

# Методы и средства сборки и развертывания ПО, 25

ФИО преподавателя: Смирнов Михаил Вячеславович

e-mail: [smirnovmgupi@gmail.com](mailto:smirnovmgupi@gmail.com)

## Лекция 4

# Основы GIT. Ветвления GIT. Инструментарий GIT.

# Что такое Git?



**Git** - это распределенная система контроля версий (DVCS).

Ее главная задача - отслеживать изменения в файлах проекта over time.

В отличие от централизованных систем, у каждого разработчика есть полная копия репозитория со всей его историей. Это обеспечивает гибкость, скорость и надежность.

Git был создан Линусом Торвальдсом для разработки ядра Linux.

# Ключевые преимущества Git



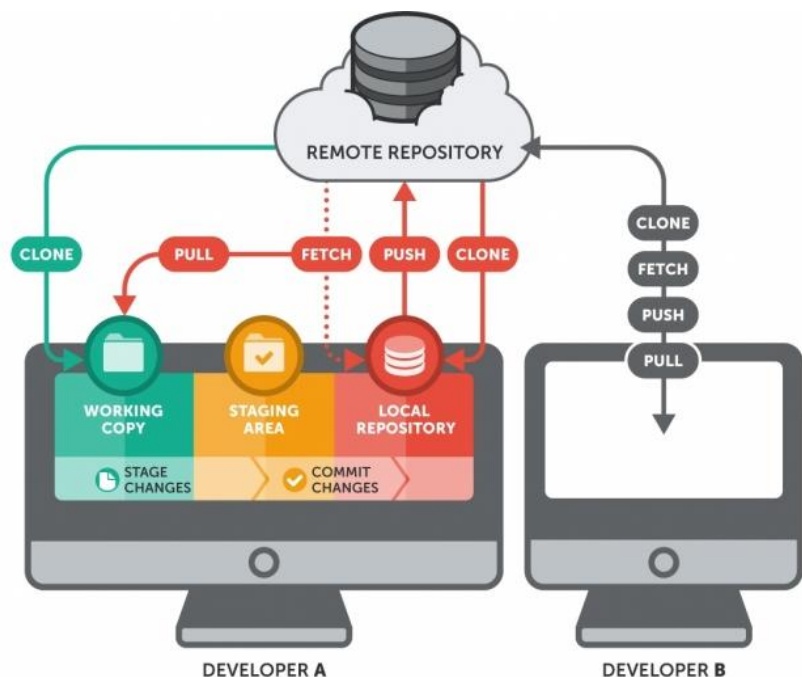
**Распределенность:** Каждая копия репозитория является полным бэкапом. Нет единой точки отказа.

**Скорость:** Операции в Git (кроме работы с сетью) выполняются локально и очень быстро.

**Ветвление и слияние:** Работа с ветками - это «визитная карточка» Git. Они легковесны и просты в использовании, что поощряет нелинейную разработку.

**Целостность данных:** Git использует хэшфункцию SHA-1 для всего. Любое изменение истории будет обнаружено.

# Основные понятия: Репозиторий



**Репозиторий (Repo)** - это виртуальное хранилище проекта. Он содержит все файлы проекта и папку `.git`, где хранится вся служебная информация: история коммитов, теги, ссылки на ветки и т.д.

**Локальный репозиторий:** Находится на локальном рабочем месте разработчика. Разработчик работает с ним напрямую, на своем компьютере.

**Удаленный репозиторий (Remote):** Находится на сервере (GitHub, GitLab, Bitbucket). Служит для обмена кодом и совместной работы.

# Основные понятия: Коммит

In case of fire



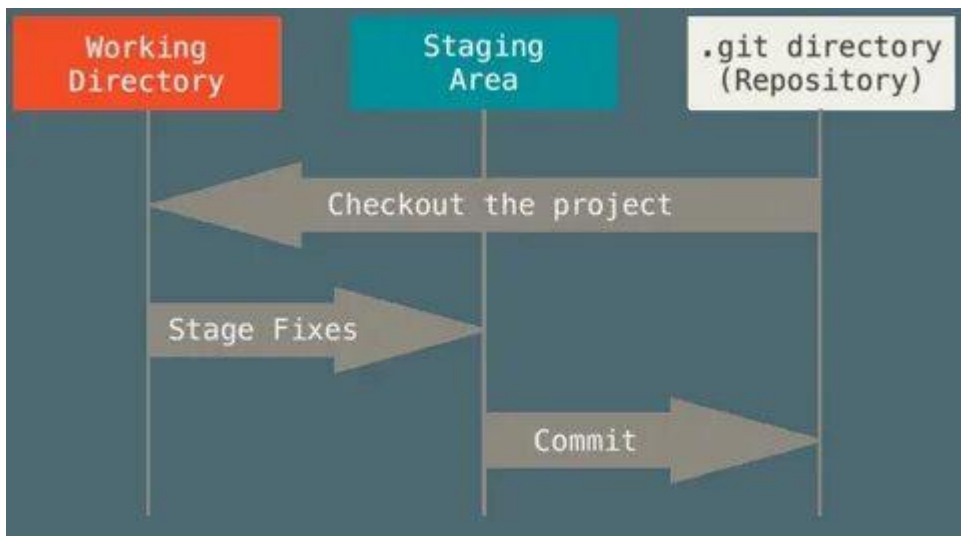
-  1. `git commit`
-  2. `git push`
-  3. `leave building`

**Коммит (Commit)** - это основная единица истории в Git. Его можно представить как снимок состояния всех файлов в проекте на определенный момент времени. Коммит неизменяем.

Каждый коммит содержит:

- **Уникальный хеш (SHA-1):** Идентификатор, например, a1b2c3d....
- **Автора и дату.**
- **Сообщение (Commit message):** Краткое и ясное описание внесенных изменений.
- **Ссылку на предыдущий коммит (родителя).**

# Процесс работы: Три дерева Git



1. **Working Directory (Рабочая копия).** Файлы на жестком диске разработчика. Он работает непосредственно с ними.
2. **Staging Area (Индекс):** Промежуточная область, куда разработчиком (*git add*) добавляются файлы, которые готовы к коммиту. Это позволяет формировать коммит именно из нужных изменений.
3. **Repository (Репозиторий):** Окончательное хранилище, куда сохраняются запланированные изменения (*git commit*).

# Базовые команды: git add и git commit



*git add <файлы> или git add.*

**Цель:** Перемещает изменения из рабочей директории в staging area. Говорят, что в этот момент ГИТ "индексирует" изменения).

**Аналогия:** Вы откладываете товары в корзину перед оформлением заказа.

*git commit -m «Передаваемое сообщение (метка)».*

**Цель:** Создает новый коммит из всех проиндексированных изменений.

**Аналогия:** Оформление и отправка заказа. Сообщение — это описание заказа.

# Базовые команды: git status и git log

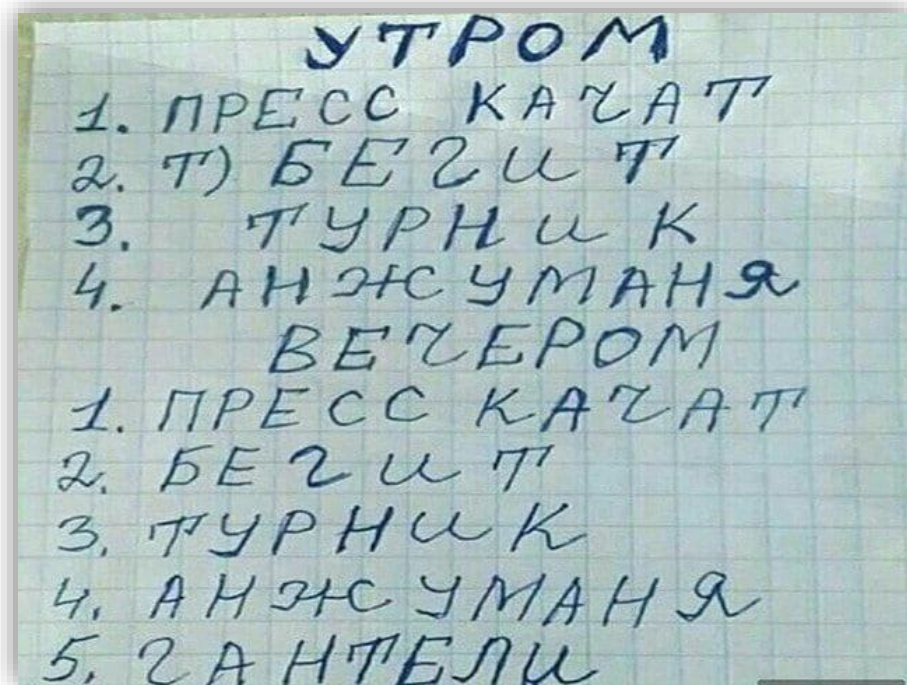
*git status*. Показывает текущее состояние рабочей директории и staging area:

какие файлы изменены, но не добавлены в индекс;

какие добавлены, но не закоммичены.

*git log*. Отображает историю коммитов в текущей ветке в обратном хронологическом порядке.

Показывает хеши, авторов, даты и сообщения.



# Базовые команды: git status и git log

```
$ git status
On branch branch-to-demo-mixed-reset
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    five.txt
    four.txt
    three.txt

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

# Базовые команды: git status и git log

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\aditi jain\GFG\GFG Articles\Customizing git log output> git log
commit 140c39d3f871d6ebf2ce45ae9d61ed5c2b51f1d5 (HEAD -> master, origin/master)
Author: Aditi Jain <jaditi930@gmail.com>
Date:   Sat Mar 2 08:43:15 2024 +0530

    added content to gfg.txt

commit 7360857a3c7dfc1770d6563146eb4f4daa615e84
Author: Aditi Jain <jaditi930@gmail.com>
Date:   Sat Mar 2 08:42:14 2024 +0530

    created gfg.txt
PS D:\aditi jain\GFG\GFG Articles\Customizing git log output> |
```

# Основные понятия: Ветвление



**Ветка (Branch)** - это указатель на определенный коммит. Создание новой ветки - это независимый путь для разработки.

*Работать над новой функцией, не затрагивая основной код (main/master).*

*Проверять всякое без риска.*

*Исправлять баги.*

# Работа с ветками: создание и переключение



*git branch <имя\_ветки>*

Создает новую ветку от текущего коммита. Сама команда branch только создает ветку, но не переключает на нее.

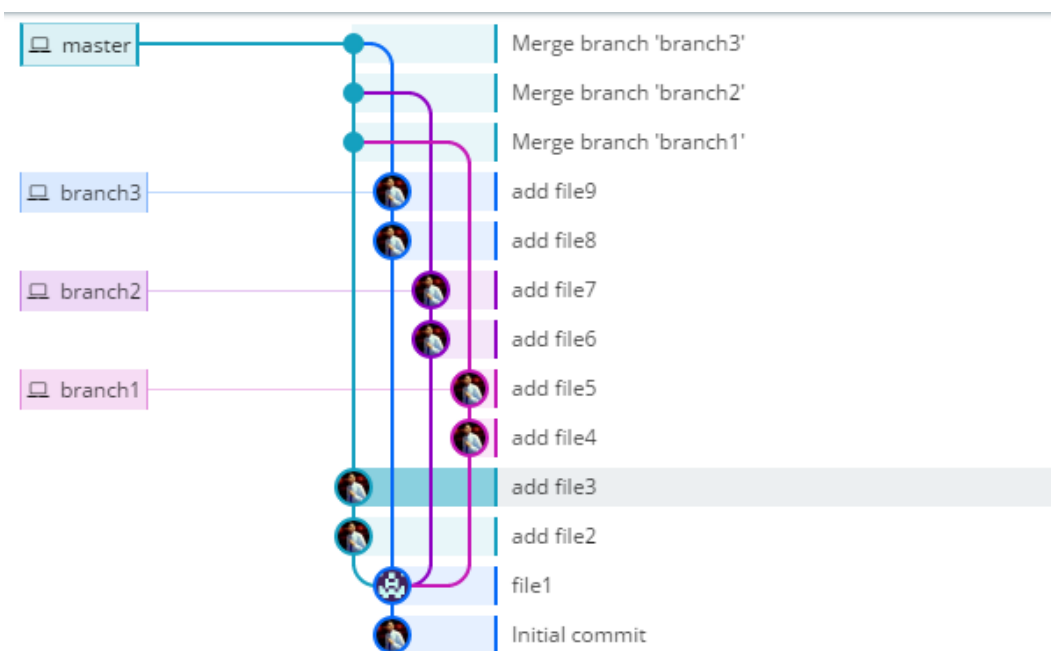
*git checkout <имя\_ветки>*

Переключает на указанную ветку. Файлы в рабочей директории приводятся в соответствие с состоянием этой ветки.

*git checkout -b <имя\_ветки>*

Комбинация двух команд: создает новую ветку и сразу переключается на нее.

# Работа с ветками: слияние



**Слияние (Merge)** - это процесс объединения истории двух веток.

1. Из ветки *master* выполняется команда *git merge new-feature*.

2. Git пытается объединить изменения из ветки *new-feature* в ветку *master*.

3. Если изменения были в разных частях файлов, Git создает **коммит слияния**, который имеет двух родителей.

# Работа с ветками: конфликт слияния



**Конфликт** возникает, когда Git не может автоматически объединить изменения, потому что в обеих ветках были изменены **одни и те же** строки в **одном и том же** файле.

Git помечает конфликтующие участки в файле специальными маркерами (<<<<<<, =====, >>>>>>).

**Задача разработчика:** Вручную отредактировать файл, оставив нужный код, и удалить маркеры.

После разрешения конфликта нужно повторно выполнить *git add* и *git commit*.

# Работа с ветками: конфликт слияния

```
colors.txt x
src > colors.txt
1  red
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2  <<<<<< HEAD (Current Change)
3  green
4  =====
5  white
6  >>>>>> his-branch (Incoming Change)
7  blue

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

react-app-demo > my-branch git merge his-branch
Auto-merging src/colors.txt
CONFLICT (content): Merge conflict in src/colors.txt
Automatic merge failed; fix conflicts and then commit the result.
x react-app-demo > my-branch •+ >M<
```

# Работа с ветками: перебазирование



**Перебазирование (*git rebase*)** - это альтернативный слиянию способ интеграции изменений. Вместо создания коммита слияния, *rebase* "перемещает" всю историю одной ветки на конец другой ветки, переписывая историю коммитов так, как если бы они разрабатывались последовательно.

**Плюс:** История становится более линейной и чистой.

**Минус:** Проблемы при *rebase* на публичные (общие) коммиты.

<https://habr.com/ru/companies/otus/articles/352640/>

# Работа с удаленным репозиторием: git push

*git push <remote\_name> <branch\_name>*

Отправляет локальные коммиты из указанной ветки на удаленный репозиторий.

*git push origin master*

*origin* - стандартное имя для удаленного репозитория, с которого вы клонировали проект.



# Работа с удаленным репозиторием: git pull

*git pull <remote\_name> <branch\_name>*

Забирает изменения из удаленной ветки и сливает их с текущей локальной веткой разработчика.

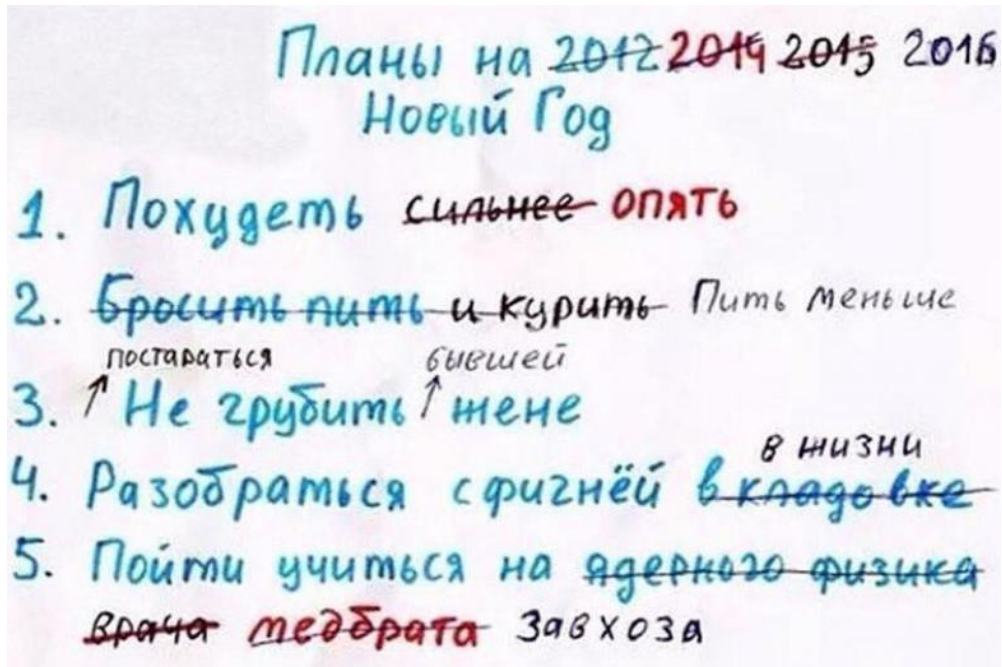
По сути, это модель из двух команд в одной:

- *git fetch* - загружает историю с удаленного репозитория.
- *git merge* - сливает эти изменения в текущую ветку локального репозитория.

*git pull* делается перед началом работы всегда, чтобы быть в курсе последних изменений.



# Работа с репозиторием: просмотр и отмена изменений



## *git diff*

Показывает разницу между рабочей директорией и последним коммитом (или staging area). Позволяет увидеть, что именно было изменено.

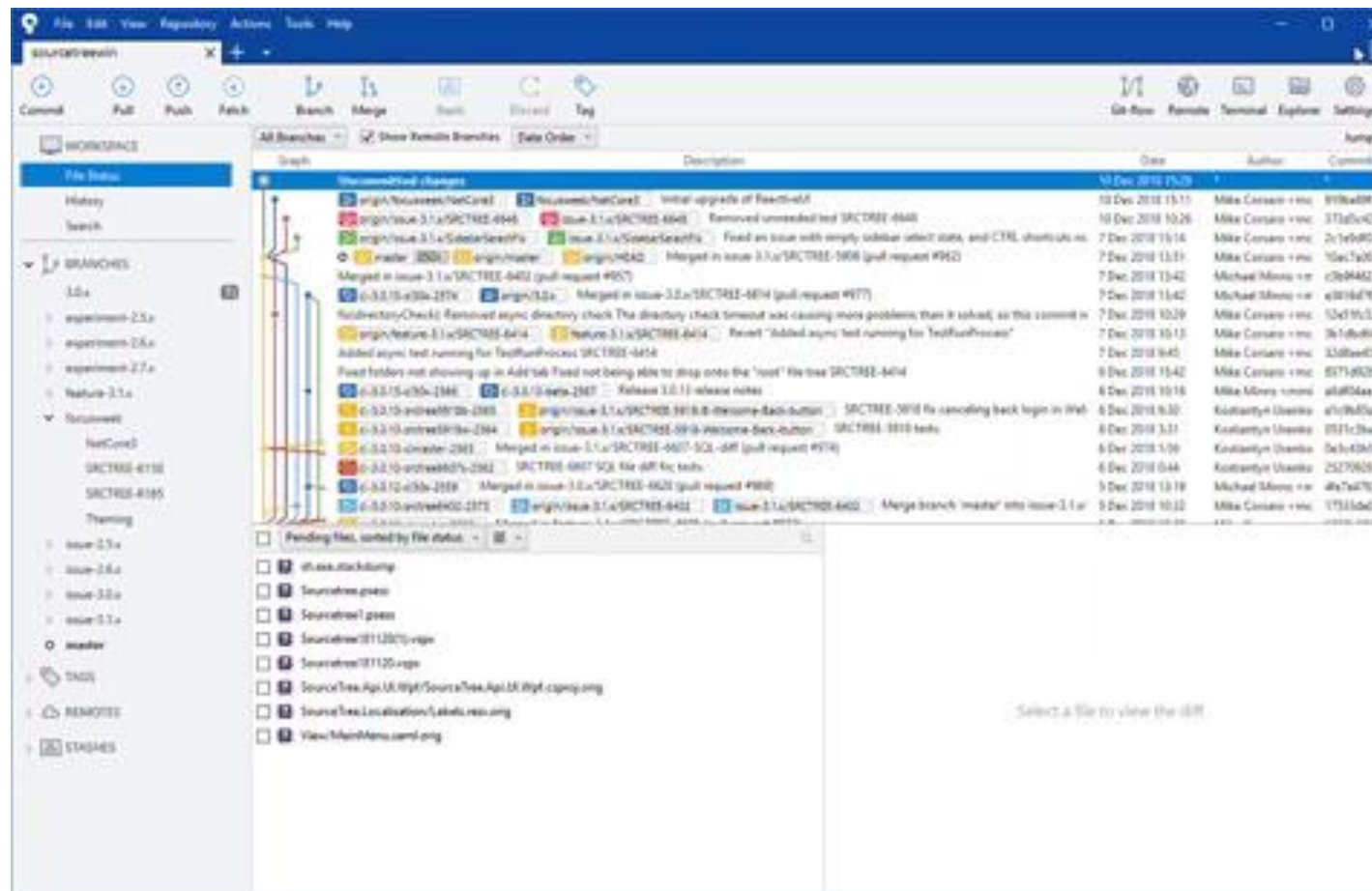
## *git restore <файл>*

Отменяет изменения в указанном файле в рабочей директории, возвращая его к состоянию последнего коммита (или состоянию в staging area, с опцией --staged).

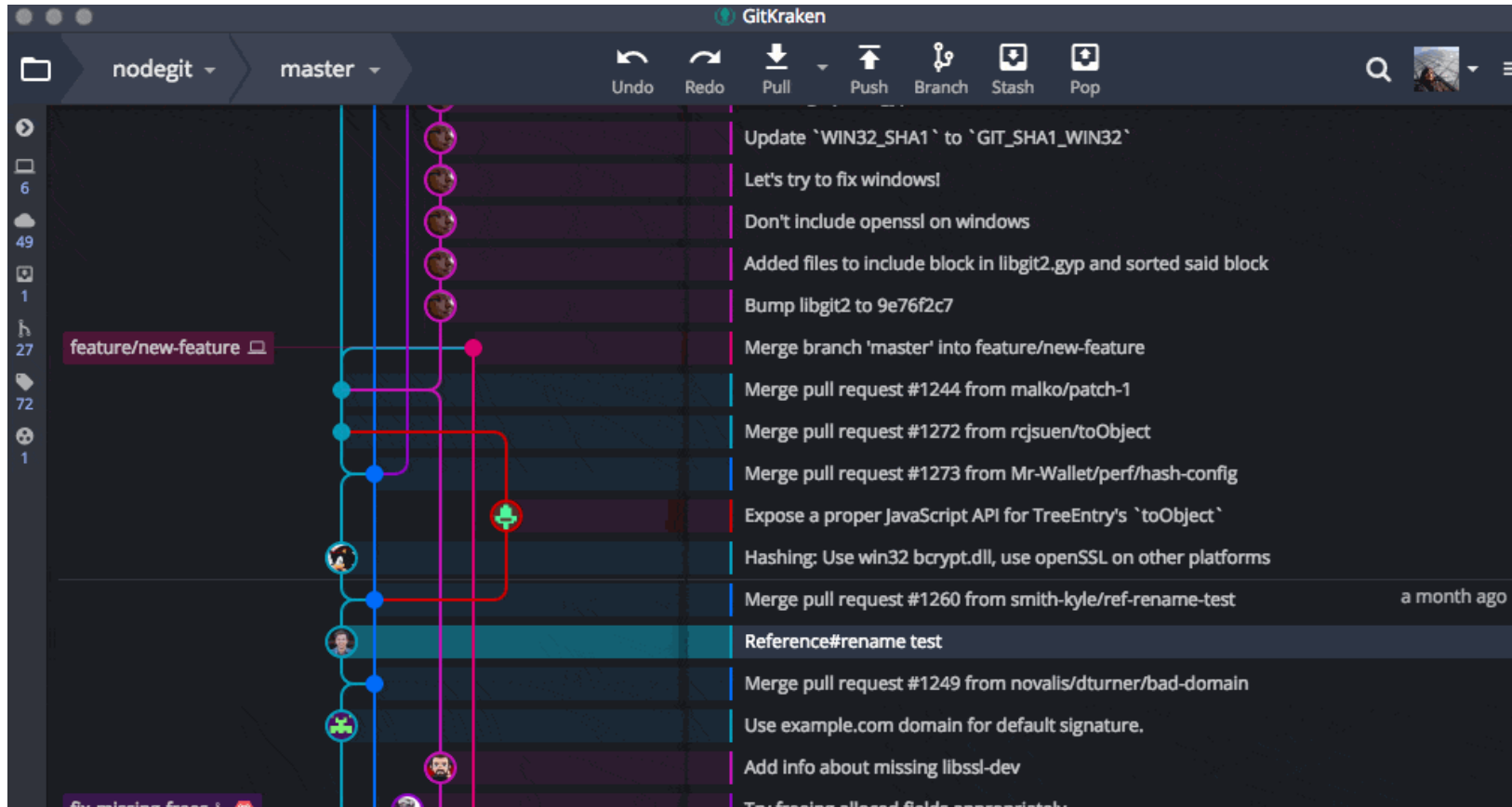
# Работа с репозиторием: просмотр и отмена изменений

```
root@shra-nest:/var/www/phoenix/www# git diff --no-index 1.html 2.html
diff --git a/1.html b/2.html
index cf82e72ab..9b62083f1 100755
--- a/1.html
+++ b/2.html
@@ -8,7 +8,7 @@
     crossorigin=""/>
-    <script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
+    <script src="https://unpkg.com/leaflet@1.7.0/dist/leaflet.js"
     crossorigin=""></script>
</head>
```

# Инструментарий: Графические клиенты (GUI), Sourcetree



# Инструментарий: Графические клиенты (GUI), GitKraken



# Спасибо за внимание!