GeekBrains

Git. Базовый курс

Урок 11

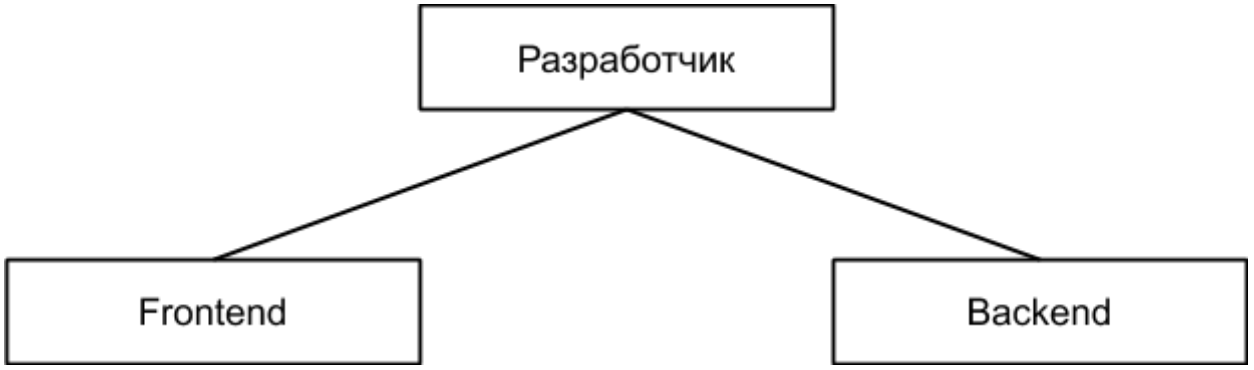
Gitflow

Роли в команде

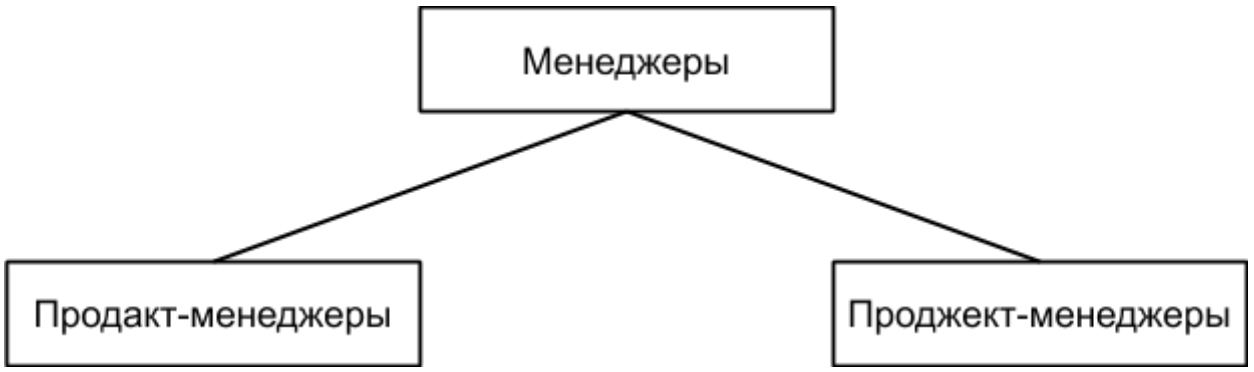
По мере того, как проект и обслуживающая его команда разрастается, необходима организация коллективной работы. В зависимости от программного продукта, коллектив может состоять из множества ролей: разработчиков, тестировщиков, менеджеров, заказчиков или замещающих заказчика бизнес-оунеров.

Разработчики создают программный код, тестировщики проверяют продукт на наличие ошибок и соответствие требованиям, менеджеры координируют работу команды, заказчик принимает продукт и описывает требования к нему.

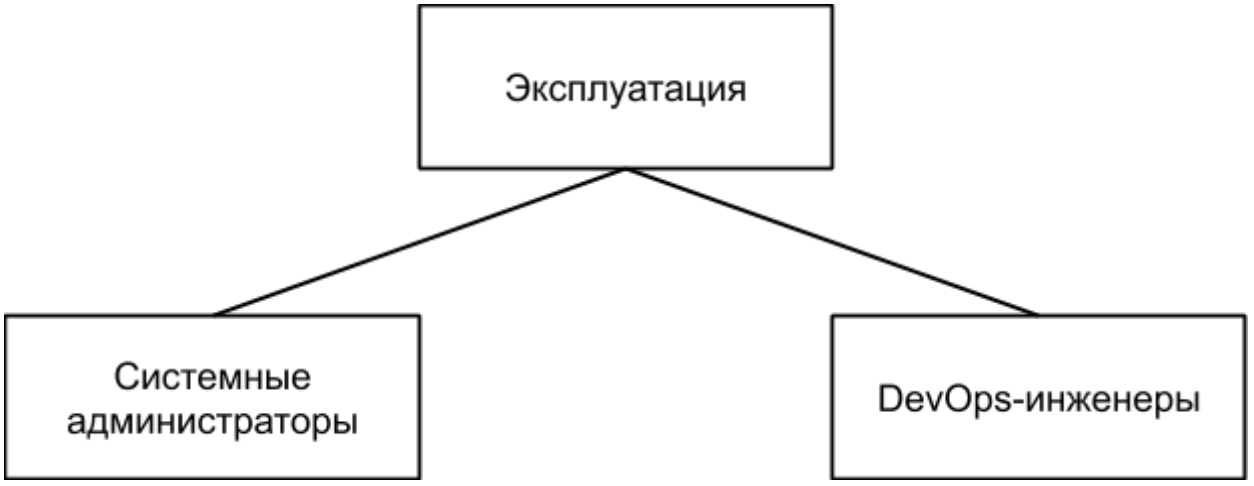
Конечно, когда проект небольшой, все эти роли могут схлопываться в одного человека, однако по мере роста клиентской базы и, возможно, доходов, команда неизбежно разрастается.



Более того, каждая из представленных ролей может дробиться еще на несколько подролей. Например, в веб-разработке программисты делятся на frontend- и backend-разработчиков.



Менеджеры могут подразделяться на продакт-менеджеров, которые продумывают логику проекта с точки зрения пользователей, и на проджект-менеджеров, которые смотрят на проект с точки зрения команды, следят за правильной организацией работы, защищают команду от переработок или отвлекающих входящих обращений.



При наличии в компании собственной облачной инфраструктуры служба эксплуатации может включать в себя системных администраторов и DevOps-инженеров, более того, они могут быть разбиты на несколько отделов.

Gitflow

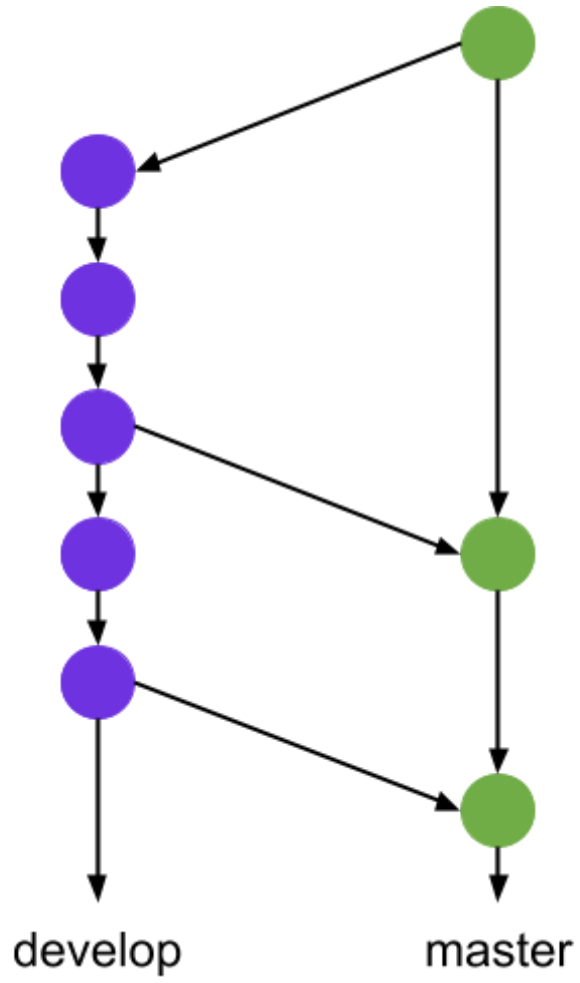
В любом случае, успешные проекты склонны к укрупнению и разрастанию команды. Рано или поздно встает вопрос, как координировать действия большой массы людей. Важно добиться, чтобы на пользователях программы, сайта или мобильного приложения не отражались изменения в новых версиях. Чем больше пользователей использует программу, тем это становится важнее.

Новый функционал в продукт обычно добавляется небольшими порциями, которые выпускают в качестве релиза. В релизе важно добавить новые возможности, проверив, что он работает так, как задумано, при этом не ломается старый функционал, разработанный в предыдущих релизах.

Решается эта задача автоматическим и ручным тестированием. Полностью автоматизировать тестирование в более или менее сложном проекте не получится. Прогнать каждую строку кода со всеми возможными вариантами данных слишком накладно и потребует усилий, времени и ресурсов на много порядков больше, чем на разработку самой программы. Поэтому помимо автоматического тестирования применяется ручное, при помощи которого тестировщики вскрывают наиболее часто повторяющиеся ситуации и ошибки. Они становятся кандидатами для создания автоматического теста.

Таким образом, тестирование — это главный барьер на пути обновленной программы к пользователю. Необходимо так организовать работу команды, чтобы код обязательно прошел проверку тестировщиками. В этом помогает Gitflow.

Gitflow — это набор правил ветвления, который позволяет управлять релизами проекта. Эти правила определяют, какие ветки нужно создавать и как производить их слияние.



Для работы по Gitflow создаются две постоянные ветки: master и develop. Это долгоиграющие ветки, они существуют на всем протяжении жизненного цикла проекта.

С веткой master мы уже знакомы, она создается по умолчанию при создании репозитория. Она считается главной, в каждый момент времени код в ней должен быть готов к релизу. Это может быть

новый или старый релиз, однако выкатка из него в продакшен-среду никогда не должна приводить к поломке.

Ветка develop используется для разработки, в ней должны содержаться самые последние изменения, необходимые для следующего релиза. Когда исходный код в ветке develop достигает стабильного состояния и готов к релизу, все изменения должны быть влиты в ветку master и помечены тегом с номером релиза.

Каждый раз, когда изменения вливаются в ветку master, получается новый релиз.

Помимо главных веток (master и develop), используются вспомогательные. Они нужны для распараллеливания разработки и тестирования. В отличие от главных веток, они всегда имеют ограниченный срок жизни. Каждая из них в конечном итоге рано или поздно удаляется.

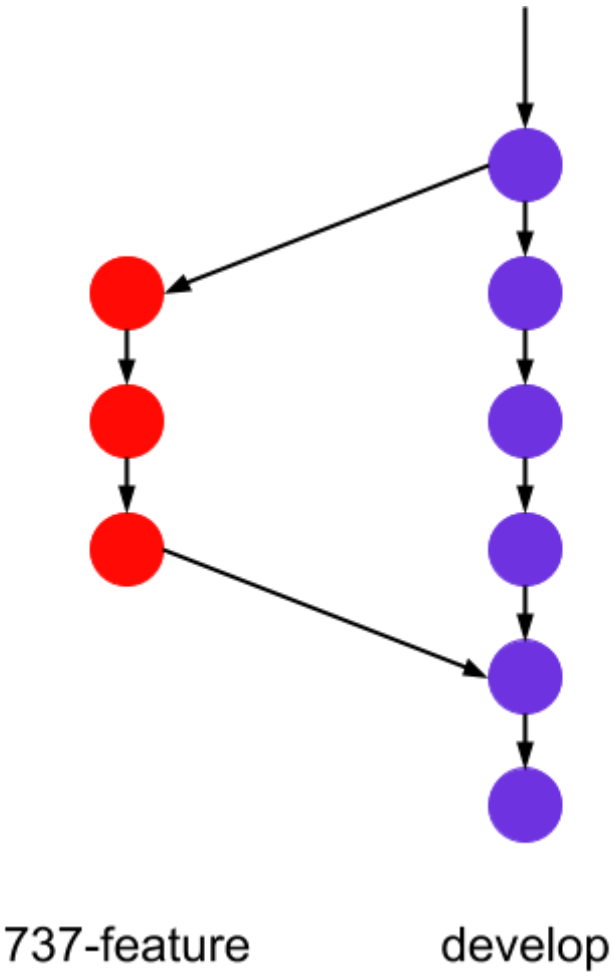
В качестве вспомогательных веток используются:

- ветки задач, в которых разработчики решают конкретные поставленные задачи,
- релизные ветки для сбора релиза,
- hotfix-ветки для срочных исправлений после релизов.

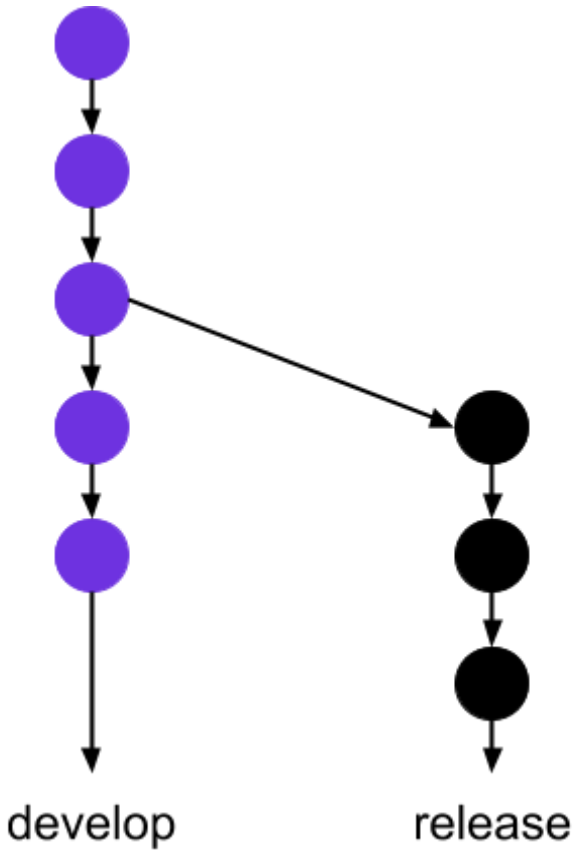
Давайте рассмотрим их более подробно.

Ветки задач используются для разработки новых функций, которые должны появиться в текущем или будущем релизах. Как правило, они отпочковываются от ветки develop, хотя это необязательно. В начале работы над задачей может быть еще не известно, в какой именно релиз она будет добавлена. Ветка живет столько, сколько продолжается разработка функциональности. После того, как функционал разработан, ветка сливается в develop.

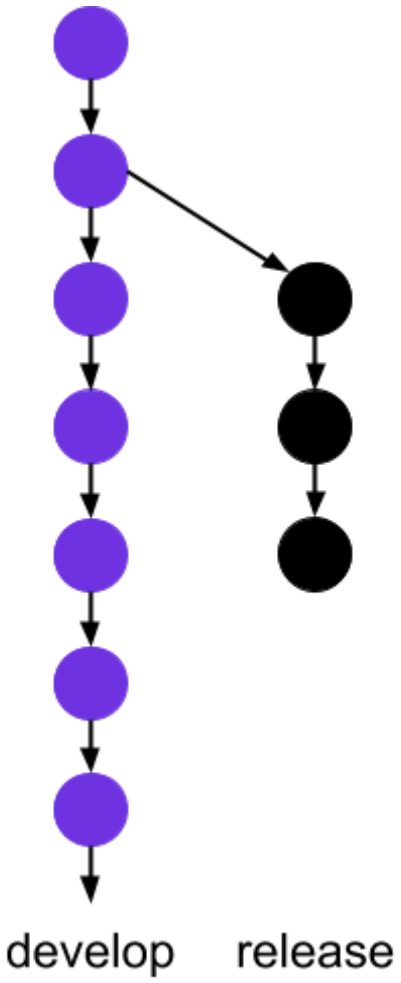
Как правило, в трекере задач заводится тикет на функционал, номер задачи обычно добавляется в название ветки, чтобы в любой момент можно было бы обратиться к трекеру за подробным описанием.



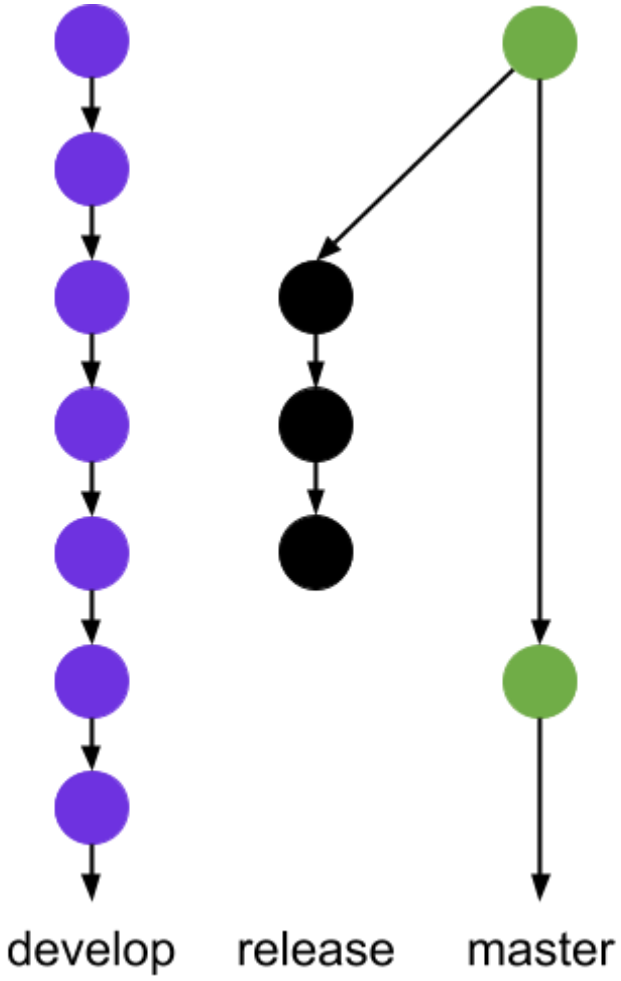
Релизная ветка используются для подготовки к выпуску новых версий продукта. Она нужна, чтобы в релиз попало не все содержимое develop-ветки, а лишь избранные задачи, например, ряд фич, которые пока нестабильны или не готовы к выкатке.



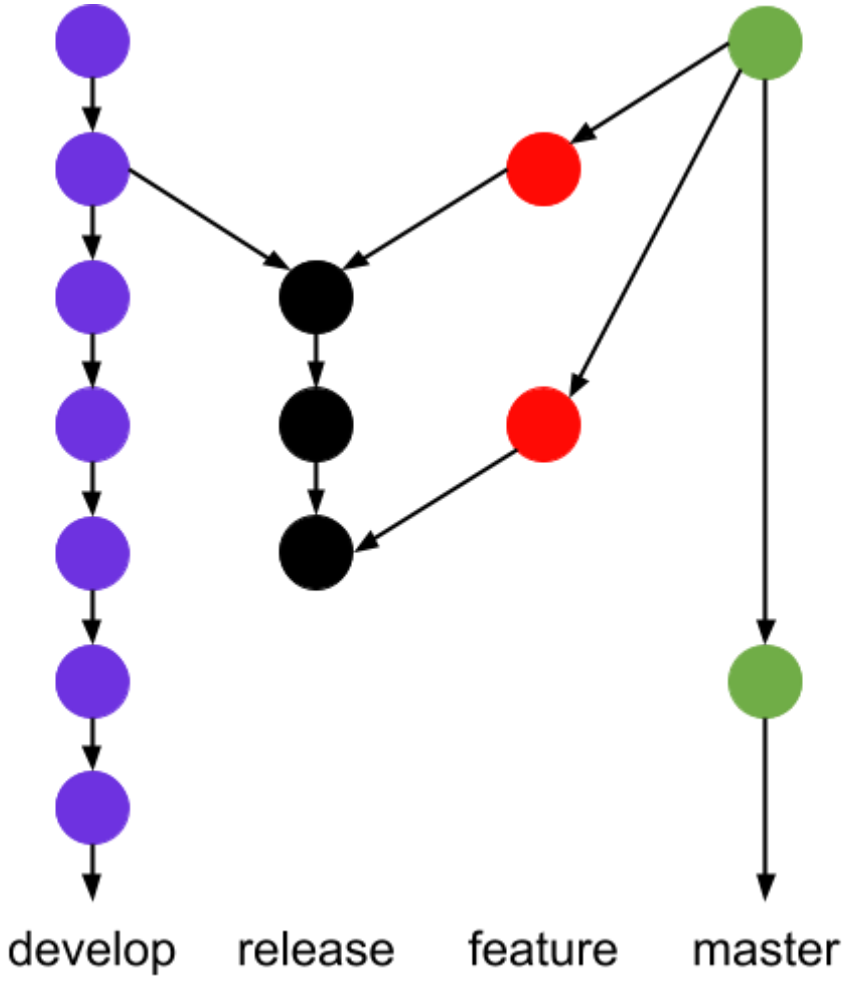
Можно собрать в релизную ветку все задачи, которые должны войти в релиз. Так появляется возможность спокойно ее тестировать и стабилизировать для запланированного релиза, оставив ветку develop свободной для разработки релизов.



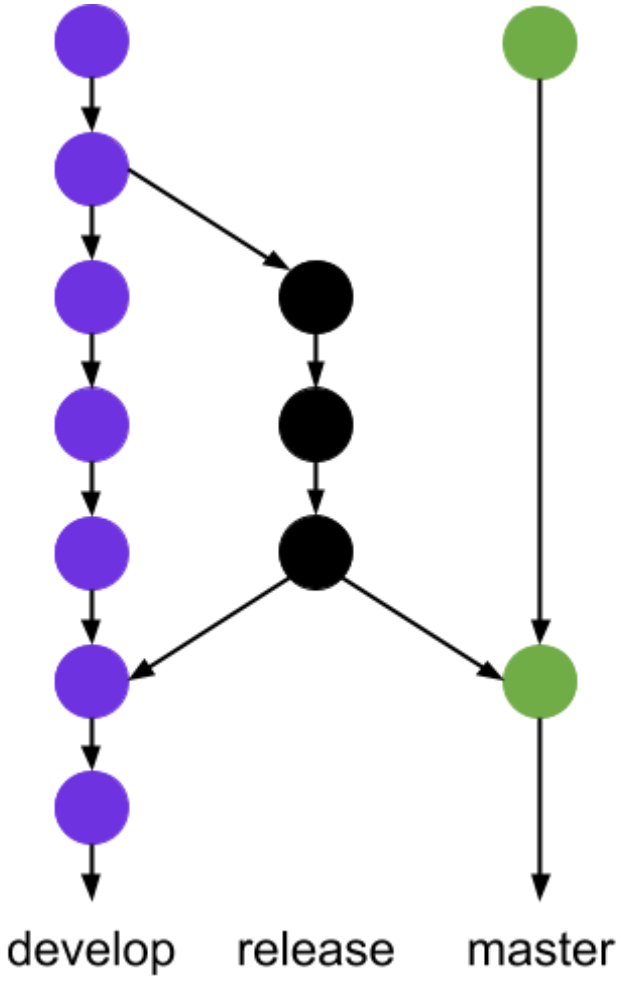
Существует два подхода для формирования релизной ветки. Согласно первому, она создается заранее, перед стартом релиза, отпочковать ее можно от develop, если он стабилен, либо от ветки master.



В этом случае ветки задач отпочковываются от master и сливаются в релизную ветку. По мере тестирования и нахождения ошибок ветка исправляется.

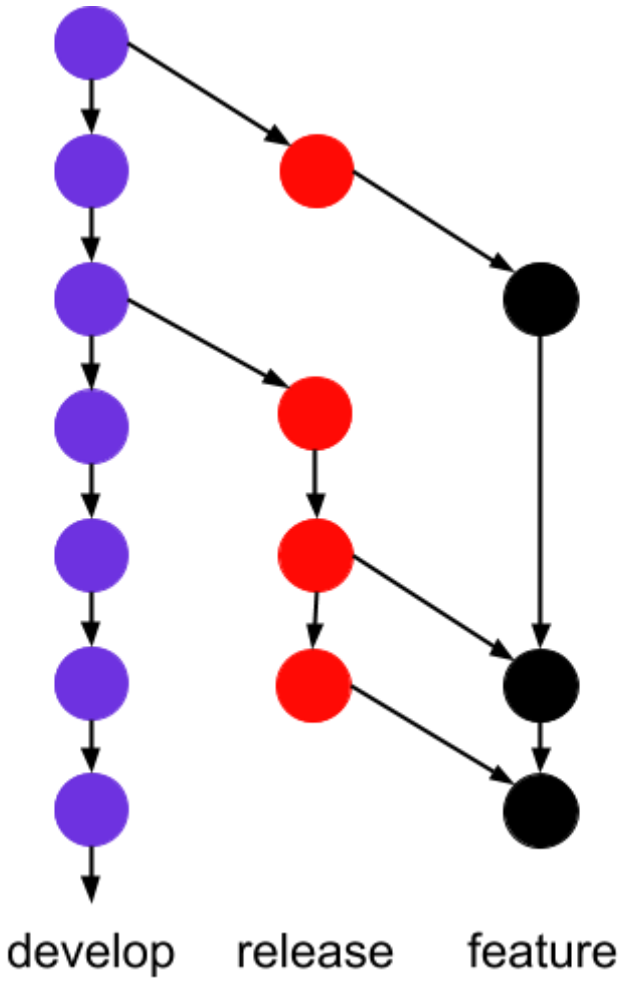


В конце релизная ветка сливается и в develop, и в master.

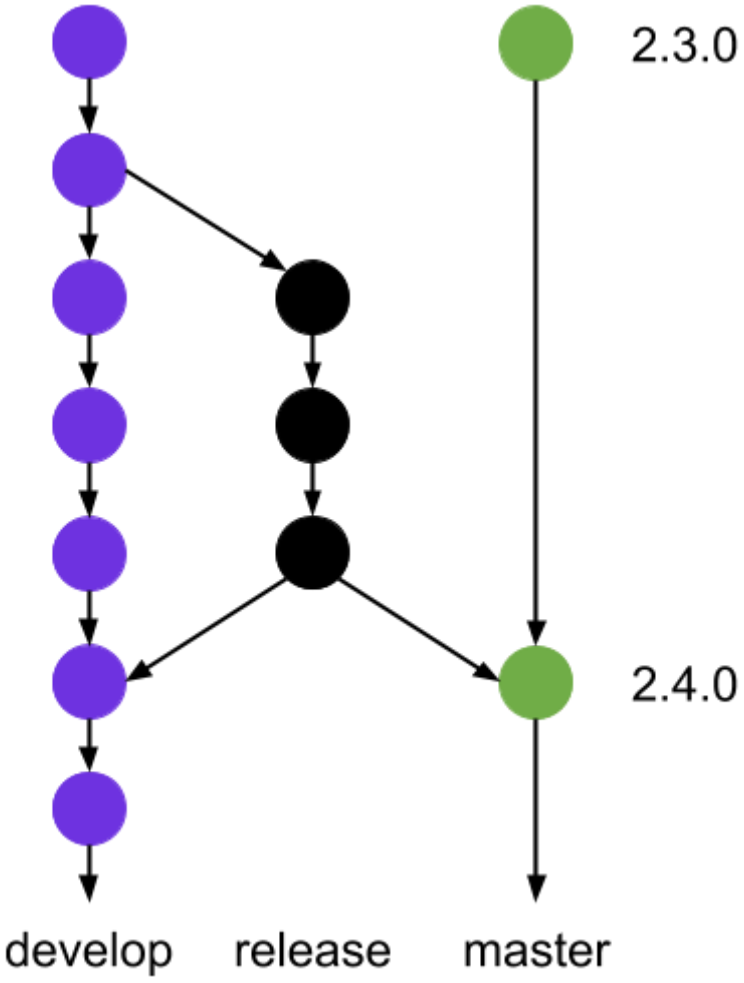


Согласно второму подходу, релизная ветка формируется из веток задач по мере их готовности. При этом ветки задач не удаляются, в любой момент можно пересобрать релизную ветку, исключив из нее ту или иную задачу.

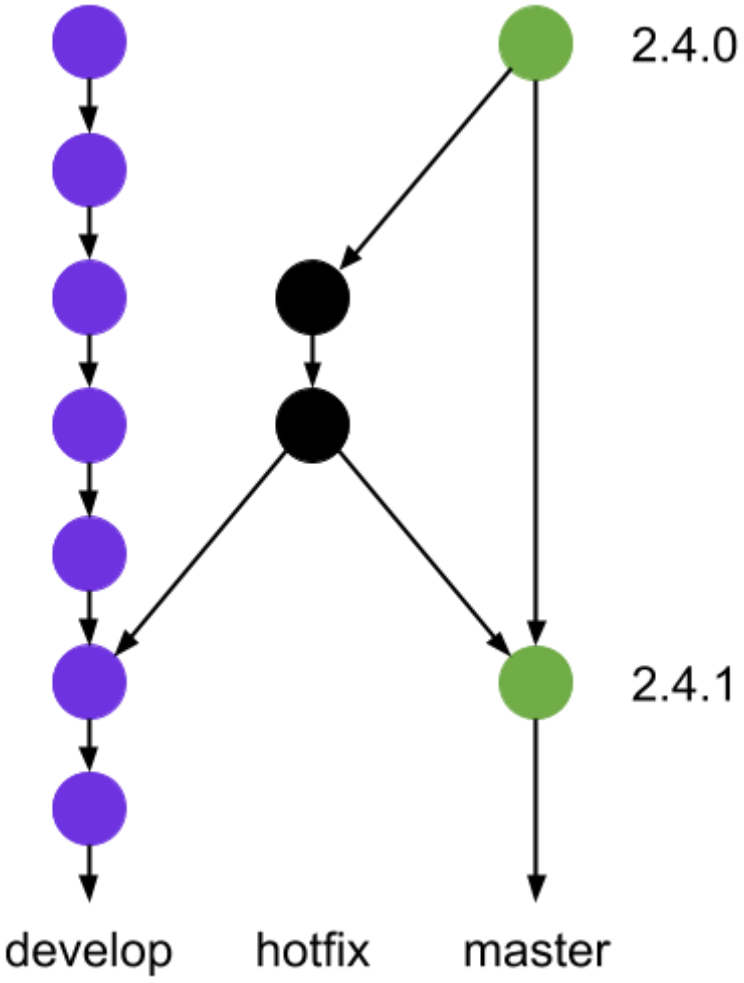
При таком подходе важно отпочковывать ветки задач от master, и вносить исправления именно в ветки задач, ни в коем случае не создавая коммиты в релизной ветке. Это связано с тем, что ветка может быть удалена и собрана снова.



После того, как релизная ветка сливается в master, она может быть удалена, считается, что master-ветка готова к релизу.



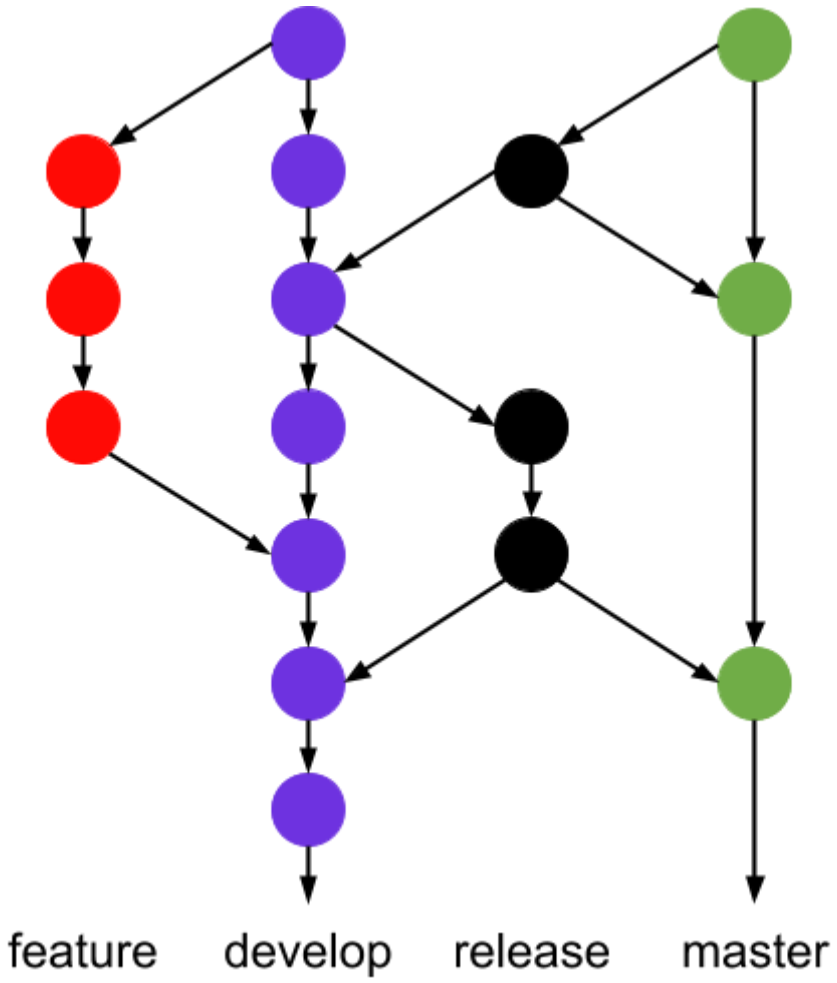
При этом на ветку master ставится тэг с версией продукта. Как правило, при каждом релизе увеличивается минорная версия, т. е. средняя цифра в версии.



Hotfix-ветки предназначены для исправлений. Бывает не просто протестировать все до конца перед выпуском продукта. Продакшен-среды могут быть больше и сложнее устроены, включать интеграции, которые сложно обеспечить на тестовых стендах. Конечным вариантом продукта пользуется более широкая аудитория, поэтому после выпуска релиза могут вскрываться дополнительные ошибки или отклонения от нормального поведения. Для их исправления используются hotfix-ветки.

Как правило, изменения, которые нужно внести в кодовую базу, очень небольшие и очевидные. Если исправление ситуации требует длительного исследования или много времени на исправление, там, где это возможно, релиз стараются откатить на предыдущую стабильную версию. Hotfix-ветки всегда отпочковываются от master-ветки и сливаются и в master, и в develop.

Так как хотфикс сливается непосредственно в ветку master, на него тоже ставится версия, при этом увеличивается номер патч-версии, т. е., последняя цифра.



Таким образом, у нас имеется четыре параллельных потока:

- ветки задач (feature),
- develop-ветка с последней версией продукта,
- релизная ветка с хотфиксами (release),
- стабильная ветка master.

Идеально, если под каждый из этих потоков имеется свой собственный стенд:

- master-ветка — это продакшен-окружение;
- релизная ветка и хотфиксы — это препрод, среда максимально близкая по своим параметрам и состоянию к продакшен-окружению;
- develop-ветка — это стейджинг-окружение;
- ветки задач — либо на втором стейджинге, либо на dev-машинах.

Разумеется, различные компании могут себе позволить разные расходы на инфраструктуру. У кого-то может быть лишь продакшен-окружение, у кого-то в дополнение может быть лишь препрод или один стейджинг.

У кого-то может быть полный набор окружений, десятки стейджей или облачная инфраструктура, позволяющая разворачивать для тестирования каждую ветку задач. В зависимости от количества окружений, работа отдела тестирования строится по-разному. Чем больше стейджей, тем тщательней можно проверить продукт, быстрее и надежнее воспроизвести и локализовать ошибку.

Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

- Официальная документация Git: **https://git-scm.com/book/ru/v2.**