Лабораторная работа №5

Зыкун Е.А.

Продвинутая командная работа с Git

О чём этот модуль

Прошлые модули были посвящены основам работы с Git и GitHub. Git — самая популярная система контроля версий. GitHub — сайт, который позволяет бесплатно размещать свои Git- репозитории. Мы показали, как пользоваться командной строкой, создавать коммиты и ветки, а также коснулись темы совместной работы — но пока в совсем небольших масштабах. В этом модуле вы погрузитесь в детали работы над реальными проектами. Такие проекты создаются командами — большими и не очень. При этом их участники заранее договариваются о «правилах игры»: о структуре веток, об их роли в процессе, о принципах оформления коммитов и так далее.

Что такое fast-forward

В этой теме расскажем о подходах к работе с ветками в команде. А именно: в какой момент создаются ветки, когда они сливаются друг с другом, какие коммиты и в какую ветку попадают. Чтобы разобрать плюсы и минусы реально используемых подходов, сначала объясним, чем плох простой и очевидный подход «все дружно „пушим“ в main по очереди!». Для этого понадобится рассмотреть понятие fast-forward (англ. «перемотка вперёд») и его связь с git push. Начнём с fast-forward: что это такое и как его распознать.

Состояние fast-forward

Две ветки находятся в состоянии fast-forward, если одну из них можно «перемотать» вперёд, и она будет содержать те же коммиты, что и другая. Разберём на примере. Есть две ветки: main и add-docs (англ. «добавить документацию»). В ветке main четыре коммита, от неё создали ветку add-docs и добавили в неё ещё два коммита.

```
$ git branch
* add-docs
main

$ git log --oneline
e08fa2a (HEAD -> add-docs) New docs 2
fd588b2 New docs 1
997d9ce (main) Commit 4
0313e8e Commit 3
5848aba Commit 2
```

```
04923d7 Commit 1
```

Можно ли отключить fast-forward

Fast-forward слияние веток можно отключить флагом --no-ff. Например: git merge --no-ff add-docs. Также его можно отключить «навсегда» (до тех пор, пока вы не вернёте настройку «как было») с помощью настройки merge.ff: git config [--global] merge.ff false. Если отключить слияние в режиме fast-forward, вместо «перемотки» ветки Git создаст в ней коммит слияния (англ. merge commit) — в обиходе его называют merge-коммит или мёрж-коммит.

Non-fast-forward

В прошлом уроке мы поближе познакомились с состоянием fast-forward. Теперь узнаем, что происходит при объединении двух веток, коммиты которых нельзя выстроить в одну цепочку. Состояние non-fast-forward Вернёмся к примеру с ветками main и add-docs и представим такую ситуацию: истории двух веток «разошлись». Это значит, что их коммиты уже нельзя выстроить в одну линию. Например, после «отделения» add-docs в ветку main добавили новый коммит Commit 5.

```
# команде git log можно указать несколько веток,
# и тогда она выведет их все
$ git log --graph --oneline main add-docs
* 15d3f04 (HEAD -> main) Commit 5
| * 8de42eb (add-docs) New docs 2
| * 4d3c346 New docs 1
|/
* 73def1e Commit 4
* 9c30ab3 Commit 3
* 83cc5ec Commit 2
* 8e87fb2 Commit 1
```

Когда Git проверяет ветки на состояние fast-forward, он не «заглядывает» в файлы и не пытается угадать, будет ли конфликт на самом деле. Для Git важно только, что конфликт теоретически возможен (или, наоборот, никак не возможен). При слиянии не-fast-forward веток Git создаёт коммит слияния.

```
# находимся в ветке main
# --no-edit избавляет от необходимости
# вводить сообщение для merge-коммита
$ git merge --no-edit add-docs
Merge made by the 'ort' strategy.
docs.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 docs.txt
# коммит слияния: 34f5f8f
$ git log --graph --oneline
* 34f5f8f (HEAD -> main) Merge branch 'add-docs'
|\
| * 8de42eb (add-docs) New docs 2
| * 4d3c346 New docs 1
* | 15d3f04 Commit 5
```

```
|/
* 73def1e Commit 4
* 9c30ab3 Commit 3
* 83cc5ec Commit 2
* 8e87fb2 Commit 1
```

Pull request и code review

В прошлом уроке мы рассказали о подходе Feature Branch Workflow, в котором вся новая функциональность разрабатывается в feature-ветках. Как только «фича» готова, ветка вливается в main. Но перед тем, как влиться в main, ветка проходит разные проверки, в том числе изменения просматривают другие участники команды. О таких проверках и пойдёт речь в этой теме.

Pull (или merge) request

В большинстве команд новые функциональности и исправления попадают в main через запрос на слияние (англ. pull request, или merge request). Его так и называют — пул-реквест или мёрж-реквест. В переводе с языка Git это значит: «Вот моя ветка, хочу «вмёржить» её в main».

Так как правильнее — пул-реквест или мёрж-реквест?

Названия «пул-реквест» и «мёрж-реквест» означают одно и то же. GitHub и BitBucket используют термин «пул-реквест», а GitLab — «мёрж-реквест». В примерах мы показывали работу с GitHub, поэтому будем называть запрос на слияние пул-реквестом, или PR. «Запросы на слияние» — это та часть процесса (workflow), которая может сильно отличаться от команды к команде. Например, в некоторых проектах строго следят за качеством изменений и тщательно проверяют их до того, как влить в main. А некоторые проще относятся к тому, что попадает в основную ветку.

Работа с PR: практика Теоретический урок позади, теперь покажем, как создавать пул-реквесты, как оставлять к ним комментарии и как «апрувить» и «мёржить» PR в интерфейсе GitHub. Представьте, что вы закончили разработку новой функциональности и готовы влить эти изменения в основную ветку. Но сначала вам нужно создать запрос на слияние, попросить коллег оценить вашу работу и, возможно, внести какие-то правки по их комментариям. Вместе с нами вы проследите типичный цикл жизни PR. В примерах будем использовать этот репозиторий: создадим в нём пул-реквест, пройдём ревью и «вмёржим» изменения в main.

Создаём пул-реквест

Участник проекта, пользователь b490 делает следующее:

1. Клонирует репозиторий.
2. Создаёт ветку feature-goose-poem.
3. Создаёт пул-реквест. Для этого используется ссылка, которую GitHub выводит через Git в консоль после выполнения команды git push.

Проводим ревью

Теперь покажем, как другой участник проекта проводит ревью. Ревьюер:

1. Заходит в пул-реквест и просматривает изменения.
2. Добавляет комментарии к тому, что кажется ему важным.
3. Выносит вердикт Request changes (англ. «нужны исправления»).

Вносим исправления

Автор пул-реквеста видит, что ему предложили внести исправления, и приступает к работе. Гуси становятся серым и белым — как положено. Пробелы в конце строк исправляют маркдаун- разметку. Затем правки «коммитятся» и «пушатся» в ту же ветку — теперь в ней три коммита.

«Апрувим» пул-реквест

Если ревьюера всё устраивает, он выносит вердикт Approve (англ. «одобрить»). Отметим, что после одобрения кнопка Merge pull request появилась как у автора пул-реквеста, так и у ревьюера, потому что они оба участники проекта, или коллабораторы.

Вливаем в main

Когда ревью пройдено, автор пул-реквеста может нажать кнопку Merge pull request, а затем Confirm merge (англ. «подтвердить объединение»). После нажатия этих кнопок GitHub вольёт ветку feature-goose-poem в main. GitHub вливает ваши изменения в ветку main в origin, а локальная ветка остаётся как была. После «мёржа» PR рекомендуем обновить локальную main: git checkout main && git pull.

Отличия для неучастников

Здесь показан типичный цикл жизни PR для разработки «в компаниях». Разве что обычно в таком цикле нет гусей. В open-source проектах автор пул-реквеста, скорее всего, не будет участником (collaborator) проекта.

Работа с PR: soft skills

Навыки специалиста принято делить на две группы: hard skills (англ. «жёсткие навыки») и soft skills (англ. «мягкие навыки», «гибкие навыки»).

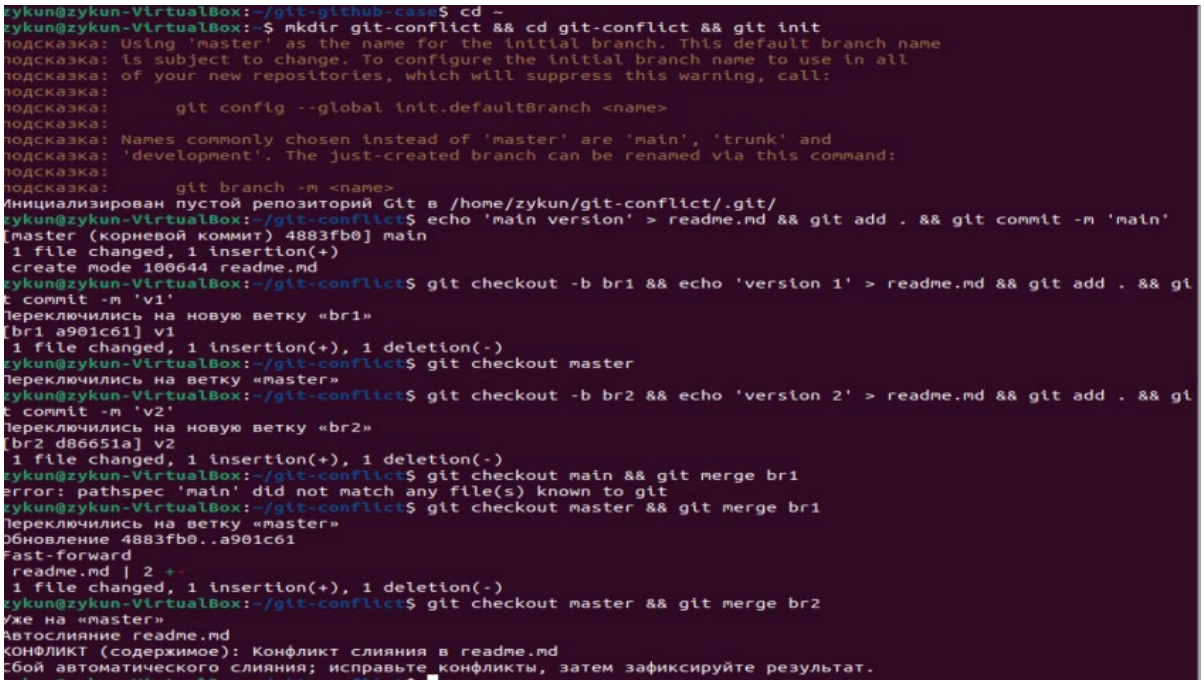
- Hard skills, или хардскилы, — это владение инструментами, причём неважно какими. Умение забивать гвозди молотком, знание команд Git, умение играть на пианино — всё это примеры хардскилов. Именно этому типу навыков мы уделяли больше внимания в нашем курсе.
- Soft skills, или софтскилы, — это социально-психологические навыки. Например: умение договариваться, умение внятно донести свои мысли или понять идеи собеседника, умение мягко преподнести критику результатов работы.

О софтскилах, а именно о том, как корректно пройти или провести ревью, пойдёт речь в этом уроке.

Разрешение конфликта вручную и через vimdiff

В этой теме поговорим о конфликтах слияния. Они могут случиться при слиянии веток (merge), если один и тот же файл был изменён в обеих ветках. Git не может сам решить, какая версия файла «правильная», а какая нет. Такие конфликты нередко возникают в командной работе. В этом уроке покажем, как разрешать их вручную и через консоль.

Подготовим новый Git-репозиторий.



Добавил в основную ветку (main или master) ветку br1. Коммиты этих веток можно выстроить в одну линию, поэтому слияние будет выполнено в режиме fast-forward. Теперь в файле readme.md содержится текст version 1, а текущее состояние веток

будет такое. Настало время создать конфликт! Для этого убедимся, что находитесь в основной ветке, а затем выполним git merge.

Поздравляем: конфликт получен!

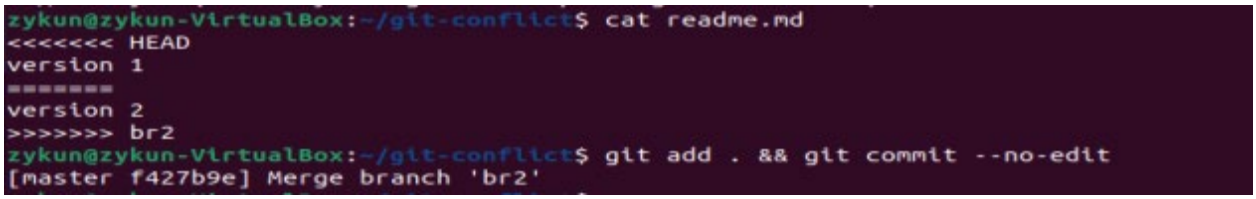
Разрешаем конфликт вручную

Когда Git выявляет конфликт, он помечает проблемные файлы и останавливает процесс слияния.

При попытке объединить ветки или применить изменения из удалённого репозитория Git добавит в файлы специальные маркеры конфликта. Убедитесь в этом. Откройте файл readme.md в графическом интерфейсе или выполните cat readme.md.

Git разметил файл. Получившиеся секции содержат изменения из каждой ветки:

- Текст между <<<<<<< HEAD и ======= указывает на изменения, которые находятся в HEAD — в данном случае это ветка main. Здесь окажутся только те строки, в которых есть конфликт.
- Текст между ======= и >>>>>>> br2 показывает на изменения, которые находятся в ветке br2. Чтобы разрешить конфликт вручную, нужно открыть файл и выбрать, какие изменения оставить, а какие отбросить. Для этого следует удалить все маркеры и ненужные изменения и оставить нужные. После разрешения конфликта файлы будут отмечены как решённые. Можно продолжить процесс слияния или выполнить коммит изменений. Допустим, нужно оставить текст version 2. Откройте файл readme.md и удалите все маркеры конфликтов, а также строку version 1.



Готово! Мы разрешили конфликт вручную и создали коммит слияния. Теперь в файле readme.md содержится текст version 2. Дерево коммитов будет выглядеть так.

Разрешение конфликта через Visual Studio Code

Эти способы помогают лучше понять, как взаимодействовать с конфликтами, но всё же в работе над реальными проектами чаще используются другие. Например, большинство разработчиков предпочитают более удобные интерфейсы, чем у vimdiff. Такие интерфейсы предлагают почти все современные среды разработки (англ. Integrated Development Environment, или IDE, «интегрированная среда

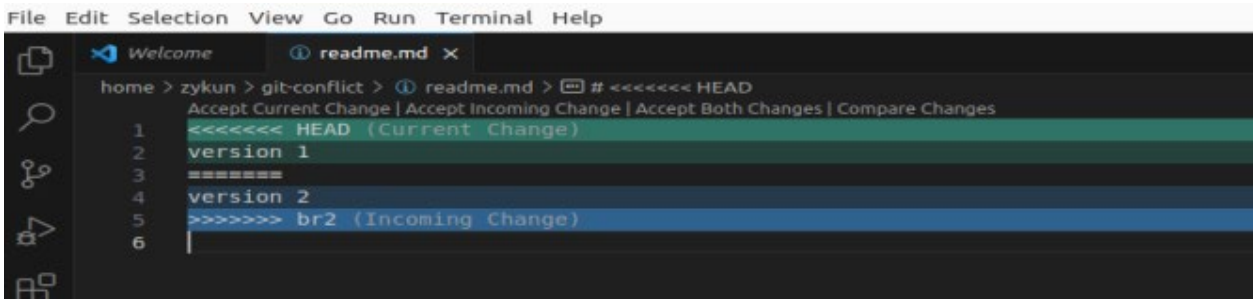
разработки») — программы, в которых собраны инструменты для быстрой и эффективной работы над проектом. В этом уроке покажем, как выглядят конфликты в среде разработки Visual Studio Code (или сокращённо VS code).

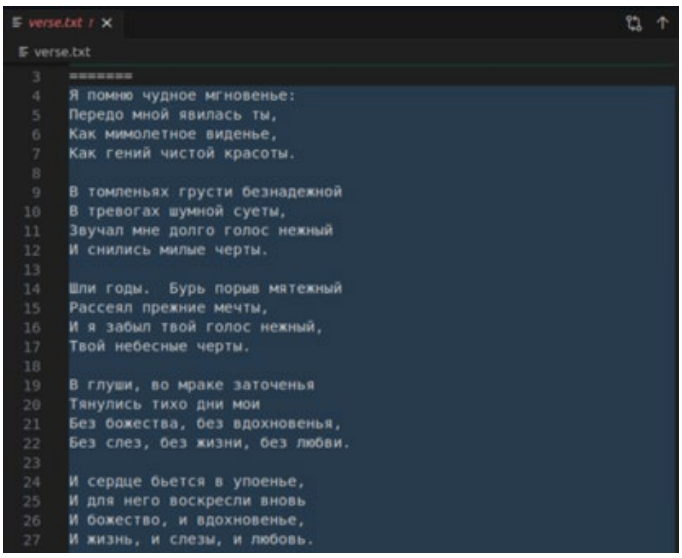
Разрешаем конфликт с помощью VS Code

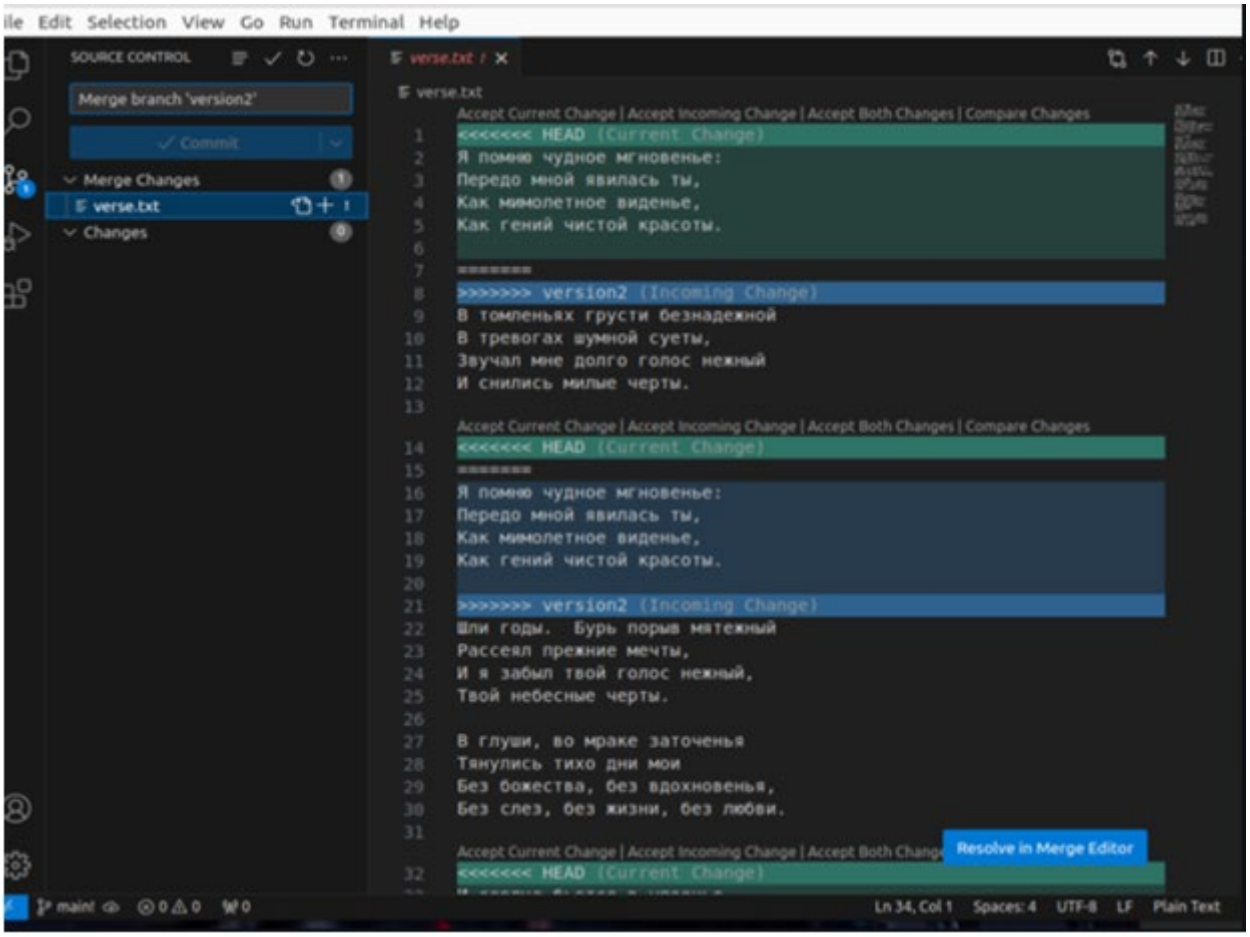
Разрешаем конфликт с помощью VS Code Разберём, какие инструменты предоставляет VS Code для разрешения конфликта. Но сперва снова сымитируйте его. Удалим текущую папку git-conflict, которую создали в прошлом уроке. Затем пересоздаем её с помощью следующей команды. Мы в одном шаге от конфликта! Но перед тем как создать его, откроем папку git-conflict в VS Code. Сделать это можно через меню File → Open... (для macOS) или Open Folder... (для Windows). Откроется окно редактора. Пришло время для конфликта! Выполним git merge br2 в консоли. Теперь в окне редактора появились маркеры конфликта. Зелёным цветом подсвечивается текущая версия, а синим — новые изменения. Мы можем разрешить конфликт прямо в файле вручную. Нажмем на кнопку Resolve in Merge Editor (англ. «разрешить в редакторе слияний») в правом нижнем углу экрана. Мы увидим окно разрешения конфликтов. Оно состоит из трёх частей: в левой части содержатся новые изменения, в правой — текущие, а в нижней — результат. Используем вспомогательные кнопки Accept Incoming (англ. «принять входящие»), Ignore (англ. «игнорировать») и Accept Current (англ. «принять текущие»), чтобы быстро добавить изменения в результат. Щёлкнем на Accept Incoming в левой верхней части экрана. Все конфликты разрешены, поэтому теперь можно нажать на Complete Merge (англ. «завершить слияние»).

Задание для самостоятельной работы

1. Открываем этот архив. pushkin.zip
2. В архиве мы найдёте папку pushkin, в которой проинициализирован Git-репозиторий с тремя ветками: main, version1 и version2. В ветках находятся разные версии стихотворения «К ***» («Я помню чудное мгновенье...»), но в обеих версиях перепутаны местами строфы.
3. Соберем правильную версию стихотворения в ветке main. Для этого сначала перейдем в main, сделаем git merge version1 из ветки main, а затем git merge version2.




verse.txt
=======
Я помню чудное мгновенье:
Передо мной явилась ты,
Как мимолетное виденье,
Как гений чистой красоты.
В томленьях грусти безнадежной
В тревогах шумной суеты,
Звучал мне долго голос нежный
И снились милые черты.
Шли годы. Бурь порыв мятежный
Рассеял прежние мечты,
И я забыл твой голос нежный,
Твой небесные черты.
В глуши, во мраке заточенья
Тянулись тихо дни мои
Без божества, без вдохновенья,
Без слез, без жизни, без любви.
И сердце бьется в упоенье,
И для него воскресли вновь
И божество, и вдохновенье,
И жизнь, и слезы, и любовь.


ile Edit Selection View Go Run Terminal Help
SOURCE CONTROL
Merge branch 'version2'
Commit
Merge Changes 1
verse.txt
Changes 0
verse.txt
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
Я помню чудное мгновенье:
Передо мной явилась ты,
Как мимолетное виденье,
Как гений чистой красоты.
=======
>>>>>>> version2 (Incoming Change)
В томленьях грусти безнадежной
В тревогах шумной суеты,
Звучал мне долго голос нежный
И снились милые черты.
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
=======
Я помню чудное мгновенье:
Передо мной явилась ты,
Как мимолетное виденье,
Как гений чистой красоты.
>>>>>>> version2 (Incoming Change)
Шли годы. Бурь порыв мятежный
Рассеял прежние мечты,
И я забыл твой голос нежный,
Твой небесные черты.
В глуши, во мраке заточенья
Тянулись тихо дни мои
Без божества, без вдохновенья,
Без слез, без жизни, без любви.
Resolve in Merge Editor
<<<<<<< HEAD (Current Change)
main
Ln 34, Col 1 Spaces: 4 UTF-8 LF Plain Text