



과학기술정보통신부

nipa 정보통신산업진흥원  
National IT Industry Promotion Agency



2021

공개SW

페스티벌

ONLINE FESTIVAL

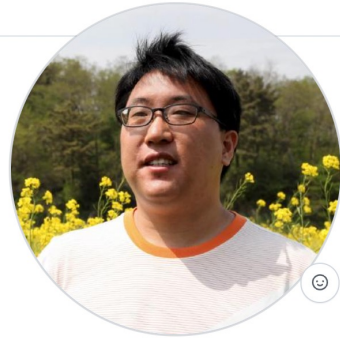


개발자와  
함께  
성장하는  
오픈소스

# CPython과 Backend.AI로 살펴보는 오픈소스 기여 프로세스

소속 : Lablup  
이름 : 김준기

# 연사 소개



**Joongi Kim**  
achimnol

Research-developer with focus on systems programming

Edit profile

193 followers · 33 following · 168

@lablup @Needworks  
South Korea  
me@daybreaker.info  
https://daybreaker.info  
@achimnol

## Achievements



## Highlights

☆ PRO

## Organizations



Overview Repositories 57 Projects Packages

## Pinned

lablup/backend.ai

Public

Backend.AI is a streamlined, container-based computing cluster orchestrator that hosts diverse programming languages and popular computing/ML frameworks, with pluggable heterogeneous accelerator su...

Shell 260 74

aio-lib/aiodocker

Public

Python Docker API client based on asyncio and aiohttp

Python 322 69

ANLAB-KAIST/NBA

Public

Network Balancing Act: A High-performance packet processing framework for heterogeneous processors

C++ 54 12

aiotools

Public

Idiomatc asyncio utilities

Python 103 7

Needworks/Textcube

Public

Textcube : Brand yourself! / Personalized web publishing platform with multi-user support

PHP 197 51

lablup/callosum

Public

An RPC Transport Library for asyncio

Python 15 2

1,495 contributions in the last year

Contribution settings

2021



Learn how we count contributions

Less More

@lablup

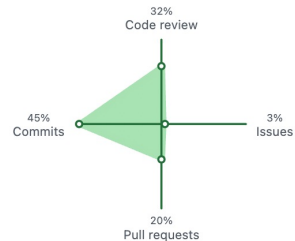
@python

@github

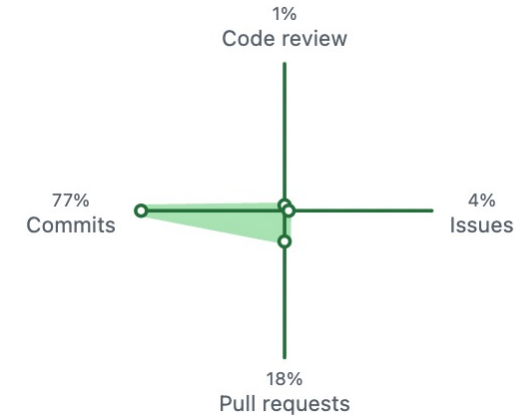
More

## Activity overview

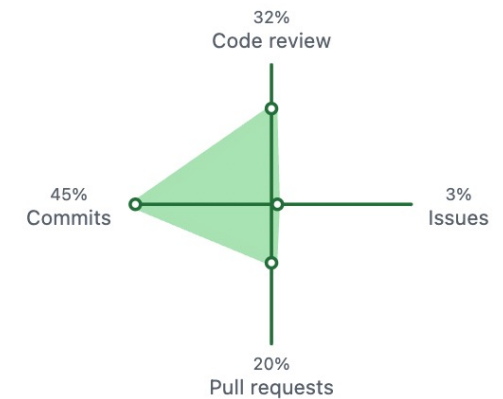
Contributed to lablup/backend.ai-manager, lablup/backend.ai-agent, lablup/backend.ai-client-py and 5 other repositories



~2020

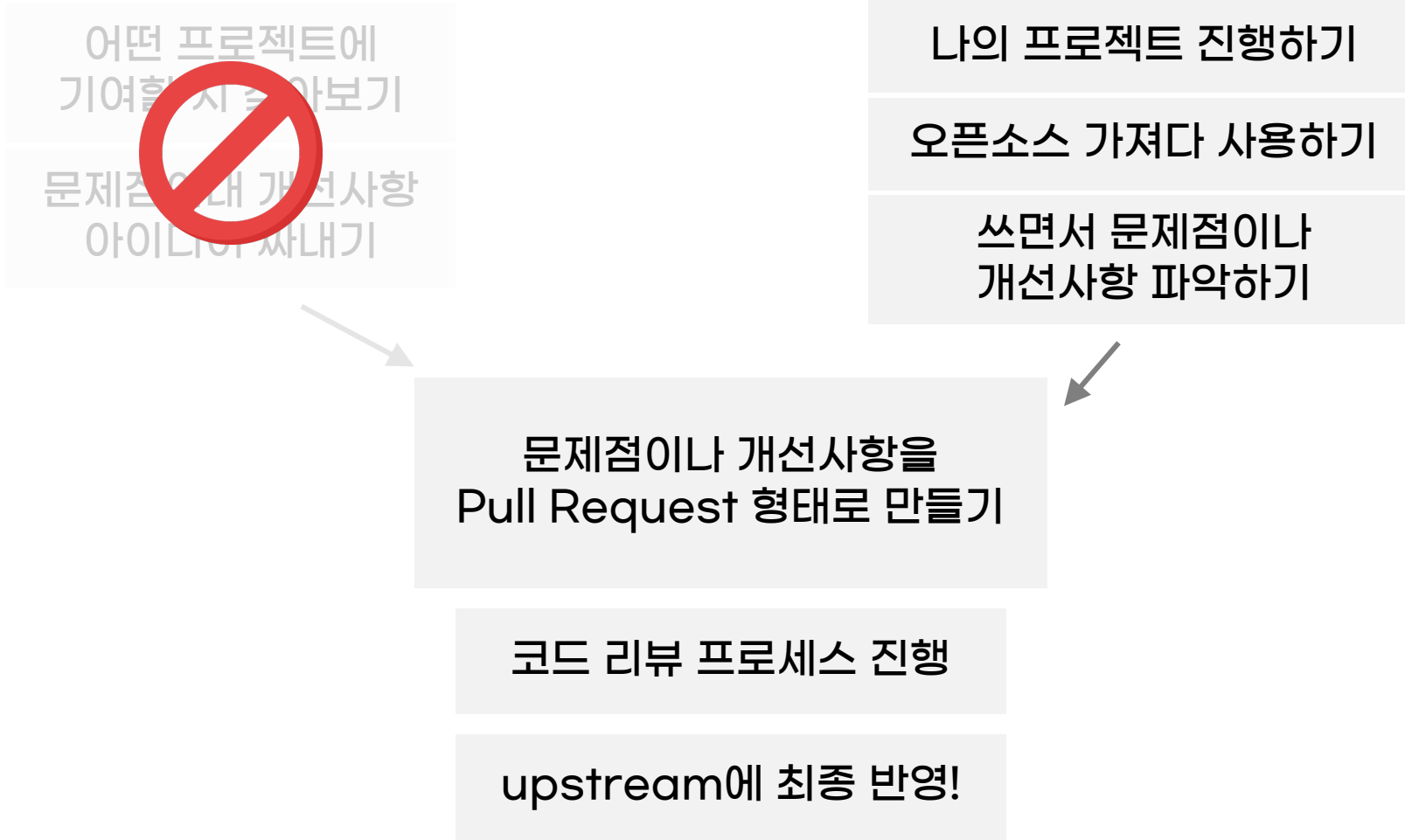


2021



- 코드 리뷰를 점점 많이 하다보니 아쉬운 점들
  - 기계적으로 검사할 있는 것들의 자동화 → 이건 기본!
    - ✓ 코딩 스타일
    - ✓ type 검사
    - ✓ 유닛 테스트 / 기능 테스트
  - 기계적으로 할 수 있을 것 같지만 바로 사용할 수 있는 검사 도구가 없는 경우
    - ✓ 특정 코딩 패턴을 금지하기 (예: 보안 취약점을 가져올 수 있다거나)
    - ✓ 코딩 스타일 도구로 검사가 되지 않거나 정의하기 복잡한 스타일 패턴
  - 테스트 코드를 잘못 작성한 경우
    - ✓ 테스트 대상이 되는 코드를 호출하지 않고 로직을 복붙한 경우
    - ✓ 테스트 코드가 정상적인 입력만 가정하고 넣어보는 경우
  - 테스트도 있고 돌아가는 코드지만 뭔가 이건 아닌 것 같을 때.... 🤖
    - ✓ 할많하않?!

# 오픈소스 기여 프로세스



# What if it goes wrong...

오픈소스 활성화를 위한  
좋은 취지에서 출발하였지만...

## DigitalOcean's Hacktoberfest is Hurting Open Source

Sep 30, 2020 | [74 comments](#) | [Tweet](#) | posted in [open source](#)

For the last couple of years, [DigitalOcean](#) has run [Hacktoberfest](#), which purports to “support open source” by giving free t-shirts to people who send pull requests to open source repositories.

In reality, **Hacktoberfest is a corporate-sponsored distributed denial of service attack against the open source maintainer community.**

ref) <https://blog.domenic.me/hacktoberfest/>

# 좋은 Pull Request 작성 방법

## ■ 나의 PR 작성하기

- 복잡하고 규모가 큰 프로젝트일수록, **작고 명확한 한 가지의 문제**를 해결하는 PR을 선호함
  - ✓ PR을 테스트 코드와 본문 코드를 합친 것으로 생각한다면, 본문 코드가 없을 때 실패하고 있을 때 성공하는 명확한 테스트 코드를 제공하는 것이 좋음
- 새로운 기능을 추가할 때는 다짜고짜 PR부터 작성하기보다는 이슈 제기를 통해 기존 개발자나 기여자들의 의중을 먼저 파악해야 함
  - ✓ 프로젝트 방향과 달라서 일부러 구현을 안 하고 있는 것일 수도 있음
- 로컬 개발환경 설정해서 직접 빌드·실행해보고 리뷰를 요청할 것
  - ✓ 리뷰어가 딱 돌려봤는데 한 번이라도 실행해봤으면 발견했을 에러가 나면 답-뻑침... 🤨

# 좋은 Pull Request 작성 방법

## ■ 기여 방법 및 주의사항 파악하기

- README, CONTRIBUTING, INSTALL, TESTING 같은 텍스트 파일이 프로젝트 최상위 디렉토리에 있다면 꼭 한번씩 읽어보기
- GitHub에서 새 이슈 작성을 눌렀을 때 요구하는 양식이 있다면 미리 한번 확인해보고 필요한 내용을 먼저 파악하기

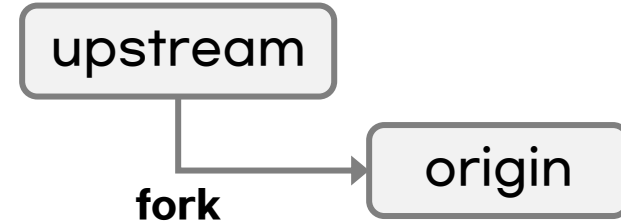
## ■ 기존의 유사한 PR 찾아보기

- 자주 언급되는 지적사항이 무엇인가?
- 커밋 메시지 및 changelog 작성을 어떤 식으로 하는가?
- 자동으로 검사하도록 적용된 lint, type 검사 툴들이 어떤 것인가?
  - ✓ 내가 로컬에서 동일한 도구로 테스트하려면 어떻게 설치·설정하는가?
- 클래스, 함수, 변수 등의 이름을 어떤 뉘앙스로 짓는가? (업계 용어를 알아야 함!)
  - ✓ 코드는 돌아가는데 이런 게 이상하면 리뷰어가 왜 이걸 굳이 바꿔야 하는지 설명하는 것이 오히려 더 어려울 때가 있다...

# 알아두면 좋은 용어

## ▪ origin

- 내가 작업하고 있는 프로젝트(의 git 저장소)
- fork한 경우 복제본 저장소 (보통 내 깃헙 계정)



## ▪ upstream

- 내가 작업하고 있는 프로젝트가 의존하는 라이브러리나 프레임워크(의 git 저장소)
- fork한 경우 원본 저장소 (보통 다른 누군가의 깃헙 계정)

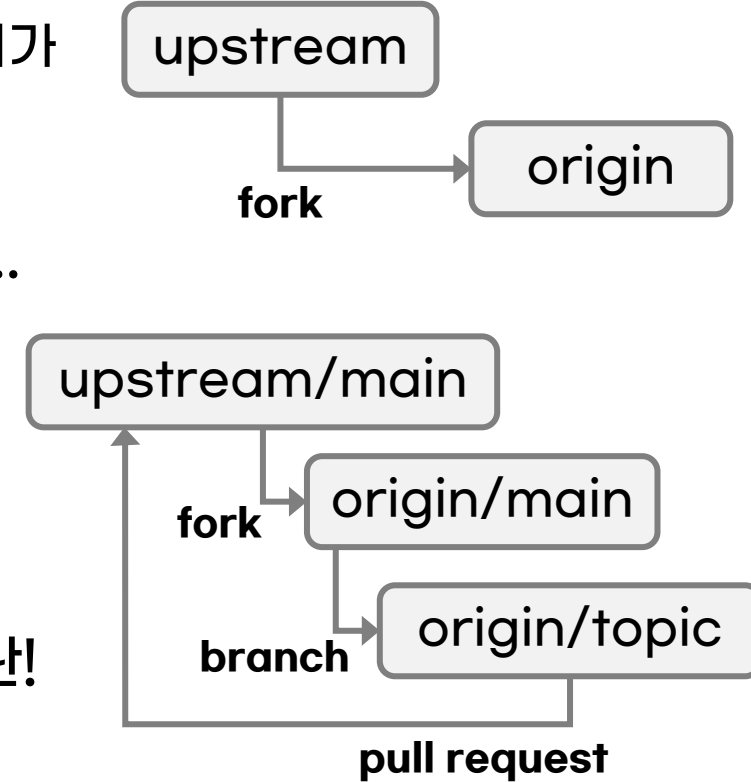
## ▪ "엇, 이건 upstream 이슈라서 해결이 불가능한데요..."

- 해석: 내 코드의 문제가 아니고 내가 의존하고 있는 다른 라이브러리의 문제라서, 거기서 문제를 해결해야 고칠 수 있음.
- 이 경우 우회 방법(workaround)을 찾거나 그 프로젝트에 직접 이슈 제기를 해야 함
- 오픈소스 기반으로 상용 제품을 개발할 때 흔히 겪는 어려움 (안 되어도 되게 해야 하니까...)

✓ 어떤 경우는 upstream 릴리즈 전까지 fork 후 임시 빌드를 따로 만들어서 함께 배포

# Git 및 GitHub 사용 시 주의사항

- fork 후 main 브랜치에 바로 작업하지 말고 별도 브랜치를 만들 것!
  - upstream의 변경사항 발생 시 이것을 가져와 병합할 때 로컬 브랜치가 있으면 충돌이 있을 때 이를 미뤄두었다가 해결한다거나 병합이 잘못되었을 때 되돌린다거나 하기 쉽다.
- forced-push는 자신의 작업 브랜치 내에서 하는 건 괜찮지만...
  - 리뷰어가 이미 봤던 코드까지 덮어치는 경우 꼭 리뷰어에게 forced-push 한다는 걸 미리 알려주고 하는 것이 좋다.
- allow edits from maintainers 옵션은 켜두자
  - 굳이 리뷰로 이야기하기 애매한 자잘한 fix를 빠르게 하는 데 도움
- 항상 작업 시작 전 자신의 작업 브랜치라도 pull을 일단 하는 습관!
- merge와 conflict 해결하는 과정, rebase 기능은 미리 연습하고 익숙해지자.



💡 origin을 git clone한 후 git remote add로 upstream 저장소 주소를 추가하면 main 브랜치 동기화 가능

# GitHub 조직 계정 사용 시 주의사항

- 사내 프로젝트에 PR을 날리고자 하는 경우
  - 각 회사의 fork 정책을 먼저 살펴볼 것
  - **Lablup**: 사내 프로젝트는 fork하지 않고 원래 저장소에 직접 브랜치를 만들도록 함 (push 권한이 없는 경우 명시적으로 요청할 것! 보통 깜빡한 경우가 많아서...)
- 회사용 GitHub 조직 계정에 외부 프로젝트를 fork 떴하고자 하는 경우
  - 회사에서 외부 오픈소스 기여에 대해 어떤 정책을 가지고 있는지 확인할 것
  - (너무 당연하지만) 원래 오픈소스의 라이선스를 지킬 것
  - **Lablup**: 제품에 보다 직접적이고 크리티컬한 영향을 주는 경우나 내부 협업이 필요한 경우 조직 계정으로 fork
  - **Lablup**: 팀원 개인이 수정할 수 있는 범위의 자잘한 기여는 각자 개인 계정에서... 제품 개발에 도움이 되는 부분이라면 업무시간에 해도 ok

## ■ 기여자로서의 마음가짐

- 내가 이 코드를 리뷰한다면 어떻게 할까?
- 리뷰어의 인지적 부하를 줄이자.
  - ✓ 기존 병합된 PR들과 가능한 비슷한 tone & manner 유지하자.
  - ✓ 변경 내용 중 어떤 게 중요한 것인지 잘 알 수 있게 설명하자.

## ■ 리뷰어로서의 마음가짐

- accept/reject에는 되도록 분명한 근거를 알려주자.
  - ✓ 항상 내가 더 많은 정보를 갖고 있음을 명심하자.
- 맘에 안 드는 부분이 있다면 더 좋은 대안을 제시하고 스스로 선택할 수 있도록 유도하자.

- 이 코드는 내가 '소유'한 것이 아니다. (저작권은 그럴지라도...)
- 리뷰의 목적은 더 나은 코드를 작성하기 위함이지, 코드 작성자를 평가하기 위한 것이 아니다.
- 비판보다는 서로 설득한다고 생각하는 것이 좋다.

# 나의 경험 - CPython

- 본문 코드 작성은 쉬운데 테스트 케이스 작성이 어려움
  - 예) bpo-45416의 경우 2줄을 삭제하면 되는데 테스트만 50줄 가량 작성
- 코어 커미터 자격을 가진 나동희 님의 도움으로 빠른 리뷰 진행
  - 다른 코어 커미터들 중 누구를 mention해야 빠르게 진행이 될지 알고 있는 것 중요
    - ✓ 예) asyncio를 잘 아는 사람은 1st1, asvetlov, aeros 등
  - 요즘 코어 커미터들이 어떤 부분에 관심을 갖고 있는지에 대한 트렌드 파악
    - ✓ 예) Python 3.11부터는 예전보다 과감한 성능 개선 시도들을 많이 받고 있음
    - ✓ 본인 관심 분야와 잘 맞는 경우 큰 도움이 될 수 있음

 **bpo-41229: Update docs for explicit aclose()-required cases and add contextlib.aclosing() method** ✓ CLA signed

expert-asyncio

 automerger

type-enhancement

#21545 by achimnol was merged on 2 Nov 2020 • Approved

 **bpo-45416: Fix use of asyncio.Condition() with explicit Lock objects** ✓ CLA signed type-bugfix

#28850 by achimnol was merged 18 days ago • Approved

## ■ 알지만 일부러 안 하는 경우

- 서버 실행 시 기존의 "--debug" 옵션에 더하여 "--warning", "--error"와 같은 추가 로그레벨 인자를 추가하자는 제안
  - ✓ 기존 "--debug" 옵션은 로그레벨을 강제 오버라이드하는 것 외에 실제 디버깅에 필요한 몇 가지 기능들을 추가로 활성화하는 의미가 포함되어 있음
  - ✓ '로그 레벨을 변경한다'라는 하나의 동작에 대해서는 로그 레벨의 '값'에 해당하는 경우의 수마다 각각 플래그가 존재하기보다는 '로그 레벨'을 나타내는 단일 플래그를 사용하는 것이 Click과 같은 명령줄 해석기를 사용하는 코드를 깔끔하게 유지할 수 있음

## ■ 영어의 문제

- 코드에 추가한 것은 "--advertised-host"인데 주석과 문서, 변경 로그에는 모두 "--advertise-host"라고 적은 경우
- "Formerly"를 의도했으나 "Formally"라고 적어서 의미 혼동이 있었던 경우
- 변수명에 사용한 영어가 콩글리쉬인 경우 / 틀린 뜻은 아닌데 업계 용어가 아닌 경우

## ▪ 자동 검증이 어려운 코드 패턴

- `asyncio.create_subprocess_shell()` 대신 `asyncio.create_subprocess_exec()`를 사용할 것
  - ✓ 외부로부터의 입력값을 조합하여 셸 명령어를 실행하는 경우 잠재적 보안 취약점
  - ✓ 처음부터 `exec()`를 사용하면 이런 문제를 피할 수 있으나, redirection 같은 기능을 사용하려면 해당 부분을 직접 `proc.stdout`을 읽어들이는 식으로 구현해야 함
- 예외 처리를 스택의 어느 부분에서 하는 것이 좋은가?
  - ✓ 해당 예외가 발생할 수 있는 스택 위치에서 되도록 가까운 곳에서 할수록 좋음
- 예외 객체의 타입이 아닌 내용을 읽어야만 구분할 수 있는 오류 처리
  - ✓ 내용을 읽어 원하는 특정 오류를 구분하는 것까지는 잘 작성하였는데 '그 외의 경우'에 대한 `raise` 구문을 빼먹는 경우
- ...

# 야생 오픈소스 vs. 회사

## ■ 야생 오픈소스 프로젝트에서의 PR과 코드 리뷰

- 리뷰 자체보다도 이 일을 할지 말지에 관한 설득의 과정이 훨씬 지난할 수 있음
- 무주공산 : 아무도 책임을 지지 않는다... 당장 커미터 발목 잡는 문제가 아니면 기약 없음...
  - ✓ "설득의 심리학" : 때로는 누군가를 찍어서 부탁하는 것이 빠르게 진행된다
  - ✓ 누구를 찍어야 할 지 알아내는 것이 센스이자 능력이자 인맥(!)

## ■ 사내 프로젝트에서의 PR과 코드 리뷰

- 고객 요구·비즈니스 목표에 따라 작업 범위가 큰 과감한 작업들(epic...)이 필요함
  - ✓ 이런 경우는 작고 명확한 하나의 변경사항으로 취급하기 어려움
  - ✓ 리뷰보다는 사전 협의와 설계가 더 중요
- 프로젝트가 성숙하고 팀의 규모가 커지면, 즉, 복잡도가 커지면 야생 오픈소스에 가까워진다.
  - ✓ 복잡도가 높을 때 어떻게 대응하게 되는가? + 어떻게 접근해야 하는가?
- 내부 주니어 개발자들을 키우고 프로젝트를 성공시켜야 하는 입장이므로 남남끼리 대면하는 야생 오픈소스보다는 그래도 좀더 친절할 가능성이 높음
  - ✓ 여러분 회사에서 코드리뷰를 하고 있다면 성장 기회로 활용하세요~!

# 감사합니다



과학기술정보통신부



정보통신산업진흥원  
National IT Industry Promotion Agency