A3.3 Appendix C: Команды Git - Основные команды

**Основные команды**

Всего несколько команд нужно для базового варианта использования Git для ведения истории изменений.

**git add**

Команда git add добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

Это одна из ключевых команд Git, мы упоминали о ней десятки раз на страницах книги. Ниже перечислены наиболее интересные варианты использования этой команды.

Знакомство с этой командой происходит в главе [Отслеживание новых файлов](https://git-scm.com/book/ru/v2/ch00/r_tracking_files).

О том как использовать git add для разрешения конфликтов слияния написано в главе [Основные конфликты слияния](https://git-scm.com/book/ru/v2/ch00/r_basic_merge_conflicts).

В главе [Интерактивное индексирование](https://git-scm.com/book/ru/v2/ch00/r_interactive_staging) показано как использовать git add для добавления в индекс лишь отдельных частей изменённого файла.

В главе [Деревья](https://git-scm.com/book/ru/v2/ch00/r_tree_objects) показано как эта команда работает на низком уровне, чтобы вы понимали, что происходит за кулисами.

**git status**

Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

Мы познакомили вас с этой командой в главе [Определение состояния файлов](https://git-scm.com/book/ru/v2/ch00/r_checking_status), разобрали стандартный и упрощённый формат вывода. И хотя мы использовали git status повсеместно в этой книге, практически все варианты использования покрыты в указанной главе.

**git diff**

Команда git diff используется для вычисления разницы между любыми двумя Git деревьями. Это может быть разница между вашей рабочей директорией и индексом (собственно git diff), разница между индексом и последним коммитом (git diff --staged), или между любыми двумя коммитами (git diff master branchB).

Мы познакомили вас с основами этой команды в главе [Просмотр индексированных и неиндексированных изменений](https://git-scm.com/book/ru/v2/ch00/r_git_diff_staged), где показали как посмотреть какие изменения уже добавлены в индекс, а какие — ещё нет.

О том как использовать эту команду для проверки на проблемы с пробелами с помощью аргумента --check можно почитать в главе [Commit Guidelines](https://git-scm.com/book/ru/v2/ch00/r_commit_guidelines).

Мы показали вам как эффективно сравнивать ветки используя синтаксис git diff A...B в главе [Determining What Is Introduced](https://git-scm.com/book/ru/v2/ch00/r_what_is_introduced).

В главе [Продвинутое слияние](https://git-scm.com/book/ru/v2/ch00/r_advanced_merging) показано использование опции -w для скрытия различий в пробельных символах, а также рассказано как сравнивать конфликтующие изменения с опциями --theirs, --ours и --base.

Использование этой команды с опцией --submodule для сравнения изменений в субмодулях показано в главе [Начало работы с подмодулями](https://git-scm.com/book/ru/v2/ch00/r_starting_submodules).

**git difftool**

Команда git difftool просто запускает внешнюю утилиту сравнения для показа различий в двух деревьях, на случай если вы хотите использовать что-либо отличное от встроенного просмотрщика git diff.

Мы лишь вкратце упомянули о ней в главе [Просмотр индексированных и неиндексированных изменений](https://git-scm.com/book/ru/v2/ch00/r_git_diff_staged).

**git commit**

Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

Вы познакомились с основами модели коммитов в главе [Коммит изменений](https://git-scm.com/book/ru/v2/ch00/r_committing_changes). Там же мы продемонстрировали использование опций -a для добавления всех изменений в индекс без использования git add, что может быть удобным в повседневном использовании, и -m для передачи сообщения коммита без запуска полноценного редактора.

В главе [Операции отмены](https://git-scm.com/book/ru/v2/ch00/r_undoing) мы рассказали об опции --amend, используемой для изменения последнего совершённого коммита.

В главе [О ветвлении в двух словах](https://git-scm.com/book/ru/v2/ch00/r_git_branches_overview) мы более подробно познакомились с тем, что делает команда git commit и почему она делает это именно так.

Мы показали вам как подписывать ваши коммиты, используя опцию -S в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

И наконец мы заглянули внутрь команды git commit в главе [Commit Objects](https://git-scm.com/book/ru/v2/ch00/r_git_commit_objects) и узнали что она делает за кулисами.

**git reset**

Команда git reset, как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочей директории при использовании параметра --hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений прежде чем использовать его.

Мы рассказали об основах использования git reset в главе [Отмена подготовки файла](https://git-scm.com/book/ru/v2/ch00/r_unstaging), где эта команда использовалась для удаления файла из индекса, добавленного туда с помощью git add.

В главе [Раскрытие тайн reset](https://git-scm.com/book/ru/v2/ch00/r_git_reset), полностью посвящённой этой команде, мы разобрались в деталях её использования.

Мы использовали git reset --hard чтобы отменить слияние в главе [Прерывание слияния](https://git-scm.com/book/ru/v2/ch00/r_abort_merge), там же было продемонстрировано использование команды git merge --abort для этих целей, которая работает как обёртка над git reset.

**git rm**

Команда git rm используется в Git для удаления файлов из индекса и рабочей директории. Она похожа на git add с тем лишь исключением, что она удаляет, а не добавляет файлы для следующего коммита.

Мы немного разобрались с этой командой в главе [Удаление файлов](https://git-scm.com/book/ru/v2/ch00/r_removing_files), показали как удалять файлы из рабочей директории и индекса и только из индекса, используя флаг --cached.

Ещё один вариант использования git rm приведён в главе [Removing Objects](https://git-scm.com/book/ru/v2/ch00/r_removing_objects), где мы вкратце объяснили как использовать опцию --ignore-unmatch при выполнении git filter-branch, которая подавляет ошибки удаления несуществующих файлов. Это может быть полезно для автоматически выполняемых скриптов.

**git mv**

Команда git mv — это всего лишь удобный способ переместить файл, а затем выполнить git addдля нового файла и git rm для старого.

Мы лишь вкратце упомянули это команду в главе [Перемещение файлов](https://git-scm.com/book/ru/v2/ch00/r_git_mv).

**git clean**

Команда git clean используется для удаления мусора из рабочей директории. Это могут быть результаты сборки проекта или файлы конфликтов слияний.

Мы рассмотрели множество опций и сценариев использования этой команды в главе [Очистка вашей рабочей директории](https://git-scm.com/book/ru/v2/ch00/r_git_clean).

# A3.4 Appendix C: Команды Git - Ветвление и слияния

## Ветвление и слияния

За создание новых веток и слияние их воедино отвечает несколько Git команд.

### git branch

Команда git branch — это своего рода "менеджер веток". Она умеет перечислять ваши ветки, создавать новые, удалять и переименовывать их.

Большая часть главы [Ветвление в Git](https://git-scm.com/book/ru/v2/ch00/ch03-git-branching) посвящена этой команде, она используется повсеместно в этой главе. Впервые команда branch была представлена в разделе [Создание новой ветки](https://git-scm.com/book/ru/v2/ch00/r_create_new_branch), а большинство таких её фич как перечисление и удаление веток были разобраны в разделе [Управление ветками](https://git-scm.com/book/ru/v2/ch00/r_branch_management).

В главе [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) мы показали как использовать сочетание git branch -u для отслеживания веток.

Наконец, мы разобрались что происходит за кулисами этой команды в главе [Ссылки в Git](https://git-scm.com/book/ru/v2/ch00/r_git_refs).

### git checkout

Команда git checkout используется для переключения веток и выгрузки их содержимого в рабочую директорию.

Мы познакомились с этой командой в главе [Переключение веток](https://git-scm.com/book/ru/v2/ch00/r_switching_branches) вместе с git branch.

В главе [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) мы узнали как использовать флаг --track для отслеживания веток.

В главе [Использование checkout в конфликтах](https://git-scm.com/book/ru/v2/ch00/r_checking_out_conflicts) мы использовали эту команду с опцией --conflict=diff3 для разрешения конфликтов заново, в случае если предыдущее решение не подходило по некоторым причинам.

Мы рассмотрел детали взаимосвязи этой команды и git reset в главе [Раскрытие тайн reset](https://git-scm.com/book/ru/v2/ch00/r_git_reset).

Мы исследовали внутренние механизмы этой команды в главе [HEAD](https://git-scm.com/book/ru/v2/ch00/r_the_head).

### git merge

Команда git merge используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.

Мы познакомили вас с этой командой в главе [Основы ветвления](https://git-scm.com/book/ru/v2/ch00/r_basic_branching). И хотя git merge встречается в этой книге повсеместно, практически все использования имеют вид git merge <branch> с указанием единственной ветки для слияния.

Мы узнали как делать "сплющенные" слияния (когда Git делает слияние в виде нового коммита, без сохранения всей истории работы) в конце главы [Forked Public Project](https://git-scm.com/book/ru/v2/ch00/r_public_project).

В главе [Продвинутое слияние](https://git-scm.com/book/ru/v2/ch00/r_advanced_merging) мы глубже разобрались с процессом слияния и этой командой, включая флаги -Xignore-all-whitespace и --abort, используемый для отмены слияния в случае возникновения проблем.

Мы научились проверять криптографические подписи перед слияниями если ваш проект использует GPG в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

Ну и наконец в главе [Слияние субдеревьев](https://git-scm.com/book/ru/v2/ch00/r_subtree_merge) мы познакомились со слиянием поддеревьев.

### git mergetool

Команда git mergetool просто вызывает внешнюю программу слияний, в случае если у вас возникли проблемы слияния.

Мы вкратце упомянули о ней в главе [Основные конфликты слияния](https://git-scm.com/book/ru/v2/ch00/r_basic_merge_conflicts) и рассказали как настроить свою программу слияния в главе [External Merge and Diff Tools](https://git-scm.com/book/ru/v2/ch00/r_external_merge_tools).

### git log

Команда git log используется для просмотра истории коммитов, начиная с самого свежего и уходя к истокам проекта. По умолчанию, она показывает лишь историю текущей ветки, но может быть настроена на вывод истории других, даже нескольких сразу, веток. Также её можно использовать для просмотра различий между ветками на уровне коммитов.

Практически во всех главах книги эта команда используется для демонстрации истории проекта.

Мы познакомились c git log и некоторыми её деталями в главе [Просмотр истории коммитов](https://git-scm.com/book/ru/v2/ch00/r_viewing_history). Там мы видели использование опций -p и --stat для получения представления об изменениях в каждом коммите, а также --pretty and --oneline для настройки формата вывода этой команды — более полным и подробным или кратким.

В главе [Создание новой ветки](https://git-scm.com/book/ru/v2/ch00/r_create_new_branch) мы использовали опцию --decorate чтобы отобразить указатели веток на истории коммитов, а также --graph чтобы просматривать историю в виде дерева.

В главах [Private Small Team](https://git-scm.com/book/ru/v2/ch00/r_private_team) и [Диапазоны коммитов](https://git-scm.com/book/ru/v2/ch00/r_commit_ranges) мы рассмотрели синтаксис branchA..branchBдля просмотра уникальных для заданной ветки коммитов. Мы часто использовали этот приём в [Диапазоны коммитов](https://git-scm.com/book/ru/v2/ch00/r_commit_ranges).

В главах [История при слиянии](https://git-scm.com/book/ru/v2/ch00/r_merge_log) и [Три точки](https://git-scm.com/book/ru/v2/ch00/r_triple_dot) мы рассмотрели синтаксис branchA...branchB и опцию --left-right для просмотра что находится в первой, либо второй ветке, но не сразу в обеих. Также в главе [История при слиянии](https://git-scm.com/book/ru/v2/ch00/r_merge_log) рассмотрели опцию --merge, которая может быть полезной при разрешении конфликтов, а также --cc для просмотра конфликтов слияния в истории проекта.

В главе [RefLog-сокращения](https://git-scm.com/book/ru/v2/ch00/r_git_reflog) мы использовали опцию -g для вывода git reflog, используя git log.

В главе [Поиск](https://git-scm.com/book/ru/v2/ch00/r_searching) мы рассмотрели использование опций -S и -L для поиска событий в истории проекта, например, истории развития какой-либо фичи.

В главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits) мы показали, как использовать опцию --show-signature для отображения строки валидации подписи для каждого коммита в git log.

### git stash

Команда git stash используется для временного сохранения всех незакоммиченных изменений для очистки рабочей директории без необходимости коммитить незавершённую работу в новую ветку.

Эта команда практически целиком раскрыта в главе [Прибережение и очистка](https://git-scm.com/book/ru/v2/ch00/r_git_stashing).

### git tag

Команда git tag используется для задания постоянной метки на какой-либо момент в истории проекта. Обычно она используется для релизов.

Мы познакомились и разобрались с ней в главе [Работа с метками](https://git-scm.com/book/ru/v2/ch00/r_git_tagging) и использовали на практике в [Tagging Your Releases](https://git-scm.com/book/ru/v2/ch00/r_tagging_releases).

Мы научились создавать подписанные с помощью GPG метки, используя флаг -s, и проверять их, используя флаг -v, в главе [Подпись результатов вашей работы](https://git-scm.com/book/ru/v2/ch00/r_signing).

# A3.5 Appendix C: Команды Git - Совместная работа и обновление проектов

## Совместная работа и обновление проектов

Не так уж много команд в Git требуют сетевого подключения для своей работы, практически все команды оперируют с локальной копией проекта. Когда вы готовы поделиться своими наработками, всего несколько команд помогут вам работать с удалёнными репозиториями.

### git fetch

Команда git fetch связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально.

Мы познакомились с ней в главе [Получение изменений из удалённого репозитория - Fetch и Pull](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) и продолжили знакомство в [Удалённые ветки](https://git-scm.com/book/ru/v2/ch00/r_remote_branches).

Мы использовали эту команду в нескольких примерах из главы [Contributing to a Project](https://git-scm.com/book/ru/v2/ch00/r_contributing_project).

Мы использовали её для скачивания запросов на слияние (pull request) из других репозиториев в главе [Pull Request Refs](https://git-scm.com/book/ru/v2/ch00/r_pr_refs), также мы рассмотрели использование git fetch для работы с упакованными репозиториями в главе [Создание пакетов](https://git-scm.com/book/ru/v2/ch00/r_bundling).

Мы рассмотрели тонкую настройку git fetch в главe и [Спецификации ссылок](https://git-scm.com/book/ru/v2/ch00/r_refspec).

### git pull

Команда git pull работает как комбинация команд git fetch и git merge, т.е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.

Мы познакомились с ней в главе [Получение изменений из удалённого репозитория - Fetch и Pull](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) и показали как узнать, какие изменения будут приняты в случае применения в главе [Просмотр удаленного репозитория](https://git-scm.com/book/ru/v2/ch00/r_inspecting_remote).

Мы также увидели как она может оказаться полезной для разрешения сложностей при перемещении веток в главе [Меняя базу, меняй основание](https://git-scm.com/book/ru/v2/ch00/r_rebase_rebase).

Мы показали как можно использовать только URL удалённого репозитория без сохранения его в списке удалённых репозиториев в главе [Checking Out Remote Branches](https://git-scm.com/book/ru/v2/ch00/r_checking_out_remotes).

И наконец мы показали как проверять криптографические подписи полученных коммитов, используя опцию --verify-signatures в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

### git push

Команда git push используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой команде нужно право на запись в репозиторий, поэтому она использует аутентификацию.

Мы познакомились с этой командой в главе [Отправка изменений в удаленный репозиторий (Push)](https://git-scm.com/book/ru/v2/ch00/r_pushing_remotes). Там мы рассмотрели основы обновления веток в удалённом репозитории. В главе [Отправка изменений](https://git-scm.com/book/ru/v2/ch00/r_pushing_branches) мы подробнее познакомились с этой командой, а в [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) мы узнали как настроить отслеживание веток для автоматической передачи на удалённый репозиторий. В главе [Удаление веток на удалённом сервере](https://git-scm.com/book/ru/v2/ch00/r_delete_branches) мы использовали флаг --delete для удаления веток на сервере, используя git push.

На протяжении главы [Contributing to a Project](https://git-scm.com/book/ru/v2/ch00/r_contributing_project) мы показали несколько примеров использования git push для совместной работы в нескольких удалённых репозиториях одновременно.

В главе [Публикация изменений в подмодуле](https://git-scm.com/book/ru/v2/ch00/r_publishing_submodules) мы использовали опцию --recurse-submodulesчтобы удостовериться, что все субмодули будут опубликованы перед отправкой на проекта на сервер, что может быть реально полезным при работе с репозиториями, содержащими субмодули.

В главе [Other Client Hooks](https://git-scm.com/book/ru/v2/ch00/r_other_client_hooks) мы поговорили о триггере pre-push, который может быть выполнен перед отправкой данных, чтобы проверить возможность этой отправки.

Наконец, в главе [Спецификации ссылок для отправки данных на сервер](https://git-scm.com/book/ru/v2/ch00/r_pushing_refspecs) мы рассмотрели передачу данных с полным указанием передаваемых ссылок, вместо использования распространённых сокращений. Это может быть полезным если вы хотите очень точно указать, какими изменениями хотите поделиться.

### git remote

Команда git remote служит для управления списком удалённых репозиториев. Она позволяет сохранять длинные URL репозиториев в виде понятных коротких строк, например "origin", так что вам не придётся забивать голову всякой ерундой и набирать её каждый раз для связи с сервером. Вы можете использовать несколько удалённых репозиториев для работы и git remote поможет добавлять, изменять и удалять их.

Эта команда детально рассмотрена в главе [Работа с удалёнными репозиториями](https://git-scm.com/book/ru/v2/ch00/r_remote_repos), включая вывод списка удалённых репозиториев, добавление новых, удаление или переименование существующих.

Она используется практически в каждой главе, но всегда в одном и том же виде: git remote add <имя> <URL>.

### git archive

Команда git archive используется для упаковки в архив указанных коммитов или всего репозитория.

Мы использовали git archive для для создания тарбола (tar.gz файла) всего проекта для передачи по сети в главе [Preparing a Release](https://git-scm.com/book/ru/v2/ch00/r_preparing_release).

### git submodule

Команда git submodule используется для управления вложенными репозиториями. Например, это могут быть библиотеки или другие, используемые не только в этом проекте ресурсы. У команды submodule есть несколько под-команд — add, update, sync и др. — для управления такими репозиториями.

Эта команда упомянута и полностью раскрыта в главе [Подмодули](https://git-scm.com/book/ru/v2/ch00/r_git_submodules).

## Осмотр и сравнение

### git show

Команда git show отображает объект в простом и человекопонятном виде. Обычно она используется для просмотра информации о метке или коммите.

Впервые мы использовали её для просмотра информации об аннотированой метке в главе [Аннотированные метки](https://git-scm.com/book/ru/v2/ch00/r_annotated_tags).

В главе [Выбор ревизии](https://git-scm.com/book/ru/v2/ch00/r_revision_selection) мы использовали её для показа коммитов, подпадающих под различные селекторы диапазонов.

Ещё одна интересная вещь, которую мы проделывали с помощью git show в главе [Ручное слияние файлов](https://git-scm.com/book/ru/v2/ch00/r_manual_remerge) — это извлечение содержимого файлов на различных стадиях во время конфликта слияния.

### git shortlog

Команда git shortlog служит для подведения итогов команды git log. Она принимает практически те же параметры, что и git log, но вместо простого листинга всех коммитов, они будут сгруппированы по автору.

Мы показали, как можно использовать эту команду для создания классных списков изменений (changelogs) в главе [The Shortlog](https://git-scm.com/book/ru/v2/ch00/r_the_shortlog).

### git describe

Команда git describe принимает на вход что угодно, что можно трактовать как коммит (ветку, тег) и выводит более-менее человекочитаемую строку, которая не изменится в будущем для данного коммита. Это может быть использовано как более удобная, но по-прежнему уникальная, замена SHA-1.

Мы использовали git describe в главах [Generating a Build Number](https://git-scm.com/book/ru/v2/ch00/r_build_number) и [Preparing a Release](https://git-scm.com/book/ru/v2/ch00/r_preparing_release) чтобы сгенерировать название для архивного файла с релизом.

## Отладка

В Git есть несколько команд, используемых для нахождения проблем в коде. Это команды для поиска места в истории, где проблема впервые проявилась и собственно виновника этой проблемы.

### git bisect

Команда git bisect — это чрезвычайно полезная утилита для поиска коммита в котором впервые проявился баг или проблема с помощью автоматического бинарного поиска.

О ней упоминается только в главе [Бинарный поиск](https://git-scm.com/book/ru/v2/ch00/r_binary_search), где она полностью и раскрыта.

### git blame

Команда git blame выводит перед каждой строкой файла SHA-1 коммита, последний раз менявшего эту строку и автора этого коммита. Это помогает в поисках человека, которому нужно задавать вопросы о проблемном куске кода.

Эта команда полностью разобрана в главе [Аннотация файла](https://git-scm.com/book/ru/v2/ch00/r_file_annotation).

### git grep

Команда git grep используется для поиска любой строки или регулярного выражения в любом из файлов вашего проекта, даже в более ранних его версиях.

Она полностью разобрана в разделе [Git Grep](https://git-scm.com/book/ru/v2/ch00/r_git_grep) и упоминается лишь там.

## Внесение исправлений

Некоторые команды в Git основываются на подходе к рассмотрению коммитов в терминах внесённых ими изменений, т.е. рассматривают историю коммитов как цепочку патчей. Ниже перечислены эти команды.

### git cherry-pick

Команда git cherry-pick используется для того чтобы взять изменения, внесённые каким-либо коммитом, и попытаться применить их заново в виде нового коммита наверху текущей ветки. Это может оказаться полезным чтобы забрать парочку коммитов из другой ветки без полного слияния с той веткой.

Мы продемонстрировали работу этой команды в главе [Rebasing and Cherry Picking Workflows](https://git-scm.com/book/ru/v2/ch00/r_rebase_cherry_pick).

### git rebase

git rebase — это "автоматизированный" cherry-pick. Он выполняет ту же работу, но для цепочки коммитов, тем самым как бы перенося ветку на новое место.

Мы в деталях разобрались с механизмом переноса веток в главе [Перебазирование](https://git-scm.com/book/ru/v2/ch00/r_rebasing), включая рассмотрение потенциальных проблем переноса опубликованных веток при совместной работе.

Мы использовали эту команду на практике для разбиения истории на два репозитория в главе [Замена](https://git-scm.com/book/ru/v2/ch00/r_replace), наряду с использованием флага --onto.

В главе [Rerere](https://git-scm.com/book/ru/v2/ch00/r_rerere) мы рассмотрели случай возникновения конфликта во время переноса коммитов.

Также мы познакомились с интерактивным вариантом git rebase, включающемся с помощью опции -i, в главе [Изменение сообщений нескольких коммитов](https://git-scm.com/book/ru/v2/ch00/r_changing_multiple).

### git revert

Команда git revert — полная противоположность git cherry-pick. Она создаёт "антикоммит" для указанного коммита, таким образом отменяя изменения, внесённые в нём..

Мы использовали её в главе [Отмена коммита](https://git-scm.com/book/ru/v2/ch00/r_reverse_commit) чтобы отменить коммит слияния (merge commit).

## Работа с помощью электронной почты

Множество проектов, использующих Git (включая сам Git), активно используют списки рассылок для координирования процесса разработки. В Git есть несколько команд, помогающих в этом, начиная от генерации патчей, готовых к пересылке по электронной почте, заканчивая применением таких патчей прямиком из папки "входящие".

### git apply

Команда git apply применяет патч, сформированный с помощью команды git diff или GNU diff. Она делает практически то же самое, что и команда patch.

Мы продемонстрировали использование этой команды в главе [Applying Patches from E-mail](https://git-scm.com/book/ru/v2/ch00/r_patches_from_email) и описали случаи, когда вы возможно захотите ею воспользоваться.

### git am

Команда git am используется для применения патчей из ящика входящих сообщений электронной почты, в частности, тех что используют формат mbox. Это используется для простого получения изменений через email и применения их к проекту.

Мы рассмотрели использование этой команды в главе [Applying a Patch with am](https://git-scm.com/book/ru/v2/ch00/r_git_am), включая такие опции как --resolved, -i и -3.

Существует набор триггеров, которые могут оказаться полезными при использовании git am для процесса разработки. О них рассказано в главе [E-mail Workflow Hooks](https://git-scm.com/book/ru/v2/ch00/r_email_hooks).

Также мы использовали git am для применения сформированного из Github’овского запроса на слияние patch-файла в главе [Email Notifications](https://git-scm.com/book/ru/v2/ch00/r_email_notifications).

### git format-patch

Команда git format-patch используется для создания набора патчей в формате mbox которые можно использовать для отправки в список рассылки.

Мы рассмотрели процесс отсылки изменений в проект, использующий email для разработки в главе [Public Project over E-Mail](https://git-scm.com/book/ru/v2/ch00/r_project_over_email).

### git send-email

Команда git send-email используется для отсылки патчей, сформированных с использованием git format-patch, по электронной почте.

Процесс отсылки изменений по электронной почте в проект рассмотрен в главе [Public Project over E-Mail](https://git-scm.com/book/ru/v2/ch00/r_project_over_email).

### git request-pull

Команда git request-pull используется для генерации примерного текста сообщения для отсылки кому-либо. Если у вас есть ветка, хранящаяся на публичном сервере, и вы хотите чтобы кто-либо забрал эти изменения без возни с отсылкой патчей по электронной почте, вы можете выполнить эту команду и послать её вывод тому человеку.

Мы показали, как пользоваться этой командой в главе [Forked Public Project](https://git-scm.com/book/ru/v2/ch00/r_public_project).

## Внешние системы

В Git есть несколько стандартных команд для работы с другими системами контроля версий.

### git svn

Команда git svn используется для работы с сервером Subversion. Это означает, что вы можете использовать Git в качестве SVN клиента, забирать изменения и отправлять свои собственные на сервер Subversion.

Мы разобрались с этой командой в главе [Git и Subversion](https://git-scm.com/book/ru/v2/ch00/r_git_svn).

### git fast-import

Для других систем контроля версий, либо для импорта произвольно форматированных данных, вы можете использовать git fast-import, которая умеет преобразовывать данные в формат, понятный Git’у.

Мы детально рассмотрели эту команду в главе [Импорт произвольного репозитория](https://git-scm.com/book/ru/v2/ch00/r_custom_importer).

## Администрирование

Если вы администрируете Git репозиторий или вам нужно исправить что-либо, Git предоставляет несколько административных команд вам в помощь.

### git gc

Команда git gc запускает сборщик мусора в вашем репозитории, который удаляет ненужные файлы из хранилища объектов и эффективно упаковывает оставшиеся файлы.

Обычно, эта команда выполняется автоматически без вашего участия, но, если пожелаете, можете вызвать её вручную. Мы рассмотрели некоторые примеры её использования в главе [Уход за репозиторием](https://git-scm.com/book/ru/v2/ch00/r_git_gc).

### git fsck

Команда git fsck используется для проверки внутренней базы данных на предмет наличия ошибок и несоответствий.

Мы лишь однажды использовали её в главе [Восстановление данных](https://git-scm.com/book/ru/v2/ch00/r_data_recovery) для поиска более недостижимых (dangling) объектов.

### git reflog

Команда git reflog просматривает историю изменения голов веток на протяжении вашей работы для поиска коммитов, которые вы могли внезапно потерять, переписывая историю.

В основном, мы рассматривали эту команду в главе [RefLog-сокращения](https://git-scm.com/book/ru/v2/ch00/r_git_reflog), где мы показали пример использования этой команды, а также как использовать git log -g для просмотра той же информации, используя git log.

Мы на практике рассмотрели восстановление потерянной ветки в главе [Восстановление данных](https://git-scm.com/book/ru/v2/ch00/r_data_recovery).

### git filter-branch

Команда git filter-branch используется для переписывания содержимого коммитов по заданному алгоритму, например, для полного удаления файла из истории или для вычленения истории лишь части файлов в проекте для вынесения в отдельный репозиторий.

В главе [Удаление файла из каждого коммита](https://git-scm.com/book/ru/v2/ch00/r_removing_file_every_commit) мы объяснили механизм работы этой команды и рассказали про использование опций --commit-filter, --subdirectory-filter и --tree-filter.

В главах [Git-p4](https://git-scm.com/book/ru/v2/ch00/r_git_p4) и [TFS](https://git-scm.com/book/ru/v2/ch00/r_git_tfs) мы использовали эту команду для исправления импортированных репозиториев.