

Mash-up

MAASH-UP

mash-up 뽀수기

발표 : 김유정



GitHub?

소프트웨어 개발 프로젝트를 위한 소스코드 관리 서비스

commit

push

pull

branch

checkout

commit : 파일을 추가하거나 변경내용을 저장소에 저장하는 작업

push : 파일을 추가하거나 변경 내용을 원격 저장소에 업로드하는 작업

pull : 원격 저장소에서의 코드를 가져오는 작업

branch : 병렬관계로 버전관리를 위해 사용

checkout : branch의 이동을 위해 사용

commit

git commit -m “짧은 메시지”

git commit

```
<type>: <subject>  
<body>  
<footer>
```

```
refactor: Commit Msg
```

– 변경사항 //명령형이나 현재시제를 사용한다.

Fixes ~ //이슈를 자동으로 닫아준다.

git commit -m “짧은 메시지”

git commit

```
<type>: <subject>  
<body>  
<footer>
```

첫 번째 줄 메시지 type

- refactor : 새로운 기능이 아닌 코드를 재정비 하였을 때 (변수 이름 바꾸기)
- fix : 빌드 스크립트의 버그를 수정
- feat : 빌드 스크립트의 기능을 위한 것이 아니라 사용자의 편의를 위한 새로운 기능
- chore : 새로운 기능을 추가
- docs : 문서 변경
- test : 누락된 테스트 추가, 생산적인 코드 생성은 없음

git commit -m "짧은 메시지"

git commit

```
feat: empty input when sending the text in chat ...  
- writeInputEditText의 전송 후 안의 text를 제거한다.  
- 입력한 글이 없을 경우에는 글을 보내지 못하게 한다.  
- 채팅창의 리사이클러 뷰가 잘리지 않도록 수정한다.
```

```
Fixes chatty-app/chatty-app#92
```

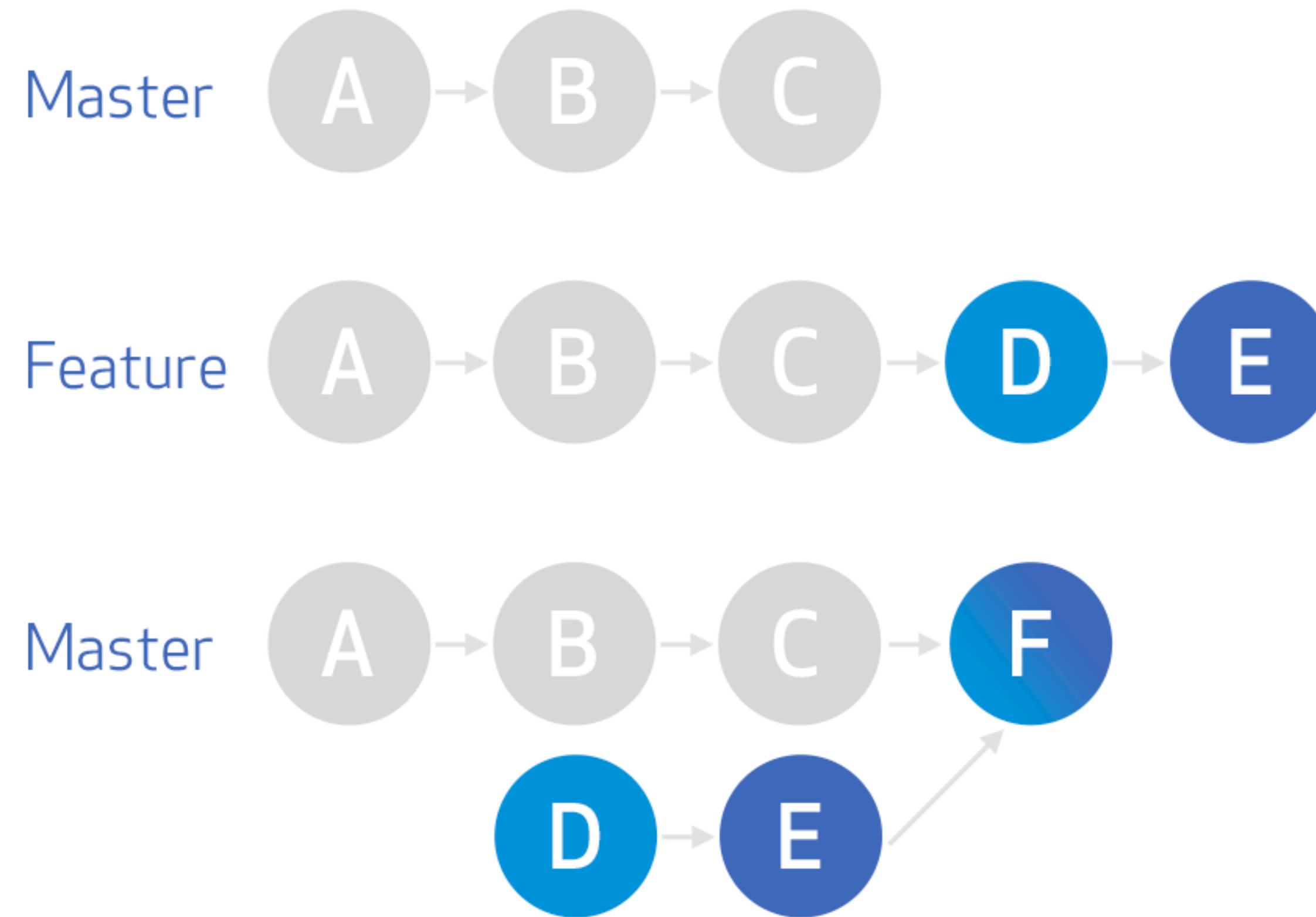
Merge commit

Squash commit

Rebase

Merge commit

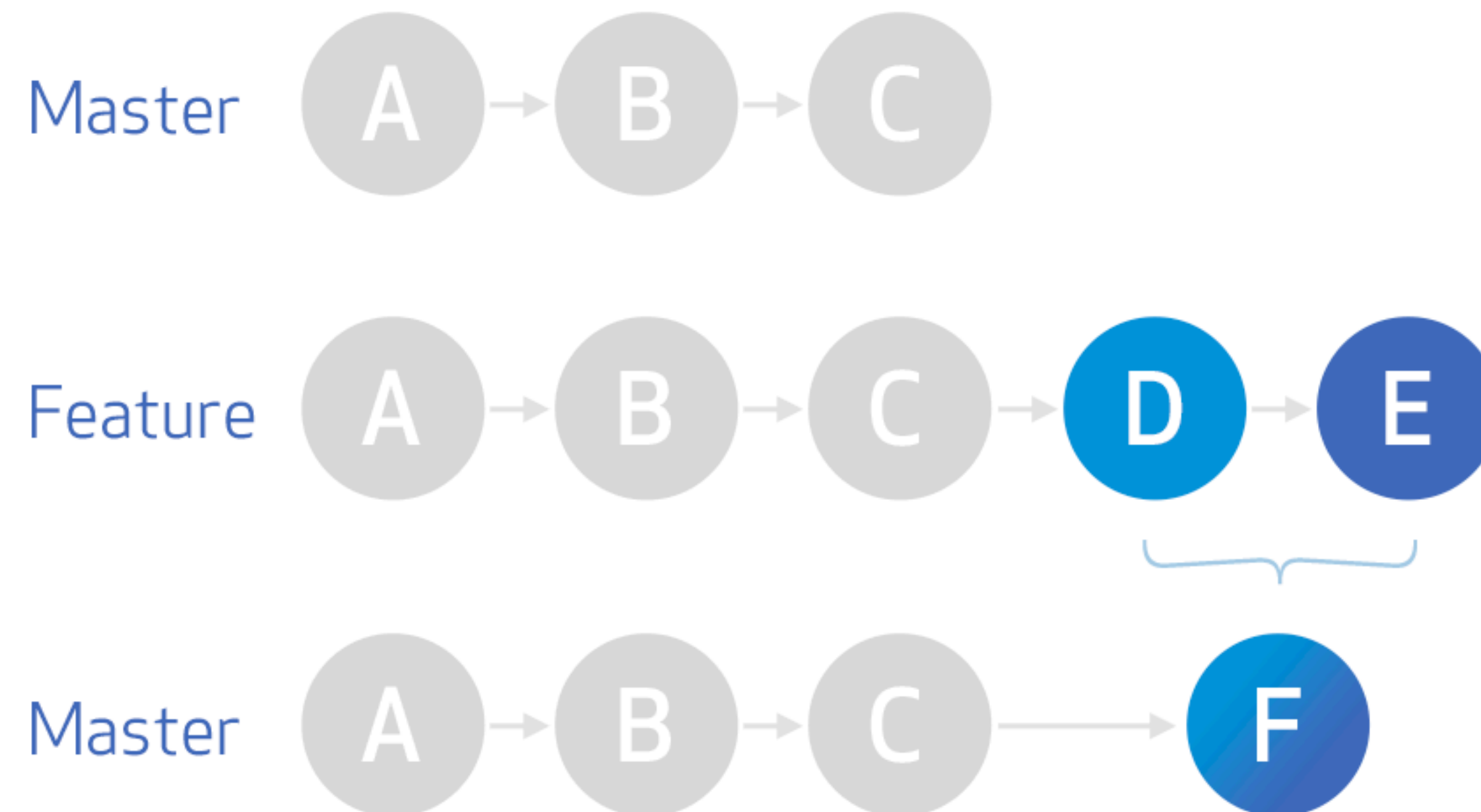
: 브랜치에서 모든 커밋을 유지하고 기본 브랜치에서 커밋으로 인터리브



Merge commit: D + E added to Master via F

Squash commit

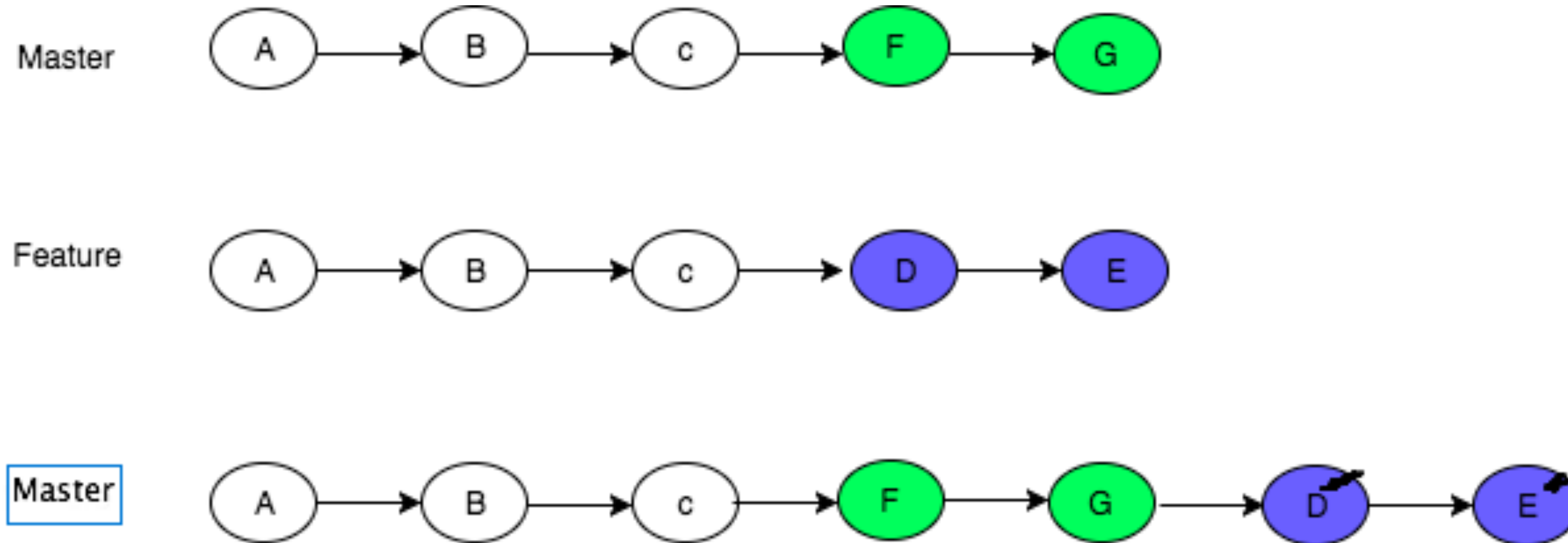
: 변경 사항은 유지하지만 기록에서 개별 커밋을 생략합니다.



Squash and merge: D + E combined into F

Rebase

: 전체 기능 분기가 마스터 분기의 끝에서 시작하여 마스터에 모든 새로운 커밋을 효과적으로 통합합니다.



barnch

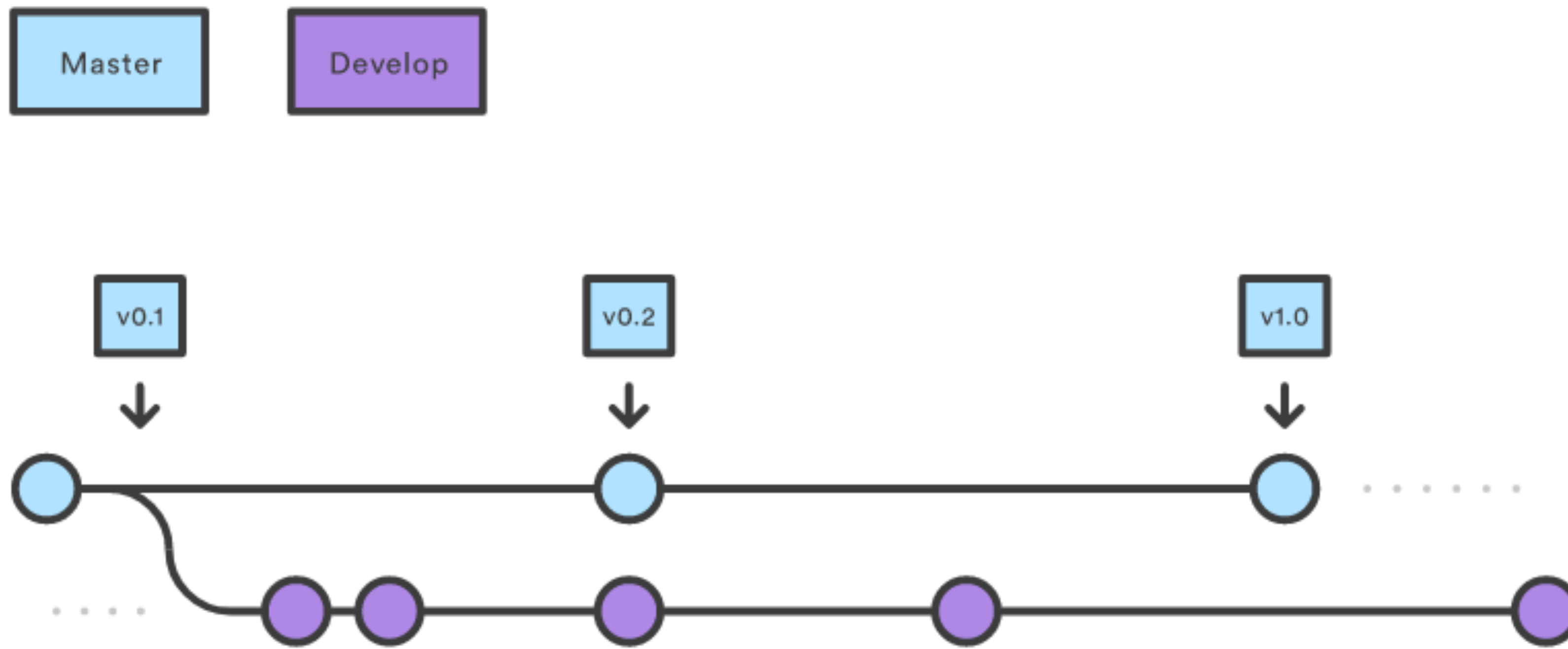
Git Branch 종류 (5가지)

항상 유지되는 메인 브랜치들(master, develop)

일정 기간 동안만 유지되는 보조 브랜치들(feature, release, hotfix)

Master Branch

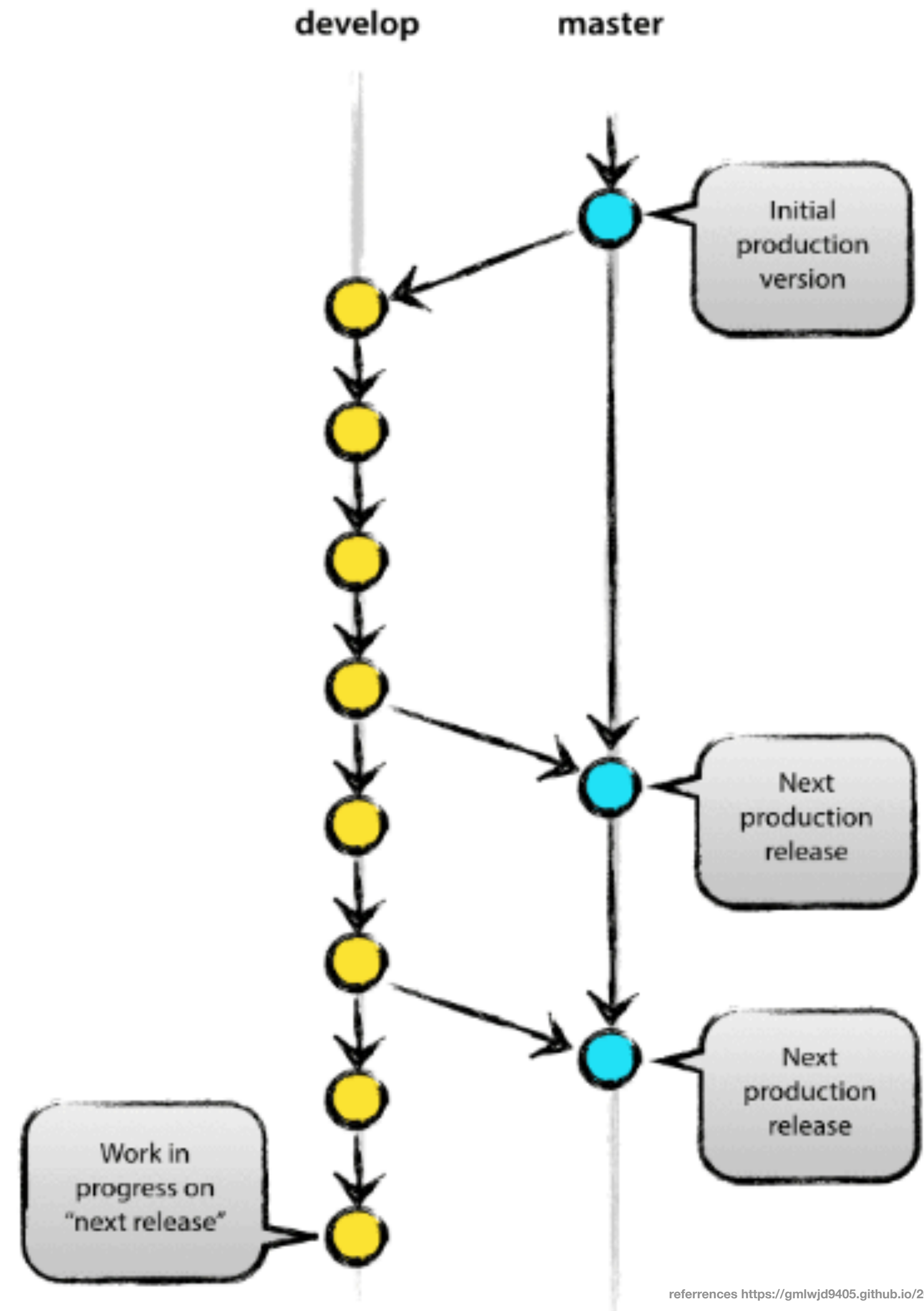
제품으로 출시될 수 있는 브랜치
배포(Release) 이력을 관리하기 위해 사용. 즉, 배포 가능한 상태만을 관리한다.



Develop Branch

다음 출시 버전을 개발하는 브랜치

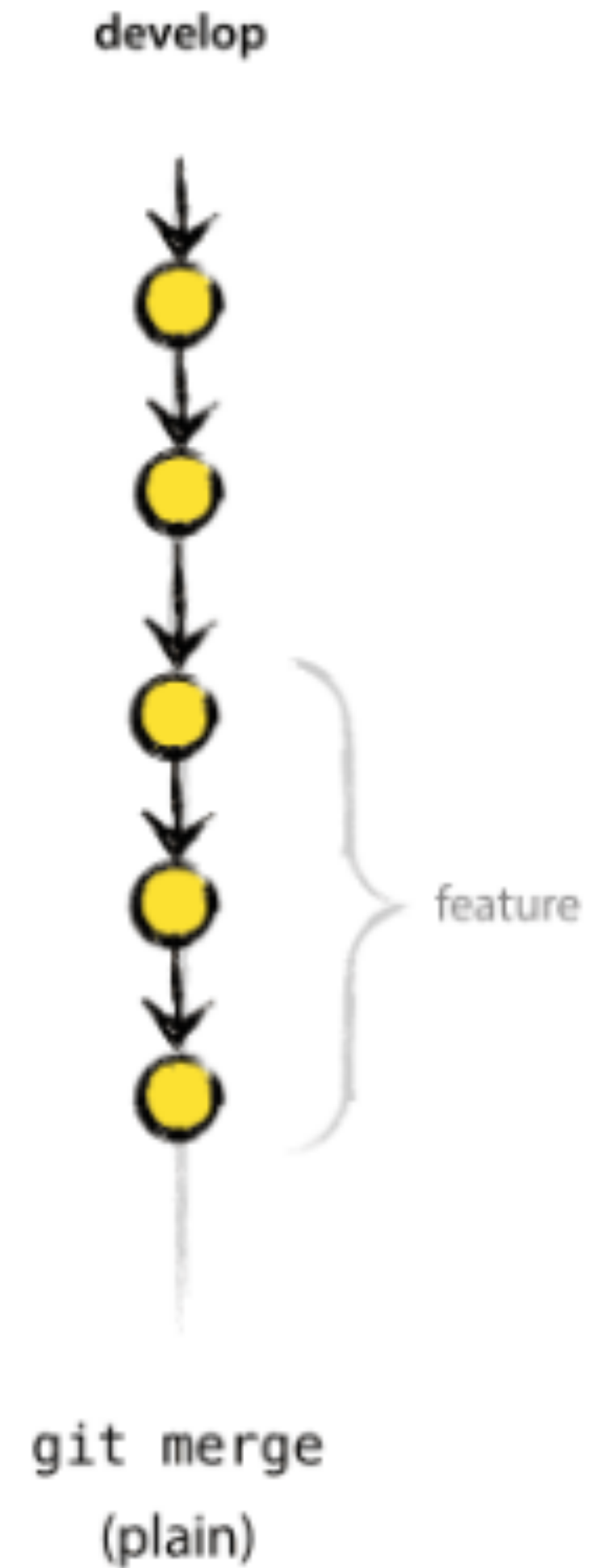
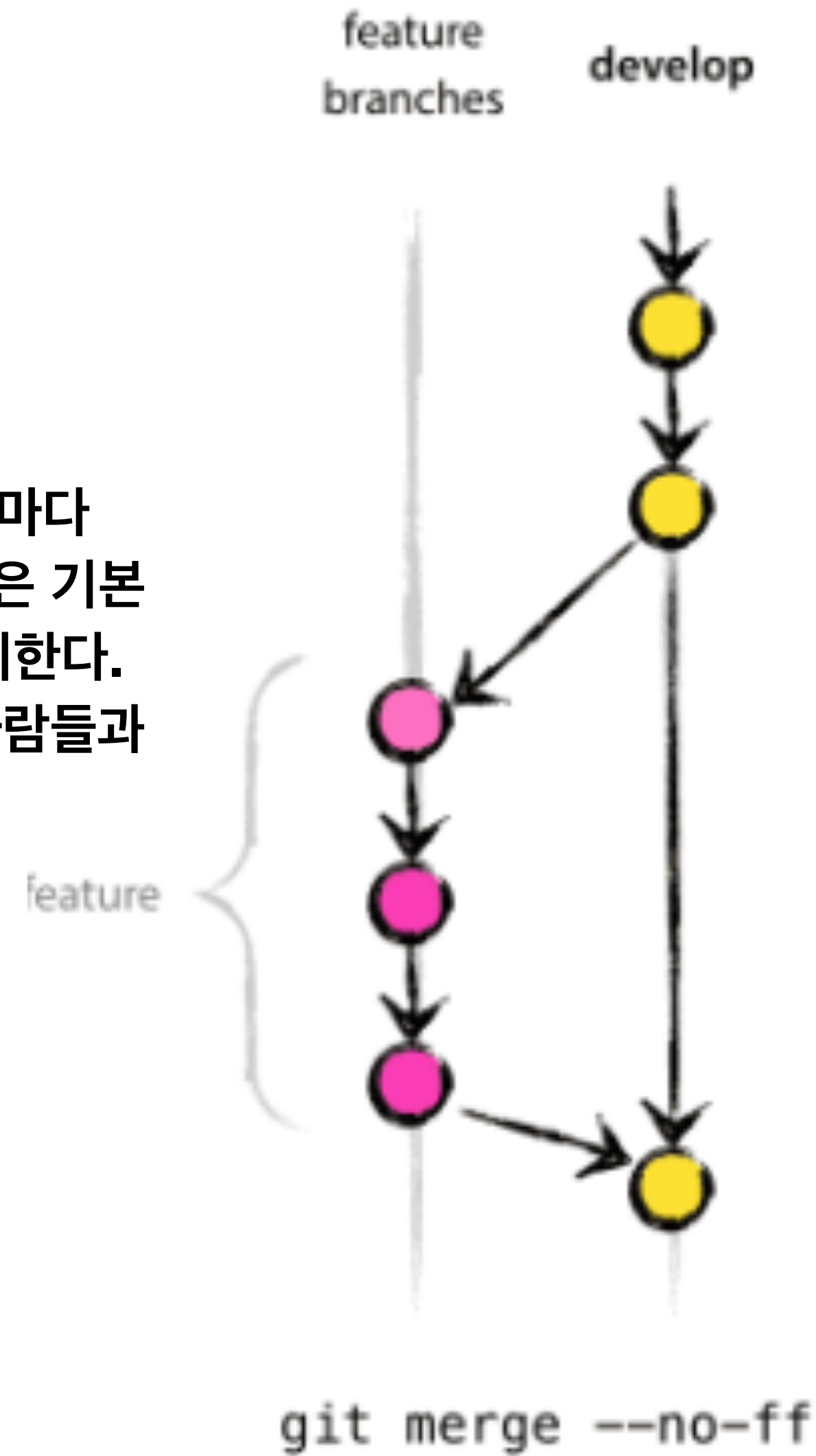
기능 개발을 위한 브랜치들을 병합하기 위해 사용. 즉, 모든 기능이 추가되고 버그가 수정되어 배포 가능한 안정적인 상태라면 **develop** 브랜치를 'master' 브랜치에 병합(merge)한다. 평소에는 이 브랜치를 기반으로 개발을 진행한다.



Feature Branch

기능을 개발하는 브랜치

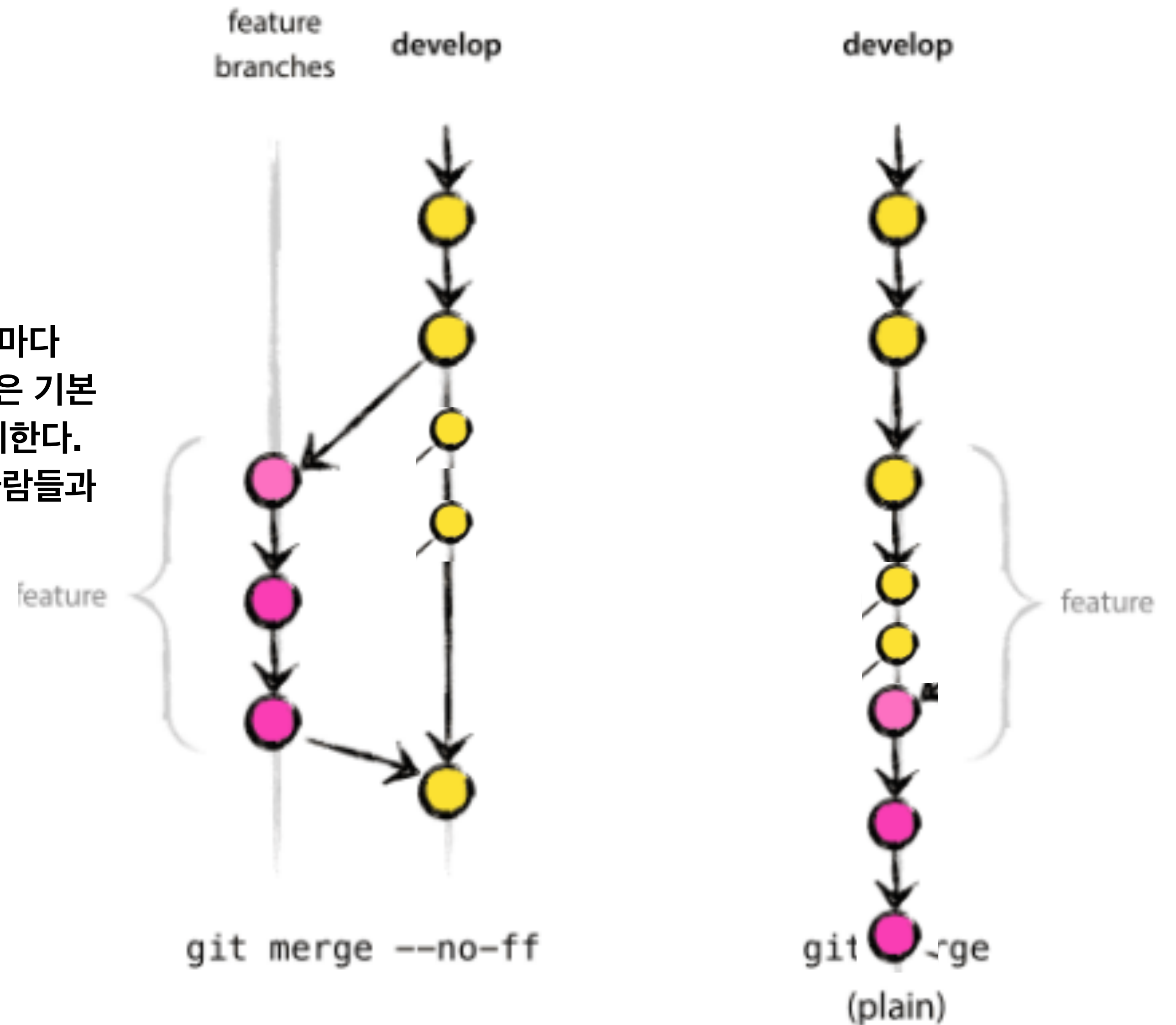
feature 브랜치는 새로운 기능 개발 및 버그 수정이 필요할 때마다 'develop' 브랜치로부터 분기한다. feature 브랜치에서의 작업은 기본적으로 공유할 필요가 없기 때문에, 자신의 로컬 저장소에서 관리한다. 개발이 완료되면 'develop' 브랜치로 병합(merge)하여 다른 사람들과 공유한다.



Feature Branch

기능을 개발하는 브랜치

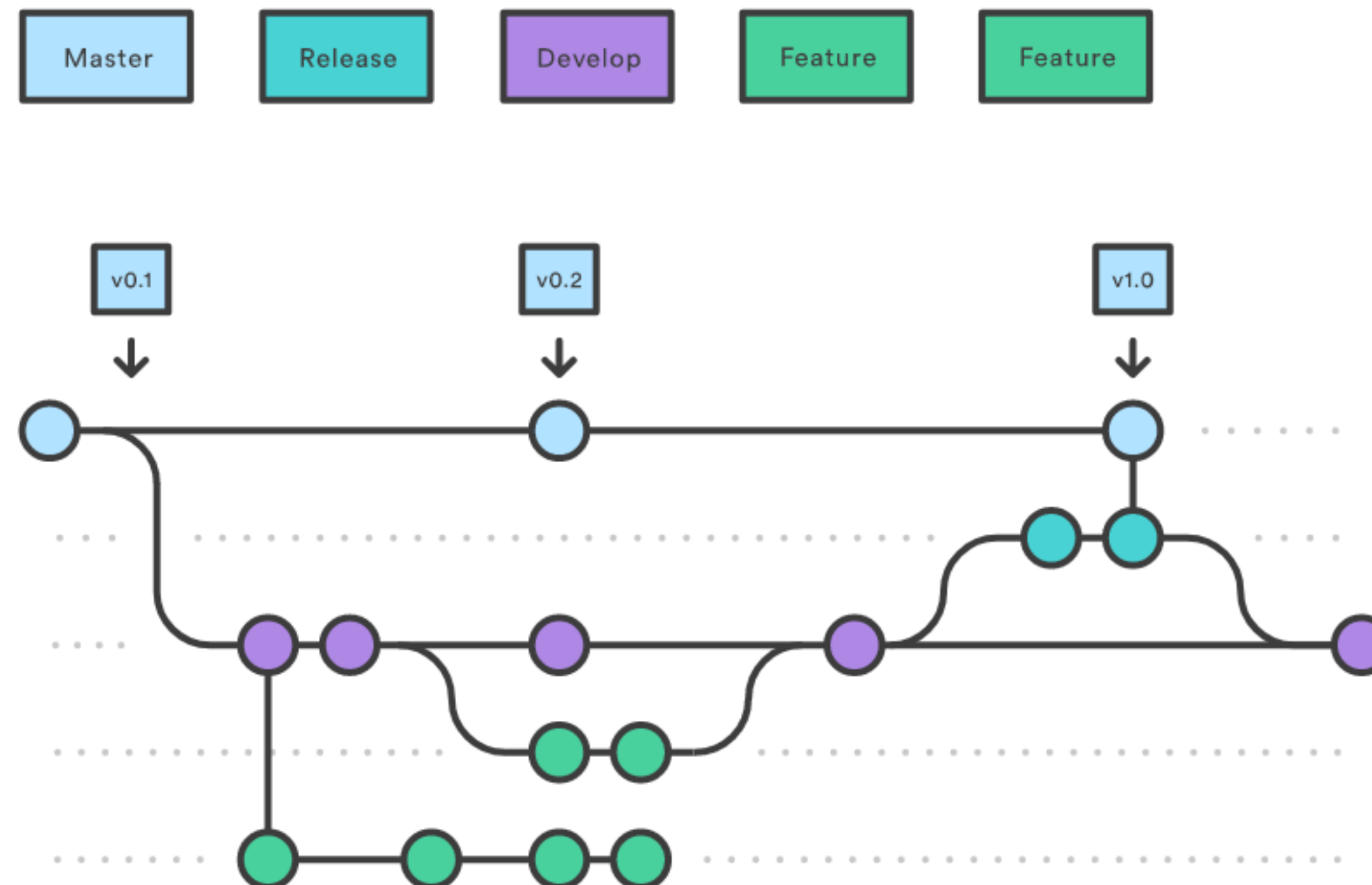
feature 브랜치는 새로운 기능 개발 및 버그 수정이 필요할 때마다 'develop' 브랜치로부터 분기한다. feature 브랜치에서의 작업은 기본적으로 공유할 필요가 없기 때문에, 자신의 로컬 저장소에서 관리한다. 개발이 완료되면 'develop' 브랜치로 병합(merge)하여 다른 사람들과 공유한다.



Release Branch

이번 출시 버전을 준비하는 브랜치

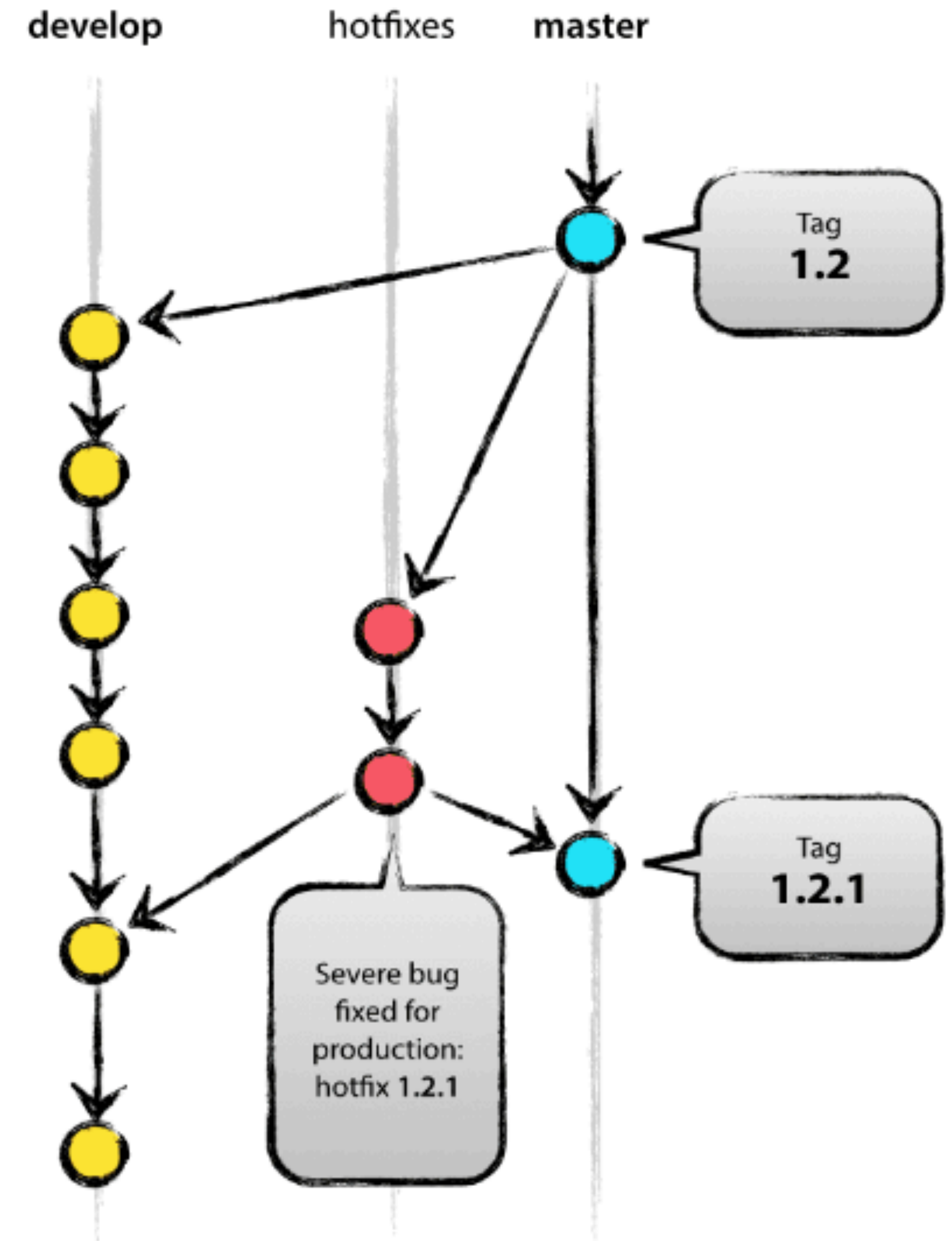
release 브랜치에서는 배포를 위한 최종적인 버그 수정, 문서 추가 등 릴리스와 직접적으로 관련된 작업을 수행
직접적으로 관련된 작업들을 제외하고는 release 브랜치에 새로운 기능을 추가로 병합(merge)하지 않는다.
‘release’ 브랜치에서 배포 준비가 완료되면 ‘master’ 브랜치에 병합한다. (이때, 병합한 커밋에 Release 버전 태그를 부여!)
배포 완료 후 ‘develop’ 브랜치에도 병합한다.
이때, 다음 번 배포(Release)를 위한 개발 작업은 ‘develop’ 브랜치에서 계속 진행해 나간다.



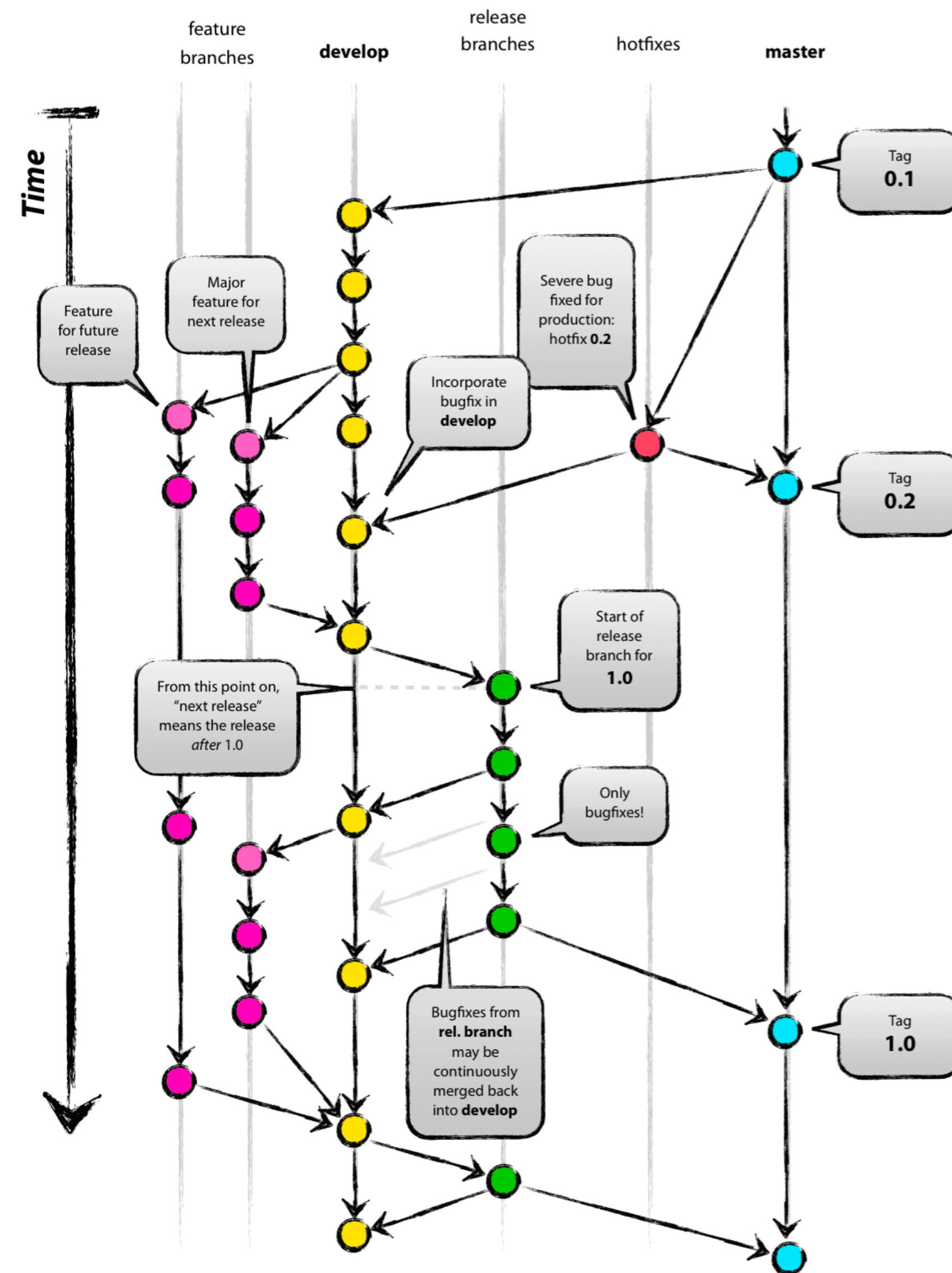
Hotfix Branch

출시 버전에서 발생한 버그를 수정 하는 브랜치

배포한 버전에 긴급하게 수정을 해야 할 필요가 있을 경우, 'master' 브랜치에서 분기하는 브랜치이다. 'develop' 브랜치에서 문제가 되는 부분을 수정하여 배포 가능한 버전을 만들기에는 시간도 많이 소요되고 안정성을 보장하기도 어려우므로 바로 배포가 가능한 'master' 브랜치에서 직접 브랜치를 만들어 필요한 부분만을 수정한 후 다시 'master' 브랜치에 병합하여 이를 배포



git flow




upstream origin downstream




upstream origin downstream


실습으로 배워봅시다


https://github.com/YuChocopie/mashup-android




[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)


 **YuChocopie** / **mashup-android**

 Unwatch

1


 Star


0


 Fork

0

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

 master


 1 branch

 0 tags





[Go to file](#)

[Add file](#)

[Code](#)


 **YuChocopie** rename


3511daf 2 days ago ⌚ 8 commits


 @types	first commit	2 days ago
 content	update	2 days ago
 src	rename	2 days ago
 static	first commit	2 days ago

About

매쉬업 안드로이드팀 블로그

 [mashup-android.yuchocopie...](#)

 [Readme](#)

 [MIT License](#)

Releases

YuChocopie / mashup-android

Unwatch

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insig

master

1 branch

0 tags

YuChocopie rename

@types	first commit
content	update
src	rename
static	first commit
.gitignore	first commit
.prettierrignore	first commit

Go to file

Add file

Code

Clone

HTTPS SSH GitHub CLI

https://github.com/YuChocopie/mashu

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

2 days ago

```
git clone https://github.com/YuChocopie/mashup-android.git
```

`git clone https://github.com/YuChocopie/mashup-android.git`
`git remote -v`
`git remote rename origin upstream`
`git remote add downstream 본인이 fork 한 repo`
`git remote -v`

```
kim-yujeong@gin-yujeong-ui-MacBook-Pro-2 ~/work/mash-up10th/AndroidB
log > master git remote -v
origin https://github.com/YuChocopie/mashup-android.git (fetch)
origin https://github.com/YuChocopie/mashup-android.git (push)
kim-yujeong@gin-yujeong-ui-MacBook-Pro-2 ~/work/mash-up10th/AndroidB
log > master git remote rename origin upstream
kim-yujeong@gin-yujeong-ui-MacBook-Pro-2 ~/work/mash-up10th/AndroidB
log > master git remote -v
upstream https://github.com/YuChocopie/mashup-android.git (fetch)
upstream https://github.com/YuChocopie/mashup-android.git (push)
```

upstream : <https://github.com/YuChocopie/mashup-android>

downstream : <https://github.com/본인git/mashup-android>

mashup-android / content / blog /

**conetne > blog > 안에 본인의 이름(원래는 제목) 폴더를 만들고
그 안에 양식대로 글을 채워넣기!**

dev 브랜치에서 진행해주세요!

git status


git add .

git commit

git checkout -b feature-yuchocopie-글제목

? git commit










[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)




feature-yuchocopie-글제목


Write

Preview

H B I         

- 테스트용 이슈

Attach files by dragging & dropping, selecting or pasting them. 

 Styling with Markdown is supported

Submit new issue

git commit -m “짧은 메시지”

git commit

```
<type>: <subject>  
<body>  
<footer>
```

```
refactor: Commit Msg
```

– 변경사항 //명령형이나 현재시제를 사용한다.

Fixes ~ //이슈를 자동으로 닫아준다.

git commit -m “짧은 메시지”

```
docs: Add Test Blog ,,,,
```

```
- 블로그에 올릴 테스트 글을 생성하다.
```

```
█
```

```
Fixed #2
```

```
# 변경 사항에 대한 커밋 메시지를 입력하십시오. '#'
```

```
# 줄은 무시되고, 메시지를 입력하지 않으면 커밋이 중
```

```
#
```

```
# 현재 브랜치 feature-yuchocopie-글제목
```

```
# 커밋할 변경 사항:
```

```
# 새 파일: content/blog/yuchoco1/ic_and
```

- feat : 빌드 스크립트의 기능을 위한 것이 아니라 사용자의 편의를 위한 새로운 기능
- chore : 새로운 기능을 추가
- docs : 문서 변경
- test : 누락된 테스트 추가, 생산적인 코드 생성은 없음

git commit -m "짧은 메시지"

git commit

```
feat: empty input when sending the text in chat ...  
- writeInputEditText의 전송 후 안의 text를 제거한다.  
- 입력한 글이 없을 경우에는 글을 보내지 못하게 한다.  
- 채팅창의 리사이클러 뷰가 잘리지 않도록 수정한다.
```

```
Fixes chatty-app/chatty-app#92
```

**commit 을 생성해
upstream 으로 push 해봅시다!**

git push upstream dev

안되는게 정상입니다!

(충돌 나야함)

git pull upstream dev - -rebase

를 하게 되면 이제 push 할 수는 있을 텐데요...

하지만 저희는 feature 별로 나누어 pr을 보내 합칠겁니다!
issue 에서 "feature-name-title" 로 이슈를 생성해주세요

git checkout -b feature-name-title

그 리 고

downstream 으로 push 해봅시다!

git push downstream feature-name-title

이제 PR 을 보내고~

다된것같쥬.. ?

