Методические указания по теме
“Работа с Git”

Содержание

1. Почему Git

Потому что не появляются задержки от работы 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, и древовидный просмотр изменений позволяет видеть что из какой ветки пришло.
Более подробно можно прочитать http://habrahabr.ru/blogs/Git/104198/

2. 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.

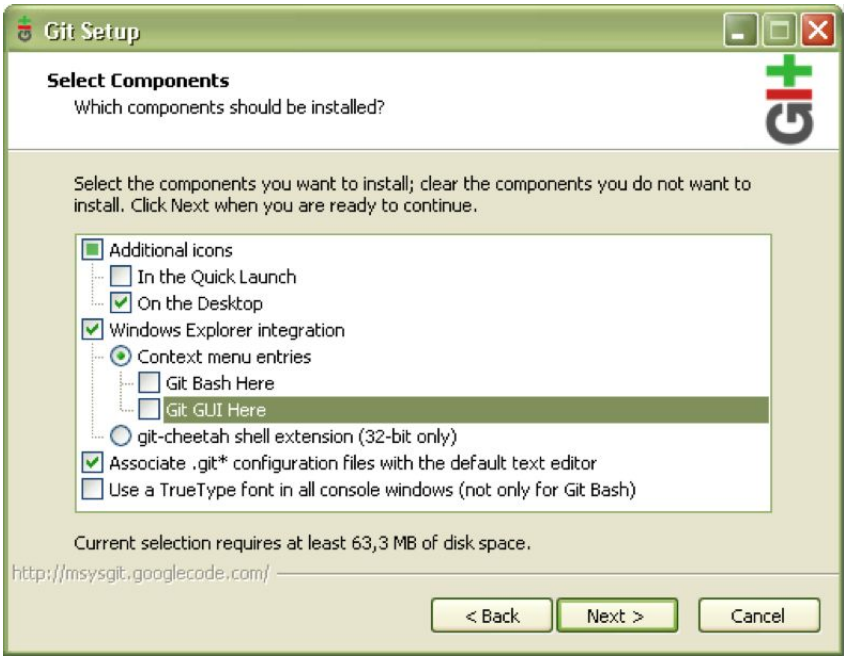
Для синхронизации изменений между разными копиями репозитория, в нужный момент делается pull чтобы скопировать изменения удалённого репозитория к себе, либо push чтобы скопировать локальные изменения в удалённый репозиторий.

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).

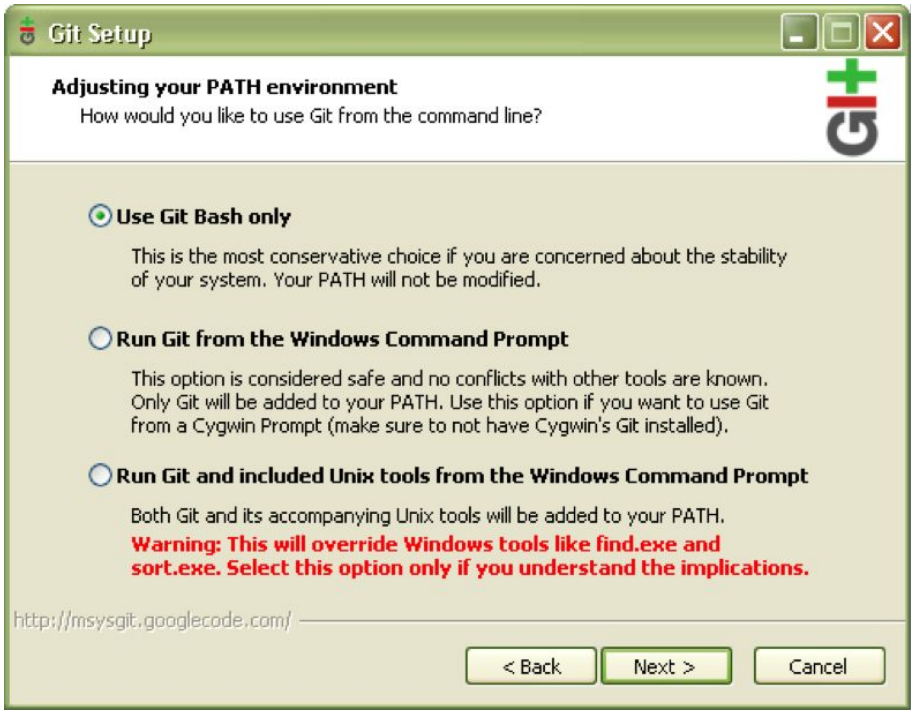
3. Установка Git под Windows

Устанавливается msysGit из проекта [http://msysgit.github.io.](http://msysgit.github.io) Нужно сказать и запустить на выполнение установочный файл.



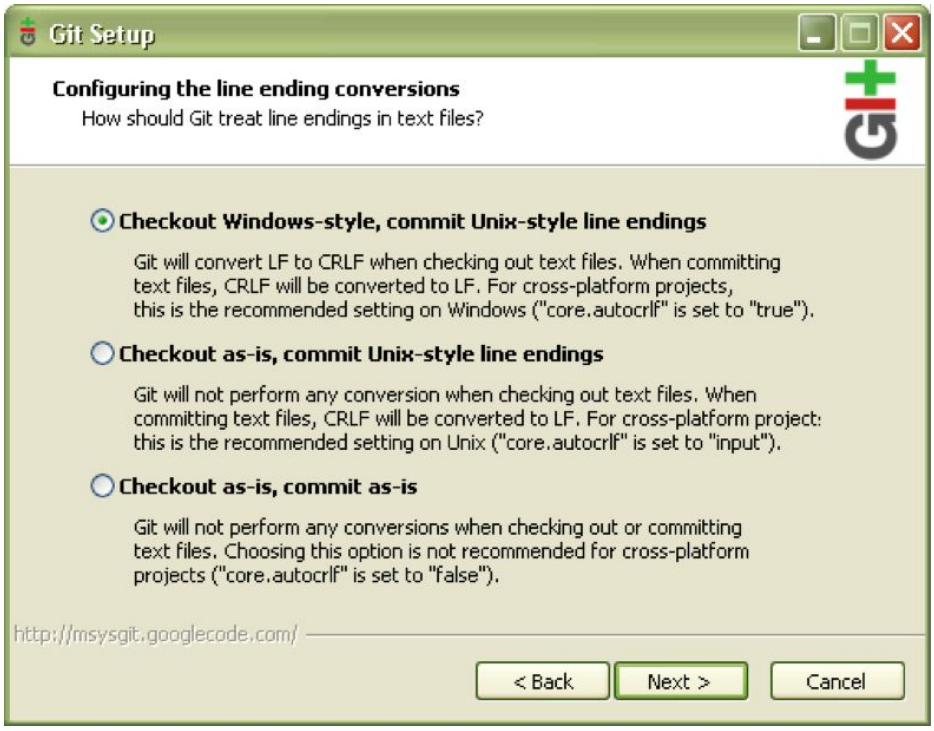
Выбор компонентов

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.

4. Работа через консоль

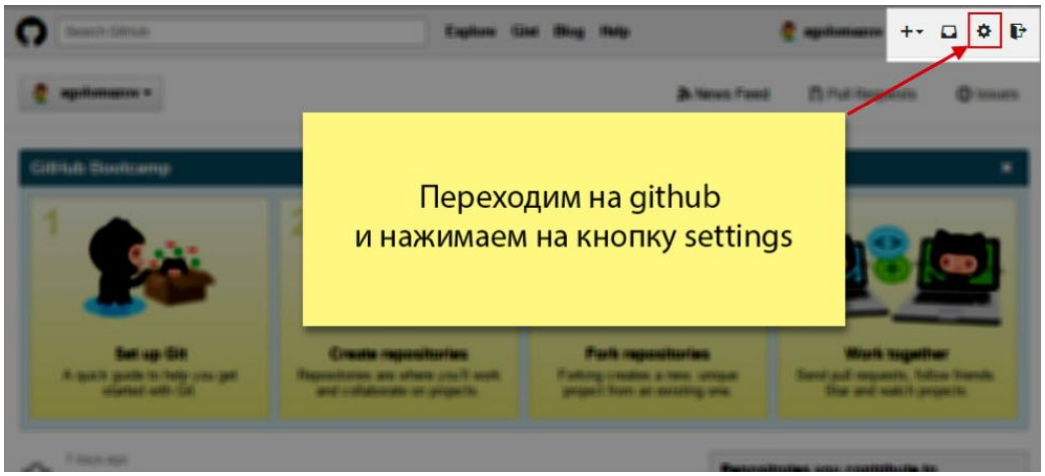
1. Открываем консоль
2. Для работы с Git необходимо настроить ФИО и Email автора:

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

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

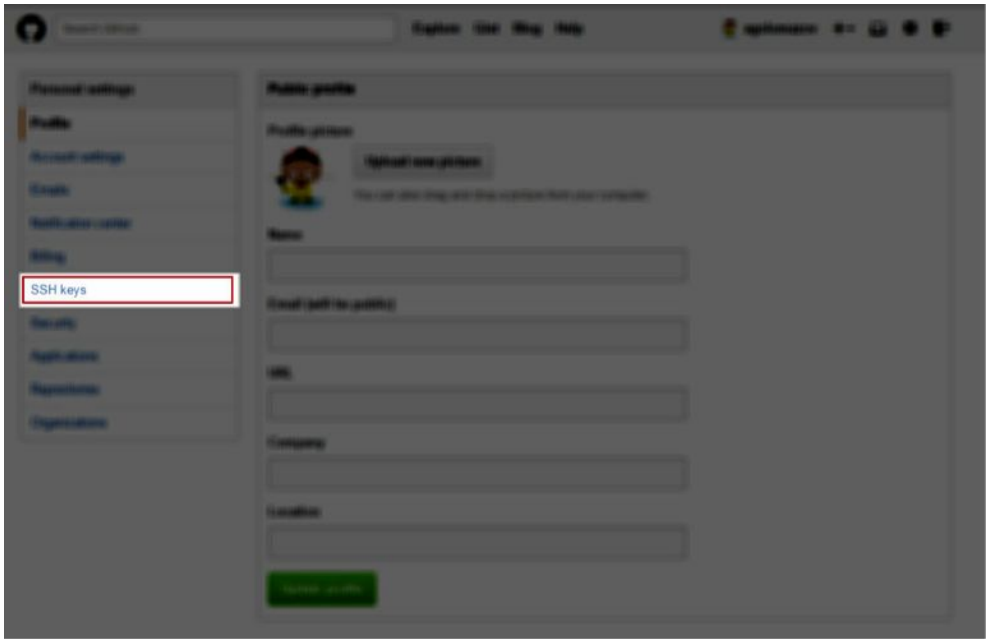
```
ssh-keygen -t dsa -C "Ivan Petrov <work@mail>"
```

4. Переходим на github и нажимаем на кнопку «Settings»



Кнопка “Settings”

5. Нажимаем на ссылку «SSH keys»

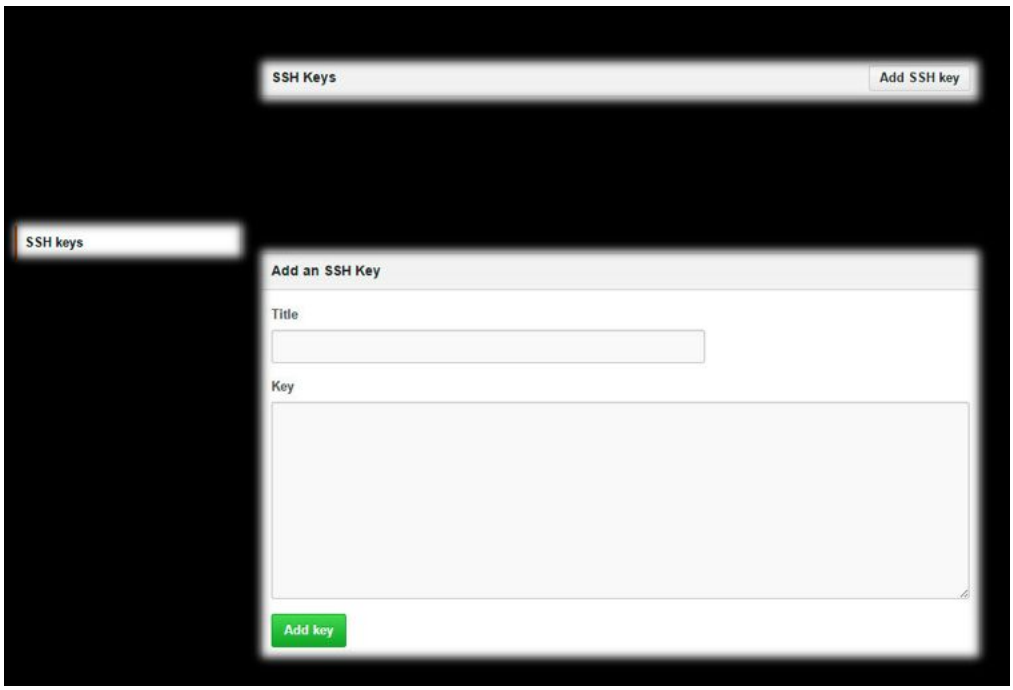


Пункт “SSH Keys”

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»

7. Далее, открываем putty-agent и добавляем сгенерированный закрытый ключ в putty



Добавление SSH-ключа

5. 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 проектов 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 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 всем 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.

6. Основные используемые функции

1. Получение всех коммитов за определенное время:

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

2. Получение списка коммитов определенного 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 введенных изменений в файл

```
git checkout -- ИМЯ_ФАЙЛА
```

6. Просмотреть изменение в буфере гита

```
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
```
1. сначало добавляем файл в .gitignore
2. git rm --cached ИМЯ_ФАЙЛА
```

17. Показать все отслеживаемые файлы в репозитории
```
git ls-tree HEAD
```

18. Показать список отслеживаемых файлов в определенной папке определенной ветки
```
git ls-tree master ИМЯ_ПАПКИ/
```

19. Список отслеживаемых файлов в определенном коммите
```
git ls-tree HASH-COMMITA
```

20. Показать сокращенный 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 определенном коммите
```
git log -p HASH-COMMIT index.html
```

23. Подробный показ изменений в ходе коммита в определенном 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 удаленного репозитория

```
git remote add origin URL_репозитория
```

33. Просмотр списка удаленных репозиториев

```
git remote –v
```

34. Удаление удаленного репозитория

```
git remote rm алиас_репозитория
```

35. Отправка локального репозитория на github.com

```
git push -u АЛИАС_УДАЛЕННОГО_РЕПОЗИТОРИЯ ЛОКАЛЬНАЯ_ВЕТКА
```

36. Просмотр списка удаленных веток

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

37. Клонирование репозитория с удаленного сервера

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

38. Отправка ветки на удаленный репозиторий

```
git push -u АЛИАС_УДАЛЕННОГО_РЕПОЗИТОРИЯ ЛОКАЛЬНАЯ_ВЕТКА
```

39. Синхронизация удаленного репозитория и локальной версии

```
git fetch АЛИАС_УДАЛЕННОГО_РЕПОЗИТОРИЯ
```

40. Объединение удаленной ветки

```
git merge origin/master
```

41. fetch + merge

```
git pull
```

42. Добавление ветки из удаленного репозитория в локальную версию

```
git merge ИМЯ_ЛОКАЛЬНОЙ_ВЕТКИ ИМЯ_ВЕТКИ_С_УДАЛЕННОГО_РЕПОЗИТОРИЯ
```

43. Удаление ветки из удаленного 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
```

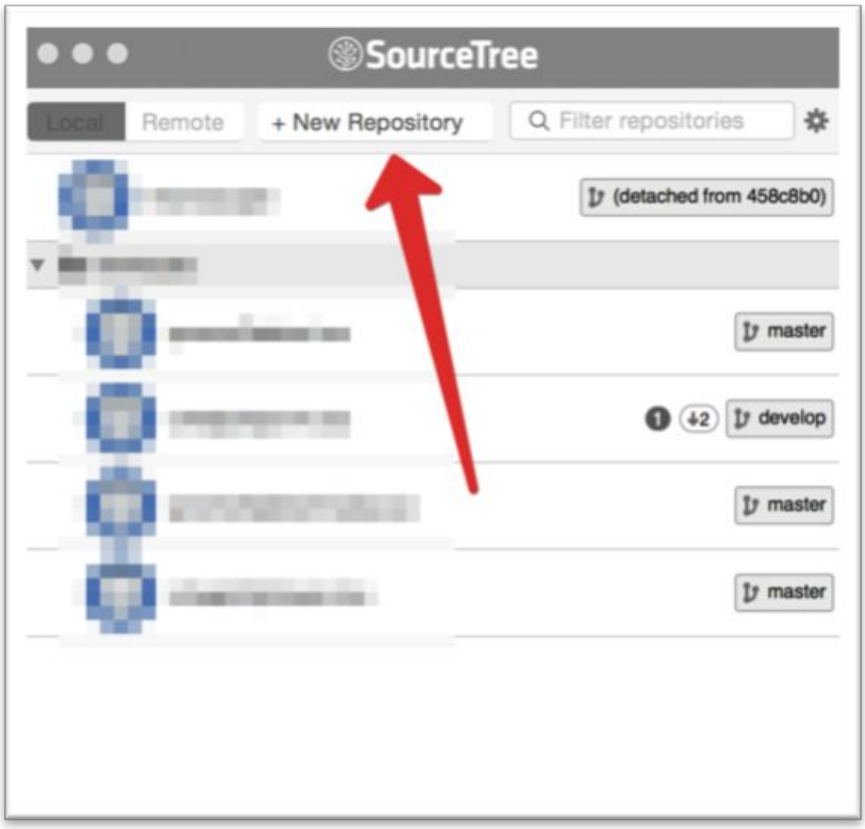
7. Работа из GUI SourceTree

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/. На ней можно найти различную промо-информацию, скриншоты программы и ссылки на получение установочных файлов.

В данном модуле будет представлено описание работы с репозиторием git и скриншоты программы в операционной системе Os X, принципиальных отличий от версии, доступной для Windows нет и все делается аналогично.

Создание/клонирование репозитория (clone/init)

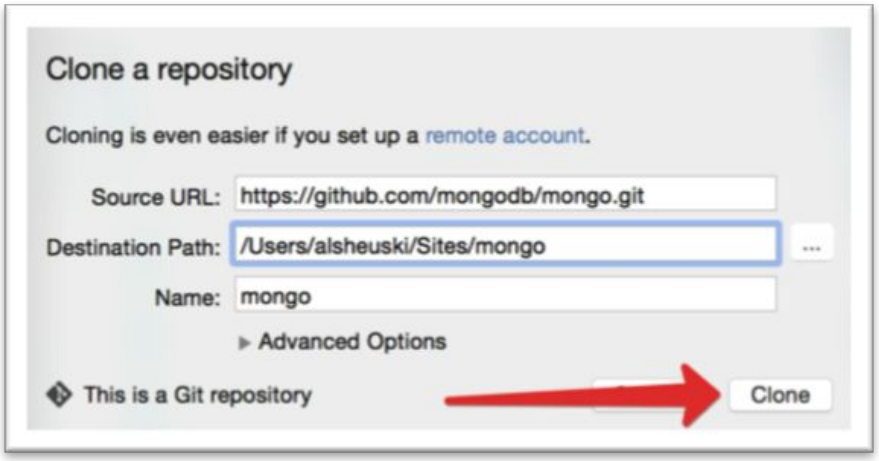
Для начала работы с проектом необходимо его клонировать/создать. Для этого запускаем SourceTree и нажимаем кнопку “New Repository” (“Новый репозиторий”) и выбираем один из пунктов. В данный момент нас интересуют пункты “Clone from URL” (“Клонировать из URL”) и “Create local repository” (“Создать локальное хранилище”), так как это основные варианты с которыми придется работать.



Менеджер проектов SourceTree

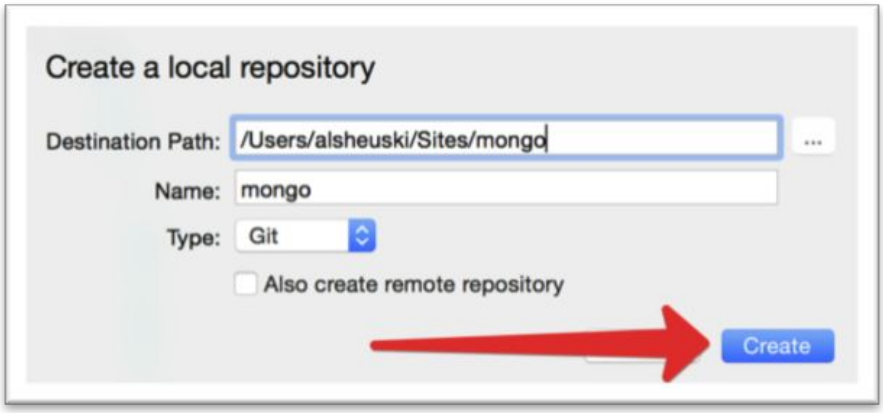
При выборе “Clone from URL” в новом окне нужно ввести в поле «Source URL» сетевой адрес репозитория, указать локальную папку в которую будет клонирован проект, название ставится

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».



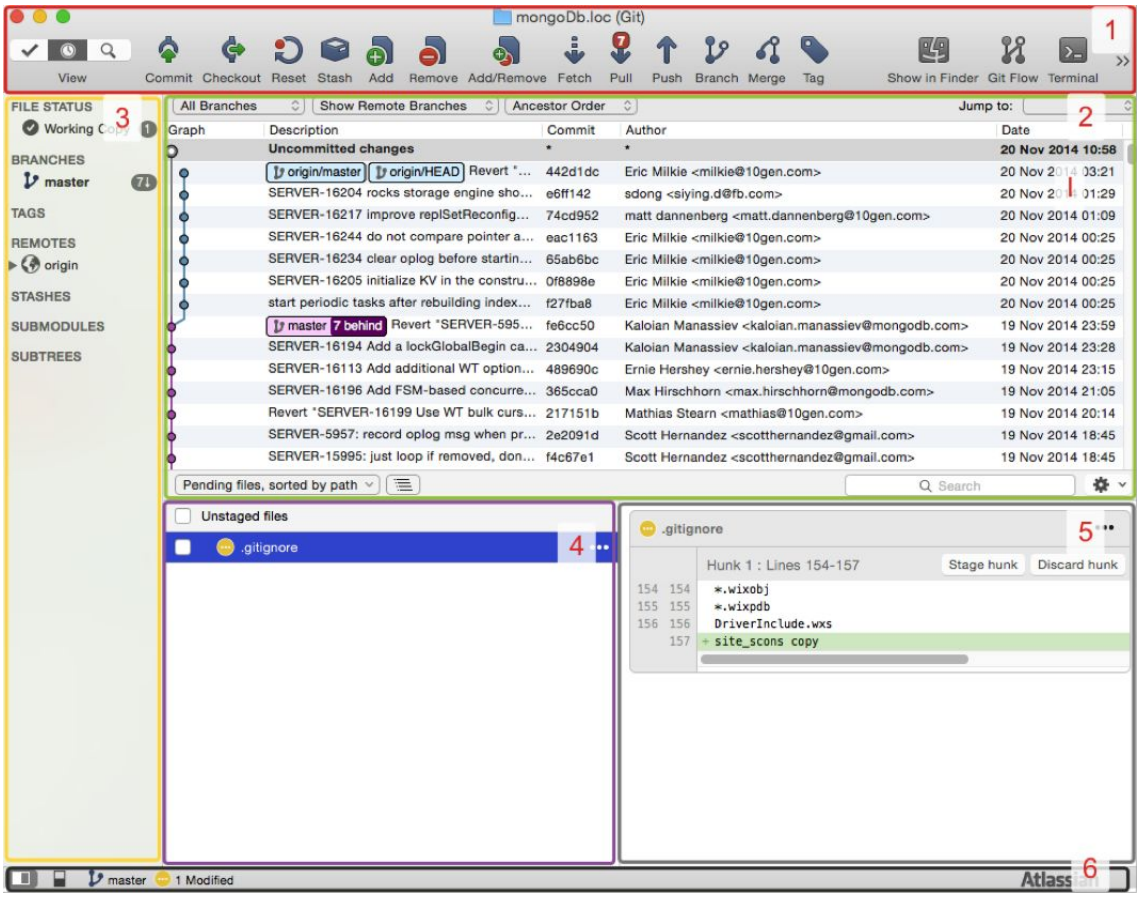
Init нового проекта

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 по номерам следующее:

1. Блок с кнопками для выполнения базовых операции git, таких как push/pull/commit/checkout и т.д.

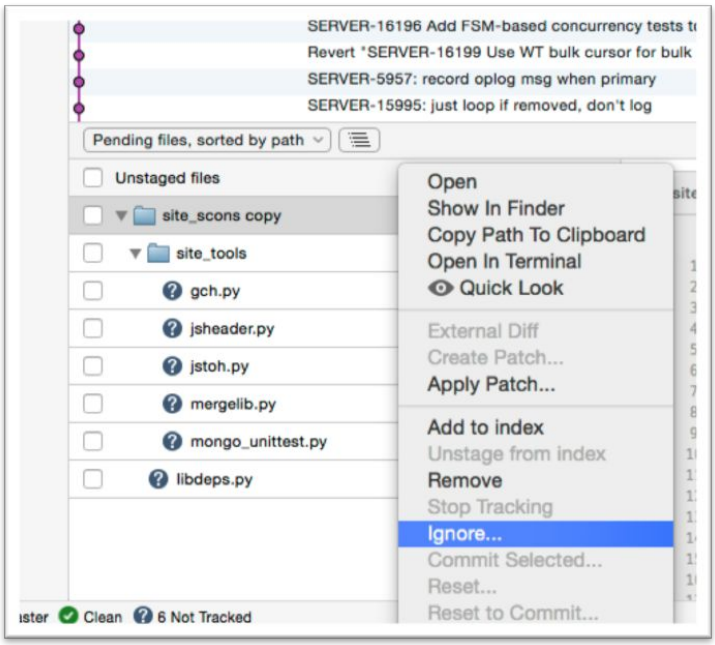
2. История ветвления и коммитов, содержащая основную информация о коммитах в виде списка и наглядно представляющее историю коммитов и их ветвление в виде дерева.
3. Блок, содержащий списки веток, тэгов как локальных так и удаленных, сохраненных (спрятанных) изменений в стэше (stash) и т.д.
4. Список 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 о установленных пакетах и их зависимостях в специальный файл. Для npm это packages.json и для последующей установки/разворачивании проекта в другом месте используют информацию из данного файла что бы 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…».

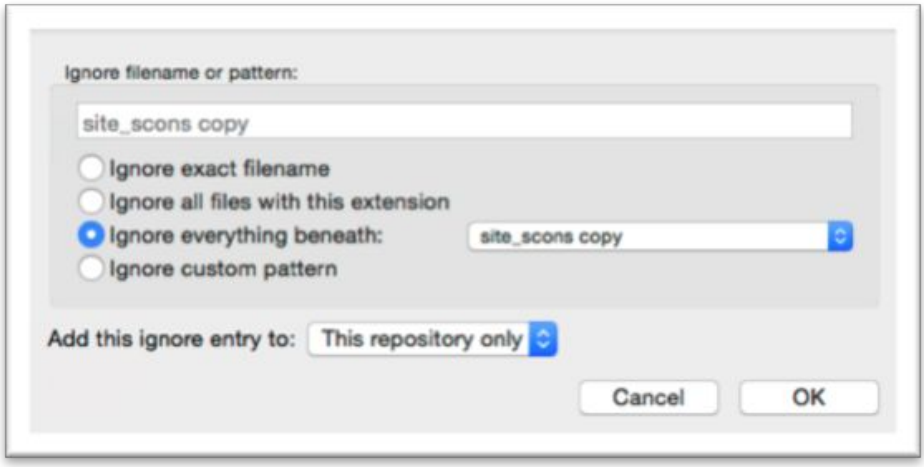


Добавление файлов в исключения

Далее выбор опций:

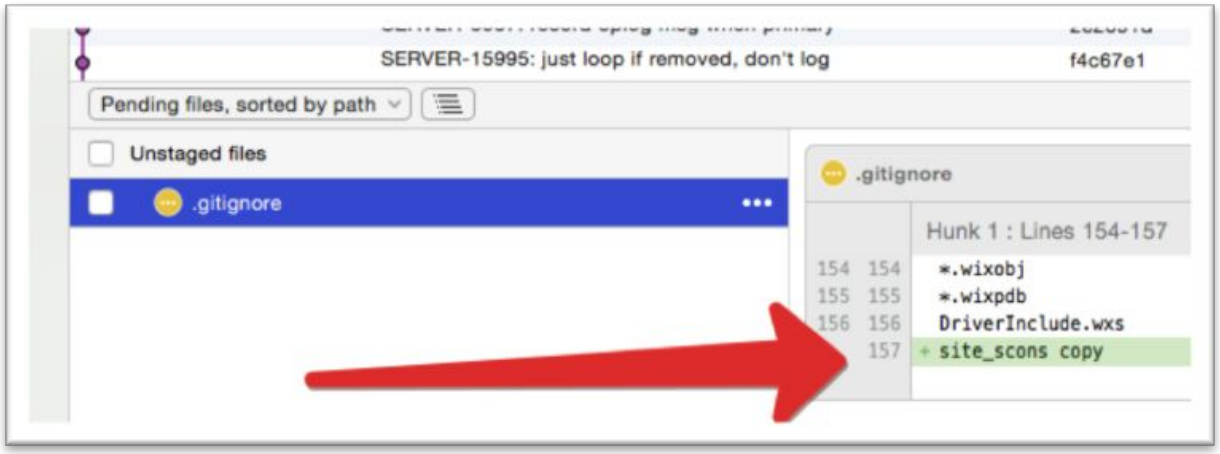
1. Ignore exact filename – игнорировать файл с указанным именем
2. Ignore all files with this extension – игнорировать все файлы с указанным расширением

3. Ignore everything beneath – игнорировать все 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 изменен и данные изменения нужно закоммитить.



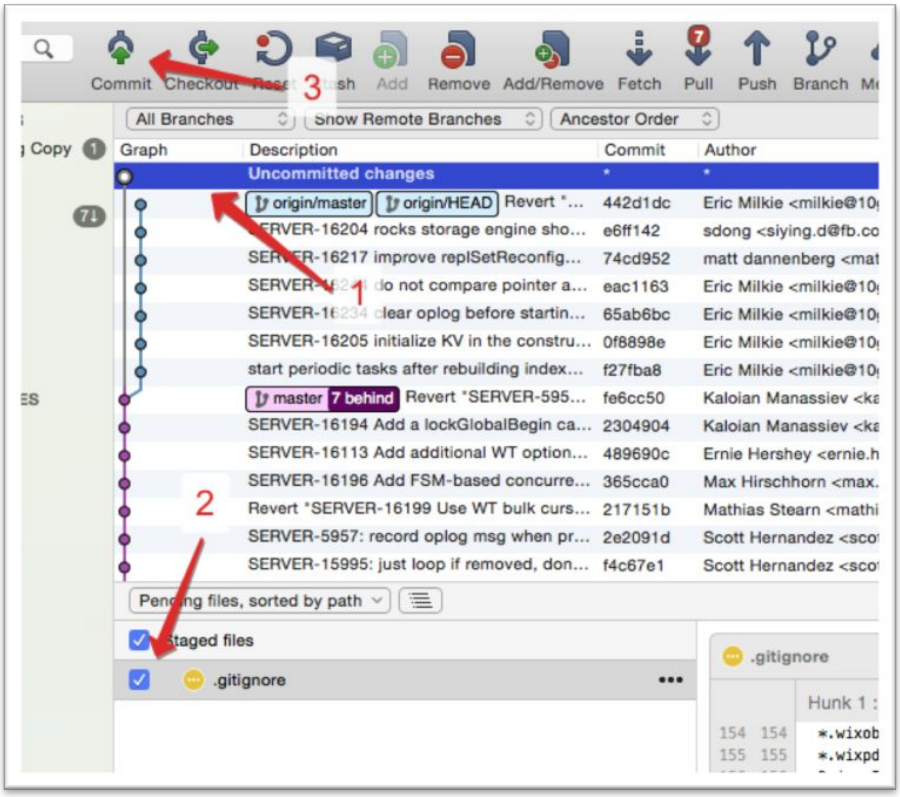
Изменение файла .gitignore

Коммит и пуш (commit/push)

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

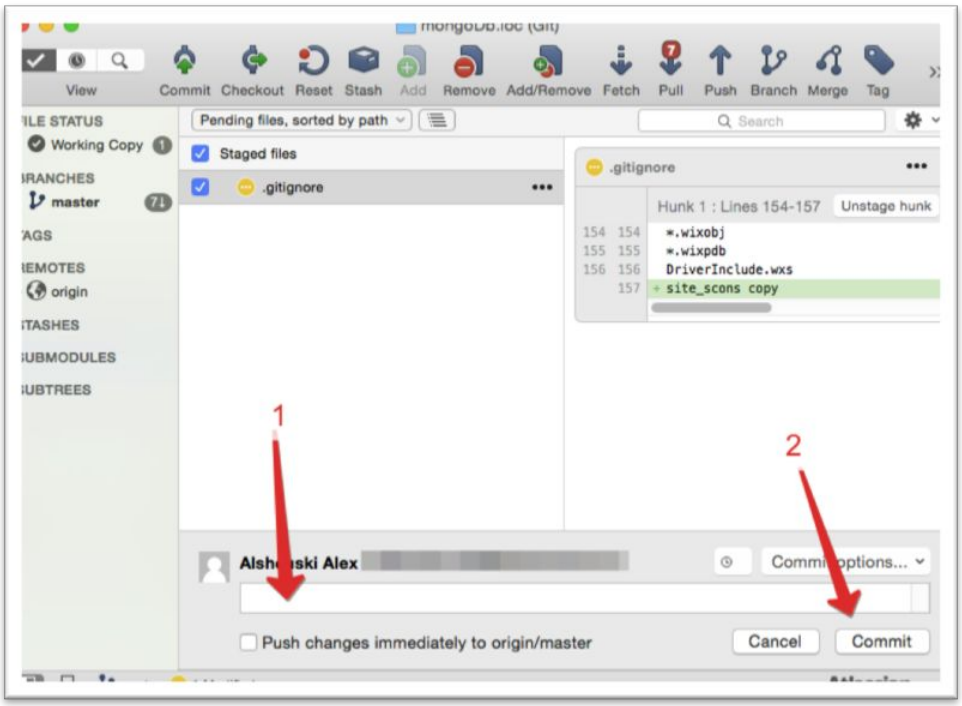
Далее (2) в блоке измененных файлов выбрать галочками выбрать необходимые для добавления в коммит файлы (выбраны не обязательно должны быть все файлы, только те, изменения в которых нужно зафиксировать на данный момент).

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



Выбор файлов для коммита

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

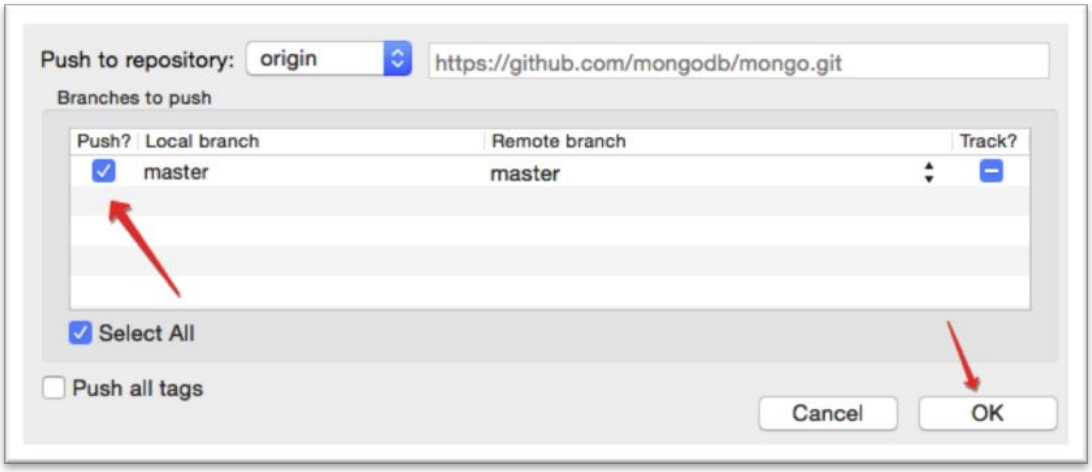


Создание коммита

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

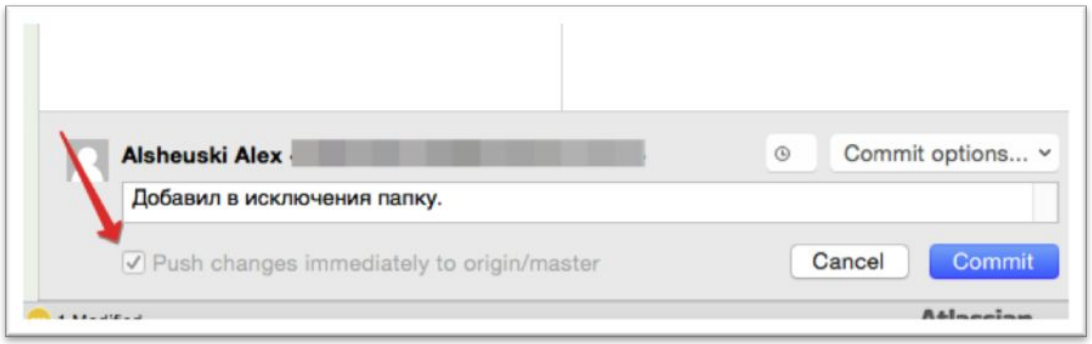
Кнопка “Push” с количеством коммитов для отправки

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



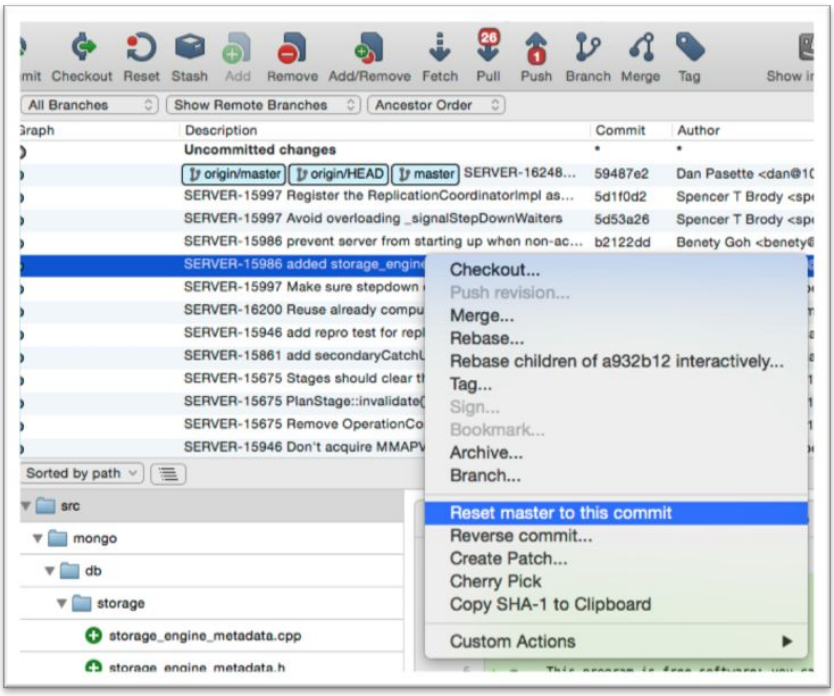
Push изменений

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



Откат изменений (reset)

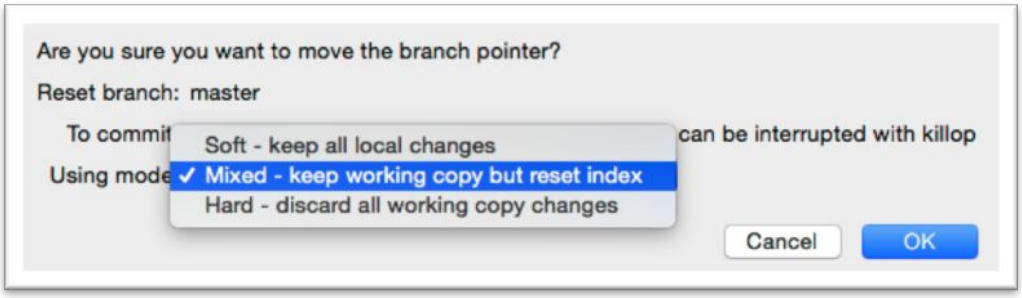
После сохранения изменений может появиться необходимость откатиться на состояние файлов в каком-либо коммите. Для этого необходимо найти и выделить нужный коммит в списке коммитов и кликнуть по нему правой кнопкой мыши и выбрать «Reset master to this commit».



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



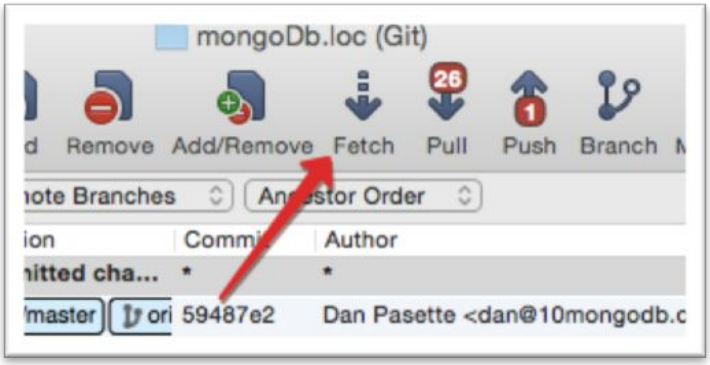
Выбор режима сброса

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

Получение обновлений (fetch/pull)

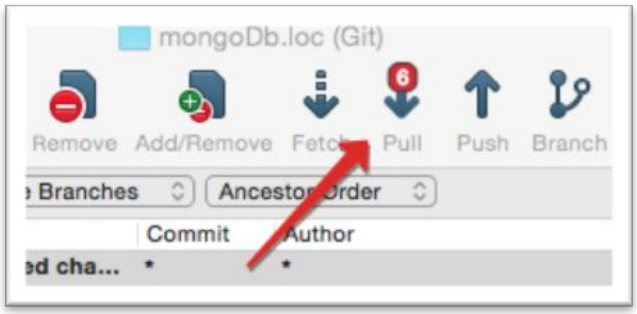
При коммандной работе над проектом либо работе соло, но в разных местах в репозитории будут появляться коммиты которых нету локально. Для синхронизации текущей работей копии приекта и получения изменений неоьбходимо сделать 2 вещи.

1. Получить изменений в удаленном репозитории. Для этого необходимо нажать кнопку «Fetch»



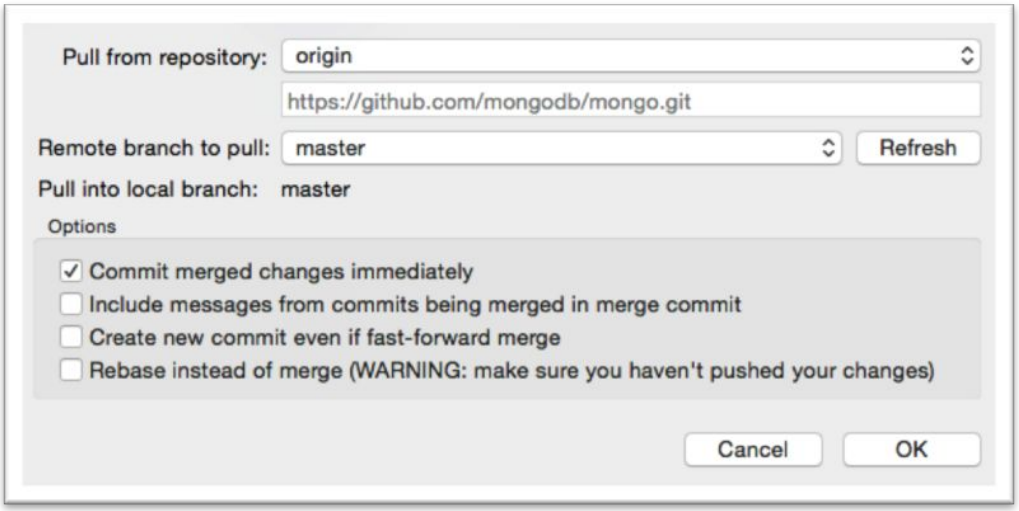
Кнопка Fetch

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



Кнопка «Pull»

Убедиться в том, что в следующем окошке выбрана верная текущая ветка и стоит галочка “Commit merged changes immediatly” и нажать «ОК».



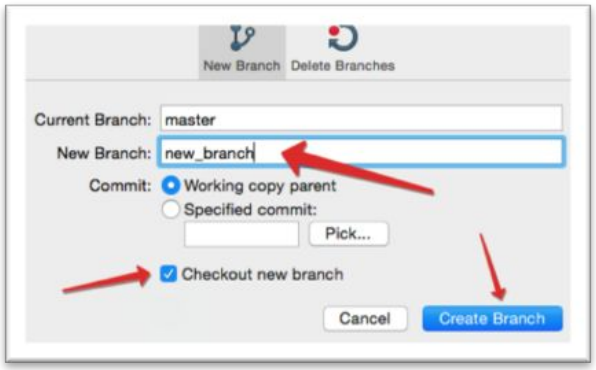
Опции «Pull»

Ветвление (checkout/branch)

В какой-то момент может появиться необходимость исправить какой-то баг либо начать работать над новым функционалом, но не поломать при этом текущий. Для этого существует механизм ветвления и что-бы создать ветку нужно нажать кнопку «Branch» на главное панели SourceTree.

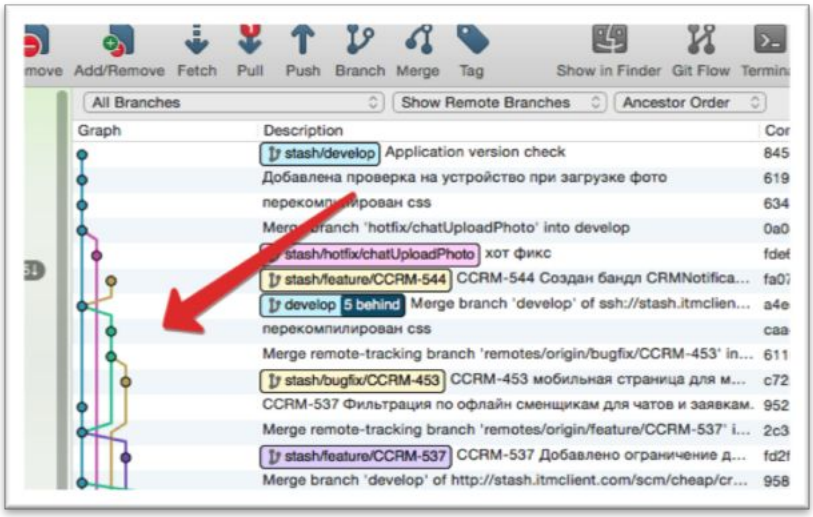
Кнопка «Branch»

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



Опции создания новой ветки

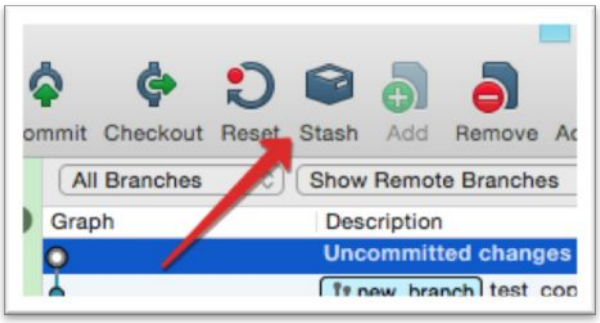
После создания новой ветки и переключения в нее (checkout) все изменения будут отслеживаться и коммититься в ней, а в списке коммитов можно увидеть дерево ветвления.



Пример дерева веток

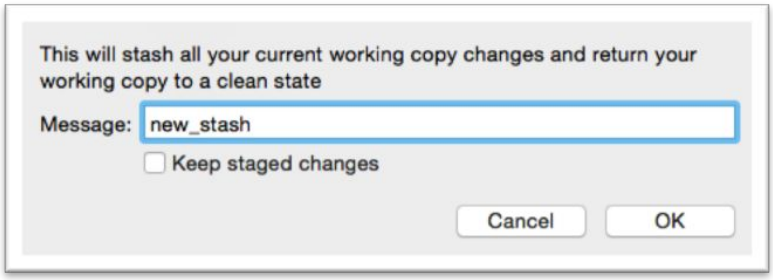
Прятки (stash)

В процессе разработки может возникнуть такая ситуация, когда необходимо переключиться в другую ветку, но текущие изменения проекта не должны быть потеряны, а для коммитить их по нем 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.



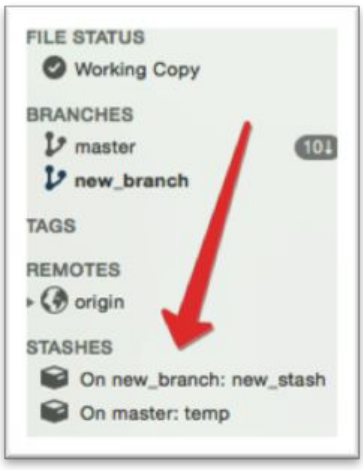
Кнопка “Stash”

После этого в открывшемся оне 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

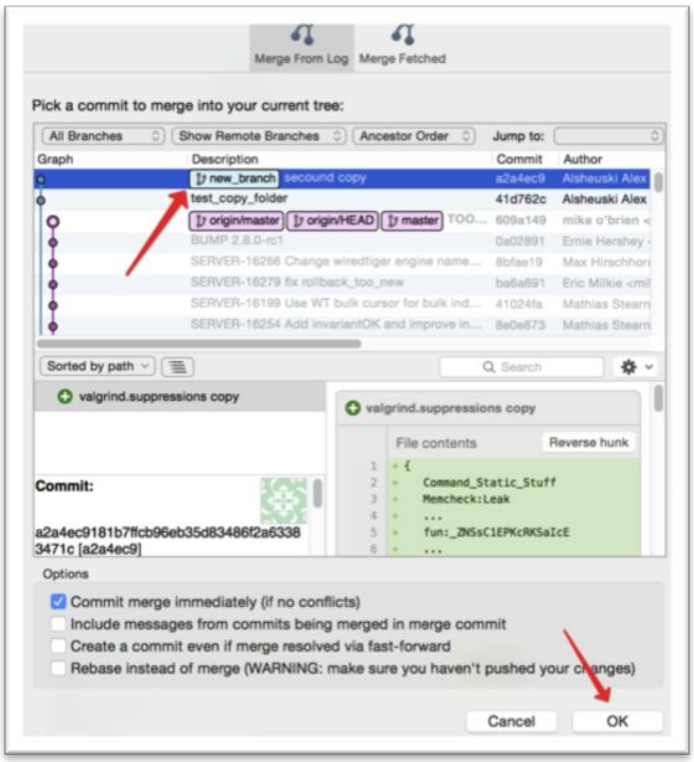
Для получения спрятанных в стэше изменений нужно кликнуть два раза по названию нужного стэша и нажать «OK».

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.

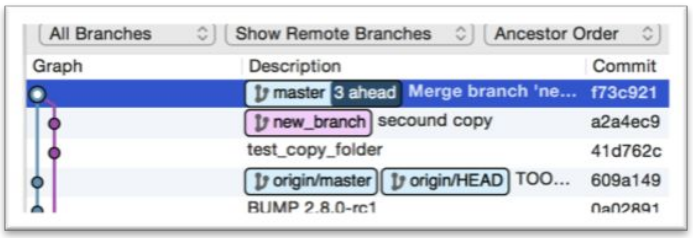
Кнопка “Merge”

Далее выбрать ветку, из которой будут получены изменения (которая будет влита в текущую), оставить опции по умолчанию и нажать “OK”.



Опции мержа веток

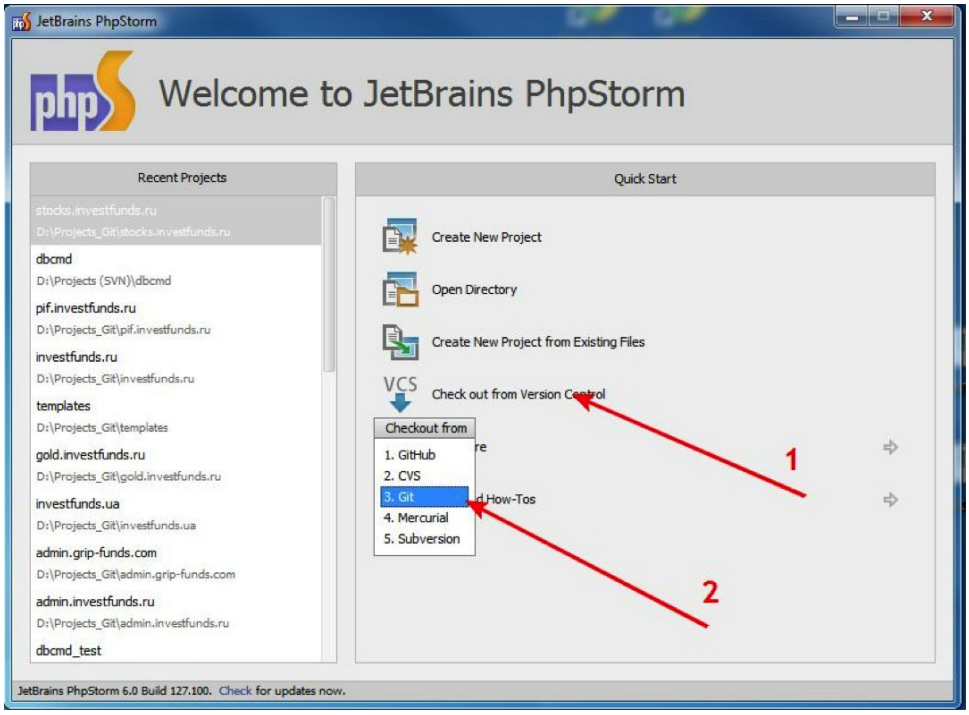
На этом мерж будет закончен.



Обновленное дерево веток после мержа

8. Работа из графического интерфейса PhpStorm

8.1. Получение репозитория

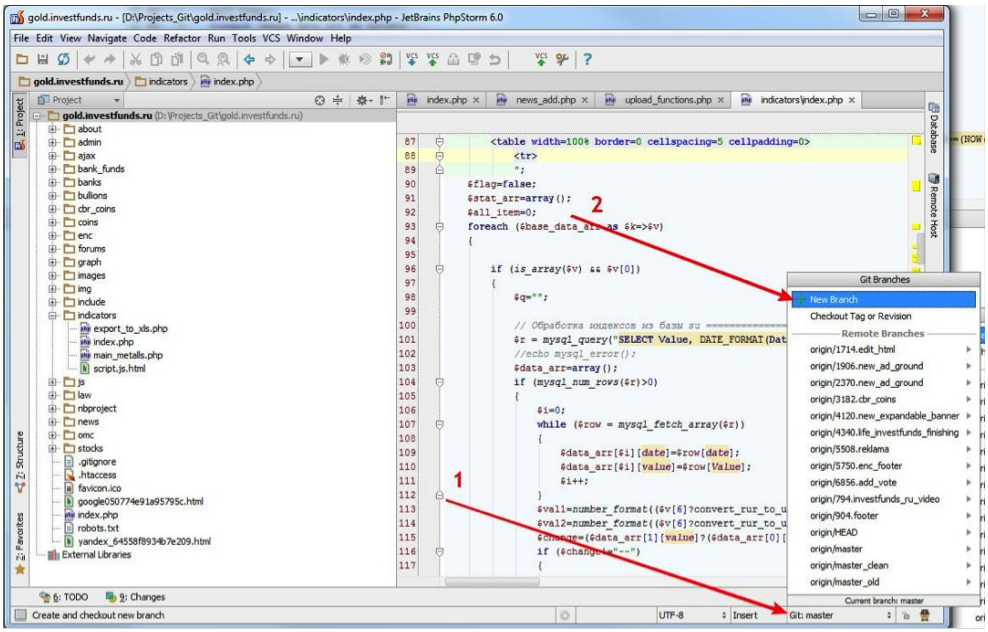


Менеджер проектов phpStorm

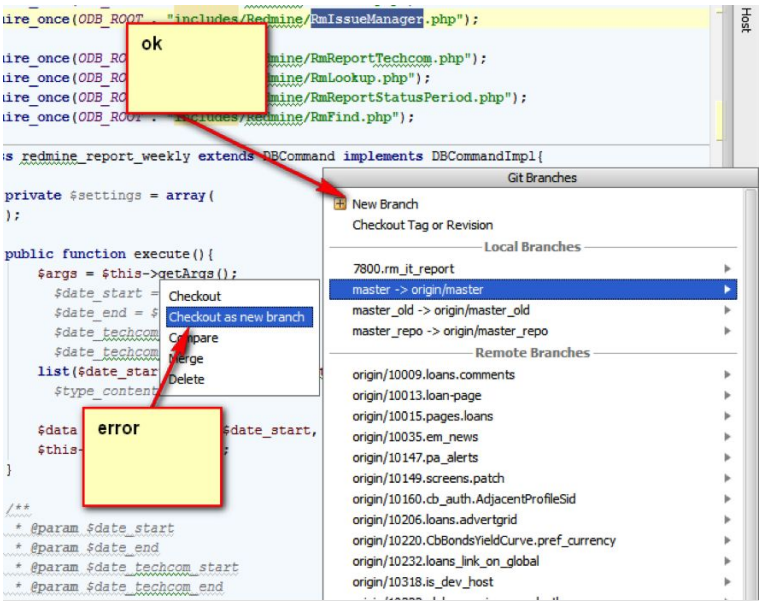
8.2. Стандартные процедуры работы

1. Начало работы над задачей

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

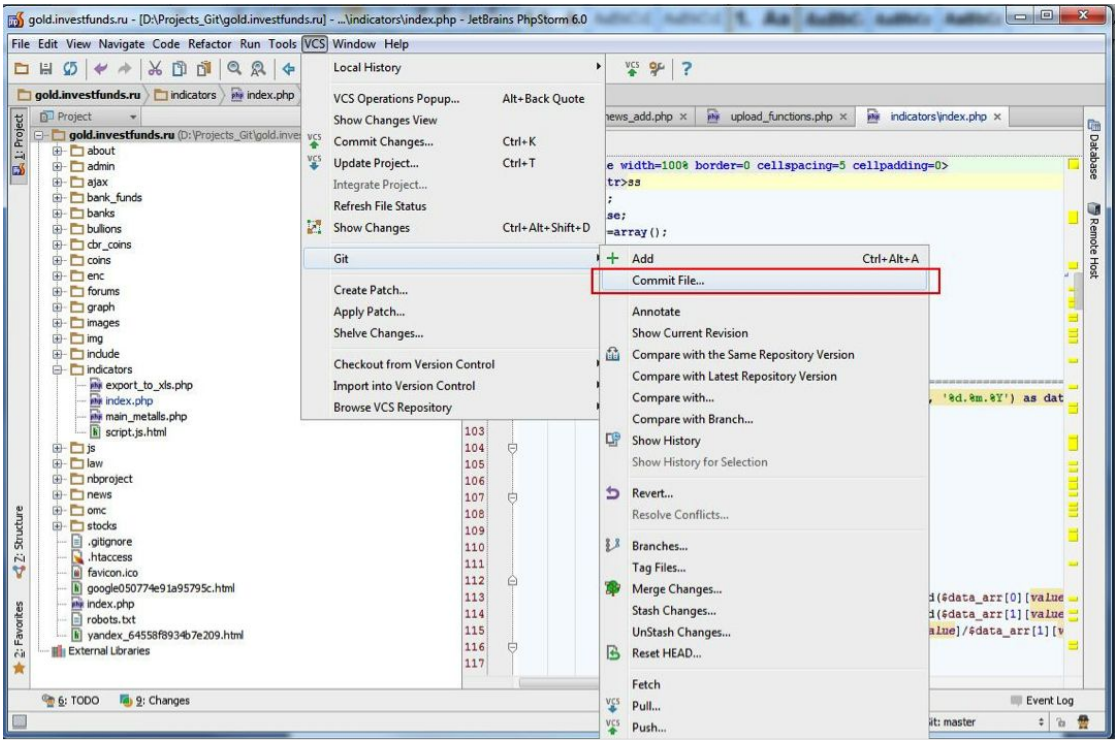


Правильный и Неправильный вариант создания ветки от мастера:

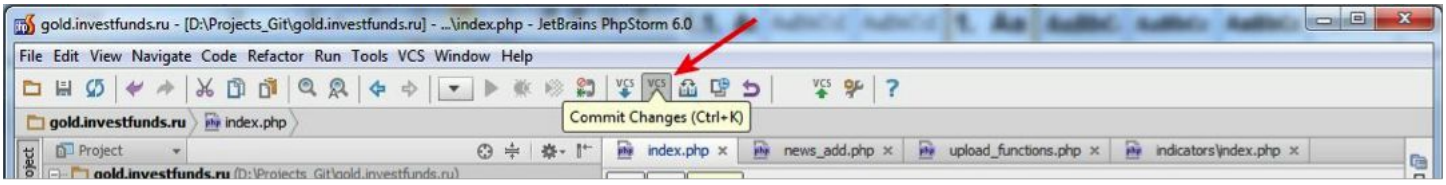


2. Коммит задачи

После изменения в файлах, выбираем пункт меню «Commit File»



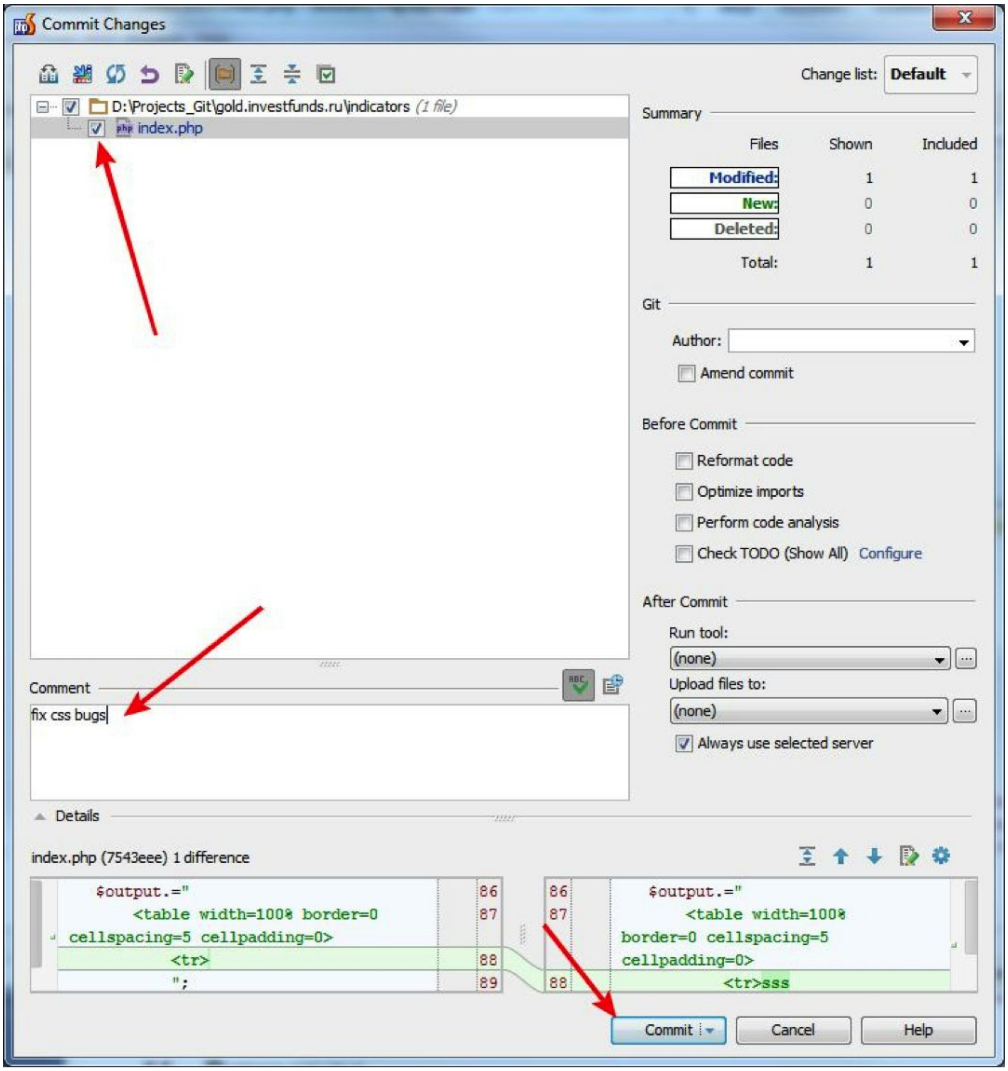
или используем кнопку на панели программы:



В появившемся окне:

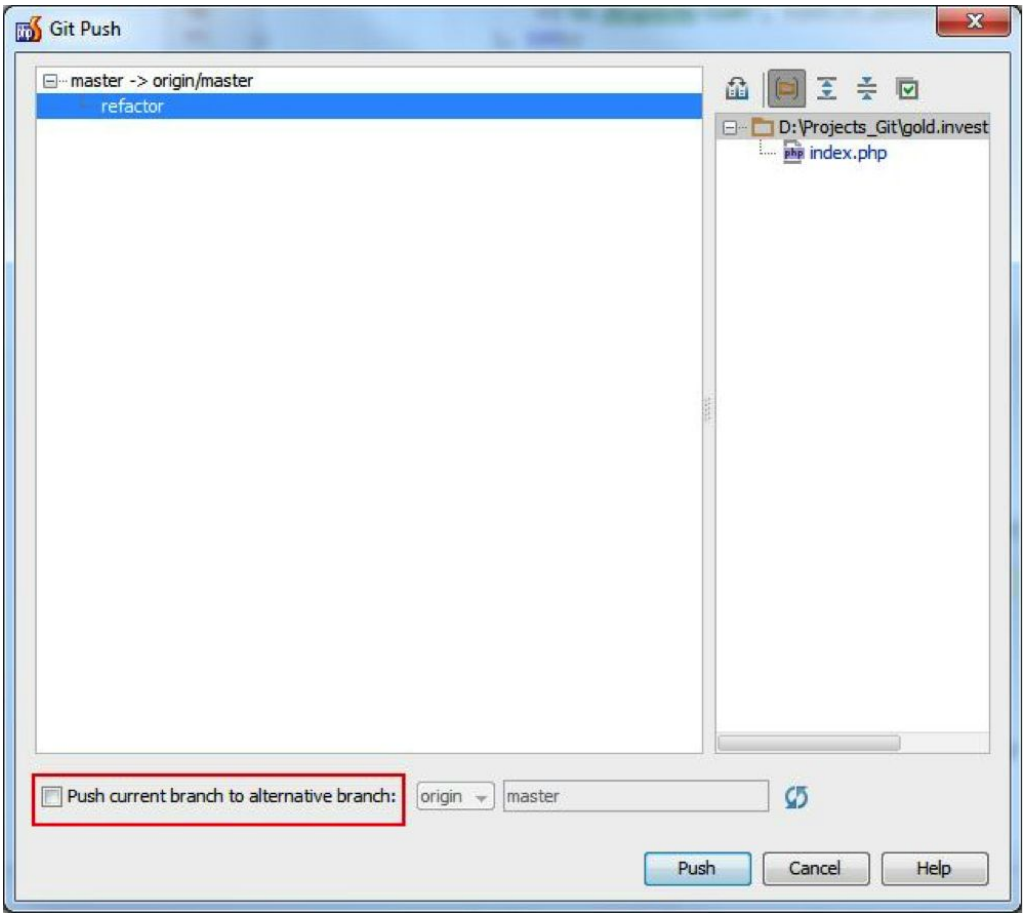
- выбираем файлы для добавления в коммит
- пишем комментарий к коммиту
- в окне можно посмотреть изменения в файлах

Нажимаем “Commit” или “Commit and Push”



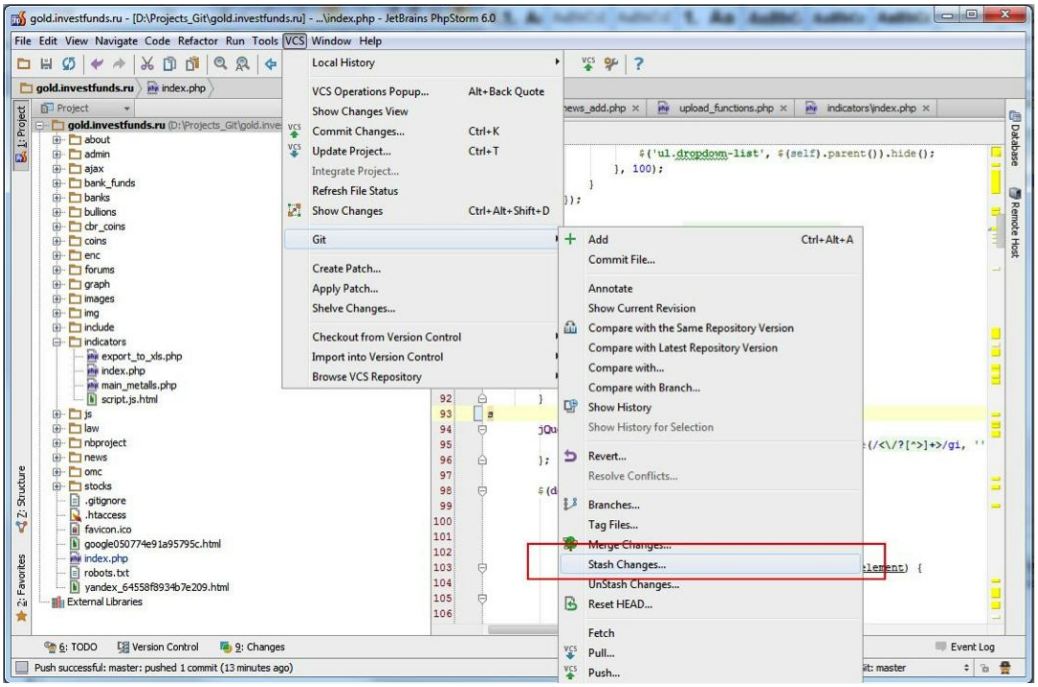
10. Отправка ветки на центральный репозиторий

Если происходит Push в первый раз в только что созданную ветку то нужно поставить галлу “Push current …” для создания этой ветки на удаленном сервере.

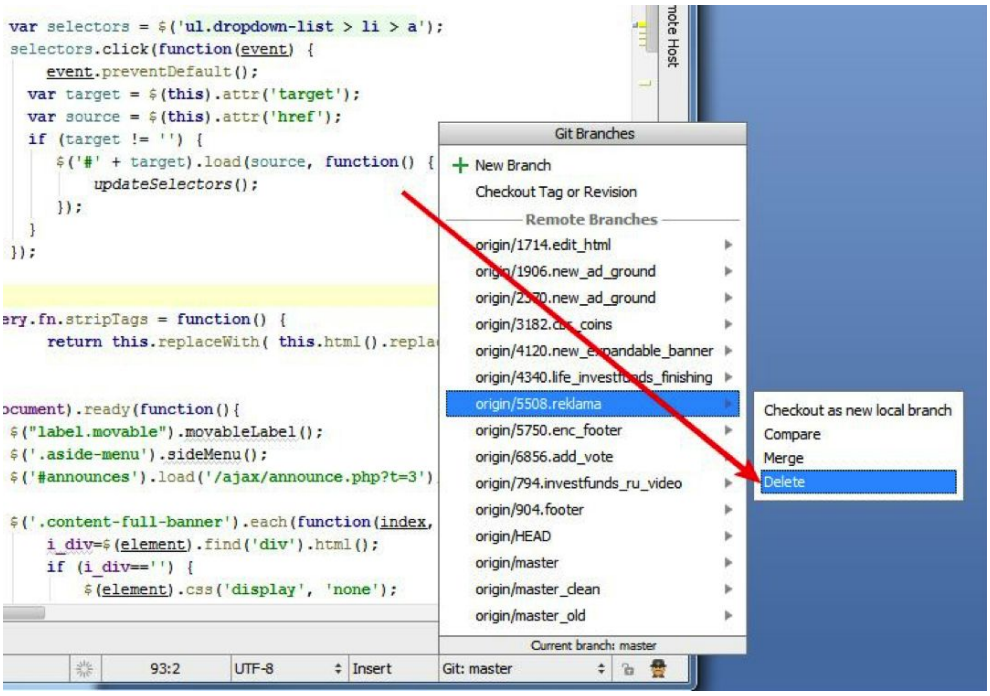


11. Кратковременное сохранение состояния изменений

Для этого нужно воспользоваться пунктом меню “Stash” – команда сохраняющая изменения в отдельное хранилище.

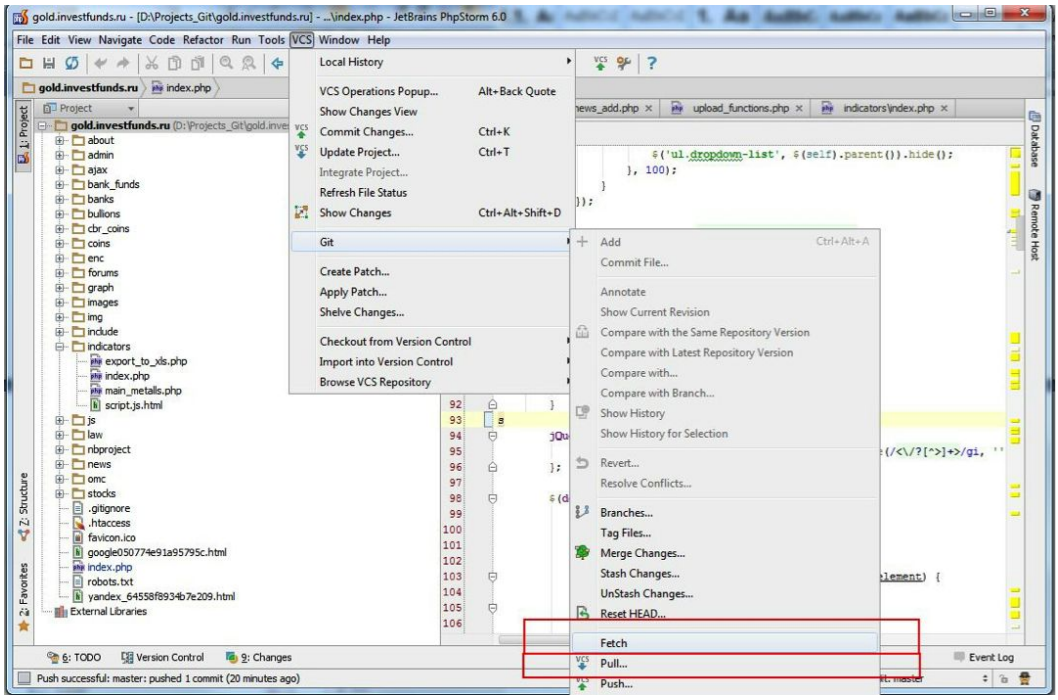


12. И удаляем ветки



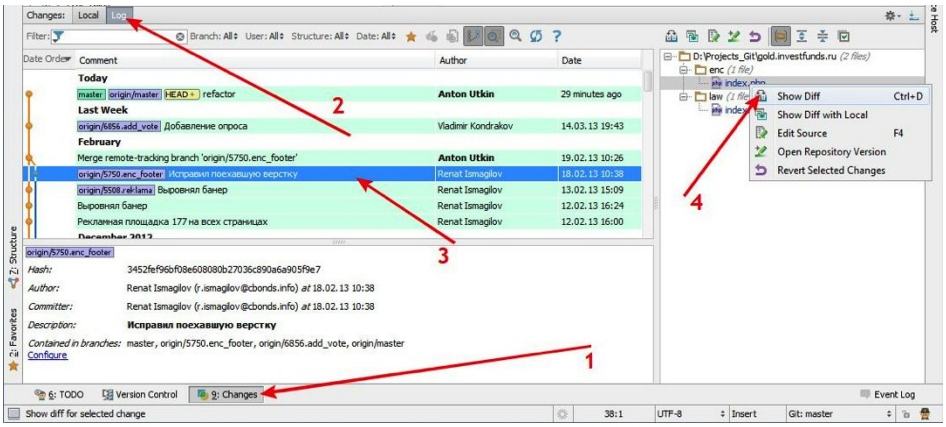
9. Ревью и выкладка задачи

Обновление локального репозитория:



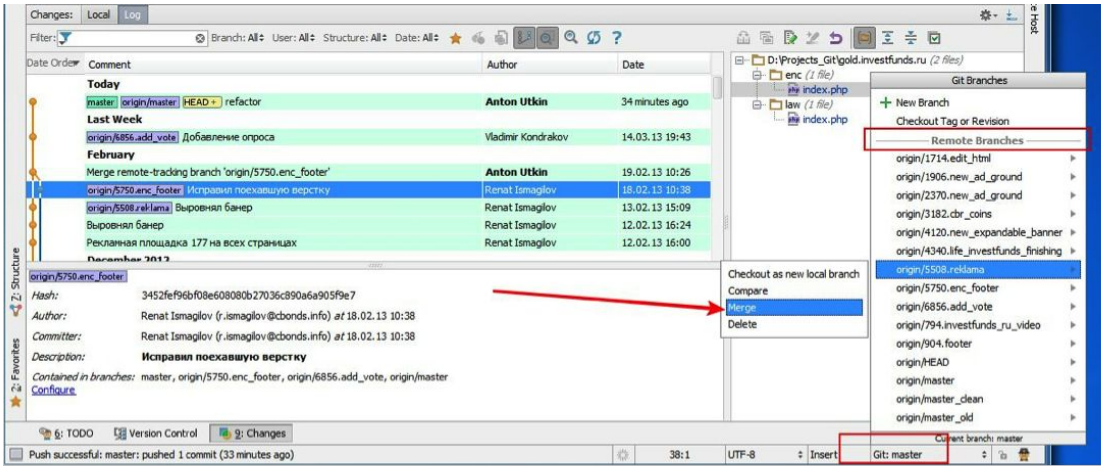
Fetch – обновляет историю, но НЕ обновляет локальные файлы
Pull – обновляет историю и локальные файлы

Просмотр истории веток:



1. Раскрываем вкладку “Changes”
2. Выбираем вкладку “Log”
3. Находим ветку с нужной задачей
4. Просматриваем изменения внесенные в файл в выбранном коммите

Добавление изменений из ветки в ветку master:

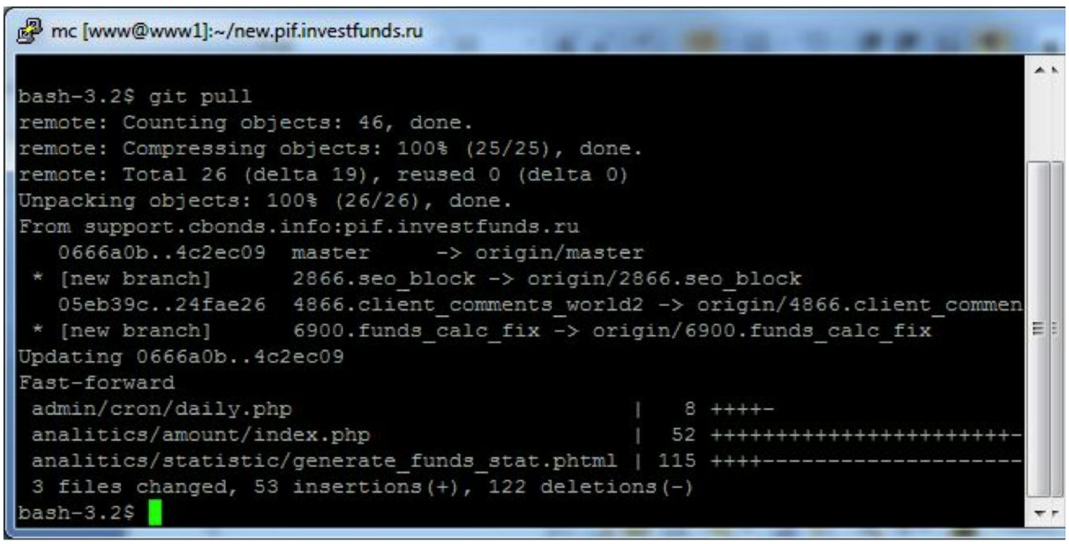


Находясь в локальной ветке master выбираем пункт меню “Merge” у ветки находящейся на удаленном сервере “Remote Branches”

· Отправка изменений в центральный репозиторий

Выбираем пункт меню “Push”

· Обновление ветки на ПРОД сервере



Набираем команду: git pull

Смотрим, что изменения пришли.

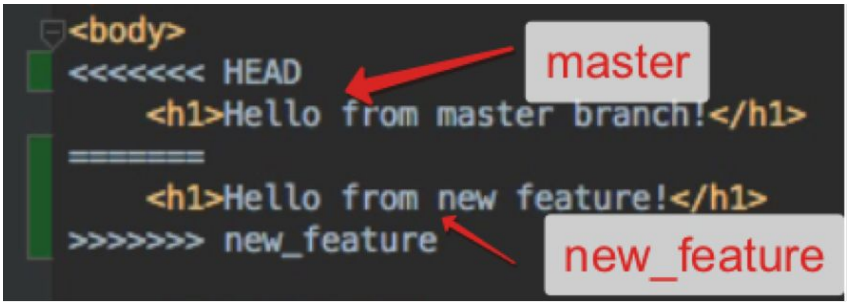
10. Устранение конфликтов при слиянии

При операциях слияния веток (merge) могут возникать конфликты и слияния при этом не будут успешно завершены до исправления данных 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 гит 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 комманду «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 коммандой «git merge --abort». Но в этом случае в проекте ничего не измениться.

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

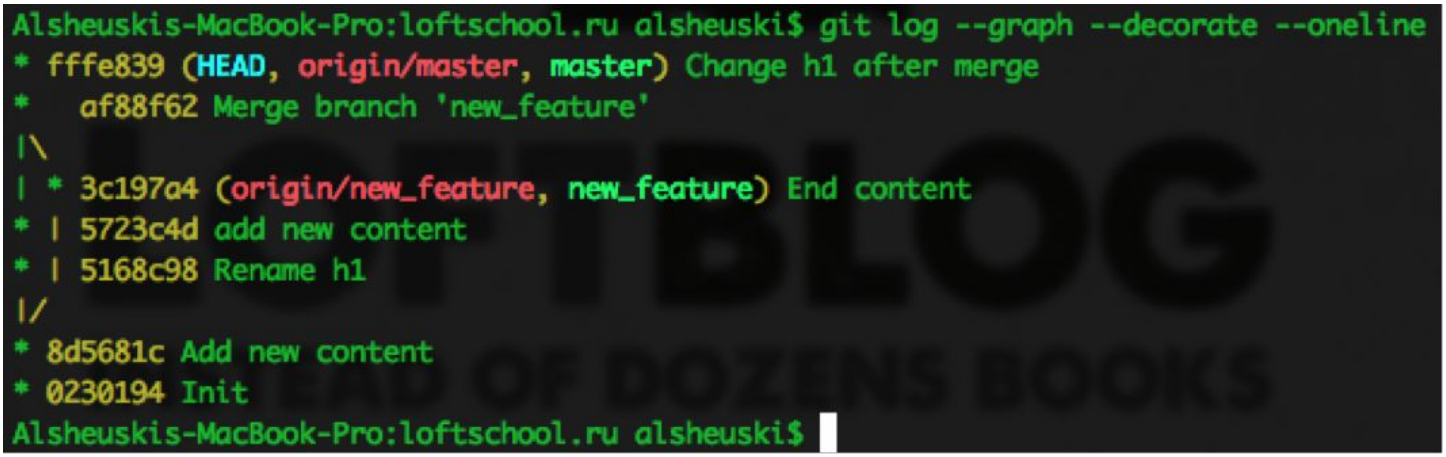


Пример конфликта в файле

Секция “HEAD” – это содержимое текущей ветки назначения. Вторая секция – это содержимое источника, в нашем примере ветки “new_feature”. Далее нужно удалить ненужные данные и све маркеты данного конфликта. В итоге мы должны получить файл, содержащий только нужную нам информацию без каких-либо дополнительных/лишних символов.

Далее нужно выполнить «git add название_файла.html» для того, чтобы показать гиту 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 слиянию будет завершена.

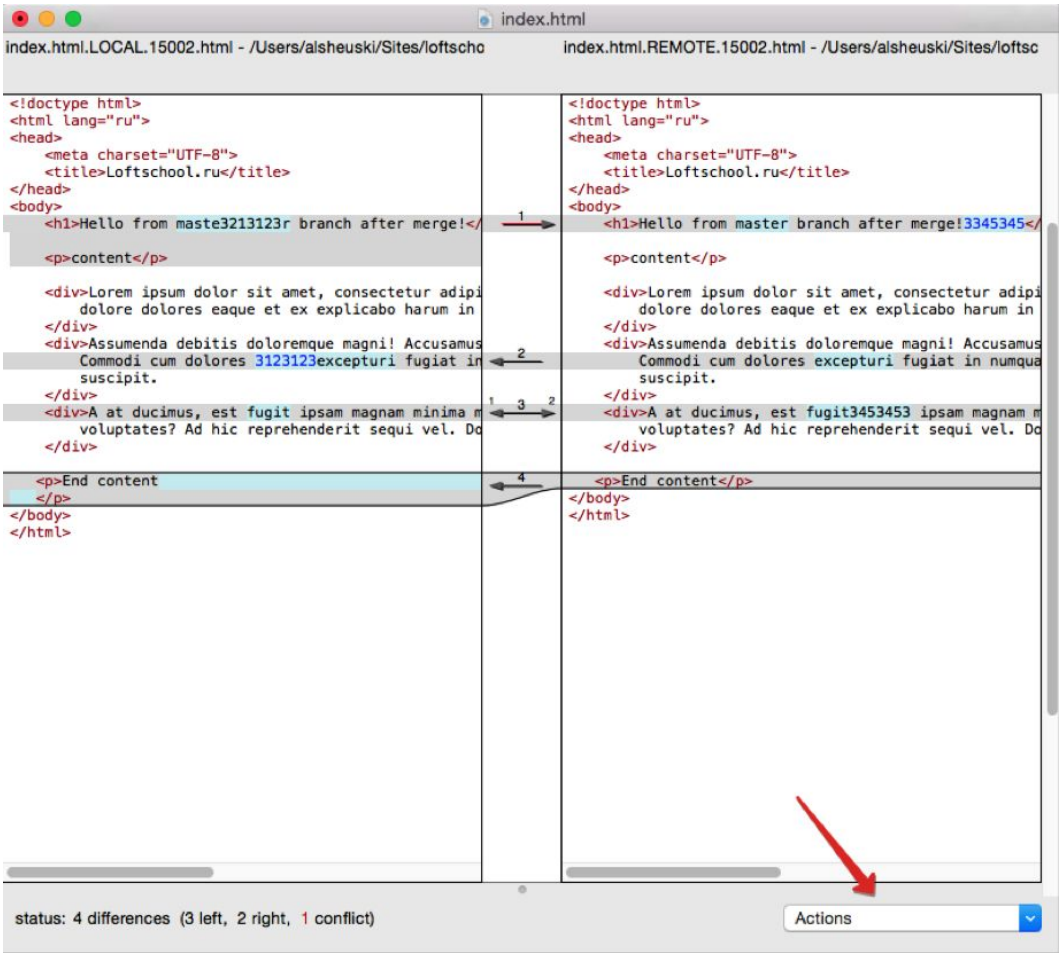
Можно просмотреть наглядно результат слияния и состояние коммитов и веток при этом. Для этого нужно выполнить комманду «git log –graph –decorate --oneline». Результат вывода данной комманды находится на скриншоте ниже.



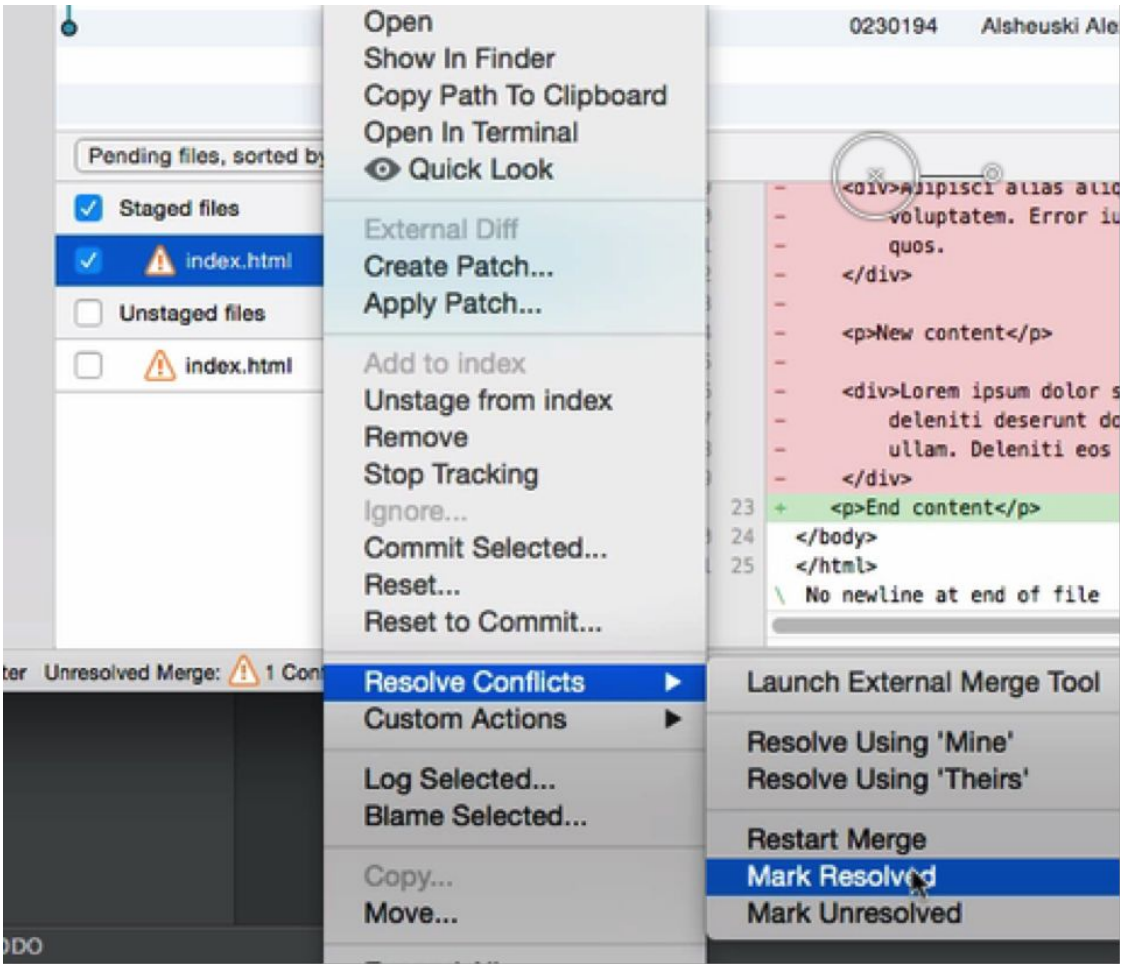
Наглядный вывод коммитов

При разрешении конфликта через программу, например, SourceTree, нам нужно сделать мерж, 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 разрешенным автоматически.



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 завершенным.

11. 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
- Доступно 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, подробнее смотри ниже в примерах,
- 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
# (корнём 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 поддиректории)
# Например /private/index.html
# НЕ БУДУТ проигнорированы файлы в /private/ivan/index.html
/private/*.html
# Игнорирование ВСЕХ html-файлов в КОНКРЕТНОЙ директории ВКЛЮЧАЯ поддиректории
# 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.

12. 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 гитом файлы:
   $ 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 удаленный, необходимо выполнить следующие команды:

   $ git remote add имя_удаленного репозитория url_удаленного_репозитория;

   $ git push -u имя_удаленного репозитория имя_ветки;

6. Для внесения изменей 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

13. Литература

http://git-scm.com/book/ru/
http://habrahabr.ru/post/60347/
http://habrahabr.ru/blogs/Git/104198/