1. Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета — а этого вам наверняка хочется — то пользоваться системой контроля версий будет очень мудрым решением. СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое.
2. Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл.
3. зафиксированном, изменённом и подготовленном. "Зафиксированный" значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.
4. Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда git add. Это многофункциональная команда, она используется для добавления под версионный контроль новых файлов, для индексации изменений, а также для других целей, например для указания файлов с исправленным конфликтом слияния
5. Простейший способ зафиксировать изменения — это набрать git commit Помните, что все, оставленное неиндек­сированным, в том числе любые созданные или измененные файлы, для которых после редактирования не была выполнена команда git add, в текущий коммит не войдет.
6. Проверить выбранные настройки позволяет команда git config --list, выводящая список всех обнаруженных в текущий момент параметров
7. Чтобы начать слежение за существующим проектом, перейдите в папку этого про­екта и введите команду $ git init В результате в существующей папке появится еще одна папка с именем .git и всеми нужными вам файлами репозитория — это будет основа вашего Git-репозитория. Пока контроль ваших файлов отсутствует.
8. Чтобы начать слежение за новым файлом, воспользуйтесь командой git add.
9. Теперь, когда область индексирования настроена нужным вам образом, можно зафиксировать внесенные туда изменения. Проще всего осуществить фиксацию командой git commit
10. Основным инструментом определения состояния файлов является команда git status Предположим, вы добавили в проект простой файл README. Если до этого момента он не существовал, команда git status покажет данный неотслеживаемый файл примерно так:   
    $ echo 'My Project' > README

$ git status  
On branch master  
Untracked files:   
(use "git add <file>..." to include in what will be committed)   
README  
nothing added to commit but untracked files present (use "git add" to track)

Понять, что новый файл README является неотслеживаемым, можно по заголовку Untracked files в выводимых командой status данных. Указанное состояние, по сути, означает, что Git видит файл, отсутствующий в предыдущем снимке со­стояния (в коммите); без явного указания с вашей стороны Git не будет добавлять этот файл в число коммитов. Подобный подход гарантирует, что в репозитории не сможет случайно появиться сгенерированный бинарный файл или другие файлы, которые вы не собирались туда добавлять.

1. Бывает так, что некоторый класс файлов вы не хотите ни автоматически до­бавлять в репозиторий, ни видеть в списке неотслеживаемых. В эту категорию, как правило, попадают автоматически генерируемые файлы, например журналы регистрации или файлы, генерируемые системой сборки. В подобных случаях создается файл .gitignore со списком соответствующих паттернов.
2. Если команда git status дает недостаточно подробный, с вашей точки зрения, результат, например если вы хотите не только получить список отредактированных файлов, но и узнать, что именно изменилось, воспользуйтесь командой git diff. Эта команда сравнивает содержимое в рабочей папке и в области индексирования. Она выводит список изменений, которые вы внесли, но пока не проиндек­сировали.
3. Впрочем, в рас­сматриваемом сейчас примере мы будем считать, что последний запуск команды git status показал все файлы как проиндексированные и все готово к фиксации изменений. Проще всего осуществить фиксацию командой git commit. Эта команда открывает выбранный вами текстовый редактор для создания сообщение фиксации. Сообщение фиксации можно задать и в команде commit, поставив перед ним флаг –m.
4. После сохранения нескольких версий файлов или клонирования уже имеющего содержимое репозитория вы, скорее всего, захотите взглянуть на то, что было сде­лано ранее. Базовым и самым мощным инструментом в данном случае является команда git log. По умолчанию при отсутствии параметров команда git log выводит в обратном хронологическом порядке список сохраненных в данный репозиторий версий. То есть первыми показываются самые свежие коммиты. Как видите, рядом с каждым коммитом указывается его контрольная сумма SHA-1, имя и электронная почта автора, дата создания и сообщение о фиксации. У команды git log существует великое множество параметров, позволяющих вывести именно ту информацию, которая вам требуется. Одним из самых полезных является параметр -p, показывающий разницу внесен­ную каждым коммитом. -2 параметр позволяет вывести на экран ту же информацию, но с добавлением внесенных изменений, отображаемых непосредственно после каждого коммита. Это удобно, когда требуется сделать обзор кода или быстро посмотреть, что происходит после серии коммитов, добавленных коллегой. Еще с командой git log могут ис­пользоваться группы суммирующих параметров. К примеру, для получения краткой статистики по каждой версии применяется параметр –stat. Еще одним крайне полезным параметром является --pretty. Он меняет формат вывода информации. Есть несколько предустановленных вариантов. Параметр oneline выводит каждый коммит в одну строку, что весьма удобно при просмотре большого числа коммитов. Параметры short, full и fuller, практически не меняя формат вывода, определяют его детализацию. Наиболее интересен параметр format, позволяющий выводить записи журнала в выбранном вами формате. Это особенно полезно при генерации данных для ма­шинного анализа, ведь формат задается в явном виде, и вы можете быть уверены, что при обновлениях Git он не изменится. Самые распространенные параметры форматирования перечислены в таблице.

|  |  |
| --- | --- |
| **Параметр** | **Описание выводимых данных** |
| %H | Хеш-код коммита |
| %h | Сокращенный хеш-код коммита |
| %T | Хеш-код дерева |
| %t | Сокращенный хеш-код дерева |
| %P | Хеш-код родительских коммитов |
| %p | Сокращенный хеш-код родительских коммитов |
| %an | Имя автора |
|  |  |
| %ae | Электронная почта автора |
| %ad | Дата создания оригинала (формат учитывает –date= option) |
| %ar | Дата создания оригинала, в относительной форме |
| %cn | Имя создателя версии |
| %ce | Электронная почта создателя версии |
| %cd | Дата создания версии |
| %cr | Дата создания версии в относительном формате |
| %s | Комментарий |

1. Необходимость отмены чаще всего возникает при слишком ранней фиксации из­менений, когда вы забыли добавить в коммит какие-то файлы или ошиблись с со­общением фиксации. Для повторного сохранения версии в такой ситуации можно воспользоваться параметром –amend
2. Просмотр уже настроенных удаленных серверов осуществляется командой git remote. Она дает список коротких имен для всех указанных вами областей удален­ной работы. Если репозиторий был клонирован, вы должны увидеть по крайней мере источник, то есть имя, которое Git по умолчанию присваивает клонируемому серверу. Параметр -v позволяет увидеть URL-адреса, которые Git хранит для сокращен­ного имени, используемого при чтении из данного удаленного репозитория и при записи в него. Если репозиториев несколько, они выводятся списком. Это означает, что мы легко можем скачать себе изменения, внесенные любым из этих пользователей. При этом у нас может быть допуск на запись в один или несколько репозиториев из списка, хотя представленный код данную деталь не показывает.
3. Как вы уже видели, извлечение данных из удаленных проектов выполняется такой командой:  
     
   $ git fetch [имя удаленного репозитория]  
     
   Эта команда связывается с удаленным проектом и извлекает оттуда все пока отсутствующие у вас данные. После этого у вас должны появиться ссылки на все ветки удаленного проекта, которые можно подвергнуть слиянию или просмотреть. При клонировании данная команда автоматически добавляет удаленный репози­торий под именем «origin». Соответственно команда git fetch origin извлекает все, что появилось на этом сервере после его клонирования (или после момента последнего извлечения информации). Важно понимать, что команда git fetch помещает все данные в ваш локальный репозиторий, — она не выполняет автома­тическое слияние с ветками, с которыми вы работаете в данный момент, и вообще никак не затрагивает эти ветки. Слияние вы выполните вручную, как только в этом возникнет необходимость. Если же у вас есть ветка, настроенная на слежение за какой-то удаленной вет­кой (подробно эта операция рассматривается в главе 3), команда git pull будет автоматическиизвлекать информацию из удаленной ветки и выполнять сли­яние с текущей веткой. В некоторых случаях такой порядок вещей оказывается проще и удобнее; кроме того, по умолчанию команда git clone автоматически настраивает вашу локальную ветку master на слежение за удаленной веткой master (она может иметь и другое имя) на сервере, с которого вы выполняли клонирование. В общем случае команда git pull извлекает данные с сервера, который вы клонировали, и автоматически пытается слить их с вашим текущим рабочим кодом.
4. Чтобы поделиться результатами своего труда, их нужно отправить в репозиторий. Это делается простой командой git push [имя удаленного сервера] [ветка]. Для отправки ветки master на сервер origin (еще раз напоминаем, что в процессе кло­нирования эти имена присваиваются автоматически) следует написать:  
   $ git push origin master  
     
   Команда сработает только при условии, что клонирование осуществлялось с сер­вера, где у вас есть доступ на запись, и за это время никто не отправлял туда свои данные. Если вы выполнили клонирование одновременно с другим пользователем и он уже отправил результаты своей работы на сервер, ваша попытка отправки данных окончится неудачей. Вам сначала нужно скачать все добавленное этим пользователем и встроить это в свои данные, и только после этого появится возмож­ность воспользоваться командой push. Более подробно процесс отправки данных на удаленные серверы рассматривается в главе 3
5. Создание новой ветки: $ git branch [имя новой ветки]  
   Основы ветвления и слияния  
   Простые случаи ветвления и слияния мы рассмотрим на схемах, которые могут пригодиться вам при решении реальных задач. Последовательность действий будет следующей:  
   Выполняем некие действия на сайте.  
   Создаем ветку для новой истории, над которой тоже нужно работать.  
   В этой ветке тоже производим некие действия.  
   А теперь предположим, что нам позвонили и сообщили о важной проблеме, требу­ющей срочного решения. Поступаем следующим образом:  
   Переключаемся в производственную ветку.  
   Создаем ветку для решения проблемы.  
   После тестирования выполняем слияние побочной ветки и отправляем ее в разработку.  
   Возвращаемся к первоначальной задаче и продолжаем работу.
6. Смена веток: $ git checkout [имя ветки]
7. Нужно выполнить слияние: git merge [ветка с которой хотите сделать слияние]
8. Процесс слияния далеко не всегда проходит гладко. Если в двух ветках, которые вы собираетесь слить, вы внесли разные изменения в один и тот же файл, Git не сможет просто взять и объединить их. Если бы при решении проблемы 53 вы от­редактировали ту же самую часть файла, что и в ветке hotfix, возник бы конфликт слияния:

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

В этом случае Git не может автоматически создать коммит слияния. Система при­останавливает процесс до момента разрешения конфликта. Посмотреть, какие файлы не прошли слияние после возникновения конфликта, позволяет команда git status:

$ git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

Все, что относится к области конфликта слияния, помечено как неслитое (unmerged). Система Git добавляет к проблемным файлам стандартные метки, позволяющие открывать эти файлы вручную и разрешать конфликты. Ваш файл содержит раздел, который выглядит примерно вот так:

<<<<<<< HEAD:index.html

<div id="footer">contact : email.support@github.com</div>

=======

<div id="footer">

please contact us at support@github.com

</div>

>>>>>>>iss53:index.htmlОсновы ветвления и слияния **81**

Версия с указателем HEAD (из вашей ветки master, так как именно в нее вы перешли перед выполнением команды merge) располагается в верхней части бло­ка (то есть выше набора символов =======), а версия из ветки iss53 показана в нижней части. Для разрешения конфликта следует или выбрать одну из версий, или каким‑то образом объединить их. К примеру, можно заменить весь блок вот этим:

<div id=»footer»>

please contact us at email.support@github.com

</div>

В это решение вошло небольшое количество информации из каждой части, а строки <<<<<<<, ======= и >>>>>>> удалены. Разобравшись с каждым таким раз­делом в каждом из проблемных файлов, выполните для каждого из этих файлов команду git add. Индексируя файл, вы помечаете его как неконфликтующий. Для разрешения конфликтов можно вызвать командой git mergetool графический интерфейс:

$ git mergetool

This message is displayed because 'merge.tool' is not configured.

See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:

opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge

araxis bc3 codecompare vimdiff emerge

Merging:

index.html

Normal merge conflict for 'index.html':

{local}: modified file

{remote}: modified file

Hit return to start merge resolution tool (opendiff):

Если вы предпочитаете инструмент слияния, отличный от заданного по умолча­нию (в данном случае система Git запустила утилиту opendiff на Mac), список доступных инструментов выводится после строчки 'git mergetool' will now attempt to use one of the following tools, — достаточно ввести название нужного вам инструмента.

**ПРИМЕЧАНИЕ**

Более специализированные инструменты, предназначенные для разрешения запутанных конфликтов слияния, рассмотрены чуть позже.

После завершения работы с инструментом слияния Git спросит, удалось ли раз­решить конфликты. В случае положительного ответа файл индексируется, что означает разрешение конфликта. Убедиться в том, что это действительно так, по­зволяет команда git status:  
$ git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

modified: index.html

Если вы довольны полученным результатом и удостоверились в том, что все ра­нее конфликтовавшие файлы проиндексированы, остается завершить слияние командой git commit. В этом случае сообщение фиксации по умолчанию выгля­дит так:

Merge branch 'iss53'

Conflicts:

index.html

#

# Кажется, вы можете зафиксировать слияние.

# Если это не так, удалите файл

# .git/MERGE\_HEAD

# и сделайте еще одну попытку.

# Введите сообщение фиксации для ваших изменений. Строки, в начале

# которых '#' игнорируются, а пустое сообщение отменяет коммит.

# On branch master

# All conflicts fixed but you are still merging.

#

# Changes to be committed:

# modified: index.html

#  
Это сообщение можно дополнить информацией о том, как вы разрешили конфликт, если считаете, что это может пригодиться другим пользователям. Если подоплека ваших действий не очевидна, имеет смысл написать, почему вы поступили именно так, а не иначе.

1. Интерфейс GitWeb чрезмерно упрощен. В качестве более современного, полно­функционального Git-сервера можно взять одно из существующих решений с открытым исходным кодом. Наиболее популярным из них является приложение GitLab, поэтому именно его мы рассмотрим в качестве примера. Оно несколько сложнее интерфейса GitWeb и требует куда большей поддержки, но предлагает несоизмеримо более богатую функциональность.