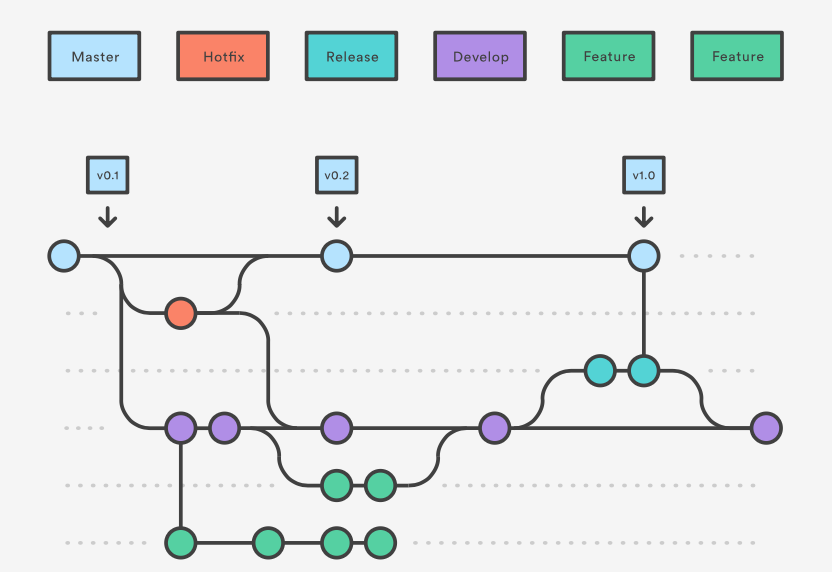
[Git Flow](http://confluence.moscow.alfaintra.net/display/HADOOP/Git+Flow)



Основные принципы Git Flow

* Ветка master отражает актуальное состояние кода на продуктивной среде. Коммит в ветку master означает выпуск новой версии. В момент коммита в ветку master, может создаваться тег с версией кода для удобной навигации.
* Разработка нового функционала ведется в feature-ветках. feature-ветки ответвляются от ветки develop. После того, как разработчик считает, что он закончил разработку в feature-ветке, из нее кидается Pull-request (далее PR) в ветку develop. После merge-а feature-ветки в master (через релизную ветку либо через hotfix), соответствующая feature-ветка больше не нужна и удаляется.
* develop-ветка однократно ответвляется от мастер ветки и содержит максимально свежий код разработки. В Develop ветку через механизм Pull Request'ов заливается код из feature-бранчей. Так же, заливается код из HotFix - бранчей. После того, как Release готов - его код вливается в Develop-ветку, это нужно для отражения всех исправлений релизной ветки в ветке Develop.
* release-ветка служит для выпуска релизов (патчей). Ветвится от develop-ветки и, таким образом, содержит внутри себя определенное количество нового функционала на точку времени ее создания. Release-ветка создается тогда, когда считается, что весь код необходимого функционала залит в Develop-ветку (все нужные Feature реализованы, и PR приняты) и следует приступать к сборке/отладке релиза.

**исключение кода из релиза**

Когда по той или иной причине принимается решение исключить какую-то feature из уже сформированного релиза (т.е. после того как был принят PR и создана release branch), все файлы этой feature откатываются на предыдущие стабильные версии либо удаляются если это новая функциональность. Разработчику надо помнить что при внедрении патча на прод, release-ветка будет влита в ветку develop и там тоже эти файлы откатятся или удалятся. Этот код останется в feature-ветке разработчика и он сможет его повторно отправить через PR в ветку develop как только это понадобится (следующий патч например), сделав предварительно pull из ветки develop и корректно разрешив конфликты при необходимости.

* **Код в релизной ветке может только отлаживаться**: могут устраняться какие-либо конфликты, исправляться код конкретных фич. Новый функционал, который по каким-то причинам не попал в Release - будет включен в следующий релиз. После того, как код в релизной ветке считается финальным осуществляется сборка патча. Затем патч проходит тестирование на тестовой среде. При обнаружении проблем, код исправляется в Release-ветке и осуществляется полная пересборка патча. Когда патч установлен в боевую среду - код из Release ветки сливается с Develop-веткой и сливается с Master-веткой. После слияние с Master и Develop ветками, Release ветка удаляется.
* HotFix-ветка служит для исправления кода, уже внедренного на продуктивную среду. Ответвление происходит от Мастер-ветки. После окончания разработки фикса, код HotFix-ветки заливается в мастер ветку и в develop-ветку. (TBD: сам хотфикс надо где-то тестировать)

Предлагается следующая имплементация:

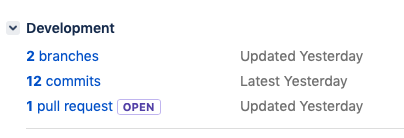
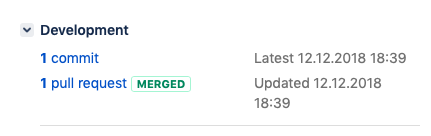
1) Разработчик получает задачу в Jira. Если задача имеет типы Task или UserStory - создается feature-бранч от ветки develop. Для задач типа Bug создается бранч HotFix-бранч от ветки Master, если его планируется установить сразу в prod, или feature-бранч, если он пойдет стандартным путем со следующим патчем. Куда войдут изменения сообщает Тим лид доносит эту информацию до разработчика.  
Сам бранч создается прямо из Jira и имеет имя вида DATALAKE-111-sometext, где DATALAKE-111 - номер задачи, а -sometext - произвольный короткий текст разработчика или название задачи. Например, DATALAKE-231-newIndex.

2) Происходит разработка, разработчик делает локальные commit'ы в свою ветку до тех пор, пока код не оттестирован. Когда код протестирован и есть уверенность что его можно публиковать, разработчик делает pull/push в свою ветку и отправляет Pull Request из своей Feature ветки в Develop. Если разработка велась в HotFix-ветке, Pull Request отправляется и в master и в develop. Когда код правильно смержен исходный бранч удаляется.

Когда PR принят, в Jira разработчик изменяет статус задачи - отправляет ее "**на сборку**".

**Сборка релиза**

1) Сборщик релиза перед созданием Release-бранча проверяет, что PR, созданные разработчиками в develop-ветку имеют статус "Merged". Т.Е. в интерфейсе Jira. проходит по всем карточкам на статусе "Сборка" и проверяет в разделе "Development" статусы pull-requests.



2) Если есть не принятые PR, которые должны пойти в патч, то сборщик релиза принимает (либо отклоняет) PR по необходимым фичам в develop ветку.

3) От ветки develop создается Release-ветвь.

4) Код собирается разработчиком в патч и отправляется в сопровождение на установку в тестовое окружение.

5) По результатам тестов в код релизной ветки вносятся фиксы, в случае необходимости.

6) Если были найдены и исправлены ошибки,то код повторно собирается разработчиком в патч и отправляется в сопровождение.

7) п.5 повторяется, если ошибок нет, то патч отправляется на установку в прод окружение.

8) По результатам установки в бой в код релизной ветки вносятся фиксы, в случае необходимости. Патч установлен в бой

9) Код релизной ветки мержится в мастер-ветку (через PR). **В dev ветке создается тэг (аннотированная метка), привязанная к текущему коммиту (коммит вида**

**Automatic merge from master -> develop).**

10) Код релизной ветки мержится в develop-ветку (через PR), затем релизная ветка удаляется.

**Разработка HotFix**

По смыслу похоже на сборку релиза. По результатам разработки HotFix так же создается мини-релиз. В таких ветках осуществляется только критичные исправления неработающего кода. Новая функциональность по старым потокам разрабатывается в рамках Feature-веток.

1) Разработчик ответвляет Hot-Fix ветку от Master-ветки.

2) В HotFix ветку заливается код исправления.

3) Собирается HotFix патч

4) HotFix патч отправляется на установку в тестовое окружение

5) По результатам тестов в код HotFix ветки вносятся изменения, в случае необходимости

6) Если были найдены и исправлены ошибки, то код повторно собирается разработчиком в патч и отправляется в сопровождение.

7) п.5 повторяется, если ошибок нет, то патч отправляется на установку в прод окружение.

8) По результатам установки в бой в код HotFix ветки вносятся фиксы, в случае необходимости. HotFix установлен в бой

9) Код HotFix-ветки мержится в мастер-ветку (через PR). **В dev ветке создается тэг (аннотированная метка), привязанная к текущему коммиту (коммит вида**

**Automatic merge from master -> develop).**

10) Код HotFix-ветки мержится в develop-ветку (через PR), затем HotFix-ветка удаляется.

Информацию о подходе в целом можно почитать тут:

<https://habr.com/post/106912/>  
<https://nvie.com/posts/a-successful-git-branching-model/>