GIT 브랜치 전략

브랜치 전략이란 **여러 개발자가 하나의 저장소를 사용하는 환경에서 저장소를 효과적으로 활용하기 위한 work-flow다.**

브랜치의 생성, 삭제, 병합 등 git의 유연한 구조를 활용해서, 각 개발자들의 혼란을 최대한 줄이며 다양한 방식으로 소스를 관리하는 역할을 한다.

즉, **브랜치 생성에 규칙을 만들어서 협업을 유연하게 하는 방법론**을 말한다.

만일 브랜치 전략이 없으면?

브랜치 전략이 없을 때 단점은 깃을 사용한지 얼마 안됐던 개발자라면 모두 겪어본 상황들일 것이다.

- 어떤 브랜치가 최신 브랜치지?
- 어떤 브랜치를 끌어와서 개발을 시작해야 하지?
- 어디에 Push를 보내야 하지?
- 핫픽스를 해야하는데 어떤 브랜치를 기준으로 수정해야할까?
- 배포 버전은 어떤 걸 골라야하지?

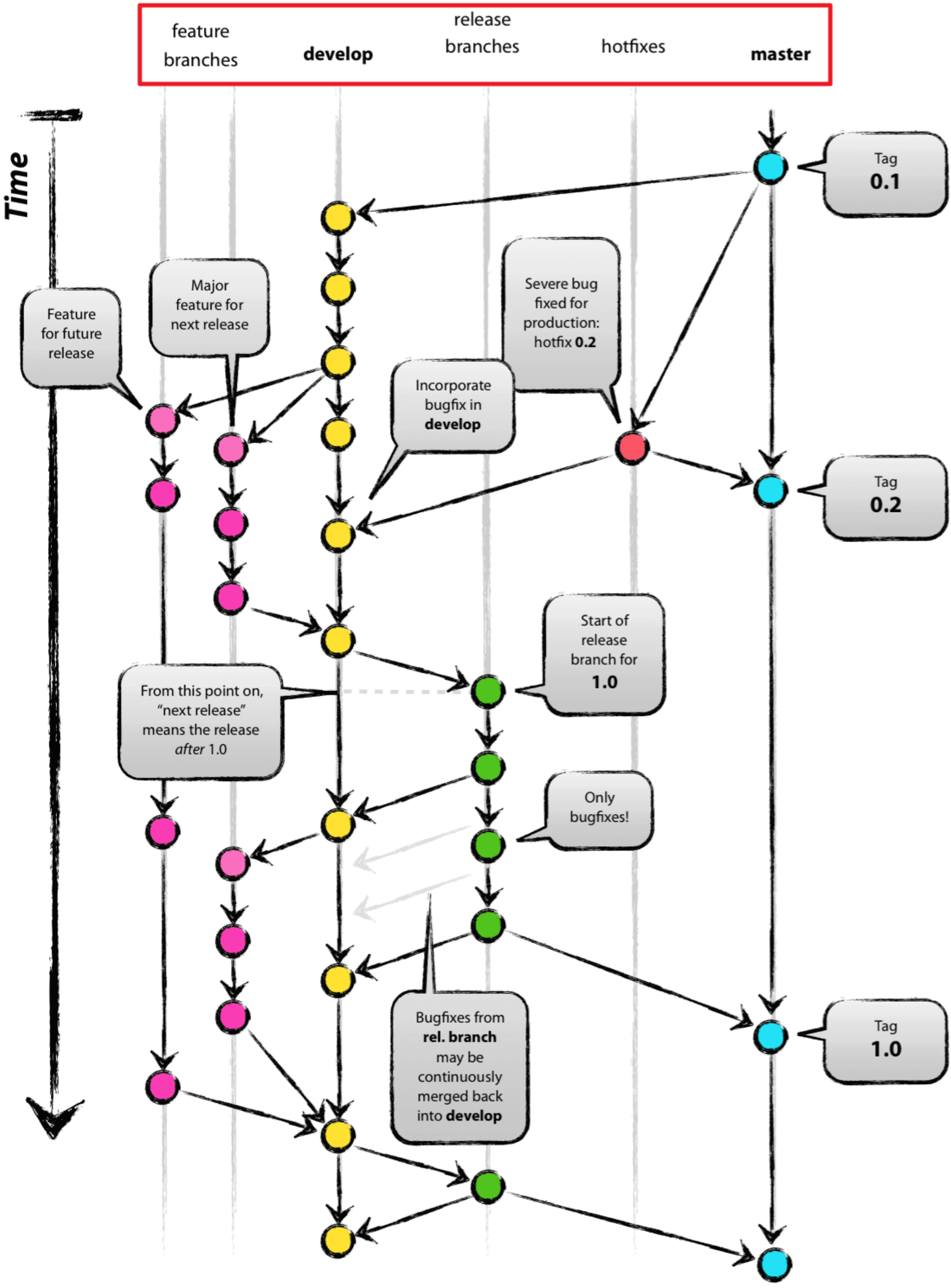
규모가 어느정도 이상 되는 저장소를 상대로 충분히 겪어본 상황일 것이다.
이런 상황을 최소화하기 위해 사용되는 것이 바로 브랜치 전략이다.

이번 시간에는 가장 널리 사용되는 2가지 브랜치 전략에 대해 알아보는 시간을 가져보겠다.

git-flow 전략

github-flow 전략

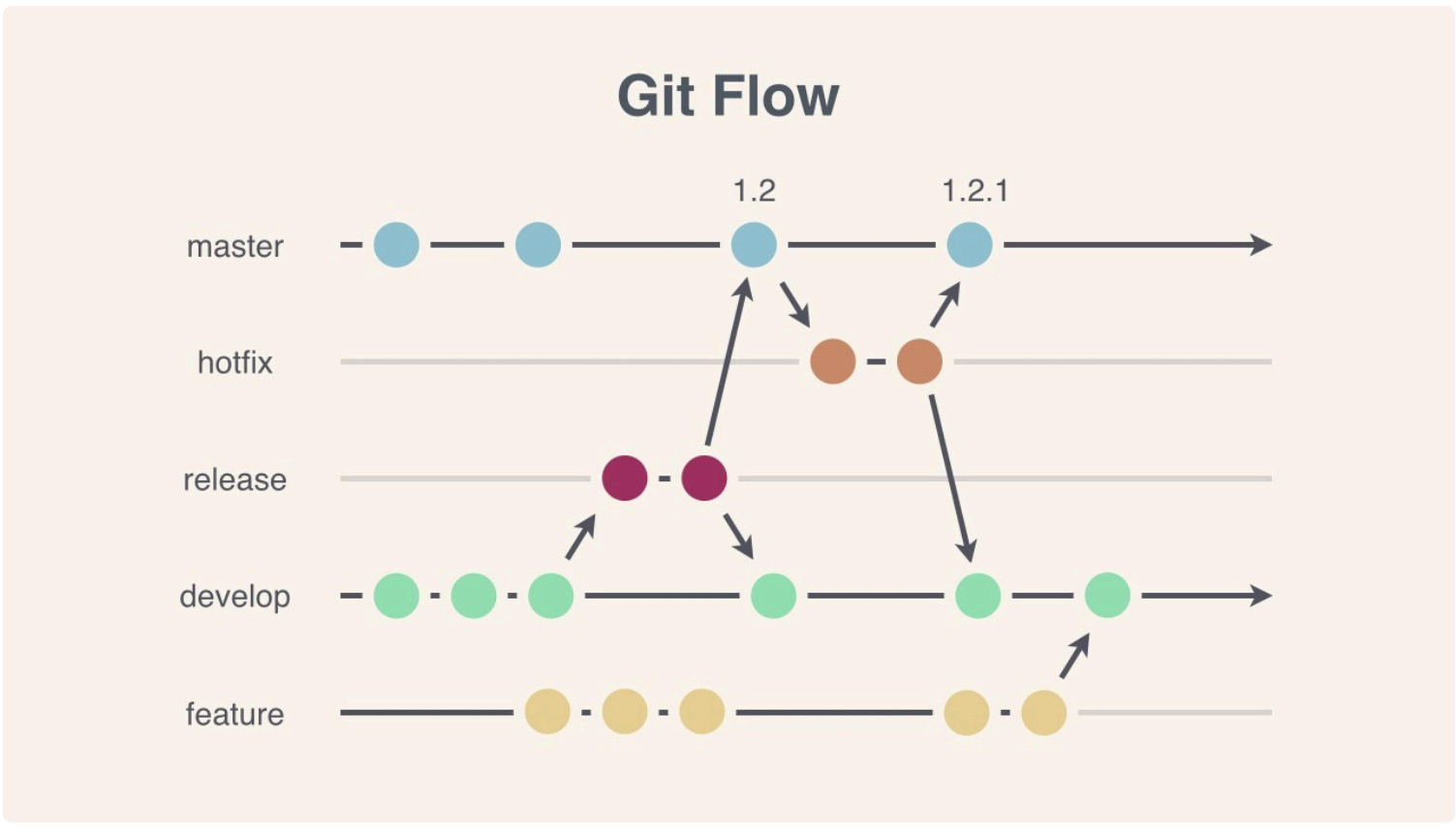
GIT-FLOW 전략



- 기본적인 가지의 이름은 아래의 5가지로 구분하곤 한다.
- **feature > develop > release > hotfix > master**

- 위 순서들은 왼쪽으로 갈수록 포괄적인 가지이며 master branch를 병합할 경우 그 왼쪽에 있는 hotfix 등 모든 가지들에 있는 커밋들도 병합하도록 구성하게 된다.
- 5가지 중, **항시 유지되는 메인 브랜치 master, develop** 2가지와 **merge 되면 사라지는 보조 브랜치 feature, release, hotfix** 3가지로 구성된다.

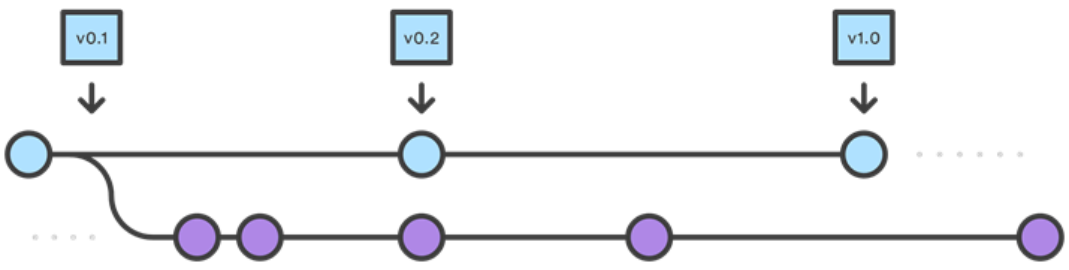
Git-flow 브랜치 구조



- **master** : 라이브 서버에 제품으로 **출시**되는 브랜치.
- **develop** : 다음 출시 버전을 대비하여 **개발**하는 브랜치.
- **feature** : **추가 기능** 개발 브랜치. develop 브랜치에 들어간다.
- **release** : **다음 버전** 출시를 준비하는 브랜치. develop 브랜치를 release 브랜치로 옮긴 후 QA, 테스트를 진행하고 master 브랜치로 합친다.
- **hotfix** : master 브랜치에서 발생한 **버그를 수정**하는 브랜치.

메인 브랜치

메인 브랜치는 **master 브랜치**와 **develop 브랜치** 두 종류를 말한다.



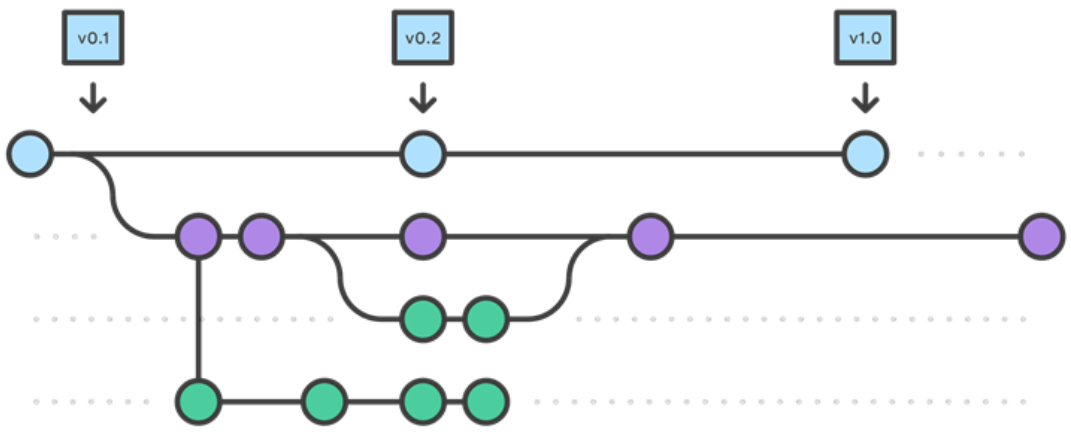
master 브랜치는 **배포 가능한 상태**만을 관리하는 브랜치를 말하며,

develop브랜치는 다음에 **배포할 것을 개발**하는 브랜치이다.

즉 develop 브랜치는 통합 브랜치의 역할을 하며, 평소에는 이 브랜치를 기반으로 개발을 진행한다.

보조 브랜치

보조 브랜치는 **피처 브랜치(feature branch)** 또는 **토픽 브랜치(topic branch)를** 말한다.



- **가지가 뻗어나오는 곳** : develop
- **뻗어나갔던 가지가 다시 합쳐지는 곳** : develop
- **이름 설정** : master, develop, release-*, hotfix-*를 제외하기만 하면 자유롭게 이름 설정이 가능하다.
- **새로운 기능을 추가**할 때 주로 사용하는 가지이다.

master 브랜치에서 develop 브랜치를 만들었고,
develop 브랜치에서 다시 **feature 브랜치를 나눠 작업**을 하고 있는 것을 그림을 통해 알 수 있다.

develop 브랜치에는 기존에 잘 작동하는 개발코드가 담겨있으며,

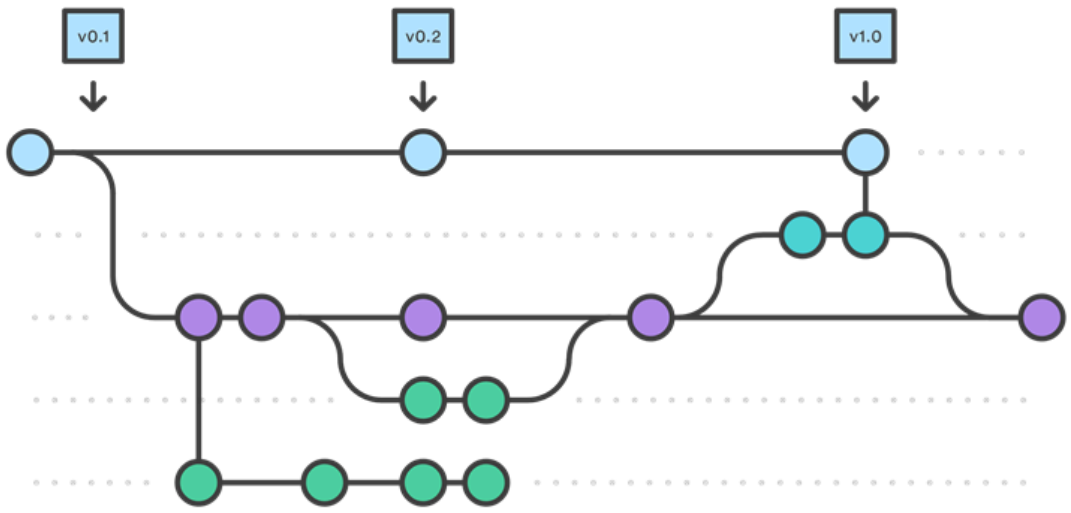
보조 브랜치는 새로 변경될 개발코드를 분리하고 각각 보존하는 역할을 한다.

즉, 보조 브랜치는 **기능을 다 완성할 때까지 유지하고, 다 완성되면 develop 브랜치로 merge 하고 결과가 좋지 못하면 버리는 방향**을 취한다.

보조 브랜치는 보통 개발자 저장소에만 있는 브랜치고, origin에는 push하지 않는다.

릴리즈 브랜치(release branch)

릴리즈 브랜치는 **배포를 위한 최종적인 버그 수정** 등의 개발을 **수행**하는 브랜치를 말한다.

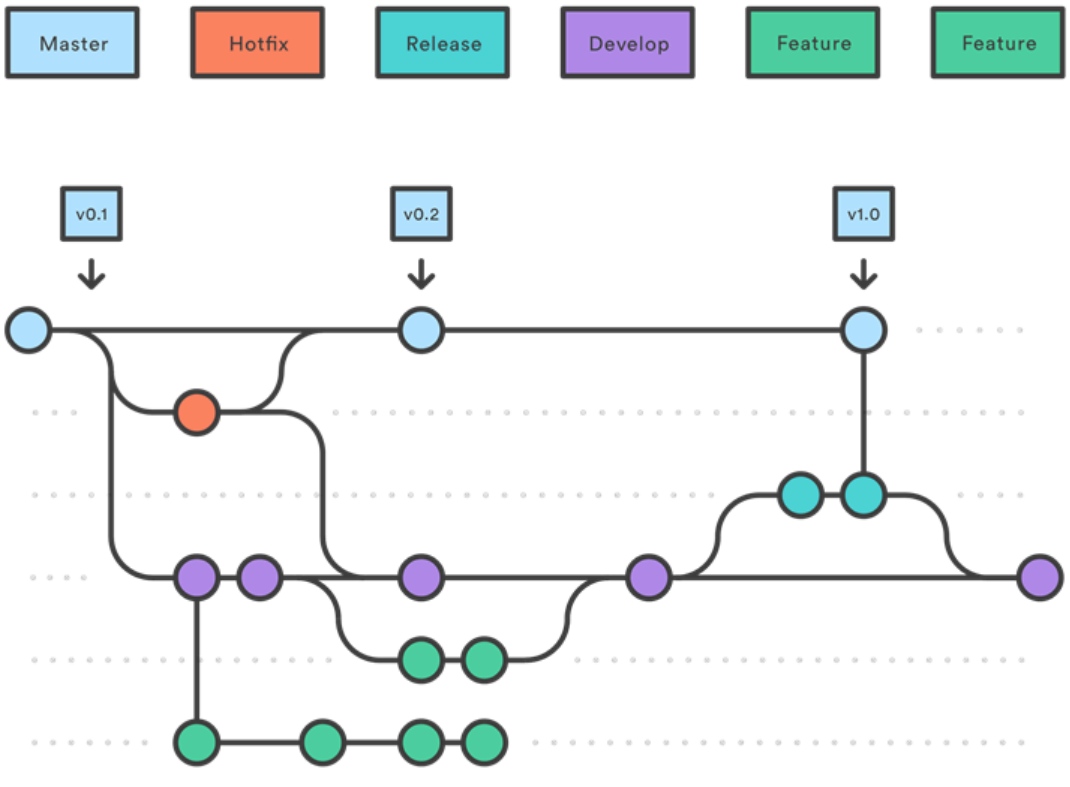


- **가지가 뻗어나오는 곳** : develop
- **뻗어나갔던 가지가 다시 합쳐지는 곳** : develop, master
- **이름 설정** : release-*
- 새로운 **제품을 배포**하고자 할 때 사용하는 가지이다.

develop 브랜치에 버전에 포함되는 기능이 merge 되었다면 QA를 위해 develop 브랜치에서부터 release 브랜치를 생성한다.
배포 가능한 상태가 되면 master 브랜치로 병합시키고, **출시된 master 브랜치에 버전 태그(ex, v1.0, v0.2)를 추가**한다.
release 브랜치에서 기능을 점검하며 발견한 버그 수정 사항은 develop 브랜치에도 적용해줘야 한다.
그러므로 배포 완료 후 develop 브랜치에 대해서도 merge 작업을 수행해야 한다.

핫픽스 브랜치(hotfix branch)

핫픽스 브랜치는 **배포한 버전에서 긴급하게 수정**할 필요가 있을 때 master 브랜치에서 분리하는 브랜치를 말한다.



- **가지가 뻗어나오는 곳** : master
- **뻗어나갔던 가지가 다시 합쳐지는 곳** : develop, master
- **이름 설정** : hotfix-*
- 제품에서 버그가 발생했을 경우에는 처리를 위해 이 가지로 해당 정보들을 모아준다. 버그에 대한 수정이 완료된 후에는 develop, master에 곧장 반영해주며 tag를 통해 관련 정보를 기록해둔다.

버그를 잡는 사람이 일하는 동안에도 다른 사람들은 develop 브랜치에서 하던 일을 계속할 수 있다.

이 때 만든 hotfix 브랜치에서의 변경 사항은 develop 브랜치에도 merge 하여 문제가 되는 부분을 처리해줘야 한다.

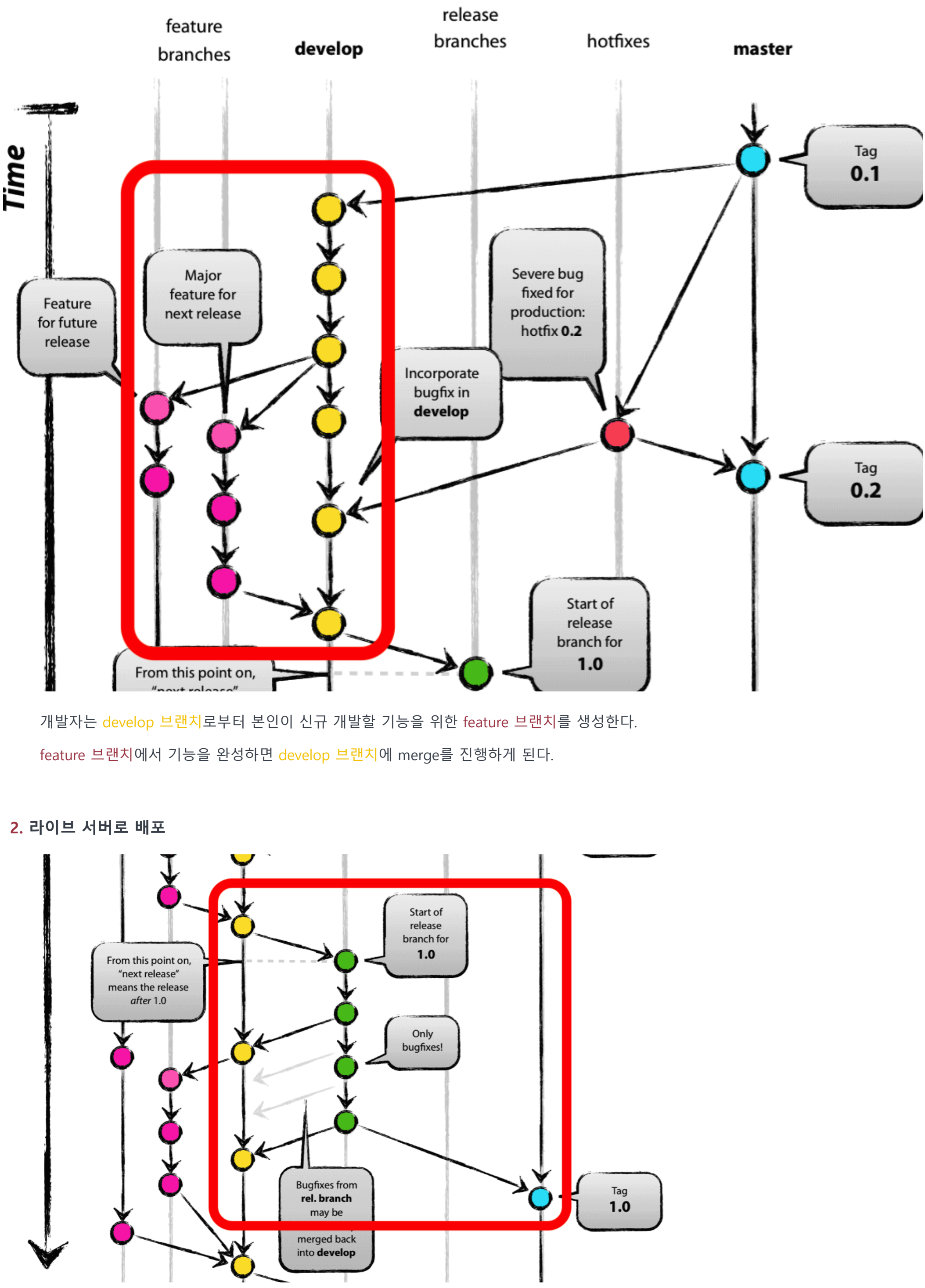
release 가지가 생성되어 관리되고 있는 상태라면 해당 가지에 hotfix정보를 병합시켜 다음번 배포 시 반영이 정상적으로 이루어질 수 있도록 해준다.

Hotfix는 보통 다급하게 버그를 고치기 위해 생성되는 가지이기 때문에 버그를 해결하면 보통 제거하는 일회성 가지다.

Git flow 흐름

- 앞에서 적었던 기본 구조 5개 중 가장 많이 사용되는 가지는 master와 develop가 되며 정상적인 프로젝트를 진행하기 위해서는 둘 모두를 운용해야 한다.
- 나머지 feature, release, hotfix branch는 사용하지 않는다면 지우더라도 오류가 발생하지 않기 때문에 깔끔한 프로젝트 진행을 원한다면 지워뒀다가 해당 가지를 활용해야 할 상황이 왔을 때 만들어줘도 괜찮다.
- 대부분의 작업은 develop에서 취합한다 생각하면 되며 테스트를 통해 정말 확실하게 더 이상 변동사항이 없다 싶을 때 master로의 병합을 진행하게 된다.
- master가 아닌 가지들은 master의 변동사항을 꾸준히 주시해야 한다.

1. 신규 기능 개발



개발자는 develop 브랜치로부터 본인이 신규 개발할 기능을 위한 feature 브랜치를 생성한다.

feature 브랜치에서 기능을 완성하면 develop 브랜치에 merge를 진행하게 된다.

2. 라이브 서버로 배포

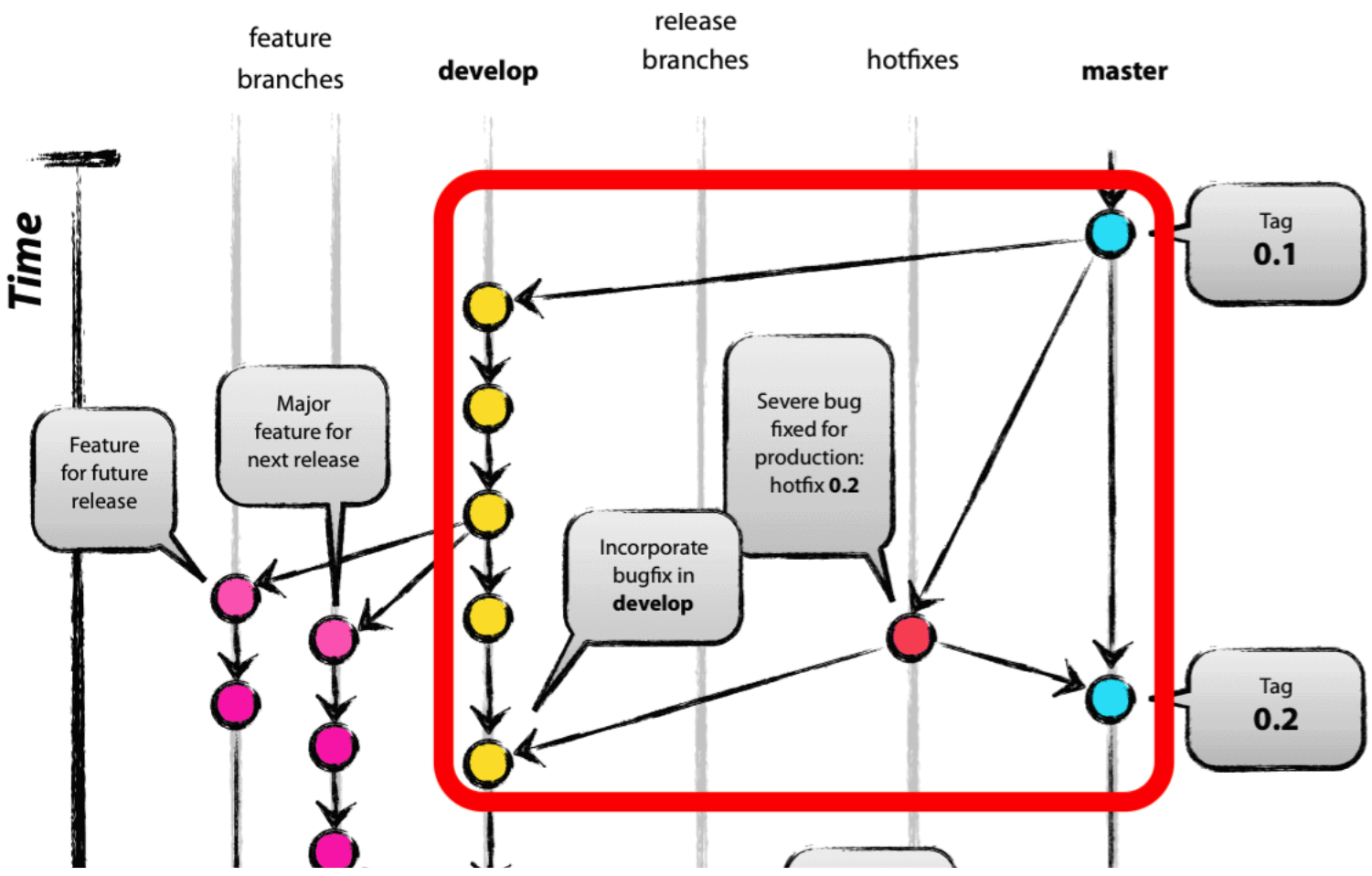
feature 브랜치들이 모두 develop 브랜치에 merge 되었다면 QA를 위해 release 브랜치를 생성한다.

release 브랜치를 통해 오류가 확인된다면 release 브랜치 내에서 수정을 진행한다.

QA와 테스트를 모두 통과했다면, 배포를 위해 release 브랜치를 master 브랜치 쪽으로 merge하며,

만일 release 브랜치 내부에서 오류 수정이 진행되었을 경우 동기화를 위해 develop 브랜치 쪽에도 merge를 진행한다.

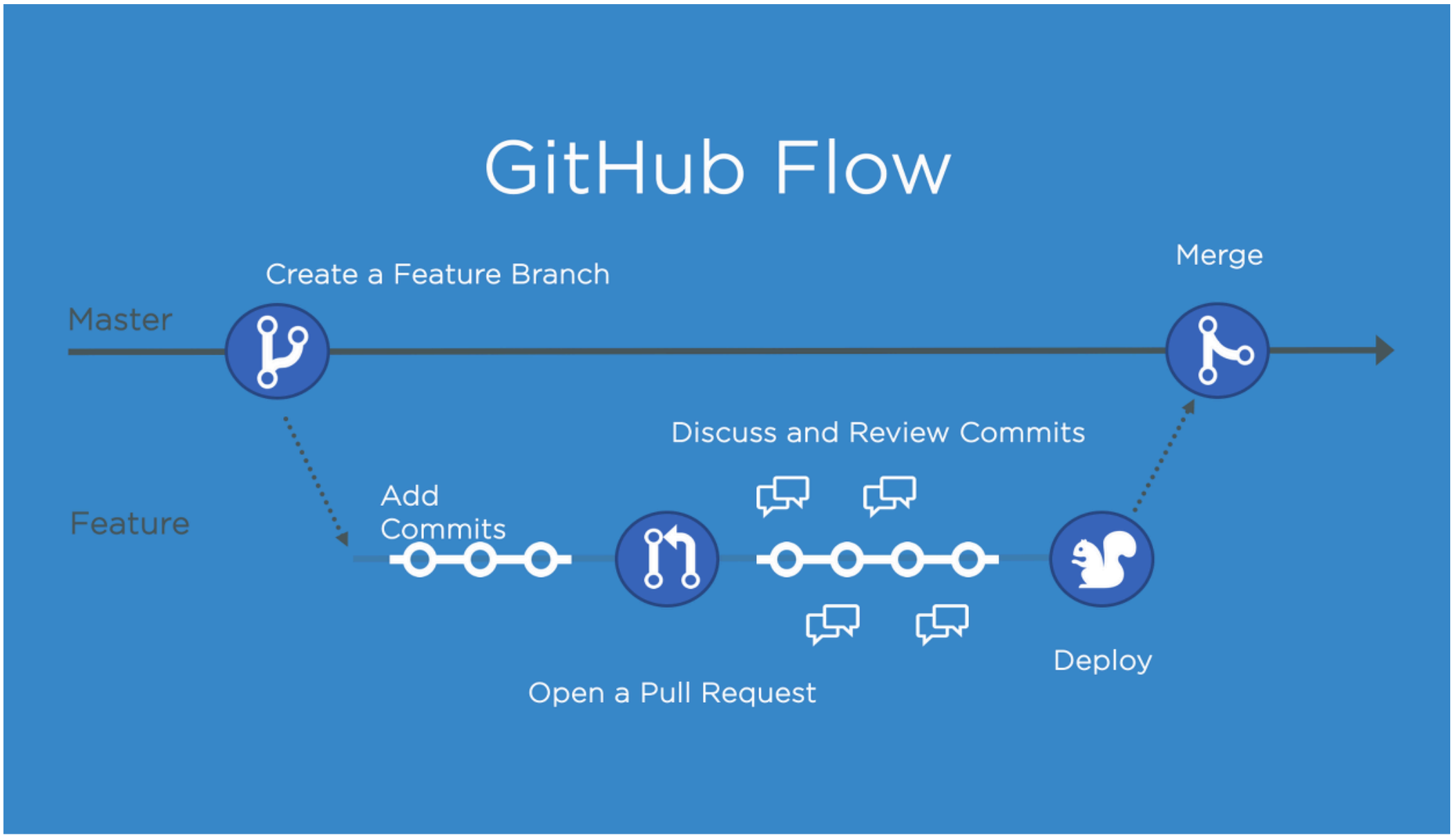
3. 배포 후 관리



만일 배포된 라이브 서버(master)에서 버그가 발생된다면, hotfix 브랜치를 생성하여 버그 픽스를 진행한다.

그리고 종료된 버그 픽스를 master와 develop 양 쪽에 merge하여 동기화 시킨다.

GITHUB-FLOW 전략



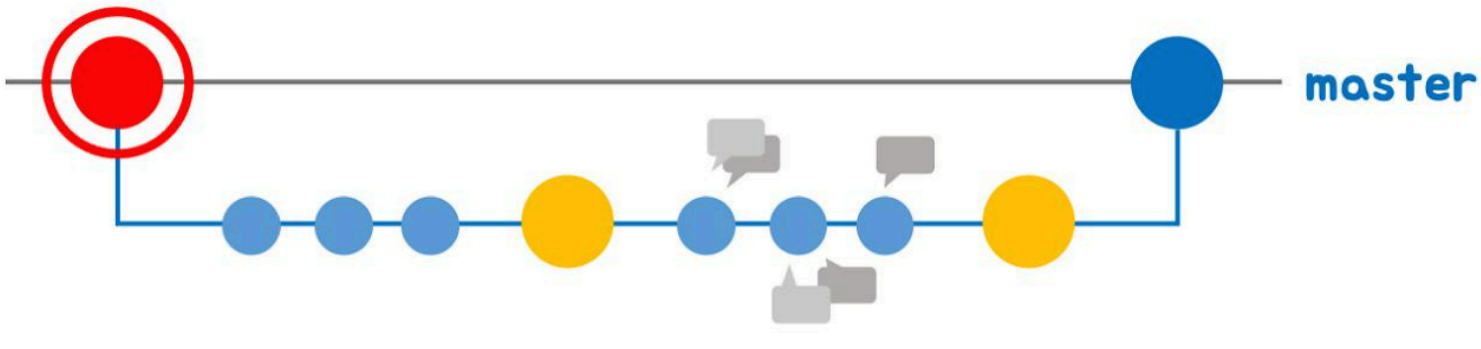
- Git flow가 좋은 방식이지만 GitHub에 적용하기에는 **복잡**하다는 Scott Chacon의 판단에 따라 만들어진 새로운 깃 관리 방식이다.
- **자동화 개념이 들어가 있다**라는 큰 특징이 존재하며 만일 자동화가 적용되어 있지 않은 곳에서만 수동으로 진행하면 된다.
- Git flow에 비해 흐름이 **단순해짐에 따라 그 규칙도 단순**해졌다.
- 기본적으로 master branch에 대한 규칙만 정확하게 정립되어 있다면 나머지 가지들에 대해서는 특별한 관여를 하지 않으며 pull request 기능을 사용하도록 권장한다.

GitHub-Flow 특징

- release branch가 명확하게 구분되지 않은 시스템에서의 사용이 유용하다.
- GitHub 자체의 서비스 특성상 배포의 개념이 없는 시스템으로 되어있기 때문에 이 flow가 유용하다.
- 웹 서비스들에 배포의 개념이 없어지고 있는 추세이기 때문에 앞으로도 Git flow에 비해 사용하기에 더 수월할 것이다.
- hotfix와 가장 작은 기능을 구분하지 않는다.
  모든 구분사항들도 결국 개발자가 전부 수정하는 일들 중 하나이기 때문이다.
  이 대신 구분하는 것은 우선 순위가 어떤 것이 더 높은지에 대한 것이다.

Github-Flow 흐름

1. 브랜치 생성

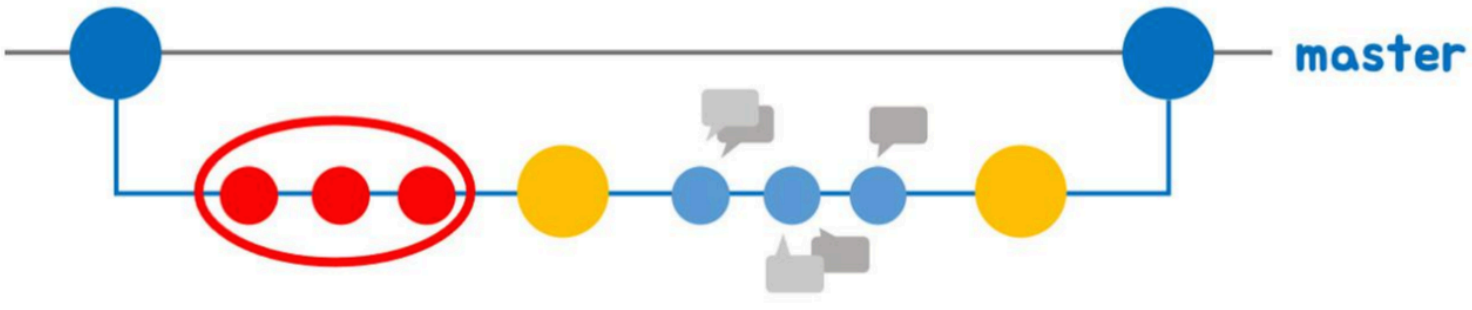


Github-flow 전략은 기능 개발, 버그 픽스 등 어떤 이유로든 **새로운 브랜치를 생성**하는 것으로 시작된다.

단, 이때 체계적인 분류 없이 브랜치 하나에 의존하게 되기 때문에 **브랜치 이름을 통해 의도를 명확하게** 드러내는 것이 매우 중요하다.

- master 브랜치는 항상 최신 상태며, stable 상태로 product에 배포되는 브랜치다. 이 브랜치에 대해서는 엄격한 role과 함께 사용한다
- **새로운 브랜치는 항상 master 브랜치에서 만든다**
- Git-flow와는 다르게 feature 브랜치나 develop 브랜치가 존재하지 않는다.
- 그렇지만, 새로운 기능을 추가하거나 버그를 해결하기 위한 **브랜치 이름은 자세하게** 어떤 일을 하고 있는지에 대해서 작성해주도록 하자

2. 개발 & 커밋 & 푸쉬

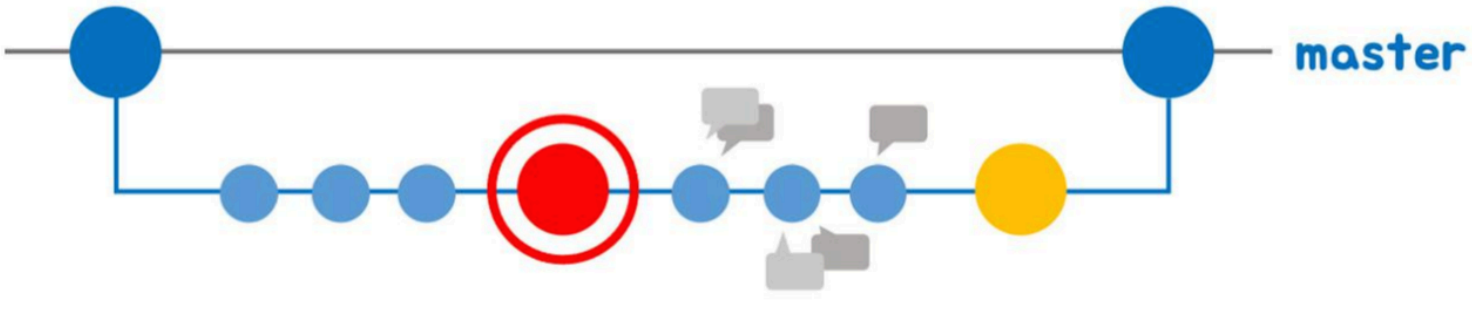


개발을 진행하면서 커밋을 남긴다.

이때도 브랜치와 같이 커밋 메세지에 의존해야 하기 때문에, 커밋 메세지를 최대한 상세하게 적어주는 것이 중요하다.

- **커밋메시지를 명확하게 작성하자**
- 원격지 브랜치로 **수시로 push** 하자
- Git-flow와 상반되는 방식
- 항상 원격지에 자신이 하고 있는 일들을 올려 다른 사람들도 확인할 수 있도록 해준다
- 이는 하드웨어에 문제가 발생해 작업하던 부분이 없어지더라도, 원격지에 있는 소스를 받아서 작업할 수 있도록 해준다

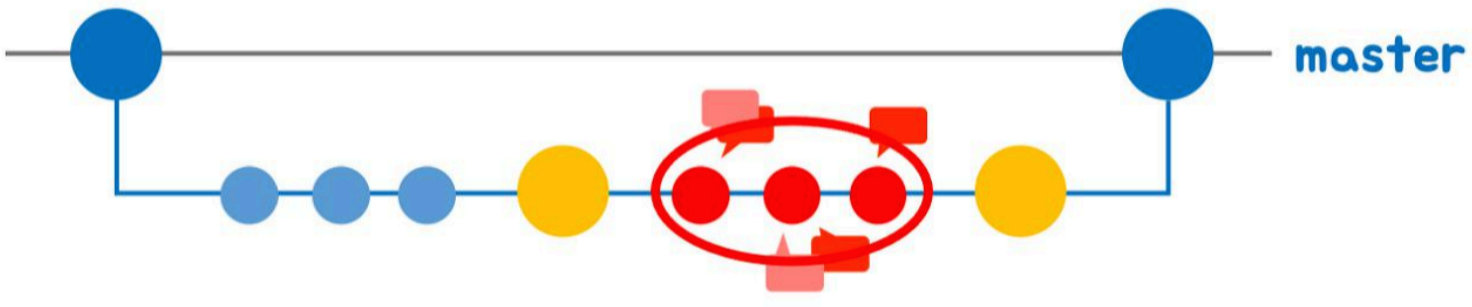
3. PR(Pull Request) 생성



피드백이나 도움이 필요할 때, 그리고 merge 준비가 완료되었을 때는 pull request를 생성한다

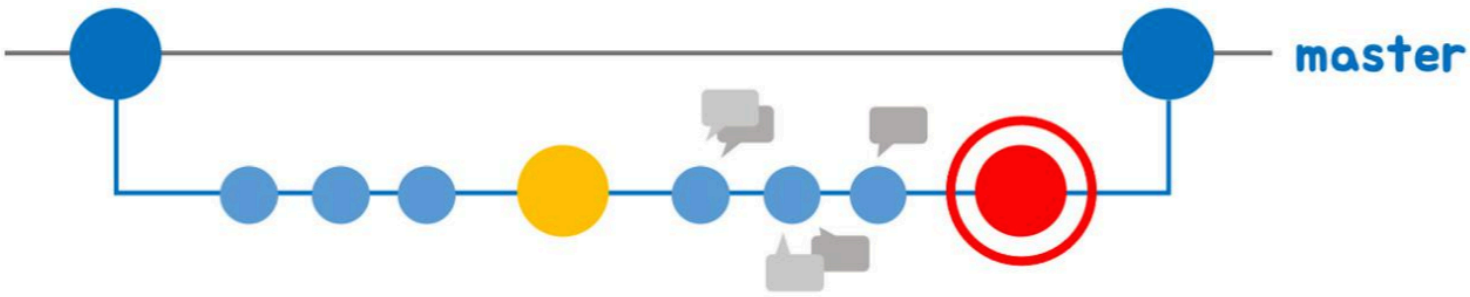
- pull request는 **코드 리뷰**를 도와주는 시스템
- 이것을 이용해 자신의 코드를 공유하고, 리뷰받는다.
- merge 준비가 완료되었다면 master 브랜치로 반영을 요구한다.

4. 리뷰 & 토의



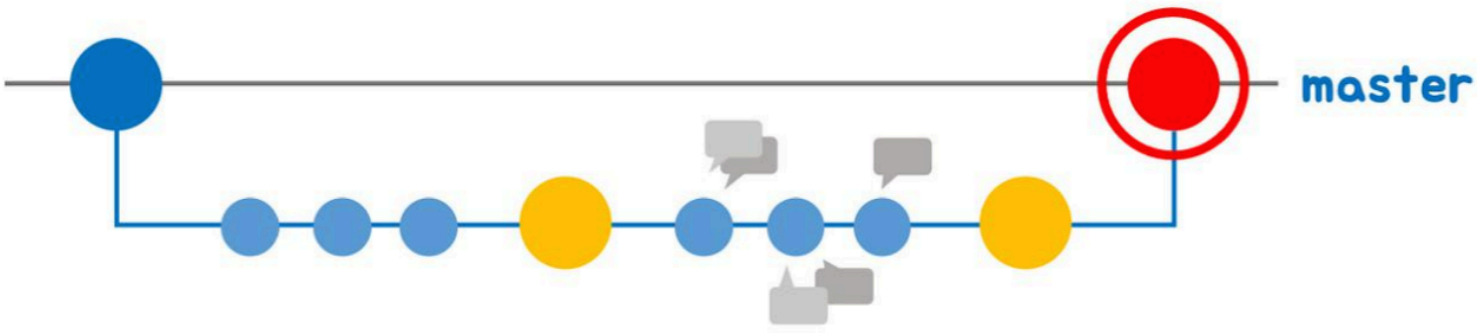
Pull-Request가 master 브랜치 쪽에 합쳐진다면 곧장 라이브 서버에 배포되는 것과 다름 없으므로, 상세한 리뷰와 토의가 이루어져야 한다.

5. 테스트



리뷰와 토의가 끝났다면 해당 내용을 라이브 서버(혹은 테스트 환경)에 배포해본다.
배포시 문제가 발생한다면 곧장 master 브랜치의 내용을 다시 배포하여 초기화 시킨다.

6. 최종 Merge

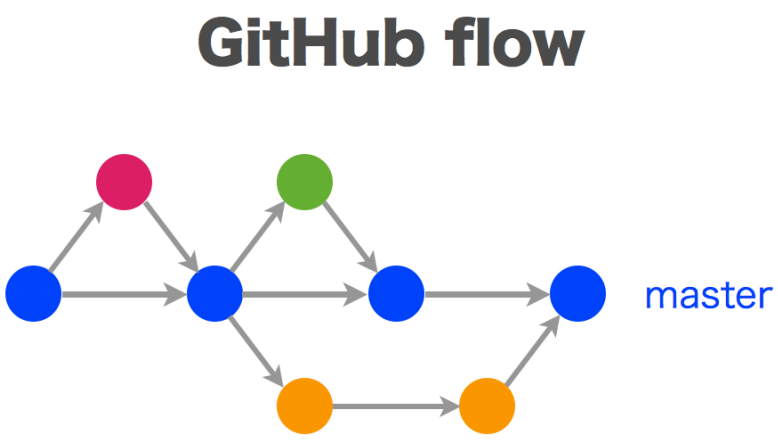


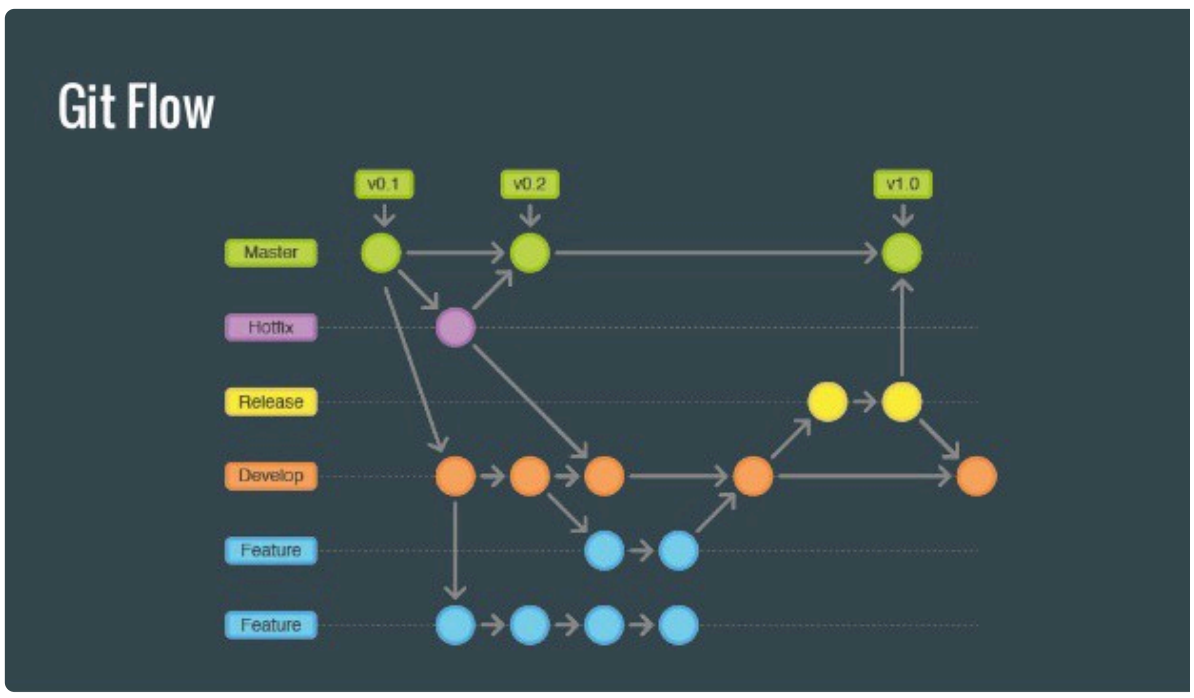
라이브 서버(혹은 테스트 환경)에 배포했음에도 문제가 발견되지 않았다면 **그대로 master 브랜치에 푸시를 하고, 즉시 배포**를 진행한다.
대부분의 Github-flow 에선 master 브랜치를 최신 브랜치라고 가정하기 때문에 배포 자동화 도구를 이용해서 Merge 즉시 배포를 시킨다.
master로 merge되고 push 되었을 때는, 즉시 배포되어야한다

- GitHub-flow의 핵심
- master로 merge가 일어나면 자동으로 배포가 되도록 설정해놓는다. (CI / CD)

---

github flow vs git flow





- 1개월 이상의 긴 호흡으로 개발하여 주기적으로 배포, QA 및 테스트, hotfix 등 수행할 수 있는 여력이 있는 팀이라면 **git-flow**가 적합하다
- 수시로 릴리즈 되어야 할 필요가 있는 서비스를 지속적으로 테스트하고 배포하는 팀이라면 **github-flow** 와 같은 간단한 work-flow가 적합하다