ASTON

Интенсив-курс по JS

astondevs.ru

Программа курса по JS

1. Git, GitHub, Команды git, Git Flow,
2. HTTP/HTTPS и принципы программирования
3. Типы данных и переменные. Приведение типов, сравнение
4. Объекты, Функции, Сложность алгоритмов (Big O), Структуры данных
5. Замыкания, Контекст выполнения. Контекст вызова, Привязка контекста
6. ООП в JavaScript, прототипы, Классы ES6
7. Event Loop, асинхронность в JavaScript, Promise,
8. Понятие окружения, DOM, BOM. Навигация по DOM. Поиск элементов
9. Изменение DOM. Атрибуты, классы и стили элементов, Браузерные события. Стадии событий. Обработка событий

Занятие 1. Основы Git

1. Git
2. GitHub
3. Основные команды git
4. Git Flow

Git

Git — это система контроля версий, которая позволяет удобно организовать рабочий процесс для команды разработчиков. Достигается это тем, что с помощью Git можно: сохранять разные состояния проекта и легко перемещаться между ними; делать новые ответвления от текущего состояния проекта, дабы “в изоляции” разрабатывать новые фичи; “подтягивать” новые изменения из других веток и довольно просто решать возникшие конфликты. При всем этом обилии полезных возможностей сам Git остаётся довольно маленькой и быстрой в работе утилитой. Большая скорость обусловлена тем, что Git ставится и работает локально на вашем ПК.

Историю изменений проекта Git хранит в виде набора “замороженных” состояний, к каждому из которых вы сможете в последствии вернуться. На каждом этапе сохранения нового состояния проекта в Git, система запоминает, как выглядит каждый файл в этот момент и сохраняет ссылку на это состояние. В конечном итоге получается длинная история из изменений, каждое из которых накладывает какой-то новый кусок кода на ваш проект.

GitHub

Насколько бы «крут» и хорош не был Git, до тех пор, пока он хранит историю вашего проекта локально, он всё ещё не решает никаких проблем командной разработки.

GitHub — это онлайн сервис для хранения проектов. Также он включает в себя и перечень возможностей систем контроля версий. Хранятся проекты на GitHub в виде репозиториев (сокращенно «репа» или «реп», если говорить в множественном числе). Каждый репозиторий имеет свой уникальный URL и хранит в себе все файлы, ветки и полную историю вашего проекта. Именно отсюда, как правило, и начинается разработка новых проектов. Первым делом всегда создается репозиторий, а все разработчики «стягивают» его себе и продолжают работать с ним локально, периодически внося изменения обратно в удаленный репозиторий.

Распределенные системы контроля версий, к которым относится Git, предоставляют одну очень важную возможность: полная «выкачка» абсолютно всей информации из репозитория к себе на ПК. В итоге каждый разработчик имеет копию всех данных проекта и в случае утери (отказа или «слета» удаленного сервера) можно будет с легкостью в полном объёме восстановить всю базу данных проекта.

Git основные команды для работы с локальным репозиторием

- git init — инициализация нового репозитория;
- git add — поэтапное добавление изменений;
- git commit — регистрация некоторого перечня изменений с описанием и назначением уникального ID этим изменениям;
- git status — отслеживание текущего состояния репозитория;
- git config — запись и чтение конфигурации Git;
- git branch — отображение текущей ветки, создание новых веток, удаление веток;
- git checkout — переключение на другие ветки;
- git merge — слияние веток, т.е. соединение изменений двух веток в одну.

git init / git add / commit

Git init

Данная команда «превращает» директорию на вашем ПК в пустой репозиторий Git. Это первый шаг на пути создания репозитория. После выполнения данной команды можно приступать к добавлению и регистрации (созданию коммитов) изменений.

Git add

С помощью данной команды можно добавлять измененные (добавленные) файлы в некоего рода промежуточный источник данных Git. Это необходимо для дальнейшей регистрации (создание коммитов на их базе) этих изменений. Существует несколько способов добавления файлов: добавление файлов каждого по отдельности, добавление директории файлов или добавление всех изменений сразу.

- git add «file» - Проиндексировать все изменения в файле
- git add . – Добавить все измененные файлы

git init / git add / commit

Git commit

Данная команда позволяет записать/зарегистрировать изменения, внесенные в файлы локального репозитория. При этом новый набор изменений (в дальнейшем коммит) получает свой уникальный идентификатор, по которому его можно найти.

Хорошей практикой считается добавление исчерпывающего описания в сообщении нового коммита. Данное описание должно приводить объяснения к внесённым изменениям. В дальнейшем хорошие сообщения коммитов очень сильно упрощают поиск по истории изменений проекта.

- git commit –m ”Сообщение”

git status / config / branch

Git status

Данная команда сообщает вам о текущем состоянии репозитория.
Вызов команды git status отобразит ветку, на которой вы находитесь в данный момент. Кроме этого, если у вас есть файлы на стадии добавления, т.е. ещё не записанные в коммит, информация об этом также будет отображена. Если же никаких изменений готовых к регистрации нет, данная команда выведет сообщение “nothing to commit, working tree clean”.

Git config

Git позволяет задавать нам множество самых разных настроек, а делается это с помощью команды git config. Самыми важными пунктами конфигурации являются user.name и user.email. Данные значения представляют собой информацию об авторе коммитов на локальном устройстве.
У команды git config есть флаг --global, который позволяет нам задать значение конфигурации для всех репозиториев сразу. Без данного флага настройки будут влиять лишь на текущий репозиторий.

Git branch

Данная команда нужна для определения текущей ветки, а также для создания и удаления веток.

- git branch <ветка> - Создание новой ветки с именем ＜ветка＞. Эта команда не выполняет переключение на эту новую ветку.
- git branch –d <ветка> - удаление ветки

git checkout / merge

git checkout

Данная команда нужна для смены рабочей (текущей) ветки. Используйте git checkout для переключения на новую ветку.
Также с помощью данной команды можно создавать новые ветки. В таком случае после вызова git checkout -b new-branch-name новая ветка будет создана и сразу же станет текущей.

git merge

Данная команда производит слияние изменений из двух веток. Как результат вызова git merge создаётся новый коммит, в котором находятся изменения текущей и целевой веток.
Частая ситуация: последние изменения из ветки dev (или develop) сливаются с изменениями на вашей ветке (feature). Это делается для того, чтобы получить доступ к новым возможностям из ветки dev или для разрешения конфликтов.

Git основные команды для работы с удаленным репозиторием

- git remote — создание связи между локальным и удалённым репозиториями;
- git clone — создание локальной копии существующего удалённого репозитория;
- git fetch — получение данных об изменениях в ветке;
- git pull — получение самой последней версии репозитория;
- git push — отправка локальных изменений в виде перечня коммитов на удалённый репозиторий.

Git remote

Данная команда нужна для соединения локального репозитория с удалённым.
При этом удалённый репозиторий может иметь какое-то название, чтобы не пришлось запоминать или хранить его полный URL-адрес.

Git clone

Данная команда нужна для создания локальной копии существующего удалённого репозитория. После вызова команды git clone, git создаст новую директорию с именем равным названию удалённого репозитория и всем его содержимым (файлы, ветки, история изменений).

Git fetch

Данная команда нужна для получения самых свежих данных с удалённого репозитория. Это могут быть новые ветки, коммиты и т.д. Однако важно понимать, что данная команда просто получает эти данные, но ничего с ними не делает.

Git pull

Данная команда по сути совмещает в себе вызовы git fetch и git merge. Т.е. сначала идёт получение нового состояния ветки в удалённом репозитории, а затем автоматическое добавление новых данных в локальную ветку. При этом если ваша ветка уже находится в актуальном состоянии, то при вызове git pull будет выведено сообщение “Already up to date.”.

Git push

Записывает локальные коммиты в удалённый репозиторий. Может принимать два параметра: название (или URL-адрес) удалённого репозитория и название ветки, изменения с которой мы хотим записать. Также в некоторых ситуациях данная команда может быть вызвана и без параметров.

Еще полезные команды Git

Для начала, конечно, стоит разобраться с командами с предыдущих слайдов, ибо они составляют основу, без которой сложно себе представить работу с системой контроля версий Git. Однако сразу после их усвоения на должном уровне, крайне рекомендую ознакомиться со следующим набором полезных команд:

1. git stash — сохранение текущего состояния репозитория и очищение директории от всех изменений;
2. git cherry-pick — вставка отдельного (-ых) коммита (-ов) в свою ветку;
3. git rebase — перемещение нескольких коммитов к новому базовому коммиту.
4. git reset — с помощью этой команды можно удалять коммиты, отменять их без удаления, восстанавливать файлы из истории.

Git stash / cherry-pick / rebase / reset

Git stash

git stash – данная команда прячет ваши изменения
git stash save “сообщение” – прячет изменения и сохраняет их с названием, чтобы потом проще их найти

Все stash хранятся в стеке (создает отдельно лежащий коммит)

git stash list – посмотреть все stash коммиты
git stash apply – достать последнее спрятанное
git stash pop – достать последнее спрятанное и сразу удалить его из стека

Git cherry-pick

Cherry-pick команда, которая позволяет выбирать произвольные коммиты git по ссылке и добавлять их к текущей ветке. (Чаще всего используется для копирования определенных коммитов из одной ветки в другую)

git cherry-pick <commit-hash> – commit-hash можно получить с помощью команды git log

Git rebase

git rebase <branch> - данная команда перенесет текущую ветку в конец ветки <branch>;
Флаг -i (интерактивный rebase – интерактивное представление всех выбранных коммитов где доступны следующий выбор команд: pick, reward, edit, squash, fixup)

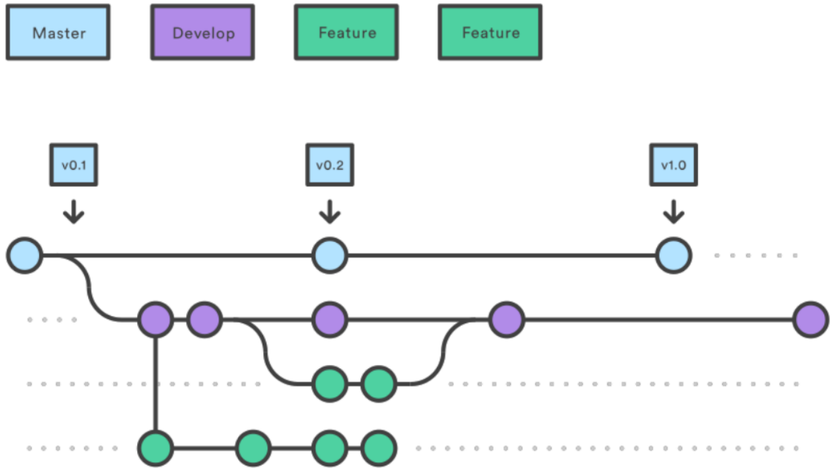
Git reset

git reset – это универсальный инструмент для отмены изменений.
Дополнительные опции(флаги):

1. --hard (указатели в истории коммитов обновляются на указанный коммит, затем происходит сброс раздела проиндексированных файлов и рабочего каталога до указанного коммита)
2. --mixed (режим по умолчанию, указатели ссылок обновляются, раздел проиндексированных файлов сбрасывается до состояния указанного коммита, любые изменения, которые были отменены в разделе проиндексированных файлов, перемещаются в рабочий каталог)
3. --soft (обновление указателей, и на этом операция сброса останавливается. Раздел проиндексированных файлов и рабочий каталог остаются неизменными)

Git Flow

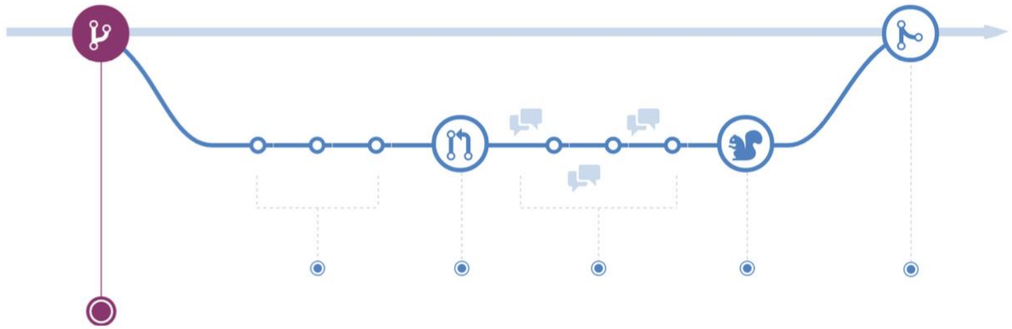
Его суть сводится к организации рабочего процесса, построенного с использованием Git.
Git Flow задаёт строгую модель для надёжного управления и расширения более менее крупных проектов.



Создание новой ветки

В любой момент времени работы на проекте у вас наверняка будет целая куча разных фич или даже собственных идей в процессе разработки. Некоторые из них уже могут быть готовы, другие — нет.
Именно для такой ситуации и существует возможность создания множества веток.

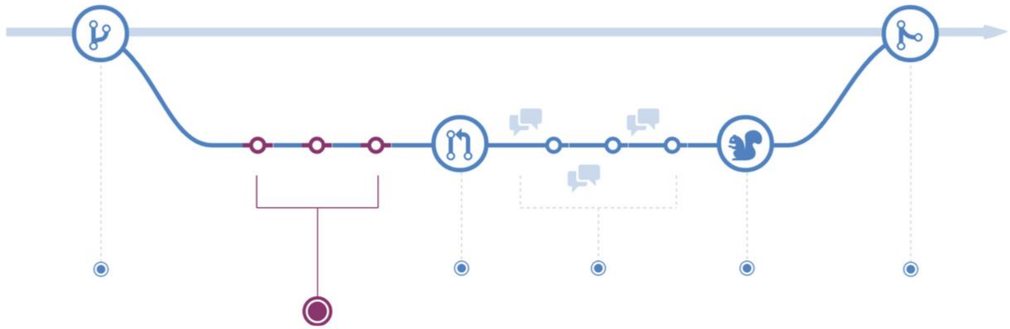
Всякий раз, создавая новую ветку в вашем проекте, вы как бы создаёте свою маленькую экосистему, которая отталкивается от текущего состояния проекта, но никак не влияет на него вплоть до момента, пока данные изменения не будут просмотрены, протестированы и внесены в основную ветку. Иными словами вы вольны экспериментировать и коммитить любые изменения в свою ветку не опасаясь “что-нибудь сломать” в основной версии проекта.



Добавление изменения

Когда ваша ветка создана — время вносить изменения. Когда вы добавляете, редактируете или удаляете файлы, вы должны создавать коммиты на основе этих изменений и периодически добавлять их с свою ветку. Данный процесс добавления коммитов позволяет отследить ваш прогресс по введению новой фичи в проект.

Кроме этого коммиты добавляют прозрачности истории вашей работы, которую впоследствии другие члены команды могут прочесть и понять, что было сделано и почему. Каждый коммит должен иметь сообщение с описанием внесённых изменений. Также при соблюдении этих правил не составит никаких проблем “откатить” вашу ветку на несколько коммитов назад в случае обнаружения бага.

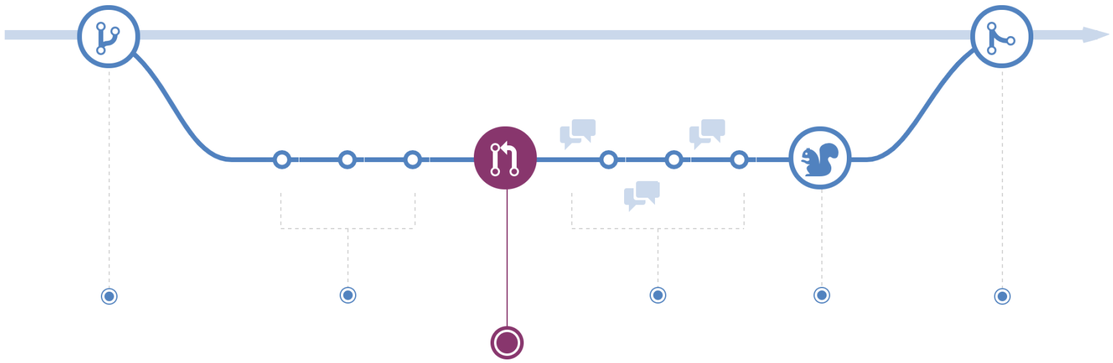


Создание запроса на внесение изменений (pull/merge)

Запрос на внесение изменений (в дальнейшем ПР) запускает процесс обсуждений ваших изменений. Т.к. данный ПР производится в рамках одного репозитория Git, каждый из членов команды может увидеть полный перечень изменений, которые будут внесены в случае апрува.

Вы можете создать ПР на любом этапе процесса разработки: когда у вас есть лишь наработки и вы хотите обсудить ваши идеи с кем-то еще, когда вы застряли на каком-то из этапов и вам нужна помощь или совет, или же когда ваша работа полностью готова.

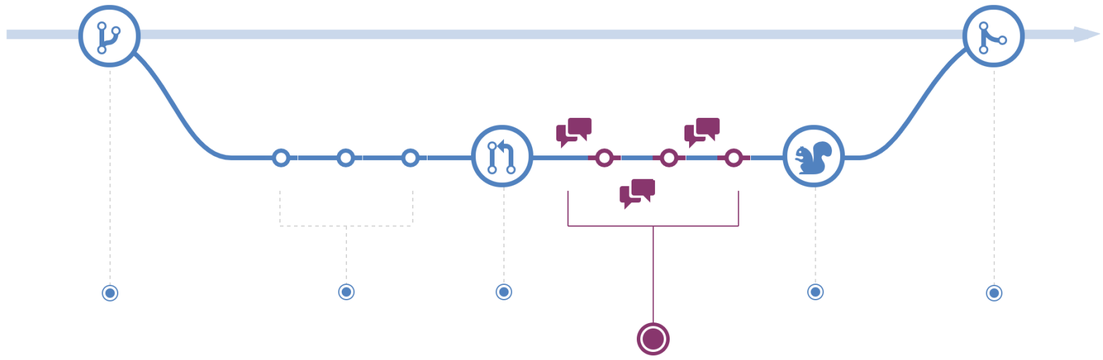
С помощью системы @упоминаний в сообщениях к вашему ПР-у вы можете запросить отзыв у конкретных людей из команды.



Обзор и обсуждение кода (внесение новых изменений)

С момента создания ПР-а любой член вашей команды может сделать ревью кода, задавая при этом вопросы и оставляя комментарии. Такое может произойти если, например, не соблюдён кодстайл проекта, или для новой функции не написан юнит тест, а может наоборот всё выглядит замечательно и ваш подход к решению задачи оказался неочевидным, но очень действенным.

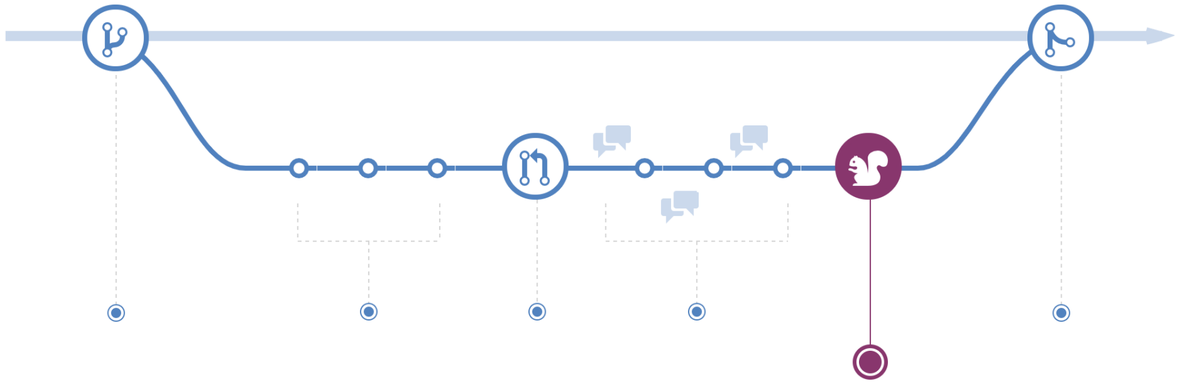
ПР-ы буквально созданы для организации такого вида коммуникаций.
Даже в процессе обсуждений и отзывов о ваших изменениях вы можете продолжать добавлять новые коммиты. Если вы что-то забыли или какой-то фрагмент кода стал причиной возникновения ошибки, вы можете поправить это в своей ветке и тут же “запушить”. Новые коммиты также будут появятся в ПР-е.



Развертывание кода и финальное тестирование

С помощью GitHub вы можете развернуть продакшн вместе с изменениями из вашей ветки еще до слияния с веткой main (master).
Когда ПР получил необходимые апрувы и все тесты прошли успешно, вы можете развернуть ваши изменения, чтобы проверить их прямо на продакшене.
Если новый код становится причиной возникновение ошибок, вы можете “откатить” его обратно (просто развернув на продавшее версию проекта с ветки main).

В разных командах также могут существовать специальные тестовые или промежуточные окружения. Данный подход на сегодняшний день является крайне популярным. Однако для некоторых именно вариант с продакшеном может оказаться наиболее подходящим.



Внесение изменений в основную ветку

Когда ваши изменения были утверждены на продакшене, самое время внести эти изменения в ветку main.

После добавления изменений, все ваши коммиты сохраняются в общей истории кодовой базы проекта. Благодаря этому в будущем любой разработчик сможет вернуться к вашему коду и понять, почему и как были приняты те или иные решения.