

Наиболее полезные конструкции системы контроля версий Git

Содержание

1	Термины и определения	1
2	Фундаментальные концепции	1
3	Конструкции Git	2
3.1	Настройка Git	2
3.2	Добавление файлов в область индексирования	2
3.3	Фиксация изменений	2
3.4	Удаление файлов	3
3.5	Переименование файлов	3
3.6	Просмотр истории коммитов	3
3.7	Отмена индексирования	4
3.8	Работа с удаленными репозиториями	4
3.9	Работа с тегами	5
3.10	Работа с ветками	5
3.11	Отправка данных на удаленный репозиторий	7
3.12	Перемещение данных	7
3.13	Перемещение отдельного коммита	8
3.14	Удаление коммитов	8
3.15	Просмотр информации по коммитам	8
3.16	Ссылки на предков	8
3.17	Диапазоны коммитов	9
3.18	Скрытие и очистка	9
3.19	Принудительно перезаписать локальные файлы	10
	Список литературы	10

1. Термины и определения

HEAD – специальный *указатель* на текущую *локальную ветку*, которая в свою очередь ссылается на последнее зафиксированное состояние, т.е. на *последний коммит*.

2. Фундаментальные концепции

При *слиянии* веток сначала нужно перейти в ту ветку, в которую требуется слить данные, а затем применить команду `git merge`, т.е.

```
git checkout master
git merge server
```

При *перемещении*¹ данных из одной ветки в другую следует сначала перейти в ту ветку, из которой требуется перенести данные, а затем воспользоваться `git rebase`, т.е.

```
git checkout experiment
git rebase master
```

Общая схема работы в небольшой команде:

- Некоторое время вы работаете в тематической ветке (например, `issue54`), и когда приходит время, сливаете результаты своего труда в ветку `master`

```
git checkout master
git merge issue54
```

- Решив, что пришло время поделиться своими наработками с коллегами, вы скачиваете данные с сервера (`git fetch origin`), и если там появились изменения, сливаете к себе ветку `origin/master`, т.е. `git merge origin/master`,
- После чего содержимое ветки `master` можно отправить на сервер `git push origin master`.

3. Конструкции Git

3.1. Настройка Git

Задать глобальные настройки можно следующим образом

```
git config --global user.name "[name]"
git config --global user.email "[email address]"
```

Для того чтобы Git при слияниях, которые сопровождаются разрешением конфликтов, использовал кэш следует воспользоваться конструкцией

```
git config --global rerere.enabled true
```

3.2. Добавление файлов в область индексирования

```
git add file_name.py
git add .
```

3.3. Фиксация изменений

Зафиксировать измененное состояние

```
git commit -m 'Initial commit'
```

Зафиксировать измененное состояние, пропустив область индексирования

```
git commit -a -m 'Some comment'
```

Исправить комментарий последнего коммита. Комментарий последнего коммита будет перезаписан

```
git commit -m 'New some comment' --amend
```

Чтобы исправить комментарий коммита (или несколько коммитов), созданного некоторое время назад (т.е. комментарий не последнего коммита) следует перейти в интерактивный режим с помощью команды

```
git rebase -i HEAD~15
```

¹Т.е. чтобы повторить изменения из одной ветки в другой

затем в открывшемся файле заменить «pick» на «reword» (изменить комментарий коммита), сохранить файл и закрыть его. Далее для каждого коммита (помеченного «reword») можно будет исправить комменатрий. Затем следует сохранить файл и закрыть его. В завершении требуется залить данные на удаленный сервер в принудительном режиме, т.е.

```
git push --force
```

После нужно принудительно обновить удаленный репозиторий

```
git push origin master --force
```

3.4. Удаление файлов

Удалить файл из *области индексирования* и заодно удалить указанный файл из рабочей папки. Чтобы система Git перестала работать с файлом, его нужно удалить из числа отслеживаемых (точнее, убрать из области индексирования) и зафиксировать данное изменение

```
$ git rm file_name.py
```

Удалить файл из области индексирования², но оставить его в рабочей папке. Данная команда в отличие от `git reset HEAD file_name.py` может использоваться как до первой фиксации (`git commit`), так и после

```
git rm --cached file_name.py
```

Удалить все файлы с расширением `.log`³ из директории `log/`

```
git rm log/*.log
```

3.5. Переименование файлов

Переименовать файл

```
git mv old_file_name new_file_name
```

Переименовать файл с использованием `{..}`

```
git mv test_file{,_new}.py
```

3.6. Просмотр истории коммитов

Вывести историю коммитов

```
git log
```

Вывести историю коммитов, ограничившись последними двумя, с указанием разницы, которую внес каждый коммит

```
git log -p -2
```

Вывести историю коммитов с краткой статистикой

```
git log --stat
```

Вывести историю коммитов с указанием сокращенного варианта хеш-кода коммита и комментария

```
git log --pretty=format:'%h %s'
```

Вывести историю коммитов за последние 2 недели

²Git перестает следить за файлом, т.е. он становится *неотслеживаемым*!

³Символ `*` экранируется

```
git log --since=2.week
```

Вывести историю коммитов с захватом интересующего слова в коммите, ограничившись последними двумя

```
git log --grep='key word' -2
```

Вывести историю коммитов, которые попали в заданный временной диапазон

```
git log --since='2020-03-01 10:00' --before ='2020-03-01 11:00'
```

Вывести историю коммитов с указанием сокращенного хеш-кода коммита, тегов, текущей ветки и собственно коммита

```
git log --oneline
```

Вывести историю коммитов, показывая места расположения указателей и точек расхождения

```
git log --oneline --decorate --all --graph
```

Отобразить только те не подвергавшиеся слиянию коммиты из ветки `origin/master`, которых нет в ветке `issue54`

```
git log --no-merges issue54..origin/master
```

Вывести информацию о том чем ветка `origin/master` будет отличаться от ветки `master`⁴ (каких коммитов нет в ветке `origin/master`)

```
git log origin/master..master -p
```

Еще данный синтаксис часто используется для просмотра информации, которую вы собираетесь отправить на удаленный сервер

```
git log origin/master..HEAD
```

или короткий вариант

```
git log origin/master..
```

так как Git вместо пропущенного фрагмента подставляет `HEAD`.

Вывести информацию из журнала ссылок⁵

```
git log -g master
```

3.7. Отмена индексирования

Отменить индексирование файла (файл удаляется из области индексирования). Данная команда может применяться только после первой фиксации (`git commit`)

```
git reset HEAD file_name.py
```

3.8. Работа с удаленными репозиториями

Добавить удаленный репозиторий под коротким именем `pb`. Теперь вместо полного URL можно использовать имя `pb`

```
git remote add pb https://github.com/paulboone/ticgit
```

⁴Этот прием бывает полезен тогда, когда требуется предварительно посмотреть данные, которые будут слиты в ветку

⁵Этот способ работает только для данных, которые все еще находятся в журнале ссылок, поэтому его невозможно использовать для просмотра коммитов, возраст которых превышает несколько месяцев

Извлечь данные из удаленного репозитория. Эта команда связывается с удаленным проектом и извлекает оттуда все пока отсутствующие в локальном репозитории данные. Она *не выполняет* автоматического слияния с ветками, и вообще никак не затрагивает эти ветки

```
git fetch origin
```

Отправить данные локальной ветки **master** на удаленный репозиторий **origin**

```
git push origin master
```

Передать данные от локальной ветки **serverfix** в ветку **awesomebranch** на удаленном репозитории

```
git push origin serverfix:awesomebranch
```

Вывести информацию о конкретном удаленном репозитории **origin**

```
git remote show origin
```

Изменить имя удаленного репозитория с **pb** на **paul**. Теперь к ветке **pb/master** нужно будет обращаться по имени **paul/master**

```
git remote rename pb paul
```

Удалить ссылку на удаленный репозиторий

```
git remote rm paul
```

3.9. Работа с тегами

Вывести список доступных тегов

```
git tag
```

Вывести список тегов, отвечающих поисковому шаблону

```
git tag -l 'v1.8.*'
```

```
git tag -l 'v0.2*.*'
```

Создать тег с комментарием. Тег привязывается к последнему коммиту

```
git log -a v1.4 -m 'My version 1.4'
```

Вывести информацию по тегу

```
git show v1.4
```

Создать легковесный тег (просто не указываются **-a**, **-s**, **-m**)

```
git tag v1.4-lw
```

Отправить все теги на удаленный репозиторий. По умолчанию команда **git push** не отправляет теги на удаленный репозиторий

```
git push origin --tags
```

3.10. Работа с ветками

Вывести список существующих веток

```
git branch
```

Создать новую ветку

```
git branch testing
```

Переключиться на новую ветку

```
git checkout testing
```

Создать новую ветку и тут же переключиться на нее

```
git checkout -b iss53
```

Внедрить внесенные изменения в готовый код

```
git merge hotfix
```

Удалить ветку

```
git branch -d hotfix
```

Вывести ветки, НЕ объединенные с текущей веткой

```
git branch --no-marged
```

Создать *локальную копию ветки* **serverfix** на основе *удаленной ветки* **origin/serverfix**. В результате будет получена локальная ветка, которая начинается там же, где и ветка **origin/serverfix**

```
git checkout -b serverfix origin/serverfix
```

или альтернативный вариант

```
git checkout --track origin/serverfix
```

Создать локальную копию ветки с именем **sf** на основе удаленной ветки **origin/serverfix**. Теперь локальная ветка **sf** поддерживает автоматический обмен данными с удаленной веткой **origin/serverfix**

```
git checkout -b sf origin/serverfix
```

Вывести только те коммиты, которых нет в ветке **master**

```
git log master..contrib
```

или так

```
git log contrib --not master
```

или так

```
git log ^master contrib
```

Вывести только те наработки из *тематической ветки*, которые появились там после расхождения с веткой **master**

```
git diff master...contrib
```

Вывести изменения, которые присутствуют только в ветке **master**

```
git diff origin/master..master
```

Для обращения к существующей ветке можно использовать краткую форму **@{u}**. К примеру, если мы следим из ветки **master** за веткой **origin/master**, то для краткости можно писать так

```
git merge @{u}
```

вместо

```
git merge origin/master
```

Вывести список веток *наблюдения*. Все цифры представляют собой показатели, зафиксированные в момент последнего скачивания данных с каждого сервера. Данная команда не обращается к серверам, а просто сообщает локальные данные из кэша. Для получения актуальной информации о количестве новых коммитов на локальных и удаленных ветках следует извлечь данные со всех удаленных серверов и только затем воспользоваться этой командой, т.е.

```
git fetch --all
git branch -vv
  iss53 7e424c3 [origin/iss53: ahead 2] forgot the brackets
  master 1ae2a45 [origin/master] deploying index fix
  serverfix 5ea463a [teamone/server-fix-good: ahead 3, behind] this should do it
  ...
```

3.11. Отправка данных на удаленный репозиторий

Для того чтобы отправить данные из локального репозитория на удаленный следует использовать конструкцию

```
git push origin master
```

но предварительно необходимо слить данные из удаленного репозитория с помощью команды

```
git pull origin master --allow-unrelated-histories
```

3.12. Перемещение данных

Изменения, зафиксированные в одной ветке, повторить в другой ветке (в **Git** это называется *перемещением*). Например, чтобы повторить изменения из ветки **experiment** в ветке **master**, следует сначала перейти в ту ветку, из которой требуется перенести изменения (ветка **experiment**), а затем воспользоваться командой **git rebase**⁶

```
git checkout experiment
git rebase master
```

Внести изменения клиентской части (ветка **client**) в окончательную версию кода (ветка **master**), оставив изменения серверной части (ветка **server**) для дальнейшего тестирования. Другими словами, взять изменения клиентской части, не связанные с изменениями на серверной стороне, и воспроизвести их в ветке **master** можно следующим образом⁷

```
git rebase --onto master server client
```

Переместить изменения из ветки **server** в ветку **master**, вне зависимости от того, в какой ветке вы находитесь, позволяет команда **git rebase [main_branch] [topic_branch]**. Эта команда переключает на тематическую ветку (в данном случае – на ветку **server**) и воспроизводит ее содержимое в основной ветке (**master**)

```
git rebase master server
```

Замечание

При перемещении изменений из одной ветки в другую, нужно перейти на ту ветку, *из которой* планируется переместить изменения

⁶Работает это следующим образом: ищется общий предок двух веток (текущей ветки и ветки, в которую выполняется перемещение), вычисляется разница, вносимая каждым коммитом текущей ветки, и сохраняется во временных файлах. После этого текущая ветка сопоставляется тому же коммиту, что и ветка, в которую осуществляется перемещение, и одно за другим происходят все изменения

⁷По сути, команда приказывает «перейти в ветку **client**, найти исправления от общего предка веток **client** и **server** и повторить их в ветке **master**»

3.13. Перемещение отдельного коммита

Взять представленные в коммите изменения и попытаться применить их в текущей ветке. Команда извлечет изменения, появившиеся в коммите, но при этом изменится контрольная сумма SHA-1 коммита, так как у него другая дата применения

```
git cherry-pick e43a6fd3e9488...
```

3.14. Удаление коммитов

Для того чтобы удалить последний коммит следует сначала удалить коммит в локальном репозитории

```
git rebase -i HEAD~2
```

а затем отправить данные в форсированном режиме на удаленный репозиторий

```
git push origin +master --force
```

Замечание

После удаления коммита или после изменения комментария коммита обязательно нужно «залить» обновления на удаленный сервер с помощью `git push origin master --force`

3.15. Просмотр информации по коммитам

Если требуется вывести информацию по коммиту (например, требуется выяснить что было удалено/добавлено в этот коммит), то можно обратиться к коммиту через его хеш-код

```
git show 06e6bbc
```

Информацию по последнему коммиту можно посмотреть следующим образом

```
git show master
```

3.16. Ссылки на предков

Для просмотра *предыдущего коммита* достаточно написать `HEAD^`, что означает «родитель HEAD»

```
git show HEAD^
```

Другое распространенное обозначение *предка* – символ `~`. Он также соответствует *ссылке на первого родителя*, поэтому записи `HEAD^` и `HEAD~` эквивалентны. А вот если указать номер после символа `~`, то проявятся различия между `~` и `^`.

Например, запись `HEAD~2` означает «первый предок первого предка», при этом происходит переход от заданного предка вглубь указанное число раз, т.е. `HEAD~3` укажет на четвертый⁸ от конца ветки коммит.

После символа `^` можно указать число: например, запись `d921970~2` означает «второй предок коммита d921970». Этот синтаксис применяется только в случае *коммитов слияния*, у которых существует несколько предков. *Первый родитель* – это ветка, на которой вы находились в момент слияния, а *второй родитель* – коммит на ветке, которая подверглась слиянию

```
git show d921970~2
```

Указанные обозначения можно комбинировать. К примеру, второго родителя четвертого от конца ветки коммита (при условии, что это коммит слияния) можно получить, написав `HEAD~3^2`.

⁸Так как отсчет ведется, начиная со второго коммита от конца ветки

3.17. Диапазоны коммитов

Вывести все коммиты, достижимые по ссылке **refA** или **refB**, но не достижимые по ссылке **refC**

```
git log refA refB ^refC
git log refA refB --not refC
```

Вывести только те коммиты, которые есть либо в ветке **master**, либо в ветке **experiment**, но не в обеих ветках одновременно

```
git log master...experiment
```

С этой командой часто используют параметр **--left-right**, позволяющий посмотреть, с какой стороны диапазона находится каждый коммит

```
git log --left-right master...experiment
```

3.18. Скрытие и очистка

Часто во время работы над проектом, все еще находится в беспорядочном состоянии, возникает необходимость перейти в другую ветку и поработать над другим аспектом. Проблема в том, что фиксировать работу, сделанную наполовину, чтобы позже к ней вернуться вы не хотите. В такой ситуации на помощь приходит команда **git stash**.

Если, к примеру, вы отредактируете два файла и только один из них проиндексируете без фиксации результатов своей работы, то с помощью команды

```
git stash save
```

можно будет перейти на другую ветку, скрыв наработки в буфере.

Теперь можно легко менять ветки и работать над другими фрагментами проекта – все изменения хранятся в стеке. Увидеть содержимое позволяет команда

```
git stash list
```

Вернуть скрытые в буфер изменения в рабочее состояние можно командой

```
git stash apply
```

Если требуется вернуться к работе над версией, сохраненной в буфере ранее, следует указать ее номер

```
git stash apply stash@{2}
```

Замечание

Вообще говоря, нет необходимости возвращать содержимое буфера в чистый рабочий каталог и в ту же ветку, из которой они были сохранены. Можно скрыть изменения одной ветки, перейти в другую и попытаться вставить измененное состояние туда

После извлечения информации из буфера файлы, которые до помещения в буфер были проиндексированы, автоматически в это состояние не вернуться. Чтобы сразу вернуть данные из буфера в исходное состояние, нужно написать

```
git stash apply --index
```

При этом команда **apply** только возвращает данные в ветку, но из стека они никуда не деваются. Убрать их из стека позволяет команда **git stash drop** с именем удаляемого файла

```
git stash drop stash@{0}
```

Впрочем, существует также команда

```
git stash pop
```

которая возвращает сохраненную в буфере информацию в ветку и немедленно удаляет ее из буфера.

3.19. Принудительно перезаписать локальные файлы

Если требуется локальные файлы перезаписать файлами с удаленного сервера, то алгоритм следующий

```
git fetch --all  
git reset --hard origin/master
```

Команда `git fetch --all` скачивает отсутствующие файлы с удаленного репозитория без попытки слить или переместить данные, а `git reset --hard origin/master` «сбрасывает» ветку `master`. Опция `--hard` изменяет все файлы в рабочем дереве таким образом, чтобы они совпадали с файлами из `master/origin`.

Список литературы

- 1.