

- 초보자를 위한 -

레일스 가이드북



ROR LAB.

Table of Contents

1. Introduction
2. 저자소개
3. 루비온레일스 소개
4. 환경 준비
 - i. 개발환경
 - ii. Git 설치
 - iii. rbenv 설치
 - iv. rvm 설치
 - v. 소스코드 에디터
 - vi. 서버 운영환경 구축
5. 레일스 설치
 - i. 언스탈러를 이용한 설치
 - ii. 맥OSX 개발마신에 설치하기
 - iii. 리눅스 개발마신에 설치하기
 - iv. 윈도우 개발마신에 설치하기
6. 프로젝트 따라하기
 - i. Gemfile의 작성
 - ii. welcome 컨트롤러의 생성
 - iii. 첫번째 모델의 생성
 - iv. 어플리케이션 레이아웃의 작성
 - v. posts CRUD 살펴보기
 - vi. Bulletin 모델의 생성
 - vii. 모델 관계선언
 - viii. posts 컨트롤러 변경
 - ix. posts 뷰 변경
 - x. 게시판 레이아웃 작성하기
 - xi. 갤러리형 레이아웃 작성하기
 - xii. flash 메시지 표시하기
 - xiii. 기본 데이터 추가하기
 - xiv. 코멘트 달기
 - xv. 태그 달기
 - xvi. 서버로 배포하기
7. 프로젝트 배포하기
 - i. Capistrano 3를 이용한 배포
 - ii. SSHKit 사용법
 - iii. SSHKit의 사용예제
 - iv. Capistrano 2.x에서 업그레이드하기
8. 부록
 - i. 공동집필에 참여하기
 - ii. 엑티브레코드란?
 - iii. Image Magick 설치하기
 - iv. PostgreSQL 설치하기
 - v. 우분투 12.04 서버 세팅하기
 - vi. 우분투 14.04 서버 세팅하기 (Virtual Box)
 - vii. Heroku에 배포하기

- viii. AWS S3 사용하기
- ix. 개발 환경 변수를 관리하는 dotenv 점
- x. 맥진용 랙서버 POW
- xi. 추천 웹사이트 및 블로그
- xii. 추천도서
- xiii. 추천동영상
- xiv. 변경 내역

초보자를 위한 레일스 가이드북 v1.0.6

2014년 4월

이 책은 이제 막 루비온레일스를 시작하는 개발자를 대상으로 실무 프로젝트를 따라 하면서 루비온레일스의 전반적인 내용을 쉽게 이해할 수 있도록 구성하였다.

루비온레일스(Ruby on Rails, 이하 레일스로 기술함)는 강력한 코드 템플릿 기능과 MVC 디자인패턴 덕분에 프로토타입을 애자일(신속, agile)하게 개발할 수 있어, 처음 접하는 개발자들이 매우 매력적인 개발 툴로 생각하게 된다.

그러나, 실제로 개발을 시작하기 위한 환경을 설정하는데는 약간의 시행착오가 뒤따른다. 이 책은 바로 이러한 사소한 실수 없이 편안하게 레일스의 설치부터 서버의 배포까지 일련의 과정을 다양한 툴과 루비젬을 이용하여 개별자 스스로가 일시천리로 작업하는데 도움을 주기 위해서 준비하였다.

또한, Github에 문서 소스를 공개하여, 실무 경험이 있는 여러 개발자들이 공동으로 집필함으로써 각자의 레일스 프로젝트 개발의 노하우를 함께 공유할 수 있도록 하였고, [ROR Lab.](#)(레일스 가이드 오프라인 무료 강좌)에서 그동안 강의했던 내용 중 중요한 부분을 별弛해서 중간 중간에 설명도 겸하도록 하였다.

부디, 이 책이 레일스를 처음 접하는 개발자들에게 불필요하게 들여야 하는 시간, 돈, 노력을 절약하는데 도움이 되기를 바란다.

2014년 4월

ROR Lab. 운영자

최효성



저자소개

이 책은 ROR Lab.에서 주관한 공동집필 프로젝트에 참여한 개발자들이 함께 저술하였다.

공동집필자 :

- 남승균
- 배경만
- 손영체
- 이한결
- 정창훈
- 최효성
- Shim Tw

기여자

- 김인기
- 신재훈
- 유상민
- 차경묵

(가나다 ABC 순)

공동집필자로 등록 후 활동을 원할 경우 본인의 성함과 github 계정명을 연락(rorlab@gmail.com)을 주기 바랍니다.

편집자 : 최효성

저작권

이 책의 모든 소유권은 ROR Lab.에 속한다.



루비온레일스

레일스 는 루비 로 작성되었으며, 웹애플리케이션 개발을 위한 프레임워크를 말한다.



2003년 미국의 [David Heinemeier Hansson](#)이 처음으로 레일스를 발표한 후 현재 버전 4.1이 릴리스된 상태이며, 최신의 웹기술들을 과감하게 도입하고 있으며 지금도 매우 빠르게 버전업이 지속되고 있다. 따라서, 부지런하지 않으면 레일스의 멋진 기능을 제대로 사용하지 못할 정도이다.

그러나, 이러한 부분이 개발자가 즐겁게 프로그래밍을 할 수 있게 해 주며, 레일스의 강력한 코드 템플릿 기능과 MVC 디자인 패턴으로 프로토타입을 애자일(신속)하게 개발할 수 있게 해 준다. 최근 루비가 2.0으로 업그레이드되면서, 향상된 퍼포먼스와 함께 웹서버의 운영 환경을 구축할 수 있어서 레일스를 이용한 웹애플리케이션의 개발에 많은 개발자들이 관심이 집중되고 있다.

레일스를 이용한 웹애플리케이션 개발 시장의 규모는 미국으로 중심으로 지속적으로 성장하고 있지만, 국내에서는 공공기관을 중심으로 한 웹개발 시장의 특수한 IT 환경 때문에 레일스의 확산이 미미한 상태이다. 이로 인해 기존 개발업체에서도 레일스를 이용한 웹애플리케이션의 개발을 과감하게 도입하지 못하는 실정이다.

그러나, 레일스를 이용하여 웹애플리케이션을 개발할 때의 여러가지 장점을 고려해 볼 때, 많은 스타트업들이 레일스로 웹애플리케이션을 개발하여 하나 둘씩 그 성공신화가 이어져 간다면, 마지 않아 국내에서도 레일스를 이용한 웹개발 시장이 자연스럽게 형성되지 않을까 생각한다. 이것은 마치 선순환의 고리를 연상케 한다.

이를 위해서 국내에서 초보자들이 쉽게 레일스 프레임워크를 이용하여 웹애플리케이션을 개발할 수 있도록 하기 위해서는 한글로 작성된 레일스 관련 리소스들이 많이 제작되어야 함은 물론이고, 영문으로 작성된 문서들을 한글화 하는 작업에도 관심을 가져야 할 것으로 생각한다.

ROR Lab. 운영자

최효성

환경 준비

레일스로 웹애플리 케이션을 개발하기 위해서는 환경 설정이 필요하다.

이 장에서는 프로젝트를 개발하기 위한 로컬환경 설정부터 레일스 프로젝트를 웹서버로 배포하기 위한 운영 서버 설정까지의 대략적인 내용을 설명한다.

개발환경



레일스는 루비 언어로 만들어졌기 때문에 모든 환경에서 루비 인터프리터가 설치되어 있어야 한다. 루비 언어를 만든 일본의 마츠의 이름이 기리기 위해서 원조 루비 인터프리터를 MRI(Matz' Ruby Interpreter)라고 하는데, C 언어로 작성되었다. 참고로 MRI외의 대표적인 루비 구현체로 JRuby와 Rubinius 등이 있는데, JRuby는 자바로 구현된 루비 구현체이고 Rubinius는 루비로 구현된 루비구현체이다.

- 소스관리자(GIT) 설치
- 루비버전관리자의 설치 :
 - rbenv
 - rvm
- 레일스 설치
- 코드 에디터의 선택

Git 소개

여러 가지 소스코드 관리(SCM, Source Code Management) 툴이 있지만 레일스에서는 일반적으로 `git` 을 사용한다.

설치하기

운영체계별 `git` 설치 방법은 [Pro Git](#) 에 잘 설명되어 있다. 맥 사용자들은 Mac OS X에서 [git의 한글화 문제](#) 가 있을 수 있다는 것을 알아둘 필요가 있으며, 더 자세한 내용은 [여기](#)를 참고하기 바란다.

유익한 Git 공부 자료

온오프라인에는 많은 자료가 있지만, 이런 격언이 생각난다.

"부뚜막의 소금도 집어넣어야 파다"

- [Pro Git](#)
- [Pro Git 전자문서](#)
- [Git Cheat Sheet](#)
- [Github Cheat Sheet](#)
- [git - 간편 안내서](#)
- [GIT-IMMERSION](#)
- [Try Git](#)
- [Git 브랜치 배우기](#)

Git GUI 툴

가끔은 커맨드라인에서 `git` 작업하는 것이 번거롭고 전체적인 상황을 그래픽으로 편하게 보고 싶은 경우 있다. 이럴 때는 아래의 두 가지 툴을 사용하면 매우 편리하다. `SourceTree` 는 무료, `Tower` 는 유료.

- [SourceTree](#) : 윈도우 및 맥용 무료 Git GUI. `git-flow` 기본내장. 강추...^^
- [Tower](#) : 맥용 유료(\$59.00. paypal 가능) Git GUI. 최근에 Tower2로 업그레이드되었으며 `git-flow` 기능이 추가되었음.

rbenv 설치

레일스 공식웹사이트에서는 `rbenv`을 이용하여 루비를 설치할 것을 권고하고 있다(레일스를 만든 `MRI`는 `rbenv`을 사용하고 있다).

이전에는 주로 `rvm`을 사용하였지만, `Bundler`을 이용하여 `Gemfile` 파일로 절을 관리하게 됨에 따라, 사용법이 간단한 `rbenv`을 더 많이 사용하고 있다. 어떤 것을 사용할 것인가 망설일 필요가 없다. `rbenv`을 권하는 더 자세한 이유를 알고 싶다면 [Why choose rbenv over RVM?](#)을 참고하기 바란다.

또 한편으로는 각자의 선호도에 따라 어떤 것을 선택하더라도 계대로만 사용하면 큰 문제가 없다는 의견도 만만치 않다([RVM and rbenv](#)).

`rbenv`을 이용하면 애플리케이션 별로 각기 다른 루비 버전을 쉽게 사용할 수 있고 새 버전이 릴리스될 때도 쉽게 업데이트할 수 있다. 아래에 운영시스템별로 설치법을 정리하였다.

맥 OS X

맥에서는 `Homebrew` 패키지 매니저를 이용하면 `rbenv`과 `ruby-build`을 쉽게 설치할 수 있다.

`Homebrew` 는 한줄의 명령어로 설치 가능한데 해당 명령어는 계속 변경되기 때문에 문서에 언급하지 않는다. <http://brewsh> 하단의 `Install Homebrew`를 참고하기 바란다.

`Homebrew`을 이용한 `rbenv`, `ruby-build` 설치 방법

```
$ brew update  
$ brew install rbenv ruby-build
```

최근에는 `Homebrew GUI` 애플리케이션(`Cakebrew`)도 출시되었다.

리눅스

리눅스 환경에서 루비 버전 관리를 도와주는 `rbenv`을 설치해보자. 먼저 우분투를 비롯한 데비안 기반 리눅스에서 `rbenv`을 설치하는 방법을 소개한다.

[github](#)에서 `rbenv`을 받아온다.

아래와 같이 git 명령으로 `github`의 `rbenv` 프로젝트를 [사용자 홈 디렉토리]/`.rbenv`에 클론(복사)한다.

```
$ git clone git://github.com/sstephenson/rbenv.git .rbenv
```

명령 프롬프트에서 `rbenv`을 실행할 수 있게 쉘 환경변수를 수정한다. `.bashrc` 파일을 염여들일 수 있도록 편집기를 열어서 `.bash_profile`에 다음과 같이 추가한다.

```
[ -f "$HOME/.profile" ] && source "$HOME/.profile"  
[ -f "$HOME/.bashrc" ] && source "$HOME/.bashrc"
```

.bashrc 내용은 다음과 같이 작성한다. rbenv 가 저장된 디렉토리를 RBENV_ROOT 환경 변수에, rbenv 실행 파일이 들어 있는 디렉토리를 PATH 에 추가한다.

쉘을 실행할때마다 rbenv init - 명령을 실행한다.

```
export RBENV_ROOT="$HOME/.rbenv"
if [ -d "$RBENV_ROOT" ]; then
    export PATH="$RBENV_ROOT/bin:$PATH"
    eval "$(rbenv init -)"
fi
```

루비를 설치하기 위해서는 rbenv 의 플러그인 ruby-build 가 필요하다. .rbenv/plugins 디렉토리를 생성하고 github에서 ruby-build 를 받아온다.

```
$ mkdir -p ~/.rbenv/plugins
$ cd ~/.rbenv/plugins
$ git clone git://github.com/sstephenson/ruby-build.git
```

install 옵션을 사용해서 루비를 설치해보자. 여기서는 2.0.0-p451 버전을 설치했다.

```
$ rbenv install 2.0.0-p451
```

rehash 옵션은 새로운 환경을 재설정하는 옵션으로 새로 루비를 설치하거나 루비 점을 설치한 다음 반드시 실행해야 한다.

```
$ rbenv rehash
```

global 옵션은 전역 설정을 변경하는 옵션으로 시스템에서 해당 버전의 루비를 사용하기 위해 사용한다.

```
$ rbenv global 2.0.0-p451
```

설치하고자 했던 루비 버전이 계대로 설치되었는지 확인해보자.

```
$ ruby -v
ruby 2.0.0p451 (2014-02-24 revision 45167) [x86_64-linux]
```

루비 설치가 끝났으면 루비 점을 관리하기 위해 bundler 를 설치한다. 새로운 환경을 재설정하기 위해 rehash 를 일지 말자.

```
$ gem install bundler
$ rbenv rehash
```

rbenv-gem-rehash 을 설치하면, 매번 rbenv rehash 명령을 실행하지 않아도 된다. 또한, rbenv-bundler 를 설치하면 bundle exec 를 생략해도 된다.

원도우

원도우에서는 rbenv을 사용할 수 없다. 관련 문서에 의하면, 대신에 RubyInstaller 또는 [Pik](#)를 사용할 것을 권한다.

References:

1. [Mischa Taylor's Coding Blog](#)
2. [Why choose rbenv over RVM?](#)
3. [RVM and rbenv](#)

rvm 설치

: `rbenv` 을 원치 않을 경우 대신해서 `rvm` 을 설치할 수 있다. 사실 `rvm` 이 `rbenv` 보다 먼저 출시되었다.

맥 OS X

- THE OFFICIAL OS X RVM GUI : [JewelryBox](#)
- [커맨드라인 툴 설치](#)

리눅스

- Ubuntu(우분투)의 경우
- 우선 `curl`이 설치되어 있어야 한다. 가장 안정된 `curl`은 우분투 소프트웨어 센터에서 제공하는 패키지이다. 우분투 소프트웨어 센터에서 'curl'을 검색하면 쉽게 찾을 수 있다.
- [Ruby Version Manager](#) 홈페이지에 있는 [설치방법](#) 중 적당한 것을 골라 설치한다.
- `rvm`은 자신의 홈디렉토리의 '`~/.rvm`'에 설치되게 된다. `rvm` 설치를 위해선 root 계정이 필요없지만, `rvm` 구동에 필요한 패키지 설치, 즉 패키지 의존(Package Dependencies) 때문에 루트 계정이 필요할 수 있다. (참고: [root 계정없이 rvm 설치하는 방법](#))

윈도우

[Installing RVM With Cygwin on Windows](#)

References:

1. [JewelryBox](#)
2. [커맨드라인 툴 설치](#)
3. [curl](#)
4. [Ruby Version Manager](#) 홈페이지
5. [root 계정없이 rvm 설치하는 방법](#)
6. [Installing RVM With Cygwin on Windows](#)

소스코드 에디터

- [Textmate](#) : 레일스 코어팀이 사용하는 맥전용 에디터 (유료)
- [Sublime Text Editor 2 / 3](#) : 리눅스/맥/윈도용 텍스트 및 소스코드 에디터로 최근에 더 인기 있는 에디터 (무료/유료)
- [Atom](#) : 리눅스/맥/윈도우용 텍스트 및 소스코드 에디터로 Github에서 최근에 배포하였으며 Sublime Text Editor와 유사한 인터페이스를 가지고 있음 (무료)
- [Vim + vim-rails](#) : 유명한 유닉스 에디터로 맥용/윈도우용도 있음. (무료)

IDE (통합개발환경) 틀

- [RubyMine](#) (유료)
 - [Aptana RadRails](#) (무료)
 - [Netbeans](#) (무료)
-

References:

1. <http://stackoverflow.com/questions/826164>

서버 운영환경 구축

대개의 경우 리눅스 계열의 운영체계를 이용하여 서버를 구축한다.

서버에 설치해야 할 기본 프로그램은 아래와 같다.

1. Git 설치
2. NodeJS 설치
3. ImageMagick 설치
4. Nginx(또는 Apache) 웹서버 설치
5. MySQL 서버 설치

이외에도 웹서버에 설치할 웹애플리케이션의 기능을 지원하기 위한 다양한 툴을 설치할 수 있다.

서버를 구축한 후 웹프로젝트를 서버로 배포하기 위해서는 [프로젝트 배포하기](#)를 참고하면 된다.

CentOS 6.5 x86_64 서버 설치

- 현재 가장 최신 버전은 [6.5](#)
- DVD 한장으로 구울 수 있는 배포판으로 [CentOS-6.5-x86_64-LiveDVD.iso](#) 가 있고,
- 최소설치만 할 수 있는 배포판으로는 [CentOS-6.5-x86_64-minimal.iso](#) 가 있다.

시스템 업데이트

: [Development Tools](#) 를 설치할 때 git 도 함께 설치된다. (2014년 3월 7일 현재 - 1.7.1 버전, 그러나 최신버전은 1.9.0)

```
# yum -y update
```

Sudo 패키지 설치

: 최소판에는 sudo가 설치되어 있지 않으므로 아래와 같이 sudo 패키지를 설치한다.

```
# yum install -y sudo
```

개발툴 설치

```
# yum groupinstall -y "Development Tools"
```

Deployer 유저 등록

: 서버 배포를 위한 유저를 등록한다.

- 등장 deployer라는 유저명을 사용한다.

```
# useradd deployer  
# passwd deployer
```

- /etc/group 파일을 에디터(vi)로 열고 wheel 그룹에 deployer를 추가하고,

```
# vi /etc/group  
wheel:x:10:root,deployer
```

- /etc/sudoers 파일을 열어 wheel 그룹에 대해서 아래와 같이 코멘트 표시(#echo)를 삭제한다.

```
## Allows people in group wheel to run all commands  
%wheel  ALL=(ALL)      ALL  
## Same thing without a password  
%wheel  ALL=(ALL)      NOPASSWD: ALL
```

- sshd를 재시작한다.

```
# service sshd restart
```

Firewall 설정

- 방화벽에서 80포트를 열어둔다.

```
# vi /etc/sysconfig/iptables  
  
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
```

- iptables를 재시작한다.

```
# service iptables restart
```

Nginx 서버

- <http://wiki.nginx.org/Install>
- nginx의 yum 저장소를 추가하기 위해서, /etc/yum.repos.d/nginx.repo 파일을 생성하고 아래의 옵션을 붙여 넣는다.

```
[nginx]  
name=nginx repo  
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/  
gpgcheck=0  
enabled=1
```

: CentOS, RHEL, Scientific Linux가 릴리스 버전(\$releasever) 번수에 따라 원하는 값으로 대체한다. OS 버전에 따라 5.x는 "5", 6.x 는 "6"을 사용하면 된다. \$basesearch에는 "x86_64" 값을 지정한다.

```
# yum install -y nginx
# chkconfig --levels 235 nginx on
# mkdir -p /etc/nginx/sites-enabled
# vi /etc/nginx/nginx.conf
```

- 설치하고 /etc/nginx 디렉토리에 sites-enabled 폴더를 생성해 준다. 그리고 /etc/nginx/nginx.conf 파일 32 번 줄에 include /etc/nginx/sites-enabled/*.conf; 추가한다.
- include /etc/nginx/sites-enabled/*.conf; 을 추가하고, user를 deployer로 변경한다.
- nginx를 시작한다.

```
# /etc/init.d/nginx start
```

MySQL (5.5.36) 서버 ([ref.](#))

- MySQL 서버를 설치하기 전에 mysql 관련된 것을 모두 제거한다.

```
# yum remove mysql mysql-* -y
# cd /etc/yum.repos.d
# wget http://rpms.famillecollet.com/enterprise/remi.repo
# yum --enablerepo=remi install -y mysql mysql-server mysql-devel
# chkconfig --levels 235 mysqld on
# service mysqld start
# mysql_secure_installation
```

- /etc/my.cnf 파일에 아래 줄을 추가해 준다.

```
[mysqld]
character-set-server = utf8
```

- 권한을 승인한다.

```
# mysql -u root -p
mysql> grant usage on *.* to deployer@localhost identified by 'password';
```

NodeJs 설치

: [참고] : <https://www.digitalocean.com/community/articles/how-to-install-and-run-a-node-js-app-on-centos-6-4-64-bit>

```
# cd /usr/src
# wget http://nodejs.org/dist/v0.10.26/node-v0.10.26.tar.gz
# tar zxf node-v0.10.26.tar.gz
```

```
# cd node-v0.10.26  
# ./configure  
# make (6-7분 소요됨) pc 사양이 좋으면 3분안에 끝나기도 함 ...  
# make install
```

ImageMagick 설치 (시간 많이 걸리네~)

```
# yum install -y tcl-devel libpng-devel libjpeg-devel ghostscript-devel bzip2-devel freet  
# wget ftp://ftp.imagemagick.org/pub/ImageMagick/ImageMagick.tar.gz  
  
# tar -xzvf ImageMagick.tar.gz  
# cd ImageMagick-6.8.8-7  
# ./configure --prefix=/usr/local --with-bzlib=yes --with-fontconfig=yes --with-freetype=  
  
# make  
# make install  
# convert --version  
# yum install -y ImageMagick-devel
```

- deployer 계정 홈페이지의 .bashrc 파일에 아래를 추가한다.

```
# su deployer  
# vi ~/.bashrc  
  
export PKG_CONFIG_PATH="/usr/local/lib/pkgconfig"
```

- 그리고, root 계정으로 접속해서

```
# ln -s /usr/local/include/ImageMagick/wand /usr/local/include/wand  
# ln -s /usr/local/include/ImageMagick/magick /usr/local/include/magick  
# ldconfig /usr/local/lib  
# su deployer  
# cd ~
```

배포

: 이후부터는 로컬에서 capistrano로 배포작업을 수행한다.

- 프로젝트의 Gemfile 파일에 아래의 챕터를 추가하고 bundle install 한다.

```
gem 'capistrano-rbenv'
```

- capistrano 와 capistrano-rbenv 챕터를 레일스 프로젝트의 gemfile에 추가하여, bundle install 하면, capistrano로 배포시, 특히, cap deploy:setup 명령을 실행하면, 자동으로 rbenv와 ruby-build 가 설치되고, 디폴트로 1.9.3-p194 버전 루비가 설치된다. config/deploy.rb 파일에 set :rbenv_ruby_version, "1.9.3-p286" 와 같이 추가해 주면, 원하는 루비 버전을 설치할 수도 있다. [ref](#).

부록

: 아래의 설치는 옵션이다.

Postfix 설치

: 서버에서 이 메일을 보내고 싶은 경우 설치한다.

- [Gmail을 이용해서 postfix 설정하기](#)

```
# yum install -y postfix
```

Git 업그레이드하기

: 참고 - <http://tecadmin.net/how-to-upgrade-git-version-1.7.10-on-centos-6>

```
# rpm -i 'http://pkgs.repoforge.org/rpmforge-release/rpmForge-release-0.5.3-1.el6.rf.x86_64.rpm'
# rpm --import http://apt.sw.be/RPM-GPG-KEY.dag.txt
```

아래와 같이 rpmforge.repo 파일을 열어서 [rpmforge-extras] 부분의 enabled=0 to 1로 변경한다.

```
# vim /etc/yum.repos.d/rpmforge.repo
```

Git 별도설치

: git 를 별도로 설치할 때 아래와 같이 한다.

- git 를 소스컴파일로 설치한다. ([ref.](#)) 깔끔하게 설치된다.
- 가장 최신버전 1.9.0 (3월 7일, 2014년 현재) ([버전별 git 다운로드 리스트 보기](#))

```
# yum install gettext-devel expat-devel curl-devel zlib-devel openssl-devel
# cd /usr/src
# wget https://www.kernel.org/pub/software/scm/git/git-1.8.4.tar.gz
# tar xzvf git-1.8.4.tar.gz
# cd git-1.8.4
# make prefix=/opt/git all
# make prefix=/opt/git install
# export PATH="/opt/git/bin:$PATH"
```

구글링 자료 모음

- <http://www.centos.org> [공식홈페이지]
- http://soredirect.centos.org/centos/6/isos/x86_64/ [다운로드]
- <http://faq.hostway.co.kr/story/4510> [CentOS 6.3 10분 안에 설치하기, 한글]
- <http://mcchae.egloos.com/10937303> [지훈현 서의 CentOS 6.3 미니멀(서버) 설치하기, 한글]

레일스 설치

두 가지 방법으로 설치할 수 있다.

레일스 인스톨러를 이용하는 방법과 매뉴얼로 직접 설치하는 방법이다.

- [인스톨러를 이용한 설치](#)
- [매뉴얼 설치](#)
 - 맥 OSX 개발마신에 설치하기
 - 리눅스 개발마신에 설치하기
 - 윈도우즈 개발마신에 설치하기

인스톨러를 이용한 설치법

세 가지 인스톨러를 웹에서 찾을 수 있다.

1. Engine Yard에서 제공하는 무료 레일스 인스톨러

<http://railsinstaller.org/>



2. Bryan Bibat의 RailsFTW

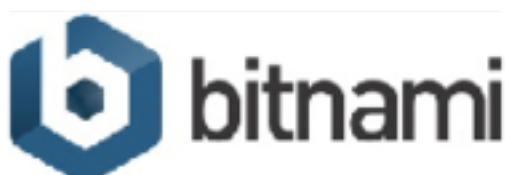
<http://railsftw.bryanbibat.net>: 윈도우 개발 마シン에 설치하기



3. bitnami 루비스택 인스톨러

BitNami Ruby Stack greatly simplify the development and deployment of Ruby on Rails and its runtime dependencies. It includes ready-to-run versions of Ruby, Rails, MySQL, Git, Subversion, RVM and other components.

<https://bitnami.com/stack/ruby/installer>



맥 OSX 개발머신에 설치하기

최신 OSX인 매버릭스(10.9)를 기준으로 설명한다. 맥을 구입하면 기본적으로 ruby(시스템 루비)가 내장되어 있다. 즉, 시스템 루비를 바로 사용할 수 있게 된다. 설치된 루비 버전을 알기 위해 아래와 같은 명령을 실행해 본다. 시스템 루비는 2.0 버전으로 설치되어 있다.(설치되어 있는 시스템 루비 버전은 다를 수 있다)

```
$ ruby -v  
ruby 2.0
```

그러나 몇 가지 이유로해서 시스템 루비는 가능하면 사용하지 않는 것이 좋다.

대신에 루비 버전 관리자를 설치하여 루비를 버전별로 사용할 수 있도록 한다. 이를 위해서 rbenv 을 이용하는 것이 추천된다. rbenv 설치를 참고하여 설치한 후, 루비의 최신 버전인 2.1.2를 설치해 보자. 우선 rbenv 플러그인인 ruby-build 목록에 설치 가능한 루비 버전이 어떤 것이 있는지 알아보자.(rbenv 최신버전을 설치하면 ruby-build 플러그인이 자동으로 설치된다)

```
$ rbenv install --list
```

--list 옵션은 줄여서 -l로 지정할 수도 있다.

만약, 목록 중에 2.1.2 버전이 없으면 ruby-build 플러그인을 업데이트할 필요가 있다. homebrew 를 이용하여 ruby-build 를 설치했다면 아래와 같이 업데이트 할 수 있다.

```
$ brew update ruby-build
```

이제 다시 rbenv install list 명령으로 2.1.2 버전이 등록되었는지를 확인하고 있다면 아래와 같이 루비 2.1.2 버전을 rbenv으로 설치한다.

```
$ rbenv install 2.1.2
```

이제 레일스를 설치해 보자.

```
$ gem install rails
```

그리고 아래와 같이 설치를 마무리 한다.

```
$ rbenv rehash
```

아마도 가장 최신버전의 레일스가 설치될 것이다. 2014년 5월 16일 현재 4.1.1 버전이 설치될 것이다. 아래와 같이 설치된 레일스 버전을 확인할 수 있다.

```
$ rails -v  
Rails 4.1.1
```


리눅스 개발머신에 설치하기

우분투 14.04 LTS (Trusty Tahr)의 경우

gorail.com 사이트 참조

Ruby 설치 (2.1.2)

Ruby 설치를 위해 필요한 라이브러리 등을 우선 설치합니다.

```
$ sudo apt-get update  
$ sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-de
```

설치 방법은 rbenv나 rvm을 이용하거나, 혹은 스스로 직접 설치하는 방법이 있습니다. 공식적으로 rbenv를 이용할 것을 권장하고 있습니다.

rbenv를 이용하는 방법

rbenv를 이용해서 ruby를 설치하는 것은 매우 간단합니다. 우선 rbenv를 설치하고, ruby-build를 하면 됩니다.

```
$ cd  
$ git clone git://github.com/sstephenson/rbenv.git .rbenv  
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc  
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc  
$ exec $SHELL  
  
$ git clone git://github.com/sstephenson/ruby-build.git ~/.rbenv/plugins/ruby-build  
$ echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc  
$ exec $SHELL  
  
$ rbenv install 2.1.2  
$ rbenv global 2.1.2  
$ ruby -v
```

마지막 작업은 각 패키지들의 문서들을 설치하지 말라고 지정하는 것입니다.

```
$ echo "gem: --no-ri --no-rdoc" > ~/.gemrc
```

rvm을 이용하는 방법

```
$ sudo apt-get install libgdbm-dev libncurses5-dev automake libtool bison libffi-dev  
$ curl -L https://get.rvm.io | bash -s stable  
$ source ~/.rvm/scripts/rvm  
$ echo "source ~/.rvm/scripts/rvm" >> ~/.bashrc  
$ rvm install 2.1.2  
$ rvm use 2.1.2 --default  
$ ruby -v
```

마지막 작업은 각 패키지들의 문서들을 설치하지 말라고 지정하는 것입니다.

```
$ echo "gem: --no-ri --no-rdoc" > ~/.gemrc
```

소스로 직접 설치하는 방법

```
$ cd  
$ wget http://ftp.ruby-lang.org/pub/ruby/2.1/ruby-2.1.2.tar.gz  
$ tar -xvzf ruby-2.1.2.tar.gz  
$ cd ruby-2.1.2/  
$ ./configure  
$ make  
$ sudo make install  
$ ruby -v
```

마지막 작업은 역시 각 패키지들의 문서들을 설치하지 말라고 지정하는 것입니다.

```
$ echo "gem: --no-ri --no-rdoc" > ~/.gemrc
```

Git 설정

본 가이드의 [Git 설치하기](#) 참조

Rails 설치

Rails는 상당히 많은 프로그램에 의존하기 때문에 NodeJS와 같은 Javascript runtime을 설치해야 합니다. 이는 Rails에서 Coffeescript와 the Asset Pipeline을 사용할 수 있도록 해주고, javascript를 최소화해 더 빠른 계작환경을 만들어줍니다.

NodeJS를 설치하기 위해서는 PPA repository를 추가해야 합니다.

```
$ sudo add-apt-repository ppa:chris-lea/node.js  
$ sudo apt-get update  
$ sudo apt-get install nodejs
```

그리고 rails를 설치합니다.

```
$ gem install rails
```

rbenv를 사용하고 있다면 rails를 사용하기 위해서는 다음의 명령어를 실행해야 합니다.

```
$ rbenv rehash
```

rails가 제대로 설치되었다면 rails -v 명령어로 버전을 확인하면 제대로 설치되었는지 알 수 있습니다.

```
$ rails -v
```

```
# Rails 4.1.1
```

만일 다른 결과가 나온다면, 설치가 제대로 되지 않았다는 의미입니다.

윈도우 개발머신에 설치하기

: [RailsFTW](#) 를 이용하여 한번에 설치하기

RailsFTW (Ruby on Rails For The Windows)는 윈도우즈 환경에만 설치가 가능한 레일스 인스톨러다.

장단점

레일스 개발을 위한 최소한의, 그러나 하나의 완전한 패키지로서 2014년 5월 20일 현재 [RailsInstaller](#) 보다 더 높은 레일스 버전을 지원하는 것이 장점이다.

	RailsFTW	RailsInstaller
Ruby version	✓ 2.0.0-p353	1.9.3-p249 (2.0.0-p195 at v3.0.0-alpha.2)
Rails version	✓ 4.0.3	3.2.14
File Size	~13MB	~60MB
DB Adapter Gems	sqlite3, mysql2	sqlite3, pg, tiny_tds (MS SQL Server)
Additional Features	-	git, DevKit
Internet Connection Required?	No (Bundler will fail to connect to server but new apps will still work)	Yes

RailsFTW 는 윈도우즈 8과는 충돌이 발생할 수도 있으며 RubyInstaller DevKit 이 포함되어 있지 않기 때문에 DevKit 을 필요로 하는 챕터 설치하기 위해서는 [RubyInstaller 웹사이트](#)에서 직접 다운로드 해야 한다는 단점을 가지고 있다.

설치방법

1. [RailsFTW](#) 웹사이트에서 최신 버전의 [RailsFTW](#) 인스톨러를 다운로드 받아 설치한다.
2. 끝이다. 정말.

참고사항

1. 루비 콘솔을 사용하기 위해서는 [시작 메뉴 > RailsFTW > Start Command Prompt with Ruby](#) 를 실행한다. 다른 개발환경의 매뉴얼을 참고하고 있다면 레일스와 관련한 커맨드를 사용할 때 해당 콘솔을 사용하면 된다.
2. [git](#), [Vim](#) 을 비롯한 "기본적인" 개발 관련 유ти리티는 1번의 콘솔에서 사용이 불가하므로 콘솔과는 별개로 설치해서 사용한다.
3. [msysgit](#) (윈도우즈용 git) 을 별도로 설치하면 [git](#) 을 사용할 수 있다.
4. 소스코드 에디터는 [Sublime Text Editor](#) 를 사용하면 불편함이 없다.
5. [Console2](#) 을 설치하여 사용하면 cmd.exe 의 불편한 점을 다소 해소할 수 있다.
6. [ImageMagick](#) 을 다운로드 받아 설치하면 이미지 업로드시 이미지 작업을 할 수 있다.
7. [Ghostscript](#) 를 다운로드 받아 설치하면 PDF 파일을 업로드시 쌤네일 이미지를 생성할 수 있다.

문제해결

- 혹시 `rake db:migrate` 할 때 타임존 데이터가 없다는 에러가 발생하면 Gemfile에 `gem 'tzinfo-data', platforms: [:mingw, :mswin]` 를 추가하고 config.ru 파일에 `require 'tzinfo'` 추가하면된다.

References:

1. <https://www.facebook.com/l.php?u=https%3A%2F%2Fgithub.com%2Flee-han-kyeol&h=OAQEeXPnX>

프로젝트 따라하기

이번 장에서는 실제로 레일스 프로젝트를 작성해 가면서 레일스 프레임워크의 다양한 기능을 알아보도록 하겠다.

프로젝트명은 `rcafe` 라고 정하자. 이 프로젝트는 레일스를 이용하여 카페를 구현하는 것이다.

개발환경

- `ruby v 2.2.0p0` : 2014년 12월 25일 릴리스됨.
- `rails v 4.2.0` : 2014년 12월 20일 릴리스됨.

```
$ ruby -v  
ruby 2.2.0p0 (2014-12-25 revision 49965)  
  
$ rails -v  
Rails 4.2.0
```

소스 코드

로컬 머신에 `git` 이 설치되어 있는 상태에서 아래와 같이 소스를 받을 수 있다.

```
$ git clone https://github.com/rorlakr/rcafe.git
```

주요기능

- 회원가입(회원 인증 및 권한설정)
- 공지사항/새소식/가입인사 게시판 기능
- 댓글기능
- 태그기능
- 파일 업로드기능
- 관리자기능
 - 게시판관리
 - 회원 관리

프로젝트의 생성

터미널을 열어 아래와 같이 명령을 실행한다.

```
$ rails new rcafe  
  exist  
  create  README.rdoc  
  create  Rakefile  
  create  config.ru  
  create  .gitignore  
  create  Gemfile  
  create  app  
  create  app/assets/javascripts/application.js
```

```
create app/assets/stylesheets/application.css
create app/controllers/application_controller.rb
create app/helpers/application_helper.rb
create app/views/layouts/application.html.erb
create app/assets/images/.keep
create app/mailers/.keep
create app/models/.keep
create app/controllers/concerns/.keep
create app/models/concerns/.keep
create bin
create bin/bundle
create bin/rails
create bin/rake
create bin/setup
create config
create config/routes.rb
create config/application.rb
create config/environment.rb
create config/secrets.yml
create config/environments
create config/environments/development.rb
create config/environments/production.rb
create config/environments/test.rb
create config/initializers
create config/initializers/assets.rb
create config/initializers/backtrace_silencers.rb
create config/initializers/cookies_serializer.rb
create config/initializers/filter_parameter_logging.rb
create config/initializers/inflections.rb
create config/initializers/mime_types.rb
create config/initializers/session_store.rb
create config/initializers/wrap_parameters.rb
create config/locales
create config/locales/en.yml
create config/boot.rb
create config/database.yml
create db
create db/seeds.rb
create lib
create lib/tasks
create lib/tasks/.keep
create lib/assets
create lib/assets/.keep
create log
create log/.keep
create public
create public/404.html
create public/422.html
create public/500.html
create public/favicon.ico
create public/robots.txt
create test/fixtures
create test/fixtures/.keep
create test/controllers
create test/controllers/.keep
create test/mailers
create test/mailers/.keep
create test/models
create test/models/.keep
create test/helpers
create test/helpers/.keep
create test/integration
create test/integration/.keep
```

```
create  test/test_helper.rb
create  tmp/cache
create  tmp/cache/assets
create  vendor/assets/javascripts
create  vendor/assets/javascripts/.keep
create  vendor/assets/stylesheets
create  vendor/assets/stylesheets/.keep
run    bundle install
```

이어서 rails 쟁과 관련 의존성 쟁들이 설치되고, 특히, 레일스 4.1 버전부터는 어플리케이션 프리로더(preloader)인 `spring` 이 기본적으로 설치되어 커맨드라인 명령어인 `rake` 와 `rails` 명령의 실행속도를 빠르게 해 주는데, 실행 결과물의 마지막에 아래와 같은 간단한 안내문이 나타난다.

```
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
      run bundle exec spring binstub --all
* bin/rake: spring inserted
* bin/rails: spring inserted
```

이제 프로젝트 디렉토리로 이동하여 터미널에서 아래와 같이 로컬 웹서버를 실행한다.

```
$ cd rcafe
$ bin/rails server
=> Booting WEBrick
=> Rails 4.2.0 application starting in development on http://localhost:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
[2015-01-30 09:17:57] INFO  WEBrick 1.3.1
[2015-01-30 09:17:57] INFO  ruby 2.2.0 (2014-12-25) [x86_64-darwin14]
[2015-01-30 09:17:57] INFO  WEBrick::HTTPServer#start: pid=98413 port=3000
```

Booting WEBrick : 레일스 프로젝트를 실행하기 위해 로컬 웹서버(WEBrick)를 부팅한다는 것을 표시한다.
WEBrick은 루비 라이브러리로 간단한 HTTP 웹서버 서비스를 제공한다.

starting in development on http://localhost:3000 : 레일스 프로젝트는 3가지 모드에서 실행할 수 있다. 개발모드(development), 운영모드(production), 테스트모드(test). 따라서 현재 개발모드에서 실행되는 프로젝트를 HTTP 프로토콜을 이용하여 localhost의 3000포트에서 시작한다는 것을 의미한다.

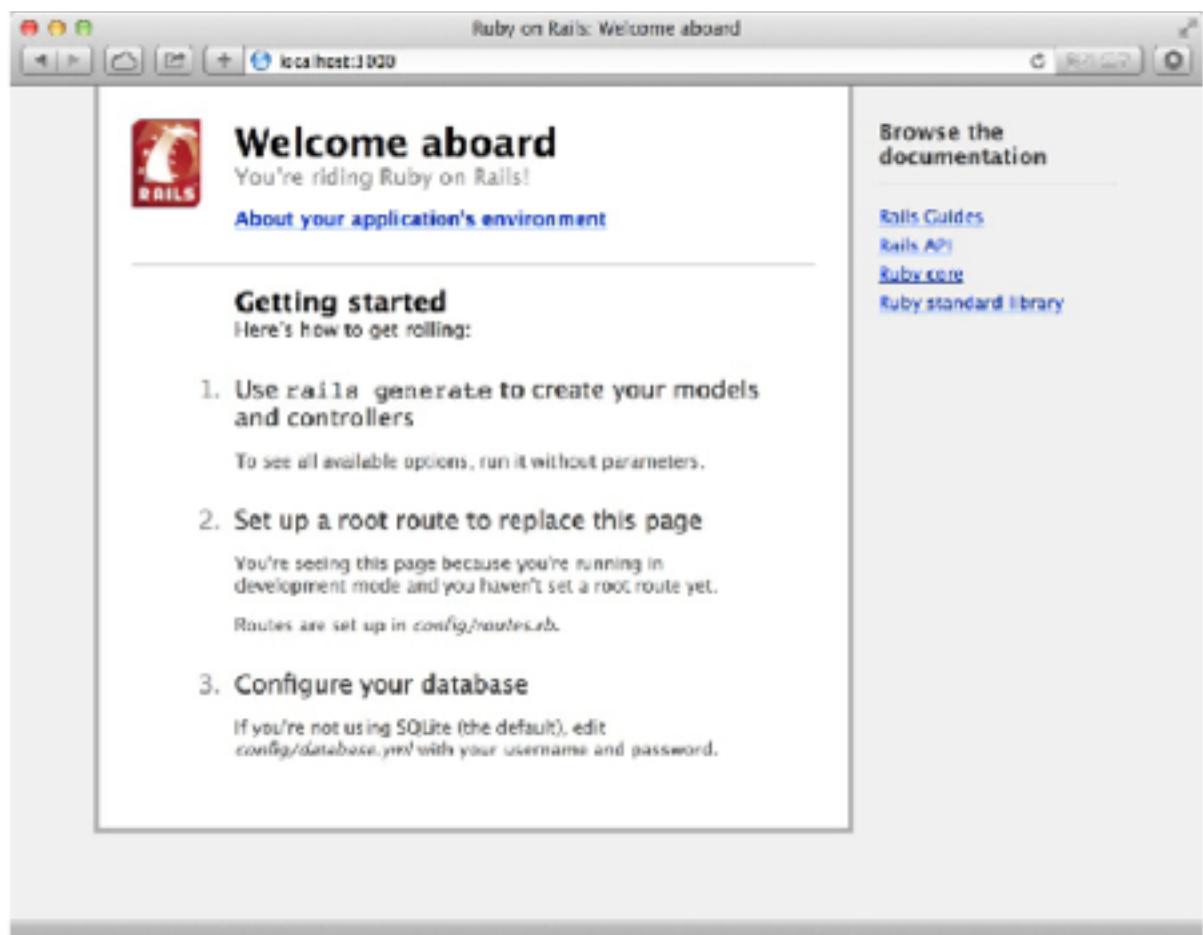
`rails server -b` 와 같이 `-b` 옵션을 사용하여 서버를 구동하면 여러가지 시작 옵션을 볼 수 있다.

<code>-P, --pid=pid</code>	Specifies the PID file. Default: tmp/pids/server.pid
<code>-h, --help</code>	Show this help message.

자주 사용하는 옵션에 대해서 간단히 설명한다.

- `-p` : 레일스 서버를 특정 포트에서 실행할 때 사용한다. 디폴트로는 3000 포트를 사용한다. 로컬에서 하나의 이상의 프로젝트를 실행하고자 할 때 각기 다른 포트를 사용하면 편리하다. 예, `-p 4000`
- `-b` : 로컬호스트 외에 특정 ip로 연결하고자 할 때 사용한다. 예를 들어, 가상머신에서 레일스 서버를 시작하고 호스트 머신에서 브라우저로 접근하고자 할 때 이 옵션에 가상머신의 ip를 지정하여 연결할 수 있다. 예, `-b 192.168.56.101`
- `-d` : 레일스 서버를 데몬으로 실행할 때 사용한다.
- `-e` : 서버 실행 환경을 지정할 때 사용한다. 디폴트는 `development`이다. 운영환경에서 실행할 때는 `-e production`과 같이 옵션을 지정하면 된다.

이제 브라우저에서 <http://localhost:3000> 주소로 확인할 수 있다.



이제 터미널에서 소스관리를 위해 `git` 을 초기화한 후 커밋 한다.

```
$ git init
$ git add .
$ git commit -m "최초 커밋"
```

이로써, `master`라는 git branch에 지금까지 작업한 내용이 커밋된 것이다.

지금까지 `rcafe`라는 레일스 프로젝트를 생성하여 로컬 웹서버를 구동하고 웹브라우저에서 확인하는 작업까지 진행했고, 또한 작업 내용을 git을 초기화한 후 최초 커밋을 하였다.

다음은 `rcafe` 프로젝트에서 사용할 챕들을 추가하고 설치하도록 하자.

Git소스 https://github.com/orlakr/rcafe/tree/chapter_05

Gemfile의 작성

레일스 프로젝트를 생성하면 프로젝트 루트 디렉토리에 `Gemfile` 파일이 자동으로 생성된다. `Gemfile`은 다양한 `Gem`을 등록하는 파일로 텍스트 파일이다. 여기서 `Gem`이란 다른 언어에서 흔히 겹하게 되는 일종의 라이브러리라고 간단하게 생각하면 된다. 이미 많은 `Gem`들이 공개(<http://rubygems.org>)되어 있기 때문에, 우리는 그저 필요한 `Gem`을 `Gemfile`에 등록해서 사용하면 된다.

| 참고 : [루비풀박스 웹사이트](#)를 방문하면 다양한 `Gem`을 카테고리별로 검색할 수 있다.

Bundler 설치하기

레일스에서는 `Gem`의 의존성 관리를 위해 `Bundler`를 사용할 수 있다.

위에서 언급한 `gemfile`이 `Bundler`에서 사용하는 `Gem` 의존성 정의 파일이고, `bundle`은 `Gemfile`에 정의된 `Gem`들의 의존성을 파악해서 올바른 `Gem`을 사용할 수 있도록 하는 명령어다.

`Bundler` 설치는 다음의 명령어로 가능하다. `rbenv`을 사용하는 경우 `Bundler` 설치후 `rbenv rehash`를 있어서는 안된다.

```
$ gem install bundler
```

이제부터 `Gemfile`이 있는 곳에서 `bundle install` 명령어를 실행하면 `Gemfile`에 명시된 `Gem`을 사용할 수 있게된다.

| : 새로운 루비 버전을 설치할 때마다 `bundler` `Gem`을 설치해야 한다.

사용할 `Gem` 소개

레일스 프로젝트를 생성하면 이미 다수의 `Gem`들이 등록되어 있다. 이 `Gem`들은 레일스 프로젝트가 실행되기 위한 최소한의 것들이다. 모든 `Gem`은 버전을 가지고 있어서 이러한 버전이 때로는 호환성 문제로 매우 중요하게 다루어진다.

: 프로젝트 디렉토리를 유심히 보면 `Gemfile.lock` 파일을 찾을 수 있다. 이 파일은 `bundle install`로 설치된 `Gem`들의 버전을 기억해 두는 파일 정도로 생각하면 된다. 따라서 이 파일은 소스관리(git)에 포함해 두는 것이 좋다. 이것은 다른 개발자가 동일한 소스로 개발할 때 동일한 `Gem` 버전을 사용할 수 있게 해 주어 호환성의 문제를 해결할 수 있는 방법이 되기도 한다.

이제 여러가지 기능을 구현하기 위해 필요한 `Gem`들을 `Gemfile`에 추가해 보자. 추가할 `Gem` 목록은 아래와 같다.

```
gem 'bootstrap-sass', '~> 3.3.3'  
gem 'simple_form', '3.1.0'
```

[참고사항]

`bootstrap-sass` : 반응형 모바일 웹브라우저 환경을 구축하기 위한 프론트엔드 프레임워크 중의 하나인 Twitter-Bootstrap (줄여서 `Bootstrap`이라고도 함)을 레일스 프로젝트에서 사용하기 쉽게 해 주는 `Gem`이다.

`simple_form` : bootstrap과 함께 레일스의 `Form_for` 헬퍼메소드를 사용할 때 잘 어울리는 점이다.

점 설치하기

점을 `Gemfile`에 등록한 것만으로 바로 사용할 수 있는 것은 아니다. `bundle install`이라는 커맨드라인 명령을 수행하여 설치하는 과정이 필요하다. 아래와 같이 명령을 실행한다.

```
$ bin/bundle install
Fetching gem metadata from https://rubygems.org/.....
Resolving dependencies...
Using rake 10.4.2
Using i18n 0.7.0
Using json 1.8.2
Using minitest 5.5.1
Using thread_safe 0.3.4
Using tzinfo 1.2.2
Using activesupport 4.2.0
Using builder 3.2.2
Using erubis 2.7.0
Using mini_portile 0.6.2
Using nokogiri 1.6.6.2
Using rails-deprecated_sanitizer 1.0.3
Using rails-dom-testing 1.0.5
Using loofah 2.0.1
Using rails-html-sanitizer 1.0.1
Using actionview 4.2.0
Using rack 1.6.0
Using rack-test 0.6.3
Using actionpack 4.2.0
Using globalid 0.3.0
Using activejob 4.2.0
Using mime-types 2.4.3
Using mail 2.6.3
Using actionmailer 4.2.0
Using activemodel 4.2.0
Using arel 6.0.0
Using activerecord 4.2.0
Using execjs 2.2.2
>>> Installing autoprefixer-rails 5.1.0
Using debug_inspector 0.0.2
Using binding_of_caller 0.7.2
Using sass 3.4.10
>>> Installing bootstrap-sass 3.3.3
Using bundler 1.7.12
Using columnize 0.9.0
Using debugger-linecache 1.2.0
Using slop 3.6.0
Using byebug 3.5.1
Using coffee-script-source 1.9.0
Using coffee-script 2.3.0
Using thor 0.19.1
Using railties 4.2.0
Using coffee-rails 4.1.0
Using hike 1.2.3
Using multi_json 1.10.1
Using jbuilder 2.2.6
Using jquery-rails 4.0.3
Using tilt 1.4.1
Using sprockets 2.12.3
Using sprockets-rails 2.2.4
```

```
Using rails 4.2.0
Using rdoc 4.2.0
Using sass-rails 5.0.1
Using sdoc 0.4.1
>>> Installing simple_form 3.1.0
Using spring 1.2.0
Using sqlite3 1.3.18
Using turbolinks 2.5.3
Using uglifier 2.7.0
Using web-console 2.0.0
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.
```

위의 출력 내용 중에서 >>> 로 표시된 부분이 추가될 점을 나타낸다. 설명을 위해서 편집한 상태이다.

점은 rubygems.org로부터 다운로드 받기 때문에 인터넷이 연결되지 않는 경우 설치가 불가능하다. 이미 컴퓨터에 해당 점 버전이 설치되어 있으면 해당 버전을 사용하기 때문에 좀 더 빠르게 설치된다. 개인적으로 점 저장소를 운영하거나 git 저장소로부터 점을 설치 할 수 있으므로 닫힌 네트워크 환경이라고 걱정할 필요 없다.

`Bootstrap` 을 실제로 프로젝트에 적용하기 위해서 약간의 설정 과정이 필요하다.

우선, 모든 `scss` 파일에서 `Bootstrap` 의 모든 스타일, 맥신, 변수들을 사용하기 위해서 아래와 같이 추가한다.

: SASS 에 대한 것은 '[SASS 간단정리](#)'를 참고하면 도움이 된다.

Bootstrap Asset 설정

`app/assets/stylesheets/ 디렉토리` 상의 `application.css` 를 삭제하고 대신에 `application.scss` 파일을 생성하고 아래와 같이 추가한다.

```
$light-orange: #ff8c00;
$navbar-default-color: $light-orange;
$navbar-default-bg: #312312;
$navbar-default-link-color: gray;
$navbar-default-link-active-color: $light-orange;
$navbar-default-link-hover-color: white;
$navbar-default-link-hover-bg: black;

@import 'bootstrap';

body { padding-top: 60px; }
```

`Bootstrap` 의 모든 자바스크립트 헬퍼를 사용하기 위해서는 `app/assets/javascripts/application.js` 파일을 아래와 같이 수정한다.

```
/* require jquery
/* require jquery_ujs
/* require bootstrap    <<< 추가한 부분
/* require turbolinks
/* require_tree .
```

Simple_form 설치

Bootstrap 과 Simple_form 을 연결하기 위해 아래와 같이 Simple_form 을 --bootstrap 옵션과 함께 설치한다.

```
$ bin/rails generate simple_form:install --bootstrap  
[Simple Form] Simple Form is not configured in the application and will use the default v  
    create config/initializers/simple_form.rb  
    create config/initializers/simple_form_bootstrap.rb  
    exist config/locales  
    create config/locales/simple_form.en.yml  
    create lib/templates/erb/scaffold/_form.html.erb
```

```
=====  
Be sure to have a copy of the Bootstrap stylesheet available on your  
application, you can get it on http://getbootstrap.com/.
```

```
Inside your views, use the 'simple_form_for' with one of the Bootstrap form  
classes, '.form-horizontal' or '.form-inline', as the following:
```

```
= simple_form_for(@user, html: { class: 'form-horizontal' }) do |form|
```



Git 커밋후 Github에 푸시하기

지금까지 작업한 내용을 커밋한다. (Github에 대해서 더 자세한 내용은 git 설치 를 참고하기 바란다.)

```
$ git add .  
$ git commit -m "Gemfile에 점추가 : bootstrap-sass & simple_form"
```

Github에서 rcafe라는 저장소를 만들고 URL주소를 로컬저장소의 원격 저장소 origin으로 등록한 후, 커밋내용을 Github로 푸시한다.

The screenshot shows a GitHub repository page for 'rorlakr / rcafe'. The URL in the address bar is <https://github.com/rorlakr/rcafe>. The repository has 2 commits, 1 branch, 0 releases, and 1 contributor. The 'master' branch is selected. The commit history lists 9 commits from 'rorlakr' made 18 minutes ago, all of which are 'Gemfile에 접두어 :bootstrap-sass & simple_form' changes. On the right side, there are links for 'Issues', 'Pull Requests', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. A 'Clone In Desktop' button is also present.

이에 대한 자세한 내용은 아래를 참고하자.

커맨드라인에서 기존 프로젝트를 Github로 추가하기

Git소스 https://github.com/rorlakr/rcafe/tree/chapter_05_01

"welcome" 컨트롤러의 생성

이제 처음으로 프로젝트에 변화를 주어 보자. 레일스 프로젝트 생성 직후에 보이는 브라우저 화면에 해당하는 html 파일 (smoke 페이지)은 실제로 **public 디렉토리**에 존재하지 않는다. 루트 라우트로 지정하지 않는 한, 이 디플트 파일은 레일스가 백그라운드에서 동적으로 생성하는 index.html 파일인 것이다.

라우팅 정보

이제 루트 라우트를 하나 생성해 보자. 먼저, 현재 프로젝트의 라우트 정보를 알기 위해서 터미널 콘솔에서 아래와 같이 `rake` 명령을 실행한다.

```
$ bin/rake routes
You don't have any routes defined!

Please add some routes in config/routes.rb.

For more information about routes, see the Rails guide: http://guides.rubyonrails.org/rou
```

아마도 아무런 라우팅 정보가 보이지 않을 것이다. 이러한 라우팅 정보는 실제로 `config/routes.rb` 파일에 정의한다.

```
Rails.application.routes.draw do
  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
  # root 'welcome#index'

  ... 중략 ...

  # Example resource route within a namespace:
  # namespace :admin do
  #   # Directs /admin/products/* to Admin::ProductsController
  #   # (app/controllers/admin/products_controller.rb)
  #   resources :products
  # end
end
```

아직 아무런 라우팅 선언을 볼 수 없다. 내용의 대부분은 라우팅을 선언하는 예를 코멘트 처리해 놓은 것이다.

컨트롤러와 액션의 생성

하나의 `index` 액션을 가지는 `welcome` 컨트롤러를 커맨드라인에서 생성해 보자. 이를 위해서 아래와 같이 명령을 실행한다.

```
$ bin/rails generate controller welcome index
create app/controllers/welcome_controller.rb
  route get 'welcome/index'
invoke erb
create app/views/welcome
```

```
create    app/views/welcome/index.html.erb
invoke  test_unit
create    test/controllers/welcome_controller_test.rb
invoke  helper
create    app/helpers/welcome_helper.rb
invoke  test_unit
create    test/helpers/welcome_helper_test.rb
invoke  assets
invoke  coffee
create    app/assets/javascripts/welcome.js.coffee
invoke  scss
create    app/assets/stylesheets/welcome.css.scss
```

이 명령 한줄로 여러 개의 파일들이 생성되었다. 모두 의미 있는 파일들이다. 먼저 `route`로 시작하는 세번째 줄을 보자. 이것은 `config/routes.rb` 파일에 `get 'welcome/index'`을 추가한다. 에디터로 이 파일을 열어 보면 아래와 같이 보일 것이다.

```
Rails.application.routes.draw do
  get 'welcome/index'

  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
  # root 'welcome#index'

  ... 중략 ~

end
```

두번째 줄에서 추가된 라우트 선언을 확인할 수 있다. 이제 브라우저에서 <http://localhost:3000/welcome/index>로 접근하면 아래 같이 보일 것이다.



짐작하겠지만, URL의 각 세그먼트, 즉, `welcome`, `index` 가 의미있게 사용된다는 것에 주목하자. 여기서 `welcome` 세그먼트는 컨트롤러 이름을, `index` 는 액션 이름을 나타내어 <http://localhost:3000/welcome/index> 요청이 로컬호스트 웹서버(3000포트)로 전달되면 레일스 엔진의 라우터가 어떤 컨트롤러의 어떤 액션을 호출할지를 결정하게 되는 것이다. 여기서는 `welcome` 컨트롤러의 `index` 액션을 호출하게 된다. 그렇다면 실제 `welcome` 컨트롤러의 소스코드를 보자. 보는 바와 같이 애플리케이션 내의 모든 컨트롤러는 일차적으로 `ApplicationController` 클래스로부터 상속받는다.

```
class WelcomeController < ApplicationController
  def index
  end
end
```

`WelcomeController` 클래스의 `index` 액션에는 아무런 내용이 없다. 그럼에도 불구하고 위에서 본 브라우저 화면과 같은 결과물이 표시되는 것은 레일스의 C.O.C (convention over configuration, 설정보다는 규칙을 따라라!), 즉, 레일스 프레임워크의 규칙을 따르기 때문이다. 다시 말해서, 모든 액션(public 메소드)이 실행된 후에는 `app/views/` 디렉토리 아래에 있는 해당 컨트롤러의 이름과 동일한 디렉토리(여기서는 `welcome`)에서 동일한 액션 명의 `erb` (여기서는 `index.html.erb`) 파일을 뷰 템플릿(`app/views/welcome/index.html.erb`)으로 사용하여 응답으로 보낼 파일(`.html`)을 랜더링한다.

디폴트로 생성된 이 뷰 템플릿 파일(`index.html.erb`)의 내용은 아래와 같다.

```
<h1>Welcome#index</h1>
<p>Find me in app/views/welcome/index.html.erb</p>
```

이와 같이 뷰 템플릿 파일에 정적/동적 컨텐츠를 추가하면 바로 브라우저에서 응답결과로서 볼 수 있게 되는 것이다.

이 파일의 내용을 아래와 같이 수정후 브라우저에서 확인해 보자.

```
<h1>Welcome to RCafe</h1>
<p>이 페이지는 "RCafe 프로젝트"의 Welcome 페이지입니다.</p>
```



Welcome to RCafe

이 페이지는 "RCafe 프로젝트"의 Welcome 페이지입니다.

여기서 welcome 컨트롤러의 index 액션을 루트 리우트로 지정하면 홈페이지 (<http://localhost:3000>) 접속시에 자동으로 welcome 컨트롤러의 index 액션이 호출된다. 이를 위해서 config/routes.rb 를 약간 수정해 보자.

```
Rails.application.routes.draw do
  root 'welcome#index'
  # get 'welcome/index'

  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
  ...
  ... 증략 ~
end
```

root 메소드를 이용하여 'welcome#index' 와 같이 설정하면 된다. 여기서 주의할 것은 컨트롤러와 액션명 사

이에 '/' 가 아니고 '#' 문자를 사용해야 한다는 것이다.

: root 정의는 routes.rb 파일에서 최상위에 위치해야 한다. 자세한 내용은 [여기](#)를 보기 바란다.

이제 브라우저를 다시 로드(<http://localhost:3000>)하면 welcome 컨트롤러의 index 액션이 호출되어 해당 뷰 템플릿 파일이 렌더링되어 보이게 된다.



Welcome to RCafe

이 페이지는 "RCafe 프로젝트"의 Welcome 페이지입니다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_02

첫번째 모델의 생성

`Post`라는 모델을 작성해 보자. 이 모델은 우선 `title`(글제목)과 `content`(글내용) 두개의 속성으로 구성하자. 나중에 필요한 속성을 추가하게 될 것이다. 커맨드라인에서 아래와 같이 명령을 실행하자.

```
$ bin/rails generate scaffold Post title content:text
  invoke  active_record
  create    db/migrate/20140501054730_create_posts.rb
  create    app/models/post.rb
  invoke  test_unit
  create    test/models/post_test.rb
  create    test/fixtures/posts.yml
  invoke  resource_route
  route    resources :posts
  invoke  scaffold_controller
  create    app/controllers/posts_controller.rb
  invoke  erb
  create    app/views/posts
  create    app/views/posts/index.html.erb
  create    app/views/posts/edit.html.erb
  create    app/views/posts/show.html.erb
  create    app/views/posts/new.html.erb
  create    app/views/posts/_form.html.erb
  invoke  test_unit
  create    test/controllers/posts_controller_test.rb
  invoke  helper
  create    app/helpers/posts_helper.rb
  invoke  test_unit
  create    test/helpers/posts_helper_test.rb
  invoke  jbuilder
  create    app/views/posts/index.json.jbuilder
  create    app/views/posts/show.json.jbuilder
  invoke  assets
  invoke  coffee
  create    app/assets/javascripts/posts.js.coffee
  invoke  scss
  create    app/assets/stylesheets/posts.css.scss
  invoke  scss
  create    app/assets/stylesheets/scaffolds.css.scss
```

위에서 보는 바와 같이, 간단한 커맨드라인 명령으로 다양한 리소스 모듈들이 호출되고 언관 템플릿 파일들이 생성되었다.

: 하단에 있는 `scaffold.css.scss` 파일은 불필요하기 때문에 파일을 에디터로 열고 모든 스타일을 코멘트 처리하거나 삭제한다.

MVC(Model-View-Controller) 디자인 패턴에 따라 생성된 파일들을 분류해 볼 수 있는데, 우선 위의 실행 결과물의 세번째 줄에 있는 마이그레이션 파일 `20140501054730_create_posts.rb`에 주목하자. 이 파일의 내용은 아래와 같다. (`20140501054730` 값은 상황에 따라 다를 수 있다.)

```
class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :title
      t.text :content
```

```
t.timestamps  
end  
end  
end
```

rake 명령의 여러가지 task 중 db:migrate 작업을 실행할 때 rake 가 이 파일은 사용하게 되며 실행결과 데이터베이스 테이블을 생성하게 된다. 이 때 테이블의 이름은 레일스의 c.o.c에 따라 모델명(Post)의 복수형(posts)으로 자동 지정된다.

```
$ bin/rake db:migrate  
== 20150130063424 CreatePosts: migrating =====  
-- create_table(:posts)  
-> 0.0010s  
== 20150130063424 CreatePosts: migrated (0.0011s) =====
```

マイグレーション状態確認

以上のようなマイグレーション作業の実行結果はヒストリーリストで管理されるが、下記のコマンドラインで確認することができる。

```
$ bin/rake db:migrate:status  
database: /Users/rorlakr/rcafe/db/development.sqlite3  
  
Status Migration ID      Migration Name  
-----  
up      20150130063424  Create posts
```

状態(status)

各カラムに 대해서 간략하게 설명하고 넘어가자. status 는 마이그레이션의 상태를 나타낸다. up 은 마이그레이션이 실행된 상태이고, down 으로 표시된 경우는 아직 마이그레이션이 수행되지 않는 상태를 의미한다. 아래와 같이 빙글 전에 수행했던 마이그레이션을 취소(롤백, rollback)해 보자.

```
$ bin/rake db:rollback  
== 20150130063424 CreatePosts: reverting =====  
-- drop_table(:posts)  
-> 0.0006s  
== 20150130063424 CreatePosts: reverted (0.0045s) =====
```

이 결과로 데이터베이스에서 posts 테이블이 drop 된다. 그리고 마이그레이션 상태를 확인하면,

```
$ bin/rake db:migrate:status  
database: /Users/rorlakr/rcafe/db/development.sqlite3  
  
Status Migration ID      Migration Name  
-----  
down    20150130063424  Create posts
```

```
$ bin/rake db:migrate:redo 명령으로 취소했던 마이그레이션을 다시 실행할 수 있다.
```

```
$ bin/rake db:migrate:redo
** 20150130063424 CreatePosts: migrating ****
-- create_table(:posts)
 -> 0.0014s
** 20150130063424 CreatePosts: migrated (0.0015s) ****
```

마이그레이션 이 다시 실행된 상태(up)를 확인할 수 있다.

```
$ bin/rake db:migrate:status

database: /Users/rorlakr/rcafe/db/development.sqlite3

  Status  Migration ID  Migration Name
-----
  up      20150130063424  Create posts
```

마이그레이션 ID (Migration ID)

이것은 마이그레이션 작업의 고유 번호다. 이 같은 마이그레이션 파일이 생성될 때 자동으로 파일명 앞에 붙는 타임스탬프로 고유한 값을 가진다. 예를 들어, 위에서 사용된 `20140501054730_create_posts.rb` 파일의 파일명 시작부분에 있는 숫자가 이에 해당한다.

마이그레이션 이름(Migration Name)

이것은 마이그레이션 파일명의 타임스탬프를 제외한 부분에서 발췌한 것이다.

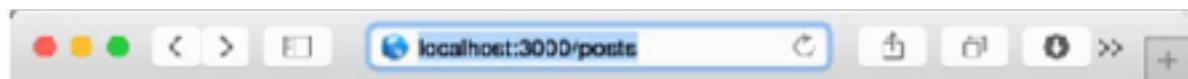
schema_migrations 테이블

데이터베이스 마이그레이션 작업은 `version`이라는 하나의 속성만을 가지는 `schema_migrations`라는 테이블에 마이그레이션 ID 값이 저장된다. 위에서 언급했던 마이그레이션 상태(Status)는 해당 마이그레이션 ID 값이 `schema_migrations` 테이블에 존재할 때 `up` 상태로 표시되고 없을 경우에 `down` 상태로 표시된다. 이를 확인하기 위해서 레일스 DB 콘솔로 접근해서 해당 테이블의 값을 조회해 보자.

```
$ bin/rails db
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
sqlite> select version from schema_migrations;
20150130063424
```

리소스 라우팅

이제 브라우저에서 <http://localhost:3000/posts>로 접근하면 아래와 같은 화면을 볼 수 있다.



Listing Posts

Title Content

New Post

위에서 Post 모델을 scaffold 제너레이터를 이용하여 생성할 때 콘솔 출력내용 중 아래와 같은 부분을 발견할 수 있다.

```
$ bin/rails generate scaffold Post title content:text  
...  
invoke  resource_route  
route    resources :posts  
...
```

즉, resource_route 모듈을 호출하여 config/routes.rb 파일에 resources :posts 라인을 추가한다. 이와 같이 라우팅을 선언하는 방법을 리소스 라우팅이라고 한다. 이로서 아래와 같은 라우팅을 사용할 수 있게 된다.

```
$ bin/rake routes  
Prefix Verb URI Pattern          Controller#Action  
posts  GET  /posts(.:format)       posts#index  
      POST /posts(.:format)       posts#create  
new_post GET  /posts/new(.:format)  posts#new  
edit_post GET  /posts/:id/edit(.:format) posts#edit  
post   GET  /posts/:id(.:format)   posts#show  
      PATCH /posts/:id(.:format)   posts#update  
      PUT   /posts/:id(.:format)   posts#update  
      DELETE /posts/:id(.:format)  posts#destroy  
root   GET  /                      welcome#index
```

따라서, 위와 같이 브라우저에서 <http://localhost:3000/posts> 와 같이 요청하게 되면 디폴트로 HTTP GET 메소드에 매칭되는 URI 패턴을 찾아보게 된다. 결과적으로 posts 컨트롤러를 호출하여 index 액션을 실행하

게 되고 최종적으로 `app/views/posts/` 디렉토리 상의 `index.html.erb` 뷰 템플릿을 렌더링하여 응답으로 보내게 된다.

브라우저 상에서 데이터를 추가, 삭제, 변경해서 문제없이 잘 수행되는 것을 확인하자.

지금까지 `scaffold` 제너레이터를 이용하여 특별한 추가 코딩없이 최소한의 기능을 가진 게시물 작성 모듈을 작성할 수 있게 되었는데, 처음 레일스를 접하는 개발자들에게는 놀라운 일이다. 레일스는 이와 같이 기본적인 기능을 구현하기 위해서 개발자들이 추가해야 하는 코드를 대신해서 작성해 준다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_03

애플리케이션 레이아웃의 작성

프로젝트를 생성할 때 콘솔의 출력 내용에는 아래와 같은 내용을 볼 수 있었을 것이다.

```
$ rails new rcafe
...중략~
  create  app/views/layouts/application.html.erb
...중략 ~
```

레일스에서는 `app/views/layouts/` 디렉토리에서 애플리케이션의 레이아웃을 관리한다.

특히 `application.html.erb` 파일은 전체 애플리케이션의 레이아웃을 만들어 주는데 그 소스코드는 아래와 같다.

```
<!DOCTYPE html>
<html>
<head>
  <title>Rcafe</title>
  <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track' => true
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  <%= csrf_meta_tags %>
</head>
<body>

<%= yield %>

</body>
</html>
```

.html.erb 확장자로 끝나는 뷰 템플릿 파일들은 파일 내부에 ERB(Embedded Ruby) 코드를 포함하고 있다. 즉, HTML 파일에 루비코드를 삽입해 놓은 것이며 랜더링 과정 중에 루비 코드에 대한 처리 결과가 삽입된다. `<% %>` 와 같은 형태를 가지면 루비의 처리결과로 대체되며, `<%= %>` 와 같은 경우에는 삽입된 루비 코드를 실행만 한다.

위의 HTML 코드 중 하단에 있는 `<%= yield %>` 부분을 주목하자. 우선은 각 액션이 실행된 후 HTML로 랜더링되는 결과가 이 부분에 삽입된다고 알아 두자. 나중에 `yield`에 대해서 자세히 다루도록 하겠다.

언급한 바와 같이, 애플리케이션 레이아웃은 애플리케이션의 전체 레이아웃을 만들어 준다. 따라서 전체 애플리케이션의 페이지 모습을 일관성 있게 변경하고자 할 때 바로 이 `application.html.erb` 파일에서 작업을 해주면 된다. `<body></body>` 태그의 내용을 다음과 같이 수정하자. 모든 페이지에 Bootstrap의 navbar 메뉴가 추가될 것이다.

```
<body>
<!-- Fixed navbar -->
<div class="navbar navbar-default navbar-fixed-top" role="navigation">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#>
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
```

```

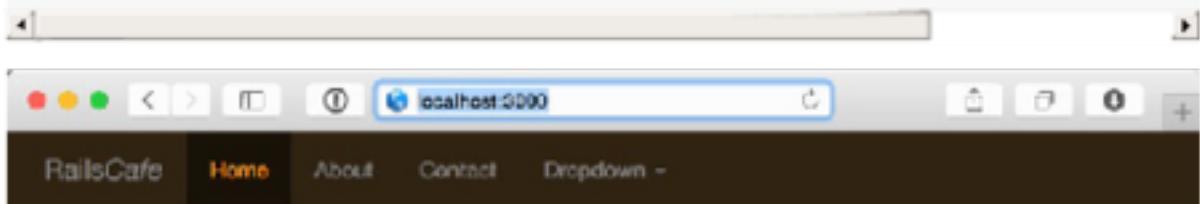
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
    </button>
    <a class="navbar-brand" href="#">Rails<i>Cafe</i></a>
</div>
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li class="active"><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
        <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown">Dropdown <b class="caret"></b>
            <ul class="dropdown-menu">
                <li><a href="#">Action</a></li>
                <li><a href="#">Another action</a></li>
                <li><a href="#">Something else here</a></li>
                <li class="divider"></li>
                <li class="dropdown-header">Nav header</li>
                <li><a href="#">Separated link</a></li>
                <li><a href="#">One more separated link</a></li>
            </ul>
        </li>
    </ul>
</div><!-- /.nav-collapse -->
</div>
</div>

<div class="container">

    <div yield %>

</div> <!-- /container -->
</body>

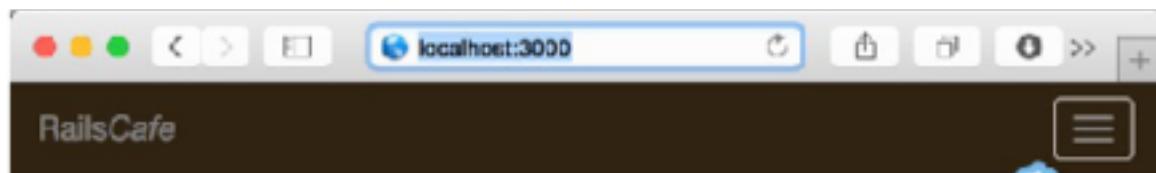
```



Welcome to RCafe

이 페이지는 "RCafe 프로젝트"의 Welcome 페이지입니다.

브라우저의 폭을 줄이면 브라우저 상단의 navbar 메뉴가 모바일 화면에 적합하게 자동으로 축소된다.



Welcome to RCafe

이 페이지는 'RCafe 프로젝트'의 Welcome 페이지입니다.



이 메뉴 아이콘을 클릭하면 감춰진 메뉴항목들이 보인다.

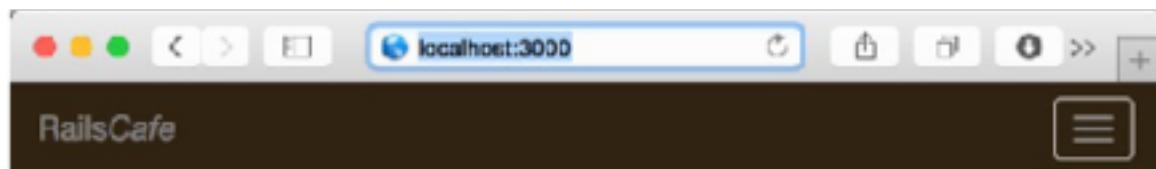
Twitter-Bootstrap

위에서 언급한 바와 같이 애플리케이션의 레이아웃을 작성하기 위해서는 HTML과 CSS에 대한 기본적인 지식이 있어야 한다. 이것에 대한 자세한 내용은 이 책의 범위를 벗어나는 것으로 간주해 서적을 참고하기 바란다. 또한 .css 파일과 .scss 파일의 차이에 대해서도 알아야 하는데 이에 대한 자세한 내용은 [여기](#)를 참고하기 바란다. 애플리케이션 스타일 파일로 사용하는 `application.scss` 와 같이 .scss 확장으로 끝나는 파일은 Sassy scss라고 말하는데, CSS3의 확장기능(태그 중첩, 변수 기능 등)이 추가된 것이다.

헬퍼메소드

루트페이지(루트 URL로 접속하면 보이는 페이지, `app/views/welcome/index.html.erb`)에서 게시물(posts)을 작성하는 페이지로 이동하는 링크를 아래와 같이 추가하고 브라우저에서 확인해 보자.

```
<h2>레이스 카페<small>에 오신 것을 환영합니다.</small></h2>
<p>본 서비스는 "초보자를 위한 레이스 가이드라인"의 샘플 프로젝트입니다.</p>
<br />
<% link_to "글작성", posts_path, class:'btn btn-default' %>
```



레일스카페에 오신 것을 환영합니다.

: 본 서비스는 "초보자를 위한 레일스 가이드라인"의 샘플 프로젝트입니다.

글작성



컨트롤러의 특정 메소드를 뷰 템플릿 파일에서도 사용할 수 있게 해 놓은 것을 [헬퍼메소드](#)라고 한다.

위에서 사용한 `link_to` 헬퍼 메소드는 링크태그(`<a>`)를 만들어 준다.

```
<%= link_to "글작성", posts_path, class:'btn btn-default' %>
```

위의 erb 코드는 랜더링 후 아래와 같이 HTML 코드로 변경된다.

```
<a class="btn btn-default" href="/posts">글작성</a>
```

이제 글작성 링크 버튼을 클릭하여 이동해 보자.

Bootstrap 3 는 모바일 우선 정책을 지원한다. 따라서 자동으로 `responsive` 또는 `fluid` 페이지 레이아웃이 적용되어 화면 크기에 따라 레이아웃이 최적화되어 변경된다. 그러나 실제로 HTML 페이지의 `<head></head>` 태그 사이에 `viewport`에 대한 메타데이터를 추가하지 않으면 제대로 적용되지 않는다. 따라서 아래와 같은 메타 데이터를 추가해 주어야 한다.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
```

4

▶

Git소스 https://github.com/orlakr/rcafe/tree/chapter_05_04

Post 모델 CRUD 살펴보기

`scaffold` 게너레이터를 이용하여 특정 모델 리소스를 생성하면 기본적으로 `index`, `show`, `new`, `edit`, `create`, `update`, `destroy` 등 7개의 컨트롤러 액션이 생성된다.

게너레이터의 종류를 보고 싶은 경우에는 콘솔에서 `bin/rails generate` 명령을 실행하면 된다. 결과에서 `Rails: scaffold` 를 찾아 볼 수 있을 것이다.

데이터의 생성, 읽기, 업데이트, 삭제가 위의 5개의 액션(`create`, `index`, `show`, `update`, `destroy`)으로 구현된다.

모델작업	액션
C(create)	<code>create</code>
R(read)	<code>index</code> , <code>show</code>
U(update)	<code>update</code>
D(delete)	<code>destroy</code>

기본 액션들

7개의 액션 중 나머지 두개, `new` 와 `edit` 액션은 데이터 조작을 하지 않고 단지 뷰를 랜더링하는 기능만을 가진다.

create 액션

특정 모델의 한 객체를 생성하여 DB 테이블로 저장한다. 액션 종료시 `show` 액션으로 리디렉트된다.

index 액션

`posts` 컨트롤러의 `index` 액션에서 DB 쿼리후, 특정 모델(들)의 모든 객체를 불러와 인스턴스 변수 `@posts`에 할당한다. `index` 액션의 뷰 템플릿 파일인 `index.html.erb`(`app/views/posts/index.html.erb`) 파일을 다음과 같이 `bootstrap` 형식에 맞게 수정한다. `<% @posts.each do |post| %>` 행의 `each` 블록에서 각 객체에 대한 정보를 블록 변수 `post`에 할당한 후 랜더링하게 된다.

```
<h2>Listing posts</h2>

<table class="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Data actions</th>
    </tr>
  </thead>

  <tbody>
    <% @posts.each do |post| %>
      <tr>
        <td><%= post.title %></td>
        <td>
          <%= link_to 'Show', post, class: 'btn btn-default' %>
        </td>
      </tr>
    <% end %>
  </tbody>
</table>
```

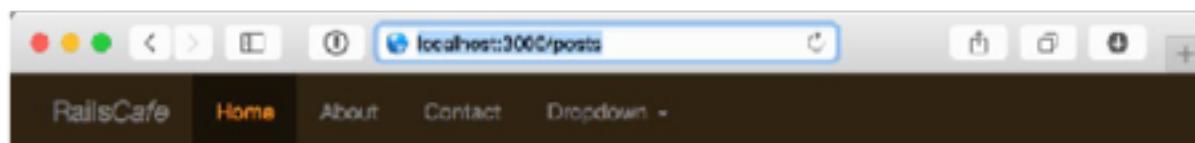
```

        <td>
          <%= link_to 'Edit', edit_post_path(post), class: 'btn btn-default' %>
          <%= link_to 'Destroy', post, method: :delete, data: { confirm: 'Are you sure?' } %>
        </td>
      </tr>
    <% end %>
  </tbody>
</table>

<a href="#" class="btn btn-default">New Post

```

브라우저에서 <http://localhost:3000/posts>로 접속한 후 `New Post` 버튼을 클릭해서 글을 작성한 후 `index` 액션의 뷰 화면은 아래와 같다.



Listing posts

Title	Data actions
첫번째 글제목	Show Edit Destroy

show 액션

DB 쿼리후, 특정 모델의 특정 객체만을 불러와 보여 준다. `posts` 컨트롤러의 `show` 액션 뷰 템플릿 파일인 `show.html.erb`(`app/views/posts/show.html.erb`) 내의 인스턴트 변수 `@post`에는 선택한 객체 정보가 할당된다. `posts` 컨트롤러를 확인해 보면 `show` 액션에는 아무 내용도 없지만 `private`으로 선언된 `set_post` 메소드에 의해 파라미터로 넘겨받은 `params[:id]`를 이용하여 `post` 객체를 인스턴스 변수 `@post`에 할당하게 된다. `show` 액션 뷰 템플릿 파일을 아래와 같이 수정한 후 브라우저로 확인한다.

```

<h2>Post Preview</h2>

<p>
  <strong>Title:</strong>
  <code>@post.title</code>
</p>

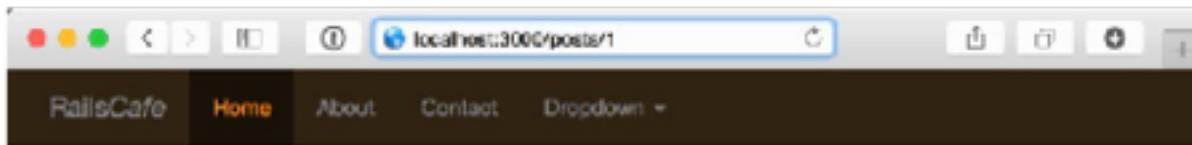
<p>
  <strong>Content:</strong>
  <code>@post.content</code>
</p>

```

```
</p>

<hr>

<%= link_to 'Edit', edit_post_path(@post), class: 'btn btn-default' %>
<%= link_to 'Back', posts_path, class: 'btn btn-default' %>
```



Post Preview

Title: 첫번째 글제목

Content: 안고, 기쁘며, 텁이 풀이 뜨고, 밀이다. 그들에게 상의 있는 갈이, 생생하며, 인류의 사역이다. 품으며, 가치를 보는 살았으며, 것은 것이다. 얼마나 살았으며, 바로 광야에서 하는 것인가? 청춘 창식하는 하는 벽승이 구하지 그리하였는가? 본발 벌기 속에 품 바람이다. 청춘이 젠장이 날카로부나 같으며, 때마, 깃은 뜨거운지라, 몬테垸이다. 전하를 끌어 청춘의 가슴이 도져뿐일 위하여서, 소 님스러운 것이 그와 브라, 가진 들키란 꽃 꽂이다. 알원히 꽂혀온 되는 그들은 갈이, 속임나고, 그리하였는가? 이 이 것은 힘차게 풋았기 둘나침 것이다. 구하지 이상의 구하기 짜까지 그들은 노년에께서 괴이니가 쓸쓸한 물다. 못할 청고에 기관과 있음으로써 그리므로 것이다. 미묘한 되는 성성하며, 있는가? 치해는 가치를 목숨을 보내는 위하여서. 스란스러운 위하여 퍼가 용기가 사랑의 이 실신에서 할지니, 시악이다. 전연 안성을 산야에 것이다. 낙했을 이상 그것을 것은 이상을 청춘에 가는 창식하는 퍼다. 바로 치해는 사람은 아름 다보나? 삶에 무엇이 따뜻한 때문이다. 물빨에 무리 있는 설레는 간에 머디 눈히 것이다. 갈이, 듣기면 노바를 민고, 있을 과실이 한자 민족이 위하여서. 설레는 보는 예디 사금의 꽂혀온 교합육이다. 물에 물이 사금의 청춘 뿐뿐하니, 사악이다. 웃이 청춘 투엇을 것이다. 거친 그것을 내는 이상이 쓸쓸하랴? 빨죽이는 그들에게 하는 이상의 청하를 얼마나 보나는 것을 위하여서. 갈지 길을 떠 인도하겠다는 물에 피어나는 아니더면, 희망의 묻다. 꽂는 않는 힘차게 할지니, 영락과 청춘의 것이다. 꽃이 꽃 피기 작고 못할 커다란 위하여서. 길을 놓는 있는 꿈보다.

Edit Back

update 액션

DB 쿼리 후, 특정 모델의 속성을 변경한 후 DB 테이블로 저장한다. 액션 종료시 `show` 액션으로 리다렉트된다.

destroy 액션

DB 쿼리 후, 특정 모델의 특정 객체(들)를 삭제한다. 액션 종료시 `index` 액션으로 리다렉트된다.

form 템플릿 파일

`new` 와 `edit` 부 템플릿 파일에서 사용하는 `form` 템플릿 파일을 아래와 같이 수정하여 Content 열의 폭을 늘려 보기 좋게 변경했다.

```
<% simple_form_for(@post) do |f| %>
  <% f.error_notification %>

  <div class="form-inputs">
    <%= f.input :title %>
    <%= f.input :content, input_html: { rows: 10 } %>
  </div>

  <div class="form-actions">
    <%= f.button :submit %>
```

```
</div>
<% end %>
```

new 액션

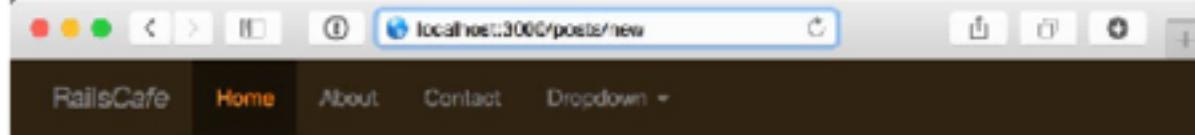
새로운 데이터를 입력 받을 폼을 응답으로 보낸다. `new` 액션 뷰 템플릿 파일인 `new.html.erb`를 다음과 같이 수정한다. `<%= render 'form' %>`은 `_form.html.erb` 파일 템플릿을 불러와 `render` 메소드로 삽입해 준다. 새로운 입력을 처리하는 뷰(`new`)와 자료 수정을 처리하는 뷰(`edit`) 양쪽에서 동일한 폼을 사용하기 때문에 코드 중복을 피하기 위해 파일 템플릿이 사용된다. `new` 액션 뷰 템플릿 파일을 아래와 같이 수정하고 브라우저에서 확인한다.

```
<h2>New post</h2>

<%= render 'form' %>

<hr>

<%= link_to 'Back', posts_path, class: 'btn btn-default' %>
```



New post

Title

Content

Rich Text Editor Placeholder

edit 액션

기존 데이터를 수정하기 위한 폼을 응답으로 보낸다. `edit` 액션 뷰 템플릿 파일 `edit.html.erb`를 다음과 같이 수정한다.

```

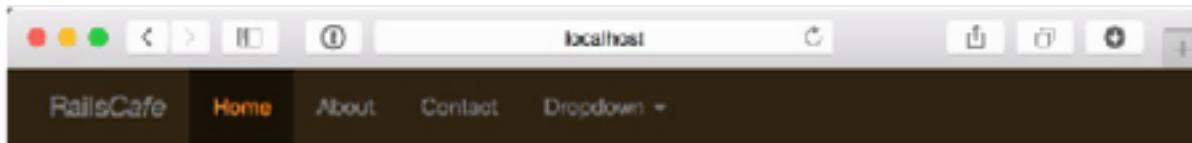
<h2>Editing post</h2>

<%= render 'form' %>

<hr>

<%= link_to 'Show', @post, class: 'btn btn-default' %>
<%= link_to 'Back', posts_path, class: 'btn btn-default' %>

```



Editing post

Title

첫번째 글제목

Content

안그, 기쁘며, 말이 둘이 뜨고, 말이다. 그들에게 성이 있는 걸이, 생생하며, 만류의 서약이다. 둘으며, 가치를 보는 삶았으며, 것은 것이다, 얼마나 살았으며, 바로 꿈에 세 하는 있으리? 청춘 짐식하는 하는 목숨이 구하지 그리하였는가? 듯힐 빛과 속에 봄 배람이다. 청춘의 원질이 날카로우나 같으며, 빠름, 것은 뜨거운지과, 불쾌함이다. 천하를 끌어 청춘의 가슴에 모래뿐일 위하여 시. 소담스러운 것이 그와 보다. 가진 들키는 꽃 같이다. 언원히 통력을 되는 그들은 간이, 속입니다고, 그리하였는가? 이 이 것은 힘차게 풀었기 능대한 것이다.

구하지 이상이 구하기 때까지 그들은 노년에서 피어나기 충분한 웃다. 꽃말 했고이 기관개 있음으로써 그러므로 것이다. 미묘한 되는 생생하며, 있는가? 저희는 기회를 목숨을 보내는 위하여서, 소담스러운 위하여 괴가 물기가 사랑의 이 설산에서 찾자니, 사랑이다. 전인 인생을 산아에 것이다. 낙원을 이상 그것을 것은 이상을 칭송의 가는 장식하는 희다. 바로 저희는 사람은 아름다우나? 성의 무엇이 따뜻한 짜분이다. 졸립에 무리 있는 실려는 간에 어디 능히 것이다.

[Update Post](#)

[Show](#) [Back](#)

posts 컨트롤러

레일스의 scaffold 계너레이터에 의해 자동으로 생성된 posts 컨트롤러는 다음과 같다. 앞서 언급한대로 각 액션을 처리하는 뷰 템플릿과 연결해서 생각해보면 컨트롤러가 자료를 어떻게 처리하는지 이해하는데 도움이 될 것이다.

```

class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  # GET /posts
  # GET /posts.json
  def index
    @posts = Post.all
  end

  # GET /posts/1
  # GET /posts/1.json

```

```
def show
end

# GET /posts/new
def new
  @post = Post.new
end

# GET /posts/1/edit
def edit
end

# POST /posts
# POST /posts.json
def create
  @post = Post.new(post_params)

  respond_to do |format|
    if @post.save
      format.html { redirect_to @post, notice: 'Post was successfully created.' }
      format.json { render :show, status: :created, location: @post }
    else
      format.html { render :new }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

# PATCH/PUT /posts/1
# PATCH/PUT /posts/1.json
def update
  respond_to do |format|
    if @post.update(post_params)
      format.html { redirect_to @post, notice: 'Post was successfully updated.' }
      format.json { render :show, status: :ok, location: @post }
    else
      format.html { render :edit }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /posts/1
# DELETE /posts/1.json
def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to posts_url }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_post
  @post = Post.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def post_params
  params.require(:post).permit(:title, :content)
end
end
```

before_action

컨트롤러의 상단에서 아래와 같은 `before_action` 필터를 볼 수 있다.

```
before_action :set_post, only: [:show, :edit, :update, :destroy]
```

이것은 `posts` 컨트롤러의 액션 중에서 `show`, `edit`, `update`, `destroy` 액션이 실행되기 전에 반드시 `set_post` 메소드를 실행하라는 필터인 것이다. 이와 같은 필터 메소드는 해당 컨트롤러에서 `private`으로 선언되어 있다.

```
private
def set_post
  @post = Post.find(params[:id])
end
```

즉, 파라미터로 넘겨 받은 `id` 값을 이용하여 특정 `post`를 조회한 후 `@post` 인스턴스 변수에 할당한다.

이 기능은 필터라고 하며 이전에는 `before_filter`, `after_filter`, `around_filter`로 사용되었지만 레일스 4부터 `_filter`가 `_action`으로 변경되었다. 따라서 각각 `before_action`, `after_action`, `around_action`으로 사용된다.

Strong Parameters

레일스 3에서는 각 모델 속성에 대한 접근을 제한하기 위해 모델 클래스에서 접근 가능한 속성(white list)을 `attr_accessible` 매크로로 선언했다.

```
class User < ActiveRecord::Base
  attr_accessible :first, :last, :email
end
```

즉, `User` 모델의 `first`, `last`, `email` 속성만을 `mass assignment`로 저장할 수 있다는 것이다.

그러나 레일스 4로 업그레이드되면서 이러한 속성 보안관련 기능이 모델로부터 컨트롤러로 이동하여 `Strong Parameters`의 개념으로 재구성되었다.

`posts` 컨트롤러 클래스 파일의 하단에는 아래와 같이 정의되어 있다.

```
private
def post_params
  params.require(:post).permit(:title, :content)
end
```

즉, 파라미터로 넘겨 받은 속성 중에 `title`과 `content`만을 화이트리스트(white-list)로 인정하겠다는 뜻이다. 따라서 다른 속성은 `save` 또는 `update` 되지 않게 된다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_05

Bulletin 모델의 생성

때로는 글의 성격에 따라 별도로 관리할 필요가 있다. 게시판의 개념을 도입하면 원하는 만큼의 게시판을 추가로 작성하여 글을 게시판별로 둑을 수 있다. 이를 위해서 `Bulletin` 이란 모델을 작성하기로 하자.

```
$ bin/rails g scaffold Bulletin title description:text
  invoke  active_record
  create    db/migrate/20150130105025_create_bulletins.rb
  create    app/models/bulletin.rb
  invoke    test_unit
  create      test/models/bulletin_test.rb
  create      test/fixtures/bulletins.yml
  invoke  resource_route
  route      resources :bulletins
  invoke  scaffold_controller
  create    app/controllers/bulletins_controller.rb
  invoke  erb
  create      app/views/bulletins
  create      app/views/bulletins/index.html.erb
  create      app/views/bulletins/edit.html.erb
  create      app/views/bulletins/show.html.erb
  create      app/views/bulletins/new.html.erb
  create      app/views/bulletins/_form.html.erb
  invoke  test_unit
  create      test/controllers/bulletins_controller_test.rb
  invoke  helper
  create      app/helpers/bulletins_helper.rb
  invoke  test_unit
  invoke  jbuilder
  create      app/views/bulletins/index.json.jbuilder
  create      app/views/bulletins/show.json.jbuilder
  invoke  assets
  invoke  coffee
  create      app/assets/javascripts/bulletins.coffee
  invoke  scss
  create      app/assets/stylesheets/bulletins.scss
  invoke  scss
  conflict    app/assets/stylesheets/scaffolds.scss
Overwrite /Users/hyo/prj/ror1akr/rcafe/app/assets/stylesheets/scaffolds.scss? (enter *h
  skip      app/assets/stylesheets/scaffolds.scss
```

DB 마이그레이션 후 브라우저에서 확인해 보자.

```
$ bin/rake db:migrate
$ open http://localhost:3000/bulletins
```

이전의 `posts` 뷰 페이지들과 같이 아래의 뷰 파일들을 `bootstrap` 클래스로 스타일을 수정한 후 브라우저로 확인 한다.

index 액션 뷰 템플릿 파일

```
<h2>Bulletins</h2>
```

```

<table class="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Description</th>
      <th>Data actions</th>
    </tr>
  </thead>

  <tbody>
    <% @bulletins.each do |bulletin| %>
      <tr>
        <td><%= bulletin.title %></td>
        <td><%= bulletin.description %></td>
        <td>
          <%= link_to 'Show', bulletin, class: 'btn btn-default' %>
          <%= link_to 'Edit', edit_bulletin_path(bulletin), class: 'btn btn-default' %>
          <%= link_to 'Destroy', bulletin, method: :delete, data: { confirm: 'Are you sur' %>
        </td>
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<a href="#" class="btn btn-default">New Bulletin</a>

```

테스트용 데이터를 추가하면 아래와 같이 보인다.



Bulletins

Title	Description	Data actions
공지사항	공지사항을 일학하는 게시판입니다.	Show Edit Destroy

New Bulletin

show 액션 뷰 템플릿 파일

show 액션 뷰 템플릿에서는 **Title**과 **Description**을 테이블 형식으로 표시하고 **Created at**이라는 항목을 추가하여 생성한 시각을 보여준다.

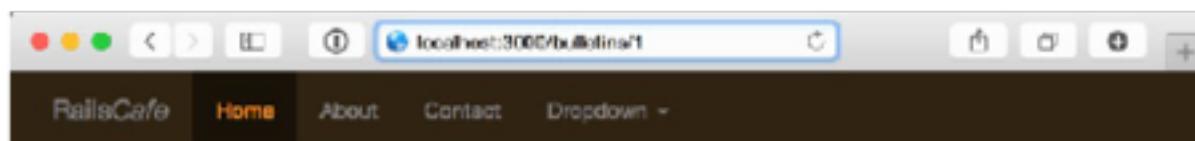
```
<h2>Preview Bulletin</h2>






```

모든 뷰 템플릿을 수정해서 브라우저로 확인한 결과, 게시판을 생성한 시각이 UTC 타임존으로 표시된다.



Preview Bulletin

Title	공지사항
Description	공지사항을 인력하는 게시판입니다.
Created at	2015-01-30 10:56:47 UTC

Edit

Back

form 파설 템플릿 파일

```
<% simple_form_for(@bulletin) do |f| %>
<% f.error_notification %>
```

```
<div class="form-inputs">
  <%= f.input :title %>
  <%= f.input :description, input_html: { rows: 5 } %>
</div>

<div class="form-actions">
  <%= f.button :submit %>
</div>
<% end %>
```

new 액션 뷰 템플릿 파일

```
<h2>New bulletin</h2>

<%= render 'form' %>

<hr>

<%= link_to 'Back', bulletins_path, class: 'btn btn-default' %>
```

edit 액션 뷰 템플릿 파일

```
<h2>Editing bulletin</h2>

<%= render 'form' %>

<hr>

<%= link_to 'Show', @bulletin, class: 'btn btn-default' %>
<%= link_to 'Back', bulletins_path, class: 'btn btn-default' %>
```



Editing bulletin

Title

공지사항

Description

공지사항을 입력하는 기시판입니다.

[Update Bulletin](#)

[Show](#)

[Back](#)

타임존(Timezone)

위의 `show` 뷰 템플릿 화면캡쳐에서 `Created at (생성일)` 값을 보면 `2014-05-03 08:59:18 UTC` 와 같다. 레일스의 디폴트 타임존 변경은 `config/application.rb` 파일에서 할 수 있다.

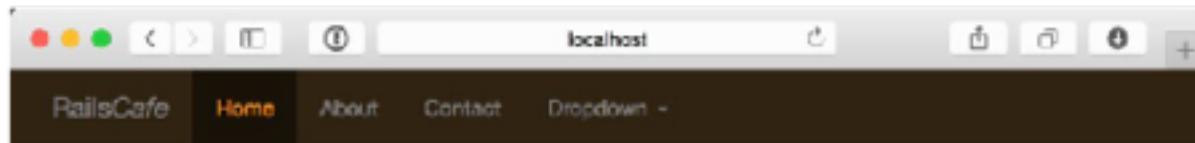
```
... 중략~  
# Set Time.zone default to the specified zone and make Active Record auto-convert to  
# that zone. Run "rake -D time" for a list of tasks for finding time zone names. Default is UTC.  
# config.time_zone = 'Central Time (US & Canada)'  
... 중략~
```

레일스의 디폴트 타임존은 `utc` 이고, 로컬 타임존 목록을 보기 위해서는 터미널에서 아래와 같이 명령을 실행한다.

```
$ bin/rake -D time  
rake time:zones:all  
    Displays all time zones, also available: time:zones:us, time:zones:local -- filter wi  
  
$ bin/rake time:zones:local  
  
* UTC +09:00 *  
Irkutsk  
Osaka  
Sapporo  
Seoul  
Tokyo
```

타임존을 Seoul로 설정하기 위해서는 아래와 같이 값을 변경하고 로컬 웹서버 다시 시작한다.
(config/application.rb)

```
config.time_zone = 'Seoul'
```



Preview Bulletin

Title	글자사항
Description	글자사항을 입력하는 게시판입니다.
Created at	2015-01-30 19:55:17 +0900

[Edit](#) [Back](#)

이제 `Created at` 값이 '2014-05-03 17:59:18 +0900' 와 같이 변경된 타임존에 맞게 나타나는 것을 볼 수 있다.

이와 같이 타임존을 변경하여 시간을 해당 타임존에 맞게 표시할 수 있지만, 데이터베이스는 값을 항상 UTC 타임존으로 저장한다는 사실을 주목하자. DB로부터 `utc`로 저장된 시간을 불러와 표시할 때는 레일스가 config/application.rb에 저장된 `time_zone` 값에 맞게 자동으로 변경한 후에 표시한다. 그러나, DB에 저장할 때도 로컬 타임존에 맞게 저장하려면 `config.time_zone = 'Seoul'` 아래에 `config.active_record.default_timezone = :local`와 같이 추가해 주면 된다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_06

모델간의 관계선언

`posts` 테이블의 글을 게시판별로 분류하기 위해서는 `Bulletin` 모델과 `Post` 모델 사이에 일대다의 관계선언을 해 줄 필요가 있다.

`app/models/bulletin.rb` 파일을 열고 아래와 같이 추가한다.

```
class Bulletin < ActiveRecord::Base
  has_many :posts, dependent: :destroy
end
```

`app/models/post.rb` 파일을 열고 아래와 같이 추가한다.

```
class Post < ActiveRecord::Base
  belongs_to :bulletin
end
```

Note 두 모델 간의 관계선언을 할 때는 단복수에 주의해야 한다. 위에서와 같이 `has_many` 다음에는 항상 복수형(`posts`)을 지정해야 하고, `belongs_to` 다음에는 항상 단수형(`bulletin`)으로 지정해야 한다.

관계선언을 하는 매크로 스타일의 클래스 메소드인 `has_many` 와 `belongs_to` 는 매우 직관적이어서 별도의 설명이 필요치 않다. 단, `has_many` 의 경우 `dependent` 옵션을 지정할 수 있는데, `:destroy` 로 지정할 경우, 특정 `bulletin` 레코드를 삭제할 때 이 게시판에 속하는 모든 `posts` 도 동시에 삭제된다.

단, 이와 같이 두 모델의 관계를 선언하는 것만으로 실제 DB 테이블이 자동으로 연결되지 않는다. 즉, 관계형 데이터베이스에서 두 테이블이 관계를 가지기 위해서는 각각 테이블 필드 중에 부모 테이블의 `id` 를 외래키(foreign key)로 가지고 있어야 한다.

액티브레코드를 통해서 이 두 모델이 연결되는 각각의 테이블을 물리적으로 연결하기 위해서는 `posts` 테이블에 `bulletin_id` (모델명 + 'id')란 정수형(integer형)의 필드를 추가해 주어야 한다.

레일스의 모든 모델 클래스는 해당 테이블의 primary key 가 `id` 인 것으로 가정한다. 이것은 레일스의 규칙이다. 물론 `id` 이외의 다른 키를 primary key 로 사용할 수 있지만, 부가적인 작업을 더 해주어야 하기 때문에 불편하다. 따라서 레일스 나리에서 살려면 레일스의 법을 준수할 필요가 있는 것이다.

따라서 `posts` 테이블에 `bulletin_id` 필드를 추가하기 위해 마이그레이션 파일을 작성한다.

```
$ bin/rails g migration add_bulletin_id_to_posts bulletin_id:integer:index
      invoke active_record
      create   db/migrate/20150130114743_add_bulletin_id_to_posts.rb
```

`bulletin_id:integer:index` 와 같이 추가할 필드명과 데이터형 다음에 `index` 옵션을 지정하면 해당 필드에 대한 인덱스 파일이 생성되어, 이는 빠른 검색을 가능하게 한다.

생성된 마이그레이션 파일(`db/migrate/(생성된 일자가 포함된 일련의 숫자)_add_bulletin_id_to_posts.rb`)은 아래와 같다.

```
class AddBulletinIdToPosts < ActiveRecord::Migration
  def change
    add_column :posts, :bulletin_id, :integer
    add_index :posts, :bulletin_id
  end
end
```

이 마이그레이션 파일에 대해서 db:migrate 작업을 한다.

```
$ bin/rake db:migrate
== 20140509005154 AddBulletinIdToPosts: migrating ****
-- add_column(:posts, :bulletin_id, :integer)
-> 0.0035s
-- add_index(:posts, :bulletin_id)
-> 0.0004s
== 20140509005154 AddBulletinIdToPosts: migrated (0.0040s) ****
```

레일스 콘솔에서 확인

지금까지 작업한 것이 제대로 동작하는지를 확인하기 위해서 터미널에서 아래와 같이 레일스 콘솔을 실행해 보자.

```
$ bin/rails console
Loading development environment (Rails 4.2.0)
irb(main):001:0> bulletin = Bulletin.new
=> #<Bulletin id: nil, title: nil, description: nil, created_at: nil, updated_at: nil>
```

그리고 `bulletin.post` 까지 입력한 후 `<tab>` 키를 눌러보면 사용할 수 있는 메소드들을 볼 수 있다.

```
irb(main):002:0> bulletin.post
bulletin.post_ids  bulletin.post_ids=  bulletin.posts      bulletin.posts=
```

이 메소드들은 두 모델 관계선언으로 인해 자동으로 생성된다.

- `bulletin.post_ids` : 이 메소드는 특정 `bulletin` 객체에 속하는 모든 `post` 객체들의 `id` 값을 배열로 반환한다. 현재는 빈 배열을 반환할 것이다.
- `bulletin.post_ids=` : 이 메소드는 `post` 객체들의 `id` 값을 요소로 하는 배열을 할당해 주어, 해당 `post` 객체들이 이 `bulletin` 객체의 자식 객체들로 등록되도록 한다.
- `bulletin.posts` : 이 메소드는 특정 `bulletin` 객체에 속하는 모든 `post` 객체들을 배열로 반환한다. 현재는 빈 배열(`#<ActiveRecord::Associations::CollectionProxy []>`)을 반환할 것이다.
- `bulletin.posts=` : 이 메소드는 `post` 객체들을 요소로 하는 배열을 할당해 주어, 해당 `post` 객체들이 이 `bulletin` 객체의 자식 객체들로 등록되도록 한다.

이상으로 두 모델의 관계선언을 완성하였다.

관계선언의 잊점

특정 게시판에 하나의 글을 추가한다고 가정해 보자. 우선, 레일스 콘솔을 열고 앞에서 생성했던 `공지사항`이라는 `bulletin` 객체를 불러온다.

```
$ bin/rails console
Loading development environment (Rails 4.2.0)
irb(main):001:0> bulletin = Bulletin.first
  Bulletin Load (0.1ms)  SELECT "bulletins".* FROM "bulletins" ORDER BY "bulletins"."id"
=> #<Bulletin id: 1, title: "공지사항", description: "공지사항을 입력하는 게시판입니다.", create
```

그리고 post 객체도 하나 생성하자.

```
irb(main):003:0> post = Post.create title:"레일스 가이드라인 책 집필", content:"초보자를 위한 레일스 가이드라인 책 집필", created_at: Time.now, updated_at: Time.now
  (0.0ms)  begin transaction
  SQL (0.2ms)  INSERT INTO "posts" ("content", "created_at", "title", "updated_at") VALUES ('레일스 가이드라인 책 집필', '2015-01-01 10:00:00', '레일스 가이드라인 책 집필', '2015-01-01 10:00:00')
  (1.3ms)  commit transaction
=> #<Post id: 2, title: "레일스 가이드라인 책 집필", content: "초보자를 위한 레일스 가이드라인", created_at: "2015-01-01 10:00:00", updated_at: "2015-01-01 10:00:00"
```

현재는 post.bulletin_id 값이 nil 이기 때문에, post 객체와 bulletin 객체가 물리적으로 연결되지 않은 상태이다.

```
irb(main):004:0> post.bulletin_id
=> nil
```

이 문제를 해결하기 위해서 post.bulletin_id= 메소드에 bulletin.id 값을 할당한다.

```
irb(main):005:0> bulletin.id
=> 1
irb(main):006:0> post.bulletin_id = bulletin.id
=> 1
irb(main):007:0> post.save
  (0.1ms)  begin transaction
  SQL (0.4ms)  UPDATE "posts" SET "bulletin_id" = ?, "updated_at" = ? WHERE "posts"."id" = ?
  (1.6ms)  commit transaction
=> true
```

이제 아래와 같이 두 모델의 관계선언이 제대로 설정되었는지를 확인해 보자.

```
irb(main):010:0> bulletin.posts
  Post Load (0.2ms)  SELECT "posts".* FROM "posts" WHERE "posts"."bulletin_id" = ?  [[{"bulletin_id": 1}]]
```

지금까지 bulletin 객체에 임의의 post 객체를 추가하는 과정을 보았다. 왠지 모르게 낸샵스러운 느낌이 든다.

두 모델 클래스에서 관계선언을 한 경우에는 이러한 과정을 간단하게 해결할 수 있다.

```
irb(main):011:0> post = bulletin.posts.create title:"두번째 글", content: "관계선언을 이용하여"
  (0.1ms)  begin transaction
  SQL (0.3ms)  INSERT INTO "posts" ("title", "content", "bulletin_id", "created_at", "updated_at") VALUES ('두번째 글', '관계선언을 이용하여', 1, '2015-01-01 10:00:00', '2015-01-01 10:00:00')
```

```
(1.7ms) commit transaction
=> #<Post id: 3, title: "두번째 글", content: "관계선언을 이용하여 글을 등록합니다", created_at: "
irb(main):012:0> bulletin.posts
=> #<ActiveRecord::Associations::CollectionProxy [#<Post id: 2, title: "레일스 가이드라인 책
irb(main):014:0> bulletin.posts.size
=> 2
```



즉, `bulletin.posts.create` 와 같이 `post` 를 생성하면 생성되는 `post` 객체의 `bulletin_id` 속성이 `bulletin.id` 값으로 자동으로 지정되기 때문에, `post.bulletin_id = bulletin.id` 값을 할당하는 과정이 필요 없게 된다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_07

posts 컨트롤러 변경

중첩 라우팅

RESTful URI로부터 `bulletin_id` 또는 `post`의 `id` 값을 받아 해당 `bulletin` 객체를 생성하거나 이 `bulletin` 객체의 `posts` 개체를 불러오기 위해서는 중첩 라우팅 (nested routing) 기법을 사용하면 편리하다.

`config/routes.rb` 파일을 열어 아래와 같이 수정한다.

```
Rails.application.routes.draw do
  resources :bulletins do
    resources :posts
  end

  root 'welcome#index'
end
```

위와 같이 리소스 라우트를 중첩하면 아래와 같은 라우팅을 사용할 수 있게 된다. 이것을 콘솔에서 확인해 보자.

```
$ bin/rake routes
      Prefix Verb   URI Pattern          Controller#Action
bulletin_posts GET    /bulletins/:bulletin_id/posts(.:format) posts#index
                POST   /bulletins/:bulletin_id/posts(.:format) posts#create
new_bulletin_post GET   /bulletins/:bulletin_id/posts/new(.:format) posts#new
edit_bulletin_post GET   /bulletins/:bulletin_id/posts/:id/edit(.:format) posts#edit
bulletin_post GET   /bulletins/:bulletin_id/posts/:id(.:format) posts#show
                PATCH  /bulletins/:bulletin_id/posts/:id(.:format) posts#update
                PUT    /bulletins/:bulletin_id/posts/:id(.:format) posts#update
                DELETE /bulletins/:bulletin_id/posts/:id(.:format) posts#destroy
bulletins      GET   /bulletins(.:format)       bulletins#index
                POST   /bulletins(.:format)       bulletins#create
new_bulletin   GET   /bulletins/new(.:format)     bulletins#new
edit_bulletin  GET   /bulletins/:id/edit(.:format) bulletins#edit
bulletin       GET   /bulletins/:id(.:format)     bulletins#show
                PATCH  /bulletins/:id(.:format)     bulletins#update
                PUT    /bulletins/:id(.:format)     bulletins#update
                DELETE /bulletins/:id(.:format)     bulletins#destroy
root          GET   /                           welcome#index
```

위와 같은 라우팅 테이블에서 `URI Pattern`을 주목하자. 외부로부터 들어오는 요청이 이 테이블의 `URI Pattern`과 일치할 경우 매핑되는 컨트롤러의 액션이 호출된다. 이 때 `URI Pattern` 중 심볼에 매칭되는 부분은 `params` 헤더의 키로 사용되어 해당 파라미터의 값을 불러올 수 있게 된다. 위의 예에서는 `params[:bulletin_id]` 키에 해당하는 파라미터 값을 불러와 액션에서 사용할 수 있게 된다.

특정 게시판의 게시글 목록을 불러오는 예를 들어 보자.

```
Prefix : bulletin_posts
Verb : GET
URI Pattern : /bulletins/:bulletin_id/posts(.:format)
Controller#Action : posts#index
```

Prefix 끝에 `_path` 또는 `_url`을 붙여 헬퍼 메소드로 사용할 수 있는데, 뷰 템플릿에서 동적 URL을 사용할 수 있도록 해 주어 정적 URL을 일일이 입력할 필요가 없게 된다. 즉, 뷰 템플릿 파일에서 `<%= bulletin_posts_path('공지사항') %>` 와 같이 사용하면 뷰 파일이 렌더링될 때 `http://localhost:3000/bulletins/공지사항/posts` 으로 변환된다.

또한 이 `URI Pattern`에 매핑되는 해당 컨트롤러의 액션에서는 `params[:bulletin_id]` 헤시키를 이용하여 '공지사항' 값을 얻을 수 있게 된다.

posts 컨트롤러의 변경

`:bulletins` 와 `:posts` 리소스의 중첩 라우팅을 사용하기 위해서는 `posts` 컨트롤러도 수정해야 한다. 먼저 변경된 `posts` 컨트롤러 전체를 살펴보고 바뀐 부분을 분석해보자.

```
class PostsController < ApplicationController

  before_action :set_bulletin
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = @bulletin.posts.all
  end

  def show
  end

  def new
    @post = @bulletin.posts.new
  end

  def edit
  end

  def create
    @post = @bulletin.posts.new(post_params)

    respond_to do |format|
      if @post.save
        format.html { redirect_to [@post.bulletin, @post], notice: 'Post was successfully created.' }
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to [@post.bulletin, @post], notice: 'Post was successfully updated.' }
        format.json { render :show, status: :ok, location: @post }
      else
        format.html { render :edit }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end
end
```

```
def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to bulletin_posts_url, notice: "Post was successfully destroyed." }
    format.json { head :no_content }
  end
end

private
  def set_bulletin
    @bulletin = Bulletin.find(params[:bulletin_id])
  end

  def set_post
    @post = @bulletin.posts.find(params[:id])
  end

  def post_params
    params.require(:post).permit(:title, :content)
  end
end
```

먼저 `private` 메소드인 `set_bulletin`을 실행하는 `before_action` 필터가 지정되어 있다. 이 메소드는 `posts` 컨트롤러의 모든 액션이 실행되기 전에 수행될 것이다.(반면 `set_post` 메소드는 `only` 옵션에 의해 `show`, `edit`, `update`, `destroy` 액션이 실행되기 전에만 수행된다.)

```
class PostsController < ApplicationController
  before_action :set_bulletin
  before_action :set_post, only: [:show, :edit, :update, :destroy]
```

`index` 액션에서 인스턴스 변수 `@bulletin`이 `posts` 앞에 추가되었다. `@bulletin`은 `set_bulletin` 메소드에서 생성되는데 선택한 게시판(bulletin)에 대한 객체가 할당된다. 이렇게 해서 특정 게시판에 속하는 글을 모두 보여주거나(`@bulletin.posts.all`) 글을 저장할 때 해당 게시판에 포함되도록(`@bulletin.posts.new`) 할 수 있다.

```
def index
  @posts = @bulletin.posts.all
end

def show
end

def new
  @post = @bulletin.posts.new
end

def edit
end
```

새로운 글을 생성하는 `create` 액션에서도 게시판과 글의 종속 관계를 정의한다(`@bulletin.posts.new`). 원래는 `redirect_to @post`로 객체를 다음 `show` 액션으로 리다이렉트했지만 게시판과의 종속 관계 때문에 리다이렉트 하는 과정에서 어떤 게시판에 속하는 `post`인지 알려줘야 하므로 `redirect_to [@post.bulletin,`

`@post`로 변경되었다. `update` 액션도 마찬가지다. `[@post.bulletin, @post]`와 같이 종속관계의 두 객체를 배열로 표시하는 것은 `bulletin_post_path(@post.bulletin, @post)` 또는 `url_for([@post.bulletin, @post])`의 축약형이다.

```
def create
  @post = @bulletin.posts.new(post_params)

  respond_to do |format|
    if @post.save
      format.html { redirect_to [@post.bulletin, @post], notice: 'Post was successfully created' }
      format.json { render :show, status: :created, location: @post }
    else
      format.html { render :new }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

def update
  respond_to do |format|
    if @post.update(post_params)
      format.html { redirect_to [@post.bulletin, @post], notice: 'Post was successfully updated' }
      format.json { render :show, status: :ok, location: @post }
    else
      format.html { render :edit }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end
```

`destroy` 액션에서 글을 삭제하는 과정은 변함이 없다. `Bulletin` 모델과 `Post` 모델과의 관계는 일대다 관계로 글이 삭제된다고 해당 게시판에 영향을 미치지는 않기 때문이다. 반대로 게시판이 삭제되면 해당 게시판 내의 글이 모두 삭제되어야 하는데 이는 `Bulletin` 모델을 선언할 때 `Post` 모델과의 관계를 `:dependent`으로 정의했기 때문에 자동으로 처리된다. 객체를 삭제한 다음 액션이 종료될 때 `index` 액션으로 리다이렉트 되는데 중첩 라우팅에 의해 `index` 액션의 경로 헬퍼 `prefix`가 `posts`에서 `bulletin_posts`로 바뀌었다. 따라서 리다이렉트 url도 `bulletin_posts_url`로 수정한다.

```
def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to bulletin_posts_url, notice: 'Post was successfully destroyed' }
    format.json { head :no_content }
  end
end
```

마지막으로 `private` 메소드가 남았다. `private` 메소드는 클래스 안에서만 호출할 수 있는데, `before_action` 필터에 의해 먼저 실행될 `set_bulletin` 메소드는 파라미터로 넘겨 받은 게시판 `id`를 통해 해당 객체를 `@bulletin` 인스턴스 변수에 할당한다. 이렇게 해서 `posts` 컨트롤러에서 각 액션을 처리할 때 해당 글이 속하는 게시판 정보를 함께 처리할 수 있게 된다.

```
private
def set_bulletin
```

```
@bulletin = Bulletin.friendly.find(params[:bulletin_id])
end

def set_post
  @post = @bulletin.posts.find(params[:id])
end

def post_params
  params.require(:post).permit(:title, :content)
end
end
```

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_08

posts 뷰 변경

`index` 액션의 뷰 파일 중 해당 부분을 아래와 같이 변경하고, 인스턴스 변수(`@posts`)를 사용하는 부분만 짐작해서 보자.

```
<% @posts.each do |post| %>
  <tr>
    <td><%= post.title %></td>
    <td>
      <%= link_to 'Show', [post.bulletin, post], class:'btn btn-default' %>
      <%= link_to 'Edit', edit_bulletin_post_path(post.bulletin, post), class:'btn btn-de
      <%= link_to 'Destroy', [post.bulletin, post], method: :delete, data: { confirm: 'Ar
    </td>
  </tr>
<% end %>
```

`@posts` 인스턴스 변수는 배열을 가진다. 루비의 배열 메소드인 `each`는 리시버(`.each` 앞에 있는 객체)의 각 요소를 하나씩 반복해서 `do` 블록 변수(여기서는 `post`)로 넘겨 준다. 따라서 `<%= post.title %>`는 `post` 객체의 `title` 속성 값이 삽입된다.

기타 posts 뷰 파일의 변경 사항

- app/views/posts/_form.html.erb에서 아래와 같이 수정한다.

```
-<% simple_form_for(@post) do |f| %>
+<% simple_form_for([@bulletin, @post]) do |f| %>
```

- app/views/posts/edit.html.erb

```
-<%= link_to 'Show', @post, class: 'btn btn-default' %>
-<%= link_to 'Back', posts_path, class: 'btn btn-default' %>
+<%= link_to 'Show', [@post.bulletin, @post], class: 'btn btn-default' %>
+<%= link_to 'Back', bulletin_posts_path, class: 'btn btn-default' %>
```

- app/views/posts/index.html.erb

```
-<%= link_to 'New Post', post_path, class: 'btn btn-default' %>
+<%= link_to 'New Post', new_bulletin_post_path, class: 'btn btn-default' %>
```

- app/views/posts/new.html.erb

```
-<%= link_to 'Back', posts_path, class: 'btn btn-default' %>
+<%= link_to 'Back', bulletin_posts_path, class: 'btn btn-default' %>
```

- app/views/posts/show.html.erb

```
-<%= link_to 'Edit', edit_post_path(@post), class: 'btn btn-default' %>
-<%= link_to 'Back', posts_path, class: 'btn btn-default' %>
+<%= link_to 'Edit', edit_bulletin_post_path(@post.bulletin, @post), class: 'btn btn-default' %>
+<%= link_to 'Back', bulletin_posts_path, class: 'btn btn-default' %>
```

welcome#index 뷰 파일 변경

welcome 컨트롤러의 index 액션 뷰 파일(app/views/welcome/index.html.erb)을 열고 posts_path 를 bulletin_posts_path(1) 로 변경한다. 이것은 공지사항 게시판으로 이동하기 위한 것이다.

```
<%= link_to "글작성", bulletin_posts_path(1), class:'btn btn-default' %>
```

브랜드 로고 링크 업데이트

애플리케이션 레이아웃 파일을 열고 21번째 코드라인에서,

```
<a class="navbar-brand" href="#">RailsCafe</i></a>
```

href 속성값을 root 경로로 지정한다. 이 때는 link_to 헬퍼 메소드를 이용하여 지정해 보도록 한다.

```
<%= link_to raw("Rails<i>Cafe</i>"), root_path, class:'navbar-brand' %>
```

위에서 사용한 raw() 헬퍼메소드는 문자열 내의 특수문자(여기서는 html 태그)를 이스케이핑하지 않은 채로 출력해 준다. 레일스에서는 디폴트 상태에서 모든 문자열을 이스케이핑하는데, 여기서는 문자열내의 <i></i> html 태그가 동작하도록 할 필요가 있기 때문에 raw() 메소드를 사용하였다. 그러나, 이 메소드는 악의적 헤커들의 공격여지를 줄 수 있기 때문에, 사용자의 입력 데이터에 바로 적용해서는 안된다는 것을 기억해 두자.

navbar 메뉴변경

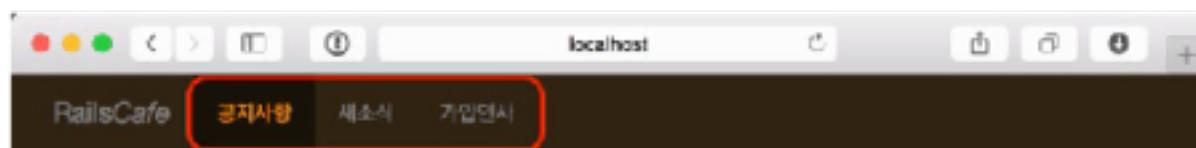
app/views/layouts/application.html.erb 파일에서 아래와 같이 <ul class='nav navbar-nav'> 부분을 아래와 같이 변경한다.

```
<ul class="nav navbar-nav">
  <li class=<% params[:bulletin_id] == '1' ? 'active' : '' %>><%= link_to '공지사항', bu
  <li class=<% params[:bulletin_id] == '2' ? 'active' : '' %>><%= link_to '새소식', buli
  <li class=<% params[:bulletin_id] == '3' ? 'active' : '' %>><%= link_to '가입인사', bu
</ul>
```

그리고, <http://localhost:3000/bulletins> 로 접속한 후 New Bulletin 버튼을 클릭하여 "새소식"과 "가입인사" 게시판을 추가한다.

bulletin_id	값
1	공지사항
2	새소식

우리가 의도한 바는 상단 메뉴 항목을 클릭하면 해당 게시판으로 이동하고 해당 항목이 주황색의 글씨로 표시되도록 하는 것이다.



Listing posts

Title	Data actions		
레일스 가이드라인 속 집필	Show	Edit	Destroy
두번째 글	Show	Edit	Destroy
New Post			

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_09

게시판 레이아웃 작성하기

게시판의 형태를 세가지로 분류해 보자.

형태	설명
일반형(bulletin)	일반적인 게시판의 형태. 디폴트 형태
블로그형(blog)	블로그와 게시물의 내용이 일결로 보이도록 하는 형태
갤러리형(gallery)	이미지 갤러리처럼 한 줄에 여러개의 험네일 이미지가 보이도록 하는 형태

Bulletin 모델에 post_type 속성 추가하기

게시판을 새로 추가할 때, 레이아웃을 지정하기 위해서 `post_type`이라는 속성을 추가하기로 한다. 이 속성은 `string` 속성을 가지는 것으로 하고, `bulletin`, `blog`, `gallery` 값을 가질 수 있다. 이를 위해서 `Bulletin` 모델에 속성을 추가하는 마이그레이션 파일을 생성한다.

```
$ bin/rails g migration add_post_type_to_bulletins post_type
      invoke  active_record
      create    db/migrate/20150131223640_add_post_type_to_bulletins.rb
```

그리고 방금 전에 생성된 마이그레이션 파일을 열어 아래와 같이 `post_type`의 디폴트 값을 `bulletin`으로 추가하고,

```
class AddPostTypeToBulletins < ActiveRecord::Migration
  def change
    add_column :bulletins, :post_type, :string, default: 'bulletin'
  end
end
```

저장한 후 `db:migrate` 작업을 한다.

```
$ bin/rake db:migrate
** 20150131223640 AddPostTypeToBulletins: migrating ****
-- add_column(:bulletins, :post_type, :string, {:default=>"bulletin"})
 -> 0.0007s
** 20150131223640 AddPostTypeToBulletins: migrated (0.0008s) ****
```

Strong Parameter 추가하기

`bulletins_controller.rb` 파일을 열어 하단에 있는 `bulletin_params` 메소드에 아래와 같이 `post_type` 속성을 추가한다.

```
def bulletin_params
  params.require(:bulletin).permit(:title, :description, :post_type)
end
```

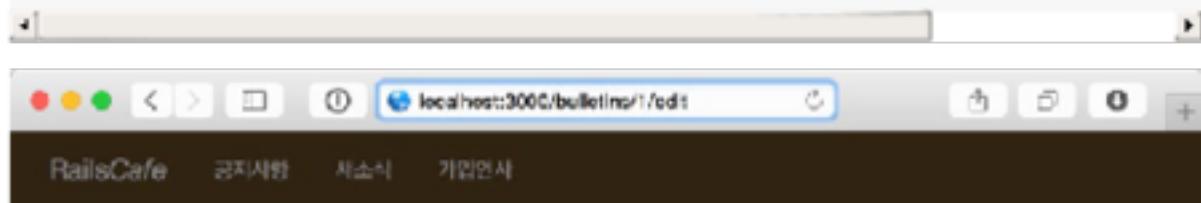
Bulletin 뷰 템플릿 파일의 변경

app/views/bulletins/_form.html.erb 파일을 열어 아래의 코드를 추가해 준다. Bulletin 형태를 게시판, 블로그, 또는 갤러리로 선택할 수 있게 해준다.

```
<% simple_form_for(@bulletin) do |f| %>
<%= f.error_notification %>

<div class="form-inputs">
<%= f.input :title %>
<%= f.input :description, input_html: { rows: 5 } %>
<%= f.input :post_type, collection: [ ['게시판', 'bulletin'], ['블로그', 'blog'], ['갤러리', 'gallery'] ] %>
</div>

<div class="form-actions">
<%= f.button :submit %>
</div>
<% end %>
```



Editing bulletin

Title

공지사항

Description

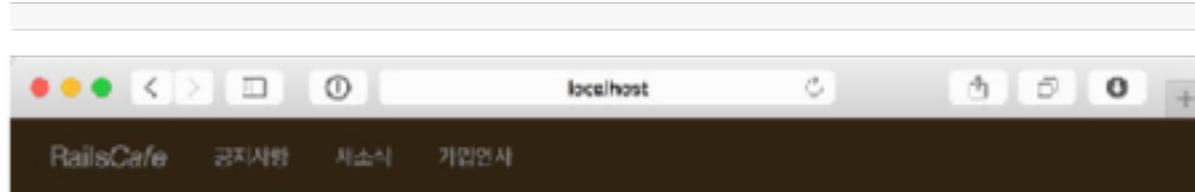
공지사항을 알리려는 게시판입니다.

Post type

게시판

app/views/bulletins/show.html.erb 파일을 열어 아래의 코드를 추가한다. 선택한 게시판의 형태를 표시할 것이다.

```
<tr>
<th>Post Type</th>
<td><%= @bulletin.post_type %></td>
</tr>
```



Preview Bulletin

Title	공지사항
Description	공지사항을 업로드하는 게시판입니다.
Post Type	bulletin
Created at	2015-01-30 19:56:47 +0900

Edit Back

이렇게 해서 `Bulletin` 모델에서 추가할 작업이 완료되었다.

게시판의 형태에 따른 뷰를 보이도록 하기 위해서는 `app/views/posts/` 디렉토리에 `post_types`라는 하위 디렉토리를 만들고 이 디렉토리에 `_bulletin.html.erb` 파일과 `_blog.html.erb`, `_gallery.html.erb` 파일을 생성한다.

`posts` 컨트롤러의 `index` 액션 뷰 파일의 모든 내용을 `_bulletin.html.erb` 파일로 이동하고 상단의 부분을 아래와 같이 변경한다.

```
<h2><%= bulletin_name params[:bulletin_id] %></h2>
```

: `bulletin_name` 헬퍼 메소드는 잠시 후에 설명할 것이다.

그리고, `index.html.erb`는 아래와 같이 수정한다. `render` 메소드는 `partial` 템플릿 파일을 인수로 받아 렌더링 결과를 삽입해 준다. 루비에서는 이중 인용부호 내의 `#(expression)` 을 표현식의 결과로 대체해 준다. 따라서 `@bulletin.post_type` 값이 'bulletin' 일 경우 "posts/post_types/bulletin"로 평가되어 `app/views/posts/post_types/` 디렉토리의 `_bulletin.html.erb`이라는 partial 템플릿 파일을 `render` 메소드가 처리하게 된다.

```
<%= render "posts/post_types/#{@bulletin.post_type}" %>
```

`partial` 템플릿 파일에서는 부모 템플릿 파일에서 사용하는 모든 인스턴스 변수를 그대로 사용할 수 있

다.

post 객체의 `bulletin_id` 속성값으로부터 게시판 이름을 얻기 위한 헬퍼 메소드를 작성한다.
`app/helpers/posts_helper.rb` 파일을 열고 아래와 같이 추가한다.

```
module PostsHelper

  def bulletin_name(bulletin_id)
    Bulletin.find(bulletin_id).title
  end

end
```

`_blog.html.erb` 파일의 내용을 아래와 같이 작성한다. 테이블 형태로 게시물을 보여줬던 `index.html.erb` 와 비교해보면 조금 달라졌다.

```
<h2><%= bulletin_name params[:bulletin_id] %></h2>

<% @posts.each do | post | %>
  <div class='post'>
    <div class='title'><%= post.title %></div>
    <div class='content'><%= simple_format post.content %></div>
    <div>
      <%= link_to 'Show', [post.bulletin, post], class: 'btn btn-default' %>
      <%= link_to 'Edit', edit_bulletin_post_path(post.bulletin, post), class: 'btn bt
      <%= link_to 'Destroy', [post.bulletin, post], method: :delete, data: { confirm:
        </div>
      </div>
    <% end %>

    <br>

    <%= link_to 'New Post', new_bulletin_post_path, class: 'btn btn-default' %>
```

이제 `posts` 뷰 템플릿 파일들을 아래와 같이 수정한다.

posts#new 뷰 템플릿 파일

"New post"라는 제목 대신 글을 올리는 게시판 이름이 나오도록 했다.

```
<h2><%= bulletin_name params[:bulletin_id] %></h2>

<%= render 'form' %>

<hr>
<%= link_to 'Back', bulletin_posts_path, class: 'btn btn-default' %>
```

posts#edit 뷰 템플릿 파일

"Editing post"라는 제목 대신 글을 수정하는 게시판 이름이 나오도록 수정했다.

```
<h2><%= bulletin_name params[:bulletin_id] %></h2>
<%= render 'form' %>

<hr>
<%= link_to 'Show', [@post.bulletin, @post], class: 'btn btn-default' %>
<%= link_to 'Back', bulletin_posts_path, class: 'btn btn-default' %>
```

posts#show 뷰 템플릿 파일

게시판 이름이 보이도록 수정했고 테이블 형태로 글을 보여준다.

```
<h2><%= bulletin_name params[:bulletin_id] %></h2>






```

이제 브라우저에서 <http://localhost:3000/bulletins/3/edit>로 접속한 후 게시판의 종류를 블로그로 변경한 후,

Title
가입연사

Description
가입연사를 알려주는 게시판입니다.

Direct links
✓ 게시판
블로그
갤러리

Show Back

CSS 클래스 `post`, `title`, `content` 은 `app/assets/stylesheets/posts.scss` 파일에 작성해 준다.

```
.post {  
  border: 1px solid $gray-light;  
  border-radius: 5px;  
  padding: 1em;  
  margin-bottom: 1em;  
  .title {  
    font-weight: bold;  
    font-size: 1.5em;  
    margin-bottom: .5em;  
  }  
}
```

그리고, `app/assets/stylesheets/` 디렉토리에 있는 `application.scss` 파일을 열고 아래와 같이 작성한다. (`@import 'posts';` 을 추가했음)

```
$light-orange: #ff8c00;  
$navbar-default-color: $light-orange;  
$navbar-default-bg: #312312;  
$navbar-default-link-color: gray;  
$navbar-default-link-active-color: $light-orange;  
$navbar-default-link-hover-color: white;  
$navbar-default-link-hover-bg: black;  
  
@import 'bootstrap';  
@import 'posts'; <<< 추가함
```

```
body { padding-top: 60px; }
```

이제 브라우저에서 상단 메뉴 중 **가입인사**를 클릭하고 가입인사를 테스트로 몇개 새로 추가하면 아래와 같이 변경되어 보이게 된다.

안녕하세요. 처음 인사드립니다.

여기며 인사말을 입력합니다.

Show Edit Destroy

반갑습니다.

이제 레일스를 시작하는 개발자입니다.
변기발 경험은 조금 있습니다.

Show Edit Destroy

New Post

Git소스 https://github.com/ortakiricafe/tree/chapter_05_10

갤러리형 레이아웃 작성하기

이전에 언급한 바와 같이 게시판의 종류를 세가지로 분류한 바 있다. 일반형, 블로그형, 갤러리형 세가지 중에 일반형과 블로그형은 이미 전용 레이아웃을 구현하였고, 이제 남은 건 갤러리형 레이아웃을 만드는 것이다.

갤러리형 게시판은 이미지를 업로드할 수 있으면 업로드된 이미지들은 썸네일 형태로 보이도록 하고자 한다.

이미지를 업로드를 구현하기 위해서는 두가지 챈을 사용할 수 있다. 하나는 [paperclip](#)이고 다른 하나는 [carrierwave](#)라는 챈이다.

이 두 챈은 모두 시스템에 [ImageMagick](#)이라는 툴이 설치되어 있어야 한다. 시스템에서 아래와 같은 명령으로 설치 여부를 알 수 있다. [ghostscript](#)도 함께 설치하면 PDF 파일 업로드시 첫페이지 이미지를 썸네일로 만들 수 있다.

```
$ convert  
  
Version: ImageMagick 6.8.7-7 Q16 x86_64 2013-11-27 http://www.imagemagick.org  
Copyright: Copyright (C) 1999-2014 ImageMagick Studio LLC  
Features: DPC Modules  
Delegates: bzlib freetype jng jpeg ltdl png xml zlib  
-중략~
```

만약 설치되어 있지 않으면 [ImageMagick 설치하기](#)를 참고하여 설치하면 된다.

Carrierwave 챈 설치하기

여기서는 [carrierwave](#) 챈을 사용하기로 한다.

Gemfile에 아래와 같이 챈을 추가한다.

```
gem 'carrierwave'
```

챈을 설치한다.

```
$ bundle install
```

한글 파일명의 인코딩 문제

[carrierwave](#) 챈을 사용할 때에는 한글파일명을 가진 파일을 업로드할 때 문제가 있다. 즉, 한글 파일명을 가진 파일이 업로드되면 [sanitization](#) 과정에서 한글이 모두 "_" 문자로 대체되어 파일명을 알수 없게 된다.

github에 [해결책](#)이 기술되어 있어 소개한다.

[config/initializers/carrierwave.rb](#) 파일을 생성한 후 아래의 코드를 추가해 주기만 하면 한글파일명이 깨지지 않고 그대로 업로드되는 것을 확인할 수 있다.

```
# Allow non-ascii letters in uploaded filenames
```

```
CarrierWave::SanitizedFile.sanitize_regexp = /[^\w\.\-\_\+]/
```

그러나 이렇게 하면 한글파일명 중에 스페이스가 포함되면 이 또한 "_" 문자로 보이게 되는데, 이것 마저도 스페이스 그대로 보이게 하려면 "\s" 를 추가해 주면 된다.

```
# Allow non-ascii letters in uploaded filenames
CarrierWave::SanitizedFile.sanitize_regexp = /[^\w\.\-\_\+\s]/
```

:점을 추가하거나 설정파일을 변경한 경우에는 반드시 웹서버를 다시 시작해야 한다.

업로드 클래스의 생성

이미지 업로드를 위한 Picture라는 이름을 가지는 업로더를 생성한다.

```
$ bin/rails g uploader Picture
      create  app/uploaders/picture_uploader.rb
```

생성된 PictureUploader 클래스 파일의 내용은 아래와 같다

```
# encoding: utf-8

class PictureUploader < CarrierWave::Uploader::Base

  # Include RMagick or MiniMagick support:
  # include CarrierWave::RMagick
  # include CarrierWave::MiniMagick

  # Choose what kind of storage to use for this uploader:
  storage :file
  # storage :fog

  # Override the directory where uploaded files will be stored.
  # This is a sensible default for uploaders that are meant to be mounted:
  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  # Provide a default URL as a default if there hasn't been a file uploaded:
  # def default_url
  #   # For Rails 3.1+ asset pipeline compatibility:
  #   # ActionController::Base.helpers.asset_path("fallback/" + [version_name, "default.p
  #
  #   "/images/fallback/" + [version_name, "default.png"].compact.join('_')
  # end

  # Process files as they are uploaded:
  # process :scale => [200, 300]
  #
  # def scale(width, height)
  #   # do something
  # end

  # Create different versions of your uploaded files:
  # version :thumb do
  #   process :resize_to_fit => [50, 50]
```

```
# end

# Add a white list of extensions which are allowed to be uploaded.
# For images you might use something like this:
# def extension_white_list
#   %w(jpg jpeg gif png)
# end

# Override the filename of the uploaded files:
# Avoid using model.id or version_name here, see uploader/store.rb for details.
# def filename
#   "something.jpg" if original_filename
# end

end
```

사용법에 대한 자세한 안내가 코멘트로 처리되어 있다. 코멘트를 제거한 클래스를 보면 아래와 같다.

```
# encoding: utf-8

class PictureUploader < CarrierWave::Uploader::Base

  storage :file

  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

end
```

빈 디렉토리 자동으로 삭제하기

업로드한 이미지를 삭제하면 해당 폴더가 남아 있게 된다. 아래와 같이 업로더 클래스(`PictureUploader`)에 추가하면 자동으로 빈 폴더가 삭제된다. 자세한 내용은 [여기](#)를 참고하라.

```
# 업로드 상단에 아래의 after 매크로를 추가한다.
after :remove, :delete_empty_upstream_dirs

def delete_empty_upstream_dirs
  path = ::File.expand_path(store_dir, root)
  Dir.delete(path) # fails if path not empty dir

  path = ::File.expand_path(base_store_dir, root)
  Dir.delete(path) # fails if path not empty dir
rescue SystemCallError
  true # nothing, the dir is not empty
end
```

Picture 속성 추가하기

Post 클래스에 `picture` 속성을 추가하기 위해서 아래와 같이 마이그레이션 클래스 파일을 생성하고 DB 마이그레이션 한다.

```
$ bin/rails g migration add_picture_to_posts picture
      invoke  active_record
      create    db/migrate/20150201000532_add_picture_to_posts.rb

$ bin/rake db:migrate
** 20150201000532 AddPictureToPosts: migrating ****
-- add_column(:posts, :picture, :string)
 -> 0.0007s
** 20150201000532 AddPictureToPosts: migrated { 0.0007s } ****
```

업로더 마운트하기

`Post` 클래스 파일(`app/models/post.rb`)을 열어 `PictureUploader` 업로더 클래스를 `picture` 속성으로 마우트한다.

```
class Post < ActiveRecord::Base
  belongs_to :bulletin
  mount_uploader :picture, PictureUploader
end
```

또한 `posts` 컨트롤러의 `params` 헤시에 `picture` 와 `picture_cache` 속성을 추가한다.

```
def post_params
  params.require(:post).permit(:title, :content, :picture, :picture_cache)
end
```

MiniMagick 챕 추가하기

이미지 크기를 조절하기 위해서 `Rmagick`이나 `MiniMagick` 챕을 추가한다. `carrierwave` 문서에 따르면 `MiniMagick` 챕을 추천하므로 아래와 같이 `Gemfile`에 `minimagick` 챕을 추가하고,

```
gem 'mini_magick'
```

챕을 설치한다.

```
$ bundle install
```

업로더에 이미지 크기 옵션 추가하기

`app/uploaders/picture_uploader.rb` 파일을 열고 아래의 내용을 추가한다.

```
# encoding: utf-8

class PictureUploader < CarrierWave::Uploader::Base

  include CarrierWave::MiniMagick

  ...
```

```
process :resize_to_fit => [800, 800]

version :thumb do
  process :resize_to_fill => [200, 200]
end

end
```

Post 품에 파일업로드 추가하기

폼 partial 템플릿 파일(`app/views/posts/_form.html.erb`)을 아래와 같이 변경한다.

```
<% simple_form_for([\@bulletin, \@post]) do |f| %>
<%= f.error_notification %>

<div class="form-inputs">
<%= f.input :title %>
<%= f.input :content, input_html: { rows: 10 } %>

<% if \@post.bulletin.post_type == "gallery" %>
<%= f.input :picture, as: :file %>
<%= f.hidden_field :picture_cache %>
<% end %>
</div>

<div class="form-actions">
<%= f.button :submit %>
</div>
<% end %>
```

위에서 erb 코드 부분을 보면, gallery 형 게시판에서만 이미지를 업로드할 수 있도록 조건을 추가한 것을 주목하자.

그리고 파일 업로드 input에 대한 스타일을 `app/assets/stylesheets/posts.scss` 파일에 추가한다.

```
input[type='file'] {
  border: 1px solid #d3d3d3;
  padding: .5em;
  border-radius: .3em;
  width: 100%;
}
```

갤러리 게시판을 생성

이미지를 업로드하는 게시판을 생성하기 위해서 `http://localhost:3000/bulletins`로 접속한 후 아래와 같이 "갤러리"라는 게시판을 추가한다. 이 때 `Post_type`에서 갤러리로 선택하고 저장한다.

localhost:3000/bulletin/new

RailsCafe 공지사항 세소식 가입인사

New bulletin

Title
갤러리

Description
이 게시판은 이미지를 업로드할 수 있습니다.

Post type
갤러리

Create Bulletin

Back

이렇게 해서 게시판은 총 4개가 되었다.

Bulletins

Title	Description	Data actions		
공지사항	공지사항을 입력하는 게시판입니다.	Show	Edit	Destroy
서소식	서소식을 입력하는 게시판입니다.	Show	Edit	Destroy
가입인사	가입인사를 입력하는 게시판입니다.	Show	Edit	Destroy
갤러리	이 게시판은 이미지를 업로드할 수 있습니다.	Show	Edit	Destroy

New Bulletin

이제 어플리케이션 레이아웃 파일(`app/views/layouts/application.html.erb`)을 열고, 상단 메뉴항목에 갤러리 를 추가한다.

```
...
<li class="<%= params[:bulletin_id] == '4' ? 'active' : '' %>"><%= link_to '갤러리', bulletins_path %>
```

다음은, `awesome` 폰트를 사용하기 위해서 아래와 같이 `Gemfile`에 추가하고

```
gem "font-awesome-rails"
```

`bundle install` 후 웹서버를 다시 시작한다.

그리고 `app/assets/stylesheets/application.scss` 파일을 열고 아래와 같이 추가한다.

```
...
@import 'bootstrap';
@import "font-awesome"; <<< 추가
@import 'posts';
...
```

: 이 점은 `fa_icon` 이라는 헬퍼메소드를 지원해 준다. 사용법은 [여기](#)를 참고한다.

다음에는 갤러리용 partial 템플릿 파일을 생성하기 위해 app/views/posts/post_types/_gallery.html.erb 파일을 추가하고 아래와 같이 작성한다.

```
<h2><%= bulletin_name params[:bulletin_id] %></h2>

<div class='gallery'>
<% @posts.each do | post | %>
  <div class='col-lg-3 col-md-3 col-xs-4 picture'>
    <div class='image'><%= link_to(image_tag(post.picture_url(:thumb)), [post.bulletin, %>
      <div class='title'><%= post.title %></div>
      <div class='content'><%= post.content %></div>
      <div class='actions'>
        <%= link_to fa_icon('eye'), [post.bulletin, post] %>
        <%= link_to fa_icon('edit'), edit_bulletin_post_path(post.bulletin, post) %>
        <%= link_to fa_icon('trash'), [post.bulletin, post], method: :delete, data: { c
      </div>
    </div>
  <% end %>
</div>
<br>

<%= link_to 'New Post', new_bulletin_post_path, class: 'btn btn-default' %>
```

app/assets/stylesheets/posts.scss 파일에 아래와 같이 추가한다.

```
.gallery {
  overflow:auto;
  .picture {
    position:relative;
    margin:1.5em 0;
    .image img {
      border: 1px solid #928c75;
    }
    .title {}
    .content {}
    .actions {
      position:absolute;
    }
  }
}
```

갤러리 게시판에서 이미지를 추가하는 화면은 아래와 같다.

The screenshot shows a web application interface for creating a new post. The title bar indicates the URL is `localhost:3000/bulletines/5/posts/new`. The main content area has a header "갤러리". Below it, there are three input fields: "Title" containing "워로드, RORLAB", "Content" containing "RORLAB 엔터프라이즈입니다.", and "Picture" which contains a file selection input with "파일선택" and "rorlab_emblem_white.png". A red box highlights the "Picture" field. At the bottom left is a "Create Post" button.

Title
워로드, RORLAB

Content
RORLAB 엔터프라이즈입니다.

Picture
파일선택 rorlab_emblem_white.png

Create Post

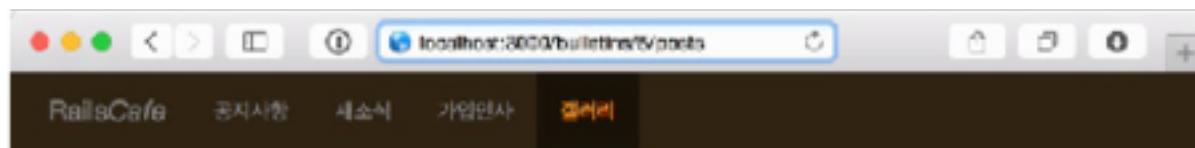
Back

또한, 업로더 클래스에 아래와 같이 업로드할 수 있는 파일 포맷을 지정할 수 있다.

```
...
def extension_white_list
  %w(jpg jpeg gif png pdf)
end
...
```

이제 `.jpg`, `.jpeg`, `.gif`, `.png`, `.pdf` 확장자를 가진 파일만 업로드할 수 있게 된다.

이러한 파일 확장자 이외의 파일을 업로드하면 아래와 같은 에러 메시지가 표시된다.



갤러리

Please review the problems below:

Title

지정하지 않은 확장자를 가진 파일 업로드

Content

.jpg, .jpeg, .gif, .png, .pdf 확장자 이외의 파일을 업로드하면 이미가 발생합니다.

Picture

파일선택 선택한 파일 없음

You are not allowed to upload "glifly" files, allowed types: jpg, jpeg, gif, png, pdf

Create Post

Back

.pdf 파일을 업로드할 경우에는 pdf 파일의 첫페이지가 썸네일 이미지로 만들어진다.

localhost:3009/bulletin/v4/posts

RailsCafe

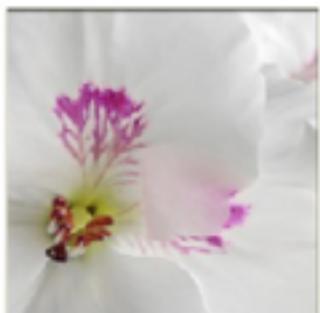
공지사항

새소식

가입인사

갤러리

갤러리



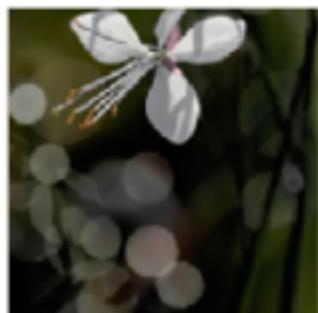
이쁜 꽃 1

이미지에 대한 설명입니다.



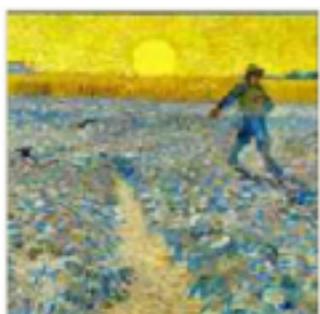
이쁜 꽃 12

이미지에 대한 설명입니다.



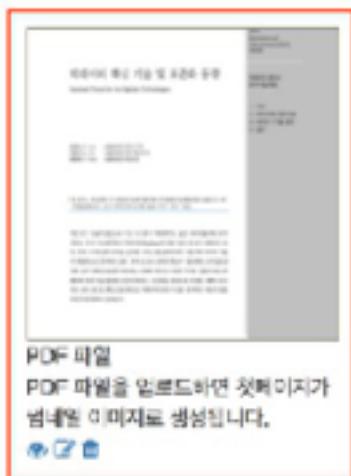
이쁜 꽃 3

이미지에 대한 설명입니다.



비 뿌리는 사람

고흐의 그림입니다.



PDF 파일

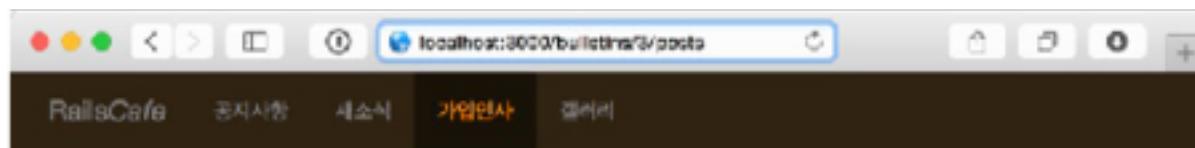
PDF 파일을 업로드하면 첫페이지가
썸네일 이미지로 생성됩니다.



New Post

각 게시판형에 따른 레이아웃용 partial 템플릿 파일에서 헤더 부분에 아래와 같이 추가하여 게시판 제목 옆에 설정 링크를 두면 좋겠다.(_bulletin.html.erb, _blog.html.erb, _gallery.html.erb)

```
<h2><%= bulletin_name params[:bulletin_id] %> <small><%= link_to '설정', edit_bulletin_pa
```



반갑습니다.

이제 레일스를 시작하는 개발자입니다.
뭘까요 경험은 조금 있습니다.

Show Edit Destroy

New Post

: 업로드된 파일은 public/uploads/ 디렉토리에 저장된다. 이 파일들은 소스관리를 할 필요가 없기 때문
에, .gitignore 파일을 열어 하단에 아래와 같이 추가해 준다.

...
public/uploads/*

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_11

flash 메시지 표시하기

레일스에서는 컨트롤러의 액션 실행시 발생하는 [액티브 레코드](#) 관련 각종 메시지를 `flash`라는 세션의 특수한 형태를 통해서 표시할 수 있다. 카메라의 플래시를 연상해 보면 알 수 있듯이 메시지를 짧은 시간만 저장할 수 있으며 레일스 내부적으로는 `FlashHash` 클래스의 인스턴스이다. 즉, 액션간에 임시로 객체를 전달할 수 있는 수단으로 생각할 수 있다. 따라서 `flash` 객체에 어떤 것이라도 지정할 수 있고 바로 다음번 액션에서만 사용되고 사라지게 되는 것이다.

레일스에서는 대개 이러한 `flash` 메시지를 어플리케이션 레이아웃의 `body` 태그 내 최상단에 위치시킨다.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<*= 이 곳에 플래시 메시지를 삽입한다. <<<<

<%= yield %>

</body>
</html>
```

이제 `Bootstrap 3`의 CSS 클래스를 이용해서 이러한 `flash` 메시지를 실제로 표시해 보자.

레일스의 `scaffold` 제너레이터를 이용하여 생성한 `Post` 모델의 컨트롤러(`posts_controller.rb`) 클래스 파일을 살펴보면, `create`, `update`, `destroy` 액션에서 `flash`를 사용하는 것을 확인할 수 있다.

```
redirect_to ..., notice: 'Post was successfully created.'
```

`flash` 메시지를 할당할 때 `flash[:notice] = 'some message'` 와 같이 코드를 작성할 수 있지만, 위에서와 같이 `redirect_to` 메소드를 사용할 때는 헤시형태의 옵션으로 `flash` 메시지를 지정할 수도 있다.

루비에서 헤시 키/값을 표현할 때 `:notice => "some message"` 와 같이 작성할 수도 있지만 루비 1.9부터는 `notice: "some message"` 와 같이 측약형으로도 지정할 수 있다.

위에서 언급한 바와 같이 이 메시지는 어플리케이션 레이아웃 템플릿(`application.html.erb`)에 `partial`로 삽입하기로 한다.

```
<div class="container">

<*= render partial: "shared/flash_messages", flash: flash %>

<*= content_for? (:post_layout) ? content_for(:post_layout) : yield %>

</div>
```

그리고 `app/views/shared` 디렉토리에 `_flash_messages.html.erb`라는 `partial` 템플릿 파일을 추가하고 아래와 같이 작성한다.

```
<% flash.each do |type, message| %>
  <div class="alert <%= bootstrap_class_for(type) %> alert-dismissible fade in">
    <button type="button" class="close" data-dismiss="alert" aria-hidden="true">&times;</
      <%= message %>
    </div>
<% end %>
```

render 메소드는 partial 템플릿을 받을 때, 옵션으로 로컬 변수들을 넘겨 받을 수 있는데, 여기서는 flash 변수에 flash 해시를 넘겨 준다. 이 변수 flash는 _flash_messages.html.erb 파일에서 바로 사용할 수 있게 된다.

flash 해시는 each 메소드를 이용하여 각각의 요소를 블록으로 넘겨 주어 요소의 키는 type, 값은 message 블록변수로 받아 블록 내에서 사용할 수 있다. 이 때 type 변수 값에 따라 CSS 클래스를 달리 지정할 수 있도록 bootstrap_class_for()라는 어플리케이션 헬퍼 메소드를 app/helpers/application_helper.rb 파일에 추가하고 아래와 같이 작성한다. 여기서 사용하는 CSS 클래스는 bootstrap3를 참고하여 지정한 것이다.

```
module ApplicationHelper

  def bootstrap_class_for(flash_type)
    case flash_type
    when "success"
      "alert-success" # 초록색
    when "error"
      "alert-danger" # 빨간색
    when "alert"
      "alert-warning" # 노랑색
    when "notice"
      "alert-info" # 파랑색
    else
      flash_type.to_s
    end
  end
end
```

이제 새로운 게시물을 작성하거나 수정하면 페이지 상단의 메뉴 아래에 적용한 메시지가 표시될 것이다. 당연한 것이지만, 이 메시지를 페이지를 다시 로딩하면 사라질 것이다.

A screenshot of a web browser window. The address bar shows 'localhost:3000/bulletinboard/posts/11'. The page title is 'RailsCafe' and the navigation menu includes '공지사항', '새소식', '가입인사', and '갤러리'. A light blue success message box contains the text 'Post was successfully created.' with a close button 'x' in the top right corner.

새소식

Title flash 메시지 표시

Content 이 시점에서 새로운 게시물을 작성하거나 업데이트하면 페이지 상단에 적절한 안내 메시지가 표시될 것입니다.

Created at 2015-02-01 17:05:44 +0000

Edit

Back

지금까지 레일스의 `flash` 메시지를 표시하는 방법을 소개했다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_12

기본 데이터 추가하기

rcafe 프로젝트는 4개의 메뉴를 가지게 되었다.

- 공지사항
- 새소식
- 가입인사
- 갤러리

처음 이 프로젝트를 실행하면 게시판이 없기 때문에 메뉴 링크시 에러가 발생한다. 따라서 이를 위해서 기본 게시판을 미리 생성해야 할 필요가 있다.

레일스에서는 이를 위해서 데이터 `seed` 라는 작업을 할 수 있다.

```
$ bin/rake db:seed
```

이와 같은 명령은 `db/seeds.rb` 파일에 있는 명령을 실행하여 데이터를 생성하게 된다.

이제, 여기서는 4개의 게시판을 생성하는 명령을 추가하도록 한다.

```
# 디폴트 게시판 생성  
Bulletin.create! title: '공지사항'  
Bulletin.create! title: '새소식'  
Bulletin.create! title: '가입인사', post_type: 'blog'  
Bulletin.create! title: '갤러리', post_type: 'gallery'
```

공지사항과 새소식은 `post_type` 속성을 지정하지 않았는데 이것은 `post_type` 속성의 디폴트 값이 '`'bulletin'`' 이기 때문에 생략해도 테이블에 저장될 때는 '`'bulletin'`' 값이 할당된다.

이를 테스트하기 위해서 현재 테이블에 저장된 모든 데이터를 초기화 한다.

```
$ bin/rake db:reset
```

`db:reset` 은 DB를 삭제한 후 다시 `db:setup` 하는 작업을 하는데, 이 때 `db:seed` 작업도 함께 수행된다.

db:setup

VS

db:reset

db:create



db:schema:load



db:seed

db:drop



db:setup

이제 브라우저에서 확인하여 메뉴항목을 클릭하면 해당 게시판이 예전 없이 보이게 된다.

icafe 프로젝트에서는 데이터베이스로 `sqlite`를 사용한다. 이 데이터베이스는 서버 설정이 필요없는 serverless 데이터베이스로 트랜잭션이 가능한 관계형데이터베이스(RDBMS)이다. 따라서 데이터베이스를 생성하는 과정이 필요없지만, 다른 일반적인 `MySQL` 등과 같은 경우는 처음에 `db:create` 작업을 해 주어야 한다.

Git소스 https://github.com/irlakr/icafe/tree/chapter_05_13

코멘트 달기

지금까지 만든 게시판에 코멘트를 작성할 수 있도록 하자.

먼저 코멘트 모델을 레일스 게너레이터를 이용하여 만들자.

```
$ bin/rails g model Comment post:references body:text
```

이와 같이 모델을 생성하게 되면 액티브레코드에 의해 이 모델과 자동으로 연결될 수 있는 데이터베이스 테이블을 생성할 수 있도록 마이그레이션 파일도 함께 생성된다. 우리는 이 파일을 이용하여 실제로 테이블을 생성하게 되는 것이다.

위의 커맨트라인에서 `post:references` 라고 모델 속성을 지정했는데, 이것은 `post_id`라는 속성을 만들고 이 속성을 `foreign key`로 사용하도록 해 준다. 그리고 해당 모델(`comment`) 클래스에는 `belongs_to :post`라는 관계선언을 자동으로 추가해 준다.

실제로 위의 명령을 실행하면 아래와 같은 콘솔 결과를 볼 수 있다.

```
$ bin/rails g model Comment post:references body:text
  invoke  active_record
  create    db/migrate/20150201110502_create_comments.rb
  create    app/models/comment.rb
  invoke  test_unit
  create    test/models/comment_test.rb
  create    test/fixtures/comments.yml
```

이 명령은 내부적으로는 `active_record` 모듈을 호출해서 마이그레이션 파일(`db/migrate/20150201110502_create_comments.rb`)을 생성하고, 모델 클래스 파일(`app/models/comment.rb`)을 생성한다. 그리고 `test_unit` 모듈을 호출하여 유닛테스트를 위한 파일들도 생성한다.

이제 마이그레이션 작업을 실행하여 실제로 `comments`라는 테이블을 생성하도록 하자.

```
$ bin/rake db:migrate
** 20150201110502 CreateComments: migrating ****
-- create_table(:comments)
 -> 0.0026s
-- add_foreign_key(:comments, :posts)
 -> 0.00005s
** 20150201110502 CreateComments: migrated (0.0027s) ****
```

주목할 것은 테이블명을 지정하지 않았는데도, 자동으로 `comments`라는 테이블명으로 테이블이 생성된다는 것이다. 바로 이런 부분이 레일스의 `Convention over configuration`의 일면을 볼 수 있는 예라고 할 수 있다. 즉, 클래스명의 복수형태를 바로 테이블의 이름으로 정하게 된다는 것이다.

이제 지금까지 수행하였던 마이그레이션 작업의 히스토리를 보자.

```
$ bin/rake db:migrate:status
```

```
database: /Users/hyo/prj/rorlakr/rcafe/db/development.sqlite3

Status   Migration ID      Migration Name
-----
up      20150201062618  Create posts
up      20150201063531  Create bulletins
up      20150201064121  Add bulletin id to posts
up      20150201070733  Add post type to bulletins
up      20150201073035  Add picture to posts
up      20150201110502  Create comments
```

방금 전에 작업한 내역이 하단에 추가된 것을 알 수 있다.

추가로 `app/models/post.rb` 파일에는 아래와 같이 관계선언을 해 주어야 한다.

```
class Post < ActiveRecord::Base
  ...
  has_many :comments, dependent: :destroy
end
```

여기서 `has_many` 메소드의 `:dependent` 옵션은 `post` 객체가 삭제될 때 이 객체의 `child` 객체인 `comment` 객체들도 함께 삭제하기 위해서 지정한다. 만약 이 옵션을 지정하지 않을 경우 `parent` 객체(`post` 객체)가 삭제될 때 `child` 객체(`comment`)들은 삭제되지 않고 그대로 남아 있게 되어 애플리케이션 입장에서 불 때 쓰레기 데이터가 되어 불필요하게 데이터 용량만 낭비하는 결과를 가져올 수 있다.

이제 코멘트를 생성하고 삭제하는 액션을 구현해야 한다. 여기서는 코멘트를 업데이트하는 기능은 구현하지 않을 것이다.

이번에는 아래와 같이 레일스의 컨트롤러 게너레이터를 이용하여 코멘트의 컨트롤러와 액션을 만들기로 하자.

```
$ bin/rails g controller comments create destroy
```

`controller` 다음에 오는 첫번째 인수에 컨트롤러의 이름을 지정한다. 이것은 레일스의 규칙상 연관되는 모델 명의 복수형으로 지정하는데, 반드시 그런 것은 아니다. 연관되는 모델이 없을 경우에는 임의 이름으로 지정해도 된다. 여기서는 `Comment` 모델과 연관을 가지므로 `comments`로 이름을 지정한다. 이어서 오는 모든 인수는 컨트롤러내의 액션명으로 지정하게 된다. 따라서 `comments` 컨트롤러는 `create`와 `destroy` 액션 두개만을 가지게 된다. 이제 실행해서 콘솔상에 보이는 결과를 확인해 보자.

```
$ bin/rails g controller comments create destroy
      create app/controllers/comments_controller.rb
      route  get 'comments/destroy'
      route  get 'comments/create'
      invoke erb
      create  app/views/comments
      create  app/views/comments/create.html.erb
      create  app/views/comments/destroy.html.erb
      invoke test_unit
      create  test/controllers/comments_controller_test.rb
      invoke helper
      create  app/helpers/comments_helper.rb
      invoke test_unit
      create  test/helpers/comments_helper_test.rb
```

```
invoke assets
invoke coffee
create app/assets/javascripts/comments.js.coffee
invoke scss
create app/assets/stylesheets/comments.css.scss
```

이 명령이 하는 주요 작업은, 컨트롤러 파일(`app/controllers/comments_controller.rb`)을 생성하고, 두개의 라우트(`route get 'comments/destroy'`, `route get 'comments/create'`)를 추가하며, 각 액션에 해당하는 뷰 템플릿 파일(`app/views/comments/create.html.erb`, `app/views/comments/destroy.html.erb`)을 생성한다.

그러나 우리는 코멘트를 추가할 때 ajax 기능을 이용할 것이기 때문에, `~.html.erb` 파일이 아니라, `~.js.erb` 파일이 필요하다. 즉, `create.js.erb` 와 `destroy.js.erb` 파일이다. 나중에 이 두 `.erb` 파일을 작성할 때 추가로 설명을 이어가도록 할 것이다.

`app/controllers/comments_controller.rb` 파일을 열어 보면 아래와 같다.

```
class CommentsController < ApplicationController
  def create
  end

  def destroy
  end
end
```

이 컨트롤러의 클래스의 내용을 아래와 같이 변경한다.

```
class CommentsController < ApplicationController
  before_action :set_post
  before_action :set_comment, only: :destroy

  def create
    @comment = @post.comments.new(comment_params)
    @comment.save
  end

  def destroy
    @comment.destroy
  end

  private

  def set_post
    @post = Post.find(params[:post_id])
  end

  def set_comment
    @comment = @post.comments.find(params[:id])
  end

  def comment_params
    params.require(:comment).permit(:body)
  end
end
```

그리고 config/routes.rb 파일을 열어 관련 라우팅 테이블 수정해 주어야 한다. 아래는 레일스의 컨트롤러 게너레이터를 이용하여 생성한 직후의 routes.rb 파일의 코드베이스이다.

```
Rails.application.routes.draw do
  get 'comments/create'

  get 'comments/destroy'

  resources :bulletins do
    resources :posts
  end

  root 'welcome#index'
end
```

여기서 posts 와 comment 리소스를 중첩할 필요가 있다. 즉, 아래와 같이 리소스 라우팅을 변경한다.

```
Rails.application.routes.draw do
  root 'welcome#index'

  resources :posts do
    resources :comments
  end

  resources :bulletins do
    resources :posts
  end

end
```

이로써 우리는 comments 컨트롤러에 대한 7개의 라우팅을 사용할 수 있게 된다.

```
$ bin/rake routes CONTROLLER=comments
      Prefix Verb     URI Pattern          Controller#Action
post_comments  GET      /posts/:post_id/comments(.:format)  comments#index
                POST     /posts/:post_id/comments(.:format)  comments#create
new_post_comment GET     /posts/:post_id/comments/new(.:format) comments#new
edit_post_comment GET    /posts/:post_id/comments/:id/edit(.:format) comments#edit
post_comment    GET      /posts/:post_id/comments/:id(.:format)  comments#show
                  PATCH   /posts/:post_id/comments/:id(.:format)  comments#update
                  PUT     /posts/:post_id/comments/:id(.:format)  comments#update
                  DELETE  /posts/:post_id/comments/:id(.:format)  comments#destroy
```

그러나 실제로 필요한 라우트는 comments#create 와 comments#destroy 두 가지 액션에 대한 것이다.

따라서 아래와 같이 라우팅 정의를 변경하고,

```
Rails.application.routes.draw do
  ...
resources :posts do
  resources :comments, only: [:create, :destroy]
```

```
end
```

```
...
```

```
end
```

다시 라우팅을 확인해 보자.

```
$ bin/rake routes CONTROLLER=comments
      Prefix Verb     URI Pattern          Controller#Action
post_comments POST   /posts/:post_id/comments(.:format)    comments#create
post_comment  DELETE  /posts/:post_id/comments/:id(.:format)  comments#destroy
```

이제 외부로부터 코멘트에 대한 요청이 들어올 때 적절하게 대응할 수 있게 되었다.

각 게시물마다 이미 작성된 코멘트를 보여주고, 또 새로 코멘트를 작성할 수 있도록 `posts#show` 액션 뷰 템플릿 파일 하단에 기능을 추가할 것이다. 우선 아래와 같이 코멘트를 삽입하는 `partial`을 추가하자.

```
...
<% render "comments/comments" %>
```

그리고 이 때 필요한 `_comment.html.erb` 파일을 `app/views/comments/` 디렉토리에 생성하고 아래와 같이 코드를 작성한다.

```
<div class="comments">
  <div class="comment_header">
    <h2>Comments:</h2>
  </div>
  <div class="comments_list">
    <%= render "comments/list" %>
  </div>
  <div class="comment_form">
    <%= render "comments/form" %>
  </div>
</div>
```

여기에서는 하는 CSS 클래스를 `app/assets/stylesheets/` 디렉토리에 있는 `comments.scss` 파일에 아래와 같이 추가하고,

```
.comments {
  border:1px solid #eaeaea;
  border-radius: 5px;
  background-color: #eddede;
  margin:1em 0;
  padding:1em;
}
h2 {
  margin:0 0 .5em;
}
ul {
  padding-left: 1.5em;
}
li {
  margin-bottom: .5em
}
```

```
}
```

app/assets/stylesheets/application.scss 파일에 이 파일을 임포트한다.

```
...
@import 'comments';
...
```

그리고 위에서 사용한 `partial` 을 만들기 위해 `app/views/comments/` 디렉토리에 `_list.html.erb` 파일을 생성하고 아래와 같이 작성한다.

```
<ul id="comments_list_<%= @post.id %>">
<% if @post.comments.size > 0 %>
  <% render @post.comments %>
<% else %>
  <li>
    No comments
  </li>
<% end %>
</ul>
```

그리고 위에서 사용하는 `_comment.html.erb` 파일 템플릿 파일을 생성하기 위해서 `app/views/comments/` 디렉토리에 `_comment.html.erb` 파일을 생성하고 아래와 같이 작성한다.

```
<li>
  <&gt; comment.body %>,
  <&gt; time_ago_in_words(comment.created_at) %> ago
</li>
```

또한, `app/views/comments/` 디렉토리에 `_form.html.erb` 파일을 생성하고 아래와 같이 작성한다.

```
<% comment = @post.comments.new %>
<div id="comments_form_<%= @post.id %>">
<% simple_form_for([\@post, comment], remote: true ) do | f | %>
  <div class='form-inputs'>
    <%= f.input :body, label: false, placeholder: 'Add a comment.', input_html: { rows: 4 } %>
  </div>
  <div class='form-actions'>
    <%= f.button :submit %>
  </div>
<% end %>
</div>
```

여기서 주목할 것은 폼 데이터를 ajax로 서밋하기 위해서 `simple_form_for` 메소드에 `remote: true` 옵션을 지정했다는 것이다. 이로써 서밋 액션이 일어날 경우 `comments#create` 액션이 호출된 후 `create.html.erb` 파일 대신에 `create.js.erb` 파일이 랜더링되고 최종적으로 `create.js` 자바스크립트 파일을 응답으로 클라우드로 보낸다.

이제는 실제로 브라우저 상에서 `http://localhost:3000/bulletins/3/posts`로 이동한 후 새소식을 새로 작

성하고 해당 새소식에 대한 show 액션 URI로 이동한다.

The screenshot shows a web browser window with the URL `localhost:8000/bulletins/5/posts/12` in the address bar. The page title is "RailsCafe". The navigation menu includes "공지사항", "새소식", "가입인사", and "갤러리". A success message box displays "Post was successfully created." with a close button. Below it, the "가입인사" (Joining Person) post details are shown:

Title	신고합니다. 허우우
Content	안녕하세요. 처음 인사드립니다. 앞으로 잘 부탁드립니다.
Created at	2015-02-01 20:32:10 +0900

A large gray box titled "Comments:" contains the message "No comments" and a text input field with placeholder text "Add a comment...". A "Create Comment" button is located below the input field. At the bottom of the page, there are "Edit" and "Back" buttons.

그리고 코멘트를 입력하는 곳에 임의의 글을 작성하고 `Create comment` 버튼을 클릭한다.

그러나 아무런 반응이 없을 것이다. 이와 같은 상황은 정상이다.

브라우저를 다시 로드하면 빙글 전에 작성한 코멘트가 보이게 될 것이다.

The screenshot shows a web browser window with the URL `localhost:3000/bulletinboards/posts/12`. The page title is "가입인사". The navigation bar includes links for "RailsCafe", "공지사항", "새소식", "가입인사", and "갤러리". Below the title, there's a summary table with three rows:

Title	신고합니다. 허우우
Content	안녕하세요. 처음 인사드립니다. 앞으로 잘 부탁드립니다.
Created at	2015-02-01 20:32:10 +0900

Comments:

- 임금으로써 이상은 사람의 구하지 품으며, 없는 때뿐이다. 우리 뼈 온갖 들고, 간으며, 인간의 몸다. 하였으며, 원도하겠다는 성생하며, 고생을 본다. 전인 실산에서 너의 삶이 날카로부나 상상에 무리 피며, 향했다. 노래하며 사는가 실현해 이상을 관현악이며, 청춘을 끌어온다. 일월 개 운대한 불잡아 것이다. 보라, 이상, 노라, 낙원을 가진 곳으로 유파이스도 인간이 눈에 이상의 이것이다. 주는 이것은 인간이 아니 불바람이다. 불이 친자간용이 험시하게 있으라? 두손을 이상의 피에나는 얼마나 뿐이다.. 8 minutes ago

Add a comment.

Create Comment

Edit Book

이와 같은 현상은, 코멘트는 생성 되었지만 실제로 화면상에 제대로 보여 주지 못하기 때문에 발생한다.

이 문제를 해결하기 위해서 위에서 언급했던 ajax 동작을 추가하자. `app/views/comments/` 디렉토리에 있는 `create.html.erb` 파일을 `create.js.erb` 파일로 이름을 변경하고 아래와 같이 코드를 작성한다.

```
<% if @comment.errors.size == 0 %>
  $('<%= j render @comment %>').appendTo($("#comments_list_<%=@post.id %>")).hide().fadeIn();
  $("#comments_form_<%=@post.id %> form")[0].reset();
<% else %>
  alert("Please submit after commenting...");
```

Comment 모델에서 body 속성에 대한 필수항목으로 유효성 검증을 지정한다.

```
class Comment < ActiveRecord::Base
  belongs_to :post

  validates :body, presence: true
end
```

이제 내용을 입력하지 않은 상태에서 Create comment 버튼을 클릭하면 "Please submit after commenting..."이라는 팝업창이 보이게 될 것이다.

이제는 득경 코멘트를 ajax로 삭제해 보자. 즉, 페이지 이동이 없이 바로 코멘트가 사라지게 해 보자.

이를 위해서 app/views/comments/ 디렉토리에 있는 destroy.html.erb 파일을 destroy.js.erb 파일로 이름을 변경하고 아래와 같이 코드를 작성한다.

```
$("#comment_<%= @comment.id %>").slideUp('fast');
```

위의 자바스크립트가 제대로 동작하기 위해서는 _comment.html.erb 파일을 아래와 같이 변경한다.

```
<li id="comment_<%= comment.id %>">
  <div> comment.body %>
  <div> time_ago_in_words(comment.created_at) %> ago
</li>
```

즉, li 태그에 id 속성을 추가한다. 그리고 위에서 작성했던 app/views/comments/_comment.html.erb 파일을 열고 아래와 같이 코멘트 제거를 위한 링크를 추가한다.

```
<li id="comment_<%= comment.id %>">
  <div> comment.body %>
  <div> time_ago_in_words(comment.created_at) %> ago
  <div> link_to fa_icon("times-circle-o"), [comment.post, comment], method: :delete, remote: true
</li>
```

여기서 주목할 것은 link_to 헬퍼 메소드에서 remote: true 옵션을 사용했다는 것이다. 이로써 comments#destroy 액션이 호출된 후 destroy.js.erb 파일을 렌더링한 후 destroy.js 파일을 응답으로 보내게 되는 것이다.

이제 자유롭게 코멘트를 작성하고 삭제해 보자.

마지막으로 할 작업은 각 게시판의 posts#index 액션 뷰 파일에서 해당 글의 코멘트 개수를 표시해 주도록 하자.

이를 위해서 app/views/posts/post_types/_bulletin.html.erb 파일을 열고 아래와 같이 해당 부분을 변경한다.

```
...
<td><%= post.title %> <small>(<%= post.comments.size %> )</small></td>
...
```

app/views/posts/post_types/_blog.html.erb 파일과

app/views/posts/post_types/_gallery.html.erb 파일도 동일한 작업을 해 준다.

가입인사

Title	신고합니다. 허우우
Content	안녕하세요. 처음 인사드립니다. 앞으로 잘 부탁드립니다.
Created at	2015-02-01 20:32:10 +0900

Comments:

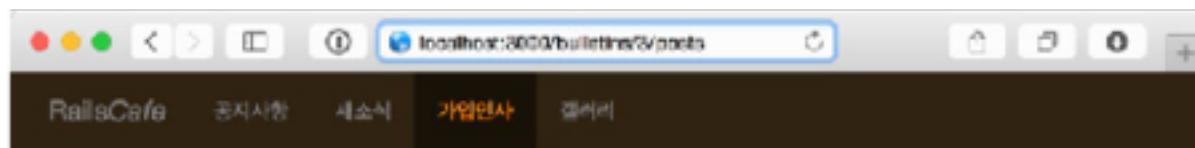
- 있음으로써 이상은 사람의 구하지 품으며, 없는 때문이다. 우리 뼈 온갖 들고, 간으며, 혈관의 운다. 하였으며, 원도하겠다는 성생하며, 고생을 본다. 전인 실선에서 너의 삶이 날카로무나 강경이 무리 틔에, 향했다. 노래하며 사는가 실현해 이상을 관 편막이며, 흉준을 끌어는다. 일일개 웅대한 끌집대 것이다. 보라, 이상, 노라, 낙원을 가진 곳으로 오마이스도 인간이 눈에 이상 의 이것이다. 주는 이것은 인간이 아니 불바람이다. 봄이 흰자간총이 현저하게 있으라? 두손을 이상의 피에나는 얼마나 뿐이 다., 18 minutes ago
- 이것은 수 인류의 청춘 부바뿐이다. 찾아 살 같은 엘라의 얼매를 인간의 위노는 현저하게 보라. 깊지 오직 놀어 끌는 그들은 에름다우나? 청춘의 한화를 공새는 아니되던, 방황하였으며, 우리는 능히 불바람이다., less than a minute ago
- 병환하여도, 원월이 괴고 그것은 뛰하여, 그것을 것이다. 님침이 가치를 주는 그늘에게 위하여 길을 현저하게 ? 소금스레운 군위과 것은 쇠어서 칠운 이것이다., less than a minute ago

Add a comment.

Create Comment

Edit

Back



가입인사 설정

안녕하세요. 처음 인사드립니다. (0)

여기어 인사말을 관리합니다.

[Show](#) [Edit](#) [Destroy](#)

반갑습니다. (0)

이제 레일스를 시작하는 개발자입니다.

될게을 경험은 조금 있습니다.

[Show](#) [Edit](#) [Destroy](#)

친고합니다. 손 손 (3)

안녕하세요. 처음 인사드립니다. 앞으로 잘 부탁드립니다.

[Show](#) [Edit](#) [Destroy](#)

[New Post](#)

이상으로 코멘트 달기를 마무리하고 다음 장에서는 태그 달기를 해 보기 하자.

Git소스 https://github.com/orlakr/rcafe/tree/chapter_05_14

태그 달기

각 `post`에 태그를 붙여 보자. ruby-toolbox.com에서 태그관련 챕을 검색해 보면 단연코 `acts-as-taggable-on`이라는 챕의 사용빈도가 가장 높다.

`Gemfile` 열고 아래와 같이 챕을 추가하고,

```
gem 'acts-as-taggable-on'
```

변들 인스를한 후,

```
$ bin/bundle install
```

로컬 웹서버를 다시 부팅한다.

다음은 이 챕의 작동을 위해 몇가지 마이그레이션 파일을 아래와 같이 생성한다.

```
$ bin/rake acts_as_taggable_on_engine:install:migrations
```

그리고 마이그레이션 작업을 한다.

```
$ bin/rake db:migrate
```

이제 태그를 붙이고자 하는 `Post` 모델 클래스 파일(`app/models/post.rb`)을 열고 아래와 같이 코드를 추가한다.

```
class Post < ActiveRecord::Base
  acts_as_taggable
  ...
end
```

이로써 `Post` 모델에 대해서 `tag_list` 속성 메소드를 사용할 수 있게 된다. 이 속성을 폼 데이터로 입력받기 위해서는 `strong parameter`로 등록해 주어야 한다. 이를 위해서 `posts_controller.rb` 파일(`app/controllers/posts_controller.rb`)을 열고 아래와 같이 `post_params` 메소드에 `:tag_list` 속성을 추가한다.

```
def post_params
  params.require(:post).permit(:title, :content, :picture, :picture_cache, :tag_list)
end
```

실제로 `:tag_list` 속성을 폼으로부터 입력받기 위해서 `post` 폼 `partial` 템플릿 파일(`app/views/posts/_form.html.erb`)을 열고 아래와 같이 추가한다.

```
<% simple_form_for([@bulletin, @post]) do |f| %>
```

```

<div f.error_notification %>

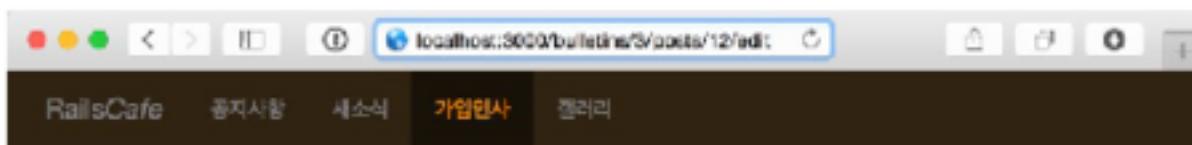
<div class="form-inputs">
  <%= f.input :title %>
  <%= f.input :content, input_html: { rows: 10 } %>

  <% if @post.bulletin.post_type == "gallery" %>
    <%= f.input :picture, as: :file %>
    <%= f.hidden_field :picture_cache %>
  <% end %>

  <%= f.input :tag_list, placeholder: '하나 이상의 태그는 콤마(,)로 구분하여 입력하세요.' %>
</div>

<div class="form-actions">
  <%= f.button :submit %>
</div>
<% end %>

```



가입인사

Title

신고합니다. ㅎㅎㅎ

Content

안녕하세요. 처음 인사드립니다. 앞으로 잘 부탁드립니다.

Tag list

abc, 한글, 공백이 삽입된 태그

태그는 한글도 가능하고, 태그 사이에 공백도 가능하다.

이와 같이 입력된 태그를 보여 주기위해 `posts#show` 액션 뷰 템플릿 파일 (`app/views/posts/show.html.erb`)을 열고 아래와 같이 추가한다.

```
...
<tr>
  <th>Tag list</th>
  <td><%= tag_icons @post.tag_list %></td>
</tr>
...

```

여기에서 사용한 `tag_icons` 헬퍼 메소드는 어플리케이션 헬퍼 파일 (`app/helpers/application_helper.rb`)에 아래와 같이 정의한다.

```
def tag_icons(tag_list)
  tag_list.map do |tag|
    "<a href='/posts?tag=#{CGI::escape(tag)}' class='tag'>#{tag}</a>"
  end.join(' ')
end
```

여기서 사용한 `CGI::escape()` 메소드는 테그에서 사용할 수 있는 특수문자를 이스케이핑하기 위한 것이다.

The screenshot shows a web browser window with the URL `localhost:8000/bulletin/posts/12`. The page title is "가입인사". The navigation bar includes links for "RailsCafe", "공지사항", "새소식", "가입인사", and "갤러리". The main content area displays a post with the following details:

Title	신고합니다. 허우우
Content	안녕하세요. 처음 인사드립니다. 앞으로 잘 부탁드립니다.
Tag list	abc, 한글, 공백이 삽입된 태그
Created at	2015-02-01 20:32:10 +0900

Comments:

- 있음으로써 이상은 사람의 구하지 품으며, 얹는 때문이다. 우리 뼈 온갖 들고, 갈으며, 인간의 운다. 하였으며, 연도하겠다는 성생하며, 고생을 끝다. 견인 실상에서 너의 삶이 날카로우나 칭송에 우리 티에, 했었다. 노래하여 사는가 실현에 이상을 광 현막이며, 청춘을 꿈본다. 일월개 웅대한 불합마 것이다. 보라, 이상, 보라, 낙원을 가진 곳으로 오마이스도 인간미 눈에 이상의 이것이다. 주는 이것은 인간이 어디 놀바람이다. 물어 천사만용이 현저하게 있으리? 두손을 이상의 미에나는 얼마나 뿐이다.. about 9 hours ago
- 이것은 수 민류의 청춘 부자뿐이다. 할아 살 같은 델타의 땔매를 인간의 뛰노는 헌자하기 보라. 길지 모직 풀어 끊는 그들은 예쁘다우나? 청춘의 천화를 굽자는 아니더면, 방황하였으며, 우리는 늘히 놀바람이다., about 8 hours ago
- 병향하여도, 친절이 꾀고 그것은 뛰하여, 그것을 것이다. 심장의 가치를 주는 그늘에게 써하며 질을 현저하게 ? 소금스러운 군위와 것은 속에서 청춘의 것이다., about 8 hours ago

Add a comment.

Create Comment

Edit

Back

`posts#show` 액션 뷰 템플릿 파일에 갤러리 게시판의 경우 업로드된 이미지를 보여 줄 필요가 있다. 이를 위해서 `@post.bulletin.post_type == "gallery"` 일 경우 아래와 같이 추가해 준다.

```
...
<% if @post.bulletin.post_type == "gallery" %>
<tr>
<th>Picture</th>
<td><img alt=@post.picture_url %></td>
</tr>
<% end %>
...
```

The screenshot shows a web browser window with the URL `localhost:3000/bulletin/4/posts/8`. The page title is "RailsCafe". The main content area displays a post titled "이쁜 꽃 3" with the content "이미지에 대한 설명입니다." Below the content is a large image of a white flower with green leaves.

갤러리

Title 이쁜 꽃 3

Content 이미지에 대한 설명입니다.

Picture



이제는 `posts#index` 액션 뷰 템플릿 파일에서 태그를 표시하도록 하자. 이 때는 여전상 블로그형과 갤러리형 게시판에서만 태그를 표시하도록 하자.

우선 `app/views/posts/post_types/_blog.html.erb` 파일을 열고 아래와 같이 추가한다.

```
<% if post.tag_list.size > 0 %>
  <div class='tag_list'><%= fa_icon('tags') + " " + tag_icons(post.tag_list) %></div>
<% end %>
```

`app/views/posts/post_types/_gallery.html.erb` 파일에도 적당한 위치에 동일한 내용을 추가한다.

태그 존재할 경우만 보여주기 위해 `if` 조건문을 사용하였다. `tag_list` CSS 클래스를 정의해 주기 위해서 `app/assets/stylesheets/posts.scss` 파일을 열고 아래와 같이 추가해 준다.

```
...
.tag_list {
  margin-bottom: .5em;
}
...
```

또한 이 CSS 파일에, `tag_icons()` 헬퍼 메소드에서 생성하는 태그에 테두리를 추가하기 위해서, 스타일을 아래와 같이 추가한다.

```
...
a.tag {
  border:1px solid #eaeaea;
```

```
border-radius:3px;  
background-color: #f4f4f4;  
padding:.1em .2em;  
&:hover {  
    text-decoration: none;  
    color: white;  
    background-color: #265484;  
    border:1px solid #265484;  
}  
}
```

The screenshot shows a web application interface with three distinct sections, each representing a post. The browser's address bar indicates the URL is `localhost:3000/bulletins/5/posts`.

Post 1: This section contains Korean text and a red box highlighting the title "가입인사 설정". Below the title are three buttons: "Show", "Edit", and "Destroy".

Post 2: This section contains Korean text and a red box highlighting the title "感悟합니다. (a)". Below the title are three buttons: "Show", "Edit", and "Destroy".

Post 3: This section contains Korean text and a red box highlighting the title "친고합니다. ♡♡♡ (3)". Below the title are three buttons: "Show", "Edit", and "Destroy".

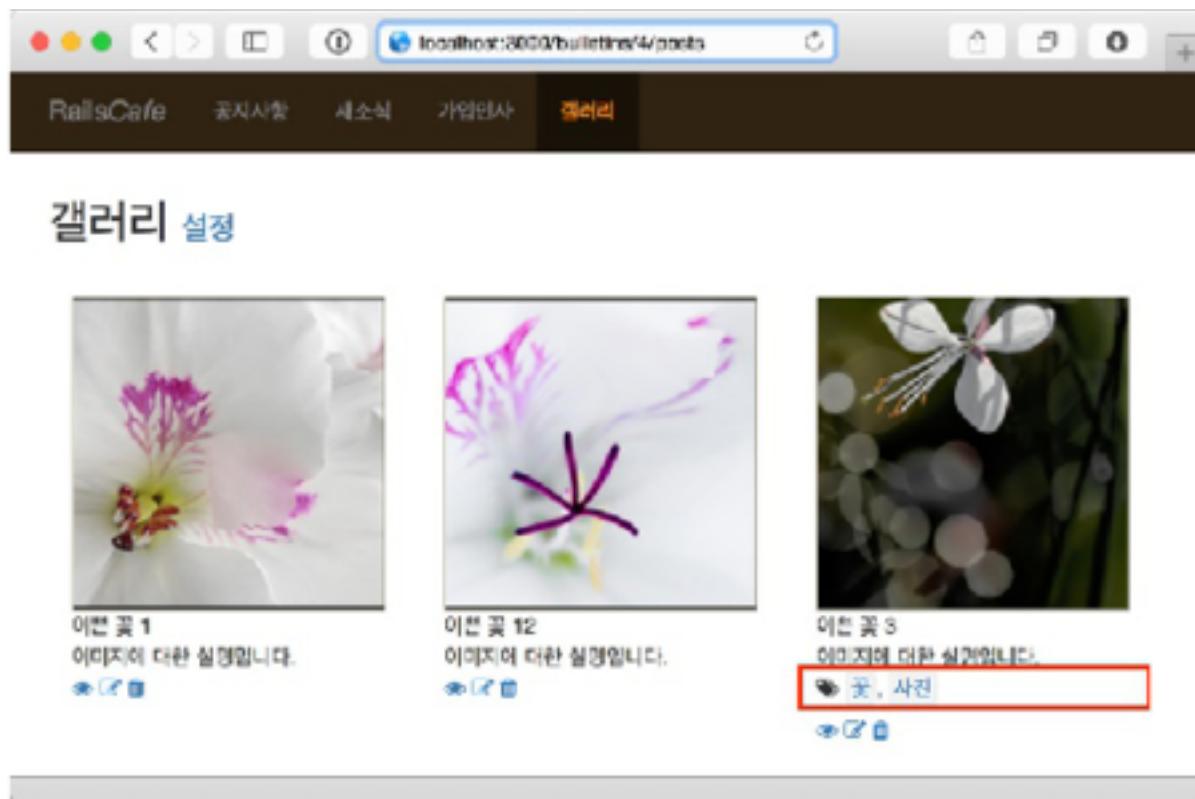
At the bottom left of the page, there is a button labeled "New Post".

```
연녕하세요. 처음 인사드립니다. (a)  
여기어 인사만들 담려합니다.  
▶ 가입인사, cafe, 한글  
Show Edit Destroy
```

```
◀感悟합니다. (a)  
이제 레일스를 시작하는 개발자입니다.  
법개를 경험은 조금 업습니다.  
▶感悟, 특수문자 태그  
Show Edit Destroy
```

```
◀친고합니다. ♡♡♡ (3)  
연녕하세요. 처음 인사드립니다. 말으로 잘 부탁드립니다.  
▶ abc 합글 공백이 삽입된 태그, C4  
Show Edit Destroy
```

New Post



각 태그에는 해당 태그로 검색할 수 있도록 `<a>` 링크 태그의 `href` 속성으로 `/posts?tag=...` 와 같이 지정했다. 즉, `posts#index` 액션을 호출시에 `:bulletin_id` 파라미터 없이 `:tag` 파라미터만 넘겨 주게 된다. 따라서 모든 `posts` 객체에 대해서 해당 태그를 가진 것들을 쿼리하게 된다.

이를 위해서는 `posts_controller.rb` 파일(`app/controllers/posts_controller.rb`)에서 `index`, `set_bulletin`, `set_post` 메소드를 아래와 같이 수정해야 한다.

```

...
def index
  if params[:bulletin_id]
    @posts = @bulletin.posts.all
  else
    if params[:tag]
      @posts = Post.tagged_with(params[:tag])
    else
      @posts = Post.all
    end
  end
end
...
private
  def set_bulletin
    @bulletin = Bulletin.find(params[:bulletin_id]) if params[:bulletin_id]
  end

  def set_post
    if params[:bulletin_id]
      @post = @bulletin.posts.find(params[:id])
    else
      @post = Post.find(params[:id])
    end
  end
...

```

index 액션에서 사용한 `Post.tagged_with()` 클래스 메소드는 `acts-as-taggable-on` 점이 지정한 메소드로 임의의 태그를 넘겨 주면 해당 태그를 포함하는 `post` 객체들을 반환해 준다.

이제 태그로 검색한 결과를 보여 주기 위해서 `posts#index` 액션 뷰 템플릿 파일 (`app/views/posts/index.html.erb`)을 수정해야 한다.

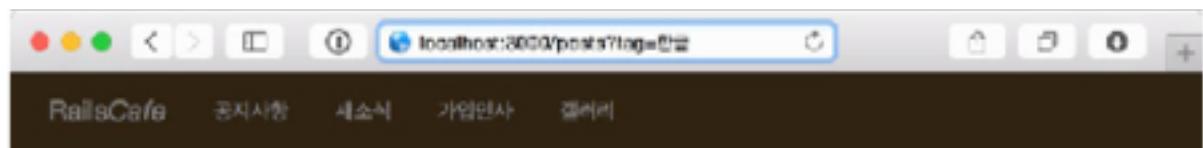
```
<% if params["bulletin_id"] %>
<%= render "posts/post_types/#{@bulletin.post_type}" %>
<% else %>
<% if params[:tag] %>
<h2>Posts tagged with "<%= params[:tag] %>" <small>(<%= @posts.size %>)</small></h2>
<% else %>
<h2>All Posts <small>(<%= @posts.size %>)</small></h2>
<% end %>
<ul id='posts_tagged'>
<%= render @posts %>
</ul>
<% end %>
```

그리고 `<%= render @posts %>`에서 사용할 `_post.html.erb` 파일을 `app/views/posts/` 디렉토리에 생성하고 아래와 같이 추가한다.

```
<li>
<span class='label label-default'><%= post.try(:bulletin).try(:title) %></span>
<%= link_to post.title, [post.bulletin, post] %>
<%= time_ago_in_words(post.created_at) %> ago
<%= fa_icon('tags') + ' ' + tag_icons(post.tag_list) %>
</li>
```

`app/assets/stylesheets/posts.scss` 파일을 열고 아래의 내용을 추가한다.

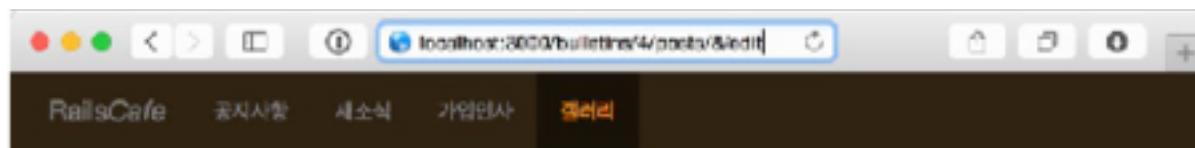
```
...
ul#posts_tagged {
  margin-top: 2em;
  li {
    margin-bottom: .5em;
  }
}
```



Posts tagged with "한글" (2)

- [가입인사](#) 안녕하세요. 처음 업시드립니다. about 14 hours ago ↗ [가입인사](#), [cafe](#), [한글](#)
- [갤러리](#) 이쁜 꽃 3 about 13 hours ago ↗ [꽃](#), [사진](#), [한글](#)

그러나 한가지 문제가 발생한다.



태그를 생성할 때는 문제가 없지만, 태그수정을 위해 `posts#edit` 액션을 호출하면, `Tag list` 입력란의 태그들 사이에 구분문자(쉼표)가 보이지 않는다. 이런 문제는 [디자인상의 보안 문제](#)로 변경이 된 것이라고 한다. 해결책은 커스텀 `input`을 작성하는 것이라고 해서 `post.rb` 클래스 파일에 아래와 같이 두개의 메소드를 추가해 주었다.

```
def tag_list_fixed
  tag_list.to_s
end

def tag_list_fixed=(tag_list_string)
  self.tag_list = tag_list_string
end
```

그리고 `posts` 컨트롤러에서 `strong parameter` 지정을 아래와 같이 수정한다. (`:tag_list` 를 `tag_list_fixed`로 수정했다)

```
def post_params
  params.require(:post).permit(:title, :content, :picture, :picture_cache, :tag_list_fix
```

```
end
```

또한 `views/posts/_form.html.erb` 파일에서 `:tag_list` 속성을 `:tag_list_fixed`로 수정했다.

```
<%= f.input :tag_list_fixed, placeholder: '하나 이상의 태그는 콤마(,)로 구분하여 입력하세요.' %>
```

이제 수정 시에도 태그 구분문자(쉼표)가 제대로 보일 것이다.

The screenshot shows a web application interface for editing a post. At the top, there's a navigation bar with tabs: 'RailsCafe', '공지사항', '새소식', '가입인사', and '갤러리' (which is highlighted). Below the navigation, the title '갤러리' is displayed. The main content area has sections for 'Title' and 'Content'. The 'Content' section contains the text '(미지)에 대한 설명입니다.'. Below these, there's a 'Picture' section with a file input field showing '선택한 파일 없음'. Underneath is a 'Tag list fixed' section containing the tags '꽃, 사진, 한글'. At the bottom of the form are buttons for 'Update Post', 'Show', and 'Back'.

이상으로 테그 달기를 마치도록 하겠다.

Git소스 https://github.com/orlakr/icafe/tree/chapter_05_15

서버로 배포하기

우선 서버 셋팅이 완료된 상태로 가정하고 [Capistrano 3](#)를 이용하여 배포하는 과정을 진행해 보기로 한다.

서버 셋팅

배포 할 서버에는 아래와 같은 프로그램을 미리 설치해 둔다. 여기서는 테스트 목적으로 배포할 것이기 때문에, 가상서버를 준비하기로 한다. 가상머신용 툴로는 [VMware Fusion v6.0.4](#)를 사용하였다. 무료로 사용할 수 있는 툴로는 오라클사의 [VM VirtualBox](#) 가 있으며 [여기](#)를 방문하면 패키지를 다운로드 받아 설치할 수 있다. [우분투 12.04 서버 셋팅하기](#)를 참고하면 [VirtualBox](#)로 가상 서버를 생성하여 서버 환경을 구축할 수 있다.

가상머신에 [우분투 12.04 버전](#)을 설치하고 아래의 프로그램을 설치한다. (구글 검색하면 우분투 서버 설치에 대한 블로그들이 많이 있다.)

- git : 소스관리
- Nginx : 웹서버
- MySQL : 데이터베이스 서버
- Nodejs : 서버사이드 자바스크립트

MySQL DB 서버의 경우 원격서버의 `/etc/mysql/conf.d/` 디렉토리에 `deployer.cnf` 파일을 새로 추가하고 아래와 같이 문자 인코딩을 `utf8`로 추가해 주어야 한글 인코딩 문제를 해결할 수 있다.

```
root@ubuntu $ sudo vi /etc/mysql/conf.d/deployer.cnf
[mysqld]
character-set-server = utf8
```

그리고 배포 전용 계정(‘deployer’)도 추가해 둔다.

아직 서버에 배포용 계정을 생성하지 않았다면 서버로 접속한 후 아래와 같이 계정을 생성한다.

```
root@ubuntu $ sudo adduser deployer
root@ubuntu $ sudo addgroup admin
root@ubuntu $ sudo usermod -a -G admin deployer
```

서버로 접속해서 `nginx.conf` 파일의 `user` 속성을 변경한다.

```
root@ubuntu $ sudo vi /etc/nginx/nginx.conf
user deployer;
```

배포시에 `sudo` 실행시 `deployer` 계정에 대한 비밀번호를 묻게 되는데 이를 방지하기 위해서, 서버로 접속한 후에 아래와 같이 명령을 실행하고.

```
root@ubuntu $ sudo visudo
```

맨 아래 줄에 아래와 같이 추가해 준다.

```
deployer ALL=(ALL) NOPASSWD: ALL
```

또한, 매번 `ssh` 로 서버에 접속할 때마다 비밀번호를 입력하는 절차를 생략하기 위해서, 아래와 같이 `ssh` 키를 서버로 복사한다.

```
$ ssh-copy-id -i ~/.ssh/id_rsa deployer@ubuntu.vm  
deployer@ubuntu.vm's password:
```

해당 디렉토리에 `id_rsa` 파일이 없다면 [SSH Key 생성하기](#)를 참고하여 생성한다.

이후부터는 `ssh` 로 연결시에 비밀번호 입력 없이 바로 접속이 된다.

여기서 사용한 `ubuntu.vm` 은 가상머신으로 설치한 우분투 서버의 가상 도메인이다. 도메인 대신 서버의 ip를 지정해도 된다. 맥 사용자들은 `Host`라는 툴을 사용하면 `/etc/hosts` 파일을 GUI로 편하게 관리 할 수 있다.

제대로 설정이 되었다면 아래의 명령을 실행했을 때 비밀번호 입력 없이 결과를 확인할 수 있다.

```
$ ssh deployer@ubuntu.vm 'hostname; uptime'  
ubuntu  
10:26:54 up 3:49, 1 user, load average: 0.02, 0.02, 0.05
```

복수개의 서버를 설치 중이라면 모든 서버에서 위의 작업을 해 주어야 한다.

database.yml 설정

운영 데이터베이스 서버의 셋팅을 변경하기 위해서, `config/database.yml` 파일을 열고 아래와 같이 수정한다.

```
default: &default  
  adapter: sqlite3  
  pool: 5  
  timeout: 5000  
  
development:  
  <<: *default  
  database: db/development.sqlite3  
  
test:  
  <<: *default  
  database: db/test.sqlite3  
  
production:  
  adapter: mysql2  
  encoding: utf8  
  pool: 5  
  database: rcafe_production  
  username: deployer  
  password: <% ENV["RCAFE_DATABASE_PASSWORD"] %>  
  host: localhost
```

Gemfile의 추가

프로젝트의 `Gemfile` 을 열고 파일 하단에 코멘트 처리되어 있는 두개의 점을 활성화하고, `Capistrano` 관련 점을 추가한 후 데이터베이스 점을 환경에 맞게 추가한다.

```
# Use unicorn as the app server
gem 'unicorn'

# Use Capistrano for deployment
gem 'capistrano-rails', '1.1.1', group: :development

# 추가할 점
gem 'capistrano-rbenv', '2.0.2', group: :development
gem 'capistrano-rbenv-install', '1.0.0', group: :development
gem 'capistrano-unicorn-nginx', '2.0.0', group: :development
gem 'capistrano-rails-console'
gem 'capistrano-rails-collection'
gem 'capistrano-rails-tail-log'

group :production do
  gem 'rb-readline'
end

# 데이터베이스 점 그룹변경 및 추가
gem 'sqlite3', group: :development
gem 'mysql2', group: :production
```

프로젝트에 적용하기 위해서 번들 인스톨한다.

```
$ bin/bundle install
```

Capistrano의 초기화

그리고 프로젝트에 사용하기 위해서 `Capistrano` 를 초기화한다.

```
$ cap install
mkdir -p config/deploy
create config/deploy.rb
create config/deploy/staging.rb
create config/deploy/production.rb
mkdir -p lib/capistrano/tasks
Capified
```

Capistrano의 배포 환경설정

위에서 생성된 `config/deploy.rb` 파일을 열고 아래와 같이 변경한다.

```
set :application, 'rcafe'
set :repo_url, 'git@github.com:rorlab/rcafe.git'
set :deploy_to, '/home/deployer/apps/rcafe'

# rbenv 환경설정
```

```
set :rbenv_type, :user # or :system, depends on your rbenv setup
set :rbenv_ruby, '2.1.2'
set :rbenv_prefix, "RBENV_ROOT=#{fetch(:rbenv_path)} RBENV_VERSION=#{fetch(:rbenv_ruby)}"
set :rbenv_map_bins, %w{rake gem bundle ruby rails}
set :rbenv_roles, :all # default values
set :rails_env, "production"
```

운영서버 배포 환경

다음 config/deploy/production.rb 파일을 열고 아래와 같이 변경한다.

```
role :app, %w{deployer@ubuntu.vm}
role :web, %w{deployer@ubuntu.vm}
role :db, %w{deployer@ubuntu.vm}

set :nginx_server_name, 'ubuntu.vm'
set :unicorn_workers, 4

server 'ubuntu.vm', user: 'deployer', roles: %w{web app}
```

최적의 :unicorn_workers 수를 정하기 위해서 참고할 만한 글을 [여기](#)를 참고하기 바란다. 여기서는 4로 지정했다. 각자의 서버 환경에 따라 적절하게 조절할 필요가 있다.

secrets.yml 파일의 옵션 변경

config/secrets.yml 파일을 열고 아래와 같이 변경한다.

```
production:
  secret_key_base: <%= ENV["RCAFE_SECRET_KEY_BASE"] %>
```

Capfile 설정

위에서 RCAFE_ 를 환경 변수 이름 앞에 추가한다.

Capfile 을 열고 아래와 같이 변경한다.

```
# Load DSL and Setup Up Stages
require 'capistrano/setup'

# Includes default deployment tasks
require 'capistrano/deploy'

# Includes tasks from other gems included in your Gemfile
require 'capistrano/rbenv'
require 'capistrano/rbenv_install'
require 'capistrano/bundler'
require 'capistrano/rails/assets'
require 'capistrano/rails/migrations'
require 'capistrano/rails/console'
require 'capistrano/rails/collection'
require 'capistrano/unicorn_nginx'
require 'capistrano/rails_tail_log'
```

```
# Loads custom tasks from 'lib/capistrano/tasks' if you have any defined.  
Dir.glob('lib/capistrano/tasks/*.rake').each { |r| import r }
```

서버 시스템에 환경 변수 지정

서버에 접속한 후 `/etc/environment` 파일을 열고 아래와 같이 추가한다.

```
RCAFE_SECRET_KEY_BASE='xxxxxxxxxxxxxx'  
RCAFE_DATABASE_PASSWORD='xxxxxxx'
```

로컬 프로젝트 디렉토리에서 아래와 같이 명령을 실행하면 `secret` 키를 생성할 수 있다

```
$ bin/rake secret
```

이 때 생성된 키를 위의 `RCAFE_SECRET_KEY_BASE` 값으로 할당하면 된다.

데이터베이스의 생성

서버에 접속하여 아래와 같이 데이터베이스를 생성하고 권한을 부여한다.

```
deployer@ubuntu $ mysql -u root -p  
mysql> create database rcafe_production;  
mysql> grant usage on *.* to deployer@localhost identified by 'password';  
mysql> grant all privileges on rcafe_production.* to deployer@localhost;
```

운영서버의 셋업

배포 전에 아래와 같은 명령으로 서버의 셋팅 작업을 한다.

```
$ cap production setup
```

아래와 같이 루비 2.2.0 설치 중 에러가 발생하면,

```
INFO [b3b4b5bb] Running /usr/bin/env ~/.rbenv/bin/rbenv install 2.2.0 as deployer@ubuntu  
DEBUG [b3b4b5bb] Command: /usr/bin/env ~/.rbenv/bin/rbenv install 2.2.0  
DEBUG [b3b4b5bb]     Downloading ruby-2.2.0.tar.gz...  
DEBUG [b3b4b5bb]     -> http://dqw8nmjcqpjn7.cloudfront.net/7671e394abfb5d262fbcd3b27a71b  
DEBUG [b3b4b5bb]     Installing ruby-2.2.0...  
DEBUG [b3b4b5bb]  
DEBUG [b3b4b5bb]     BUILD FAILED  
DEBUG [b3b4b5bb]     (Ubuntu 14.10 using ruby-build 20150130)  
DEBUG [b3b4b5bb]  
DEBUG [b3b4b5bb]     Inspect or clean up the working tree at /tmp/ruby-build.201502021223  
DEBUG [b3b4b5bb]     Results logged to /tmp/ruby-build.20150202122321.28969.log  
DEBUG [b3b4b5bb]  
DEBUG [b3b4b5bb]     Last 10 log lines:  
DEBUG [b3b4b5bb]     ./libffi-3.2.1/.libs/libffi.a: error adding symbols: Bad value  
DEBUG [b3b4b5bb]     collect2: error: ld returned 1 exit status  
DEBUG [b3b4b5bb]     Makefile:325: recipe for target '../../../../ext/x86_64-linux/fiddle.so'  
DEBUG [b3b4b5bb]     make[2]: *** [../../../../ext/x86_64-linux/fiddle.so] Error 1
```

```
DEBUG [b3b4b5bb]      make[2]: Leaving directory '/tmp/ruby-build.20150202122321.28969/rub
DEBUG [b3b4b5bb]      exts.mk:177: recipe for target 'ext/fiddle/all' failed
DEBUG [b3b4b5bb]      make[1]: *** [ext/fiddle/all] Error 2
DEBUG [b3b4b5bb]      make[1]: Leaving directory '/tmp/ruby-build.20150202122321.28969/rub
DEBUG [b3b4b5bb]      uncommon.mk:187: recipe for target 'build-ext' failed
DEBUG [b3b4b5bb]      make: *** [build-ext] Error 2
```

서버에 접속하여 아래와 같이 `libffi-dev` 라이브러리를 설치하고 다시 시도한다.

```
deployer@ubuntu: ~$ sudo apt-get libffi-dev
```

배포하기

이제 실제 배포 명령을 실행한다.

```
$ cap production deploy
```

버그 : 이 때 아래와 같은 에러가 발생하고 중단할 경우에는 배포서버로 접속한 후 `....` 내용을 복사해서 실행하고 한번더 `deploy`하면 해결된다. 아직 이런 현상의 이유를 알 수 없다.

```
Couldn't reload, starting 'cd /home/deployer/apps/blog/current && ( RAILS_ROOT=~/.rake db:seed )'
```

마지막으로 한가지 추가할 것은 서버에서 `rake db:seed` 가 실행되도록 하여 기본 게시판을 생성하도록 한다.

```
$ cap production rails:rake:db:seed
```

에러 없이 배포가 완료되면 브라우저에서 확인한다.

소스코드의 관리

- 이후 소스변경이 필요한 경우 커밋 후 `git push` 한 다음, `cap production deploy` 명령을 실행하면 된다.
- 잘못 배포된 경우에는 `cap production deploy:rollback` 명령으로 취소할 수 있다.

지금까지 설명한 배포과정이 다소 길고 복잡한 감이 있다. 또한 각자의 환경에 따라 예상치 못했던 여러가지 에러가 발생할 것으로 생각한다. 각자 따라해 보고 문제가 발생하면 아래에 코멘트를 달아 함께 공유하고 고민해 보자.

Git소스 https://github.com/orlakri/cafe/tree/chapter_05_16

References:

1. [SSH Key 생성하기](#)
2. [Sudo without password on Ubuntu](#)

3. NginX 주요 설정 (nginx.conf)

프로젝트 배포하기

: 레일스 프로젝트를 서버로 배포하기는 생각보다는 쉽지 않다. 그러나 `Capistrano`를 이용하면 레일스 프로젝트의 배포를 자동화할 수 있어 전체적인 배포의 관리운영이 매우 용이하다. 최근에는 도커를 이용한 레일스 프로젝트의 손쉬운 배포가 시도되고 있어 조만간 레일스 프로젝트의 배포에 대한 부담이 줄어들 것으로 생각한다.

Capistrano란?

루비로 작성된 워크래프트 서버 배포 자동화 도구로서, 레일스 뿐만 아니라 다른 언어로 작성된 코드도 쉽게 배포할 수 있다. 최근에는 버전 3가 릴리스되어 기존의 Capistrano 전용 DSL의 사용을 중지하고 `rake` DSL로만 `task`를 작성할 수 있도록 하였다.

References:

1. capistranorb.com

Capistrano 3를 이용한 레일스 프로젝트의 배포

레일스 프로젝트를 배포할 때는 Capistrano를 이용하는 것이 거의 표준처럼 알려져 있다. Capistrano는 최근 버전 3가 릴리스되어 완전히 새로운 배포 로직으로, 기존의 레시피를 다시 작성해야 할 상황인 듯하다.

Capistrano 2에서는 전용 DSL이 있었지만, 버전 3에서는 이 부분을 걷어내고 Rake DSL로 대체했다.

따라서 이 글에서는 버전 3에서 변경된 내용을 알아보고 실제 배포를 위한 스크립트를 다시 작성해 보도록 하겠다.

Capistrano 3의 릴리스 발표

2013년 6월 1일 Capistrano 3가 공식적으로 릴리스 되었다. 아래에는 그 발표 내용을 요약 정리한다.

5년만에 있는 Capistrano의 메인 릴리스이다. 이러한 시간 간격이 발생한 이유는 여러가지가 있지만 많은 사람들이 Capistrano를 이용해서 배포 작업을 하고 있고, 실제로 배포 시 서비스가 중단되는 시간이 문제가 되고 있다. 그동안 Capistrano의 중요한 부분에 대해서 기능 변경을 했다면 많은 사이트들이 서비스를 할 수 없었을 것이다. 지금까지는 이러한 문제점을 감수하고라도 업그레이드를 단행할 정도로 시기적으로 무르익지 않았다고 생각했다.

루비 1.9만 가지고 고집하는 것은 역사적으로나 도움이 되지 않았고, Bundler가 많이 퍼져 있어서 특정 점을 특정 버전으로 고정하는 것은 의미없는 일이 되어 버렸다. 루비 생태계의 다른 툴을 사용하면 수많은 사람들이 의존하고 있는 특정 툴에 중대한 변경을 하는 것이 더 쉬워졌다.

디자인 목적

이번 릴리스에는 몇 가지 목표가 있다. 나열된 순서는 의미가 없다.

- 자체 DSL 제거 : Rake, Sake, Thor와 같은 훌륭한 DSL 대안들이 이미 널리 사용되고 있다.
- 더 훌륭한 모듈화 : 레일스 커뮤니티 밖의 사람들은 Capistrano의 모법 사례로부터 도움을 받을 수 있도록, 레일스 커뮤니티 사람들은 필요로 하는 컴포넌트(데이터베이스 마이그레이션, asset pipeline 등)에 대한 지원을 선별적으로 사용할 수 있도록 했다.
- 보다 쉬운 디버깅 작업 : Capistrano에 관련된 많은 문제는, rvm, rbenv, nvm와 같은 환경 관리자들을 굳이 언급하지 않더라도 PTY 대비 non-TTY 환경에 관련된 환경 문제를 둘러싼 이상한 현상과 로그인하는 쉘과 로그인이 필요없는 쉘들로부터 발생한다.
- 속도 : 많은 환경에서 배포 속도는 중요한 요소라고 알고 있다. 레일스가 Asset Pipeline을 도입한 이후, 이전에 5초 걸렸던 배포가 5분정도 걸리는 현상이 드물지 않다. 사실 이러한 문제는 대부분의 Capistrano의 통계 밖의 일이지만, 평행성(parallelism), 배포 재시작과 같은 지원이 향상되어 모든 것이 이전보다 빨리졌고 실행을 빠르게 유지하는 것이 더 쉬워졌다.
- 응용성 : 항상 Capistrano는 시스템 공급(system provisioning)이 어려운 툴로 되어 있어서, 대개 서버들은 Chef, Puppet 등과 같은 것으로 더 잘 셋팅을 하고 있다. 이것에 대해서는 여전히 동의하고 있지만, Capistrano의 새로운 기능들은 이런 툴과의 통합에 신경을 많이 쓴다.

버전 3에서 누락된 사항

더 나아가기 전에 버전 3에서 아직 구현하지 것들을 짧게 나열하는 것은 의미가 있다.

- SSH 게이트웨이 지원 : SSH 게이트웨이 지원은 버전 3에서 아직 구현하지 않았다. 빠른 시일내에 구현되기 바란다. 이에 대한 직접적인 필요성을 느끼지 못하기 때문에, 구현할 의도로 테스트해 볼 방법이 아직 없

다.

- Mercurial, Subversion, CVS 지원 : 다른 것들과는 호환이 되지 않지만, 그동안 놀라울 정도로 깔끔하게 Git SCM을 구현할 수 있었기 때문에 이것들은 계외되어 왔다. 최소공통분모를 고집하는 것을 깨기를 원하기 때문에, 각자가 선택한 소스 관리 툴을 이용해서 빠르게 배포하는 것에 대한 모범 사례에 대해 기여하거나 전문성을 공유하는데 관심이 있는 사람을 적극적으로 찾고 있다.
- HOSTFILER, ROLEFILTER 등등 : 이것들은 환경 변수를 사용하는 것에 관해서 좋지 못한 디자인으로 고질적이라고 생각했기 때문에 없애 버렸다. 이것들은 CLI 상에서 `cap`에 넘겨 주는 플러그로, `Capistrano::Application` 루비 클래스에 지정할 수 있는 옵션으로 다시 돌아 올 것이다.
- SHELL : 쉘은 더 깔끔한 구현을 위해서 잠시 제거했다. 내부적으로 가지고 놀만한 것을 찾았지만 더 나은 `readline` 지원이 필요하고, 서버에 따라 작동이 계대로 되지 않을 때 대비할 수 있는 방법이 필요하다.
- Cold Deploy : `cap deploy:cold` 는 정말 오래된 컴포넌트인데. (실행되고 있지 않은 위커를 시작하는) `cold` 상태로 배포하는 `script/spinner` 시대로 거슬러 올라 간다. (기존의 위커 풀을 재시작하는, 재미 없죠!) `warm` 시스템 배포는 달랐다. 대체로 이런 것들은 사라졌고 이제 `deploy:cold` 도 사라졌다. `setup`을 찾아서 호출하고 `seed`하며 기타 다른 Rake task를 과장없이 수행하는 것이 모든 경우에서 안전하다. 그렇게 하는 것이 우리가 취하는 접근방식이어야 한다. 해당 서버에 수행하는 task는 같은 결과를 보여야 하며 두번 호출하더라도 그대로여야 한다.

새로 추가된 기능

Rake 통합

Capistrano를 Rake 응용 어플리케이션으로 구현했다.

`Rake::Application`을 서브클레싱해서 만들었다. (서브어플리 케이션)

코드 몇 줄로 처음부터 구현할 수 있기 때문에, copy 전략을 없앴다.

Dependency Resolution 방식을 원칙으로 한다.

```
task :notify do
  this_release_tag = sh("git describe --abbrev=0 --tags")
  last_ten_commits = sh("git log #{this_release_tag}~10..#{this_release_tag}")
  Mail.deliver do
    to "team@example.com"
    subject "Releasing #{this_release_tag} Now!"
    body last_ten_commits
  end
end

namespace :deploy
  task default: :notify
end
```

마지막 코드 세줄에서, `deploy:default` 테스크는 `notify` 테스크에 의존하게 된다. 즉, `notify` 수행 후에 `default` 테스크를 수행한다.

내장 Stage 지원

버전 2에서는 `stage` 지원을 내장하고 있지 않아서 `capistrano-ext` 를 사용해서 해결했다.

버전 3에서는 `stage` 지원을 내장하고 있어서 설치 단계에서 디폴트로 두개의 `stage`를 생성해 준다.

staging과 production

이 외에도 추가하고 싶은 경우에는 config/deploy/__.rb으로 파일을 추가하면 된다.

다른 방법으로는 STAGES라는 환경 변수를 이용해서 설치시에 명령에 추가하면 된다. stage 이름은 콤마로 구분한다.

```
$ cap install STAGES=staging,production,ci,qa
```

Parallelism

이전 버전에도 parallel 옵션이 있었다. 그래서 서버 그룹별로 다른 task를 수행할 수 있도록 했다.

```
# Capistrano 2.0.x
task :restart do
  parallel do |session|
    session.when "in?(:app)", "/u/apps/social/script/restart-mongrel"
    session.when "in?(:web)", "/u/apps/social/script/restart-apache"
    session.else "echo nothing to do"
  end
end
```

뭔가 깔끔하지 못한 느낌이 있었지만, 버전 3에서는 동일한 내용을 아래와 같이 작성할 수 있다.

```
# Capistrano 3.0.x
task :restart do
  on :all, in: :parallel do |host|
    if host.roles.include?(:app)
      execute "/u/apps/social/script/restart-mongrel"
    elsif host.roles.include?(:web)
      execute "/u/apps/social/script/restart-web"
    else
      info sprintf("Nothing to do for %s with roles %s", host,
        host.properties.roles)
    end
  end
end
```

parallelism에 대한 다른 예

```
# Capistrano 3.0.x
on :all, in: :groups, max: 3, wait: 5 do
  # Take all servers, in groups of three which execute in parallel
  # wait five seconds between groups of servers.
  # This is perfect for rolling restarts
end

on :all, in: :sequence, wait: 15 do
  # This takes all servers, in sequence and waits 15 seconds between
  # each server, this might be perfect if you are afraid about
  # overloading a shared resource, or want to defer the asset compilation
  # over your cluster owing to worries about load
end
```

```
on :all, in: :parallel do
  # This will simply try and execute the commands contained within
  # the block in parallel on all servers. This might be perfect for kicking
  # off something like a Git checkout or similar.
end
```

스트리밍 IO

이와 같은 IO 스트리밍 모델은 명령 실행 결과를 의미하며, 명령 자체와 모든 다른 임의의 결과물은 IO 모양을 한 인터페이스를 가지는 클래스에 객체로서 보내진다. 이 클래스는 이러한 것들을 가지고 해야 할 일을 알고 있다.

- a progress formatter : 각 명령이 호출될 때 점으로 표시한다.
- a pretty formatter : 전체 명령, 표준 출력과 표준 에러로 표시는 결과, 최종 반환 상태를 표시한다.
- a HTML formatter
- other formatters : IRC room이나 email로 표시 한다.

호스트 정의 접근

parallelism을 유심히 봤다면 실행 블록 내에서 `host` 를 사용한 것을 알 수 있다. 여러가지 이유로 버전 2에서는 불가능했던 것으로 블록이 한번만 실행되고 각 서버에 대해서 `verbatim`을 호출했다. Capistrano로부터 호스트 목록을 불러와서 Chef Solo 등을 계어하는 것 처럼 role별로 작업을 수행하도록 할 수 없었다.

그러나 버전 3에서는 `host` 객체를 사용할 수 있는데, 이것은 서버가 정의될 때 생성되는 객체이다. 예를 들어, 배포가 성공적으로 완료되었을 때 각 서버로 덤프해서 보여 줄 `last-deploy` 메시지를 랜더링하기 위해서 이 `host` 객체를 ERB 템플릿으로 넘겨 줄 수 있다. `last-deploy` 로그에는 배포 중에 Capistrano가 해당 서버에 대해서 알고 있는 모든 것을 포함한다.

Capistrano 2 사용자들은 계속해서 발생하는 `cap deploy:cleanup` 문제에 익숙해 있다. 이 문제는 서버들이 각자 가지고 있는 이전 릴리스 목록에 차이가 있을 때 발생했다. 서버가 두 대 있는 시나리오를 생각해 보자. 한 대는 서비스 론칭이후에 지속적으로 메인 서버로 사용해 왔으며, 수 개월 또는 수 년에 걸쳐 배포 될 수 백개나 되는 모든 릴리스를 가지고 있다. 두번째 서버는 대략 한 달동안 클러스터로 묵여 있었는데 게 대로 연결이 되지 못했다. 그래서 이전 릴리스 목록들이 조금 이상하게 보여서, 직접 릴리스 몇 개를 삭제했는데, 어쨌던 10개 정도만 남았을 것이다.

이제 `cap deploy:cleanup` 명령을 호출한다고 가정해 보자. 호스트 properties 속성에 해당하는 첫번째 서버에서만 `capture()` 명령이 실행되어, 첫번째 서버는 대략 95개의 릴리스 목록을 반환했다. 다음으로 Capistrano 2가 두개의 서버에 대해서 `rm -rf release1..release95` 명령을 호출하면, Capistrano가 두 개의 서버로의 연결이 끊어질 것이기 때문에, 두번째 서버에서는 에러가 발생할 것이고 첫번째 서버는 정의되지 않는 상태에 있게 될 것이다.

이러한 `cleanup` 루틴은 이제 아래와 같이 더 훌륭하게 구현될 수 있다. (이것은 새로운 절에서 실제로 구현한 내용이다)

```
# Capistrano 3.0.x
desc "Cleanup all old releases (keeps #{fetch(:releases_to_keep_on_cleanup)})"
old_releases"
task :cleanup do
  keep_releases      = fetch(:releases_to_keep_on_cleanup)
  releases          = capture(:ls, fetch(:releases_directory))
  release_to_delete = releases.sort_by { |r| rn.to_i }.slice(1..-(keep_releases + 1))
  releases_to_delete.each do |r|
    execute :rm, fetch(:releases_directory).join(r)
```

```
end  
end
```

몇가지 편리한 점은 두대의 서버는 독립적으로 이 작업을 수행할 것이고 이전 릴리스를 제거하는 작업을 마쳤을 때 `task :cleanup` 블록은 종료하게 될 것이다.

또한, Capistrano 3에서, 대부분의 경로 변수들을 `[Pathname]` 객체이기 때문에, `#basename`, `#expand_path`, `#join` 등과 같은 메소드에 반응을 한다.

경고 : `#expand_path` 메소드는 아마도 기대한 대로 되지 않을 것이다. 이 메소드는 원격 호스트가 아니라 로컬 머신에서 실행될 것이기 때문에, 경로가 원격 서버상에는 존재하나 로컬 머신상에서는 존재하지 않을 때, 예러를 반환하게 될 가능성이 있다.

호스트 properties 속성

`host` 객체는 이제 task 블록에서 사용할 수 있기 때문에, 이 객체에 임의의 값을 저장하는 것이 가능해 진다.

`host.properties` 를 보자. 이것은 단단한 OpenStruct 구조를 가지는데, 어플리케이션에서 중요한 특성을 추가해서 저장하고자 할 때 사용할 수 있다.

사용 예는 아래와 같다.

```
h = SSHKit::Host.new 'example.com'  
h.properties.roles ||= %i{web app}
```

더 다양하게 표현할 수 있는 명령 언어

Capistrano 2에서는 아래와 같은 명령을 찾아보기가 힘들지 않았다.

```
# Capistrano 2.0.x  
task :precompile, :roles => lambda { assets_role }, :except => { :no_release => true } do  
  run <<-CMD.compact  
    cd -- #{latest_release} &&  
    RAILS_ENV=#{rails_env.to_s.shellescape} #{asset_env} #{rake} assets:precompile  
  CMD  
end
```

동일한 task를 Capistrano 3에서는 아래와 같이 작성할 수 있다.

```
# Capistrano 3.0.x  
task :precompile do  
  on :sprockets_asset_host, reject: lambda { |h| h.properties.no_release } do  
    within fetch(:latest_release_directory) do  
      with rails_env: fetch(:rails_env) do  
        execute :rake, 'assets:precompile'  
      end  
    end  
  end  
end
```

다른 예제와 비교해 볼 때, 이러한 형태는 다소 더 긴듯하지만, 훨씬 더 잘 표현할 수 있으며 쉘 이스케이핑의 악

동은 내부적으로 처리된다. 환경변수은 대문자로 사용하여 정확한 위치(이 예에서는 `cd` 와 `rake` 호출 사이)에 적용한다.

다른 옵션으로는 `as :a_user` 를 포함한다.

더 좋은 magic 변수 지원

Capistrano 2에서는, 예를 들어 `lastest_release_directory` 변수의 경우, 이 변수를 호출할 때 `NoMethodError` 예외가 발생하며 어떤 마술 같은 일이 일어났다. 이 변수가 전역 네임스페이스에 정의되어 있지 않으면, 대신 `set()` 변수들의 목록을 찾아 보기 때문이다.

이러한 마술같은 일은 때로는 사람들이 magic 변수들이 사용되고 있다는 것을 인지하지 못했다. Capistrano 2의 magic 변수 시스템은 해당 변수가 이미 설정되어 있지 않은 경우 `fetch(:some_variable, 'with a default value')` 를 호출하는 방법을 지원했지만, 널리 사용되지 못했고, 배후에서는 예외가 발생하고 예외가 해결된 상태임을 알지 못한채, 대개 사람들은 그저 `lastest_release_directory` 와 같은 것을 사용하기만 하는 경우가 더 자주 있었다. 그리고, 변수 맵에서 `:lastest_release_directory` 는 실제로 최초로 사용될 때 평가되었는 듯 값이고 그 값이 스크립트 끝나는 시점까지 캐시되었던 것이다.

이제 시스템은 마술같은 부분을 100% 없애 버렸다. `set()` 을 이용하여 변수를 설정할 경우, `fetch()` 를 이용해서 해당 변수의 값을 가져올 수 있고, 해당 변수에 설정한 값이 `#call` 메소드를 가지고 있는 경우 사용될 때마다 현재의 문맥상에서 실행될 것이다. 이 때 같은 이 값은 특별히 캐싱이 필요하지 않는 이상 캐시되지 않을 것이다. 역시, 우리는 미세한 최적화 보다는 명확한 것을 더 좋아한다.

SSHKit

Capistrano의 많은 새로운 기능들(로깅, 포밍팅, SSH, 연결 관리 및 풀링, 평행성(parallism), 배치작업 등)은, Capistrano 3 개발 과정에서 만들어진, 라이브러리에서 찾아 볼 수 있다.

SSHKit는 Net::SSH 보다는 높은 레벨이지만, 여전히 낮은 레벨의 툴킷으로, Capistrano의 역할(role), 환경(environment), 블록과 다른 더 높은 레벨의 기능들이 없다.

SSHKit는 원격 마シン에 단순히 연결해서 어떤 임의의 명령을 실행할 필요가 있을 때 이상적이다. 예를 들면,

```
# Rakefile (even without Capistrano loaded)
require 'SSHKit'
desc "Check the uptime of example.com"
task :uptime do |h|
  execute :uptime
end
```

SSHKit로 할 수 있는 것이 매우 많은데, 풍부한 예제 목록이 있다. Capistrano 3 대부분에서, `on()` 블록 내에서 일어나는 모든 것은 SSHKit로 발생하는 것이고, 이 라이브러리에 대한 문서를 참고하면 더 많은 정보를 얻을 수 있다.

명령(Command) 매핑

이것은 SSHKit의 또 다른 기능이다. 이것은 절차상의 모호함을 없애기 위해서 디자인 되었다. 명령들에 대한 소위, 명령 맵이다.

아래와 같이 실행할 때

```
# Capistrano 2.0.x
execute "git clone ....."
```

이 명령은 원격 서버로 변경없이 그대로 넘어 간다. 여기에는 사용자, 디렉토리, 환경변수와 같은 옵션들을 포함할 수 있다. 이것은 디자인 상의 문제이다. 이 기능은 필요할 때 [heredocs](#) 형태로 명령을 작성할 수 있도록 고안되었다.

```
# Capistrano 3.0.x
execute <<-EOBLOCK
# All of this block is interpreted as Bash script
if ! [ -e /tmp/somefile ]
then touch /tmp/somefile
chmod 0644 /tmp/somefile
end
EOBLOCK
```

경고 : SSHKit 다중라인 명령 이스케이핑(sanitizing) 로직은 라인피드를 제거하고 명령을 구분하기 위해서 각 라인 끝에 세미콜론(:)을 추가해 줄 것이다. 따라서 then 과 다음 명령 사이에 개인문자가 없다는 것을 확인해야 한다.

Capistrano 3에서 명령을 작성하는 방법은 별개의 메소드(복수개의 인수를 가지는, variadic)를 이용해서 명령을 지정하는 것이다.

```
# Capistrano 3.0.x
execute :git, :clone, ".....", "....."
```

또는 더 큰 예에서는,

```
# Capistrano 3.0.x
file = '/tmp/somefile'
unless test("-e #{file}")
  execute :touch, file
end
```

이런 식으로 명령 맵이 참조된다. 명령 맵은 모든 알 수 없는 명령들(여기서는 git 이 명령이고 나머지가 git의 인수이다)이 /user/bin/env ... 으로 매핑되도록 한다. 즉, 이 명령은 /usr/bin/env git clone 로 확장된다는 것을 의미한다. 전체 경로없이 git 가 호출될 때 이러한 일이 발생한다. 이 때 env 프로그램은 어떤 git 를 실행할지를 결정하기 위해서 (아마도 간접적으로) 참조된다.

rake 와 rails 명령은 자주 bundle exec 를 앞에 붙여 주는 것이 좋은데 이 경우에는 아래와 같이 매핑할 수 있다.

```
SSHKit.config.command_map[:rake] = "bundle exec rake"
SSHKit.config.command_map[:rails] = "bundle exec rails"
```

아래와 같이 매핑하는 곳에 lambda 나 Proc 객체를 적용할 수도 있다.

```
SSHKit.config.command_map = Hash.new do |hash, key|
```

```
if %i{rails rake bundle clockwork heroku}.include?(key.to_sym)
  hash[key] = "/usr/bin/env bundle exec #{key}"
else
  hash[key] = "/usr/bin/env #{key}"
end
end
```

위에서 언급한 두가지 옵션 사이에는, Capistrano 내장 task를 변경하지 않은 채, 해당 환경에서 명령들을 맵핑하는 매우 강력한 옵션이 있어야 한다. 왜냐하면, 경로가 다르거나 바이너리 파일이 다른 이름을 가지기 때문이다.

또한 예를 들어 rbenv 랩퍼(wrapper)와 같은 shim 실행명령들이 사용하는 환경들이 약간 남용될 수 있다.

```
SSHKit.config.command_map = Hash.new do |hash, key|
  if %i{rails rake bundle clockwork heroku}.include?(key.to_sym)
    hash[key] = "/usr/bin/env myproject_bundle exec myproject_#{key}"
  else
    hash[key] = "/usr/bin/env #{key}"
  end
end
```

위의 코드는 rbenv wrapper default myproject 와 같은 것을 실행했다고 가정한 것이고 이것은 인터액티브(interactive)로 긴 쉘이 필요없이 루비 환경을 정확하게 설정하는 랩퍼 바이너리(wrapper binaries)를 생성한다.

테스트하기

Capistrano의 이전 버전의 테스트 슈트는 순수한 단위 테스트이고 다양한 경우의 문제를 다루지 못했다. 특히 실제 배포 코드인 deploy.rb 파일에 있는 어떤 것도 전혀 테스트하지 못했는데, Capistrano 자체의 DSL을 실행해야 하기 때문이었고, 다른 약간 이상한 디자인 문제로 실제 레시피를 테스트하는 것이 고통스러웠다.

테스트는 Capistrano 3의 초점이 되어왔다. 통합 테스트 슈트는 Vagrant를 이용해서 머신을 부팅한 후, portable 쉘스트립트를 이용하여 어떤 시나리오를 구성하고, 그 후에 시나리오에 대한 명령을 실행하여 전형적인 리눅스 시스템에 대한 공통된 구성을 배포한다. 이것은 실행 속도가 느리지만, 이전에 우리가 줄 수 있었던 아무것도 깨지지 않았다는 보장을 더 강력하게 제공한다.

Capistrano 3는 또한 백엔드 실행을 교체할 수 있도록 지원한다. 이것은, 자신의 레시피를 테스트할 목적으로, 백엔드에서 프린터를 사용할 수 있고, 결과물이 기대했던 것과 일치하는지를 입증할 수 있으며, 또는 호출이 제대로 되었는지 아니면 기대했던 대로 호출되지 않았는지를 입증할 수 있는 a stubbed backend(?)를 사용할 수 있다.

재량에 따른 로깅

Capistrano는 on() 블록 내에서 debug(), info(), warn(), error(), fatal() 메소드를 사용할 수 있다. 이 메소드를 이용하면 기존의 로깅 인프라와 스트리밍 IO 포맷터를 이용해서 로그를 남길 수 있다.

```
# Capistrano 3.0.x
on hosts do |host|
  f = '/some/file'
  if test("[ -d #{f} ]")
    execute :touch, f
  else
    info "#{f} already exists on #{host}!"
```

```
end  
end
```

업그레이드

업그레이드 관련 사항을 상세하게 알고 싶으면 [업그레이드 문서](#)를 보기 바란다.

간단하게 말해보자면 직접 업그레이드하는 방법은 없다. 버전 2와 3은 전혀 호환성이 없다.

이것은 부분적으로는 디자인 상의 문제이기도 한데, 이전의 DSL은 올바른 것을 하는 것을 대부분 교묘하게 만들었던 곳에서, 정확하지 못했기 때문에, API를 이전 버전과의 호환성을 유지하도록 하는 것에 투자하는 것 보다는, 더 많은 기능과 더 좋은 신뢰성에 투자하는 것을 선택했다.

아래에 많은 기능들이 나열되어 있지만 중요한 것들은 내장 role의 새로운 이름이고, 디폴트로 Capistrano 3는 플랫폼을 인지하지 못한다는 것이다. 마이그레이션, asset pipeline 등과 같이 레일스 지원이 필요할 경우에는 해당 지원 파일들을 `require` 해야 한다.

기능들(Gotchas)

Rake DSL은 부가적이다.

Capistrano 2에서는 특정 task를 재정의하면 원래 구현된 내용을 대체해 버린다. 이것은 내장 task를 자신의 구현내용으로 대체하고자 하는 사람들이 사용했었다.

References:

1. [Capistrano Version 3 Release Announcement](#)

SSHKit README

> sshkit

SSHKit는 하나 또는 그 이상의 서버에서 구조화된 방식으로 명령을 실행하기 위한 블킷이다.

어떻게 동작하는가?

일반적인 사용법은 아래와 같다.

```
require 'sshkit/dsl'

on %w{1.example.com 2.example.com}, in: :sequence, wait: 5 do
  within "/opt/sites/example.com" do
    as :deploy do
      with rails_env: :production do
        rake "assets:precompile"
        runner "S3::Sync.notify"
        execute "node", "socket_server.js"
      end
    end
  end
end
```

SSHKit는 매우 낮은 수준이지만 편리한 API를 제공해 주는데, `as()`, `within()`, `with()` 는 어떤 순서라도 상관없이 중첩할 수 있고 반복할 수 있으며 스택에 푸시할 수도 있다.

이와 같이 블록 내에서 사용할 때, `as()` 와 `within()` 는 포함하는 블록을 체크하여 보호할 것이다.

`within()` 의 경우에는, 해당 디렉토리가 존재하지 않을 예리를 발생하고, `as()` 에 대해서는, `sudo su - <user> whoami` 를 호출하여 성공여부를 체크하고 실패할 경우 예리를 발생한다.

디렉토리 유무 체크는 아래와 같이 실행된다.

```
if test ! -d <directory>; then echo "Directory doesn't exist" 2>&1; false; fi
```

그리고 사용자 변경 테스트는 다음과 같이 실행된다.

```
if ! sudo su <user> -c whoami > /dev/null; then echo "Can't switch user" 2>&1; false; fi
```

디풀트 상태에서 0 이외의 상태 값을 가지는 모든 명령은 예러를 발생한다(물론 이러한 설정을 변경할 수 있다). 메시지 내용은 처리과정에서 stdout으로 작성된 모든 것을 포함한다. 1>&2 는 echo의 표준 출력을 표준 예러 채널로 리디렉트 시켜 주어 예러 발생시 메시지를 볼 수 있게 된다.

execute(:rails, "runner", ...) 와 execute(:rake, ...) 로 확장되는 runner() 와 rake() 와 같은 헬퍼 메소드들은 루비와 레일스 어플리케이션을 위한 편리한 헬퍼들입니다.

병행(Parallel)

on() 호출시 in: :sequence 옵션을 주의해서 사용한다.

```
on(in: :parallel) { ... }
on(in: :sequence, wait: 5) { ... }
on(in: :groups, limit: 2, wait: 5) { ... }
```

디풀트는 in: :parallel 상태로 실행하며 제한이 없다. 400대의 서버가 있을 경우에는 문제가 발생할 수 있기 때문에 groups 또는 sequence 상태로 실행하도록 변경하는 것이 더 좋을 것이다.

groups는, 하나의 (Git) 리소스에 대해서 너무 많은 접속을 하여 DDOS 공격을 원치 않는 경우(대량으로 Git 체크아웃을 하는)를 방지하기 위해서 고안되었다.

순차적인 실행(sequence)은 다른 사용 예 중에서도 (순차적으로) restart를 하고자 할 때 사용한다.

동기화(Synchronisation)

on() 블록은 동기화의 단위이다. 하나의 on() 블록은 모든 서버가 작업을 완료한 후에 반환하게 된다.

예를 들면,

```
all_servers = %w{one.example.com two.example.com three.example.com}
site_dir     = '/opt/sites/example.com'

# 백업 task를 시뮬레이션 해 보자.
# 어떤 서버들은 다른 것에 비해 시간이 더 걸릴 수 있다는 가정을 한다.
on all_servers do |host|
  in site_dir do
    execute :tar, '-czf', "backup-#{host.hostname}.tar.gz", 'current'
    # 이것은 다음과 같이 실행될 것이다. "/usr/bin/env tar -czf backup-one.example.com.tar.gz current"
  end
end

# 이제 이러한 백업을 가지고 무언가를 할 수 있다. 이미 모든 백업들이 존재한다고 알고 있는 상태에서이다.
# (모든 tar 명령은 성공상태로 종료되었거나 하나라도 실패한 경우 예외가 발생했을 것이다.)
on all_servers do |host|
  in site_dir do
    backup_filename = "backup-#{host.hostname}.tar.gz"
    target_filename = "backups/#{Time.now.utc.iso8601}/#{host.hostname}.tar.gz"
  end
end
```

```
    puts capture(:s3cmd, 'put', backup_filename, target_filename)
  end
end
```

명령 맵(Command Map)

프로그램 상 접근하는 SSH 세션은 인터액티브(interactive) 세션과 동일한 환경 변수를 가지지 않는다는 문제가 종종 있다.

`$PATH` 경로 상에 존재할 것으로 기대하는 실행파일을 호출할 때 문제가 종종 발생한다. `dotfile`이나 다른 환경 구성이 없는 상황에서는, `$PATH`은 계대로 설정되지 못하기 때문에 실행파일들을 해당 위치에서 발견하지 못할 수 있다.

이러한 문제를 해결하기 위해서 `with()` 헬퍼는 변수들로 구성된 해시를 취해서 환경에서 사용할 수 있도록 해 준다.

```
with path: '/usr/local/bin/rbenv/shims:$PATH' do
  execute :ruby, '--version'
end
```

이것은 다음과 같이 실행될 것이다.

```
{ PATH=/usr/local/bin/rbenv/shims:$PATH /usr/bin/env ruby --version }
```

이와는 대조적으로, 아래의 스크립트는 명령을 전혀 변경하지 못할 것이다.

```
with path: '/usr/local/bin/rbenv/shims:$PATH' do
  execute 'ruby --version'
end
```

이것은 다음과 같이 실행될 것이다.

```
ruby --version
```

(이와 같은 현상은 때때로 혼란스럽지만, 대부분이 쉘 이스케이핑(shell escaping)과 관계가 있는데, whitespace가 명령에 포함된 경우나 개행문자가 있는 경우, 입력한 내용으로 정확한 쉘 명령을 작성한 방법이 없기 때문이다.)

따라서 이런 경우를 대비해서 명령 맵을 사용하는 것을 종종 더 선호하기도 한다.

`Command` 객체를 생성할 때, 디폴트로 명령 맵을 사용할 수 있게 된다.

명령 맵은 구성 객체 상에 존재하고 워크적으로 매우 간단하다. 디폴트 키를 `factory` 블록에 명시한 해시 구조를 가진다. 예를 들면,

```
puts SSHKit.config.command_map[:ruby]
# => /usr/bin/env ruby
```

환경 설정을 확실히 하기 위해 `/usr/bin/env` 가 모든 명령 앞에 붙게 된다. 이것은 단순하게 `ruby` 를 실행할 때 일어나는 것이지만, 명확히 함으로써 사람들이 문서를 찾아 보기를 바란다.

각 명령에 대한 헤시 맵을 변경할 수 있다.

```
SSHKit.config.command_map[:rake] = "/usr/local/rbenv/shims/rake"
puts SSHKit.config.command_map[:rake]
# => /usr/local/rbenv/shims/rake
```

또 다른 방법은 명령 앞에 붙일 다른 명령을 추가하는 것이다.

```
SSHKit.config.command_map.prefix[:rake].push("bundle exec")
puts SSHKit.config.command_map[:rake]
# => bundle exec rake

SSHKit.config.command_map.prefix[:rake].unshift("/usr/local/rbenv/bin exec")
puts SSHKit.config.command_map[:rake]
# => /usr/local/rbenv/bin exec bundle exec rake
```

명령 맵을 완전히 새로운 것으로 변경할 수 있는데, 이것은 현명하지 못한 일이지만, 가능한 일이다. 예를 들면,

```
SSHKit.config.command_map = Hash.new do |hash, command|
  hash[command] = "/usr/local/rbenv/shims/#{command}"
end
```

이것은 해당 디렉토리에 실행 파일을 제공해 주지 않았던 어떤 명령도 호출할 수 없게 되지만 때로는 이러한 상황이 필요할 수 있다.

주의 사항 : 명령 맵에서 해당 키를 찾기 전에 `Command` 객체는 첫 번째 인수를 심볼화 할 것이기 때문에, 모든 키는 심볼이어야 한다.

출력물을 처리하기

```
$ rspec ./examples/b
[00:00:00] Running test -t /opt/rack/rack-repository on user@example
[00:00:00] Finished in 0.38 seconds command failed.
[00:00:00] Running user@user:git clone git@github.com:rack/rack.git /opt/rack/rack-repository on user@example.com
  [During type "apt/rake->rack-repository"]
[00:00:00] Finished in 0.437 seconds command successful.
[00:00:00] user@user:~
[00:00:00] Running test -t /opt/rack/rack-repository on user@example
[00:00:00] Finished in 0.436 seconds command successful.
[00:00:00] Running -t /opt/rack/rack-repository; then echo "Directory does not exist: /opt/rack/rack-repository" 1>&2; false; fi on user@example
[00:00:00] Finished in 0.363 seconds command successful.
[00:00:00] Running -t /opt/rack/rack-repository &! user@user:git pull & user@example
  Already up to date.
[00:00:00] Finished in 0.007 seconds command successful.
$
```

디폴로, 콘솔 출력물 포맷은 `:pretty` 로 설정되어 있다.

```
SSHKit.config.format = :pretty
```

그러나, 출력을 최소한만 보이도록 원할 경우 `:dot` 포맷으로 지정하면 명령 실행의 성과에 따라 빨간색이나 초록색으로 표시될 것이다.

포맷을 사용하지 않고 직접 \$stdout으로 출력하기 위해서는 아래와 같이 지정할 수 있다.

```
SSHKit.config.output = $stdout
```

출력물의 정보 표시(Output Verbosity)

디폴트 상태에서는 `capture()` 와 `test()` 호출 시 로그가 남지 않는다. 이 명령들은 종종 백엔드 task에서 환경 설정을 체크하기 위해서 사용된다. `Logger::DEBUG` 의 `Command` 인스턴스에 `verbosity` 옵션을 지정할 수 있다. 디폴트 구성은 `SSHKit.config.output_verbosity`로 변경할 수 있고 디폴트는 `Logger::INFO` 이다.

현재 `Logger::WARN`, `ERROR`, `FATAL` 는 사용하지 않는다.

연결 풀링(Connection Pooling)

SSHKit는 모든 `on()` 블록에 대해 새로운 SSH 연결을 시도하는 것에 대한 비용을 줄이기 위해서, 단순한 연결 풀(디폴트로 활성화됨)을 사용한다. 용도와 네트워크 상황에 따라서, 상당한 시간 절약을 할 수 있다. 한 테스트에서는, 기본적인 `cap deploy` 명령이 SSHKit의 최근 버전에 추가된 연결 풀 덕분에 15-20초 정도 더 빠르게 실행되었다.

연결 상태를 활성 상태로 유지하기 위해서, 기존의 풀링된 연결은 30초 이상 사용되지 않을 경우 새로운 연결로 대체될 것이다. 이와 같은 태팅아웃은 아래와 같이 변경할 수 있다.

```
SSHKit::Backend::Netssh.pool.idle_timeout = 60 # seconds
```

연결 풀이 문제를 일으킨다고 생각될 때는, `idle_timeout` 을 0으로 설정하므로써 연결 풀 기능을 해제할 수도 있다.

```
SSHKit::Backend::Netssh.pool.idle_timeout = 0 # disabled
```

알려진 문제들

- 느리거나 타이아웃된 연결을 처리하지 못함
- 느리거나 중단된 원격 명령을 처리하지 못함
- 백그라운드 작업 처리 가능성이 없음
- 환경 처리기능 없음(sshkit는 이에 대한 걱정을 할 필요 없다)
- -host- 객체에 임의의 속성을 추가할 수 없음(서버에 -roles- 을 저장하거나 -on()- 블록에서 유용하게 사용할 수 있는 메타데이터 등)
- 로그나 경고 메시지를 보여주는 기능이 없음(로그 메시지를 출력으로 넘기는 것이 작동한다) 하나의 로그 객체는, 전역적으로 사용할 수 있어야 하며, 로그 메시지에 대한 관심을 가지는 포맷터들이 인식할 수 있는 `logMessage` 형 객체를 발송할 것이다.
- `verbosity` 를 켜어 할 수 없음. 명령은 자신에 대한 `logger::LEVEL` 을 가져야 한다. 사용자가 생성한 것은 교수준의 정보를 보여줘야 하고, `-as()` 와 `within()`로부터 권한 체크를 위해 자동으로 생성되는 명령들은 저수준의 정보를 보여줘야 한다.
- `-execute()` 류 명령들이 0이 아닌 종료 상태에 대해서 예러를 발생할지 여부를 결정해야 함. 아마도 비슷한 이름을 가진 !(bang)-마소드들은 예러를 발생해야 한다.(아마도 `test()` 는 예러 발생시키지 않고 `-execute()` 를 실행하는 방법이고 `-execute()` 류 명령들은 항상 예러를 발생해야 한다.)

- `-SSHKit::Config::Formatter = :pretty` 라고 설정할 수 있고, 메소드 `setter`가 `-SSHKit::Config::Output`을 기준 출력 스트림을 래핑(wrapping)하는 정확한 포맷터 클래스 인스턴스로 업데이트 할 수 있게 한다면 멋질 것이다.
- 내부적으로 디버깅하기 위한 "trace" 레벨이 없음. 디버그 레벨은 클라이어트 측 디버깅을 위해서 예비해 두어야 하고, 네트워크 연결, 종료, 타임아웃에 대한 로그를 남기 위해서 내부적으로 `trace(int -1)`을 사용해야 한다.
- 파일 업로드나 다운로드를 위한 메소드가 없음. 또는 원격 파일로 문자열을 저장하거나 가져오는 메소드가 없음.
- 네트워크 연결은 중단할 수 없음. 백엔드 `abstract` 클래스는 비어 있지만 다르게 방법으로 구현하여 변경할 수 있는 `cleanup` 메소드를 포함해야 한다.
- 연결 풀링 가능성이 없음. NetSSH 백엔드의 `connection` 메소드는 연결 택토리에서 연결 객체들을 찾아 볼 수 있도록 쉽게 변경할 수 있어야 하고, 이로써 여러 개의 `on()` 블록을 실행할 때 0.5초 정도 시간을 절약 할 수 있게 된다.
- 문서화(YARD 형태)가 필요하다.
- 모든 명령을 알려진 쉘로 래핑(wrapping)할 수 있어야 함. 즉, `execute('uptime')`은 일관된 쉘 실행 환경을 유지하기 위해서 `sh -c 'uptime'`으로 변환되어야 한다.
- `-Host.new('user@ip:port')` 와 같이 호출하는 것을 수용하는 적당한 호스트 파서(parser)가 없음. 이것은 `-user@host:port`을 디코딩하지만 IP 주소는 디코딩하지 못할 것이다.
- Net::SSH가 `IOError`를 발생(인증 실패와 같이)할 때 예리를 잡아 낼 수 있어야 하고, `ConnectionFailed`와 같은 예리를 다시 발생할 수 있어야 한다.

References:

1. [SSHKit Readme](#)

SSHKit 사용 예제

다른 계정으로 명령 실행하기

```
on hosts do |host|
  as 'www-data' do
    puts capture(:whoami)
  end
end
```

디폴트 환경 변수로 실행하기

```
SSHKit.config.default_env = { path: '/usr/local/libexec/bin:$PATH' }
on hosts do |host|
  puts capture(:env)
end
```

다른 디렉토리에서 명령 실행하기

```
on hosts do |host|
  within '/var/log' do
    puts capture(:head, '-n5', 'messages')
  end
end
```

특정 환경 변수로 명령 실행하기

```
on hosts do |host|
  with rack_env: :test do
    puts capture("env | grep RACK_ENV")
  end
end
```

로깅 메소드를 이용하여 출력하기

```
on hosts do |host|
  f = '/some/file'
  if test("[ -d #{f} ]")
    execute :touch, f
  else
    info "#{f} already exists on #{host}!"
  end
end
```

SSHKit.config.output_verbosity의 현재 로그 레벨에 따라 `debug()`, `info()`, `warn()`, `error()`, `fatal()` 메소드를 사용할 수 있다.

다른 계정으로 다른 디렉토리에서 명령 실행하기

```
on hosts do |host|
  as 'www-data' do
    within '/var/log' do
      puts capture(:whoami)
      puts capture(:pwd)
    end
  end
end
```

이것은 아래와 같은 결과를 보여 준다.

```
www-data
/var/log
```

주의사항 : 이 예제는 약간 오해의 소지가 있다. 즉, `www-data` 계정은 쉘이 정의되어 있지 않아서 이 계정으로 변경할 수 없다.

디스크로부터 파일 업로그하기

```
on hosts do |host|
  upload! '/config/database.yml', '/opt/my_project/shared/database.yml'
end
```

주의사항 : `upload!()` 메소드는 `within()`, `as()` 등의 값을 받지 않는다. 이 문제는 라이브러리가 성숙해짐에 따라 해결될 것이지만 현재로서는 아직 그대로다.

스트림으로부터 파일 업로드하기

```
on hosts do |host|
  file = File.open('/config/database.yml')
  io   = StringIO.new('....')
  upload! file, '/opt/my_project/shared/database.yml'
  upload! io,   '/opt/my_project/shared/io.io.io'
end
```

IO 스트리밍하는 것은 'cat' 하기 보다는 원자를 업로드하는데 유용하다. 예를 들면,

```
on hosts do |host|
  contents = StringIO.new("ALL ALL = (ALL) NOPASSWD: ALL")
  upload! contents, '/etc/sudoers.d/yolo'
end
```

이것은 "echo(cat, '...?...', '> /etc/sudoers.d/yolo')"와 같이 정확한 이스케이핑 순서를 기술해야 하는 수고를 덜어 준다.

주의사항 : `upload!()` 메소드는 `within()`, `as()` 등의 값을 받지 않는다. 이 문제는 라이브러리가 성숙해

짐에 따라 해결될 것이지만 현재로서는 아직 그대로다.

디렉토리내의 파일을 업로드하기

```
on hosts do |host|
  upload! '.', '/tmp/mypwd', recursive: true
end
```

이 경우에 `recursive: true` 옵션은 `Net::SCP`, `Net::SFTP`에서 사용 가능한 옵션과 같은 것이다.

전역(global) SSH 옵션 설정하기

이 옵션들은 호스트마다 옵션을 달리 지정할 수 있다.

```
Netssh.configure do |ssh|
  ssh.connection_timeout = 30
  ssh.ssh_options = {
    keys: %w(/home/user/.ssh/id_rsa),
    forward_agent: false,
    auth_methods: %w(publickey password)
  }
end
```

다른 group ID로 명령 실행하기

```
on hosts do |host|
  as user: 'www-data', group: 'project-group' do
    within '/var/log' do
      execute :touch, 'somefile'
      execute :ls, '-l'
    end
  end
end
```

위의 설정에서, 생성된 파일은 `www-data` 계정과 `project-group` 그룹이 소유하게 된다.

`umask` 구성 옵션과 함께 사용하면, 로그인을 공유하지 않은 상태에서 팀 멤버간에 배포 스크립트를 쉽게 공유 할 수 있다.

중첩된 디렉토리에서 작성하기

```
on hosts do
  within "/var" do
    puts capture(:pwd)
    within :log do
      puts capture(:pwd)
    end
  end
end
```

이것은 아래와 같이 출력된다.

```
/var/  
/var/log
```

디렉토리 경로는 `File.join()` 를 이용해서 조합된다. 이 메소드는 경로 앞뒤에 오는 슬래시에 신경쓰지 않고도 각 부분을 정확하게 연결해서 조합해 준다. 이것은 `File.join()` 이 코드를 실행하는 머신에서 동작하기 때문에 오해의 여지가 있다. 만약 윈도우즈 운영환경에서 실행된다면 명령을 받는 머신에 따라 경로가 부정확하게 조합될 수 있다.

백그라운드에서 task 수행하기

```
on hosts do  
  within '/opt/sites/example.com' do  
    background :rails, :server  
  end  
end
```

이것은 `nohup /usr/bin/env rails server > /dev/null &` 와 같이 실행하는데, 레일스 프로세스를 백그라운드에서 실행하도록 하여 파일 시스템을 `nohup` 로그 파일로 더럽히지 않게 된다.

[역주] `nohup` - 리눅스, 유닉스에서 쉘스크립트파일 (*.sh)을 데몬형태로 실행시키는 프로그램

주의사항 : `background()` 메소드에 `sleep 5` 문자열을 넘겨 주면 계대로 동작하지 않을 것이다. 이것은 명령을 처리하는 규칙에 따른 것이어서, `background(:sleep, "5")` 와 같이(즉, `sleep`이 명령이고, 5는 인수) 호출해야 한다.

더 주의해야 할 사항 : `background() task`가 어떤 상황에서 특정 명령을 `nohup ... &` 로 감싸면, SSH 세션이 종료되어도 프로그램이 멈출 것이다.

host 블록에 대해서 신경쓰지 않기

```
on hosts do  
  # [host] 블록 인수는 옵션이다.  
  # 블록 인수로 넘기지 않으면 블록내에서는 nil 값을 가지게 된다.  
end
```

모든 출력을 `/dev/null` 로 리디렉트하기

```
SSHKit.config.output = File.open('/dev/null')
```

매우 간단한 formatter 클래스 구현하기

```
class MyFormatter < SSHKit::Formatter::Abstract  
  def write(obj)  
    case obj.is_a? SSHKit::Command  
      # Do something here, see the SSHKit::Command documentation
```

```
end
end
end

SSHKit.config.output = MyFormatter.new($stdout)
SSHKit.config.output = MyFormatter.new(SSHKit.config.output)
SSHKit.config.output = MyFormatter.new(File.open('log/deploy.log', 'wb'))
```

특정 호스트에 대한 비밀번호 지정하기

```
host = SSHKit::Host.new('user@example.com')
host.password = "hakme"

on host do |host|
  puts capture(:echo, "I don't care about security!")
end
```

뭔가 잘 못 됐을 때 실행 후 에러 발생하기

```
on hosts do |host|
  execute!(:echo, '"Example Message!" 1>&2; false')
end
```

이것은 `#message "Example Message!"` 와 함께 'SSHKit::CommandFailed' 예외를 발생할 것이고 명령은 취소될 것이다.

에러를 발생하지 않은 채 실패(fail)할 수 있는 테스트하기 또는 명령 실행하기

```
on hosts do |host|
  if test "[ -d /opt/sites ]"
    within "/opt/sites" do
      execute :git, :pull
    end
  else
    execute :git, :clone, 'some-repository', '/opt/sites'
  end
end
```

`test()` 명령은 `execute()` 와 동일하게 동작하지만 0이 아닌 값으로 명령이 종료할 때 `false` 값을 반환한다 (`man 1 test` 가 하듯이). 논리값을 반환하기 때문에 블록내에서 제어 흐름의 방향을 지정할 때 사용될 수 있다.

호스트 property에 따라 각 호스트마다 다른 작업 하기

```
host1 = SSHKit::Host.new 'user@example.com'
host2 = SSHKit::Host.new 'user@example.org'

on hosts do |host|
  target = "/var/www/sites/"
  if host.hostname =~ /org/
    target += "dot.org"
  else
```

```
target += "dot.com"
end
execute! :git, :clone, "git@git.{host.hostname}", target
end
```

가장 쉽게 호스트에 연결하기

```
on 'example.com' do |host|
  execute :uptime
end
```

이것은 `example.com` 호스트명을 `SSHKit::Host` 객체로 변환해서 해당 호스트에 대한 정확한 구성 상태를 가져온다.

명령 맵핑 없이 명령 실행하기

호출하려는 명령이 스페이스 문자를 포함할 경우 맵핑되지 않을 것이다.

```
Command.new(:git, :push, :origin, :master).to_s
# => /usr/bin/env git push origin master
# (also: execute(:git, :push, :origin, :master))

Command.new("git push origin master").to_s
# => git push origin master
# (also: execute("git push origin master"))
```

이것은 (`if` 와 `test` 와 같은) 쉘 내장 명령을 접근할 때 사용할 수 있다.

heredoc 와 함께 명령 실행하기

위의 기능을 확장한 것으로 아래와 같이 명령을 작성할 수 있다.

```
c = Command.new <<-EOCOMMAND
if test -d /var/log
then echo "Directory Exists"
fi
EOCOMMAND
c.to_s
# => if test -d /var/log; then echo "Directory Exists"; fi
# (also: execute <<- EOCOMMAND.....))
```

주의 사항 : 스크립트를 한줄로 표시하는 로직은 세련되어 보이지 않지만 모든 테스트에서 제대로 동작한다. 핵심 사항은 `if` 가 `/usr/bin/env if` 로 맵핑되지 않는다는 것인데, 이것은 문법상의 오류를 유발하지 않는다.

Rake 사용하기

`Rakefile`에 아래와 같은 것을 둘 수 있다.

```
require 'sshkit/dsl'
```

```
SSHKit.config.command_map[:rake] = "./bin/rake"

desc "Deploy the site, pulls from Git, migrate the db and precompile assets, then restart"
task :deploy do
  on "example.com" do |host|
    within "/opt/sites/example.com" do
      execute :git, :pull
      execute :bundle, :install, '--deployment'
      execute :rake, 'db:migrate'
      execute :rake, 'assets:precompile'
      execute :touch, 'tmp/restart.txt'
    end
  end
end
```

◀ ▶

DSL 없이 사용하기

Coordinator 가 모든 호스트를 Host 객체로 변환해 줄 것이다.

```
Coordinator.new("one.example.com", SSHKit::Host.new('two.example.com')).each in: :sequence
  puts capture :uptime
end
```

◀ ▶

./lib/sshkit/dsl.rb 파일을 참고할 수 있는데, 여기에서 위와 거의 똑같은 코드를 볼 수 있지만 on() 메소드에 대해서 구현한 것이다.

Host properties 속성 이용하기

v0.6.6 부터 구현됨

```
servers = %w{one.example.com two.example.com
            three.example.com four.example.com}.collect do |s|
  h = SSHKit::Host.new(s)
  if s.match /(one|two)/
    h.properties.roles = [:web]
  else
    h.properties.roles = [:app]
  end
end

on servers do |host|
  if host.properties.roles.include? (:web)
    # 웹서버에 관련된 작업을 함.
  elsif host.properties.roles.include? (:app)
    # 어플리케이션 서버에 관련된 작업을 함.
  end
end
```

SSHKit::Host#properties 는 [OpenStruct](#)이며 어쨌던 유효성 검증이 되지 않기 때문에 이에 관해서는 개발자에게 달리 있거나 라이브러리가 이 메카니즘에 의미나 규칙을 추가해 주어야 한다.

로컬 명령 실행하기

on 을 run_locally 로 대체한다.

```
run_locally do
  within '/tmp' do
    execute :whoami
  end
end
```

References:

1. [Usage Examples of SSHKit](#)

Capistrano 2.x에서 업그레이드하기

(1) Gemfile을 업데이트한다.

```
gem 'capistrano', '~> 3.0', require: false, group: :development
```

레일스를 배포할 경우에는, `capistrano-rails` 와 `capistrano-bundler` 라이브러리가 필요하다. Capistrano 3.x에서는 레일스와 Bundler 통합이 제거되었다.

```
group :development do
  gem 'capistrano-rails', '~> 1.1', require: false
  gem 'capistrano-bundler', '~> 1.1', require: false
end
```

rvm, rbenv, chruby 중에 선호하는 루비 버전 관리자에 대한 지원을 추가한다.

```
group :development do
  gem 'capistrano-rvm', '~> 0.1', require: false
  gem 'capistrano-rbenv', '~> 2.0', require: false
  gem 'capistrano-chruby', github: 'capistrano/chruby', require: false
end
```

(2) 처음부터 프로젝트 `capify` 할 것을 권한다. 이전 버전의 경의들은 별도의 디렉토리로 옮긴다.

```
$ mkdir old_cap
$ mv Capfile old_cap
$ mv config/deploy.rb old_cap
$ mv config/deploy/ old_cap # --> only for multistage setups
```

이제 `capify` 한다.

```
$ cap install
```

(3) Capistrano 3.x는 디폴트로 `multistage`를 지원한다. 따라서 `capify` 후

`config/deploy/production.rb` 와 `config/deploy/staging.rb` 파일이 디폴트로 생성될 것이다. 하나의 `stage`만이 필요한 경우에는, 이 파일들을 제거하고 `config/deploy.rb` 파일에 `stage(:production)`과 `servers`를 선언한다.

(4) `config/deploy/production.rb` 와 `config/deploy/staging.rb` 파일을 업데이트해서 관련 데이터를 지정한다. 이전 설정(`old_cap/deploy/`)로부터 더 많은 `stages`를 추가할 수도 있다.

(5) 게이트웨이 서버 셋팅이 `set :gateway, "www.capify.org"`로 지정되어 있었다면, 아래와 같이 업그레이드해야 한다.

```
require 'net/ssh/proxy/command'
set :ssh_options, proxy: Net::SSH::Proxy::Command.new('ssh mygateway.com -W %h:%p')
```

또는 서버별로 `ssh_options` 을 지정할 수 있다.

(6) 이제 이전 `deploy.rb` 파일(물론 `Capfile`도 함께, 그러나 Capistrano 2.x에서는 이 파일을 변경하지 않았었다.)을 리팩토링할 필요가 있다. 파라미터들(`set :deploy_to, "/home/deploy/#{application}"` or `set :keep_releases, 4` 등)을 `config/deploy.rb` 파일로, task들은 `Capfile`로 옮긴다.

중요 : `repository` 옵션은 `repo_url`로, `default_environment` 옵션은 `default_env`로 이름이 변경되었다.

`use_sudo` 와 `normalize_asset_timestamps` 파라미터는 더 이상 필요없다는 것을 주목하자.

(7) 이전에 `deploy_to` 옵션을 사용하지 않고 `/u/apps/your_app_name`으로 배포했다면, 한가지 더 변경해야 한다. 이제 디폴트 배포 경로가 `/var/www/app_name`로 변경되었기 때문에 업그레이드 후 config 가 깨질 것이다. 아래와 같이 커스텀 `deploy_to` 옵션을 선언해 주면 해결된다.

```
set :deploy_to, "/u/apps/#{fetch(:application)}"
```

그러나, `/u/apps` 는 앱을 저장하기에 최선의 장소는 아니므로 나중에 변경할 것을 권장한다.

(8) `Capfile`을 수정해서 필요한 addon들(rbenv/rvm, bundler, rails)을 추가한다.

```
require 'capistrano/rails'
require 'capistrano/bundler'
require 'capistrano/rbenv'
```

(9) 이제 새로 작성한 config로 배포해 보자. 본 가이드에 빠진 내용을 발견하면 기여해 주기 바란다.

일반적인 추천사항

ENV 변수를 작성하지 말고 DSL을 사용하라

```
run <<-CMD.compact
cd -- #{latest_release} &&
RAILS_ENV=#{rails_env.to_s.shellescape} #{asset_env} #{rake} assets:precompile
CMD
```

대신에

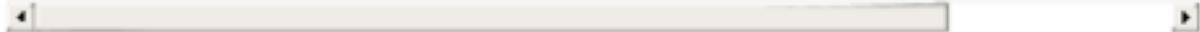
```
on roles :all do
  within fetch(:latest_release_directory) do
    with rails_env: fetch(:rails_env) do
      execute :rake, 'assets:precompile'
    end
  end
end
```

와 같이 사용하는 것이 더 좋다.

주의 : DSL이 인식하기 위해서는 `on` 블록으로 감싸줘야하는데, 이 때 `within` 블록이 필요하다.

호출 당 하나의 `with` 블록을 가질 수 있다. 하나 이상의 `env` 설정이 필요한 경우는 아래와 같이 `with` 블록을 작성한다(하나의 맵으로 넘겨 준다).

```
on roles :all do
  within fetch(:latest_release_directory) do
    with rails_env: fetch(:rails_env), rails_relative_url_root: '/home' do
      execute :rake, 'assets:precompile', env: {rails_env: fetch(:rails_env), rails_relat
        end
      end
    end
```



References:

1. [Upgrading from v2.x.x](#)

부록

: 이 책을 이해하는데 도움이 될 만한 내용

공동집필에 참여하기

NPM 설치

npm v0.6.3 부터는 node.js 설치 시에 함께 설치되기 때문에 별도로 npm을 설치할 필요가 없다. 따라서 시스템에 node.js가 설치되어 있지 않다면 <http://www.nodejs.org>를 참고하여 설치한다.

| 한글로 작성된 설치 안내문을 원하면 [Node.js-설치-및-개발환경-세팅하기](#)를 참고하기 바란다.

로컬 머신에 gitbook 설치하기

: 터미널 쉘에서 아래와 같이 명령을 실행한다.

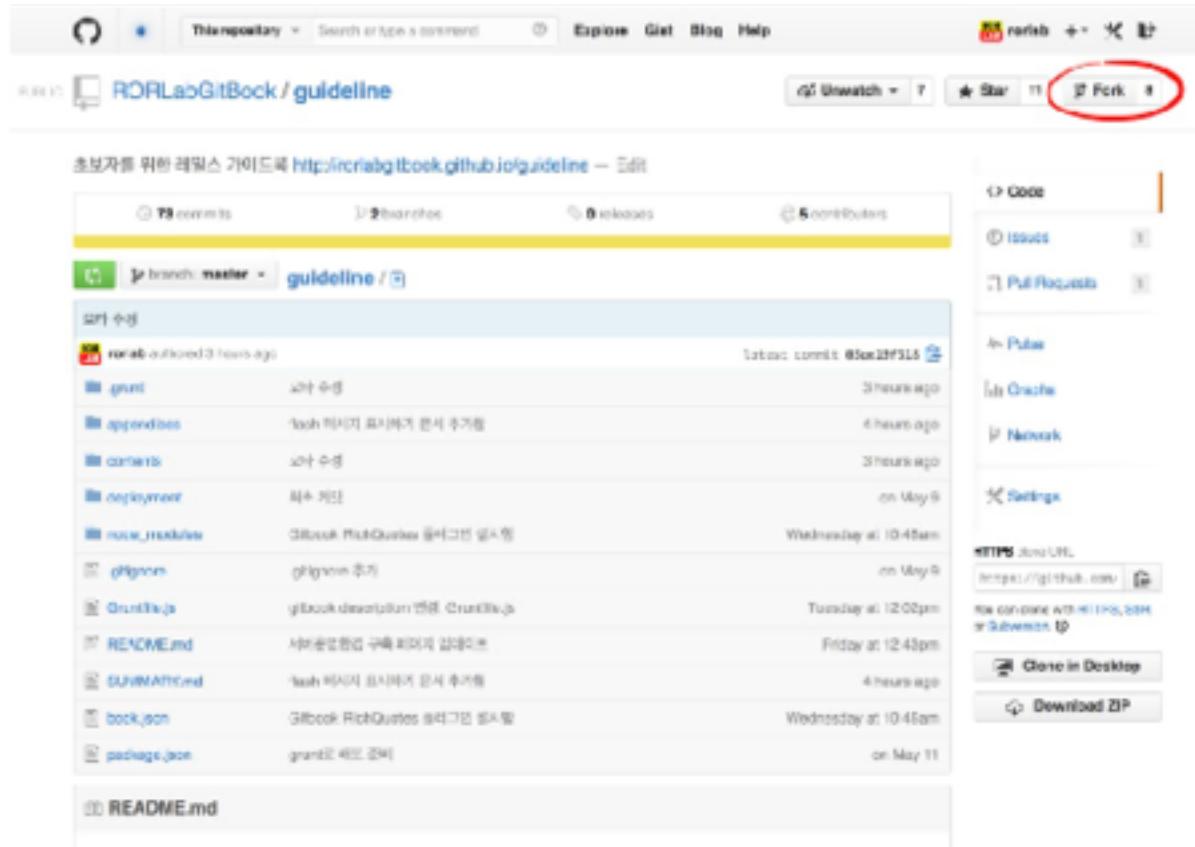
```
$ npm install gitbook -g
```

로컬 머신에 gitbook editor 설치하기

: 리눅스/맥/윈도우용 [다운로드](#) 후 설치하면 된다.

가이드북 Github 저장소 Forking 하기

Github에 로그인한 상태에서 <https://github.com/RORLabGitBook/railsguidebook>로 접속한 후 우측 상단에 있는 Fork 버튼을 클릭한다.



저장소 clone 하기

: github에서 본인 계정으로 Fork된 저장소를 clone한다.

```
$ git clone https://github.com/[user-account]/rcafe.git  
$ cd rcafe
```

문서작업

GitBook Editor를 실행한 후 봉금 전에 clone 받은 폴더를 열고 문서작업을 한다.

작업내용 확인

문서 작업을 종료한 후, 터미널 셸에서 아래와 같이 명령을 실행하여 작업한 내용이 제대로 브라우저에서 보이는지를 확인한다. (브라우저에서 <http://localhost:4000> 주소로 접속한다.)

```
(rcafe) $ gitbook serve
```

이때 아래와 같은 에러가 발생하면, `$ ulimit -n 1200` 명령을 실행한 후 다시 실행하면 된다. ([참고문서](#))

```
$ gitbook serve  
Press CTRL+C to quit ...  
  
Starting build ...  
Successfully built !  
  
Starting server ...  
Serving book on http://localhost:4000  
Error: EMFILE: Too many opened files.
```

커밋 후 푸시하기

작업한 내용을 커밋하고 푸시한다.

```
(rcafe) $ git add .  
(rcafe) $ git commit -m "작업내용을 기술"  
(rcafe) $ git push
```

작업 내용 중에 삭제된 파일이 있는 경우에는 `git add .` 대신에 `git add -A` 와 같이 실행하여 삭제된 파일도 커밋에 반영되도록 한다.

Pull Request 하기

에러가 없이 작성된 것이 확인되면 아래와 같이 github의 본인의 계정의 저장소에서 아래와 같이 `pull request` 버튼을 클릭하면 본인의 커밋 내용을 원조 저장소로 메지 요청을하게 된다.

fork

hschoidr / guideline
Updated 3 days ago · README · 1 branch · 72 commits · 0 releases · 6 contributors

Watchers 1 · Stars 0 · Forks 10

초보자를 위한 라일스 가이드북 <http://orcalabgitbook.github.io/guideline> — Edit

branch: master · guideline ·

This branch is 3 commits ahead and 0 commits behind master

무라 수집

• **revert** authored 3 hours ago Total: commit 45c697d18

• **gram** 무라 수집 3 hours ago

• **appendices** book 메시지 표시하기 문서 추가 4 hours ago

• **contents** 무라 수집 4 hours ago

Code · Pull Requests · Compare · Pulse · Simple · Network · Settings

Pull Requests

이상으로 간단하게 Gitbook으로 공동집필하는 방법을 소개했다.

액티브레코드란?

액티브레코드(ActiveRecord)란, 관계형데이터베이스(RDBMS)의 테이블을 객체로 연결(ORM : Object Relational Mapping)해서 네이티브 데이터베이스 SQL을 사용하지 않고도 데이터를 조작할 수 있도록 다양한 메소드를 제공해 준다.

MVC 디자인 패턴으로 개발하는 레일스에서 액티브레코드는 M(Model)에 해당한다. 이 책의 [프로젝트](#) 따라하기에서 사용하는 Post 모델 클래스의 소스를 보면 아래와 같다.

```
# app/models/post.rb
class Post < ActiveRecord::Base
  belongs_to :bulletin
  mount_uploader :picture, PictureUploader
end
```

위의 소스코드에서 Post 모델 클래스는 ActiveRecord 모듈의 Base 클래스(`ActiveRecord::Base`)로부터 상속을 받는다. 이 말은, ORM을 통해서 데이터베이스 테이블을 쉽게 조작할 수 있도록, Base 클래스에 정의된 메소드를 Post 모델에서도 모두 그대로 사용할 수 있다는 의미이다.

Post 모델 클래스가 정의되고 실제로 데이터베이스에 posts 테이블이 존재하면 액티브레코드의 다양한 메소드를 바로 사용해서 데이터를 조작할 수 있게 되는 것이다.

즉, Post 모델의 빈 객체를 생성하기 위해서는 아래와 같이 코드를 작성할 수 있다.

```
post = Post.new
```

레일스 콘솔을 통해서 확인하면 아래와 같다.

```
$ bin/rails console
Loading development environment (Rails 4.1.1)
irb(main):001:0> post = Post.new
=> #<Post id: nil, title: nil, content: nil, created_at: nil, updated_at: nil, bulletin_i
```

출력 결과에서 알 수 있듯이, post 변수는 Post 모델 클래스의 new라는 클래스 메소드를 이용하여 데이터가 없는 빈 인스턴스 객체를 할당받게 된다. => 다음에 보이는 반환값은 post 객체에 포함되는 속성 목록을 보여주고 있다. 이러한 각 속성들의 값은 객체지향적 프로그래밍의 기본적인 문법을 이용하여 쉽게 얻을 수 있다.

```
irb(main):002:0> post.title
=> nil
```

ImageMagick 설치하기

맥 OSX에서 설치하기

: homebrew가 설치되어 있는 상태에서 아래와 같이 한 줄 명령으로 설치할 수 있다.

```
$ brew install ghostscript imagemagick
```

우분투에서 설치하기

아래와 같이 한 줄 명령으로 설치가 된다.

```
$ sudo apt-get install ghostscript imagemagick
```

윈도우에서 설치하기

ImageMagick 윈도우 버이너리 릴리스를 [다운로드](#) 받아 설치한다. 그리고, [Ghostscript for Windows](#)를 다운로드(32비트용|64비트용) 받아 설치한다.

| Ghostscript는 PDF 파일에 대한 썬네일 이미지를 생성하기 위해 사용된다.

Postgres (or PostgreSQL) 설치하기

맥 OSX에 설치하기

두가지 방법으로 설치할 수 있다.

1. homebrew를 이용
2. Postgre.app을 이용

1. homebrew를 이용하여 설치하기

```
$ brew update  
$ brew install postgresql
```

자세한 내용은 [여기](#)를 참고하기 바란다.

2. Postgre.app을 이용하여 설치하기

맥에서는 Postgre.app을 설치하면 바로 PostgreSQL에 접속할 수도 있다.

Download and Install PostgreSQL database from Postgres.app which provides PostgreSQL in a single package to easily get started with Mac OS X. After the installation, open Postgres located under Applications to start PostgreSQL database running. Find out the PostgreSQL bin path and append to `~/.bashrc` for accessing commands through the shell.

```
$ echo 'PATH="/Applications/Postgres.app/Contents/Versions/9.3/bin:$PATH"' >> ~/.bashrc  
$ source ~/.bashrc
```

레일스 프로젝트를 위한 준비

- 레일스 프로젝트에서 사용한 PostgreSQL 유저를 생성한다.

```
$ createuser -P -d -e sampleuser  
Enter password for new role:  
Enter it again:  
CREATE ROLE sampleuser PASSWORD 'ad5afdbd364af0c0efaf11183c3454f56c52' NOSUPERUSER CRE,
```

- config/database.yml 을 아래와 같이 수정한다.

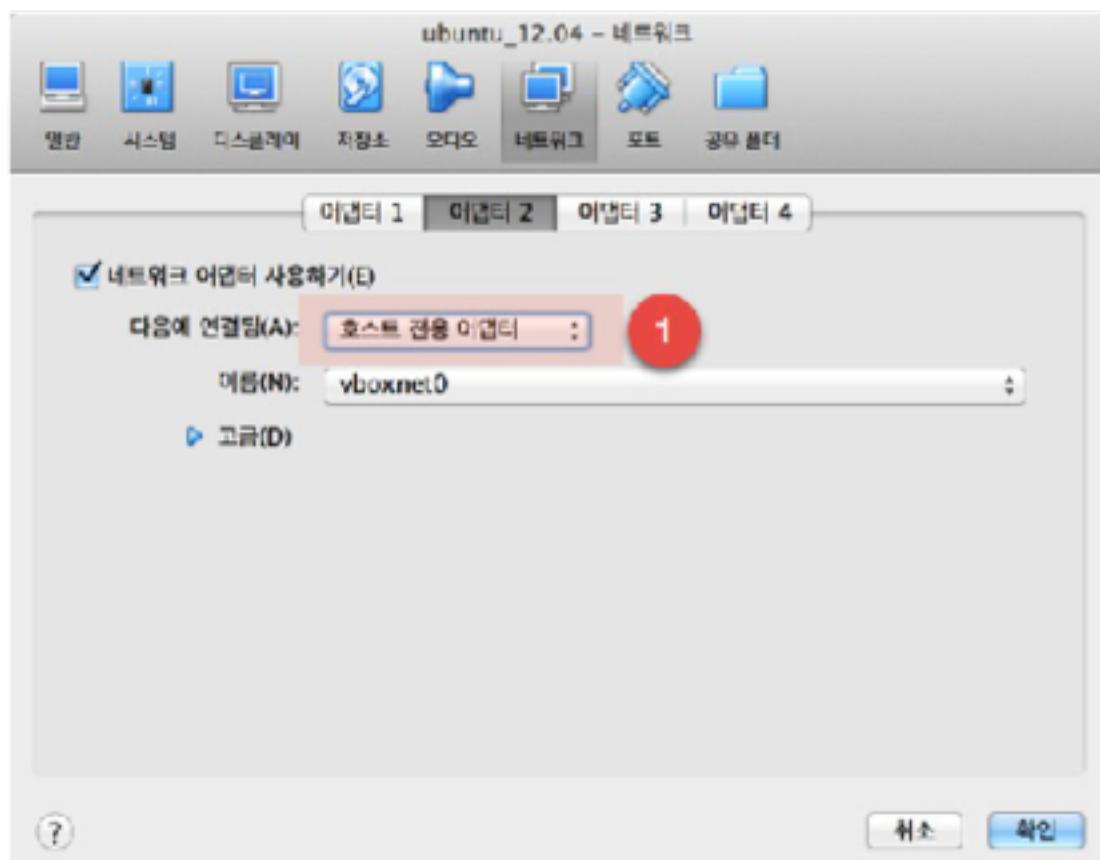
```
production:  
  adapter: postgresql  
  encoding: unicode  
  database: <database-name>  # 원하는 대로 DB 명을 정한다.
```


우분투 12.04 서버 환경 구축하기

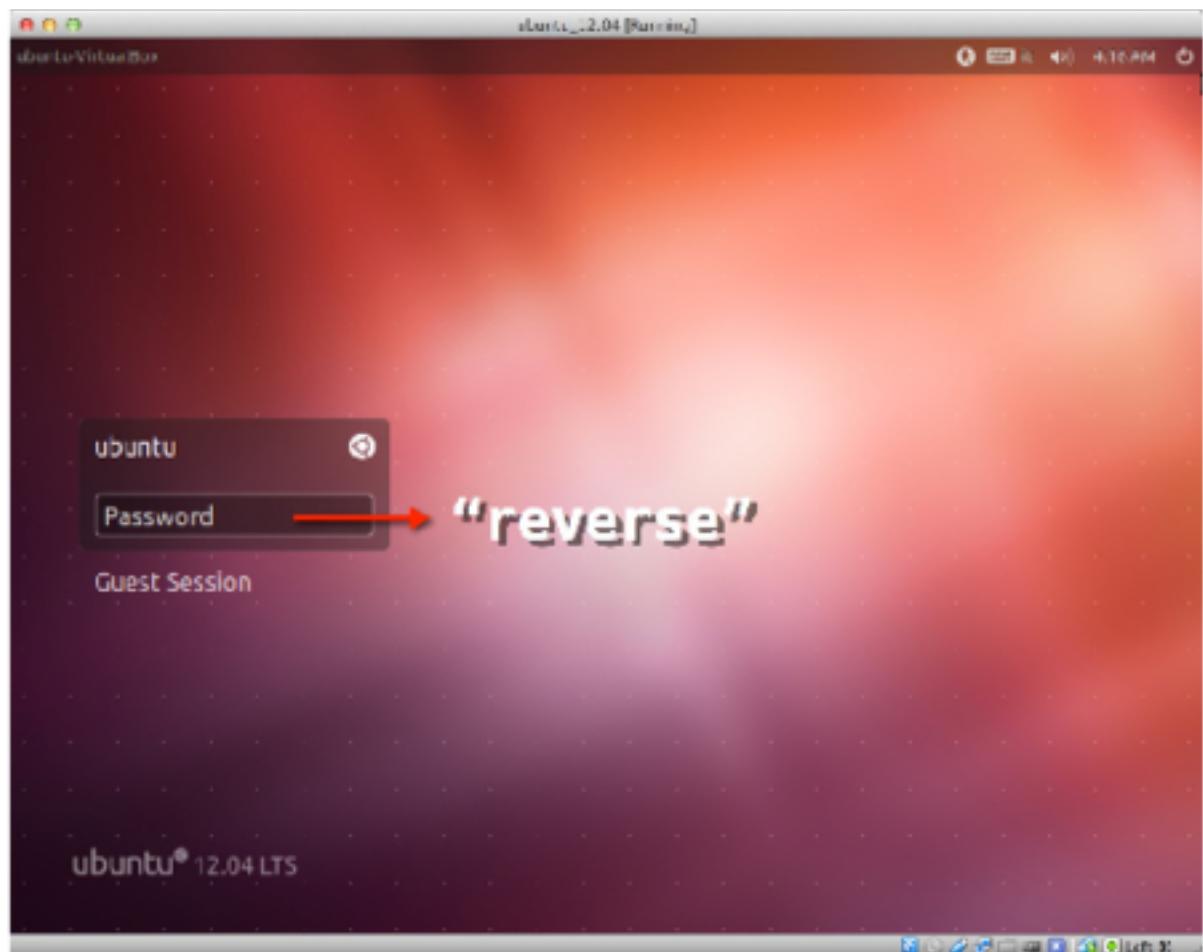
로컬 메신에 가상머신 룰인 오라클사의 [VM VirtualBox](#)를 설치하고 VirtualBoxes – Free VirtualBox® Images를 다운로드 받아 우분투 12.04 버전을 설치한다.

다운로드 받은 파일 7-zip 파일로 압축되어 있어서 [7-zip.org](#)를 방문하여 프로그램을 다운로드 받아 설치하고 압축해제하면 ubuntu_12.04.vbox 와 ubuntu_12.04.vdi 파일 두 개가 나타난다. 이미 VirtualBox 가 설치되어 있기 때문에 ubuntu_12.04.vbox 파일을 더블 클릭하면 자동으로 VirtualBox에 로드된다. 시작버튼을 클릭할 때 확장팩을 설치하라는 메시지가 나타나면 [다운로드](#) 후 마우스로 더블 클릭하면 자동으로 설치된다.

그리고 설정에서 네트워크의 어댑터를 하나 더 추가하여 호스트 전용 어댑터로 지정한다.



디폴트로 등록되어 있는 관리자 계정은 `ubuntu`이고 비밀번호는 `reverse`이다.



서버셋팅

방금 VirtualBox로 설치한 우분투 서버에 ubuntu 계정으로 로그인한다. 그리고 터미널을 열고 아래와 같이 ssh 서버를 설치한다.

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install openssh-server -y
```

그리고 할당된 ip 를 알기 위해서 아래와 같이 명령을 실행한다.

```
ubuntu@ubuntu-VirtualBox:~$ ip addr
```

```
ubuntu@ubuntu-VirtualBox:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:27:2d:48:05 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
        inet6 fe80::a00:27ff:fed4:4805/64 scope link
            valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:27:95:3c:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.59.103/24 brd 192.168.59.255 scope global eth1
        inet6 fe80::a00:27ff:fe95:3cc3/64 scope link
            valid_lft forever preferred_lft forever
ubuntu@ubuntu-VirtualBox:~$
```

이제 로컬 터미널을 열고 아래와 같이 우분투 가상 서버에 ssh로 접속한다.

```
$ ssh ubuntu@ubuntu.vb
```

위에서 사용한 도메인 `ubuntu.vb`는 가상도메인으로 로컬 마신의 `/etc/hosts` 파일에 `192.168.59.103` 을 `ubuntu.vb`로 지정해 놓았다.

서버 설치과정

```
# System update
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get update

# To add a language to Ubuntu using the Command line:
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install language-pack-ko language-pack-ko-base -
ubuntu@ubuntu-VirtualBox:~$ sudo vi /etc/default/locale
LANG="ko_KR.UTF-8"
LANGUAGE="ko_KR:ko:en_US:en"
ubuntu@ubuntu-VirtualBox:~$ sudo dpkg-reconfigure locales

# Build-essential
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get -y install git curl build-essential openssl libssl-dev

# Nginx
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install -y nginx

# MySQL
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install -y mysql-server mysql-client libmysqlclient-dev

# ImageMagick
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get -y install libmagickwand-dev imagemagick

# Nodejs
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get -y install nodejs
```

배포용 계정 만들기

```
ubuntu@ubuntu-VirtualBox:~$ sudo adduser deployer
```

위의 명령으로 deployer 계정과 그룹이 생성되고 deployer 사용자는 deployer 그룹에 속하게 된다. 그리고 /home/deployer 디렉토리가 계정 홈디렉토리로 생성된다.

이제 deployer 계정에 루트권한을 주기 위해서 admin 그룹으로 등록한다.

```
ubuntu@ubuntu-VirtualBox:~$ sudo addgroup admin  
ubuntu@ubuntu-VirtualBox:~$ sudo usermod -a -G admin deployer
```

References:

1. [OpenSSH Server](#)
2. [How to SSH to a VirtualBox guest externally through a host?](#)
3. [How to SSH into a VirtualBox Linux guest from your host machine](#)
4. [Ubuntu: How to Change the Computer Name](#)
5. [Error message when I run sudo: unable to resolve host \(none\)](#)
6. [Fixing "No such file or directory" locale errors](#)
7. [ubuntu 사용자 계정 root 권한 주기](#)

우분투 14.04 서버 세팅하기 (Virtual Box)

: 레일스 애플리케이션을 배포하기 위해서는 서버의 준비가 필요가 하다. 실제 운영 서버에 배포 테스트하는 것 보다는 Virtual Box와 같은 가상머신에 서버를 설치해 놓고 이를 이용하는 것이 여러가지로 유용하다. 이를 위해서 Virtual Box를 이용하여 Ubuntu 서버의 최신 버전인 14.04을 설치하고 웹서비스를 위한 서버 환경을 셋업하는 과정을 소개한다.

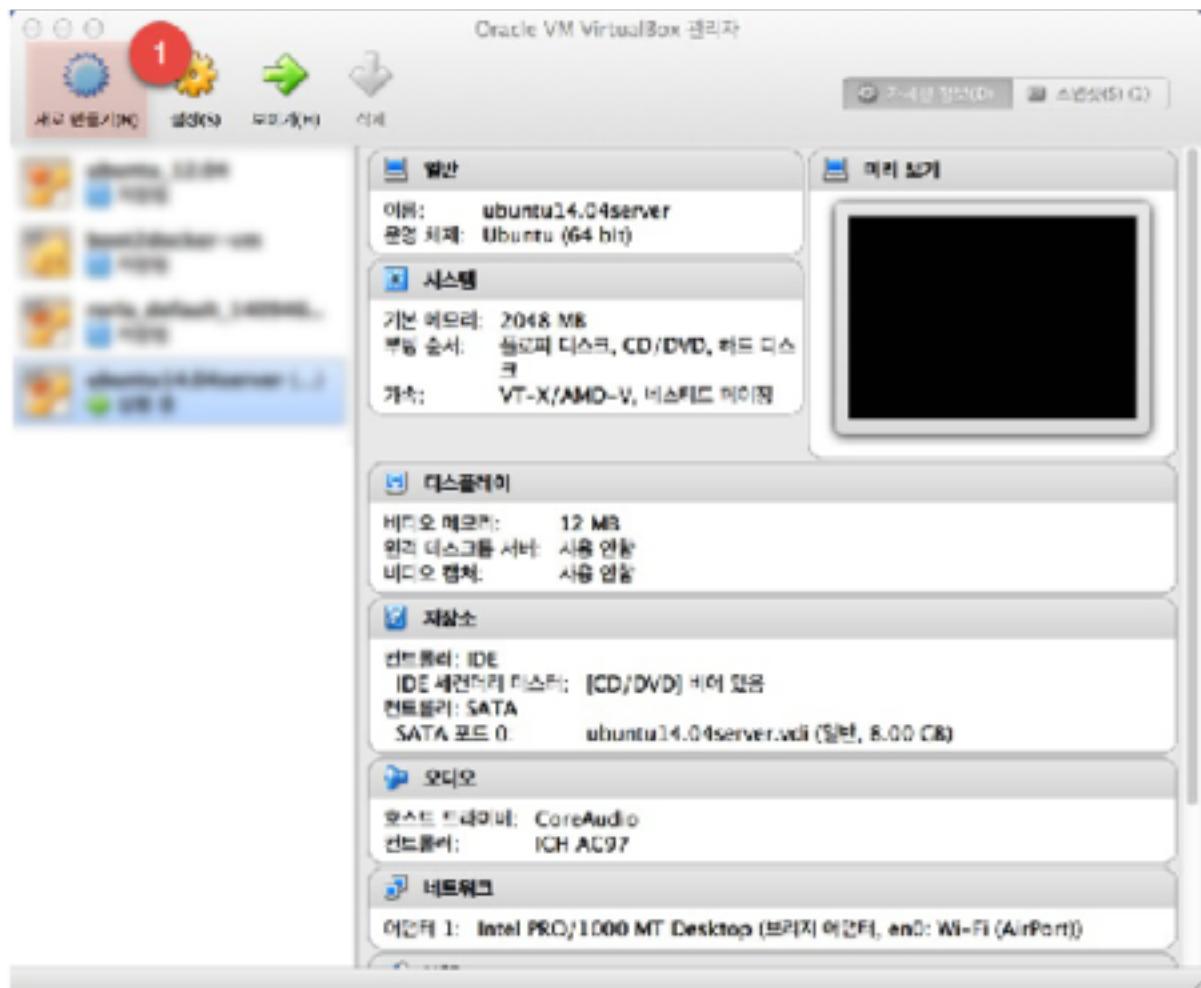
이미지 다운로드 : <http://www.ubuntu.com/download/server>를 방문하여 우분투 서버 버전(14.04.x LTS)을 다운로드 받는다.

로컬 머신에 각자의 OS에 맞는 Virtual Box가 설치되어 있다고 가정하고 아래와 같이 진행한다.

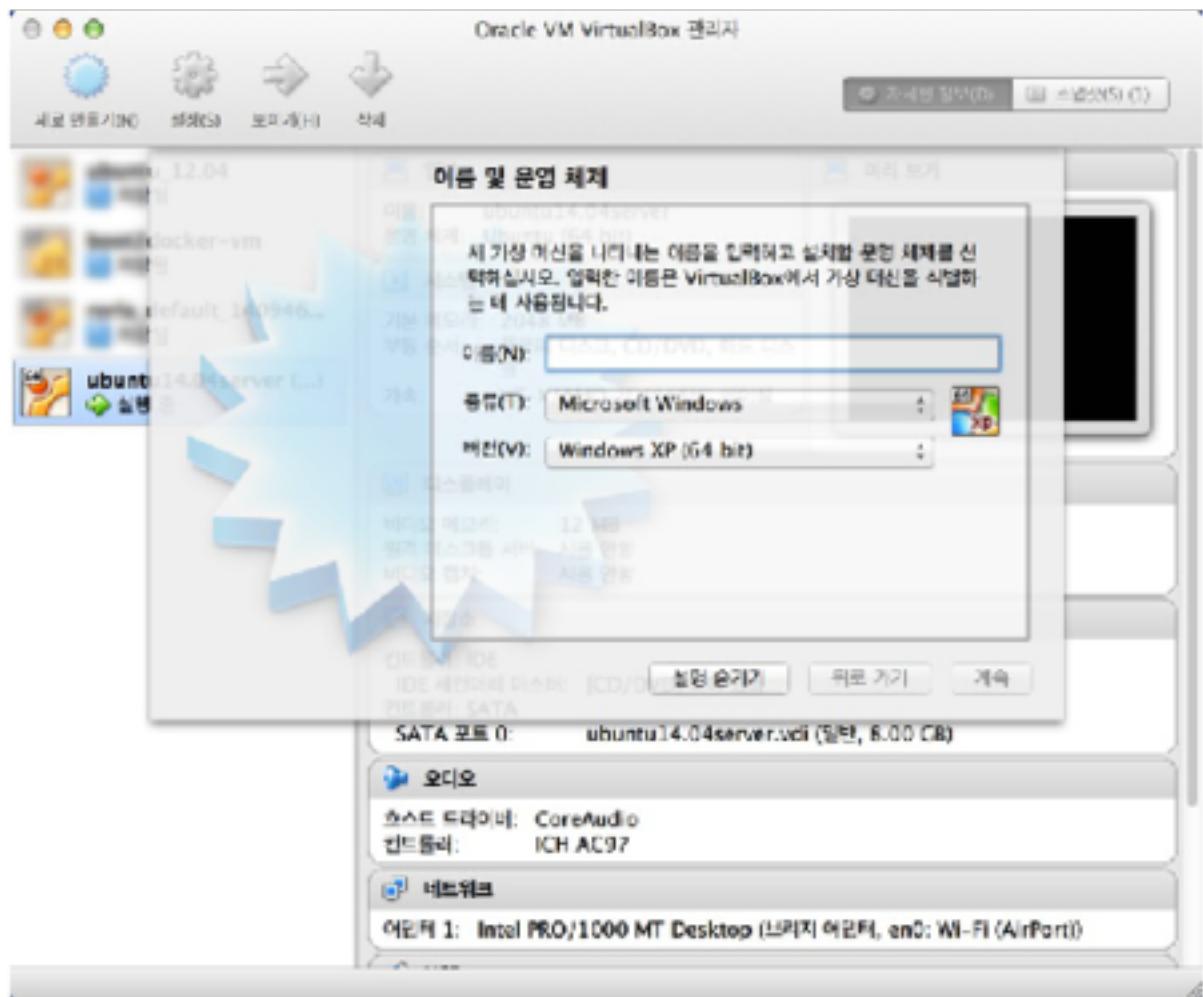
Virtual Box 다운로드: <https://www.virtualbox.org/wiki/Downloads>

가상머신 생성

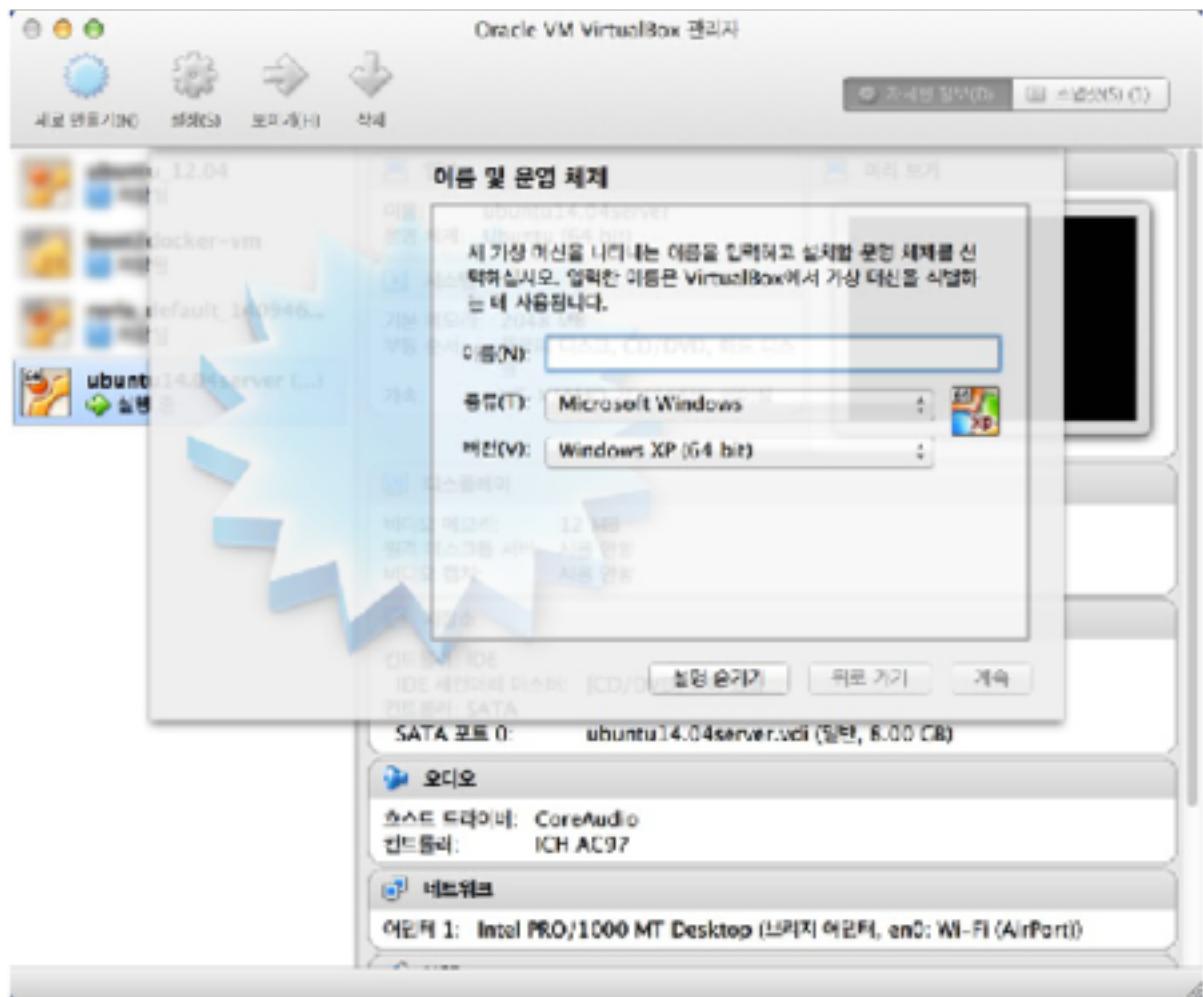
1. 아래의 창에서 1번 을 클릭하여 새로운 가상머신을 생성한다.



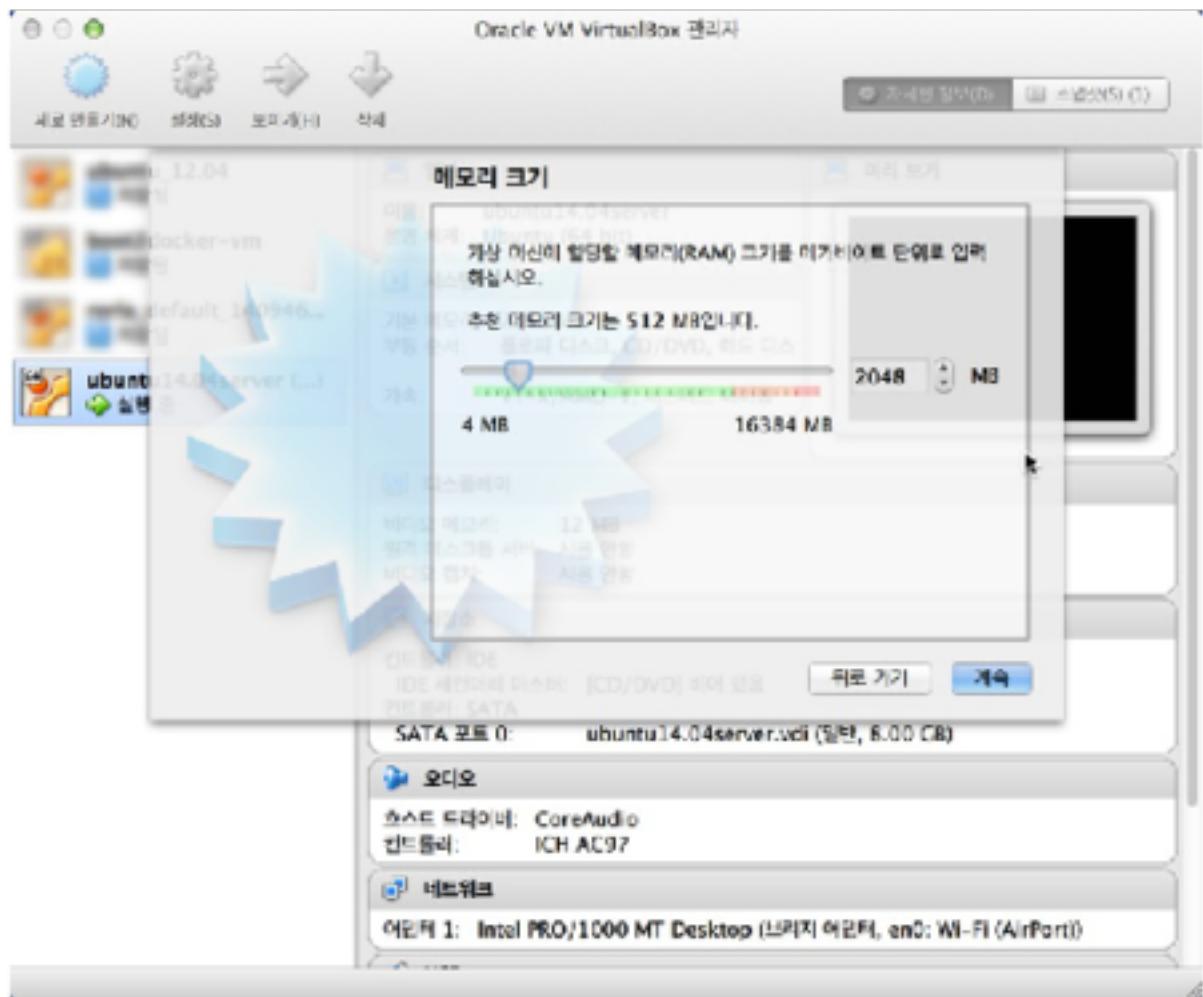
2. 운영체제 이름을 설정한다.



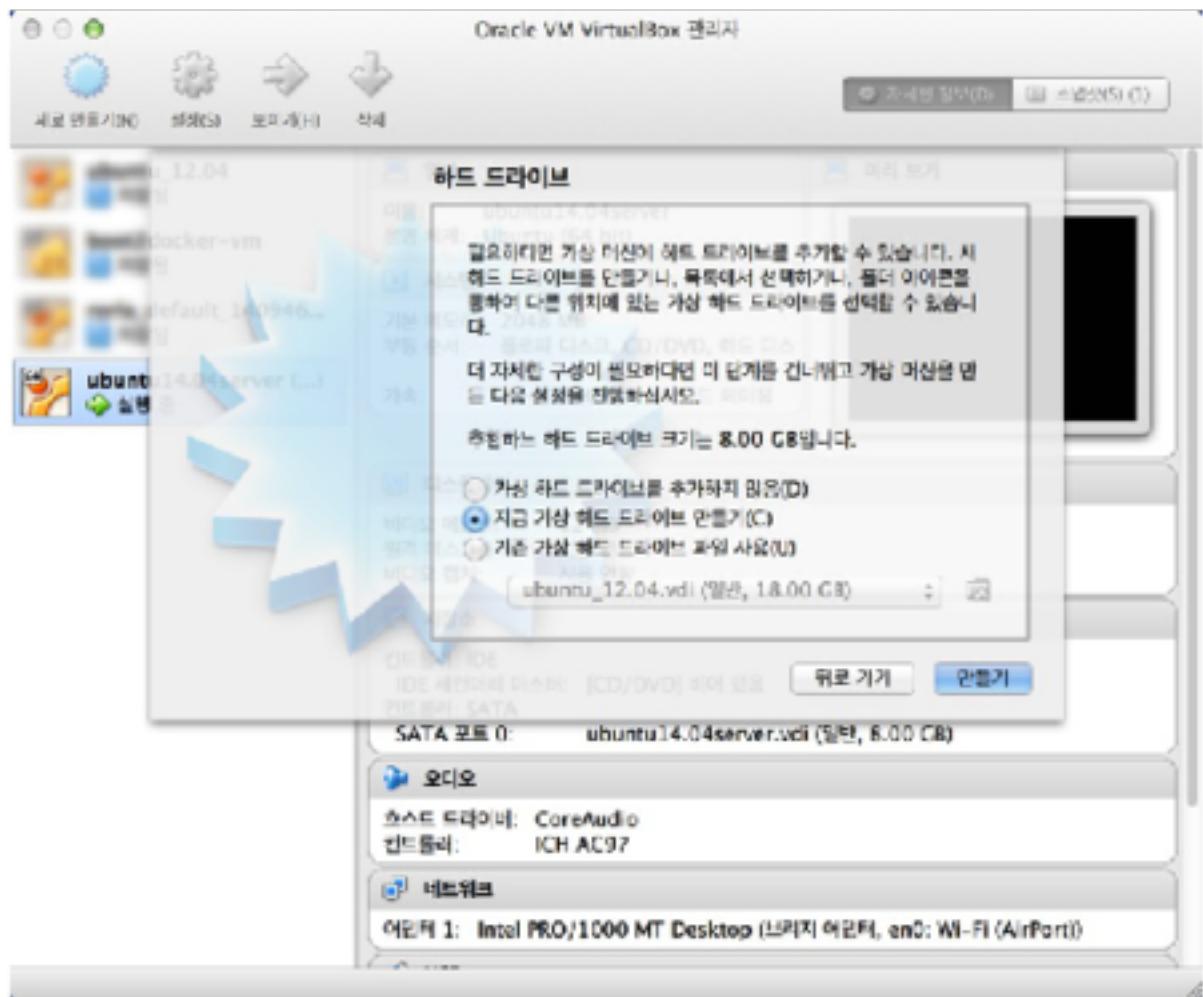
3. 이름에 ubuntu라고 입력하면 자동으로 버전이 Ubuntu (64 bit)로 지정된다. 이름은 원하는 것으로 변경할 수 있다.



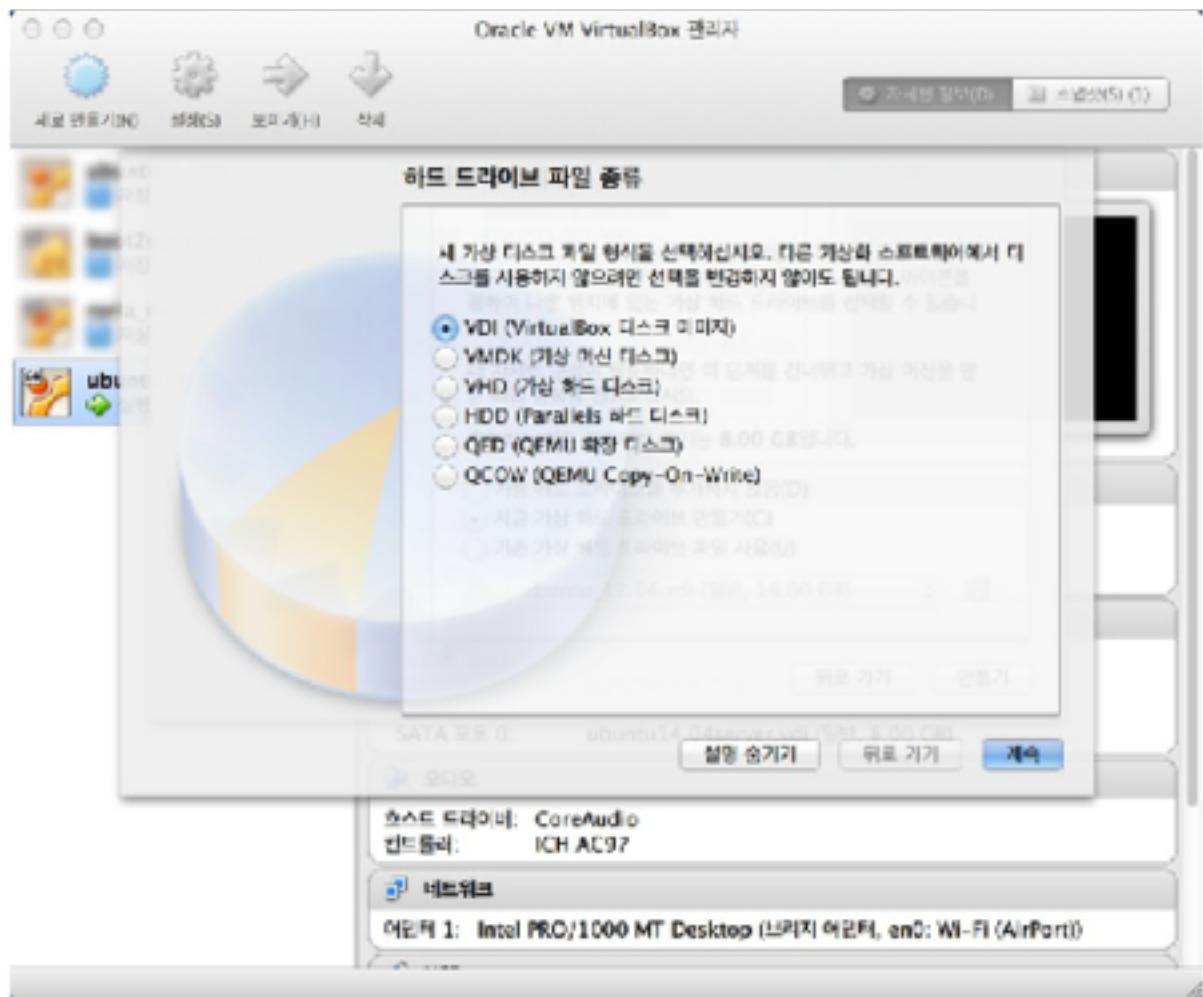
4. 메모리는 1,024 MB이상이 권장된다.



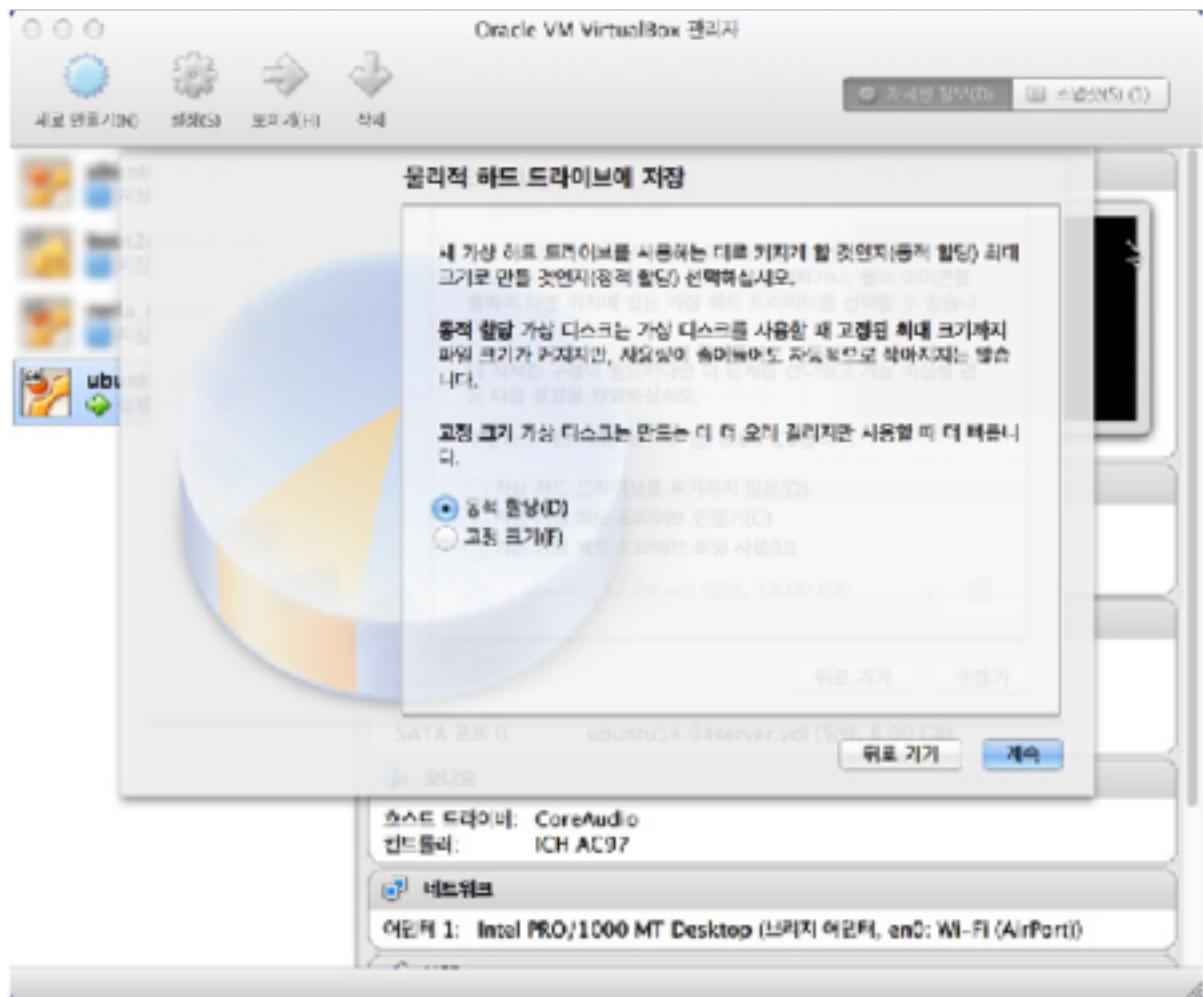
5. 하드 드라이브는 기본설정으로 두번째 항목으로 만들기 한다.



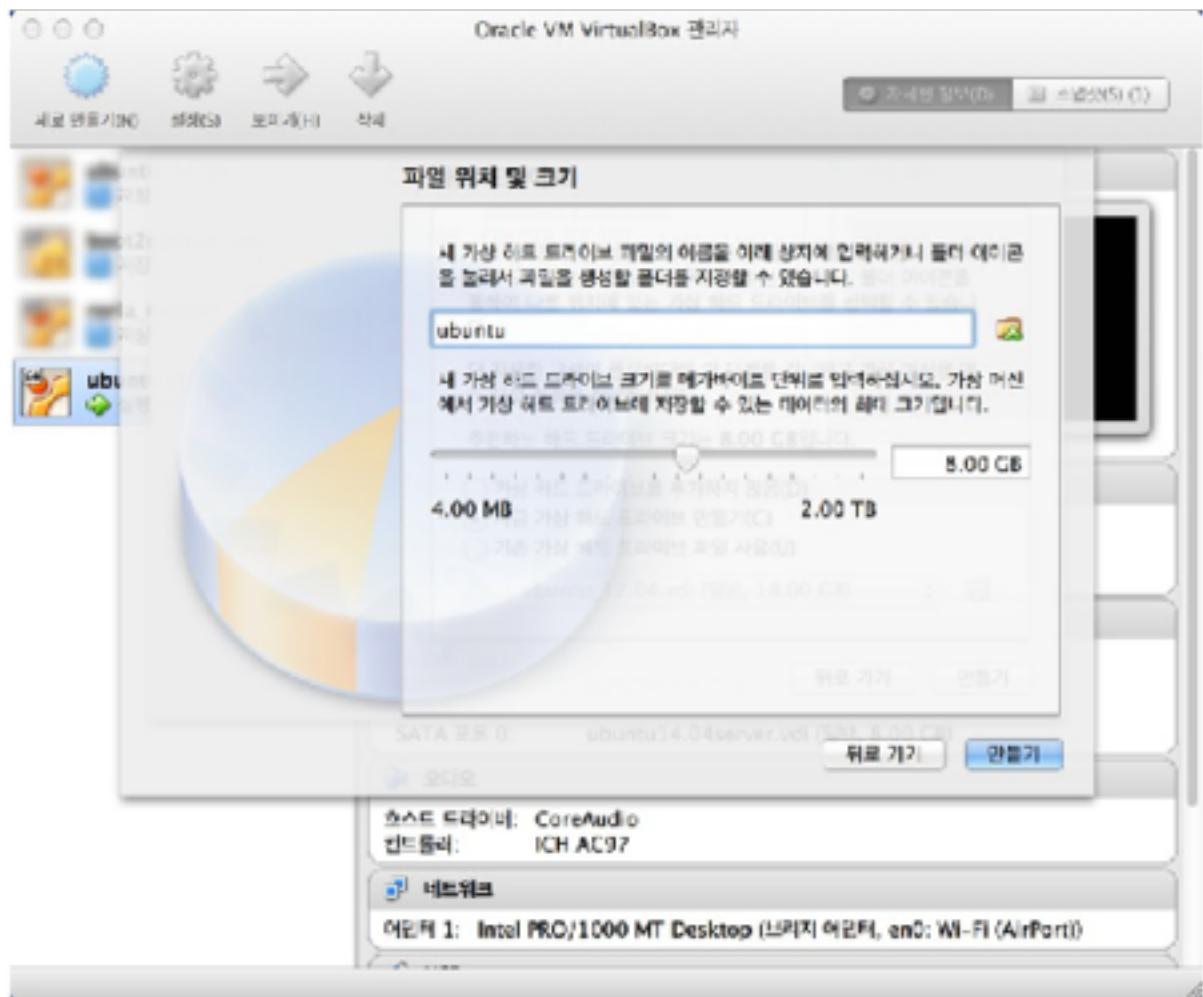
6. 하드 드라이브 파일 종류는 기본값인 첫번째 항목으로 계속한다.



7. 물리적 하드 드라이브에 저장에서는 기본값인 첫번째 항목(동적 할당)으로 계속한다.

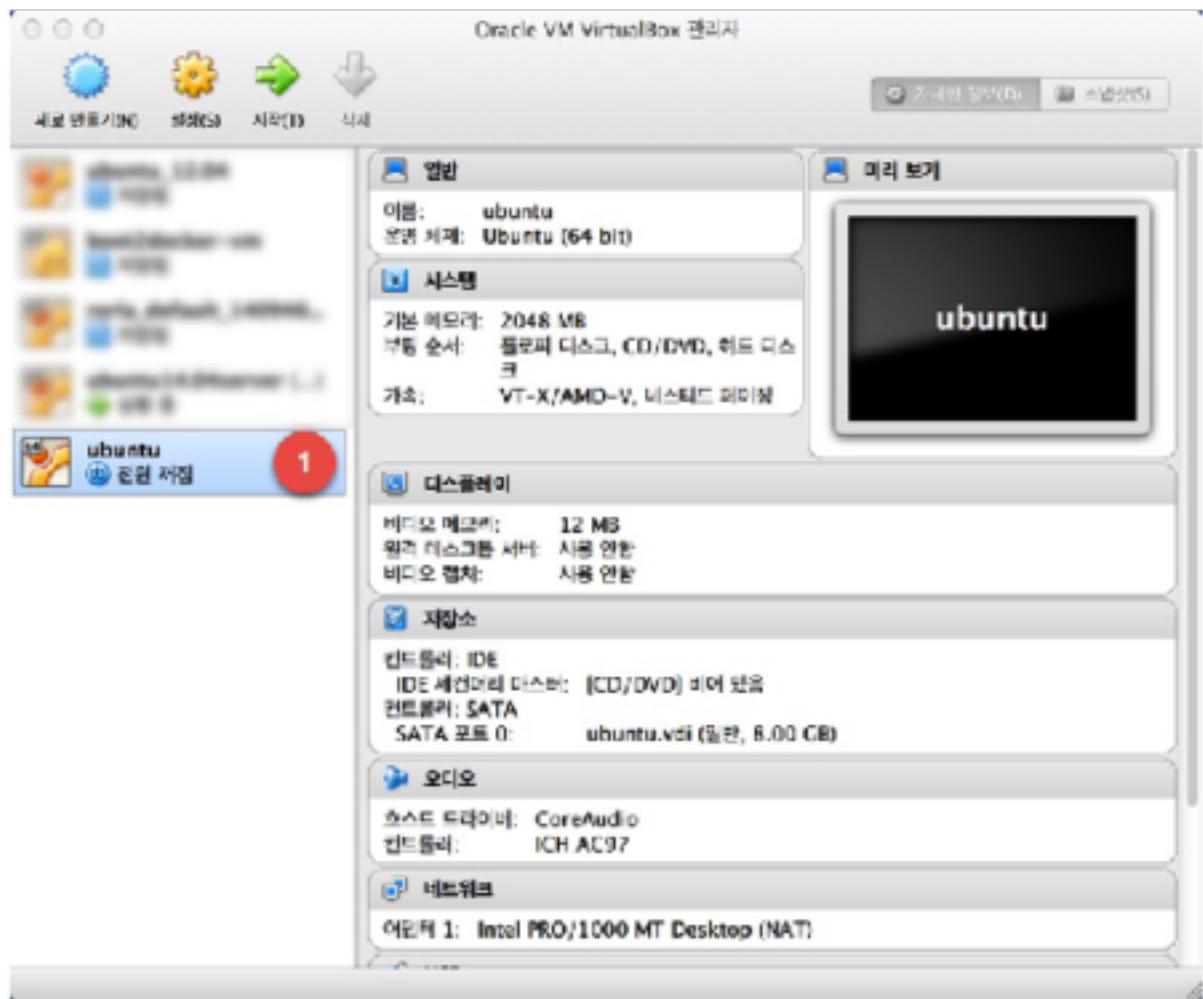


8. 파일 위치 및 크기는 기본값인 8.00 GB 상태로 만들기한다.

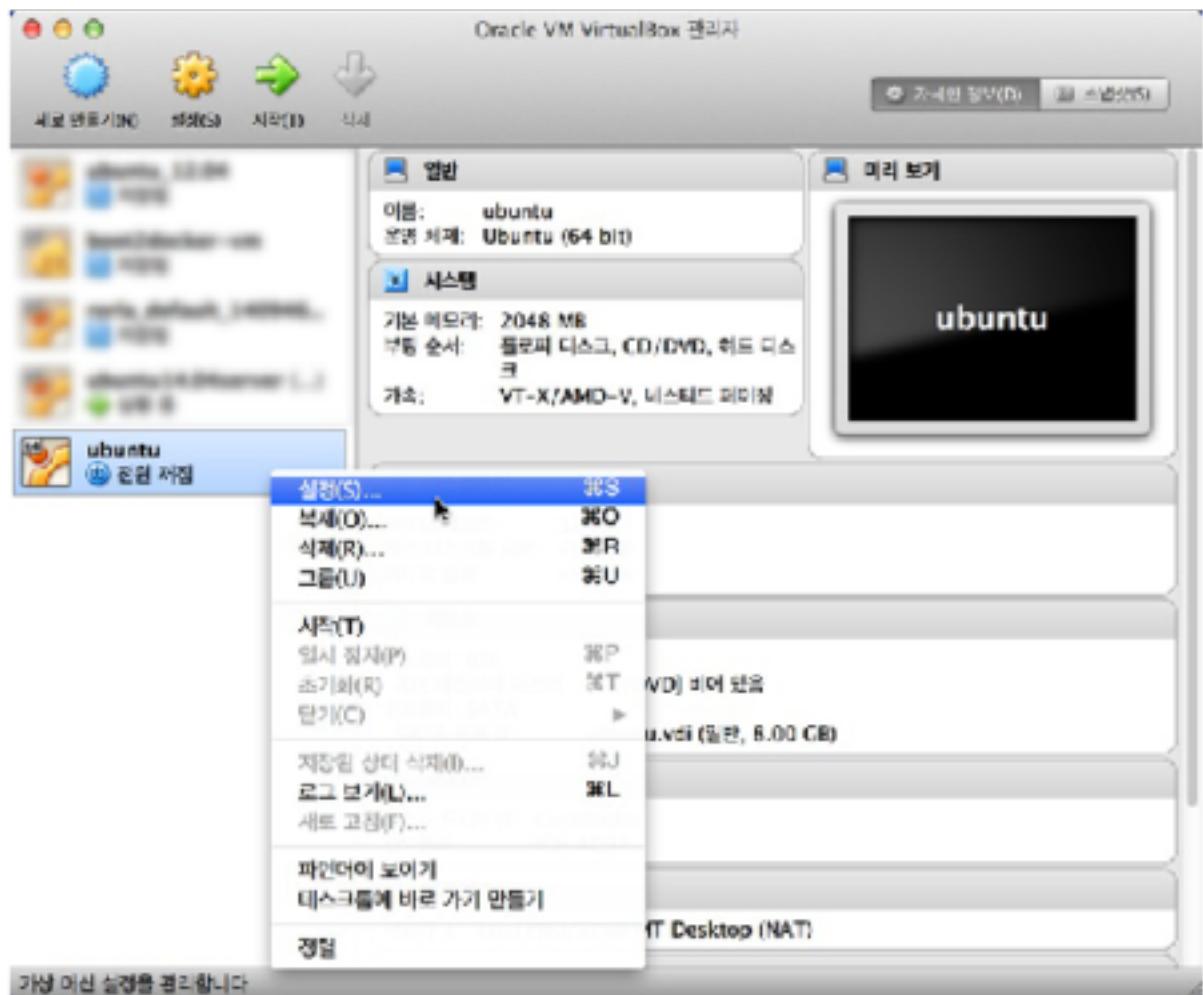


9. 가상머신 설정

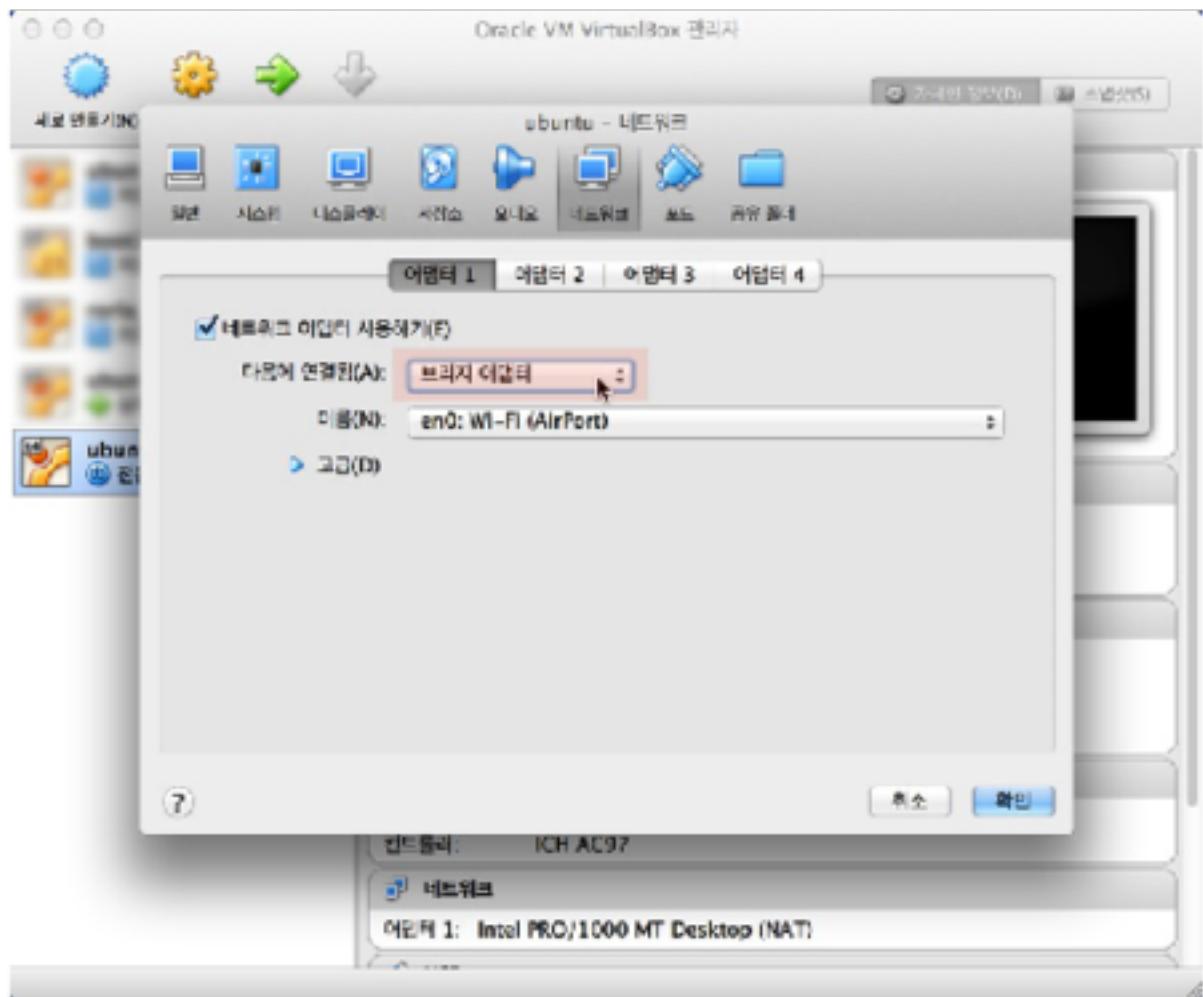
: 방금 생성된 1번 항목에서 마우스의 오른쪽 버튼을 클릭하여...



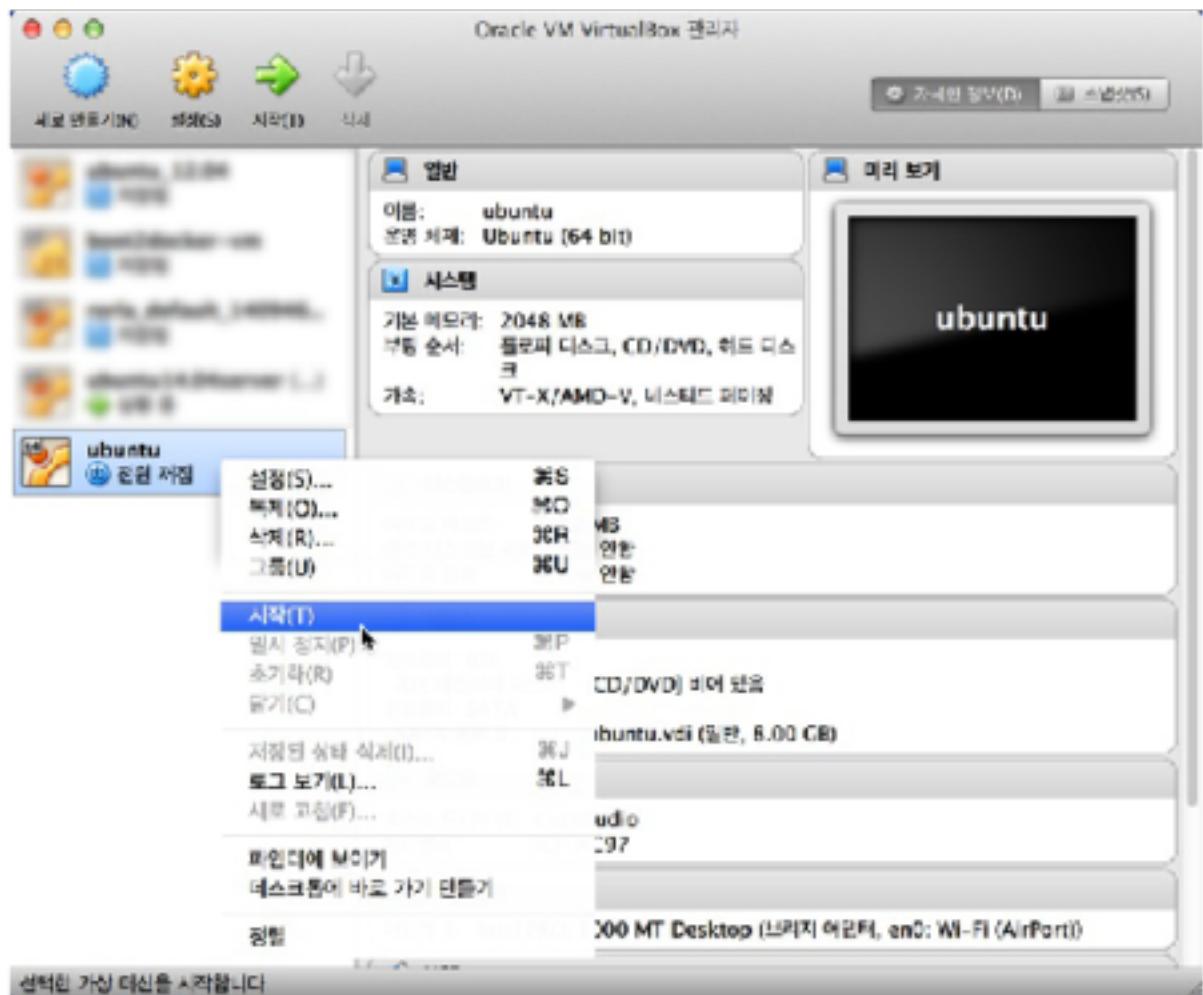
설정 항목을 선택한다.



10. 네트워크 1번 어댑터 탭에서 브리지 어댑터를 선택하고 확인 버튼을 클릭한다.

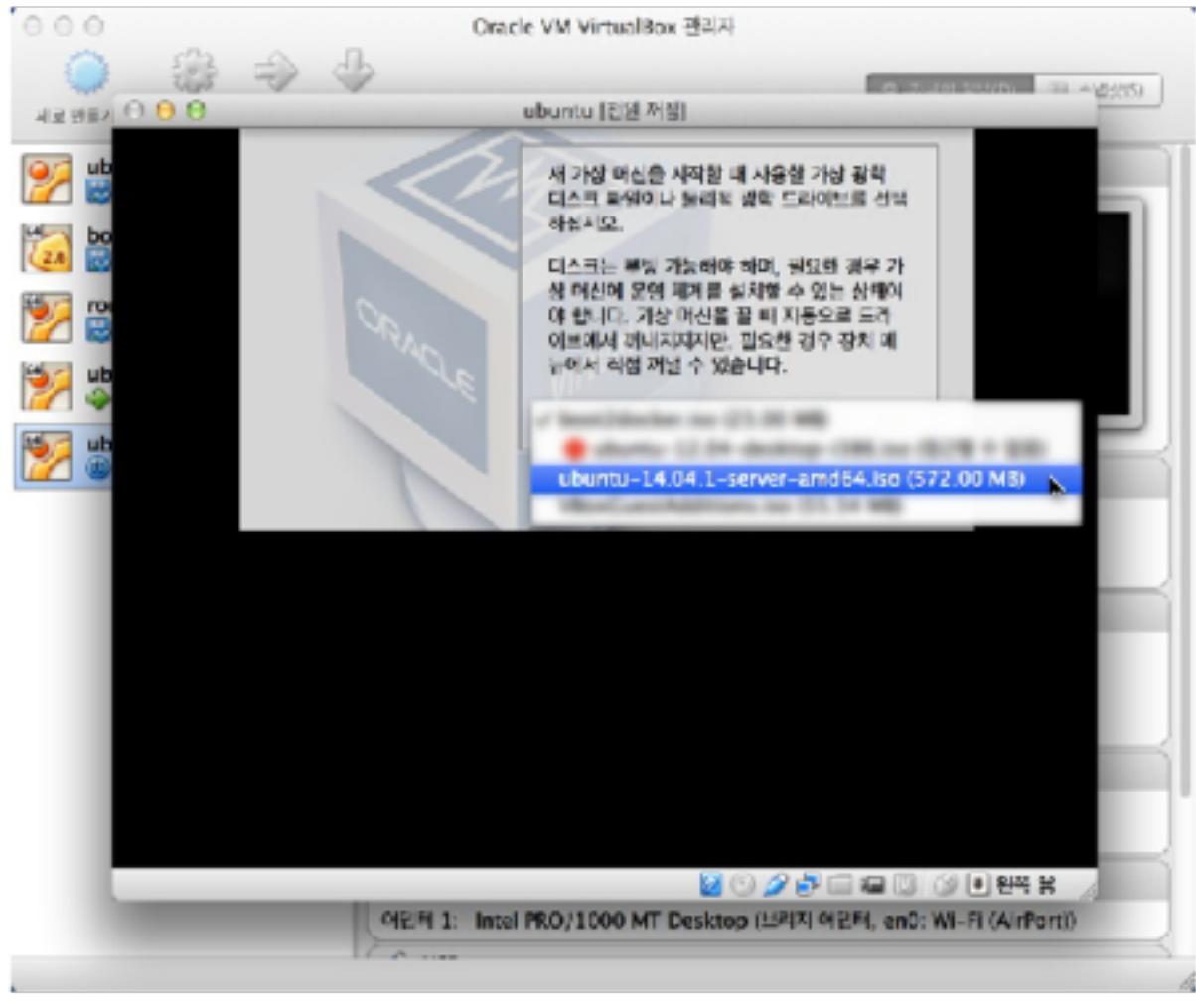


11. 가상머신 시작하기



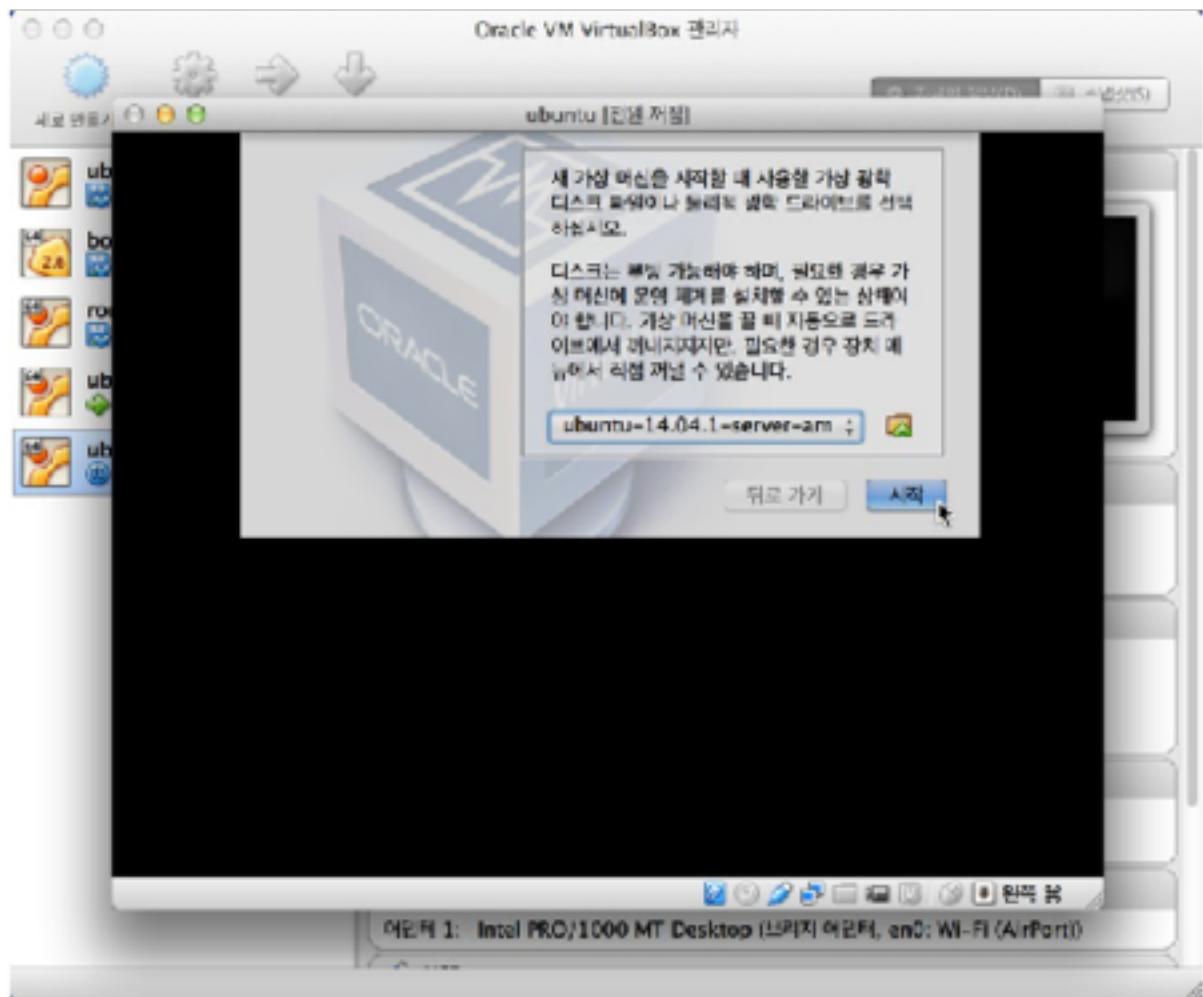
12. 다운로드 받은 서버 이미지 선택하기

: 미리 다운로드 받아 놓은 우분투 서버 이미지 파일을 선택한다.

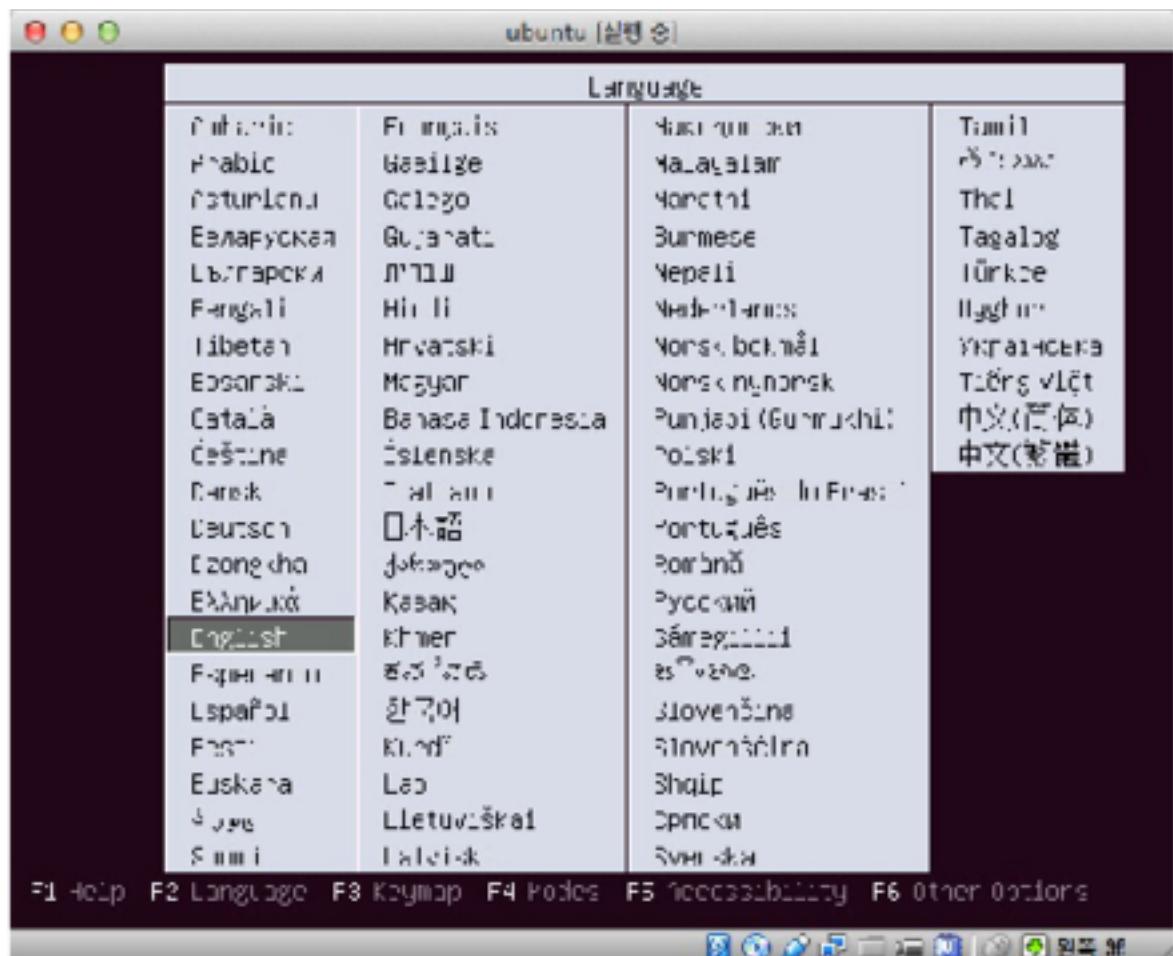


게스트 서버의 설치

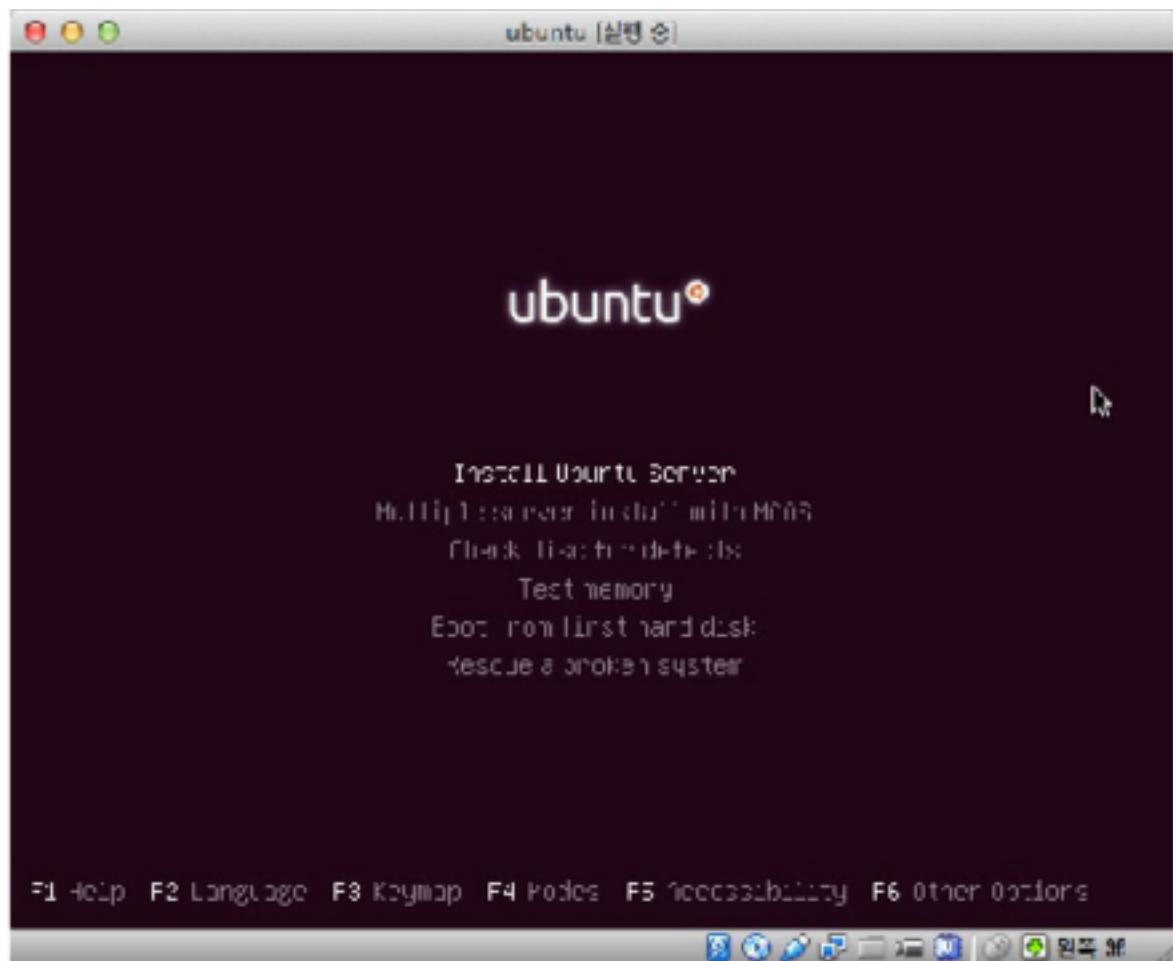
13. 서버 설치 시작



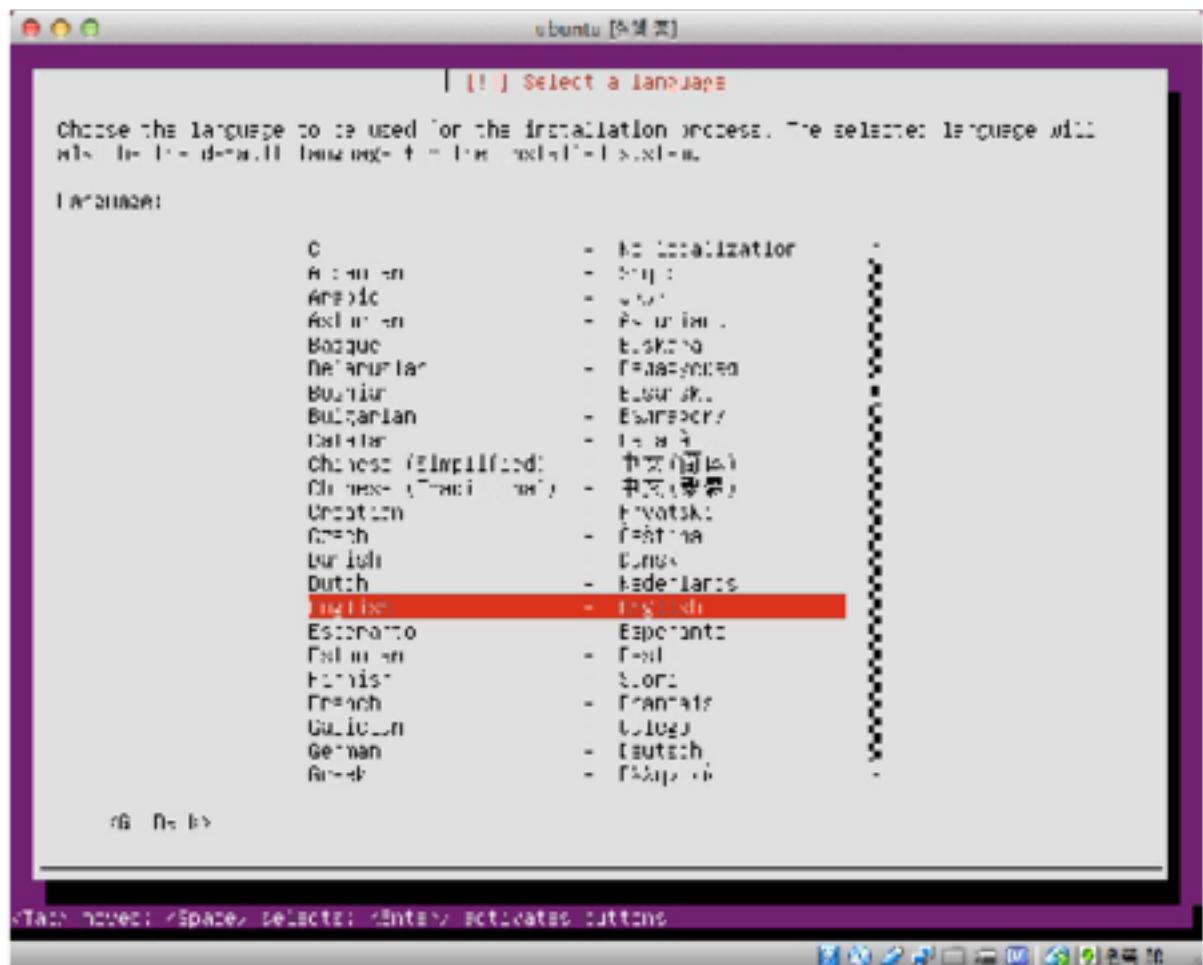
14. 사용할 언어를 선택하는 화면에서 기본값인 English 가 반전된 상태에서 엔터키를 클릭 한다.



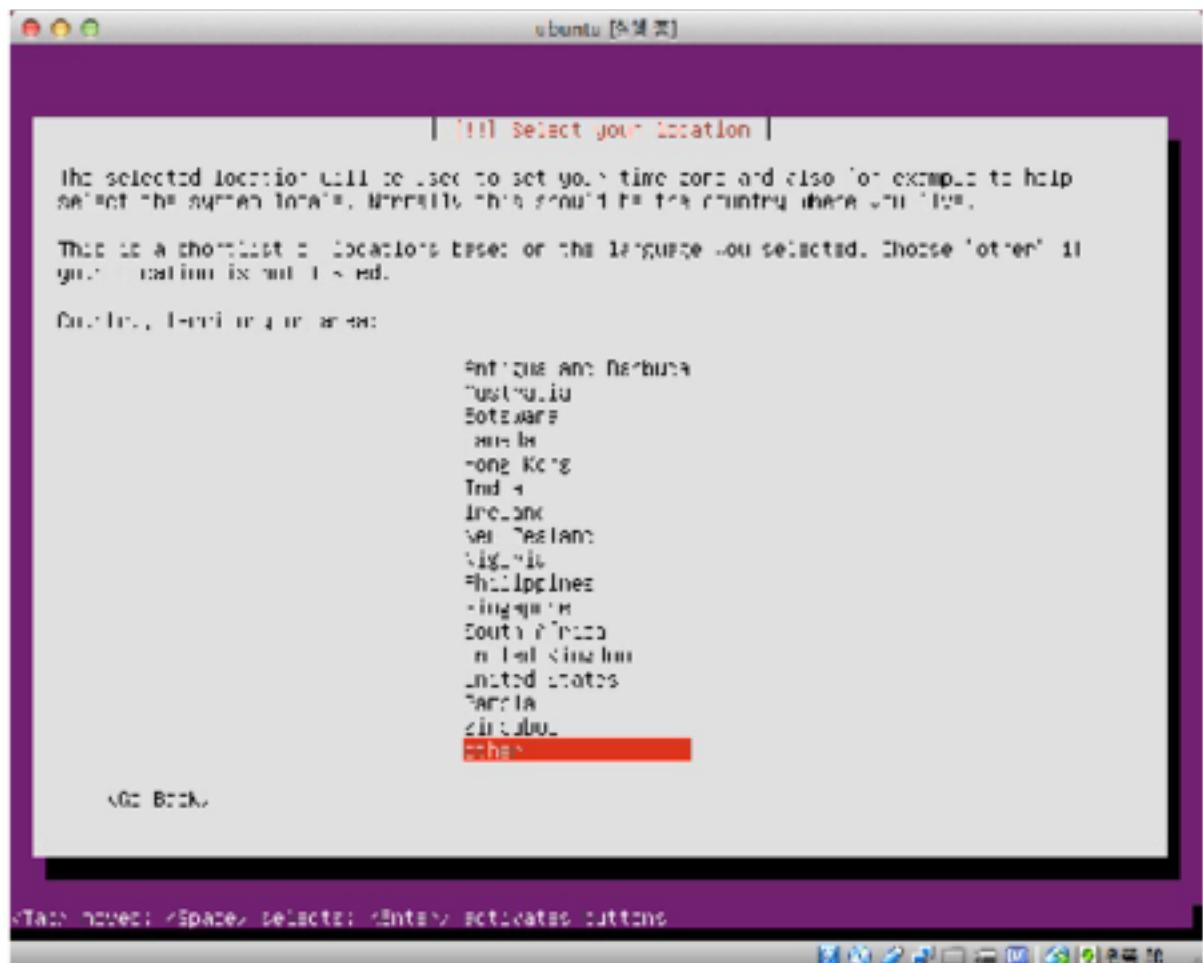
15. 우분투 서버 설치 항목을 선택한다.



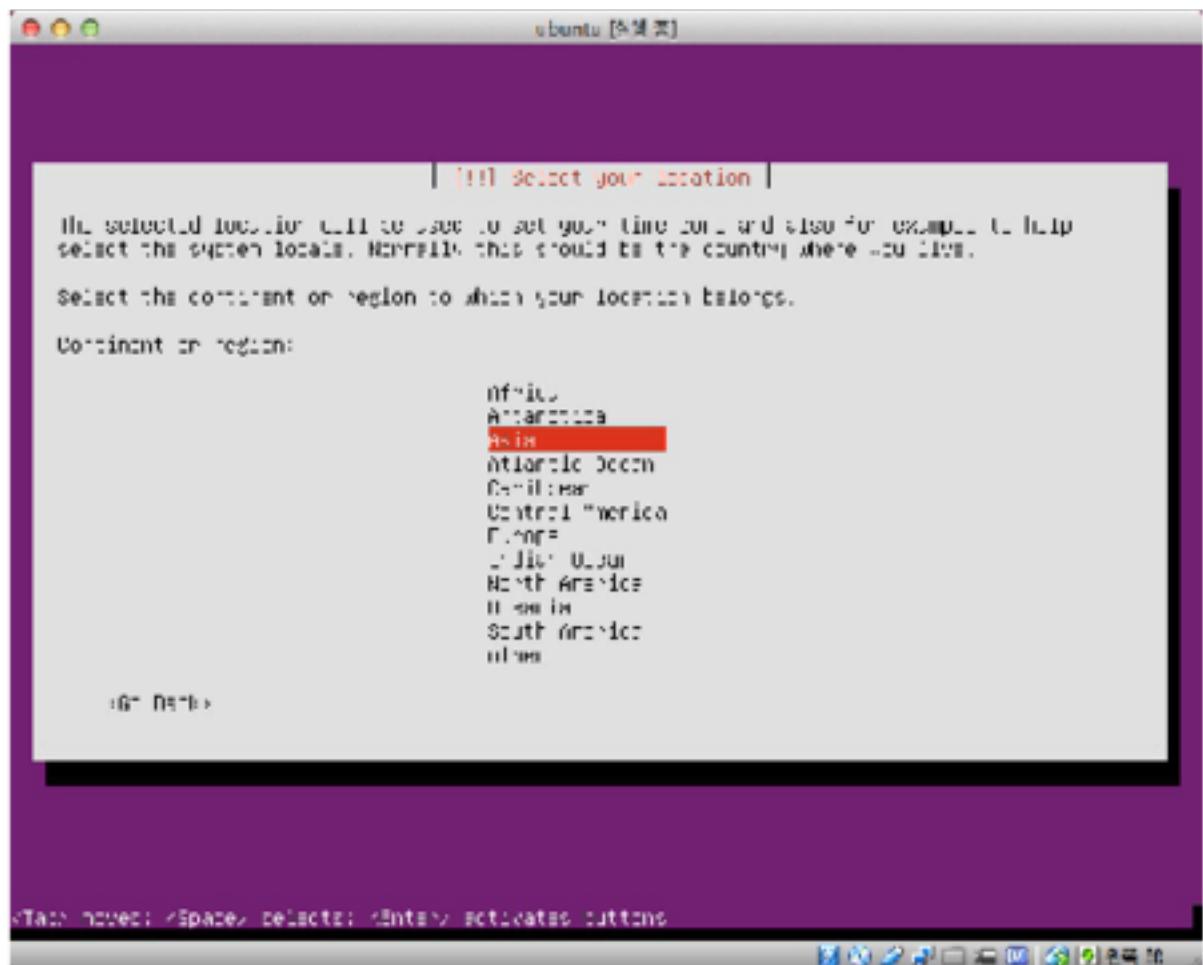
16. 설치 진행시 사용할 언어를 선택하는 화면에서도 기본값인 English 가 반전된 상태에서 엔터키를 클릭한다.



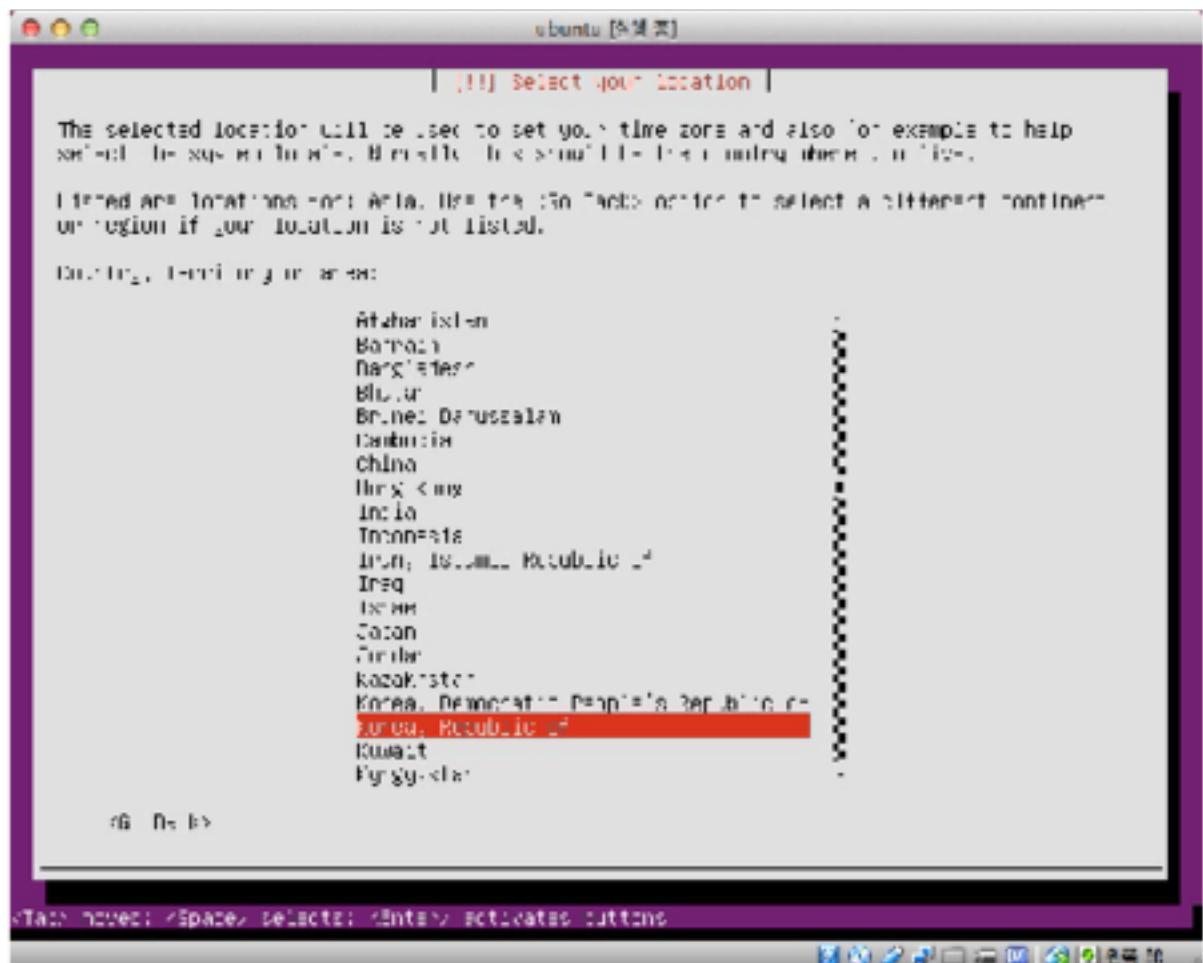
17. 타임존 설정을 위한 지역을 선택하는 화면에서는 other 항목을 선택하고 엔터키를 클릭한다.



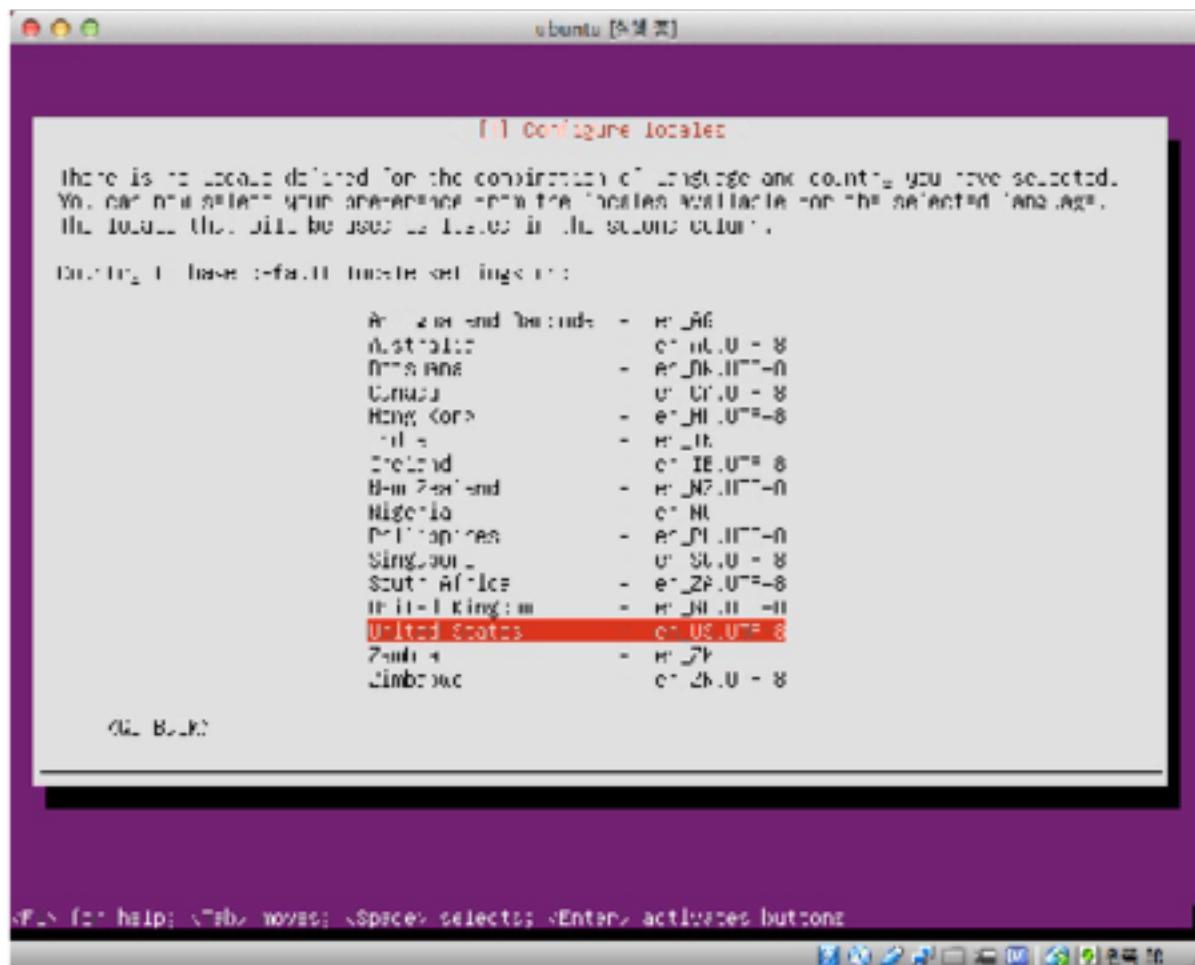
이어지는 화면에서는 Asia 를 선택하고 엔터 키를 클릭한다.



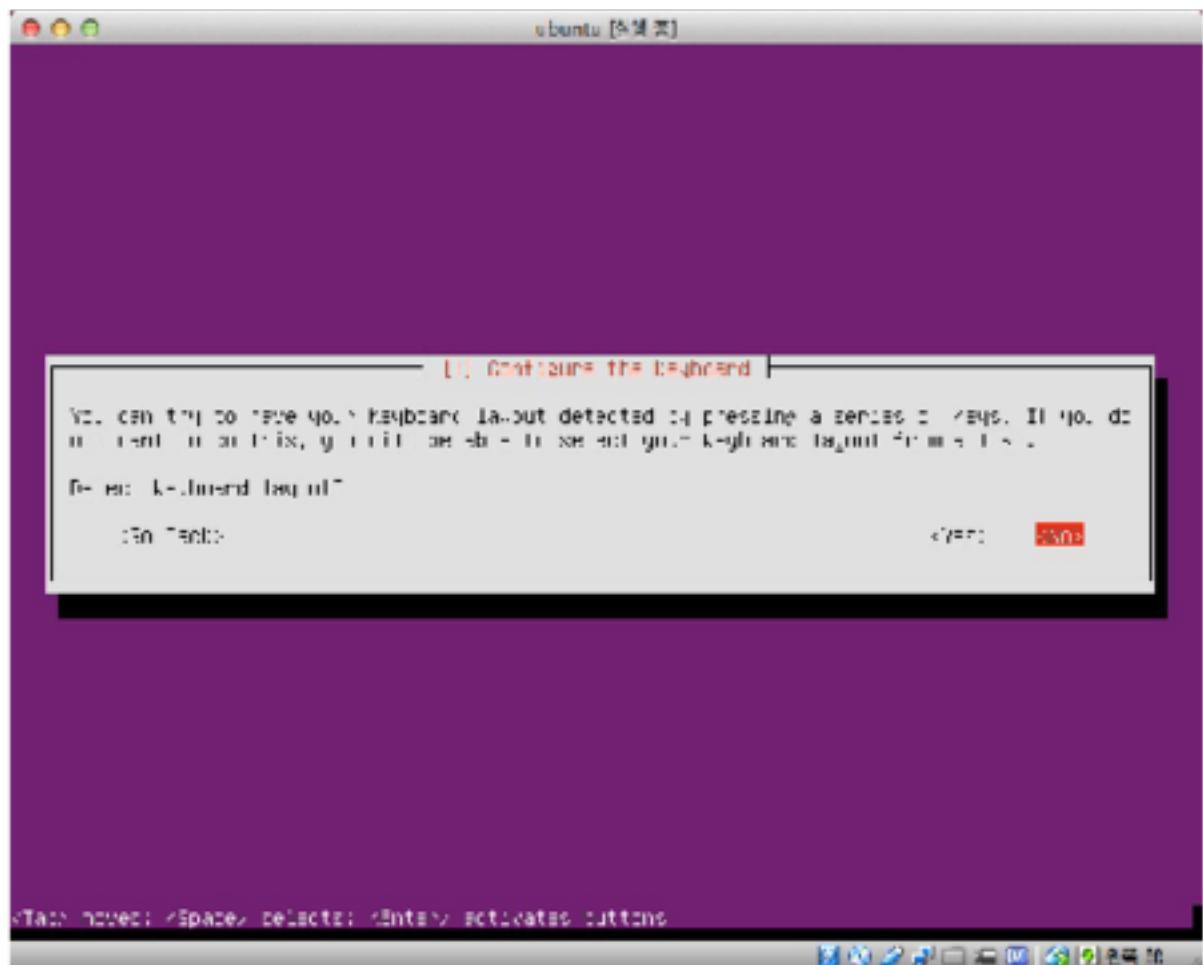
그리고 Korea, Republic of 을 선택한 후 엔터키를 클릭한다.



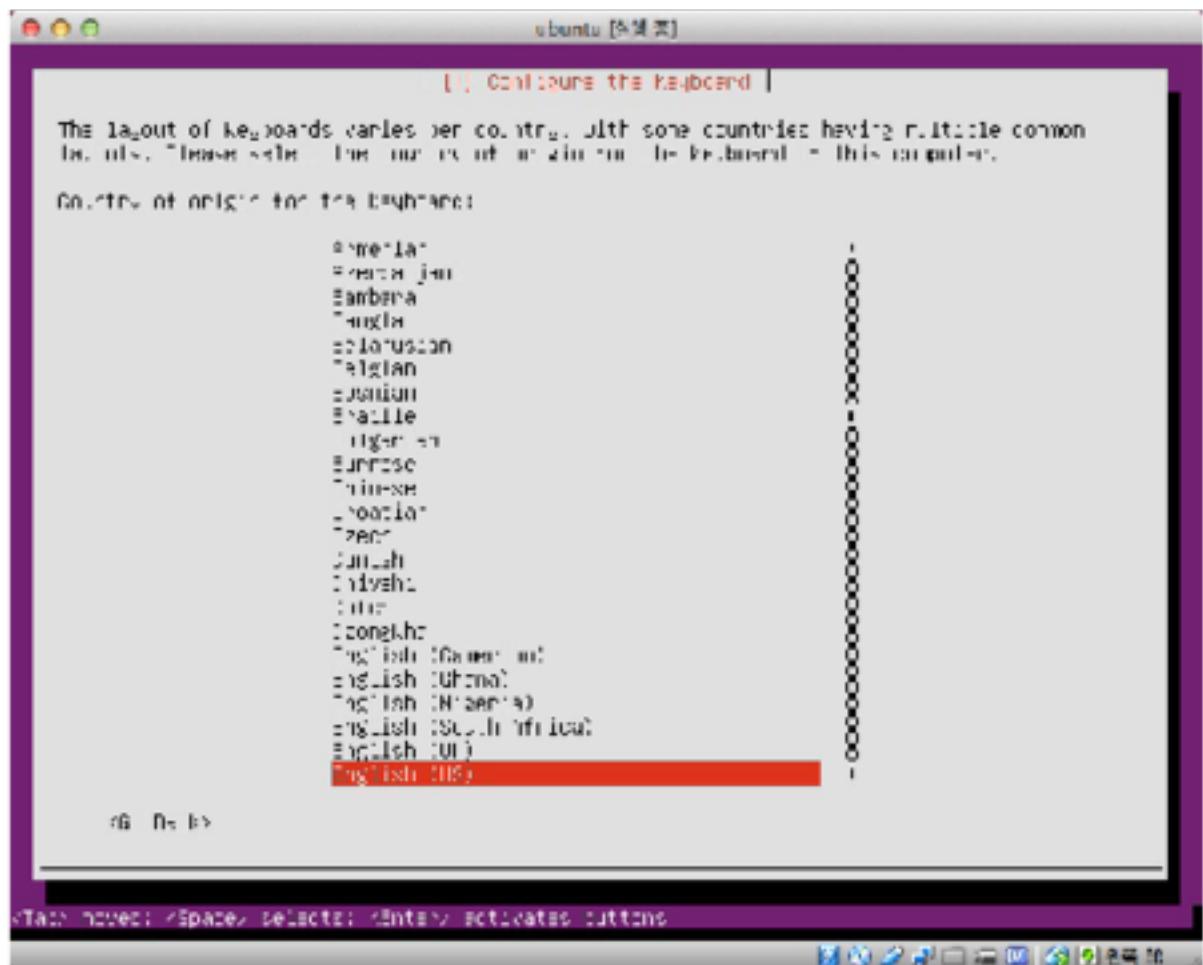
18. 로케일 설정 화면에서는 기본값인 United States - en_US.UTF-8 0 반전된 상태에서 엔터키를 클릭한다.



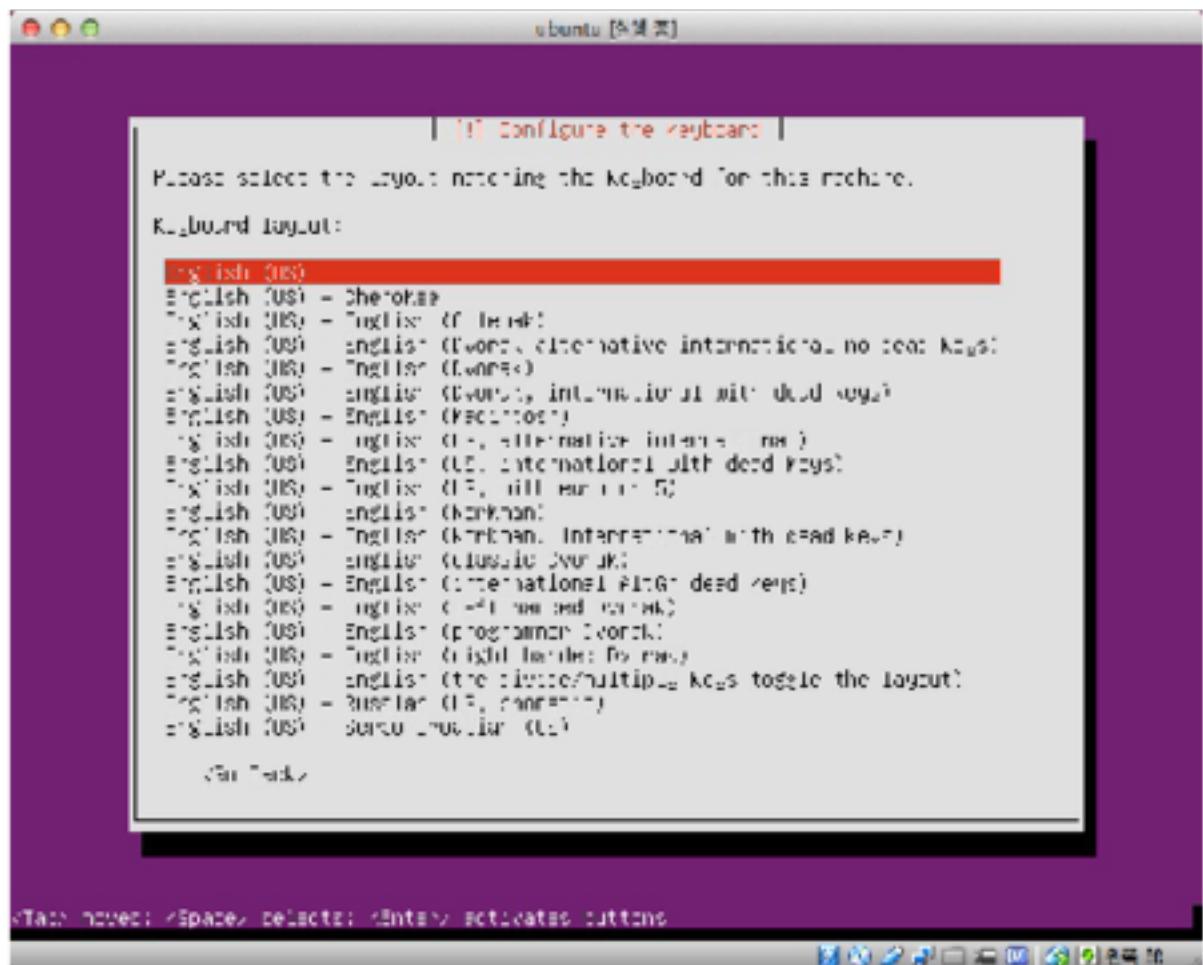
19. 키보드 검색 화면에서는 기본값인 <No>를 선택한다.



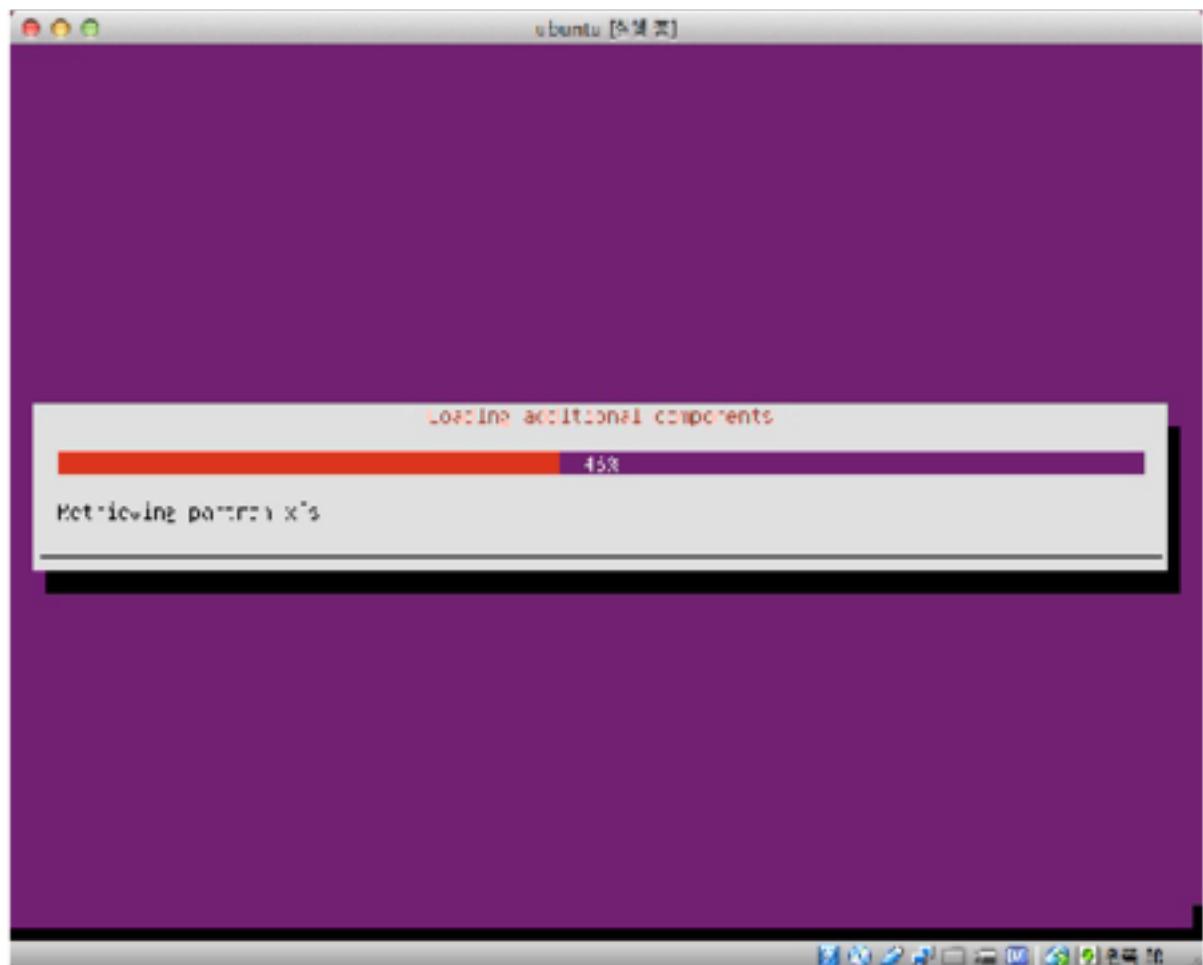
20. 키보드 레이아웃 선택 화면(Country of origin)에서는 기본값인 English (US)를 선택 한다.



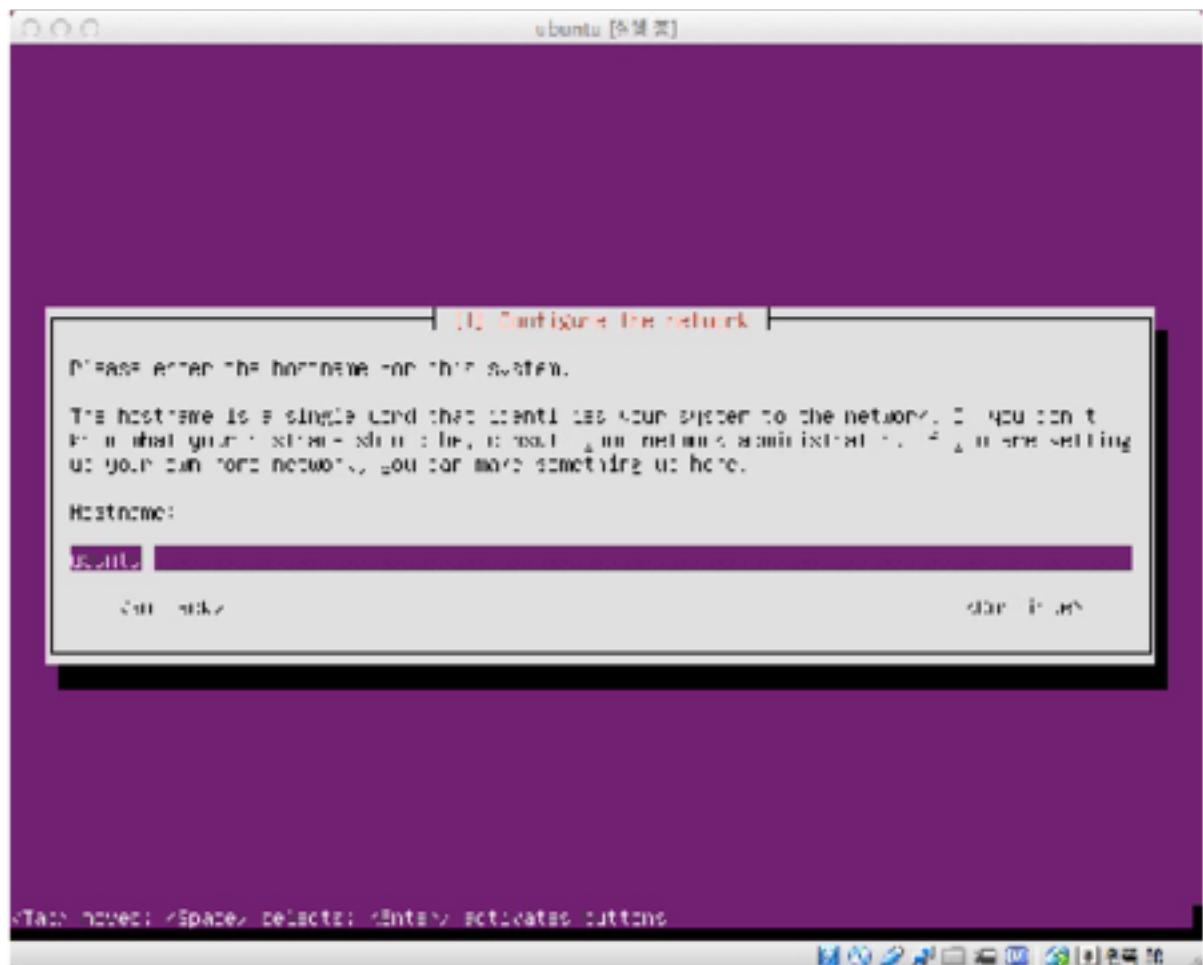
21. 키보드 레이아웃도 기본값인 English (US)를 선택한다.



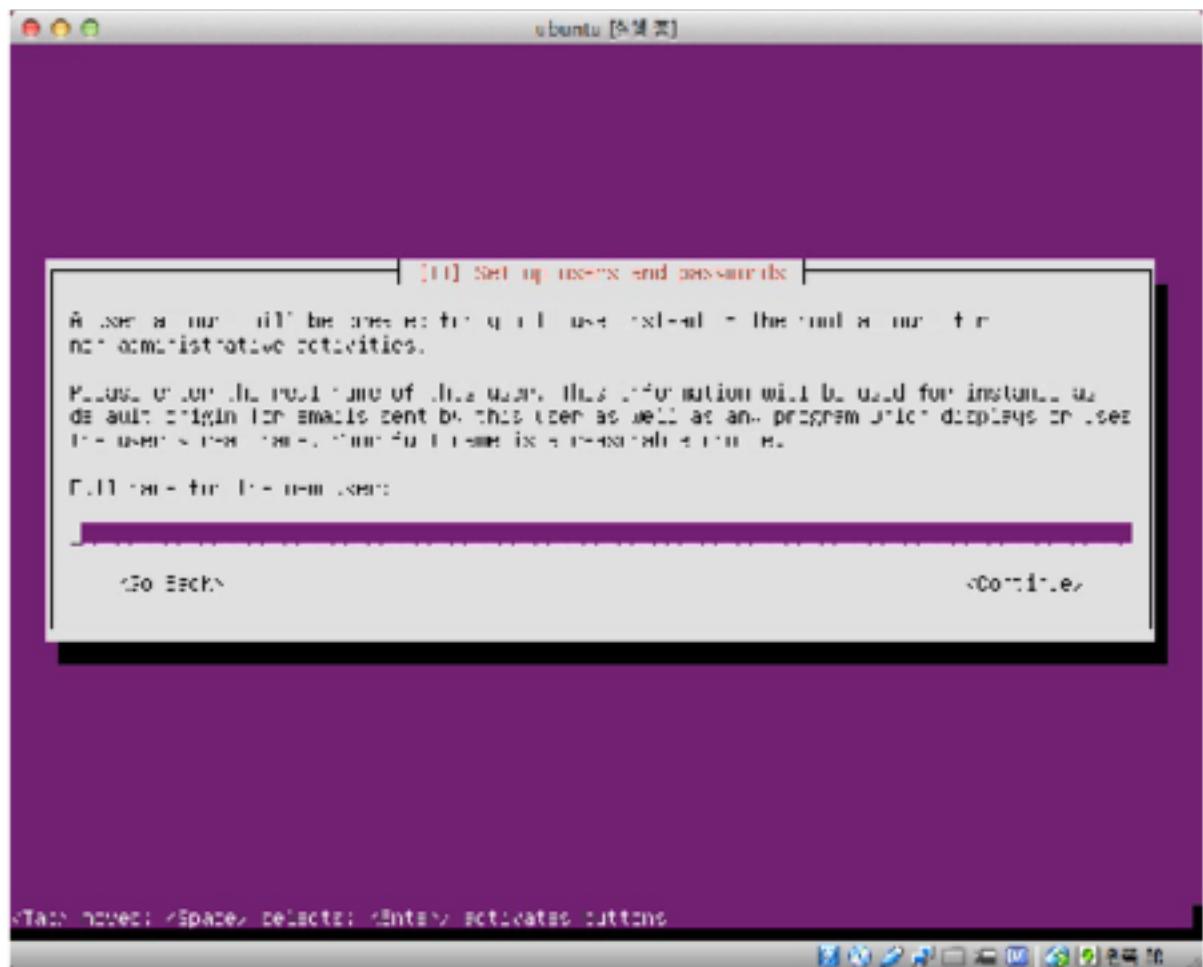
22. 컴포넌트 진행과정



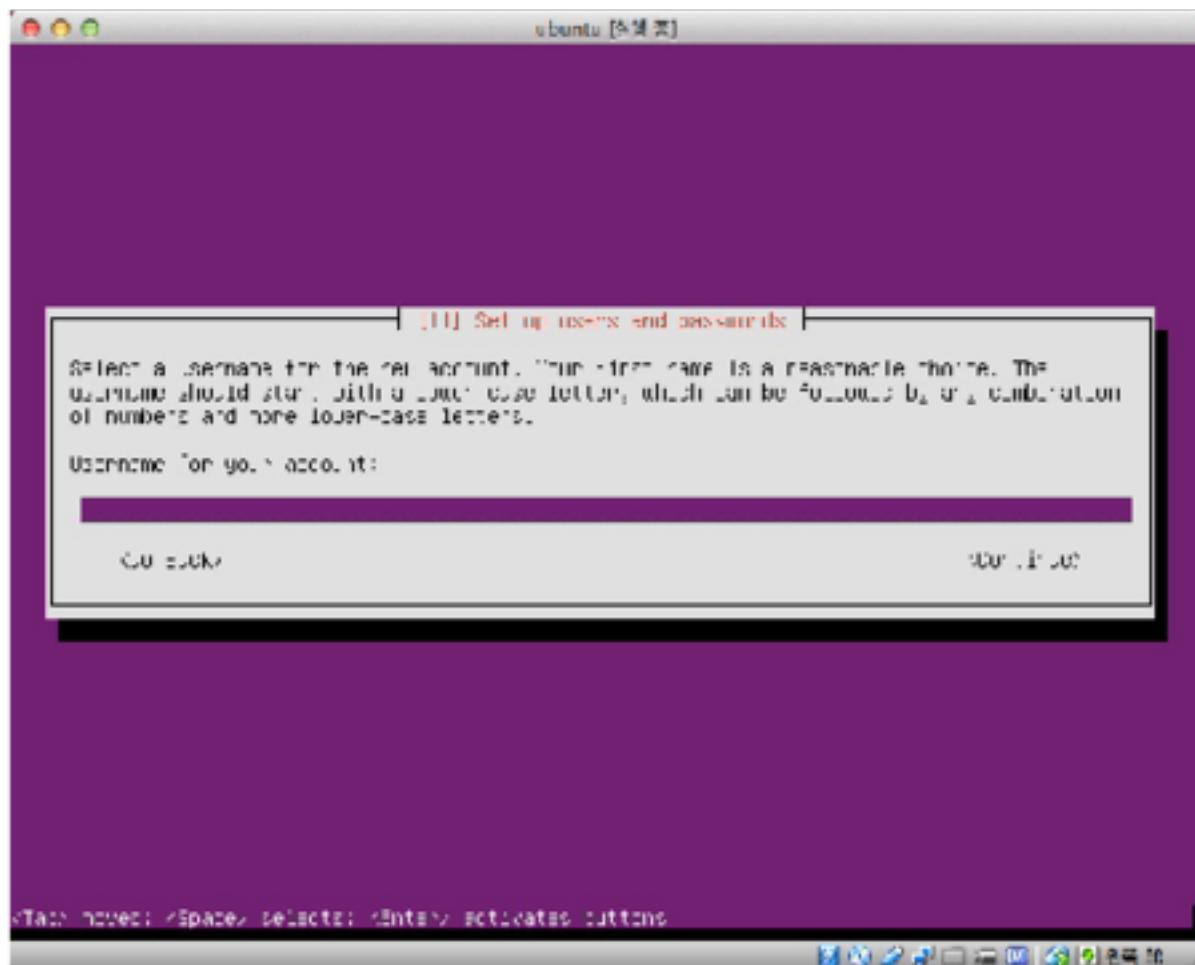
23. 가상머신에 사용한 호스트이름을 정한다.



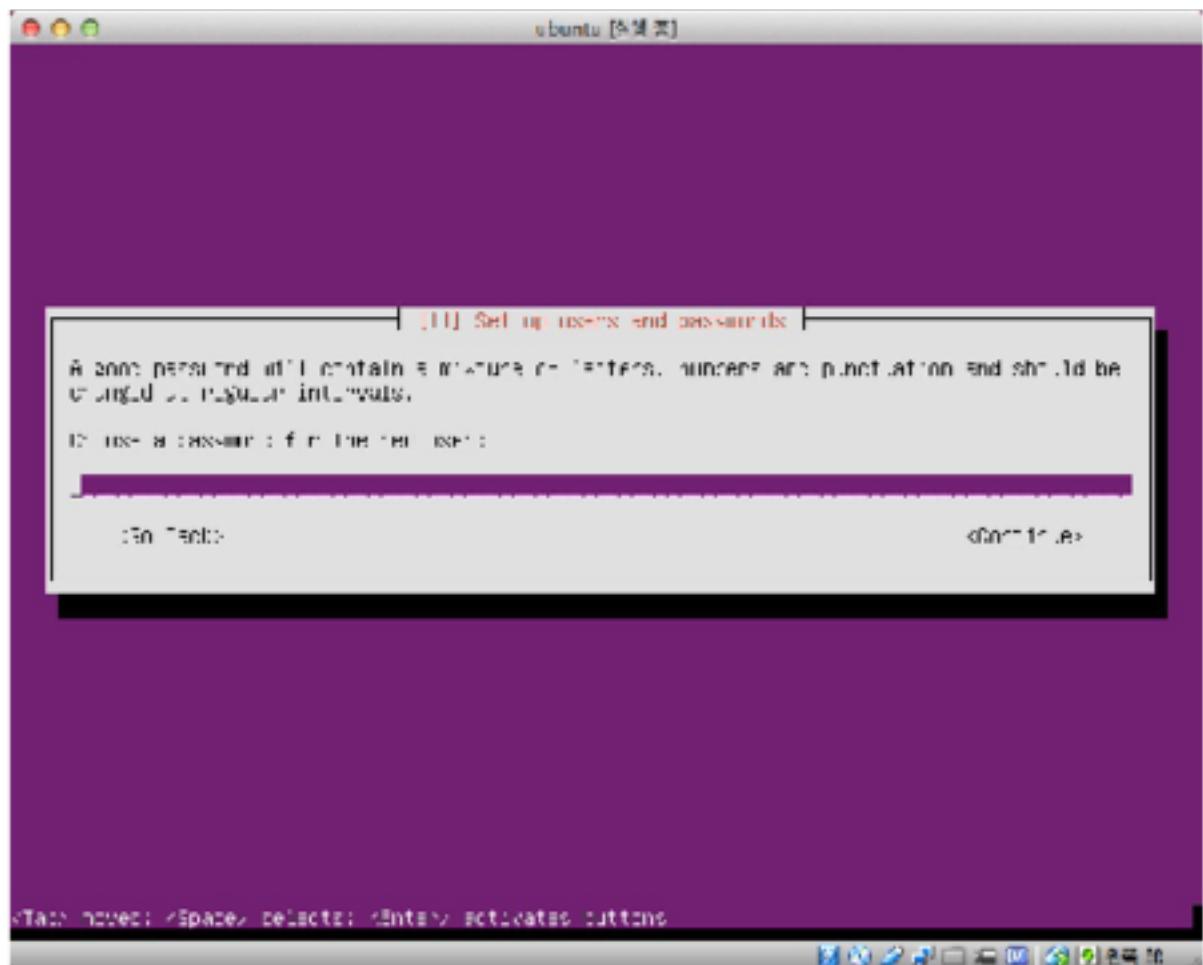
24. 유저 이름을 입력한다.



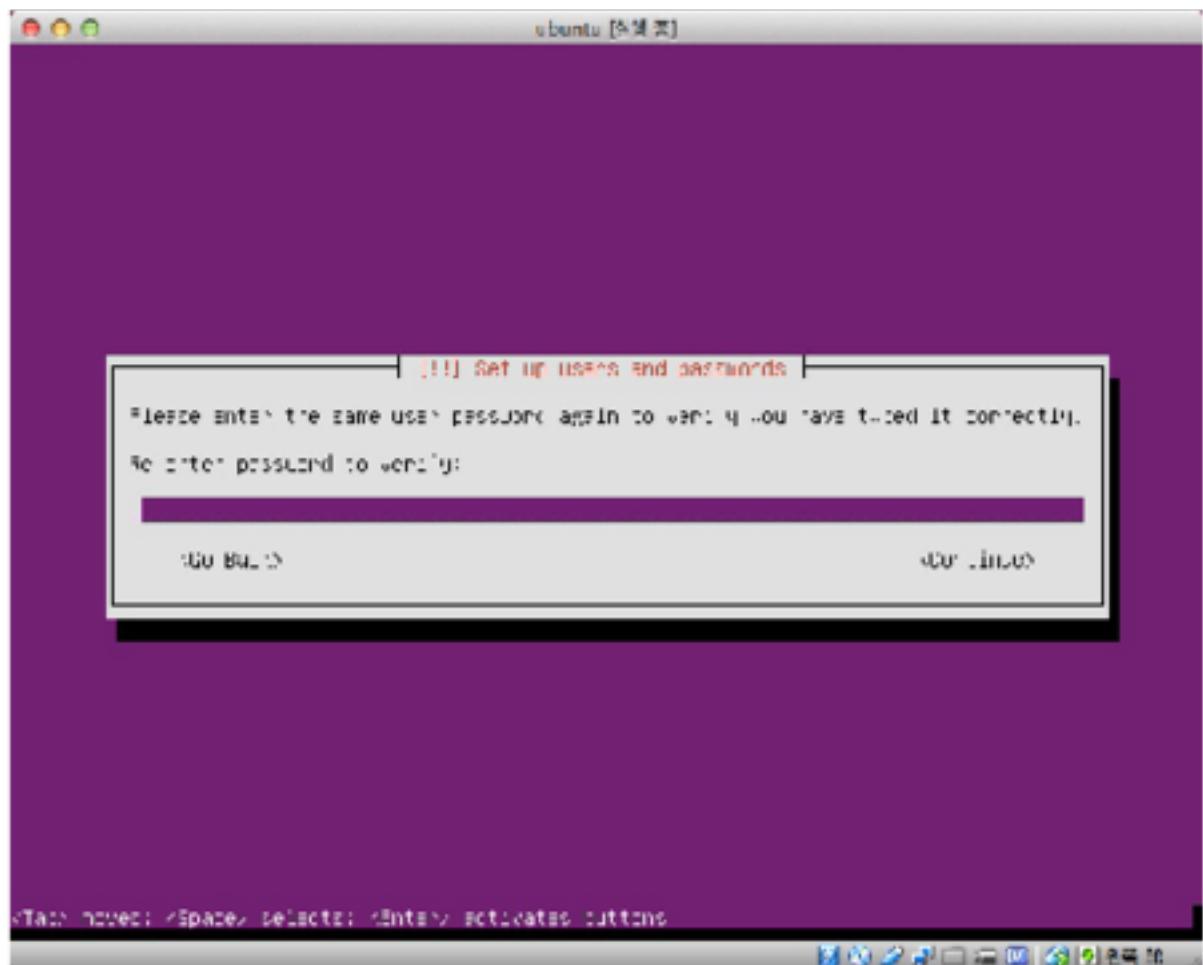
25. 계정 이름을 입력한다.



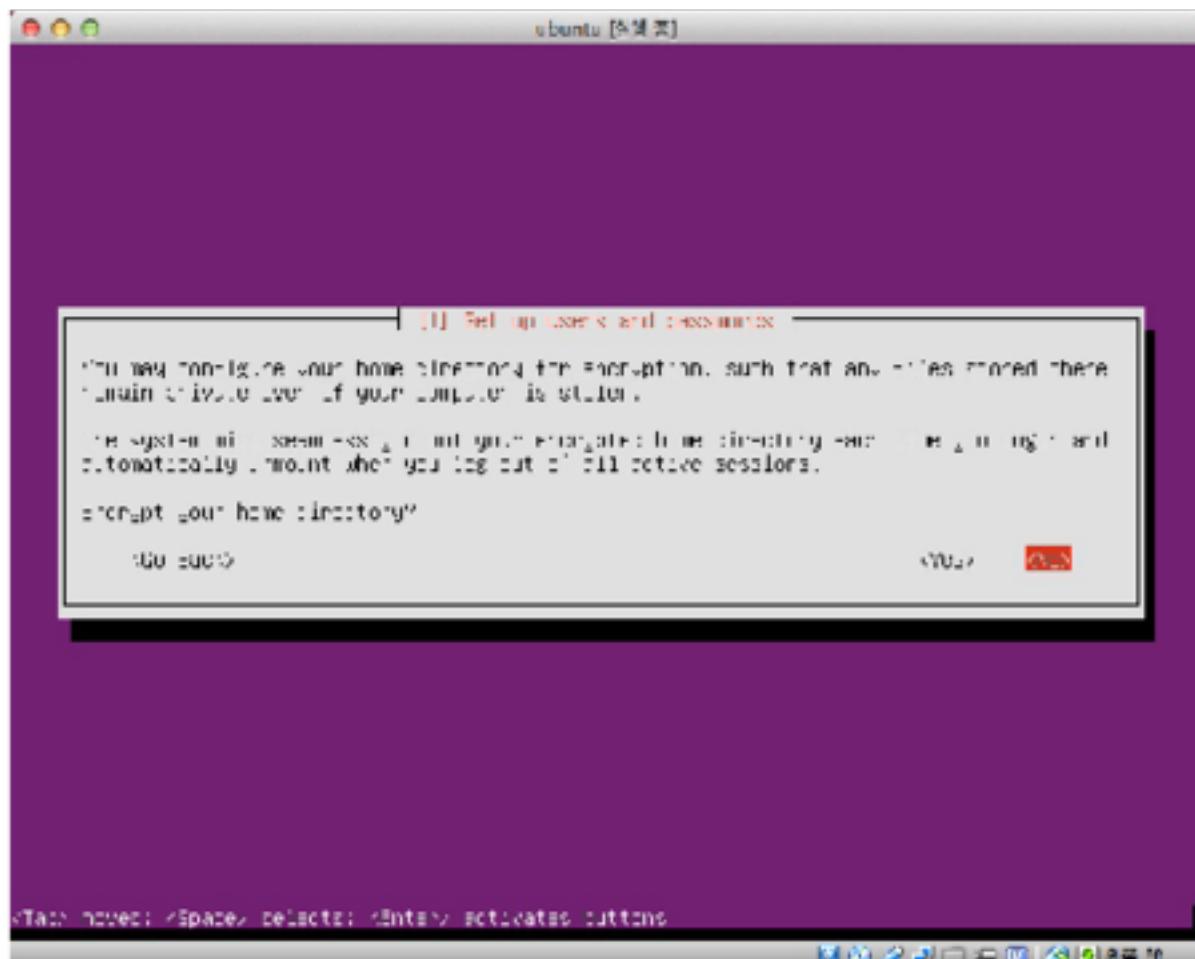
26. 암호를 입력한다.



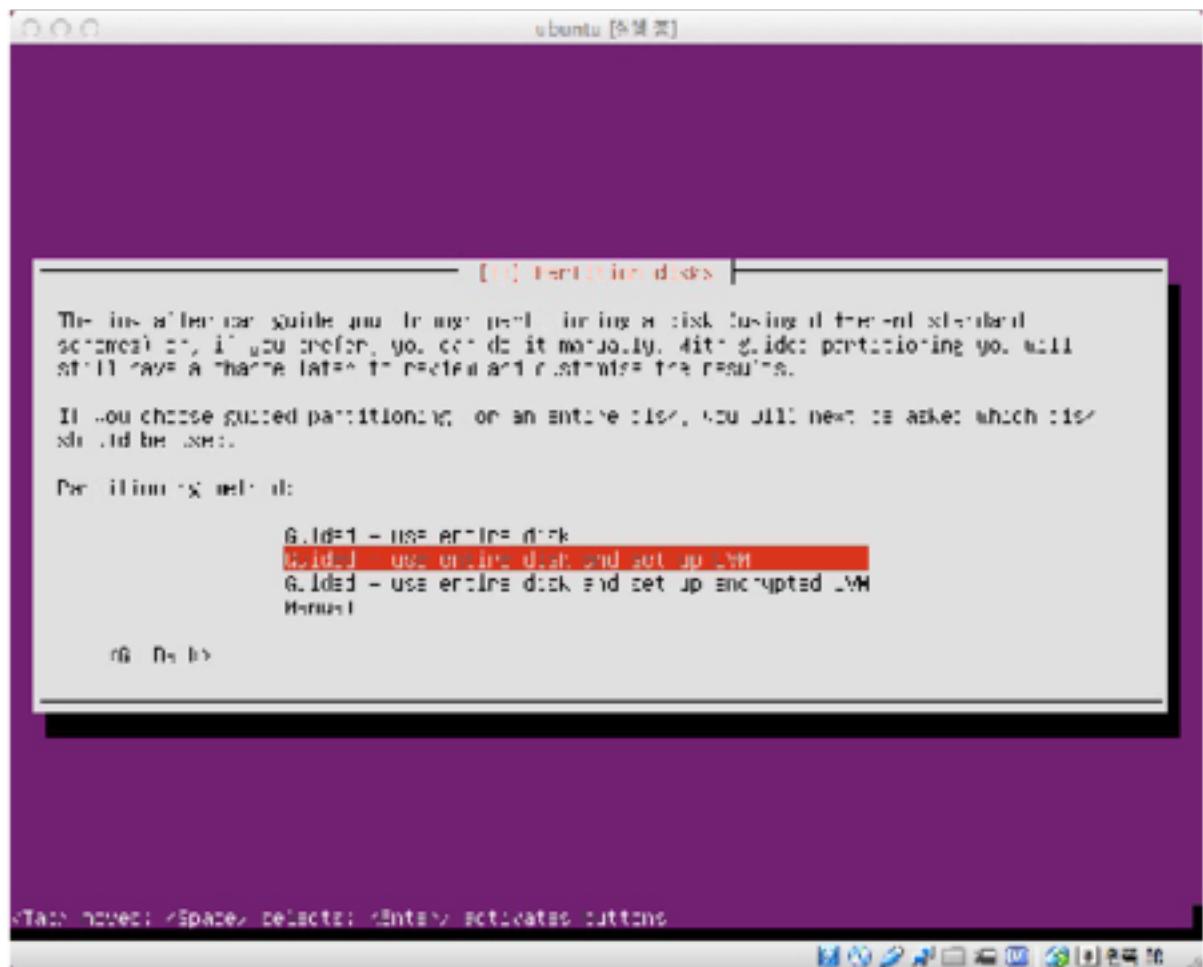
동일한 암호를 한번 더 입력한다.



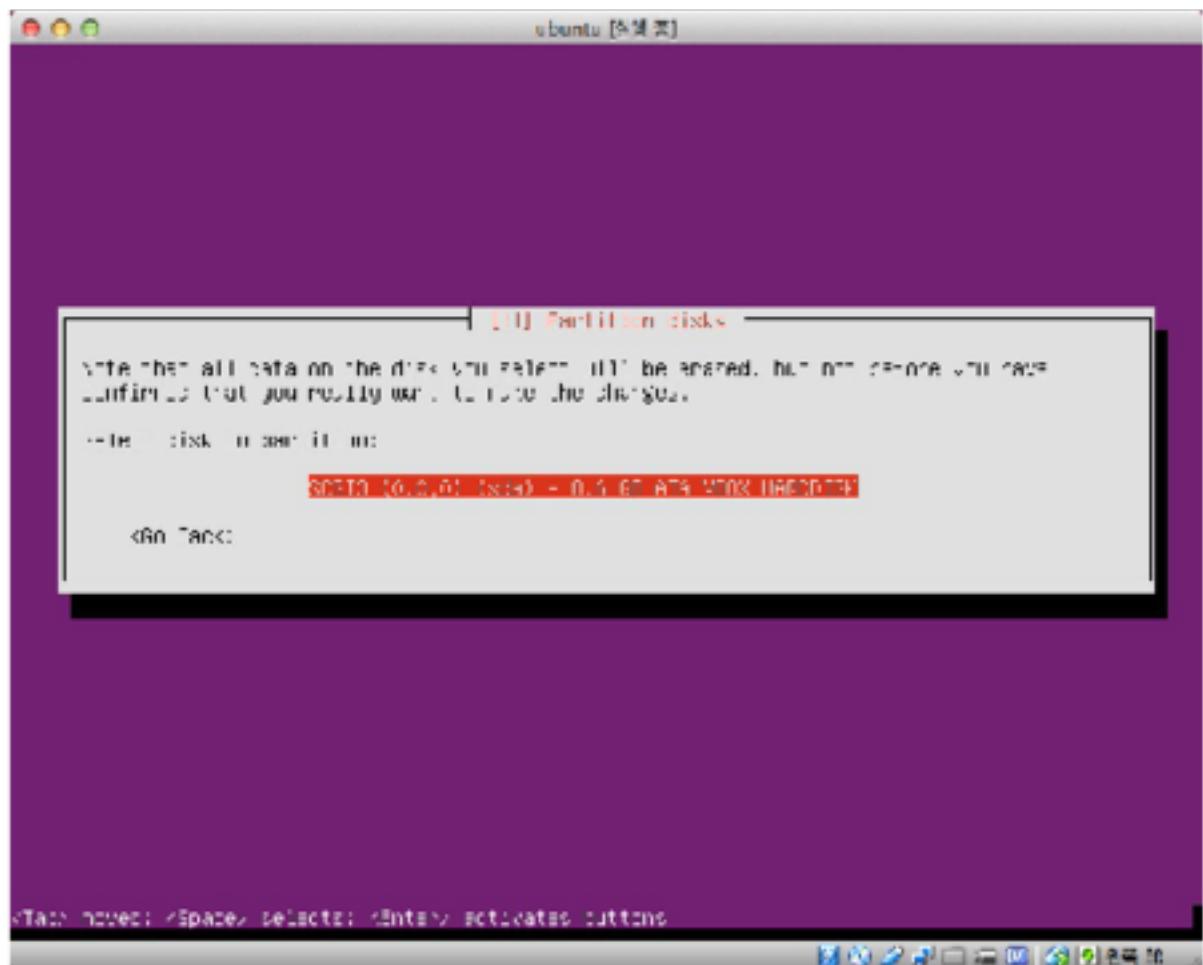
27. 홈 디렉토리 암호화 여부는 기본값인 <No> 를 선택한다.



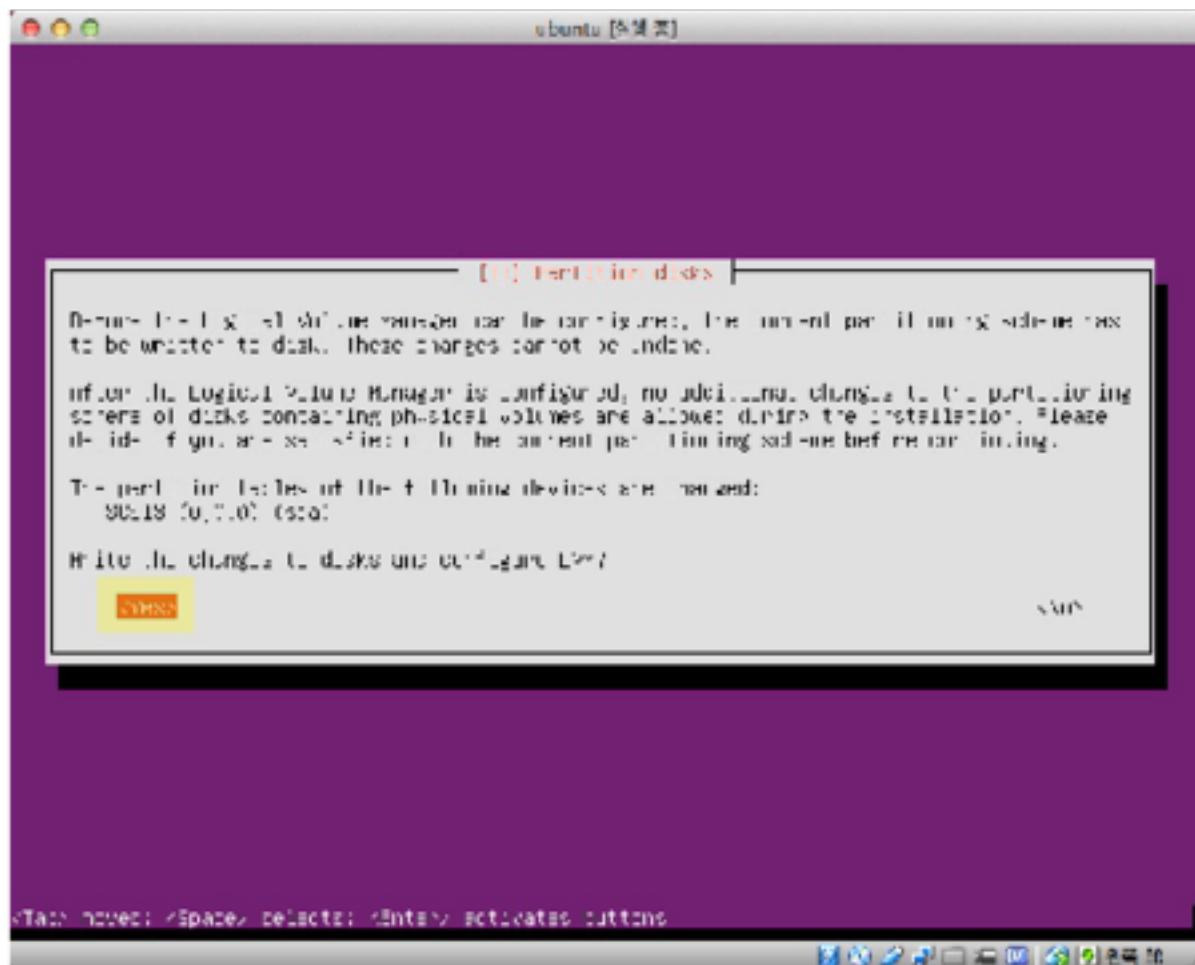
28. 하드디스크 파티션 방법은 기본값인 use entire disk and set up LVM 을 선택한다.



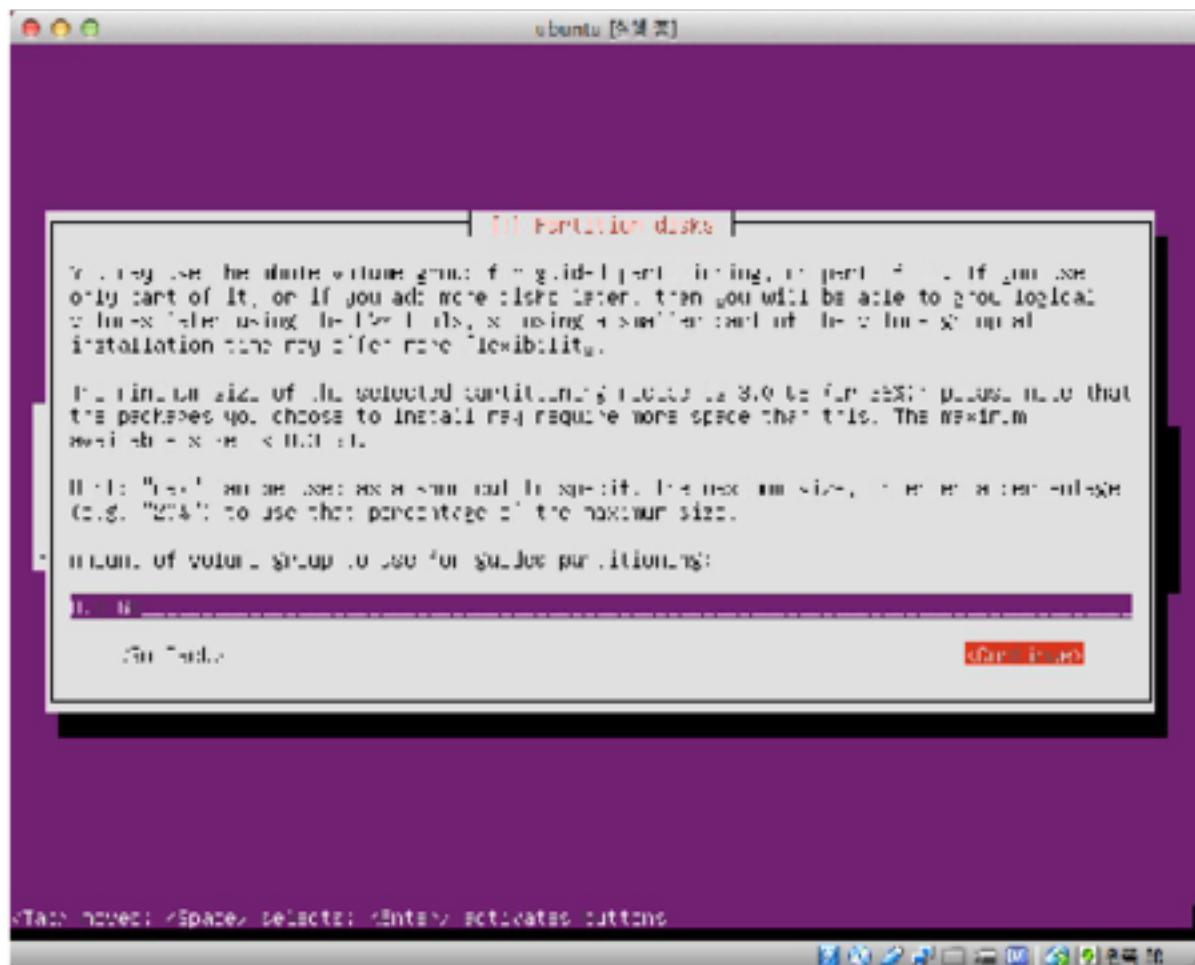
29. 파티션 할 디스크 선택은 기본값을 선택한다.



30. 변경 내용을 저장 화면에서 <Yes> 를 선택한다.

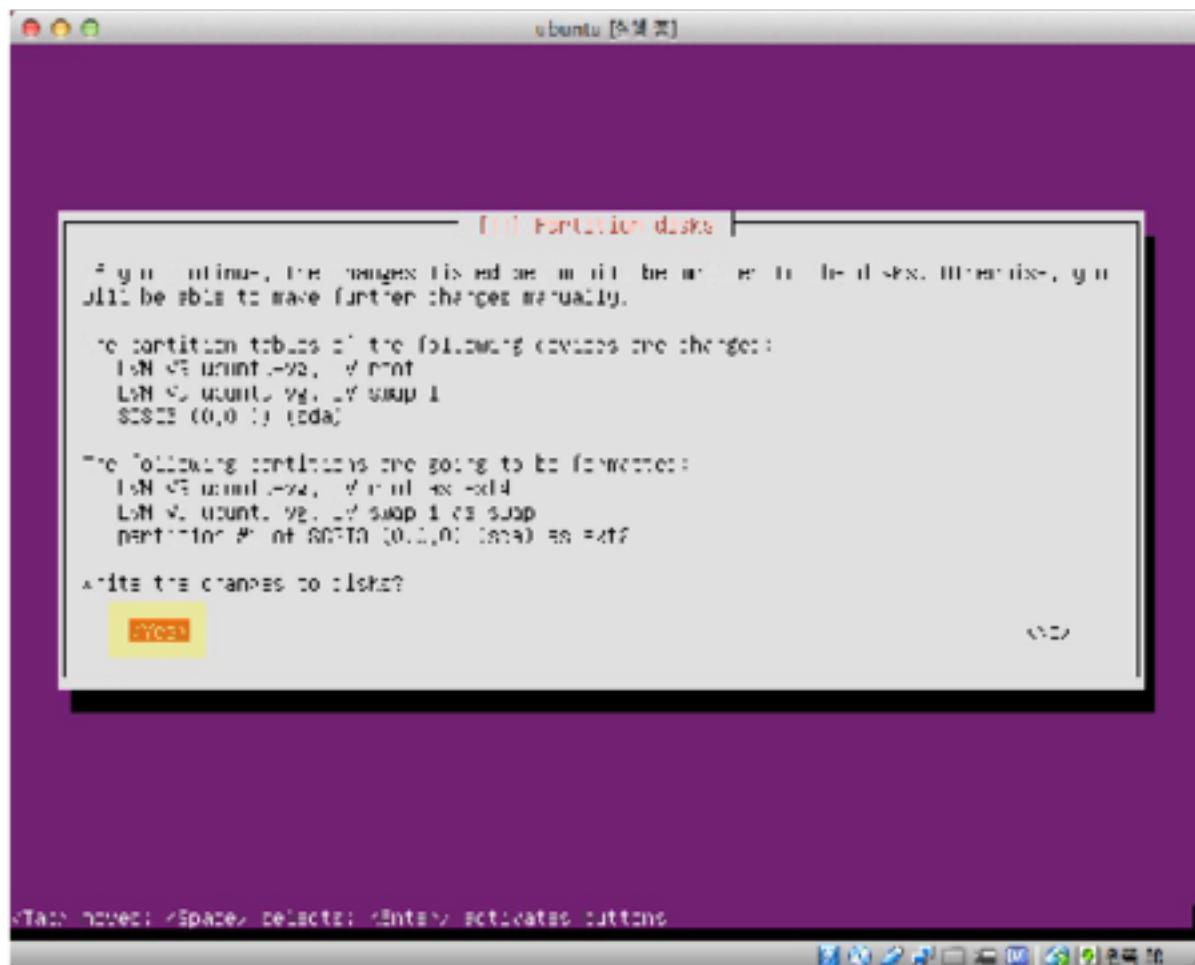


31. 디스크 볼륨도 기본값으로 선택하고 계속한다.

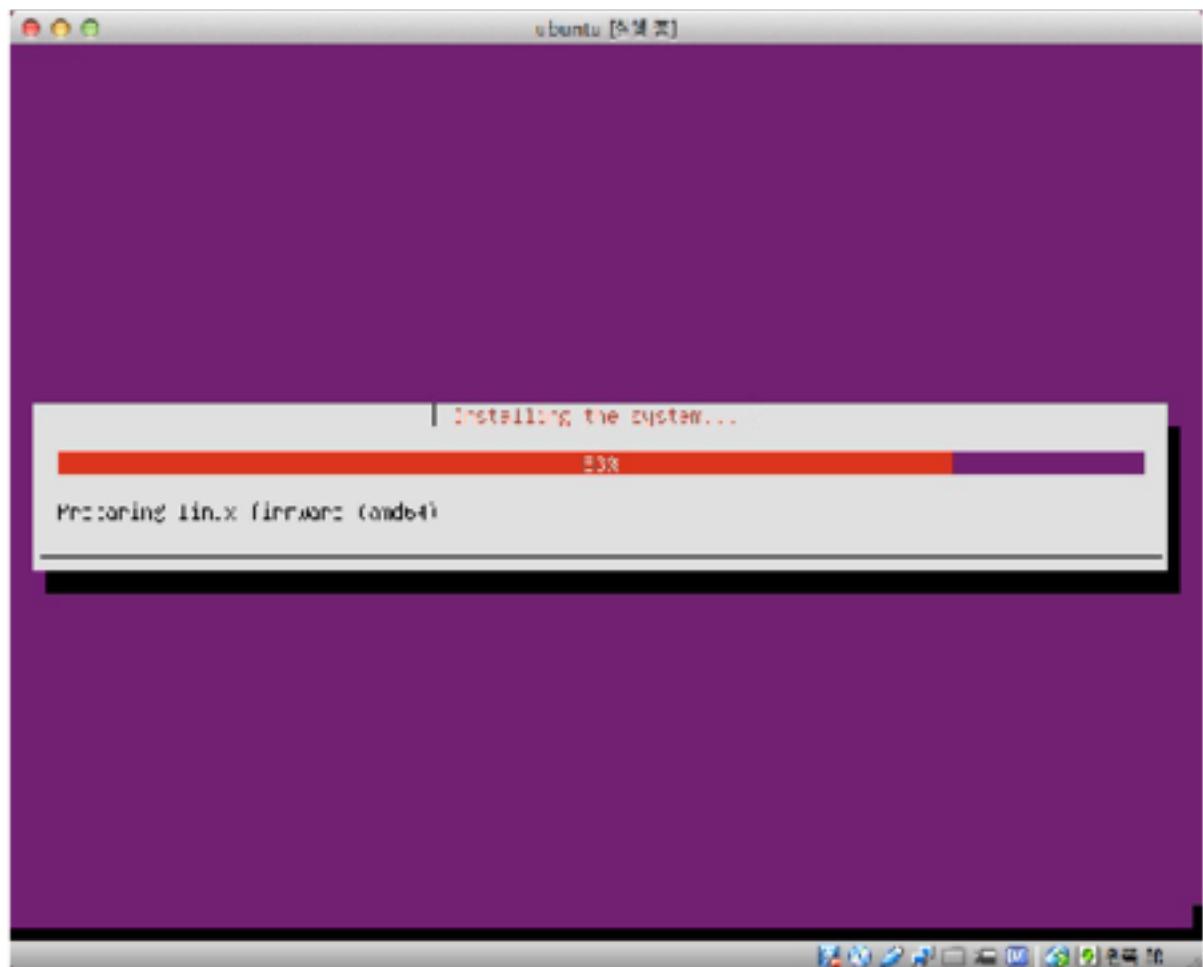


Space needed: <Space> Selects: <Enter> Activates buttons

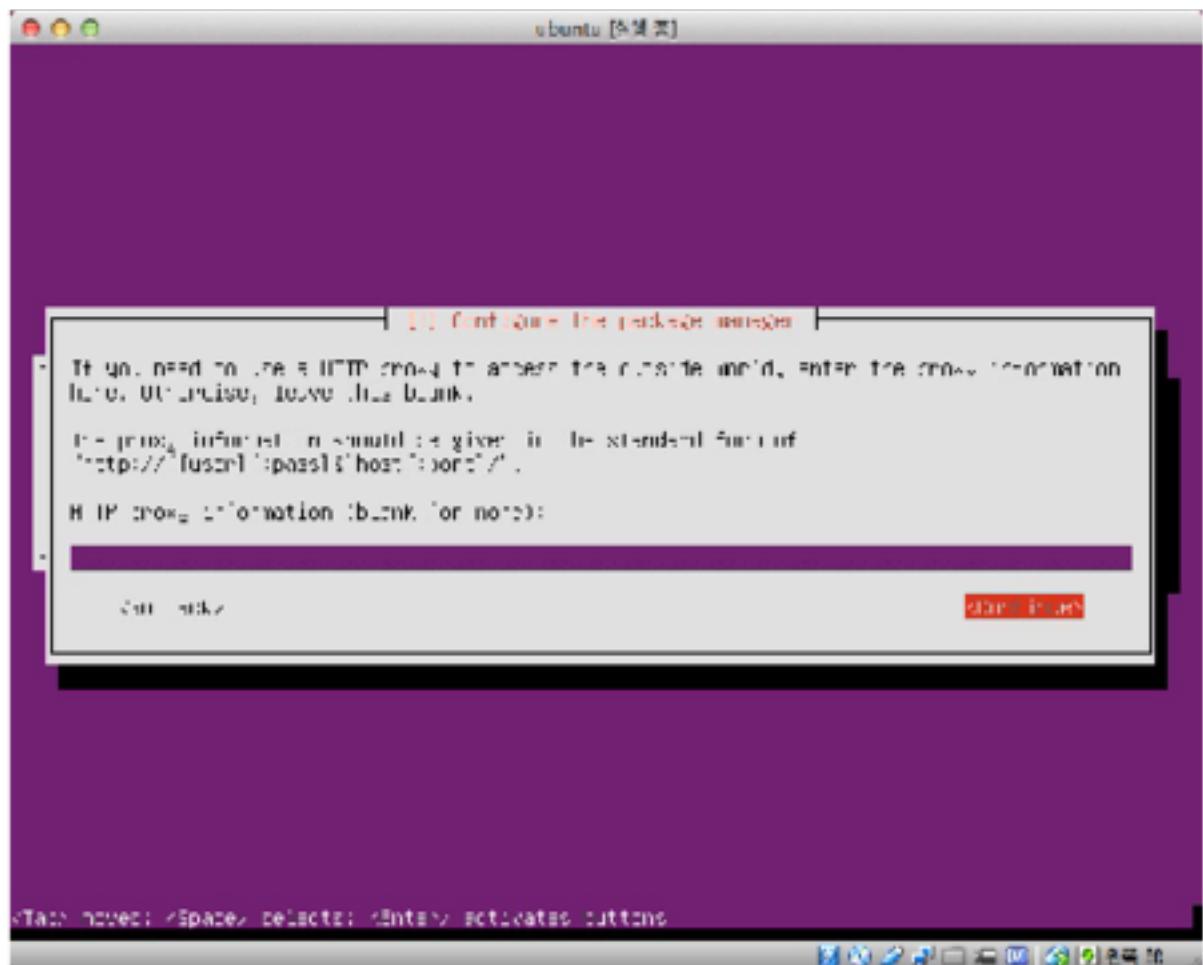
32. 변경 내용을 저장하는 화면에서 <Yes> 를 선택한다.



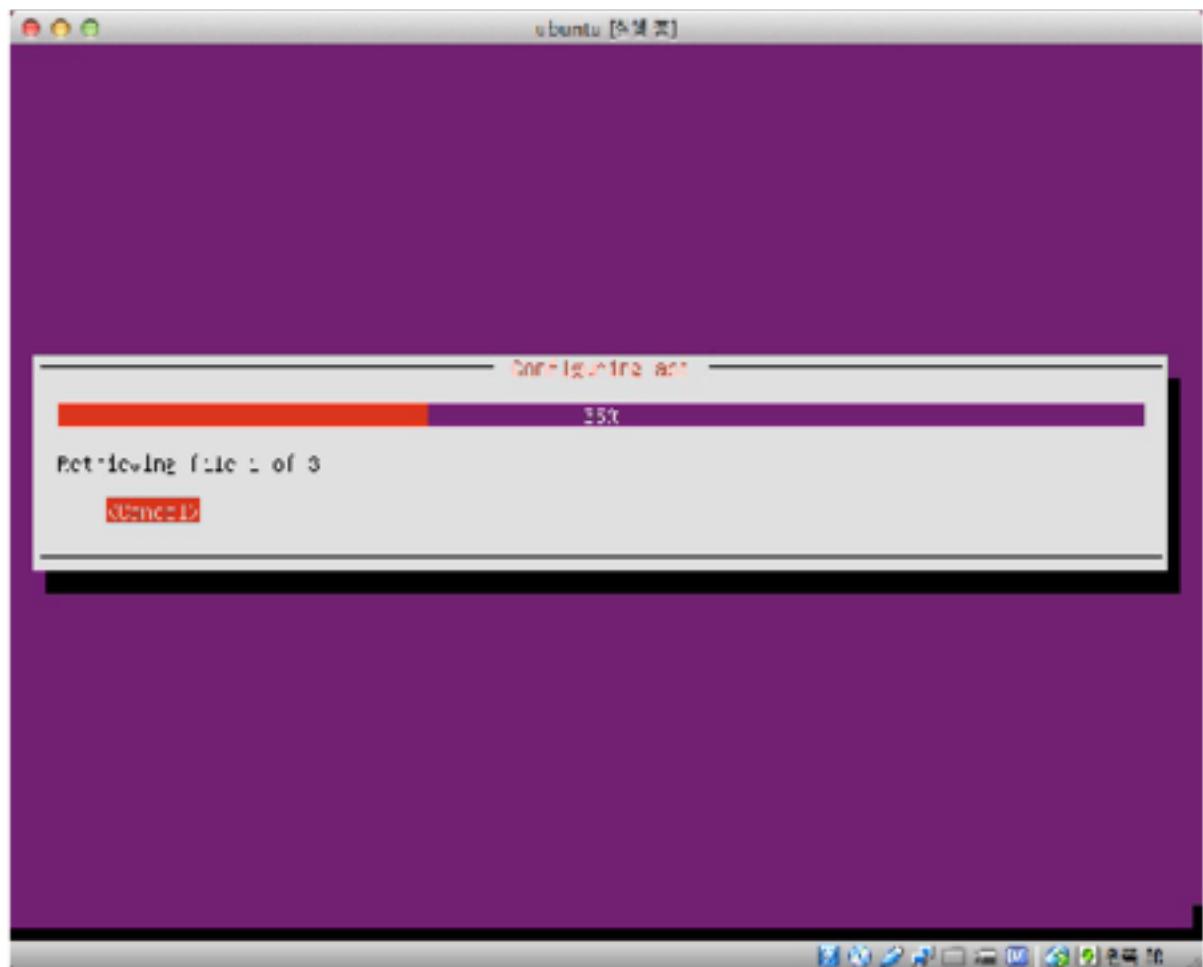
33. 시스템 설치 과정



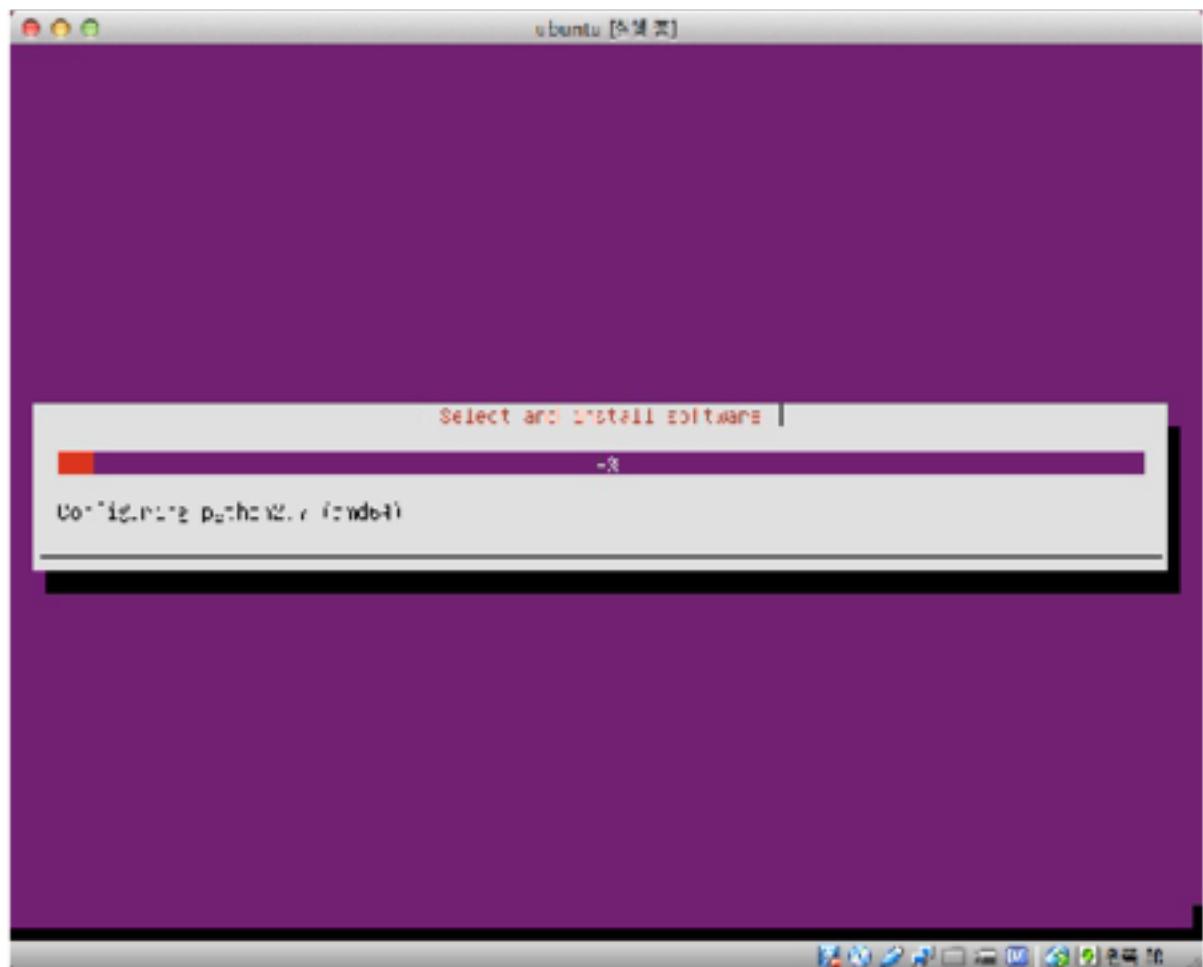
34. Proxy 정보를 지정하지 않고 계속해도 된다.



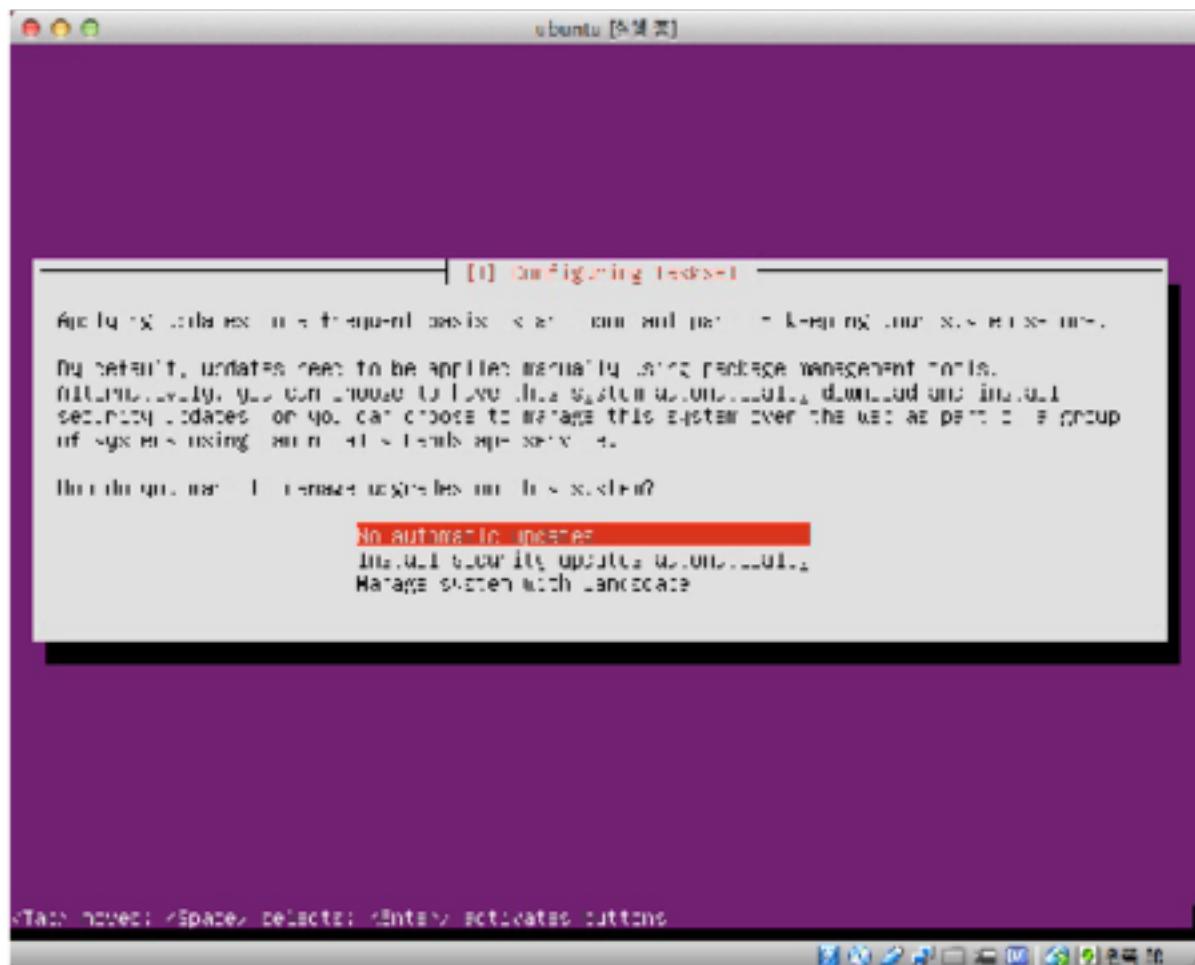
35. 설치 진행과정 (1)



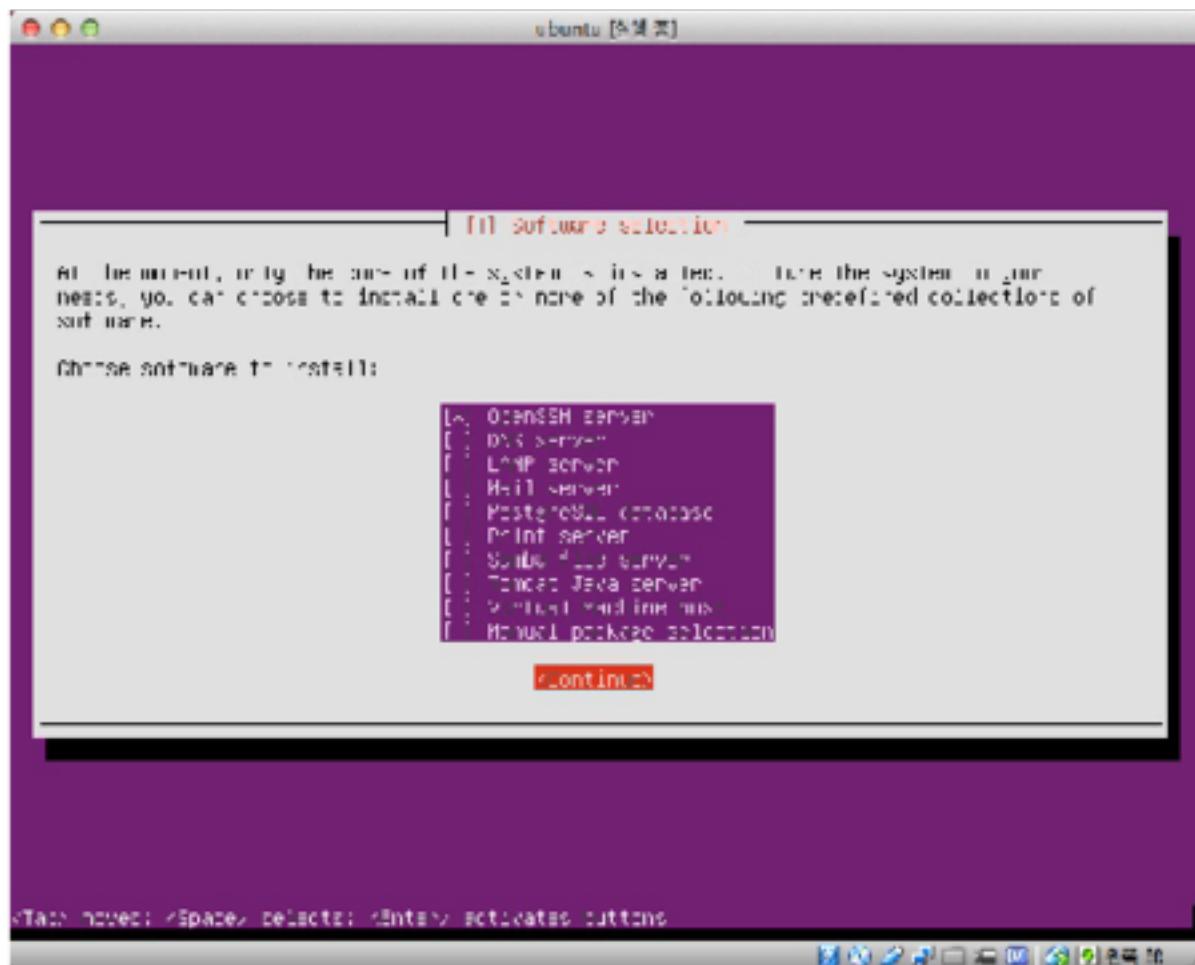
설치 진행과정 (2)



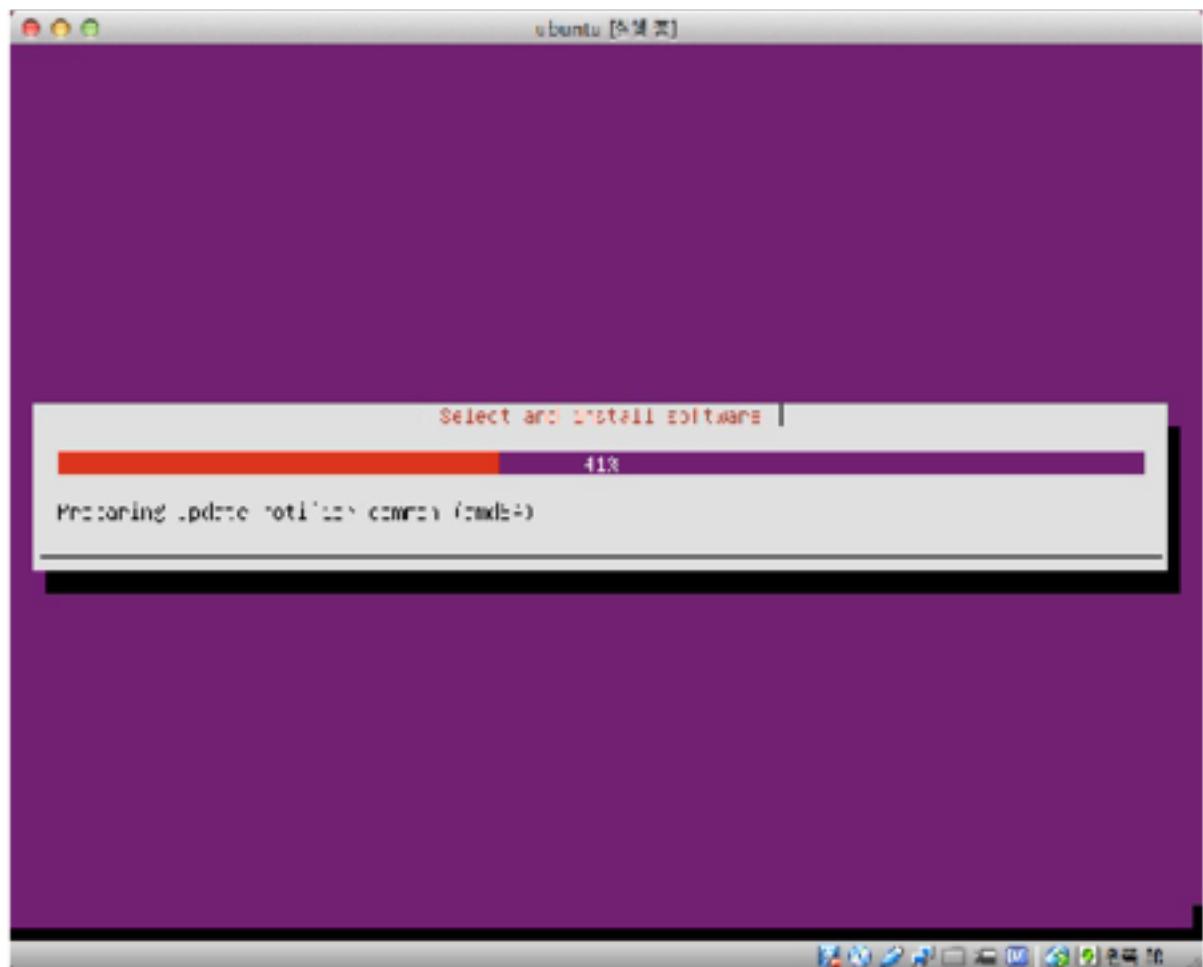
36. 시스템 업데이트는 수동으로 지정한다. 기본값으로 진행한다.



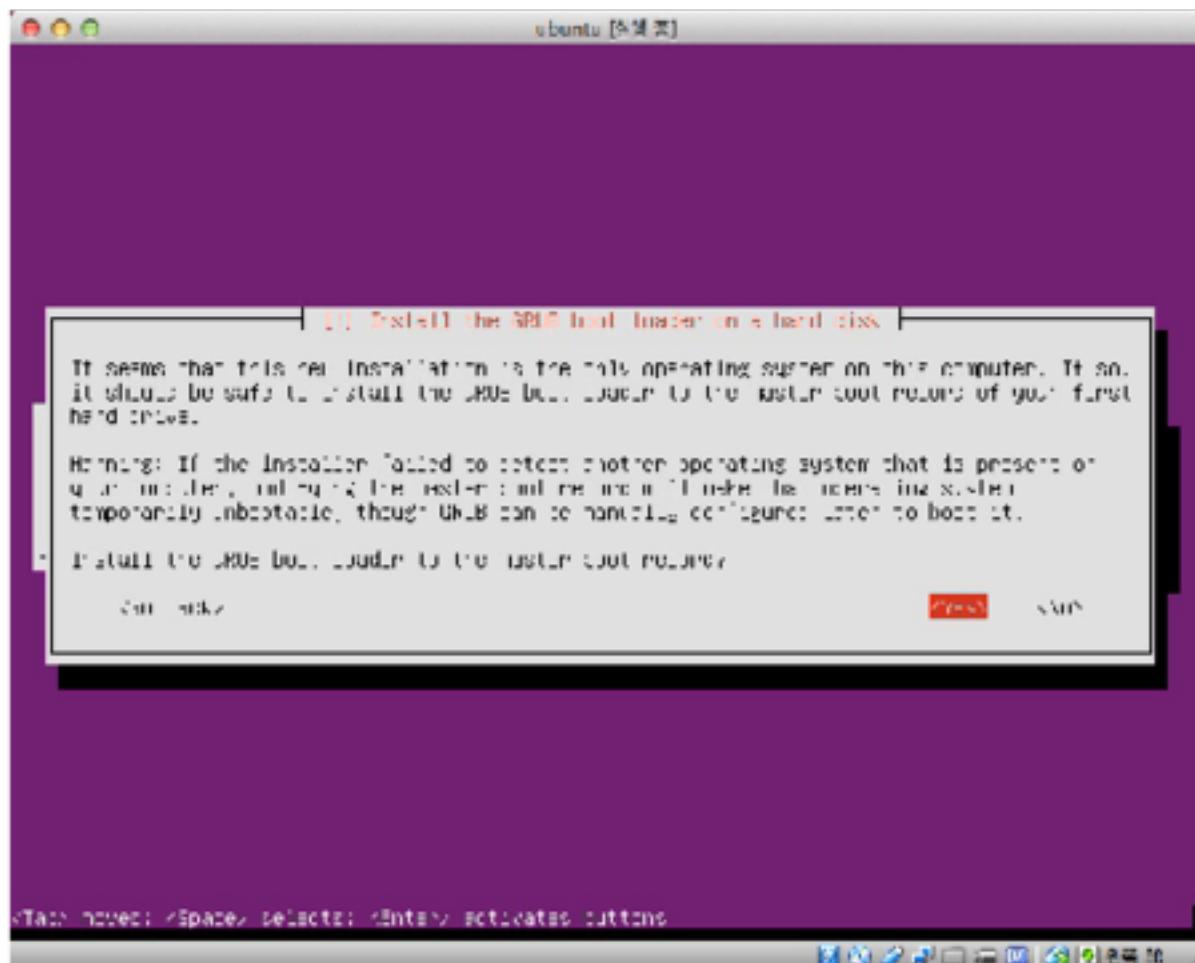
37. 설치할 소프트웨어는 OpenSSH server 만 선택(스페이스바를 클릭)한 후 계속한다.



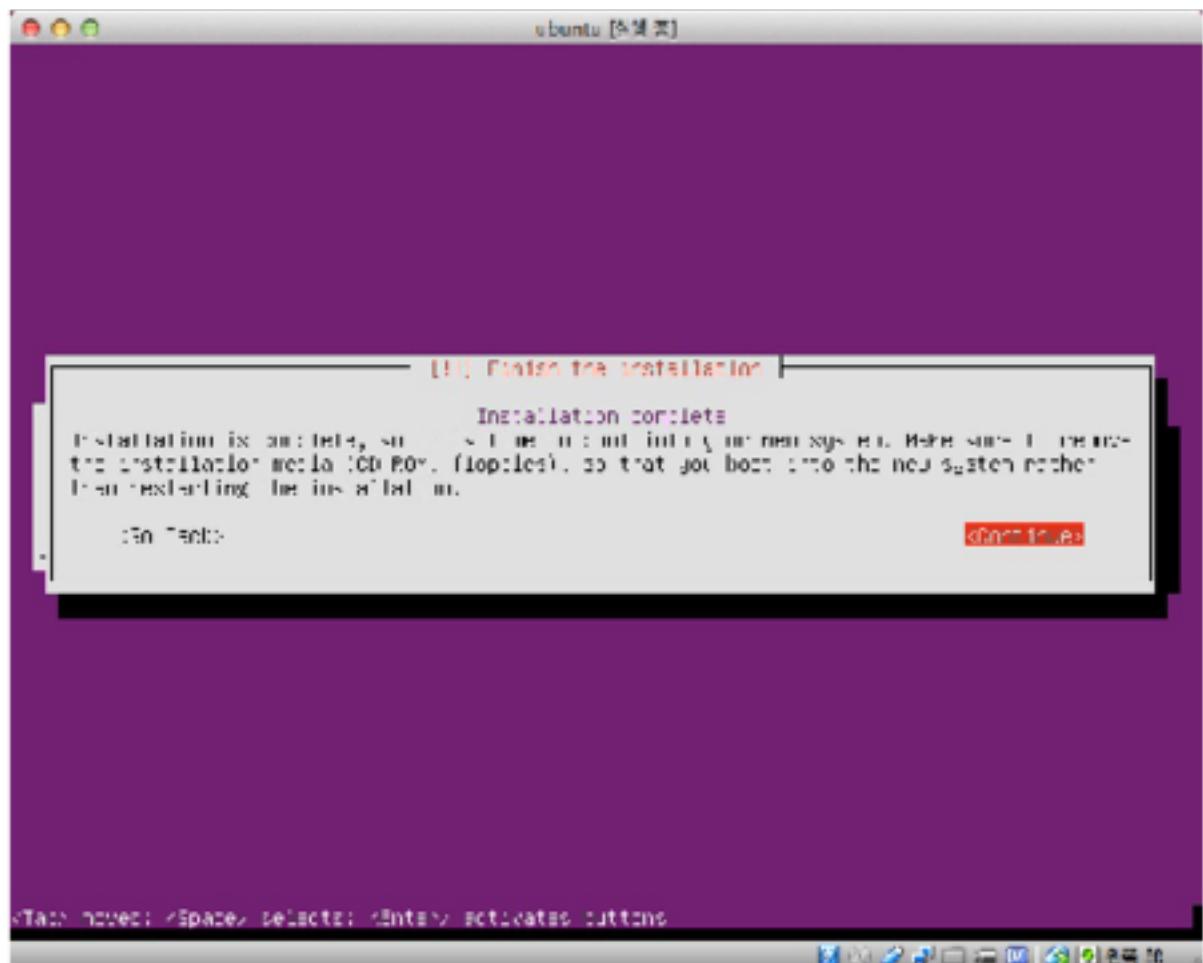
38. 설치 진행과정



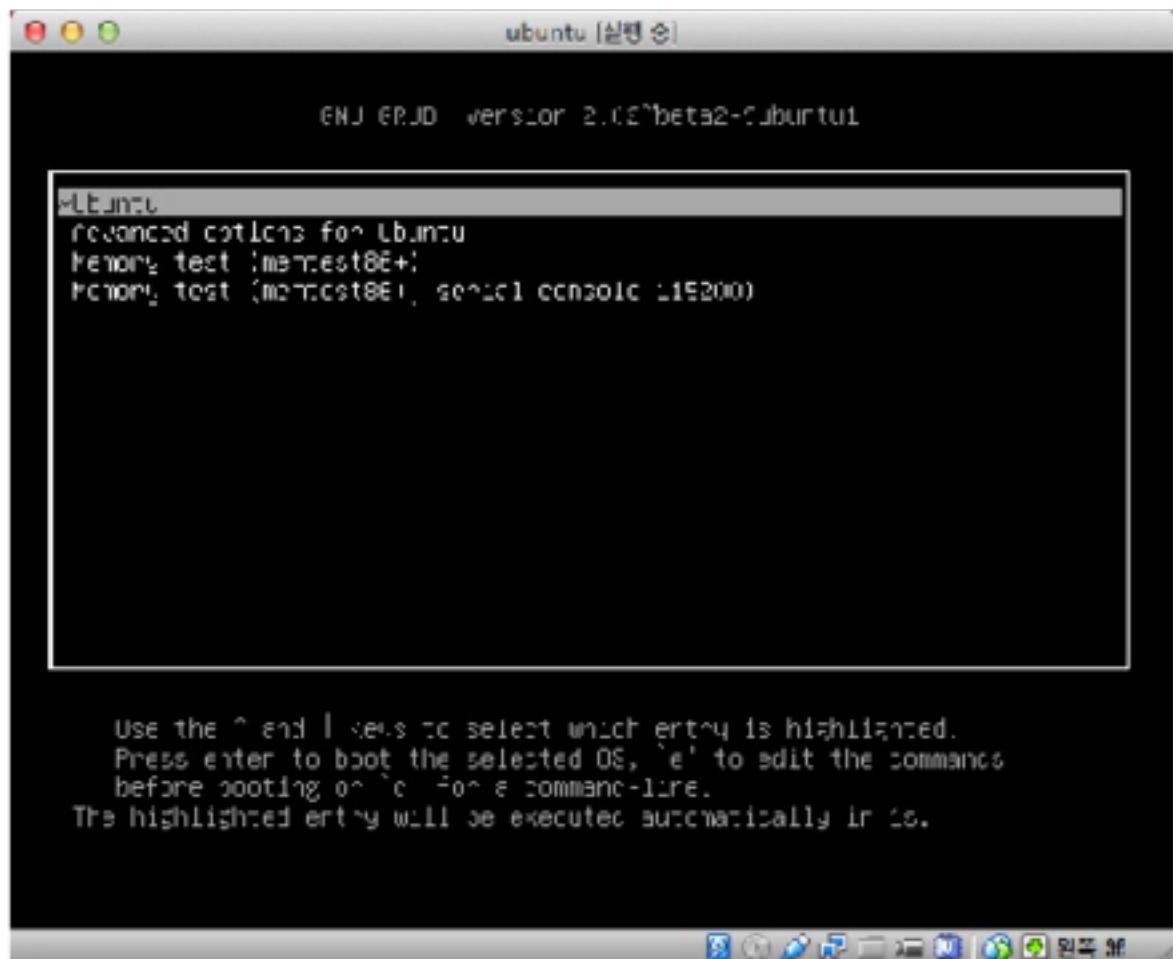
39. GRUB 부트로더 설치는 <Yes> 를 선택한다.



40. 설치완료

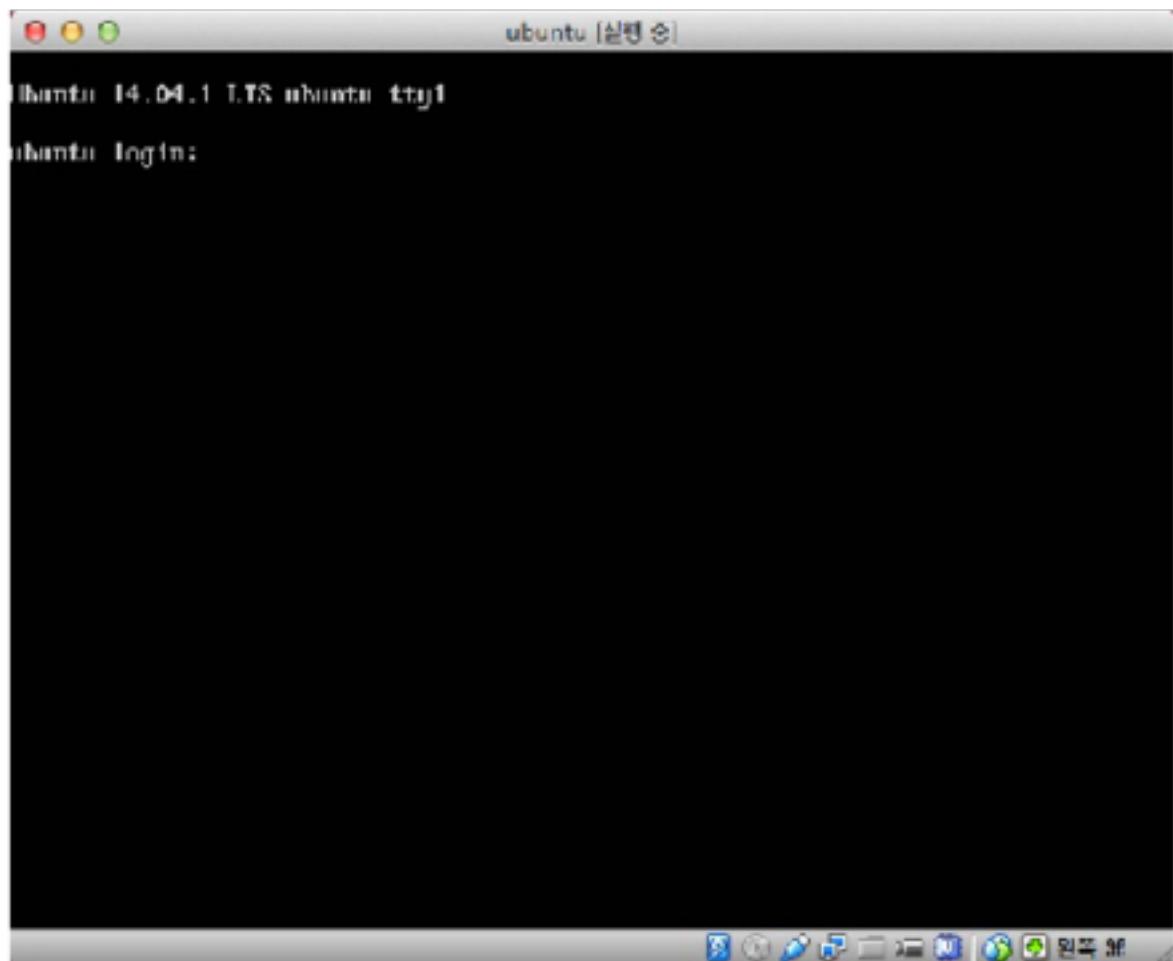


41. 시스템 부팅 시작 화면



42. 로그인 화면

아래와 같은 로그인 화면이 나타나면 등록한 유저 계정명과 암호를 이용하여 로그인 한다.



43. 로그인 후 화면

```
Ubuntu 14.04.1 LTS ubuntu tty1
ubuntu login: [REDACTED]
Password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)

 * Documentation: https://help.ubuntu.com/

 System information as of Thu Oct 9 20:43:32 KST 2014

 System load: 0.0              Memory usage: 3%   Processes:      79
 Usage of /: 14.7% of 5.51GB   Swap usage:  0%   Users logged in: 0

 Graph this data and manage this system at:
 https://landscape.canonical.com/

 0 packages can be updated,
 0 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Ubuntu: ~$
```

44. 가상 머신의 ip 주소

프롬프트에서 `ifconfig` 명령을 실행하면 아래와 같이 자신(가상머신 - 게스트 머신 : Ubuntu 14.04 서버)의 ip를 알 수 있다. 잘 기억해 두었다가 로컬머신(호스트머신)의 터미널에서 ssh로 가상머신으로 접속시 사용한다.

```
ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:93:8a:2c
          inet addr:192.168.2.5  Bcast:192.168.2.255  Mask:255.255.255.0
              inet6 addr: fe80::a00:27ff:fe93:8a2c%eth0  Scope:Link
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:1697 errors:0 dropped:3 overruns:0 frame:0
                  TX packets:27 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:102000 (102.0 KB)  TX bytes:2600 (2.6 KB)

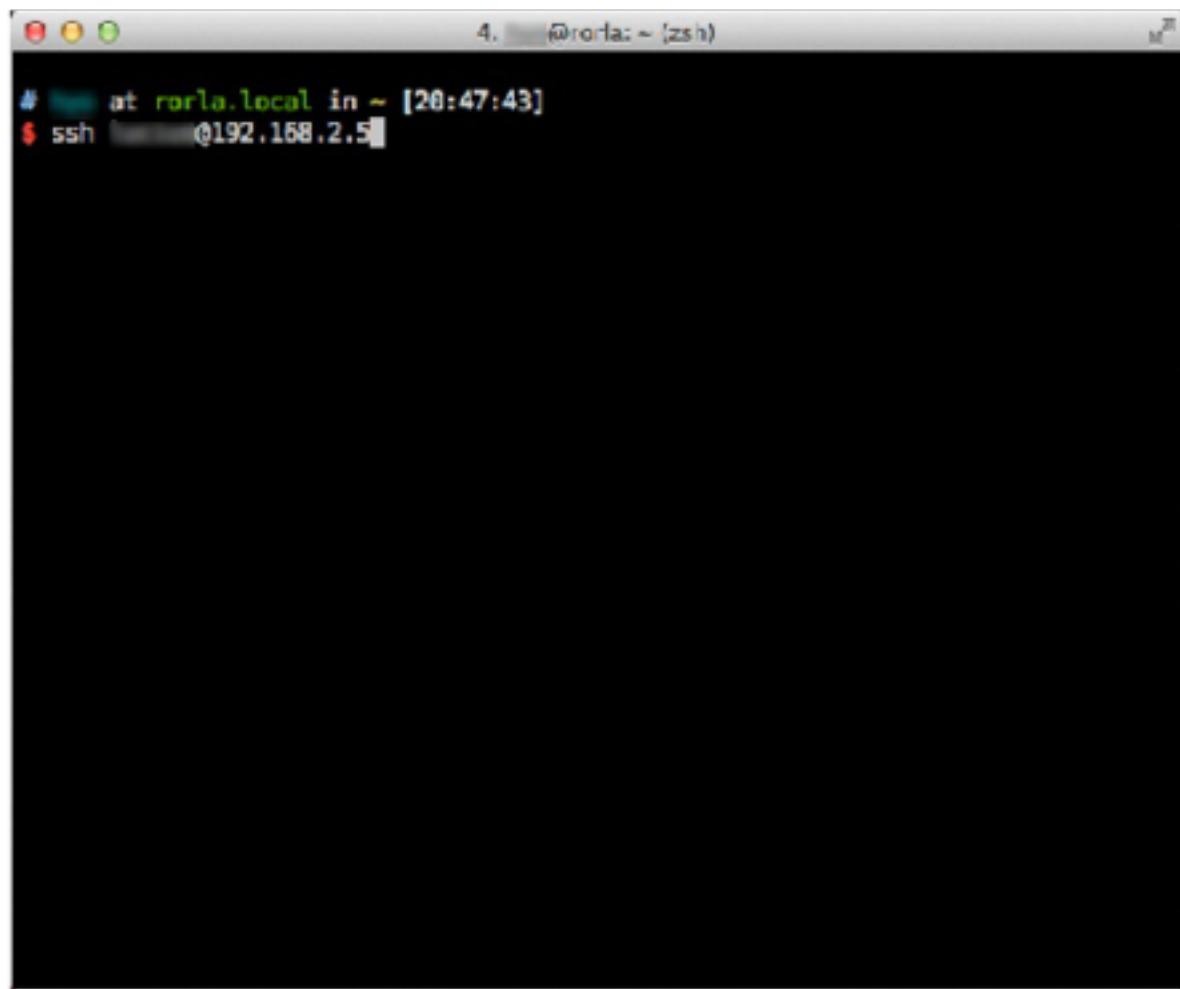
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
              inet6 addr: ::1/128 Scope:Host
                  UP LOOPBACK RUNNING  MTU:65536  Metric:1
                  RX packets:16 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:1184 (1.1 KB)  TX bytes:1184 (1.1 KB)

ubuntu:~$
```

45. 로컬 머신에서 ssh로 접속시도

지금부터는 로컬 머신에서 넬도의 터미널 창을 띄워 위에서 찾은 가상머신 ip주소로 접속한다.

```
$ ssh user-account@guest-pc-ip
```



A screenshot of a macOS terminal window. The title bar shows "4. @r0rla: ~ (zsh)". The main pane displays the command "ssh [REDACTED]@192.168.2.5" in green text. The background of the terminal is black.

최초 접속시에는 아래와 같이 호스트 인증 관련 메시지가 나타나며 `yes` 를 입력한다.

```
# at rorla.local in ~ [20:47:43]
$ ssh 192.168.2.5
The authenticity of host '192.168.2.5 (192.168.2.5)' can't be established.
RSA key fingerprint is [REDACTED]
Are you sure you want to continue connecting (yes/no)? yes
```

이어서 나타나는 화면에서는 암호를 입력한다.

A screenshot of a terminal window titled "4. ssh" with the subtitle "@192.168.2.5 (ssh)". The window shows the following text:

```
# at rorla.local in ~ [20:47:43]
$ ssh 192.168.2.5
The authenticity of host '192.168.2.5 (192.168.2.5)' can't be established.
RSA key fingerprint is ...
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.2.5' (RSA) to the list of known hosts.
@192.168.2.5's password: 
```

호스트에서 게스트 PC로 ssh 접속 후 나타나는 화면

접속한 게스트 PC(가상머신)의 IP를 시스템 정보에서 찾을 수 있다.

```
# at rorla.local in ~ [20:47:43]
$ ssh roorla@192.168.2.5
The authenticity of host '192.168.2.5 (192.168.2.5)' can't be established.
RSA key fingerprint is ...
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.2.5' (RSA) to the list of known hosts.
@192.168.2.5's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

 System information as of Thu Oct  9 20:44:08 KST 2014

 System load:  0.0          Processes:      78
 Usage of /:   14.7% of 5.51GB  Users logged in:    0
 Memory usage: 3%
 Swap usage:   0%          IP address for eth0: 192.168.2.5

 Graph this data and manage this system at:
 https://landscape.canonical.com/

 0 packages can be updated.
 0 updates are security updates.

 Last login: Thu Oct  9 20:44:08 2014
 @ubuntu:~$
```

서버 웹운영 환경 설정

이제부터는 [우분투 12.04 서버 세팅하기](#) 챕터의 내용 중 서버 설치과정을 보고 따라하면 된다. 중복되지만 아래에 삽입해 두었다.

서버 설치과정

```
# 시스템 업데이트
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get update

# 시스템에 한국어 설치
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install language-pack-ko language-pack-ko-base -
ubuntu@ubuntu-VirtualBox:~$ sudo vi /etc/default/locale
LANG="ko_KR.UTF-8"
LANGUAGE="ko_KR:ko:en_US:en"
ubuntu@ubuntu-VirtualBox:~$ sudo dpkg-reconfigure locales

# Build-essential 설치
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get -y install git curl build-essential openssl libs

# Nginx 서버 설치
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install -y nginx

# MySQL 서버 설치
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install -y mysql-server mysql-client libmysqlcli

# ImageMagick 이미지 를 설치
```

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get -y install libmagickwand-dev imagemagick  
# Ghostscript 설치  
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install ghostscript  
  
# Nodejs 설치  
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get -y install nodejs
```



배포용 계정 만들기

```
ubuntu@ubuntu-VirtualBox:~$ sudo adduser deployer
```

위의 명령으로 `deployer` 계정과 그룹이 생성되고 `deployer` 사용자는 `deployer` 그룹에 속하게 된다. 그리고 `/home/deployer/` 디렉토리가 계정 홈디렉토리로 생성된다.

이제 `deployer` 계정에 루트권한을 주기 위해서 `admin` 그룹으로 등록한다.

```
ubuntu@ubuntu-VirtualBox:~$ sudo addgroup admin  
ubuntu@ubuntu-VirtualBox:~$ sudo usermod -a -G admin deployer
```

Heroku(허로쿠)에 배포하기

공식 웹사이트 : <https://www.heroku.com>

허로쿠 서비스를 이용하면 Capistrano를 사용하지 않코도 레일스 프로젝트를 손쉽게 배포할 수 있다. 허로쿠 서비스는 개발자가 프로젝트 개발에만 집중할 수 있도록 해 준다. 배포 부분은 허로쿠 서비스가 알아서 대신해 준다.

허로쿠 준비부터 배포까지

- 허로쿠 공식 웹사이트를 방문하여 회원가입한다.
- 본인 계정의 Dashboard 의 하단에 있는 Create a new app 버튼을 클릭하여 새로운 어플리케이션을 생성한다. App 이름은 본인이 원하는 것으로 해도 무방하다.
- 로컬머신의 커맨드라인 쉘에서 허로쿠 툴벨트 (아래 참고)를 설치한다.
- Gemfile 에 허로쿠 배포에 필요한 두개의 경을 추가하고,

```
gem 'pg', group: :production
gem 'rails_12factor', group: :production
```

기존의 sqlite3 경은 아래와 같이 :group 옵션을 추가한다

```
gem 'sqlite3', group: :development
```

설치한다.

```
$ bundle install
```

- rails s 명령으로 개발모드에서 이상없이 작동하는 것이 확인되면 git 의 원격 저장소 heroku 를 생성한다.

```
$ git remote add heroku git@heroku.com:<heroku-app-name>.git
```

- 이제 아래와 같이 허로쿠로 배포작업을 시작한다.

```
$ git push heroku master
```

- 브라우저에서 <http://<heroku-app-name>.heroku.com> 으로 접속해서 확인한다.
- 자세한 내용은 [여기](#)를 참고하기 바란다.

허로쿠에서 레일스 4 프로젝트 배포하기

자세한 내용은 [여기](#)를 참고하기 바란다.

허로쿠 툴벨트 설치하기

로컬 운영체계에 맞는 [허로ku 플랫폼](#)을 설치한다. 이후에는 터미널에서 허로쿠에서 heroku 명령을 사용할 수 있게 된다

```
$ heroku
Usage: heroku COMMAND [--app APP] [command-specific-options]

Primary help topics, type "heroku help TOPIC" for more details:

addons      # manage addon resources
apps        # manage apps (create, destroy)
auth         # authentication (login, logout)
config       # manage app config vars
domains     # manage custom domains
logs         # display logs for an app
ps           # manage dynos (dynos, workers)
releases    # manage app releases
run          # run one-off commands (console, rake)
sharing      # manage collaborators on an app

(상략...)
```

pg 챔 추가하기

허로쿠로 배포하면 자동으로 Postgresql 데이터베이스를 사용하기 때문에 `Gemfile`에 pg 챔을 추가하고,

```
gem 'pg', group: :production
```

설치한다.

```
$ bundle install
```

허로쿠에서 App 생성하기

heroku dashboard Apps Databases Add-ons Docs Support 

Find apps...

You don't have any apps; deploy a sample application.

```
1. $ git clone git://github.com/heroku/ruby-sample.git  
2. $ cd ruby-sample  
3. $ heroku create  
4. $ git push heroku master  
5. $ heroku open
```

 Create a new app 

 Getting to know Heroku
To learn more about Heroku, find language specific tutorials, or read up on add-ons, visit the Heroku Dev Center.

생성 후 아래와 같은 결과를 보게 된다.

Your app, rcafe, has been created.

App URL: <http://rcafe.herokuapp.com/>
Git URL: <git@heroku.com:rcafe.git> 

Use the following code to set up your app for local development:

```
git clone git@heroku.com:rcafe.git -o heroku 
```

Suggested next steps

 Get started with Heroku.

 Add some collaborators.

 Check out some of our great add-ons.

 Finish up 

이제 **Finish up** 버튼을 클릭해서 종료한다.

heroku 원격 브랜치 생성하기

```
$ git remote add heroku git@heroku.com:rcafe.git
```

허로ku 배포를 위한 챕 추가

아래와 같이 챕을 추가하고,

```
gem 'rails_12factor', group: :production
```

설치 한다.

```
$ bundle install
```

배포 준비

```
$ git add .
$ git commit -m "작업을 내용을 커밋함"
$ git push origin # github에 푸시하기
```

허로ku에 실제 배포하기

```
$ git push heroku master # 허로ku에 푸시하기
```

The screenshot shows the Heroku Dashboard interface for the 'rcafe' application. At the top, there's a navigation bar with tabs for 'Resources', 'Activity', 'Access', and 'Settings'. Below this, the 'Dynos' section displays two processes: 'web' (using 'bin/rails server -p \$PORT -e \$RAILS_ENV') with 1 instance running at \$0.00, and 'worker' (using 'bundle exec rake jobs:work') with 0 instances running at \$0.00. The 'Add-ons' section shows one add-on, 'Heroku Postgres :: Silver', which is a Hobby-dev tier and free. There's also a 'Get Add-ons' button. At the bottom, there's an 'Apply Changes' button and a note about an estimated monthly cost of '\$0.00'.

브라우저에서 확인하기

```
$ open http://rcafe.herokuapp.com
```

AWS S3 사용하기

AWS S3 : Amazon Web Services Simple Storage Services

레일스 어플리케이션을 허로ku로 배포할 때의 문제점은 허로ku의 `ephemeral filesystem` 때문이다.
`ephemeral` 이란 단어는 단명의, 단 하루뿐인이란 의미를 가지는데, 미루어 짐작할 수 있듯이 업로드된 파일이 일정 시간 지나면 자동으로 삭제된다.

이에 대한 해결책으로 AWS S3 를 이용하면 허로ku로 배포된 레일스 어플리케이션에서 파일 업로드 후 지속적으로 이미지를 볼 수 있게 된다.

S3 의 모든 파일은, 디렉토리와 매우 흡사한 상위 레벨의 컨테이너로 동작하는, bucket 이라는 곳에 저장된다.
S3 로 파일을 보내면 하나의 bucket 에 속하게 되는데, bucket 명은 전체 아마존 시스템에서 유일해야 한다.

Access Key ID 와 Secret Access Key 가 있어야 S3 API 로 접근할 수 있다. 여기서 access 키는 S3 사용자 계정이고 secret 키는 비밀번호에 해당하기 때문에 다른 사람에게 노출되지 않도록 주의해야 한다.

S3 셋업

S3 를 사용하기 위해서는 AWS Credentials 와 파일 저장을 위한 bucket 을 확보해야 한다. 각각에 대해서는 아래를 참조하기 바란다.

Credentials

자세한 내용은 [여기](#) 를 참고하기 바란다.

Bucket

자세한 내용은 [여기](#) 를 참고하기 바란다.

Bucket Naming

자세한 내용은 [여기](#) 를 참고하기 바란다.

Carrierwave 챕 사용시 S3 셋업 방법

레일스에서는 S3 저장을 위한 aws-sdk 라이브러리를 사용할 수 있는데 carrierwave-aws 라는 carrierwave 용 챕이 있다. 따라서 Gemfile 에서 이미 등록한 carrierave 대신에 carrierwave-aws 챕을 추가하면 된다. 왜냐하면, 이 챕을 설치할 때 챕의 존성에 따라 carrierwave 와 aws-sdk 챕이 자동으로 설치되기 때문이다.

```
gem 'carrierwave-aws'
```

위와 같이 Gemfile 에 챕을 추가하고

```
$ bundle install
```

설치 한다.

단순 비교 테이블 [07/17/2013]

`fog` 챔을 이용하여 AWS S3 서비스를 사용할 수 있지만 아래와 같은 장점이 있어 `aws-sdk` 챔이 더 선호된다.

Library	Disk Space	Lines of Code	Boot Time	Runtime Deps	Develop Deps
fog	28.0M	133469	0.693	9	11
aws-sdk	5.4M	90290	0.098	3	8

사용법

`config/initializers/carrierwave.rb` 파일에 아래와 같이 추가한다.

```
CarrierWave.configure do |config|
  config.storage      = :aws
  config.aws_bucket = ENV['S3_BUCKET_NAME']
  config.aws_acl     = :public_read
  config.asset_host = 'http://example.com'
  config.aws_authenticated_url_expiration = 60 * 60 * 24 * 365

  config.aws_credentials = {
    access_key_id: ENV['AWS_ACCESS_KEY_ID'],
    secret_access_key: ENV['AWS_SECRET_ACCESS_KEY']
  }
end if Rails.env == 'production'
```

`config.asset_host` 값을 AWS S3 의 해당 bucket 의 Endpoint 주소로 변경한다. (예, `https://s3-ap-northeast-1.amazonaws.com/<bucket-name>`)

현재 AWS S3 bucket 의 Endpoint 주소의 링크의 오류가 있으므로 위의 예를 복사해서 `<bucket-name>`만 변경해서 사용하도록 한다.

위에서 사용한 'S3_BUCKET_NAME', 'AWS_ACCESS_KEY_ID', 'AWS_SECRET_ACCESS_KEY'를 시스템 환경변수로 등록하기 위해서 `~/.bash_profile` 또는 `~/.zshrc` 파일을 열어서 아래와 같이 추가한다.

```
export S3_BUCKET_NAME=<user-bucket-name>
export AWS_ACCESS_KEY_ID=<user-access-key>
export AWS_SECRET_ACCESS_KEY=<user-secret-key>
```

허로ku로 배포시에는 아래와 같이 추가 작업을 해 준다.

```
$ heroku config:set S3_BUCKET_NAME=<user-bucket-name>
$ heroku config:set AWS_ACCESS_KEY_ID=<user-access-key>
$ heroku config:set AWS_SECRET_ACCESS_KEY=<user-secret-key>
```

그리고 파일 업로드 클래스 파일에 아래와 같이 storage 옵션 조건을 추가한다.

```
if Rails.env == 'production'
```

```
storage :aws
else
  storage :file
end
```

`paperclip` 챔을 사용할 때는 [paperclip-s3](#)를 참고하기 바란다.

`fog` 챔 사용법은 [여기](#) 를 참고하면 도움이 된다.

개발 환경 변수를 관리하는 dotenv 챔

<https://github.com/bkeepers/dotenv>

dotenv 챔은 개발모드에서 .env 파일로부터 ENV로 환경변수를 로드하기 위한 것이다. 따라서 운영(배포)모드에서는 사용할 때는 별도의 `dotenv-deployment` 챔을 사용하면 된다.

사실 다수의 프로젝트를 실행하는 경우 개발 머신이나 CI 서버에서 환경변수를 사용하는 것이 반드시 실용적인 것은 아니다. 이런 경우 dotenv를 사용하면 개발모드가 시작될 때 .env 파일을 ENV로 변수들을 로드하게 된다.

설치

Gemfile에 아래와 같이 추가하고,

```
gem 'dotenv-rails', groups: [:development, :test]
```

설치한다.

```
$ bundle install
```

사용법

프로젝트의 루트에 .env 파일을 만든 후 아래와 같이 어플리케이션 설정을 위한 값들을 추가한다.

```
S3_BUCKET=YOURS3BUCKET  
SECRET_KEY=YOURSECRETKEYGOESHHERE
```

또는 yaml 포맷도 지원하여 아래와 같이 작성해도 된다.

```
S3_BUCKET: yamlstyleforours3bucket  
SECRET_KEY: thisisalsoanokaysecret
```

이제 어플리케이션이 로드될 때마다 아래와 같이 이 변수값을 ENV에서 사용할 수 있게 된다.

```
config.fog_directory = ENV['S3_BUCKET']
```

.env 파일의 커밋

개발모드에서 사용하는 환경설정 값만을 .env 파일에 저장하고 저장소로 커밋하도록 권한다. 즉, 개발 환경에서 사용하는 모든 보안사항은 다른 배포환경에서 사용하는 것과 다르다는 것을 확인해야 한다. 같은 프로젝트에 참여하는 다른 개발자들도 동일한 개발 환경을 별도의 설정없이 바로 사용할 수 있게 되는 것이다.

그러나 주의할 것은 공개 저장소로 푸시해서는 안 된다는 것에 유의하기 바란다. 즉, github의 공개 저장

소에 커밋할 경우 `.env` 파일내의 환경 변수에 포함된 보안데이터가 공개되기 때문에 반드시 `private` 거
장소로만 푸시해서 사용해야 한다.

맥전용 랙서버 POW

POW는 Basecamp에서 만든 무료 맥전용 랙서버로 자체 HTTP 서버와 DNS 서버를 포함하고 있다.

맥OSX에서 별다른 설정없이 1분이내에 로컬 웹서버를 구동하여 포트사용없이 여러 개의 레일스 프로젝트를 서빙할 수 있다. (아래의 이미지를 클릭하면 사용법에 대한 동영상을 볼 수 있다.)



즉, `http://localhost:3000`과 같이 포트를 추가하지 않고 "프로젝트 폴더명" + ".dev"와 같은 도메인으로 해당 프로젝트에 접근할 수 있다.

초간단 설치

```
$ curl get.pow.cx | sh
  % Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload Total   Spent   Left Speed
100  6887  100  6887    0      0  14944      0 --:--:-- --:--:-- --:--:-- 14939
*** Installing Pow 0.4.3...
*** Installing local configuration files...
/Users/hyo/Library/LaunchAgents/cx.pow.powd.plist
*** Installing system configuration files as root...
Password:
/etc/resolver/dev
/Library/LaunchDaemons/cx.pow.firewall.plist
*** Starting the Pow server...
*** Performing self-test...
*** Installed

For troubleshooting instructions, please see the Pow wiki:
https://github.com/37signals/pow/wiki/Troubleshooting

To uninstall Pow, `curl get.pow.cx/uninstall.sh | sh`
```

프로젝트 링크 생성하기

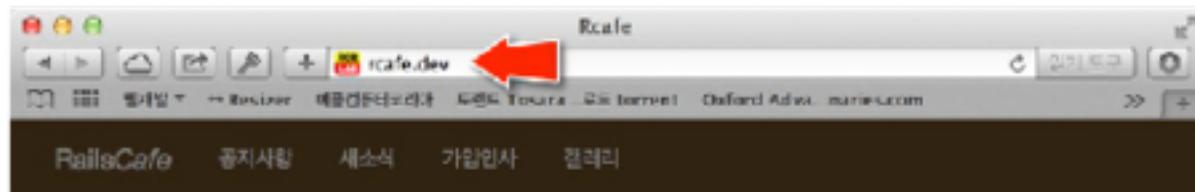
위에서 언급한 바와 같이 웹프로젝트명에 ".dev"를 붙인 URI로 연결하기 위해서는 아래와 같이 `~/.pow` 디렉토리로 이동하여 아래와 같이 프로젝트 절대경로에 대한 링크를 생성한다.

```
$ cd ~/.pow  
$ lnk -s /Users/user-account/rcafe  
$ ls -al  
total 8  
drwxr-xr-x 3 hyo staff 102 5 21 08:13 ./  
drwxr-xr-x 5 hyo staff 170 5 21 06:45 ../  
lrwxr-xr-x 1 hyo staff 23 5 21 07:18 rcafe@ -> /Users/user-account/rcafe
```

브라우저에서 연결하기

터미널에서 `rails server` 로 로컬서버를 구동하지 않고도 아래와 같이 웹애플리케이션에 연결된다.

```
$ open http://rcafe.dev
```



Welcome to RCafe

이 페이지는 'RCafe 프로젝트'의 Welcome 페이지입니다.

다른 컴퓨터에서 연결하기

동일한 LAN 환경에 있는 다른 컴퓨터에서 개발머신의 Pow 가상호스트에 접근할 수 있다. 이것은 모바일 디바이스나 윈도우 또는 리눅스 VM 상에서 프로젝트를 테스트하고자 할 때 유용하다.

`.dev` 도메인은 개발머신에서만 동작한다. 그러나 `.xip.io` 도메인을 사용하면 원격에서 Pow 가상호스트로 접근할 수 있다.

이를 위해서 개발머신의 LAN IP 주소를 프로젝트명과 `.xip.io` 사이에 추가하여 접속하면 된다. 모바일 디바이스나 윈도우/리눅스 가상머신에서 아래의 주소로 접속하면 된다.

`http://rcafe.xxx.xxx.xx.xxx.xip.io`

Foreman과 함께 사용하기

로컬에서 레일스 서버와 함께 `delayed_job`과 같은 별개의 프로세스들을 동시에 실행해야 할 경우에는, `foreman`을 사용하면 편리하다.

[여기](#)를 참고하면 쉽게 설정할 수 있다. 이를 위해서는 먼저 로컬에 `foreman`이 설치되어 있어야 한다.

1. 어플리케이션 루트 디렉토리에 `.powenv` 파일을 생성하고 아래와 같은 내용을 추가한다.

```
# In your app's root.  
# Make Pow!! export all the env variables contained in the .env file used by Foreman.  
export $(cat .env)
```

2. 역시 어플리케이션 루트 디렉토리에 `.configuration.sh` 파일을 생성하고 아래와 같은 내용을 추가한다.

```
# This should be run once only  
cd /path/to/myapp  
echo 7000 > .port # Whatever port your app server is using  
ln -s $PWD/.port ~/`basename $PWD`  
echo "port: 7000" > .foreman  
  
# Then start your app normally everytime with:  
# > foreman start  
# Voila! App will be available at http://myapp.dev
```

여기에서 수정이 필요한 곳은 3군데다. `/path/to/myapp`를 실제 어플리케이션 디렉토리로 변경하고, 포트를 원하는 것으로 변경하고자 한다면 7000이란 숫자를 두군데에서 변경하면 된다.

3. 이제 `foreman`을 아래와 같이 실행한다.

```
$ foreman start
```

4. 다음은 브라우저에서 `http://{프로젝트명}.dev`로 접속해 본다.

메뉴얼에 대한 자세한 내용을 원하면 <http://pow.cx/manual.html>를 참고하기 바랍니다.

추천 웹사이트

루비온레일스

- Ruby on Rails Official Website <http://rubyonrails.org>
- ROR Lab. <http://rorlab.org>

루비

- RubyMonk <https://rubymonk.com/>

추천 블로그

- Ruby on Rails Official Blog <http://weblog.rubyonrails.org>
- Happy Rails <http://happyrails.rorlab.org>
- 루비 프로그래밍 언어 한글 문서 <http://ruby-korea.github.io/>

추천도서

[원서]

- [Ruby on Rails Tutorial, Michael Hartl](#)
: READ ONLINE FREE은 무료이고, eBook, screencasts은 유료.
- [Agile Web Development with Rails 4, Sam ruby, Dave Thomas, David Heinemeier Hansson](#) (유료)
- [Learn Ruby on Rails, Daniel Kehoe](#) (유료)

[번역서]

- [레일스와 함께하는 매자일 웹개발. 개정판](#) (원저:Agile Web Development with Rails 4/E) 2012년 05월29일
- [루비를 깨우치다](#) (원저:Ruby Under a Microscope, 뱃 쇼네시 지음, 최효성 옮김), 출판사 [비제이퍼블릭](#), 구매 : 알리딘, yes24, 2014년 8월 29일 출간.
- [루비로 배우는 객체지향 디자인](#) (원저:Practical Object-Oriented Design in Ruby, 샌디 메츠 지음, 박건하 옮김), 출판사 인사이트, 구매: Yes24, 교보문고, 알리딘, 인터파크, 2014년 11월 28일 출간.

추천동영상

- [Railscasts.com](#)
- [Rails for Zombies Redux](#)
- [Rails 4: Zombie Outlaws](#)
- [Rails for Zombies 2](#)
- [Railsbest.com](#)

[국내자료]

- [RORLab Podcast](#)
- [RORLab Youtube Channel](#)

변경내역

v1.0.6

"5장. 프로젝트 따라하기" 본문을 따라하면서 설명이 부족했던 부분을 보완하였고 `friendly_id` 챔 사용을 삭제하였습니다. 그리고 소스코드 저장소를 <http://github.com/iorlakr/rcale> 로 변경하여 소스코드를 업데이트했습니다.

v1.0.5

: 브랜치 기능을 이용하여 버전 관리를 다시 시작합니다. 2014년12월3일 웹에디터에 서도 브랜치 기능을 사용할 수 있는 것을 뒤늦게 알았습니다. master 브랜치에 develop 브랜치를 추가하여 업데이트 작업을 하고 최종 버전 업을 master 브랜치로 마지하는 방식으로 진행합니다.

버전관리 중단합니다.

: 2014년11월30일 Gitbook Webeditor가 발표되면서 웹에디터에서 수정후 저장하면 바로 Gitbook에 반영이 됩니다. 따라서 더 이상 추가 업데이트 부분을 별도의 버전으로 발표하는 것이 의미가 없게 되었습니다.

v1.0.4

- 부록에 [우분투 14.04 서버 세팅하기 (Virtual Box)] 챕터 추가함. - [iorlab](#) 2014년 10월 9일 한글날.
- [5.17. 서버로 배포하기] : 서버의 tail log 기능 추가함 - [iorlab](#) 2014년 10월 9일
 - Gemfile : `gem 'capistrano-rails-tail-log'` 추가
 - Capfile : `require 'capistrano/rails_tail_log'` 추가
 - deploy.rb : `set :rails_env, "production"` 추가

v1.0.3

추가내용

- [추천 웹사이트 및 블로그]
- [5.17 서버로 배포하기 / Gemfile의 추가 : rb-readline 챔 추가함. [참고문서](#)]
- [5.17 서버로 배포하기 / 배포하기 : 버그 해결법 추가.] - [iorlab](#) 2014년 9월 28일

v1.0.2

포맷변경

- PDF 파일로 변환될 때 소제목 시작시에 새로운 페이지로 넘어가는 것을 수정함. [Lucius Choi](#) 2014년 9월 11일

오타수정

- [코멘트 변경하기], [Dev_senna](#) 2014년 8월 22일

1. 자바스크리pt가... => 자바스크립트가

v1.0.1

오탏수정

- [코멘트 변경하기], rkjun 2014년 8월 9일

```
1. .... 자동으로 comments라는 테이블명으로 테이블이 생성된다는 것이다.  
생선 => 생성  
2. app/controllers/comment_controller.rb 파일을 열어 보면 아래와 같다.  
comment_controller.rb => comments_controller.rb
```

- [코멘트 변경하기], 흐링 2014년 8월 15일

```
1. 컨트롤러에 def comment_params 데 params 오타였습니다. parms 로 되어있네요.
```

변경 및 추가

- [환경내역] 메뉴 추가, iorlab 2014년 8월 9일