

Работа с Git

Оглавление

Почему Git	3
Общие сведения о Git	4
Установка Git под Windows	5
Работа через консоль	8
Правила ведения чистых коммитов	11
Основные используемые функции	13
Работа из GUI SourceTree	20
Работа из графического интерфейса PhpStorm	42
Ревью и выкладка задачи	48
Устранение конфликтов при слиянии	51
Gitignore	55
Сценарий работы с системой контроля версий (на примере GIT)	59

Почему Git



Потому что не появляются задержки при работе с системой контроля версий. Git хранит всё локально, включая историю, ветки, коммиты и позволяет работать без обращения к сети. Git позволяет легко работать с ветками, без видоизменений раскладки репозитория, а древовидный просмотр изменений позволяет видеть, что из какой ветки пришло.

Более подробно можно прочитать здесь: <http://habrahabr.ru/blogs/Git/104198/>

Общие сведения о Git

Git относится к классу DVCS (Distributed Version Control System). При этом рабочая копия содержит все коммиты, историю, ветки, всё необходимое для ведения разработки без обращения к какому-либо серверу.

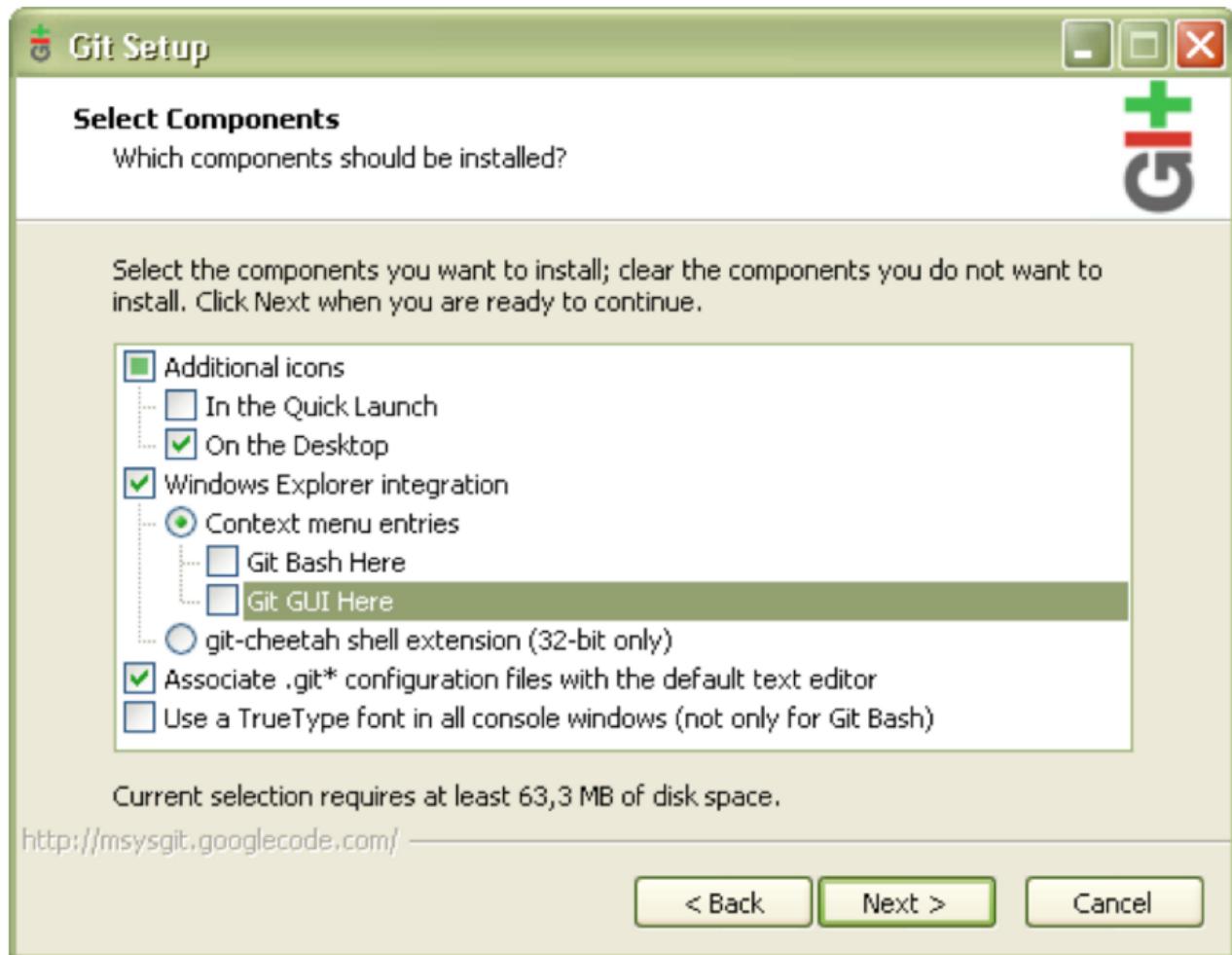
Для синхронизации изменений между разными копиями репозитория в нужный момент делается **pull**, чтобы скопировать изменения удалённого репозитория к себе, либо **push**, чтобы скопировать локальные изменения в удалённый репозиторий.

В случае с Git каждый **коммит** имеет уникальный ID в виде хеша, содержащий в себе все файлы, относящиеся к нему. Каждый коммит имеет один коммит-родитель, и, возможно, коммит-источник слияния. Таким образом, коммиты представляют собой дерево наборов файлов.

«**Веткой**» является указатель на какой-либо коммит. Чтобы слить две ветки, одна из которых начинается с конца другой, можно просто передвинуть указатель второй ветки на новый коммит (это называется Fast-Forward).

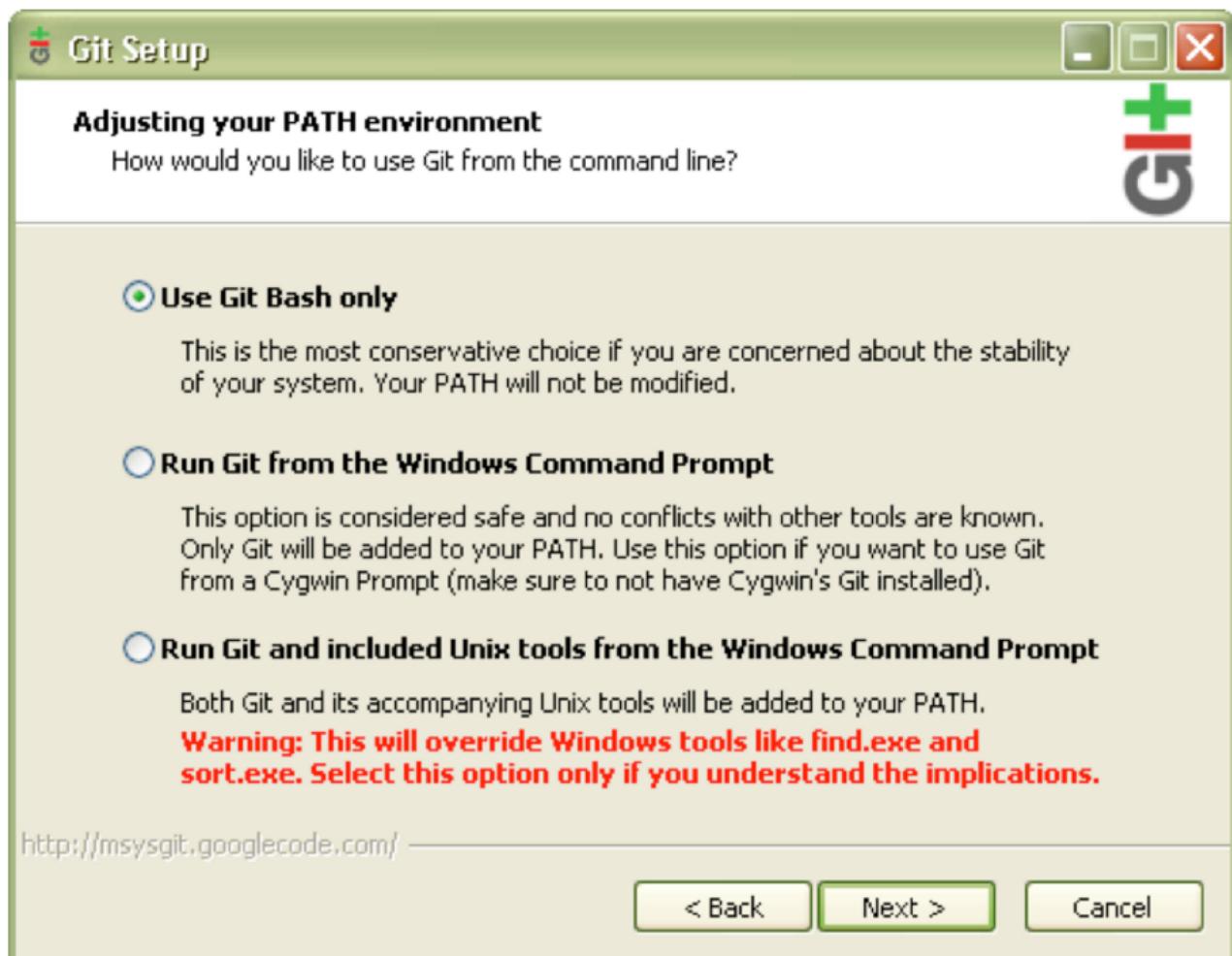
Установка Git под Windows

Устанавливается **msysGit** из проекта <http://msysgit.github.io>. Нужно скачать и запустить на выполнение установочный файл.



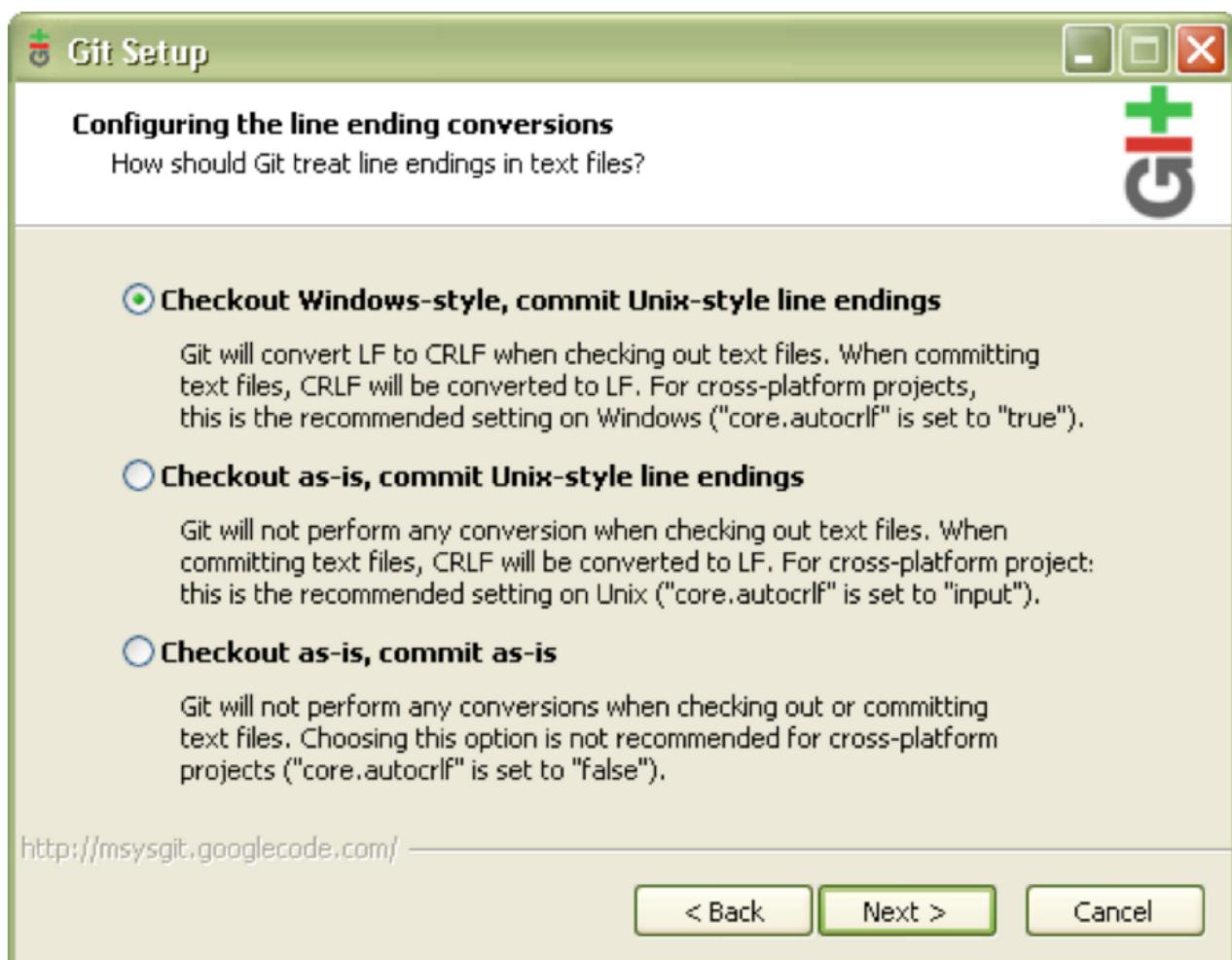
Выбор компонентов

Выбрать опции при установке «**Use Git Bash only**» или «**Run Git from the Windows Command Prompt**»



Установка переменной окружения PATH

Выбрать «Use Windows style line endings».



Настройка способа переноса строк

После завершения копирования компонентов нужно нажать кнопку «Finish» и на этом установка будет окончена.

Работа через консоль

1. Открываем консоль (Правой кнопкой в проводнике и выбрать git bash — для Windows, любой другой терминал — для других ОС).
2. Для работы с Git необходимо настроить ФИО и E-mail автора:

```
git config --global user.name "Ivan Petrov"  
git config --global user.email "work@mail"
```

3. Создаем приватный ключ, введя в консоли следующую команду:

```
ssh-keygen
```

Нажимаем Enter один раз для подтверждения адреса для хранения пароля. По умолчанию — нам подходит.

Нажимаем Enter два раза на вопрос о пароле. Пароль нам не требуется, ведь мы как раз генерируем ключ для того, чтобы избавиться от пароля.

После выполнения команды будет выведена надпись:

```
Your public key has been saved in ....ssh/id_dsa.pub.
```

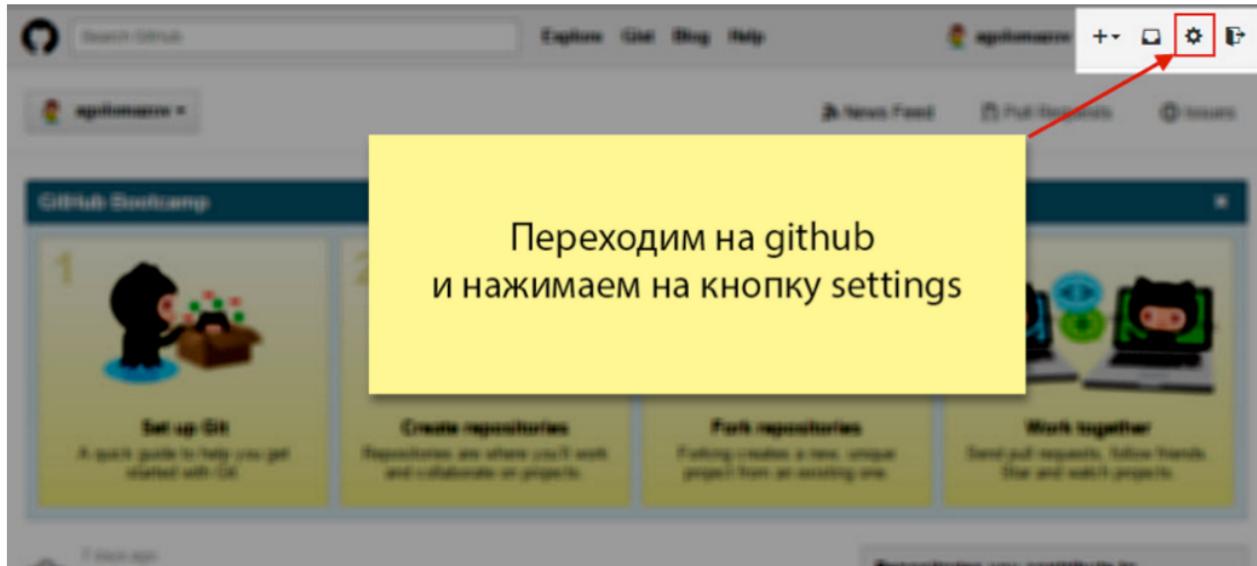
4. Предыдущая команда создала SSH ключ (id_rsa.pub) и поместила его в вашу домашнюю папку (C:/Users/username/ — для Windows), в подпапку «.ssh». Наша задача — прочитать содержимое этого файла. Вы можете сделать это с помощью Проводника и Блокнота, но самый простой способ — с помощью консоли:

```
cat ~/.ssh/id_rsa.pub
```

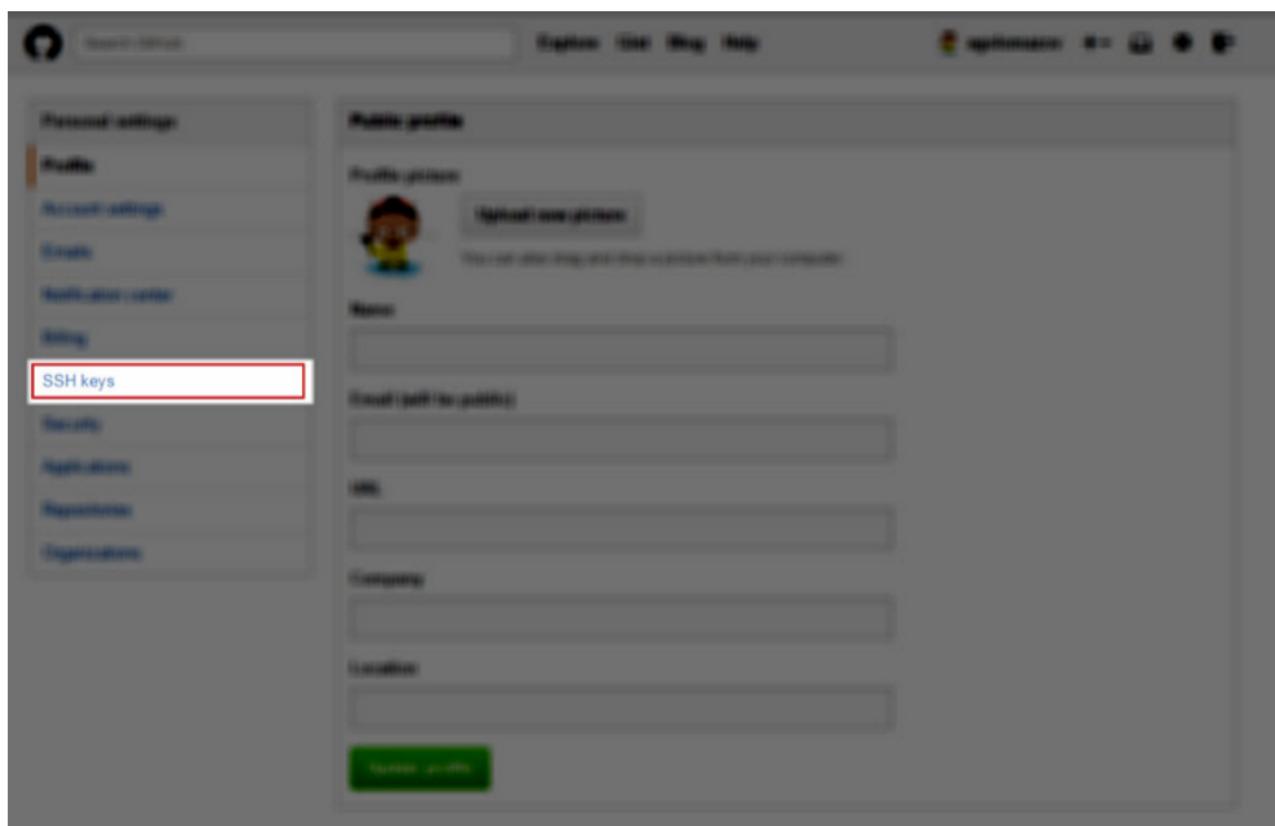
Эта команда читает файл id_rsa.pub в домашней папке (~) в подпапке .ssh и выводит содержимое на экран. Содержимое этого файла необходимо скопировать.

```
tmp@arku-laptop /home/arku $ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQC/UQh+ScwtuEyl0GEpv2Qhh0q8HujHmk2e1mDZcDEy
E/dmbJU9PUsoyWjfa0z+Iuf0puUnSVs6H3apxeI4rwRrHr10yUShBDyzgyKlsb0G9Tt/HTBwkCL5Kdgy+
Qmwjp/Vt4KDoFpXBp1NAg8EGj9wyYIDfs0g0zsL+kwgAF06+i7WNg00cbqJqtw2gxhCcf3C5picqACPq
uM0TCL3sToCd0uTlVQpDsS181uo1C/K/LvZG5KgBuyWduf8Jp3WT9CLkHBfBzFF33XoU8QqUUUVUR5D+
eRvQ0WbEdYCKMG3cKuFqAXXnf0qF6VkKaTgEXTToUwRwAsDn9ai5f/RVjx/5 tmp@arku-laptop
```

5. Переходим на [github](#) и нажимаем на кнопку «**Settings**»

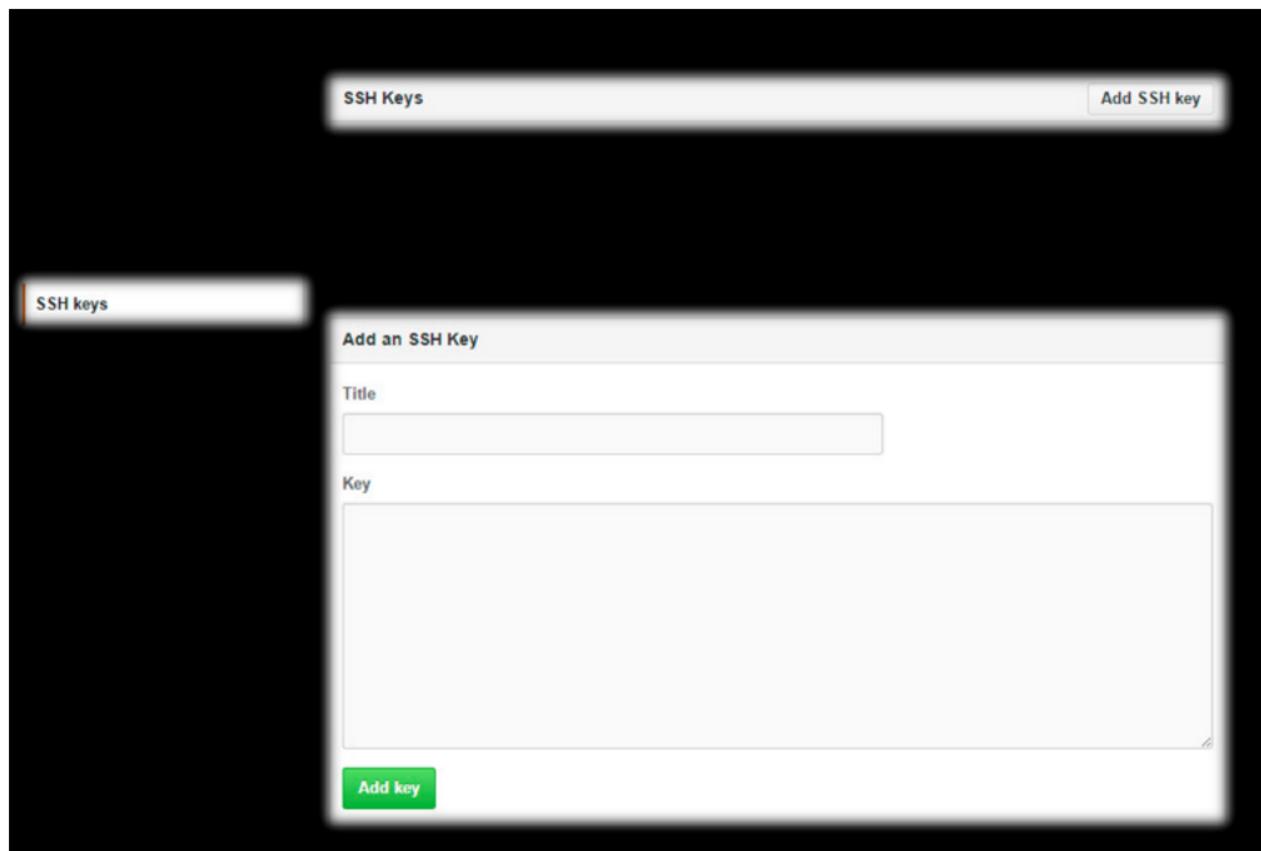


Нажимаем на ссылку «**SSH keys**»

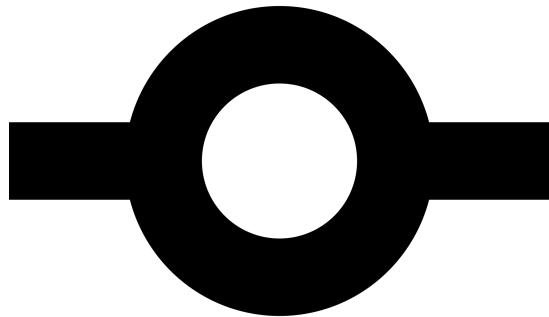


6. Заполняем форму:

- В поле «**title**» пишем любое имя для ключа.
- В поле «**key**» помещаем текст открытого ключа, который был сгенерирован командой ssh-keygen.
- Нажимаем на кнопку «**Add key**».



Правила ведения чистых коммитов



Все коммиты, которые попадают в центральную ветку, должны следовать следующим правилам:

- 1. Автором должен быть прописан разработчик – Имя, фамилия, рабочий e-mail.**
- 2. Текст комментария должен содержать #номер задачи и краткое описание изменения, формат:**
«#NNNN[.branch_name_after_dot] What exactly done» Символы «[» и «]» – означают что опционально. Для зарубежных проектов описание обязано быть на английском языке, для российских проектов приемлемо комментировать на русском. По желанию в комментарии к коммиту можно писать название ветки, в таком случае между номером задачи и названием ветки ставиться точка, например: «#9388.contact-actions-append-ita-profiles то, что я сделал».
- 3. Коммит не должен содержать в себе файлы, не относящиеся к изменениям.** Если ваша IDE, OS, какой-то плагин к какому-либо софту, использующемуся при разработке создают технические файлы, то либо добавьте их в .gitignore, либо не добавляйте к коммиту, либо удаляйте перед коммитом.
- 4. Коммит не должен добавлять/убирать пустые строки, менять пробелы на табы и наоборот, менять число пробелов и т. п. нигде, кроме случаев, относящихся к сути коммита.** То есть при

рефакторинге это нормально, но если ваш редактор поменял во всём файле пробелы на табы или наоборот — меняйте настройки редактора или перед коммитом приводите всё к виду «как было».

5. Минимизация конфликтов. При добавлении кода следует стараться форматировать код так, чтобы модификация его приводила к минимуму конфликтов при слиянии.

Основные используемые функции

1. Получение всех коммитов за определённое время:

```
git log --since=YYYY-mm-dd (без включения последнего дня)  
git log --until=YYYY-mm-dd (с включением последнего дня)
```

2. Получение списка коммитов определённого автора:

```
git log --author="Имя_автора"
```

3. Получение N последних коммитов:

```
git log -n N
```

4. Поиск коммита по регулярному выражению:

```
git log --grep="Init"
```

5. Отмена только что введённых изменений в файл:

```
git checkout -- ИМЯ_ФАЙЛА
```

6. Просмотреть изменение в буфере Git:

```
git diff -staged
```

7. Просмотреть изменения и подсветить их:

```
git diff --color-words ИМЯ_ФАЙЛА
```

8. Удаление файлов из репозитория:

```
git rm ИМЯ_ФАЙЛА
```

9. Перемещение файлов:

```
git mv ИМЯ_ФАЙЛА
```

10. Добавление и одновременный коммит всех файлов:

```
git commit -am "Текст коммита"
```

11. Сброс изменений в файле:

```
git checkout ИМЯ_ФАЙЛА
```

12. Сброс изменений в буфере:

```
git reset HEAD ИМЯ_ФАЙЛА
```

13. Редактирование последнего коммита:

```
git commit --amend -m "Комментарий к коммиту"
```

14. Откат к предыдущему коммиту:

```
git reset --soft ХЭШ_КОММИТА  
git reset --mixed ХЭШ_КОММИТА  
git reset --hard ХЭШ_КОММИТА
```

15. Добавление файла глобального игнорирования:

```
git config --global core.excludefiles ПУТЬ_К ФАЙЛУ
```

16. Игнорирование файла, который лежит в репозитории:

Сначала добавляем файл в **.gitignore**. Затем:

```
git rm --cached ИМЯ_ФАЙЛА
```

17. Показать все отслеживаемые файлы в репозитории:

```
git ls-tree HEAD
```

18. Показать список отслеживаемых файлов в определённой папке определённой ветки:

```
git ls-tree master ИМЯ_ПАПКИ/
```

19. Список отслеживаемых файлов в определённом коммите:

```
git ls-tree HASH-COMMITA
```

20. Показать сокращённый вариант логов репозитория:

```
git log --oneline -N
```

21. Показать коммиты в диапазоне от 1-го SHA-1 до 2-го SHA-1:

```
git log 8713cbc..e5dfaf3 --oneline
```

22. Показ всех изменений, сделанных в файле на определённом коммите:

```
git log -p HASH-COMMIT index.html
```

23. Подробный показ изменений в ходе коммита в определённом файле:

```
git show HASH_COMMITA index.html
```

24. Создание новой ветки:

```
git branch ИМЯ_ВЕТКИ
```

25. Переключение между ветками:

```
git checkout ИМЯ_ВЕТКИ
```

26. Создание и одновременное переключение между ветками:

```
git checkout -b ИМЯ_ВЕТКИ
```

27. Сравнение веток:

```
git diff ИМЯ_ВЕТКИ1..ИМЯ_ВЕТКИ2
```

28. Переименование ветки:

```
git branch -m СТАРОЕ_ИМЯ_ВЕТКИ НОВОЕ_ИМЯ_ВЕТКИ
```

29. Удаление ветки:

```
git branch -d ИМЯ_ВЕТКИ
```

30. Слияние ветки:

```
git merge ИМЯ_ВЕТКИ
```

31. Обрывание процесса слияния:

```
git merge --abort
```

32. Добавление удалённого репозитория:

```
git remote add origin URL_репозитория
```

33. Просмотр списка удалённых репозиториев:

```
git remote -v
```

34. Удаление удалённого репозитория:

```
git remote rm алиас_репозитория
```

35. Отправка локального репозитория на github.com:

```
git push -u АЛИАС_УДАЛЁННОГО_РЕПОЗИТОРИЯ ЛОКАЛЬНАЯ_ВЕТКА
```

36. Просмотр списка удалённых веток:

```
git branch -r  
git branch -a # просмотр и локальных и удалённых веток
```

37. Клонирование репозитория с удалённого сервера:

```
git clone URL_ИЗ_GITHUB [имя_папки, в  
которую_надо_положить_репозиторий]
```

38. Отправка ветки на удалённый репозиторий:

```
git push -u АЛИАС_УДАЛЁННОГО_РЕПОЗИТОРИЯ ЛОКАЛЬНАЯ_ВЕТКА
```

39. Синхронизация удалённого репозитория и локальной версии:

```
git fetch АЛИАС_УДАЛЁННОГО_РЕПОЗИТОРИЯ
```

40. Объединение удалённой ветки:

```
git merge origin/master
```

41. fetch + merge:

```
git pull
```

42. Добавление ветки из удалённого репозитория в локальную версию:

```
git merge ИМЯ_ЛОКАЛЬНОЙ_ВЕТКИ  
ИМЯ_ВЕТКИ_С_УДАЛЁННОГО_РЕПОЗИТОРИЯ
```

43. Удаление ветки из удалённого репозитория:

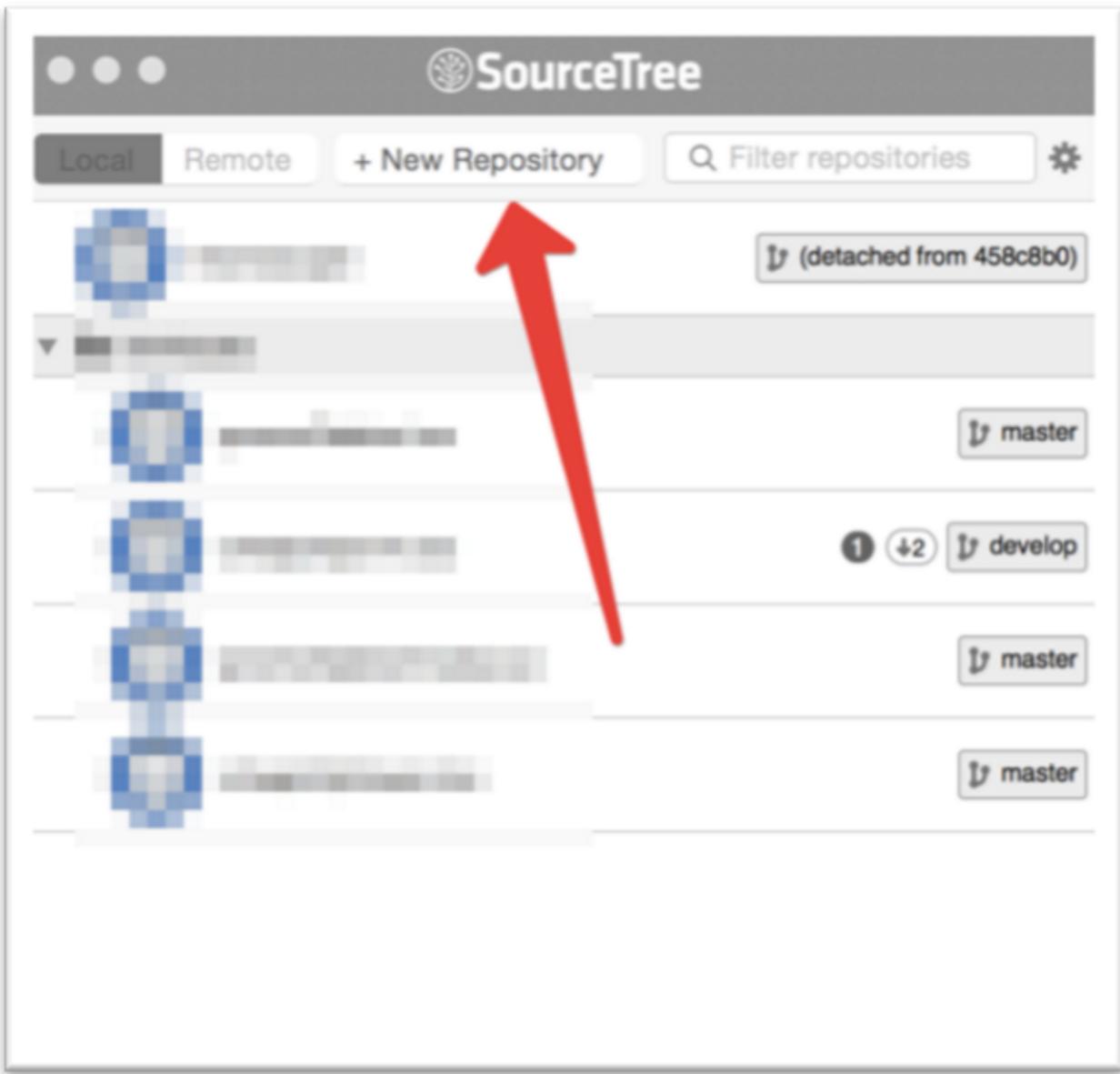
```
git push origin --delete ИМЯ_ВЕТКИ
```

44. Добавление алиасов к командам Git:

```
git config --global alias.СОКРАЩЕНИЕ_СКОМАНДЫ "КОМАНДА"
```

45. Посмотреть, кем в последний раз правилась каждая строка файла:

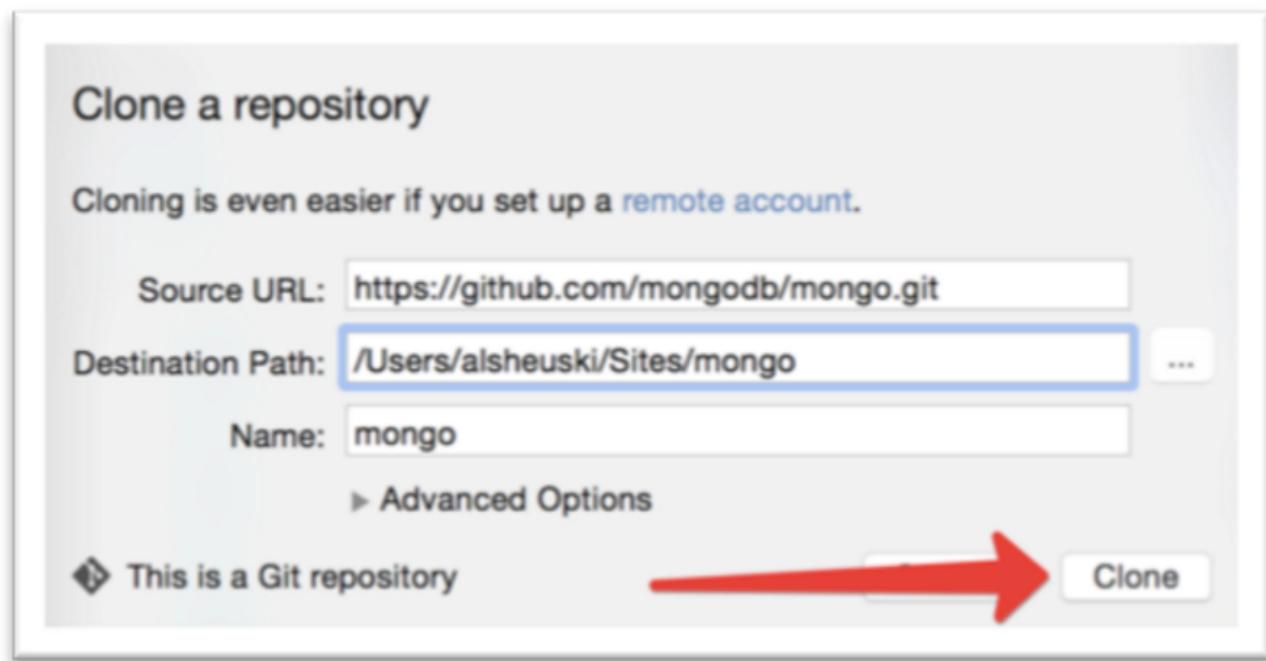
```
git blame ИМЯ_ФАЙЛА
```



Работа из GUI SourceTree



SourceTree – это GUI-клиент для работы с git- и mercurial-репозиториями от компании Atlassian. Данный клиент является бесплатным и доступен для установки на операционные системы Windows и Os X. Домашняя страница в сети находится по адресу <http://www.sourcetreeapp.com/>. Там можно найти различную промо-информацию, скриншоты программы и ссылки на получение установочных файлов.



В этой главе будет представлено описание работы с репозиторием Git и скриншоты программы в операционной системе OS X. Принципиальных отличий от версии, доступной для Windows, нет и всё делается аналогично.

Создание/клонирование репозитория (clone/init)

Для начала работы с проектом необходимо его клонировать/создать. Для этого запускаем SourceTree и нажимаем кнопку «**New Repository**» («Новый репозиторий») и выбираем один из пунктов.

В данный момент нас интересуют пункты «**Clone from URL**» («Клонировать из URL») и «**Create local repository**» («Создать локальное хранилище»), так как это основные варианты, с которыми придется работать.

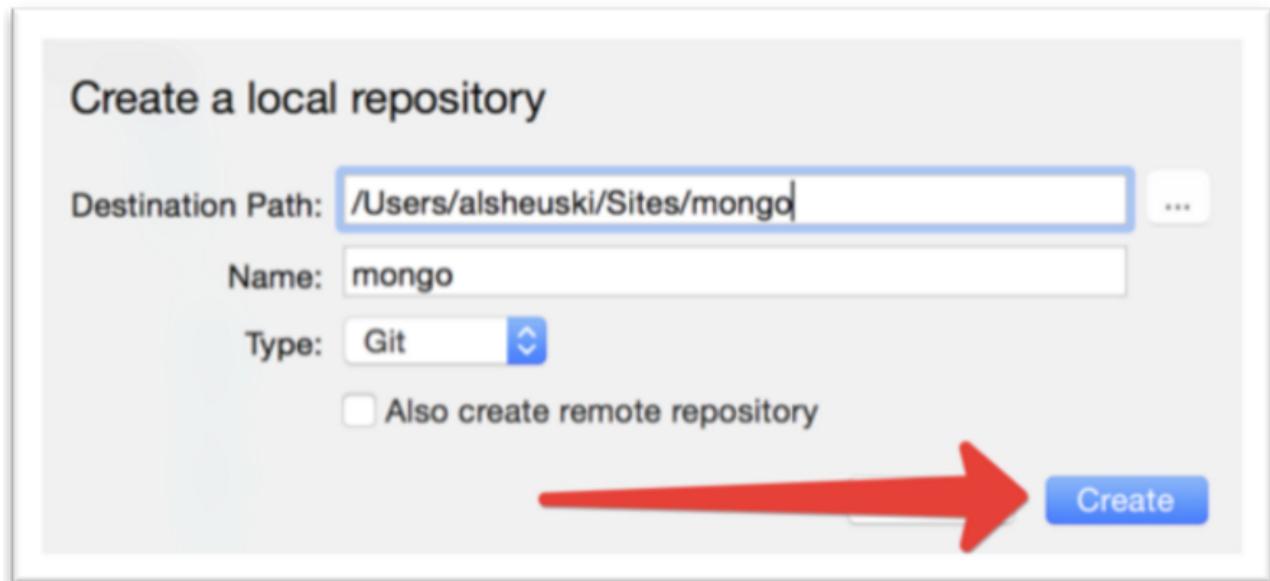
Менеджер проектов SourceTree

При выборе «Clone from URL» в новом окне нужно ввести в поле «Source URL» сетевой адрес репозитория, указать локальную папку, в которую будет клонирован проект.

Название ставится автоматически по названию папки назначения, но его можно изменить, либо указать необходимое. Далее нажать кнопку «Clone».

Настройки клонирования проекта

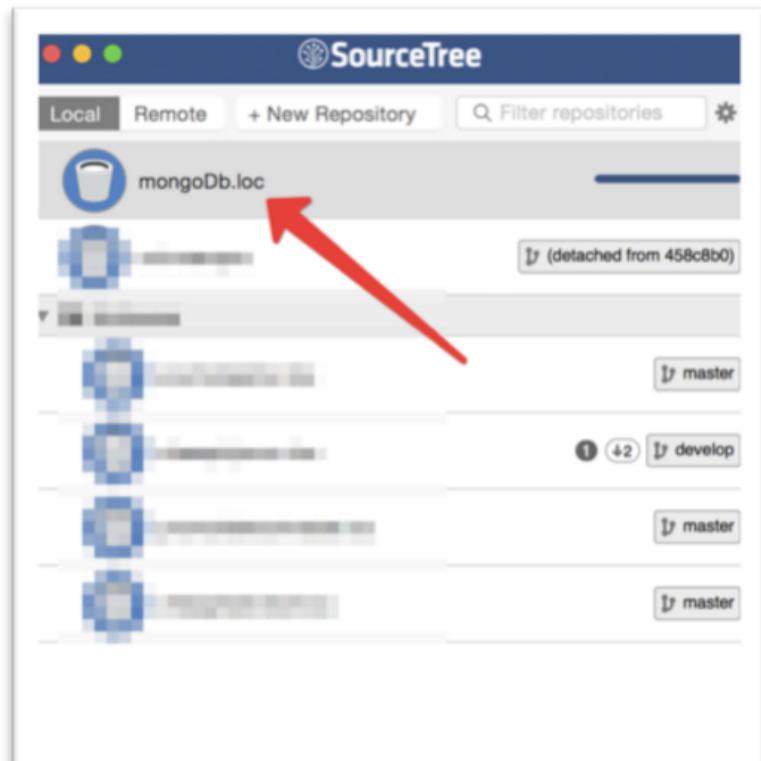
Для создания (инита) нового репозитория локально, нужно указать папку назначения, название проекта и тип хранилища (Git либо Mercurial). Далее нажать кнопку «Create».



Init нового проекта

Таким образом, произошел **clone/init** проекта, при этом было создано локальное хранилище проекта, находящее в папке **.git** и новый проект добавился в списке проектов SourceTree.

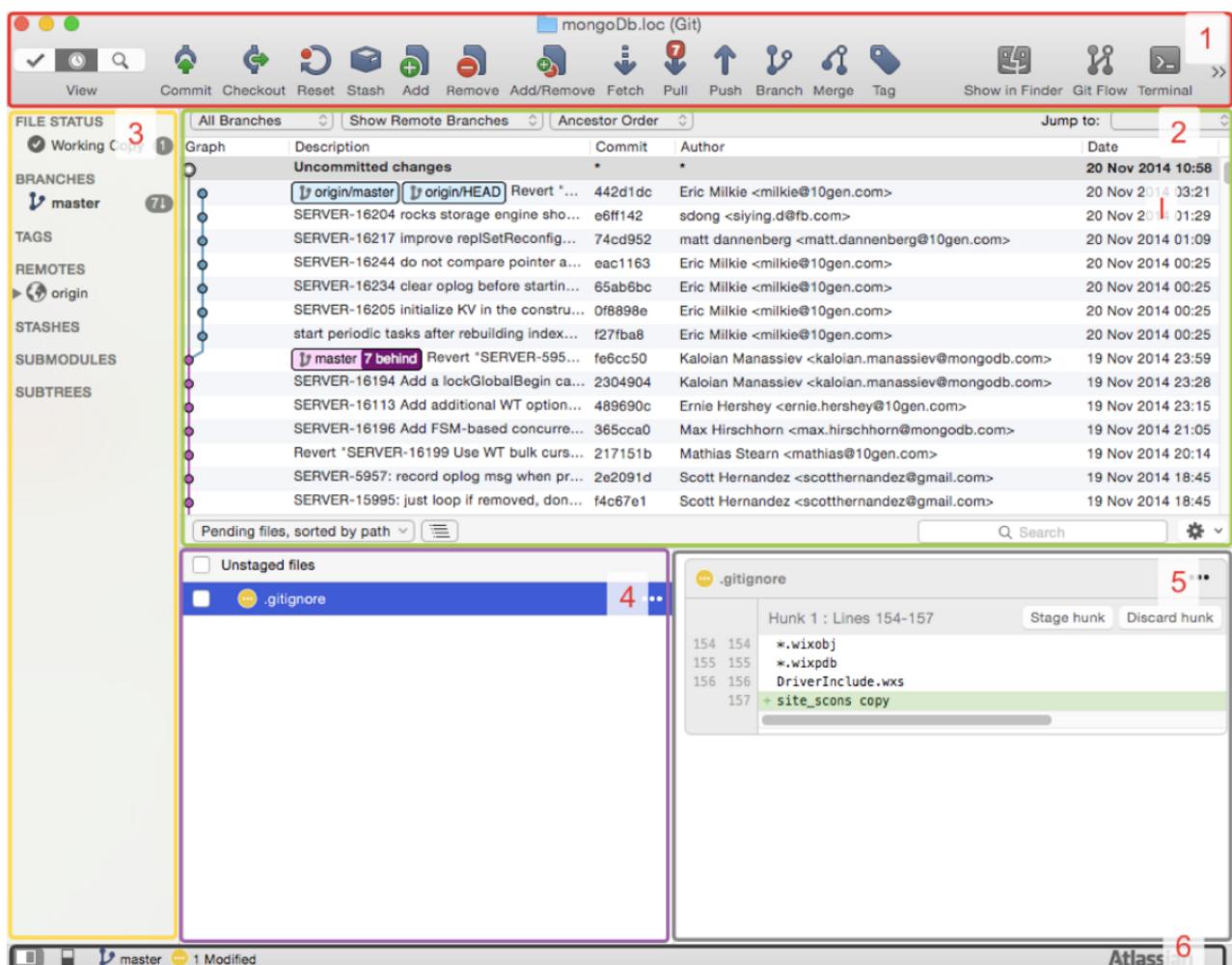
Далее при работе над проектом, Git отслеживает все изменения, происходящие с файлами данного проекта.



Менеджер проектов

Основное окно SourceTree

Основное рабочее окно SourceTree показано ниже на скриншоте. Оно содержит несколько областей с различной информацией и элементами управления.



Основное рабочее окно SourceTree

Назначение блоков:

- Блок с кнопками для выполнения базовых операций Git,** таких, как push/pull/commit/checkout и т. д.
- История ветвления и коммитов,** содержащая основную информацию о коммитах в виде списка и наглядно представляющая историю коммитов и их ветвление в виде дерева.

3. **Блок, содержащий списки веток, тэгов**, как локальных, так и удалённых, сохранённых (спрятанных) изменений в стэше (stash) и т. д.
4. **Список изменений для выбранного коммита**, либо незафиксированных изменений.
5. **Детальный просмотр изменений** в конкретном выбранном в блоке 4 файле.
6. **Строка состояния** SourceTree с дополнительной информацией.

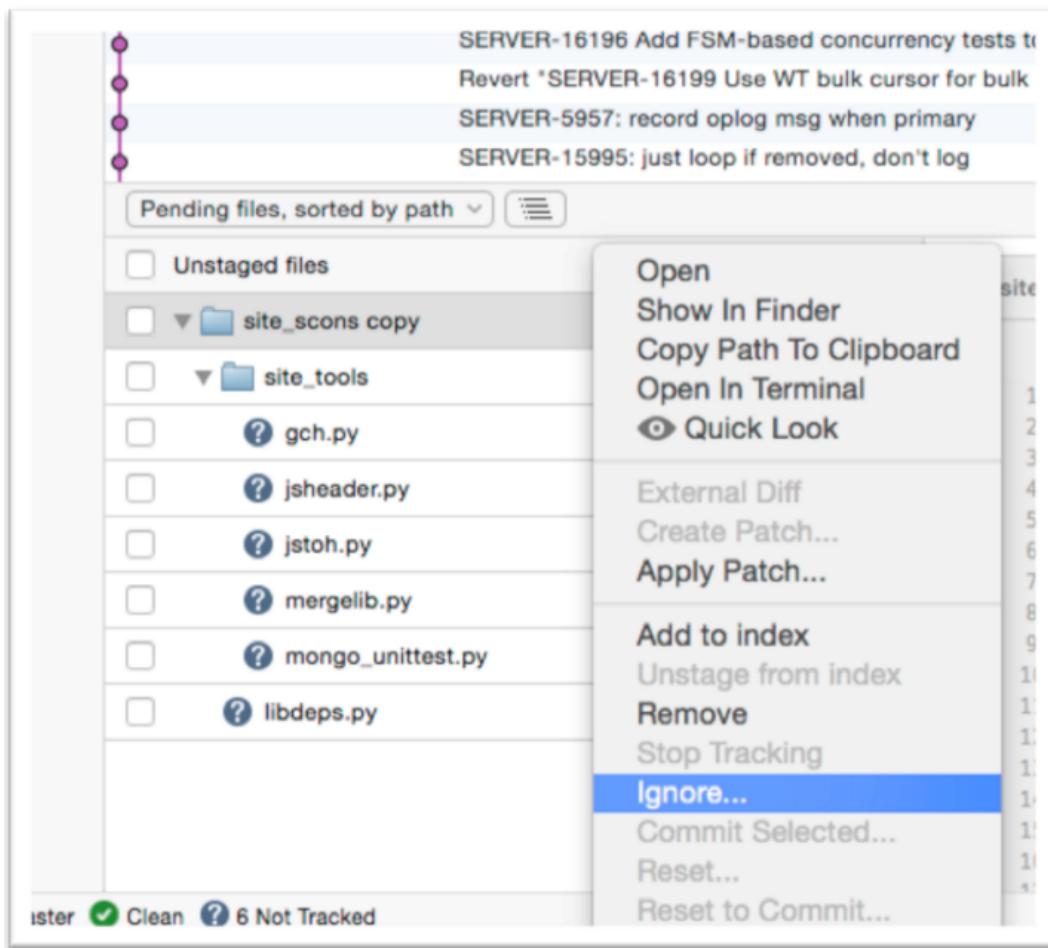
.gitignore



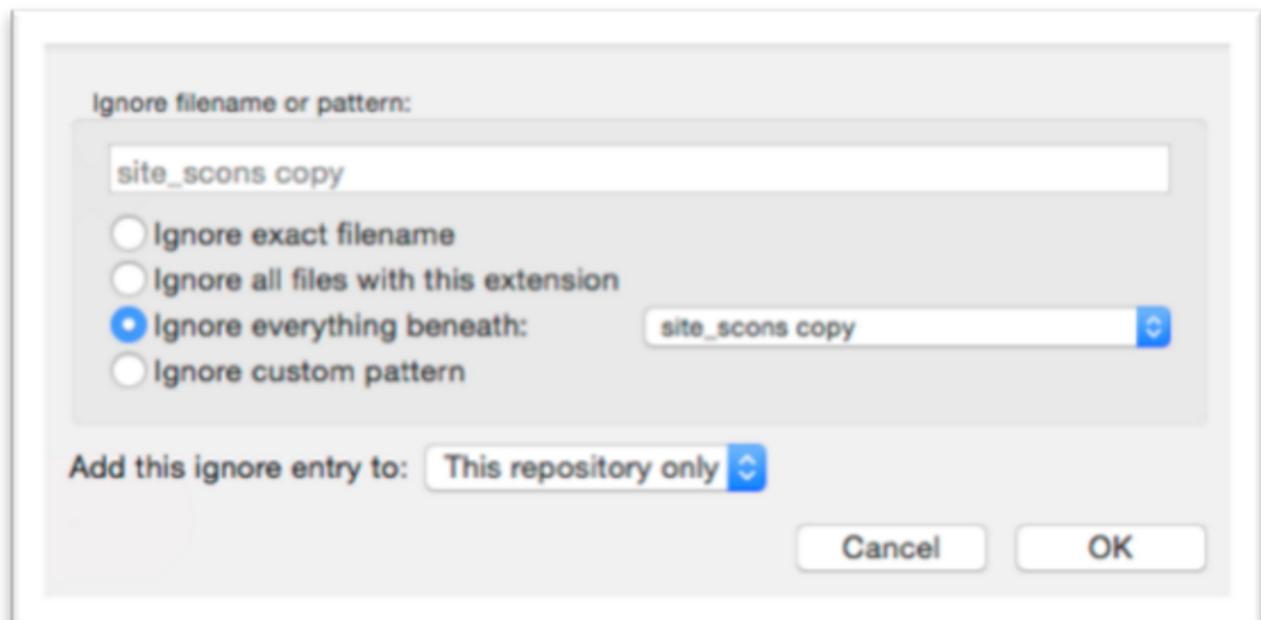
При работе над проектом возникают ситуации, когда не все файлы необходимо отслеживать и добавлять в удаленный репозиторий. Отличным приметом являются файлы локальных конфигов, либо пакеты, установленные через npm/bower.

Пакетные менеджеры устанавливают пакеты в папку, например, /node_modules, в корне проекта и записывают необходимые данные об установленных пакетах и их зависимостях в специальный файл. Для npm это packages.json и для последующей установки / разворачивания проекта в другом месте используют информацию из данного файла, чтобы установить все необходимое. Таким образом, **нет никакого смысла добавлять в репозиторий что-то из папки / node_modules, достаточно отслеживать только файл packages.json.**

Для исключения из поля видимости Git какие-либо файлы или папки служит файл **.gitignore**. Он находится в корне проекта и представляет собой простой текстовый файл с перечислением исключений. В SourceTree добавление исключений в .gitignore происходит через контекстное меню, вызываемое кликом правой кнопкой мыши по



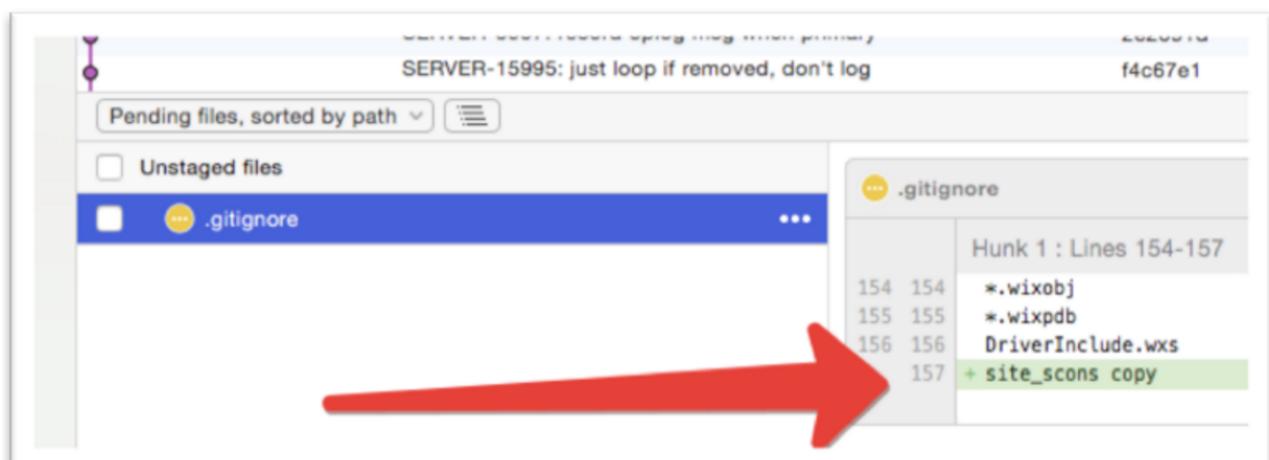
исключаемым файлам или папкам. В данном меню необходимо выбрать «**Ignore...**».



Добавление файлов в исключения

Далее выбор опций:

- 1. Ignore exact filename** – игнорировать файл с указанным именем.
- 2. Ignore all files with this extension** – игнорировать все файлы с указанным расширением.
- 3. Ignore everything beneath** – игнорировать всё в указанной папке.
- 4. Ignore custom pattern** – игнорировать по совпадению с шаблоном.



Опции добавления исключения в .gitignore

В данном примере после нажатия кнопки «OK» в .gitignore добавится новая строка «site_scons copy», а сам файл .gitignore будет изменён и данные изменения нужно закоммитить.

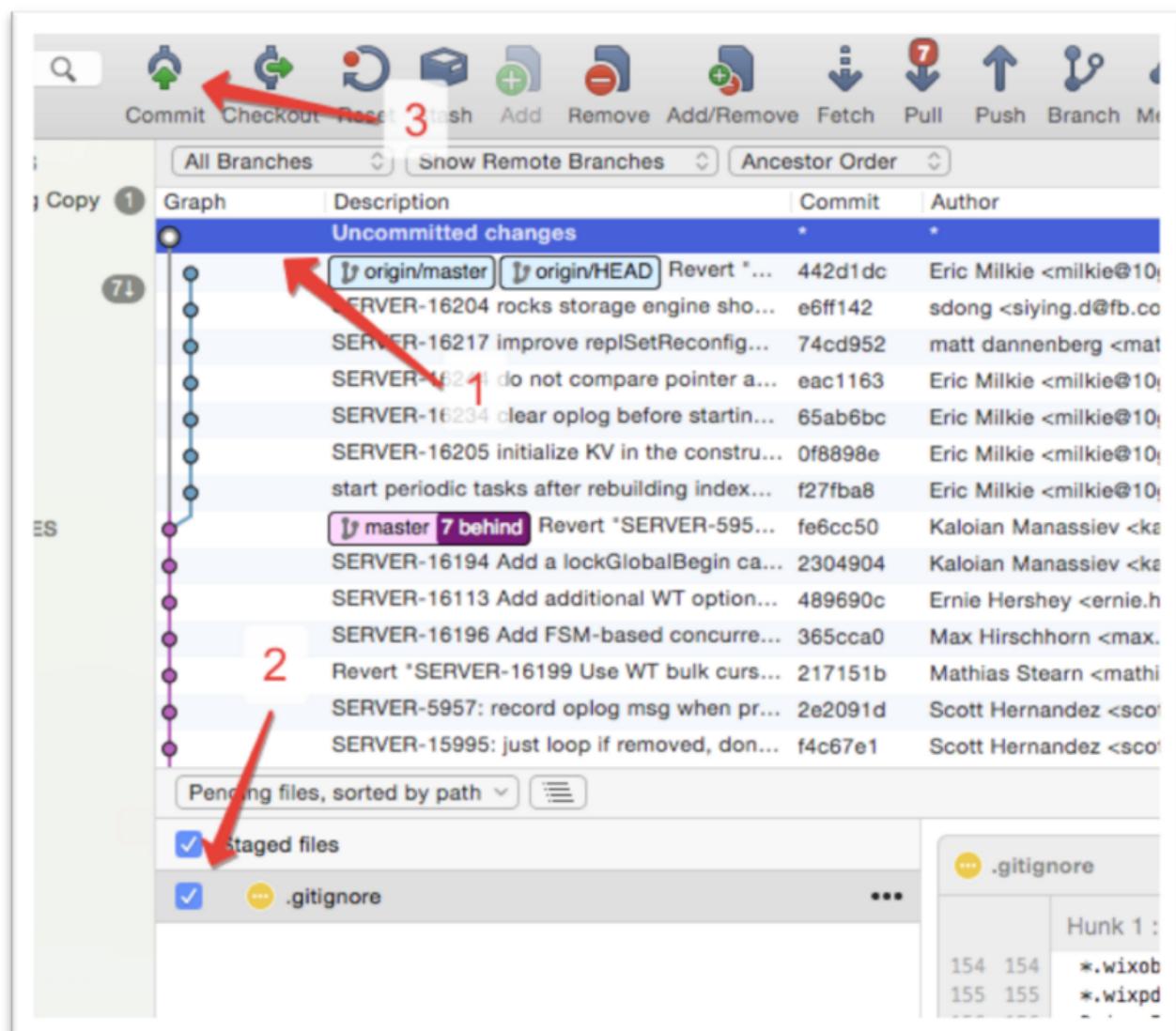
Изменение файла .gitignore

Коммит и пуш (commit/push)

Для фиксации изменений (commit) нужно убедиться в том, что (1) в блоке истории коммитов выбрана строка «**Uncommitted changes**».

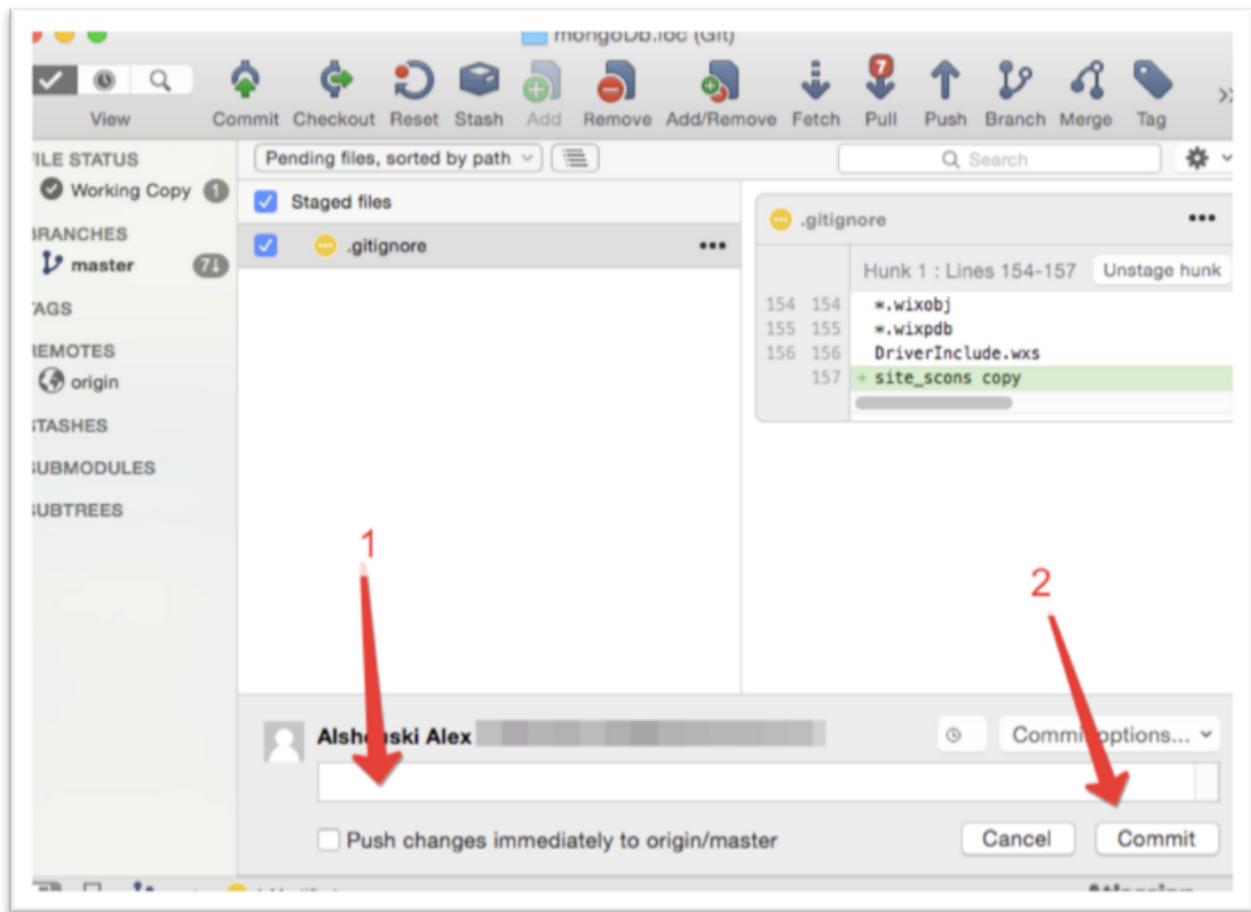
Далее (2) в блоке изменённых файлов галочками выбрать необходимые для добавления в коммит файлы (выбраны не обязательно должны быть все файлы, только те, изменения в которых нужно зафиксировать на данный момент).

После этого нажать (3) на кнопку «**Commit**».



Выбор файлов для коммита

Далее откроется окно для ввода описания коммита. В нем ещё можно изменить выбор файлов для коммита. Нужно ввести текст коммита (1) и нажать кнопку «Commit» (2).



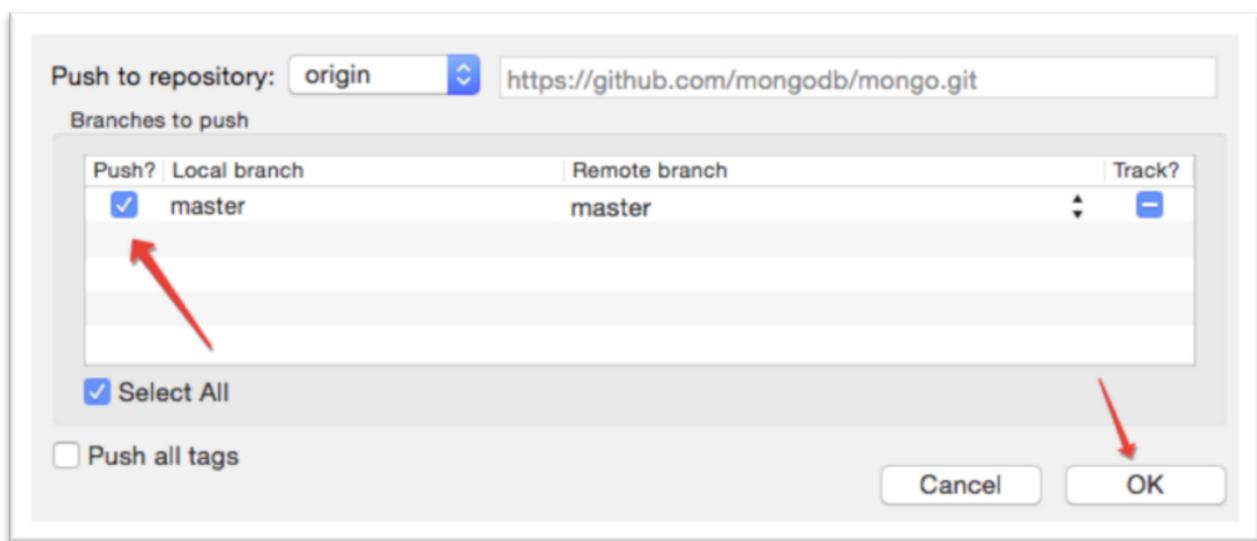
Создание коммита

Поздравляю! Коммит создан локально и для отправки изменений на сервер нужно нажать кнопку «Push» в блоке с основными операциями SourceTree.



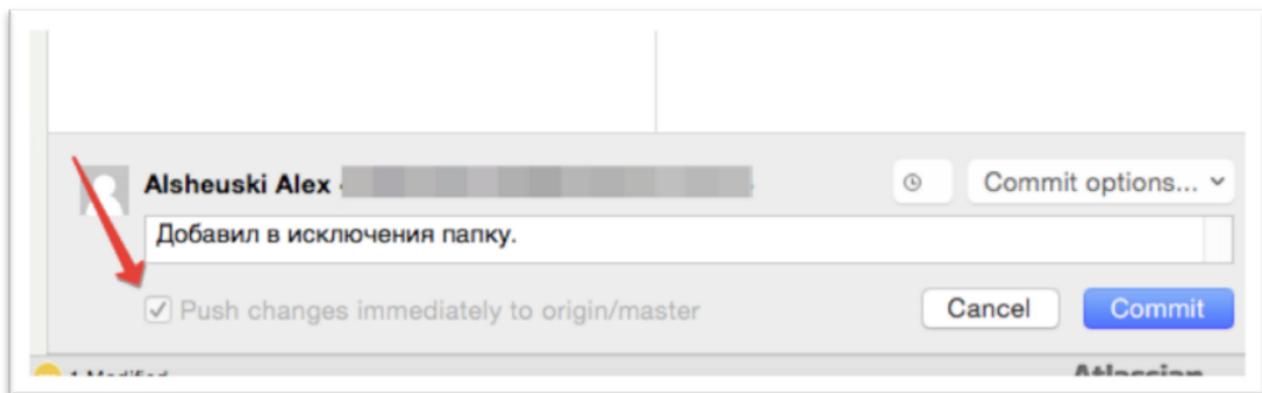
Кнопка «Push» с количеством коммитов для отправки

В появившемся окне выбрать ветку(и) для отправки изменений и нажать «OK».



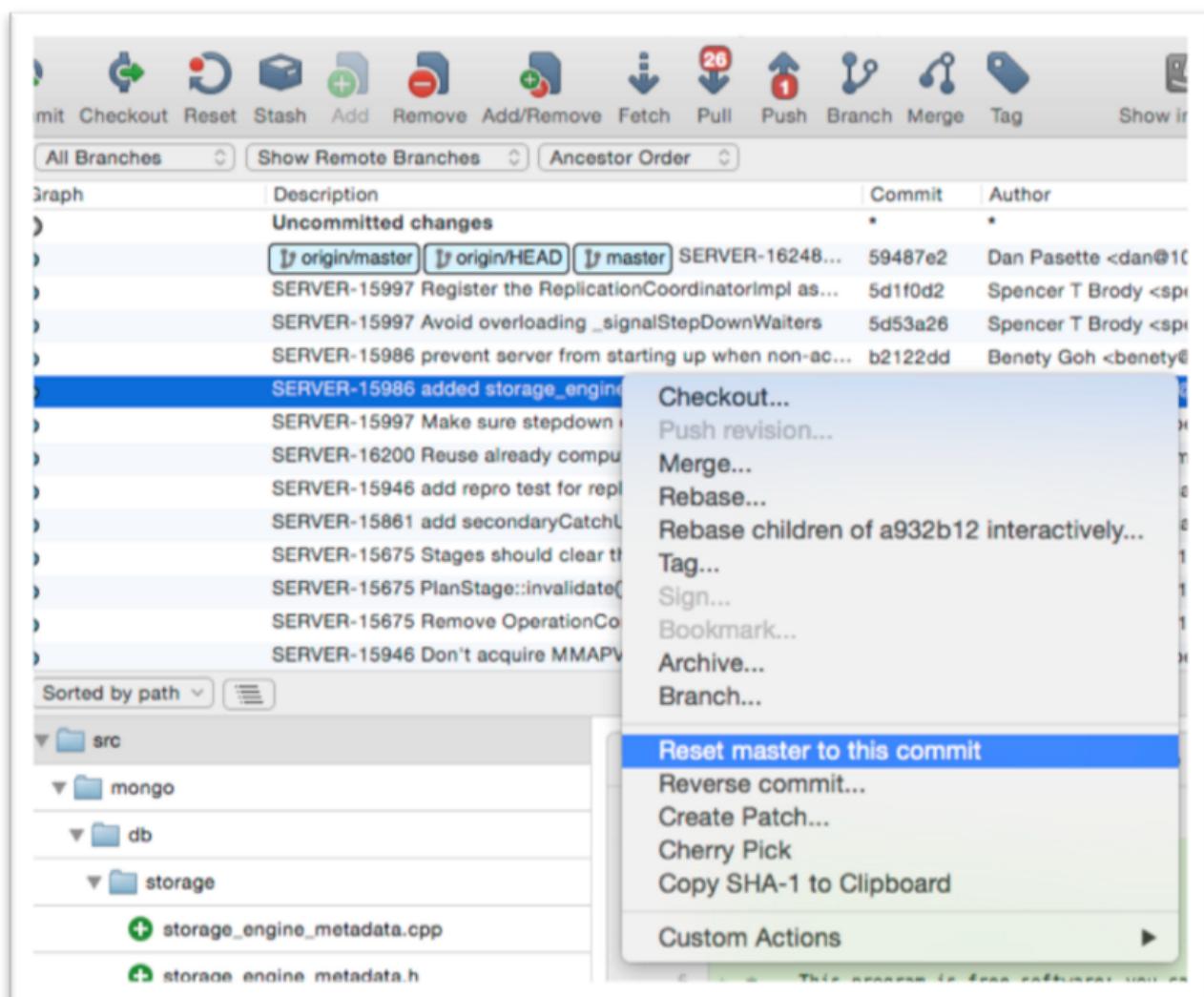
Push изменений

Также можно отправить (push) коммит в удалённый репозиторий сразу на предыдущем шаге. Для этого после ввода описания коммита нужно установить галочку **«Push changes immediately to название ветки»** и нажать **«Commit»**. При этом сразу же после создания коммита он будет автоматически отправлен в удаленный репозиторий.



Откат изменений (reset)

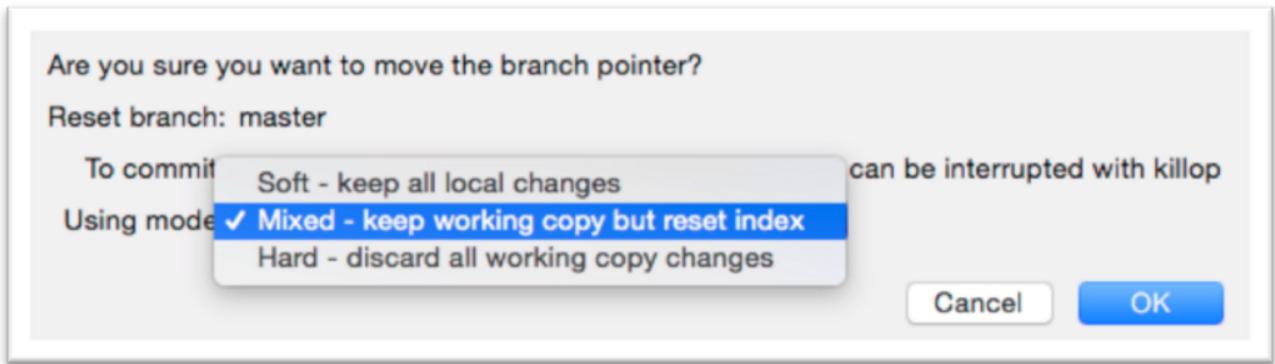
После сохранения изменений может появиться необходимость откатиться на состояние файлов в каком-либо коммите. Для этого необходимо найти и выделить нужный коммит в списке коммитов и кликнуть по нему правой кнопкой мыши и выбрать «Reset master to this commit».



Reset ветки на необходимый коммит

Далее выбрать режим отката:

1. **Soft** – с сохранением всех локальных изменений
2. **Mixed** – сохранение рабочей копии, но со сбросом индекса
3. **Hard** – сброс всех изменений



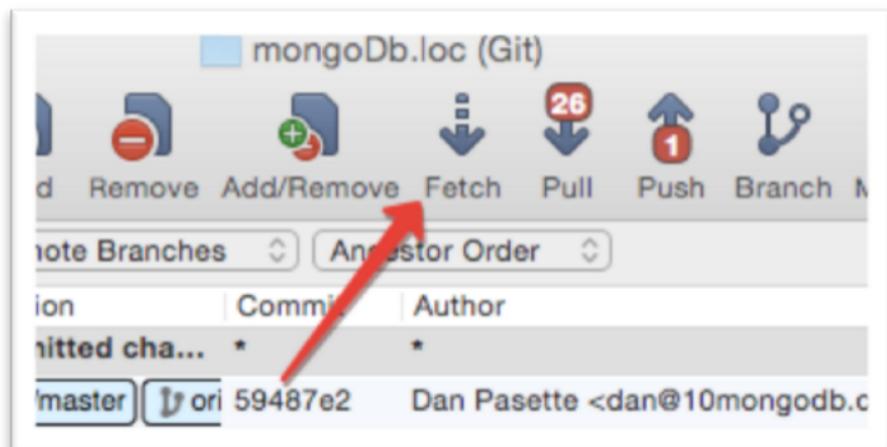
Выбор режима сброса

После нажатия кнопки «OK» индекс будет перемещен на выбранный коммит и файлы проекта изменятся в соответствии с выбранным режимом отката.

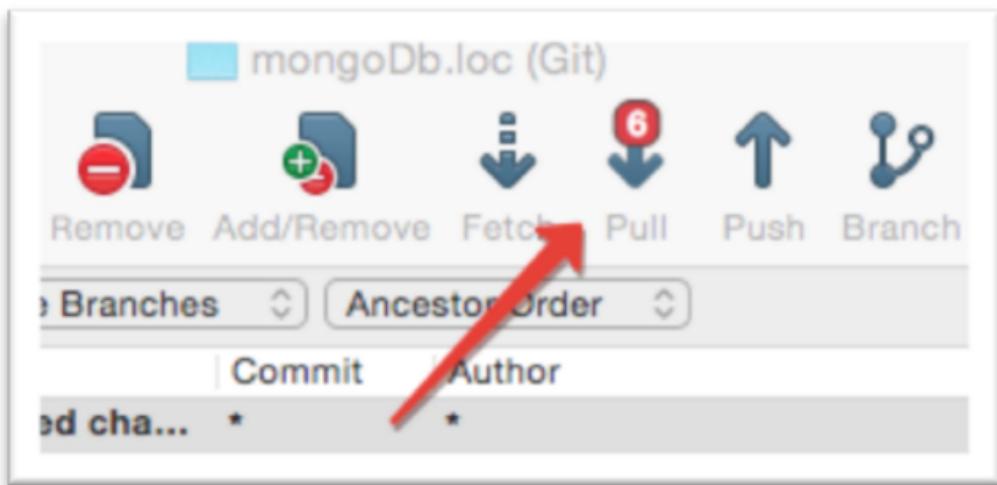
Получение обновлений (fetch/pull)

При командной работе над проектом, либо при работе в одиночку, но в разных местах, в репозитории будут появляться коммиты, которых нет локально. Для синхронизации текущей рабочей копии проекта и получения изменений необходимо сделать 2 вещи:

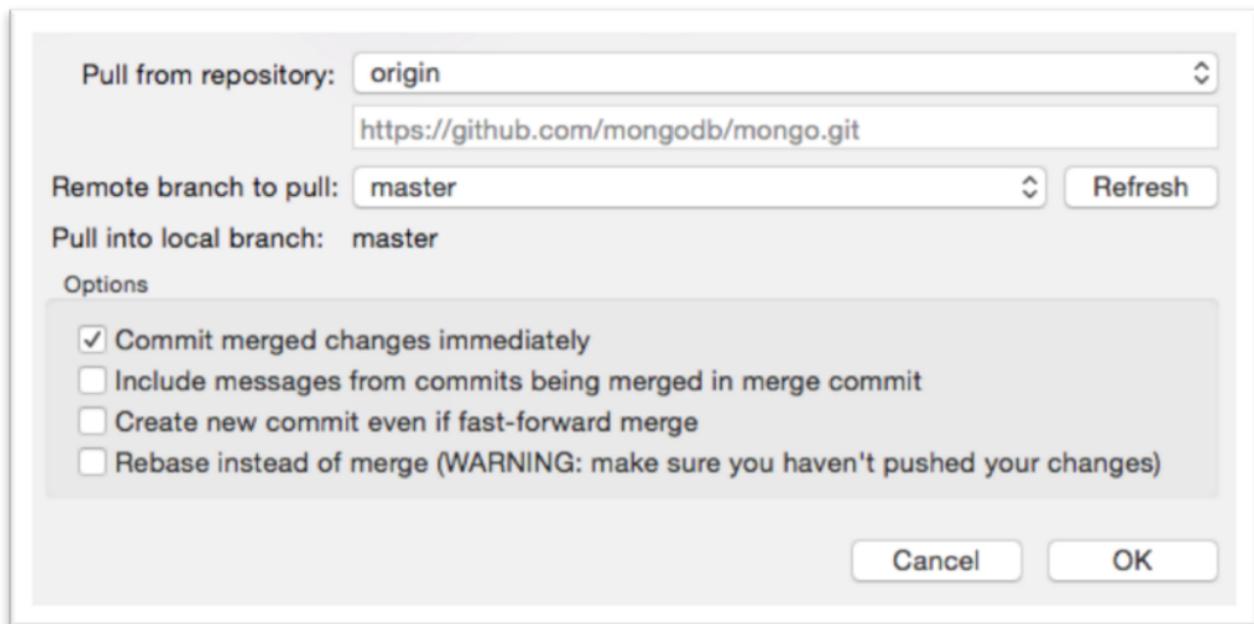
1. Получить изменения в удалённом репозитории. Для этого необходимо нажать кнопку «Fetch»



2. Применить изменения к рабочей копии, нажав кнопку «Pull».



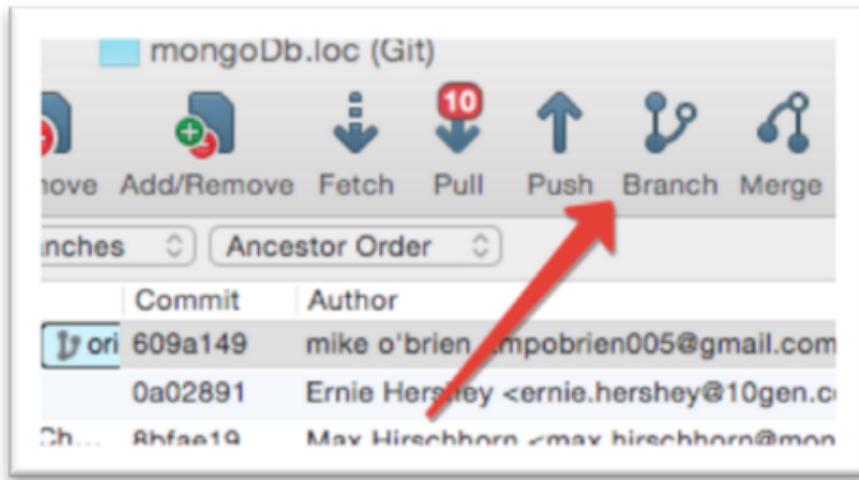
Убедитесь в том, что в следующем окошке выбрана верная текущая ветка и стоит галочка «Commit merged changes immediately» и нажмите «OK».



Опции «Pull»

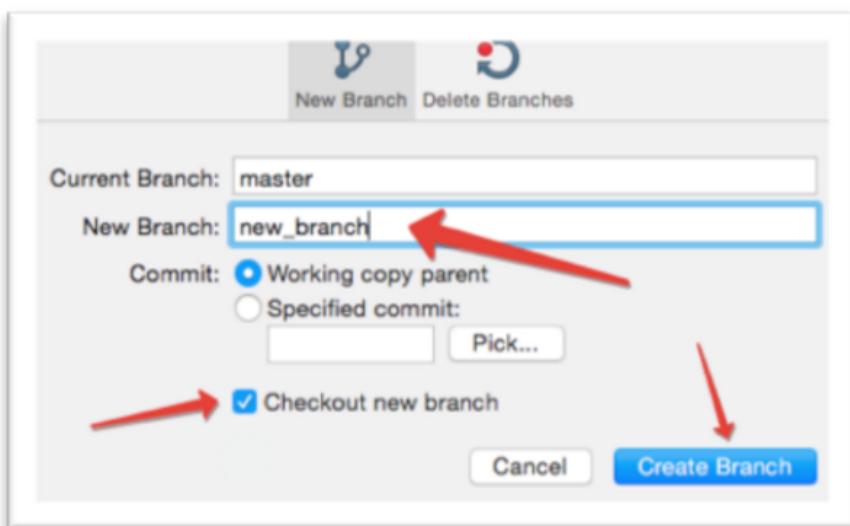
Ветвление (checkout/branch)

Бывает, что появляется необходимость исправить какой-то баг, либо начать работать над новым функционалом, но не поломать при этом текущий. Для этого существует механизм ветвления и, чтобы создать ветку, нужно нажать кнопку «Branch» на главной панели SourceTree.



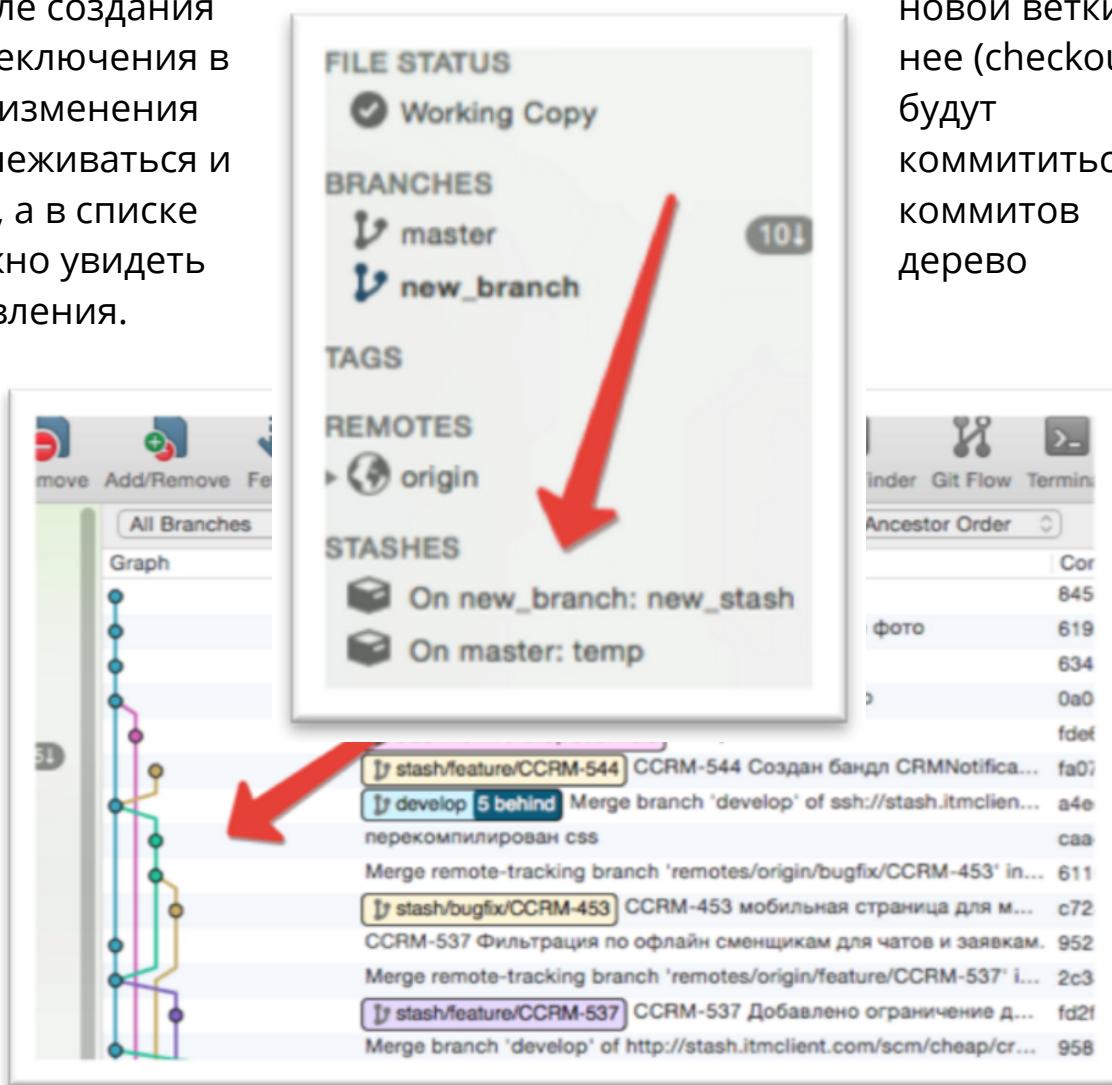
Кнопка «Branch»

В появившемся окне ввести название новой ветки, проверить, что установлена галочка «Checkout new branch» (переключиться на созданную ветку) и нажать кнопку «Create branch».



Опции создания новой ветки

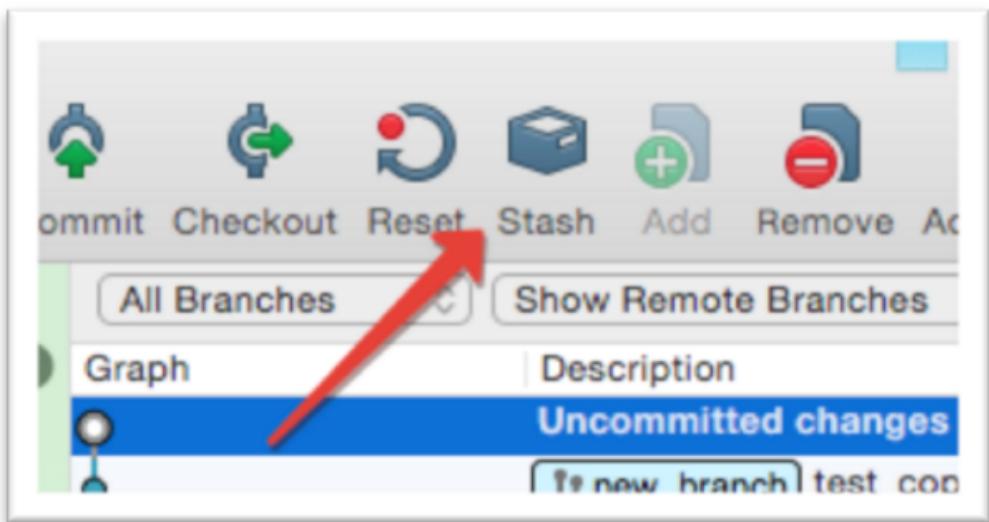
После создания переключения в все изменения отслеживаться и ней, а в списке можно увидеть ветвления.



Пример дерева веток

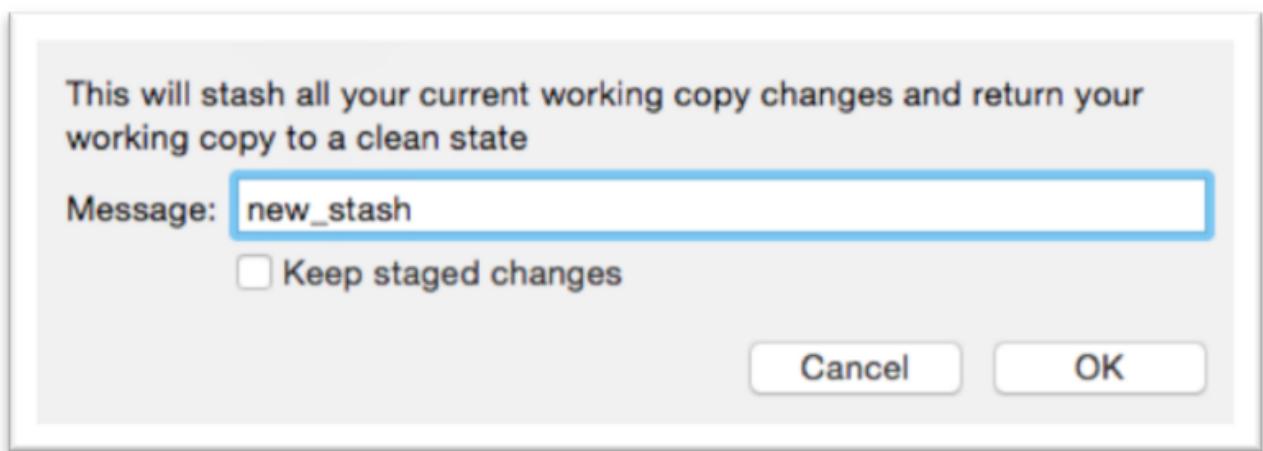
Прятки (stash)

В процессе разработки может возникнуть такая ситуация, когда необходимо переключиться в другую ветку, но текущие изменения проекта не должны быть потеряны, а коммитить их по тем или иным причинам — не стоит. Для решения этой проблемы данные изменения можно спрятать (положить в stash), а потом достать их из стэша и применить к рабочей копии. Для этого необходимо выделить файлы для сохранения и нажать кнопку «**Stash**» на главной панели инструментов.



Кнопка «Stash»

После этого в открывшемся окне ввести название стэша и нажать кнопку «OK».



Создание стэша

После этого созданный стэш появится в списке существующих стэшей проекта в левом блоке статуса файлов проекта SourceTree.

Список стэшей проекта

Для получения спрятанных в стэше изменений нужно кликнуть два раза по названию нужного стэша и нажать «**OK**».

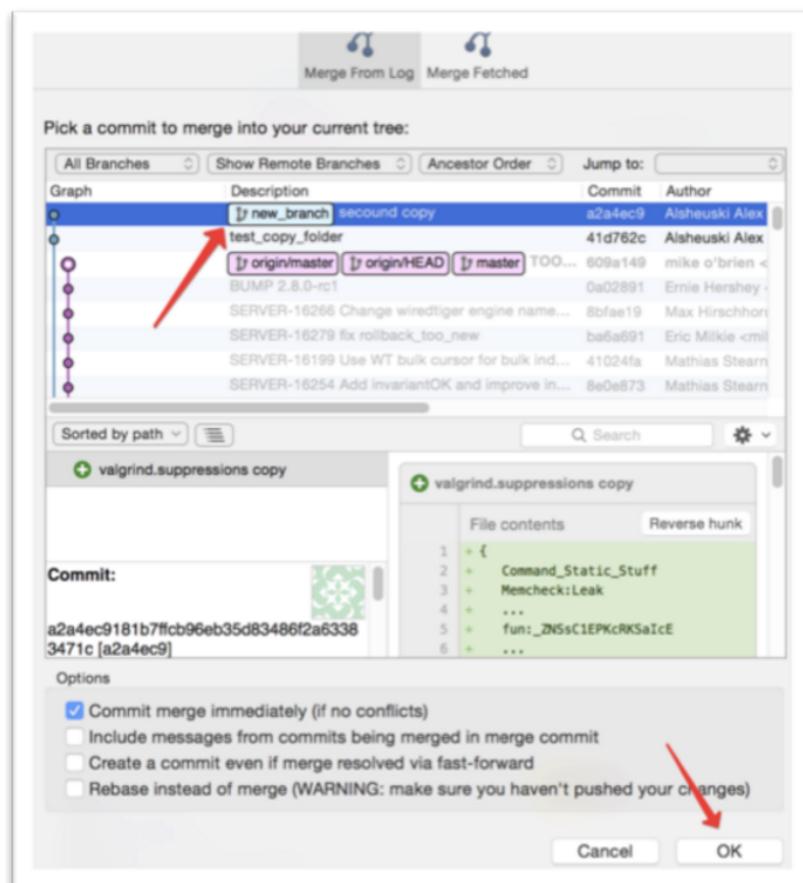
Слияние (merge)

После того, как новая фича реализована, либо поправлен баг, нужно добавить новые изменения в основную ветку разработки и для этого слить ветки (*merge*). Для операции мержа нужно переключиться (*checkout*) на ветку, в которую будут добавлены изменения из другой ветки и нажать кнопку «**Merge**» на главной панели инструментов.



Кнопка "Merge"

Далее
ветку, из
будут
изменения
будет
текущую),
опции по



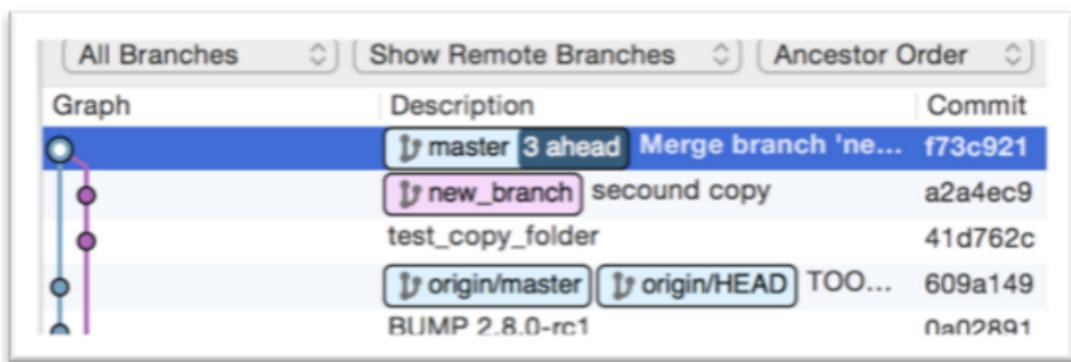
умолчанию
«OK».

выбрать
которой
получены
(которая
влияла в
оставить

и нажать

Опции мержса веток

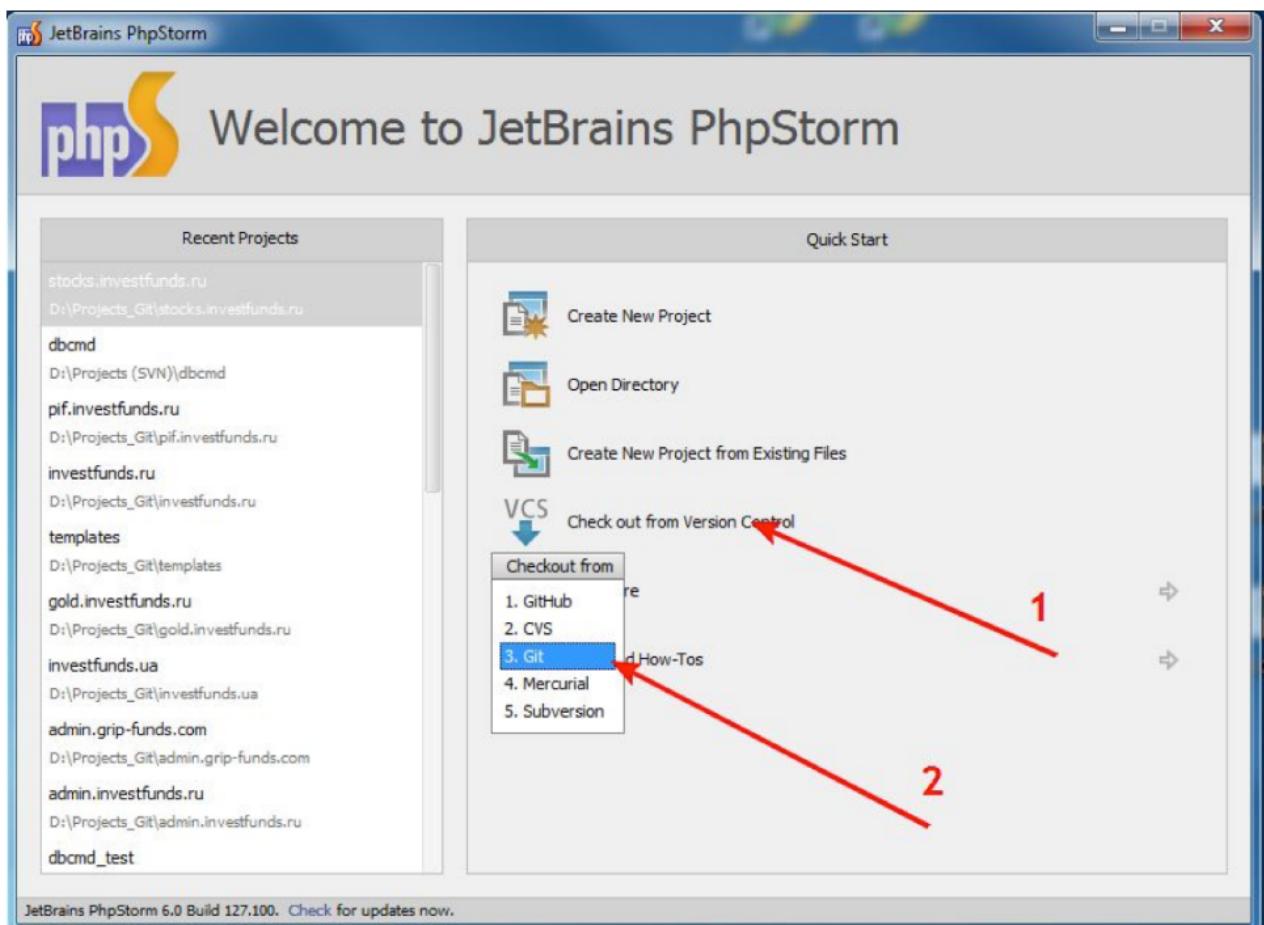
На этом мерж будет закончен.



Обновленное дерево веток после мержса

Работа из графического интерфейса PhpStorm

Получение репозитория



Менеджер проектов PhpStorm

Стандартные процедуры работы

1. Перед созданием своей ветки очень важно переключиться в ветку **master**, т.к. создаваемая ветка будет наследована от активной ветки.

Annotation 1 points to the 'Create and checkout new branch' button at the bottom of the editor.

Annotation 2 points to the 'New Branch' button in the Git branches dialog.

Git branches dialog:

- New Branch
- Checkout Tag or Revision
- Remote Branches
 - origin/1714.edit_html
 - origin/1906.new_ad_ground
 - origin/2370.new_ad_ground
 - origin/3182.cbr_coins
 - origin/4120.new_expandable_banner
 - origin/4340.life_investfunds_finishing
 - origin/5508.reklama
 - origin/5750.enc_footer
 - origin/6856.add_vote
 - origin/794.investfunds_ru_video
 - origin/904.footer
 - origin/HEAD
 - origin/master
 - origin/master_clean
 - origin/master_old
- Current branch: master

Правильный и неправильный вариант создания ветки от мастера:

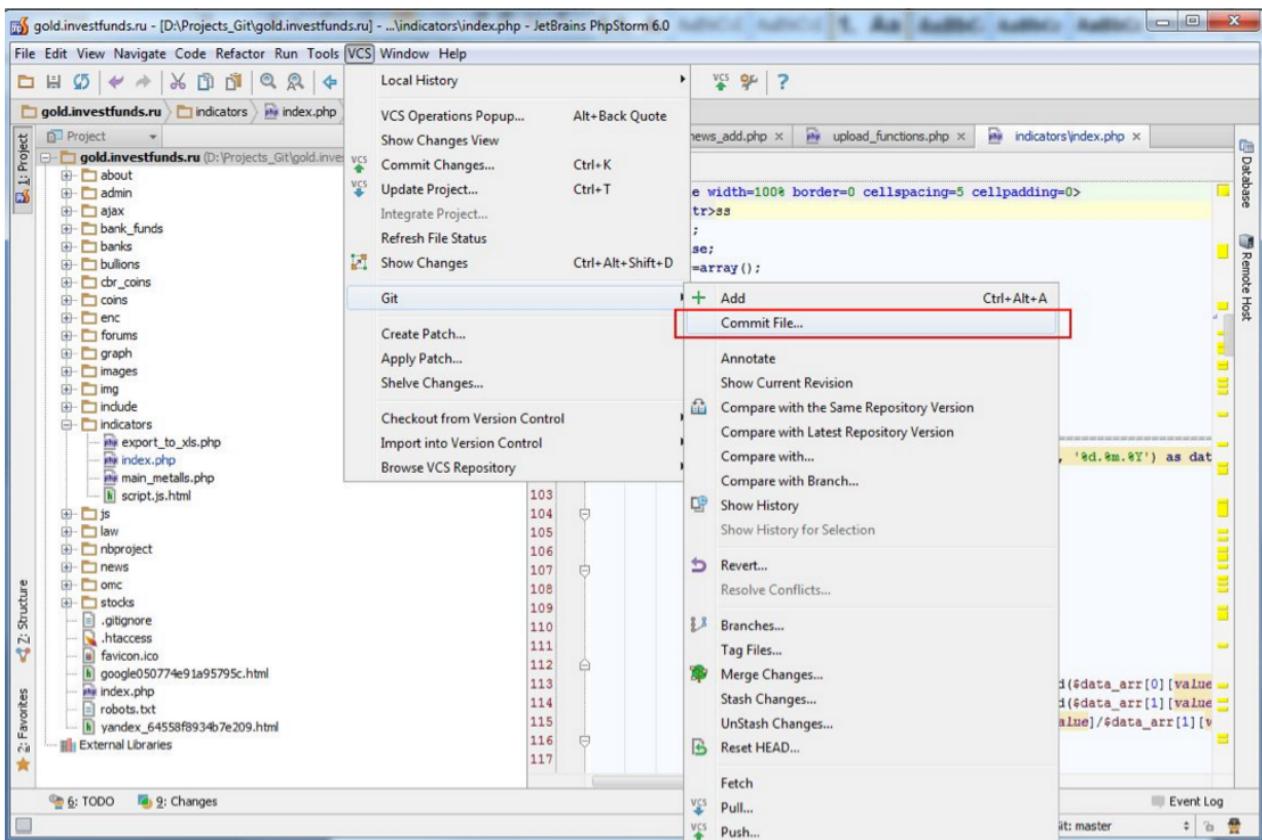
Annotation 'ok' highlights a successful creation of a new branch named '7800.rm_it_report' from the 'master' branch.

Annotation 'error' highlights an attempt to create a new branch from a non-existent branch '7800'.

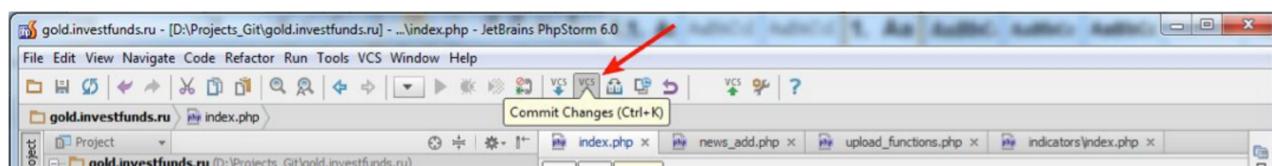
Git branches dialog:

- New Branch
- Checkout Tag or Revision
- Local Branches
 - 7800.rm_it_report
 - master -> origin/master
 - master_old -> origin/master_old
 - master_repo -> origin/master_repo
- Remote Branches
 - origin/10009.loans.comments
 - origin/10013.loan-page
 - origin/10015.pages.loans
 - origin/10035.em_news
 - origin/10147.pa_alerts
 - origin/10149.screens.patch
 - origin/10160.cb_auth.AdjacentProfileSid
 - origin/10206.loans.advertgrid
 - origin/10220.CbBondsYieldCurve.pref_currency
 - origin/10232.loans_link_on_global
 - origin/10318.is_dev_host

2. Коммит задачи. После изменения в файлах выбираем пункт меню «Commit File».



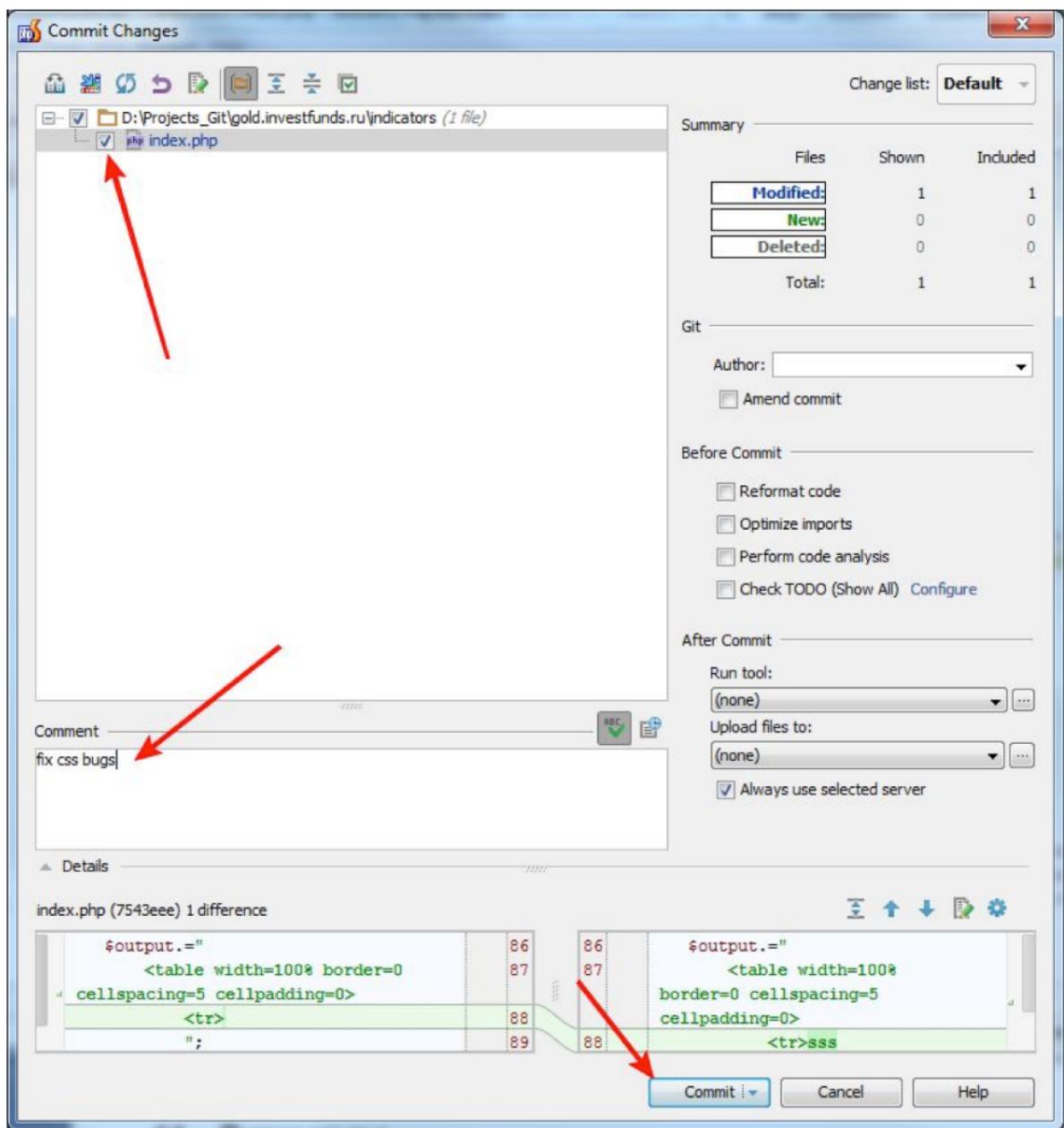
Или используем кнопку на панели программы:



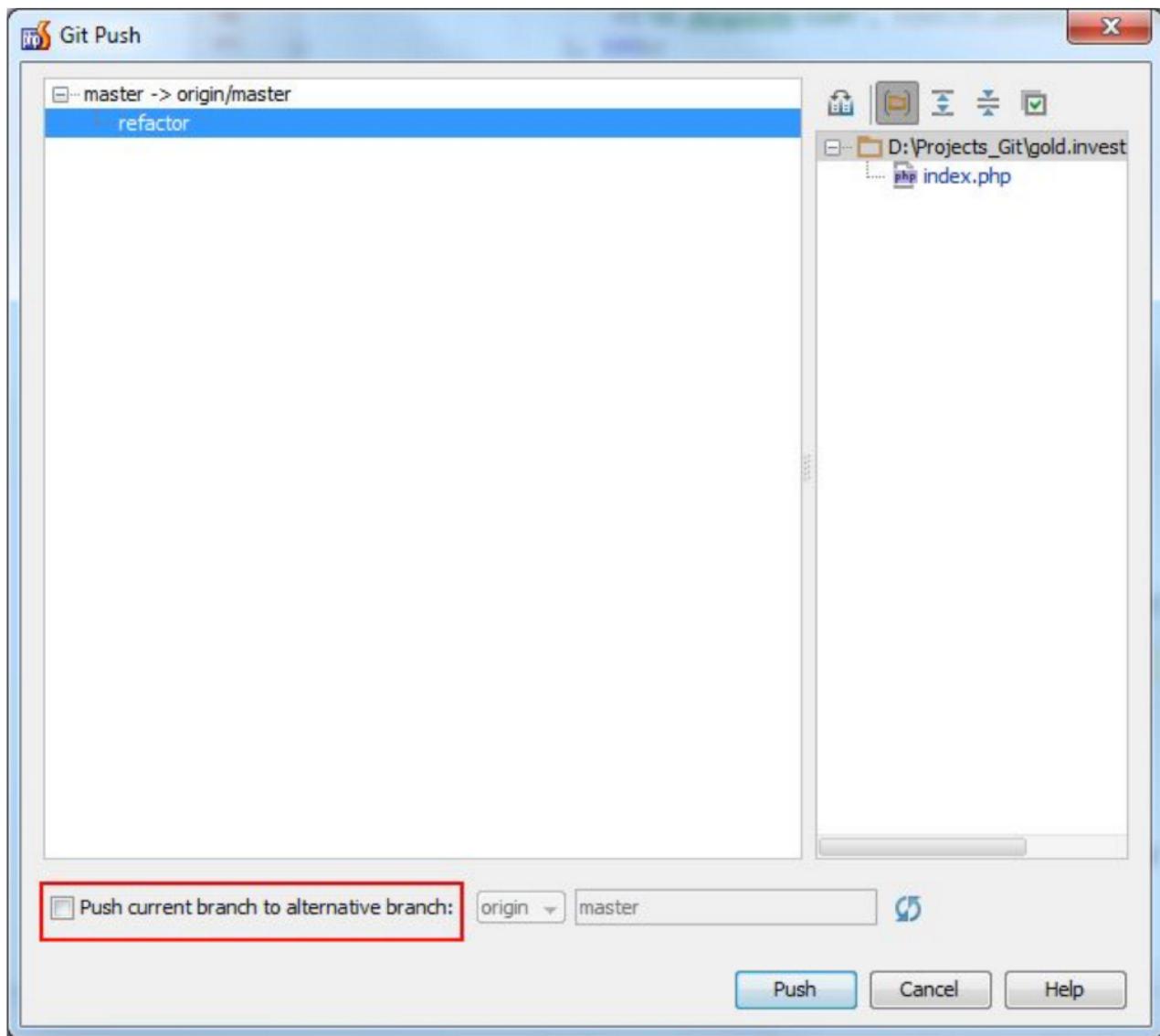
В появившемся окне:

- выбираем файлы для добавления в коммит;
- пишем комментарий к коммиту;
- в окне можно посмотреть изменения в файлах.

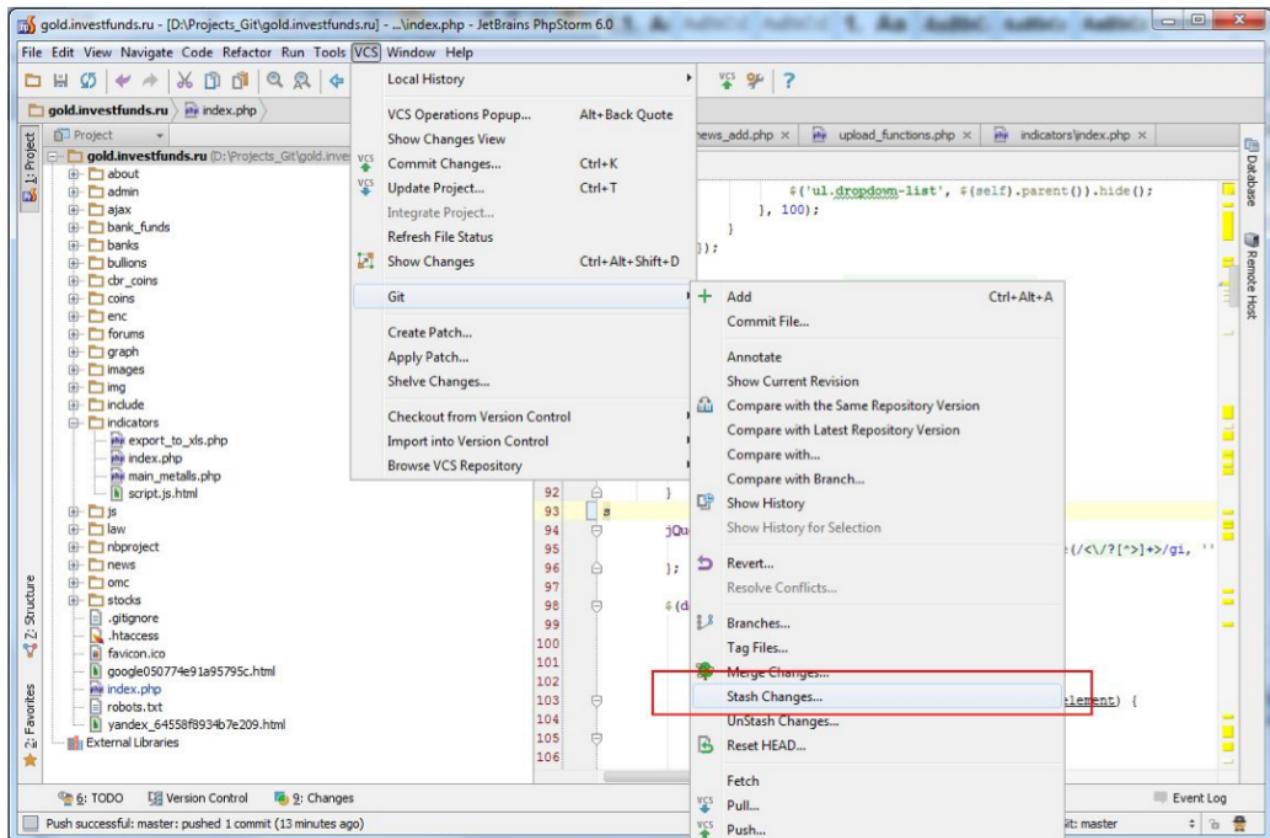
Нажимаем «Commit» или «Commit and Push»



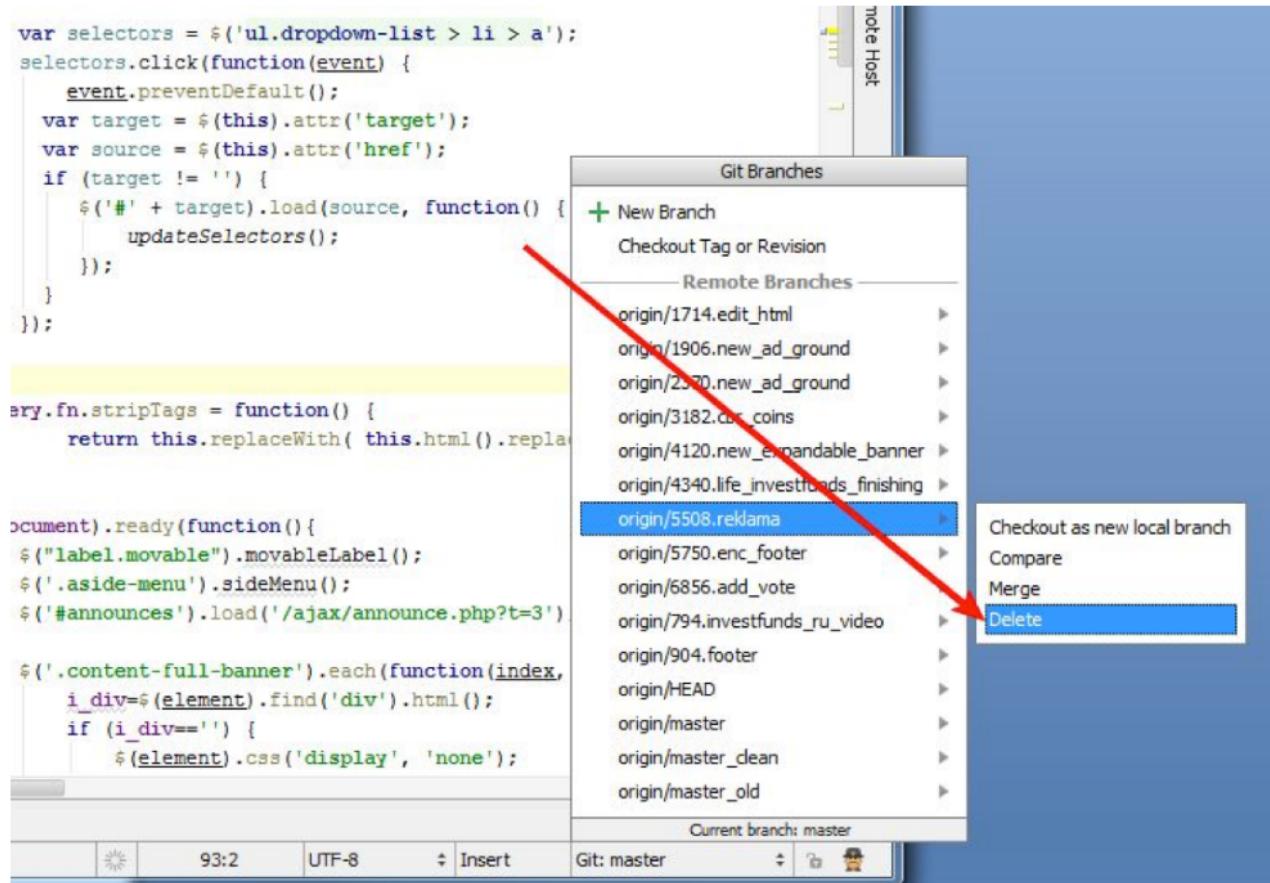
3. Отправка ветки на центральный репозиторий. Если Push происходит в первый раз в только что созданную ветку, то нужно поставить галку **“Push current ...”** для создания этой ветки на удалённом сервере.



4. Кратковременное сохранение состояния изменений.
Для этого нужно воспользоваться пунктом меню «**Stash**» – команда, сохраняющая изменения в отдельное хранилище.

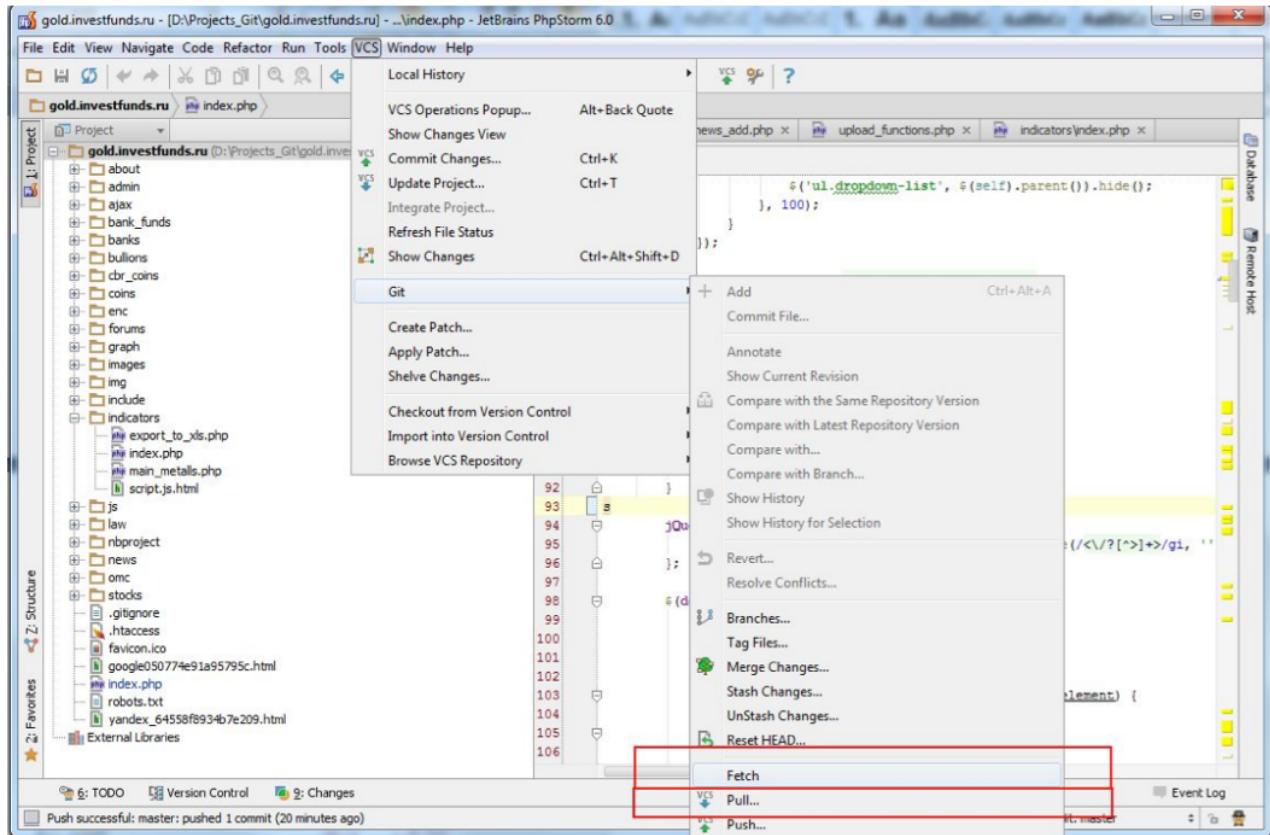


5. Удаляем ветки.



Ревью и выкладка задачи

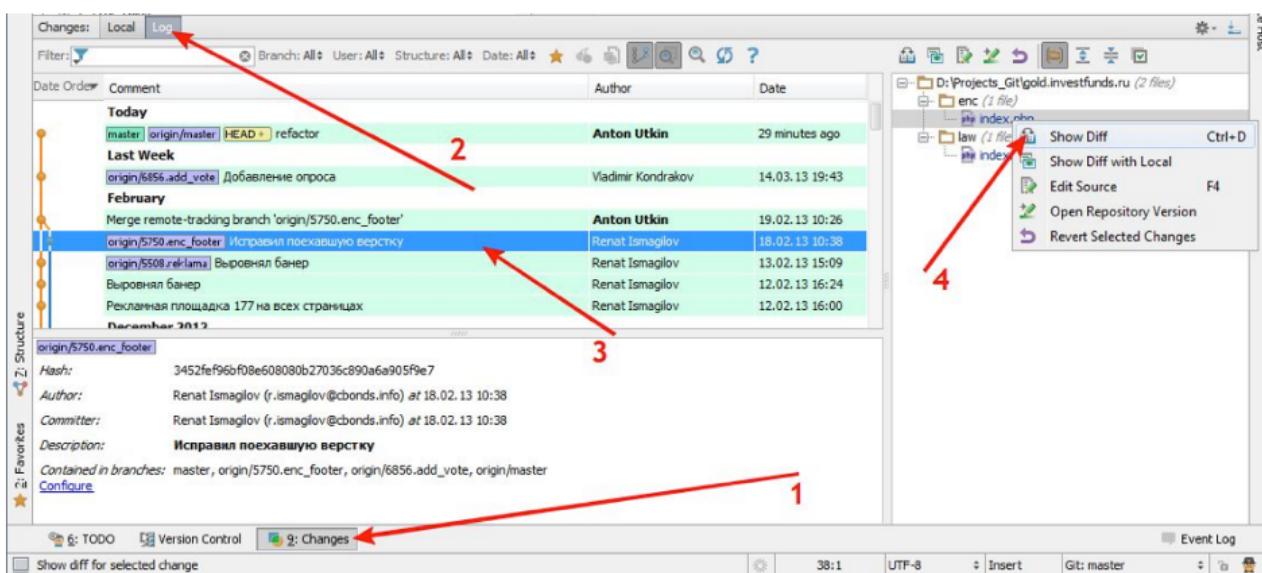
Обновление локального репозитория:



Fetch – обновляет историю, но не обновляет локальные файлы.

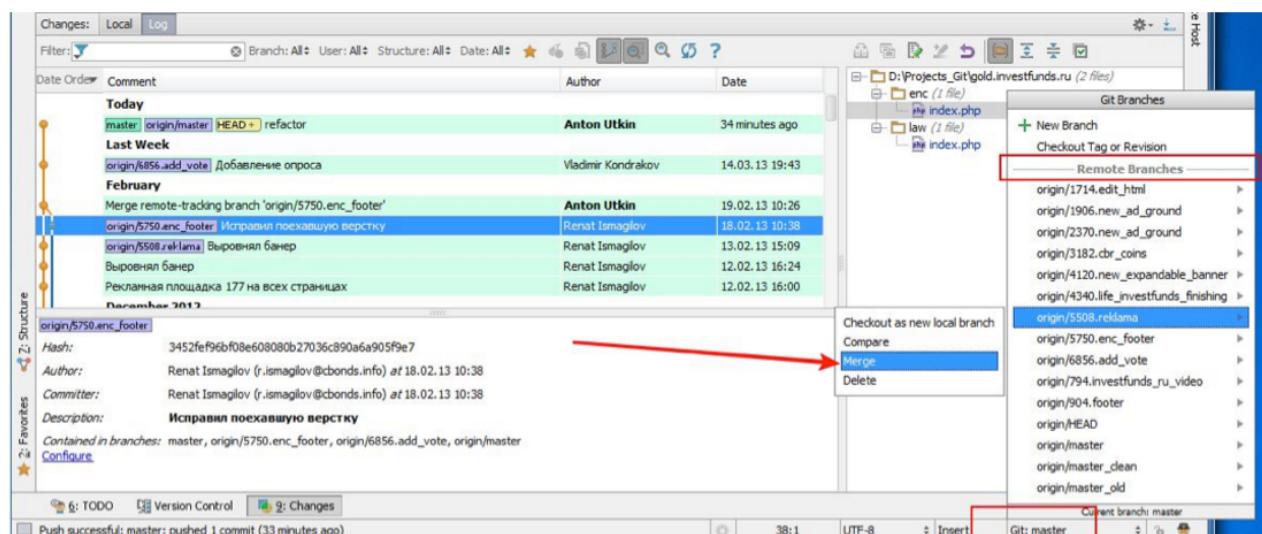
Pull – обновляет историю и локальные файлы.

Просмотр истории веток:



1. Раскрываем вкладку «Changes».
2. Выбираем вкладку «Log».
3. Находим ветку с нужной задачей.
4. Просматриваем изменения, внесённые в файл в выбранном коммите.

Добавление изменений из ветки в ветку master:



Находясь в локальной ветке **master**, выбираем пункт меню «**Merge**» у ветки, находящейся на удалённом сервере «**Remote Branches**»

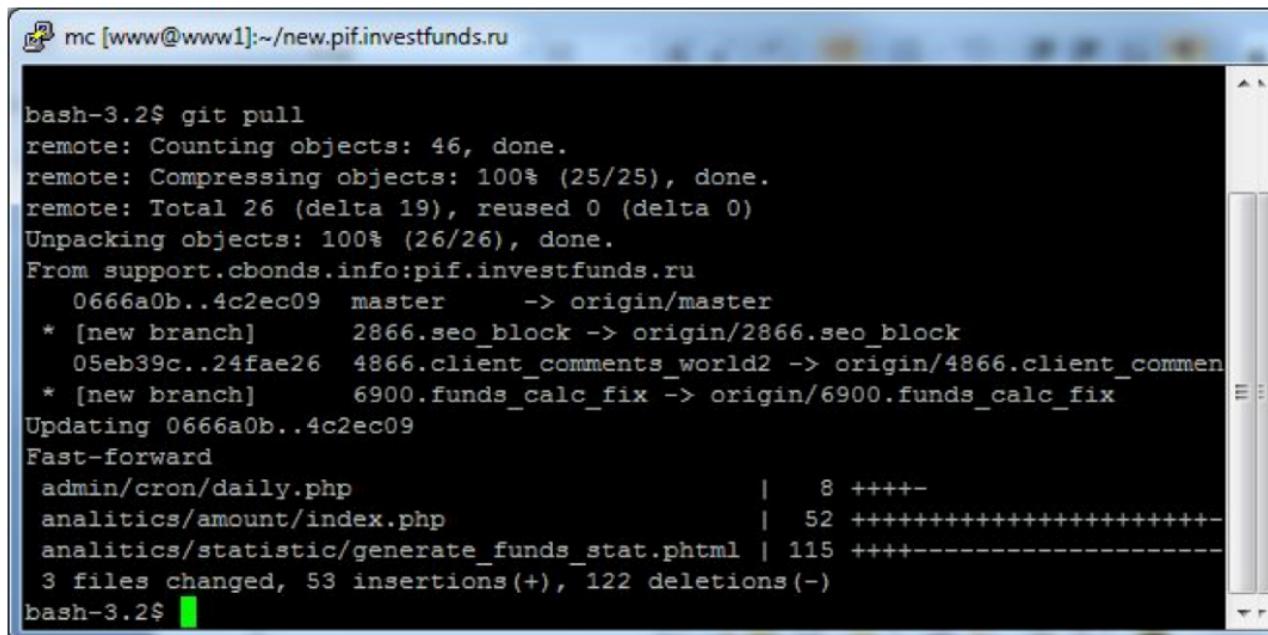
- Отправка изменений в центральный репозиторий:

Выбираем пункт меню «**Push**».

- Обновление ветки на ПРОД сервере

Набираем команду: **git pull**.

Смотрим, что изменения пришли.



```
mc [www@www1:~/new.pif.investfunds.ru]
bash-3.2$ git pull
remote: Counting objects: 46, done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 26 (delta 19), reused 0 (delta 0)
Unpacking objects: 100% (26/26), done.
From support.cbonds.info:pif.investfunds.ru
  0666a0b..4c2ec09  master      -> origin/master
 * [new branch]    2866.seo_block -> origin/2866.seo_block
   05eb39c..24fae26  4866.client_comments_world2 -> origin/4866.client_commen
 * [new branch]    6900.funds_calc_fix -> origin/6900.funds_calc_fix
Updating 0666a0b..4c2ec09
Fast-forward
 admin/cron/daily.php           |   8 +-----
 analytics/amount/index.php     |  52 ++++++-----+
 analytics/statistic/generate_funds_stat.phtml | 115 +-----+
 3 files changed, 53 insertions(+), 122 deletions(-)
bash-3.2$
```

Устранение конфликтов при слиянии

При операциях слияния веток (merge) могут возникать конфликты и слияния при этом не будут успешно завершены до исправления этих конфликтов.

Конфликт – это, фактически, различные изменения в одном и том же месте файла в различных ветках и приводящие к остановке операции мержа, так как Git не может самостоятельно решить, какие из изменений необходимо оставить.

Во время возникновения конфликта Git сообщит о том, что возник конфликт и в каких файлах он возник. Для успешного завершения слияния нужно вручную изменить файлы, содержащие конфликты, таким образом, чтобы в них осталась только нужная нам информация, а затем сообщить Git о том, что конфликты исправлены и закоммитить результат.

Для просмотра списка конфликтных файлов нужно выполнить команду **«git status»**. Во время выполнения слияния и появления конфликта Git переходит в состояние слияния и не выходит из него при конфликте. Можно не разрешать конфликт, а прервать слияние командой **«git merge --abort»**. Но в этом случае в проекте ничего не изменится.

Для разрешения конфликта вручную нужно открыть файл в каком-либо текстовом редакторе. В самом файле место конфликта помечается маркерами, как показано на скриншоте.

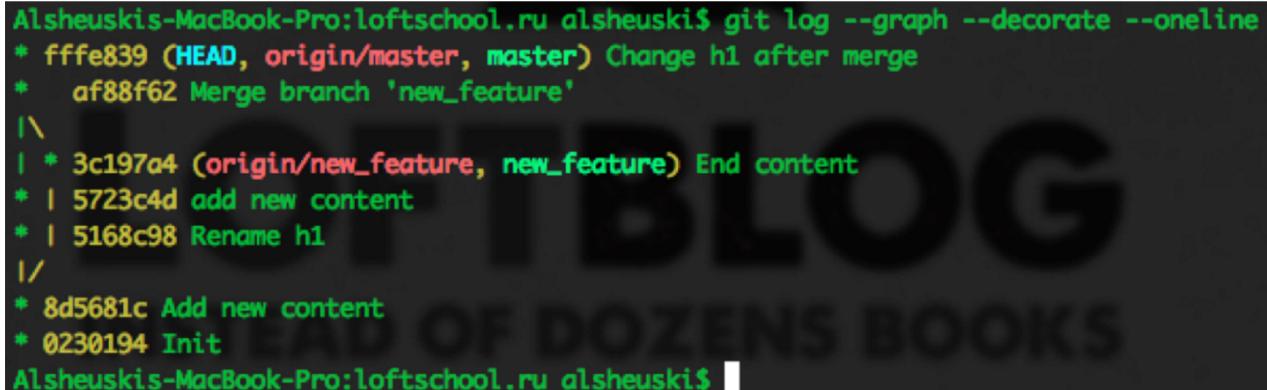
```
<body>
<<<<< HEAD
<h1>Hello from master branch!</h1>
=====
>>>>> new_feature
```

Секция «**HEAD**» – это содержимое текущей ветки назначения. Вторая секция – это содержимое источника, в нашем примере, ветки «`new_feature`».

Далее нужно удалить лишние данные и маркеры этого конфликта. В итоге мы должны получить файл, содержащий только нужную нам информацию без каких-либо дополнительных/лишних символов.

Затем нужно выполнить «**git add название_файла.html**» для того, чтобы показать Git, что все готово к коммиту и «**git commit**», чтобы их зафиксировать. Далее Git покажет список конфликтов, который нужно очистить, так как мы разрешили их вручную. После этого процедура слияния будет завершена.

При этом можно просмотреть результат слияния и состояние коммитов и веток. Для этого нужно выполнить команду «**git log --graph --decorate --oneline**». Результат вывода данной команды находится на скриншоте ниже.



```
Alsheuski-MacBook-Pro:loftschool.ru alsheuski$ git log --graph --decorate --oneline
* fffe839 (HEAD, origin/master, master) Change h1 after merge
* af88f62 Merge branch 'new_feature'
  \\
  | * 3c197a4 (origin/new_feature, new_feature) End content
  * | 5723c4d add new content
  * | 5168c98 Rename h1
  |
* 8d5681c Add new content
* 0230194 Init
Alsheuski-MacBook-Pro:loftschool.ru alsheuski$
```

При разрешении конфликта через программу, например, SourceTree, нам нужно сделать мердж, при возникновении конфликта кликнуть правой кнопкой мыши по конфликтному файлу для появления контекстного меню, далее выбрать «**Launch External Merge Tool**».

Появится окошко, содержащее с одной стороны содержимое одной версии файла, с другой стороны – другой версии файла. Различия между файлами будут выделены и направления принятия изменений буду показаны стрелочками. Какие различия нужно оставить,

выбирается из списка в нижней правой части экрана. Далее «Edit»-«Save Merge» и закрыть окно. Статус конфликта станет разрешённым автоматически.

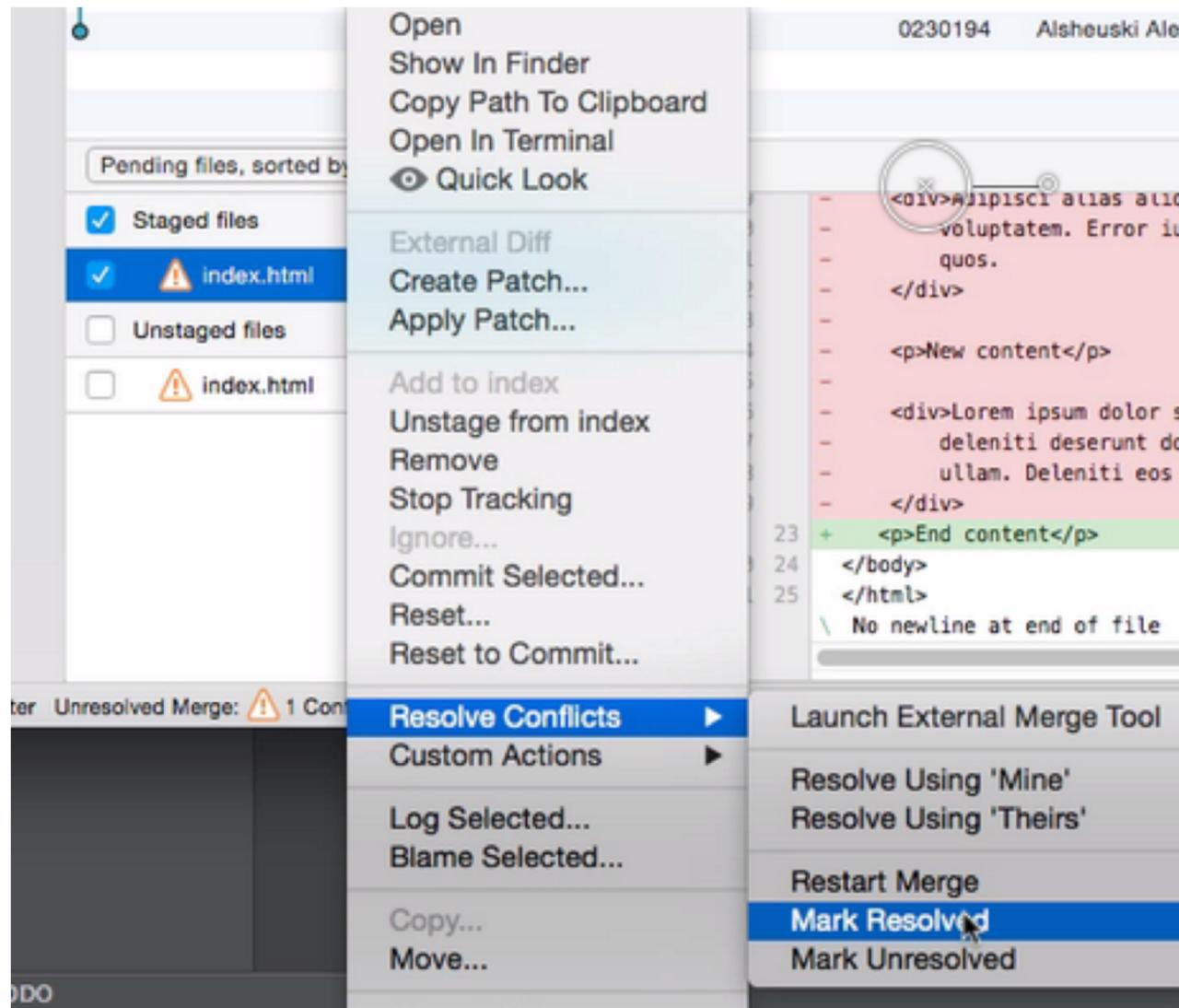
The screenshot shows a conflict resolution interface with two code editor panes. The left pane is titled 'index.html.LOCAL.15002.html' and the right pane is titled 'index.html.REMOTE.15002.html'. Both panes display the same HTML code for a 'loftschool.ru' page. Red arrows numbered 1 through 4 indicate specific conflict markers: 1 points to a 'h1' tag in the master branch; 2 points to a 'div' tag in the remote branch; 3 points to another 'div' tag in the master branch; and 4 points to the closing 'p' tag in the local branch. A large red arrow at the bottom right points towards the 'Actions' button in the status bar. The status bar at the bottom left shows 'status: 4 differences (3 left, 2 right, 1 conflict)'.

```
<!doctype html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Loftschool.ru</title>
</head>
<body>
    <h1>Hello from maste3213123r branch after merge!</h1>
    <p>content</p>
    <div>Lorem ipsum dolor sit amet, consectetur adipi
        dolore dolores eaque et ex explicabo harum in
    </div>
    <div>Assumenda debitis doloremque magni! Accusamus
        Comodi cum dolores 3123123excepturi fugiat in
        suscipit.
    </div>
    <div>A at ducimus, est fugit ipsam magnam minima m
        voluptates? Ad hic reprehenderit sequi vel. Do
    </div>
    <div>End content
    </p>
</body>
</html>
```

status: 4 differences (3 left, 2 right, 1 conflict)

Actions

Также можно перейти к конфликтному файлу в текстовом редакторе и выполнить редактирование конфликта. Уже после этого перейти к конфликтному файлу в SourceTree, кликнуть правой кнопкой мыши по нему для вызова контекстного меню и выбрать «Resolve Conflicts»-«Mark Resolved». После этого сделать коммит.



Пометка разрешения конфликта

После этого слияние можно считать завершённым.

Gitignore

.gitignore служит для указания в нём файлов и папок, которые необходимо скрыть от системы контроля версий git.

Как правило, скрывают файлы или папки, которые автоматически генерируют своё содержимое, либо имеют конфигурационные параметры, которые могут различаться у тех, кто совместно работает над проектом.

Правила синтаксиса

- Одна строчка - одно правило.
- Пустые строки игнорируются.
- Комментарии доступны через решётку (#) в начале строки.
- Символ "/" в начале строки указывает, что правило применяется только к файлам и папкам, которые располагаются в той же папке, что и сам файл .gitignore
- Можно использовать спецсимволы: звёздочка (*) заменяет любое количество символов (ноль или больше), вопросик (?) заменяет от нуля до одного символа. Можно размещать в любом месте правила.
- Две звёздочки (**) используются для указания любого количества поддиректорий.
- Восклицательный знак (!) в начале строки означает инвертирование правила, необходим для указания исключений из правил игнорирования.
- Символ "\" используется для экранирования спецсимволов, например, чтобы игнорировать файл с именем "!readme!.txt", нужно написать такое правило: «\!readme!.txt».
- Для игнорирования всей директории, правило должно оканчиваться на слэш (/), в противном случае правило считается именем файла.

Пример

```
# Игнор-лист файлов проекта.  
# Игнорировать ВСЕ файлы и директории, включая  
поддиректории и файлы в них.  
*  
# ----- ФАЙЛЫ -----  
# Игнорирование по типу файла, будут игнорироваться в  
АБСОЛЮТНО всех директориях.  
# Например: /files/data.zip, /server.log, /uploads/  
users/data/info.xls  
.zip  
.log  
.pdf  
.xls  
# Игнорирование файла во ВСЕХ директориях.  
# Например: /params/db/config.php, /config.php  
config.php  
# Игнорирование конкретного файла ТОЛЬКО в корне проекта  
# (корнем считается расположение файла .gitignore).  
# Например, НЕ БУДЕТ проигнорирован файл /db/config.php  
/config.php  
# Игнорирование конкретного файла ТОЛЬКО в указанной  
директории.  
# Например, НЕ БУДЕТ проигнорирован файл /prod/params/  
config.php  
/params/config.php  
# ----- ДИРЕКТОРИИ -----  
# Игнорирование всех файлов и папок ТОЛЬКО в конкретной  
директории (включая поддиректории и файлы в них).  
# Например: /images/user.jpg, /images/company/logo.png  
# НЕ БУДУТ проигнорированы файлы и папки /prod/images/  
user.jpg  
/images/*  
# Игнорирование всех файлов и папок в ЛЮБЫХ директориях  
с указанным именем.  
# Например: /images/user.jpg, /core/images/user.jpg
```

```
images/*
# Игнорирование ВСЕХ html-файлов в ОДНОЙ КОНКРЕТНОЙ
директории (НЕ ВКЛЮЧАЯ под директории).
# Например: /private/index.html
# НЕ БУДУТ проигнорированы файлы в /private/ivan/
index.html
/private/*.html
# Игнорирование ВСЕХ html-файлов в КОНКРЕТНОЙ директории,
ВКЛЮЧАЯ под директории.
# Например: /private/info.html, /private/users/ivan/
info.html
/private/**/*.*.html
# ----- РАЗНОЕ -----
# Исключение из игнорирования.
# Игнорирование ВСЕХ файлов и папок внутри директории /secret,
# за исключением файла /secret/free.txt, он не будет
проигнорирован
/secret/*
!/secret/free.txt
# Игнорирование файла с именем, содержащим спецсимволы
# Например: !readme!.txt
\!readme!.txt
# Игнорирование всех JPG и JPEG файлов внутри директорий,
# которые начинаются на "h" и МОГУТ содержать ещё один
символ после.
# Например: /images/h4/user.jpg, /images/h/company.jpeg
/images/h?/*.jp?g
```

Исключение для компьютера

Когда у вас несколько проектов и везде создается что-либо, что вы не хотите коммитить (например, *.swp файлы Vim) используйте **~/.gitconfig**. Вышеприведённый пример папки .idea, которая создается для каждого проекта, как раз подходит сюда. Создайте файл **.gitexcludes** и выполните:

```
git config --global core.excludesfile ~/.gitexcludes
```

или вручную добавьте в **~/.gitconfig**:

```
[core]
  excludesfile = ~/.gitexcludes
```

В файл **.gitexcludes** добавьте правила глобального игнорирования файлов во всех репозиториях на вашем компьютере.

Сценарий работы с системой контроля версий (на примере GIT)

1. В папке проекта создаем минимум два файла:

- **README.md** - файл с описанием вашего проекта
- **.gitignore** - файл со списком игнорируемых файлов и папок в репозитории

2. В консоли переходим в папку с проектом и выполним команду:

```
$ git init
```

3. Затем далее добавляем файлы в отслеживание через Git:

```
$ git add .
```

4. После добавления всех файлов в отслеживание нашим Git выполним команду:

```
$ git commit -m "Текст комментария к коммиту"
```

5. Для того, чтобы отправить все изменения из локального репозитория на удалённый, необходимо выполнить следующие команды:

```
$ git remote add имя_удалённого_репозитория  
url_удалённого_репозитория;  
$ git push -u имя_удалённого_репозитория имя_ветки;
```

6. Для внесения изменений в коммиты выполните один из следующих пунктов:

Изменение последнего коммита:

```
git commit --amend
```

Отмена индексации файла:

```
git reset HEAD ИМЯ_ФАЙЛА
```

Отмена изменений файла:

```
git checkout -- ИМЯ_ФАЙЛА
```