

Git – 07

목차


- Part 3. Git의 다양한 활용 방법
 - Chapter 10. Git 고급
 - ~~▪ Chapter 11. Git GUI~~
 - Chapter 12. GitHub로 협업하기
- 생략 내용
 - Part 2. 네 가지 주요 IDE의 Git 활용
 - 이클립스, VS, IntelliJ IDEA, Xcode
 - MAC OS X 관련 내용
 - part 3. Git GUI
 - 소스트리에서 깃 사용으로, GUI 부분 제외하면 part 1과 같음.


Chapter 12. GitHub로 협업하기


- GitHub의 협업 도구
 - 이슈 트래커
 - 위키
 - 풀 리퀘스트
 - GitHub에서의 코드 리뷰
- 프로젝트를 위한 협업 준비 규칙
 - 커밋 단위
 - 커밋 메시지 작성 규칙
 - 브랜치 이름 작성 규칙
 - 태그와 버전 이름 작성 규칙
- 프로젝트 유형별 협업 흐름
 - git-flow : 게임이나 SI 개발 환경에 권장
 - github-flow : 웹 어플리케이션
 - gitlab-flow : 모바일 앱과 게임


GitHub의 협업 도구

- 이슈 트래커
 - <https://github.com/frabcus/house/>
 - 이슈 트래커는 쉽게 말하자면 게시판이다.
 - 버그 보고, 기능 개선 건의, 그 외 프로젝트에 관련된 주제를 등록할 수 있는 공간
 - 일반적인 게시판과의 차이
 - 담당자 : issue 담당자 지정 가능
 - 알림 : @<name> 형식으로 특정 그룹이나 특정 사용자에게 알림
 - 라벨 : 카테고리 역할의 라벨 지정 가능
 - 커밋 레퍼런스 : 커밋 해시를 써두면 자동으로 해당 커밋에 링크
 - 마일스톤 : 이슈들을 그룹으로 만드는 표식을 지정

 **Support only last n (3?) iOS versions in jQuery 4.0+** Docs 7

#3950 opened 20 days ago by mgol  4.0.0

 **Infrastructure for the grunt commands & check their work** Blocker Build Docs 2

#3922 opened on Jan 5 by markelog  3.4.0

- issue number #3950, label Docs, milestone 4.0.0

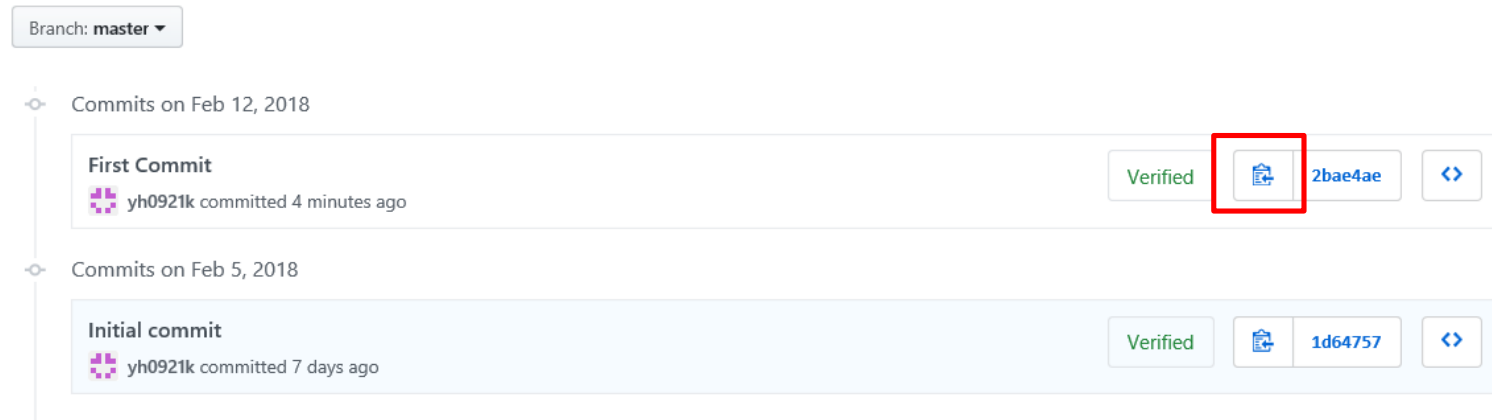
GitHub의 협업 도구

- 이슈 트래커
 - GitHub 원격 저장소 > Issues

The screenshot shows the GitHub interface for the repository 'yh0921k / NetSys'. At the top, there are buttons for 'Watch', 'Star', and 'Fork', each with a count of 0. Below these are navigation tabs: 'Code', 'Issues' (which is highlighted with an orange bar and shows 0 issues), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. A message box states: 'Label issues and pull requests for new contributors. Now, GitHub will help potential first-time contributors discover issues labeled with **help wanted** or **good first issue**.' Below this is a search bar with the filter 'is:issue is:open' and buttons for 'Labels' and 'Milestones'. A green 'New issue' button is on the right. The main content area has a large question mark icon and the text 'Welcome to Issues!'. It explains that issues are used to track todos, bugs, feature requests, and more, and that they will appear in a searchable and filterable list. It concludes with the instruction: 'To get started, you should [create an issue](#).'

GitHub의 협업 도구

- 이슈 트래커
 - New issue
 - Title : First Issue
- 이슈는 commit 내역을 참조할 때 의미가 있다.
 - 저장소의 commit 내역에서 아래 버튼으로 해당 commit의 SHA-1 값을 복사 가능
 - 이를 이슈 작성 화면에서 붙여 넣는다.



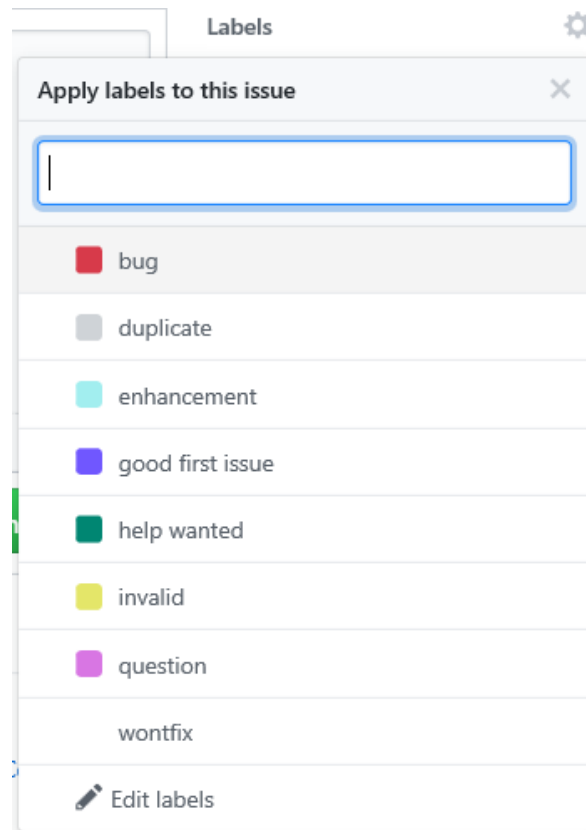
GitHub의 협업 도구

- 이슈 트래커
 - 이슈를 최종 등록했을때 자동으로 해당 커밋 내역의 링크를 표시한다.
 - 다음으로 우측의 Labels, Milestone, Assignee 메뉴가 있다.

The screenshot shows the GitHub interface for creating a new issue in the 'NetSys' repository by user 'yh0921k'. The top navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. On the right, there are buttons for Watch (0), Star (0), and Fork (0). The main content area is titled 'First Issue' and has a 'Write' tab selected. The issue body contains the text 'hello' followed by a commit hash '2bae4ae898d8e363cc12dff8df87e0dee14701d4'. Below the text area, there is a note: 'Attach files by dragging & dropping or selecting them.' At the bottom right, there is a green button labeled 'Submit new issue'. On the right side of the page, there are sections for 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), and 'Milestone' (No milestone), each with a settings gear icon.

GitHub의 협업 도구

- 이슈 트래커
 - Labels 설정에서는 개발상에서 발생할 수 있는 상황들의 키워드로 미리 생성된 라벨이 있다.
 - 여기서는 bug와 help wanted 두 가지를 선택한다.



GitHub의 협업 도구

- 이슈 트래커
 - 새로운 라벨을 생성하고 싶다면, issue의 메인 화면에서 label 탭을 선택한 후 새로운 라벨 이름과 색상을 지정하고 생성한다.

yh0921k / NetSys

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Labels Milestones New label

8 labels Sort ▼

bug	Edit Delete
duplicate	Edit Delete
enhancement	Edit Delete
good first issue	Edit Delete
help wanted	Edit Delete
invalid	Edit Delete
question	Edit Delete
wontfix	Edit Delete

GitHub의 협업 도구

- 이슈 트래커
 - 마일스톤
 - 한 이슈에는 하나의 마일스톤만 할당할 수 있다.
 - 같은 마일스톤을 지정한 이슈가 해결될 때마다 막대의 진행률이 올라간다.
 - 담당자 지정
 - 이슈를 처리하는 권한이 있는 담당자를 지정할 수 있다.
 - 톱니바퀴 버튼을 누르면 현재 저장소의 공헌자 중 한 명을 선택해서 지정할 수 있다.

First Issue

Write Preview

hello

2bae4ae898d8e363cc12dff8df87e0dee14701d4

Attach files by dragging & dropping or [selecting them](#).

Styling with Markdown is supported

Submit new issue

Assignees

yh0921k

Labels

bug

help wanted

Projects

None yet

Milestone

First Release

GitHub의 협업 도구

- 이슈 트래커

yh0921k / NetSys

Watch 0 Star 0 Fork 0

Code Issues 1 Pull requests 0 Projects 0 Wiki Insights Settings

First Issue #1

Open yh0921k opened this issue 10 seconds ago · 0 comments

yh0921k commented just now

Owner

hello

2bae4ae

yh0921k added bug help wanted labels just now

yh0921k added this to the First Release milestone just now

yh0921k self-assigned this just now

Assignees

yh0921k

Labels

bug

help wanted

Projects

None yet

Milestone

First Release

Notifications

Unsubscribe

You're receiving notifications because you were assigned.

1 participant

Write Preview

AA B i “ < > ↻ ⋮ ⋮ ⋮ ↶ @ *

Leave a comment

Attach files by dragging & dropping or selecting them.

Styling with Markdown is supported

Close issue Comment

Lock conversation

GitHub의 협업 도구

- 이슈 트래커
 - 이슈가 등록되면 등록된 이슈에 댓글을 남기거나 이슈가 참조하는 커밋 등을 살펴보면서 이슈를 해결하게 된다.
 - 아래의 Write 탭은 댓글을 입력하는 곳으로, 이슈 작성과 마찬가지로 커밋 SHA-1 checksum 값이나 그림을 넣을 수 있다.
 - Preview 탭은 입력한 댓글이 어떻게 보이는지 확인할 수 있다.
 - 첨부한 이미지 확인에 사용
 - 이슈가 해결되었다면 Close issue를 클릭해 해당 이슈를 닫을 수 있다.
 - 댓글이 입력 중인 상태라면 Close and comment로 버튼 이름이 바뀐다.
 - Issue 메인의 closed 탭에서 닫힌 이슈를 다시 열고 재논의할 수 있다.

GitHub의 협업 도구

- 위키

- 흔히 알고 있는 위키와 같이 특정 주제나 단어 등에 대한 정보를 담아둔 개별 페이지를 말한다.
- Edit mode를 통해 문서를 작성할 때 사용하는 여러 가지 문법을 사용할 수 있다.
 - 자신이 선호하는 마크다운 문법을 사용한다.
 - 물음표 버튼을 누르면 각 문법에 대한 설명을 볼 수 있다.
- Edit message 항목에는 문서를 간략하게 설명하거나 변경 사항이 무엇인지 간단하게 적어둔다.
- 문서 링크는 'https://github.com/wizplan/git_test/wiki/문서이름'의 형태로 생성된다.

GitHub의 협업 도구

- 위키

Welcome to the NetSys wiki!

프로젝트

커밋

커밋 시기

커밋은 언제 하는가?

커밋 메시지 컨벤션

브랜칭

브랜칭은 어떻게 하는가?

위키

위키 사용법

위키를 작성할 때 지켜야할 규칙

용어 목록

* 용어1

* 용어2

 * 용어 2-1

문서 링크

[TEST1](https://github.com/wizplan/git_test/wiki/Test1)

[테스트 링크](https://github.com/wizplan/git_test)

[미생성된 문서](https://github.com/wizplan/git_test/wiki/미생성된 문서)

GitHub의 협업 도구

- 위키

yh0921k / NetSys

Watch 0Star 0Fork 0

CodeIssues 1Pull requests 0Projects 0WikiInsightsSettings

Home

yh0921k edited this page just now · 3 revisions

Welcome to the NetSys wiki!

프로젝트

커밋

커밋 시기

커밋은 언제 하는가?

커밋 메시지 컨벤션

브랜칭

브랜칭은 어떻게 하는가?

위키

위키 사용법

위키를 작성할 때 지켜야할 규칙

용어 목록

- 용어1
- 용어2
 - 용어 2-1

문서 링크

[TEST1](#)

[테스트 링크](#)

[미생성된 문서](#)

+ Add a custom footer

Pages 1

Home

+ Add a custom sidebar

Clone this wiki locally

<https://github.com/yh0921k/>

Clone in Desktop

GitHub의 협업 도구

- 위키

- 아직 생성하지 않은 문서를 하이퍼링크로 만들었다면 빨간색으로 표시되며, 만약 내부 위키 문서 주소인 경우 하이퍼링크를 클릭하면 해당 문서의 이름으로 생성 화면 페이지가 열린다.
 - 만약 미리 문서 이름을 정해둔다면 하이퍼링크만 만들어둔 다음 천천히 내용을 채워나갈 수 있다.
- 만들어지지 않은 내부 위키 문서로 하이퍼링크를 걸거나, 이미 있는 내부 위키 문서로 하이퍼링크를 설정할 때는 다른 방법을 사용할 수 있다.
 - [[Home]]
 - 위와 같이 대괄호를 두 개 사용해서 표시하면 자동으로 위키 내부의 문서 링크가 만들어진다.

[[HOME]]

[[여기|<https://github.com/yh0921k/NetSys/wiki/HOME>]]를 클릭하세요.

HOME

여기를 클릭하세요.

GitHub의 협업 도구





- 위키
 - Add a custom footer, Add a custom sidebar
 - 문서 하단에 나타나는 Custom Footer와 Custom Index를 작성하는데 유용한 Custom Sidebar를 만든다.

Editing _Footer





Write


Preview

h1h2h3



B*i*`<>`





Edit mode:

Markdown

custom footer는 모든 문서 하단에 나타나게 됩니다.
[[HOME]]

Editing _Sidebar

Page History





New Page

Delete Page





Write


Preview

h1h2h3



B*i*`<>`





Edit mode:

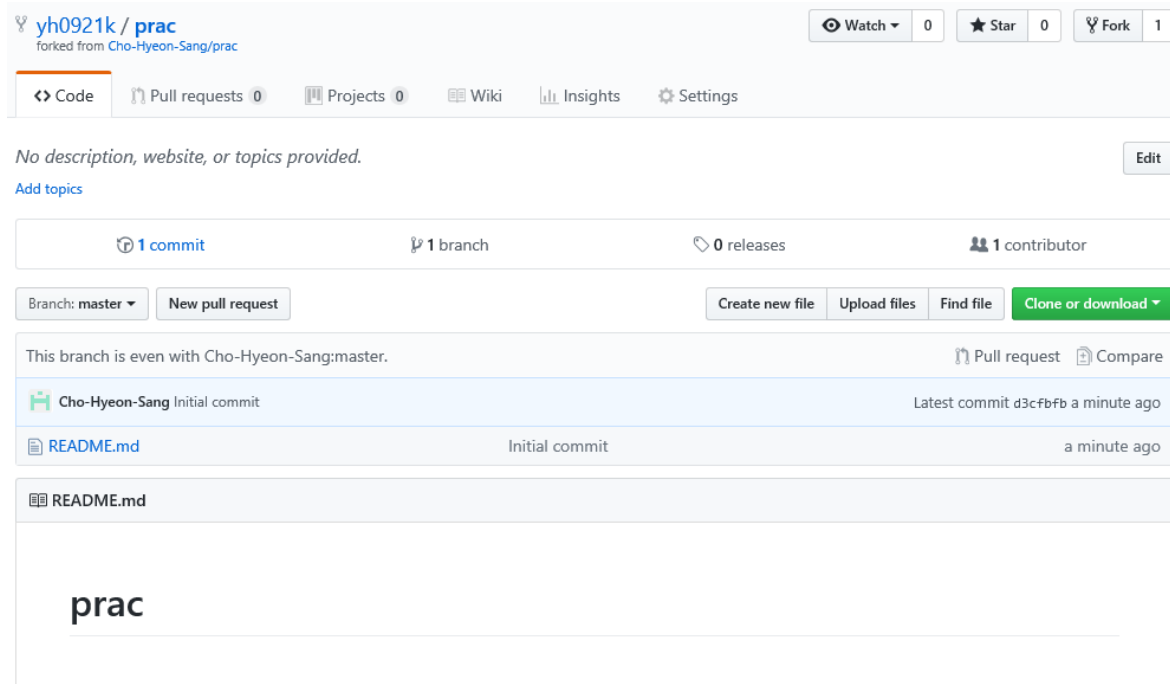
Markdown

- [[HOME]]
- [[위키]]
- [[프로젝트]]
 - [[커밋]]
 - [[브랜칭]]

GitHub의 협업 도구

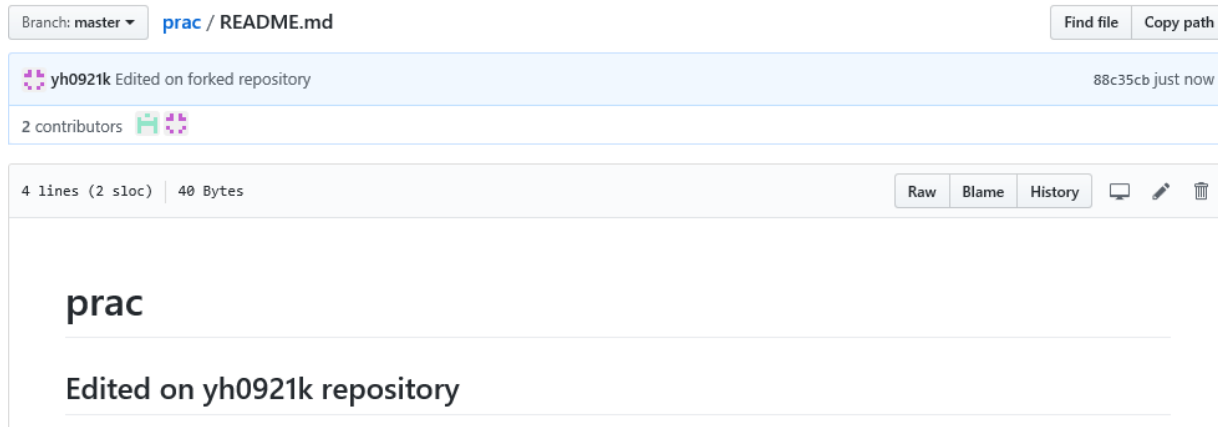
- 풀 리퀘스트

- 기술적으로는 원 저장소의 소유자가 포크한 다른 저장소의 변경 내역을 병합하는 것이다.
 - 아무나 저장소를 병합해서는 안되므로 요청이라는 형태로 다른 저장소 소유자가 수행하게끔 한다.
- yh0921k 계정에서 Cho-Hyeon-Sang 계정의 prac 저장소 fork



GitHub의 협업 도구

- 풀 리퀘스트
 - 변경 내역을 만들어 이를 커밋한다.
 - README.md 파일 수정



- Pull request > New pull request or Create a pull request

GitHub의 협업 도구

- 풀 리퀘스트

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base fork: Cho-Hyeon-Sang/prac

base: master

head fork: yh0921k/prac

compare: master

✓ Able to merge. These branches can be automatically merged.

Create pull request

Discuss and review the changes in this comparison with others.

1 commit

1 file changed

0 commit comments

1 contributor

Commits on Feb 12, 2018

yh0921k

Edited on forked repository

Verified

88c35cb

Showing 1 changed file with 3 additions and 1 deletion.

Unified

Split

4 README.md

<>

📄

View

🗨

⌵

... .. @@ -1 +1,3 @@

1 -# prac

1 +# prac

2 +

3 +## Edited on yh0921k repository

No commit comments for this range

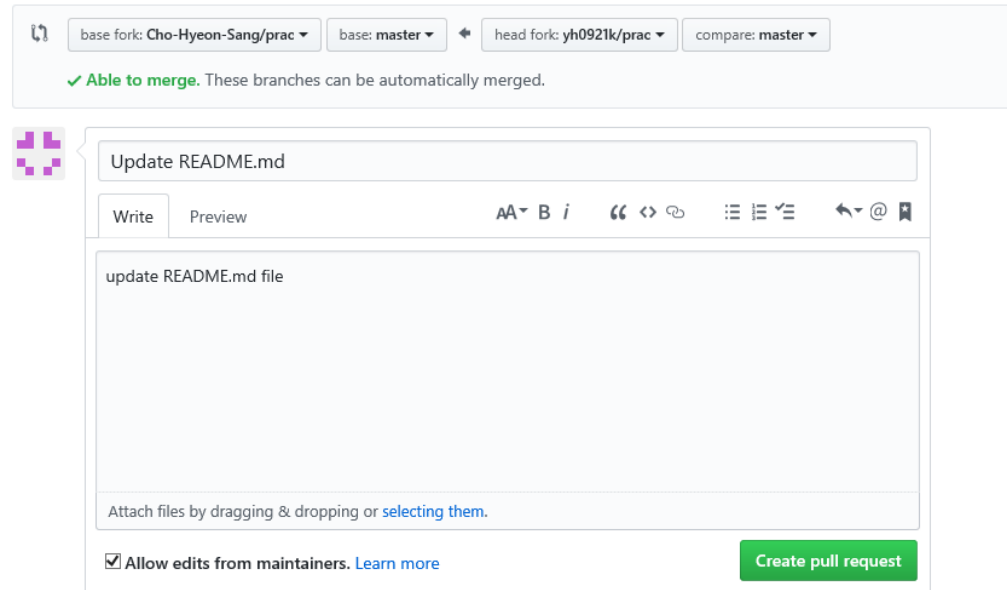
- base fork : 병합의 기반이 될 저장소와 브랜치를 선택
- head fork : 병합 대상이 될 저장소와 브랜치를 선택

GitHub의 협업 도구

- 풀 리퀘스트
 - Create pull request

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



base fork: Cho-Hyeon-Sang/prac base: master head fork: yh0921k/prac compare: master

✓ Able to merge. These branches can be automatically merged.

Update README.md

Write Preview

update README.md file

Attach files by dragging & dropping or [selecting them](#).

☒ Allow edits from maintainers. [Learn more](#)

Create pull request

- 생성된 pull request는 원 저장소에서 확인할 수 있다.
 - 자신이 fork해 온 저장소에는 해당 요청이 보이지 않는다.
 - 기본적으로 pull request는 받은 요청만 표시하고 보낸 요청은 표시하지 않는다.

GitHub의 협업 도구

- 풀 리퀘스트

- 원 저장소에서는 생성된 pull request에 대한 토론을 댓글의 형태로 할 수 있고, 관리자가 승인하면 변경 내용이 반영된다.
- 알림 설정에 따라 pull request가 추가되면, 메일을 발송하기도 한다.
- 어떤 커밋이 있었는지, 어떤 파일이 어떻게 변경이 되었는지 확인하고 토론을 진행할 수 있다.

Update README.md #1

[Edit](#)[Open](#) yh0921k wants to merge 1 commit into Cho-Hyeon-Sang:master from yh0921k:master[Conversation](#) 0[Commits](#) 1[Files changed](#) 1

Changes from all commits ▾ Jump to... ▾ +3 -1 ■■■■

Unified

[Split](#)[Review changes ▾](#)

4 ■■■■ README.md

[<>](#)[File](#)[View](#)[Diff](#)[Edit](#)[More](#)

... @@ -1 +1,3 @@

1 -# prac

1 +# prac

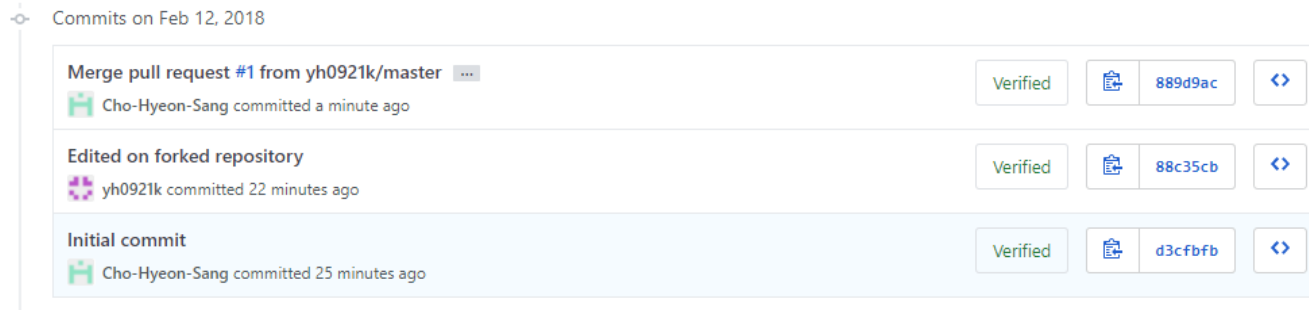
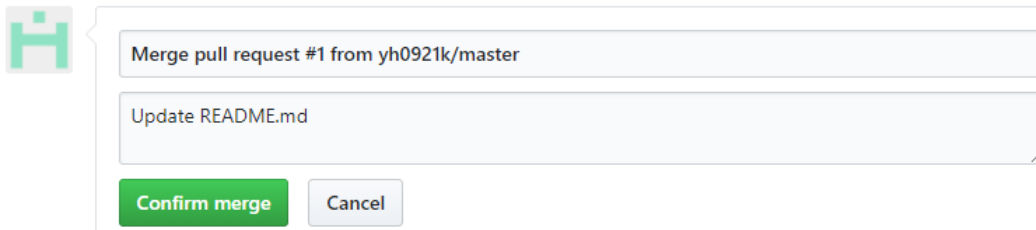
2 +

3 +## Edited on yh0921k repository

GitHub의 협업 도구

- 풀 리퀘스트

- 최종적으로 코드를 반영하기로 결정했다면, Merge pull request를 사용해 병합한다.
- 반영하지 않는다면, Close pull request를 클릭해 닫을 수 있다.
 - 물론 나중에 다시 열어서 요청 과정을 재개할 수 있다.
- 병합 또한 코드를 변경하는 작업이므로 적절한 Confirm message를 남긴다.



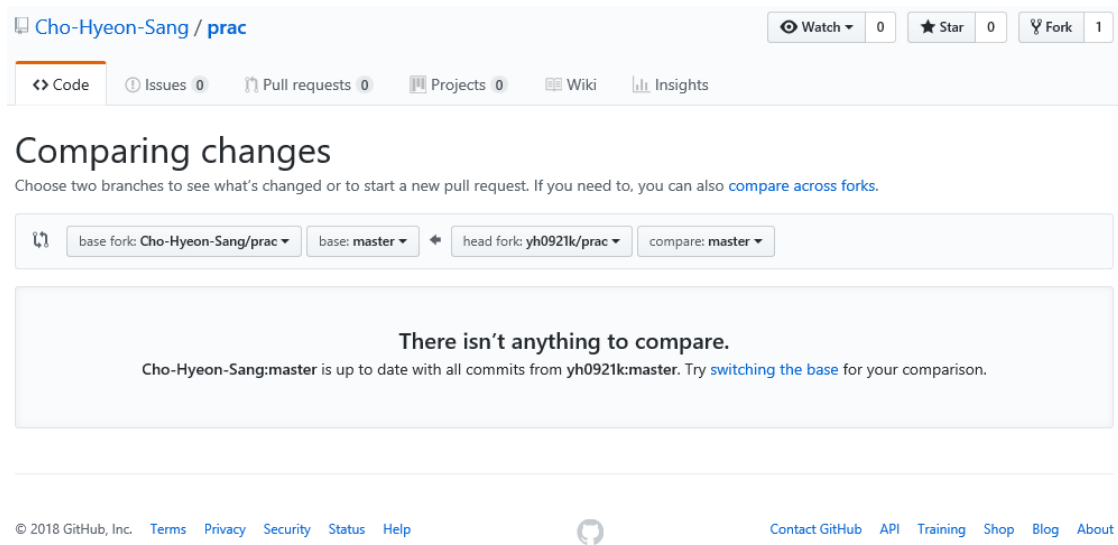
- 저장소의 메인 페이지에서 commit을 확인하면 반영된 커밋을 확인할 수 있다.

GitHub의 협업 도구

- 코드를 최신 상태로 유지하기
 - Git은 기본적으로 커밋 내역을 통해서 최신 상태인지를 확인한다.
 - 따라서 풀 리퀘스트를 받아들인다면 저장소의 내용은 같지만 커밋 내역에는 차이가 발생할 수 있다.
 - 때문에 코드를 최신 상태로 유지하려면 다시 풀 리퀘스트를 이용해야 한다.
 - 풀 리퀘스트를 이용해서 포크한 자신의 저장소를 최신으로 유지하는 방법은 간단하다.
 - 새로운 풀 리퀘스트를 만들어서 자신의 저장소를 원 저장소와 같은 상태로 만들면 된다.
 - 즉, 풀 리퀘스트의 생성과 수락을 혼자서 다 하는 것이다.

GitHub의 협업 도구

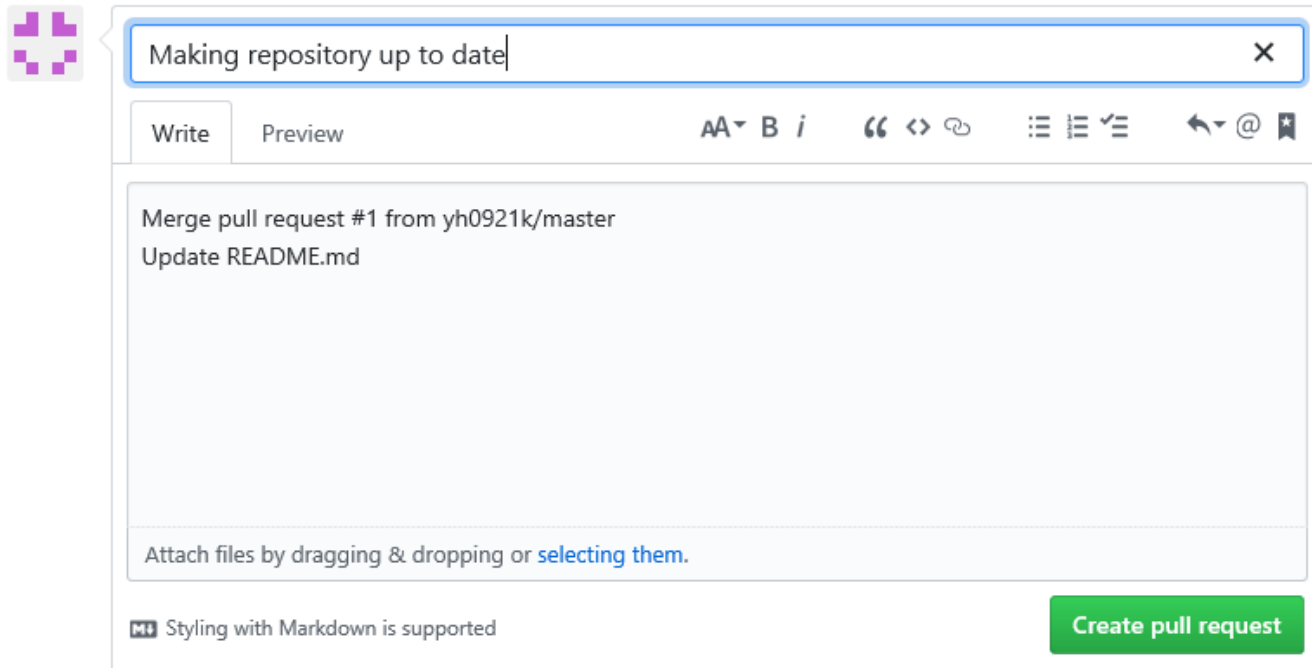
- 코드를 최신 상태로 유지하기
 - 포크했던 저장소에서 다시 새로운 풀 리퀘스트를 만든다.
 - 하지만 포크한 저장소를 원 저장소보다 커밋이 뒤쳐져 있는 상태이다.
 - 따라서 비교할 것이 아무것도 없다는 메시지와 함께 기준을 바꿔야 한다고 알려준다.



- switching the base를 클릭한다.

GitHub의 협업 도구

- 코드를 최신 상태로 유지하기
 - Base fork는 현재 작업중인 포크한 저장소이고 head fork는 원 저장소로 설정되어 있다.
 - 뒤쳐진 커밋만큼 풀 리퀘스트를 할 수 있다.
 - Create pull request
 - Title : Making repository up to date



Making repository up to date

Write Preview

Merge pull request #1 from yh0921k/master
Update README.md

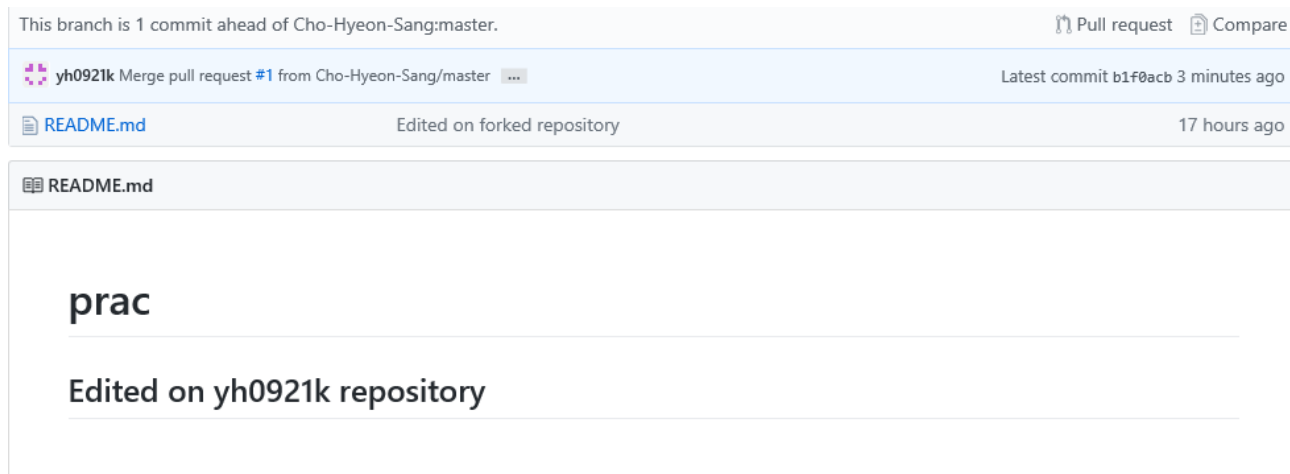
Attach files by dragging & dropping or [selecting them](#).

Styling with Markdown is supported

Create pull request

GitHub의 협업 도구

- 코드를 최신 상태로 유지하기
 - Merge pull request를 통해 해당 내역을 merge 한다.
 - 이후 저장소의 메인으로 가서 원 저장소의 내용들이 제대로 포크한 저장소로 이동했는지 확인한다.
 - 이해가 빠르다면, 원 저장소를 원격 저장소에 다른 이름으로 추가해서 풀이나 페치를 해도 된다는 것을 알 수 있다.
 - 하지만 여기서 설명하는 방법은 GitHub 안에서 해결하는 비교적 간단한 방법을 설명한 것이며, 정석으로는 위 줄의 방법이 더 맞다고 볼 수 있다.



GitHub의 협업 도구

- GitHub에서의 코드 리뷰
 - 앞에서 살펴본 것처럼 GitHub에서는 프로젝트 진행 전 과정에서 댓글 기능을 사용할 수 있다.
 - 이를 잘 활용하면 전문적인 코드 리뷰 도구를 사용하지 않아도 GitHub상에서 코드 리뷰를 진행할 수 있다.
 - 커밋 내역을 하나 선택해서 댓글을 작성한다.
 - 아래와 같이 커밋 내역 전체에 관한 리뷰 메시지를 작성할 수 있다.

0 comments on commit 88c35cb

 Lock conversation



Write

Preview

AA ▾ B i “ <> 🔗 ☰ ☷ ✓ ↩ @ ★

Leave a comment

Attach files by dragging & dropping or [selecting them](#).

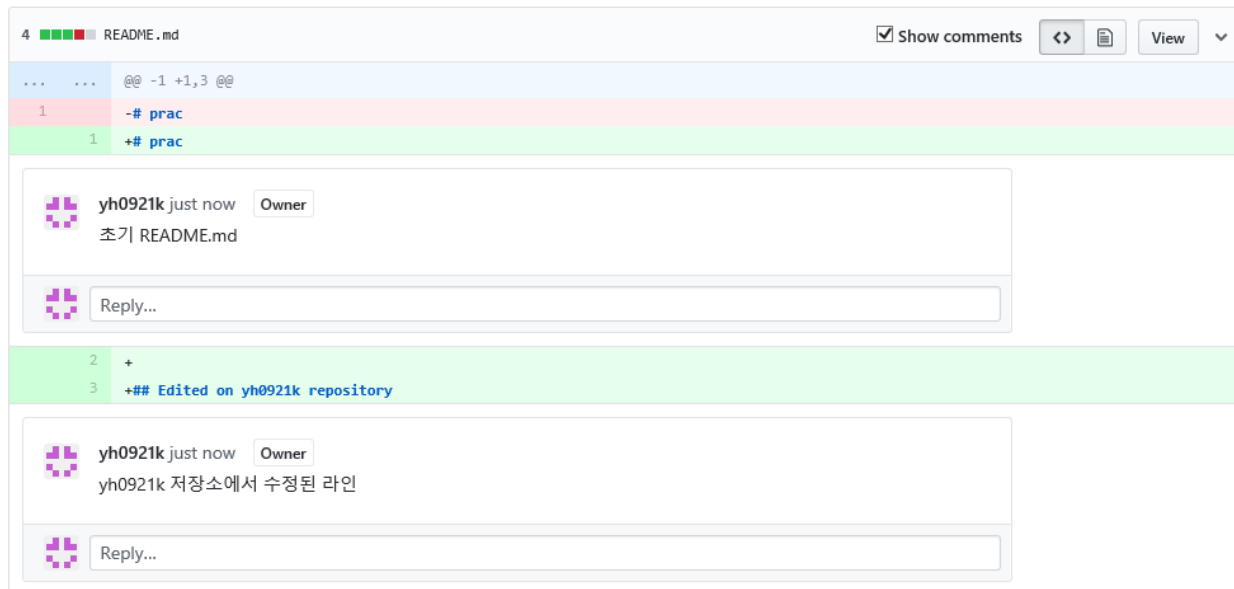
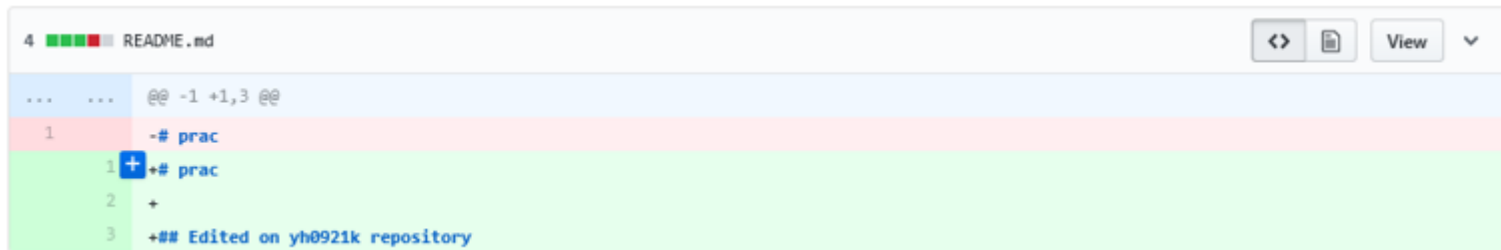
 Styling with Markdown is supported

Comment on this commit

GitHub의 협업 도구

- GitHub에서의 코드 리뷰

- 또한, 커밋의 변경 내역 라인 위에 커서를 올리면 파란색 [+] 기호가 나타나는데, 이를 사용해서 해당 라인에 댓글을 작성할 수 있다.
 - 이렇게 댓글을 작성하면 코드와 코드 사이에서 댓글이 바로 보이게 된다.



프로젝트를 위한 협업 준비 규칙

- 이전까지는 Git과 GitHub라는 도구의 사용법 자체를 배웠다면, 여기서부터는 Git과 GitHub를 언제 어떻게 사용해야 할지를 알아본다.
 - 프로젝트 관리나 소프트웨어 공학에 가까운 내용이기 때문에 따라서 내용이 다소 추상적일 수 있다.
 - Git을 이용해 프로젝트를 관리하는 방법에는 특별히 정해진 규칙이 없다.
 - 언제든지 브랜치를 만들어서 새로운 기능을 시험해볼 수 있고, 원 저장소와는 상관없는 자신만의 로컬 저장소를 만들어서 작업할 수 있는 것이 Git이다.
 - 하지만 협업의 상황에서는 무엇보다 중요한 것이 브랜칭 규칙이다.
 - 모두가 다 master 브랜치를 브랜칭해서 자신의 이름을 딴 브랜치에서 작업할 수 있다.
 - 하지만 그것보다 프로젝트 전체를 관리하는 훨씬 더 쉬운 방법이 있다.
 - 이번 절에서는 프로젝트를 관리하기 전에 세워야 할 규칙을 정의한다.

프로젝트를 위한 협업 준비 규칙

- 커밋 단위
 - 커밋에 포함될 수 있는 내용이 여러 개로 나누어질 수 있을 만큼 크다면, 이를 쪼개서 커밋해야 한다.
 - 즉, 커밋의 내용을 최소 단위로 유지하는 것이다.
 - 이를 지키기 위해 다음과 같은 규칙을 지키는 것이 좋다.
 - 커밋 하나는 하나의 의도와 의미만을 가져야 한다. 한번에 여러 파일을 수정하더라도 그 의도는 단 하나여야 한다. 그것이 버그 수정이든 새로운 기능 추가든 마찬가지이다.
 - 파일을 하나만 수정하더라도 두 개 이상의 의도가 있다면 하지 말아야 한다. 즉, 버그 수정과 새 기능 추가를 동시에 하지 않아야 한다.

프로젝트를 위한 협업 준비 규칙

- 커밋 메시지 작성 규칙

- 커밋 메시지는 자유롭게 작성할 수 있지만, 정말 자유롭게 작성한다면 협업에는 방해가 된다.
 - 처음으로 Git을 도입하는 팀에 알맞은 간단한 커밋 메시지 작성 규칙이 있다.

[category] - [simple message]

[detailed description]

- category는 커밋의 성격이 무엇인지 한번에 알 수 있는 단어로 작성한다.
 - 가능하면 짧고 명확하게 카테고리를 분류한다.
 - 예를 들어 fix, add, mod, rm 등의 카테고리가 있을 수 있다.
 - 차례로 잘못된 부분 수정, 기능 추가, 코드 수정, 기능 삭제의 의미를 담고 있다.
- message는 해당 커밋에 대한 간단한 한 줄 설명을 작성한다.
 - 일반적으로 터미널에 보이는 글자 수를 고려하여 영문 기준 70자 정도가 적절하다.
- detailed description에 포함될 수 있는 내용은 다양하다.
 - 커밋의 자세한 내용을 여기에 기술한다.

프로젝트를 위한 협업 준비 규칙

- 커밋 메시지 작성 규칙
 - detailed description에 포함될 수 있는 내용은 다양하다.
 - 커밋의 자세한 내용을 여기에 기술한다.
 - detailed description을 작성할 때, 아래와 같은 내용은 지키도록 한다.
 - 왜 커밋을 했는가?
 - 버그 수정의 경우 원래 어떤 문제가 있었는가?
 - 사용 중인 이슈 트래커가 있다면 해당 이슈의 하이퍼링크를 포함해야 한다.
 - 위의 규칙들을 염두에 두면서 커밋을 작성하면, 작성된 커밋만 보고도 많은 정보를 얻을 수 있고 새로 협업에 참여하는 사람이 이전의 커밋 로그만을 보고도 전체적인 프로젝트를 잘 이해할 수 있다.
 - 좋은 커밋 메시지와 나쁜 커밋 메시지의 예를 볼 수 있는 Git Commit Good Practice
 - <https://wiki.openstack.org/wiki/GitCommitMessages>
 - 위의 내용을 반드시 읽어보도록 하자.

프로젝트를 위한 협업 준비 규칙

- 브랜치 이름 작성 규칙
 - 새로운 브랜치를 만들 때 어떻게 이름을 지을지를 협업자들 사이에 공유한다면, 브랜치 이름만을 보고서 어떤 목적으로 브랜치를 만들었는지를 바로 알 수 있다.
 - 자세한 설명이 없이도 사람들이 바로 알아볼 수 있어야 한다는 점에서 브랜치 네이밍 작성 규칙은 커밋 메시지 작성 규칙과 유사한 점이 있다.
 - 커밋 메시지와 마찬가지로 몇 가지 카테고리를 만들어서 분류하는 것이 좋다.
 - new : 새 기능 추가가 목적인 브랜치
 - test : 무언가를 테스트하는 브랜치(새 라이브러리, 배포 환경, 실험 등)
 - bug : 버그 수정이 목적인 브랜치
 - 사람이 한 번에 알아볼 수 없거나(숫자로만 된 이름, 의미가 없는 이름) 너무 긴 설명조의 이름은 사용하지 않는다.

프로젝트를 위한 협업 준비 규칙

- 태그와 버전 이름 작성 규칙
 - 프로젝트를 진행하다 보면 자연스레 버전 이름을 붙이게 된다.
 - <https://semver.org/lang/ko/>
 - 위의 웹 페이지를 참고해 버전 이름을 커밋에 올바르게 태깅하는 것만으로도 많은 정보를 제공할 수 있다.
 - 간단한 x. y. z 설명
 - x는 기존과 호환이 되지 않는 변경이 발생할 때 증가시킨다.
 - y는 기존과 호환이 되며, 새로운 기능이 추가될 때 증가시킨다.
 - z는 기존과 호환이 되며, 버그 수정 등이 될 때 증가시킨다.
 - 위의 세 가지 규칙을 기억하고 필요에 따라 버전 이름 뒤에 간단한 라벨을 붙여서 부가 설명을 하면 된다.

프로젝트 유형별 협업 흐름

- Git을 이용한 협업은, 다르게 말하면 브랜칭 생성 규칙을 공유하는 것으로 말할 수 있다.
 - 어떤 브랜칭 생성 규칙을 선택할 것인지는 아래의 표를 고려해 결정할 수 있다.

요소 / 플랫폼	데스크톱 애플리케이션	웹	모바일 앱
특징	배포 후에 유지보수가 힘들	배포와 개발의 구분이 없음, 언제나 배포가능	배포 이후 지속적인 업데이트 가능
프로젝트 마감	최초 배포 시	해당 없음	버전마다
배포 단위	최초 한번, 업데이트는 패치 등으로 제공됨	해당 없음, 무결정성	버전마다
배포 시기 조절 가능 여부	가능	해당 없음	불가능
추천 Flow	git-flow	github-flow	gitlab-flow

- 이번 절에서는 추천 흐름의 세 가지인 git-flow, github-flow, gitlab-flow를 소개한다.

프로젝트 유형별 협업 흐름

- git-flow : 게임이나 SI 개발 환경에 권장
 - 데스크톱 애플리케이션은 한번 배포된 후에는 유지보수가 쉽지 않다.
 - 사용 환경이 오프라인일 경우, 도중에 실행을 중단할 수 없는 경우, 사용자가 유지보수에 관심을 가지지 않을 경우 등 다양한 이유가 있다.
 - 이런 애플리케이션의 경우 매우 견고하게 만들어져야 하며, 견고함을 유지하기 위해 프로젝트 작업 흐름 또한 여러 가지 결함을 최소화하고 결함이 있다고 해도 빠른 시간 안에 감지해 수정할 수 있는 모델이어야 한다.
 - 이러한 조건을 만족하는 것으로 빈센트 드라이센이 제안한 작업 흐름 모델인 'A successful Git branching model'이 있다.
 - 이 작업 흐름 모델은 브랜치 그룹에 역할을 부여하고 각 브랜치의 상호작용을 엄격하게 제한한다.
 - 이 작업 모델은 브랜치를 다섯 가지의 역할로 나눈다.
 - develop branch
 - feature branch
 - release branch
 - master branch
 - hotfix branch

프로젝트 유형별 협업 흐름

- git-flow : 게임이나 SI 개발 환경에 권장



- develop branch
 - 하나만 존재한다.
 - 모든 개발이 시작되는 부분이다.
 - 절대로 develop 브랜치에서 바로 커밋하지 않는다.
 - 이 브랜치에 병합되는 것은 feature 브랜치와 release 혹은 hotfix의 버그 수정이다.
 - 이 브랜치는 오직 병합 커밋만 할 수 있다.

프로젝트 유형별 협업 흐름

- git-flow : 게임이나 SI 개발 환경에 권장



■ feature branch

- 여러 개가 존재할 수 있다.
- 여기에 속하는 branch는 develop branch를 기반에 두고 브랜치되어 새로운 기능 개발이나 버그 수정을 담당한다.
- 각각의 브랜치는 하나의 기능(의도)만을 맡는다.
- 따라서 네이밍이 중요하다.
- feature branch들은 오직 develop branch에 병합될 때만 관계성이 생긴다.
- 갈라져 나오는 것도, 다시 병합하는 것도 오직 develop branch와만 진행한다.

프로젝트 유형별 협업 흐름

- git-flow : 게임이나 SI 개발 환경에 권장



release branch

- develop branch에서 나와서 배포 준비를 하는 브랜치이다.
- 새로운 기능 추가는 더 하지 않고 오로지 버그 수정만 한다.
- 즉, 배포본의 완성도를 높이는 branch이다.
- 당연히 수정된 버그는 develop 브랜치로 병합되어야 한다.

프로젝트 유형별 협업 흐름

- git-flow : 게임이나 SI 개발 환경에 권장



- hotfix branch
 - master branch, 즉 현재 배포 중인 코드에 버그가 있어 급히 수정할 때만 사용한다.
 - hotfix branch로 수정한 내용은 master와 develop branch에만 반영한다.

프로젝트 유형별 협업 흐름

- git-flow : 게임이나 SI 개발 환경에 권장
 - 정리 : develop branch를 중심으로 feature branch들을 통해 기능을 추가하고 release branch를 통해 배포 준비와 코드의 버그를 수정하며 master로 배포하고 hotfix로 배포된 버전의 버그를 수정해 master와 develop branch에 반영하는 것이 반복된다.
 - 순서
 - develop branch를 기반으로 feature branch 생성
 - feature branch에서 기능 개발 시작
 - 기능 개발이 완료되면 develop branch로 풀 리퀘스트 혹은 병합
 - 배포 시기가 되면 develop branch를 기반에 두고 release branch 생성
 - release branch에서 이미 알려진 버그 수정에 주력, 수정된 버그는 develop branch에 반영
 - 배포 시기가 되면 release branch를 기반에 두고 master branch를 생성하여 배포
 - master branch(=배포된 버전)에서 버그가 발견되면 hotfix branch를 생성
 - hotfix branch에서 버그 수정이 끝나면 develop branch와 master branch에 반영
 - 이 작업 흐름은 주기적으로 작업 결과를 배포하는 프로젝트에 적합한 방식이다.
 - 매우 견고한 코드를 생산하면서 배포 간격이 충분히 긴 프로그램이나 솔루션을 다루는 프로젝트에 적합하다.
 - 빠른 개발과 배포에는 알맞지 않는다.
 - 예를 들어, 하루에도 몇 번씩 작업 결과를 배포해야 하는 웹 서비스 등에는 알맞지 않는다.

프로젝트 유형별 협업 흐름

- github-flow : 웹 애플리케이션
 - git-flow의 단점을 해결하고자 GitHub에서 사용하는 github-flow를 소개한다.
 - 이 작업 흐름은 GitHub에서 사용 중인 작업 흐름이다.
 - master와 feature, 두 가지 브랜치만 존재한다.
 - 앞에서 언급한 git-flow와 비교하면 매우 가벼운 프로젝트 작업 흐름 모델이다.
 - 따라서 빠른 기능 추가와 수정이 필요한 분야에 적합하다.
 - 대표적으로 하루에도 몇 번씩 배포될 수 있는 웹 애플리케이션 등이 이 작업 흐름을 사용하기에 적합하다.

프로젝트 유형별 협업 흐름

- github-flow : 웹 애플리케이션
 - master branch
 - 언제나 배포할 수 있는 상태로 유지되는 브랜치
 - master branch가 곧 배포 브랜치가 된다.
 - 하나만 존재하며, 오직 병합 커밋만 할 수 있다.
 - feature branch
 - 여러 개가 존재할 수 있다.
 - master branch에서 갈라져서 새 기능을 추가하거나, 버그를 수정하거나, 그 외의 모든 코드 수정을 담당하는 branch group이다.
 - 다른 작업 흐름과 마찬가지로 한 번에 하나의 의도만을 구현하는 브랜치 그룹이며, 그에 따라 이름 짓기가 중요하다.

프로젝트 유형별 협업 흐름

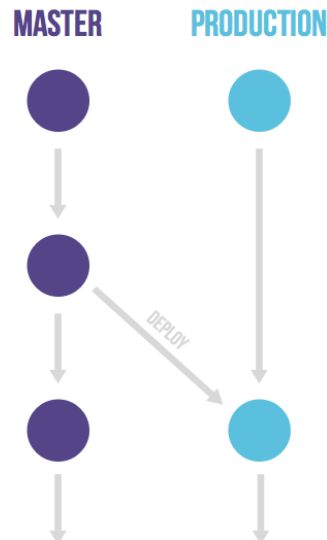
- github-flow : 웹 애플리케이션
 - 순서
 - master branch를 기반에 두고 feature branch 생성
 - feature branch에서 기능 개발 시작
 - 기능이 완성되면 master branch에 풀 리퀘스트
 - feature branch에서 받은 풀 리퀘스트는 협업자들의 코드 리뷰 진행
 - 코드 리뷰를 반영해 feature branch에서 작업 진행
 - 위의 3~5를 필요한 만큼 반복
 - feature branch가 master branch에 병합되고 새 기능 배포 완료

프로젝트 유형별 협업 흐름

- gitlab-flow : 모바일 앱과 게임
 - 양극단으로 볼 수 있는 git-flow와 github-flow의 중간에 gitlab-flow가 있다.
 - GitLab Flow 웹 문서에서 github-flow를 기본으로 여러 가지 변형 형태를 gitlab-flow라는 이름으로 소개한다.
 - <https://about.gitlab.com/2014/09/29/gitlab-flow/>
 - 작업 흐름은 github-flow를 따르지만 배포 과정을 GitLab에서 개선한 작업 흐름이라고 생각하면 된다.
 - github-flow의 부족한 점인 안정성과 배포 시기 조절 가능성을 추가 브랜치를 두어 보강하는 전략이다.

프로젝트 유형별 협업 흐름

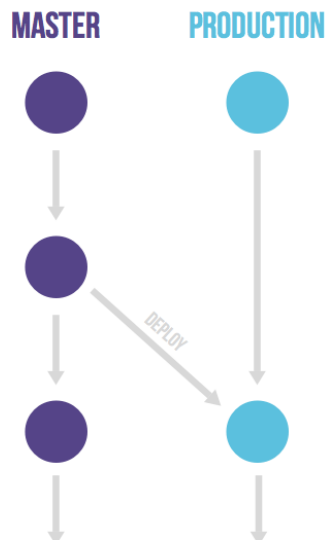
- gitlab-flow : 모바일 앱과 게임
 - Github-flow with Production
 - production branch는 배포 코드가 있는 branch이다.



- feature branch의 작업 결과가 마스터로 병합되고, 배포 준비가 되면 마스터에서 production branch로 병합한다.
 - git-flow의 release branch와 비슷하다.
 - 다만, production branch는 오직 배포만을 담당한다.

프로젝트 유형별 협업 흐름

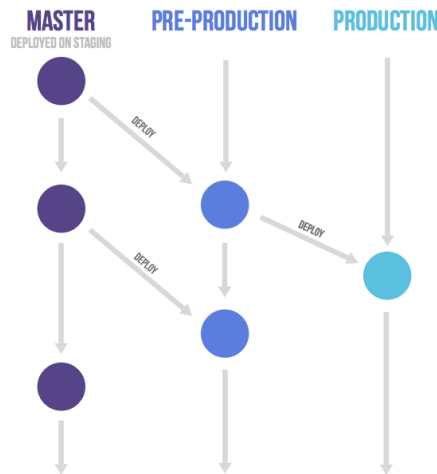
- gitlab-flow : 모바일 앱과 게임
 - Github-flow with Production
 - production branch는 배포 코드가 있는 branch이다.



- 이 작업 흐름은 사용자의의도대로 배포할 수 없는 환경일 때 적절하다.
 - 앱 스토어나 구글 플레이 마켓 등으로 배포하는 모바일 앱 개발에 어울리는 흐름이다.
 - 개발 속도가 충분히 빠르더라도 배포 시기를 정할 수 없는 경우
- 즉, production 브랜치로 배포하고 외부에서 배포 승인을 기다리는 것으로 생각하면 된다.
 - 또는 배포 시점을 일정하게 통제하고 싶을 때 사용할 수 있다.
 - 또는 github-flow의 특징인 잦은 배포에 대한 부담을 줄이는 방법으로 쓸 수 있고, 한 번의 배포가 매우 큰 일이라면 이 작업 흐름을 사용할 수 있다.

프로젝트 유형별 협업 흐름

- gitlab-flow : 모바일 앱과 게임
 - Github-flow with Pre-production and Production
 - 이전 모델에 pre-production branch를 추가한 작업 모델이다.



- 이 브랜치는 테스트 브랜치로, 개발 환경에서 바로 배포하지 않고 사용 환경과 동일한 테스트 환경에서 코드를 테스트하는 것이다.
- 모바일 앱 개발 시 각종 하드웨어에서 제대로 실행되는지 테스트할 필요가 있다면, 이 작업 흐름을 사용할 수 있다.
- 웹 개발에서도, 로컬 저장소에서 기능 개발을 마친 다음, 테스트 서버에서 시험하는 것을 pre-production branch를 만드는 것으로 생각할 수 있다.
 - 실제 서버로 배포하는 것을 production branch에 병합하는 것으로 생각

프로젝트 유형별 협업 흐름

- 앞에서 설명한 두 가지 모델 외에도 다양한 gitlab-flow model이 있다.