

Head First

Git

Лучший способ
понять Git
изнутри

Раджу Ганди



Прокачай свой мозг!

Оглавление (краткое)

Вступление. Как использовать эту книгу.....	17
1. Знакомство с Git. Приступим к делу	36
2. Ветвление кода. Свободный полет вашей мысли.....	84
3. Осмотритесь вокруг. Изучим ваш репозиторий Git.....	144
4. Отмена действий. Исправляем свои ошибки	185
5. Командная работа с Git — часть I. Удаленная работа.....	239
6. Совместная работа с Git — часть II. Команда, вперед!	290
7. Поиск в репозитории Git. Git идет искать.....	370
8. Эффективная работа с Git. #Советы профессионалов	414
Приложение: остатки. Пять важных тем, о которых мы не говорили.....	455

Оглавление (с пояснениями)

Вступление

Заставьте мозг принять Git. Сейчас вы пытаетесь чему-то научиться, в то время как ваш мозг делает вам одолжение, следя за тем, чтобы обучение не закрепилось. Ваш мозг думает: "Лучше оставить место для более важного, например, того, каких диких животных следует избегать и почему кататься голым на сноуборде — плохая идея". Так как же заставить свой мозг уверовать, что ваша жизнь зависит от знания Git?

Кому адресована эта книга?	18
Я знаю, о чем вы думаете	19
Метапознание: мышление о мышлении.....	21
Вот что МЫ сделали	22
Многое зависит от ВАС.....	23
Вам нужно установить Git (macOS).....	25
Вам нужно установить Git (Windows).....	26
Вам нужно установить текстовый редактор (macOS).....	28
Вам нужно установить текстовый редактор (Windows).....	29
Вам понадобится учетная запись GitHub.....	30
Немного об организации ваших файлов и проектов	32
Технические рецензенты.....	33
Благодарности	34
Для тех, кто подумал, что благодарности закончились	35

Знакомство с Git

1

Приступим к делу



Нам нужен контроль версий! Каждый программный проект начинается с идеи, реализованной в исходном коде. Эти файлы — сердце наших приложений, поэтому следует относиться к ним с осторожностью. Мы должны быть уверены, что храним их в безопасности, сохраняем историю изменений и приписываем заслуги (или вину!) законным авторам. Мы также должны обеспечить комфортное сотрудничество между несколькими членами команды.

Вдобавок нам нужно, чтобы все это было в одном инструменте, который всегда под рукой, но не мешает нам и срабатывает только в нужный момент.

Существует ли такой волшебный инструмент? Наверное, вы догадываетесь, каков будет ответ. Конечно, это Git! Разработчики и организации по всему миру любят Git. Так что же делает Git таким популярным?

Зачем нам нужен контроль версий	37
Разговор в офисе разработчиков	39
Освоим Git?	40
Краткий обзор командной строки: как узнать, где вы находитесь, с помощью <code>pwd</code>	42
Еще про командную строку: создание каталога при помощи <code>mkdir</code>	43
Больше командной строки: список файлов с командой <code>ls</code>	44
Почти закончили знакомство: смена каталога с командой <code>cd</code>	45
Команде нужны аргументы	46
Сотрите ненужное	48
Создайте свой первый репозиторий	49
Как работает команда <code>init</code>	50
Расскажите Git о себе	52
Как нам пригодится Git?	53
Подключаем Git к работе	54
Использование репозитория Git	56
Что означает слово "коммит"?	58
Смотри, прежде чем прыгать	60
Три этапа Git	61
Git в командной строке	63
Заглянем за кулисы Git	64
Разные состояния файлов в репозитории Git	65
Индекс — это ваш "блокнот"	68
Компьютер, доложи обстановку!	70
Вы создали историю версий!	76

Ветвление кода

2 Свободный полет вашей мысли

Вы можете ходить и жевать резинку одновременно. Стародавние приверженцы Git скажут вам, откинувшись в шезлонге и потягивая крафтовый зеленый чай, что одним из самых больших преимуществ Git является легкость, с которой вы можете создавать ветки кода. Возможно, вам поручили создать новую опцию интерфейса, и пока вы над ней работаете, ваш менеджер просит вас исправить ошибку в производственной версии. Или, может быть, вы только что закончили вносить изменения в код, но внезапно пришло вдохновение, и вы придумали лучший способ реализации алгоритма. Ветки позволяют вам работать над несколькими, совершенно не связанными частями одной и той же кодовой базы в одно и то же время, независимо друг от друга. Давайте разберемся, как это устроено!

Все начинается с электронной почты.....	85
Обновление ресторанный меню	88
Сколько возможностей... глаза разбегаются!.....	91
Переключая дорожки.....	92
Вернитесь к началу!	94
Визуализация веток.....	96
Ветки, коммиты и файлы в них	97
Параллельная работа	100
Что такое ветка на самом деле?.....	102
Переключаем ветки или каталоги?	104
Торт и глазурь, объединяйтесь!.....	107
Читайте FAQ инструкцию!.....	109
Делать слияние не всегда легко!.....	112
Направление слияния имеет значение	113
Еще немного про настройку Git	114
Эй, ты куда?.....	117
Это коммит слияния	120
Нужно действовать очень осторожно!.....	124
Я очень конфликтный!	124
Очистка (слитых) веток.....	130
Удаление обособленных веток.....	133
Типичный рабочий процесс.....	134



Осмотритесь вокруг

3

Изучим ваш репозиторий Git

Вы готовы копнуть глубже, Шерлок? Продолжая работать в Git, вы будете создавать ветки, делать коммиты и объединять свою работу в ветках интеграции. Каждый коммит — это шаг вперед, а история коммитов показывает, как вы туда попали. Время от времени вы будете оглядываться назад, чтобы увидеть, как вы оказались в текущем положении, или понять, как две ветки отошли друг от друга. Мы начнем эту главу с демонстрации того, как Git помогает визуализировать историю коммитов.

Просмотр истории коммитов важен, но Git также помогает увидеть, как изменился ваш репозиторий. Напомним, что коммиты представляют собой изменения, а ветки отражают последовательность изменений. Как узнать, где именно произошло изменение — между коммитами, между ветвями или даже между вашим рабочим каталогом, индексом и базой данных объектов? Это другая тема данной главы.

Вместе мы продедем очень интересную детективную работу и разберемся, как устроен Git. Итак, давайте прокачаем навыки расследования!

Бриджит ищет работу.....	145
Коммитов недостаточно.....	147
Кто на свете всех милее, форматирован ровнее?.....	149
Как работает <code>git log</code> ?.....	153
Как заставить <code>git log</code> делать всю работу	154
В чем заключаются различия?	158
Визуализация различий	159
Визуализация различий: один файл за раз	160
Визуализация различий: один ханк за раз	161
Делаем команду <code>diff</code> более наглядной	162
Поиск поэтапных изменений	165
Различия между ветками	168
Поиск различий в коммитах	174
Как <code>diff</code> работает с новыми файлами?.....	175



Отмена действий

4

Исправляем свои ошибки

Мы все делаем ошибки, верно? Люди совершают ошибки с незапамятных времен, и долгое время ошибки обходились нам довольно дорого (в эпоху перфокарт и пишущих машинок нам приходилось все переделывать). Причина очевидна — тогда у нас не было системы контроля версий. Но теперь она есть! Git предоставляет широкие возможности для легкого и безболезненного исправления ошибок. Если вы случайно добавили файл в индекс, допустили опечатку в сообщении коммита или сделали неправильный коммит, Git раскрывает перед вами множество рычагов и кнопок, которыми можно воспользоваться, чтобы никто никогда не узнал о вашем... кхм... промахе.

Прочитав эту главу, вы точно будете знать, что делать — независимо от того какую ошибку вы допустили. Итак, давайте сделаем несколько ошибок и научимся их исправлять.

Вечеринка по случаю помолвки	186
Ошибка в планировании	188
Отмена изменений в рабочем каталоге	190
Отмена изменений в индексе	192
Удаление файлов из репозитория Git	195
Коммит удаления	196
Переименование и перемещение файлов	198
Редактирование сообщений коммитов	199
Переименование веток	202
Создаем альтернативный план.....	205
Роль и значение HEAD.....	209
Обращение к коммитам через HEAD	211
Обход коммитов слияния	212
Отмена коммитов	214
Удаление коммитов командой <code>reset</code>	215
Три разновидности команды <code>reset</code>	216
Другой способ отменить коммиты	221
Реверсная отмена коммитов	222
И-и-и-и-и-и-и... все получилось!.....	225



Командная работа с Git — часть I

5 Удаленная работа

Работа в одиночестве может быстро надоесть. В этой книге вы многое узнали о том, как устроен Git и как работать с его репозиториями. Вместе с вами мы использовали репозитории, которые инициализировали локально с помощью команды `git init`. Мы проделали большую работу — создали ветки, объединили их и использовали утилиты Git, такие как команды `git log` и `git diff`, чтобы отследить развитие репозитория. Но большинство проектов не такие. Мы часто работаем в командах с друзьями или коллегами. Git предлагает очень мощную модель совместной работы, в которой мы все можем делиться своими достижениями, используя общий репозиторий. Все начинается с того, что наш репозиторий становится публичным, что делает историю коммитов проекта "общей". В общедоступном репозитории вы можете делать все, чему научились до сих пор, точно так же, как раньше (за некоторыми исключениями). Вы можете создавать ветки и коммиты и добавлять их в историю коммитов наравне с коллегами или друзьями. Так устроена командная работа в Git. Но прежде чем мы перейдем к командной работе, давайте вместе разберемся, как работают публичные репозитории. Вперед, друзья!

Клонирование репозитория Git.....	240
На старт, внимание, клон!	244
Что происходит при клонировании?	248
Распределенная система контроля версий	250
Выгрузка обновлений	254
Смотрите, куда выгружаете!	259
Не фотографировать: публичные и частные коммиты.....	261
Стандартная рабочая процедура: ветки	263
Слияние веток: вариант 1 (локальные слияния)	265
Выгрузка локальных веток	269
Слияние веток: вариант 2 (pull request)	273
Создание запроса на слияние	280
Pull requests или merge requests?.....	278
Выполнение запросов на слияние.....	280
Что дальше?	282



Совместная работа с Git — часть II

6 Команда, вперед!

Готовы собрать команду? Git — фантастический инструмент для совместной работы, и у меня возникла блестящая идея научить вас всему этому — читая эту главу, вы объединитесь с кем-то еще! Вы будете опираться на сведения, полученные в предыдущей главе. Вы знаете, что работа с распределенной системой, такой как Git, включает в себя множество компонентов. Итак, что же Git предлагает нам для упрощения этой задачи и что вам нужно помнить, когда вы начинаете совместную работу с коллегами? Существуют ли какие-то специальные функции, облегчающие совместную работу? Сейчас вы об этом узнаете.

На старт. Внимание. Клон!

Параллельная работа	292
Параллельная работа... в Гитландии	293
Совместная работа в стиле Git	295
Работа двух коллег на GitHub	296
Как "догнать" изменения	304
Как "догнать" изменения (git pull)	306
Ваши посредники — ветки слежения	310
Первая причина использовать ветки слежения	311
Отправка в удаленный репозиторий: итог	319
Получение веток слежения	320
Вторая причина использовать ветки слежения: получение всех обновлений с удаленного репозитория	321
Совместная работа	325
Совместная работа: подведем итог	329
Третья причина использовать ветки слежения: не забыть отправить ветки в репозиторий	330
Четвертая причина использовать ветки слежения: подготовка к выгрузке в репозиторий	332
git pull = git fetch + git merge!	337
Используйте git fetch + git merge. Избегайте git pull	338
Идеальный сценарий	341
Типичный рабочий процесс: начало	342
Типичный рабочий процесс: подготовка к слиянию	343
Типичный рабочий процесс: локальное слияние или pull request?	344
Наглядная схема рабочего процесса	346
Удаление рабочих веток	349



Поиск в репозитории Git

7 Git идет искать!

Ваш проект и его история коммитов неизбежно будут расти. Время от времени вам придется искать в ваших файлах определенный фрагмент текста. Возможно, вы захотите узнать, кто изменил файл, когда он был изменен, а также выяснить, какой коммит его изменил. Git охотно поможет вам в этом.

А еще есть ваша история коммитов. Каждый коммит представляет собой изменение. Git позволяет вам искать не только каждый экземпляр фрагмента текста в вашем проекте, но и время его добавления (или удаления). Это облегчает поиск нужных сообщений коммитов. В довершение всего, иногда нужно найти коммит, который привел к ошибке или опечатке. Git предлагает специальную команду, которая позволяет вам быстро сосредоточиться на этом коммите.

Чего же мы ждем? Давайте попробуем найти что-нибудь в репозиториях Git!



Выходим на новый уровень	371
Изучение истории коммитов	373
Использование git blame.....	375
git blame и менеджер репозитория Git	376
Поиск в репозитории Git	378
Поиск в репозитории Git с командой grep	379
Параметры команды git grep.....	380
Комбинация флагов команды git grep	381
Что не умеет делать git blame	383
Если копнуть поглубже логи (-S)	384
git log -S или blame?.....	385
Использование флага -p с git log	386
Еще один флаг глубокого поиска с git log (-G).....	389
Окрошка из флагов Git.....	392
Просмотр коммитов.....	395
Отвязка указателя HEAD	396
История о потерянном указателе.....	397
Поиск коммитов с командой git bisect	401
Использование git bisect	402
Завершение git bisect.....	404

Эффективная работа с Git

8

#Советы профессионалов

До сих пор в этой книге вы учились использовать Git. Но вы также можете подчинить Git своей воле. Именно поэтому так важно изучить детали его настройки. Вы уже начали настраивать Git в предыдущих главах, а в этой главе мы рассмотрим гораздо больше настроек, облегчающих вашу жизнь. Вы научитесь также определять ярлыки: долой многословные команды Git!

Вы можете значительно упростить взаимодействие с Git. Я покажу, как заставить Git игнорировать определенные типы файлов, чтобы вы случайно не добавили их в коммит. Вы освоите продуманные способы написания сообщений коммитов и узнаете, как я люблю называть свои ветки. И в заключение мы рассмотрим, как графический пользовательский интерфейс Git может сыграть важную роль в вашем рабочем процессе. #поехали #недождусь

Конфигурирование Git	415
Файл глобальных настроек .gitconfig	416
Конфигурация Git для конкретного проекта	419
Просмотр конфигурации Git	421
Псевдонимы Git как личные пути	423
Настройка поведения псевдонимов Git	424
Как заставить Git игнорировать некоторые файлы	427
Действие файла .gitignore	428
Управление файлом .gitignore	429
Пример файла .gitignore	431
Коммит раньше, коммит чаще	433
Пишите осмысленные сообщения коммитов	435
Анатомия хорошего сообщения коммита	436
Анатомия хорошего сообщения коммита: заголовки	437
Анатомия хорошего сообщения коммита: тело	439
Слишком сложно?	440
Используйте понятные имена веток	442
Работа с графическим интерфейсом	444



Приложение: остатки

Пять важных тем, о которых мы не говорили

Мы рассмотрели много разных тем, и вы почти закончили читать эту книгу. Мне будет вас не хватать, но прежде чем мы расстанемся, было бы неправильно отправить вас в мир без дополнительной подготовки. Git предлагает множество функций, и я не смог бы уместить их все в одной книге. Поэтому я приберег несколько действительно интересных описаний для этого приложения.

#1 Теги (запомни меня навсегда)	456
#2 Команда cherry-pick (копирование коммитов)	457
#3 Тайники (псевдокоммиты)	458
#4 Команда reflog (reference log)	460
#5 Команда rebase (альтернатива merge)	461
Мы не прощаемся!	464

