**GIT**

**Настройка GIT**

**Есть 3 состояния файлов в папке:**

1 – созданы в папке,

2 – Git следит за ними,

3 – Git запоминает все изменения.

$ git log --help – вызов справки\описания

$ git init

Initialized empty Git repository in C:/Users/╨Р╨╗╨╡╨║╤Б╨░╨╜╨┤╤А/Desktop/HTML\_CSS\_JS/╨Я╤А╨░╨║╤В╨╕╨║╨░ ╨┐╨╛ ╨Я╨╡╤В╤А╨╕╤З╨╡╨╜╨║╨╛/.git/

Александр@LAPTOP-8QRP69PM MINGW64 ~/Desktop/HTML\_CSS\_JS/Практика по Петриченко (master)

$ echo >> README.md

Александр@LAPTOP-8QRP69PM MINGW64 ~/Desktop/HTML\_CSS\_JS/Практика по Петриченко (master)

$ git config --local user.name "Alex"

Александр@LAPTOP-8QRP69PM MINGW64 ~/Desktop/HTML\_CSS\_JS/Практика по Петриченко (master)

$ git config -- local user.email katruxin.85@mail.ru

Александр@LAPTOP-8QRP69PM MINGW64 ~/Desktop/HTML\_CSS\_JS/Практика по Петриченко (master)

$ ^C

$ git add -A

warning: LF will be replaced by CRLF in README.md.

The file will have its original line endings in your working directory

Александр@LAPTOP-8QRP69PM MINGW64 ~/Desktop/HTML\_CSS\_JS/Практика по Петриченко (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: Index.html

new file: README.md

new file: proba.js

new file: script.js

new file: style.css

new file: "~$\320\275\321\201\320\277\320\265\320\272\321\202.docx"

new file: ~WRL0003.tmp

new file: "\320\232\320\276\320\275\321\201\320\277\320\265\320\272\321\202.docx"

$ git commit -a -m "Это мой первый комит"

[master (root-commit) a295d33] ╨н╤В╨╛ ╨╝╨╛╨╣ ╨┐╨╡╤А╨▓╤Л╨╣ ╨║╨╛╨╝╨╕╤В

8 files changed, 233 insertions(+)

create mode 100644 Index.html

create mode 100644 README.md

create mode 100644 proba.js

create mode 100644 script.js

create mode 100644 style.css

create mode 100644 "~$\320\275\321\201\320\277\320\265\320\272\321\202.docx"

create mode 100644 ~WRL0003.tmp

create mode 100644 "\320\232\320\276\320\275\321\201\320\277\320\265\320\272\321\202.docx"

Александр@LAPTOP-8QRP69PM MINGW64 ~/Desktop/HTML\_CSS\_JS/Практика по Петриченко (master)

$ git status

On branch master

nothing to commit, working tree clean

**Если что-то поменял, то делай:**

Проверяй

$ git status

Увидел изменения:

On branch master

Your branch is up to date with 'testo/master'.

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: ╨Ъ╨╛╨╜╤Б╨┐╨╡╨║╤В.docx

Untracked files:

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

project/

project\_v2/

добавляй изменения:

$ git add –A

Добавляй комментарии - комить:

$ git commit -a -m "Мой третий комит"

Пушь изменения на git:

$ git push testo

Enumerating objects: 7, done.

Counting objects: 100% (7/7), done.

Delta compression using up to 4 threads

Compressing objects: 100% (4/4), done.

Writing objects: 100% (5/5), 23.63 KiB | 5.91 MiB/s, done.

Total 5 (delta 1), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (1/1), completed with 1 local object.

To https://github.com/AKatryukhin/testo.git

0efc9ef..70aee86 master -> master

**Всё. При обновлении окна браузера увидим все изменения и комментарии.**

## Контроль отображения записей

Есть много вариантов выбора, какие элементы отображаются в логе. Вот лучший:

git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short

Давайте рассмотрим его в деталях:

* --pretty="..." — определяет формат вывода.
* %h — укороченный хэш коммита
* %d — дополнения коммита («головы» веток или теги)
* %ad — дата коммита
* %s — комментарий
* %an — имя автора
* --graph — отображает дерево коммитов в виде ASCII-графика
* --date=short — сохраняет формат даты коротким и симпатичным

## Общие алиасы

Для пользователей Windows:

#### ВЫПОЛНИТЬ:

git config --global alias.co checkout

git config --global alias.ci commit

git config --global alias.st status

git config --global alias.br branch

git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"

git config --global alias.type 'cat-file -t'

git config --global alias.dump 'cat-file -p'

## Алиасы команд (опционально)

Если ваша оболочка поддерживает алиасы или шорткаты, вы можете добавить алиасы и на этом уровне. Я использую:

#### ФАЙЛ: .profile

alias gs='git status '

alias ga='git add '

alias gb='git branch '

alias gc='git commit'

alias gd='git diff'

alias gco='git checkout '

alias gk='gitk --all&'

alias gx='gitx --all'

alias got='git '

alias get='git '

Сокращение gco для команды git checkout особенно полезно. Оно позволяет мне вводить:

gco <branch>

для переключения в отдельную ветку.

И да, я достаточно часто пишу вместо git get или got, поэтому создам алиасы и для них.

# Получение старых версий

### Цели

* Научиться возвращать рабочий каталог к любому предыдущему состоянию.

Возвращаться назад в историю очень просто. Команда checkout скопирует любой снимок из репозитория в рабочий каталог.

## 01Получите хэши предыдущих версий

#### ВЫПОЛНИТЕ:

git hist

**Примечание:** Вы не забыли задать hist в вашем файле .gitconfig? Если забыли, посмотрите еще раз урок по [алиасам](https://githowto.com/ru/aliases).

#### РЕЗУЛЬТАТ:

$ git hist

\* fa3c141 2011-03-09 | Added HTML header (HEAD, master) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Изучите данные лога и найдите хэш для первого коммита. Он должен быть в последней строке данных git hist. Используйте этот хэш-код (достаточно первых 7 знаков) в команде ниже. Затем проверьте содержимое файла hello.html.

#### ВЫПОЛНИТЕ:

git checkout <hash>

cat hello.html

**Примечание:** Многие команды зависят от хэшевых значений в репозитории. Поскольку ваши хеш-значения будут отличаться от моих, когда вы видите что-то вроде <hash> или <treehash> в команде, подставьте необходимое значение хэш для вашего репозитория.

Вы увидите…

#### РЕЗУЛЬТАТ:

$ git checkout 911e8c9

Note: checking out '911e8c9'.

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 performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b new\_branch\_name

HEAD is now at 911e8c9... First Commit

$ cat hello.html

Hello, World

Выходные данные команды checkout очень хорошо объясняют ситуацию. Старые версии git будут ругаться, что не расположены в локальной ветке. В любом случае, сейчас об этом не беспокойтесь.

Обратите внимание на то, что содержимое файла hello.html является значением по умолчанию.

## 02Вернитесь к последней версии в ветке master

#### ВЫПОЛНИТЕ:

git checkout master

cat hello.html

Вы увидите…

#### РЕЗУЛЬТАТ:

$ git checkout master

Previous HEAD position was 911e8c9... First Commit

Switched to branch 'master'

$ cat hello.html

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

«master» — имя ветки по умолчанию. Переключая имена веток, вы попадаете на последнюю версию выбранной ветки.

# Создание тегов версий

### Цели

* Узнать, как создавать теги для коммитов для использования в будущем

Давайте назовем текущую версию страницы hello первой (v1).

## 01Создайте тег первой версии

#### ВЫПОЛНИТЕ:

git tag v1

Теперь текущая версия страницы называется v1.

## 02Теги для предыдущих версий

Давайте создадим тег для версии, которая идет перед текущей версией и назовем его v1-beta. В первую очередь нам надо переключиться на предыдущую версию. Вместо поиска по хэшу, мы будем использовать ^, обозначающее «родитель v1».

Если обозначение v1^ вызывает у вас какие-то проблемы, попробуйте также v1~1, указывающее на ту же версию. Это обозначение можно определить как «первую версию предшествующую v1».

#### ВЫПОЛНИТЕ:

git checkout v1^

cat hello.html

#### РЕЗУЛЬТАТ:

$ git checkout v1^

Note: checking out 'v1^'.

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 performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b new\_branch\_name

HEAD is now at 8c32287... Added standard HTML page tags

$ cat hello.html

<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Это версия c тегами <html> и <body>, но еще пока без <head>. Давайте сделаем ее версией v1-beta.

#### ВЫПОЛНИТЕ:

git tag v1-beta

## 03Переключение по имени тега

Теперь попробуйте попереключаться между двумя отмеченными версиями.

#### ВЫПОЛНИТЕ:

git checkout v1

git checkout v1-beta

#### РЕЗУЛЬТАТ:

$ git checkout v1

Previous HEAD position was 8c32287... Added standard HTML page tags

HEAD is now at fa3c141... Added HTML header

$ git checkout v1-beta

Previous HEAD position was fa3c141... Added HTML header

HEAD is now at 8c32287... Added standard HTML page tags

## 04Просмотр тегов с помощью команды tag

Вы можете увидеть, какие теги доступны, используя команду git tag.

#### ВЫПОЛНИТЕ:

git tag

#### РЕЗУЛЬТАТ:

$ git tag

v1

v1-beta

## 05Просмотр Тегов в логах

Вы также можете посмотреть теги в логе.

#### ВЫПОЛНИТЕ:

git hist master --all

#### РЕЗУЛЬТАТ:

$ git hist master --all

\* fa3c141 2011-03-09 | Added HTML header (v1, master) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (HEAD, v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Вы можете видеть теги (v1 и v1-beta) в логе вместе с именем ветки (master). Кроме того HEAD показывает коммит, на который вы переключились (на данный момент это v1-beta).

## Отмена локальных изменений (до индексации)

### Цели

* Научиться отменять изменения в рабочем каталоге

## 01Переключитесь на ветку Master

Убедитесь, что вы находитесь на последнем коммите ветки master, прежде чем продолжить работу.

#### ВЫПОЛНИТЕ:

git checkout master

## 02Измените hello.html

Иногда случается, что вы изменили файл в рабочем каталоге, и хотите отменить последние коммиты. С этим справится команда checkout.

Внесите изменение в файл hello.html в виде нежелательного комментария.

#### ФАЙЛ: hello.html

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

**<!-- This is a bad comment. We want to revert it. -->**

</body>

</html>

## 03Проверьте состояние

Сначала проверьте состояние рабочего каталога.

#### ВЫПОЛНИТЕ:

git status

#### РЕЗУЛЬТАТ:

$ git status

# On branch master

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout -- <file>..." to discard changes in working directory)

#

# modified: hello.html

#

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

Мы видим, что файл hello.html был изменен, но еще не проиндексирован.

## 04Отмена изменений в рабочем каталоге

Используйте команду checkout для переключения в версию файла hello.html в репозитории.

#### ВЫПОЛНИТЕ:

git checkout hello.html

git status

cat hello.html

#### РЕЗУЛЬТАТ:

$ git checkout hello.html

$ git status

# On branch master

nothing to commit (working directory clean)

$ cat hello.html

<html>

**<head>**

**</head>**

<body>

<h1>Hello, World!</h1>

</body>

</html>

Команда status показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге. И «нежелательный комментарий» больше не является частью содержимого файла.

## Отмена проиндексированных изменений (перед коммитом)

### Цели

* Научиться отменять изменения, которые были проиндексированы

## 01Измените файл и проиндексируйте изменения

Внесите изменение в файл hello.html в виде нежелательного комментария

#### ФАЙЛ: HELLO.HTML

<html>

<head>

**<!-- This is an unwanted but staged comment -->**

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Проиндексируйте это изменение.

#### ВЫПОЛНИТЕ:

git add hello.html

## 02Проверьте состояние

Проверьте состояние нежелательного изменения.

#### ВЫПОЛНИТЕ:

git status

#### РЕЗУЛЬТАТ:

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: hello.html

#

Состояние показывает, что изменение было проиндексировано и готово к коммиту.

## 03Выполните сброс буферной зоны

К счастью, вывод состояния показывает нам именно то, что мы должны сделать для отмены индексации изменения.

#### ВЫПОЛНИТЕ:

git reset HEAD hello.html

#### РЕЗУЛЬТАТ:

$ git reset HEAD hello.html

Unstaged changes after reset:

M hello.html

Команда reset сбрасывает буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали.

Команда reset (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий. Мы можем использовать команду checkout из предыдущего урока, чтобы удалить нежелательные изменения в рабочем каталоге.

## 04Переключитесь на версию коммита

#### ВЫПОЛНИТЕ:

git checkout hello.html

git status

#### РЕЗУЛЬТАТ:

$ git status

# On branch master

nothing to commit (working directory clean)

Наш рабочий каталог опять чист.

# Отмена коммитов

### Цели

* Научиться отменять коммиты в локальный репозиторий.

## 01Отмена коммитов

Иногда вы понимаете, что новые коммиты являются неверными, и хотите их отменить. Есть несколько способов решения этого вопроса, здесь мы будем использовать самый безопасный.

Мы отменим коммит путем создания нового коммита, отменяющего нежелательные изменения.

## 02Измените файл и сделайте коммит

Измените файл hello.html на следующий.

#### ФАЙЛ: hello.html

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

**<!-- This is an unwanted but committed change -->**

</body>

</html>

#### ВЫПОЛНИТЕ:

git add hello.html

git commit -m "Oops, we didn't want this commit"

## 03Сделайте коммит с новыми изменениями, отменяющими предыдущие

Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом.

#### ВЫПОЛНИТЕ:

git revert HEAD

Перейдите в редактор, где вы можете отредактировать коммит-сообщение по умолчанию или оставить все как есть. Сохраните и закройте файл. Вы увидите…

#### РЕЗУЛЬТАТ:

$ git revert HEAD --no-edit

[master 45fa96b] Revert "Oops, we didn't want this commit"

1 files changed, 1 insertions(+), 1 deletions(-)

Так как мы отменили самый последний произведенный коммит, мы смогли использовать HEAD в качестве аргумента для отмены. Мы можем отменить любой произвольной коммит в истории, указав его хэш-значение.

**Примечание:** Команду --no-edit можно проигнорировать. Она была необходима для генерации выходных данных без открытия редактора.

## 04Проверьте лог

Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.

#### ВЫПОЛНИТЕ:

git hist

#### РЕЗУЛЬТАТ:

$ git hist

\* 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (HEAD, master) [Alexander Shvets]

\* 846b90c 2011-03-09 | Oops, we didn't want this commit [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Эта техника будет работать с любым коммитом (хотя, возможно, возникнут конфликты). Она безопасна в использовании даже в публичных ветках удаленных репозиториев.

## 05Далее

Далее давайте посмотрим на технику, которая может быть использована для удаления последних коммитов из истории репозитория.

# Удаление коммитов из ветки

### Цели

* Научиться удалять самые последние коммиты из ветки

Revert из предыдущего раздела является мощной командой, которая позволяет отменить любые коммиты в репозиторий. Однако, и оригинальный и «отмененный» коммиты видны в истории ветки (при использовании команды git log).

Часто мы делаем коммит, и сразу понимаем, что это была ошибка. Было бы неплохо иметь команду «возврата», которая позволила бы нам сделать вид, что неправильного коммита никогда и не было. Команда «возврата» даже предотвратила бы появление нежелательного коммита в истории git log.

## 01Команда reset

Мы уже видели команду reset и использовали ее для согласования буферной зоны и выбранного коммита (мы использовали коммит HEAD в нашем предыдущем уроке).

При получении ссылки на коммит (т.е. хэш, ветка или имя тега), команда reset…

1. Перепишет текущую ветку, чтобы она указывала на нужный коммит
2. Опционально сбросит буферную зону для соответствия с указанным коммитом
3. Опционально сбросит рабочий каталог для соответствия с указанным коммитом

## 02Проверьте нашу историю

Давайте сделаем быструю проверку нашей истории коммитов.

#### ВЫПОЛНИТЕ:

git hist

#### РЕЗУЛЬТАТ:

$ git hist

\* 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (HEAD, master) [Alexander Shvets]

\* 846b90c 2011-03-09 | Oops, we didn't want this commit [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Мы видим, что два последних коммита в этой ветке - «Oops» и «Revert Oops». Давайте удалим их с помощью сброса.

## 03Для начала отметьте эту ветку

Но прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти.

#### ВЫПОЛНИТЕ:

git tag oops

## 04Сброс коммитов к предшествующим коммиту Oops

Глядя на историю лога (см. выше), мы видим, что коммит с тегом «v1» является коммитом, предшествующим ошибочному коммиту. Давайте сбросим ветку до этой точки. Поскольку ветка имеет тег, мы можем использовать имя тега в команде сброса (если она не имеет тега, мы можем использовать хэш-значение).

#### ВЫПОЛНИТЕ:

git reset --hard v1

git hist

#### РЕЗУЛЬТАТ:

$ git reset --hard v1

HEAD is now at fa3c141 Added HTML header

$ git hist

\* fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Наша ветка master теперь указывает на коммит v1, а коммитов Oops и Revert Oops в ветке уже нет. Параметр --hard указывает, что рабочий каталог должен быть обновлен в соответствии с новым head ветки.

## 05Ничего никогда не теряется

Что же случается с ошибочными коммитами? Оказывается, что коммиты все еще находятся в репозитории. На самом деле, мы все еще можем на них ссылаться. Помните, в начале этого урока мы создали для отмененного коммита тег «oops». Давайте посмотрим на все коммиты.

#### ВЫПОЛНИТЕ:

git hist --all

#### РЕЗУЛЬТАТ:

$ git hist --all

\* 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (oops) [Alexander Shvets]

\* 846b90c 2011-03-09 | Oops, we didn't want this commit [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке master. Если бы мы не отметили их тегами, они по-прежнему находились бы в репозитории, но не было бы никакой возможности ссылаться на них, кроме как при помощи их хэш имен. Коммиты, на которые нет ссылок, остаются в репозитории до тех пор, пока не будет запущен сборщик мусора.

## 06Опасность сброса

Сброс в локальных ветках, как правило, безопасен. Последствия любой «аварии» как правило, можно восстановить простым сбросом с помощью нужного коммита.

Однако, если ветка «расшарена» на удаленных репозиториях, сброс может сбить с толку других пользователей ветки.

# Удаление тега oops

### Цели

* Удаление тега oops (уборка)

## 01Удаление тега oops

Тег oops свою функцию выполнил. Давайте удалим его и коммиты, на которые он ссылался, сборщиком мусора.

#### ВЫПОЛНИТЕ:

git tag -d oops

git hist --all

#### РЕЗУЛЬТАТ:

$ git tag -d oops

Deleted tag 'oops' (was 45fa96b)

$ git hist --all

\* fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

# Внесение изменений в коммиты

### Цели

* Научиться изменять существующие коммиты

## 01Измените страницу, а затем сделайте коммит

Добавьте в страницу комментарий автора.

#### ФАЙЛ: hello.html

**<!-- Author: Alexander Shvets -->**

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

#### ВЫПОЛНИТЕ:

git add hello.html

git commit -m "Add an author comment"

## 02Ой... необходим email

После совершения коммита вы понимаете, что любой хороший комментарий должен включать электронную почту автора. Обновите страницу hello, включив в нее email.

#### ФАЙЛ: hello.html

**<!-- Author: Alexander Shvets (alex@githowto.com) -->**

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

## 03Измените предыдущий коммит

Мы действительно не хотим создавать отдельный коммит только ради электронной почты. Давайте изменим предыдущий коммит, включив в него адрес электронной почты.

#### ВЫПОЛНИТЕ:

git add hello.html

git commit --amend -m "Add an author/email comment"

#### РЕЗУЛЬТАТ:

$ git add hello.html

$ git commit --amend -m "Add an author/email comment"

[master 6a78635] Add an author/email comment

1 files changed, 2 insertions(+), 1 deletions(-)

## 04Просмотр истории

#### ВЫПОЛНИТЕ:

git hist

#### РЕЗУЛЬТАТ:

$ git hist

\* 6a78635 2011-03-09 | Add an author/email comment (HEAD, master) [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email». Этого же эффекта можно достичь путем сброса последнего коммита в ветке, и повторного коммита новых изменений.

# Перемещение файлов

### Цели

* Научиться перемещать файл в пределах репозитория.

## 01Переместите файл hello.html в каталог lib

Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем страницу в каталог lib.

#### ВЫПОЛНИТЕ:

mkdir lib

git mv hello.html lib

git status

#### РЕЗУЛЬТАТ:

$ mkdir lib

$ git mv hello.html lib

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# renamed: hello.html -> lib/hello.html

#

Перемещая файлы с помощью git, мы информируем git о 2 вещах

1. Что файл hello.html был удален.
2. Что файл lib/hello.html был создан.

Оба эти факта сразу же проиндексированы и готовы к коммиту. Команда git status сообщает, что файл был перемещен.

## 02Второй способ перемещения файлов

Позитивной чертой git является то, что вы можете забыть о версионном контроле до того момента, когда вы готовы приступить к коммиту кода. Что бы случилось, если бы мы использовали командную строку операционной системы для перемещения файлов вместо команды git?

Оказывается, следующий набор команд идентичен нашим последним действиям. Работы здесь побольше, но результат тот же.

Мы могли бы выполнить:

mkdir lib

mv hello.html lib

git add lib/hello.html

git rm hello.html

## 03Коммит в новый каталог

Давайте сделаем коммит этого перемещения.

#### ВЫПОЛНИТЕ:

git commit -m "Moved hello.html to lib"

# Подробнее о структуре

### Цели

* Добавить еще один файл в наш репозиторий

## 01Добавление index.html

Давайте добавим файл index.html в наш репозиторий. Следующий файл отлично подойдет для этой цели.

#### ФАЙЛ: index.html

<html>

<body>

<iframe src="lib/hello.html" width="200" height="200" />

</body>

</html>

Добавьте файл и сделайте коммит.

#### ВЫПОЛНИТЕ:

git add index.html

git commit -m "Added index.html."

Теперь при открытии index.html, вы должны увидеть кусок страницы hello в маленьком окошке.

# Git внутри: Каталог .git

### Цели

* Узнать о структуре каталога .git

## 01Каталог .git

Настало время провести небольшое исследование. Для начала, из корневого каталога вашего проекта…

#### ВЫПОЛНИТЕ:

ls -C .git

#### РЕЗУЛЬТАТ:

$ ls -C .git

COMMIT\_EDITMSG MERGE\_RR config hooks info objects rr-cache

HEAD ORIG\_HEAD description index logs refs

Это магический каталог, в котором хранятся все «материалы» git. Давайте заглянем в каталог объектов.

## 02База данных объектов

#### ВЫПОЛНИТЕ:

ls -C .git/objects

#### РЕЗУЛЬТАТ:

$ ls -C .git/objects

09 24 28 45 59 6a 77 80 8c 97 af c4 e7 info

11 27 43 56 69 6b 78 84 91 9c b5 e4 fa pack

Вы должны увидеть кучу каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git.

## 03Углубляемся в базу данных объектов

#### ВЫПОЛНИТЕ:

ls -C .git/objects/<dir>

#### РЕЗУЛЬТАТ:

$ ls -C .git/objects/09

6b74c56bfc6b40e754fc0725b8c70b2038b91e 9fb6f9d3a104feb32fcac22354c4d0e8a182c1

Смотрим в один из каталогов с именем из 2 букв. Вы увидите файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Они сжаты и закодированы, поэтому просмотр их содержимого нам мало чем поможет. Рассмотрим далее каталог .git внимательно

## 04Config File

#### ВЫПОЛНИТЕ:

cat .git/config

#### РЕЗУЛЬТАТ:

$ cat .git/config

[core]

repositoryformatversion = 0

filemode = true

bare = false

logallrefupdates = true

ignorecase = true

[user]

name = Alexander Shvets

email = alex@githowto.com

Это файл конфигурации, создающийся для каждого конкретного проекта. Записи в этом файле будут перезаписывать записи в файле .gitconfig вашего главного каталога, по крайней мере в рамках этого проекта.

## 05Ветки и теги

#### ВЫПОЛНИТЕ:

ls .git/refs

ls .git/refs/heads

ls .git/refs/tags

cat .git/refs/tags/v1

#### РЕЗУЛЬТАТ:

$ ls .git/refs

heads

tags

$ ls .git/refs/heads

master

$ ls .git/refs/tags

v1

v1-beta

$ cat .git/refs/tags/v1

fa3c1411aa09441695a9e645d4371e8d749da1dc

Вы должны узнавать файлы в подкаталоге тегов. Каждый файл соответствует тегу, ранее созданному с помощью команды git tag. Его содержание – это всего лишь хэш коммита, привязанный к тегу.

Каталог heads практически аналогичен, но используется для веток, а не тегов. На данный момент у нас есть только одна ветка, так что все, что вы увидите в этом каталоге – это ветка master.

## 06Файл HEAD

#### ВЫПОЛНИТЕ:

cat .git/HEAD

#### РЕЗУЛЬТАТ:

$ cat .git/HEAD

ref: refs/heads/master

Файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка master.

# Git внутри: Работа непосредственно с объектами git

### Цели

* Исследовать структуру базы данных объектов
* Научиться использовать SHA1 хэши для поиска содержимого в репозитории

Давайте исследуем объекты git с помощью некоторых инструментов.

## 01Поиск последнего коммита

#### ВЫПОЛНИТЕ:

git hist --max-count=1

Эта команда должна показать последний коммит в репозиторий. SHA1 хэш в вашей системе, вероятно, отличается от моего, но вы увидите что-то наподобие этого.

#### РЕЗУЛЬТАТ:

$ git hist --max-count=1

\* 8029c07 2011-03-09 | Added index.html. (HEAD, master) [Alexander Shvets]

## 02Вывод последнего коммита

С помощью SHA1 хэша из коммита, указанного выше…

#### ВЫПОЛНИТЕ:

git cat-file -t <hash>

git cat-file -p <hash>

Вот что выходит у меня…

#### РЕЗУЛЬТАТ:

$ git cat-file -t 8029c07

commit

$ git cat-file -p 8029c07

tree 096b74c56bfc6b40e754fc0725b8c70b2038b91e

parent 567948ac55daa723807c0c16e34c76797efbcbed

author Alexander Shvets <alex@githowto.com> 1299684476 -0500

committer Alexander Shvets <alex@githowto.com> 1299684476 -0500

Added index.html.

**Примечание:** Если вы задали алиасы «type» и «dump», как описано в [уроке об алиасах](https://githowto.com/ru/aliases), можете вводить команды git type и git dump вместо длинных команд (которые я никогда не запоминаю).

Это вывод объекта коммита, который находится во главе ветки master.

## 03Поиск дерева

Мы можем вывести дерево каталогов, ссылка на который идет в коммите. Это должно быть описание файлов (верхнего уровня) в нашем проекте (для конкретного коммита). Используйте SHA1 хэш из строки «дерева», из списка выше.

#### ВЫПОЛНИТЕ:

git cat-file -p <treehash>

Вот как выглядит мое дерево…

#### РЕЗУЛЬТАТ:

$ git cat-file -p 096b74c

100644 blob 28e0e9d6ea7e25f35ec64a43f569b550e8386f90 index.html

040000 tree e46f374f5b36c6f02fb3e9e922b79044f754d795 lib

Да, я вижу index.html и каталог lib.

## 04Вывод каталога lib

#### ВЫПОЛНИТЕ:

git cat-file -p <libhash>

#### РЕЗУЛЬТАТ:

$ git cat-file -p e46f374

100644 blob c45f26b6fdc7db6ba779fc4c385d9d24fc12cf72 hello.html

Существует файл hello.html.

## 05Вывод файла hello.html

#### ВЫПОЛНИТЕ:

git cat-file -p <hellohash>

#### РЕЗУЛЬТАТ:

$ git cat-file -p c45f26b

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

А вот и он. Мы вывели объекты коммитов, объекты деревьев и объекты блобов непосредственно из репозитория git. Это все, что есть – блобы, деревья и коммиты.

## 06Исследуйте самостоятельно

Исследуйте git репозиторий вручную самостоятельно. Смотрите, удастся ли вам найти оригинальный файл hello.html с самого первого коммита вручную по ссылкам SHA1 хэша в последнем коммите.

# Создание ветки

### Цели

* Научиться создавать локальную ветку в репозитории

Пора сделать наш hello world более выразительным. Так как это может занять некоторое время, лучше переместить эти изменения в отдельную ветку, чтобы изолировать их от изменений в ветке master.

## 01Создайте ветку

Давайте назовем нашу новую ветку «style».

#### ВЫПОЛНИТЕ:

git checkout -b style

git status

**Примечание:**git checkout -b <имяветки> является шорткатом для git branch <имяветки> за которым идет git checkout <имяветки>.

Обратите внимание, что команда git status сообщает о том, что вы находитесь в ветке «style».

## 02Добавьте файл стилей style.css

#### ВЫПОЛНИТЕ:

touch lib/style.css

#### ФАЙЛ: lib/style.css

h1 {

color: red;

}

#### ВЫПОЛНИТЕ:

git add lib/style.css

git commit -m "Added css stylesheet"

## 03Измените основную страницу

Обновите файл hello.html, чтобы использовать стили style.css.

#### ФАЙЛ: lib/hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

**<link type="text/css" rel="stylesheet" media="all" href="style.css" />**

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

#### ВЫПОЛНИТЕ:

git add lib/hello.html

git commit -m "Hello uses style.css"

## 04Измените index.html

Обновите файл index.html, чтобы он тоже использовал style.css

#### ФАЙЛ: index.html

<html>

**<head>**

**<link type="text/css" rel="stylesheet" media="all" href="lib/style.css" />**

**</head>**

<body>

<iframe src="lib/hello.html" width="200" height="200" />

</body>

</html>

#### ВЫПОЛНИТЕ:

git add index.html

git commit -m "Updated index.html"

## 05Далее

Теперь у нас есть новая ветка под названием **style** с 3 новыми коммитами. Далее мы узнаем, как осуществлять навигацию и переключаться между ветками.

# Навигация по веткам

### Цели

* Научиться перемещаться между ветками репозитория

Теперь в вашем проекте есть две ветки:

#### ВЫПОЛНИТЕ:

git hist --all

#### РЕЗУЛЬТАТ:

$ git hist --all

\* 07a2a46 2011-03-09 | Updated index.html (HEAD, style) [Alexander Shvets]

\* 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]

\* 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]

\* 8029c07 2011-03-09 | Added index.html. (master) [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

## 01Переключение на ветку Master

Просто используйте команду git checkout для переключения между ветками.

#### ВЫПОЛНИТЕ:

git checkout master

cat lib/hello.html

#### РЕЗУЛЬТАТ:

$ git checkout master

Switched to branch 'master'

$ cat lib/hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Сейчас мы находимся на ветке Master. Это заметно по тому, что файл hello.html не использует стили style.css.

## 02Вернемся к ветке «style».

#### ВЫПОЛНИТЕ:

git checkout style

cat lib/hello.html

#### РЕЗУЛЬТАТ:

$ git checkout style

Switched to branch 'style'

$ cat lib/hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Содержимое lib/hello.html подтверждает, что мы вернулись в ветку **style**.

# Изменения в ветке master

### Цели

* Научиться работать с несколькими ветками с различными (и, возможно, конфликтующими) изменениями.

Пока вы меняли ветку «style», кто-то решил обновить ветку master. Они добавили README.

## 01Создайте файл README в ветке master.

#### ВЫПОЛНИТЕ:

git checkout master

#### ФАЙЛ: README

This is the Hello World example from the git tutorial.

## 02Сделайте коммит изменений README в ветку master.

#### ВЫПОЛНИТЕ:

git add README

git commit -m "Added README"

# Просмотр отличающихся веток

### Цели

* Научиться просматривать отличающиеся ветки в репозитории.

## 01Просмотрите текущие ветки

Теперь у нас в репозитории есть две отличающиеся ветки. Используйте следующую лог-команду для просмотра веток и их отличий.

#### ВЫПОЛНИТЕ:

git hist --all

#### РЕЗУЛЬТАТ:

$ git hist --all

\* 6c0f848 2011-03-09 | Added README (HEAD, master) [Alexander Shvets]

| \* 07a2a46 2011-03-09 | Updated index.html (style) [Alexander Shvets]

| \* 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]

| \* 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]

|/

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Это наша первая возможность увидеть в действии --graph в git hist. Добавление опции --graph в git log вызывает построение дерева коммитов с помощью простых ASCII символов. Мы видим обе ветки (style и master), и то, что ветка master является текущей HEAD. Общим предшественником обеих веток является коммит «Added index.html».

Метка --all гарантированно означает, что мы видим все ветки. По умолчанию показывается только текущая ветка.

# Слияние

### Цели

* Научиться сливать две отличающиеся ветки для переноса изменений обратно в одну ветку.

## 01Слияние веток

Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем master с style.

#### ВЫПОЛНИТЕ:

git checkout style

git merge master

git hist --all

#### РЕЗУЛЬТАТ:

$ git checkout style

Switched to branch 'style'

$ git merge master

Merge made by recursive.

README | 1 +

1 files changed, 1 insertions(+), 0 deletions(-)

create mode 100644 README

$ git hist --all

\* 5813a3f 2011-03-09 | Merge branch 'master' into style (HEAD, style) [Alexander Shvets]

|\

| \* 6c0f848 2011-03-09 | Added README (master) [Alexander Shvets]

\* | 07a2a46 2011-03-09 | Updated index.html [Alexander Shvets]

\* | 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]

\* | 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]

|/

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Путем периодического слияния ветки master с веткой style вы можете переносить из master любые изменения и поддерживать совместимость изменений style с изменениями в основной ветке.

Однако, это делает графики коммитов действительно уродливыми. Позже мы рассмотрим возможность перебазирования, как альтернативы слиянию.

## 02Далее

Но что если изменения в ветке master конфликтуют с изменениями в style?

# Создание конфликта

### Цели

* Создание конфликтующих изменений в ветке master.

## 01Вернитесь в master и создайте конфликт

Вернитесь в ветку master и внесите следующие изменения:

git checkout master

#### ФАЙЛ: lib/hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

**<!-- no style -->**

</head>

<body>

<h1>Hello, World! **Life is great!**</h1>

</body>

</html>

#### ВЫПОЛНИТЕ:

git add lib/hello.html

git commit -m 'Life is great!'

**Внимание:** используйте для этого коммита одинарные кавычки, дабы избежать проблем с символом !. В bash он считается служебным.

## 02Просмотр веток

#### ВЫПОЛНИТЕ:

git hist --all

#### РЕЗУЛЬТАТ:

$ git hist --all

\* 454ec68 2011-03-09 | Life is great! (HEAD, master) [Alexander Shvets]

| \* 5813a3f 2011-03-09 | Merge branch 'master' into style (style) [Alexander Shvets]

| |\

| |/

|/|

\* | 6c0f848 2011-03-09 | Added README [Alexander Shvets]

| \* 07a2a46 2011-03-09 | Updated index.html [Alexander Shvets]

| \* 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]

| \* 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]

|/

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

После коммита «Added README» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с style.

## 03Далее

Последнее изменение в master конфликтует с некоторыми изменениями в style. На следующем шаге мы решим этот конфликт.

# Разрешение конфликтов

### Цели

* Научиться разрешать конфликты во время слияния

## 01Слияние master с веткой style

Теперь вернемся к ветке style и попытаемся объединить ее с новой веткой master.

#### ВЫПОЛНИТЕ:

git checkout style

git merge master

#### РЕЗУЛЬТАТ:

$ git checkout style

Switched to branch 'style'

$ git merge master

Auto-merging lib/hello.html

CONFLICT (content): Merge conflict in lib/hello.html

Automatic merge failed; fix conflicts and then commit the result.

Если вы откроете lib/hello.html, вы увидите:

#### ФАЙЛ: lib/hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<<<<<<< HEAD

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

=======

<!-- no style -->

>>>>>>> master

</head>

<body>

<h1>Hello,World! Life is great!</h1>

</body>

</html>

Первый раздел - версия во главе текущей ветки (style). Второй раздел - версия ветки master.

## 02Решение конфликта

Вам необходимо вручную разрешить конфликт. Внесите изменения в lib/hello.html для достижения следующего результата.

#### ФАЙЛ: lib/hello.html

<!-- Author: Alexander Shvets (alex@githowto.com) -->

<html>

<head>

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

</head>

<body>

<h1>Hello, World! Life is great!</h1>

</body>

</html>

## 03Сделайте коммит решения конфликта

#### ВЫПОЛНИТЕ:

git add lib/hello.html

git commit -m "Merged master fixed conflict."

#### РЕЗУЛЬТАТ:

$ git add lib/hello.html

$ git commit -m "Merged master fixed conflict."

Recorded resolution for 'lib/hello.html'.

[style 645c4e6] Merged master fixed conflict.

## 04Расширенные возможности слияния

Git не предоставляет никаких графических инструментов слияния, но будет с удовольствием работать с любыми сторонними инструментами слияния, которые вы хотите использовать ([обсуждение таких инструментов на StackOverflow](https://stackoverflow.com/questions/137102/whats-the-best-visual-merge-tool-for-git)).

# Перебазирование как альтернатива слиянию

### Цели

* Узнать различия между перебазированием и слиянием.

### Обсуждение

Давайте рассмотрим различия между слиянием и перебазированием. Для того, чтобы это сделать, нам нужно вернуться в репозиторий в момент до первого слияния, а затем повторить те же действия, но с использованием перебазирования вместо слияния.

Мы будем использовать команду reset для возврата веток к предыдущему состоянию.

# Сброс ветки master

### Цели

* Сбросить ветку master в точку до конфликтующего коммита.

## 01Сброс ветки master

Добавив интерактивный режим в ветку master, мы внесли изменения, конфликтующие с изменениями в ветке style. Давайте вернемся в ветке master в точку перед внесением конфликтующих изменений. Это позволяет нам продемонстрировать работу команды rebase, не беспокоясь о конфликтах.

#### ВЫПОЛНИТЕ:

git checkout master

git hist

#### РЕЗУЛЬТАТ:

$ git hist

\* 454ec68 2011-03-09 | Life is great! (HEAD, master) [Alexander Shvets]

\* 6c0f848 2011-03-09 | Added README [Alexander Shvets]

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README».

#### ВЫПОЛНИТЕ:

git reset --hard <hash>

git hist --all

Просмотрите лог. Он должен выглядеть, как будто репозиторий был перемотан назад во времени к точке до какого-либо слияния.

#### РЕЗУЛЬТАТ:

$ git reset --hard 6c0f848

$ git hist --all

\* 6c0f848 2011-03-09 | Added README (HEAD, master) [Alexander Shvets]

| \* 07a2a46 2011-03-09 | Updated index.html (style) [Alexander Shvets]

| \* 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]

| \* 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]

|/

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

# Перебазирование

### Цели

* Использовать команду rebase вместо команды merge.

Итак, мы вернулись в точку до первого слияния и хотим перенести изменения из ветки master в нашу ветку style.

На этот раз для переноса изменений из ветки master мы будем использовать команду rebase вместо слияния.

#### ВЫПОЛНИТЕ:

git checkout style

git rebase master

git hist

#### РЕЗУЛЬТАТ:

$ go style

Switched to branch 'style'

$

$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: Added css stylesheet

Applying: Hello uses style.css

Applying: Updated index.html

$

$ git hist

\* 6e6c76a 2011-03-09 | Updated index.html (HEAD, style) [Alexander Shvets]

\* 1436f13 2011-03-09 | Hello uses style.css [Alexander Shvets]

\* 59da9a7 2011-03-09 | Added css stylesheet [Alexander Shvets]

\* 6c0f848 2011-03-09 | Added README (master) [Alexander Shvets]

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

## 01Слияние VS перебазирование

Конечный результат перебазирования очень похож на результат слияния. Ветка style в настоящее время содержит все свои изменения, а также все изменения ветки master. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка master является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной.

## 02Когда использовать перебазирование, а когда слияние?

Не используйте перебазирование …

1. Если ветка является публичной и расшаренной. Переписывание общих веток будет мешать работе других членов команды.
2. Когда важна точная история коммитов ветки (так как команда rebase переписывает историю коммитов).

Учитывая приведенные выше рекомендации, я предпочитаю использовать rebase для кратковременных, локальных веток, а слияние для веток в публичном репозитории.

[35. Слияние в ветку master](https://githowto.com/ru/merging_back_to_master)

[33. Сброс ветки master](https://githowto.com/ru/resetting_the_master_branch)

# Слияние в ветку master

### Цели

* Мы поддерживали соответствие ветки style с веткой master (с помощью rebase), теперь давайте сольем изменения style в ветку master.

## 01Слияние style в master

#### ВЫПОЛНИТЕ:

git checkout master

git merge style

#### РЕЗУЛЬТАТ:

$ git checkout master

Switched to branch 'master'

$

$ git merge style

Updating 6c0f848..6e6c76a

Fast-forward

index.html | 2 +-

lib/style.css | 8 ++++++++

lib/hello.html | 6 ++++--

3 files changed, 13 insertions(+), 3 deletions(-)

create mode 100644 lib/style.css

Поскольку последний коммит ветки master прямо предшествует последнему коммиту ветки style, git может выполнить ускоренное слияние-перемотку. При быстрой перемотке вперед, git просто передвигает указатель вперед, таким образом указывая на тот же коммит, что и ветка style.

При быстрой перемотке конфликтов быть не может.

## 02Просмотрите логи

#### ВЫПОЛНИТЕ:

git hist

#### РЕЗУЛЬТАТ:

$ git hist

\* 6e6c76a 2011-03-09 | Updated index.html (HEAD, master, style) [Alexander Shvets]

\* 1436f13 2011-03-09 | Hello uses style.css [Alexander Shvets]

\* 59da9a7 2011-03-09 | Added css stylesheet [Alexander Shvets]

\* 6c0f848 2011-03-09 | Added README [Alexander Shvets]

\* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]

\* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]

\* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]

\* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]

\* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]

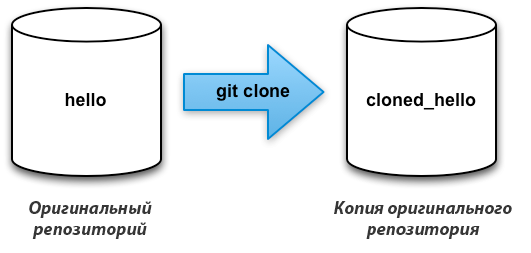
\* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]

\* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]

# Несколько репозиториев

До сих пор мы работали с одним git репозиторием. Однако, git удается отлично работать с несколькими репозиториями. Эти дополнительные репозитории могут храниться локально, или доступ к ним может осуществляться через сетевое подключение.

В следующем разделе мы создадим новый репозиторий с именем «cloned\_hello». Мы покажем, как перемещать изменения из одного репозитория в другой и как разрешать конфликты, возникающие в результате работы с двумя репозиториями.



А пока что поработаем с локальными репозиториями (т.е. репозиториями, хранящимися на вашем локальном жестком диске), однако практически все, что вы узнаете в этом разделе, будет применяться к нескольким репозиториям, несмотря на то, хранятся ли они локально или являются публичными.

**Примечание:** Мы будем вносить изменения в обе копии наших репозиториев. Обращайте внимание на то, в каком репозитории вы находитесь на каждом шаге в следующих уроках.

[37. Клонирование репозиториев](https://githowto.com/ru/cloningrepositories)

[35. Слияние в ветку master](https://githowto.com/ru/merging_back_to_master)

# «Гитхаб»: повторение

С «Гитхабом» вы уже на «ты»: умеете вести историю коммитов и основные команды, а ещё настраивать связь между локальным и удалённым репозиторием.

На этом его возможности не заканчиваются. Перед началом работы над дипломом мы посвятим несколько уроков дополнительному интерфейсу: научимся отправлять запрос на изменение из одной ветки в другую через Pull request. Это потребуется для дипломной работы и пригодится на практике.

Сначала давайте проверим, хорошо ли вы помните предыдущие темы..

Какая команда сопоставляет два коммита?



git diff и хеш двух коммитов.

Всё верно.



git diff --committed.

Как перенести изменения из одной ветки в другую?



Переключиться в ветку слияния и ввести git merge название ветки.

Да, всё правильно!



Переключиться в сливаемую ветку и ввести git merge название ветки.

Какой командой можно спрятать изменения, а потом их вернуть?



Командой git stash спрятать изменения, а git stash pop вернуть их.

Всё верно, pop вернёт изменения.



Командой git stash спрятать изменения, а git stash list вернуть их.

А вот и нет, команда git stash list покажет список спрятанных изменений.

Как сделать коммит?



Вызвать git add -A.



Зафиксировать изменения командой git add -A. Вызвать git commit -m для создания коммита.

Всё верно!



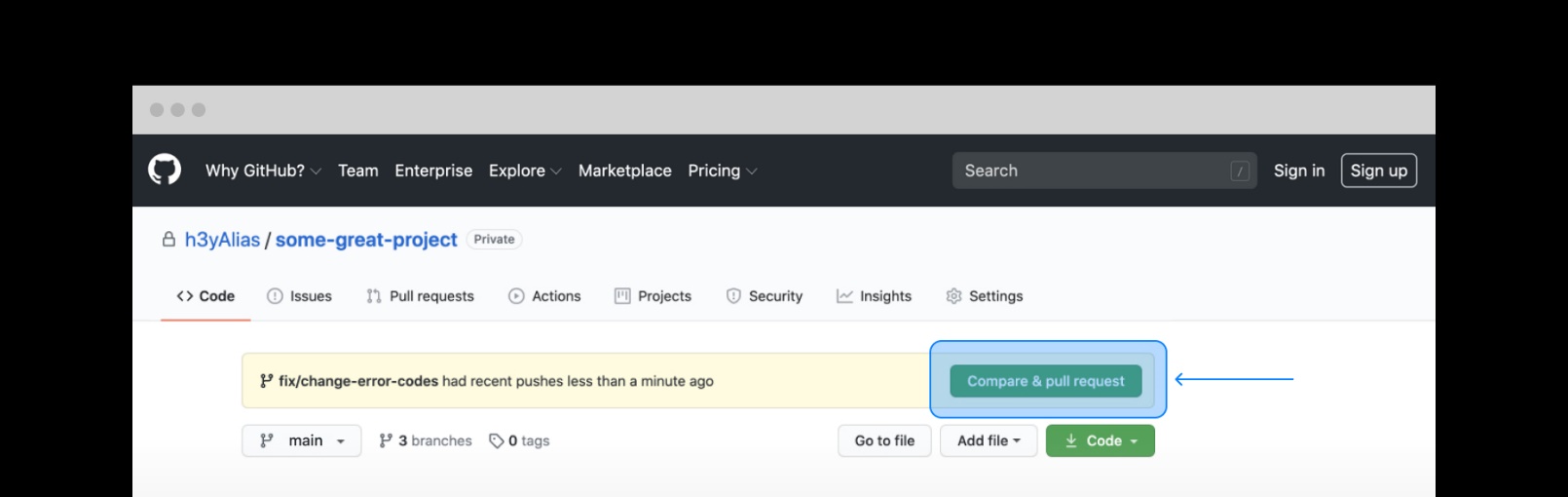
Выполнить git status.

## Pull request

Вы уже знаете про ветвление в Git. Ветки нужны, когда в проекте задействовано несколько человек: так у каждого будет своя версия кода, где можно работать и не беспокоиться об изменениях, которые вносят другие. Также работа в ветках, отличных от main, помогает сырому коду не попадать в «прод» — к конечному пользователю.

Представим, что вам нужно внести много изменений в проект. Первое, что вы делаете после клонирования, — создаёте новую ветку. Когда вы внесли кучу полезных изменений, дело за малым — загрузить обновления из вашего ответвления в ветку main репозитория. Для этого в «Гитхабе» существует Pull Request. Его удобно использовать в командной работе.

Когда «Гитхаб» замечает, что можно создать пул реквест из текущей ветки, появляется кнопка “Compare & pull request”:

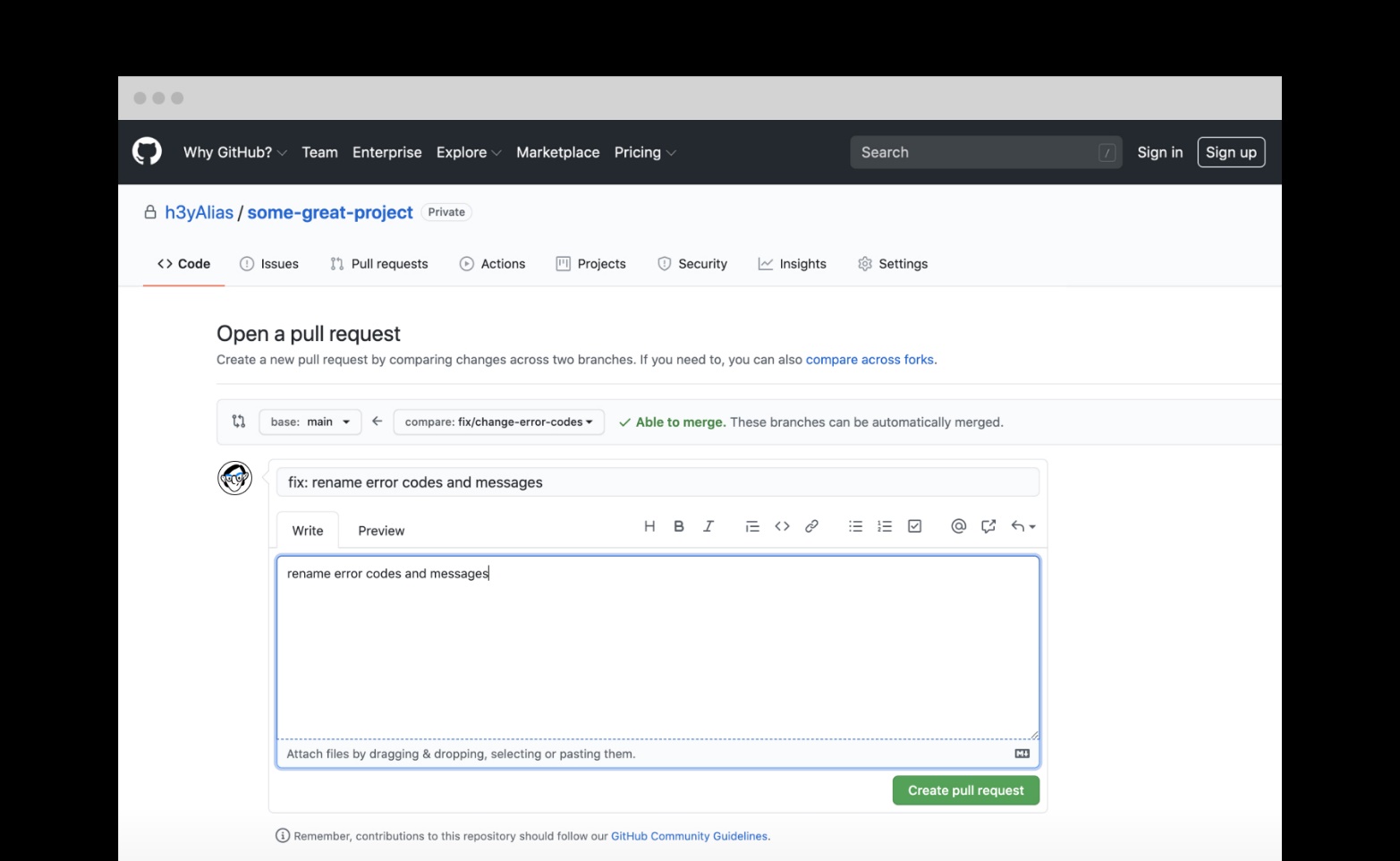


Вы на экране создания пул реквеста. В интерфейсе над кнопкой “Create pull request” — меню с ветками и репозиториями.

Дальше важно не перепутать: первая ветка отвечает за то, куда отправятся изменения, вторая — откуда (из какой ветки) эти изменения придут. «Направление изменений» указывает стрелка.

Вы можете открыть пул реквест между ветками внутри одного репозитория.

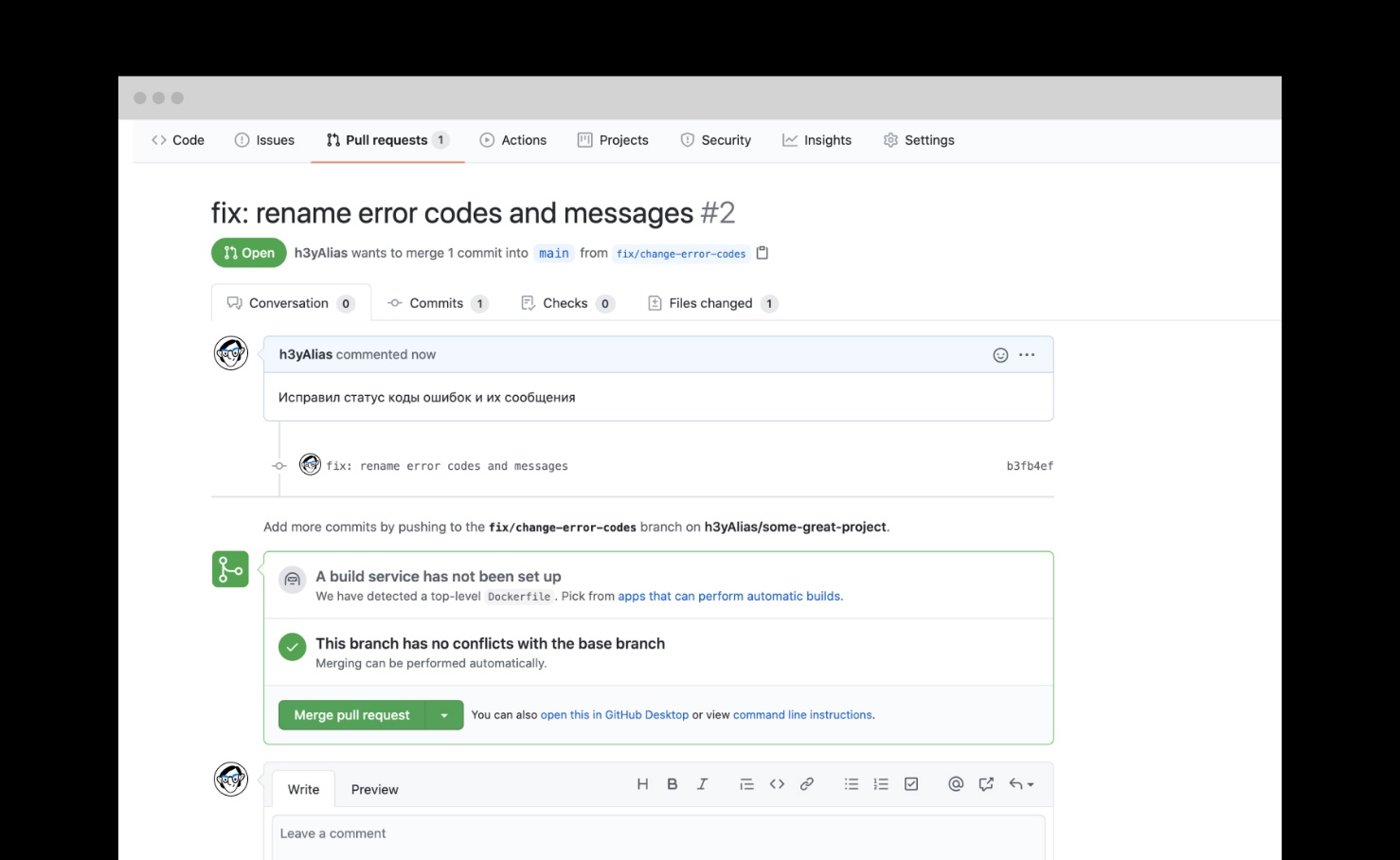
Здесь видно, какие файлы изменены, откуда и куда они попадут:



Создание pull request. В интерфейсе есть ветки, из которых открывают pull request, а также описание и заголовок для pull request

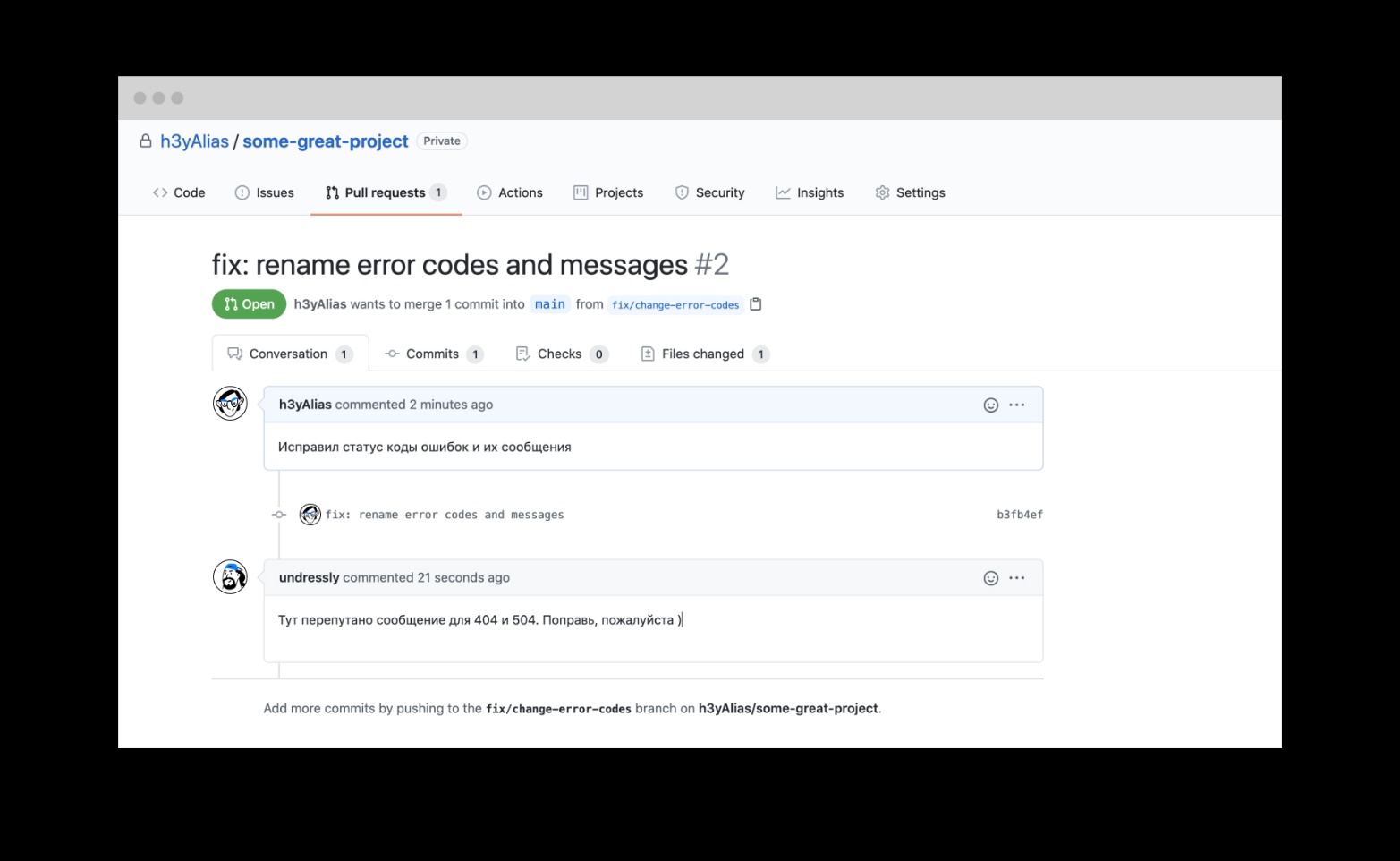
Пришло время зелёной кнопки: “Create pull request”. После не забудьте заполнить название изменения и кратко опишите, что именно вы сделали.

После нажатия кнопки “Create pull request” — вы создаёте пул реквест в репозитории. Во вкладке “Pull requests” видны обсуждения изменений, которые вы предложили:



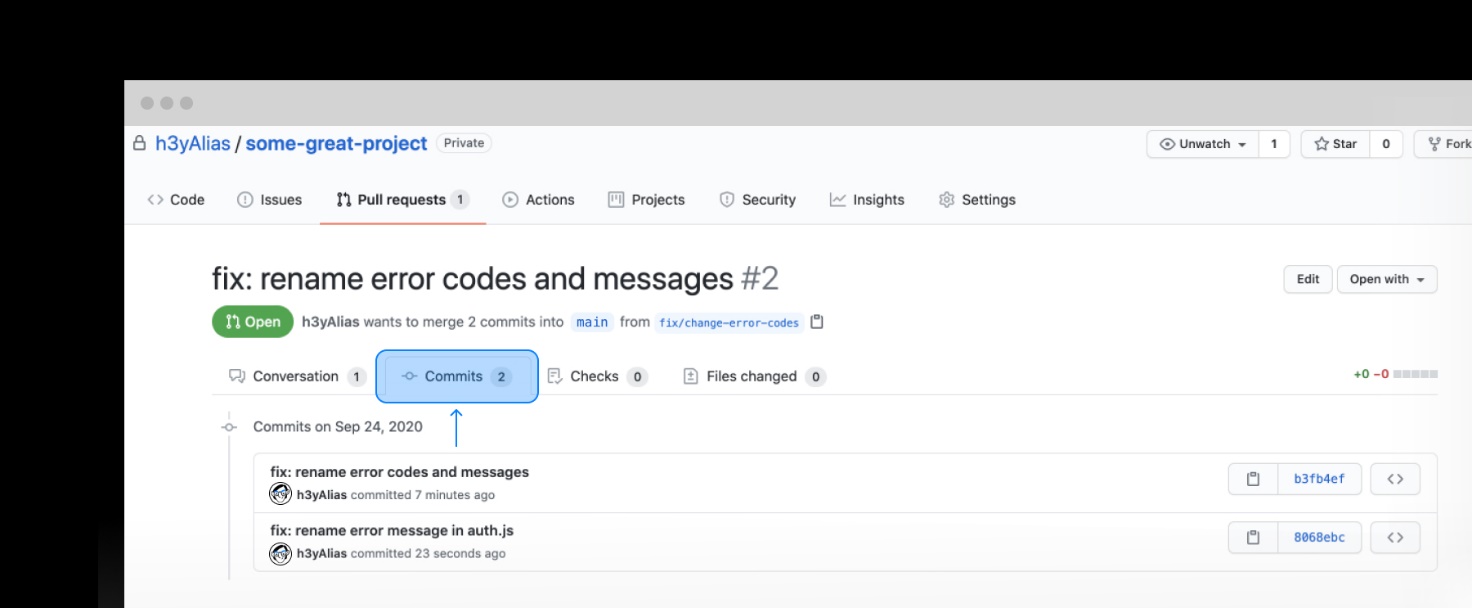
Обсуждение во вкладке “Pull requests”

Если тимлиду не понравится код, он может оставить комментарий:



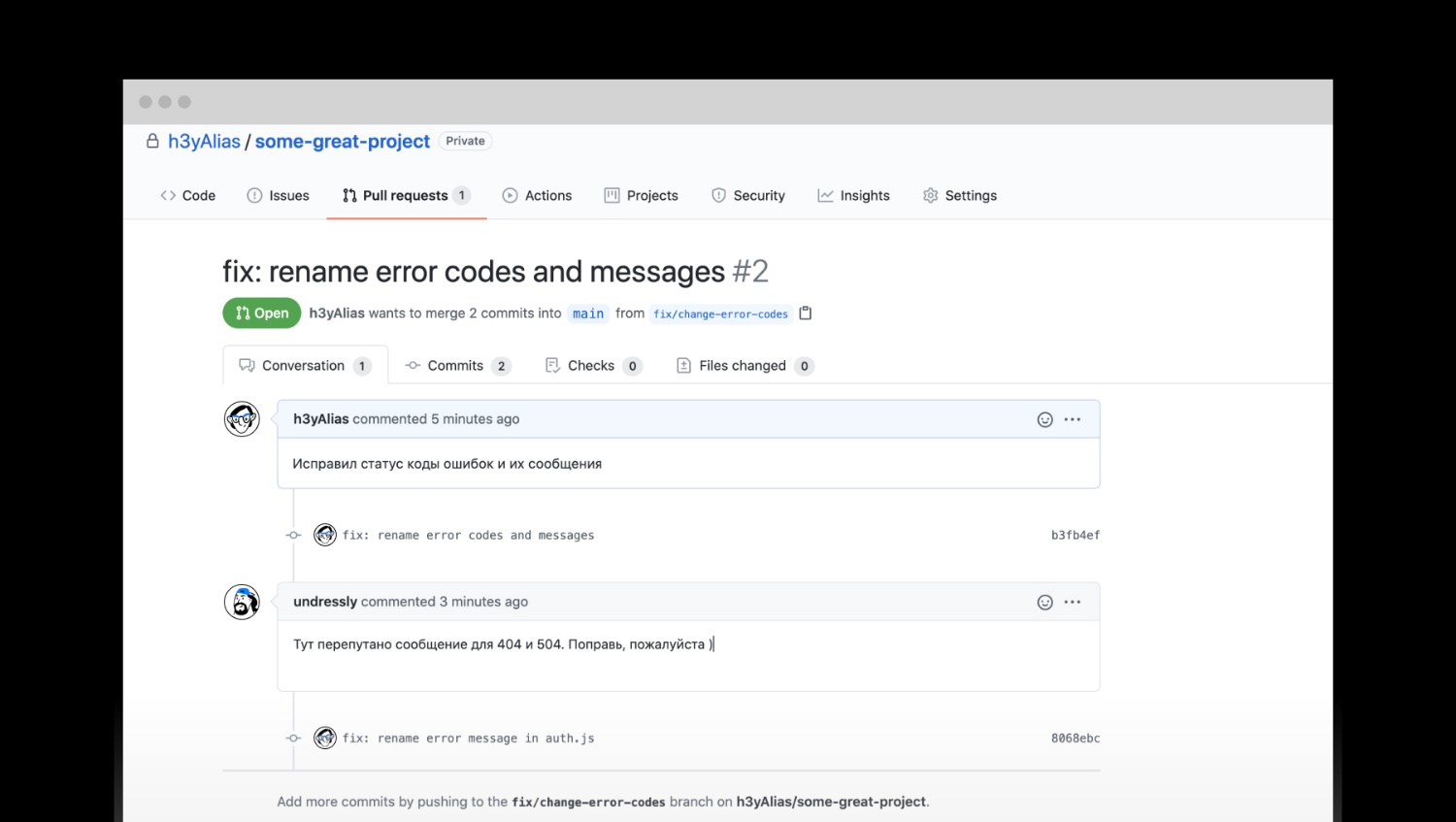
Кажется, код снова придётся дорабатывать...

Когда поправите код и сделаете коммит, изменения автоматически добавятся в пул реквест. Их можно посмотреть во вкладке “Commits”:



История коммитов во вкладке “Commits”

Новый коммит автоматически отобразится во вкладке “Conversation”:



Новые коммиты отображаются во вкладке “Conversation”

Когда код будет всех устраивать, тимлид (или другой ответственный человек) нажмёт заветную кнопку “Merge”, и изменения попадут в ветку main.

## Merge!

Резюмируем. Когда вы захотите внести изменения в основную ветку — воспользуйтесь пул реквестом.

В пул реквесте на «Гитхабе» удобно обсуждать предлагаемые изменения. Вот что [пишут](https://github.com/yandex/ym-helpers/pull/14) в комментариях разработчики.

У каждого пул реквеста есть:

* **Название** Краткое описание предлагаемых изменений. Например, «Адаптивный хедер сайта» или «Замена альбома на галерею».
* **Описание** В нём более развёрнуто, чем в названии, опишите, что именно сделали. Это опционально, но так же удобнее, правда?
* **Исходная ветка** В ней вы писали код.
* **Целевая ветка** В неё вы хотите внести изменения.

Пул реквест завершают двумя способами:

* Merge — изменения приняты, код вливается в целевую ветку, вы великолепны.
* Close — изменения не приняты, пул реквест закрывается, а вы всё равно великолепны.

С пул реквестами вы столкнётесь уже совсем скоро — при сдаче дипломной работы. Она будет разбита на несколько этапов, поэтому рекомендуем называть ветки соответствующим образом. Например:

* level-1 — при вёрстке и написании базовых компонентов на «Реакте»,
* level-2 — когда будете выполнять последний этап по добавлению основной функциональности в проект.

На этом изучение «Гитхаба» в Практикуме подошло к концу. Да пребудет с вами сила кнопки Merge! Смело переходите к описанию дипломного проекта — вы готовы.