Конспект по Git / **"Что? ~~Где? Когда?"~~ Зачем? Как работает?"**

1. **Что такое система контроля версий и зачем она нужна?**

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.

1. **Как происходит инициализация git?**

С помощью команды $ git init в необходимой папке, которая становится репозиторием

1. **Какие изменения происходят при этом в файловой системе?**

Создается скрытая папка «.git», в которой находятся все файлы Git’a

1. **Из каких логических частей состоит git?**
2. Рабочий каталог
3. Область подготовленных файлов
4. Каталог Git’a

**метки в git**

Как и большинство СКВ, Git имеет возможность помечать (tag) определённые моменты в истории как важные. Как правило, этот функционал используется для отметки моментов выпуска версий (v1.0, и т.п.).

Просмотр имеющихся меток (tag) в Git'е делается просто. Достаточно набрать git tag

Данная команда перечисляет метки в алфавитном порядке; порядок их появления не имеет значения.

Git использует два основных типа меток: **легковесные и аннотированные**. Легковесная метка — это что-то весьма похожее на ветку, которая не меняется — это просто указатель на определённый коммит. А вот аннотированные метки хранятся в базе данных Git'а как полноценные объекты. Они имеют контрольную сумму, содержат имя поставившего метку, e-mail и дату, имеют комментарий и могут быть подписаны и проверены с помощью GNU Privacy Guard (GPG). Обычно рекомендуется создавать аннотированные метки, чтобы иметь всю перечисленную информацию; но если вы хотите сделать временную метку или по какой-то причине не хотите сохранять остальную информацию, то для этого годятся и легковесные метки.

**Создание аннотированной метки:** git tag -a {название метки, например v1.4} -m '{сообщение метки, например, my version 1.4}' [тут можно указать хэш коммита, если нужно поставить метку не на текущий]

**Создание легковесная метки.** В сущности, это контрольная сумма коммита, сохранённая в файл — больше никакой информации не хранится. Для создания легковесной метки не нужно передавать аргументы -a, -s и -m: git tag {имя метки}

**отправить метки в удаленный репозиторий**

Это делается так же, как и выкладывание в совместное пользование удалённых веток — нужно выполнить git push origin [имя метки].

Если у вас есть много меток, которые хотелось бы отправить все за один раз, можно использовать git push origin --tags. В таком случае все ваши метки отправятся на удалённый сервер (если только их уже там нет).

**просмотреть удаленный репозиторий**

Чтобы просмотреть, какие удалённые серверы у вас уже настроены, следует выполнить команду **git remote –v**

Единственный способ прямого "просмотра" удаленного репозитория - войти в сервер, на котором он находится, и сделать это или настроить gitweb или подобное.

Также можно отобразить только удаленные ветки. Нужно сначала выбрать их (чтобы убедиться, что у вас есть локальная ссылка на них), а затем перечислить:

git fetch

git branch -r

**Добавить удаленный репозиторий**

**remote add [сокращение] [url]** например: **remote add newr git@github.com:mojombo/grit.git чтобы вносить изменения нужно обязательно указывать репозиторий как SSH-ссылку**

Чтобы извлечь все из удаленного репозитория, нужно ввести **git fetch** newr

После этого, ветка master из репозитороия newr доступна локально как newr/master. Вы можете слить (**git** **merge**) её в одну из своих веток или перейти на эту ветку (**checkout** newr master), если хотите её проверить. Также действует **pull**

**Удаление удаленного репозитория git remote rm p**

**отправить изменения на удаленный репозиторий**

Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий. Команда для этого действия простая: **git push [удал. сервер] [ветка].** Чтобы отправить вашу ветку master на сервер origin, вы можете выполнить следующую команду для отправки наработок на сервер:

**git push origin master**

**Работа с коммитами.**

команда Git для записи индексированных изменений в репозиторий. Коммит будет включать текущие состояния индексированных файлов плюс последние сохраненные состояния неиндексированных (но отслеживаемых) файлов. Обратите внимание: коммит включает в себя не изменения (дельты, патчи) относительно предыдущего коммита, а "снимок" (англ. shapshot) текущего состояния рабочей области.

**команда git add .** (нужна для индексирования файлов перед коммитом)

Команда git add принимает параметром путь к файлу или каталогу. Если это каталог, команда рекурсивно добавляет (индексирует) все файлы в данном каталоге.

**Ветвление.**

Новая ветка: $ git branch <имя\_ветки>

Склонировать ветку: git clone --branch=branch-name http://whatever.git

Удалить ветку локально: $ git branch -d <имя\_ветки>

Удалить ветку в репозитории: $ git push origin --delete <имя\_ветки>

Переименовать ветку: git branch -m <oldname> <newname>

Переместить ветку: git branch -f <имя ветки> <куда переместить, можно указать хэш коммита или относительную ссылку, например, относительная ссылка HEAD~3 обозначает перемещение на три родителя назад от HEAD >

задать ветку как ветку по умолчанию для git pull git push с помощью git branch --set-upstream-to remoteRep/remoteBranch находясь на этой ветке.

**Что такое режим Detaching HEAD?**

HEAD - это символическое имя текущего выбранного коммита — это, по сути, тот коммит, над которым мы в данный момент работаем. HEAD всегда указывает на последний коммит из локального дерева.

Обычно HEAD указывает на имя ветки.

Переходим к сути вопроса:

Отделение (detaching) HEAD означает лишь присвоение его не ветке, а конкретному коммиту с помощью того же checkout

**Не перемещайте коммиты, которые вы уже отправили в публичный репозиторий.**

Перемещение работает следующим образом: находится общий предок для двух веток (на которой вы находитесь сейчас и на которую вы выполняете перемещение); для каждого из коммитов в текущей ветке берётся его дельта и сохраняется во временный файл; текущая ветка устанавливается на тот же коммит, что и ветка, на которую выполняется перемещение; и, наконец, одно за другим применяются все изменения. Рисунок 3-29 иллюстрирует этот процесс.



вы можете вызвать интерактивный диалог передав параметр '-i' или '--interactive' в команду 'git rebase'. Это вызовет интерактивный режим выполнения операции ребазирования

Как только вы выполнили команду 'rebase -i', тут же откроется ваш редактор по умолчанию

это даст вам одну линию на каждый коммит следующего формата:

(action) (partial-sha) (short commit message)

Теперь, вы можете изменить действие (по умолчанию 'pick'-выбрать) на 'edit'-редактировать или 'squash'- сдавить или просто оставить это как 'pick'. Вы можете также переупорядочить коммиты просто передвигая линии так как вам этого хочется. Затем, когда вы выйдете из редактора, git попытается уложить коммиты тем образом каким вы это определили и выполнить действия определенные вами.

Если определен 'pick', то он просто попробует применить патч и сохранить коммит с тем же сообщением-описанием как и было до этого.

Если определено 'squash', то он будет комбинировать это коммит с предыдущим чтобы создать новый коммит. Вы опять попадете в редактор чтобы объединить оба сообщение-описания обоих коммитов сложив их вместе.

Если определен 'edit', то процесс будет идти также, но приостановится перед тем как двигаться к следующему вас выбросит в коммандую строку и вы сможете изменить коммит, или изменить содержимое коммита некоторым образом.

Если вы хотите разделить коммит, например, вы определите 'edit' для этого коммита:

И затем когда вы попадете в командную строку, вы вернетесь к этому коммиту и создадите два (или более) новых.

Последняя вещь которую интерактивное выполнение ребазирования может делать для вас - это сбрасывать коммиты. Если вместо того чтобы выбрать 'pick', 'squash' или 'edit' для линии коммита, вы просто удалите линию, то это удалит коммит из истории.

**Изменение сообщений нескольких коммитов**

В Git отсутствуют инструменты для изменения истории, но вы можете использовать команду rebase, чтобы перебазировать группу коммитов туда же на HEAD, где они были изначально, вместо перемещения их в другое место. С помощью интерактивного режима команды rebase, вы можете останавливаться после каждого нужного вам коммита и изменять сообщения, добавлять файлы или делать что-то другое, что вам нужно. Вы можете запустить rebase в интерактивном режиме, добавив опцию -i к git rebase. Вы должны указать, какие коммиты вы хотите изменить, передав команде коммит, на который нужно выполнить перебазирование.

Например, если вы хотите изменить сообщения последних трех коммитов, или сообщение какого-то одного коммита этой группы, то передайте как аргумент команде git rebase -i родителя последнего коммита, который вы хотите изменить – HEAD~2^ или HEAD~3. Может быть, проще будет запомнить ~3, так как вы хотите изменить последние три коммита; но не забывайте, что вы, в действительности, указываете четвертый коммит с конца – родителя последнего коммита, который вы хотите изменить:

$ git rebase -i HEAD~3

Напомним, что это команда перебазирования – каждый коммит, входящий в диапазон HEAD~3..HEAD, будет изменен вне зависимости от того, изменили вы сообщение или нет. Не включайте в такой диапазон коммит, который уже был отправлен на центральный сервер: сделав это, вы можете запутать других разработчиков, предоставив вторую версию одних и тех же изменений.

Выполнение этой команды отобразит в вашем текстовом редакторе список коммитов.

Команда rebase в интерактивном режиме предоставит вам скрипт, который она будет выполнять. Она начнет с коммита, который вы указали в командной строке (HEAD~3) и повторит изменения, внесенные каждой из коммитов, сверху вниз. Наверху отображается самый старый коммит, а не самый новый, потому что он будет повторен первым.

Вам необходимо изменить скрипт так, чтобы он остановился на коммите, который вы хотите изменить. Для этого измените слово ‘pick’ на слово ‘edit’ напротив каждого из коммитов, после которых скрипт должен остановиться. Например, для изменения сообщения только третьего коммита, измените файл следующим образом:

Когда вы сохраните сообщение и выйдете из редактора, Git переместит вас к самому раннему коммиту из списка и вернет вас в командную строку со следующим сообщением:

$ git rebase -i HEAD~3

Stopped at f7f3f6d... changed my name a bit

You can amend the commit now, with

git commit --amend

Once you’re satisfied with your changes, run

git rebase --continue

Эти инструкции говорят вам в точности то, что нужно сделать. Введите

$ git commit --amend

Измените сообщение коммита и выйдите из редактора. Затем выполните

$ git rebase --continue

Эта команда автоматически применит два оставшиеся коммита и завершится. Если вы измените ‘pick’ на ‘edit’ в других строках, то можете повторить эти шаги для соответствующих коммитов. Каждый раз Git будет останавливаться, позволяя вам исправить коммит, и продолжит, когда вы закончите.

**Объединение коммитов**

С помощью интерактивного режима команды rebase также можно объединить несколько коммитов в один.

Если вместо “pick” или “edit” вы укажете “squash”, Git применит изменения из текущего и предыдущего коммитов и предложит вам объединить их сообщения.

Когда вы сохраните скрипт и выйдете из редактора, Git применит изменения всех этих коммитов и затем вернет вас обратно в редактор, чтобы вы могли объединить сообщения коммитов:

После сохранения сообщения, вы получите один коммит, содержащий изменения всех трех коммитов, существовавших ранее.

**Разбиение коммита**

Разбиение коммита отменяет его и позволяет затем по частям индексировать и фиксировать изменения, создавая таким образом столько коммитов, сколько вам нужно. Например, предположим, что вы хотите разбить средний коммит на три. Вы можете добиться этого, изменив в скрипте rebase -iинструкцию для разбиваемой коммита на “edit”:

Затем, когда скрипт вернет вас в командную строку, вам нужно будет отменить индексацию изменений этого коммита, и создать несколько коммитов на основе этих изменений. Когда вы сохраните скрипт и выйдете из редактора, Git переместится на родителя первого коммита в вашем списке, применит первый коммит (f7f3f6d), применит второй (310154e), и вернет вас в консоль. Здесь вы можете отменить коммит с помощью команды git reset HEAD^, которая, фактически, отменит этот коммит и удалит из индекса измененные файлы. Теперь вы можете добавлять в индекс и фиксировать файлы, пока не создадите требуемые коммиты, а после этого выполнить команду git rebase --continue:

**Удаление коммитов**

Вы также можете использовать интерактивное перебазирование для переупорядочивания или полного удаления коммитов. Если вы хотите удалить коммит и изменить порядок, в котором были внесены другие коммиты, то вы можете изменить порядок следования в скрипте и удалив ненужные коммиты.

Когда вы сохраните скрипт и выйдете из редактора, Git переместит вашу ветку на родителя этих коммитов, применит первый коммит, затем еще один и еще, и после этого остановится, не переходя к удаленным коммитам.

**Отмена/удаление коммитов**

**git reset** отменяет изменения, перенося ссылку на ветку назад, на более старый коммит. Это своего рода "переписывание истории"; git reset перенесёт ветку назад, как будто некоторых коммитов вовсе и не было.

Reset работает на локальных ветках, в локальных репозиториях. Но этот метод переписывания истории не сработает на удалённых ветках, которые используют другие пользователи.

команда git reset HEAD~1 отменит один коммит в текущей ветке (один от HEAD).

**git reset знает несколько "режимов"**

**мягкий**

При использовании git reset --soft HEAD~1 отменяется последний коммит, но изменения файла остаются в рабочем дереве. Также изменения останутся в индексе, поэтому выполнение git commit создаст коммит с теми же изменениями, что и ранее удаленный.

**смешанный**

Это режим по умолчанию и очень похож на мягкий. При "удалении" коммита с помощью git reset HEAD~1 вы все равно сохраните изменения в рабочем дереве, но не в индексе; так что если вы хотите "повторить" индексацию, вам нужно будет добавить изменения (git add) перед коммитом.

**жесткий**

При использовании git reset --hard HEAD~1 вы потеряете все незафиксированные изменения в дополнение к изменениям, внесенным в последний коммит. Изменения не останутся в вашем рабочем дереве, поэтому выполнение команды git status скажет вам, что в вашем хранилище нет никаких изменений.

Если набрать просто **git reset без всяких аргументов**, что сделает git по умолчанию?

Без опции "git reset" интерпретируется как "git reset --mixed HEAD".

git reset имеет три режима: мягкий, смешанный и жесткий (по умолчанию "смешанный" ).

Как и многие другие команды git, git reset принимает аргумент, который является ссылкой на коммит. По умолчанию, если ссылка не указана, используется HEAD. По сути, команда в целом отменяет индексацию со времени последнего коммита.

**git revert**

Чтобы отменить изменения на удаленных репозиториях, надо использовать **git revert**.

revert действует иначе, чем reset - он создает новый коммит, который содержит изменения, полностью противоположные тем, что сделаны в коммите, изменения которого мы хотим "отменить".

После revert можно сделать push и поделиться отменёнными изменениями с остальными пользователями.

команда git reverb HEAD создаст новый коммит, который будет содержать изменения, полностью противоположные тем, что сделаны в одном коммите в текущей ветке (одном от HEAD)

команда git reverb HEAD~2 создаст новый коммит с изменениями, противоположными второму от текущего положения HEAD

**Логи Git**

**git log -3** вывести последние 3 коммита

**git log -p -2** вывести последние 2 коммита с разницей между ними

**Чистка репозитория:**

Это приведет к удалению всех файлов, которые вы могли создать с помощью git add:

**git reset**

Это вернет все локальные незафиксированные изменения (должны быть выполнены в корне репозитория):

**git checkout .**

Вы также можете отменить незафиксированные изменения только для определенного файла или каталога:

**git checkout [some\_dir|file.txt]**

Еще один способ отменить все незафиксированные изменения (дольше печатать, но работает из любого подкаталога):

**git reset --hard HEAD**

Это удалит все локальные неотслеживаемые файлы, поэтому остаются только файлы, отслеженные git:

**git clean -fdx**

ВНИМАНИЕ: -x также удалит все пропущенные файлы, в том числе указанные .gitignore ! Вы можете использовать -n для предварительного просмотра файлов, которые будут удалены.

**Ссылки:**

<https://htmlacademy.ru/blog/99-github-as-hosting>  -  использование github как хостинга

<https://www.youtube.com/watch?v=iQqDce_9y3k> - Как пользоваться GIT в Phpstorm. Инструкция  
<https://www.youtube.com/watch?v=MnU1U7GCWLk>  - Работа с SSH

[https://habr.com/ru/company/mailru/blo.../267595/#1](https://habr.com/ru/company/mailru/blog/267595/#1)  
[https://htmlacademy.ru/blog/86-useful-...g-with-git](https://htmlacademy.ru/blog/86-useful-commands-for-working-with-git)

[https://webdevkin.ru/posts/raznoe/izuc...inayushhix](https://webdevkin.ru/posts/raznoe/izuchaem-git-merge-vs-rebase-dlya-nachinayushhix)   
<https://tonyganch.io/git/rebase/>  
<https://habr.com/ru/post/195674/>

Интерактивная игра по git - [https://learngitbranching.js.org](https://learngitbranching.js.org/). Раскрывает много неочевидных моментов  
Официальная документация по git <https://git-scm.com/book/ru/v2>

<https://habr.com/post/231839/>  6 мифов, мешающих разработчикам использовать Git