1

Инициализируйте в папке репозиторий командой ***git init***

После её выполнения git создаст скрытую папку .git и поместит туда всё необходимое для дальнейшей работы.

Создайте новый файл. Заполните README.md любым интересным текстом.

echo "# Как пользоваться Git" > README.md

Давайте посмотрим на этот проект глазами Git. Выполните команду git status. Вот что вы увидите в консоли после этого:

On branch main

No commits yet

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

README.md

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

Git показал, в какой ветке находятся изменения (о ветках мы поговорим немного позднее) и вывел список найденных файлов. README.md помечен как «неотслеживаемый». Это значит, что он не добавлен в индекс: изменения в этом файле не будут записаны в следующий коммит. Коммитом называют набор изменений. Это та самая «точка сохранения», которая добавляется в историю.

Укажите Git, чтобы он следил за изменениями README.md:

git add README.md

Если захотите добавить сразу все файлы, то воспользуйтесь командой:

git add .

Однако, стоит помнить, что без грамотно оформленного файла .gitignore в коммит могут попасть «мусорные» файлы.

Осталось самое сложное: придумать название для коммита. Именно по ним вы и другие разработчики будете вспоминать причины внесения изменений

в код. В момент создания коммита вы достаточно погружены в задачу, чтобы написать дельное описание. Пользуйтесь этим!

Плохая история коммитов выглядит так:

573a2c6 - V.Petrov, 2 days ago : Добавил тесты
0153a95 - V.Petrov, 3 days ago : fix
b0e36e9 - V.Petrov, 3 days ago : Теперь точно починил баг
9e1d923 - V.Petrov, 4 days ago : Починил баг
7f22e9c - V.Petrov, 4 days ago : Изменил схему апи

Ужасная так:

0153a95 - V.Petrov, 3 days ago : other changes
b0e36e9 - V.Petrov, 3 days ago : ! it works
9e1d923 - V.Petrov, 4 days ago : some code
7f22e9c - V.Petrov, 4 days ago : refactoring
7f22e9c - V.Petrov, 4 days ago : fhoianjl

Хорошая история коммитов рассказывает о конкретных изменениях:

0153a95 - V.Petrov, 3 days ago : Исправлены грамматические ошибки в документации
b0e36e9 - V.Petrov, 3 days ago : Добавлена сортировка пользователей по дате регистрации
9e1d923 - V.Petrov, 4 days ago : Django обновлена до 3.1.1
7f22e9c - V.Petrov, 4 days ago : Увеличена минимальная длина пароля до 16 символов
7f22e9c - V.Petrov, 4 days ago : Добавлен модуль авторизации

Для создания коммита примените команду commit с параметром -m (message) и напишите короткое, но качественное описание.

git commit -m "Добавлен README.md"

В ответном сообщении Git напишет, какие изменения были зафиксированы.

[main (root-commit) 8b4f5c2] Добавлен README.md
1 file changed, 1 insertion(+)
create mode 100644 README.md

Сделайте ещё несколько изменений. Напишите что-то ещё в README.md или добавьте новые файлы, а потом зафиксируйте изменения.

echo '* Перед коммитом изменений нужно убедиться, что все файлы добавлены в индекс.' >> commit_help.md
echo '* Посмотреть состояние индекса можно командой git status.' >>

commit_help.md
echo '* Используйте `git add filename` для добавления конкретного файла или git add . для добавления всех файлов' >> commit_help.md
echo '* Чтобы одной командой добавить все изменённые файлы и сделать коммит, выполните git commit -m “commit message"' >> commit_help.md
echo '* Новые файлы добавлены не будут' >> commit_help.md
echo '- [Как сделать новый коммит](./commit_help.md)' >> README.md

Сохраните изменения, используя git commit -m “сообщение коммита”

После того как вы добавили файлы в индекс и закоммитили их, ваша цепочка изменений увеличилась. Посмотрите на неё командой git log.

```
commit 72d968f94fc8a0b593a2a8e21a8b3946b7018403 (HEAD -> main)
Author: Guido van Rossum <guido@python.org>
Date:   Thu Jan 01 00:05:32 2020 +0300

    Добавлена информация о команде commit

commit e98adc2eeb56d59285c6cf3f66b0b85458c186ab
Author: Guido van Rossum <guido@python.org>
Date:   Thu Jan 01 00:00:14 2020 +0300

    Добавлен README.md
```

Стандартный вывод команды log содержит полный хеш коммита, данные автора, дату и описание изменений. У команды log широкий набор инструментов для форматирования — вы можете ознакомиться с документацией.

3

Чтобы постоянно не печатать длинную команду, вы можете создать для неё короткое имя — alias. Для этого внесите изменения в файл .git/config.

```
[alias]
    history = log --graph --oneline --decorate
```

В этом файле хранятся настройки Git. Всего таких файлов три:

- /etc/gitconfig — содержит общие для всей системы настройки.
- ~/.gitconfig — содержит настройки для вашего пользователя.

- .git/config — находится в папке репозитория. В нём указаны настройки конкретного проекта.

Если в нескольких файлах будут по-разному настроены одинаковые параметры, то будут использоваться те, чья область применения меньше всего. То есть ваш .git/config важнее, чем ~/.gitconfig, а они оба перекрывают /etc/gitconfig.

Посмотреть все настройки, которые применены к проекту, можно командой git config --list --show-origin.

Теперь в вашем Git есть команда git history, делающая то же самое, что и git log --graph --oneline --decorate.

4

Чтобы повернуть время вспять и вернуть состояние проекта к любому коммиту, используйте reset.

Для практики немножко “накосячим”. Сделайте ещё один коммит:

echo '* Список всех коммитов показывает команда `git log`' >> log_help.md
echo '- [Как просмотреть историю](./log_help.md)' >> README.md
echo >> reset_help.md

git add .
git commit -m "Добавлена информация про log и файл для информации про reset"

Погодите! Если в сообщении коммита есть союз «и», значит изменений в нём слишком много. Давайте вернёмся к предыдущему коммиту и сделаем новый, включив в него только log_help.md и README.md.

Команда git reset откатывает все изменения в текущей ветке до указанной версии коммита. Так можно делать только в ветке, с которой работаете только вы и никто другой. При применении команды переписывается история ветки.

При ресете передаётся один из трёх ключей:

- --soft — откатывает изменения до указанного комита. При этом изменения остаются в индексе, будто вы сделали git add, но не закоммитили их.

- --mixed (стоит по умолчанию) — аналогичен варианту выше, но изменения уже не будут отслеживаться. Если после ресета выполнить команду git status, то Git предложит добавить изменения командой git add.
- --hard — как видно из названия, это самый жёсткий вариант. Он полностью откатывает изменения и заменяет данные в рабочей директории. Все закоммиченные и незакоммиченные изменения удаляются.

Чтобы перейти к состоянию предыдущего коммита, выполните команду

git reset --soft HEAD~1

HEAD — это указатель на коммит, на которым вы находитесь. HEAD~1 возвращает хеш предыдущего коммита. То же самое вы можете сделать, скопировав нужный хеш из команды git log: git reset --soft abc123def456.

Если вы заглянете в git log, то увидите, что ваш последний коммит пропал из списка. А git status покажет, что все файлы остались в индексе. Нужно убрать оттуда пустой файл reset_help.md. Для этого пригодится команда git rm с параметром --cached — удалить из индекса, но оставить в файловой системе.

git rm --cached reset_help.md

5

Чтобы просмотреть разницу между индексом и последним коммитом, выполните команду git diff.

В результате вы получите вариант, похожий на этот:

diff --git a/README.md b/README.md
index ed22cde..0d0e9b2 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,3 @@
 # Как пользоваться Git
- [Как сделать новый коммит](./commit_help.md)
+- [Как просмотреть историю](./log_help.md)
diff --git a/log_help.md b/log_help.md
new file mode 100644
index 0000000..de044e0

--- /dev/null
+++ b/log_help.md
@@ -0,0 +1 @@
+ Список всех коммитов показывает команда `git log`*
diff --git a/reset_help.md b/reset_help.md

Знак - напротив строки означает, что её удалили. Знак + означает, что строку добавили. Если строчка была просто изменена, то её старая версия отобразится со знаком -, а новая — со знаком +. Этот подход нужен, когда под рукой нет IDE или merge-инструмента.

Если нужно сравнить два конкретных коммита, то команда будет выглядеть так:

git diff cf5c7edcd8de031fa8dfd28fa893ef2832bdf280 35d5ce7b29fbba8caf22e090f2cf74710fe9cc01

Названия коммитов можно посмотреть, выполнив команду git log.

Сделайте два коммита: один с файлами README.md и log_help, а второй — с reset_help.md. Теперь ваша история должна выглядеть так:

c9067c0 (HEAD -> main) Добавлен файл reset_help.md
facb9fb Добавлена информация о команде log
7374bd1 Добавлена информация о команде commit
0647209 Добавлен README.md

Только что вы исправили небольшой беспорядок, применив git reset и git commit. Теперь стоит рассказать, что изменить последний коммит можно гораздо проще. Давайте дополним файл reset_help.md:

echo '* Команда `reset` откатывает проект к выбранному коммиту' >> reset_help.md
echo '- [Как вернуться к прошлой версии](./reset_help.md)' >> README.md

Вам нужно добавить эти изменения в предыдущий коммит и дать ему новое название. Вы можете сделать это, использовав параметр --amend.

git add .
git commit --amend -m "Добавлена информация о команде reset"

Этот параметр означает, что актуальный индекс будет применён к последнему коммиту. С её помощью вы можете:

- исправить опечатку в названии последнего коммита;
- добавить или убрать файлы из коммита.

Теперь ваша история выглядит так:

d3aff95 (HEAD -> main) Добавлена информация о команде reset
facb9fb Добавлена информация о команде log
7374bd1 Добавлена информация о команде commit
0647209 Добавлен README.md

7

В каждом проекте есть файлы, которые нет смысла хранить в истории. Если вы пользуетесь Pycharm, это будет папка с вашими настройками IDE (.idea). При использовании виртуального окружения вам не стоит добавлять в Git директорию, в которую установлен Python и его зависимости (обычно она называется venv). Интерпретатор python наверняка будет создавать папку pycache. Байт-код в .pyc-, .pyd- и .pyo-файлах, файлы логов, отчёты о результатах тестов и прочие производственные артефакты не важны для работы проекта, поэтому нужно спрятать их от всевидящего ока Git.

Для этого заведите в корне проекта файл .gitignore. В нём вы можете указать конкретный файл или glob-выражение.

```
ignore_me.txt       # Игнорировать файл
app/ignored/        # Игнорировать папку
*.py[cdo]           # Игнорировать все файлы с расширением pyc/pyd/pyo
app/**/*.log        # Игнорировать log-файлы в app- и поддиректориях
!app/main.log       # ... кроме файла main.log
```

В разных языках и разных проектах набор правил будет отличаться.

8

Чтобы привязать удаленный репозиторий, выполните следующие команды:

Указываем Git-сервер для репозитория
git remote add origin ссылка

Отправляем изменения в ветку main
git push origin main

Обновите страничку в GitHub и убедитесь, что изменения попали в репозиторий.

9

Пока вы работаете в одиночестве, для планирования задач вам хватит любого текстового редактора: выписываете туда список дел, добавляете файл в .gitignore и периодически сверяетесь с планом, вычёркивая завершённые задачи. Когда в вашем проекте начнётся командная работа, вам и вашим коллегам захочется знать, кто над чем сейчас работает и на каком этапе сейчас задача.

В GitHub есть удобный инструмент для визуализации задач - Projects. Сначала вам нужно создать проект в вашем репозитории. GitHub предоставляет несколько готовых шаблонов. Выберите Board, чтобы получить полностью настроенную доску.

После этого вы получите доску для размещения задач с 3-5 колонками по умолчанию:

- To do — задачи, за которые ещё никто не взялся.
- In progress — задачи в работе.
- Review in progress — задачи, ожидающие код-ревью.
- Reviewer approved — задачи, прошедшие код-ревью.
- Done — выполненные задачи.

Теперь перейдите на вкладку Issues и добавьте актуальную проблему: у вас в проекте до сих пор нет информации о ветвлении в Git.

Обратите внимание на поля справа: вы можете назначить ответственного за решение проблемы, указать, к какому проекту она относится, и повесить метки, облегчающие поиск задач.

Если вы привязали задачу к проекту, в нём появится новая карточка.

Назначьте её на себя, передвиньте в колонку *In progress* и приступайте к выполнению.

10

Пока в вашем проекте довольно линейная история. Посмотреть его схему можно командой ***git log --graph***. Полоски слева — это дерево ветвлений. Сейчас ваша история больше похожа на фонарный столб. Давайте добавим ей ветку.

Все работы из примера проводились в ветке main. В ней хранится последняя стабильная версия приложения. Чтобы случайно не сломать мастер-ветку, для разработки новой функциональности или исправления ошибок создаются отдельные ветки. Такой подход называется git-flow. На курсе мы рассмотрим его облегчённую версию.

Для разработки новой функциональности в название новой ветки добавляют префикс feature/. Он нужен, чтобы отличать её от тех, где исправляют ошибки — к названию таких веток добавляется префикс fix/.

git checkout -b feature/branches

Команда checkout используется в Git, чтобы:

- переключиться на произвольную существующую ветку (git checkout branch-name);
- создать новую ветку от текущей (git checkout -b new-branch-name).

При команде checkout Git не даст переключиться на другую ветку, если в текущей есть незакоммиченные изменения. Если вы не хотите коммитить изменения, используйте команду git stash — она их спрячет. Позже изменения можно достать с помощью команд git pop или git apply.

Добавьте в ветку изменения и закоммитьте их.

echo "Команда checkout используется в Git, чтобы:" > branch_help.md
echo "* переключиться на произвольную существующую ветку (git checkout branch-name)" >> branch_help.md
echo "* создать новую ветку от текущей (git checkout -b new-branch-name)" >> branch_help.md
echo '- [Ветвление](./branch_help.md)' >> README.md

git add .
git commit -m "Добавлена информация о ветвлении"

Посмотреть список веток, которые доступны локально, можно командой git branch, а добавив параметр -a, вы увидите ещё и ветки в репозитории GitHub.

```
* feature/branches
  main
  remotes/origin/main
```

Теперь отправьте изменения на сервер.

git push origin feature/branches

Проверьте GitHub — там появилось сообщение о новой ветке с предложением создать pull-request. Соглашайтесь.

Pull-request — это не просто совмещение двух веток. Это способ показать ваши изменения коллегам, ознакомить их с кодом и получить дружелюбные комментарии и одобрение от тимлида.

После создания pull-request укажите задачу, к которой он относится.

Теперь ссылка на pull-request отображается на доске. Кстати, теперь ваша карточка вполне заслуживает переместиться в колонку Review in progress.

Теперь, чтобы завершить работу, вам нужно совместить изменения в ветке feature/branches и main. Пока что вы сами и постановщик задач, и исполнитель, и ревьюер. Но скоро это изменится — для выполнения заданий вы объединитесь с другими студентами. Ритуалы, которые вы проделываете с GitHub помогут скоординировать вашу работу.

Откройте pull-request и нажмите большую зелёную кнопку Merge pull-request. В описани добавьте текст “fixes #1” , если issue об отсутствии информации по ветвлению имеет такой номер. Это позволит закрыть issue автоматически, и также автоматически закрыть задачу в проекте в состояние done, если pull request будет выполнен. Не забудьте удалить ветку после завершения merge — для этого тоже есть отдельная кнопка :) После слияния вам больше незачем хранить изменения в отдельной ветке.

Все ваши изменения попали в main. Теперь они официальная часть проекта.

10.1

Merge также можно осуществить через консоль.

Для начала переключитесь на мастер-ветку и заберите изменения из GitHub.
git checkout main

Git напомнит вам, что между локальной и удалённой веткой есть различия:

Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
(use "git pull" to update your local branch)
git pull

А потом создайте новую ветку и внесите правки.

git checkout -b feature/merge
echo "Для слияния двух веток нужна команда merge" > merge_help.md
echo "[Слияние веток](./merge_help.md)" >> README.md

Не забывайте проверять статус индекса и добавлять в него файлы.

git status
git add merge_help.md README.md
git commit -m "Добавлена информация о merge"
git log

Теперь переключитесь на ветку, в которую хотите влить feature/merge и выполните слияние.

git checkout main

Теперь нужно слить изменения ветки feature/merge с основной веткой разработки.

git merge feature/merge

При желании вы можете отправить локальную мастер-ветку в GitHub командой git push origin branch-name. Пока вы работаете один и полностью контролируете код это допустимо. При командной разработке все задачи принято выполнять в отдельных ветках и вливать изменения через pull-request, а main заблокирован для прямого push.

Команда **git revert** создаёт новые коммиты, в которых откачены старые изменения. Команду revert обычно используют в публичных ветках — тех, в

которых ведут работу несколько человек. Например, её применяют, если кто-то записал в мастер-ветку неработающий код. Также её можно использовать и для своих личных веток.

11

Бывают ситуации, когда один и тот же файл разработчики одновременно изменили в разных ветках, а при слиянии этих веток происходят конфликты. Потренируемся их создавать и ликвидировать.

Сейчас ваша история выглядит примерно так:

32a3cd9 (HEAD -> main, origin/main) Добавлена информация о merge
6ca257d Merge pull request #2 from username/feature/branches
16e30f8 Добавлена информация о ветвлении
add34c9 Добавлена информация о команде reset
2b6a10d Добавлена информация о команде log
689a900 Добавлена информация о команде commit
91d0e04 Добавлен README.md

Переключитесь на ветку feature/yoda и добавьте коммит.

git checkout -b feature/yoda main
echo "Always more questions than answers there are." >> quotes.md
git add quotes.md
git commit -m 'Цитата о вопросах и ответах'

Теперь перейдите в ветку human и сделайте похожие изменения.

git checkout -b human main
echo "There are always more questions than answers." >> quotes.md
git add quotes.md
git commit -m 'Цитата об ответах и вопросах'

И попробуйте вмёржить ветку feature/yoda.

git merge feature/yoda

Git выведет сообщение о конфликте...

CONFLICT (add/add): Merge conflict in quotes.md
Auto-merging quotes.md
Automatic merge failed; fix conflicts and then commit the result.

...и добавит в файлы пометки, где именно это произошло.

cat quotes.md

```
<<<<<<< HEAD
There are always more questions than answers.
=======
Always more questions than answers there are.
>>>>>>> feature/yoda
```

В файле появились непонятные дополнения:

- <<<<<<< HEAD;
- =======;
- >>>>>>> feature/yoda.

Эти строки можно назвать «разделителями конфликта». Строка ======= — «центр» конфликта. Всё содержимое между строкой <<<<<<< HEAD и ======= находится в текущей мастер-ветке. На неё и указывает ссылка HEAD. А всё между центром и строкой >>>>>>> feature/yoda — содержимое той ветки, с которой происходит слияние.

Попробуйте решить конфликты самостоятельно и закоммитьте изменения.

Полюбуйтесь на ваше ветвистое дерево коммитов.

git log --oneline --graph

У вас должно получиться что-то вроде этого:

```
*   61b5884 (HEAD -> human) Merge yoda to human
|\
| * 8efe6b8 (feature/yoda) Цитата о вопросах и ответах
* | cf721a5 Цитата об ответах и вопросах
|/
* 32a3cd9 (main, feature/merge, feature/branches) Добавлена информация
о merge
*   6ca257d (origin/main) Merge pull request #2 from
username/feature/branches
|\
| * 16e30f8 (origin/feature/branches) Добавлена информация о ветвлении
|/
* add34c9 Добавлена информация о команде reset
* 2b6a10d Добавлена информация о команде log
```

```
* 689a900 Добавлена информация о команде commit
* 91d0e04 Добавлен README.md
```

12 (*)

rebase создаёт более чистую историю коммитов. Рассмотрим на примере команды pull.

Переключитесь на новую ветку, начав её от первого коммита в репозитории. Подставьте свой hash, посмотрев его в git log.

git checkout -b feature/rebase 91d0e04

Создайте и добавьте новый файл.

echo >> empty_file.txt
git add .
git commit -m "Добавил пустой файл про запас"

```
* 003bf78 (HEAD -> feature/rebase) Добавил пустой файл про запас
* 91d0e04 Добавлен README.md
```

Выполните ***git pull origin main*** и посмотрите в историю коммитов ***git log --graph --oneline --decorate.***

```
*   23e23e7 (HEAD -> feature/rebase) Merge branch 'main' of github.com:username/git_intro into feature/rebase
|\
| *   003bf78 (origin/main) Merge pull request #2 from username/feature/branches
| |\
| | * 16e30f8 (origin/feature/branches) Добавлена информация о ветвлении
| |/
| * add34c9 Добавлена информация о команде reset
| * 2b6a10d Добавлена информация о команде log
| * 689a900 Добавлена информация о команде commit
* | eec8dae Добавил пустой файл про запас
|/
* 91d0e04 Добавлен README.md
```

Мастер-ветка вмёржилась в feature/rebase, создан автоматический коммит о слиянии.

Теперь вернитесь в исходное состояние ветки.

git reset --soft 003bf78
git stash
git reset --hard 91d0e04
git stash pop
git commit -m "Добавил пустой файл про запас"

В такой последовательности команд сначала откатывается первый коммит из ветки feature/rebase с сохранением нового файла empty_file.txt. Этот файл будет помечен как новый, но не закоммиченный. Затем empty_file.txt помещается во временное хранилище stash. После этого изменения в ветке feature/rebase откатываются до изначального состояния. Таким образом ветка feature/rebase не будет отличаться от мастера (коммит 91d0e04). Далее новый файл достаётся обратно из stash. Такая последовательность действий позволяет изменить название коммита и не потерять изменения в файлах.

Выполните команду ***git pull --rebase origin main*** и сравните историю.

```
* bf9d8da (HEAD -> feature/rebase) Добавил пустой файл про запас
*   6ca257d (origin/main) Merge pull request #2 from
username/feature/branches
|\
| * 16e30f8 (origin/feature/branches) Добавлена информация о ветвлении
|/
* add34c9 Добавлена информация о команде reset
* 2b6a10d Добавлена информация о команде log
* 689a900 Добавлена информация о команде commit
* 91d0e04 Добавлен README.md
```

В отличие от merge, который выполняет слияние веток в отдельном коммите, rebase перемещает ваш коммит в самый верх ветки.

Rebase с изменением коммитов:
Ещё одно преимущество ребейза — одной командой все коммиты схлопываются в один. История становится более чистой, а коммиты с мелкими фиксами или исправлениями комментариев ревьюверов отсутствуют.

Для примера создайте пачку мелких изменений.

echo >> counter.md && git add . && git commit -m "init counter"
for i in {1..5}; do echo $i >> counter.md; git add . ; git commit -m $i; done

Загляните в лог.

```
a24359e (HEAD) 5
4139f15 4
16517e6 3
a64a3fa 2
fa732c5 1
8829b18 init counter
```

Запустите rebase в интерактивном режиме редактирования истории.

git rebase -i HEAD~6

Вы увидите список коммитов и инструкцию по применению rebase.

```
pick 8829b18 init counter
pick fa732c5 1
pick a64a3fa 2
pick 16517e6 3
pick 4139f15 4
pick a24359e 5

# Rebase 0eba08d..a24359e onto 0eba08d (6 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
```

Ваша задача — указать для каждого коммита необходимое изменение. Например, чтобы соединить коммиты со второго по пятый, нужно объявить их как squash.

```
pick 8829b18 init counter
pick fa732c5 1
```

```
squash a64a3fa 2
squash 16517e6 3
squash 4139f15 4
squash a24359e 5`
```

После нажатия ctrl+c (перейти в режим ввода команд в vim) и ввода :x (выйти и сохранить) Git предложит ввести сообщение для нового коммита. Очистите поле, последовательно нажав d (удалить) и G (перейти в конец файла), перейдите в режим ввода кнопкой i и введите нужный текст.

Проверьте историю после завершения rebase. Она стала гораздо чище!

```
3f446fb (HEAD -> feature/rebase) add numbers
8829b18 init counter
```

Будьте аккуратны!
Если вы меняете историю коммитов, используя rebase, ammend или reset в ветке, которая уже есть в origin, вы получите сообщение об ошибке. Git не сможет автоматически проассоциировать имеющиеся в origin коммиты с вашей изменённой версией. Чтобы решить проблему, вам придётся обновить ветку «силой», использовав push --force. К этому можно прибегать, только если вы работаете один в своей ветке, иначе после смены хэша коммитов конфликты получат все остальные разработчики.

Как откатить rebase, если его сделали не от нужной ветки?
Попробуйте разрулить следующую ситуацию: разработчики создали ветку и закоммитили в неё кучу изменений. Затем сделали ребейз от ветки feature/1. Прибегает тимлид и говорит, что надо было ребейзиться от ветки feature/2. Как быть?

Один из ключей к решению этой проблемы — команда git reflog.

Недостатки rebase
Из-за того, что ребейз меняет историю коммитов, можно потерять контекст, в котором их написали разработчики. Ещё теряется информация о слияниях веток: она нужна для определения причины ошибки. Помните об этом.

Заключение

- git commit --amend <commit message> — изменить сообщение последнего коммита.
- git stash — спрятать текущие изменения и откатить их. Удобно, если не хочется удалять изменения, но нужно откатить их на время ради переключения в другую ветку.
- git pull — получить изменения с сервера.
- git log --graph --oneline --decorate — отразить историю коммитов в виде графа.
- Фильтрация истории коммитов:
    - По дате:
        - git log --after="2020-4-1"
        - git log --after="yesterday"
        - git log --after="1 week ago"
        - git log --after="2020-4-1" --before="2014-4-6"
    - По автору:
        - git log --author="Alex" — найти коммиты Алекса.
        - git log --author="Alex\|Ivan" — найти коммиты Алекса и Ивана.
    - По другим параметрам:
        - git log --grep="ISSUE-777:" — по паттерну сообщения.
        - git log -- foo.py bar.py — по файлам, которые были изменены.
        - git log -S "git" — поиск по содержимому файлов. Например, когда в файлы была добавлена строчка git.
    - git log --no-merges — убрать из выборки мёрж-коммиты.
    - git log --merges — выбрать только мёрж-коммиты.