Journal of the Korea Academia-Industrial cooperation Society
Vol. 17, No. 2 pp. 8-14, 2016

http://dx.doi.org/10.5762/KAIS.2016.17.2.8
ISSN 1975-4701 / eISSN 2288-4688

오픈 소스 프로젝트를 위한 도커 기반 버전 관리 기법

이용전[1], 임성락[1*]
[1]호서대학교 컴퓨터공학부

A scheme of Docker-based Version Control for Open Source Project

Yong-Jeon Lee[1], Seong-Rak Rim[1*]
[1]**Division of Computer Engineering, Hoseo University**

요 약 오픈 소스 프로젝트가 다수의 개발자들에 의해 진행될 때 동일한 파일에 대한 여러 버전을 관리해 주는 버전 관리 시스템은 매우 유용한 도구로 사용되고 있다. 그러나 대부분의 기존 버전 관리 시스템들(SVN, Git 등)은 소스 코드 혹은 문서의 변경 이력을 주로 관리하고 있기 때문에 개발 환경의 변경이 발생할 때마다 개발자마다 직접 변경해야 하는 불편함이 있다. 이러한 불편함을 해소하기 위하여 본 논문에서는 오픈 소스 프로젝트를 위한 버전 관리 기법을 제시한다. 제시한 기법의 기본 개념은 컨테이너 방식의 가상화 도구인 도커를 이용하여 개발 환경을 포함한 이미지를 생성하고 이를 새로운 버전으로 관리한다. 제시한 기법의 기능적 타당성을 검토하기 위하여 서로 다른 OS(우분투12.04, 센트OS7)를 사용하는 호스트에 도커를 구축한 후 개발 환경의 변경 이력이 포함된 버전 관리를 실험하고 기존 버전 관리 시스템들과 비교 평가하였다. 그 결과 제시한 기법은 오픈 소스 프로젝트를 위한 편리한 버전 관리 기법이 될 수 있을 것으로 보인다.

Abstract When Open Source Projects are processed by multiple developers, the Version Control Systems, which control the different versions of the same file being used, is a very useful tool. On the other hand, because most of conventional VCS(SVN, Git, etc.) mainly control the history of the modifications of the source codes or documents, there is an inconvenience that each developer should modify the development environment whenever the development environment is modified. To overcome this inconvenience, this paper suggests a scheme of VC for OSP. The basic concept of the suggested scheme is that an image, including the development environment and controls, is created as a new version using the Docker, virtualization tool of the container method. To review the functional appropriateness of the suggested scheme, after establishing the Docker on the hosts that use the different OS(Ubuntu12.0.4, CentOS7), this study tested a VC that could control the different versions including the history of modifications of the development environment and evaluated them by a comparison with the conventional VCS. The results show that the suggested scheme is a convenient scheme of VC for the OSP.

Keywords : Docker Engine & Hub, Open Source Project, Version Control System

1. 서론

복잡한 소프트웨어를 다수의 개발자들이 공동으로 개발하는 과정에서 상당한 수의 소스 코드나 문서와 같은 파일들이 생성된다[1]. 이 과정에서 파일이나 데이터의 손실, 개발자들 간의 지식전달 등의 문제가 생긴다. 버전 관리 시스템(Version Control System)은 이러한 문제를 해결하는 것에 도움을 주는 유용한 도구이다. 버전 관리 시스템은 동일한 파일에 대한 변경 이력을 저장하고 관리해주는 시스템으로 기본적으로 파일들을 마스터 복사

*Corresponding Author : Seong-Rak Rim(Hoseo Univ.)
Tel: +82-41-540-5708 email: srrim@hoseo.edu
Received November 4, 2015 Revised (1st November 20, 2015, 2nd December 15, 2015)
Accepted February 4, 2016 Published February 29, 2016

본(master copies)과 작업용 복사본(working copies)으로 나눈다[2]. 개발자들은 저장소(repository)에 저장되어 있는 마스터 복사본을 자신의 로컬로 다운로드 한 작업용 복사본에서 필요한 변경 작업을 수행하고 다시 저장소에 업로드 한다. 개발자들이 이러한 과정을 반복하면서 동일한 파일에 대한 여러 버전의 파일들이 만들어진다. 이 과정에서 다수의 개발자가 동일한 버전의 파일 내용을 변경할 경우 서로 다른 개발자에 의해 다른 내용이 덮어씌워지게 되는 충돌 문제가 발생하게 된다. 버전 관리 시스템은 이러한 충돌 문제를 예방하기 위한 기능들을 제공한다. 또한 현재 작업용 복사본에 예상치 못한 오류가 발생할 경우 저장소에 저장되어 있는 마스터 복사본을 통해 이전 버전으로 복귀할 수 있도록 도움을 준다.

기존 버전 관리 시스템들은 주로 문서나 소스코드 파일들의 버전을 관리한다. 그러나 오픈 소스 프로젝트(Open Source Project)[3]를 진행하다 보면 개발 환경에 대한 변경이 필요한 경우가 발생한다. 개발 환경은 소프트웨어 개발을 위한 필수 조건으로 운영체제나 커널 버전, IDE(Integrated Development Environment), 각종 도구, 시스템 및 환경 변수 등 필요한 모든 것을 포함하는 매우 중요한 요소이다. 이러한 개발 환경이 변경되면 모든 개발자들은 동일한 개발 환경을 개별적으로 변경해야 한다. 이로 인해 예상치 못한 다양한 형태의 문제들이 발생한다. 또한 다른 사람이 개발 중인 프로그램을 이어서 개발할 경우에도 기존 개발자와 동일한 개발 환경을 구축하는 과정에서 많은 시간을 소비하게 된다. 이를 위해 개발환경에서 필요한 특정 부분만을 변경하여 개별적으로 관리하는 것은 번거로움과 복잡성이 따르고 개발 환경을 포함한 시스템 전체를 버전으로 관리하는 것은 자원의 낭비가 심한 문제점이 있다. 또한 기존 버전 관리 시스템들은 변경 이력 관리를 위한 중앙 서버를 별도로 구축해야하는 번거로움이 있다.

이러한 문제점을 해결하기 위하여 본 논문에서는 도커[4]를 기반으로 한 버전 관리 기법을 제안한다.

2. 관련 연구

2.1 버전 관리 시스템(Version Control System)

버전 관리 시스템은 구성 방식에 따라 로컬 버전관리 시스템, 중앙 집중식 버전관리 시스템(Centralized Version Control System; CVCS), 분산 버전 관리 시스템(Distributed Version Control System; DVCS)으로 나누어진다.

로컬 버전 관리 시스템은 자신의 로컬 컴퓨터에 생성한 데이터베이스에 파일의 변경 사항을 기록하는 방식이다. 시스템 외부에 있는 개발자들과 공동 작업을 할 경우 많은 제한이 따른다. 대표적으로 RCS(Revision Control System)[5]가 이에 속한다.

중앙 집중식 버전 관리 시스템은 버전 관리되는 모든 파일을 저장하는 하나의 서버와 다수의 클라이언트로 구성된다. 중앙 집중식 버전 관리는 다른 개발자 무엇을 하고 있는지 알 수 있고, 관리자는 누가 무엇을 할 수 있는지 로컬 버전 관리 방식보다 쉽고 정확하게 관리할 수 있다. 그러나 중앙 집중식 버전 관리 시스템은 중앙 서버가 다운될 경우 전체 시스템에 영향을 준다. 중앙 데이터베이스의 데이터가 손실된다면 사람들은 각자 자신의 컴퓨터에 가지고 있던 데이터를 제외하고 그동안 쌓인 프로젝트의 이력을 모두 잃게 된다. 중앙 집중식 버전 관리 프로그램으로는 Subversion(SVN)[6], CVS(Concurrent Versions System) 등이 이에 속한다.

분산 버전 관리 시스템은 중앙 집중식 버전 관리 시스템의 문제를 해결하기 위해 개발 되었다. 기본적인 구성은 중앙 집중식 버전 관리 시스템과 같지만 분산 버전 관리 시스템은 클라이언트가 저장소(repository)를 통째로 복제한다. 따라서 서버에 문제가 생겨도 어느 클라이언트든 복제된 저장소를 다시 서버로 복사하면 서버가 복구된다. Git[7], Mecurial[8] 등이 이에 속한다.

2.2 Subversion(SVN)

SVN은 중앙 집중식 버전 관리 시스템으로 오픈소스로 제공되어 현재 많은 회사나 단체에서 사용하고 있다. 또한 소스 코드를 자유롭게 사용 가능하여 사용자의 버전에 맞추어 개발을 할 수도 있다[6]. SVN에서 제공하는 기능으로는 Import, Check Out, Commit, Update 등이 있다.

Import : 서버의 저장소에 파일을 포함시키는 작업이다. 기존에 있는 저장소에 포함 시키거나 새로운 저장소를 생성하여 포함시킬 수 있다.

Check out : 저장소로부터 선택한 디렉터리를 자신의 작업 공간으로 복사해오는 작업이다.

Commit : 자신의 작업 공간에서 변경한 내용을 서버에 알려주는 작업이다. Commit을 해야만 다른 컴퓨터나 서버에서 변경한 내용을 알 수 있다.
Update : 서버로부터 자신의 작업 공간에 변경된 사항을 가져오는 작업이다. 변경된 파일은 현재 자신이 가지고 있는 파일과 병합된다.

SVN의 기본적인 동작 과정은 'Import'를 통해 로컬에 있는 디렉터리나 파일들을 서버의 저장소에 포함시킨다. 다음으로 'Check out'을 통해 서버의 저장소에 있는 디렉터리나 파일들을 자신의 작업 공간으로 복사해온다. 복사해온 파일들로 자신이 원하는 작업을 한 후에 'Commit'을 통해 서버에 변경 사항을 알려준다. 그럼 다른 사용자는 'Update'를 통해 해당 저장소 내의 디렉터리와 파일들의 변경 사항에 대한 정보를 받아온다. 즉 'Commit'과 'Update'를 통해 저장소와 로컬 디렉터리의 파일을 동기화하여 충돌 문제를 예방한다.

2.3 Git

Git는 분산 버전 관리 시스템으로 오픈 소스로 공개되어 있으며 프로젝트 규모에 상관없이 사용될 수 있도록 설계되었다[7]. Git는 분산된 환경에 각각 완전한 저장소를 가지고 저장소 내의 파일 버전, 변경된 모든 파일에 대한 스냅숏을 저장한다[9]. Git는 저장소와 작업용 디렉터리 사이에 working tree(Staging area)가 존재한다. 개발자는 woring tree를 통해 저장소에 파일을 저장하기 전에 파일들을 검토한다. Git의 기능으로는 Init, Clone, Add, Rm, Commit, Check out 등이 있다.

Init : 새로운 Git 저장소(Repository)를 생성하거나 존재하는 저장소를 재초기화 한다.
Clone : 새롭게 생성된 디렉터리(Working Directory)에 안에 저장소를 복제하고 현재 복제된 저장소의 분기점을 초기화한다.
Add : 파일 내용들을 인덱스에 추가한다. 이 명령어는 다음 commit을 위하여 working tree에서 사용하는 인덱스를 갱신한다.
Rm : 인덱스와 working tree로부터 내용들을 제거한다. Rm 명령어로 Working Directory의 파일이 지워지지는 않는다.
Commit : 저장소에 변경 사항을 기록한다. 사용자가 변경 사항에 대한 내용을 기록한 로그와 덧붙여 인덱스의 현재 내용물들을 저장소에 저장한다.
Check out : 분기점을 working tree로 가져온다. 인덱스의 버전을 동기화하기 위해 working tree의 파일들을 갱신한다.

Git의 기본적인 동작 과정은 'Init'으로 새로운 저장소를 생성한다. 그리고 'Clone'을 통해 자신의 로컬로 저장소를 복제한다. 'Check out'을 통해 저장소 변경 내용과 working tree 내용을 동기화한다. 작업용 디렉터리에서 원하는 작업을 한 후에 'Add' 명령어를 통해 변경한 내용들을 Staging area(working tree)에 추가한다. 만약 변경한 내용을 저장소에 반영하는 것을 원하지 않는다면 'Rm'을 통해 working tree에서 제거한다. 그리고 'Commit' 명령어를 통해 working tree 내용을 저장소에 반영한다. working tree를 통해 저장소와 작업용 디렉터리의 변경 이력을 동기화하면서 충돌 문제를 예방한다.

3. 도커 기반 버전 관리

도커는 lxc(Linux Containers)기반의 오픈 소스 가상화 도구이다. 개발자들의 소프트웨어 빌드를 더 빠르게 할 수 있게 도와주고 분산된 응용프로그램들이 어디서나 간편하게 실행되고 공유될 수 있게 해준다[4].

도커에서의 모든 작업은 컨테이너를 통해 이루어진다. 컨테이너는 특정 소프트웨어가 실행되기 위한 코드, runtime, system tools, system libraries를 포함하고 있다. 컨테이너들은 OS의 커널을 공유하며 유저영역에서 독립적으로 동작한다.

컨테이너를 생성하기 위해서는 이미지가 필요하다. 이미지는 컨테이너 생성 및 실행을 위한 바이너리와 라이브러리, 패키지와 같은 필수적인 요소들과 소스 코드나 문서와 같은 파일들까지 포함하고 있다.

도커는 2014년 초, 가상 이미지 누적 다운로드 수가 약 54만 건에서 연말에 1억 25만 건으로 늘어났으며 개발자 허브에 도커 관련 프로젝트가 6만 5천 건이 생성되는 등 기하급수적인 성장세를 보이고 있다[10].

도커에서 지원하는 Table 1과 같은 기능들을 이용하여 버전 관리 기법을 제안한다.

Table 1. Docker's facilities

facilities	explanation
run	create a container
commit	create a new image
tag	attach a tag
pull	download an image from repository
push	upload an image to repository

3.1 시스템 구성

도커 기반 버전 관리는 중앙 집중식 버전 관리의 구조로 Fig. 1과 같이 서버 역할을 담당하는 도커 허브와 클라이언트 역할을 담당하는 도커 엔진으로 구성한다.

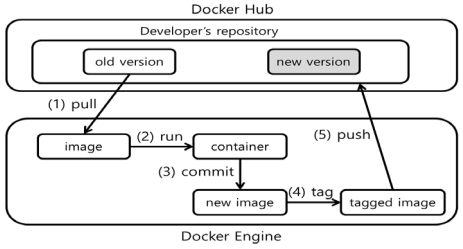


Fig. 1. Docker-based Version Control

개발자들은 도커 엔진을 통하여 도커 허브의 저장소로부터 이전 버전의 이미지를 자신의 로컬로 다운로드한 후 이를 기반으로 컨테이너를 생성한다(1)(2). 컨테이너에서 필요한 변경작업을 수행하고 새로운 이미지를 생성한다(3). 새로운 이미지에 태그를 지정하여 도커 허브의 저장소에 업로드 한다(4)(5). (1)-(5)의 과정을 반복하면서 새로운 버전들이 생성된다.

현재 시점에서 의도하지 않은 파일 삭제나 시스템 설정 오류 등의 문제가 발생하면 기존 이미지를 통해 문제를 해결한다. 컨테이너 내부에서 행하는 작업은 컨테이너 생성 시 기반이 되었던 이미지에 아무런 영향을 주지 않는다. 따라서 문제가 발생했을 때 도커 허브의 저장소로부터 이전 버전의 이미지를 다시 다운로드할 필요 없이 기존 이미지를 기반으로 컨테이너를 다시 생성하면 이전 버전으로 복귀할 수 있다.

버전 충돌 문제는 Fig. 2와 같이 이미지에 태그를 첨부하는 방법으로 예방한다. 도커 엔진에서 이미지에 태그를 첨부하여 새로운 이미지를 생성한다(1). 개발자는 태그를 통해 생성된 새로운 이미지를 저장소에 저장하는 방식으로 원본 이미지를 보호하고 동일한 내용을 저장소에 저장한다(2). 또한 태그가 첨부되지 않은 이미지를 저장소에 업로드 할 경우에는 도커 허브에서 'latest' 태그를 자동으로 첨부해주기 때문에 버전 충돌 문제를 예방할 수 있다.

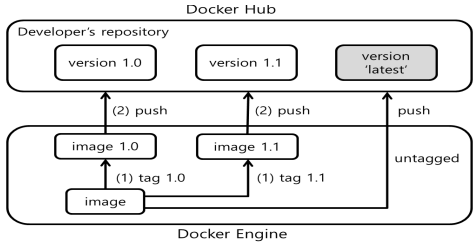


Fig. 2. Collision prevention with image tag

3.2 개발환경 관리

개발환경의 변경 이력 관리는 이미지 관리를 통해 이루어진다. 이미지에는 컨테이너 생성을 위한 필수적인 요소들이 모두 포함되어 있다. 특정 이미지를 기반으로 생성한 컨테이너는 이미지가 포함하고 있는 개발 환경 및 파일 등 여러 요소를 가지게 된다. 따라서 변경되거나 새롭게 구축된 개발환경의 내용을 포함하는 이미지를 저장소에 업로드하면 개발환경의 변경 이력에 대한 새로운 버전이 생성되고 도커 허브는 이러한 버전들을 관리한다.

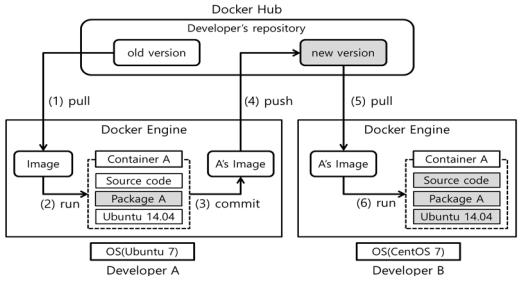


Fig. 3. Development environment control

예를 들어 Fig. 3과 같이 서로 다른 OS를 사용하는 두 개발자 A, B가 Ubuntu 14:04에서 동작하는 프로그램을 개발하려고 할 경우 동일한 개발 환경이 요구된다. 개발자 A는 도커 허브의 저장소로부터 old version의 이미지를 다운로드 한 후(1) 컨테이너로 생성한다(2). 생성한 컨테이너는 Ubuntu 14.04 환경으로 동작하며 소스 코드만을 포함하고 있다. 개발자 A는 소스 코드를 컴파일 하기 위해 패키지 A를 설치한다. 그리고 변경 내용을 포함하는 A's Image를 생성한다(3). 생성된 이미지를 도커

허브의 저장소에 업로드 하면(4) 저장소에는 새로운 버전이 생성되고 개발자 B는 이것을 다운로드 한다(5). 그리고 다운받은 이미지를 기반으로 자신의 도커 엔진에서 컨테이너를 생성한다(6). 개발자 B의 운영체제는 CentOS 7에서 동작하지만 컨테이너는 Unbuntu 14.04로 동작하며 패키지 A와 소스코드를 포함하고 있다. 따라서 개발 중인 프로그램에서 요구하는 패키지나 시스템 설정 등 개발환경이 변경되는 경우에도 변경된 내용을 포함하는 이미지를 저장소에 업로드 하고, 다른 개발자들은 업로드 된 새로운 버전을 다운로드 하여 컨테이너를 생성함으로써 프로그램 실행에 필요한 모든 변경 사항을 제공받을 수 있다.

4. 실험 및 평가

4.1 실험

제시한 시스템의 기능적 타당성을 검토하기 위한 실험 환경은 Table 2와 같다.

Table 2. experimental environment

	A	B
OS	Ubuntu	CentOS
OS version	12.04 (LTS 64bit)	7 (64 bit)
Kernel	3.10	3.10

4.1.1 이전 버전으로 복귀

A 실험 환경에서 Fig. 4와 같이 'run' 명령어를 통해 이전 버전의 이미지(oldversion)를 기반으로 컨테이너를 생성한다(1). 생성과 동시에 컨테이너 내부에 접속되고 컨테이너 내부에서는 커맨드 라인이 컨테이너의 ID로 변경된다(2).

인위적인 문제를 발생시키기 위해 리눅스의 기본 명령어들이 저장되어 있는 '/bin' 디렉터리를 삭제한다(3). 삭제 후에 디렉터리 내부의 파일 목록을 보는 'ls' 명령어가 동작하지 않는 문제가 발생한 것을 확인할 수 있다(4).

```
root@lyj:~# docker run -i -t oldversion /bin/bash…(1)
root@947bc4c3cc3e:/# ls / …(2)
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@947bc4c3cc3e:/# rm -rf /bin/ …(3)
root@947bc4c3cc3e:/# ls …(4)
bash: /bin/ls: No such file or directory
```

Fig. 4. remove /bin to generate a problem

Fig. 5와 같이 이전 버전으로 복귀하기 위해 컨테이너를 종료한 후(1) 이전 버전의 이미지로 컨테이너를 재생성한다(2). 컨테이너 내부에서 'ls' 명령어가 동작하고 '/bin' 디렉터리가 존재하는 것을 확인할 수 있다(3).

```
root@6232933f0e33:/# exit …(1)
exit
root@lyj:~# docker run -i -t oldversion /bin/bash …(2)
root@88fc0772ce72:/# ls /
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr   …(3)
```

Fig. 5. recovery from /bin

4.1.2 버전 충돌 예방

Fig. 6과 같이 'tag' 명령어를 통해 태그를 첨부할 수 있다(1). 태그명은 개발자명과 저장소의 이름, 버전 번호를 명시한다. leeyongjeon/repo:1.0은 leeyongjeon 개발자의 repo 저장소에 1.0 버전을 의미한다. 그 다음 'push' 명령어를 통해 해당 태그명을 첨부하면(2) 도커 허브의 저장소에 업로드가 되는 것을 확인할 수 있다(3).

```
root@lyj:~# docker tag -f newimage leeyongjeon/repo:1.0 …(1)
root@lyj:~# docker push leeyongjeon/repo:1.0 …(2)
The push refers to a repository [leeyongjeon/repo] (len: 1)
Sending image list
Pushing repository leeyongjeon/repo (1 tags)
0105f98ced6d: Image already pushed, skipping
66395c31eb82: Image already pushed, skipping
002fa881df8a: Image already pushed, skipping
a005e6b7dd01: Image already pushed, skipping
0b1e8fc1bbf0: Image successfully pushed …(3)
```

Fig. 6. tag and push

Fig. 7과 같이 태그명 1.0이라는 이미지가 'repo' 저장소에 등록된 것을 확인할 수 있다(1).

Fig. 7. image stored in repository

Fig. 8과 같이 동일한 이미지를 별도의 태그를 하지 않고 업로드 한다(1).

```
root@lyj:~# docker push leeyongjeon/repo  …(1)
The push refers to a repository [leeyongjeon/repo]
Sending image list
Pushing repository leeyongjeon/repo (2 tags)
```

Fig. 8. upload an untagged image to repository

태그가 첨부되지 않았지만 Fig. 9와 같이 ‘latest'라는 태그가 자동적으로 첨부되고 저장소에 저장된 것을 확인할 수 있다(1).

Fig. 9. automatic tagging

4.1.3 개발환경 관리

실험 A에서 Fig.10과 같이 도커 엔진을 통해 Ubuntu 14.04 이미지를 기반으로 컨테이너를 생성한다(1). '/source'를 생성하고 ‘hello'를 출력하는 ’main.c‘ C 소스 파일을 생성한다(2). 그리고 gcc 명령어를 통해 컴파일을 한다(3). 하지만 gcc 컴파일러가 설치되어 있지 않기 때문에 컴파일이 되지 않는 것을 확인할 수 있다(4).

```
root@lyj:~# docker run -i -t ubuntu:14.04 /bin/bash …(1)
root@a79f97bfb742:/# mkdir /source & vi /source/main.c …(2)
[1] 16
[1]+  Done                    mkdir /source
root@a79f97bfb742:/# gcc -o /source/hello /source/main.c …(3)
bash: gcc: command not found  …(4)
```

Fig. 10. create a container

Fig. 11과 같이 gcc 컴파일러를 설치한 후에(1) 컴파일이 정상적으로 완료되고(2) 생성된 프로그램을 실행한 결과(3) 'hello'라는 문구가 출력되는 것을 확인할 수 있다(4).

```
root@a79f97bfb742:/# apt-get install gcc -y …(1)
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 22 newly installed, 0 to remove and 7 not upgraded
root@a79f97bfb742:/# gcc -o /source/hello /source/main.c  …(2)
root@a79f97bfb742:/# ./source/hello …(3)
hello  …(4)
```

Fig. 11. compiling source code and execution

Fig. 12와 같이 'commit' 명령어를 통해 gcc 컴파일러가 설치된 컨테이너를 'gccimage'라는 이름의 이미지로 생성한다(1). 생성이 완료되면 이미지의 ID 값이 화면에 출력되는 것을 확인할 수 있다(2). 그리고 생성된 이미지를 도커 허브의 저장소에 업로드 한다(3).

```
root@lyj:~# docker commit a79f97bfb742 gccimage …(1)
bf168302a8ebe940c4cb4e42a02678224b943cbb6aa32e9399461dbbb8f4682d …(2)
root@lyj:~# docker push leeyongjeon/repo:gccimage …(3)
The push refers to a repository [leeyongjeon/repo] (len: 1)
Sending image list
Pushing repository leeyongjeon/repo (1 tags)
```

Fig. 12. create and push gccimage

B 실험 환경에서 Fig. 13과 같이 도커 허브의 저장소로부터 gccimage를 다운로드 하고(1) 이것을 기반으로 컨테이너로 생성한다(2). B 실험 환경은 CentOS 환경으로 동작하지만 컨테이너 내부는 의 비교 평가는 다음 Table 3과 같다.

ubuntu14:04 환경으로 동작하는 것을 확인할 수 있다(3).

```
[root@localhost yjlee]# docker pull leeyongjeon/repo:gccimage …(1)
Trying to pull repository docker.io/leeyongjeon/repo ...
Status: Downloaded newer image for docker.io/leeyongjeon/repo:gccimage
[root@localhost yjlee]# docker run -i -t gccimage /bin/bash …(2)
root@d4743ad6597c:/# cat /etc/issue
Ubuntu 14.04.3 LTS \n \l …(3)
```

Fig. 13. pull gccimage and create a container

Fig. 14와 같이 A 실험 환경에서 설치한 gcc 컴파일러가 존재하고(1) '/source' 디렉터리와 ‘main.c' 소스 코드도 존재하는 것을 확인할 수 있다(2). gcc 컴파일러로 소스 코드를 컴파일한 후(3) 생성된 파일을 실행한 결과(4) 'hello'라는 문구가 출력되는 것을 확인할 수 있다(5).

```
root@d4743ad6597c:/# gcc --version  …(1)
gcc (Ubuntu 4.8.4-2ubuntu1~14.04) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
root@d4743ad6597c:/# ls /source/…(2)
main.c
root@d4743ad6597c:/# gcc -o /source/hello /source/main.c …(3)
root@d4743ad6597c:/# ./source/hello …(4)
hello  …(5)
```

Fig. 14. compiling source code and execution

4.2 시스템 평가

SVN과 Git, 그리고 도커 기반 버전 관리 기법도커를 이용할 경우 도커 허브가 중앙 서버 역할을 하기 때문에 별도의 서버를 구축하지 않아도 된다. 또한 도커 기반 버전 관리 시스템은 버전 관리를 이미지 단위로 관리를 하고 SVN과 Git는 파일 단위로 관리한다. SVN과 Git는 충돌 문제 해결을 위한 명령어들을 수동으로 사용하여 예방하고 도커 기반 버전 관리 시스템에서는 이미지에 수동으로 태그를 첨부하거나 자동으로 첨부되는 태그로 충돌 문제를 예방한다. SVN이나 Git는 개발환경처럼 비교적 크고 복잡한 데이터를 버전으로 관리하는 것에 어려움이 있지만 도커 기반 시스템에서는 이미지에 모두 포함되어 있기 때문에 관리가 용이하다.

Table 3. comparison of Version Control Systems

	SVN	Git	Docker
system configuration	centralized	distributed	centralized
server	necessary	necessary	unnecessary
unit of version control	file	file	image
collision prevention	commit & update	add & commit	tag
	manual	manual	semi automatic
unit of access permission	directory	directory	repository
development environment control	manual	manual	automatic

5. 결론

소프트웨어 개발 프로젝트의 규모가 커질수록 소스 코드나 문서의 버전 관리는 더욱이 중요한 요소이다. 현재 가장 널리 사용되는 버전 관리 시스템들(SVN, Git)은 문서나 소스코드 같은 파일 단위의 버전을 관리하는 것에는 효과적이지만 오픈 소스 프로젝트의 진행 과정에서 종종 발생되는 개발환경의 변경 이력은 개발자 마다 관리해야 하는 불편함이 있다. 본 연구에서 제안한 도커 기반 버전 관리 기법은 개발 중인 프로그램의 개발 환경까지도 버전으로 관리해준다. 따라서 다수의 개발자들이 오픈 소스 프로젝트를 진행할 경우 개발 환경의 변경에 쉽게 대처할 수 있다. 또한 현재 급격한 상승세를 보이고 있는 도커를 이용한 프로젝트나 개발 환경의 관리에 어려움을 겪고 있는 솔루션 업체, 클라우드 서비스 등에 많은 이점을 가져올 것이라 기대된다.

References

[1] Nam-ho. Kim, Young B. Park, “An Implementation of Automation Version Management System”, KIISE, p.55-60, Oct, 2007.

[2] Bieniusa. A, Thiemann. P, Wehr. S, “The Relation of Version Control to Concurrent Programming”, Computer Science and Software Engineering, P.461 - 464. Dec, 2008

[3] Orucevic-Alagic. A, Höst. M. "Network Analysis of a Large Scale Open Source Project", SEAA, 40th EUROMICRO, p.25 - 29, Aug, 2014.

[4] Docker, "https://www.docker.com/"

[5] Sagi Ifrash, David H. Lorenz, "Crosscutting Revision Control System", ICSE, 2012 34th International Conference on, P.321-330, Jun, 2012.

[6] TortoiseSVN, "http://tortoisesvn.net/"

[7] git, "http://git-scm.com/"

[8] mercurial, "https://mercurial.selenic.com/"

[9] Russell G. Shirey, Kenneth M. Hopkinson, Kyle E. Stewart, Douglas D. Hodson, Brett J. Borghetti, "Analysis of Implementations to Secure Git for Use as an Encrypted Distributed Version Control System", HICSS, Jan, 2015.

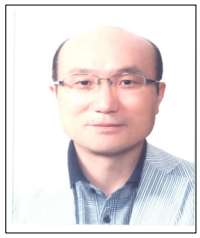
[10] Won-Yong Kim, Myung-Woo Kang, Hye-Jung Park, Man-Soo Kim, Eui-Nam Huh, Chan-Ho Yong, "A Study on Operating-System Level Virtualization based on Linux Container", KIISE, p.1226-1228, Jun, 2015.

이 용 전(Yong-Jeon Lee) **[준회원]**

- 2008년 2월 : 호서대학교 컴퓨터공학과 (학사)
- 2014년 2월 ~ 현재 : 호서대학교 컴퓨터공학과 (석사)

<관심분야>
임베디드 시스템

임 성 락(Seong-Rak Rim) **[정회원]**



- 1983년 2월 : 서울대학교 공과대학원 컴퓨터공학과(공학석사)
- 1992년 8월 : 서울대학교 공과대학원 컴퓨터공학과(공학박사)
- 1983년 3월 ~ 1990년 2월 : (주) 금성반도체연구소 선임연구원
- 1993년 3월 ~ 현재 : 호서대학교 컴퓨터공학과 교수

<관심분야>
임베디드 시스템, 리눅스 커널