780efc7f7193b5ca4	HEAD@{1}	rebase: Finish f2
5bfe1169adc0dd43da1146147d9f67b1aec6472c	HEAD@{2}	rebase: Continue f2
4768b5ff96d7f68f4660699752933aee93ca38db	HEAD@{3}	rebase: Start f2
5081627773e4edbfbf1a6bb12a9bec5b8209bc5c	HEAD@{4}	rebase: checkout 5081627773e4edbfbf1a6bb
10b11c454cb7aa27ba2b94d999f613560e56b0ad	HEAD@{5}	commit: Finish f2
eb104cc3ce6813084f80adf76da25bca7905ec1c	HEAD@{6}	commit: Continue f2
93c5ea6d7f1270ab47ab3213c0b5830619971700	HEAD@{7}	commit: Start f2
142ace9e932951b85e46183f50449863d0f99f1e	HEAD@{8}	checkout: moving from f1 to f2
142ace9e932951b85e46183f50449863d0f99f1e	HEAD@{9}	checkout: moving from master to f1

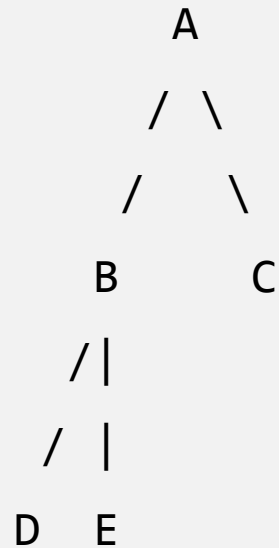
Reflog в консоли

```
$ git reflog
38e8c1c (HEAD -> f2) HEAD@{0}: rebase finished: returning to refs/heads/f2
38e8c1c (HEAD -> f2) HEAD@{1}: rebase: Finish f2
160b3f3 HEAD@{2}: rebase: Continue f2
14c5d89 HEAD@{3}: rebase: Start f2
b9e3b60 (master) HEAD@{4}: rebase: checkout master
c11810f HEAD@{5}: commit: Finish f2
b6b258e HEAD@{6}: commit: Continue f2
fce2e46 HEAD@{7}: commit: Start f2
84a27c0 (tag: v0.1, f1) HEAD@{8}: checkout: moving from f1 to f2
84a27c0 (tag: v0.1, f1) HEAD@{9}: checkout: moving from master to f1
```

Как ничего не терять?

1. Отмечать дорогие сердцу коммиты тегами перед сложными манипуляциями
2. Помнить про особенность `git log`. По умолчанию показывает предков HEAD, а не все коммиты
3. В крайнем случае использовать `reflog`

Относительная адресация коммитов



$$A = A^{\wedge 0}$$

$$B = A^{\wedge 1} = A^{\sim 1}$$

$$C = A^{\wedge 2}$$

$$D = A^{\wedge 1^{\wedge 1}} = A^{\wedge \wedge} = A^{\sim 2}$$

$$E = B^{\wedge 2} = A^{\wedge \wedge 2}$$

$\sim N$ – 1-ый родитель из N-ого поколения

$\wedge N$ – N-ый родитель предыдущего поколения

А это точно надо запомнить?

А это точно надо запомнить?

Нет!

Удобный алиас

```
git config --global alias.undo "reset --soft HEAD^"
```

Эта команда отменяет commit, но про режим soft будет позже...



Задание 4. Tag

Задание 5. Feature Branches

Structure

Actions

Remote

S1. Все локально

S2. Хранятся
состояния директории,
постепенная сборка коммита

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

H1. Гибкое конфигурирование и качественная документация

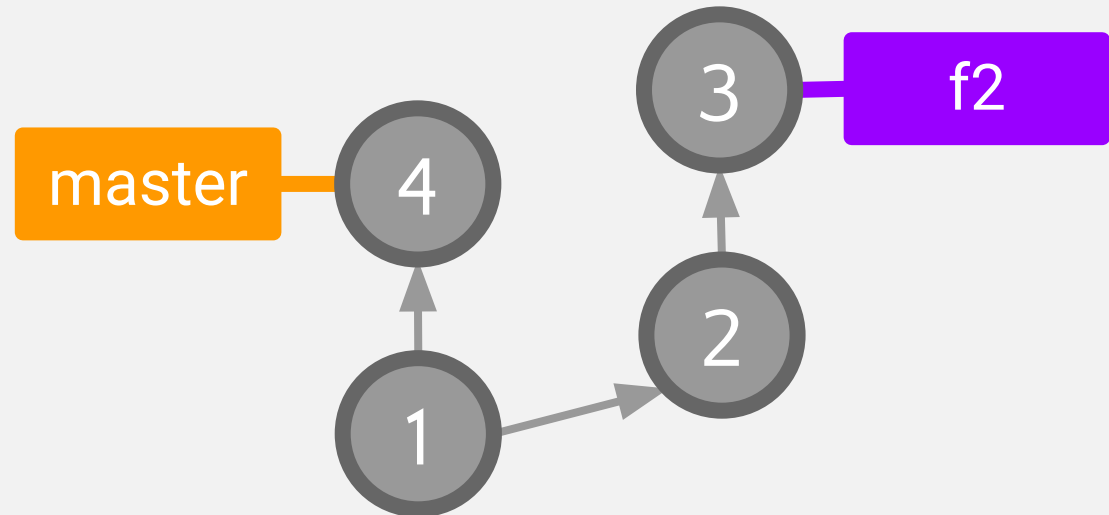
A1. Трехсторонний merge в три шага

Два состояния можно объединить через merge, mergetool и commit

Участвуют три стороны: current, incoming и base

Слияние

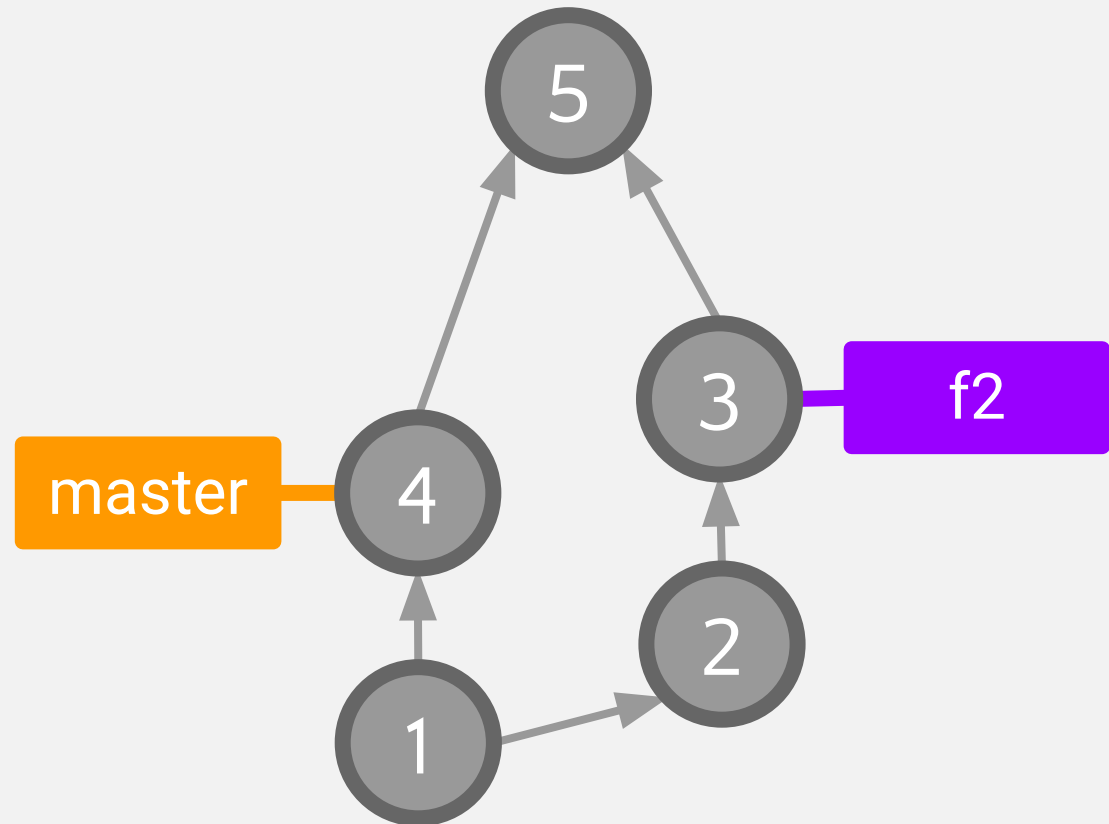
Разрабатывать в отдельных ветках правильно, но в конце концов надо **объединить функционал в одной версии**



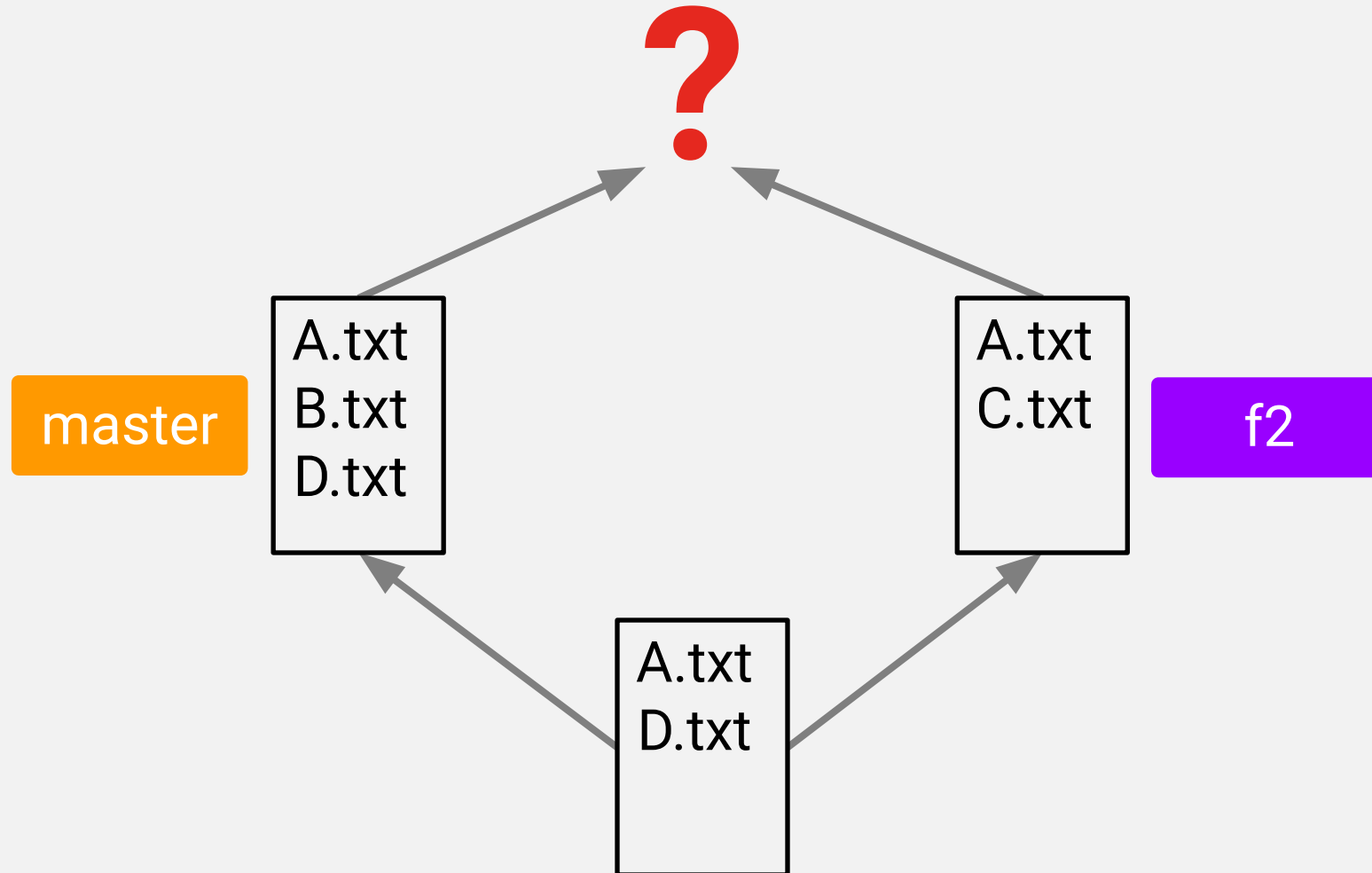
Слияние

Разрабатывать в отдельных ветках правильно, но в конце концов надо **объединить функционал в одной версии**

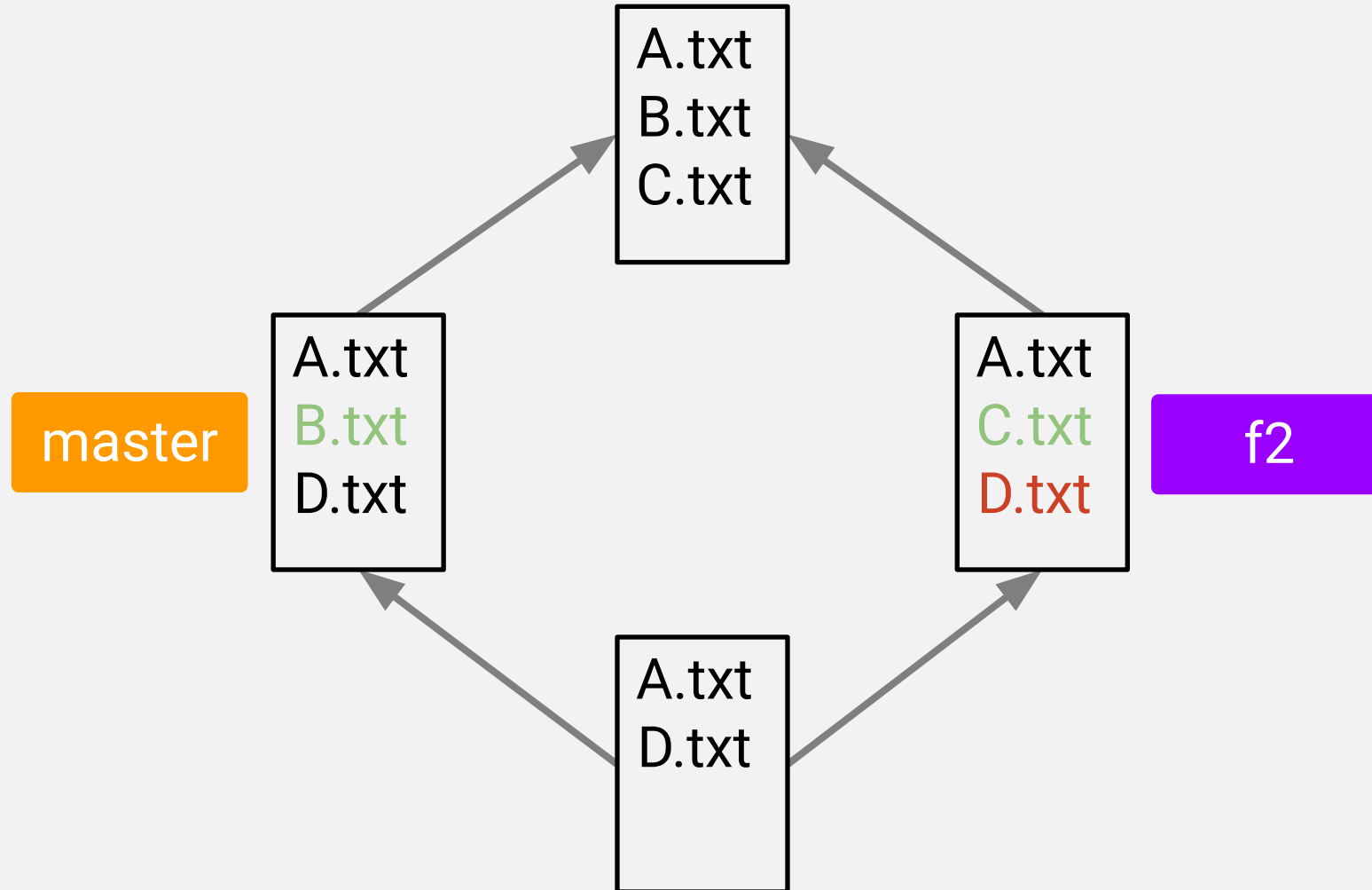
Значит надо получить новое состояние – КОММИТ



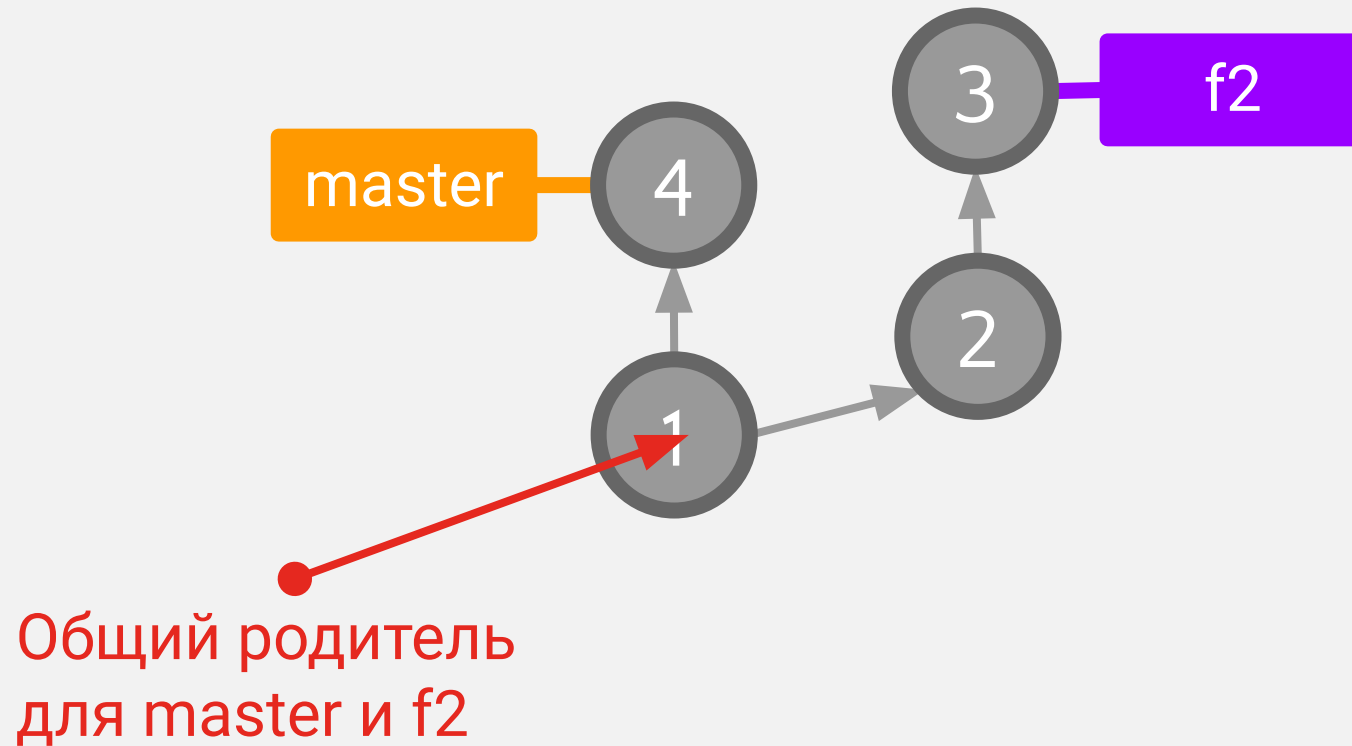
Как объединить изменения?



Как объединить изменения?

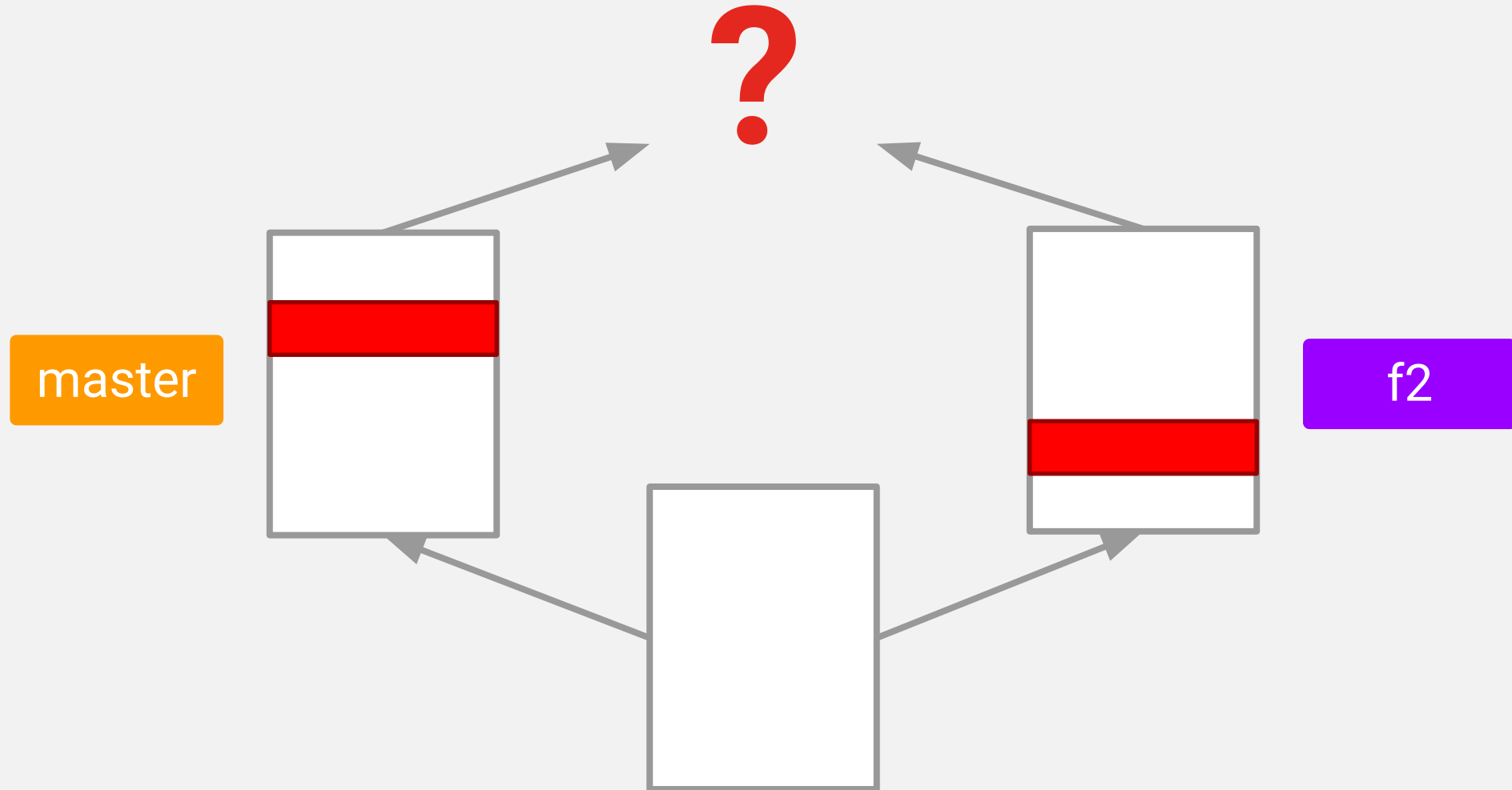


Общий родитель

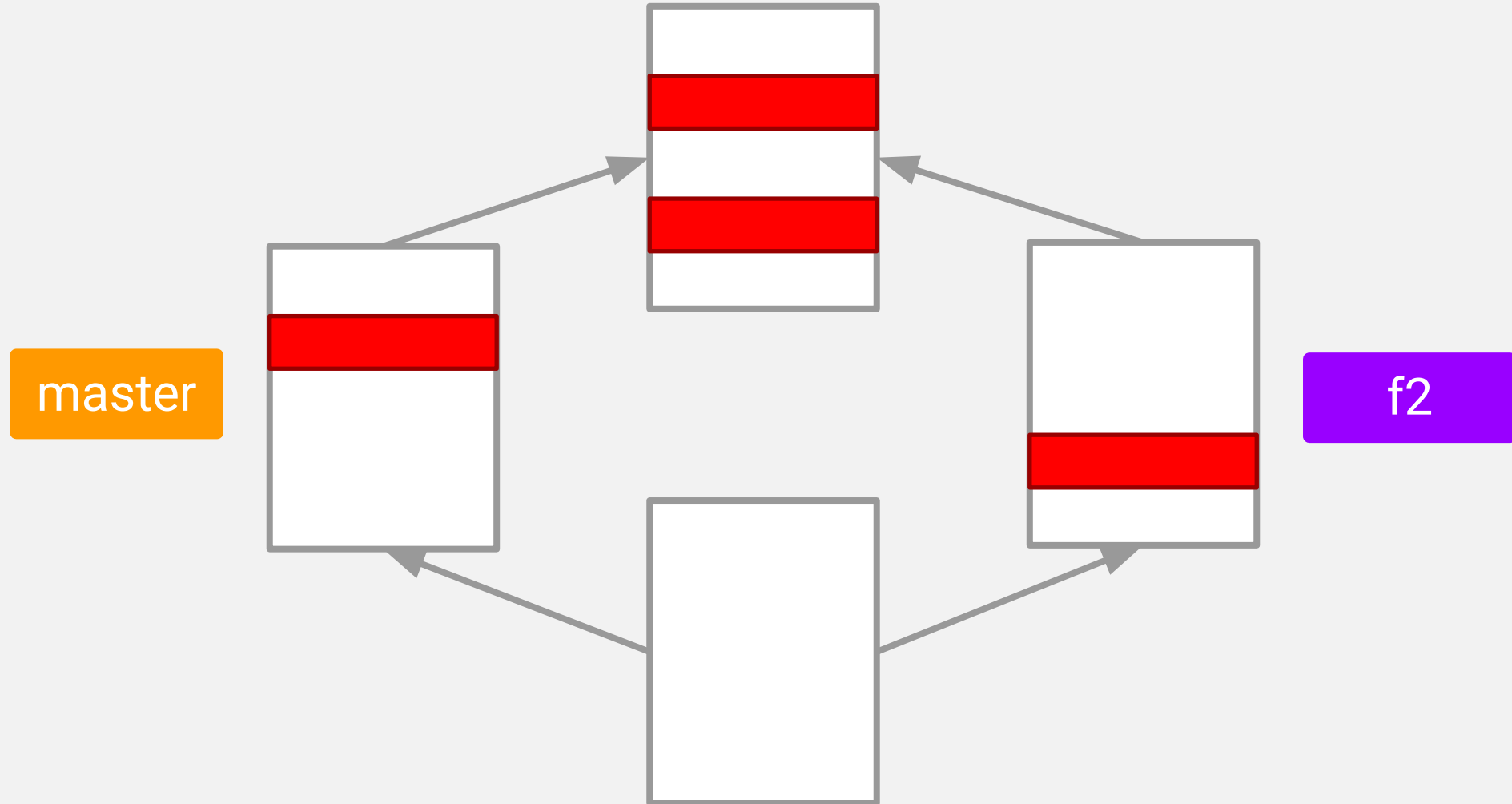


А если был изменен
один и тот же файл?

Как объединить изменения?



Как объединить изменения?



А если одна и та же строчка?

Конфликт

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

Конфликт

```
<<<<<< HEAD:index.html
```

```
<div id="footer">contact : email.support@github.com</div>
```

```
=====
```

```
<div id="footer">
```

```
  please contact us at support@github.com
```

```
</div>
```

```
>>>>>> f2:index.html
```

Создается текстовый блок,
содержащий изменения из обоих коммитов

Разрешение конфликтов

Необходимо заменить все «объединенные» текстовые блоки на правильное содержимое

Стратегии

1. Взять один вариант
2. Взять второй вариант
3. Взять оба варианта последовательно
4. Написать что-то совершенно иное

Разрешение конфликта в VS Code

```
213 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
214 <<<<<< HEAD (Current Change)
215     readFileAndGenerating(file, split);
216 =====
217     readFileAndGenerate(input, split);
218 >>>>>> master (Incoming Change)
219
220     i18nFileGenerate(output, options || null);
221
```


В конфликтах
HEAD указывает на **current**
Другая ветка — это **incoming**

При merge **HEAD** указывает на ветку,
в которую вливается другая ветка

Current и Incoming

```
213 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
214 <<<<<< HEAD (Current Change)
215     readFileAndGenerating(file, split);
216 =====
217     readFileAndGenerate(input, split);
218 >>>>>> master (Incoming Change)
219
220     i18nFileGenerate(output, options || null);
221
```

Как объединять изменения?

- Надо объединять на уровне **списка файлов** и на уровне **содержимого файлов**
- Нужно знать две объединяемые версии, а также **общего предка**
- Часто происходит **автоматическое объединение**
- В остальных случаях необходимо **вручную решать конфликты**

Алгоритм слияния

Действие

1. Объединить изменения
автоматически

2. Разрешить конфликты
вручную

3. Закоммитить результат

Алгоритм слияния

Действие	Git Bash
1. Объединить изменения <i>автоматически</i>	<code>merge <branch></code>
2. Разрешить конфликты <i>вручную</i>	<code>mergetool</code>
3. Закоммитить результат	<code>commit</code>

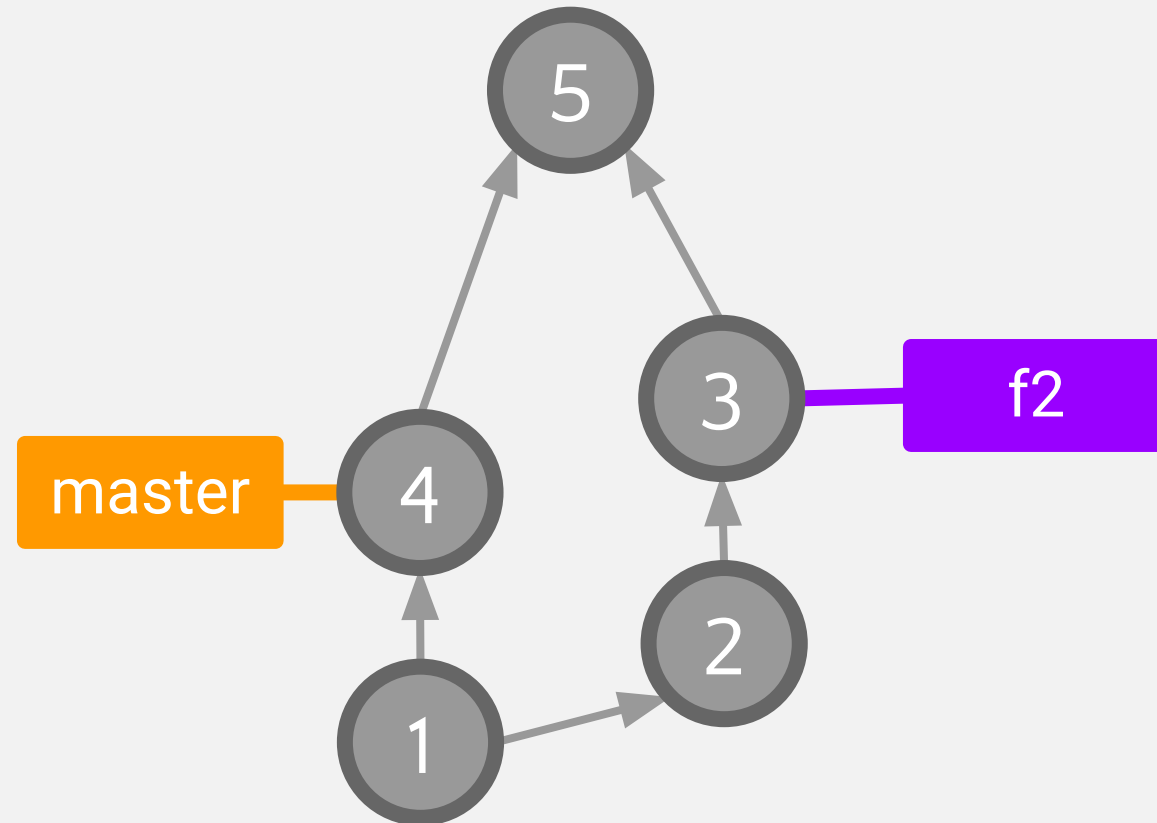
Алгоритм слияния

Действие	Git Bash	Git Extensions
1. Объединить изменения <i>автоматически</i>	<code>merge <branch></code>	Контекстное меню / Merge into current branch
2. Разрешить конфликты <i>вручную</i>	<code>mergetool</code>	Главное меню / Commands / Solve merge conflicts
3. Закоммитить результат	<code>commit</code>	Главное меню / Commands / Commit

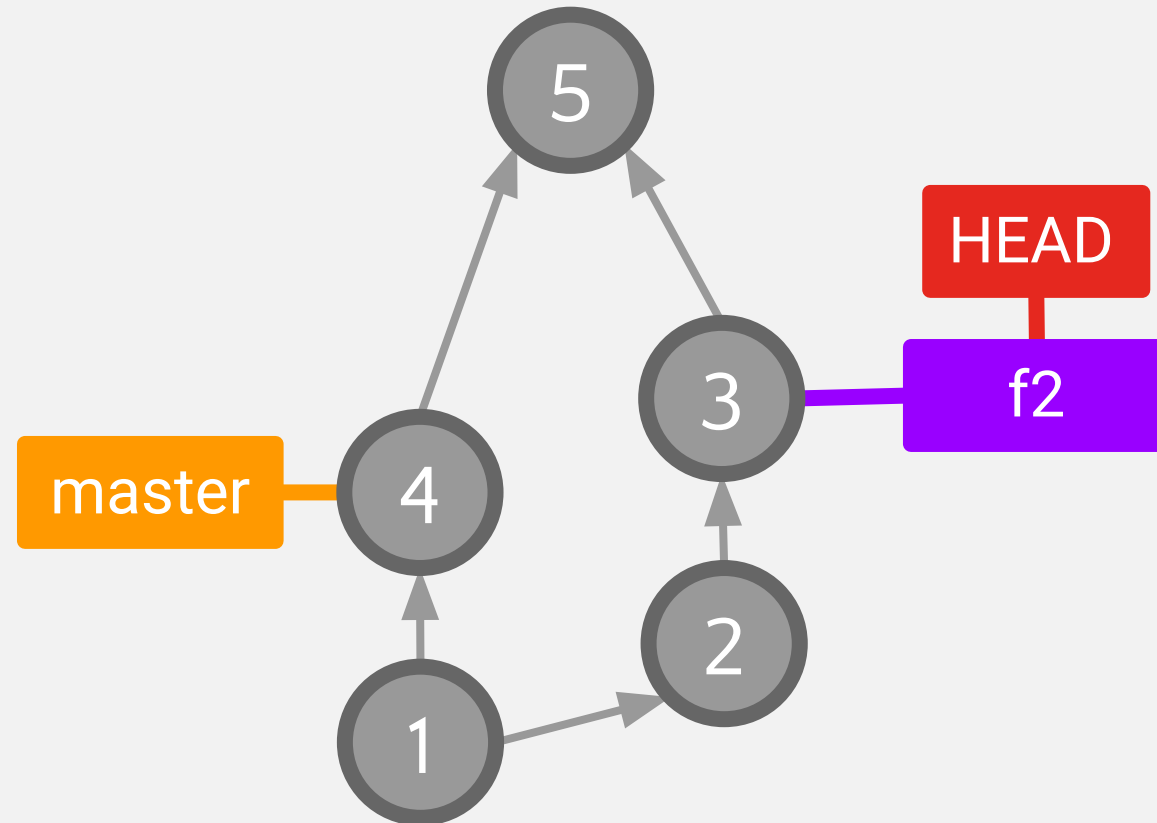
*Git Extensions автоматически
переходит к следующему шагу*

*Можно остановиться,
а затем продолжить*

Как поведут себя ветки?

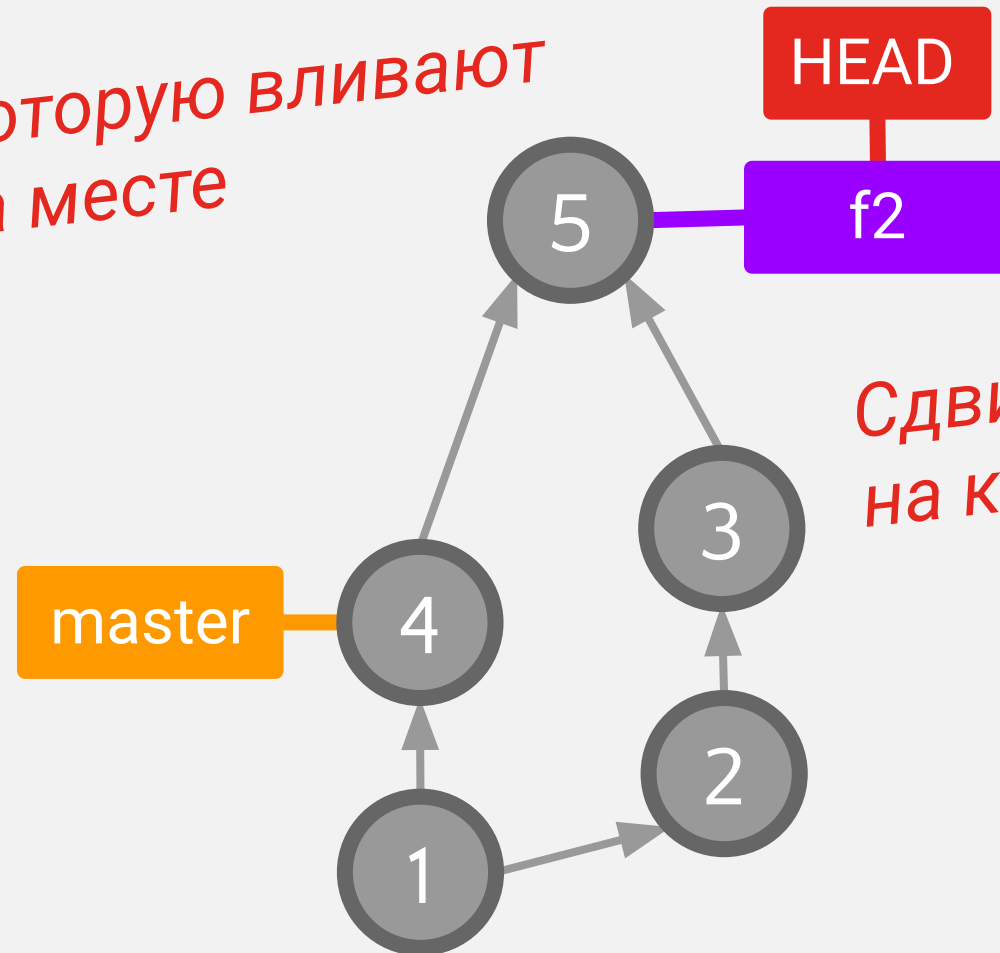


Как поведут себя ветки?



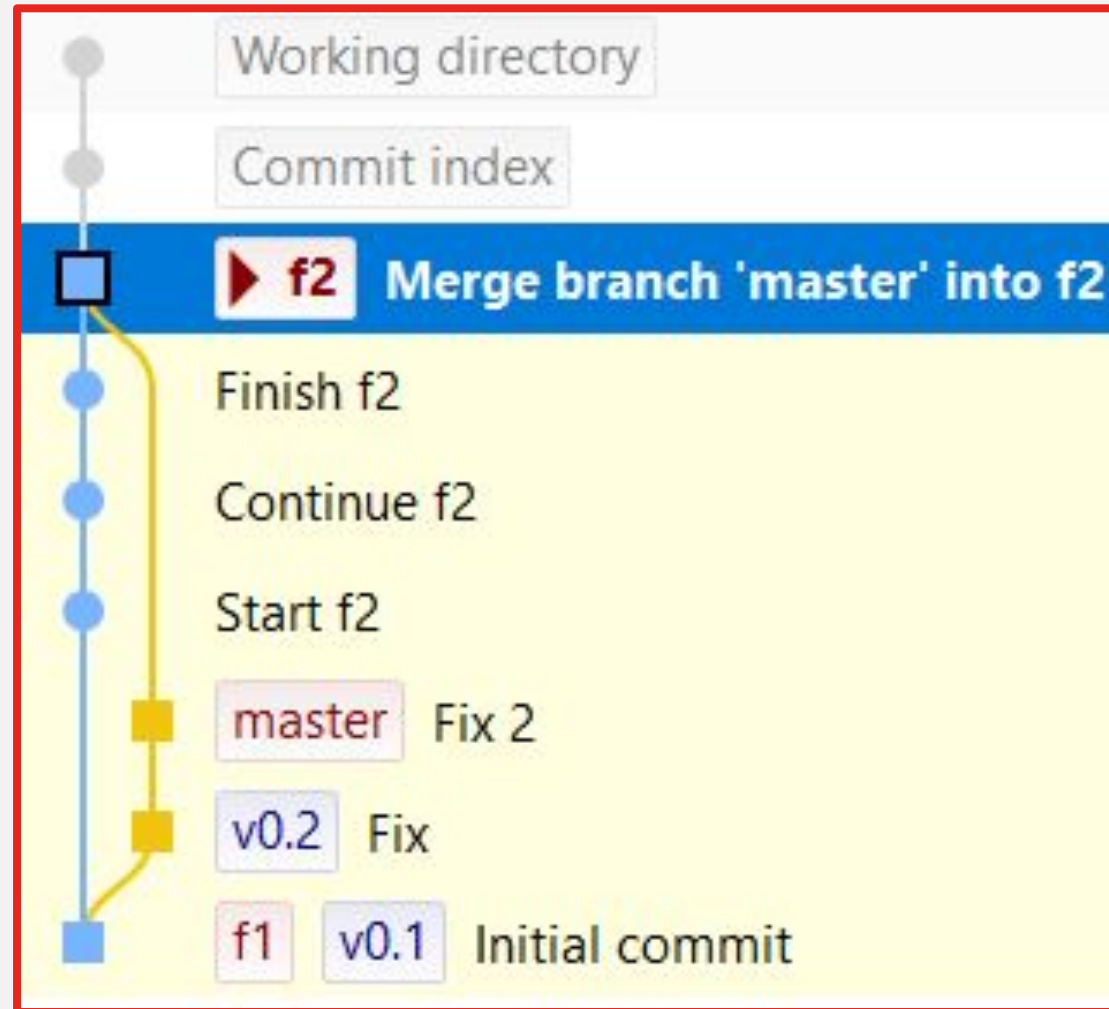
Как поведут себя ветки?

Ветка, которую вливают
стоит на месте

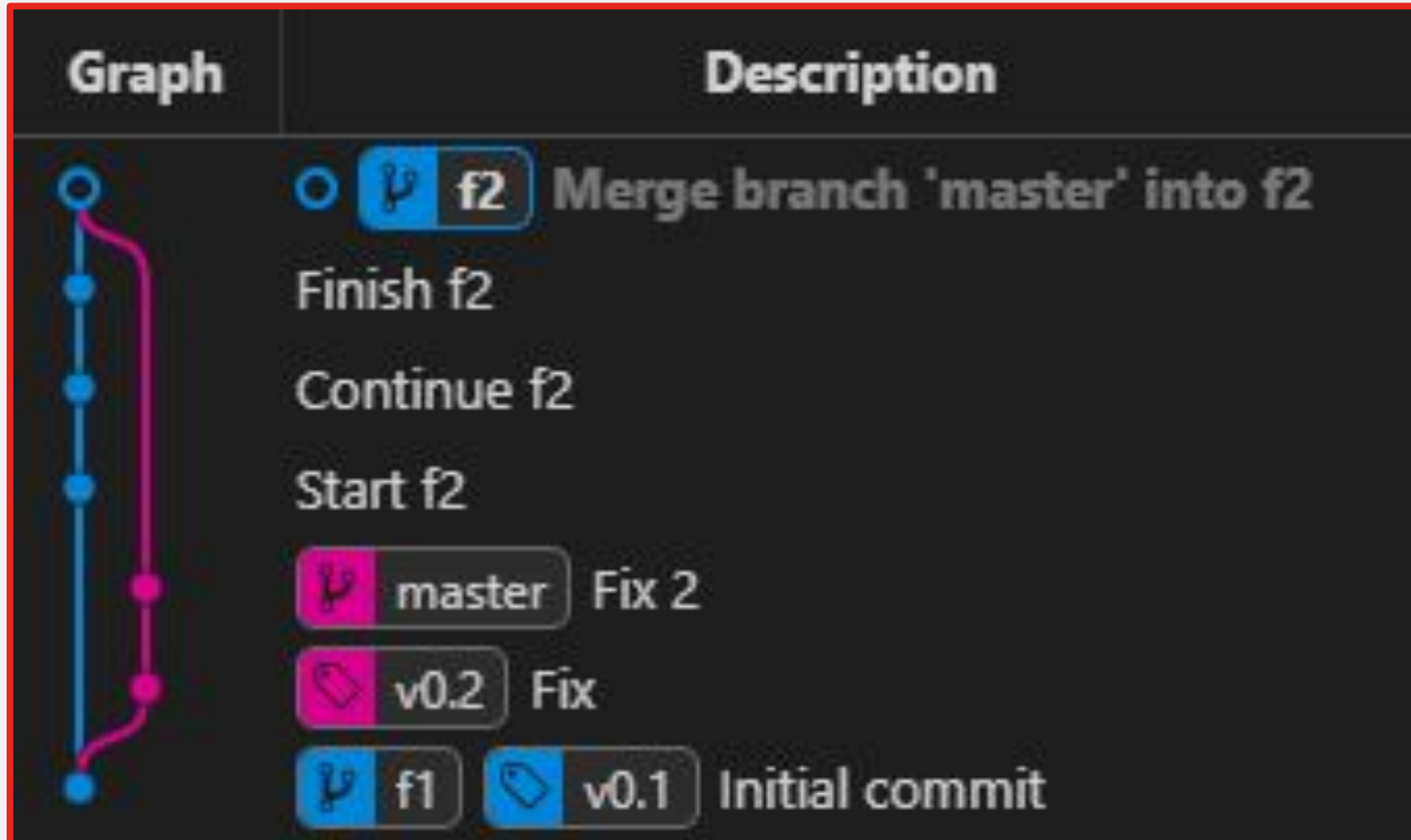


Сдвинется ветка,
на которой был HEAD

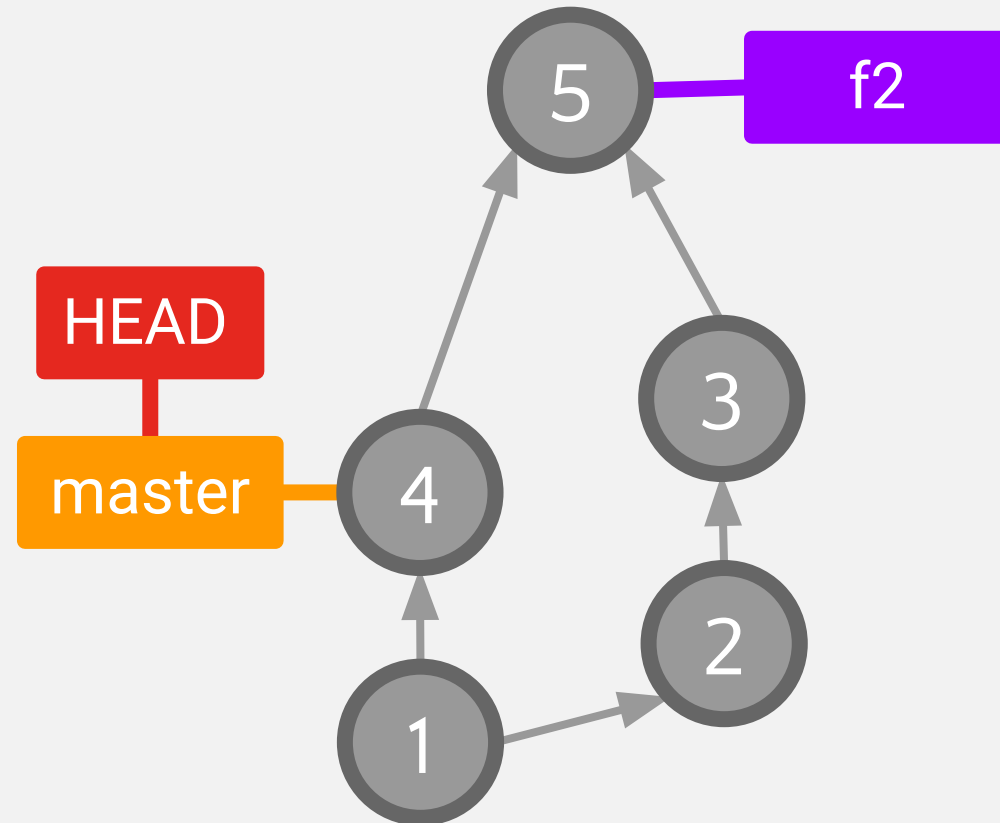
Слияние в Git Extensions



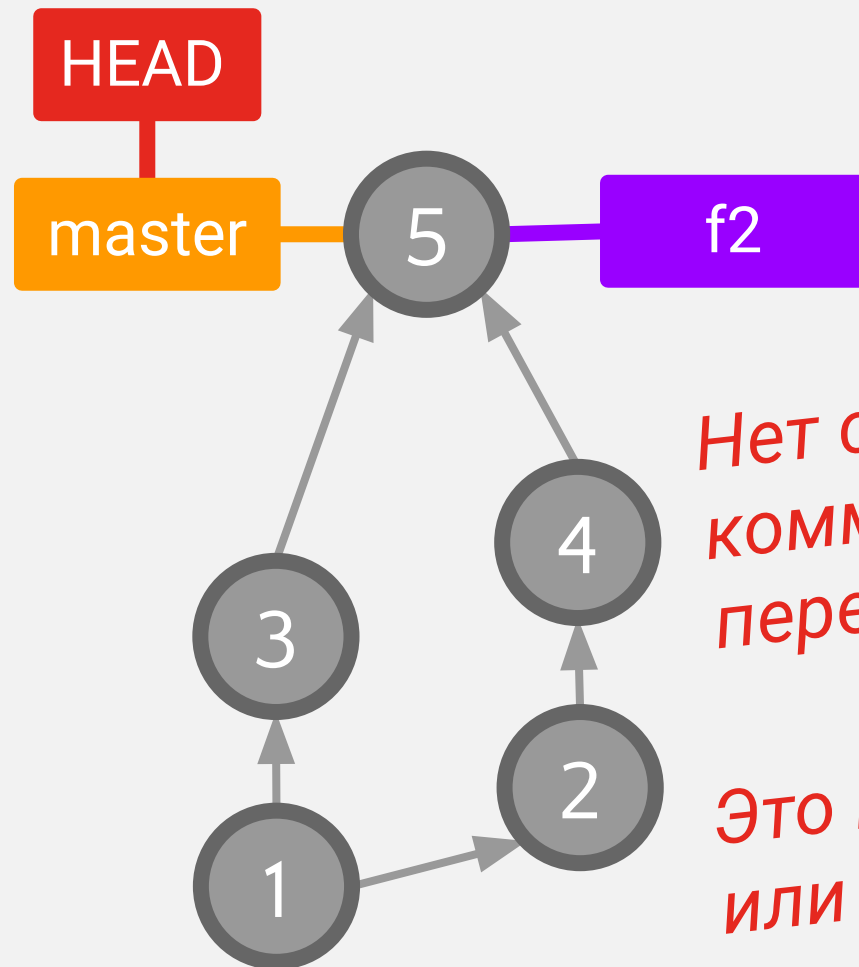
Слияние в Git Graph



А что если обратно влить?



Fast-Forward Merge

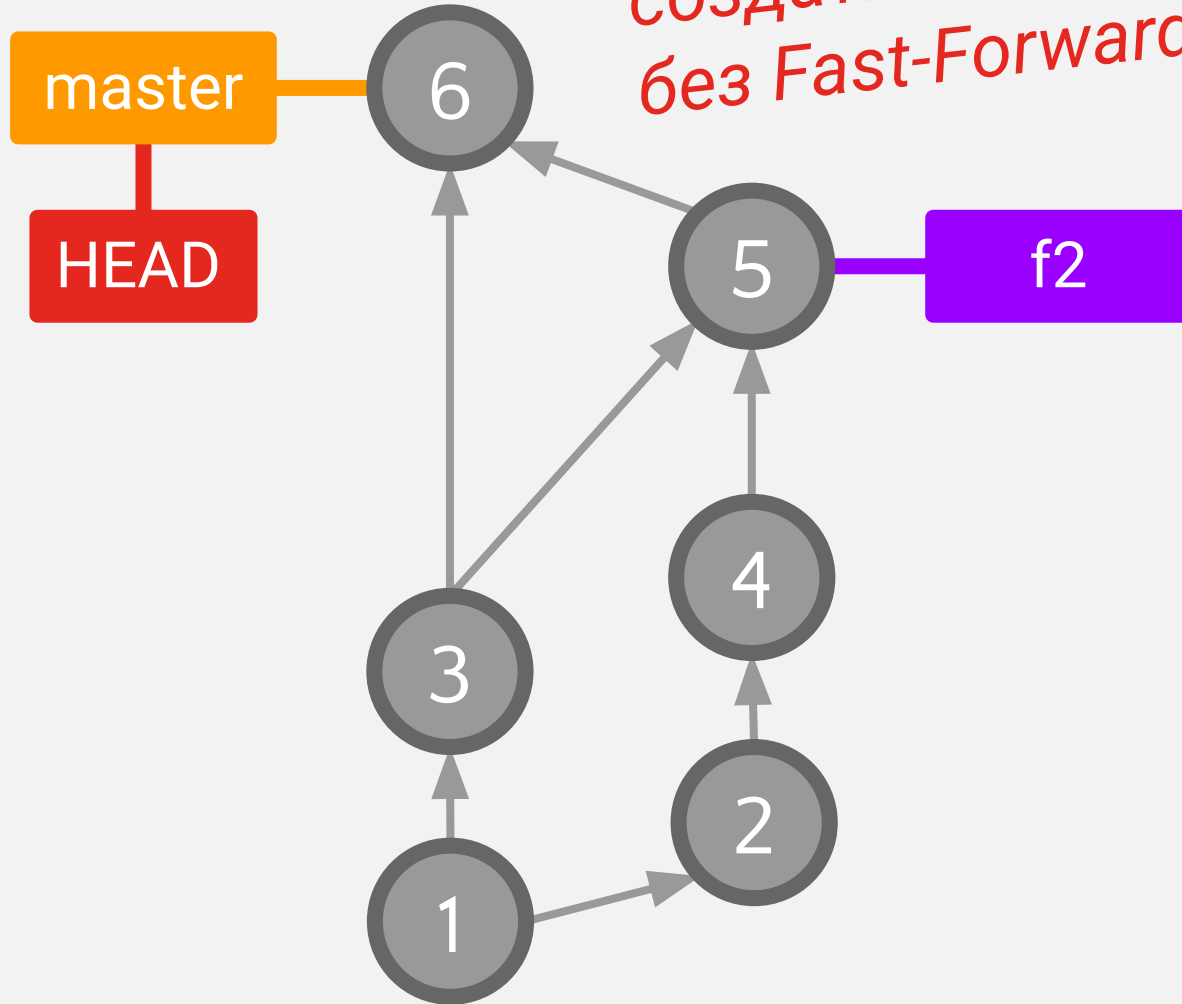


Нет смысла создавать
коммит – просто
передвигается ветка

Это Fast-Forward Merge
или «Перемотка»

merge --no-ff

Можно заставить Git
создать НОВЫЙ КОММИТ
без Fast-Forward



А если merge не удался?

Иногда вследствие неведомых действий состояние директории полно конфликтов, **merge** не завершен и его **хочется отменить**



А если merge не удался?

Иногда вследствие неведомых действий состояние директории полно конфликтов, **merge** не завершен и его **хочется отменить**



Переходим на исходный коммит с помощью `checkout` или `reset`, а затем повторяем `merge`

А если merge не удался?

Иногда вследствие неведомых действий состояние директории полно конфликтов, **merge** не завершен и его **хочется отменить**



Переходим на исходный коммит с помощью checkout или reset, а затем повторяем merge

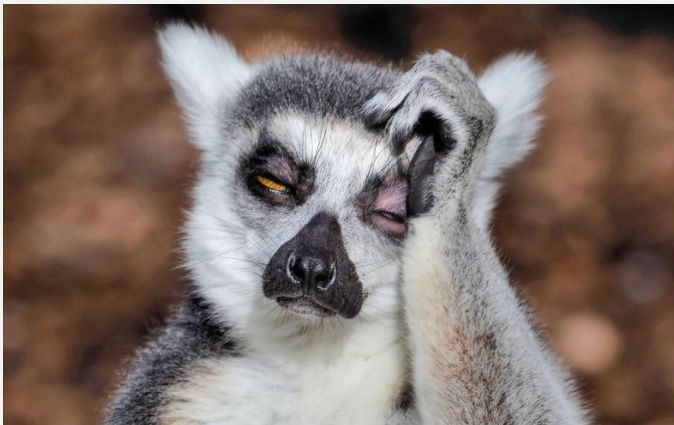


При этом затрутса все локальные изменения, поэтому **перед слиянием все изменения** рекомендуется **сохранять** (commit или ~~stash~~)

Более корректный способ

Иногда вследствие неведомых действий состояние директории полно конфликтов, **merge** не завершен и его **хочется отменить**

```
git merge --abort
```



*Предыдущий вариант
тоже ок. Разве нет?*

Лишние файлы после merge?

При создании конфликта Git генерирует .orig-файлы с версиями до слияния

Они исчезнут после успешного merge, если выставлено в настройках

```
[mergetool]  
  keepbackup = false
```



Задание 6. Merge Conflict

Задание 7. Hidden Conflict

Задание 8. Fast-Forward Merge

Structure

S1. Все локально

S2. Хранятся
состояния директории,
постепенная сборка коммита

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

Actions

A1. Трехсторонний merge
в три шага

Remote

H1. Гибкое конфигурирование и качественная документация

A2. rebase, cherry-pick и amend, чтобы пересоздать историю

Нельзя переписать историю — можно создать новую

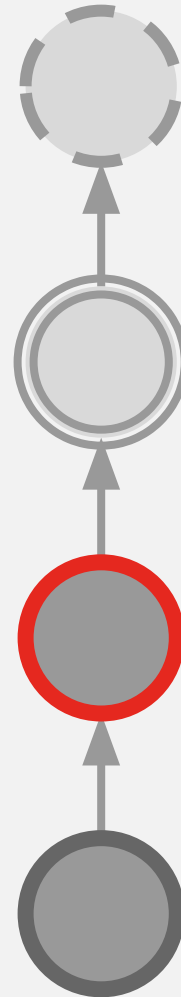
Даже если кажется,
что Git редактирует коммиты,
на самом деле он создает новые

Amend Commit

Working directory

Commit index

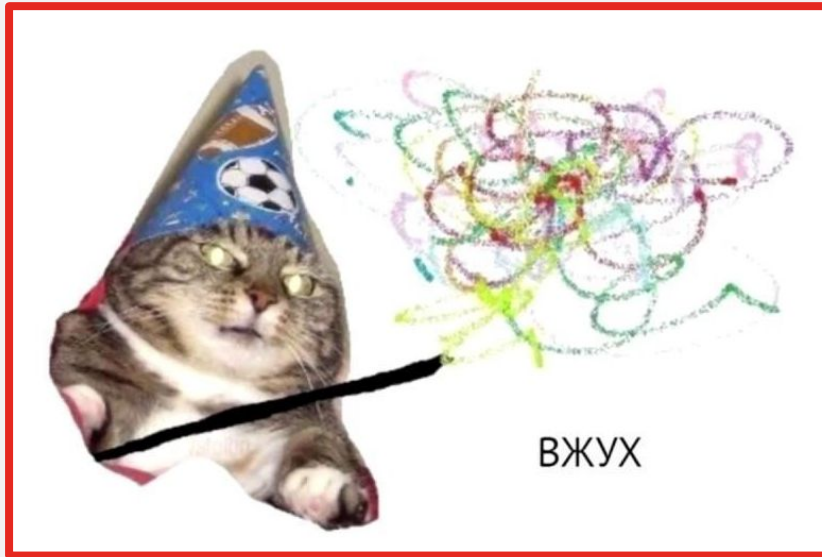
Последний коммит



*Хочется дополнить
последний коммит*

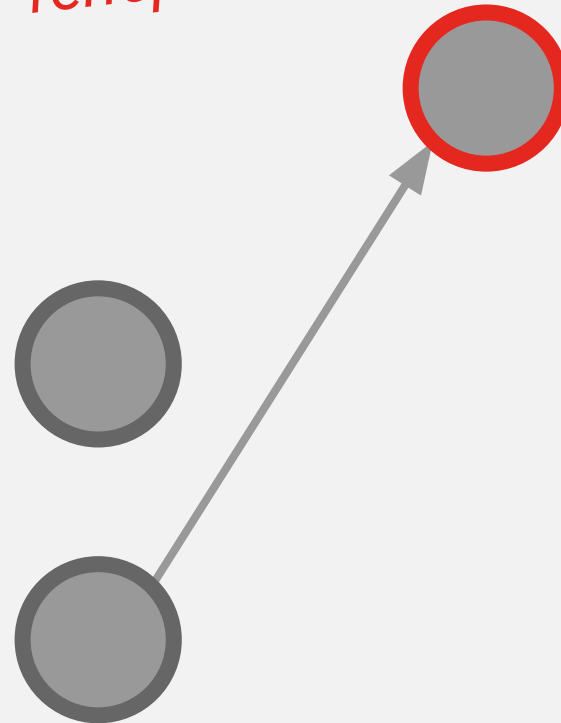
Amend – англ. Вносить правки

Amend Commit



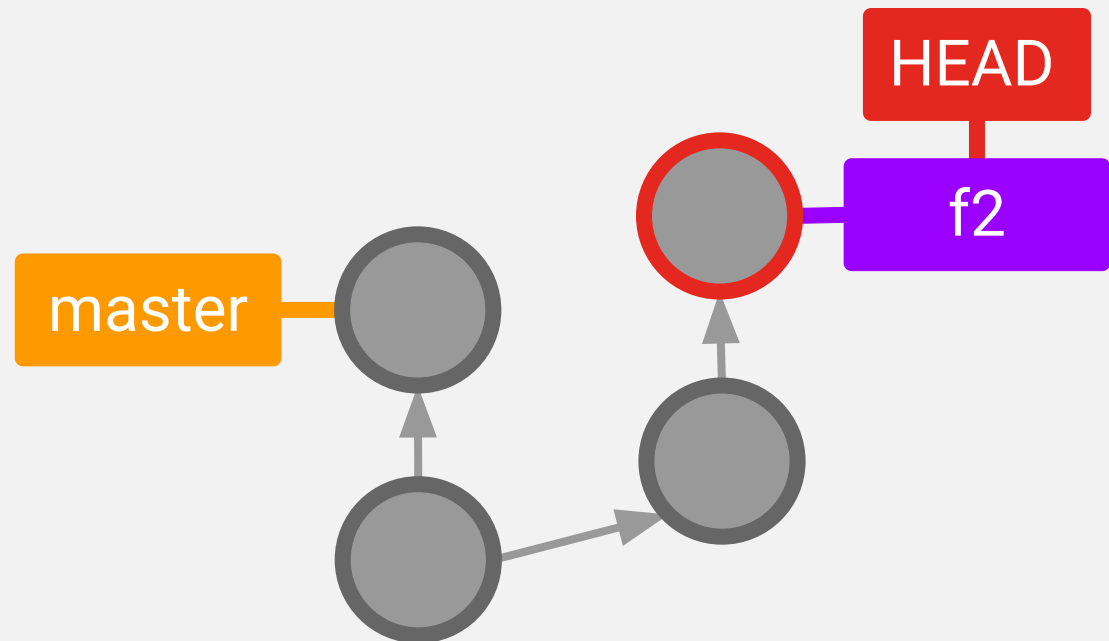
Старый коммит остался,
но никому не нужен

Изменения из Commit index
и старого коммита
теперь в новом коммите



Rebase

Слияния порождает лишние коммиты,
а история становится нелинейной

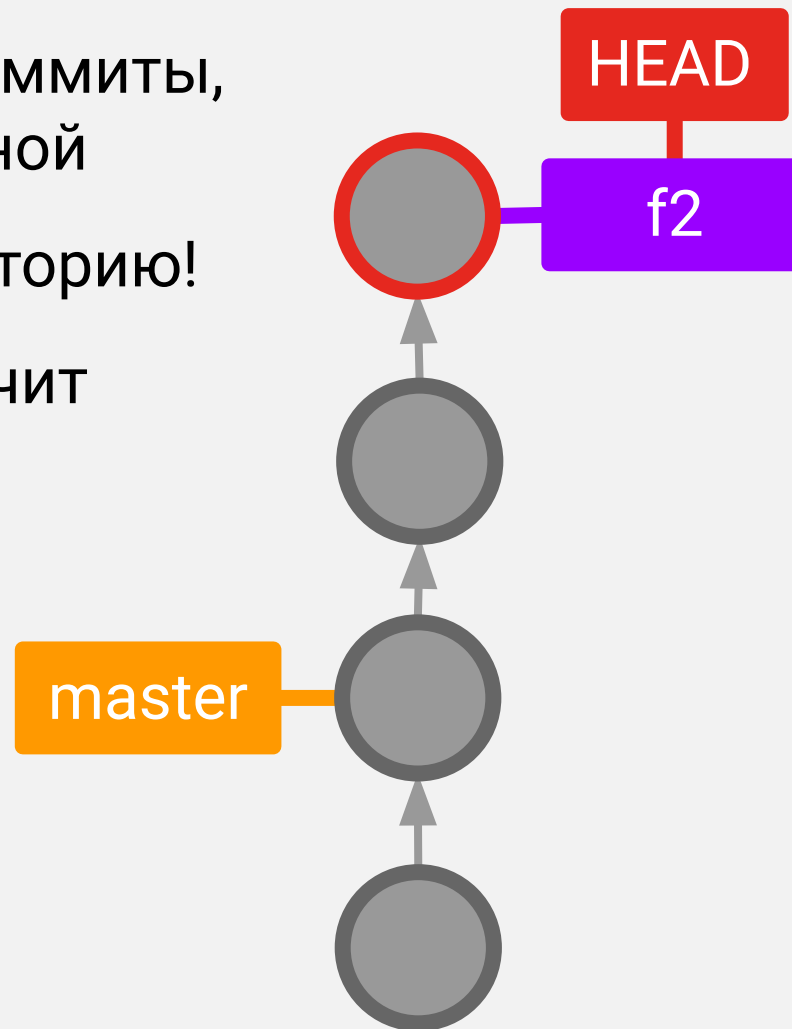


Rebase

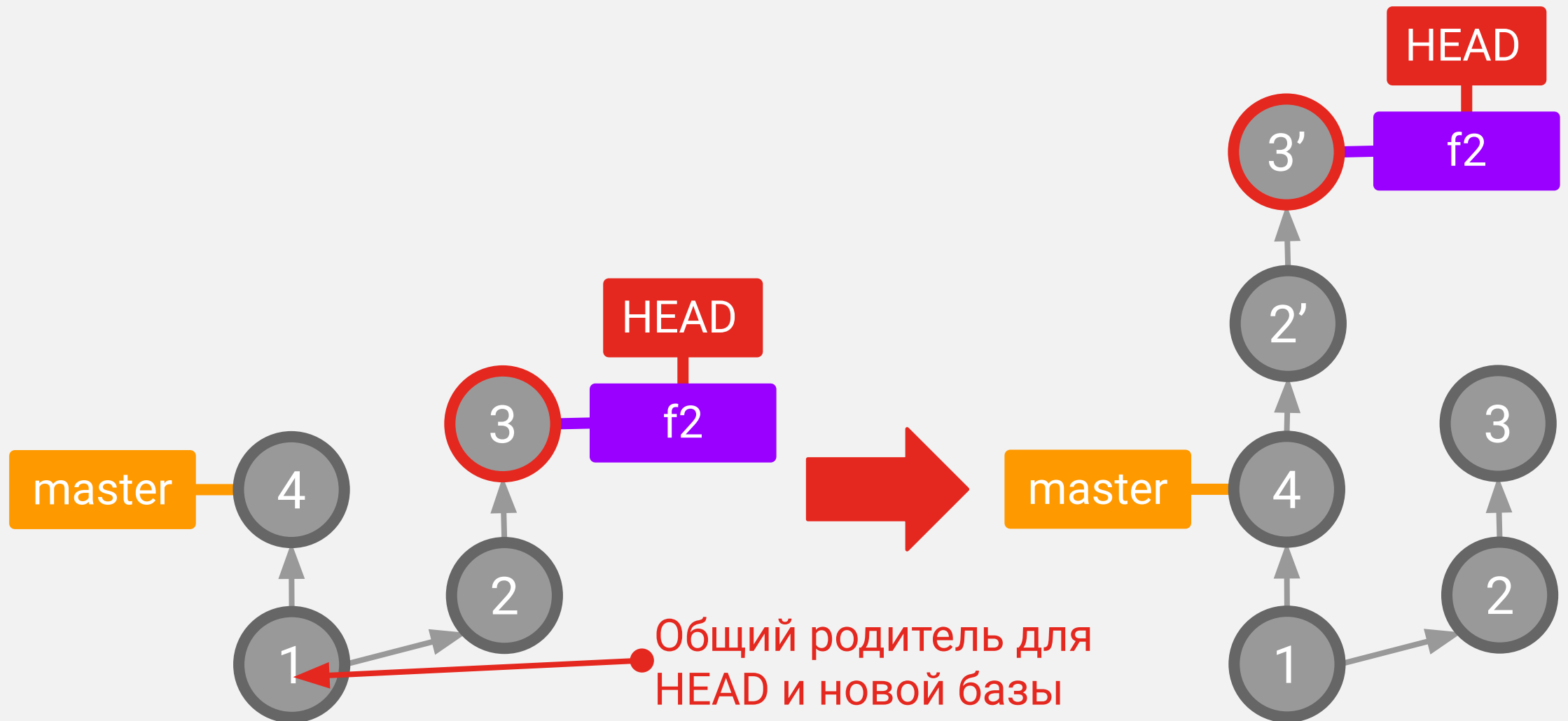
Слияния порождает лишние коммиты,
а история становится нелинейной

Хочется получать линейную историю!

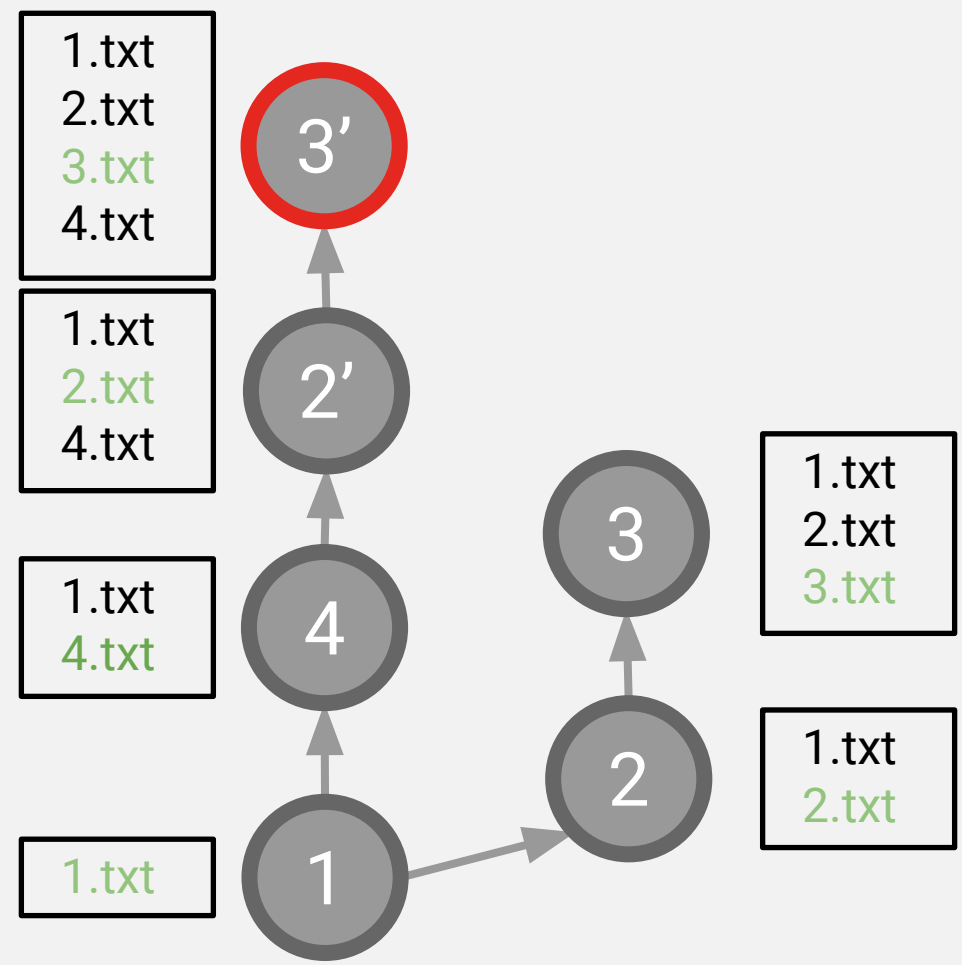
Менять коммиты нельзя – значит
надо создавать копии



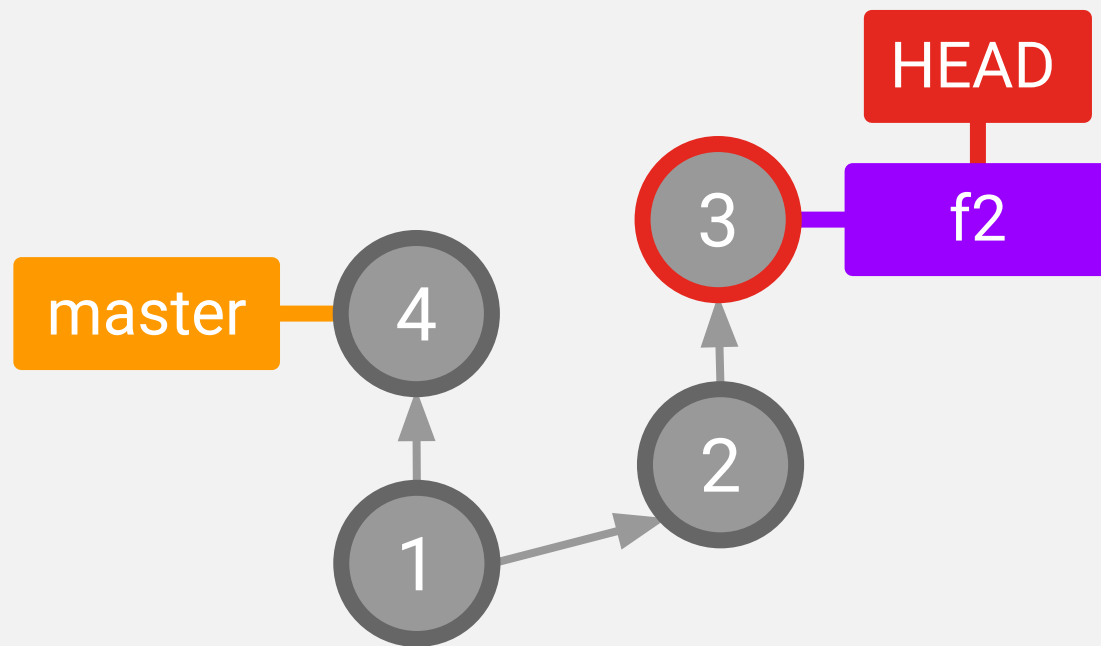
Rebase создает новые коммиты



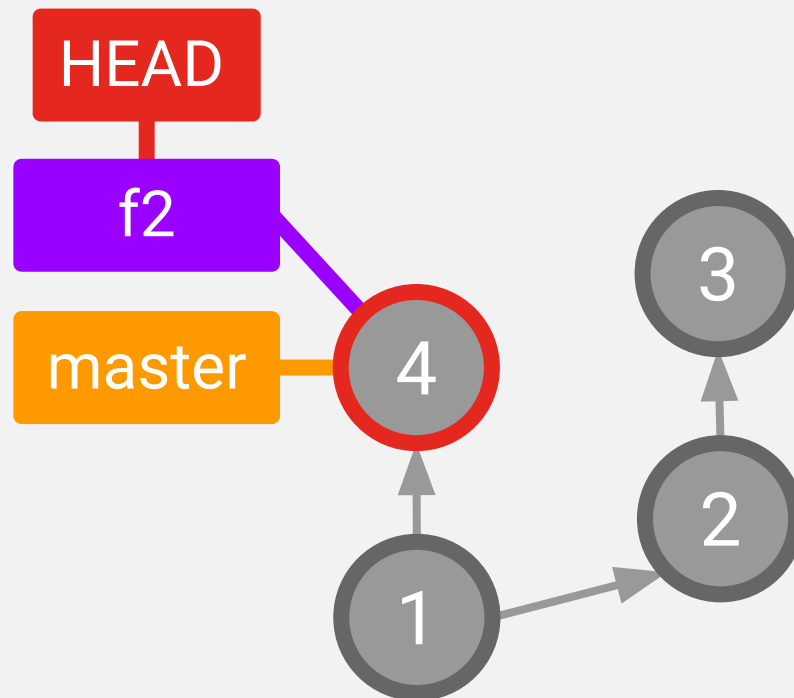
Те же изменения на новой «базе»



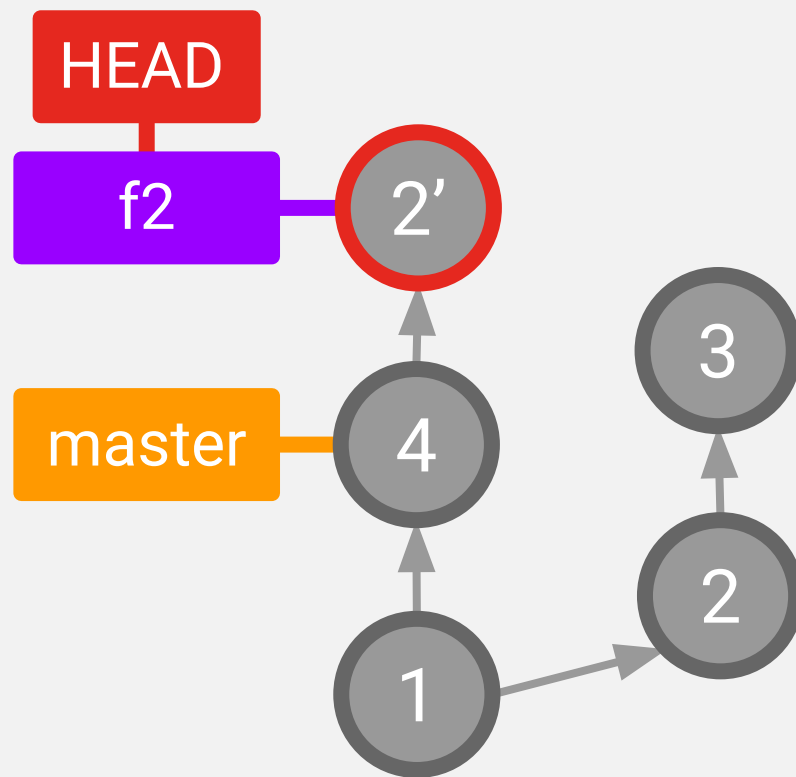
Rebase по шагам



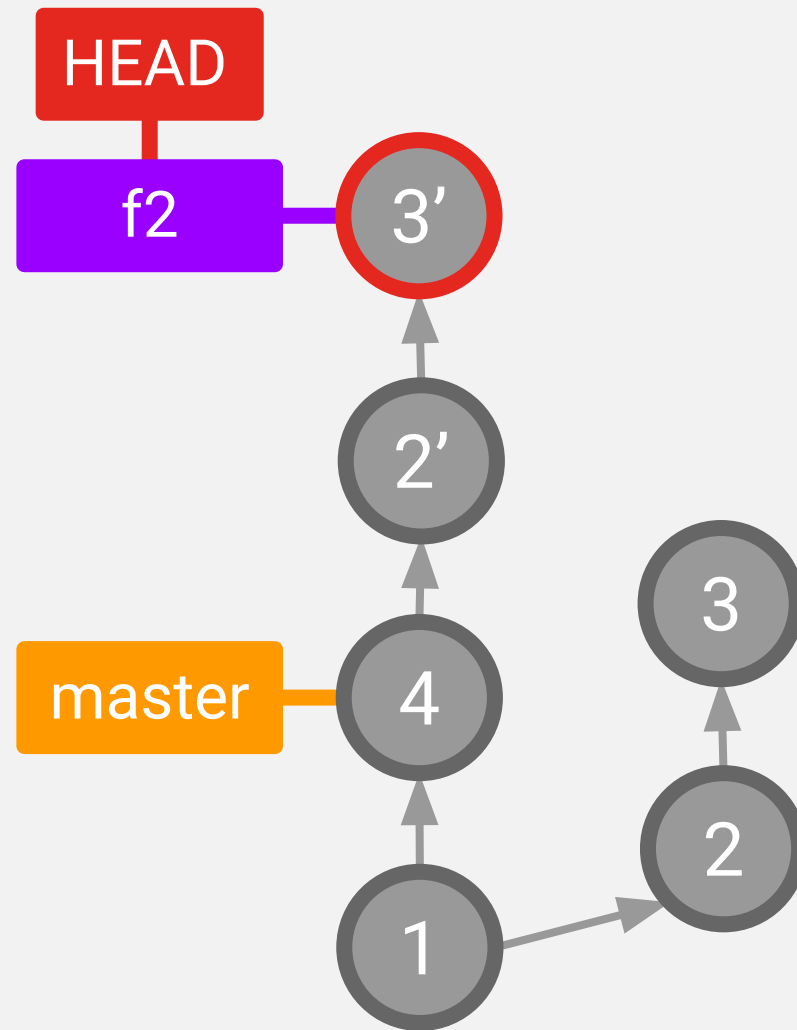
Rebase по шагам



Rebase по шагам



Rebase по шагам

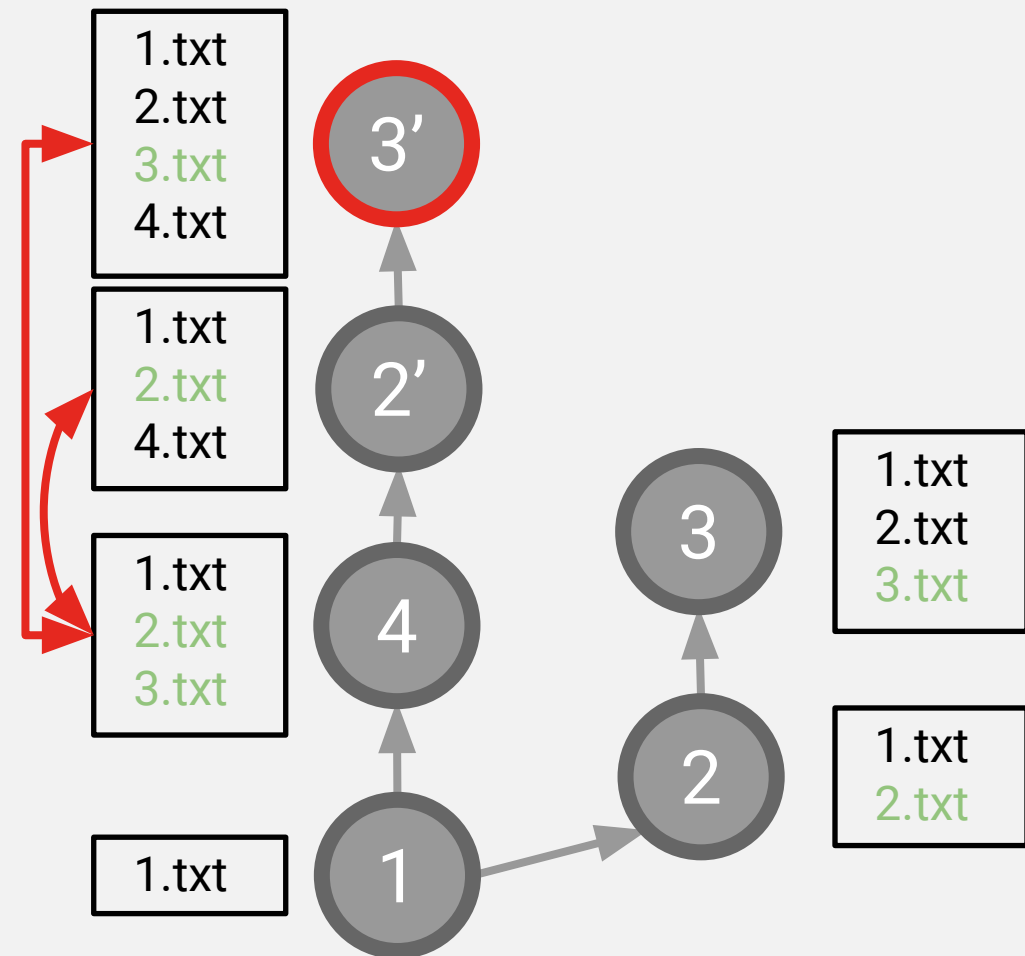


Конфликты при rebase

Изменения в коммите 4, новой базе, могут конфликтовать с изменениями в 2 и в 3

Следовательно, перенос **каждого коммита** может породить конфликт

Конфликты разрешаются аналогично merge



В конфликтах
HEAD указывает на **current**
Другая ветка — это **incoming**

После начала rebase **HEAD** перемещается
на коммит построения **новой истории**,
а в конце оказывается на вершине построенной ветки

После решения конфликтов — продолжение

git rebase --continue

The screenshot shows the 'Rebase' dialog box in a Git GUI. The window title is 'Rebase'. On the left, a commit graph illustrates the rebase process: a local branch 'other' (commits f, e) is being rebased onto a remote branch 'current' (commits d, c, b, a). The 'current' branch is marked as 'REMOTE'. A legend indicates that squares represent the current branch and green circles represent new commits. A table titled 'Commits to re-apply:' shows the following data:

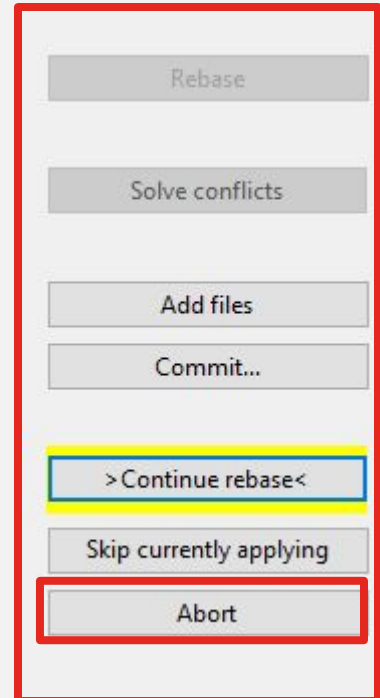
Name	Subject	Author	Date	Status
0001	3	digi	Sun, 18 Oct 2020 23:06:03	Applying...

On the right side of the dialog, there are several buttons: 'Rebase', 'Solve conflicts', 'Add files', 'Commit...', '>Continue rebase<', 'Skip currently applying', and 'Abort'. The '>Continue rebase<' button is highlighted with a red and yellow border.

А если rebase идет как-то не так?

Иногда приходит понимание, что не надо было ввязываться в этот rebase с кучей конфликтов при переносе каждого коммита и его **ХОЧЕТСЯ ОТМЕНИТЬ**

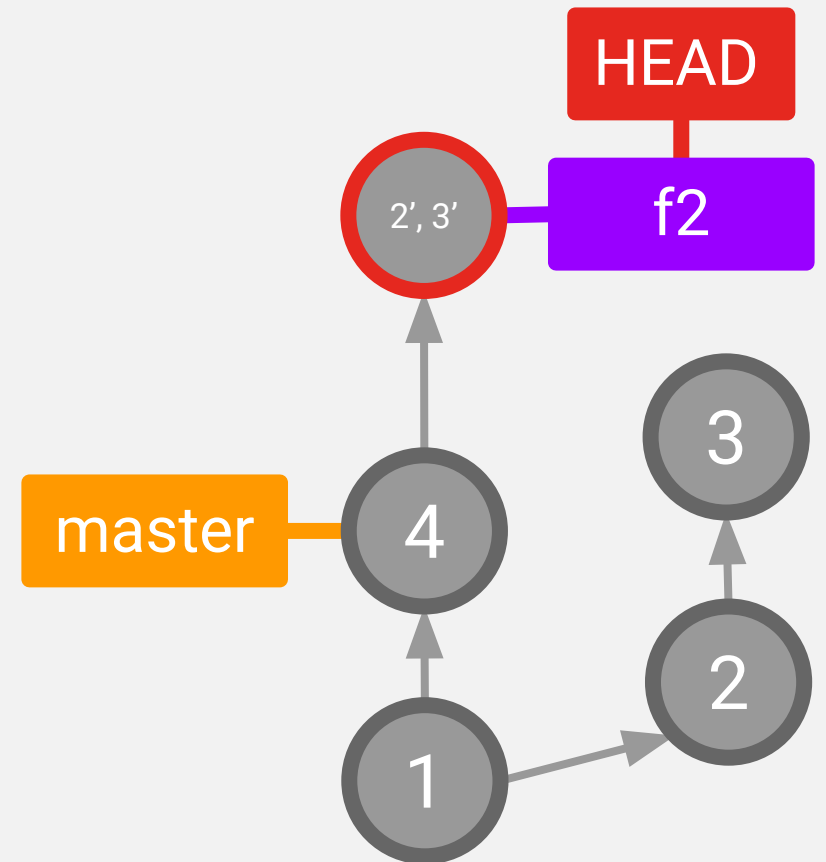
```
git rebase --abort
```



Интерактивный rebase

Позволяет указать, как
применять каждый коммит

Например, можно делать
squash всех переносимых
КОММИТОВ В ОДИН



Сценарий интерактивного rebase

```
git-rebase-todo - Блокнот
Файл Правка Формат Вид Справка
pick 7b722d0 Sheet markup
pick 5a9e1f5 Add fetch.md
pick f2335b7 Add help.md

# Rebase 5c65d9c..f2335b7 onto 5c65d9c (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the comm
# e, edit <commit> = use commit, but stop for amendi
# s, squash <commit> = use commit, but meld into pre
# f, fixup <commit> = like "squash", but discard thi
```

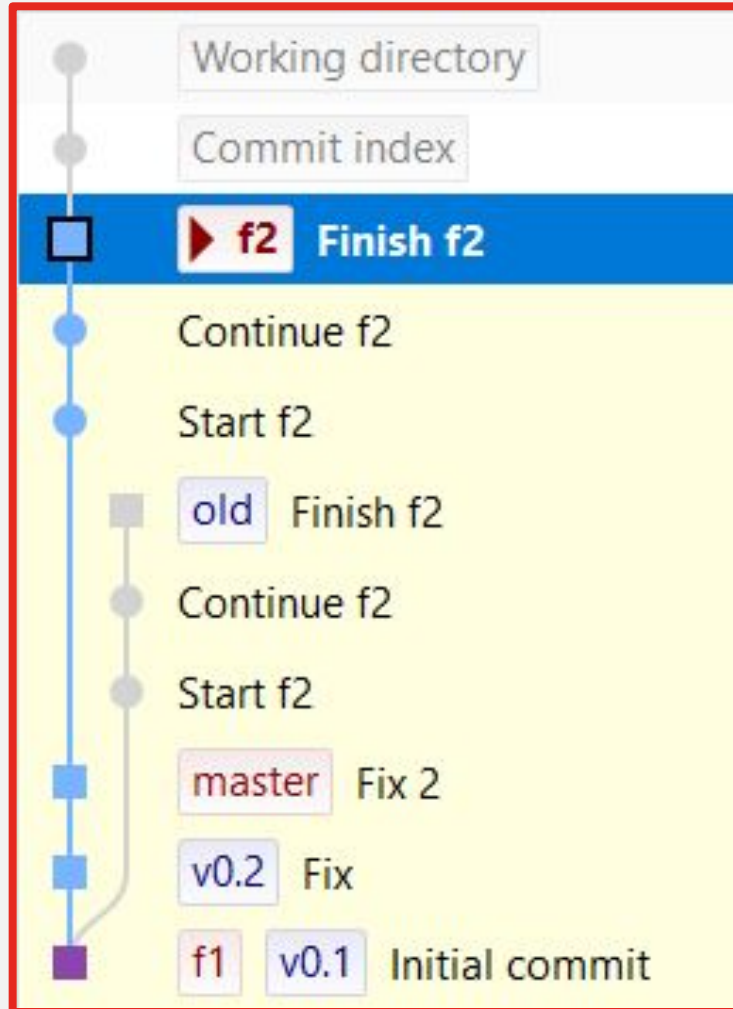
Обычный вариант

```
git-rebase-todo - Блокнот
Файл Правка Формат Вид Справка
reword 7b722d0 Sheet markup
fixup 5a9e1f5 Add fetch.md
fixup f2335b7 Add help.md

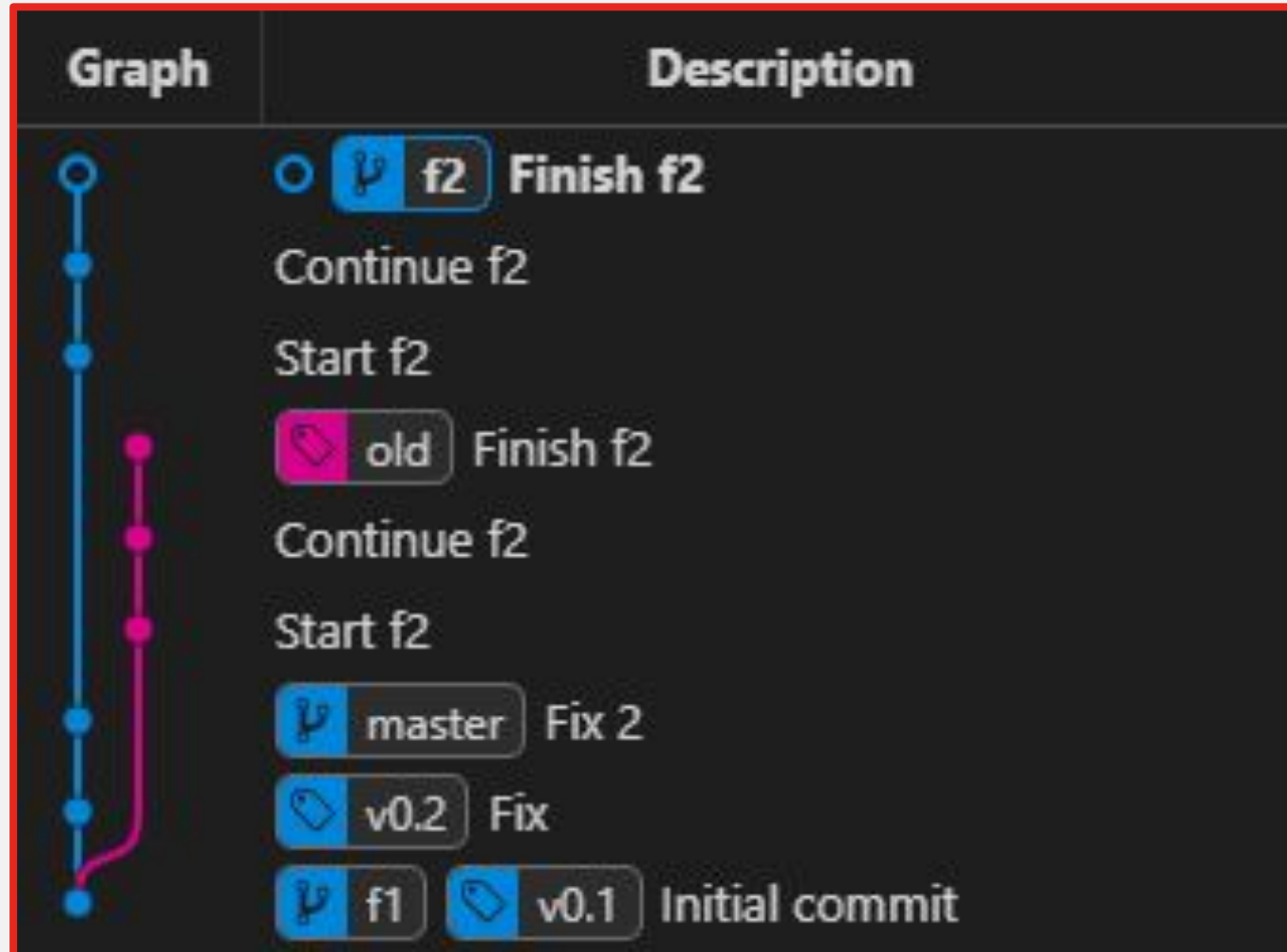
# Rebase 5c65d9c..f2335b7 onto 5c65d9c (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the comm
# e, edit <commit> = use commit, but stop for amendi
# s, squash <commit> = use commit, but meld into pre
# f, fixup <commit> = like "squash", but discard thi
```

С объединением коммитов

Rebase в Git Extensions



Rebase в Git Graph

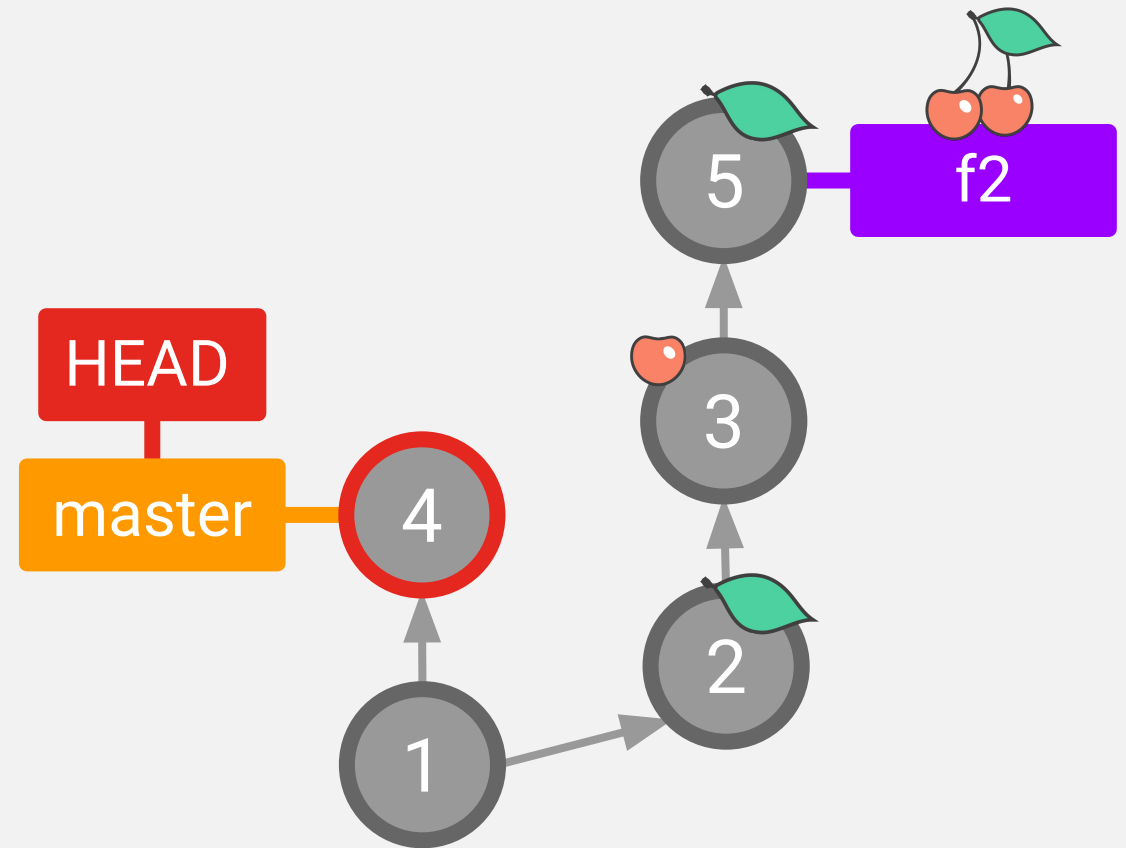


Cherry Pick

Позволяет создать копию изменений произвольного коммита в новом месте



cherry picking — идиома, означающая взять лучшее и оставить все остальное

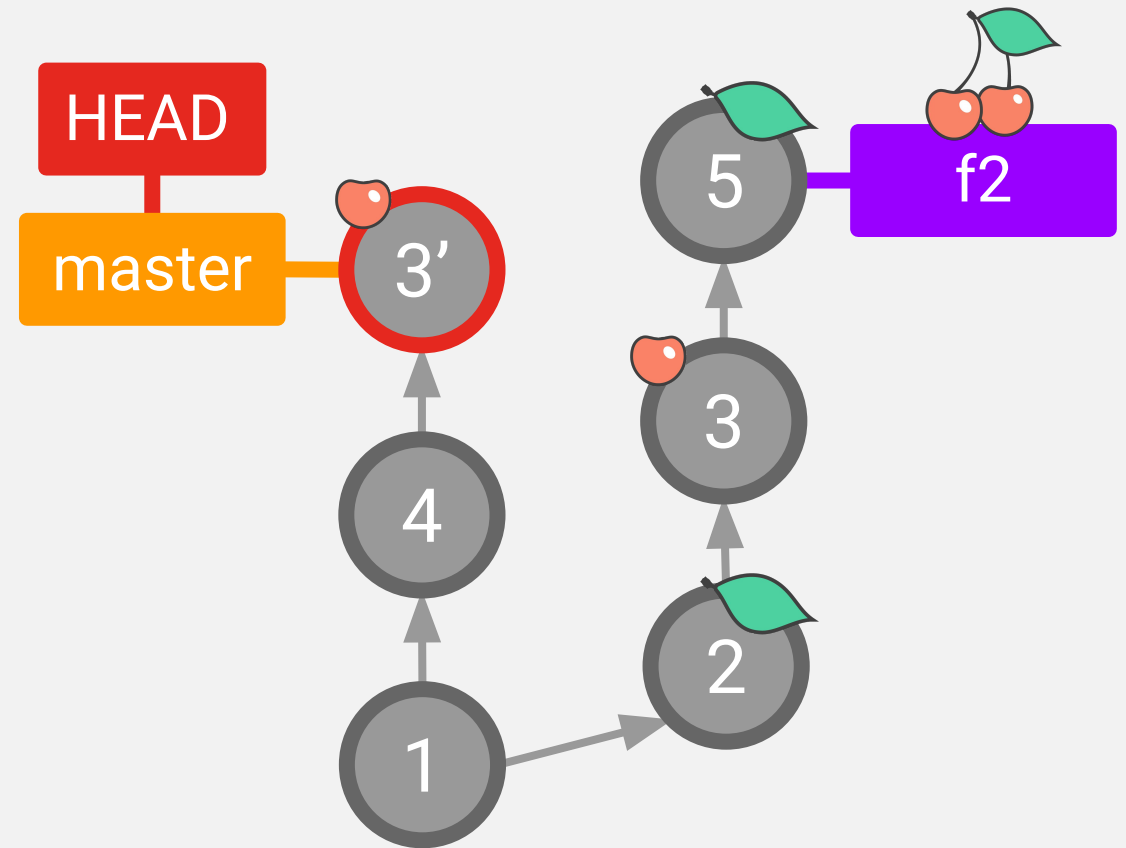


Cherry Pick

Позволяет создать копию изменений произвольного коммита в новом месте



cherry picking — идиома, означающая взять лучшее и оставить все остальное



Задание 9. Hot Rebase
Задание 10. Reflog

Structure

Actions

Remote

S1. Все локально

A1. Трехсторонний merge
в три шага

S2. Хранятся
состояния директории,
постепенная сборка коммита

A2. rebase, cherry-pick и amend,
чтобы пересоздать историю

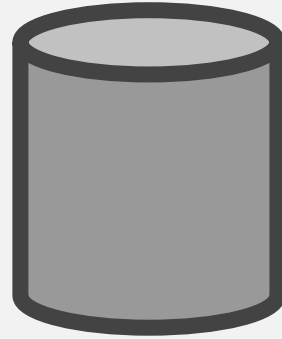
S3. Манипуляции
через ссылки,
нет ссылки — в мусор

H1. Гибкое конфигурирование и качественная документация

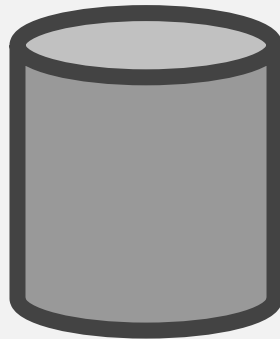
R1. Доступен fetch коммитов
любого репозитория в любой момент

Клонирование

`https://repos/repo`



`git clone https://repos/repo`



origin

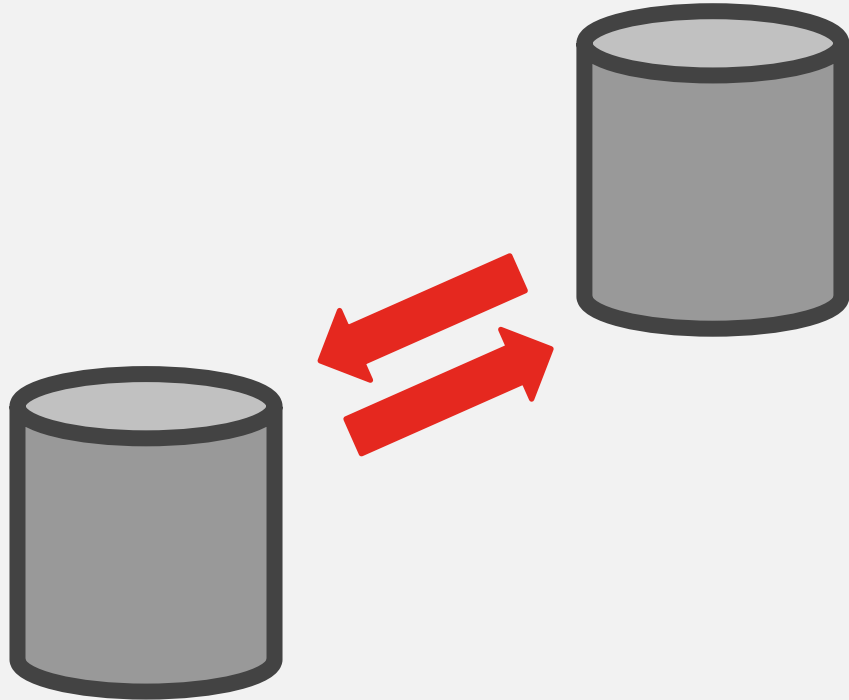
`https://repos/repo`



`origin = https://repos/repo`

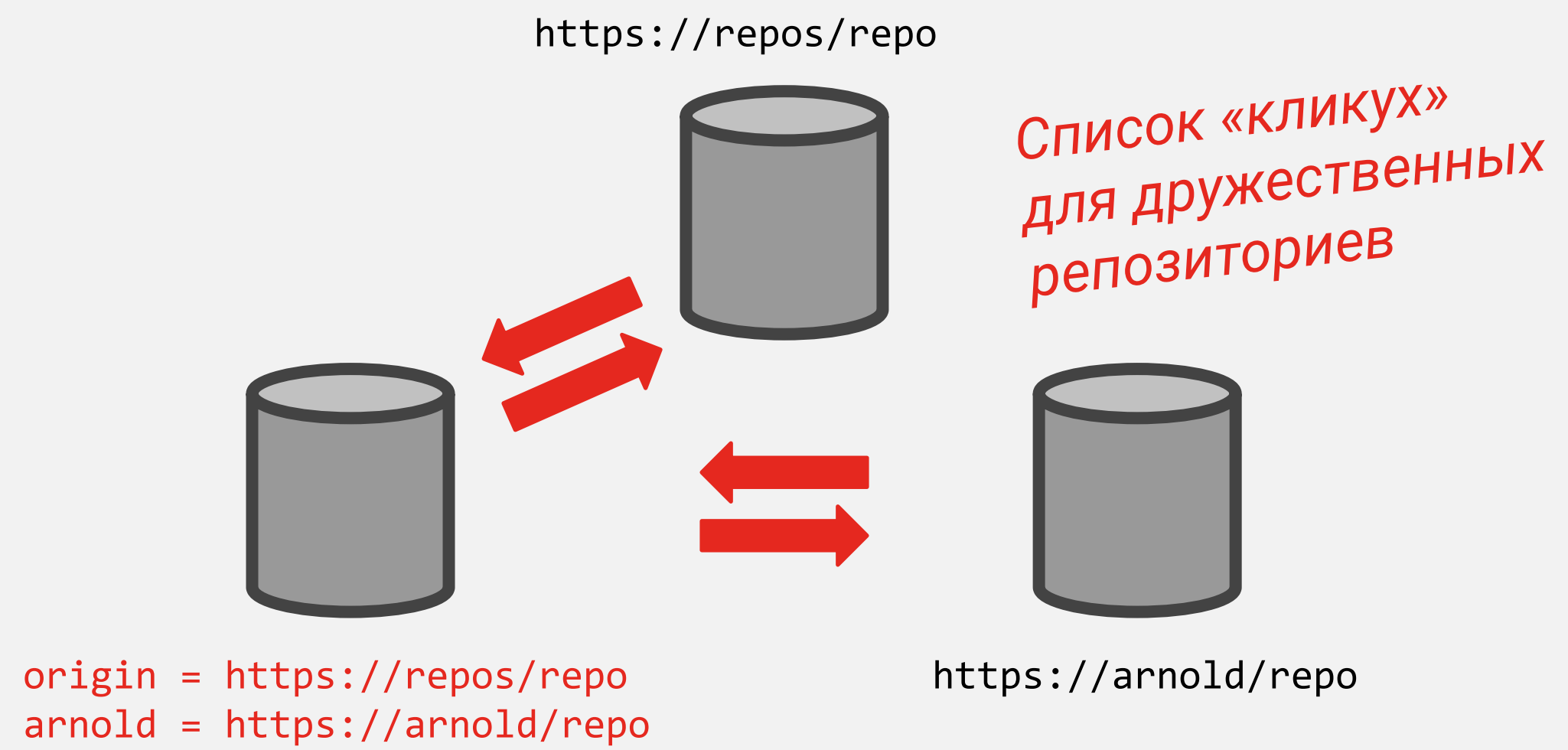
remote

`https://repos/repo`



`origin = https://repos/repo`

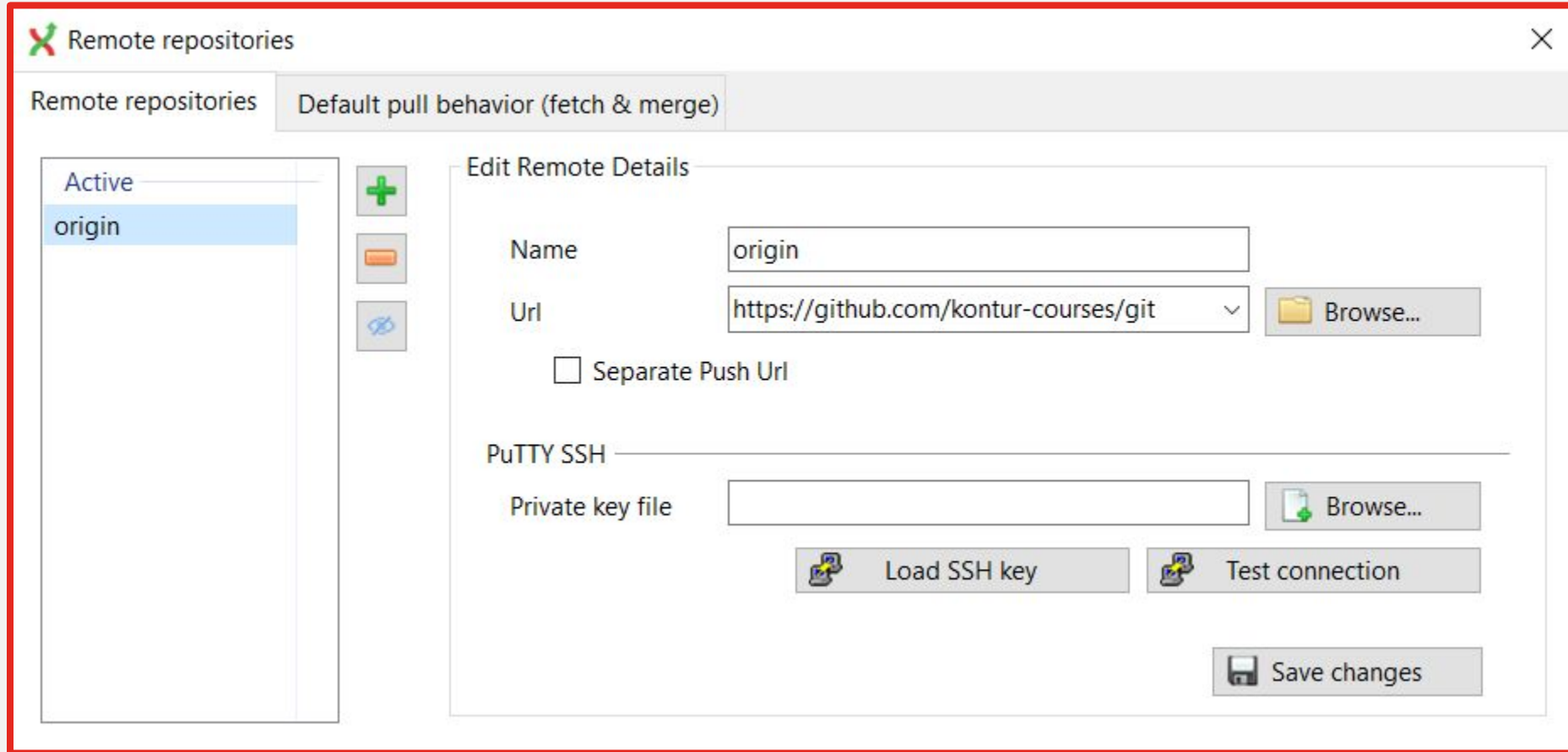
Таблица remote



Remote в Git Extensions



Remote в Git Extensions



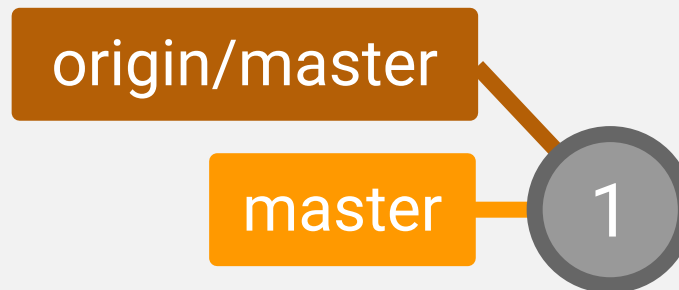
Remote в консоли

```
$ git remote -v  
origin https://github.com/kontur-courses/git (fetch)  
origin https://github.com/kontur-courses/git (push)
```

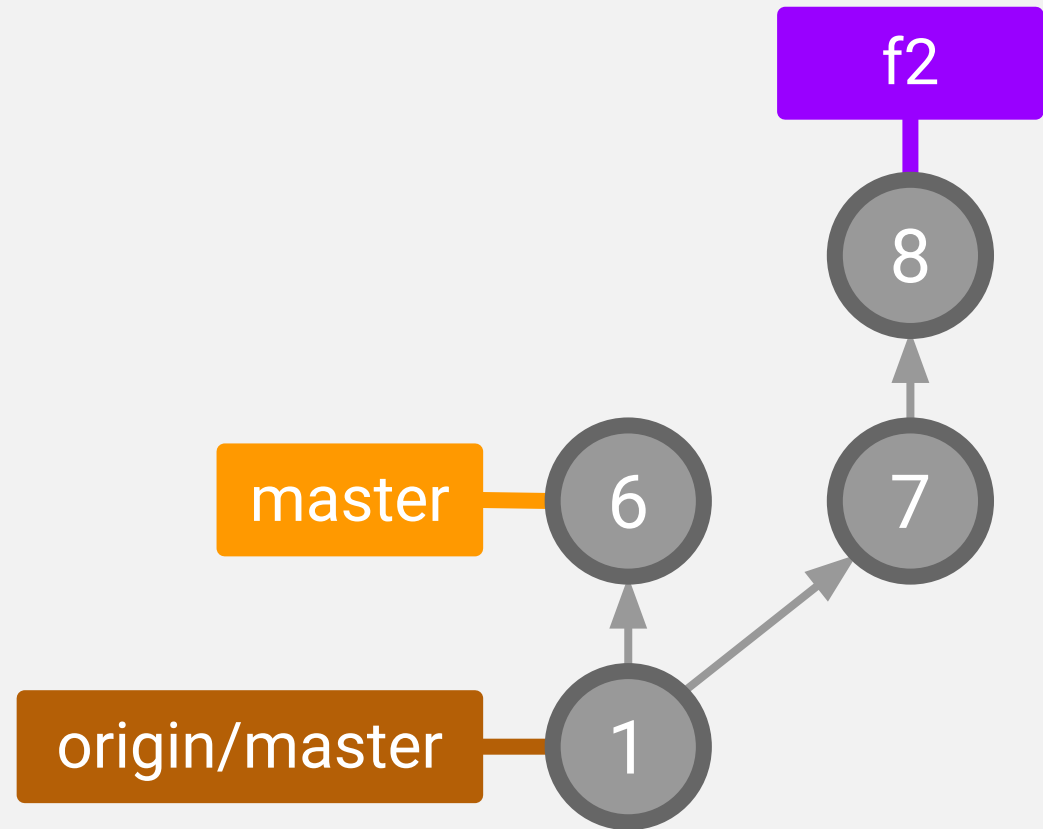
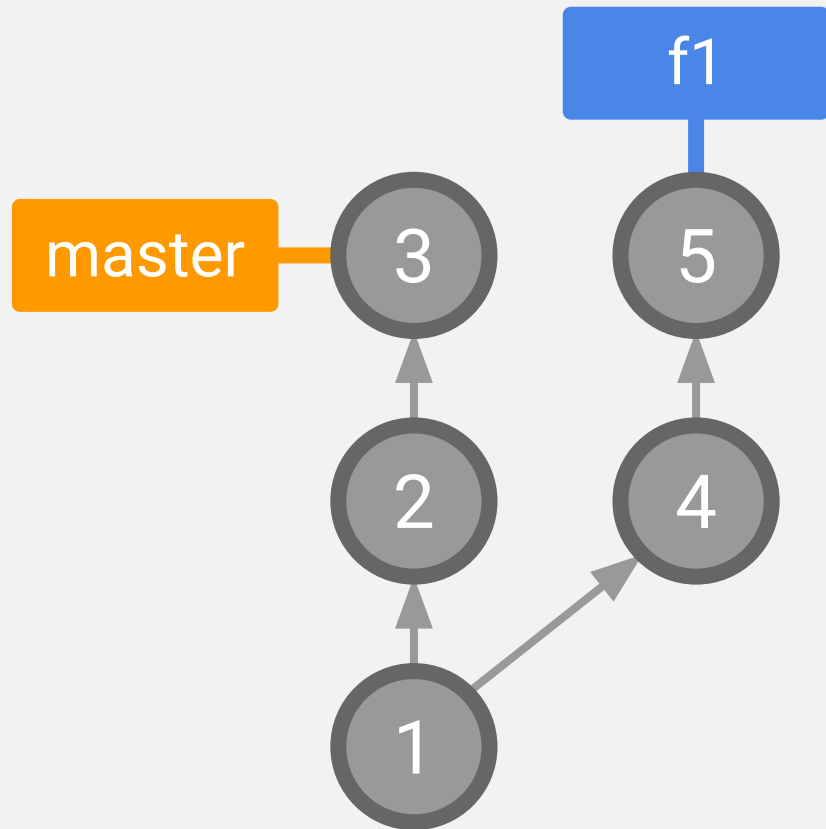
Обмен изменениями



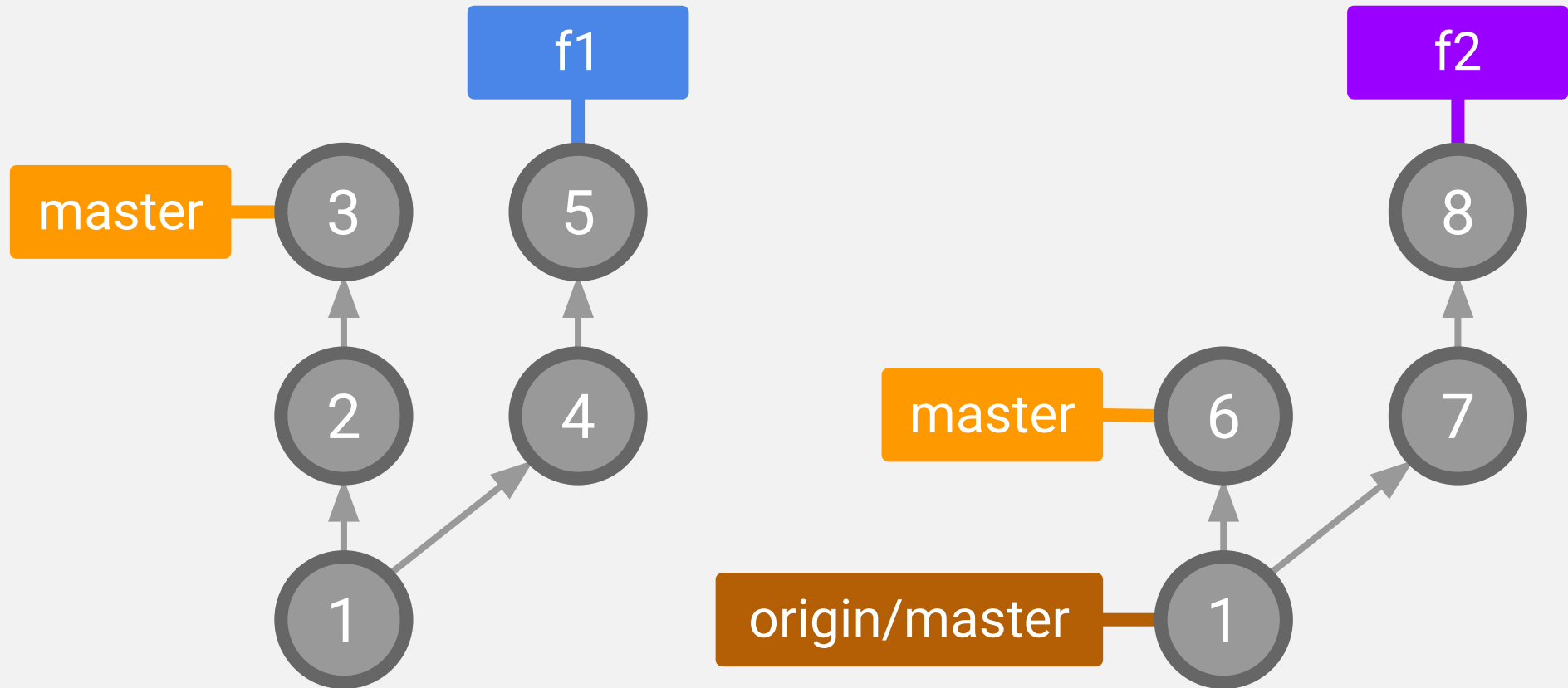
Появился клон



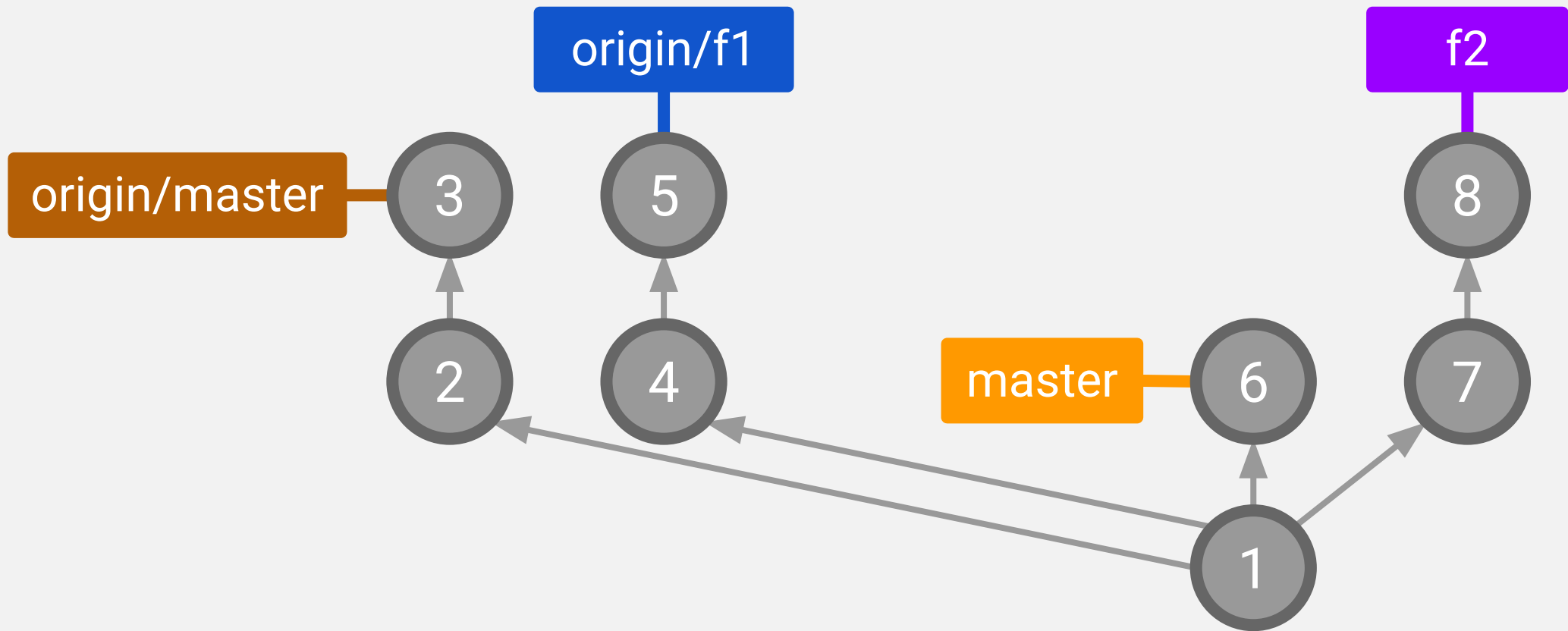
Прошло время



А что если в правый добавить из левого?



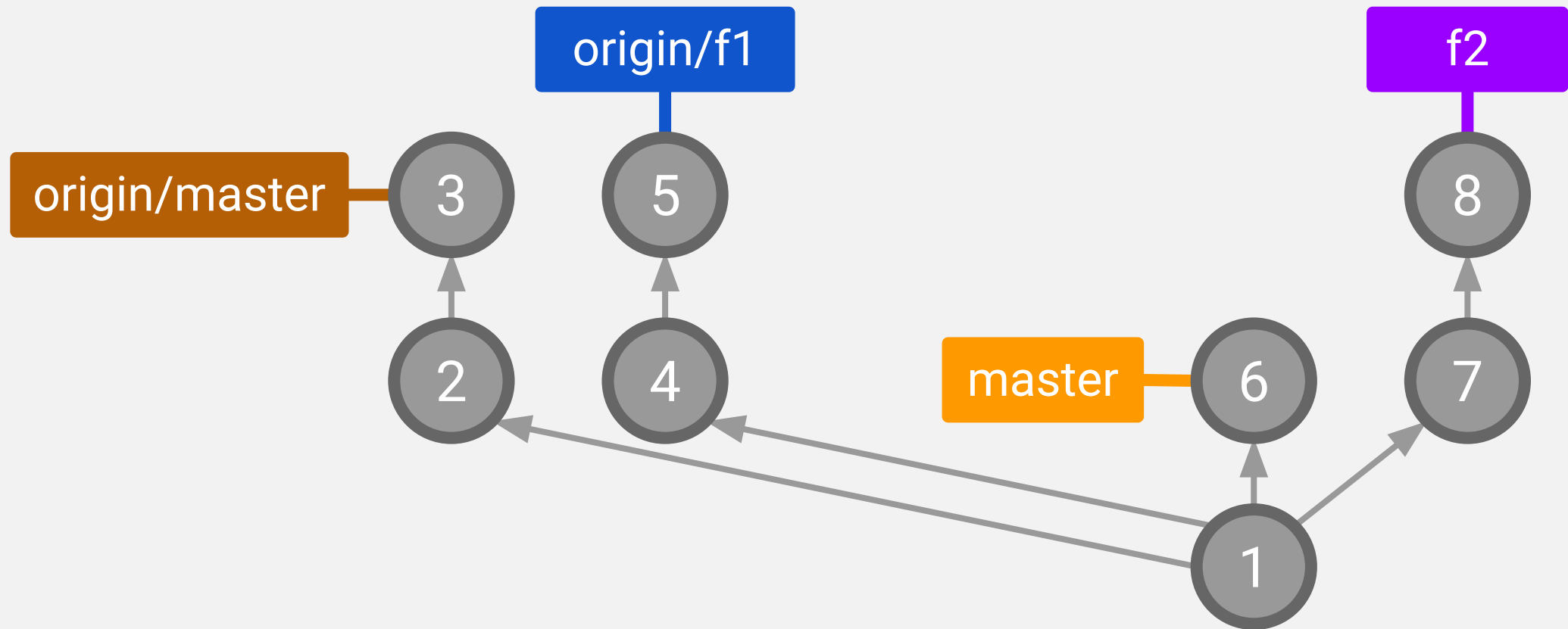
А что если в правый добавить из левого?



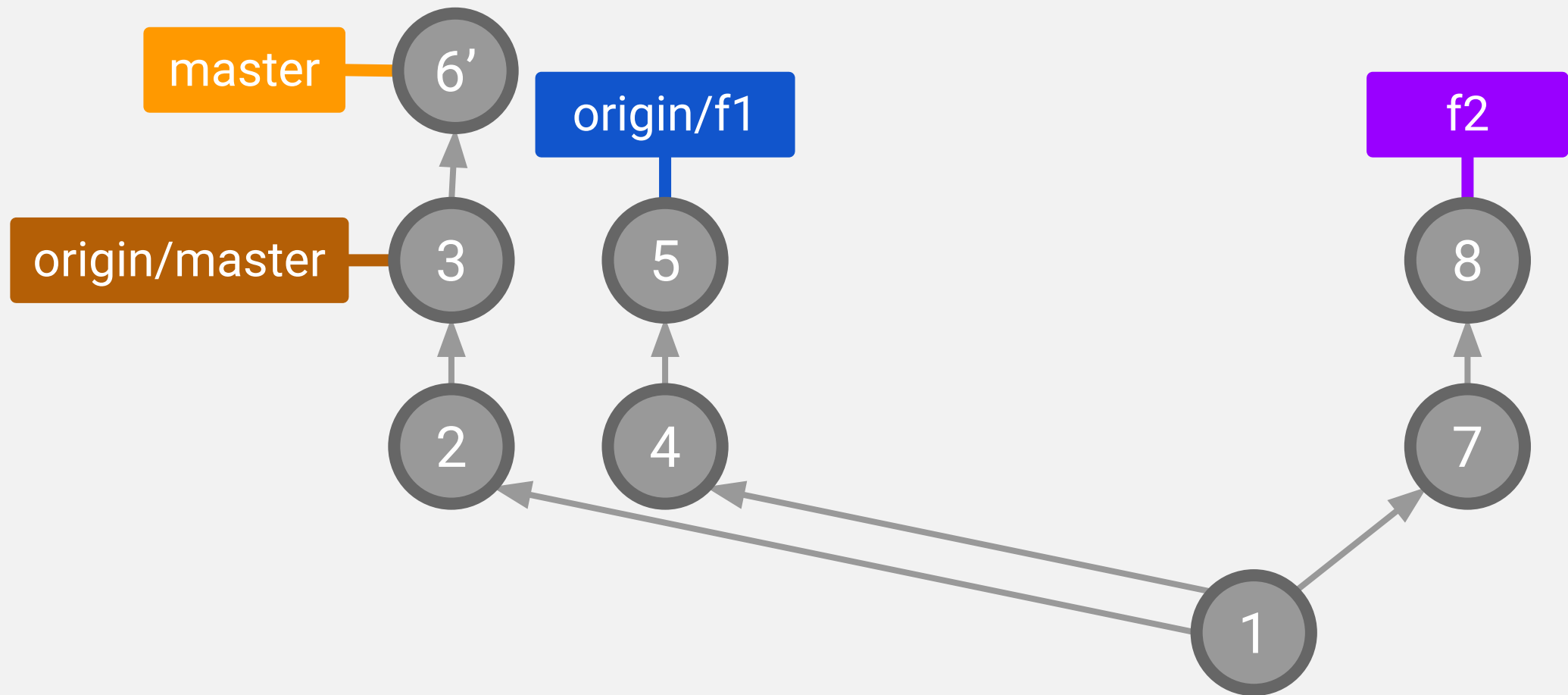
Fetch

- **fetch** – операция получения изменений из другого репозитория
- Это **безопасная операция** за счет неизменности истории
- В коммитах может быть «каша», но это уже отдельный вопрос
- Fetch ничего не меняет, просто **позволяет взглянуть шире**: на изменения из других репозиториях, а не только на локальные
- Можно сделать fetch из любого репозитория, **даже если нет общих коммитов**

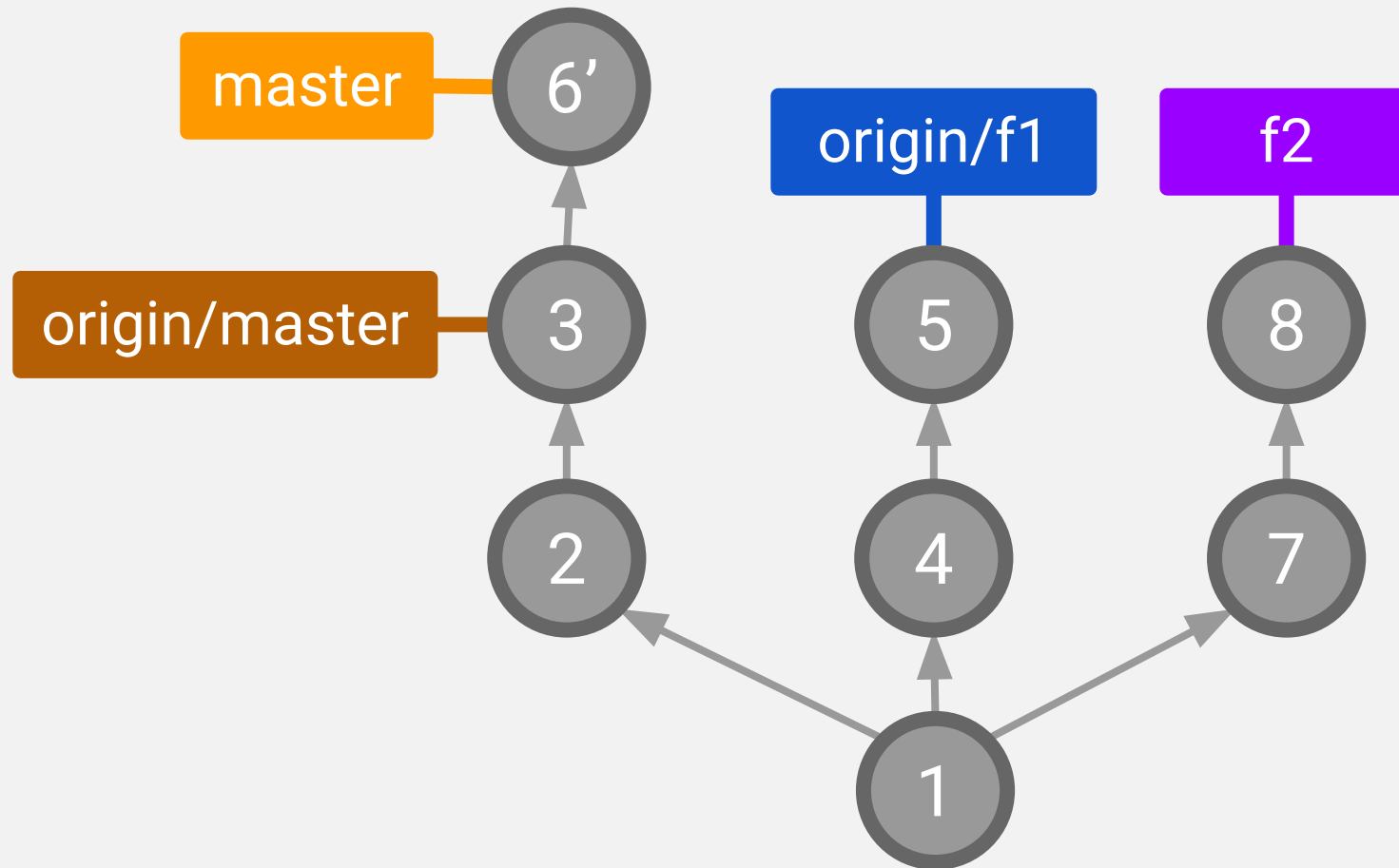
Все же, что здесь делать?



Rebase локальной ветки



Теперь все хорошо



Задание 11. Fetch From Remote

Задание 12. Interactive Rebase

Задание 13. Cherry Pick

Structure

Actions

Remote

S1. Все локально

A1. Трехсторонний merge
в три шага

R1. Доступен fetch коммитов
любого репозитория
в любой момент

S2. Хранятся
состояния директории,
постепенная сборка коммита

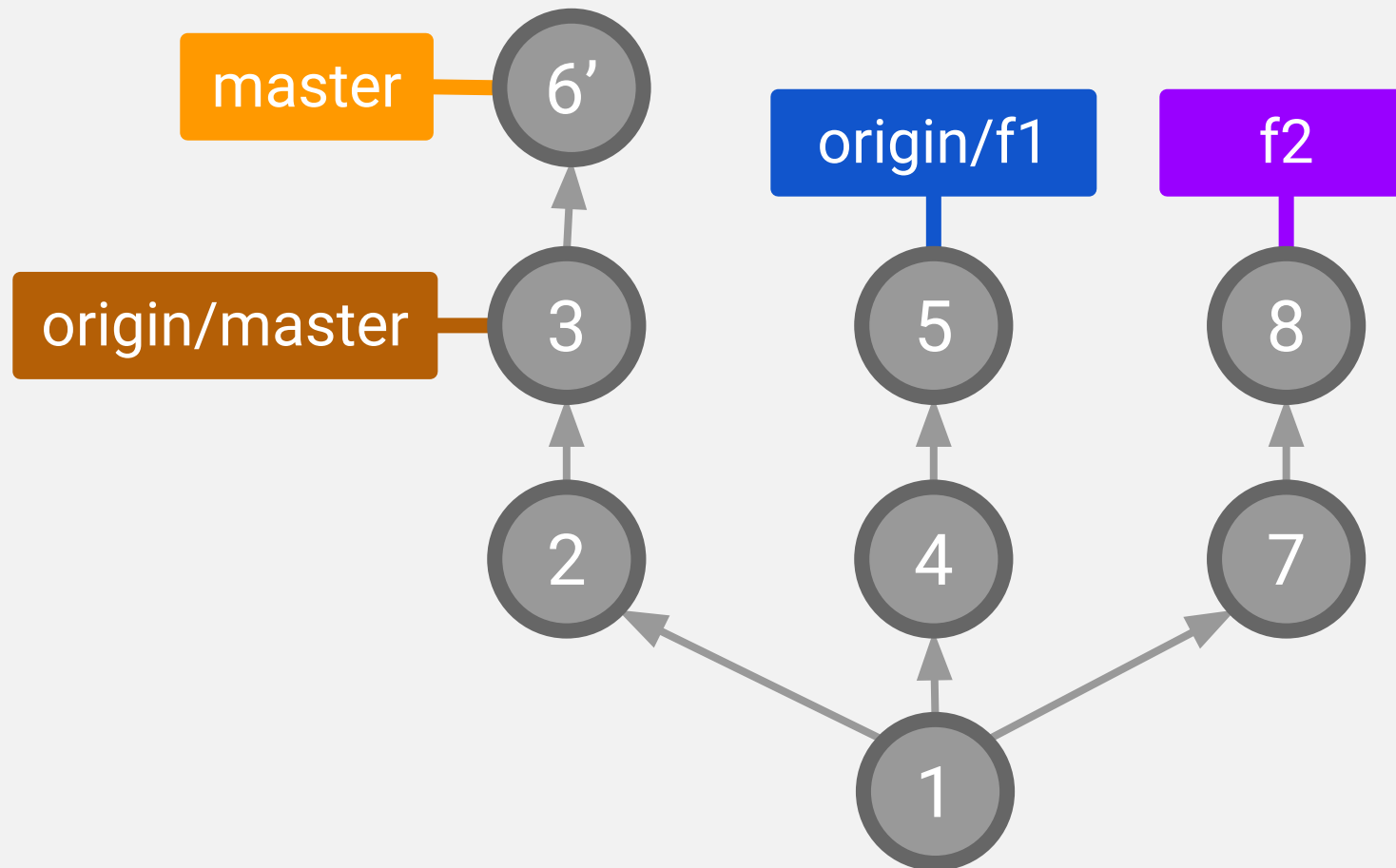
A2. rebase, cherry-pick и amend,
чтобы пересоздать историю

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

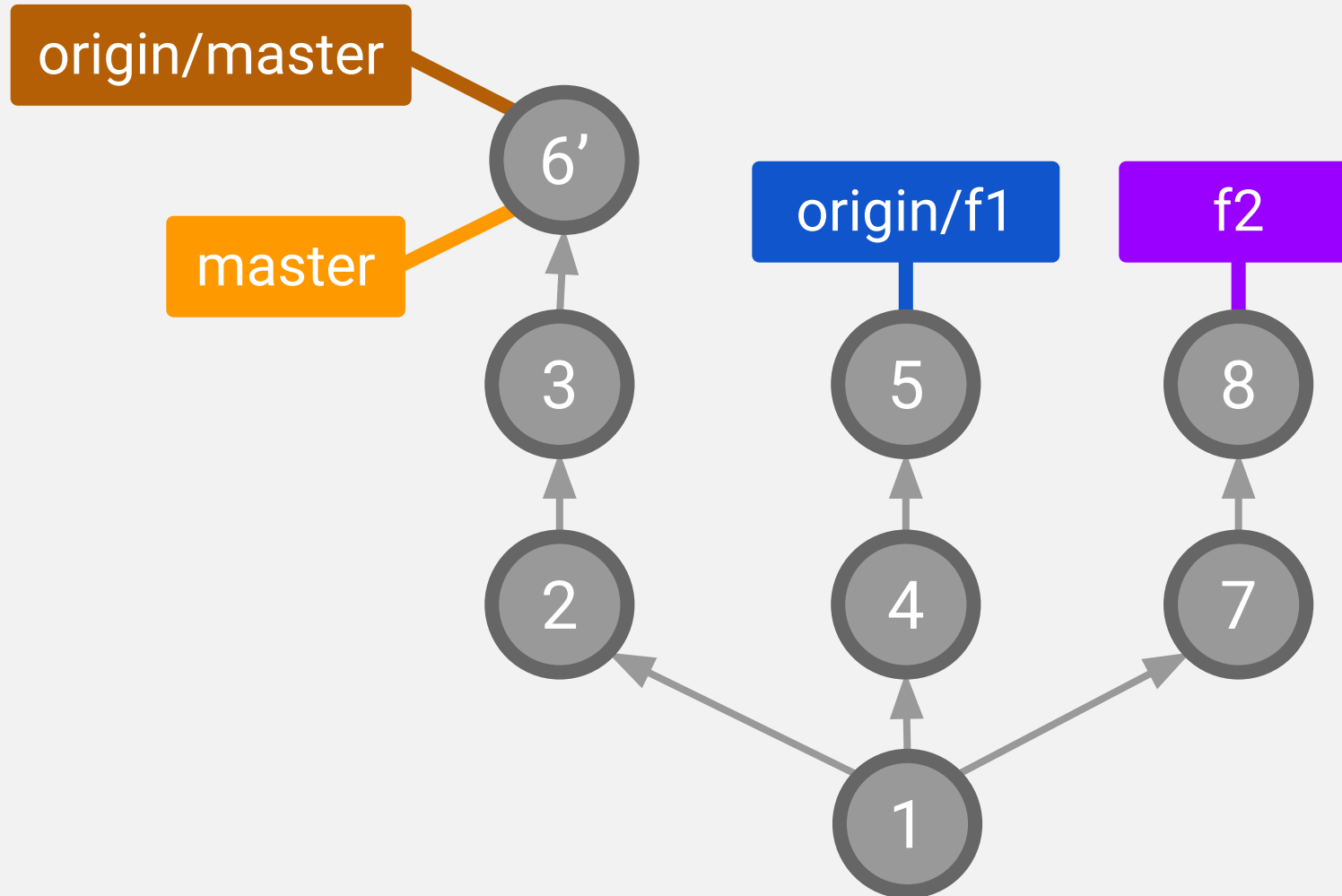
H1. Гибкое конфигурирование и качественная документация

R2. Удаленное изменение — это push

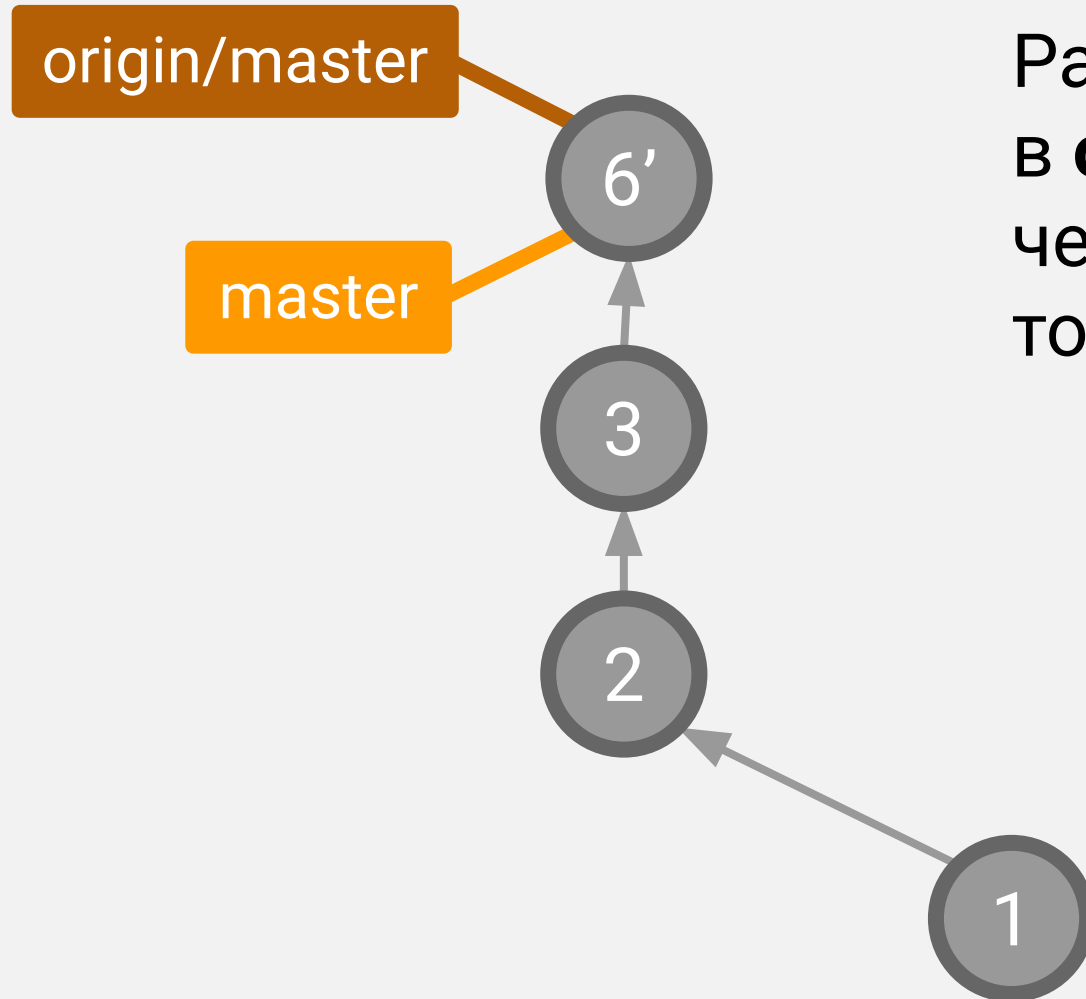
Надо отправить изменения



Применим push



Применим push



Раз **master** можно влить в **origin/master** через **fast-forward merge**, то push будет успешным

Push

1. Отправляет коммиты в удаленный репозиторий
2. Сдвигает ветки в удаленном репозитории

Требует права на запись в удаленный репозиторий, а следовательно некий способ аутентификации

Удаление удаленной ~~удаленки~~ ветки

Удаление ветки – это тоже push, только другой

```
git push <remote> -d <branch>
```

В Git Extensions разница между удалением локальной и удаленной ветки особо не чувствуется

Теги в удаленном репозитории

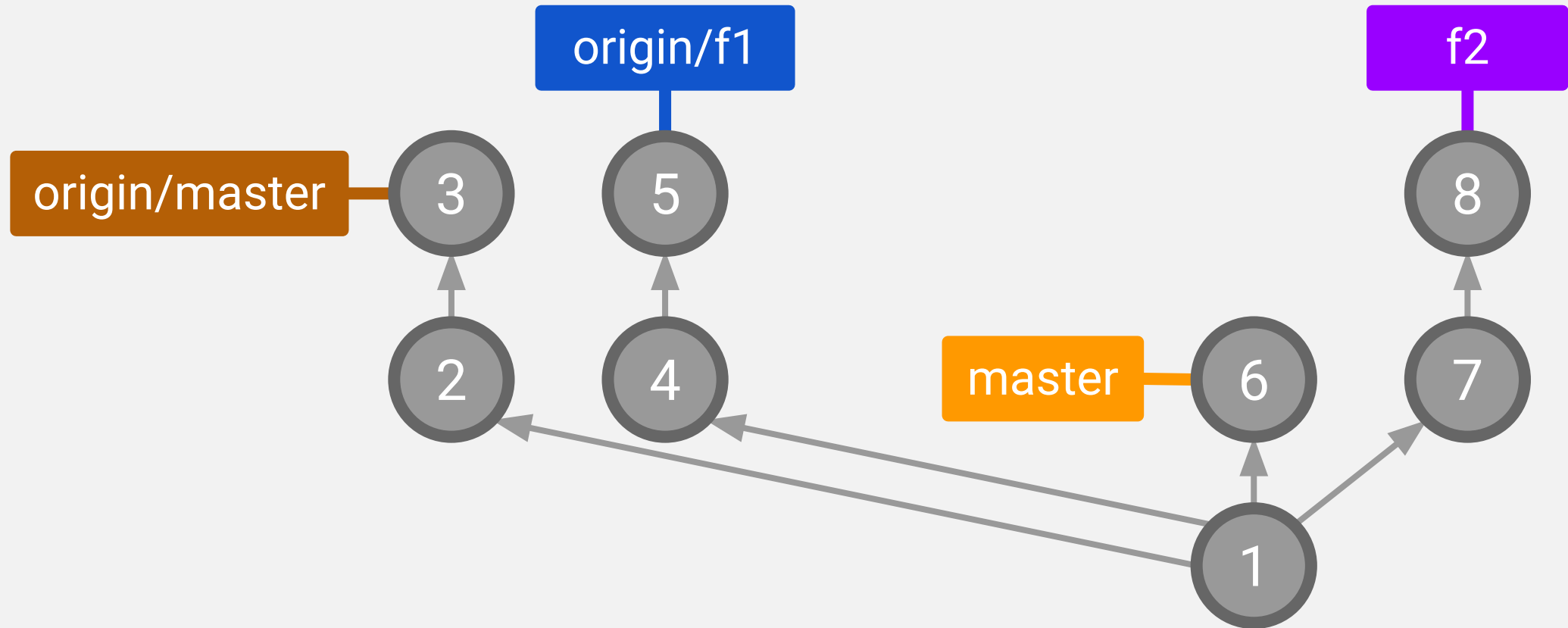
Добавление или удаление тегов в удаленном репозитории – это тоже push

```
git push <remote> tag <tag>
```

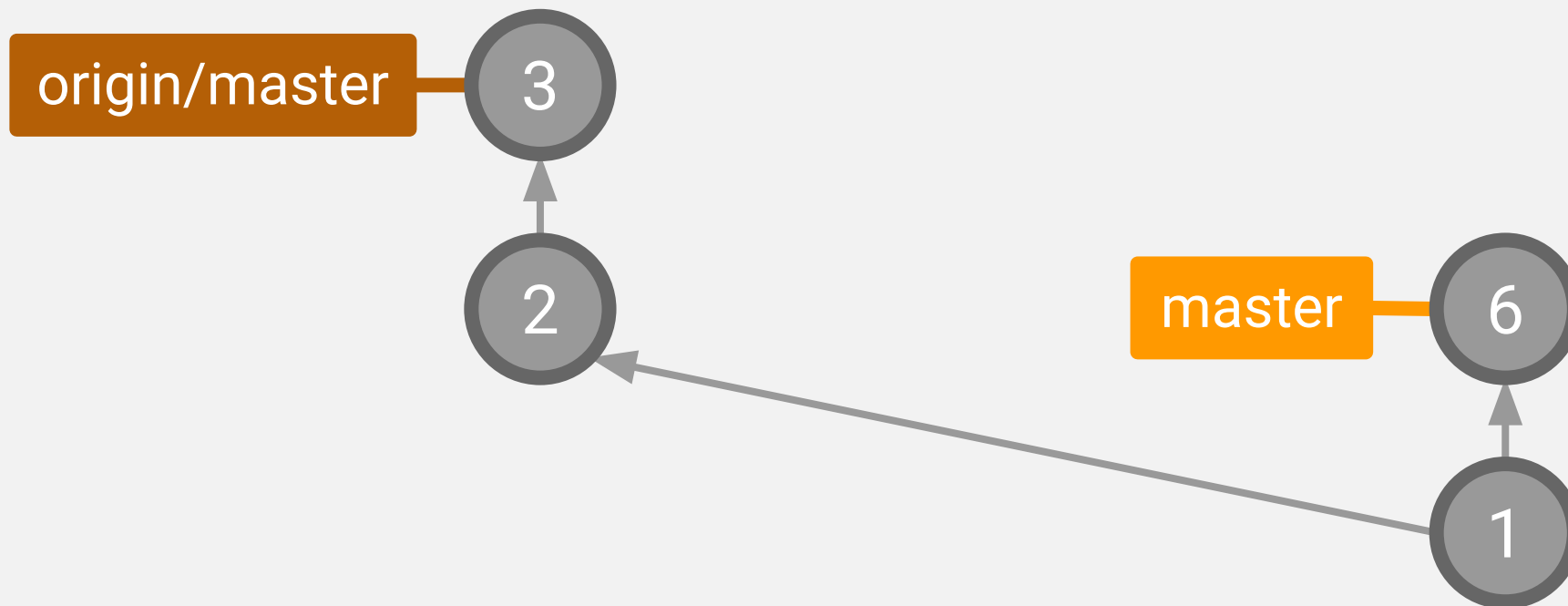
```
git push <remote> --tags
```

```
git push <remote> -d <tag>
```

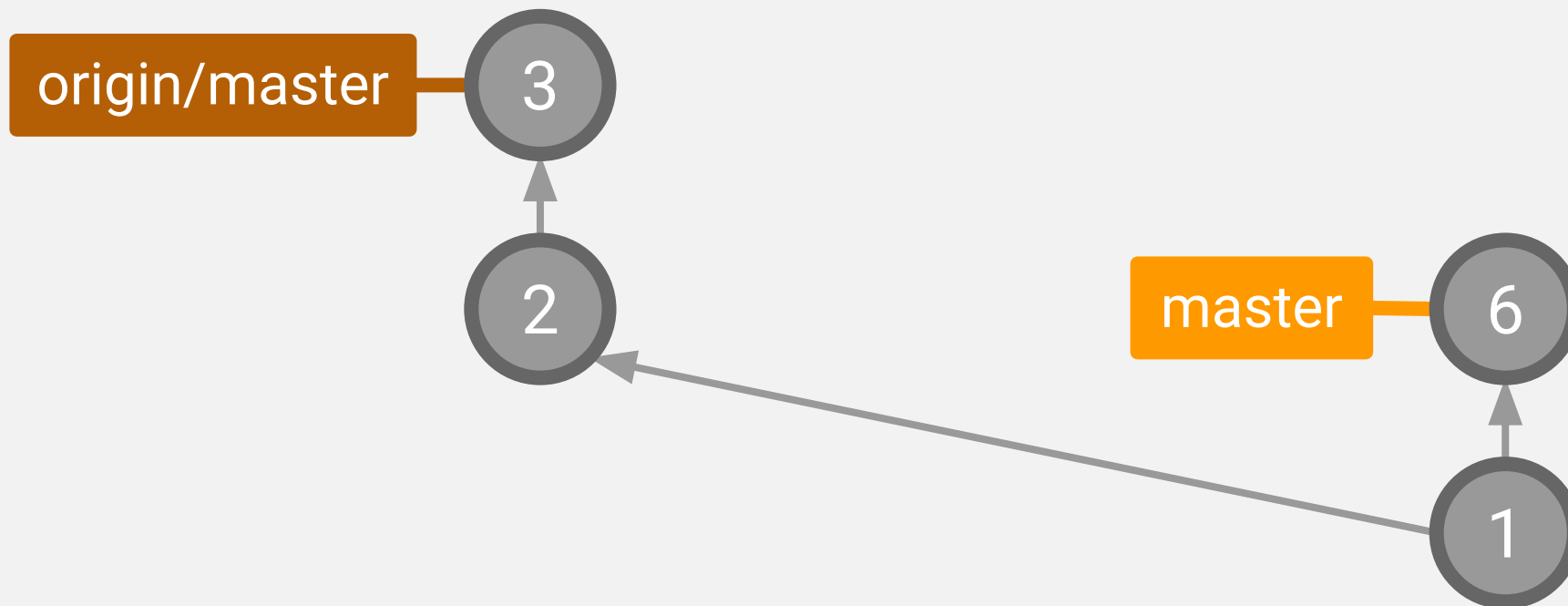
Ситуация без rebase



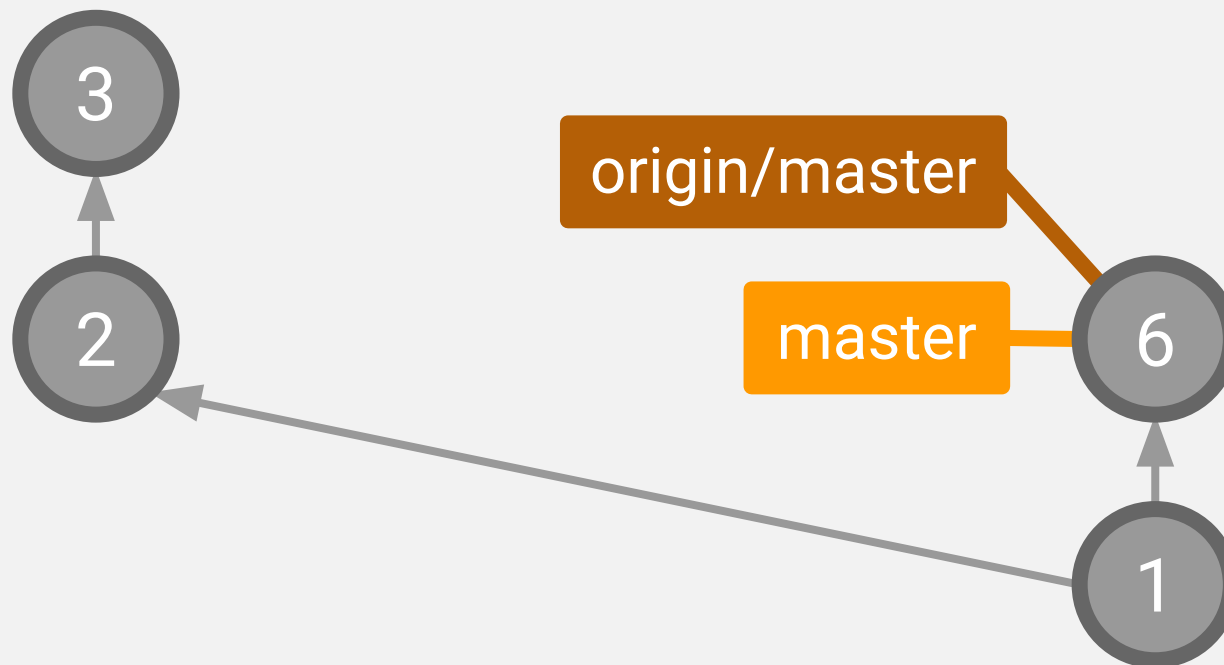
Fast-Forward Merge невозможен



Force Push может помочь...



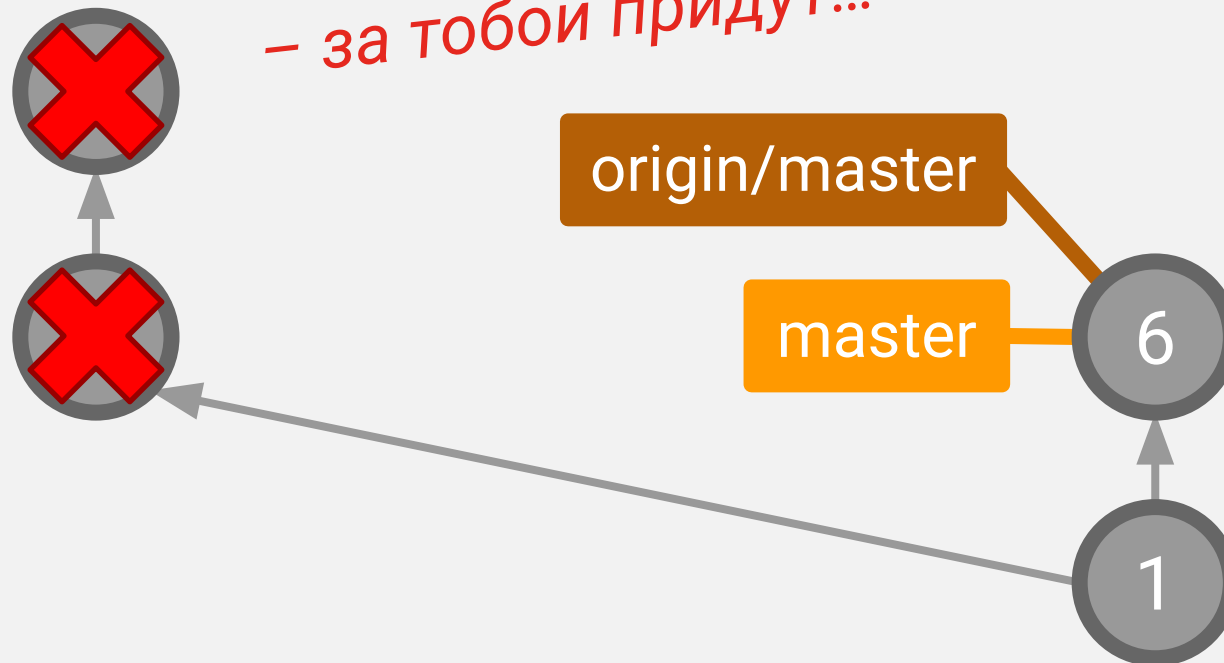
Force Push заставит ветку сдвинуться



Force Push заставит ветку сдвинуться

Коммиты будут
потеряны

Если их использовали
– за тобой придут...



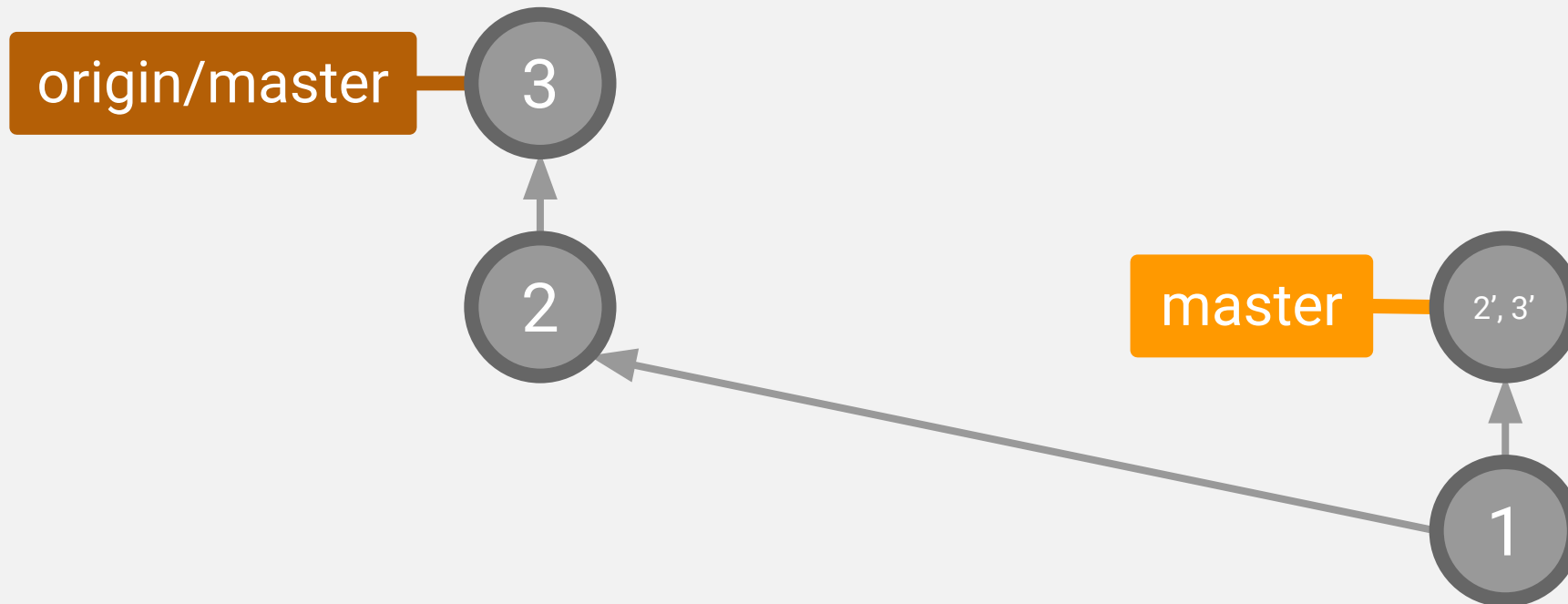


Никогда не делай
Force Push
в master

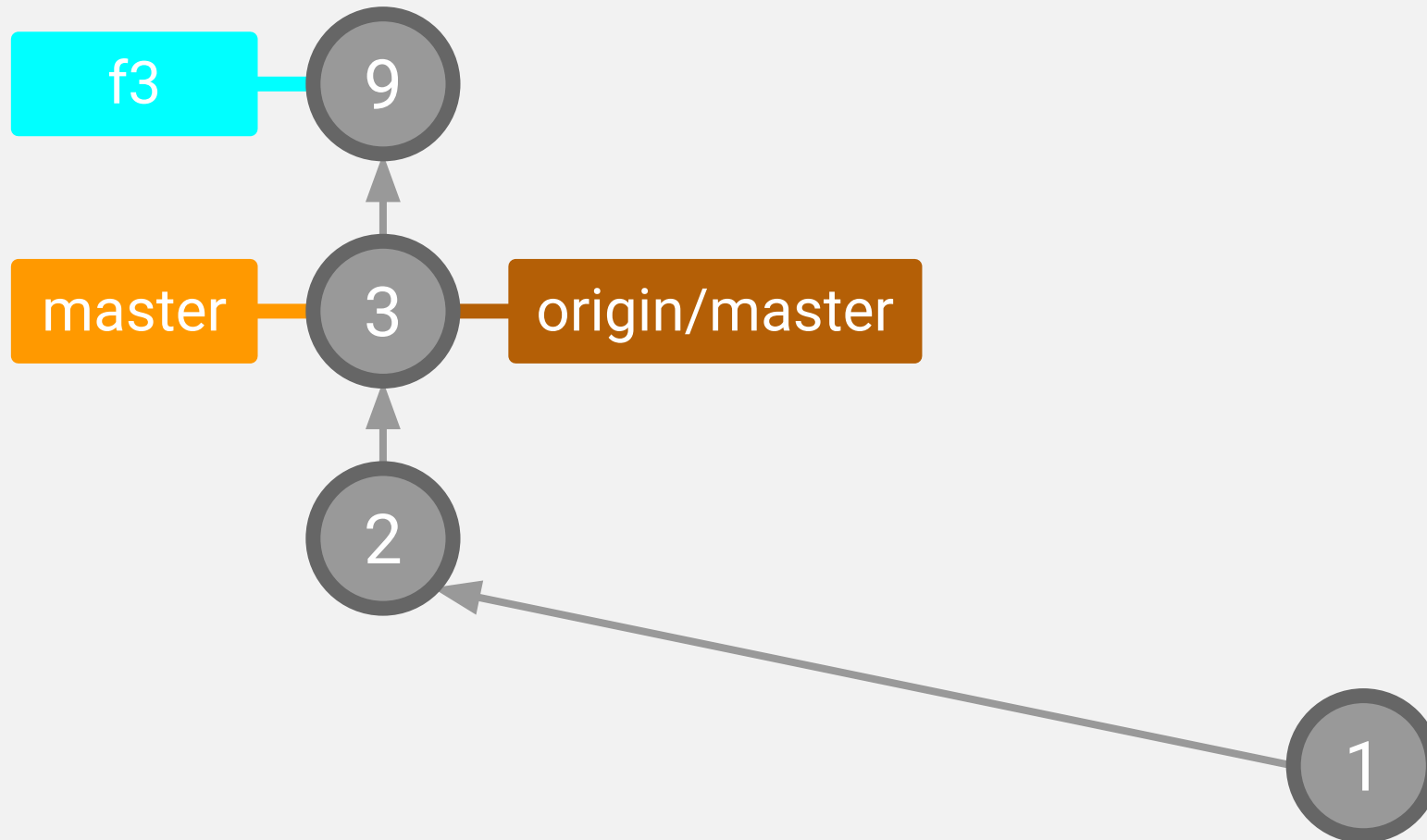
*Со своими ветками
делай что хочешь*

Реалистичный пример

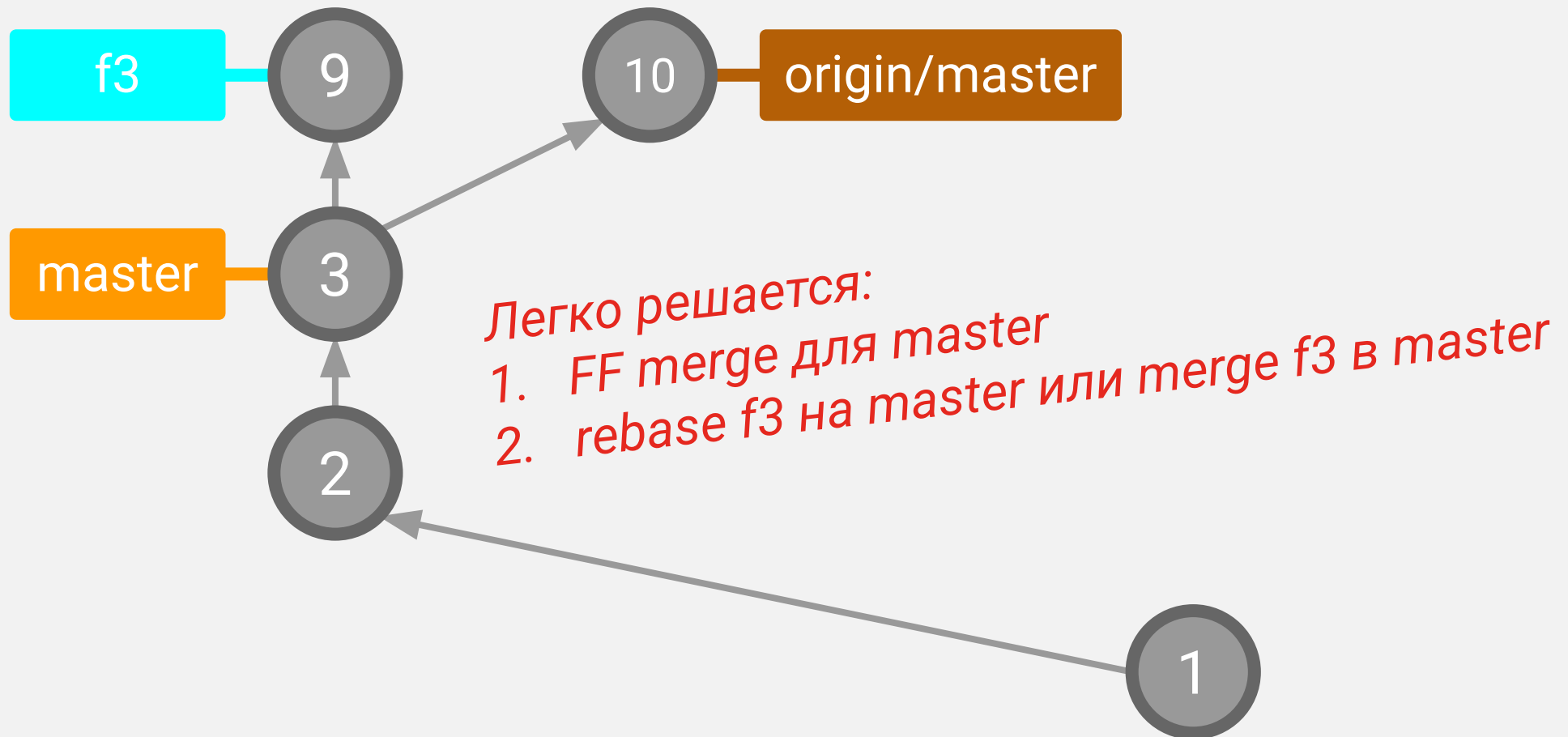
Картина мира первого разработчика



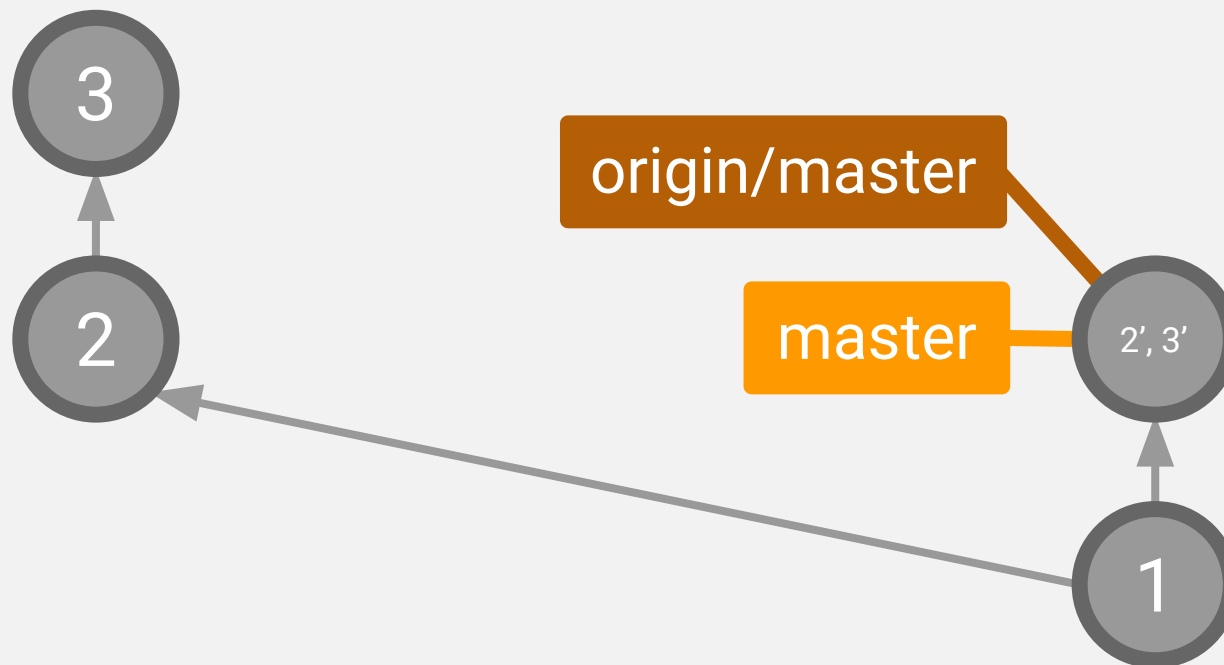
Картина мира второго разработчика



Худшие ожидания второго



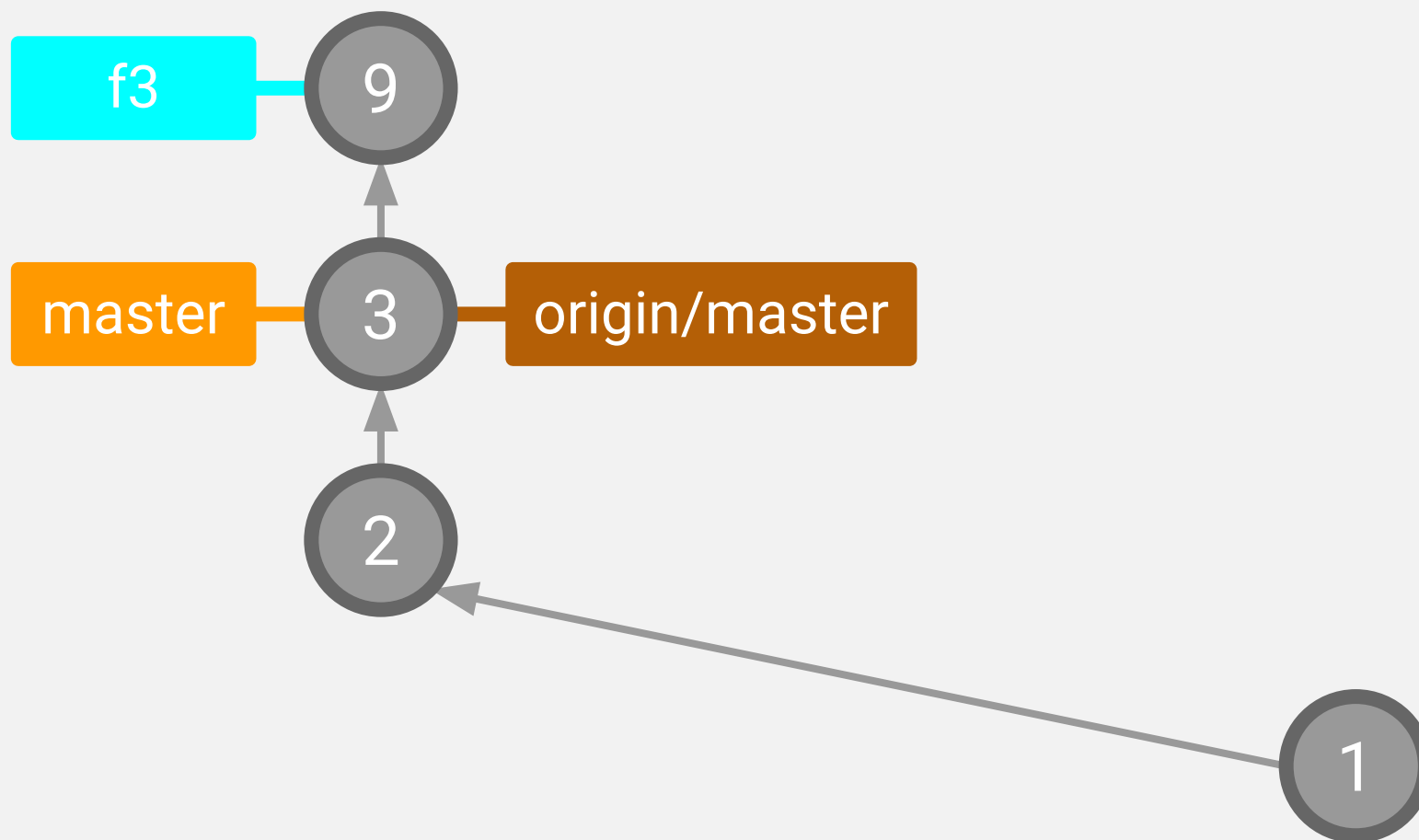
Но первый делает подлянку



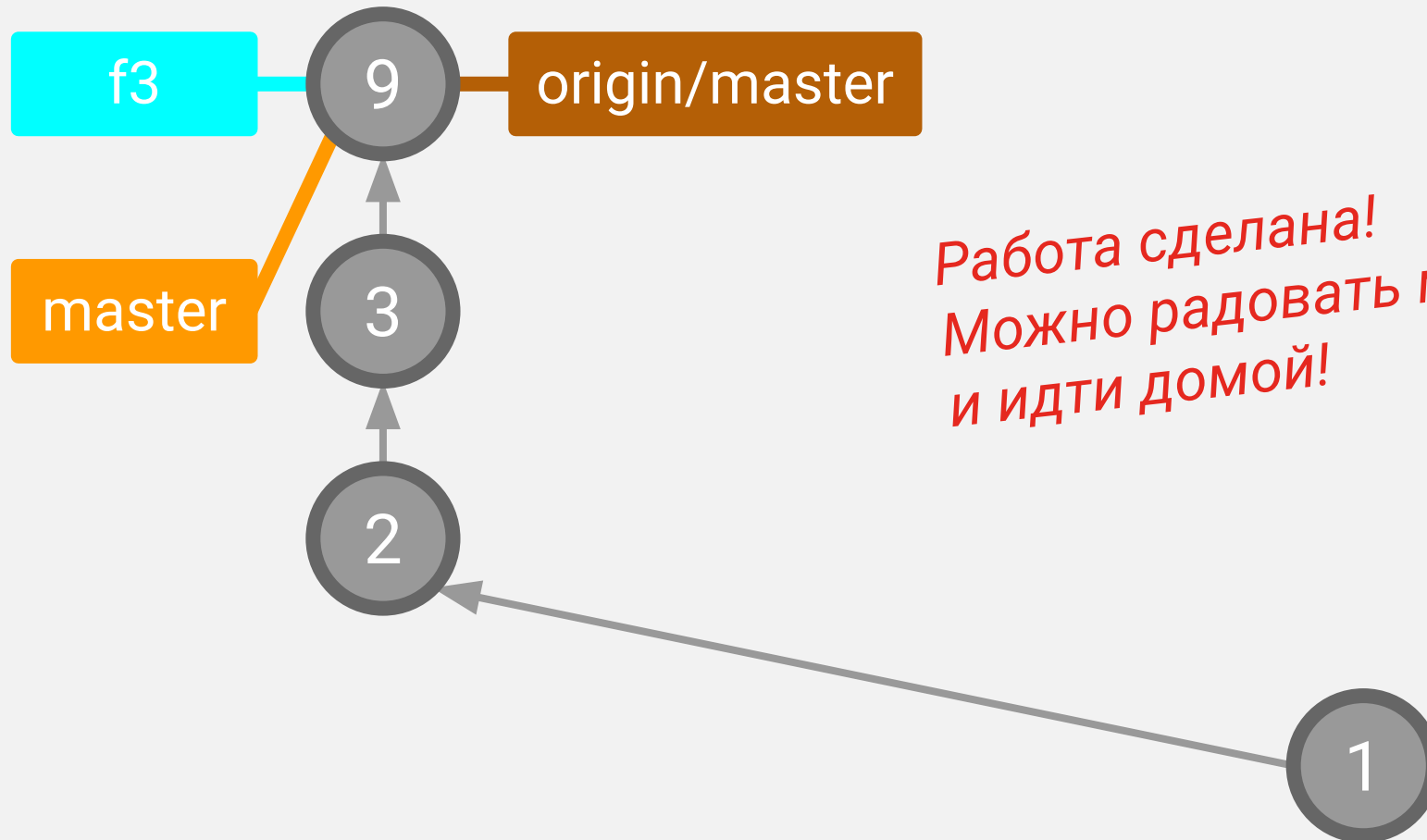
Что видит второй после fetch



А что если второй будет быстрее?

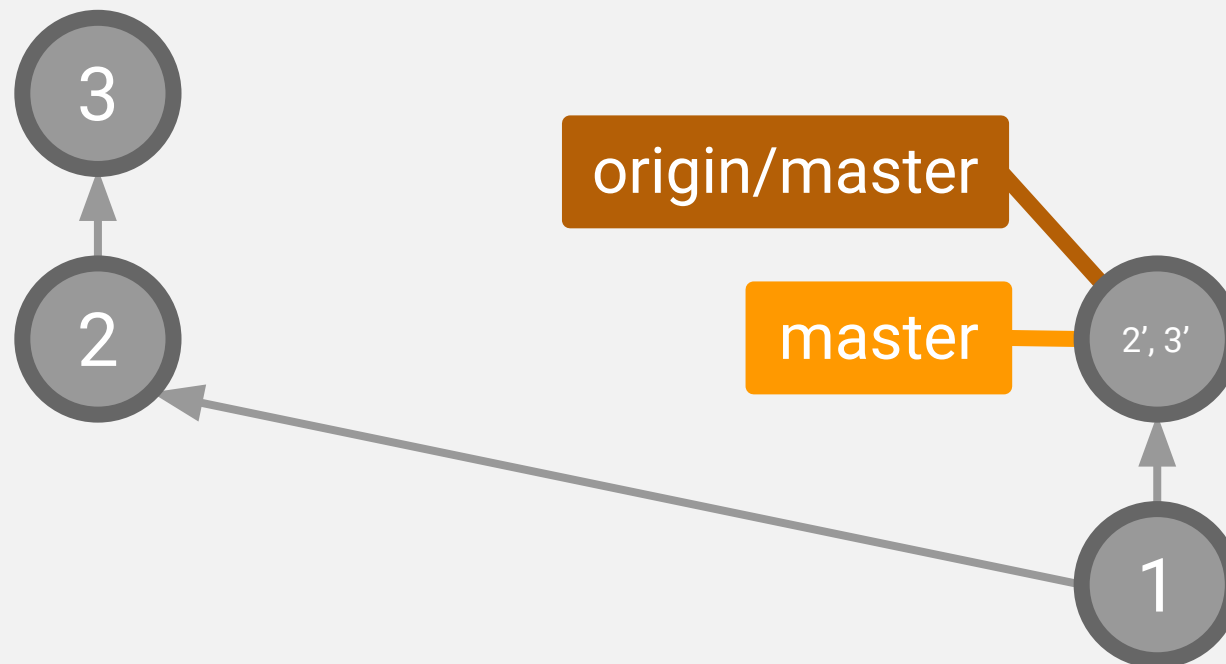


Вольет свою ветку в master и запустит?

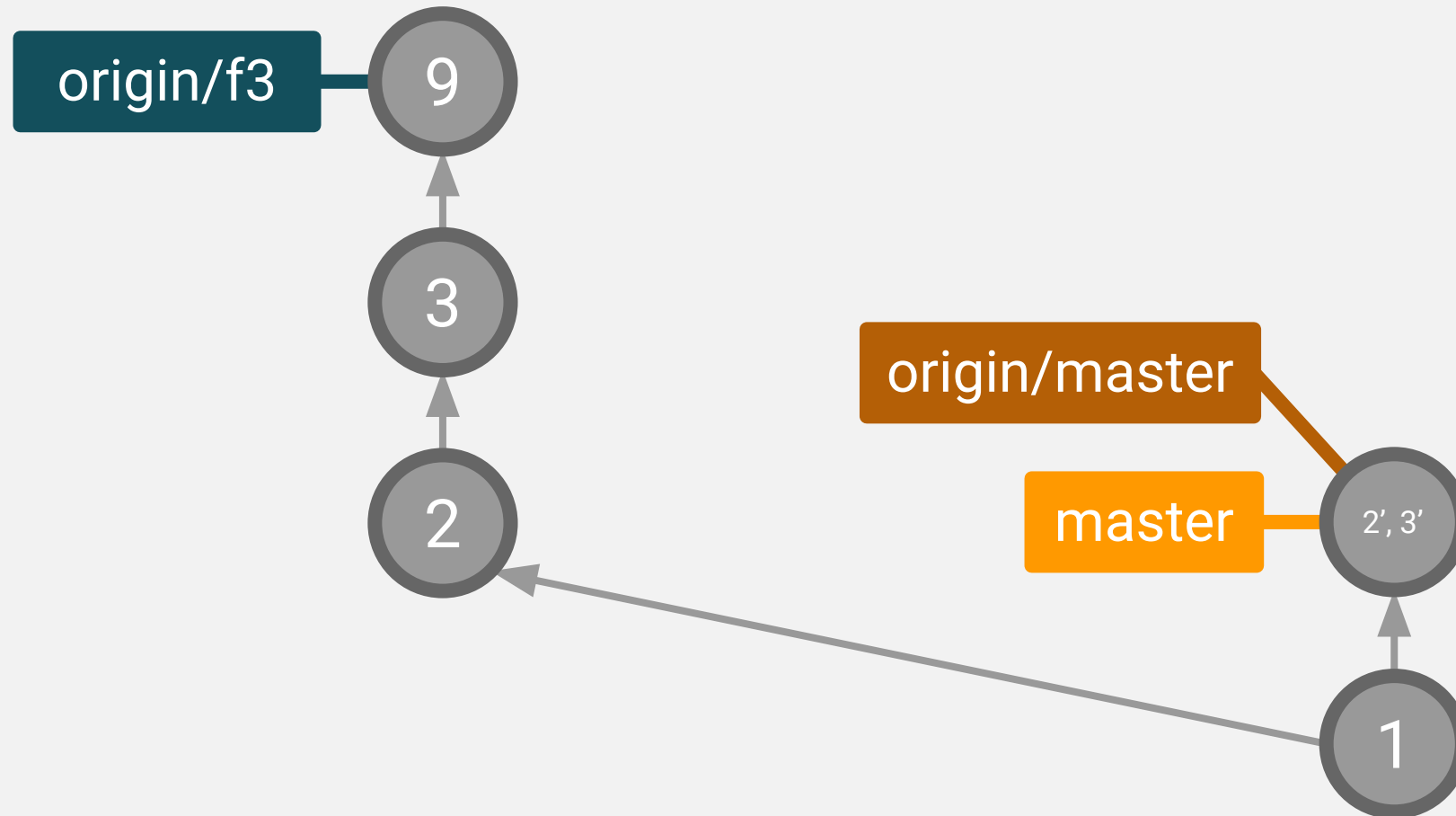


*Работа сделана!
Можно радовать менеджера
и идти домой!*

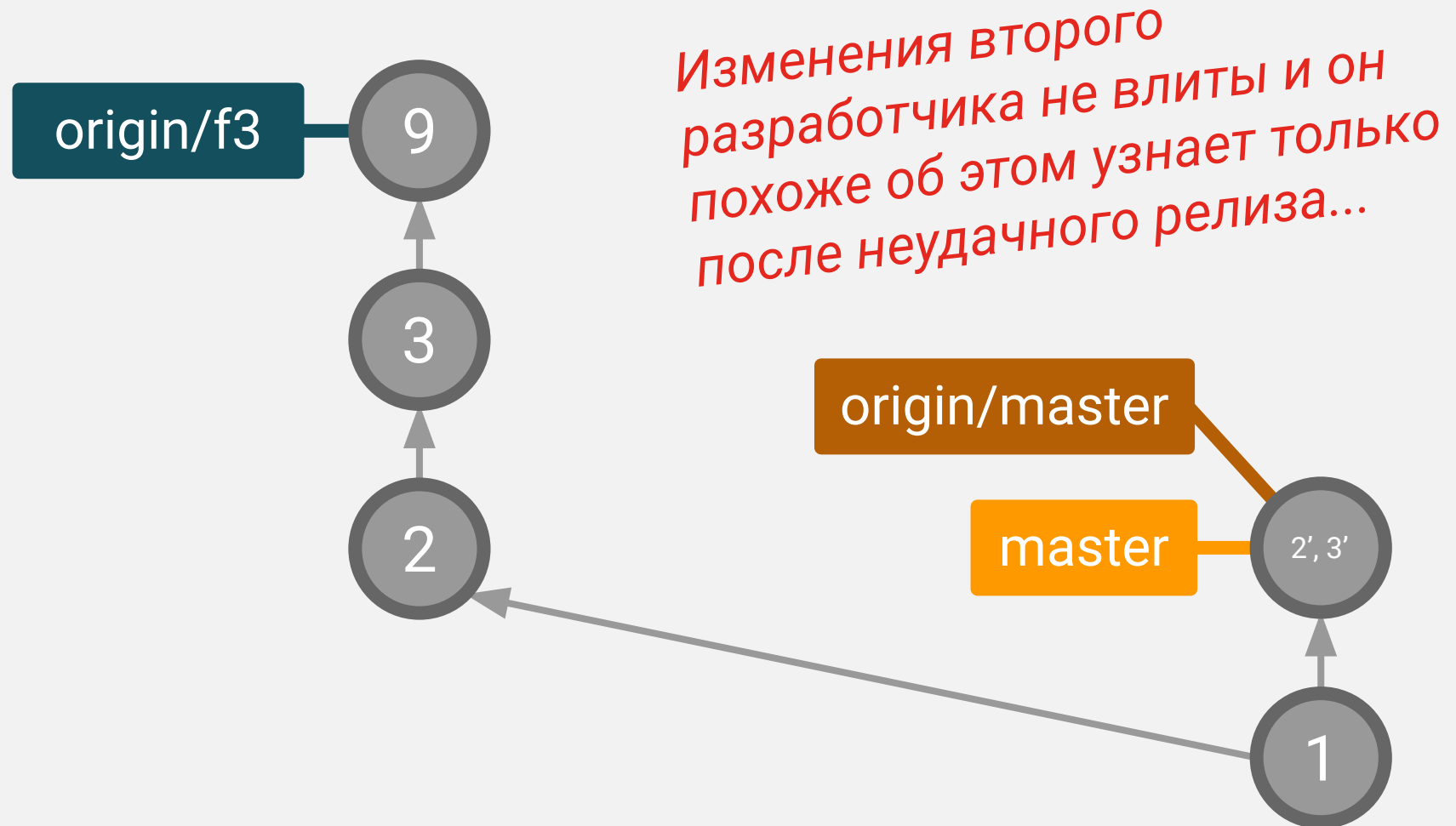
Затем первый в неведении делает подлянку



Итоговая картина



Итоговая картина

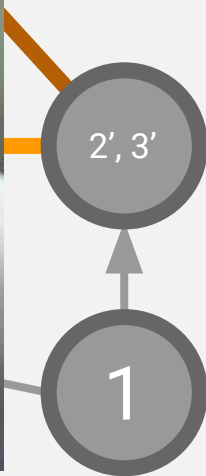


Итоговая к

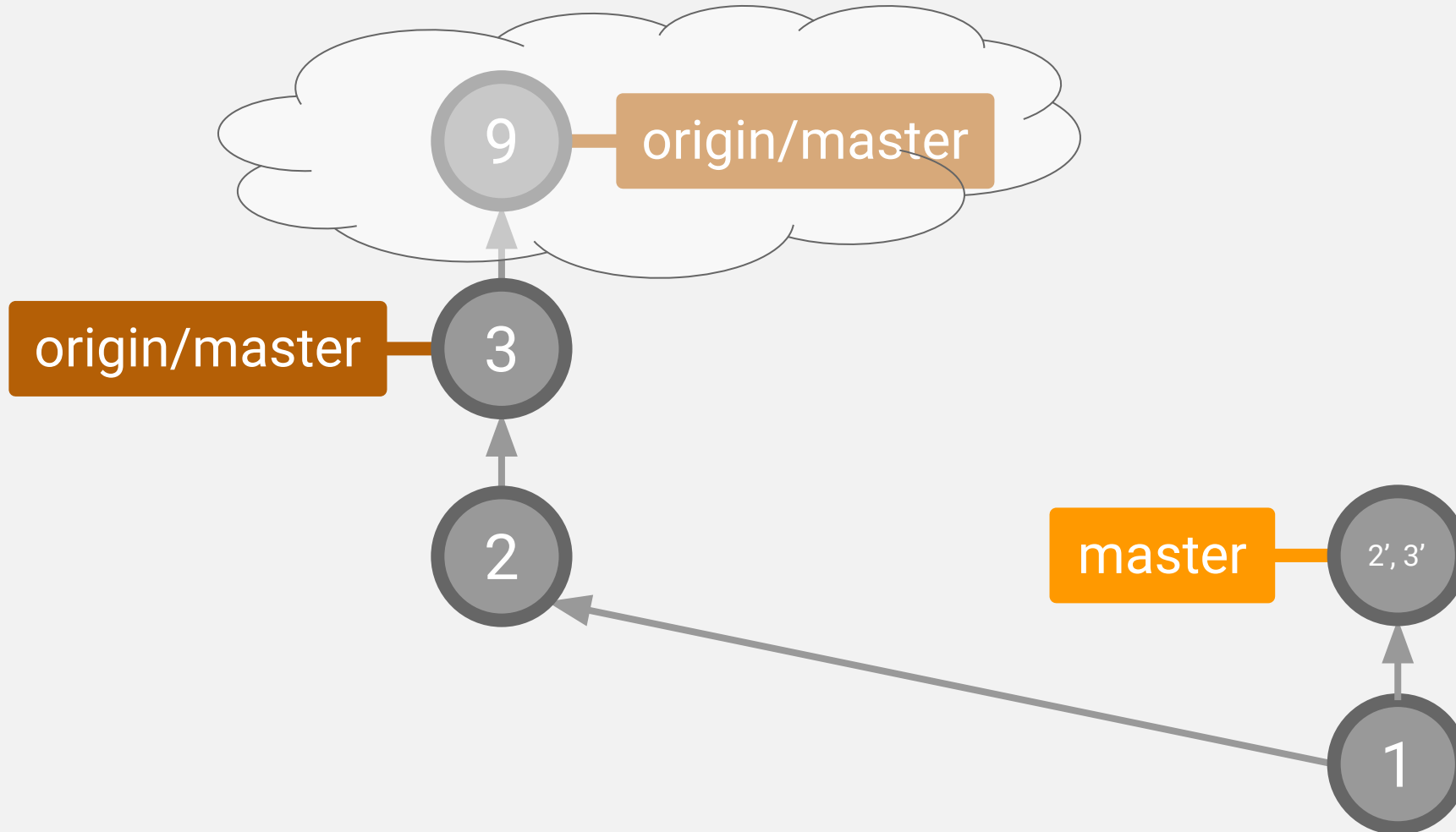
origin/f3



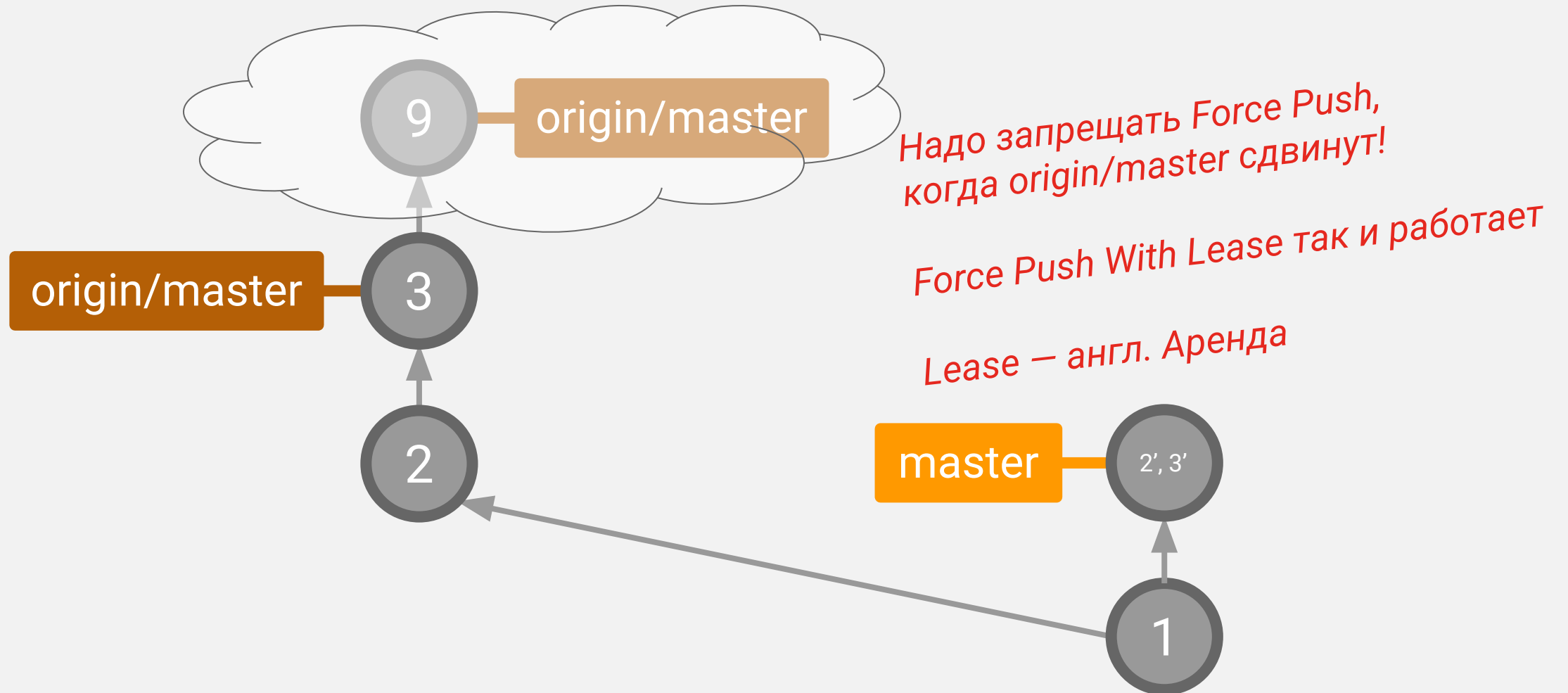
ты и он
ет только
лиза...



Пристальный взгляд на проблему



Пристальный взгляд на проблему





Никогда не делай Force Push
Не делай Force Push With Lease в master

*Со своими ветками делай что хочешь,
но используй Force Push With Lease*

Удобный алиас

```
git config --global alias.please "push --force-with-lease"
```



Задание 14. Push

Structure

Actions

Remote

S1. Все локально

A1. Трехсторонний merge
в три шага

R1. Доступен fetch коммитов
любого репозитория
в любой момент

S2. Хранятся
состояния директории,
постепенная сборка коммита

A2. rebase, cherry-pick и amend,
чтобы пересоздать историю

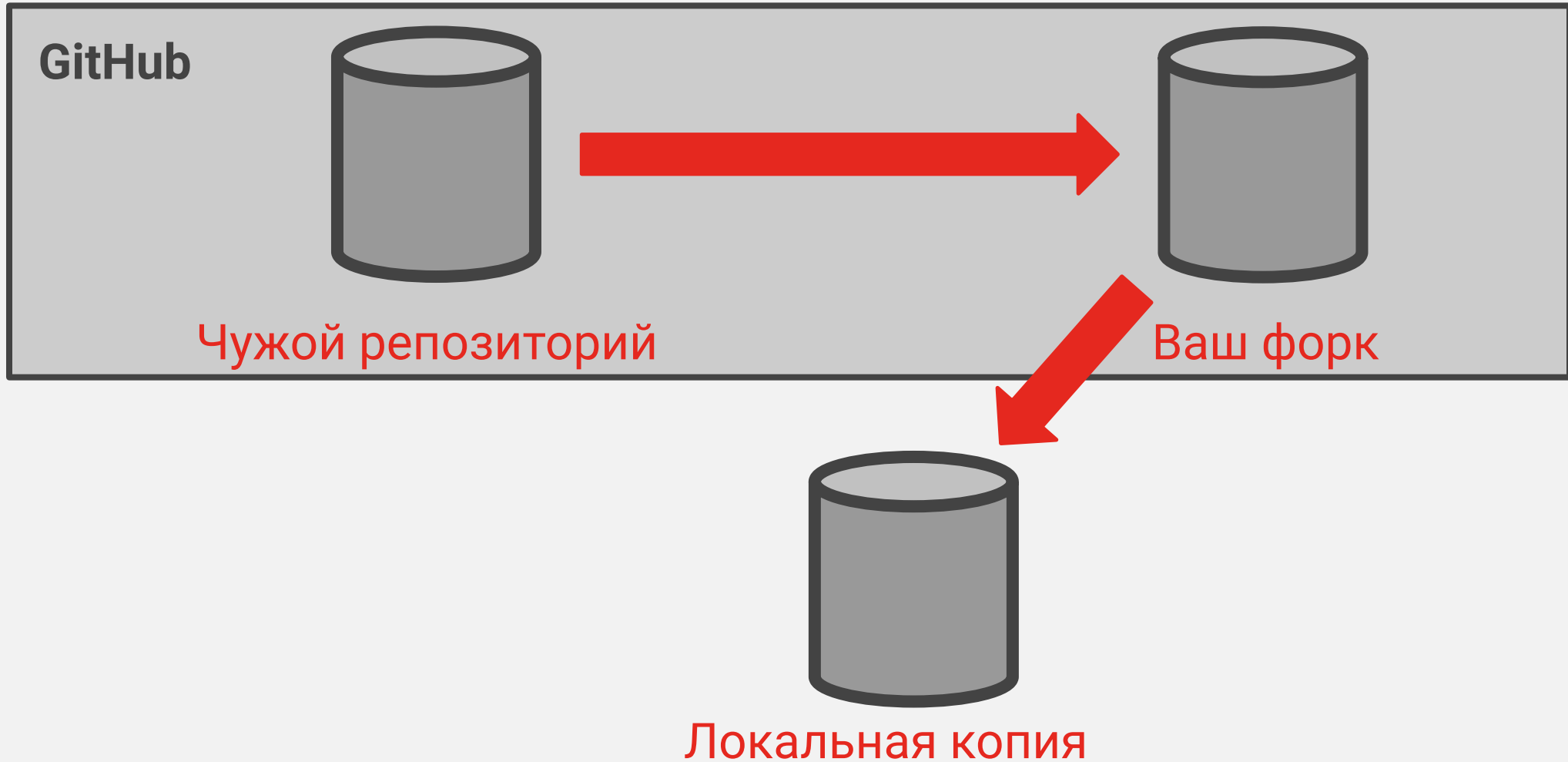
R2. Удаленное изменение
— это push

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

H1. Гибкое конфигурирование и качественная документация

R3. Явное сопоставление локальных веток с upstream

Upstream repository



Upstream repository



Как push понимает какую ветку сдвигать?

1. Никак – надо всегда прописывать обе ветки явно
2. Должно совпадать имя
3. Есть внутренняя таблица сопоставления

Git использует **все** способы,
режим сопоставления можно настраивать

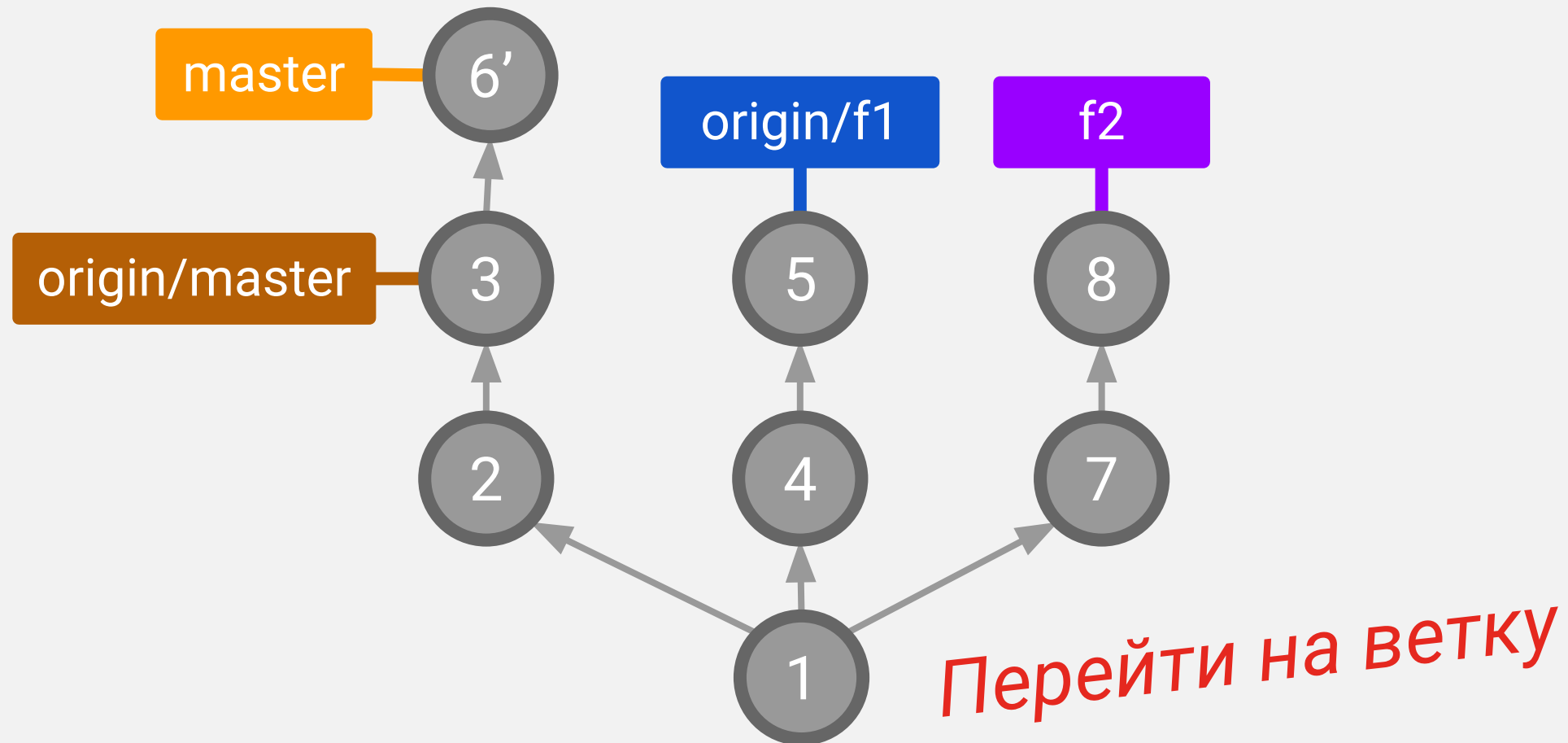
По умолчанию режим simple

Для основного репозитория

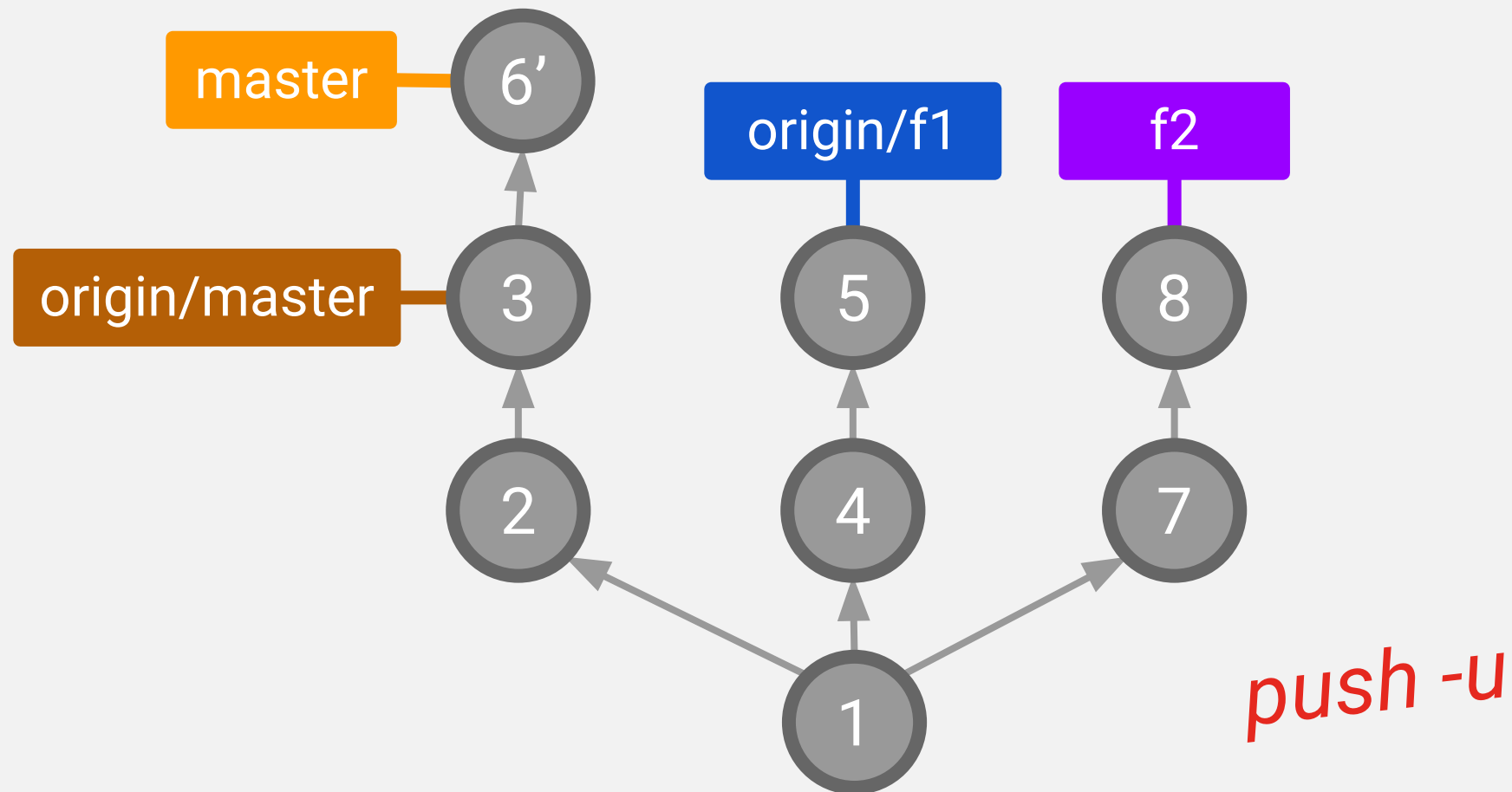
- Используется сопоставление из таблицы
- При push новой ветки достаточно указать намерение
`git push -u origin HEAD`
- Чтобы создать локальную ветку для удаленной с записью в таблицу достаточно на нее перейти
`git checkout <branchname>`

~~Для других репозиториях идет сопоставление по имени ветки~~
можно не запоминать

Как создать привязку для f1?



Как создать привязку для f2?



Удобный алиас

```
git config --global alias.pushup "push -u origin HEAD"
```



Upstream mapping в Git Extensions

The screenshot shows the 'Remote repositories' dialog box in Git Extensions. It has two tabs: 'Remote repositories' and 'Default pull behavior (fetch & merge)'. The 'Remote repositories' tab contains a table with the following data:

Local branch name	Remote repository	Default merge with
f2		
master	origin	master
f1	origin	f1

The 'Default pull behavior (fetch & merge)' tab is active and shows configuration for the selected 'master' branch:

- Local branch name: master
- Remote repository: origin
- Default merge with: master

There is a 'Save changes' button with a floppy disk icon. At the bottom of the dialog, there are two buttons: 'Prune remote branches' and 'Update all remote branch info'.

Upstream mapping в консоли

```
$ git branch -vv
f1      f9ea67c [origin/f1] commit in f1 branch
* f2    f8e8ab2 commit in f2 branch
master ace37b6 [origin/master] Initial commit
```

Local branch

Remote name

Upstream branch

Pull тоже использует upstream mapping

```
git pull origin =  
  git fetch + git merge origin/<upstream_branch>
```

```
git pull --rebase origin =  
  git fetch + git rebase origin/<upstream_branch>
```

Pull нужно знать с какой веткой делать merge или rebase
и здесь тоже нужно сопоставление веток

Безопасный pull

Либо fast-forward merge, либо без merge

С помощью опций:

```
pull --ff-only
```

Установить для pull по умолчанию:

```
git config --global pull.ff only
```

Удобный алиас

```
git config --global alias.pure "pull --rebase --autostash"
```



Задание 15. Upstream

Structure

Actions

Remote

S1. Все локально

A1. Трехсторонний merge
в три шага

R1. Доступен fetch коммитов
любого репозитория
в любой момент

S2. Хранятся
состояния директории,
постепенная сборка коммита

A2. rebase, cherry-pick и amend,
чтобы пересоздать историю

R2. Удаленное изменение
— это push

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

R3. Явное сопоставление
локальных веток
с upstream

H1. Гибкое конфигурирование и качественная документация

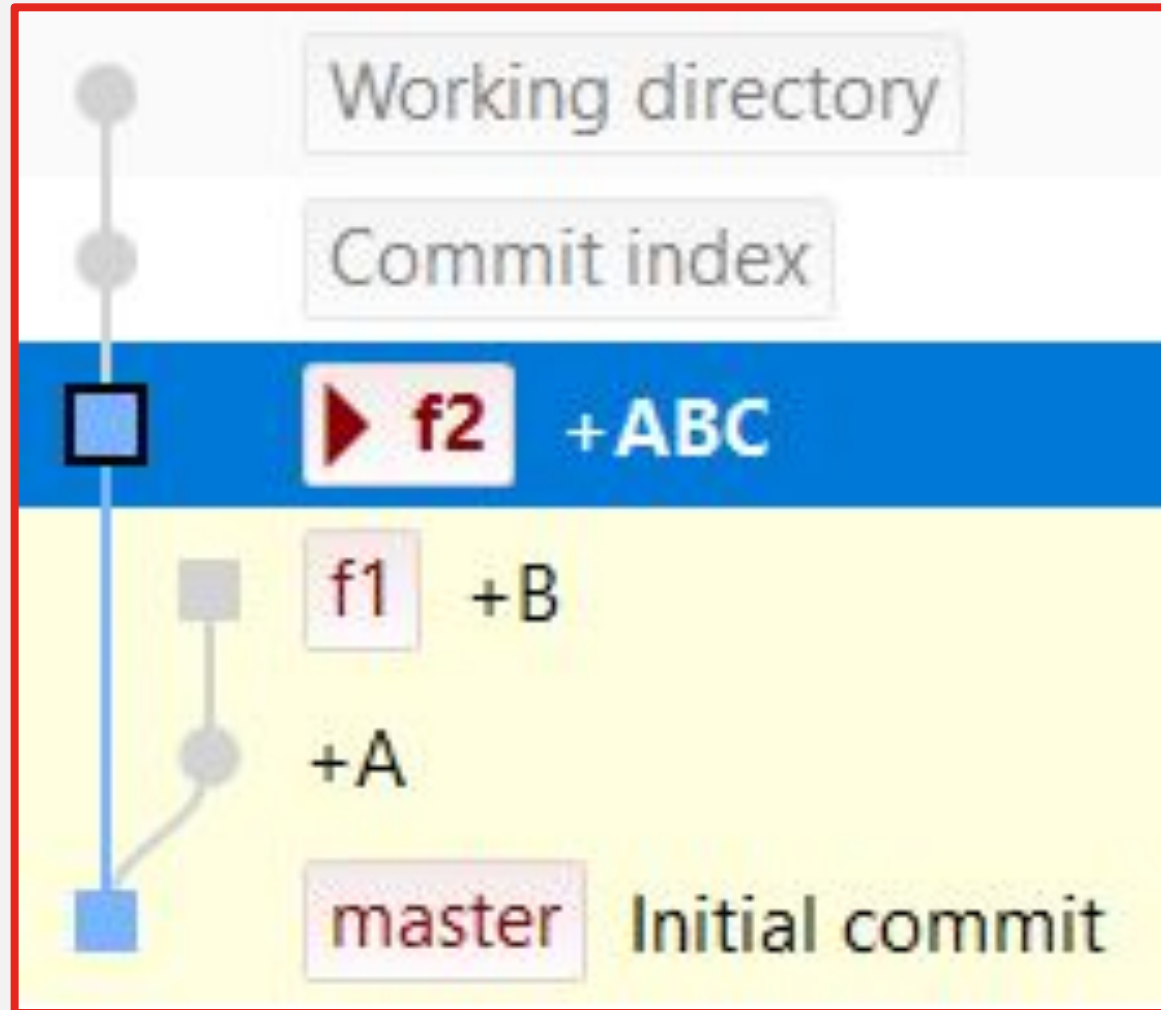
A3. stash, reset, revert для управления изменениями

Изменения можно временно припрятать

Можно получить разницу между любыми коммитами

Коммит можно отменить другим коммитом

Что произойдет при влитии f1 в f2?

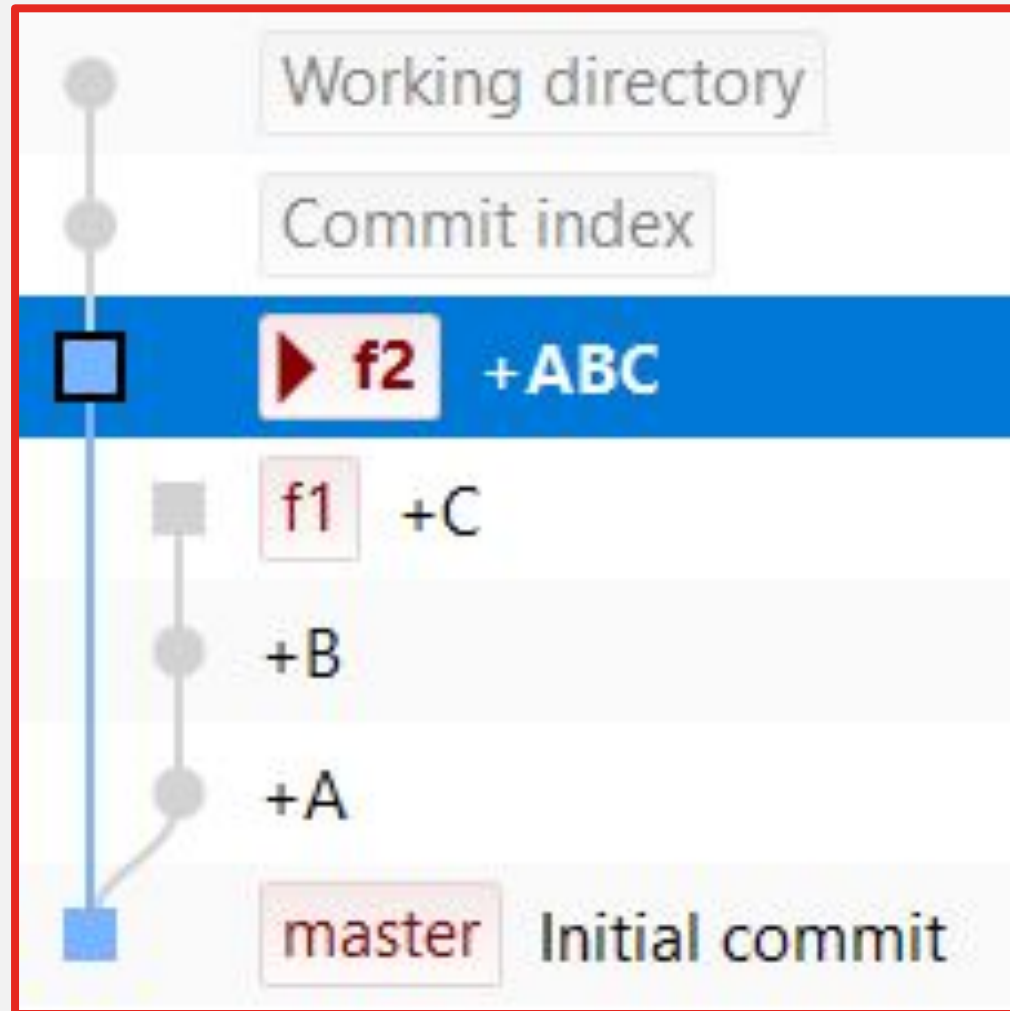


Произойдет конфликт

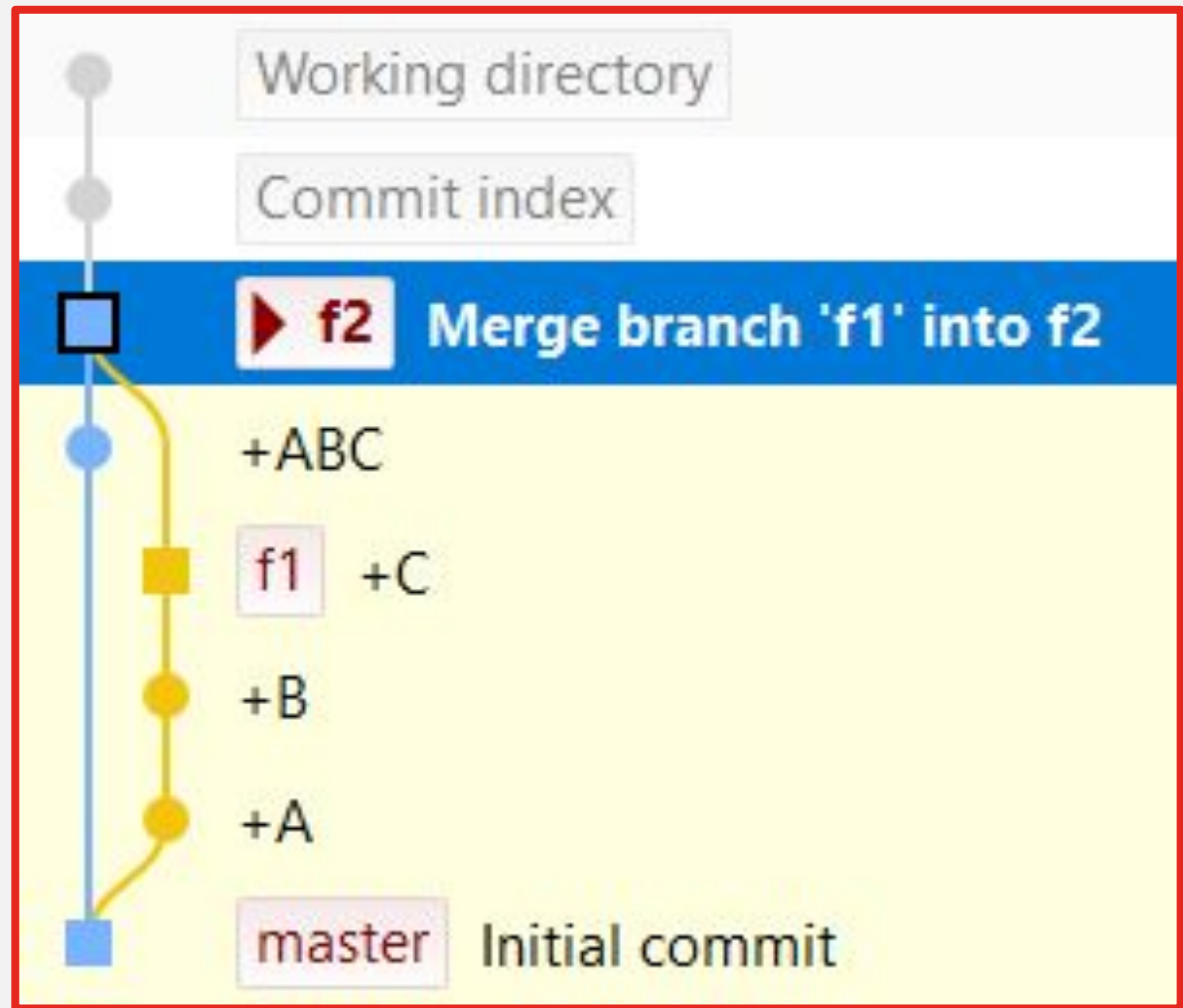
```
file.txt x file.txt: Current Changes ↔ Incoming Changes
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<<<< HEAD (Current Change)
2 ABC
3 =====
4 AB
5 >>>>>>> f1 (Incoming Change)
6
```

```
file.txt file.txt: Current Changes ↔ Incoming Changes x
1 - ABC 1 + |AB
2 2
```

Что произойдет при влитии f1 в f2?

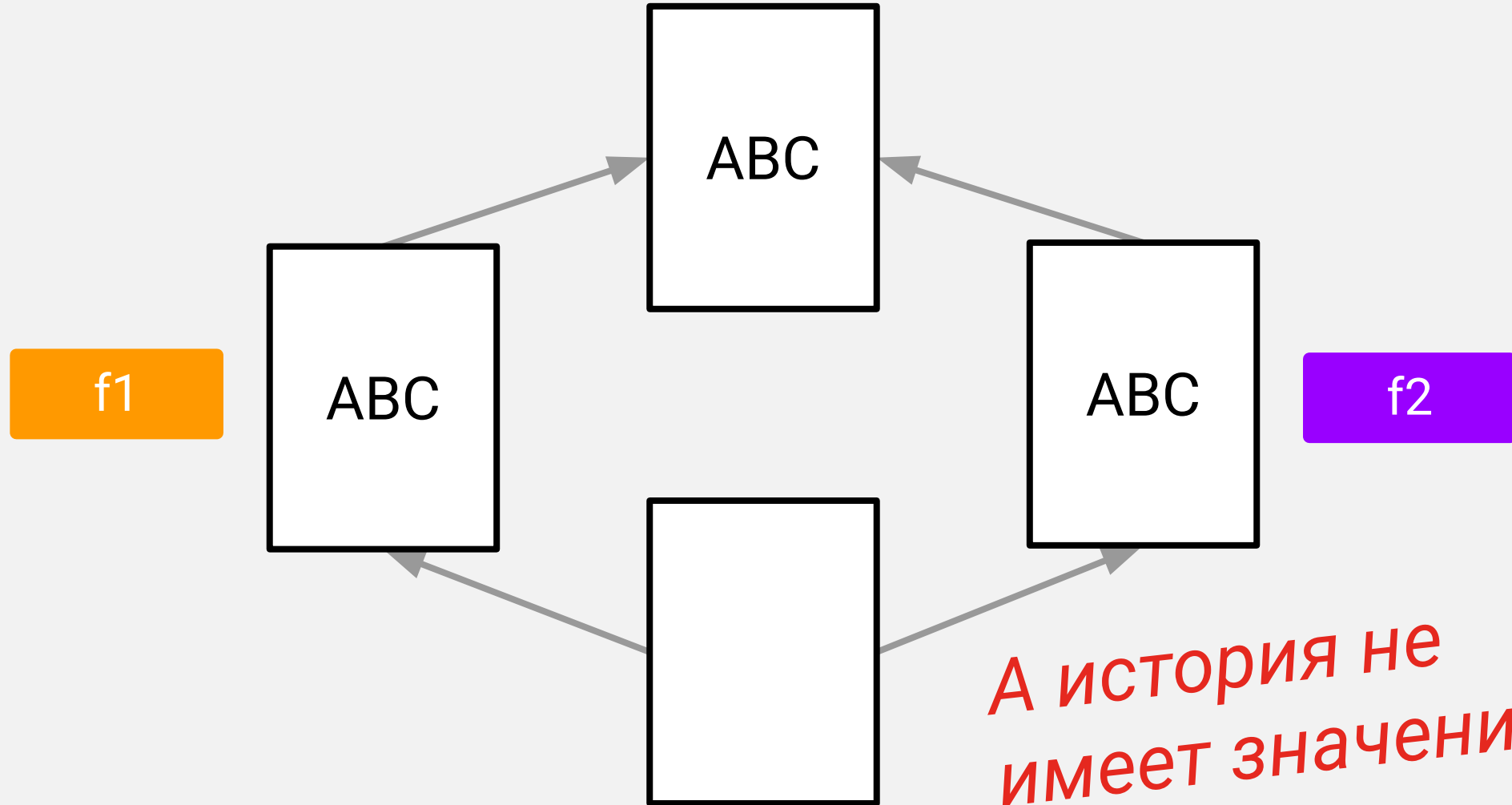


Бесконфликтное слияние



Почему?

Состояния слева и справа совпадают

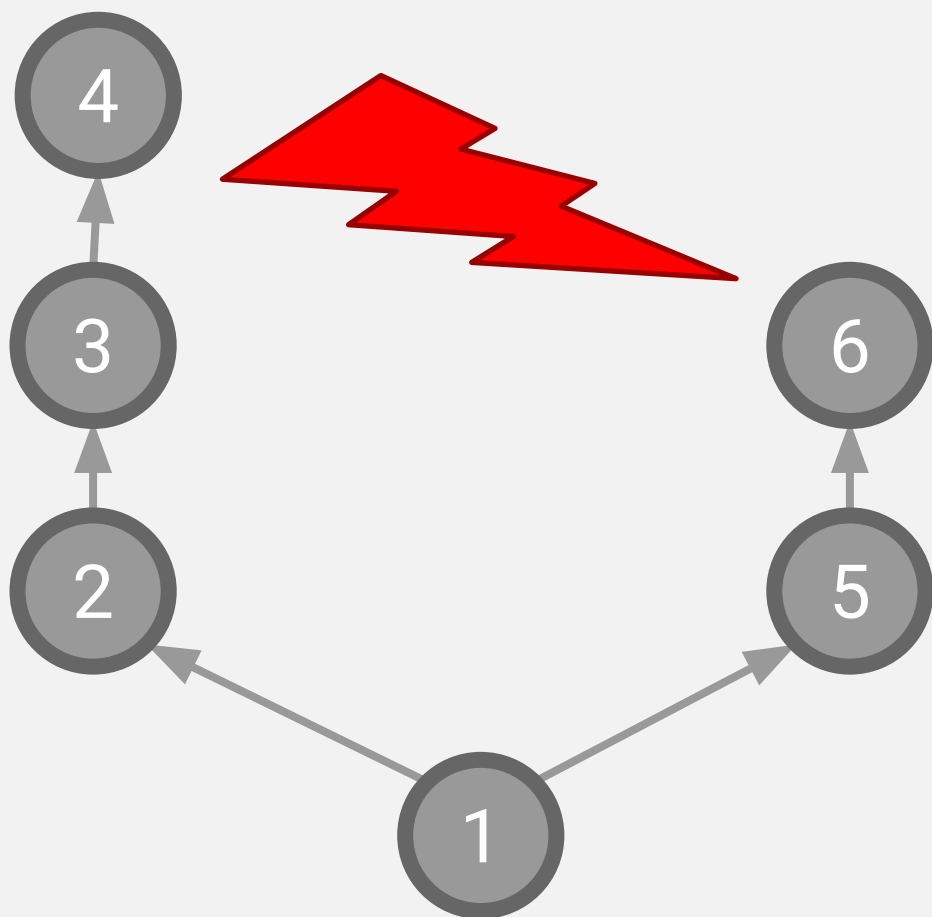


А история не имеет значения

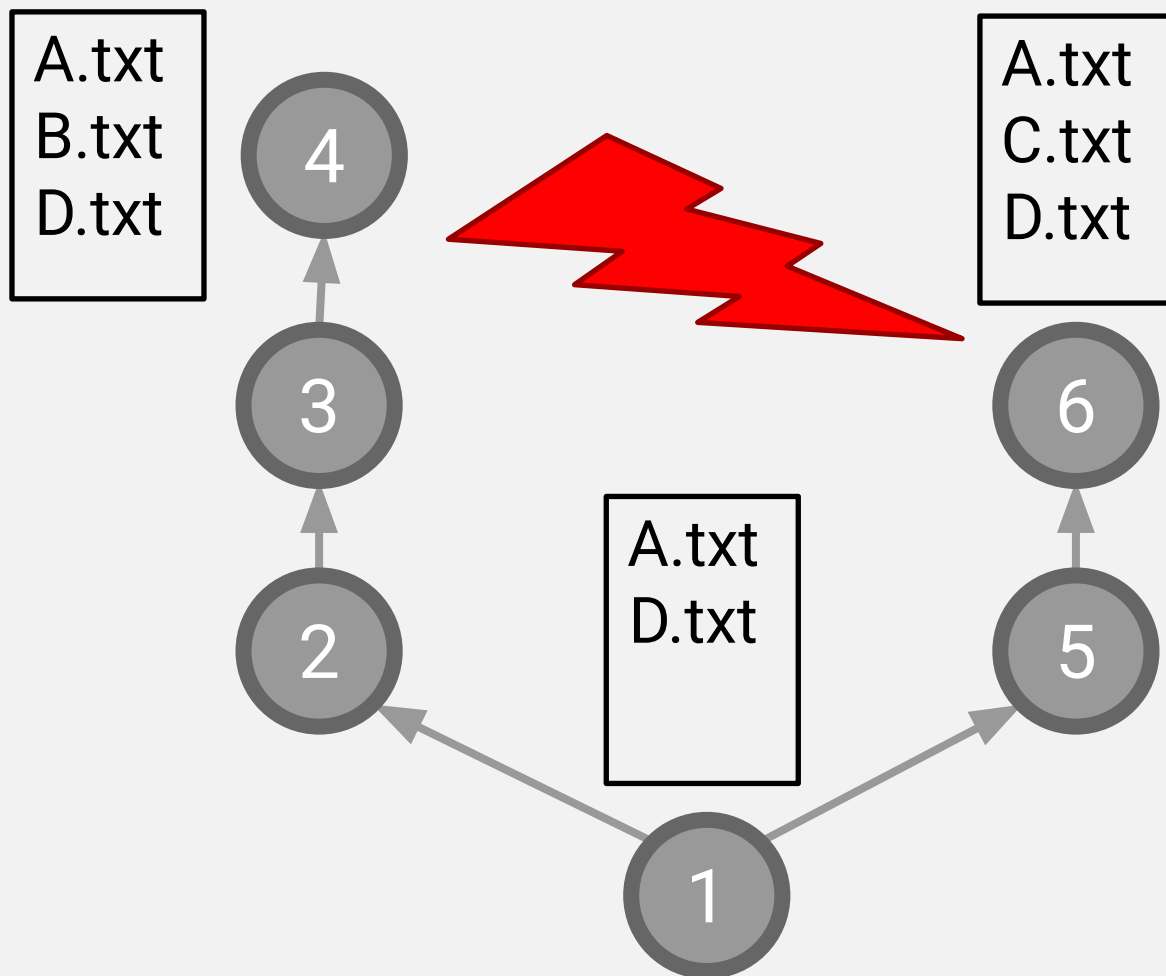
Хранятся файлы, разница вычисляется на лету

1. Каждый коммит хранит структуру каталога и все файлы состояния директории
2. Хранение файлов оптимизировано: **файлы не хранятся повторно**, потому что в структуре каталога хранятся не сами файлы, а ссылки по хэшу на них
3. **Используется сжатие**, чтобы текстовые данные занимали меньше места
4. Разница между коммитами вычисляется на лету и с родителем и с любым другим коммитом

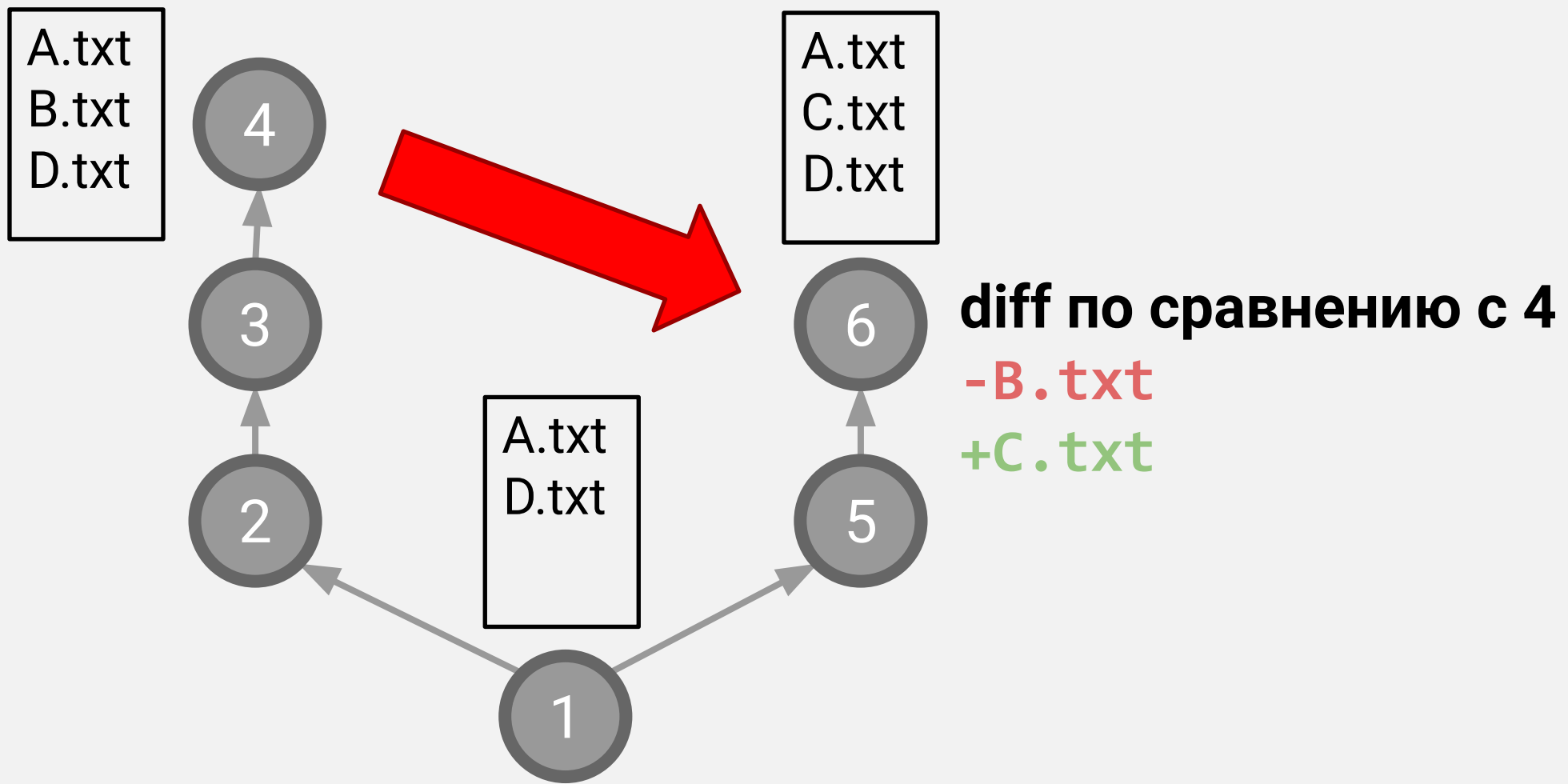
Diff можно получить между любыми коммитами



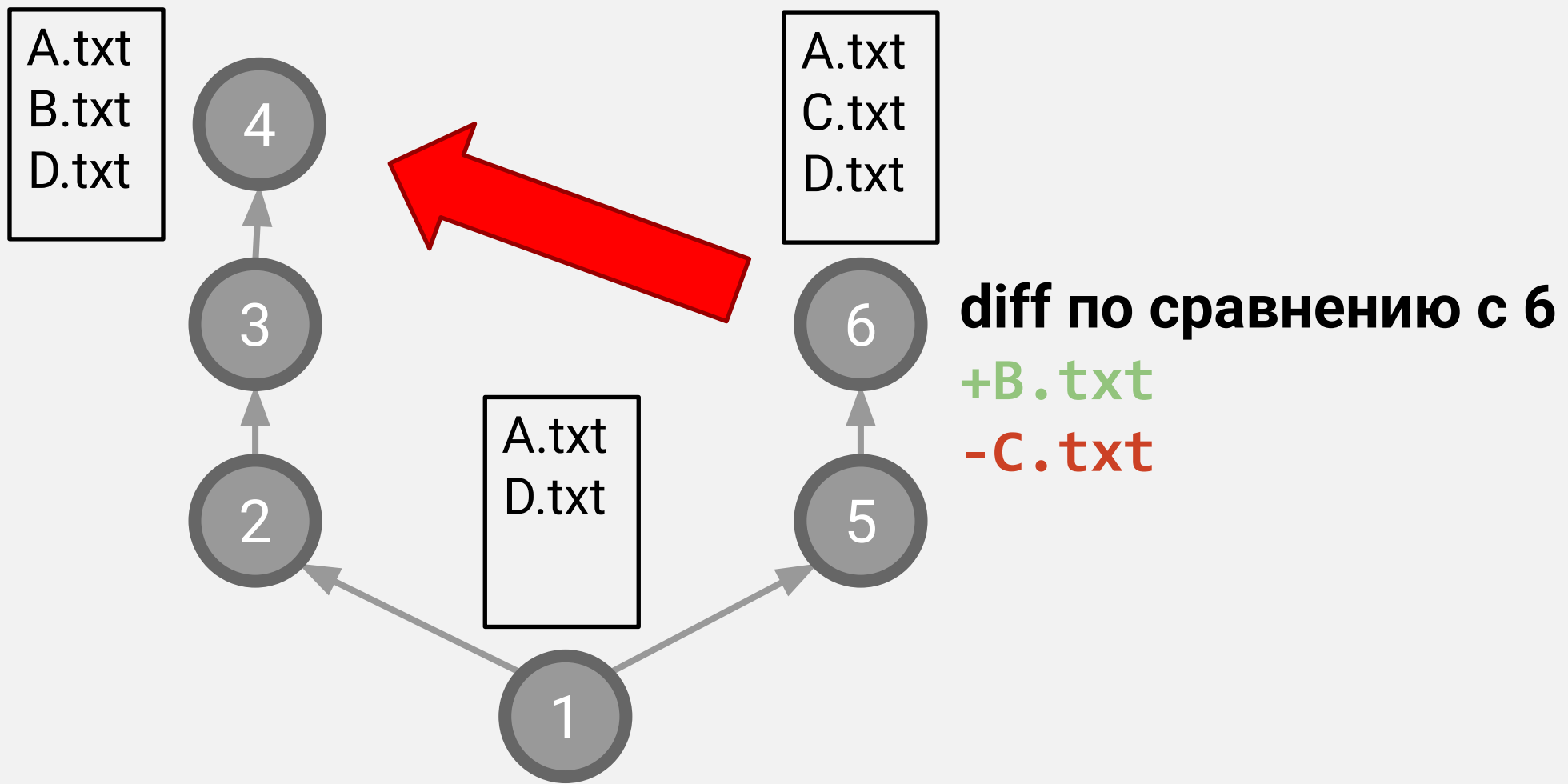
Diff можно получить между любыми коммитами



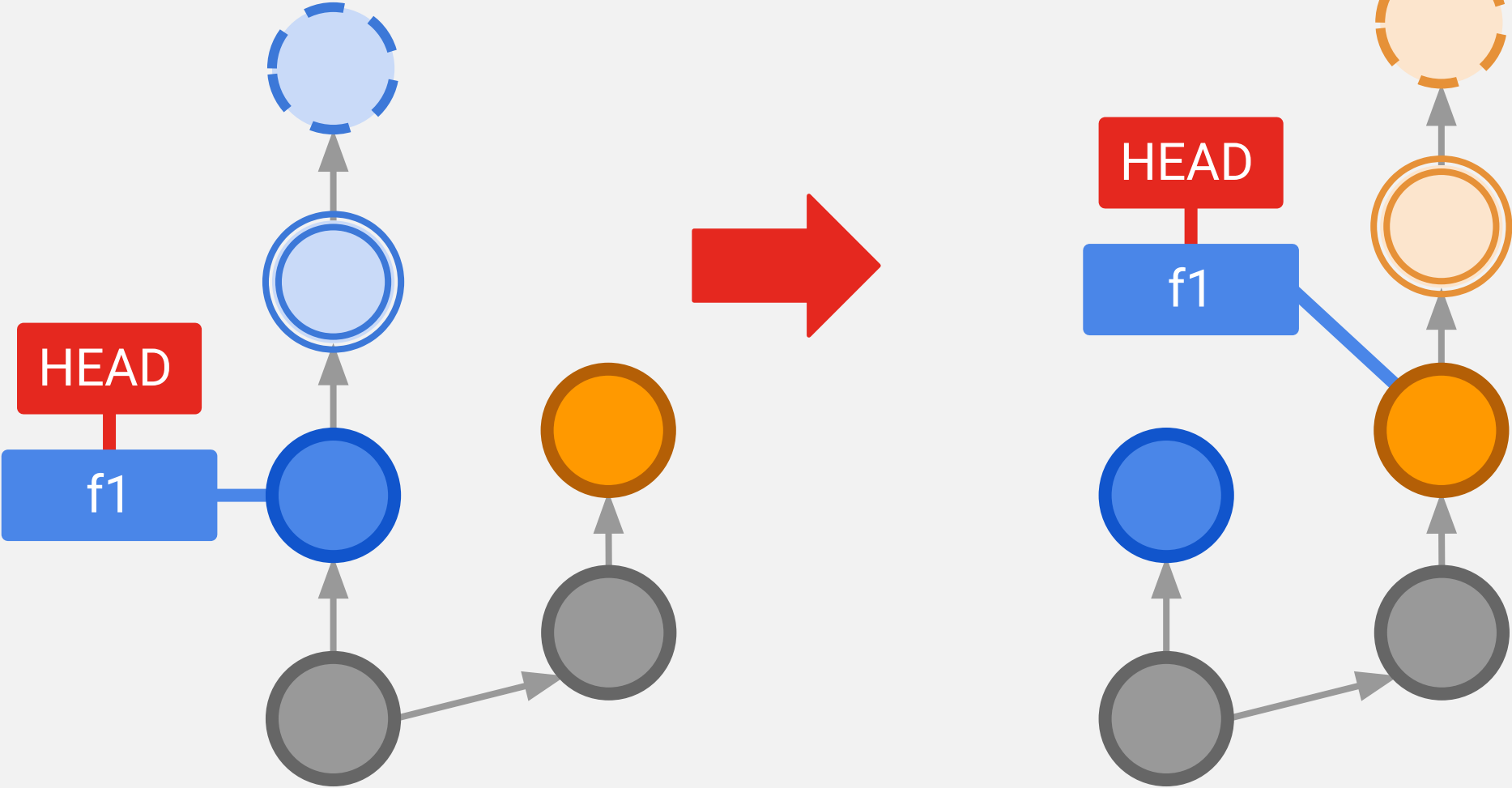
Что изменится при переходе от 4 к 6?



Что изменится при переходе от 6 к 4?



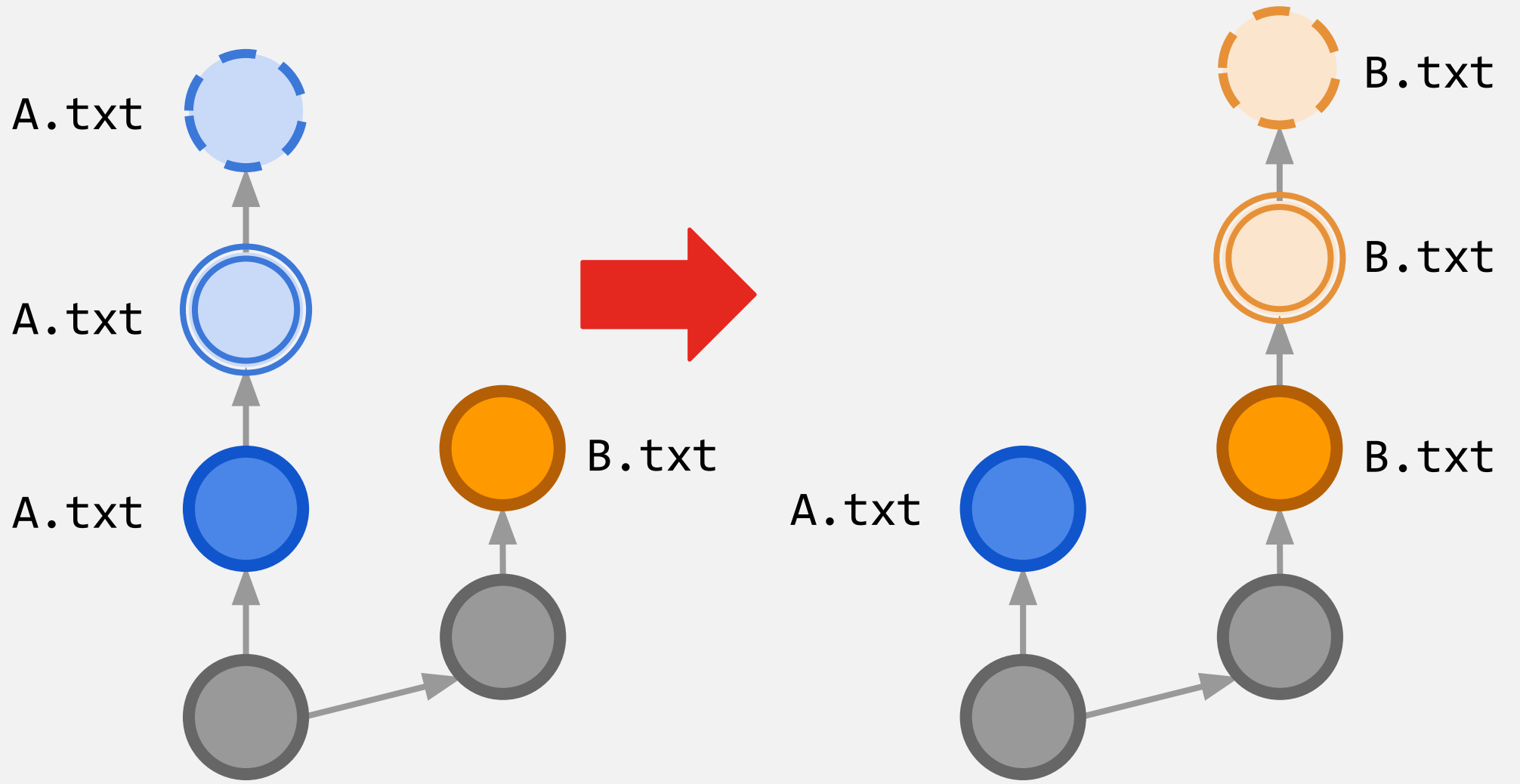
reset --hard



reset --hard

- **Переносит ветку** вслед за HEAD
- Выставляет индекс и директорию согласно коммиту, **устраняет разницу**

reset --hard





Working directory changes ▾

There are no unstaged changes



Unstage

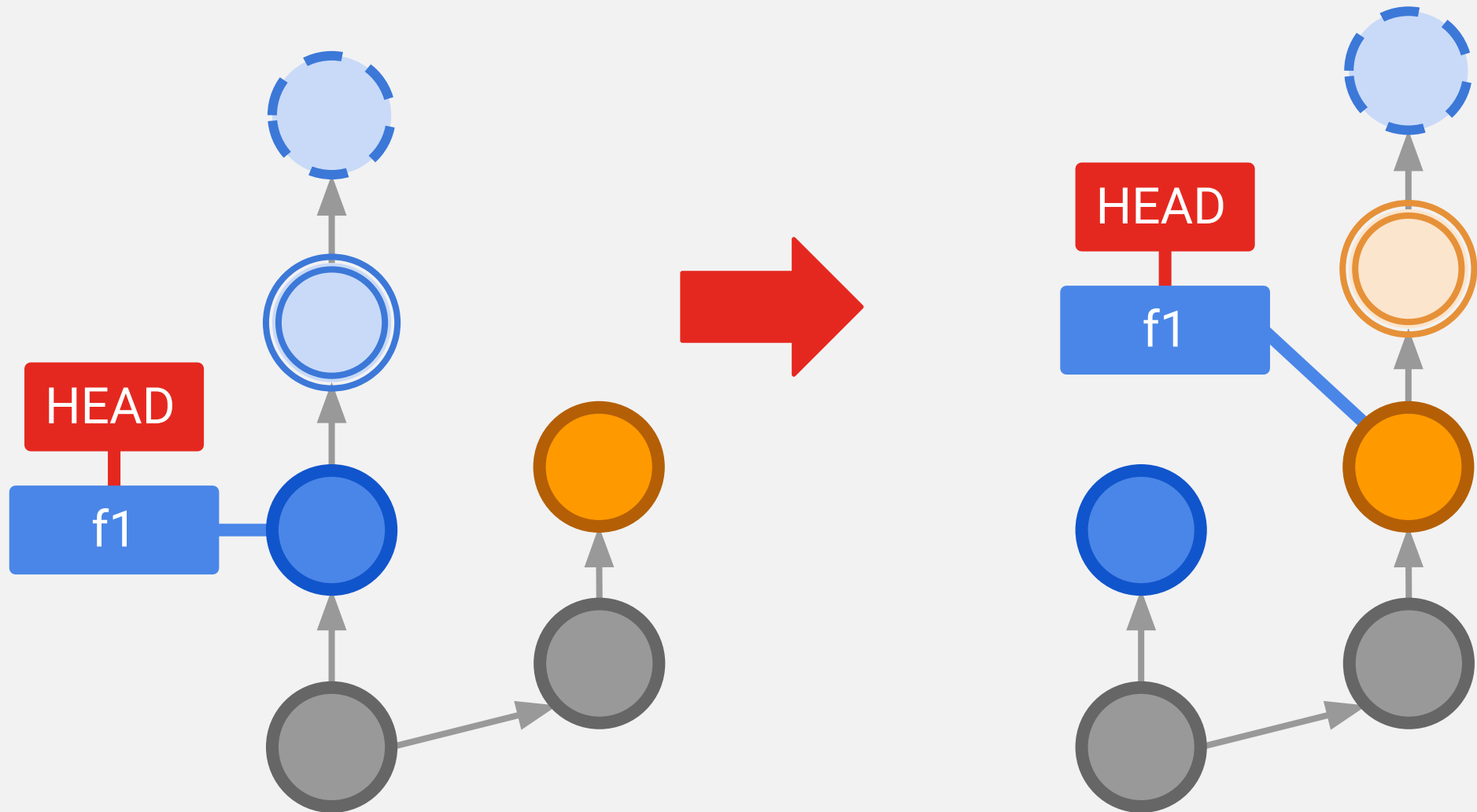


Stage



There are no staged changes

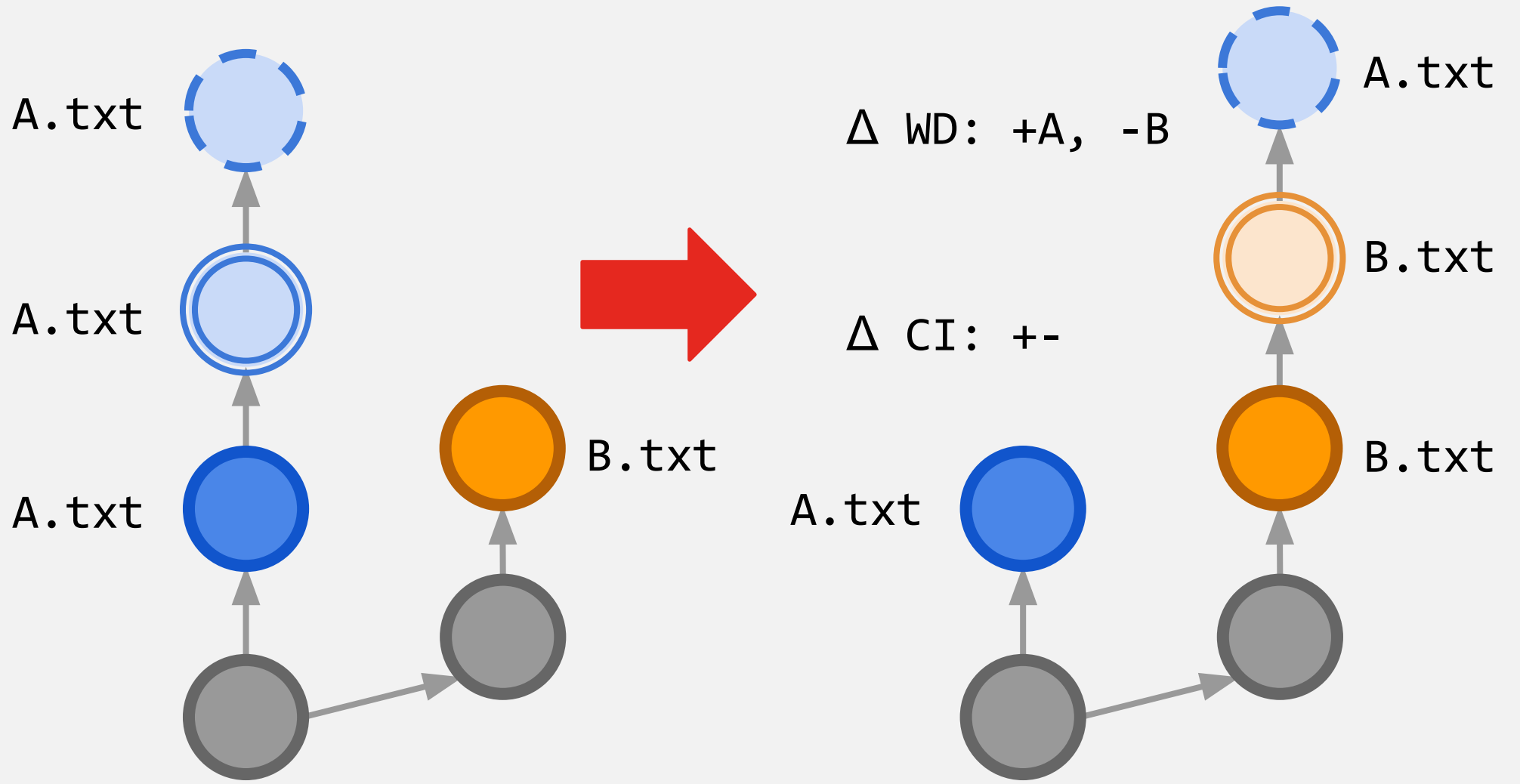
reset --mixed

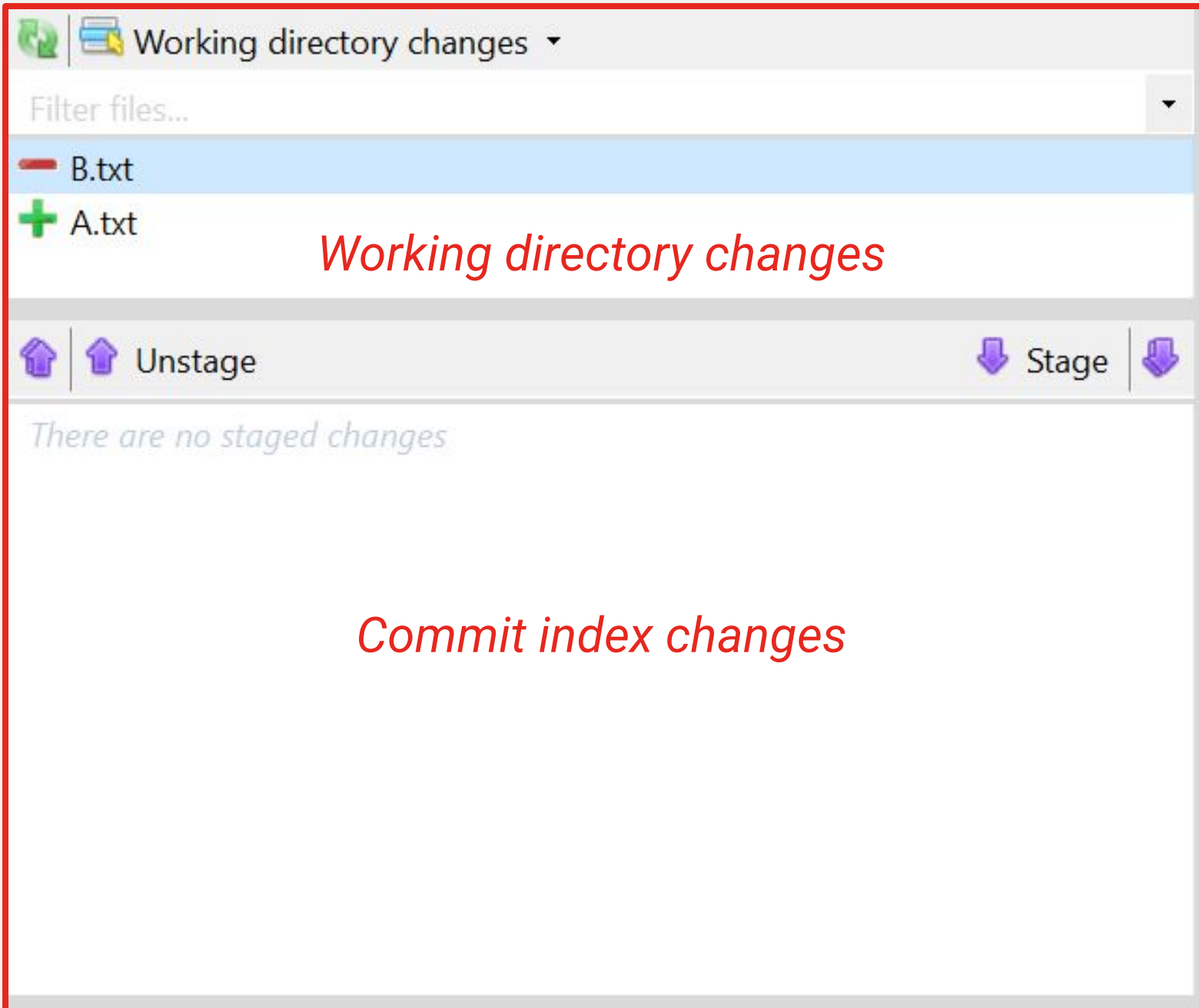


reset --mixed

- **Переносит ветку** вслед за HEAD
- **Задаёт индекс** согласно коммиту
- **Оставляет разницу** между исходным и новым состоянием **в директории**

reset --mixed

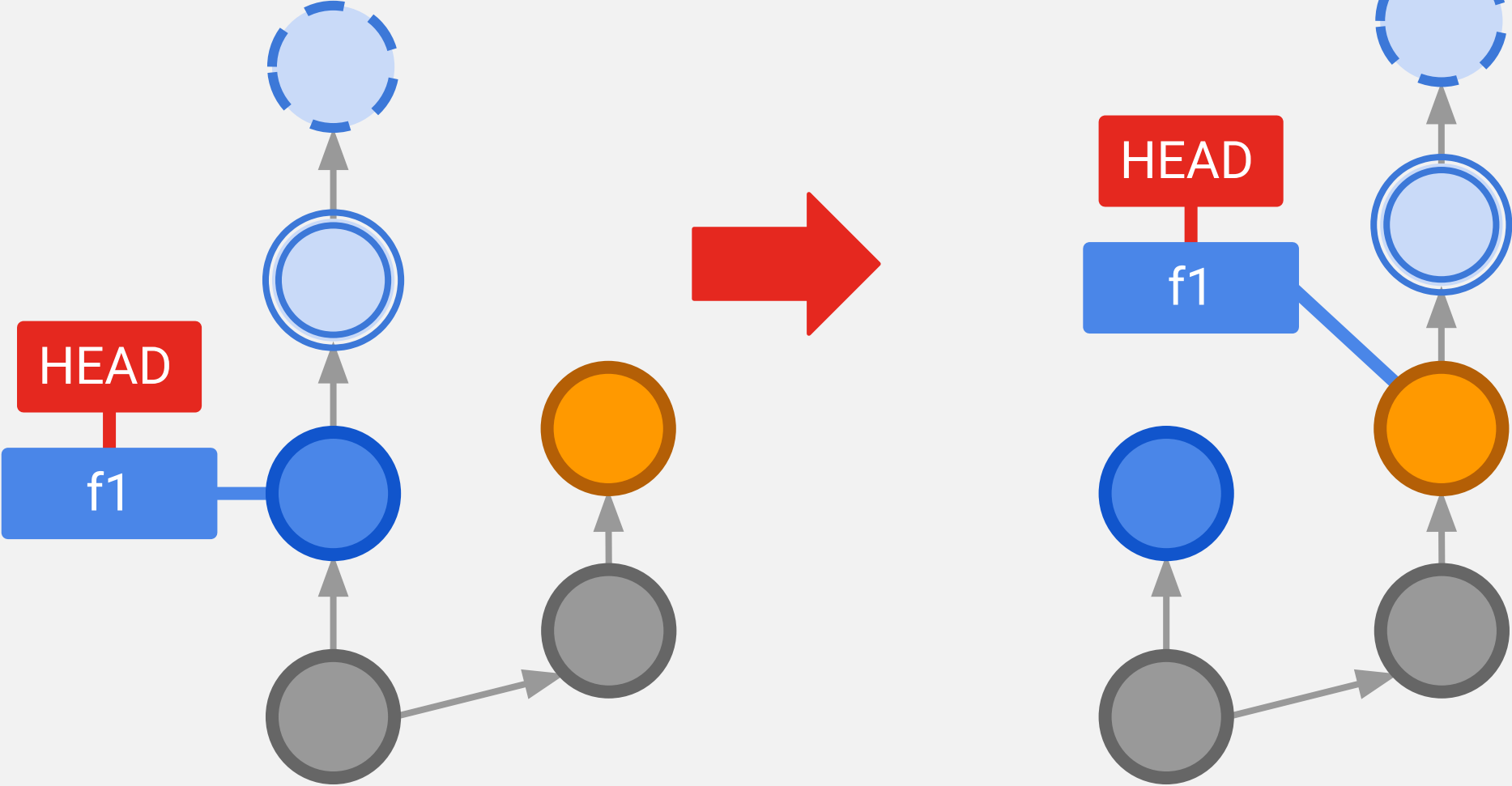




Working directory changes

Commit index changes

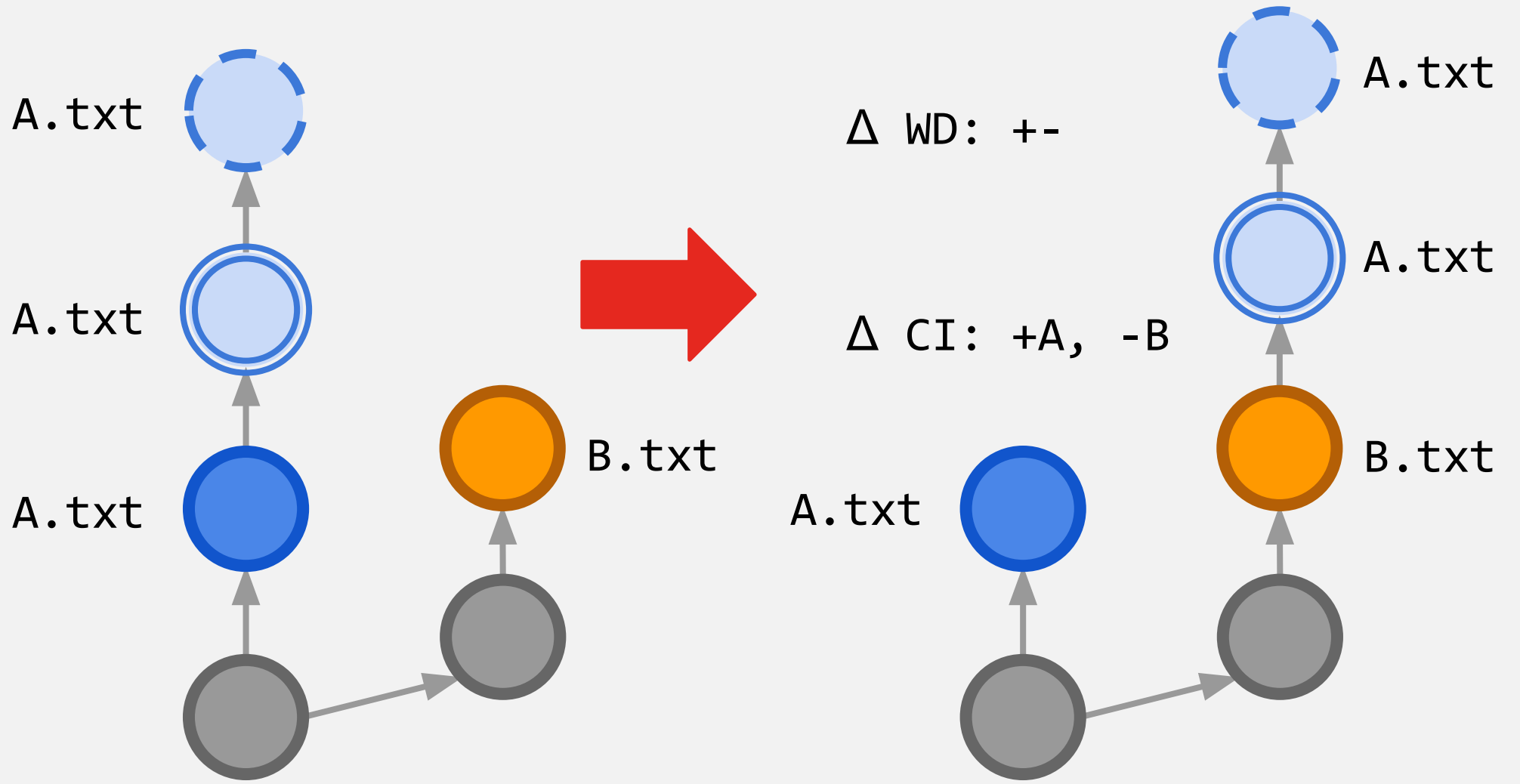
reset --soft



reset --soft

- **Переносит ветку** вслед за HEAD
- Не задает ни индекс, ни директорию согласно коммиту
- **Оставляет разницу** между исходным и новым состоянием **в индексе** и директории



reset --soft




  Working directory changes ▾


There are no unstaged changes

Working directory changes

  Unstage

 Stage 

 A.txt

 B.txt

Commit index changes

Удобный алиас

```
git config --global alias.undo "reset --soft HEAD^"
```



Stash

Тайник для временного **сохранения изменений**

После сохраненные изменения можно восстановить
в том же или в другом месте

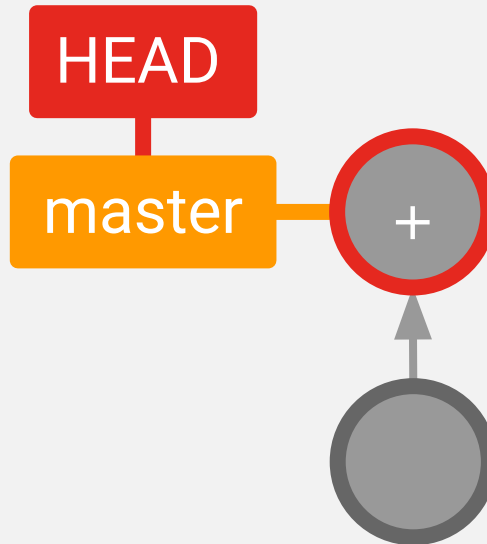
Работает по принципу стека: push, pop

Можно применять сохраненные изменения
неоднократно



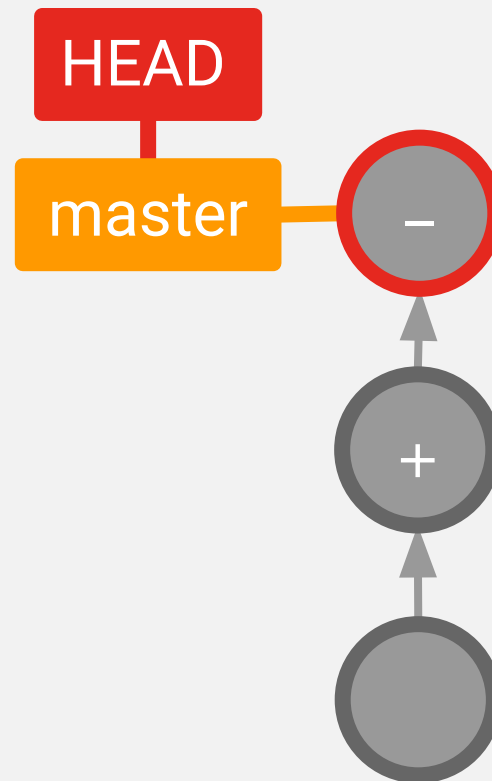
Revert

Неудачные изменения одного коммита можно отменить противоположными изменениями в следующем коммите



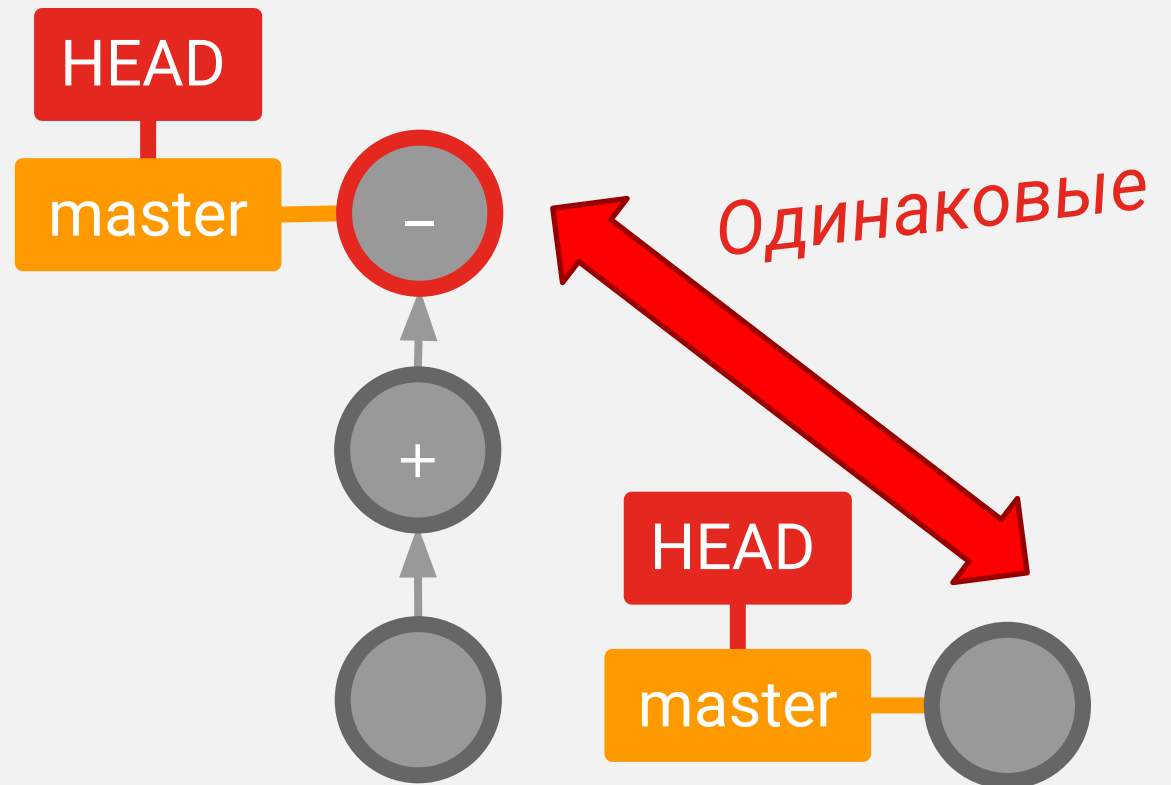
Revert

Неудачные изменения одного коммита можно отменить противоположными изменениями в следующем коммите



Revert

Неудачные изменения одного коммита можно отменить противоположными изменениями в следующем коммите



Revert полезен,
когда надо отменить изменение,
а перестраивать историю нельзя

Задание 16. Stash

Задание 17. Hard Reset

Задание 18. Soft Reset

Structure

Actions

Remote

S1. Все локально

A1. Трехсторонний merge
в три шага

R1. Доступен fetch коммитов
любого репозитория
в любой момент

S2. Хранятся
состояния директории,
постепенная сборка коммита

A2. rebase, cherry-pick и amend,
чтобы пересоздать историю

R2. Удаленное изменение
— это push

S3. Манипуляции
через ссылки,
нет ссылки — в мусор

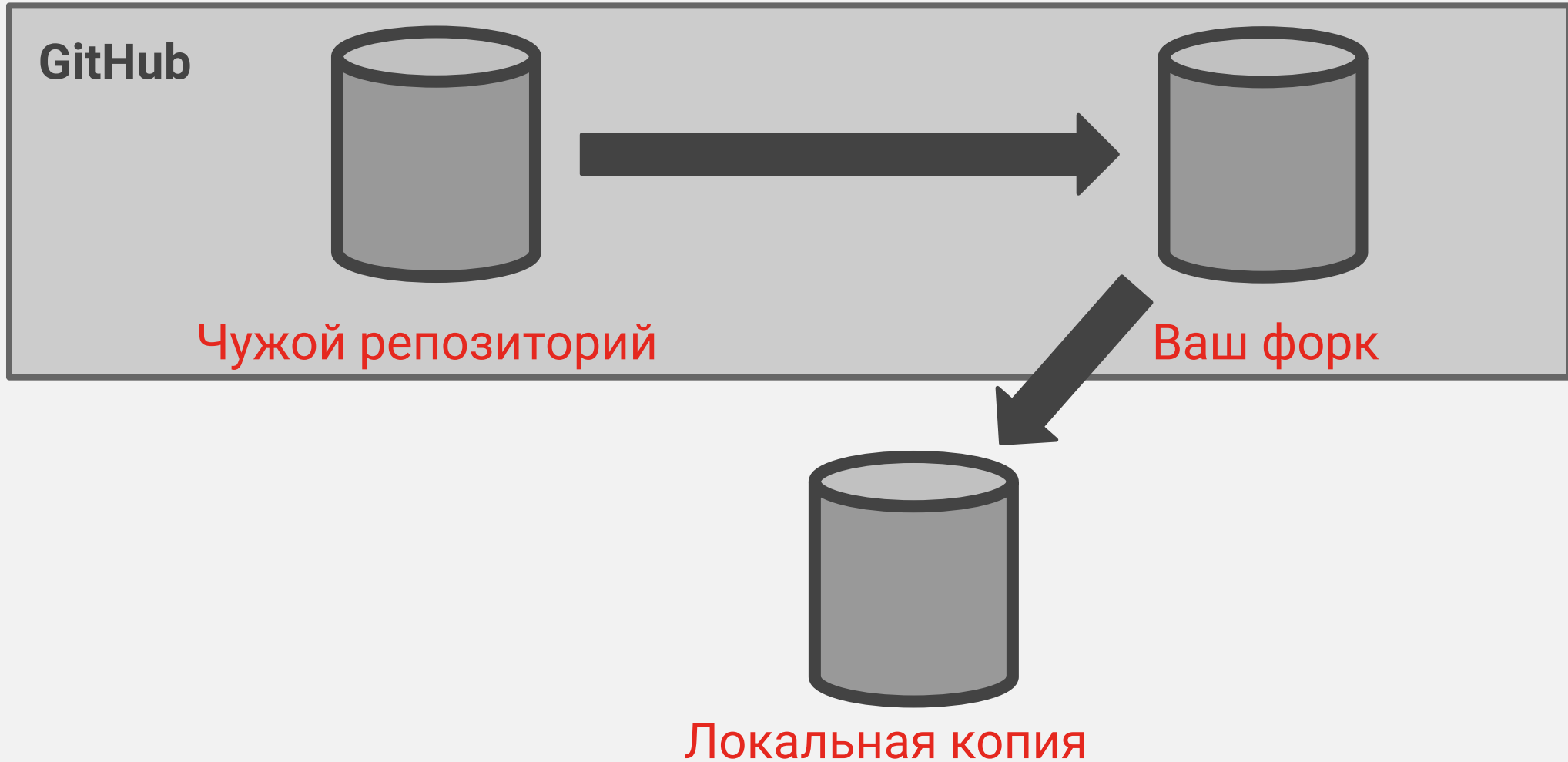
A3. stash, reset, revert
для управления изменениями

R3. Явное сопоставление
локальных веток
с upstream

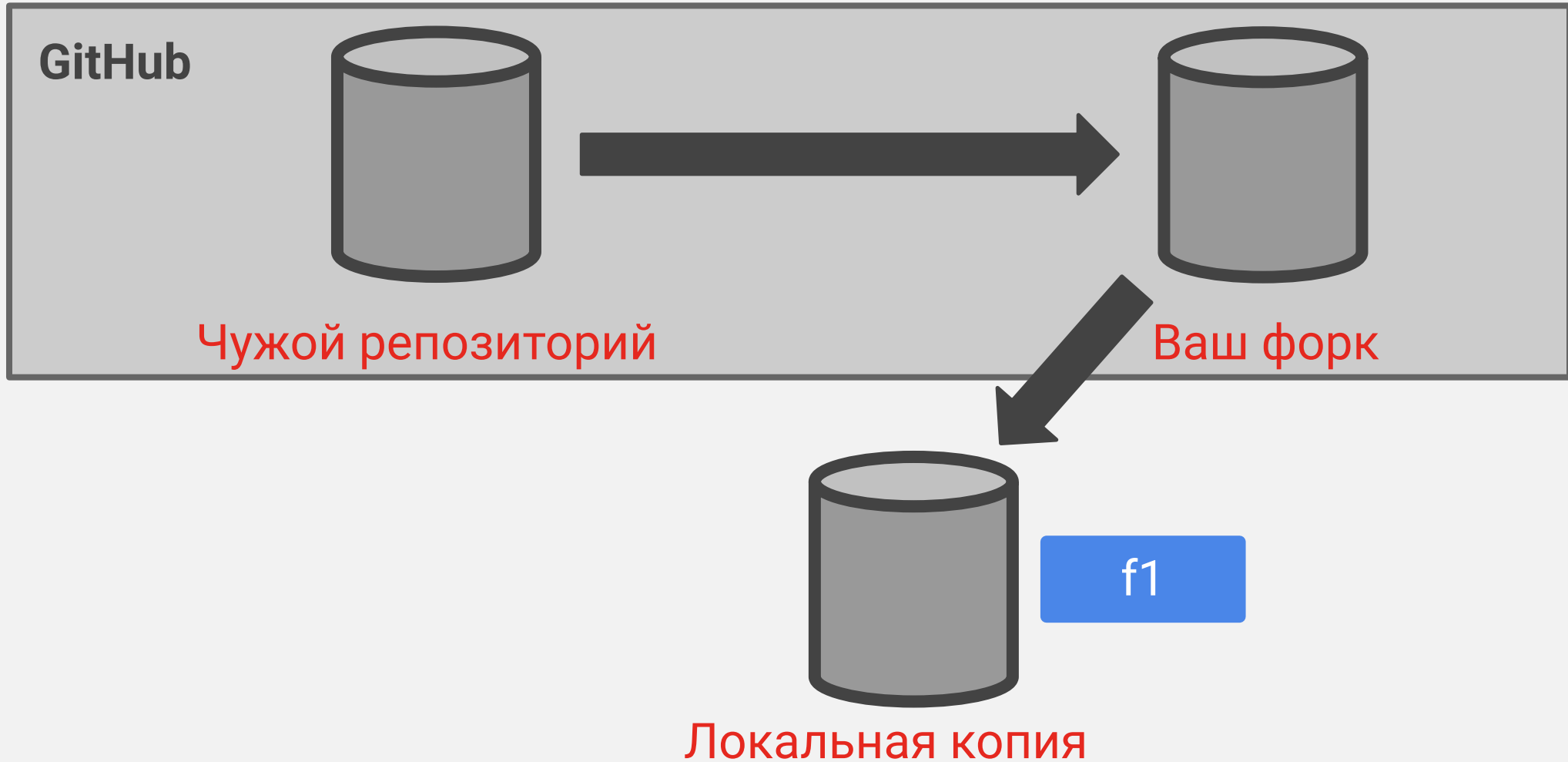
H1. Гибкое конфигурирование и качественная документация

Разное

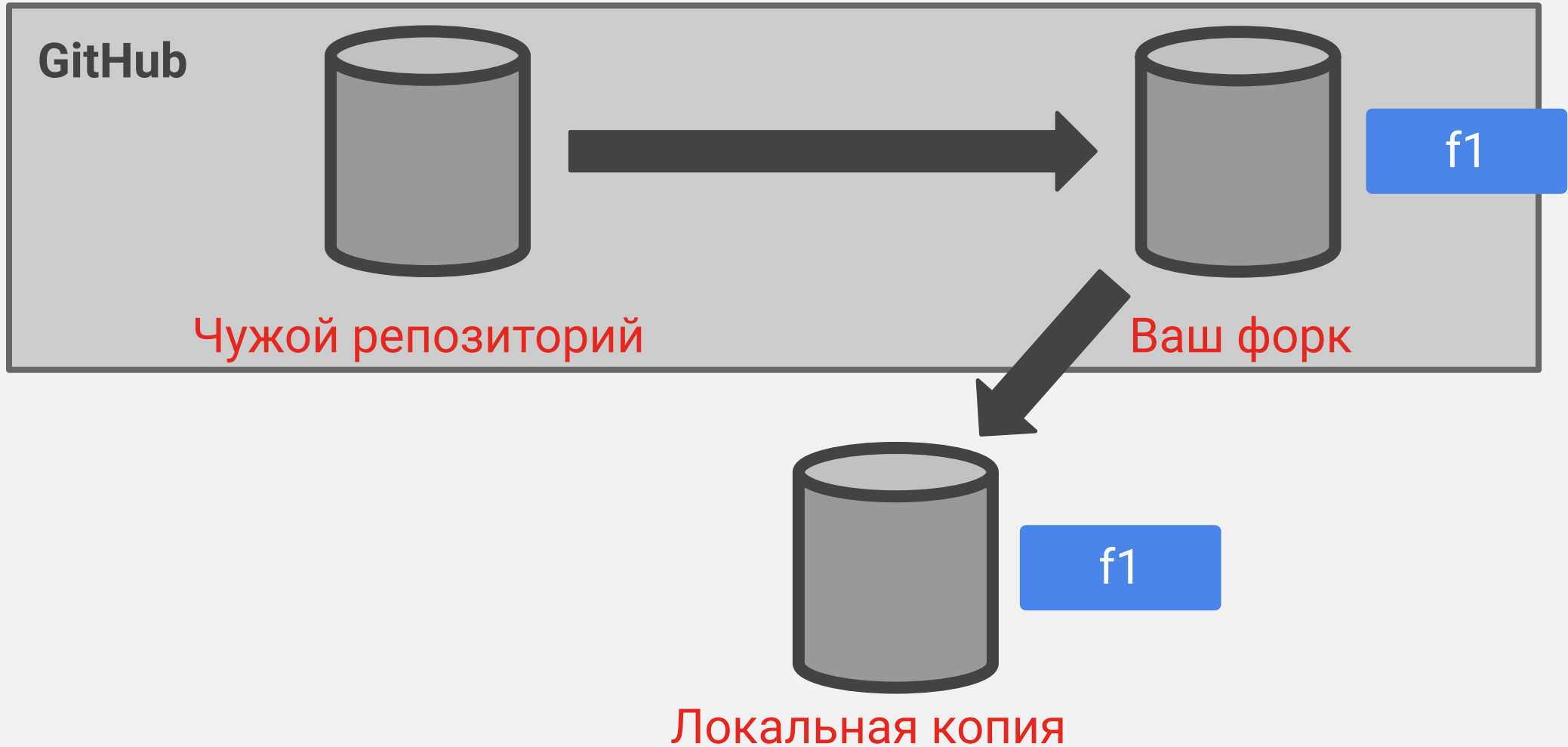
Что такое Pull Request



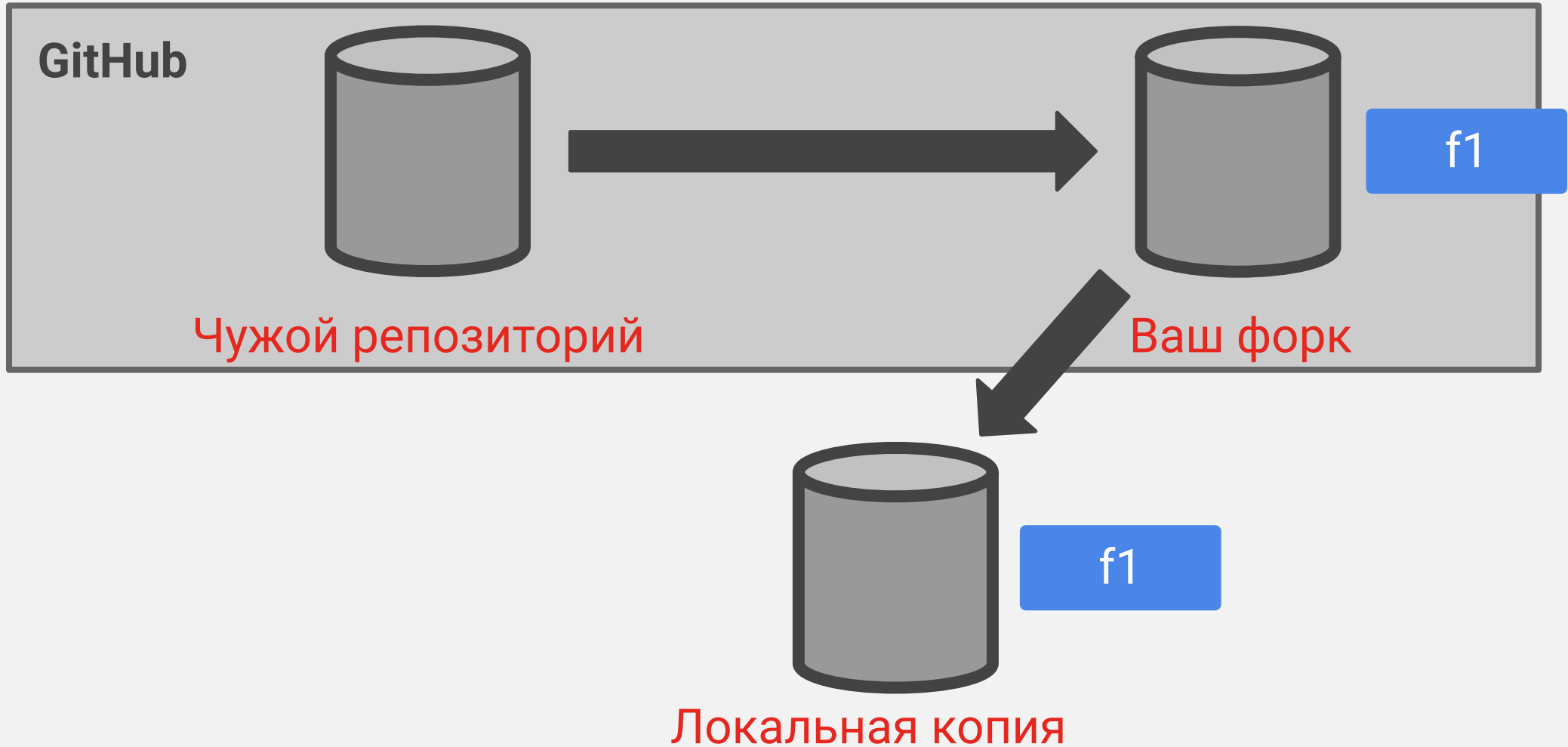
Изменения в локальном репозитории



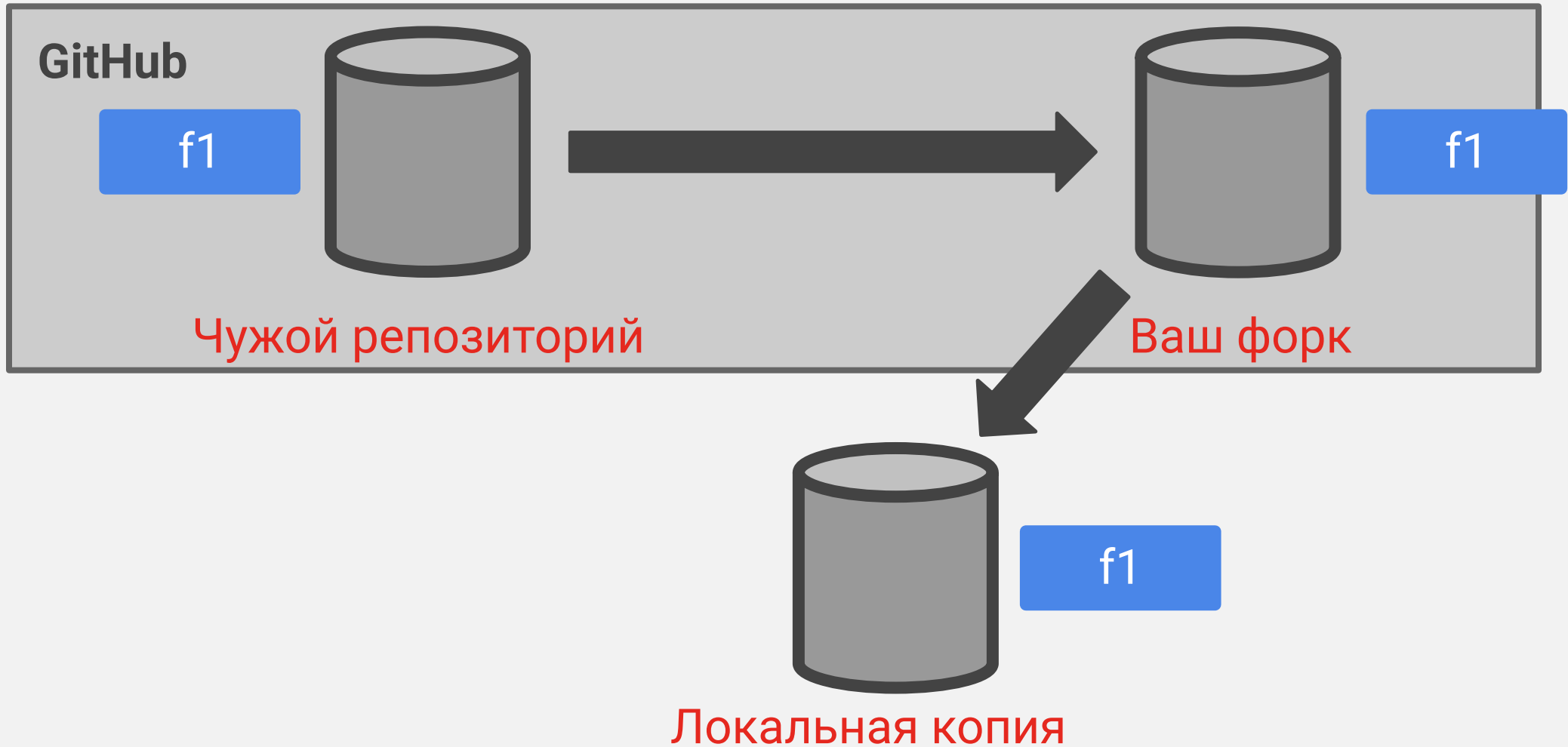
Отправка изменений в свой удаленный



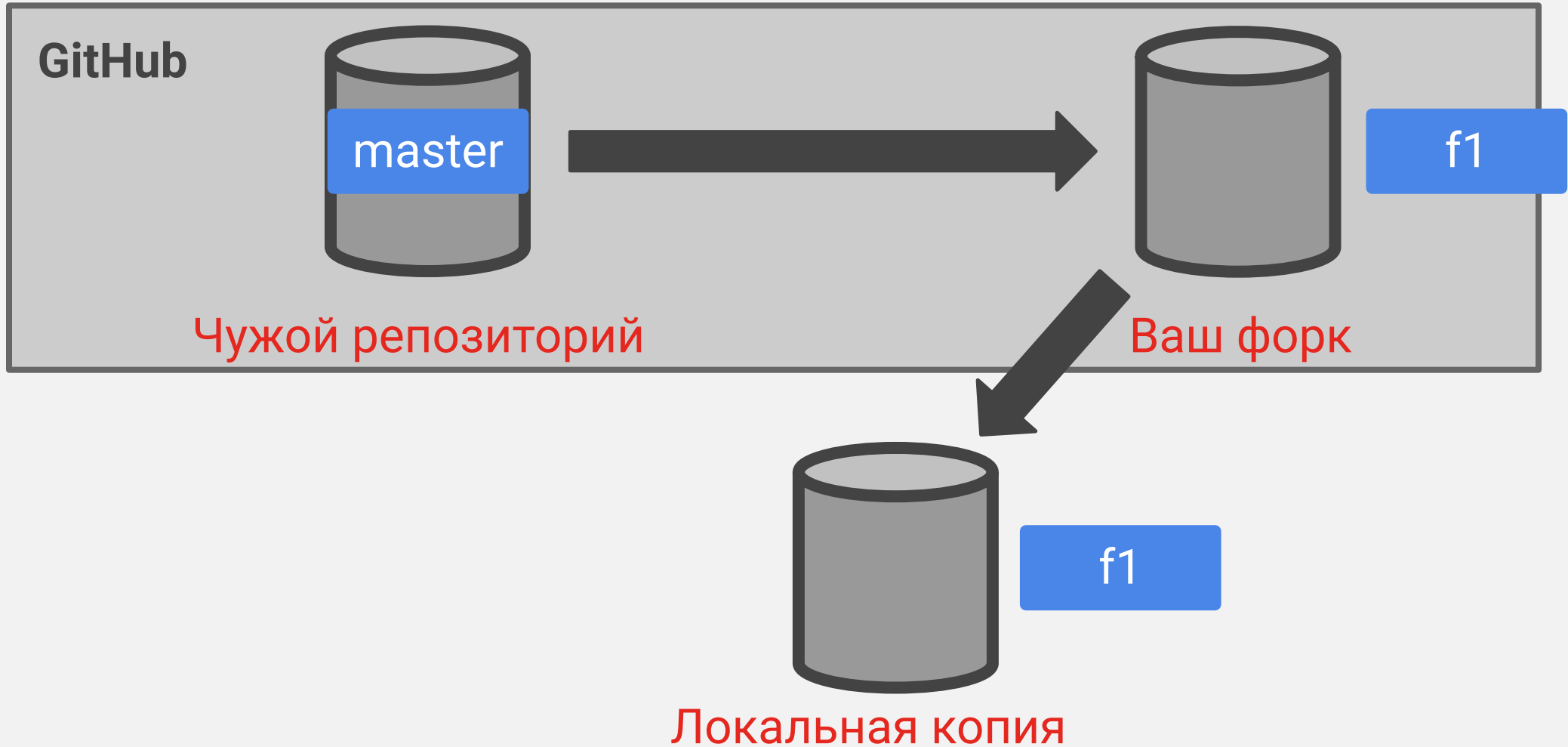
PR – запрос на fetch + merge



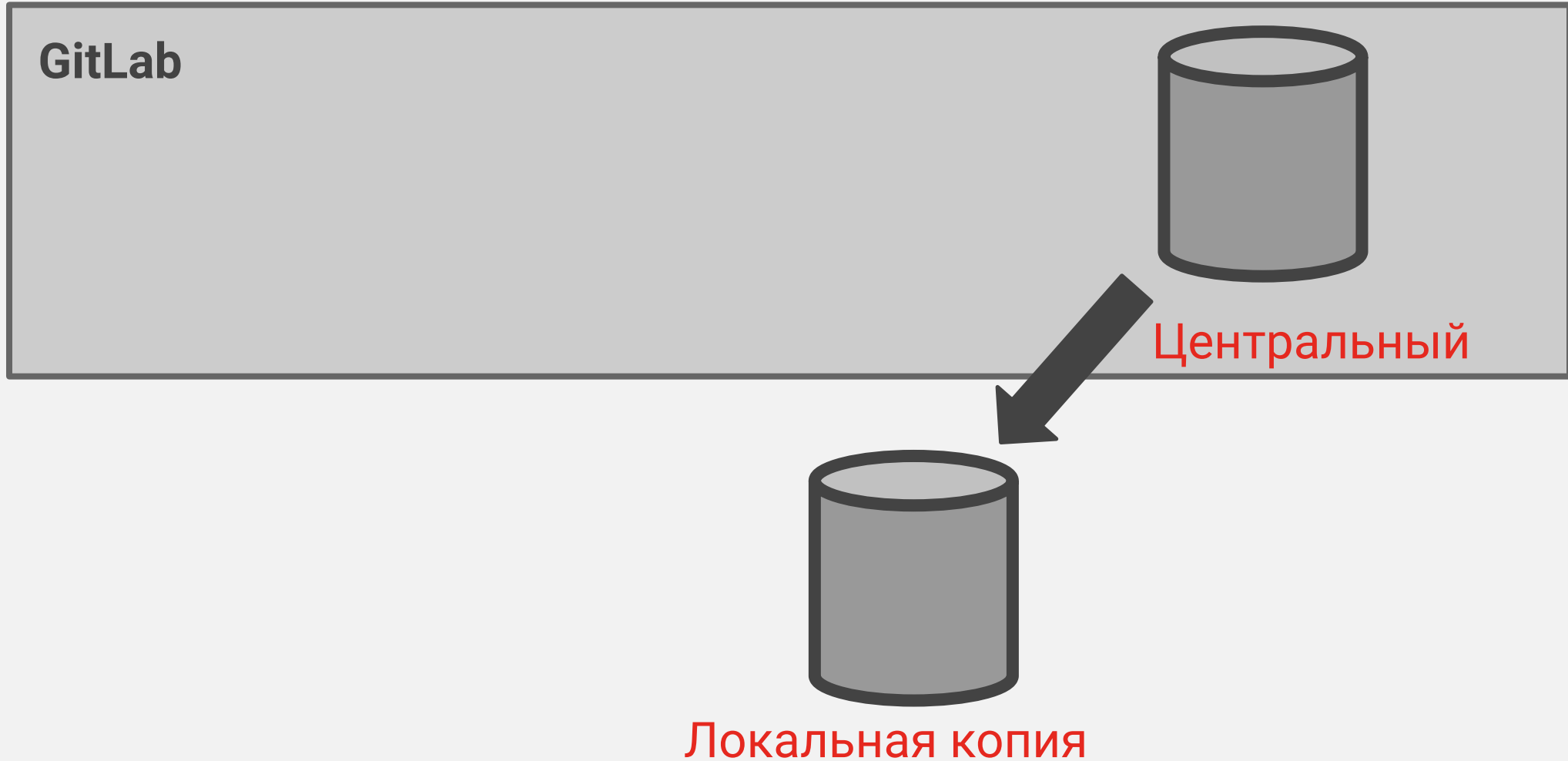
Если PR принят, то fetch



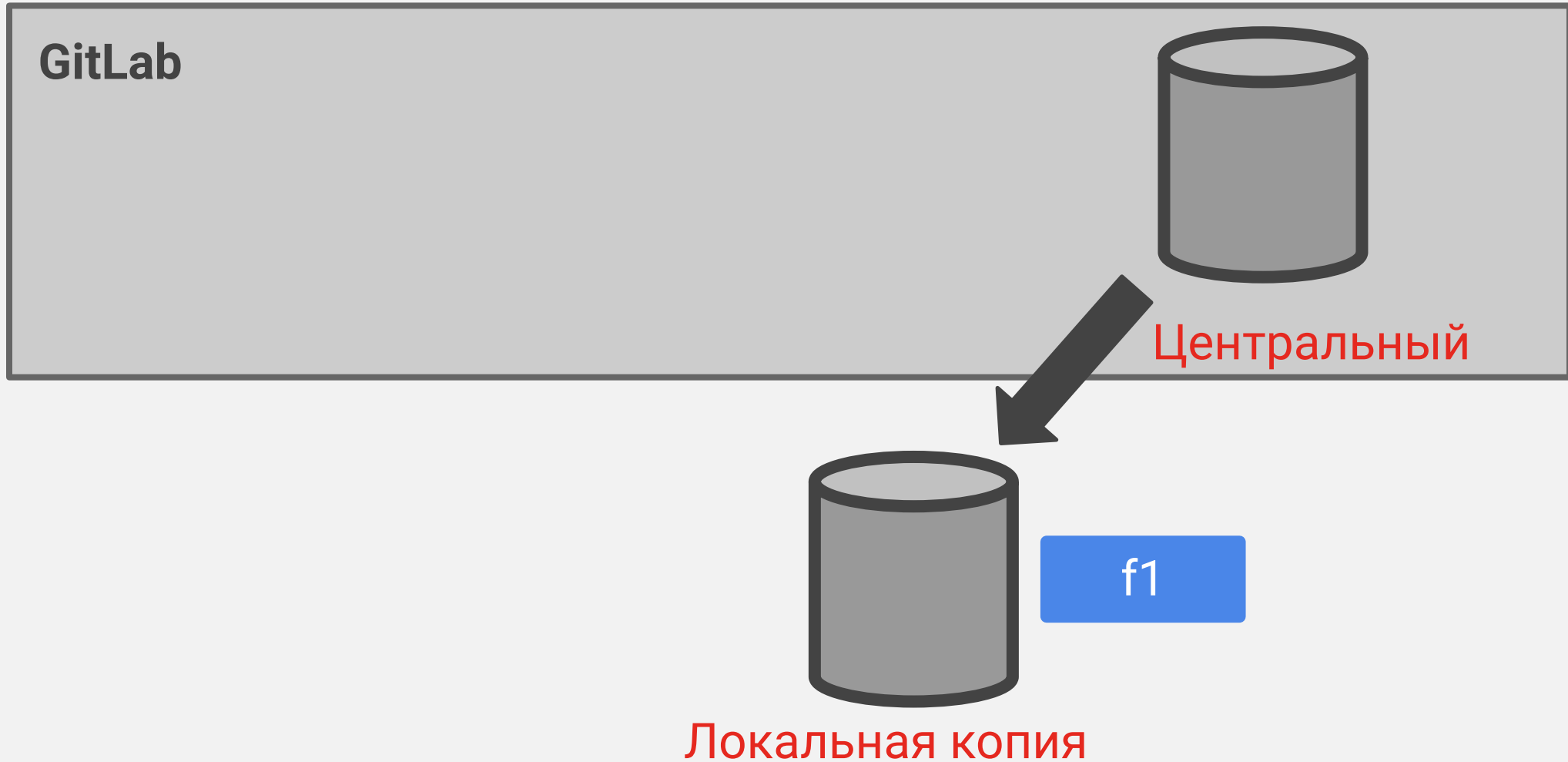
Если PR принят, то merge



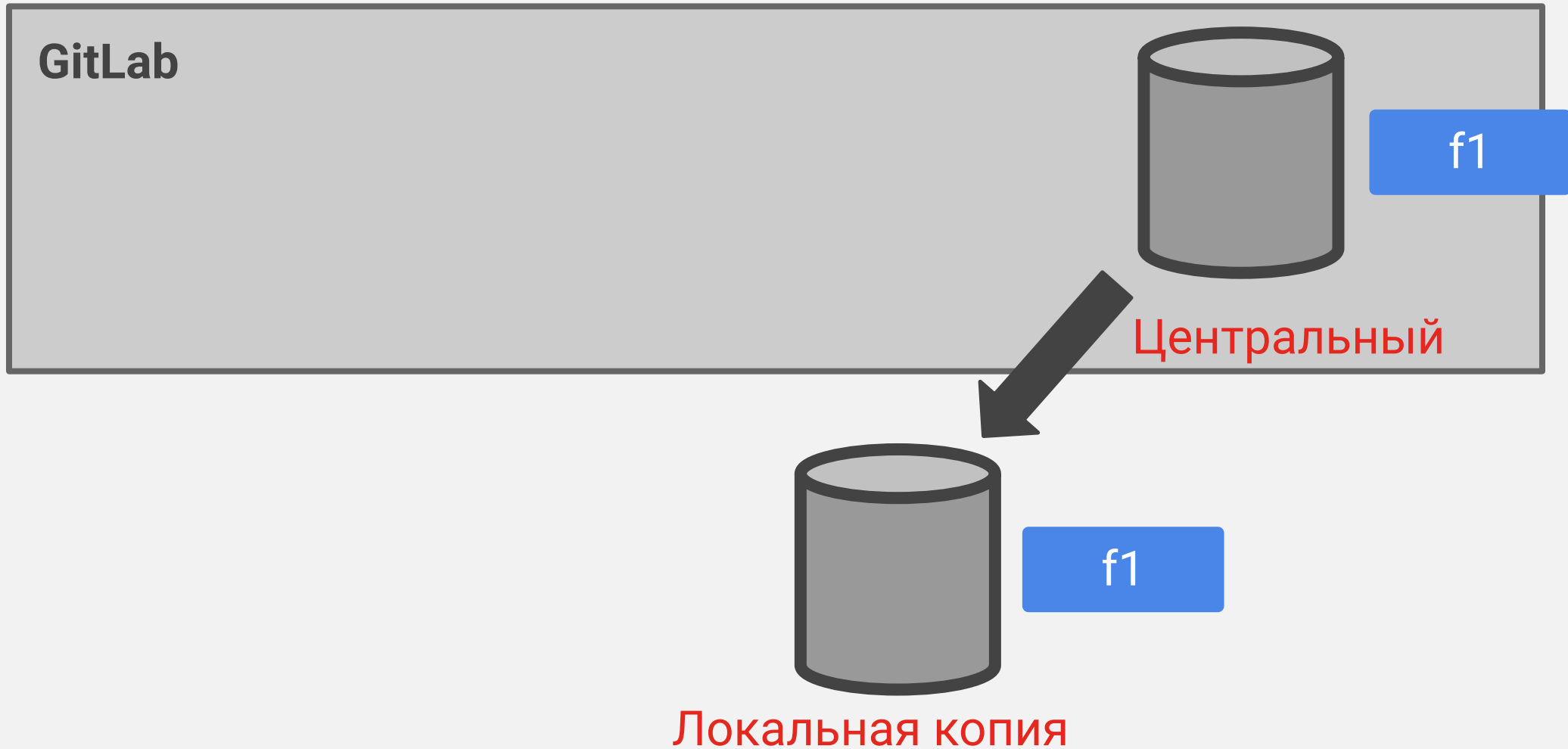
Что такое Merge Request



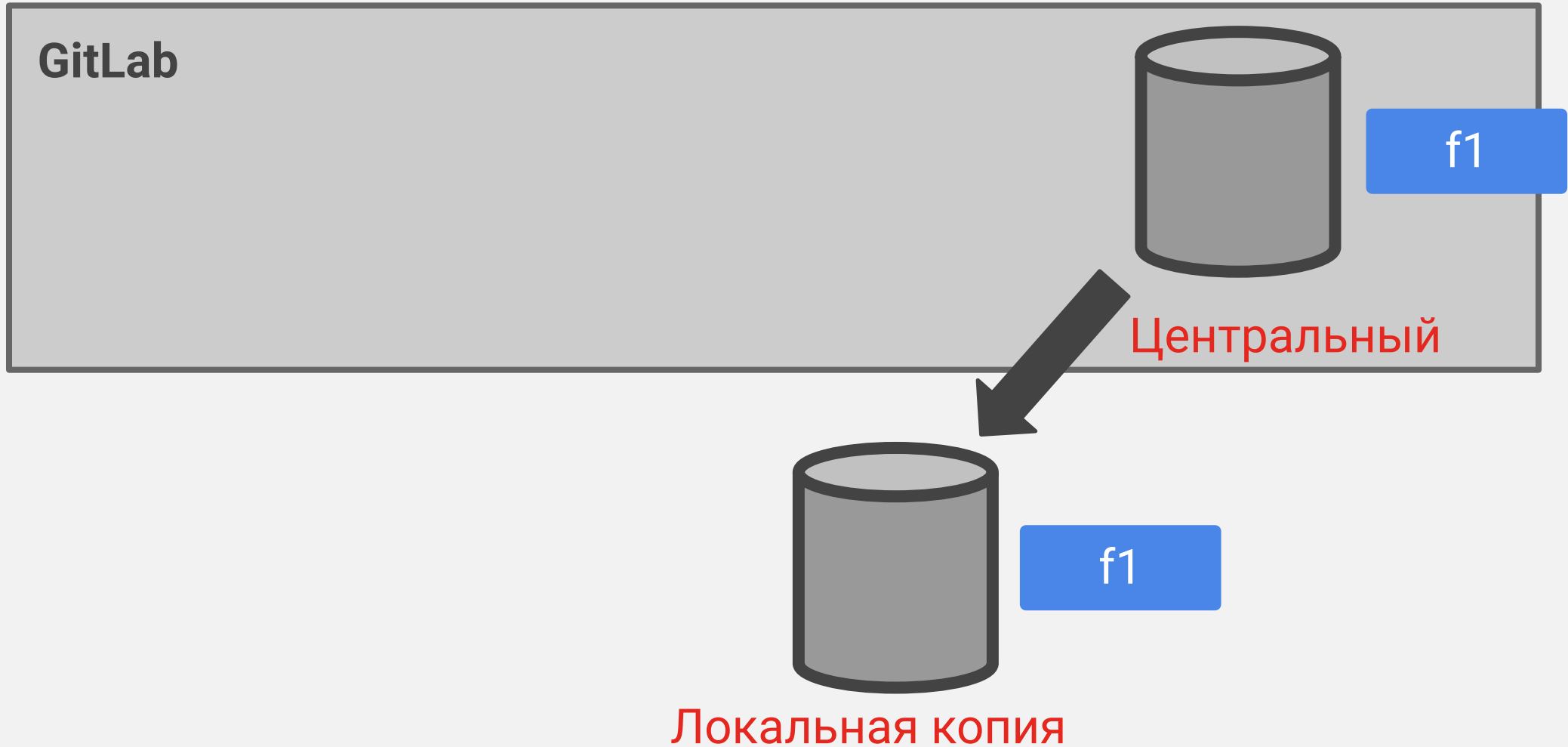
Изменения в локальном репозитории



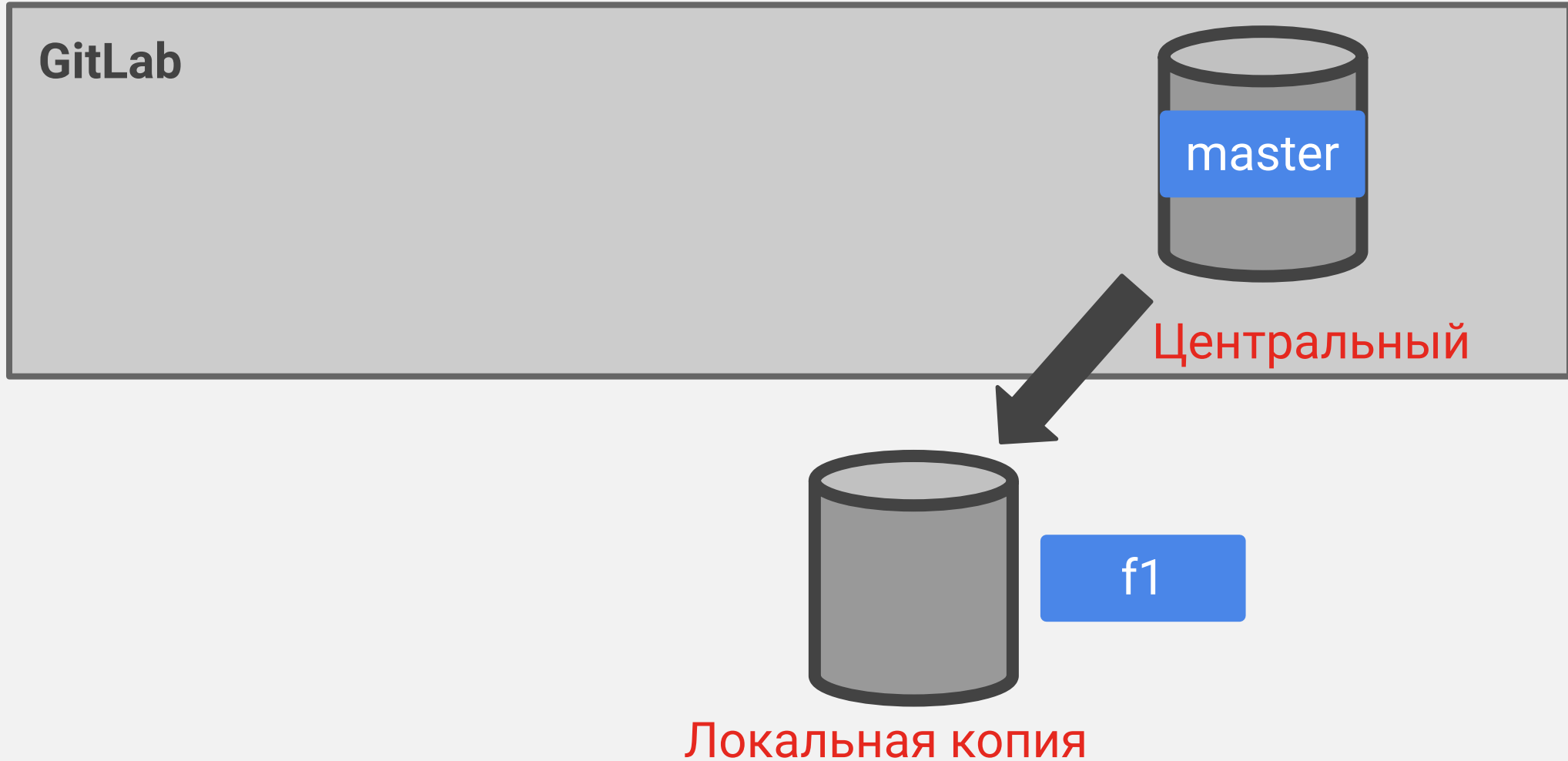
Отправка изменений в удаленный



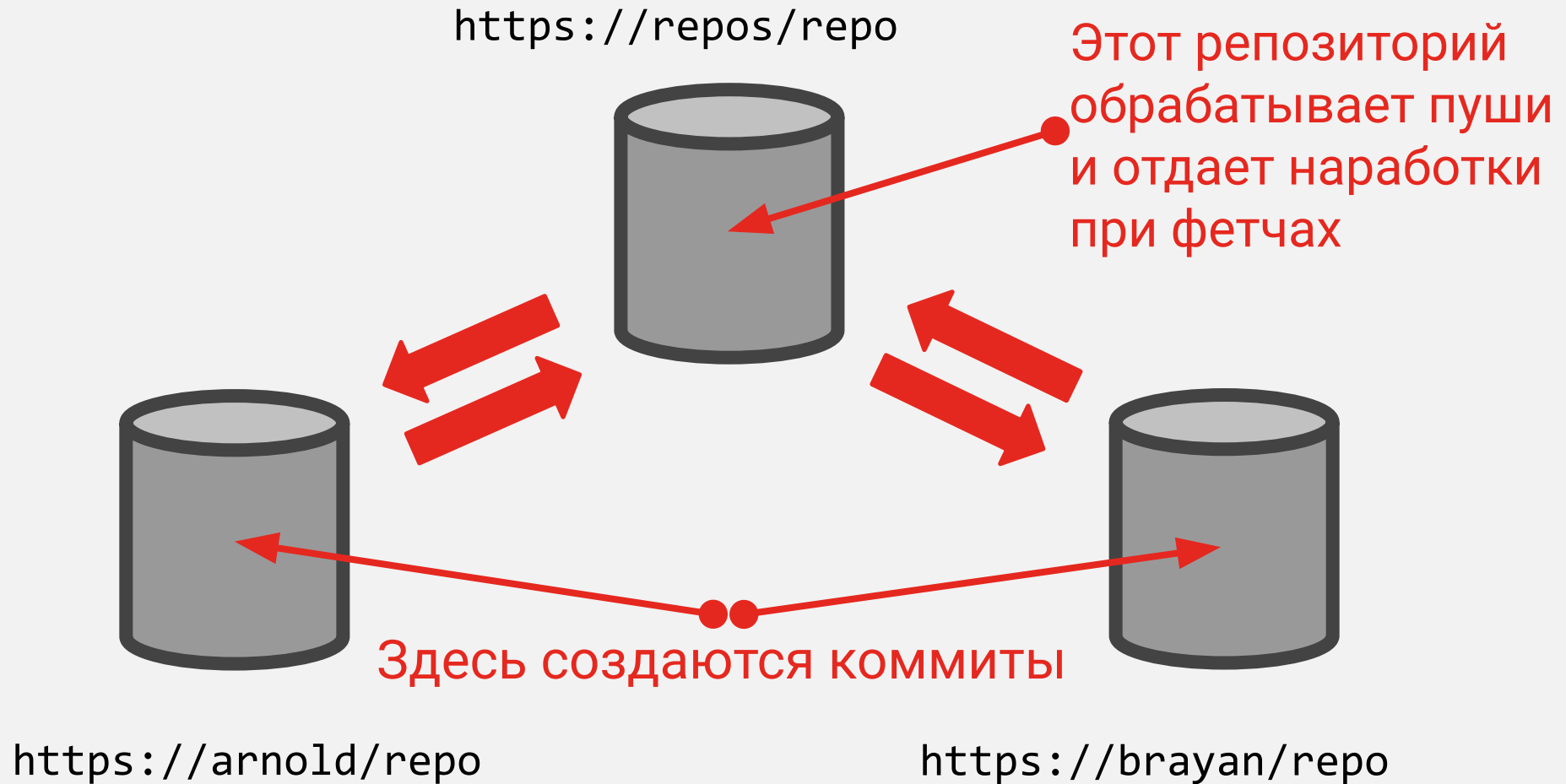
MR – запрос на merge



Если MR принят, то merge



Картина для централизованного репозитория



Алгоритм для разработчика

1. Получить последние изменения
2. Создать локальную ветку
3. Разработать в ней фичу
4. Получить последние изменения
5. Влить мастер в свою ветку (опционально)
6. Влить свою ветку в мастер
7. Запустить (если кто-то успел запустить раньше, то повторить 4-7)

Блиц



Надо подключить Git к папке,
в который ты делаешь тестовое задание

Надо загрузить локальный репозиторий
с тестовым заданием на GitHub для проверки

Надо подключиться к разработке сервиса
в отдельном репозитории

Надо начать разработку новой фичи

Сделаны некоторые изменения,
надо их сохранить в репозитории

Ты забыл добавить кое-что
в последний коммит, а надо

Надо сделать так, чтобы новая фича была доступна для тестирования

Тестировщик попросил,
чтобы последние исправления из master
тоже были в ветке с фичей

При влитии исправлений
из master возникли конфликты

Пора сделать релиз фичи,
у вас фичи релизятся из master

Надо помочь другому разработчику
сделать фичу

За день ты сделал некоторые изменения
и решил ими поделиться

Оказалось, что твой напарник
тоже сделал изменения

Надо их получить, а затем
опубликовать свои изменения

На следующий день ты продолжил
работать над фичей,
но тебя срочно попросили починить
опечатку прямо в master

ЧТО ТВОРЯТ =/

Ты поправил опечатку, закоммитил в master
и получил изменения в origin/master

Ты решил переключиться на origin/master,
Git Extension предложил сделать reset master
на origin/master, а ты машинально согласился...

Ты вернулся к фиче,
которую делал с другим разработчиком

У твоего напарника сегодня выходной

Ты доделал изменения,
о которых вы договаривались
Их надо сохранить перед тем,
как переходить к следующей задаче

Ты начал делать новую задачу
и сделал первый коммит
Но понял, что забыл завести ветку,
а коммит ушел в master

Разработка не клеилась...
Ты сделал десяток коммитов, перед тем
как понял, как решать задачу
Чтобы избежать позора, ты решил заменить
эти коммиты на один

Чтобы список веток был коротким,
ты решил удалить влитые локальные ветки

Бурная неделя закончилась
Пора отдохнуть!

Обратная связь



Заполни форму обратной связи по ссылке

<http://bit.ly/kontur-courses-feedback>

или

по ярлыку *feedback* в корне репозитория