**ОТЧЕТ**

**о выполнении лабораторной работы №2**

**Дисциплина «МДК 03.01 Моделирование и анализ программного обеспечения»**

**Тема: «Основные команды Git»**

**Специальность: 09.02.07 Информационные системы и программирование**

**Квалификация: Специалист по информационным системам**

Выполнил: Иванов Никита Вадимович

Студент группы: ИС50-11-23

Проверил:

Преподаватель: Кретова Е.М.

Дата: «8» Октября 2024 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

[**Теоретические сведения** 3](#_Toc179305382)

[**Порядок выполнения работы** 4](#_Toc179305383)

[**Выполнение задания** 5](#_Toc179305384)

# **Теоретические сведения**

**Контроль версий**, также известный как управление исходным кодом, — это практика отслеживания изменений программного кода и управления ими.

**Системы контроля версий** — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени.

**Git** — система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Преимущества:

* Git показывает очень высокую производительность в сравнении со множеством альтернатив. Это возможно благодаря оптимизации процедур фиксации коммитов, создания веток, слияния и сравнения предыдущих версий.
* При разработке в Git прежде всего обеспечивается целостность исходного кода под управлением системы. Содержимое файлов, а также объекты репозитория, фиксирующие взаимосвязи между файлами, каталогами, версиями, тегами и коммитами, защищены при помощи криптографически стойкого алгоритма хеширования SHA1.
* Гибкость — одна из основных характеристик Git. Она проявляется в поддержке различных нелинейных циклов разработки, эффективности использования с малыми и крупными проектами, а также совместимости со многими системами и протоколами.

# **Порядок выполнения работы**

1. Изучить книгу «Pro Git»
2. Выписать из нее основные команды с описанием их действия для работы с консолью
   1. Проверка настроек

Если вы хотите проверить используемую конфигурацию, можете использовать команду **git config --list**, чтобы показать все настройки, которые Git найдёт:



* 1. Создание репозитория в существующем каталоге
  2. Клонирование существующего репозитория
  3. Определение состояния файлов
  4. Отслеживание (добавление) новых файлов
  5. Индексация изменённых файлов
  6. Сокращённый вывод статуса
  7. Игнорирование файлов
  8. Просмотр индексированных и неиндексированных изменений
  9. Коммит изменений
  10. Удаление файлов
  11. Перемещение файлов
  12. Просмотр истории коммитов
  13. Операции отмены
  14. Отмена индексации файла
  15. Отмена изменений в файле
  16. Просмотр удалённых репозиториев
  17. Добавление удалённых репозиториев
  18. Получение изменений из удалённого репозитория
  19. Отправка изменений в удалённый репозиторий
  20. Просмотр удалённого репозитория
  21. Удаление и переименование удалённых репозиториев
  22. Просмотр списка тегов
  23. Создание ветки
  24. Переименование ветки
  25. Отправка изменений (ветка на удаленный сервер)
  26. Отслеживание веток
  27. Получение изменений (ветка на удаленный сервер)
  28. Удаление веток на удалённом серве

# **Выполнение задания**

2.1. Проверка настроек

Если вы хотите проверить используемую конфигурацию, можете использовать команду **git config --list**, чтобы показать все настройки, которые Git найдёт:



Рисунок - Проверка настроек

2.2. Создание репозитория в существующем каталоге

Создать репозиторий в Git можно одним из двух способов:

1. Вы можете взять локальный каталог, который в настоящее время не находится под

версионным контролем, и превратить его в репозиторий Git, либо

2.3. Вы можете клонировать существующий репозиторий Git из любого места.

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git, содержащий все необходимые файлы репозитория — структуру Git репозитория. На этом этапе ваш проект ещё не находится под версионным контролем.



Рисунок - Создание репозитория в существующем каталоге

2.4. Клонирование существующего репозитория

Для получения копии существующего Git-репозитория, например, проект необходимо использовать команду git clone.



Рисунок - Клонирование существующего репозитория

Эта команда создаёт каталог libgit2, инициализирует в нём подкаталог .git, скачивает все данные для этого репозитория и извлекает рабочую копию последней версии. Если вы перейдёте в только что созданный каталог libgit2, то увидите в нём файлы проекта, готовые для работы или использования.



Рисунок - Название результирующего каталога

Эта команда делает всё то же самое, что и предыдущая, только результирующий каталог будет назван mylibgit.

2.5. Определение состояния файлов

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда git status.



Рисунок - Определение состояния файлов

При добавлении в свой проект нового файла, простой файл README. Если этого файла раньше не было, и вы выполните git status, вы увидите свой неотслеживаемый файл вот так:

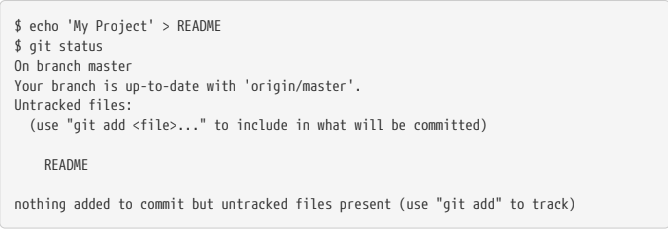


Рисунок - Неотслеживаемые файлы

2.6. Отслеживание (добавление) новых файлов

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл,

используется команда git add. Чтобы начать отслеживание файла README, вы можете выполнить следующее:



Рисунок - Добавление файла для отслеживания

При новом запуске команды status, можно увидеть, что файл README теперь отслеживаемый и добавлен в индекс:



Рисунок - Результат добавления отслеживания

2.7. Индексация изменённых файлов

Для индексации файла его необходимо немного модифицировать, если мы изменим отслеживаемый файл CONTRIBUTING.md и после этого снова выполните команду git status, то результат будет примерно следующим:

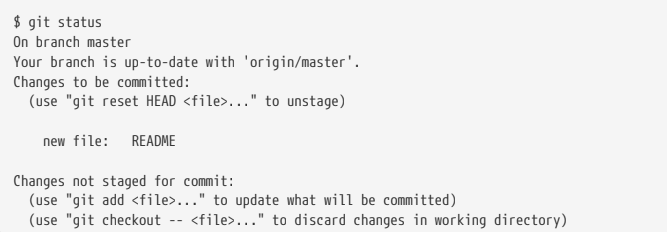


Рисунок - Индексация измененных файлов

2.8. Сокращённый вывод статуса

Вывод команды git status довольно всеобъемлющий и многословный. Git также имеет флаг вывода сокращённого статуса, так что вы можете увидеть изменения в более компактном виде. Если вы выполните git status -s или git status --short вы получите гораздо более упрощённый вывод:

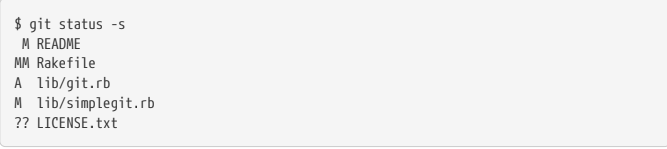


Рисунок – Сокращенный вывод статуса

2.9. Игнорирование файлов

Для того, чтобы фильтровать файлы (игнорировать их), можно создать файл .gitignore. с перечислением шаблонов соответствующих таким файлам. Вот пример файла .gitignore:

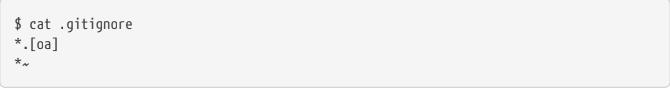


Рисунок - Игнорирование файлов

2.10. Просмотр индексированных и неиндексированных изменений

Допустим, вы снова изменили и проиндексировали файл README, а затем изменили файл CONTRIBUTING.md без индексирования. Чтобы увидеть, что же вы изменили, но пока не проиндексировали, наберите git diff без аргументов:

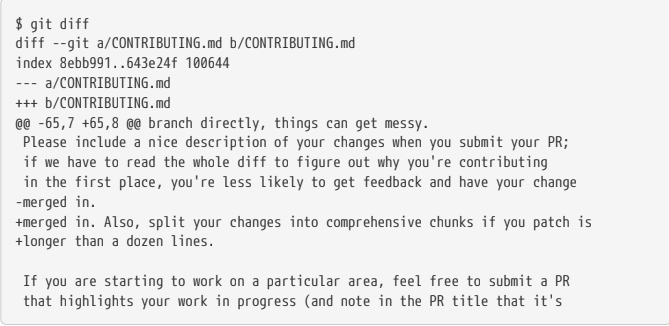


Рисунок - Просмотр изменений

Если вы хотите посмотреть, что вы проиндексировали и что войдёт в следующий коммит, вы можете выполнить git diff --staged. Эта команда сравнивает ваши проиндексированные изменения с последним коммитом:

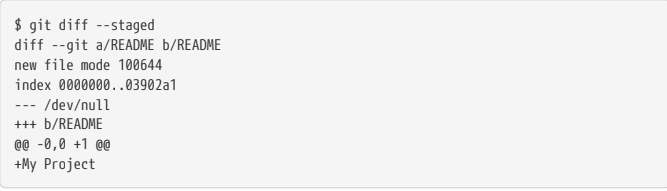


Рисунок - Сравнение проиндексированных изменений с последним коммитом

Используйте git diff для просмотра непроиндексированных изменений.



Рисунок - Просмотр непроиндексированных изменений

Также git diff --cached для просмотра проиндексированных изменений (--staged и --cached синонимы):

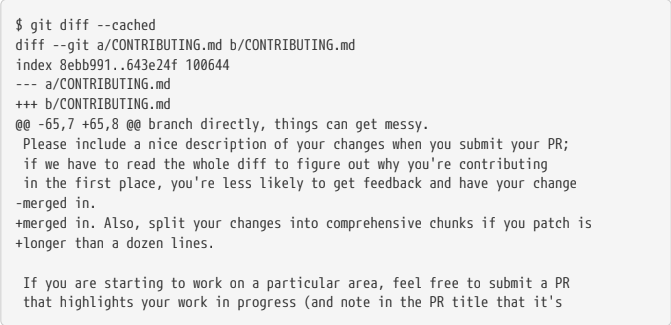


Рисунок - Просмотр проиндексированных изменений

2.11. Коммит изменений

Простейший способ зафиксировать изменения — это набрать git commit:



Рисунок - Коммит

Эта команда откроет выбранный вами текстовый редактор со следующим текстом.



Рисунок - Результат выполнения команды

2.12. Удаление файлов

Для того чтобы удалить файл из Git, вам необходимо удалить его из отслеживаемых файлов (точнее, удалить его из вашего индекса) а затем выполнить коммит. Это позволяет сделать команда git rm, которая также удаляет файл из вашего рабочего каталога, так что в следующий раз вы не увидите его как «неотслеживаемый».



Рисунок - Удаление файла из Git

Затем, если вы выполните команду git rm, удаление файла попадёт в индекс:



Рисунок - Результат удаления файла

После следующего коммита файл исчезнет и больше не будет отслеживаться. Если вы изменили файл и уже проиндексировали его, вы должны использовать принудительное удаление с помощью параметра -f.

2.13. Перемещение файлов

Для перемещения файла, можно воспользоваться командой mv:



Рисунок - Перемещение файлов

2.14. Просмотр истории коммитов

После того, как вы создали несколько коммитов или же клонировали репозиторий с уже существующей историей коммитов, вероятно вам понадобится возможность посмотреть что было сделано — историю коммитов. Одним из основных и наиболее мощных инструментов для этого является команда git log.

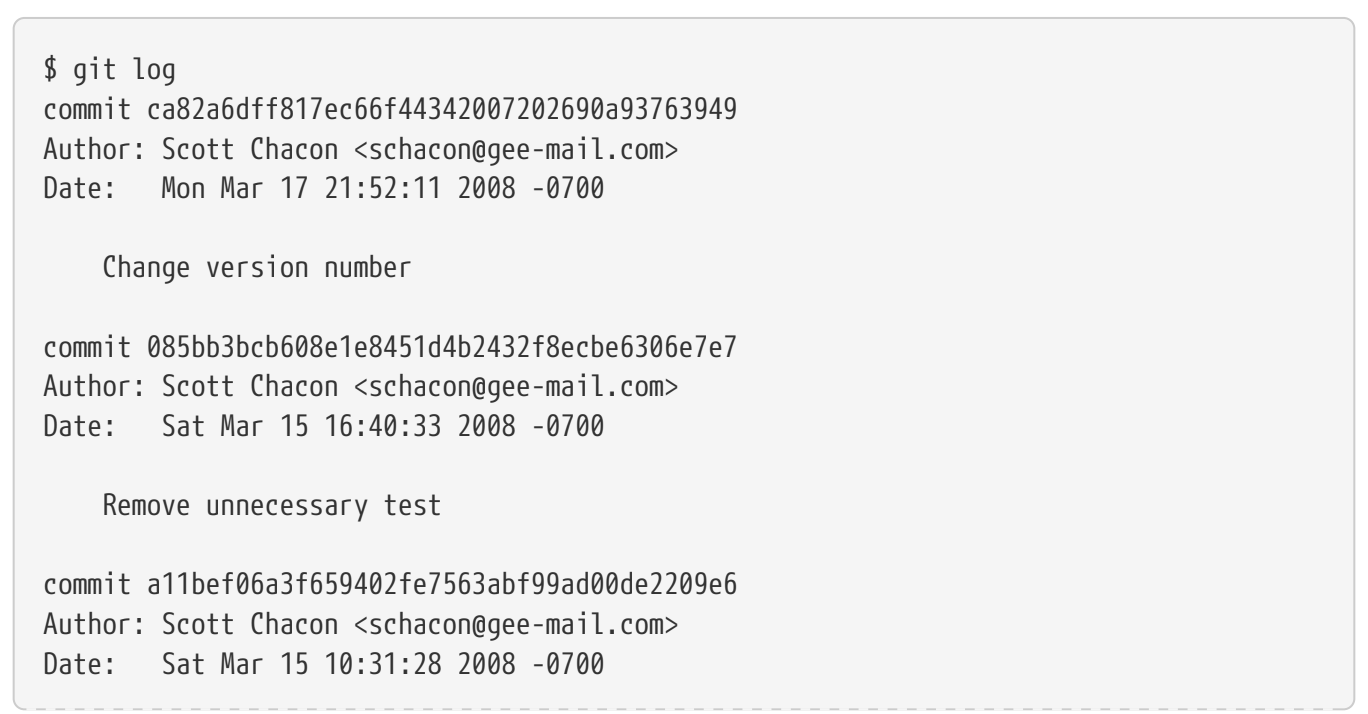




Рисунок - История коммитов

2.15. Операции отмены

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать коммит — внесите необходимые изменения, добавьте их в индекс и сделайте коммит ещё раз, указав параметр --amend:



Рисунок - Операция отмены

Если вы сделали коммит и поняли, что забыли проиндексировать изменения в файле, который хотели добавить в коммит, то можно сделать следующее:



Рисунок - Отмена после выполненного коммита

2.16. Отмена индексации файла

Используйте git reset HEAD <file>... для исключения из индекса.



Рисунок - Отмена индексации файла

2.17. Отмена изменений в файле

Для отмены изменений используется команда git reset HEAD <file> …



Рисунок - Отмена изменений в файле

2.18. Просмотр удалённых репозиториев

Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете

запустить команду git remote. Она выведет названия доступных удалённых репозиториев.

Если вы клонировали репозиторий, то увидите как минимум origin — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование:

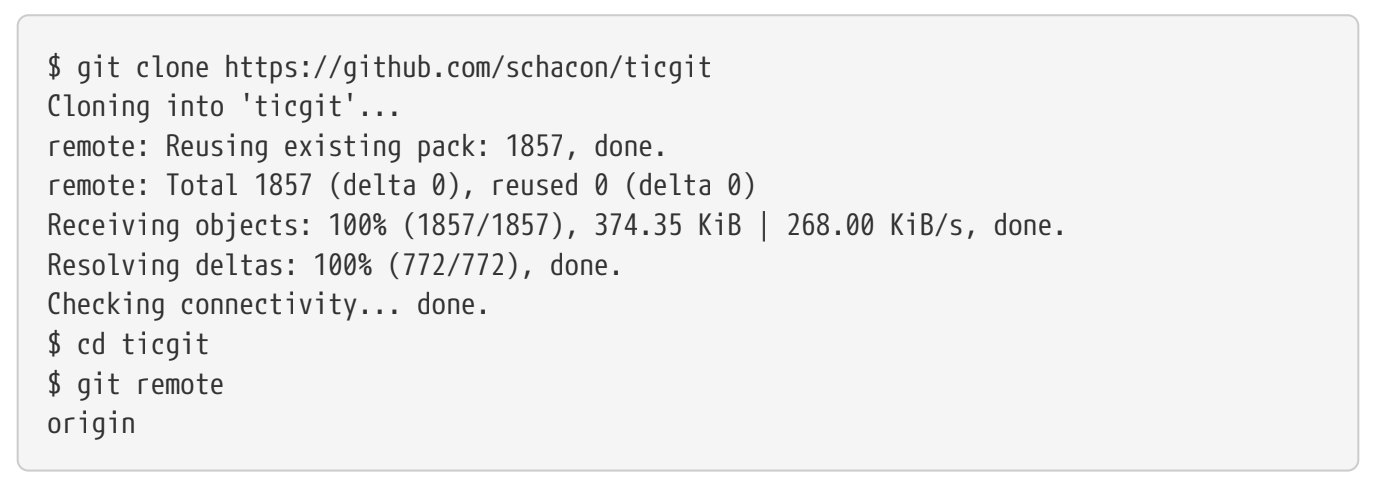


Рисунок - Просмотр удалённых репозиториев

2.19. Добавление удалённых репозиториев

В предыдущих разделах мы уже упоминали и приводили примеры добавления удалённых репозиториев, сейчас рассмотрим эту операцию подробнее. Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду git remote add <shortname> <url>:

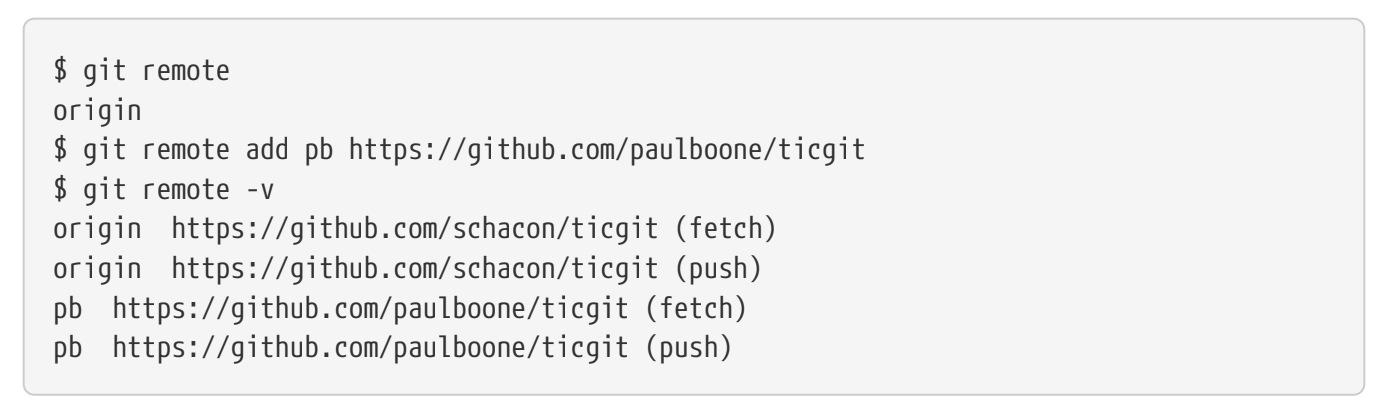


Рисунок - Добавление удаленных репозиториев

2.20 .Получение изменений из удалённого репозитория

Как вы только что узнали, для получения данных из удалённых проектов, следует выполнить следующую команду

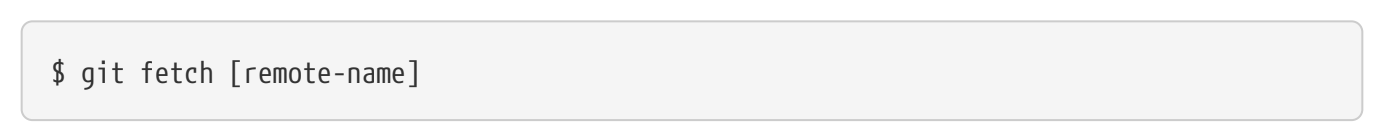


Рисунок - Получение изменений

2.21. Отправка изменений в удалённый репозиторий

Когда вы хотите поделиться своими наработками, вам необходимо отправить их в удалённый репозиторий. Команда для этого действия простая: git push <remote-name> <branch-name>. Чтобы отправить вашу ветку master на сервер origin (повторимся, что клонирование обычно настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки ваших коммитов:



Рисунок - Отправка изменений в удалённый репозиторий

2.22. Просмотр удалённого репозитория

Если хотите получить побольше информации об одном из удалённых репозиториев, вы можете использовать команду git remote show <remote>. Выполнив эту команду с некоторым именем, например, origin, вы получите следующий результат:

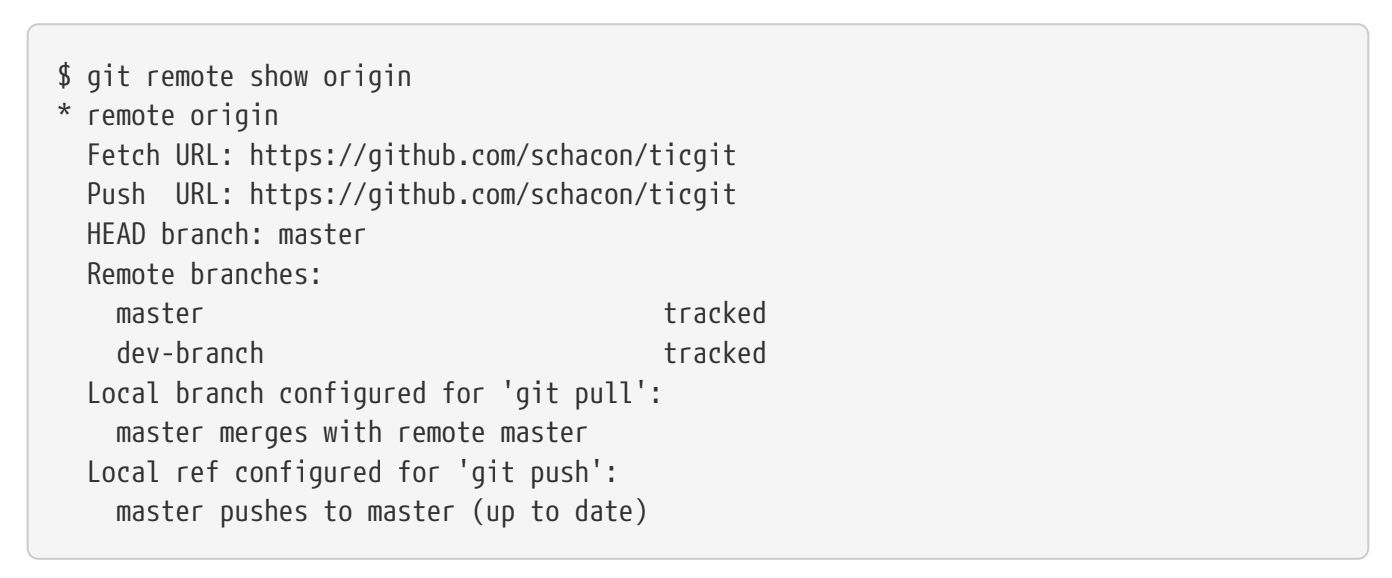


Рисунок - Просмотр удалённого репозитория

2.23. Удаление и переименование удалённых репозиториев

Для переименования удалённого репозитория можно выполнить git remote rename.

Например, если вы хотите переименовать pb в paul, вы можете это сделать при помощи git remote rename:



Рисунок - Удаление и переименование удалённых репозиториев

2.24. Просмотр списка тегов

Просмотреть список имеющихся тегов в Git можно очень просто. Достаточно набрать команду git tag (параметры -l и --list опциональны):



Рисунок - Просмотр списка тегов

2.25. Создание ветки

При создании ветки создаётся новый указатель для дальнейшего перемещения. Допустим вы хотите создать новую ветку с именем testing. Вы можете это сделать командой git branch:



Рисунок - Создание новой ветки

2.26. Переименование ветки

Переименуйте ветку локально с помощью команды git branch --move:

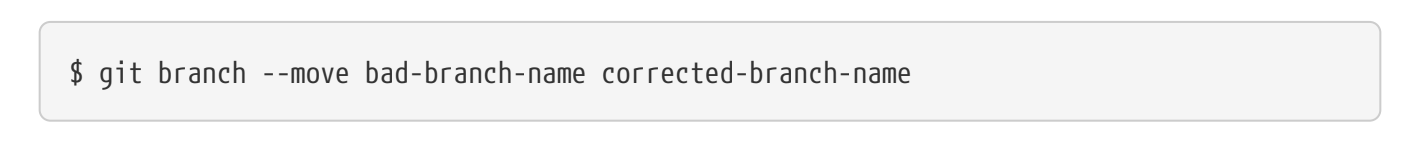


Рисунок - Переименование ветки

2.27. Отправка изменений (ветка на удаленный сервер)

Например, если у вас есть ветка serverfix, над которой вы хотите работать с кем-то ещё, вы можете отправить её точно так же, как вы отправляли вашу первую ветку. Выполните команду git push <remote> <branch>:

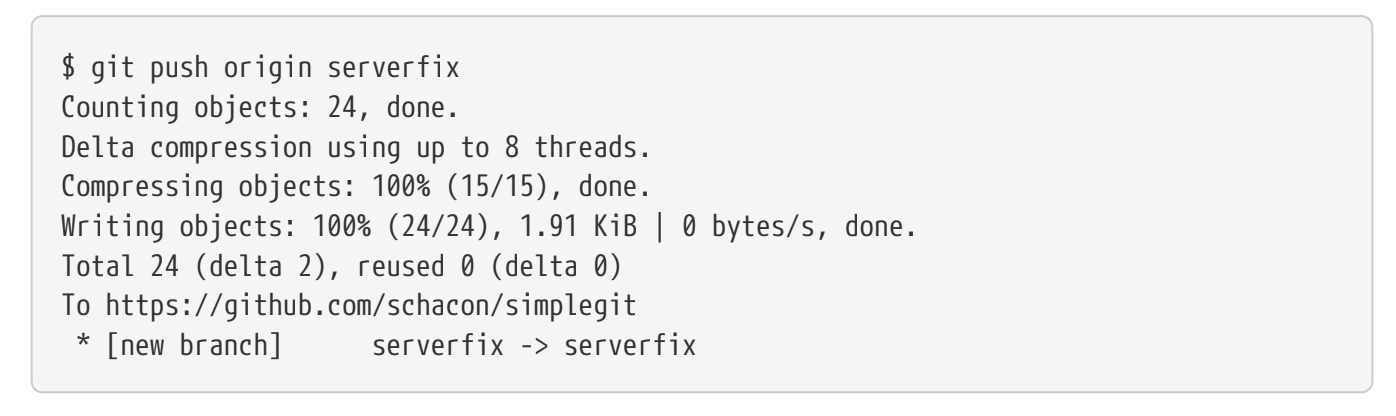


Рисунок - Отправка изменений на удаленный сервер

2.28. Отслеживание веток

Это можно с помощью команды git checkout -b <branch> <remote>/<branch>. Это часто используемая команда, поэтому Git предоставляет сокращённую форму записи в виде флага --track:



Рисунок - Отслеживание веток

2.29 Получение изменений (ветка на удаленный сервер)

Команда git fetch получает с сервера все изменения, которых у вас ещё нет, но не будет изменять состояние вашей рабочей копии. Эта команда просто получает данные и позволяет вам самостоятельно сделать слияние. Тем не менее, существует команда git pull, которая в большинстве случаев является командой git fetch, за которой непосредственно следует команда git merge. Если у вас настроена ветка слежения как показано в предыдущем разделе, или она явно установлена, или она была создана автоматически командами clone или checkout, git pull определит сервер и ветку, за которыми следит ваша текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

Обычно, лучше явно использовать команды fetch и merge, поскольку магия git pull может часто сбивать с толку.

* 1. Удаление веток на удалённом сервере

Скажем, вы и ваши соавторы закончили с нововведением и слили его в ветку master на

удалённом сервере (или в какую-то другую ветку, где хранится стабильный код). Вы можете удалить ветку на удалённом сервере используя параметр --delete для команды git push. Для удаления ветки serverfix на сервере, выполните следующую команду:

