오픈소스 프로젝트 과제

김민중
kl4314@likelion.org

January 6, 2018

Contents

1 OSS 주요 라이선스 **3**
1.1 GPL(GNU General Public License) 2.0 . . . 3
1.1.1 주요내용 . . . 3
1.1.2 공개범위 . . . 3
1.2 GPL(GNU General Public License) 3.0 . . . 4
1.2.1 주요내용 . . . 4
1.2.2 공개범위 . . . 4
1.3 LGPL(GNU Lesser General Public License) 2.1 . . . 5
1.3.1 주요내용 . . . 5
1.3.2 공개범위 . . . 5
1.4 BSD(Berkeley Software Distribution) License . . . 6
1.4.1 주요내용 . . . 6
1.4.2 공개범위 . . . 6
1.5 MIT(Massachusetts Institute of Technology) License . . . 6
1.5.1 주요내용 . . . 6
1.5.2 공개범위 . . . 7

2 OSS 소개 **7**
2.1 Elasticsearch . . . 7
2.1.1 개발 과정 . . . 7

3 Git **9**
3.1 사용법 . . . 9
3.2 Git과 Github, GitLab의 차이점 . . . 14

3.3 Git의 repository 14
3.4 Git 주요 명령어 15

1 OSS 주요 라이선스

1.1 GPL(GNU General Public License) 2.0

1.1.1 주요내용

GPL 2.0은 현재 가장 많은 공개SW가 채택하고 있는 라이선스로 자유소프트웨어 재단에서 만들고 배포하였다. 공개 SW 라이선스 중에서 의무사항이 매우 강력한 라이선스이다.

복제와 배포가 이루어 질 때는 본 허가서와 프로그램에 대한 보증이 제공되지 않는다는 사실에 대해서 언급되었던 모든 내용을 그대로 유지시켜야 하며, 영문판 GPL 라이선스를 함께 제공해야 한다.

파일을 개작할 때는 파일을 개작한 사실, 내용 및 그 날짜 등을 파일 안에 명시해야 한다.

소프트웨어를 수정하거나 새로운 소프트웨어를 링크 시키는 경우 GPL에 의해 소스코드를 제공해야 한다.

프로그램의 일부를 본 허가서와 배포 기준이 다른 자유 프로그램과 함께 결합하고자 할 경우에는 해당 프로그램의 저작자로부터 서면 승인을 받아야 한다.

자신의 특허를 구현한 프로그램을 GPL로 배포하는 경우에는 그 프로그램을 GPL 조건에 따라 이용하는 이용자에게 특허에 대한 사용료를 받을 수 없으며, 제3자의 특허를 구현한 그 프로그램인 경우에는 그 특허권자가 GPL조건에 따라 이용하는 프로그램 이용자에 대하여 특허 사용료를 받지 않을 때에만 그 프로그램을 GPL로 배포하는 것이 가능하다.

1.1.2 공개범위

GPL 프로그램의 소스 코드를 개발중인 프로그램 코드에 삽입하거나 링크시켜 이를 배포하는 경우에 개발한 프로그램의 소스코드도 공개해야 한다.

다만 원본 프로그램과 별개의 독립된 프로그램을 GPL프로그램과 같이 배포하는 경우에는 GPL 2.0이 아닌 다른 라이선스조건에 의해 배로 할 수 있다.

하지만 구체적으로 어떠한 경우가 파생물에 해당하는지는 또는 독립된 프로그램의 단순한 집합물에 해당하는지를 구별하는 것은 쉽지 않다.

FSF는 이러한 혼란을 막기 위해 GPL FAG를 통하여 몇 branch의 구별 기준을 제시하고 있다.

1.2 GPL(GNU General Public License) 3.0

1.2.1 주요내용

기본적인 내용은 GPL 2.0과 비슷하지만 GPL 3.0에서는 DRM관련 내용, 소프트웨어 특허문제, 양립성 문제 등이 추가되었다.

GPL 3.0의 소스코드를 사용자 제품에 포함시키거나 혹은 그와 함께 배포하는 경우에는 해당 소스에 설치정보를 함께 제공해야 한다.

'설치정보'란 소프트웨어를 수정하여 해당 제품에 설치하고 실행하는데 필요한 방법, 절차, 인증키 혹은 여타 정보 모두를 의미한다.

다만 소프트웨어가 롬에 설치된 경우처럼 해당 제품의 제조업체나 여타 제 3자도 수정된 코드를 제품에 설치할 수 없는 경우에는 설치정보를 제공하지 않아도 된다.

특허와 관련해서 원래의 소스코드를 개선하여 배포한 기여자의 경우 자신이 기여한 부분에 대해서는 비차별적이고 특허 사용료가 없다는 내용의 라이선스를 제공해야 한다.

특허와 관련해서 라이선스 등으로부터 특허소송이 제기되는 경우 소송을 제가한 날에 특허소송을 제기한 라이선스의 공개SW 라이선스는 종료 된다.

소프트웨어를 작성하고자 할 경우 기준에 만들어진 코드를 재사용하거나 결합하는 경우가 많은데 결합되는 각 코드의 라이선스가 상호 상충되는 경우 같이 사용할 수 없다.

1.2.2 공개범위

GPL 2.0과 동일하다.

1.3 LGPL(GNU Lesser General Public License) 2.1

1.3.1 주요내용

FSF에서 LGPL을 만들어 사용하고 있는 이유는 공개 SW의 사용을 장려하기 위한 전략적인 차원이다

상용 라이브러리와 동일한 기능을 제공하고 공개SW 라이브러리에 GPL과 같은 엄격한 라이선스를 적용하게 되면 라이브러리를 사용하는 SW의 소스코드를 공개해야하기 때문에 상용SW 개발자들은 공개SW 라이브러리의 사용을 꺼려할 것이다.

오히려 이미 널리 사용되고 있는 상용라이브러리와 동일한 기능을 제공하는 공개 SW 라이브러리를 LGPL로 배포하여 원 프로그램의 소스코드는 공개하지 않고, 이에 사용된 해당 공개SW 라이브러리의 소스코드만 공개하게 함으로써 공개 SW 라이브러리의 사용을 장려하고 사실상의 표준으로 유도하는 한편 관련된 다른 공개SW를 보다 더 많이 사용할 수 있도록 하겠다는 것이다.

소프트웨어를 배포하는 경우 저작권 표시, 보증 책임이 없다는 표시 및 LGPL에 의해 배포된다는 사실을 명시해야 한다.

LGPL 라이브러리의 일부를 수정하는 경우 수정한 라이브러리의 소스코드 공개해야 한다.

LGPL 라이브러리에 응용프로그램의 소스코드를 공개할 필요가 없다.

다만 사용자가 라이브러리 수정 후 동일한 실행 파일을 생성할 수 있도록 Static Linking시에는 응용프로그램의 Object Code를 제공해야 한다.

특허의 경우 GPL과 동일하다.

1.3.2 공개범위

어떠한 경우에도 LGPL 소프트웨어 자체는 공개해야 하지만 LGPL 소프트웨어와 링크되는 부분의 소프트웨어 소스코드는 공개해야할 의무가 발생하지 않으므로 기업의 입장에서는 LGPL소프트웨어를 좀 더 선호하게 되었다.

사용여부 명시 등은 GPL과 동일하게 반영하면 되고 공개해야할 소스코드의 공개 역기 GPL과 동일한 방식을 이용하면 된다.

LGPL은 일정한 요건을 충족시키는 경우 LGPL 라이브러리를 이용하는 프로그램, 다시 말해 링킹(linking)을 통해 LGPL 라이브러리와 함께 작동하도록 설계된

프로그램을 배포할 경우에는 소스코드를 제공하지 않아도 된다.

1.4 BSD(Berkeley Software Distribution) License

1.4.1 주요내용

BSD 라이선스는 SW의 소스코드를 공개하지 않아도 되는 대표적인 공개SW 라이선스이다.

BSD 라이선스의 허용범위가 넓은 이유는 BSD 라이선스로 배포되는 프로젝트가 미국정부에서 제공한 재원으로 운영되었기 때문이며, 이는 SW에 대한 대가를 미국 국민의 세금으로 미리 지불했기 때문에 사람들에게 그들이 원하는 방식으로 소프트웨어를 사용하거나 만들 수 있도록 허가한 것이다.

따라서 BSD 라이선스의 소스코드를 이용하여 새로운 프로그램을 개발하여도 새로운 프로그램의 소스코드를 공개하지 않고 BSD가 아닌 다른 라이선스를 적용하여 판매할 수 있다.

SW를 배포하는 경우 저작권표시, 보증 책임이 없다는 내용을 표시해야 한다.

수정 프로그램에 대한 소스코드의 공개를 요구하지 않기 때문에 상용 SW에 무재한 사용이 가능하다.

1.4.2 공개범위

BSD 라이선스의 경우 의무사항만 준수한다면 소스코드를 공개하지 않아도 된다.

1.5 MIT(Massachusetts Institute of Technology) License

1.5.1 주요내용

- MIT 라이선스(MIT License)는 미국 매사추세츠 공과대학교(MIT)에서 해당 대학의 소프트웨어 공학도들을 돕기 위해 개발한 라이선스다. 라이선스와 저작권 관련 명시만 지켜주면 되는 라이선스로 가장 느슨한 조건을 가진 라이선스 중 하나기 때문에 인기가 많다. 있다.

1.5.2 공개범위

의무사항만 준수한다면 소스코드를 공개하지 않아도 된다.

2 OSS 소개

2.1 Elasticsearch

2.1.1 개발 과정

엘라스틱서치의 탄생은 자상한 남편의 이야기로 시작한다. 엘라스틱 공동 설립자이자 현재 CTO 직을 맡고 있는 샤이 배넌(Shay Banon)은 2004년, 런던으로 이사를 가야 했다. 당시 그의 아내가 요리사의 길을 준비하고 있어 이를 지원해주기 위해서였다. 새로운 곳으로 이사를 하다 보니 샤이 배넌은 개발자로서 새 직장을 구하는 데 시간이 필요했고, 자연스레 집에 있는 시간이 많았다. 그는 새로운 기술을 공부하면서 남는 시간을 보냈는데, 마침 그의 아내는 요리 수업 시간에 배워온 자료를 정리하고 있었다. 그런 모습을 본 샤이 배넌은 아내에게 맞춤화된 요리법 검색 서비스를 만들어주기로 결심했다. 그때 발견한 기술이 바로 '루신'이다. 루신은 자바에서 사용할 수 있는 검색 기술 라이브러리로, 당시 꽤 유명한 오픈소스 기술이었다. 샤이 배넌은 루신을 기반으로 필요한 검색 기능을 구축하기 시작했고, 자체 오픈소스 기술인 '컴파스(Compass)'를 개발하게 한다.

컴파스에 투자한 지 몇 달이 지나자, 샤이 배넌은 해당 기술이 요리법 검색 이상으로 활용될 수 있다는 것을 깨달았다. 검색 기술을 적용할 수 있는 분야는 무궁무진했기 때문이다. 실제로 현재 엘라스틱 고객군은 금융, 미디어, 유통, IT 등 다양하다.

컴파스를 기반으로 조금 더 정교하게 만들어진 기술이 바로 엘라스틱서치다. 샤이 배넌은 취업 후 틈틈이 엘라스틱서치를 개발했지만, 그 성장 속도는 생각보다 빨랐다. 그는 아예 직장을 그만두고 엘라스틱서치에만 집중하기로 했다. 그러면서 뜻이 맞는 다른 개발자 3명과 함께 엘라스틱서치란 스타트업을 설립하게 된다. 아내를 위해 만들려던 요리검색 기술은 엘라스틱서치에 집중하느라 결국 완성하지 못했다고 한다.

버그 픽스의 과정은 https://github.com/elastic/elasticsearch/issues이 깃허브 주소에 나와 있습니다.

아래의 표에서 볼 수 있듯이 현재도 개발중에 있으며 현재는 2.4버전까지만 지원합니다.

버전	원래 출시일	마지막 버전	출시일	유지 보수 상태
0.4	2010-02-08	0.4.0	2010-02-08	더 이상 지원하지 않음
0.5	2010-03-05	0.5.1	2010-03-09	더 이상 지원하지 않음
0.6	2010-04-09	0.6.0	2010-04-09	더 이상 지원하지 않음
	⋮	⋮	⋮	
	⋮	⋮	⋮	
2.4	2016-08-31	2.4.6	2016-07-27	현재 지원하고 있음
5.0	2016-10-26	5.0.2	2016-11-29	현재 지원하고 있음
5.1	2016-12-08	5.1.2	2017-01-12	현재 지원하고 있음
5.2	2017-01-31	5.2.2	2017-02-28	현재 지원하고 있음
5.3	2017-03-28	5.3.3	2017-06-01	현재 지원하고 있음
5.4	2017-05-04	5.4.3	2017-06-27	현재 지원하고 있음
5.5	2017-07-08	5.5.3	2017-07-06	현재 지원하고 있음
5.6	2017-09-11	5.6.3	2017-10-10	최신
6.0	2017-05-09	6.0.0-rc1	2017-09-28	최신 프리뷰

Table 1: 위표는 Elasticsearch의 출시 과정입니다.

3 Git

3.1 사용법

새로운 저장소 만들기

폴더를 하나 만들고, 그 안에서 아래 명령을 실행하세요.

git init

새로운 git 저장소가 만들어집니다.

저장소 받아오기

로컬 저장소를 복제(clone)하려면 아래 명령을 실행하세요.

git clone /로컬/저장소/경로

원격 서버의 저장소를 복제하려면 아래 명령을 실행하세요.

git clone 사용자명@호스트:/원격/저장소/경로

흐름

여러분의 로컬 저장소는 git이 관리하는 3개의 branch로 구성 돼 있습니다. 첫번째 branch인 작업 디렉토리(Working directory)는 실제 파일들로 이루어져있고, 두번째 branch인 인덱스(Index)는 준비 영역(staging area)의 역할을 하며, 마지막 branch인 HEAD는 최종 확정본(commit)을 나타냅니다.

add와 commit

변경된 파일은 아래 명령어로 (인덱스에) 추가할 수 있습니다..

git add <파일 이름>
git add *

이것이 바로 git의 기본 작업 흐름에서 첫 단계에 해당돼요. 하지만 실제로 변경 내용을 확정하려면 아래 명령을 내려야 한답니다.

git commit -m "이번 확정본에 대한 설명"

자, 이제 변경된 파일이 HEAD에 반영됐어요. 하지만, 원격 저장소에는 아직 반영이 안 됐답니다.

변경 내용 push하기

현재의 변경 내용은 아직 로컬 저장소의 HEAD 안에 머물고 있어요. 이제 이 변경 내용을 원격 서버로 올려봅시다. 아래 명령을 실행하세요.

git push origin master

(다른 branch를 push하려면 master를 원하는 branch 이름으로 바꿔주세요.)

만약 기존에 있던 원격 저장소를 복제한 것이 아니라면, 원격 서버의 주소를 git에게 알려줘야 합니다.

git remote add origin <원격 서버 주소>

이제 변경 내용을 원격 서버로 발행할 수 있어요.

branch 만들기

branch는 안전하게 격리된 상태에서 무언가를 만들 때 사용해요. 여러분이 저장소를 새로 만들면 기본으로 master branch가 만들어집니다. 이제 다른 branch를 이용해서 개발을 진행하고, 나중에 개발이 완료되면 master branch로 돌아와 병합하면 돼요. 아래 명령으로 "hotfix"라는 이름의 branch를 만들고 갈아탑니다.

git checkout -b hotfix

아래 명령으로 master branch로 돌아올 수 있어요.

git checkout master

아래 명령으로는 branch를 삭제할 수 있어요.

git branch -d hotfix

여러분이 새로 만든 branch를 원격 저장소로 전송하기 전까지는 다른 사람들이 접근할 수 없어요.

git push origin <branch 이름>

갱신과 병합(merge)

여러분의 로컬 저장소를 원격 저장소에 맞춰 갱신하려면 아래 명령을 실행하세요.

git pull

이렇게 하면 원격 저장소의 변경 내용이 로컬 작업 디렉토리에 받아지고(fetch), 병합(merge)된답니다. 다른 branch에 있는 변경 내용을 현재 branch(예를 들면, master branch)에 병합하려면 아래 명령을 실행하세요.

git merge <branch 이름>

첫번째 명령이든 두번째 명령이든, git은 자동으로 변경 내용을 병합하려고 시도해요. 문제는, 항상 성공하는 게 아니라 가끔 충돌(conflicts)이 일어나기도 한다는 거예요. 이렇게 충돌이 발생하면, git이 알려주는 파일의 충돌 부분을 여러분이 직접 수정해서 병합이 가능하도록 해야 하죠. 충돌을 해결했다면, 아래 명령으로 git에게 아까의 파일을 병합하라고 알려주세요.

git add <파일 이름>

변경 내용을 병합하기 전에, 어떻게 바뀌었는지 비교해볼 수도 있어요.

git diff <원래 branch> <비교 대상 branch>

꼬리표(tag) 달기

소프트웨어의 새 버전을 발표할 때마다 꼬리표를 달아놓으면 좋아요. (물론 꼬리표는 SVN 등에 이미 존재하는 기능이지요.) 아래 명령을 실행하면 새로운 꼬리표인 1.0.0을 달 수 있어요.

git tag 1.0.0 1b2e1d63ff

위 명령에서 1b2e1d63ff 부분은 꼬리표가 가리킬 확정본 식별자입니다. 아래 명령으로 확정본 식별자를 얻을 수 있어요.

git log

확정본 식별자의 앞부분 일부만 입력해도 꼬리표를 붙일 수 있지만, 그 일부분이 반드시 고유하다는 조건이 필요해요.

로컬 변경 내용 되돌리기

만약 여러분이 (물론 그럴 일은 없겠지만 ;) 실수로 무언가 잘못한 경우, 아래 명령으로 로컬의 변경 내용을 되돌릴 수 있어요.

git checkout – <파일 이름>

위 명령은 로컬의 변경 내용을 변경 전 상태 (HEAD) 로 되돌려줘요. 다만, 이미 인덱스에 추가된 변경 내용과 새로 생성한 파일은 그대로 남는답니다.

만약, 로컬에 있는 모든 변경 내용과 확정본을 포기하려면, 아래 명령으로 원격 저장소의 최신 이력을 가져오고, 로컬 master 가지가 저 이력을 가리키도록 할 수 있어요.

git fetch origin
git reset –hard origin/master

3.2 Git과 Github, GitLab의 차이점

Git

Git은 버전 관리를 위한 도구이다. 전체 개발 소스를 공유하면서 개발 파트를 나누어 공유한다.

Github

Github는 좋은 Web UI를 제공하며 여러 질문과 답변, 이슈 사항을 기록하기 위한 Issues 페이지를 제공하고 Wiki를 제공한다. Github는 Git 기반의 커뮤니티 공간이라고 말할 수 있다. 보통 OpenSource Project가 진행되어진다
private repository로 사용할 경우 일부 비용을 내야해 개인 프로젝트 저장소로는 적합하지 않다.

Gitlab

Github와 Gitlab의 차이점은 Gitlab은 Private project더라도 비용이 없고 비용을 추가로 내면 기술지원도 받을 수 있다.

3.3 Git의 repository

repository

저장소(Git repository)란 말그대로 파일이나 폴더를 저장해 두는 곳입니다. 그런데 Git 저장소가 제공하는 좋은 점 중 하나는 파일이 변경 이력 별로 구분되어 저장된다는 점입니다. 비슷한 파일이라도 실제 내용 일부 문구가 서로 다르면 다른 파일로 인식하기 때문에 파일을 변경 사항 별로 구분해 저장할 수 있습니다.

원격 저장소와 로컬 저장소)

원격 저장소(Remote Repository): 파일이 원격 저장소 전용

서버에서 관리되며 여러 사람이 함께 공유하기 위한 저장소입니다.
로컬 저장소(Local Repository): 내 PC에 파일이 저장되는 개인 전용 저장소입니다.

3.4 Git 주요 명령어

명령어	기능
git add [파일]	stage area에 파일을 추가하여 commit 할 수 있도록 한다.
git commit -m "mention"	stage area에 있는 파일들을 commit 한다.
git commit -a -m "mention"	이미 추가된 파일이 수정 중인 상황에서 stage area에 올리지 않아도 stage area에 올리고 바로 commit 한다.
git remote add [저장소] [저장소주소]	원격 저장소를 추가한다.
git remote -v	원격 저장소 목록을 보여준다.
git clone [주소] [저장될 폴더]	git 원격 저장소에 있는 프로젝트를 내려받는다.
git clone –depth [숫자] [주소]	프로젝트가 많은 커밋들을 가지고 있을 경우 내려받는데 오래 걸리므로 depth 옵션을 사용하면 해당 숫자만큼의 최신 커밋들만 가지고 프로젝트를 내려받는다.
git push origin master	origin 원격 저장소에 master 브랜치에 추가된 스냅샷들을 올린다.
git tag	만들어진 태그 목록을 보여준다.
git tag [태그]	태그를 만든다.
git tag -l ‘v1.0′	1.0버전의 태그들만 검색하여 보여준다.
git branch	브랜치 목록을 보여준다.
git branch [브랜치]	브랜치를 생성한다.
git checkout [브랜치]	해당 브랜치로 이동한다.
git checkout -b [브랜치]	브랜치가 없으면 브랜치를 생성하고 이동한다.
git merge [브랜치]	현재 브랜치에서 입력한 브랜치와 합친다.
git reset [커밋] [파일]	stage area에 있는 파일들을 모두 특정 커밋으로 되돌린다.
git reset –soft 커밋	수정사항을 유지하고 특정 커밋으로 되돌린다.
git reset –hard 커밋	수정사항을 무시하고 특정 커밋으로 되돌린다.

Table 2: 위표는 Git 주요 명령어입니다.