

Южно-Уральский
государственный
университет

Национальный
исследовательский
университет

КРОК

Git. Локальный и удаленный репозитории

КРОК

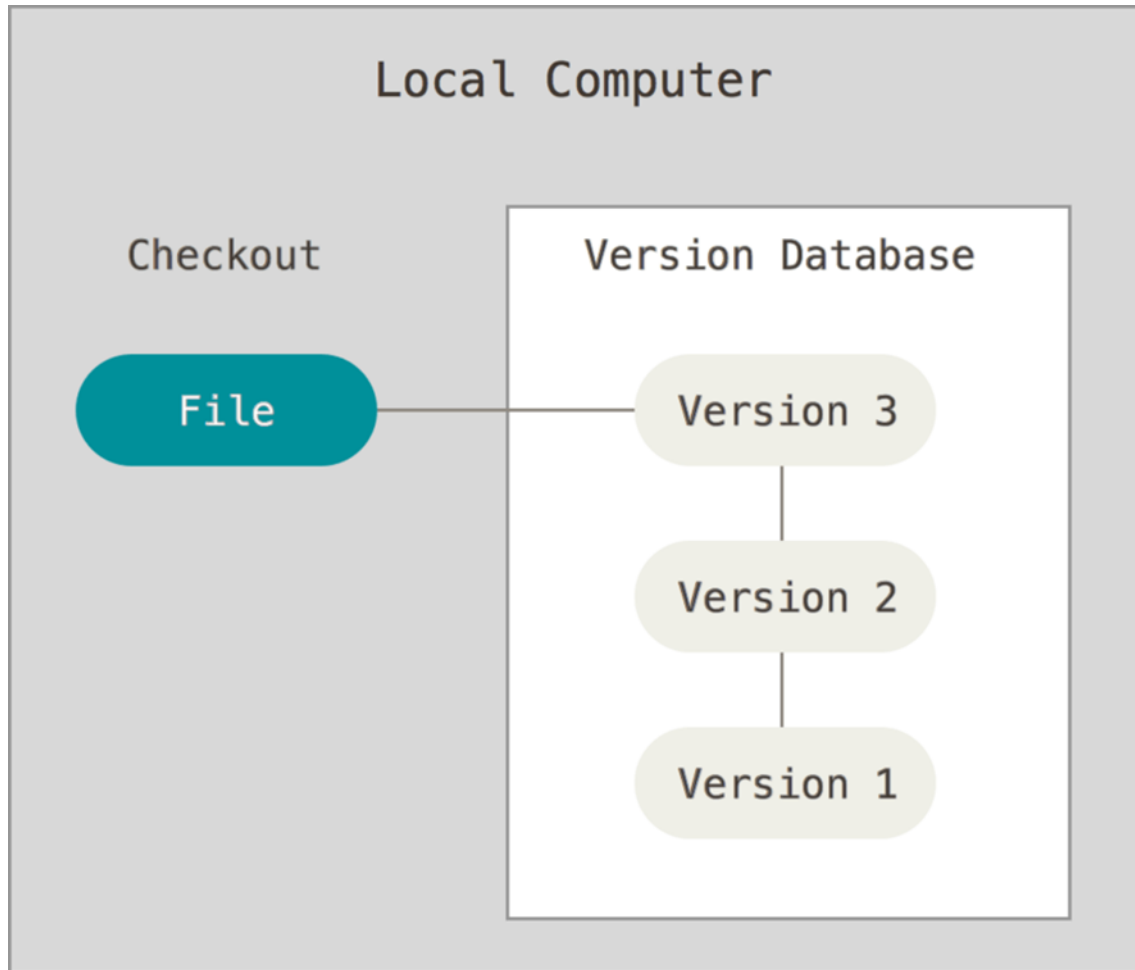
Челябинск, ул. Карла Маркса, д. 38

Смирнов Анатолий
Технический менеджер

Кузнецов Сергей
Старший инженер-разработчик

Фоменко Алексей
Младший инженер-разработчик

- **Система контроля версий** - это программное обеспечение, которое позволяет хранить несколько версий одного и того же документа, отслеживать изменения в документах, а так же, в случае необходимости, возвращаться к предыдущим версиям документов
- **Аббревиатуры:** СКВ, VCS (Version Control System)



Примеры локальных СКВ:

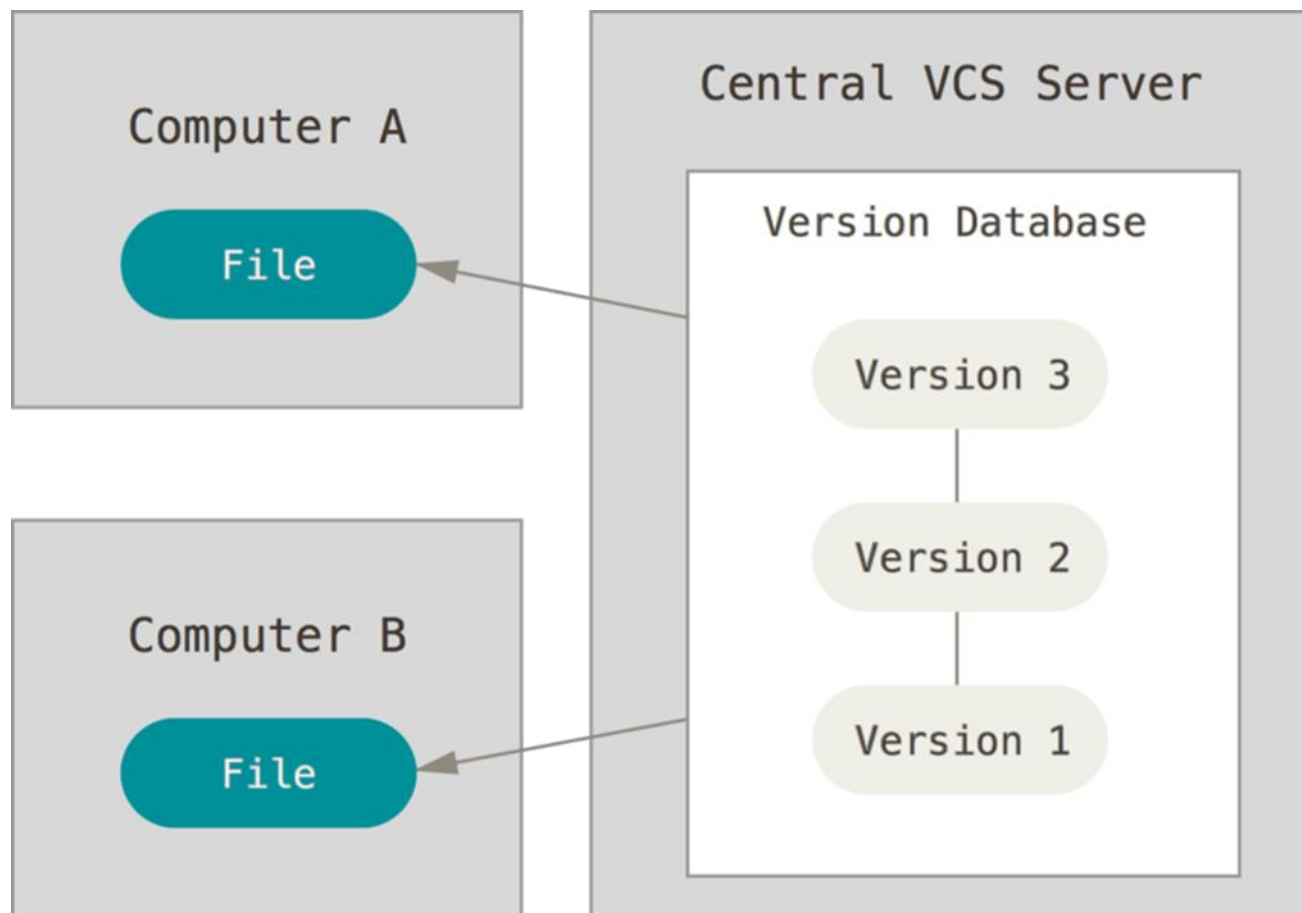
- Revision Control System (RCS), 1982

Достоинства:

- простота использования
- быстрая скорость извлечения определенной версии документа

Недостатки:

- отсутствует возможность совместного параллельного использования
- есть вероятность потери данных, в случае выхода из строя оборудования



Примеры централизованных СКВ:

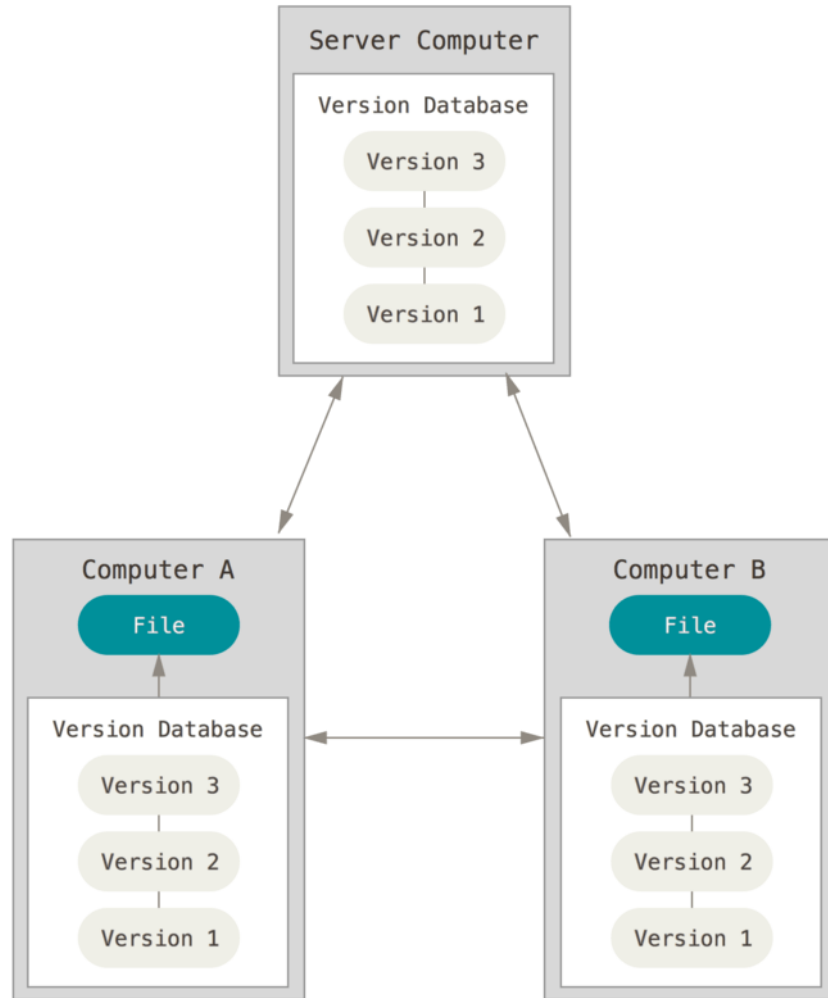
- Concurrent Versions System (CVS), 1990
- Perforce, 1995
- Subversion (SVN), 2000

Достоинства:

- возможность ведения командной работы

Недостатки:

- есть вероятность потери данных, в случае выхода из строя оборудования



Примеры распределенных СКВ:

- Git, 2005
- Mercurial (Hg), 2005

Достоинства:

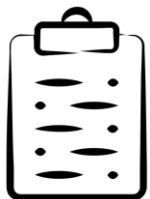
- отсутствие единой точки отказа
- возможность ведения командной работы
- более высокая скорость проведения части операций за счёт их локального выполнения по сравнению с централизованными СКВ

Недостатки:

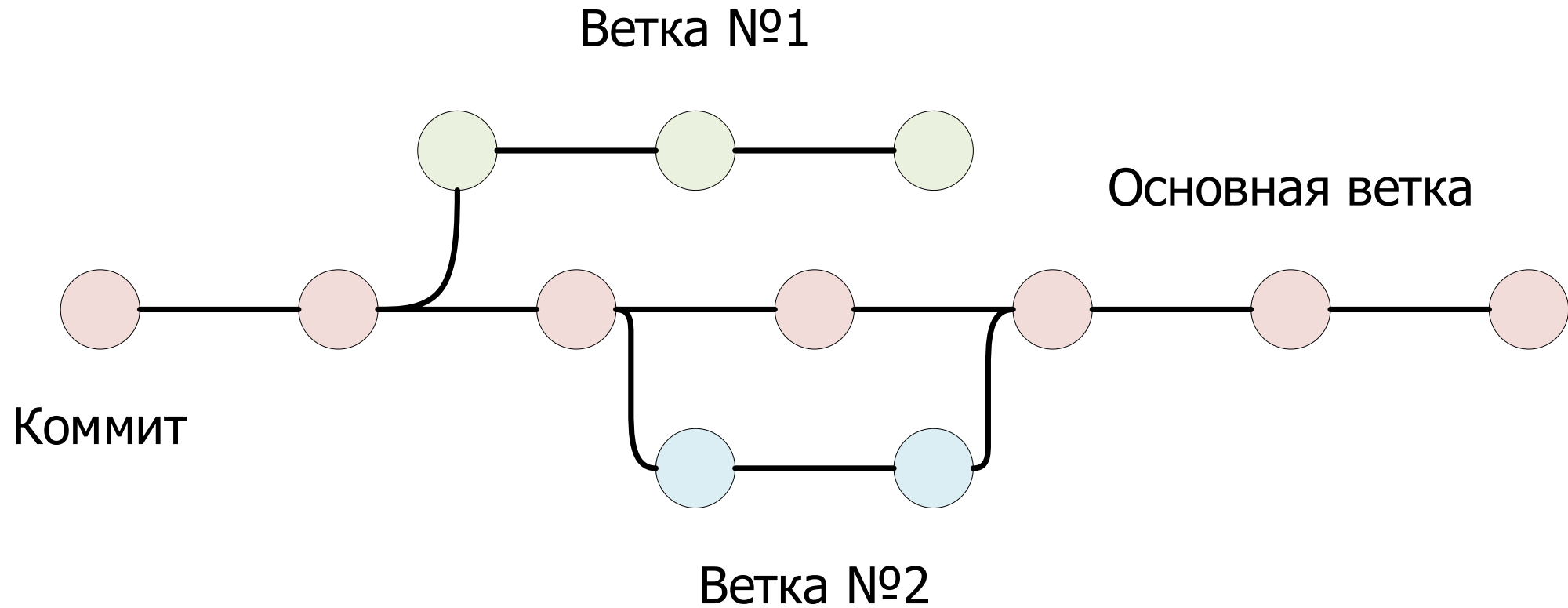
- повышенные затраты дисковой памяти по сравнению с централизованными СКВ
- при начальном клонировании проекта требуется длительное время для загрузки данных

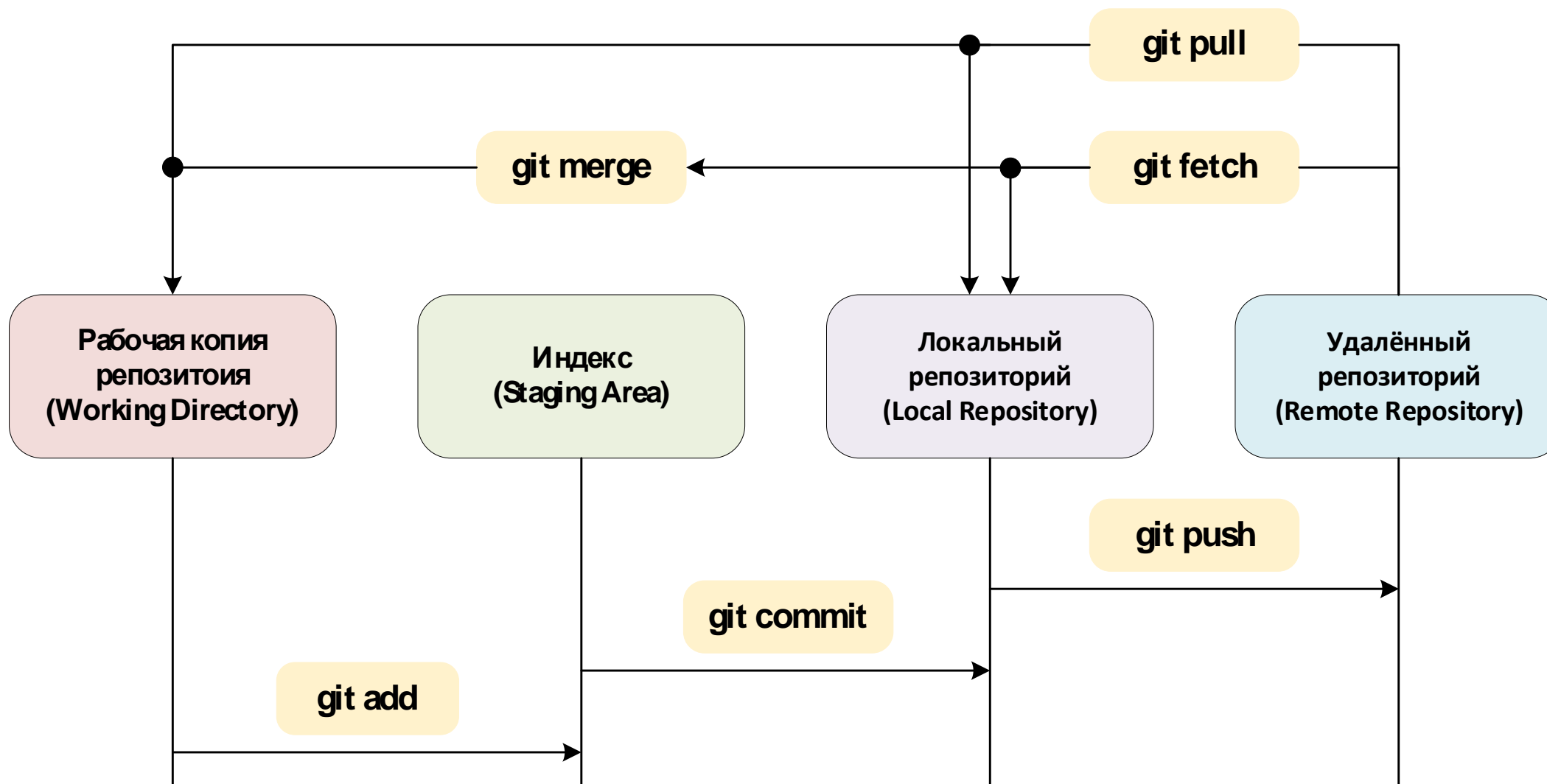


- **Git** – это распределенная система контроля версий с открытым исходным кодом
- Первоначальная реализация Git была разработана Линусом Торвальдсом в 2005 году для управления разработкой ядра Linux
- Консольная утилита, но имеет множество графических оболочек
- Официальный сайт: <https://git-scm.com/>



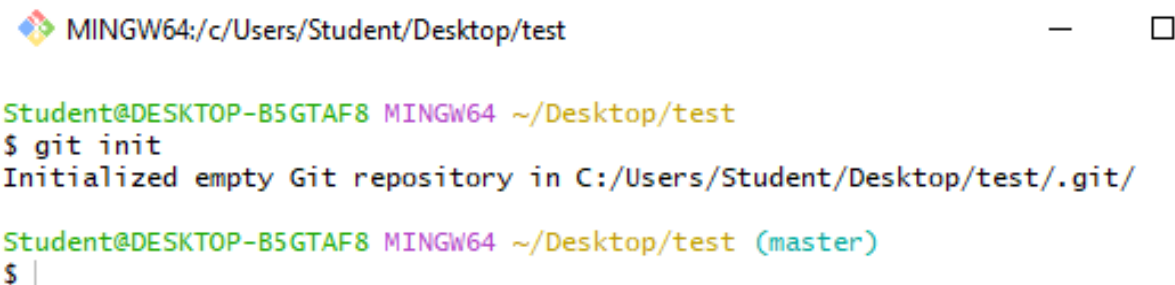
- **Репозиторий** – это каталог файловой системы, содержащий файлы проекта, состояние которых отслеживается системой контроля версий
- **Коммит** – это зафиксированный в определенный момент времени набор изменений в системе контроля версий
- **Ветка** – это параллельная версия проекта





1. git init

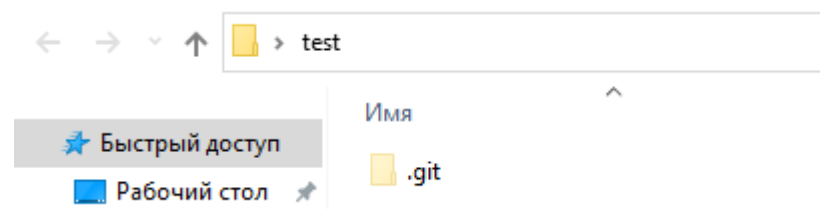
- Команда **git init** предназначена для создания (инициализации) нового репозитория и выполняется в первую очередь
- После выполнения данной команды в каталоге появляется скрытый каталог **.git**, который содержит все необходимые файлы репозитория



```
MINGW64:/c:/Users/Student/Desktop/test

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test
$ git init
Initialized empty Git repository in C:/Users/Student/Desktop/test/.git/

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ |
```



- Команда **git config** предназначена для конфигурации репозитория, глобальных настроек **git**
- После создания репозитория необходимо настроить имя пользователя и рабочую почту. Эта обязательная информация. Она будет отображаться рядом с каждым коммитом
- Для настройки имени пользователя и почты используются команды:
 - **git config user.name "Имя пользователя"**
 - **git config user.email "почта"**

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)  
$ git config user.name "Ivanov Ivan"
```

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)  
$ git config user.email "ivanov@mail.com"
```

- Команда **git add** предназначена для добавления файлов с изменениями в индекс
- Любой новый файл, который добавляется в репозиторий, по умолчанию не отслеживается Git. Поэтому, если необходим версионный контроль файла, то требуется выполнить команду **git add**
- Синтаксис: **git add имя_файла.расширение**

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)  
$ git add a.txt
```

- Команда **git status** позволяет понять, в каком состоянии находятся файлы в репозитории, текущую ветку
- Файлы в репозитории делятся на две категории неотслеживаемые (**untracked**) СКВ и отслеживаемые.
- В свою очередь, отслеживаемые файлы подразделяются на неизмененные (**unmodified**), измененные (**modified**) и подготовленные к коммиту (**staged**)
- Синтаксис: **git status**

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git status
On branch master

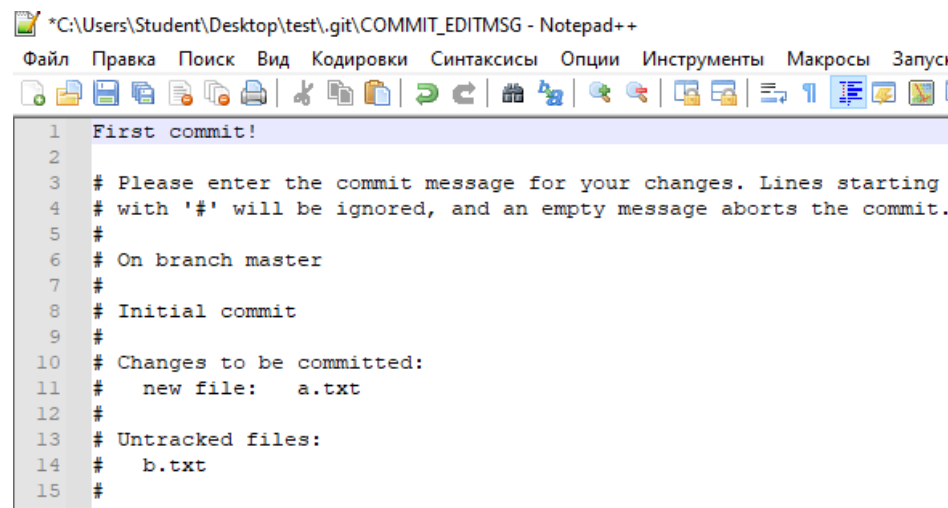
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   a.txt

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

- Команда **git commit** позволяет зафиксировать изменения в файлах, находящихся в индексе
- В процессе выполнения коммита, требуется оставить комментарий к коммиту, в котором описывается, причина изменений/добавлений файлов и т.д.
- Синтаксис: **git commit**

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git commit
[master (root-commit) 29ca927] First commit!
1 file changed, 1 insertion(+)
create mode 100644 a.txt
```



The screenshot shows a Notepad++ window titled '*C:\Users\Student\Desktop\test\.git\COMMIT_EDITMSG - Notepad++'. The menu bar includes 'Файл', 'Правка', 'Поиск', 'Вид', 'Кодировки', 'Синтаксисы', 'Опции', 'Инструменты', 'Макросы', and 'Запуск'. The text area contains the following content:

```
1 First commit!
2
3 # Please enter the commit message for your changes. Lines starting
4 # with '#' will be ignored, and an empty message aborts the commit.
5 #
6 # On branch master
7 #
8 # Initial commit
9 #
10 # Changes to be committed:
11 #   new file:   a.txt
12 #
13 # Untracked files:
14 #   b.txt
15 #
```

- Команда **git log** позволяет посмотреть историю коммитов, хэши коммитов, авторов, время создания
- Команда **git log** по умолчанию выводит коммиты в обратном хронологическом порядке
- Синтаксис: **git log**

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git log
commit decdde9eed489858f353704c0afee5cfe4eb3610 (HEAD -> master)
Author: Ivanov Ivan <ivanov@mail.com>
Date:   Tue Feb 13 01:26:26 2024 +0500
```

Second commit

```
commit 29ca927b36e3dca5c7ead766fa4f5e9502dcf3f5
Author: Ivanov Ivan <ivanov@mail.com>
Date:   Tue Feb 13 01:17:58 2024 +0500
```

First commit!

7. git checkout

- Команда **git checkout** позволяет перейти на предыдущую версию состояния репозитория, либо на предыдущую версию файла
- Синтаксис: **git checkout** хэш_коммита

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git checkout 29ca927b36e3dca5c7ead766fa4f5e9502dcf3f5
Note: switching to '29ca927b36e3dca5c7ead766fa4f5e9502dcf3f5'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

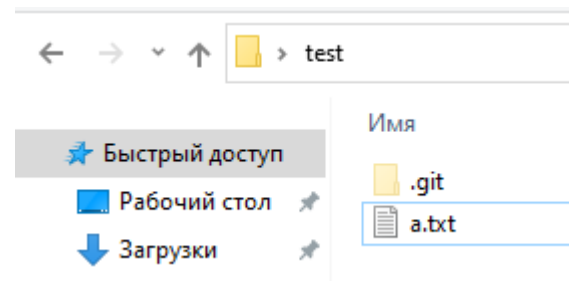
Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 29ca927 First commit!

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test ((29ca927...))
$ git status
HEAD detached at 29ca927
nothing to commit, working tree clean
```



- Команда **git branch** позволяет создать новую ветку от текущего коммита
- По умолчанию в репозитории присутствует одна ветка **master**
- Синтаксис: **git branch имя_ветки**
- Чтобы посмотреть список всех локальных веток, можно воспользоваться командой: **git branch --list**

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)  
$ git branch feature-1
```

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)  
$ git branch --list  
feature-1  
* master
```

- Команда **git checkout** также позволяет перейти с одной ветки на другую
- Синтаксис: **git checkout имя_ветки**
- Можно одновременно создать новую ветку и сразу перейти на нее при помощи команды:
git checkout -b имя_ветки

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git checkout feature-1
Switched to branch 'feature-1'

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (feature-1)
$ |
```

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git checkout -b feature-2
Switched to a new branch 'feature-2'

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (feature-2)
$ |
```

- Команда **git merge** позволяет выполнить слияние двух веток. Для этого сначала необходимо перейти в ту ветку, в которую будет сливаться другая ветка
- Синтаксис: **git merge имя_ветки**
- В команде **git merge** указывается название той ветки, которая будет сливаться в текущую ветку.

```
Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (feature-1)
$ git commit
[feature-1 633547c] A commit in the second branch
1 file changed, 1 insertion(+)
create mode 100644 c.txt

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (feature-1)
$ git checkout master
Switched to branch 'master'

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git log --graph
* commit decdde9eed489858f353704c0afee5cfe4eb3610 (HEAD -> master, feature-2)
| Author: Ivanov Ivan <ivanov@mail.com>
| Date: Tue Feb 13 01:26:26 2024 +0500
|
|     Second commit
|
* commit 29ca927b36e3dca5c7ead766fa4f5e9502dcf3f5
| Author: Ivanov Ivan <ivanov@mail.com>
| Date: Tue Feb 13 01:17:58 2024 +0500
|
|     First commit!
|

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
$ git merge feature-1
Updating decdde9..633547c
Fast-forward
 c.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 c.txt

Student@DESKTOP-B5GTAF8 MINGW64 ~/Desktop/test (master)
```



| Для локальной работы | Для взаимодействия с удаленным репозиторием | Для командной работы и устранения конфликтов |
|--|--|--|
| <ul style="list-style-type: none">▪ <code>git init</code>▪ <code>git config</code>▪ <code>git status</code>▪ <code>git log</code>▪ <code>git add</code>▪ <code>git diff</code>▪ <code>git commit</code>▪ <code>git branch</code>▪ <code>git merge</code> | <ul style="list-style-type: none">▪ <code>git remote</code>▪ <code>git push</code>▪ <code>git clone</code>▪ <code>git pull</code> | <ul style="list-style-type: none">▪ <code>git merge</code> (с конфликтами)▪ <code>git reset</code>▪ pull-реквесты▪ <code>gitFlow</code> |



- Для того, чтобы внести вклад в какой-либо Git-проект, необходимо уметь работать с удаленными репозиториями. Удаленные репозитории представляют собой версии проекта, сохраненные в интернете или еще где-то в сети.
- Может быть несколько удаленных репозиториях, каждый из которых может быть доступен для чтения или для чтения-записи.
- Взаимодействие с другими пользователями предполагает управление удаленными репозиториями, а также отправку и получение данных из них.





Удаленный репозиторий — это репозиторий, размещенный в локальной или интернет сети. Удаленный репозиторий используется для того, чтобы делиться и обмениваться кодом между разработчиками в рамках сети.



Самые распространенные решения:

- GitHub
- GitLab
- Bitbucket



Git ≠ GitHub

Git — это инструмент, позволяющий реализовать распределенную систему контроля версий, а GitHub — это сервис для проектов, использующих Git.



Для того, чтобы добавить удаленный репозиторий и присвоить ему имя (shortname), просто выполните команду `git remote add <shortname> <url>`.

Чтобы добавить удаленный репозиторий и присвоить ему название, которое используется для удобства как сокращение адреса необходимо выполнить:

```
git remote add origin <>
```



Команда `git branch -M <>` позволяет переименовать ветку master в main для совместимости с удаленным репозиторием:

```
git branch -M main
```



Команда `git push` позволяет отправлять локальную ветку на удаленный репозиторий:

```
git push origin main
```

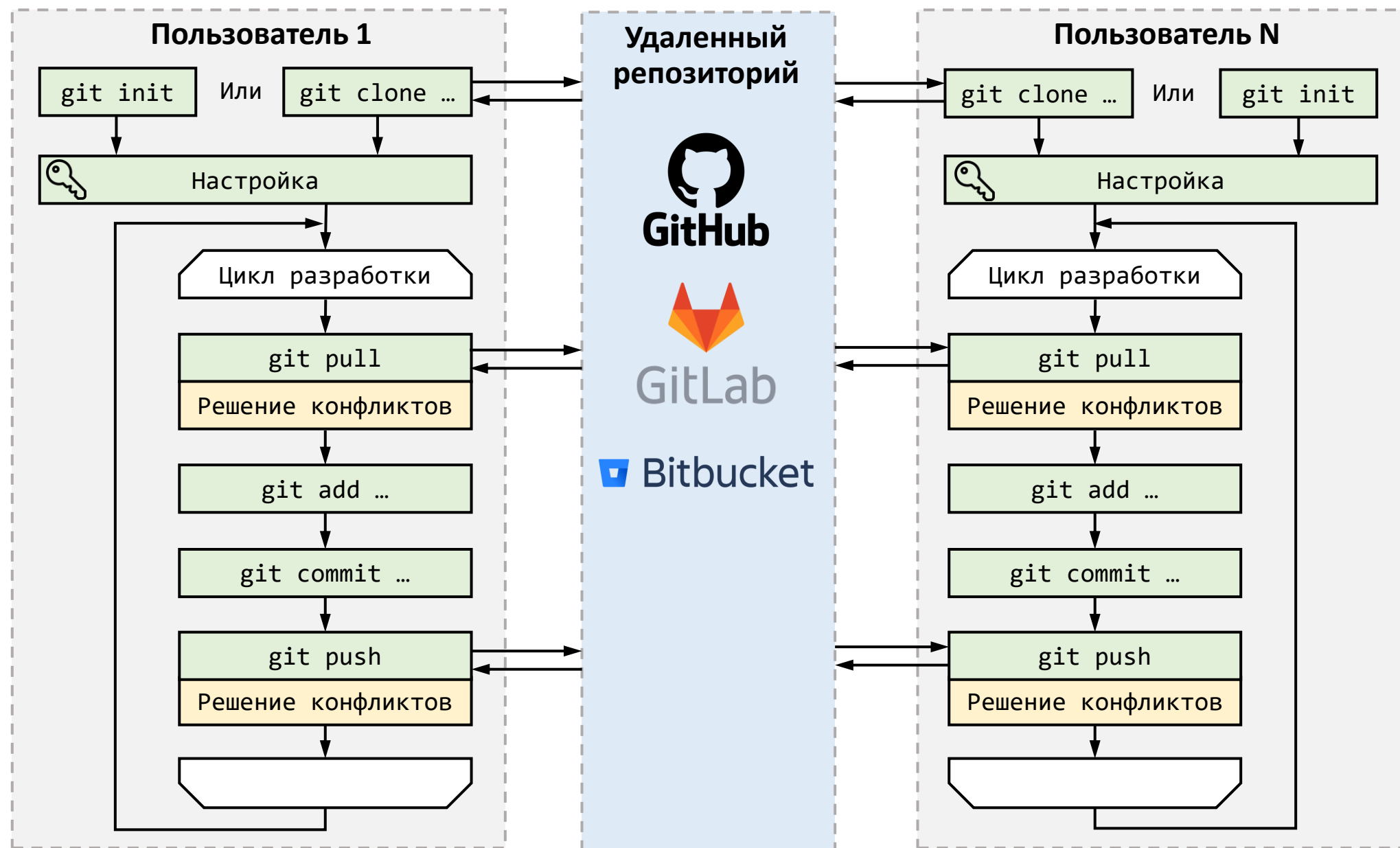
Такое каждый раз писать слишком громоздко. Для этого единожды набираем предыдущую команду с флагом `-u`:

```
git push -u origin main
```

После этого можно писать более коротко, так как `git` запомнил, что пушить надо на сервер `origin` ветку под именем `main`:

```
git push
```

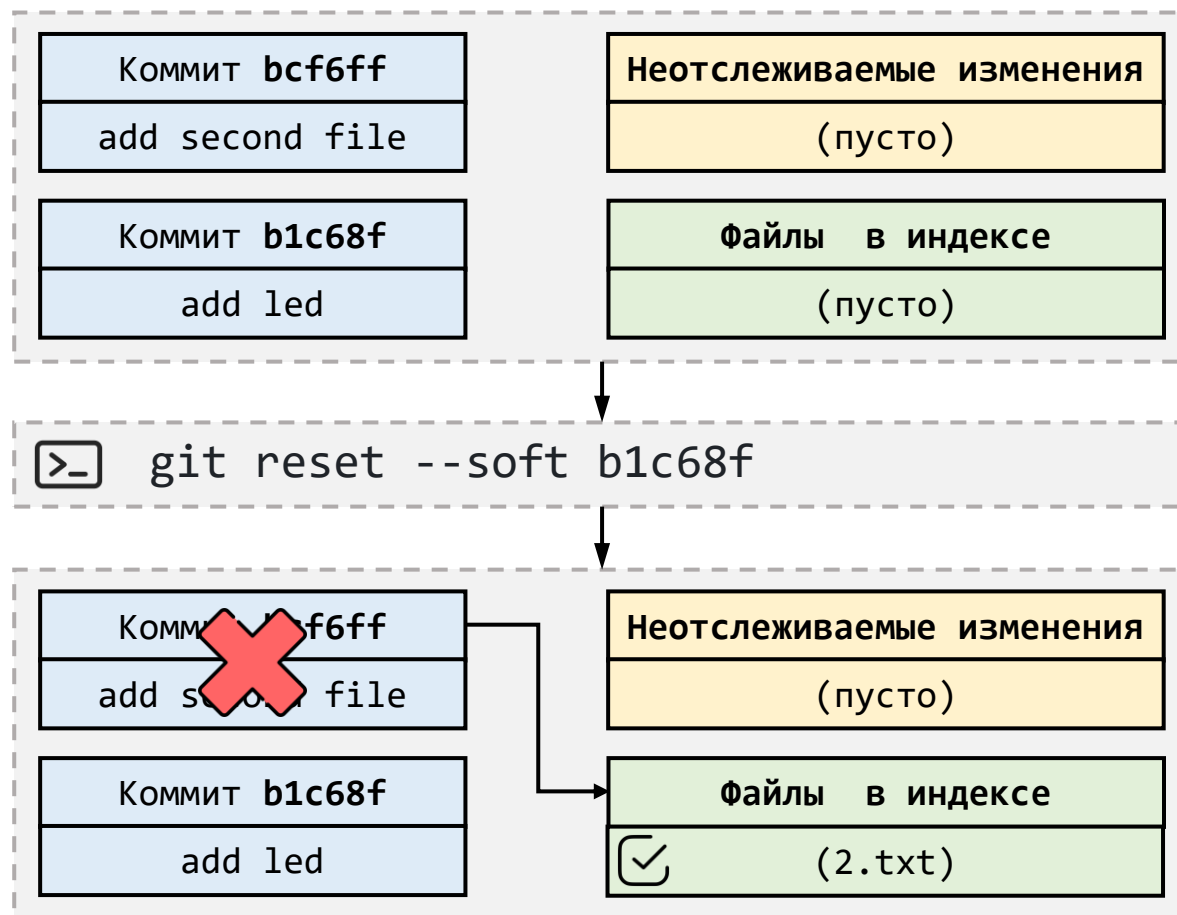

Процесс разработки при работе с одной веткой



Команда git reset --soft



Переносит все изменения в индекс



```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git log
commit bcf6ffdfc1e1898fd7147c36aebd11a59cb4c880 (HEAD -> master)
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 11:59:30 2024 +0300

    add second file

commit b1c68fa181027c0cac0216819747987cfd60613a
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 11:44:34 2024 +0300

    add led

User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git status
On branch master
nothing to commit, working tree clean
```

```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git reset --soft b1c68f

User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   2.txt

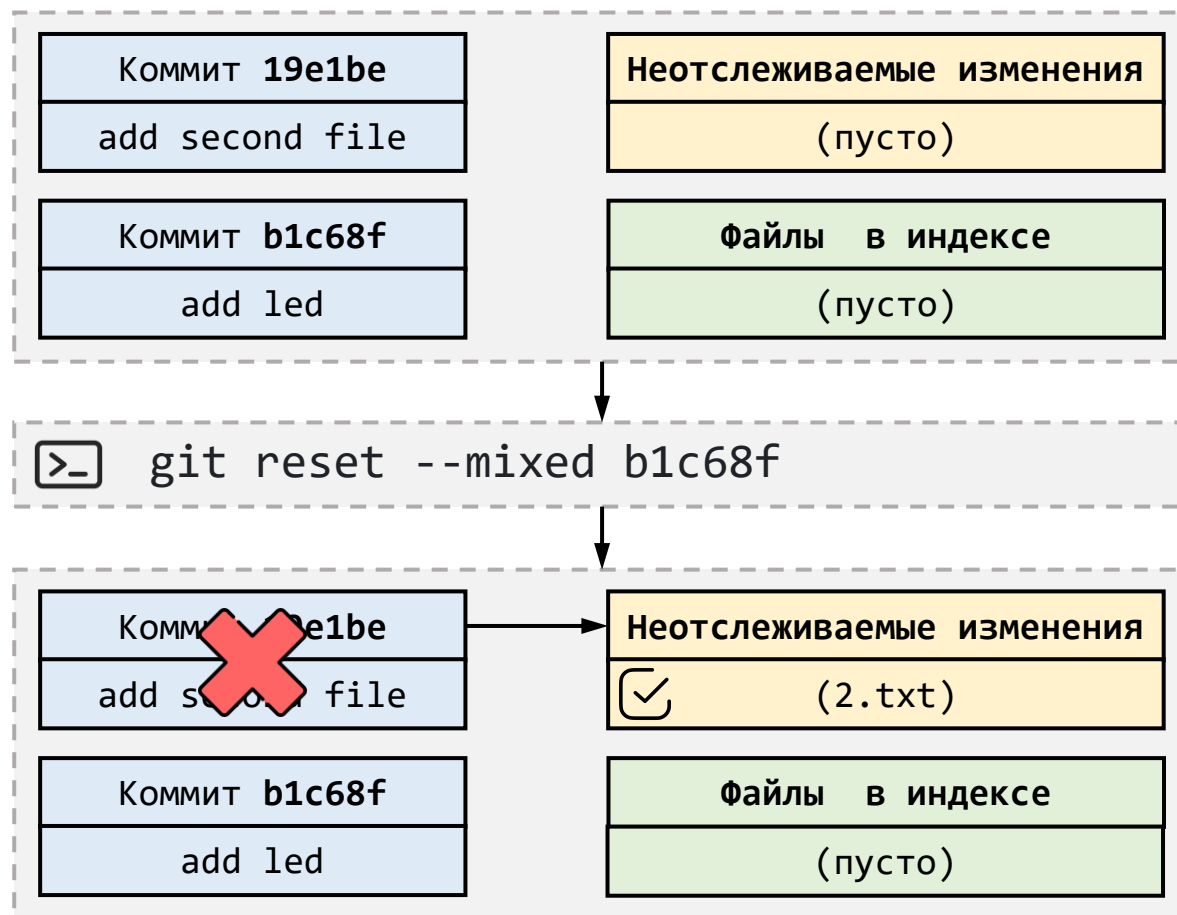
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git log
commit b1c68fa181027c0cac0216819747987cfd60613a (HEAD -> master)
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 11:44:34 2024 +0300

    add led
```

Команда git reset --mixed



Переносит все изменения в неотслеживаемую область. Аналог просто git reset



```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git log
commit 19e1be77eebe5931becebdbb4f99f3adacb5f8d2 (HEAD -> master)
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 12:16:04 2024 +0300
```

add second file

```
commit b1c68fa181027c0cac0216819747987cfd60613a
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 11:44:34 2024 +0300
```

add led

```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git status
On branch master
nothing to commit, working tree clean
```

```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git reset --mixed b1c68f

User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        2.txt

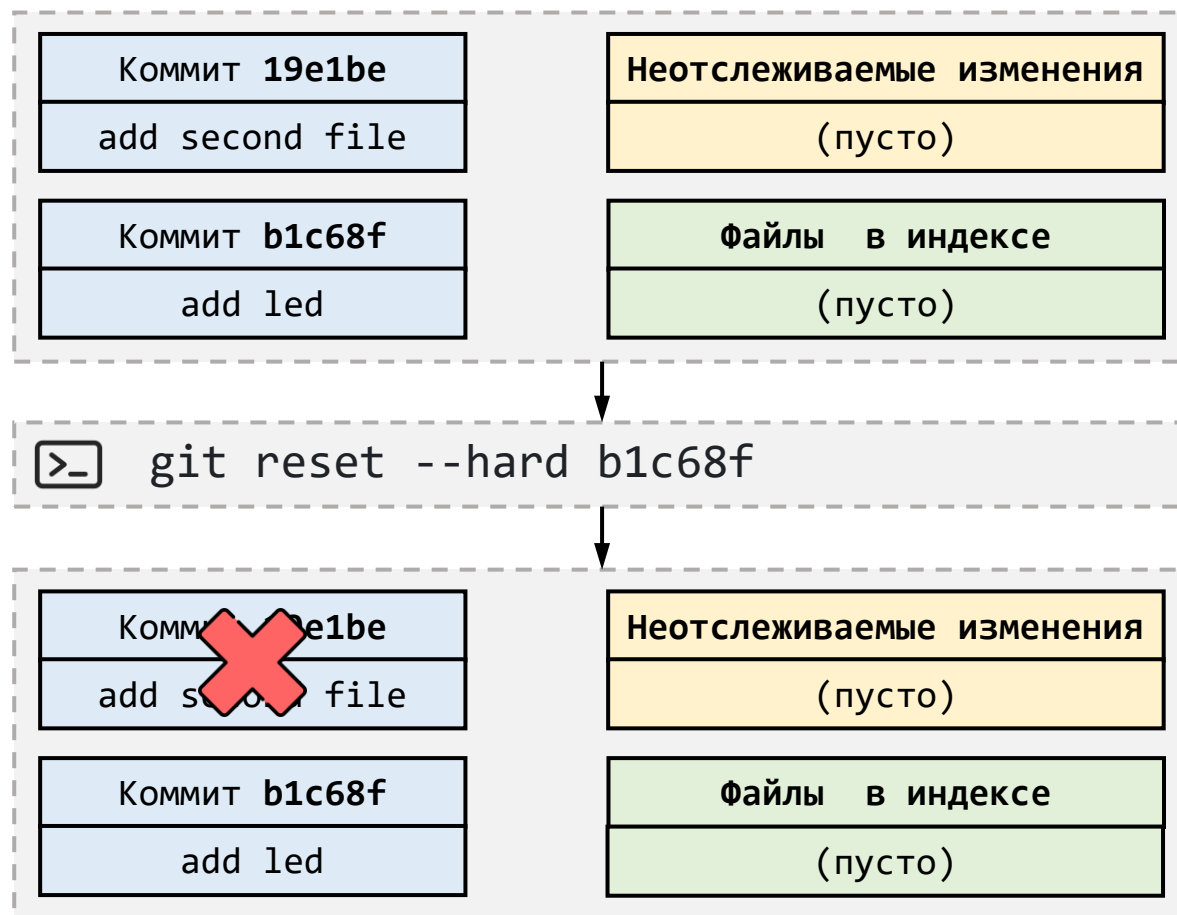
nothing added to commit but untracked files present (use "git add" to track)

User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$
```

Команда git reset --hard



Удаляем все изменения без
ВОЗМОЖНОСТИ ВОССТАНОВЛЕНИЯ



```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git log
commit d4572a9e13c5817e649da7f6ff73d09c0df33b90 (HEAD -> master)
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 12:21:30 2024 +0300
```

add second file

```
commit b1c68fa181027c0cac0216819747987cfd60613a
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 11:44:34 2024 +0300
```

add led

```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git status
On branch master
nothing to commit, working tree clean
```

```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git reset --hard b1c68f
HEAD is now at b1c68fa add led
```

```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git status
On branch master
nothing to commit, working tree clean
```

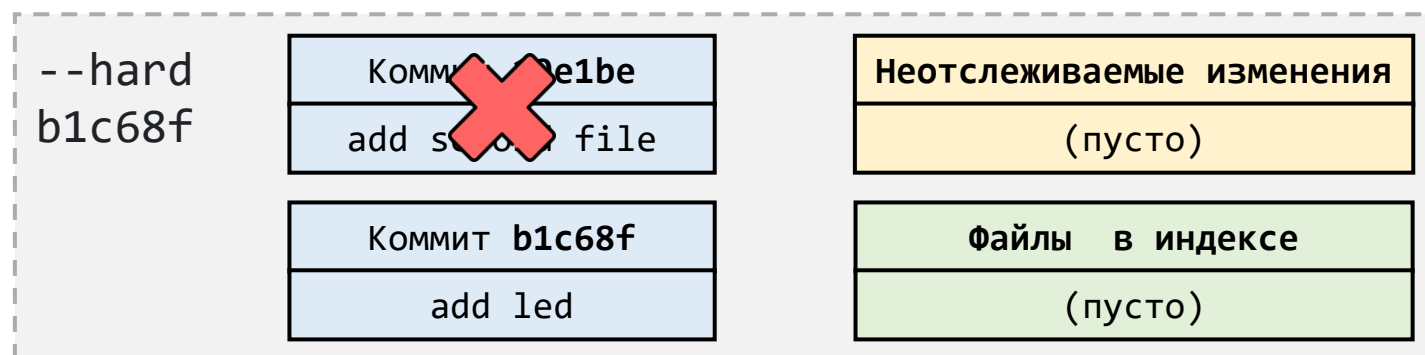
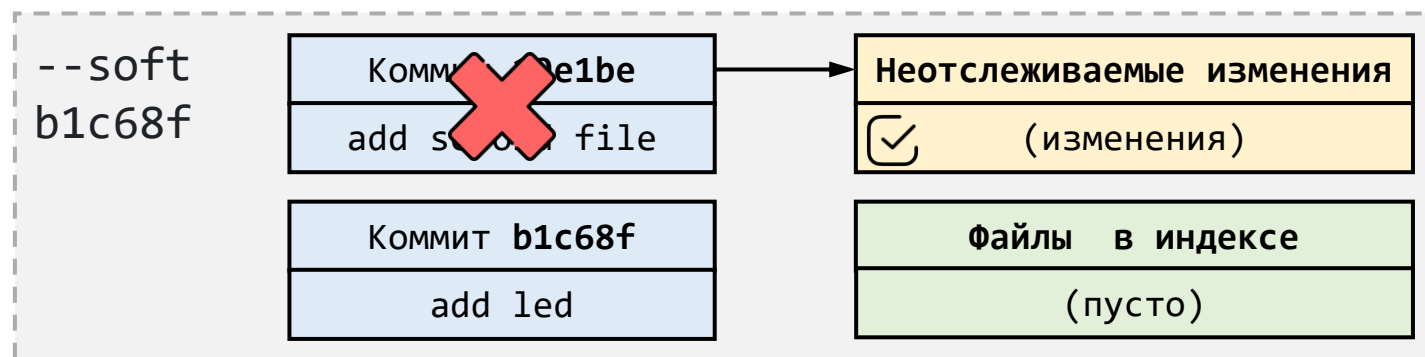
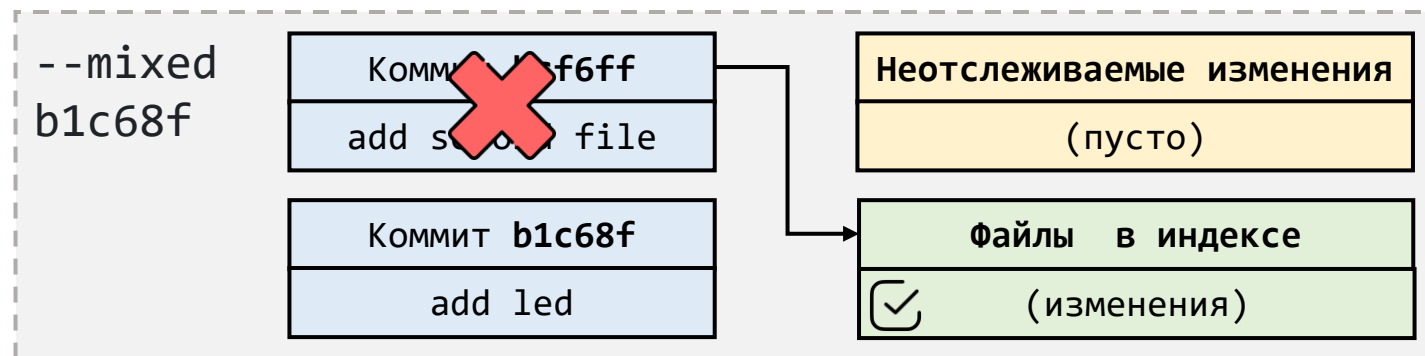
```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git log
commit b1c68fa181027c0cac0216819747987cfd60613a (HEAD -> master)
Author: Сергей Кузнецов <you@example.com>
Date: Thu Feb 15 11:44:34 2024 +0300
```

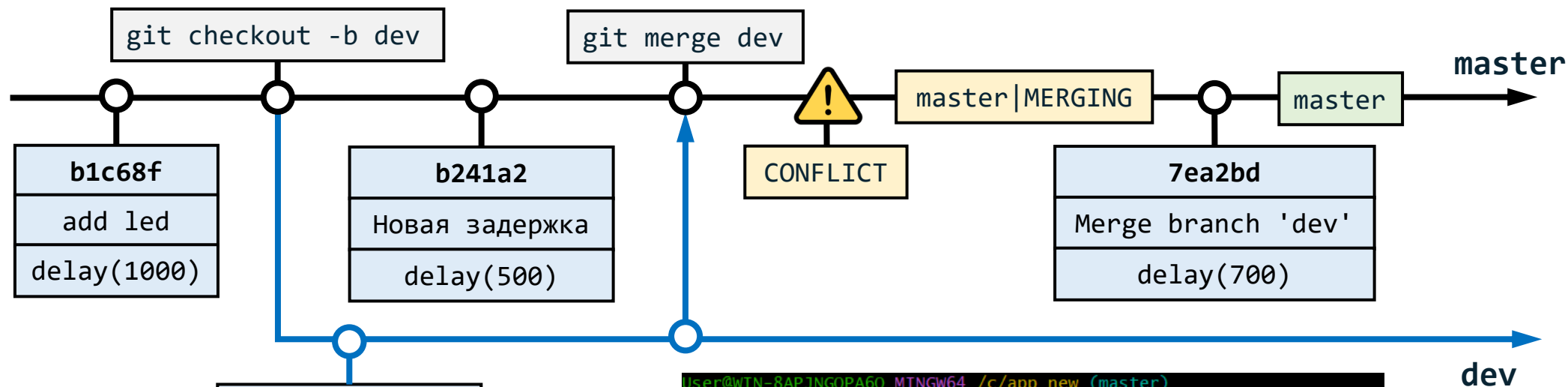
add led

Команды git reset. Обобщение



Если коммиты уже присутствуют на удаленном репозитории, то такой способ будет некорректным!





```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git log --all --graph
* commit 7ea2bd961c934c23ae25340c0d6ab0ccfe6e4c2b (HEAD -> master)
  Merge: b241a24 c3d1ecf
  Author: Сергей Кузнецов <you@example.com>
  Date: Thu Feb 15 13:31:22 2024 +0300

    Merge branch 'dev'

* commit c3d1ecf44bf562bc2b14a3b5ad90b6b4b7959628 (dev)
  Author: Сергей Кузнецов <you@example.com>
  Date: Thu Feb 15 13:19:11 2024 +0300

    Change delay

* commit b241a24bce394fe87deb8c95c81c274124385127
  Author: Сергей Кузнецов <you@example.com>
  Date: Thu Feb 15 13:22:42 2024 +0300

    Новая задержка

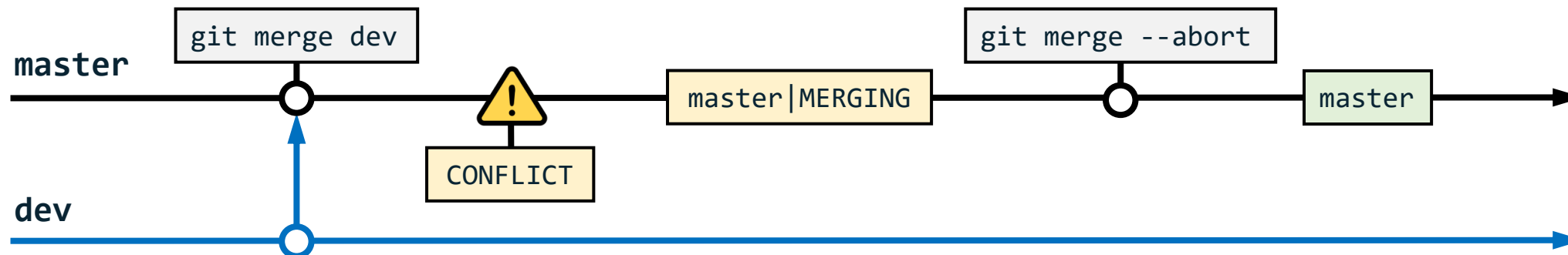
* commit b1c68fa181027c0cac0216819747987cfd60613a
  Author: Сергей Кузнецов <you@example.com>
  Date: Thu Feb 15 11:44:34 2024 +0300

    add led
```



Из режима устранения конфликтов можно выйти при помощи команды

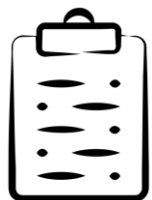
`git merge --abort`



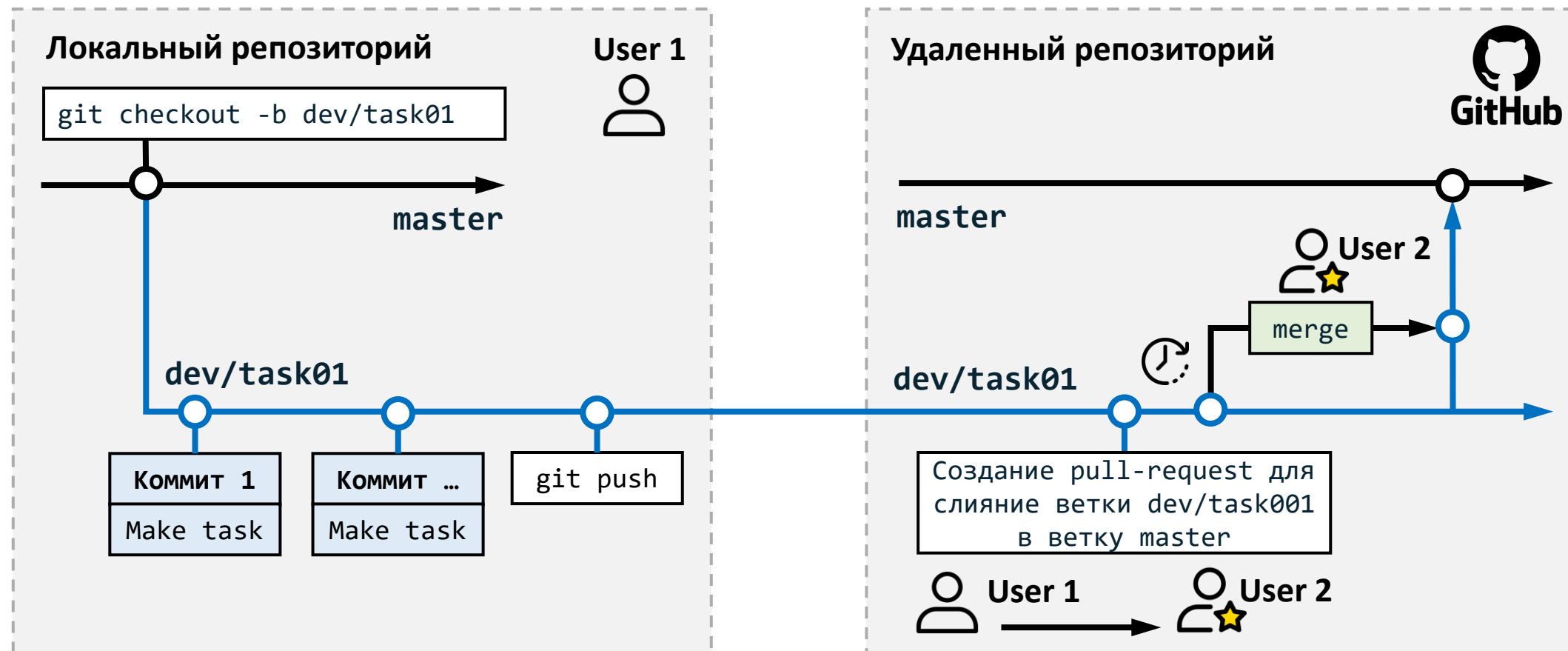
```
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
$ git merge dev
Auto-merging led.c
CONFLICT (content): Merge conflict in led.c
Automatic merge failed; fix conflicts and then commit the result.

User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master|MERGING)
$ git merge --abort

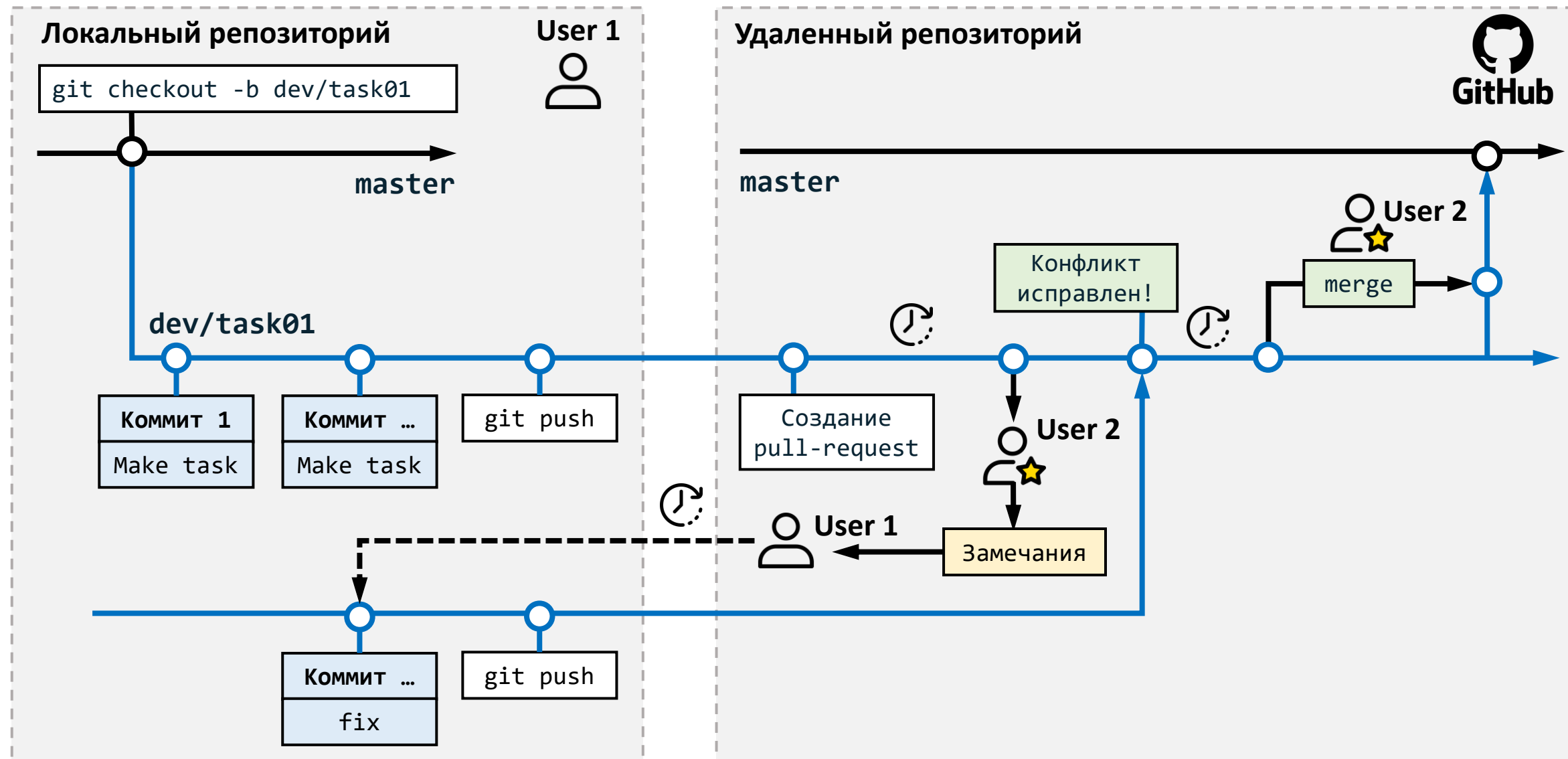
User@WIN-8APJNGOPA60 MINGW64 /c/app_new (master)
```



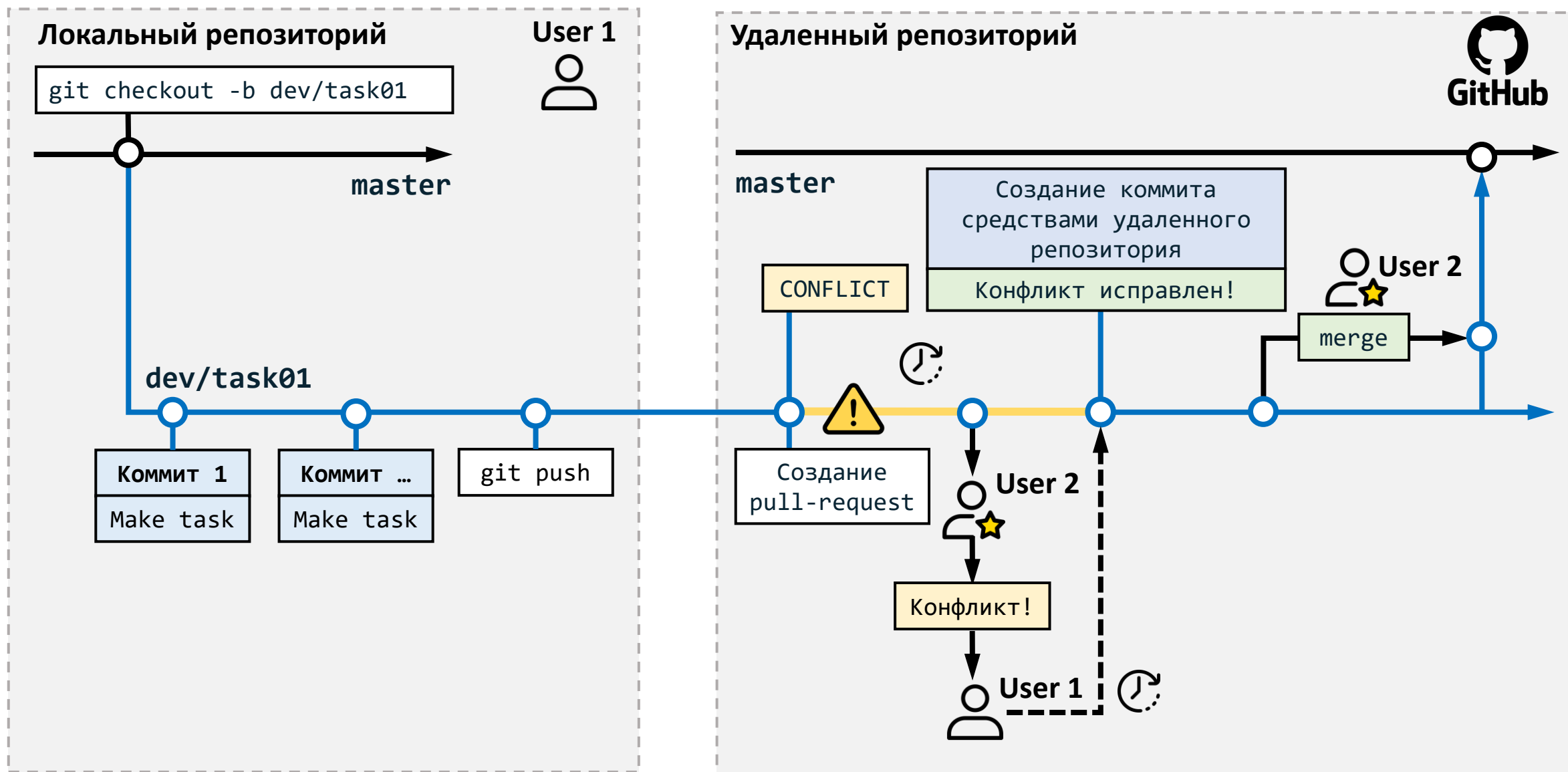
Обычно pull-request создает один пользователь, а рассматривает и принимает решения другой пользователь.



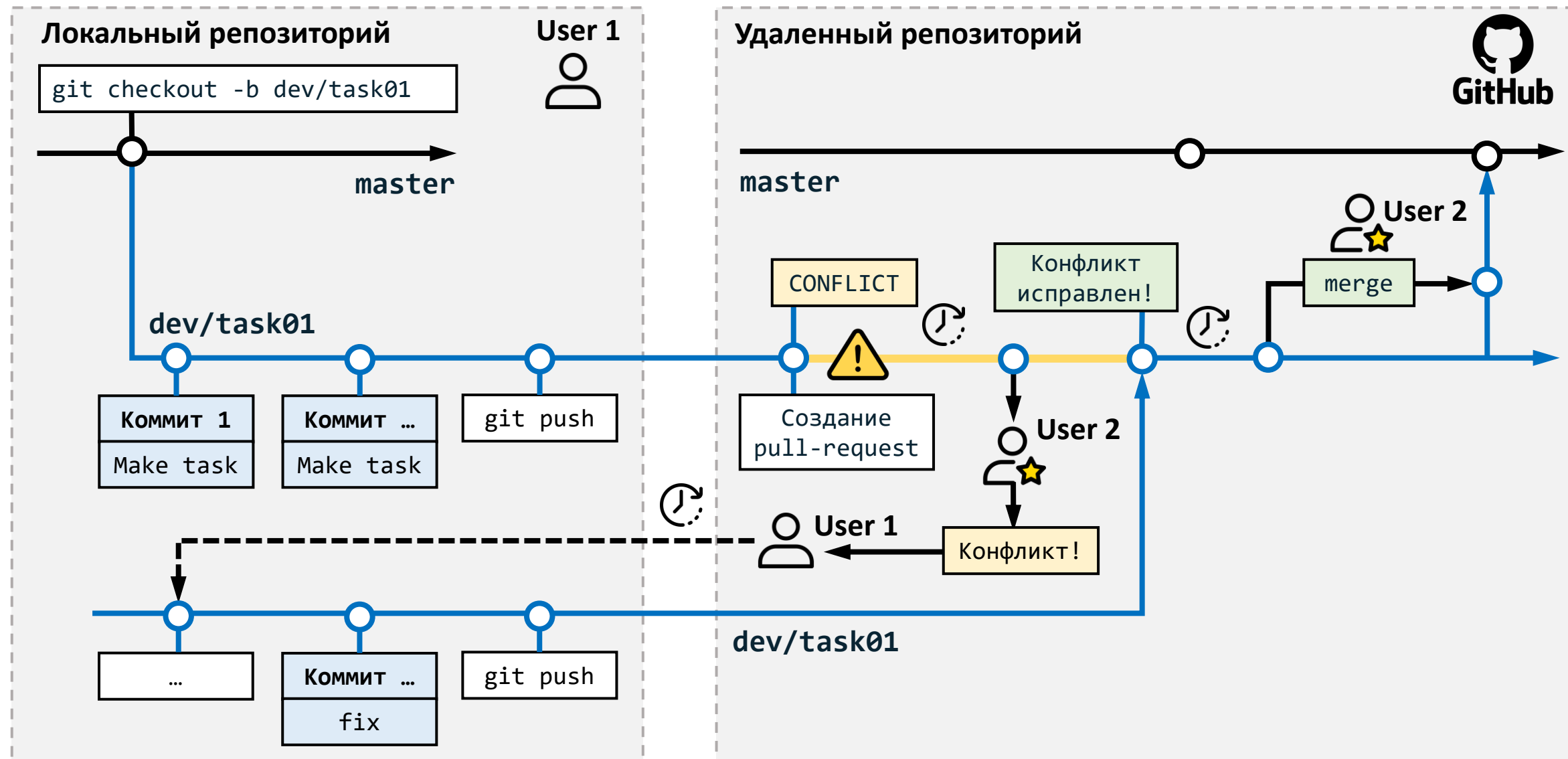
Pull request с замечаниями



Pull request с конфликтами. Вариант 1

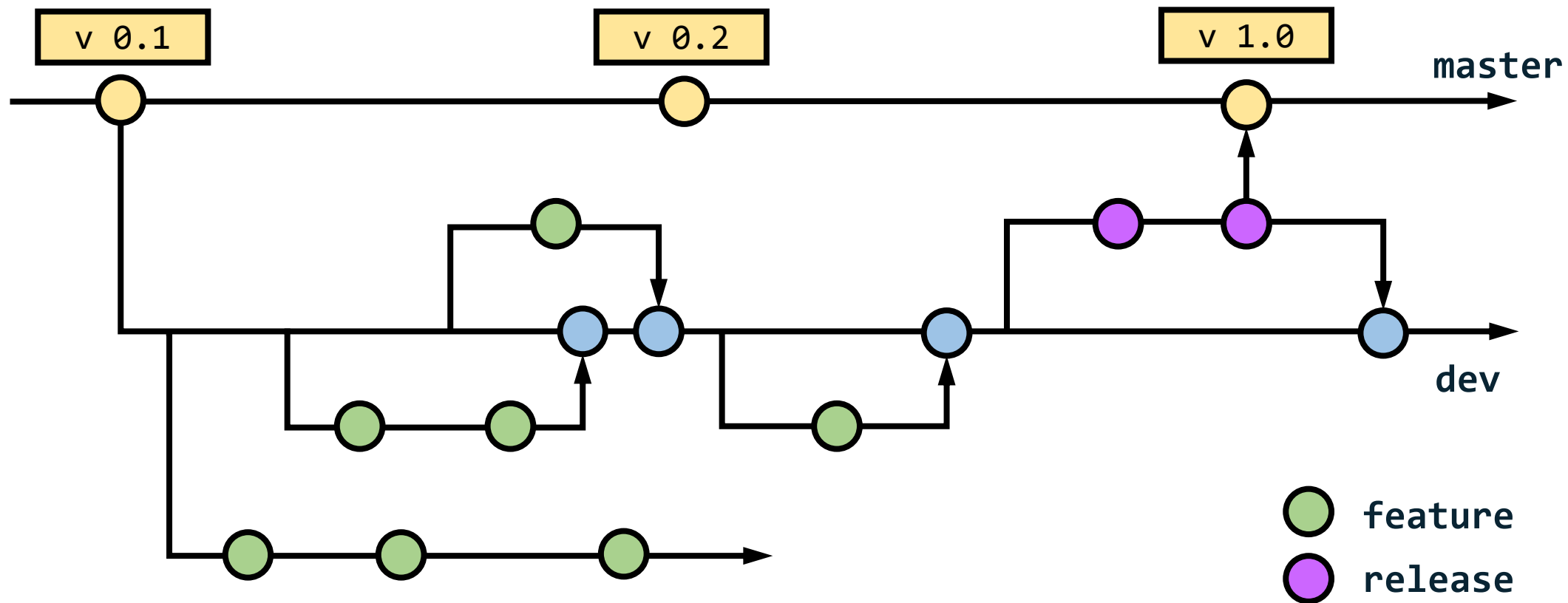


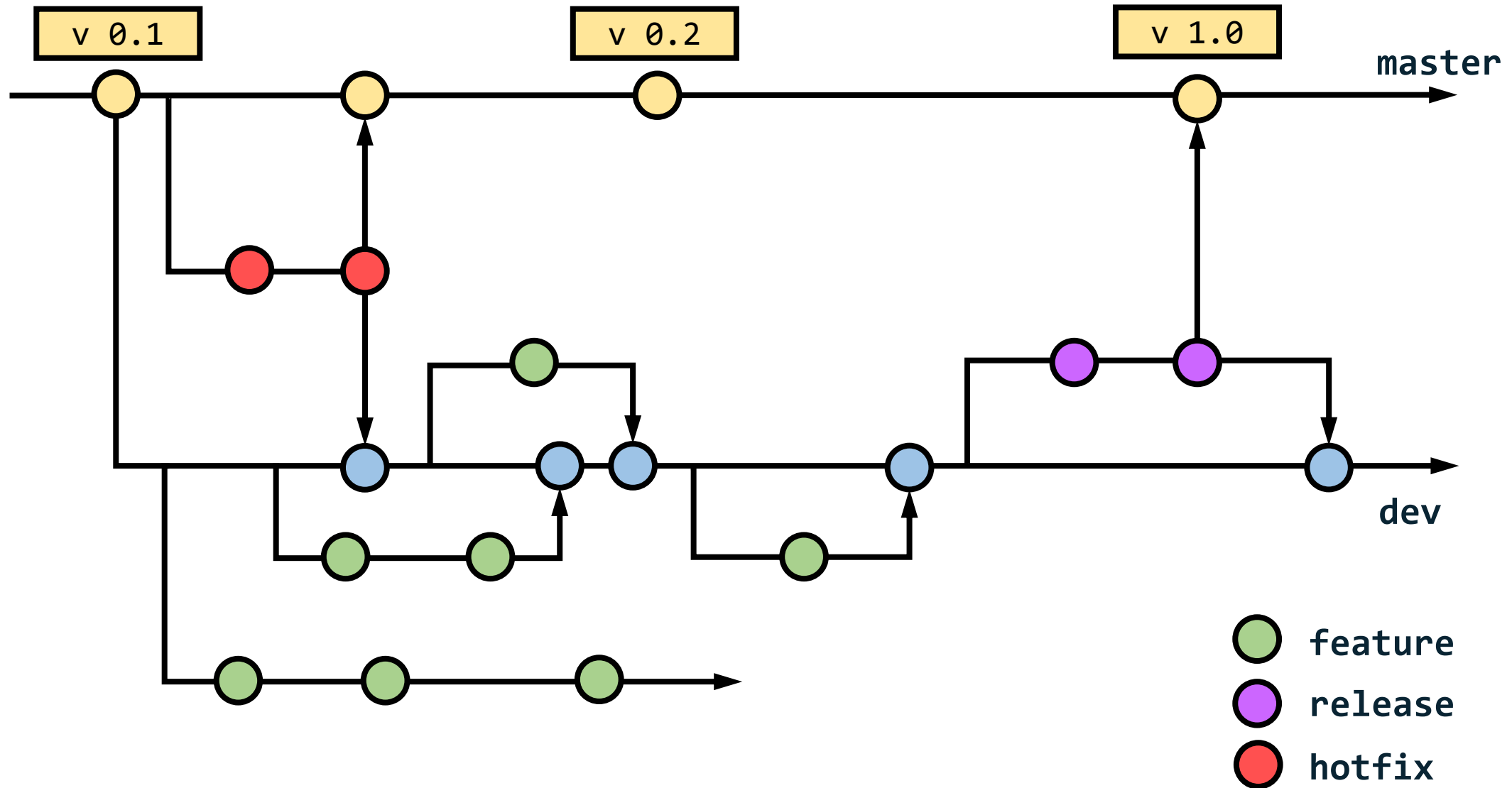
Pull request с конфликтами. Вариант 2





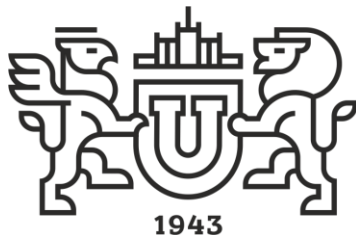
- Предполагает выстраивание строгой модели ветвления вокруг релиза проекта
- Подходит для проектов, которые имеют спланированный цикл релиза
- Вместо использования одной ветки master, в этой модели используется две ветки для записи истории проекта. В ветке master хранится официальная история релиза, а ветка develop служит в качестве интеграционной ветки для новых функций. Также, удобно тегировать все коммиты в ветке master номером версии.







- Из ветки master создается ветка dev.
- Из ветки dev создается ветка release.
- Из ветки dev создаются ветки feature.
- Когда работа над веткой feature завершается, она сливается в ветку dev.
- Когда работа над веткой release завершается, она сливается с ветками dev и master.
- Если в ветке master обнаруживается проблема, из master создается ветка hotfix.
- Когда работа над веткой hotfix завершается, она сливается с ветками dev и master



Южно-Уральский
государственный
университет

Национальный
исследовательский
университет

КРОК

Спасибо за внимание!



КРОК

Челябинск, ул. Карла Маркса, д. 38