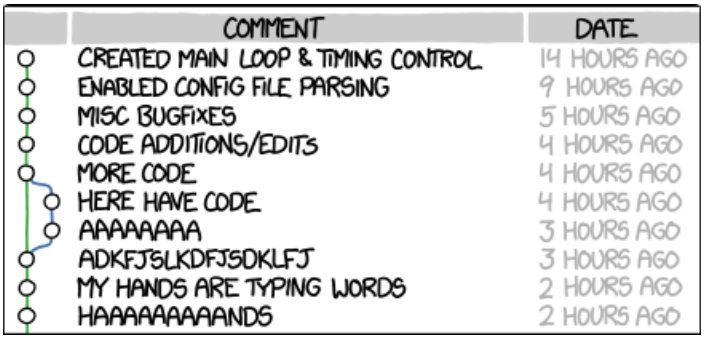
# 좋은 git 커밋 메시지를 작성하기 위한 7가지 약속






본 글은 Chris Beams의 How to Write a Git Commit Message (https://chris.beams.io/posts/git-commit/) 블로그 내용을 이해하기 빠르고 쉽게 전달하기 위해 번역, 편집하고 도움이 될만한 내용을 추가한 것입니다.

## git 커밋 메시지를 잘 쓰려고 노력해야 하는 이유

왜 커밋 메시지를 잘 쓰기 위해 노력해야 할까요? 이유는 서로 다를 수 있지만, 잘 쓰인 커밋 메시지가 더 유익하다는 사실에는 많은 프로그래머들이 동의할 것으로 여겨집니다. 그 중 대표적인 3가지를 꼽아보면 다음과 같습니다.

1. 더 좋은 커밋 로그 가독성
2. 더 나은 협업과 리뷰 프로세스
3. 더 쉬운 코드 유지보수

"좋은 커밋 메시지를 고민하는 건 괜찮은 생각인 것 같아요! 하지만 좋은 메시지에 딱부러진 정답이 있을지 모르겠네요. 사람들마다 좋은 커밋 메시지로 생각하는 기준이 모두 다를 것 같은데, 그렇다면 그나마 좀 더 많은 사람들이 공감하고 실천하고 있는, 좋은 커밋 메시지를 만드는 약속 같은 것이 있을까요?"

이번 글에서는 그 좋은 커밋 메시지의 보편적인 기준을 고민해 봅니다.

## 좋은 git 커밋 메시지를 작성하기 위한 7가지 약속

> 이하 약속은 커밋 메시지를 `English` 로 작성하는 경우에 최적화되어 있습니다. 한글 커밋 메시지를 작성하는 경우에는 더 유연하게 적용하셔도 좋을 것 같네요.

1. 제목과 본문을 한 줄 띄워 분리하기
2. 제목은 영문 기준 50자 이내로
3. 제목 첫글자를 대문자로
4. 제목 끝에 `.` 금지

5. 제목은 `명령조`로
6. 본문은 영문 기준 72자마다 줄 바꾸기
7. 본문은 `어떻게`보다 `무엇을`, `왜`에 맞춰 작성하기

## 1. 제목과 본문을 한 줄 띄워 분리하기

시작부터 가장 핵심적인 꿀팁 한 가지를 알려드리겠습니다. 결론부터 말씀드리자면 제목과 본문 사이에 공백 한 줄을 추가하는 것은 **생각보다 상당히 효과적인 테크닉**입니다. 우선 `git commit`의 `man` 페이지를 참조하면 다음과 같은 문장이 나옵니다.

> Though not required, it's a good idea to begin the commit message with a single short (less than 50 character) line summarizing the change, followed by a blank line and then a more thorough description. The text up to the first blank line in a commit message is treated as the commit title, and that title is used throughout Git. For example, git-format-patch(1) turns a commit into email, and it uses the title on the Subject line and the rest of the commit in the body.

요약하자면 커밋 메시지는 **50자 이내의 요약문장**과 **빈 줄 하나**, 그리고 **설명문**으로 구성하면 좋다는 내용입니다.

이 내용이 강제사항은 아니며 모든 커밋 메시지를 제목과 본문으로 구성할 필요는 없습니다. 때에 따라서는 아래의 예와 같이 변경사항을 한 줄로 요약한 커밋 메시지가 더 효율적입니다.

```
아 직전 커밋 중 `README.md`에 들어간 오타 한글자 고침 아 (부끄)
```

```
쌍따옴표 한 개 빼먹었다..--; 수정
```

```
어제부로 저장소 URL이 바뀜. URL 한 개만 업데이트
```

하지만 충분히 내용이 있고 잘 갖춰진 커밋 메시지를 작성할 땐 git의 제안을 따라보세요.

제목과 본문 사이에 한 줄 공백을 두면, 어떤 마법이 일어나는지 보여드리겠습니다. 다음과 같이 제안을 따른 커밋 메시지가 있습니다. 제목, 공백 한 줄, 본문으로 구성되어 있습니다.

```
Derezz the master control program

MCP turned out to be evil and had become intent on world domination.
This commit throws Tron's disc into MCP (causing its deresolution)
and turns it back into a chess game.
```

`git log`는 이 커밋 메시지를 다음과 같이 보여줍니다.

```
$ git log
commit 42e769bdf4894310333942ffc5a15151222a87be
Author: Kevin Flynn <kevin@flynnsarcade.com>
Date:   Fri Jan 01 00:00:00 1982 -0200

 Derezz the master control program

 MCP turned out to be evil and had become intent on world domination.
 This commit throws Tron's disc into MCP (causing its deresolution)
 and turns it back into a chess game.
```

별다를게 없네요. 여기서 `git log --oneline` 옵션을 사용해 봅니다.

```
$ git log --oneline
42e769 Derezz the master control program
```

짠! 제목만 보여줍니다. 한 줄 공백으로 분리하지 않았다면 어떻게 보였을까요?

```
$ git log --oneline
42e769 Derezz the master control program MCP turned out to be evil and had become intent on world domination. This commit throws
```

더 뚱뚱한 본문을 가진 커밋 메시지였더라면 어떻게 보였을지 예상이 되시죠?

`git shortlog` 라는 명령어도 있습니다. 이 명령어를 사용해서 로그를 볼 때, 제목과 본문이 한 줄 공백으로 나뉘어진 커밋들은 이렇게 보여집니다.

```
$ git shortlog
Kevin Flynn (1):
    Derezz the master control program

Alan Bradley (1):
    Introduce security program "Tron"

Ed Dillinger (3):
    Rename chess program to "MCP"
    Modify chess program
    Upgrade chess program

Walter Gibbs (1):
    Introduce protoype chess program
```

## 2. 제목은 영문 기준 50자 이내로

'50자 이내'라는 규칙은 지키기 그리 어려운 제약사항이 아닙니다. 이 작은 강제로 커밋 메시지 작성자는 더 읽기 좋은 커밋 제목을 만들 수 있고, 가장 간결히 요약된 제목을 작성할 수 있습니다. 또는 그렇게 만들 방법을 고민하는 습관을 들이는데 도움을 주죠.

50자가 정 힘들다면, 우선 "69자"를 도전해보세요.

## 3. 제목 첫글자를 대문자로

이건 거의 영문법의 문제네요. 첫글자는 대문자, 아시죠? 영어를 모국어로 사용하는 사람들이 의외로 이 문법에 민감하다고 합니다. 그 느낌이 마치 `않해!` 를 보는 수준이라고 하네요. 그러니까 이렇게 하지 마시고,

- **accelerate** to 88 miles per hour

이렇게 해주세요.

- **Accelerate** to 88 miles per hour

## 4. 제목 끝에 `.` 금지

이 역시 영문법의 문제네요. 제목에는 보통 점을 찍지 않죠. 이러지 마시고,

- Open the pod bay **doors.**

이렇게 해주세요.

- Open the pod bay **doors**

## 5. 제목은 `명령조`로

아마도 이 글에서 제일 낯선, 이전에는 모르셨을 법한, 그리고 가장 **중요한** 내용이 아닐까 생각합니다. 제목을 작성하거나 한 줄 메시지로 커밋을 할 때, 즉 커밋 메시지 가장 첫 문장의 영문법은 `명령조`로 합시다. 이는 첫 단어를 `동사원형`으로 쓰라는 의미입니다. 처음 보신 분은 이게 왜 그래야 할까 싶으실 겁니다. 사실 우리는 우리말로 커밋 메시지를 읽을 때, 누군가에게 안 일어난 일을 하도록 "지시"하는 `명령문`보다는 일어난 일을 "보고"하는 `설명문`을 더 편안히 받아들입니다. 어쨌든 커밋 메시지는 과거의 기록이니까요. 예를 들면 이런 메시지를 봤다고 가정해 보겠습니다.

```
인증 메소드 고쳐라

* CABVerification.java: 15번째 줄 인증 메소드의 인자를 현 정책에 맞게 고쳤다.
* JSONFormat.java: 이 커밋은 이 파일에 적절한 로깅 메소드를 추가한다.
* MainView.java: 약간의 리팩토링.
```

이 메시지 제목을 이렇게 고치고 싶으실 겁니다.

```
인증 메소드 고쳤다
```

우리말로 제목을 요약하는 경우라면 때로는 '문장'보다 '구문'을 더 선호할 때도 있습니다.

```
인증 메소드 수정
```

그럼 왜 영문 커밋 메시지의 제목을 `명령문`으로 써야 하는지 알려드리겠습니다.

위에서 말씀드린 것처럼 평소에는 `명령문`이 사람이 읽기에 다소 무례하거나 로그 메시지라는 상황에 어울리지 않는 것처럼 느껴질 수 있습니다. 외국인도 우리와 똑같이 생각하고 있습니다. 그런데 `명령문`은 git 커밋의 제목으로는 아주 딱입니다. 그 한가지 좋은 이유는 **git 스스로가 자동 커밋을 작성할 때 `명령문`을 사용하고 있습니다.** 예를 들어, 기억을 되돌아보면 `git merge`를 실행했을 때 커밋 메시지 기본값이 이렇습니다.

```
Merge branch 'myfeature'
```

`git revert` 명령어를 실행하면 어떤 메시지가 기본값으로 들어갈까요?

```
Revert "Add the thing with the stuff"

This reverts commit cc87791524aedd593cff5a74532befe7ab69ce9d.
```

Github에서 풀 리퀘스트의 "Merge" 버튼을 클릭하면 자동으로 채워지는 메시지가 어땠나요?

```
Merge pull request #123 from someuser/somebranch
```

이처럼 커밋 메시지를 `명령문`으로 작성한다는 것은, **git의 빌트-인 컨벤션(Built-in Convention)을 그대로 따른다는 것을 의미합니다.** 때문에 커밋 제목을 `명령문`으로 작성하면, 자동 메시지로 채워진 커밋 사이에 자연스레 녹아듭니다. 앞선 항목에서 설명드린 `git shortlog`와 같은 타 명령어에도 연계되어 잘 어울리죠.

물론 이것은 제목이나 한 줄 커밋 메시지의 이야기이고요, 커밋 메시지의 본문을 `명령조`로 시작할 필요는 없습니다. 자연스럽게 과거형이나 현재형 시제를 사용하여 변경점을 서술하면 됩니다.

혼란스러운 점을 말끔히 해소할 수 있는 아주 간단한 팁을 한가지 알려드리겠습니다. 제목을 작성할 때 어떤 문법으로 작성해야 할지 잘 모르시겠다면, 다음 문장 뒤에 제목을 배치해서 문법이 잘 어울리나 확인해 보세요.

- If applied, this commit will `{제목}`

예문을 보여드리겠습니다.

- (If applied, this commit will) **Refactor subsystem X for readability**
- (If applied, this commit will) **Update getting started documentation**
- (If applied, this commit will) **Remove deprecated methods**
- (If applied, this commit will) **Release version 1.0.0**
- (If applied, this commit will) **Merge pull request #123 from user/branch**

안어울리는 문장을 쓰면 어떻게 되는지 볼까요?

- (If applied, this commit will) **Fixed bug with Y**
- (If applied, this commit will) **Changing behavior of X**
- (If applied, this commit will) **More fixes for broken stuff**
- (If applied, this commit will) **Sweet new API methods**

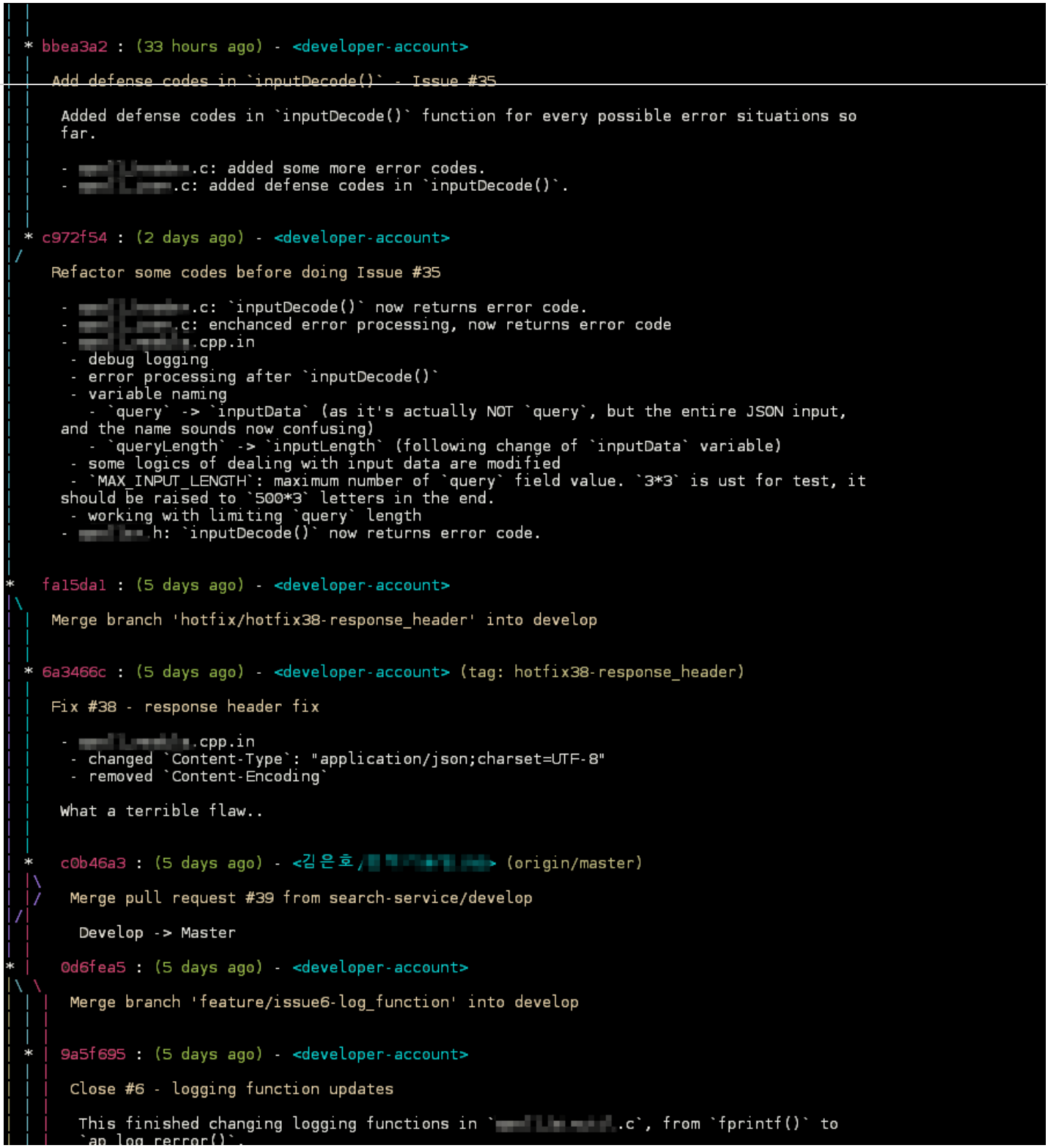
이 규칙은 오로지 "제목"에만 적용되고, "본문"을 작성할 때는 평서문으로 작성한다는 것을 다시 한 번 기억해 주세요.

## 6. 본문은 영문 기준 72자마다 줄 바꾸기

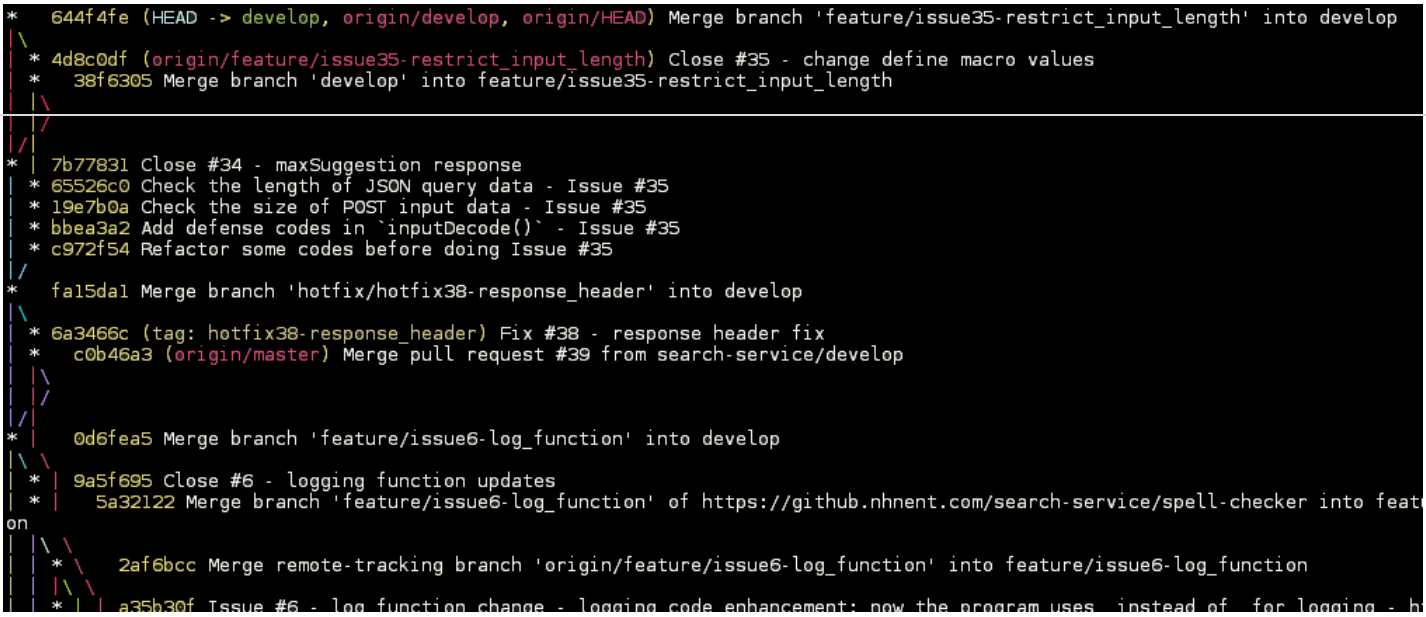
git은 자동으로 커밋 메시지 줄바꿈을 하지 않습니다. `git log`에 스타일을 지정해서 자동 줄바꿈이 적용된 로그 출력을 만들 수는 있지만요. 단순한 `git log` 명령어 입력만으로도 보기 좋은 메시지를 만들고자 한다면 적당한 위치에서 엔터키를 눌러 줄을 바꿔주세요. 그 적당한 위치로 `72자`를 추천드립니다.

> 꿀팁: `git log`에 스타일을 지정하는 방법을 모르셨다면, git이 설치된 시스템의 커맨드라인에 다음 명령어를 입력해서 사용해 보세요. 커스텀 스타일 옵션이 적용된 `git log`를 실행시켜주는 "alias"를 만들어주는 설정입니다. 한 번 설정한 이후 로그 확인은 `git log` 대신 `git lg`을 간단하게 입력하시면 됩니다. 자동 줄바꿈이 적용됨은 물론 Magically! 예뻐진 git 로그를 감상해 보세요. 제가 직접 편집해서 사용하고 있는 명령어입니다. 물론 기존의 `git log`도 원래대로 동작합니다. 이 설정을 `~/.gitconfig` 파일에 작성해두면 매번 입력할 필요없이 다음 git 실행부터 항상 적용됩니다.

```
$ git config --global alias.lg "log --graph --abbrev-commit --decorate --date=relative --format=format:'%C(bold red)%h%C(reset)
```




[그림1] git lg 실행 결과




[그림2] git lg --oneline 실행 결과

## 7. 본문은 어떻게 보다 무엇을, 왜 에 맞춰 작성하기

제목이 내용을 충분히 전달하는 것 같습니다. 좋은 사례로 Bitcoin Core 프로젝트에 실제로 사용된 커밋 메시지를 보여드리겠습니다.

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

    Simplify serialize.h's exception handling

    Remove the 'state' and 'exceptmask' from serialize.h's stream
    implementations, as well as related methods.

    As exceptmask always included 'failbit', and setstate was always
    called with bits = failbit, all it did was immediately raise an
    exception. Get rid of those variables, and replace the setstate
    with direct exception throwing (which also removes some dead
    code).

    As a result, good() is never reached after a failure (there are
    only 2 calls, one of which is in tests), and can just be replaced
    by !eof().

    fail(), clear(n) and exceptions() are just never called. Delete
    them.
```

# PLUS TIP! 커밋 메시지로 Github 이슈(issue)를 자동 종료시키기

만약 커밋 메시지를 영문으로 작성하신다면 좋은 팁 하나 더 소개해 드릴게요. Github에는 커밋 메시지에 특정한 단어를 사용해 자동으로 이슈를 종료시키는 편리한 기능이 탑재되어 있습니다. 이 예약어는 커밋 메시지 안의 어느 위치에서나 사용 가능합니다. 이제 커밋 메시지로 Github 이슈를 닫아보세요! 방법은 간단합니다.

키워드 #이슈번호

---

Github가 이슈 종료로 인식하는 키워드는 다음과 같습니다.

- close
- closes
- closed
- fix
- fixes
- fixed
- resolve
- resolves
- resolved

각 키워드마다 기능에 차이는 전혀 없고요, 문법이나 맥락에 맞게 사용하면 됩니다. 다만 `close` 계열은 일반 개발 이슈, `fix` 계열은 버그 픽스나 핫 픽스 이슈, `resolve` 계열은 문의나 요청 사항에 대응한 이슈에 사용하면 적당하다는 관례는 있습니다. 그럼 예문을 보여드릴게요.

```
# 제목에 이슈 한 개 닫기를 적용한 사례
Close #31 - refactoring wrap-up


* This is wrap-up of refactoring main code.
* main.c
  * removed old comments
  * fixed rest indentations
  * method extraction at line no. 35


# 본문에 이슈 여러 개 닫기를 적용한 사례
Update policy 16/04/02

* This closes #128 - cab policy, closes #129 - new hostname, and fixes #78 - bug on logging.
* cablist.txt: changed ACL due to policy update delivered via email on 16/04/02, @mr.parkyou
* hostname.properties: cab hostname is updated
* BeautifulDeveloper.java: logging problem on line no. 78 is fixed. The `if` statement is never happening. This deletes the `if`
```

이렇게 커밋을 작성하여 푸시하면, 푸시한 브랜치에 따라 이슈가 자동으로 닫힙니다. 예를 들어 Github 저장소의 default branch를 `master`로 설정해뒀을 경우, `master` 브랜치에 커밋 후 푸시하면 즉시 해당 번호의 이슈가 닫힙니다. `develop` 브랜치에 푸시했다면, 이슈는 닫히지 않고 있다가 나중에 `develop -> master Merge`가 되었을 때 알아서 닫힙니다.

기능을 좀 더 살펴보시면, 다른 저장소의 이슈를 닫는 것도 가능하지만, 여기서는 여기까지만 간단히 소개해 드리겠습니다. 더 자세한 내용은 밑의 링크를 참조하세요.

보기 좋은 git! 편리한 git! 누려보아요~

감사합니다.

# References

- Chris Beams : How to Write a Git Commit Message (https://chris.beams.io/posts/git-commit/)
- erlang/otp Wiki : Writing good commit messages (https://github.com/erlang/otp/wiki/Writing-good-commit-messages)
- git.kernel.org : The core git plumbing (https://git.kernel.org/cgit/git/git.git/tree/Documentation/SubmittingPatches)
- git-scm : Distributed Git - Contributing to a Project (https://git-scm.com/book/ch5-2.html)
- git-scm : Pretty Formats (https://git-scm.com/docs/pretty-formats)
- Github : Closing issues via commit messages (https://help.github.com/articles/closing-issues-via-commit-messages/)

김은호 기술본부
협업 도구, 코드의 형상 관리, 문서화에 관심이 많은 서버 프로그래머입니다. 잘 정제된 지식 정보 공유에 관심이 많습니다.

이전글 (/posts/107) 다음글 (/posts/105)