O'REILLY®

bhv®

Head First

Git

Лучший способ понять Git изнутри

Раджу Ганди

Прокачай свой мозг!

УДК 004.4'23
ББК 32.973.26-018

Г19

Ганди Р.

Г19 Head First. Git: Пер. с англ. — СПб.: БХВ-Петербург, 2023. — 464 с.: ил.

ISBN 978-5-9775-1777-5

Книга поможет быстро и легко изучить самый популярный в мире инструмент контроля версий Git. В ней использована уникальная методика Head First, выходящая за рамки синтаксиса и инструкций по решению конкретных задач, а эффектное визуальное оформление разработано с учетом того, как работает и наиболее продуктивно усваивает информацию мозг.

Рассмотрены основы Git, свойства ветвления кода, слияние, коммиты, устройство репозитория Git и поиск в нем, отмена действий и исправление ошибок. Особое внимание уделено командной работе с Git, передовым методам взаимодействия и советам профессионалов по эффективной работе.

Для программистов

УДК 004.4'23
ББК 32.973.26-018

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Людмила Гауль*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Людмилы Гауль*
Корректор	*Светлана Крутоярова*
Оформление обложки	*Зои Канторович*

«БХВ-Петербург», 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-1-492-09251-3 (англ.)
ISBN 978-5-9775-1777-5 (рус.)

Оглавление (краткое)

Оглавление (с пояснениями)

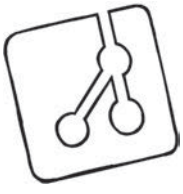
Вступление

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?

1 Знакомство с Git

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?

2 Ветвление кода
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!

3 Осмотритесь вокруг

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

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.

5 Командная работа с Git — часть I
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!

6 Совместная работа с Git — часть II
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!

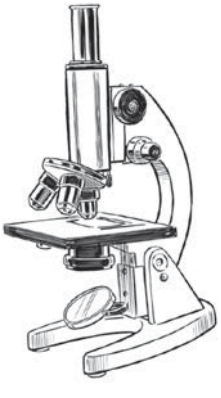
7 Поиск в репозитории Git

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!



8 Эффективная работа с Git #Советы профессионалов

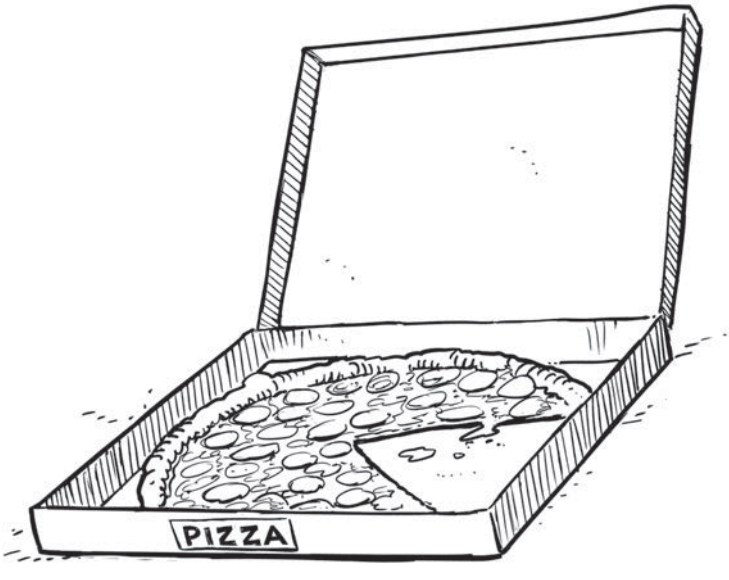
До сих пор в этой книге вы учились использовать Git. Но вы также можете подчинить Git своей воле. Именно поэтому так важно изучить детали его настройки. Вы уже начали настраивать Git в предыдущих главах, а в этой главе мы рассмотрим гораздо больше настроек, облегчающих вашу жизнь. Вы научитесь также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

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.



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

Возможно, вы играли в видеоигры, для прохождения которых требуется больше одного подхода. По мере прохождения игры вы выигрываете и проигрываете несколько сражений и можете приобрести оружие или создать армию. Время от времени вы пытаетесь выполнить определенное задание более одного раза.

Многие игры позволяют сохранять ваши достижения. Допустим, вы только что сразили огненного дракона, и теперь по сюжету игры вам предстоит пробиться к огромной сокровищнице.

Вы решаете на всякий случай сохранить своего персонажа, а затем продолжить приключение. Вы создаете "моментальный снимок"игры в ее нынешнем виде. Хорошая новость заключается в том, что теперь, даже если вас внезапно настигнет безвременная кончина, когда на вас нападут гадкие ящерицы, плюющиеся кислотой, вам не придется возвращаться к самому началу. Вместо этого вы просто загружаете снимок игры и повторяете попытку. Огненный дракон остался позади!

Контроль версий позволяет вам делать то же самое с вашей работой — он дает вам возможность сохранить достижения. Вы можете немного поработать, сохранить проект и продолжить работу. Такой "моментальный снимок" — это способ записи набора изменений, поэтому, даже если вы внесли изменения в кучу файлов в своем проекте, все они сохранятся в одном снимке.

Это означает, что если вы допустили ошибку или, возможно, вас не устраивает текущее решение, вы можете просто вернуться к своему предыдущему снимку. С другой стороны, если вы довольны, вы просто создаете еще один снимок и продолжаете стучать по клавиатуре.

И это еще не все. Система контроля версий, такая как Git, позволяет вам легко работать вместе с коллегами над одним и тем же набором файлов, не наступая друг другу на пятки. Мы подробнее остановимся на этом в последующих главах, а пока вам достаточно просто знать о такой возможности.

Git — это ваш банк памяти, система безопасности и платформа для совместной работы, объединенные в одном инструменте! Понимание принципа контроля версий и Git, в частности, — понимание того, на что он способен и как влияет на рабочие процессы, — сделает вашу работу *по-настоящему* продуктивной.

Поздравляем!

Ваша компания только что получила контракт на создание "ГавЛав" — первого в мире приложения для знакомств среди пушистых друзей человека. Тем не менее конкуренты не дремлют даже в мире собак, и у нас не так много времени, чтобы тратить его впустую!

г. Громколайск,
ул. Коротких Хвостов, 12

Поздравляем, вы победили в конкурсе на разработку лучшего в мире приложения для собачьих знакомств "ГавЛав".

Это приложение позволит вашему пушистому другу расширить круг общения, найти друзей, а может быть, и спутника жизни! Используя самые последние достижения в области машинного обучения и интуитивно понятный интерфейс, специально разработанный с учетом потребностей вашей собаки, мы постараемся стать лидером отрасли в короткие сроки.

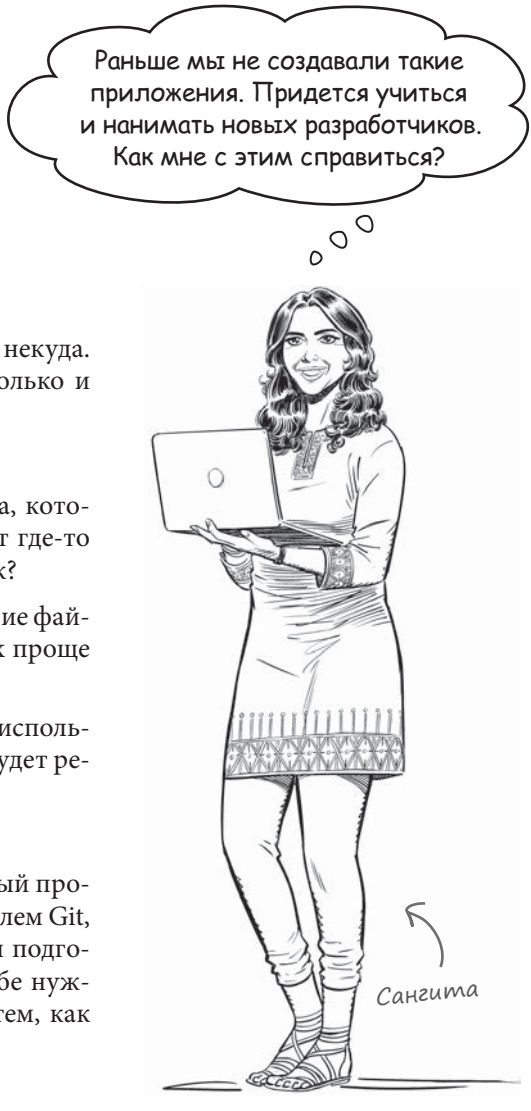
Мы считаем, что правильно рассчитали время для выхода на рынок, но понимаем, что конкуренция высока. Кроме того, это первая попытка сделать что-то подобное, значит, мы должны проверять любые идеи. Мы ожидаем, что будем тесно сотрудничать с вами и вашими разработчиками по мере того, как мы приближаемся к нашему первому релизу.

Мы с нетерпением ждем наброски дизайна и альфа-версию приложения.

Искренне Ваш,

Директор Джонни Гавус

Разговор в офисе разработчиков



Мардж: Нам стоит использовать систему контроля версий.

Сангита: Я слышала о такой системе, но никогда не пользовалась. Вряд ли сейчас подходящее время, чтобы изучать что-то новое.

Мардж: Начать работу с Git — проще некуда. Нужно лишь создать репозиторий Git, только и всего.

Сангита: Что-что нужно создать?

Мардж: Репозиторий — это особая папка, которой управляет Git. Тебе ведь нужно будет где-то разместить все файлы нового проекта, так?

Сангита: Мне нравится хранить все рабочие файлы проекта в одной главной папке. Так их проще найти.

Мардж: Отлично! Когда создашь папку, используй Git, чтобы указать, что именно в ней будет репозиторий. Это очень просто.

Сангита: И зачем это нужно?

Мадж: Всякий раз, когда ты создаешь новый проект, который будет находиться под контролем Git, тебе нужно запустить команду Git, чтобы подготовить папку к работе. Представь, что тебе нужно завести двигатель автомобиля перед тем, как поехать.

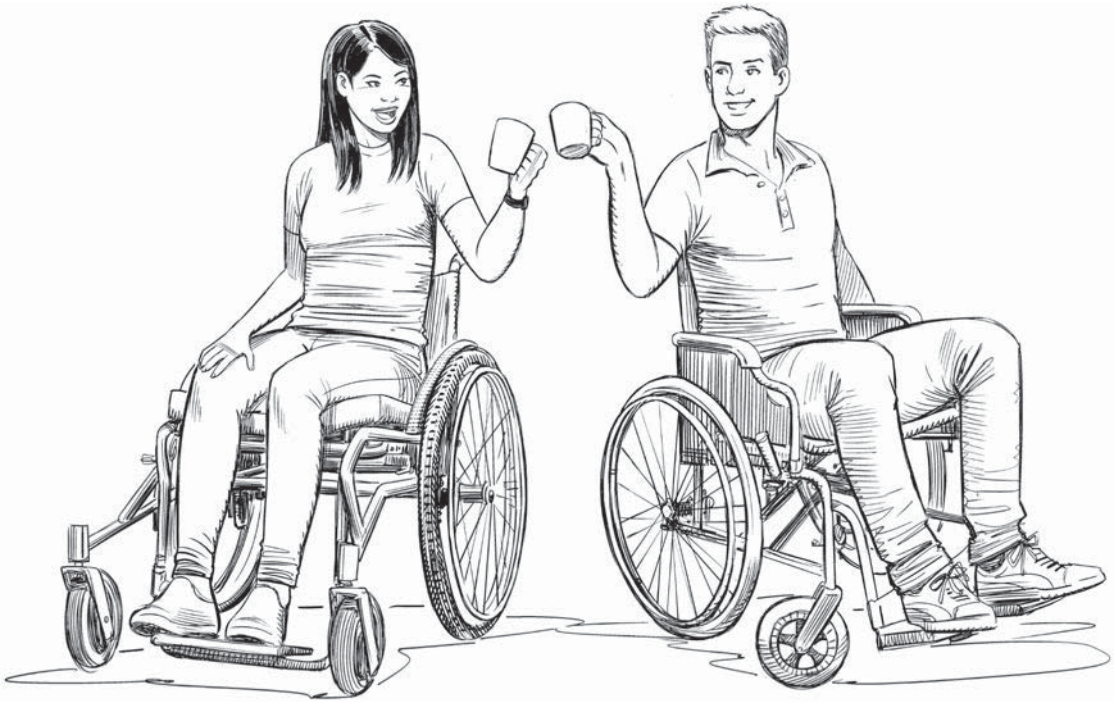
Сангита: Хм-м-м... Ну ладно...

Мардж: Достаточно одной команды, и твоя папка готова к работе с Git. Это так же просто, как повернуть ключ зажигания и завести мотор.

Сангита: Да, звучит неплохо!

Мардж: Позови меня, если тебе что-то понадобится. Я буду поблизости на всякий случай.

Освоим Git?



Мы не сможем двигаться дальше, если у вас не установлен Git. Если вы еще не сделали этого, сейчас самое время. Вернитесь к разделу "Вам потребуется установить Git" во введении, чтобы узнать, как это сделать, и начать работу.

Даже если у вас установлен Git, будет полезно установить его новую версию, чтобы убедиться, что все, о чем мы говорим в этой книге, работает должным образом.

Заводим двигатель...

Любой проект, над которым вы работали, состоит из одного или нескольких файлов — это могут быть файлы исходного кода, файлы документации, сценарии сборки и т. д. Если вы хотите управлять этими файлами с помощью Git, то первым шагом будет создание репозитория Git.

Но что такое репозиторий Git? Вспомните, что одна из причин использования системы контроля версий заключается в том, что мы можем периодически сохранять моментальные снимки нашей работы. Конечно, Git нужно место для хранения этих снимков. Это и есть репозиторий Git.

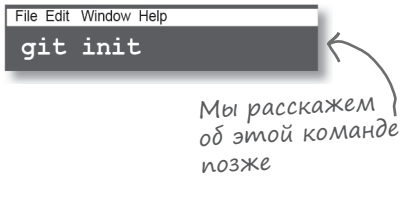
Следующий вопрос — где расположен этот репозиторий? Обычно мы храним все файлы проекта в одной папке. Если мы собираемся использовать Git в качестве нашей системы контроля версий для этого проекта, мы сначала создадим репозиторий в этой папке, чтобы у Git было место для хранения наших снимков. Создание репозитория Git включает запуск команды **`git init`** в верхней папке вашего проекта.

Вскоре мы углубимся в детали, а пока вам достаточно знать, что без создания репозитория Git вы действительно мало что можете сделать с Git.

Независимо от того, насколько велик ваш проект (другими словами, независимо от того, сколько файлов или подкаталогов имеет ваш проект), в **верхней** (или корневой) папке этого проекта необходимо запустить команду **`git init`**, чтобы начать работу с Git.

1. Создайте папку проекта.

2. Инициализируйте Git.



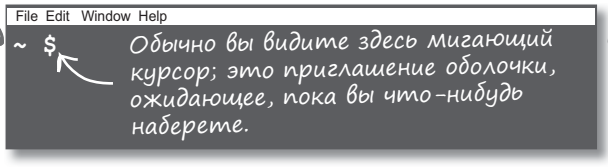
3. Инициализация репозитория Git внутри папки дает ей сверхспособности. Это место часто называют *рабочим каталогом*.

Краткий обзор командной строки: как узнать, где вы находитесь, с помощью pwd

Если вам непонятно, о чем речь, вернитесь к введению. Вы найдете инструкции в разделе "Вам потребуется установить Git".

Во время выполнения упражнений из этой книги вы будете часто использовать командную строку, так что давайте потратим немного времени на то, чтобы освоиться с ней. Начните с запуска терминала, как мы это делали во введении, и перейдите в нужную папку на жестком диске. Напоминаем, что на Mac вы найдете **`Terminal.app`** в папке **`Applications > Utilities`**. В Windows нажмите кнопку **Пуск** главного меню, и вы увидите пункт **Git Bash** в меню **Git**. Вас встретит приглашение, означающее что терминал готов принимать команды.

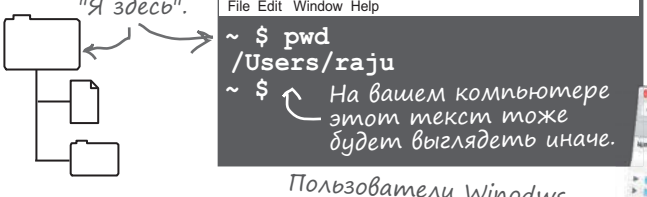
Приглашение может выглядеть иначе, если вы используете другой терминал.



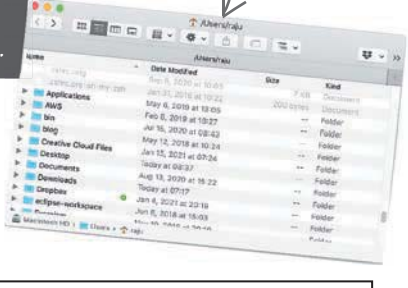
У пользователей Windows это окно Git Bash.

Давайте начнем с чего-нибудь простого. Введите **`pwd`** и нажмите <Enter>; `pwd` означает "печать рабочего каталога" и отображает путь к каталогу, в котором в данный момент работает терминал. Другими словами, если вы создадите новый файл или новый каталог, они появятся в этом каталоге.

pwd значит "Я здесь".



Видите этот путь в верхней части окна? Это эквивалентно pwd в окне Finder или Проводника.



Ответы на задания вы найдете в конце каждой главы.

Пользователи Winodws, когда мы говорим "терминал" — мы подразумеваем... Git Bash!!!

Возьмите карандаш

Давайте потренируемся. Запустите терминал, введите после приглашения команду `pwd` и запишите здесь вывод терминала, который вы получили:

Отлично! Если вы впервые используете терминал или не очень хорошо с ним знакомы, то это задание может показаться немного сложным. Но знайте: мы будем направлять вас на каждом этапе пути не только к этому упражнению, но и ко всем упражнениям в этой книге.

→ Ответ на стр. 78.

Еще про командную строку: создание нового каталога при помощи команды `mkdir`

Знание местоположения текущего каталога в терминале с помощью `pwd` очень полезно, потому что почти все, что вы делаете, относится к текущему каталогу, включая создание новых папок. Но если мы хотим создать новый каталог (папку), то используем команду `mkdir`, что означает make directory (создать каталог).

В отличие от `pwd`, которая просто сообщает вам путь к текущему каталогу, `mkdir` принимает аргумент, который представляет собой имя каталога, который вы хотите создать:

Это эквивалент mkdir в окне Finder или в Проводнике.

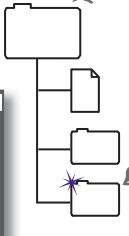
Не делайте этого пока. Сейчас у нас есть для вас другие упражнения.

```
File Edit Window Help
~ $ mkdir created-using-the-command-line
~ $
```

Если все в порядке, вы просто увидите новое приглашение.

Это аргумент, то есть имя новой папки.

Текущий каталог (pwd)

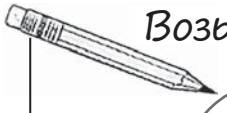


mkdir создает новый каталог внутри текущего каталога (pwd).

Сравните свой ответ с ответом в конце главы.

`mkdir` выдаст ошибку, если вы попытаетесь создать папку, которая уже существует!

Если вы попытаетесь создать папку, которая уже существует в этом месте, `mkdir` просто сообщит `"File exists"` (Файл существует) и не сделает ничего. Обратите внимание, что в данном случае слово `File` обозначает папку, а не файл.



Возьмите карандаш

Напомним, что в Windows терминал — это Git Bash.

Ваш ход. В окне терминала при помощи команды `mkdir` создайте новую папку с названием `my-first-commandline-directory`

Напишите здесь команду и аргумент.

Попробуйте ввести эту команду второй раз. Запишите здесь, что получилось:

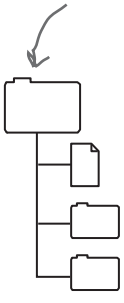
Запишите здесь ошибку.

Ответ на стр. 78.

Еще больше командной строки: список файлов с командой `ls`

Вывод команды `mkdir`, мягко говоря, не очень информативный. Пока вы не совершили никаких ошибок, она молча делает свое дело. Чтобы убедиться, что что-то получилось, вы можете получить список файлов в текущем каталоге. Команда списка называется `ls` (сокращение от list).

Запуск ls здесь означает вывод списка всех файлов и папок в текущем (pwd) каталоге.



Вы можете использовать Finder (Mac) или Проводник (Windows), и увидите то же самое.

```
File Edit Window Help
~ $ ls
Applications  Movies
Desktop       Music
Documents     Pictures
Downloads     bin
Library       created-using-the-command-line
```

Мы обрезали вывод для краткости.

Команда `ls` по умолчанию отображает только обычные файлы и папки. Время от времени нам нужно видеть скрытые файлы и папки. Для этого вы можете поставить `ls` с *флагом*. Флаги, в отличие от аргументов, начинаются с дефиса (чтобы отличить их от аргументов). Чтобы увидеть все файлы и папки (включая скрытые), мы можем использовать флаг `-A` (да-да, заглавная буква "A"), например:

Соблюдайте регистр. После дефиса идет заглавная буква "A".

Флаг "All". Не забывайте про дефис!

Команда ls.

```
ls -A
```

Это вывод. У вас он может выглядеть совсем иначе.

```
File Edit Window Help
~ $ ls -A
.bash_history
.bash_profile
.bashrc
Applications
Desktop
Documents
Downloads
Library
```

У скрытых файлов и папок имя начинается с точки.

Напомним, что мы обрезали вывод для краткости.

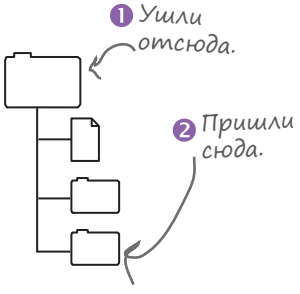
Возьмите карандаш

При помощи терминала получите список файлов и папок в текущем каталоге. Проверьте, видите ли вы в этом списке недавно созданную папку `my-first-commandline-directory`.

Теперь используйте флаг `-A`, чтобы узнать, есть ли в текущем каталоге скрытые файлы и папки.

→ Ответ на стр. 78.

Почти закончили знакомство с терминалом: смена каталога с командой cd



Двигаемся дальше! Мы создали новый каталог, но как нам перейти к нему? Для этого у нас есть команда `cd`, что означает change directory (изменить каталог). Как только мы изменим каталоги, можем использовать `pwd`, чтобы убедиться, что действительно сменили местоположение.

Большинство терминалов поддерживают автозавершение ввода. Введите первые буквы имени каталога, нажмите клавишу <Tab>, и терминал сделает остальное.

Убедитесь, что правильно ввели имя каталога.

```
File Edit Window Help
~ $ cd created-using-the-command-line
~/using-the-command-line $ pwd
~/using-the-command-line $ /Users/raju/created-using-the-command-line
```

В зависимости от своего типа терминал может показать вам полный или сокращенный путь. Вы всегда можете использовать pwd, чтобы проверить, где вы находитесь в данный момент.

Команда `cd` переносит нас в подкаталог текущего каталога. Чтобы вернуться в родительский каталог, мы также можем использовать `cd`, например:

Между cd и двумя точками (..) нужен пробел.

```
cd ..
```

```
File Edit Window Help
~/using-the-command-line $ cd ..
~ $
```

Две точки обозначают родительский каталог. — *Это две точки.*

Всегда следите за рабочим каталогом, используя pwd. Большинство команд работают относительно текущего каталога.

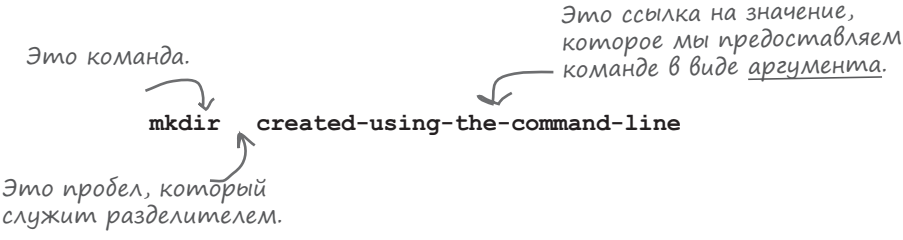
Упражнение

Попробуйте изменить текущий каталог. Используйте `cd`, чтобы перейти в только что созданную папку `my-first-commandline-directory`, затем используйте `pwd`, чтобы убедиться, что вы сменили каталог, и наконец, используйте `cd ..`, чтобы вернуться в родительский каталог. Запишите здесь команды по мере их использования.

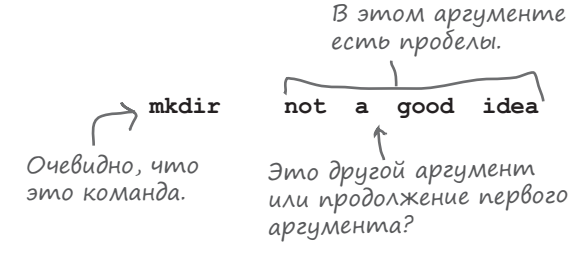
→ Ответ на стр. 79.

Команде нужны аргументы

Функции командной строки, такие как `pwd` и `mkdir`, являются *вызываемыми командами*. Некоторые команды, например `mkdir` и `cd`, ожидают подсказки о том, что вы хотите создать или куда перейти. Этой подсказкой служат *аргументы* команды.



Хотите знать, почему мы решили использовать дефисы вместо пробелов? Оказывается, наличие пробелов в аргументах создает проблемы. В командной строке пробелы отделяют команду от ее аргументов. Если в ваших аргументах тоже есть пробелы, может возникнуть путаница.



В командной строке пробел служит разделителем. Но если пробел окажется в аргументе, командная строка не поймет, передаете ли вы несколько аргументов или один аргумент из нескольких слов. Если у вас есть аргумент, состоящий из нескольких слов или символов, его следует заключить в кавычки.

```
mkdir    "this is how it is done"
```

Теперь понятно, что это аргумент.

Как видите, легко допустить ошибку, если использовать пробелы в аргументах. Советуем избегать пробелов в имени файла или пути к нему.

Например, лучше использовать в качестве пути C:\my-projects\ вместо C:\my projects\

Отличный вопрос.

Командной строке все равно, используете ли вы двойные или одинарные кавычки. Важно лишь соблюдать единообразие. Если вы начинаете имя аргумента с одинарных кавычек, завершите его одинарной кавычкой. Аналогично поступайте с двойными кавычками.

Как правило, большинство людей, использующих командную строку, предпочитают двойные кавычки, и мы тоже; однако есть одна ситуация, когда вы должны использовать двойные кавычки — если ваш аргумент содержит одинарную кавычку.

Обратите внимание, что в данном случае мы используем одинарную кавычку в слове `sangita's`:

```
mkdir "sangita's_home-folder"
```

Чтобы использовать здесь одинарную кавычку, вам нужно заключить аргумент в двойные кавычки.

Обратное также верно: если вам нужно использовать двойные кавычки в своем аргументе, то придется заключить этот аргумент в одинарные кавычки. Однако мы уже говорили, что лучше всего избегать пробелов в аргументах, особенно в именах каталогов и файлов. **Если вам нужен пробел в аргументе, просто используйте дефис или подчеркивание.** Это поможет вам избежать кавычек любого рода при использовании аргументов.

Кто за что отвечает?

В терминале командной строки есть много команд и флагов. В этой головоломке о том, кто за что отвечает, сопоставьте каждую команду с ее описанием.

`cd`	Показывает путь к текущему каталогу.
`pwd`	Создает новый каталог.
`ls`	Переходит в родительский каталог.
`mkdir`	Меняет каталог.
`ls -A`	Выдает список обычных файлов в папке.
`cd ..`	Выдает список **всех** файлов в папке.

→ Ответ на стр. 80.

Сотрите ненужное

Мы закончили работу с этим разделом, и теперь можно удалить созданные вами ненужные папки, такие как `my-first-commandline-directory` и любые другие. Для этого просто воспользуйтесь Проводником Windows или окном Finder и удалите их. Хотя командная строка тоже позволяет сделать это, файлы, удаленные с помощью командной строки, обычно не попадают в корзину. Если вы случайно удалили не ту папку, восстановить ее будет очень сложно!

В будущем, когда вы лучше познакомитесь с командной строкой, вероятно, вы сможете использовать команду для удаления файлов, но пока давайте не будем рисковать.

Создайте свой первый репозиторий

Давайте потратим немного времени на знакомство с Git. Вы его уже установили, а теперь не помешает убедиться, что все настроено, и получить представление о том, что нужно для создания репозитория Git. Для этого вам понадобится окно терминала. Только и всего!

Начните с открытия окна терминала, как мы делали в предыдущем упражнении. Чтобы облегчить работу, мы предлагаем вам создать папку `headfirst-git-samples` для хранения **всех** примеров из этой книги. Внутри нее создайте новую папку для нашего первого упражнения в главе 1 и назовите ее `ch01_01`.

Если вы плохо знакомы с командной строкой, можете использовать Finder (Mac) или Проводник (Windows) для создания новой папки. Тем не менее мы будем часто использовать командную строку, поэтому вам следует научиться с ней работать.

```
File  Edit  Window  Help
headfirst-git-samples $ mkdir ch01_01
headfirst-git-samples $ cd ch01_01
ch01_01 $
```

Начните с создания каталога ch01_01. directory.

Затем перейдите в новый каталог.

cd означает "change directory."

mkdir означает "make directory".

Оказавшись в новом каталоге, создайте свой первый репозиторий Git. Для этого просто выполните команду `git init` внутри только что созданной папки.

Выполните команду init.

Обязательно проверьте регистр. Команды Git всегда пишутся строчными буквами.

```
File  Edit  Window  Help
ch01_01 $ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:    git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:    git branch -m <name>
Initialized empty Git repository in ~/headfirst-git-samples/ch01_01/.git/
ch01_01 $
```

Слово "hint" мы обсудим в следующей главе.

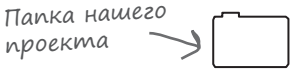
Git сообщает нам, что все в порядке.

Это было совсем не трудно, правда? Теперь у вас есть первый репозиторий Git.

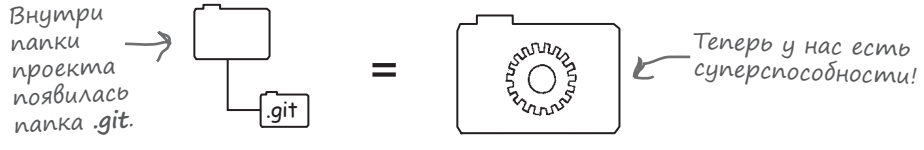
Как работает команда `init`

Итак, что именно мы только что сделали? Команда `git init` выглядит довольно скромно, но она определенно эффективна. Давайте заглянем за кулисы и посмотрим, как она работает.

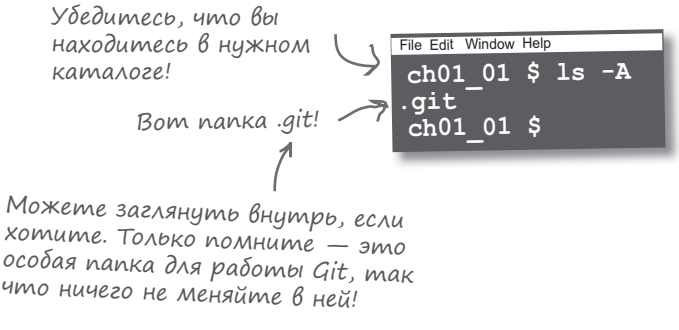
Для этого мы создадим новую пустую папку проекта.



С помощью терминала мы переходим в нужный каталог и произносим волшебное заклинание `git init`, где `init` — это сокращение от ***initialize*** (инициализация). Git понимает, что мы просим его создать репозиторий в этом месте, и в ответ создает скрытую папку с именем `.git` и помещает внутрь нее файлы конфигурации и подпапку, в которой будет сохранять снимки проекта, когда нам это потребуется.



Один из способов убедиться, что команда сработала, — вывести список файлов, расположенных внутри папки проекта:



Эта скрытая папка представляет репозиторий Git. Ее назначение — хранить все, что связано с вашим проектом, включая все коммиты, историю проекта, файлы конфигурации и все, что у вас есть. Она также хранит любую конкретную конфигурацию и настройки Git, которые вы могли сделать для этого конкретного проекта.

Вопрос — ответ

Мне нравится использовать обычный файловый Проводник. Можно ли использовать его для просмотра папки `.git`?

Конечно! По умолчанию стандартный Проводник не показывает скрытые файлы и папки. Вы должны включить отображение скрытых объектов в настройках Проводника.

Что произойдет, если кто-то случайно удалит рабочую папку `.git` проекта?

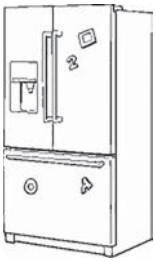
Старайтесь не допускать такого. Эта папка — своего рода "кошелек", в котором Git хранит всю историю проекта и много рабочих файлов, включая настройки Git под ваш проект. Если вы удалите папку `.git`, то потеряете эти данные. Но все остальные файлы вашего проекта останутся нетронутыми.

Что случится, если я случайно выполню `git init` больше одного раза с той же самой папкой?

Хороший вопрос. Ответ — ничего страшного. Git просто сообщит вам, что выполнит повторную инициализацию, но вы ничего не потеряете и не испортите. Попробуйте сделать это с папкой `ch01_01`. Мы только начали работу, вам нечего терять, и это лучшее время для экспериментов.

У других систем управления версиями есть серверный компонент. Нужен ли он здесь?

Начать работу с Git *действительно* легко. Команда `git init` создает репозиторий Git, и можно начинать работу. В конце концов, вам понадобится механизм, позволяющий поделиться своей работой с коллегами, и мы обещаем, что скоро доберемся до этого. Но пока достаточно того, что вы знаете.



Магниты с командами

У нас есть все команды, нужные для создания новой папки, перехода к ней и инициализации для создания нового репозитория Git. Будучи добросовестными разработчиками, мы часто проверяем, находимся ли мы в правильном каталоге. Чтобы помочь нашим коллегам, мы красиво разместили команды на офисном холодильнике с помощью магнитов, но они упали на пол. Ваша задача — собрать их и водрузить на место. Обратите внимание, что некоторые магниты можно использовать многократно.

Разместите здесь магниты правильно.

Ответ на стр. 80.

Расскажите Git о себе

Есть еще один шаг, прежде чем мы приступим к работе с Git и репозиториями. Git ожидает, что вы расскажете ему некоторые подробности о себе. Благодаря этому, когда вы создаете "моментальный снимок" проекта, Git знает, кто его создал. Мы собираемся начать разговор о создании снимков, так что давайте закончим с этим прямо сейчас. Вам нужно предоставить данные только один раз, и они будут применяться ко всем проектам, с которыми вы работаете на своем компьютере.

Мы начнем с нашего старого верного друга — терминала. **Обязательно используйте свои имя и адрес электронной почты вместо наших!** (Мы знаем, что вы нас любите, но не хотим приписывать себе ваши заслуги!) Начните с открытия нового окна терминала. Не беспокойтесь об изменении каталогов — сейчас не имеет значения, где вы запустите терминал.

Запустите терминал и повторяйте за нами.

❶ Сначала сообщите Git свое полное имя.

```
File Edit Window Help
~ $ git config --global user.name "Raju Gandhi"
```

Эту команду можно выполнить в любом каталоге.

Это команда ввода конфигурации.

Вы всегда можете изменить настройки, просто выполнив команду с новыми значениями. Если вы решите использовать рабочий e-mail вместо личного, просто замените его. Сделайте закладку на этой странице книги.

❷ Теперь укажите адрес электронной почты. Пока можно использовать личный адрес; вы сможете изменить его в любое время.

```
File Edit Window Help
~ $ git config --global user.email "me@i-love-git.com"
```

Укажите здесь свой e-mail.

Как нам пригодится Git?

Давайте посмотрим, как выглядит типичное взаимодействие с Git. Помните, мы говорили о видеоиграх, позволяющих сохранять достижения? Так вот, просьба к Git "сохранить ваши достижения" подразумевает отправку вашей работы в Git. По сути, это означает, что Git хранит версию вашей работы. Сделав это, вы можете продолжать спокойно работать, пока не почувствуете, что пришло время сохранить еще одну версию кода, и цикл повторится. Теперь разберемся, как это работает.