**Jenkins 설치/사용 매뉴얼**

**(Ubuntu)**

Innovation&QE Part, SW Platform Lab

Chief Technology Officer

**Copyright** ⓒ 2011 LG Electronics Co.,LTD.

**Copyright © 2011 LG Electronics Co, Ltd. All Rights Reserved.**

Though every care has been taken to ensure the accuracy of this document, LG Electronics Co, Ltd. cannot accept responsibility for any errors or omissions or for any loss occurred to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

LG Electronics Co, Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of LG Electronics Co, Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

|  |  |
| --- | --- |
| Software Information | Document Information |
| Innovation&QE Part, SW Platform Lab  Chief Technology Officer  LG Electronics Co., Ltd.  Seocho R&D Campus, Seocho-Gu,  Seoul, Korea 137-724  Contact us: @lge.com | Innovation&QE Part, SW Platform Lab  Chief Technology Officer  LG Electronics Co., Ltd.  Seocho R&D Campus, Seocho-Gu,  Seoul, Korea 137-724  Contact us: @lge.com |

About This Document

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Comment | Author | Approval |
| 0.1 | 2011-03-09 |  | Hyejin Oh, Bokyung Kim, Jeonggon Seo |  |
| 0.2 | 2011-8-24 | 1. Redmine- 설정  2. Jenkins project 분리 | Eunyoung Lee, Chiwook jeong |  |
| 0.21 | 2012-1-11 | 1. Trouble shooting | Eunyoung Lee, Chiwook jeong |  |
| 0.3 | 2012-02-27 | 1. Jenkins Plugin : 기타 추가 | Hyejin.oh |  |
| 0.31 | 2012-03-09 | 1. Jenkins LDAP 수정  2. Role strategy plugin 사용법 수정 | Eunyoung Lee  Sol.namgung |  |

Purpose

Ubuntu에 Jenkins과 Plugins(N’SIQ Collector, N’SIQ CppStyle, FIndbugs, Redmine)을 설치/사용하는 방법에 대한 가이드를 제공한다.

Scope

Audience

Contents

[1 Jenkins 설치 7](#_Toc319068262)

[1.1 소개 7](#_Toc319068263)

[1.1.1 Jenkins 동작 원리 7](#_Toc319068264)

[1.2 설치 8](#_Toc319068265)

[1.2.1 Java 설치 8](#_Toc319068266)

[1.2.2 Tomcat 8](#_Toc319068267)

[1.2.2.1 User 별 단독 Tomcat 설치 12](#_Toc319068268)

[1.2.2.2 단독 Tomcat 설치 12](#_Toc319068269)

[1.2.2.3 Tomcat 복사 12](#_Toc319068270)

[1.2.2.4 환경 설정 13](#_Toc319068271)

[1.2.2.5 설정 확인 15](#_Toc319068272)

[1.2.3 Jenkins 17](#_Toc319068273)

[1.2.4 Ant 20](#_Toc319068274)

[1.2.4.1 소개 20](#_Toc319068275)

[1.2.4.2 설치 20](#_Toc319068276)

[1.2.4.3 설정 21](#_Toc319068277)

[1.2.4.4 실행 결과 23](#_Toc319068278)

[1.2.5 Maven 24](#_Toc319068279)

[1.2.5.1 소개 24](#_Toc319068280)

[1.2.5.2 설치 24](#_Toc319068281)

[1.2.5.3 설정 25](#_Toc319068282)

[1.2.5.4 실행 결과 26](#_Toc319068283)

[1.3 설정 27](#_Toc319068284)

[1.3.1 프로젝트 공통 설정 27](#_Toc319068285)

[1.3.2 프로젝트 빌드 설정 31](#_Toc319068286)

[1.3.2.1 일반 shell 빌드 31](#_Toc319068287)

[1.3.2.2 Ant 빌드 33](#_Toc319068288)

[1.3.2.3 ~~maven 빌드~~ 36](#_Toc319068289)

[1.3.3 분산빌드 환경 구축 37](#_Toc319068290)

[1.3.3.1 분산빌드 구조 37](#_Toc319068291)

[1.3.3.2 Jenkins에서 master&slave 설정 37](#_Toc319068292)

[1.3.4 LDAP 설정 39](#_Toc319068293)

[1.3.4.1 사전 작업 39](#_Toc319068294)

[1.3.4.2 설정 39](#_Toc319068295)

[1.4 실행 결과 43](#_Toc319068296)

[1.5 Trouble shooting 46](#_Toc319068297)

[1.5.1 Tomcat log 확인 방법 46](#_Toc319068298)

[1.5.2 Tomcat 재 설치 방법 46](#_Toc319068299)

[1.5.3 Plug-in 충돌에 의한 지속적인 tomcat의 down 대책 47](#_Toc319068300)

[1.5.4 AWT관련 Headless error 대책 48](#_Toc319068301)

[2 Jenkins Plugin 설치 : N’SIQ Collector 51](#_Toc319068302)

[2.1 소개 51](#_Toc319068303)

[2.2 설치 51](#_Toc319068304)

[2.3 설정 51](#_Toc319068305)

[2.4 Window용 53](#_Toc319068306)

[2.5 실행결과 53](#_Toc319068307)

[3 Jenkins Plugin : N’SIQ CppStyle 56](#_Toc319068308)

[3.1 소개 56](#_Toc319068309)

[3.2 설치 56](#_Toc319068310)

[3.2.1 공통 설치 56](#_Toc319068311)

[3.3 설정 58](#_Toc319068312)

[3.3.1 Filefilter.txt 설정 58](#_Toc319068313)

[3.3.2 Linux 용 59](#_Toc319068314)

[3.3.3 Window용 61](#_Toc319068315)

[3.3.4 3.2.3 N’SIQ CppStyle 리포팅 설정 62](#_Toc319068316)

[3.4 실행 결과 64](#_Toc319068317)

[4 Jenkins Plugin 설치 : ~~Findbugs~~ 65](#_Toc319068318)

[4.1 소개 65](#_Toc319068319)

[4.2 설치 65](#_Toc319068320)

[4.2.1 플러그인 설치 65](#_Toc319068321)

[4.2.2 Findbugs 설치 66](#_Toc319068322)

[4.3 설정 68](#_Toc319068323)

[4.3.1 설정build.xml 설정 68](#_Toc319068324)

[4.3.2 Jenkins설정 70](#_Toc319068325)

[4.4 실행 결과 72](#_Toc319068326)

[5 Jenkins Plugin : Redmine 74](#_Toc319068327)

[5.1 소개 74](#_Toc319068328)

[5.2 설치 74](#_Toc319068329)

[5.2.1 Window 용 74](#_Toc319068330)

[5.2.2 Linux용 75](#_Toc319068331)

[5.3 설정 76](#_Toc319068332)

[5.4 실행 결과 78](#_Toc319068333)

[5.4.1 프로젝트 생성 및 일감등록 78](#_Toc319068334)

[5.4.2 Jenkins 연동 81](#_Toc319068335)

[5.4.3 Jenkins translate 오류 해결 82](#_Toc319068336)

[6 Jenkins Plugin : 기타 83](#_Toc319068337)

[6.1 Sidebar Links 설치 83](#_Toc319068338)

[6.2 Jenkins SSH plugin설치 83](#_Toc319068339)

[7 Appendix 84](#_Toc319068340)

[7.1 리눅스 (Ubuntu)관련 Tip 84](#_Toc319068341)

[7.1.1 Ubuntu 네트워크 설정 84](#_Toc319068342)

[7.1.2 Putty접속이 안될경우 84](#_Toc319068343)

[7.1.3 MySQL의 database위치 변경 방법 85](#_Toc319068344)

[7.1.4 외부 서버로 파일 보내는 방법 86](#_Toc319068345)

[7.2 Maven 86](#_Toc319068346)

[7.2.1 Maven POM 파일의 구조 86](#_Toc319068347)

[7.2.2 Maven Plugins 87](#_Toc319068348)

[7.2.3 Maven POM 파일의 설정 방법 93](#_Toc319068349)

[7.2.3.1 일반적 설정 94](#_Toc319068350)

[7.2.3.2 Build Plugin 설정 97](#_Toc319068351)

[7.2.3.3 Reporting Plugin 설정 101](#_Toc319068352)

# Jenkins **설치**

## 소개

|  |  |  |
| --- | --- | --- |
| **제품** | **내 용** | |
| **Jenkins** | 플랫폼 유연성 | Servlet Container (War파일) , Stand-alone |
| Java Builder | Ant, Maven 2, Kundo |
| SCM | 기본제공: CVS, Perforce, Subversion  플러그인 제공: AccuRev, Bazaar, BitKeeper, Clearcase, CMVC, Git, Mercurial, PVCS SourceSafe, StarTeam, TFS |
| 알림 기능 | Android, E-mail, Google Calendar, IRC, XMPP, RSS, Twitter |
| IDE 연동 | Eclipse, IntelliJ IDEA, NetBeans |
| 버그 관리 연동 | Bugzilla, Google Code, JIRA, Mantis, Trac |
| Test tool 연동 | Junit, Cobertura, Clover |
| 라이선스 및 | 무료 오픈소스 |

* 자바로 개발된 CI 시스템
* 쉬운 설치와 실행 (Servlet Container와 Stand-alone 방식으로 실행)
* 웹 기반의 운영환경 (직관적인 UI)
* 다양한 기능을 편리하게 확장 (약 140여개의 플러그인 제공)
* 활발한 커뮤니티 활동
* 오픈소스
* MIT 라이선스 (사용, 수정, 배포 가능)

Jenkins은 Continuous Integration(점진적 통합)을 지원하는 도구로서 여러 개의 Java Project 또는 File들을 Build를 편리하고 쉽게 할 수 있도록, Commit에 따른 자동 build를 제공하여 Source Code가 Source관리 시스템에 Commit이 되었을 때 마다 CI 툴이 이를 감지하고 자동으로 build를 수행하도록 설정할 수 있다. 또한 일일 Check out과 build가 가능하여 개발자가 출근 후 Source 관리 시스템에서 최신 Code를 내려받고, 출근 전에 현재 Code를 Source관리 시스템에 저장함으로써 Source Code에 대한 무결성을 유지한다.

### Jenkins 동작 원리

Local

Source

Local

Source

Original

Source

Hudson

Source

SVN

Repository

SVN

Hudson

* Original Source를 SVN에 Import한다.
* Local에서는 SVN에 Import한다
* 각 Local에서 작업된 Source를 SVN에 등록한다
* Jenkins에서 변경된 Source를 통하여 build 및 배포를 수행한다.

## 설치

### Java 설치

1. 필수 **패키지 다운로드**

$sudo add-apt-repository ppa:sun-java-community-team/sun-java6

$sudo apt-get update

$sudo apt-get install sun-java6-jdk

$sudo update-alternatives --config java

설치하고자 하는 버전의 java를 선택한다.

1. PATH 설정(/etc/profile)

JAVA\_HOME=/usr/lib/jvm/java-6-sun

PATH=$JAVA\_HOME/bin:$PATH

export JAVA\_HOME

1. Java가 설치되어 있는 위치를 확인할 수 없다면
   1. Find 명령어를 사용해서 java가 설치되어 있는 폴더를 찾는다.

$cd /

$find –name java

우분투 10.04 이후 부터는 apt-get, update 자동으로 되지 않는다. 그렇기 때문에 우분투를 10.10로 업그레이드 한 후 위의 명령어를 설정해주면 쉽게 java 패키지가 설치 된다. 또한 우분투 10.04 부터는 openjdk가 설치가 되었기 때문에 설치하고자 하는 버전의 자바를 update-alternatives --config java를 이용하여 버전을 설치해야 된다

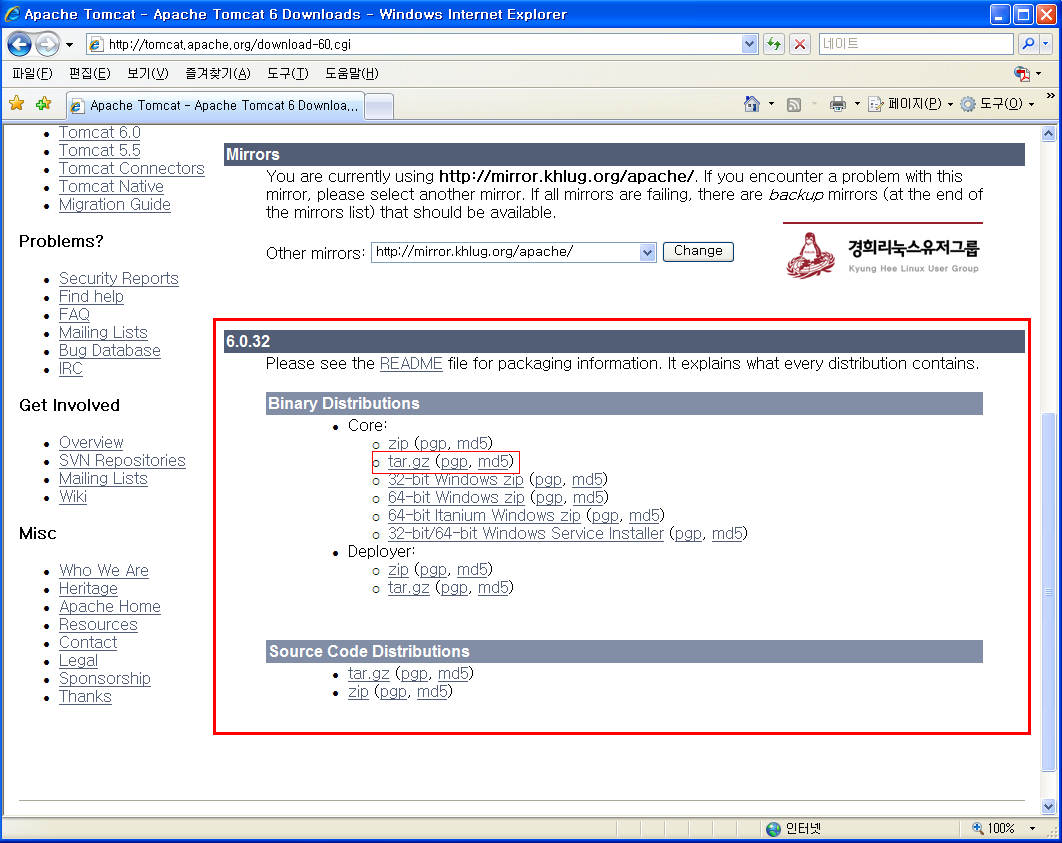
### Tomcat

1. **Tomcat 다운로드**

<http://tomcat.apache.org/> 방문한다.



Core에 있는 tar.gz 파일을 다운로드 한다.



1. 압축 푸는 명령어를 이용하여 다운로드 한 파일의 압축을 풀어준다.

$ tar xvfz apache-tomcat-6.0.32

1. /usr/local에 web 디렉토리 하나를 만들어 둔다.

$ sudo mkdir web

1. apache-tomcat-6.0.32를 /usr/local/web 으로 옮긴다.

$ sudo mv apache-tomcat-6.0.32 /usr/local/web/

1. 이동시킨 apache-tomcat-6.0.32의 이름을 간단하게 tomcat으로 바꾸어준다

$ mv apache-tomcat-6.0.32 tomcat

1. Tomcat이 Java 컴파일러와 라이브러리를 자유롭게 사용하기 위해서 시스템변수에 환경변수를 설정해 주어야 한다. etc/profile 이란 파일을 수정하도록 한다.

$ vi /etc/profile

* PATH 설정

apache-tomcat-6.0.32가 있는 설치되 있는 위치를 PATH로 넣어주어 PATH를 설정해 준다.

CATALINA\_HOME=/usr/local/web/tomcat

PATH=$JAVA\_HOME/bin:$CATALINA\_HOME/bin:$PATH

export JAVA\_HOME CATALINA\_HOME

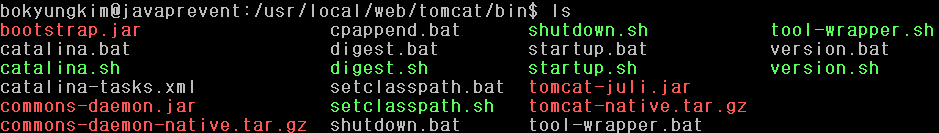
1. Vi 에디터를 이용하여 /etc/profile의 파일 첫 부분에 위의 내용을 추가해 주도록 한다. 추가한 후에는 저장하고 vi 에디터를 종료한다.
2. 다음의 명령어를 이용하여 설정을 완료시켜준다.

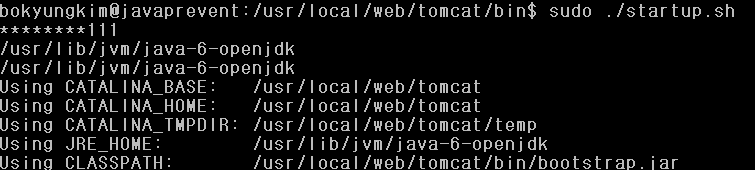
$ source /etc/profile

보통 기본으로 설정이 되어있지만 Tomcat 설정이 필요하다면 /usr/local/web/tomcat/conf 명령어를

이용하여 설정을 해준다.

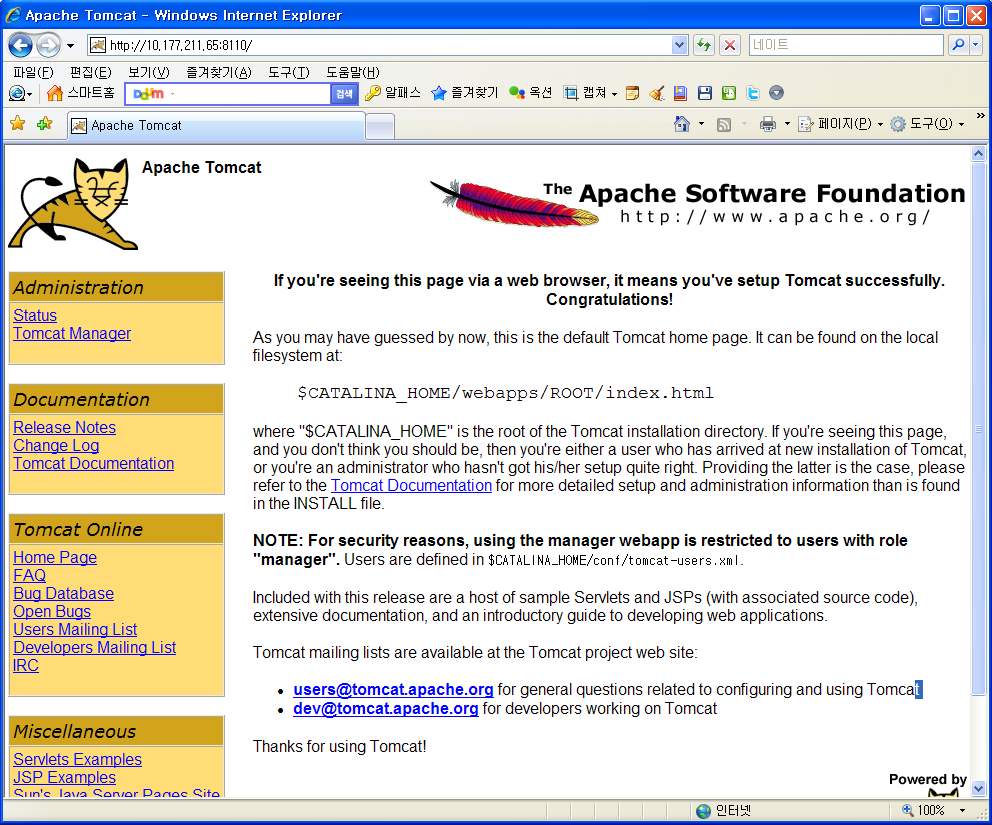
1. /usr/local/web/tomcat/bin에 가서 startup.sh 를 통해 Tomcat을 구동한다





1. Tomcat을 정상적으로 구동하면 <http://자신의서버> IP:8080(포트번호)으로 접속해본다.

아래와 같이 뜨면 성공한 것이다.



만약 실행이 되지 않는다면 포트를 변경해주어야 하는 상황일 수 도 있으니 포트번호를 확인한 후

사용하지 않는 포트 번호를 사용하도록 한다.

1. Server.xml 설정

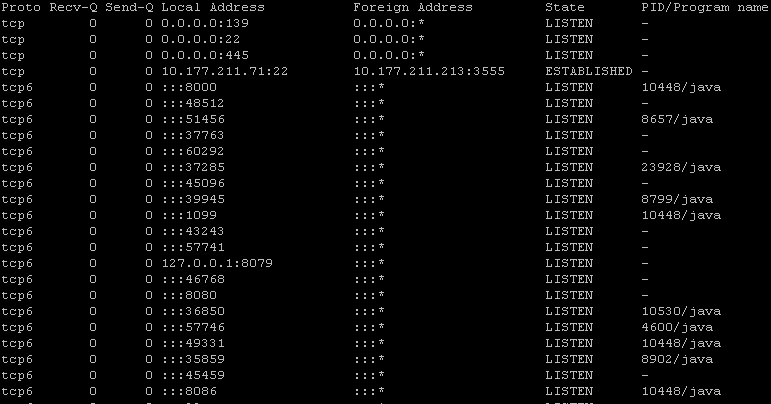
server.xml 파일은 포트번호를 변경하기 위해서 필요한 설정 파일이다.

netstat –a 명령어를 사용하여 열려있는 포트번호를 확인한 후 사용 할 수 있는 포트번호를 지정해 준다.

이 페이지의 포트번호가 8110 으로 되어있는데 자신의 서버에 열려있는 포트 번호로 변경하기 위해서 /usr/local/web/tomcat/conf 아래에 있는 server.xml 파일에서 포트번호를 변경시켜준다.

아래의 코드로 모든 포트를 확인할 수 있다.

$ sudo netstat -nap



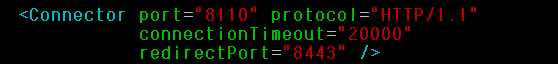
포트를 닫기 위해 sudo kill -9 (PID번호)를 아래의 코드와 같이 입력해 준다.

$ sudo kill -9 10488

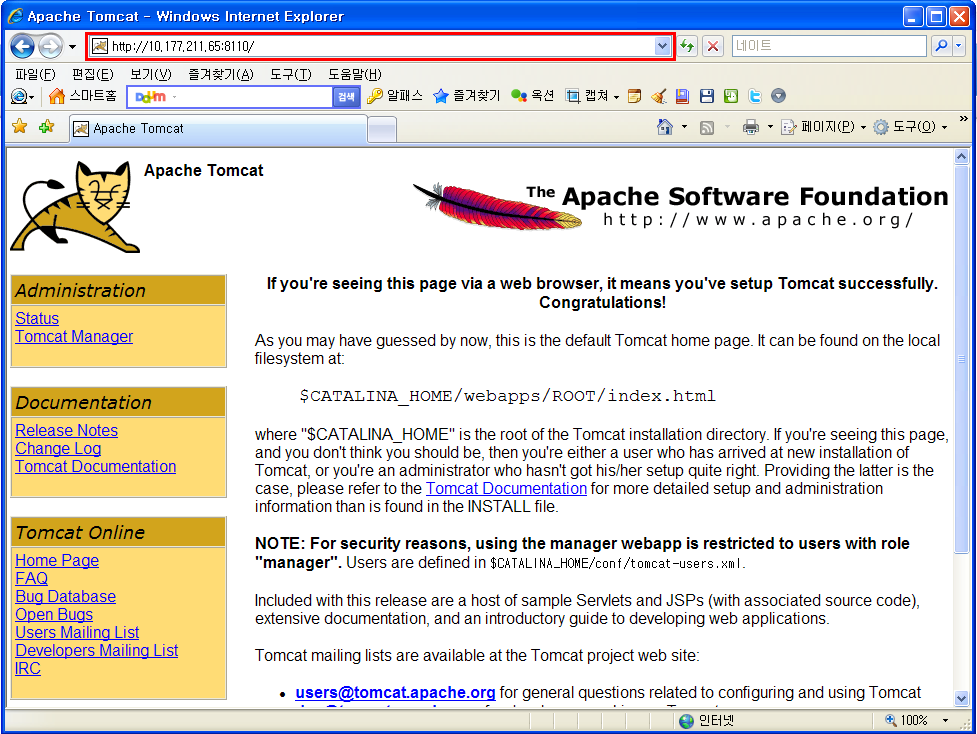
이와 같은 방법으로 포트가 죽었는지 확인한 뒤 다시 실행하면 포트오류를 해결할 수 있다. 만일 실행시킬

만한 포트번호를 찾았을 경우 Vi 에디터를 실행시켜서 server.xml의 포트번호를 변경해 준다.

$ sudo vi server.xml



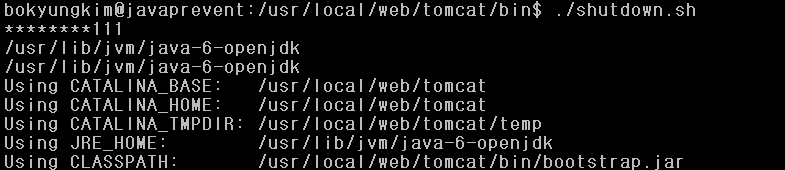
다음과 같이 8080으로 접속이 안되어서8110으로 변경해 주었더니 Tomcat이 실행되는 것을 확인 할 수 있다.



<http://10.177.211.65:8110> 으로 접속한 화면이다.

1. Tomcat을 종료하는 명령어는 ./shutdown.sh 이다. bin 폴더에 있는 shutdown.sh 명령을 실행시켜서

Tomcat 을 종료한다. 다음과 같이 화면에 뜬다면 Tomcat이 종료된 것이다.



#### User 별 단독 Tomcat 설치

Jenkins 는 하나의 tomcat에 서비스로 운영되고 있다. 단독 tomcat은 user가 직접 tomcat을 구성하여 단독으로 tomcat의 자원을 사용하는 것이다. 반면 공용 tomcat은 하나의 tomcat으로 버추얼호스트나 context path를 이용하여 쓰레드를 사용하여 여러유저가 하나의 tomcat자원을 사용한다. 그렇기 때문에 공용tomcat을 사용하면 jenkins의 프로젝트를 별도로 관리 할 수 없게 된다. 그렇기 때문에 사용자별 단독 tomcat을 사용하여 모든 jenkins서비스를 독립적으로 운영하도록 한다.

#### 단독 Tomcat 설치

1)사용자 추가

tomcat 서버를 구축할 사용자 mcp를 추가 한다.

asapadamin@asapadamin-desktop:~$ sudo adduser "mcp"

2)사용자 권한 변경

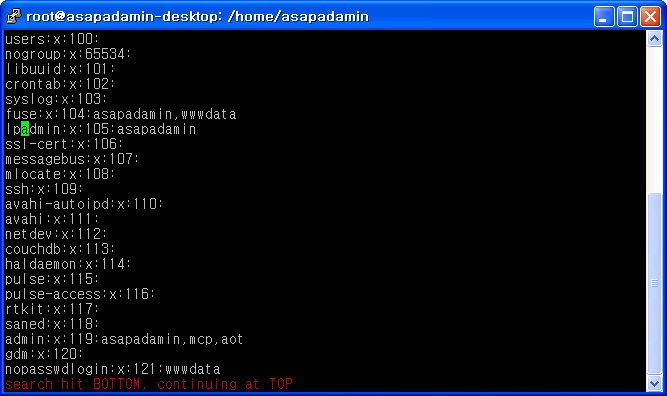
2.1) 생성된 mcp 유저의 권한을 쉽게 tomcat 서버를 구축하기 위하여 root 계정으로 변경한다.

asapadamin@asapadamin-desktop:~$ sudo su

2.2) /etc/group 파일의 admin 권한에 사용자를 추가 해주어야 하므로 사용자를 root 계정으로 전환한다.

root@asapadamin-desktop:/home/asapadamin# vi /etc/group

2.3) /etc/group 파일을 열어 “**admin:x:xxx:”** 을 찾아 사용자를 추가한다.

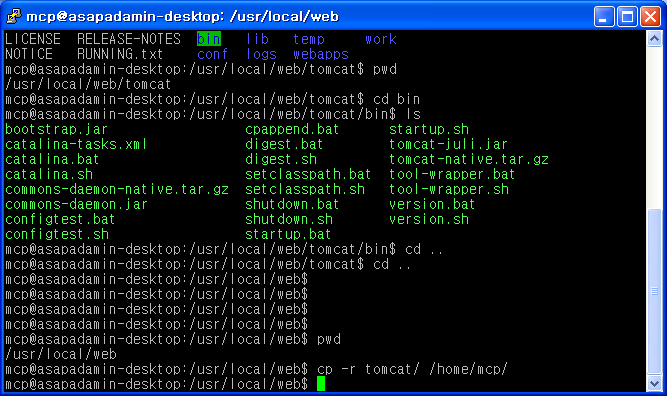


#### Tomcat 복사

1)사용자로 접속하여apache tomcat을 복사한다.

root@asapadamin-desktop:/home/asapadamin# su asapadamin

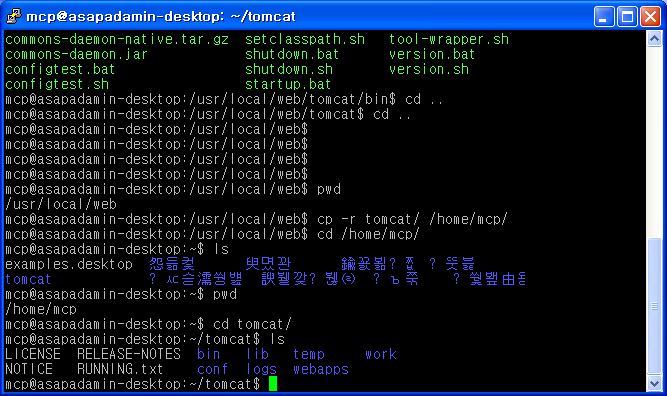
asapadamin@asapadamin-desktop:~$ cp –r tomcat/ home/mcp



2)사용자 계정으로 이동하여 복사한 tomcat 이 존재 하는지 확인한다.

mcp@asapadamin-desktop:/usr/local/web$ cd /home/mcp/tomcat/

mcp@asapadamin-desktop:~/tomcat$ ls



3) tomcat의 권한을 확인한다.

mcp@asapadamin-desktop:/usr/local/web$ ls -l

만일 tomcat의 권한이 root인 경우 user의 권한으로 바꿔준다.

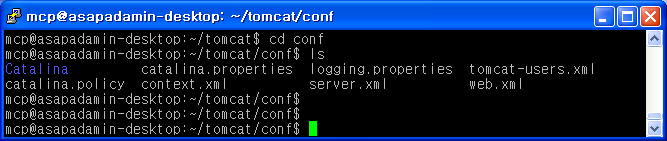
mcp@asapadamin-desktop:/usr/local/web$ sudo chown –R eun:eun /home/eun/tomcat

mcp@asapadamin-desktop:/usr/local/web$ ls -l

#### 환경 설정

환경 설정을 위하여 **/config/server.xml을 수정해야 한다.**

1) tomcat 이 복사된 위치로 이동하여, conf 폴더로 이동한다.



2) vi 편집기로 **server.xml 아래를 참고하여 포트 번호를 수정한다.**



**K-1.png 설정시 포트 번호를 잘못 설정하면 아무것도 보이지 않을 수 있으니 중복되지 않게 설정하여야 한다.**

|  |
| --- |
| **<?xml version='1.0' encoding='utf-8'?><Server port="9000" shutdown="SHUTDOWN"> <Listener className="org.apache.catalina.security.SecurityListener" /> <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" /> <Listener className="org.apache.catalina.core.JasperListener" /> <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" /> <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" /> <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" /> <GlobalNamingResources> <Resource name="UserDatabase" auth="Container" type="org.apache.catalina.UserDatabase" description="User database that can be updated and saved"**  **factory="org.apache.catalina.users.MemoryUserDatabaseFactory" pathname="conf/tomcat-users.xml" /> </GlobalNamingResources>**  **<Service name="Catalina">**  **<Connector port="9001" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="9002" /> <Connector port="9003" protocol="AJP/1.3" redirectPort="9002" />** |

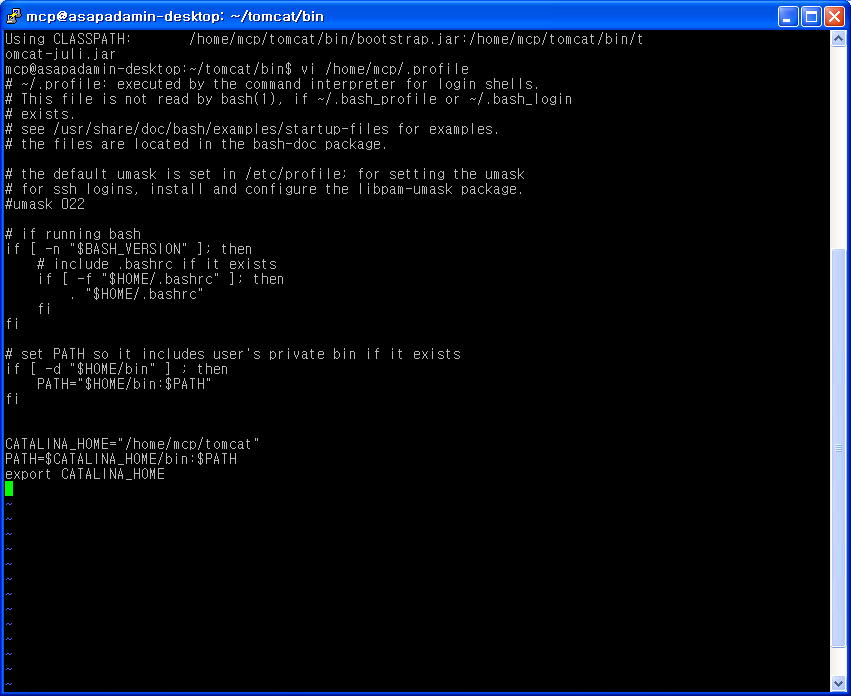
3) Path 설정하기 위해 /home/사용자 디렉토리/.profile 을 편집한다.

파일 내 아래와 같이 세팅해준다.

CATALINA\_HOME="/home/mcp/tomcat"

PATH=$CATALINA\_HOME/bin:$PATH

export CATALINA\_HOME



4) path를 반영한다..

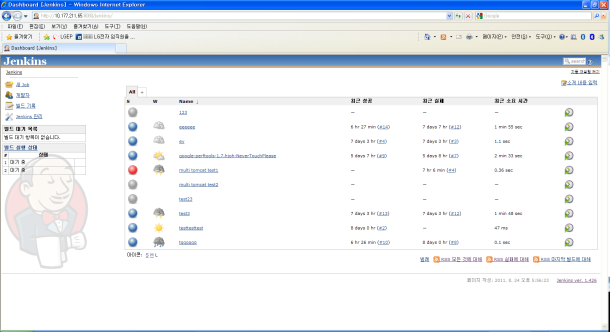
**mcp@admin$ source ./profile**

K-1.png path를 설정하고 반영하지 않으면 복사해온 tomcat 의 서버에 영향을 줄 수 있으니 반드시 tomcat을 복사하면 source ./profile 을 적용하여야 한다.

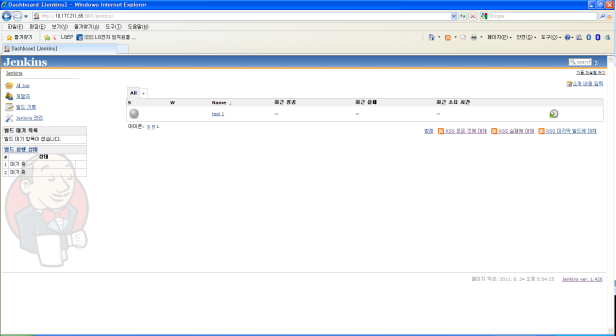
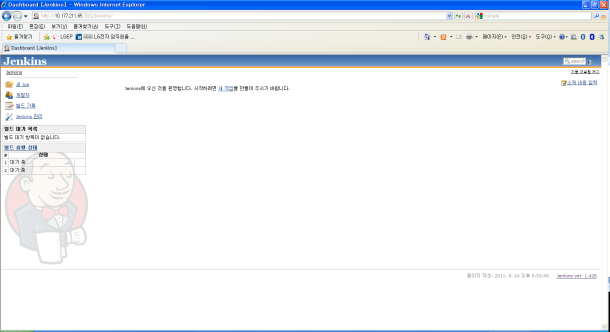
#### 설정 확인

1) localhost:포트번호/Jenkins 로 확인한다.

<asapadmin>

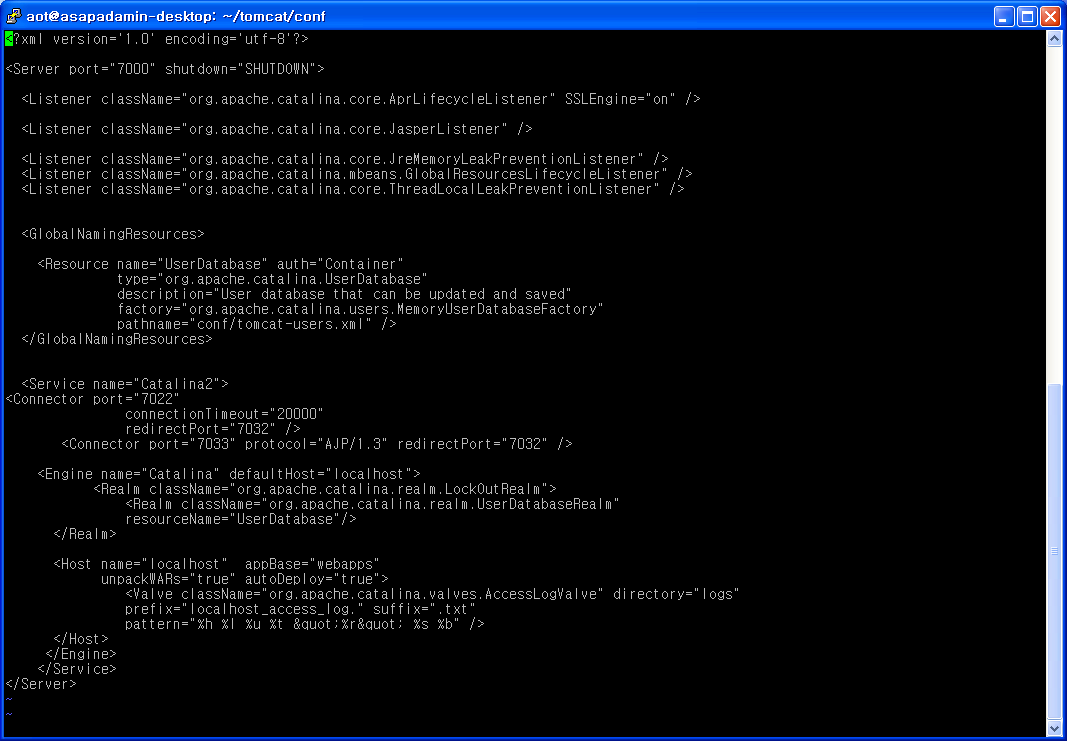


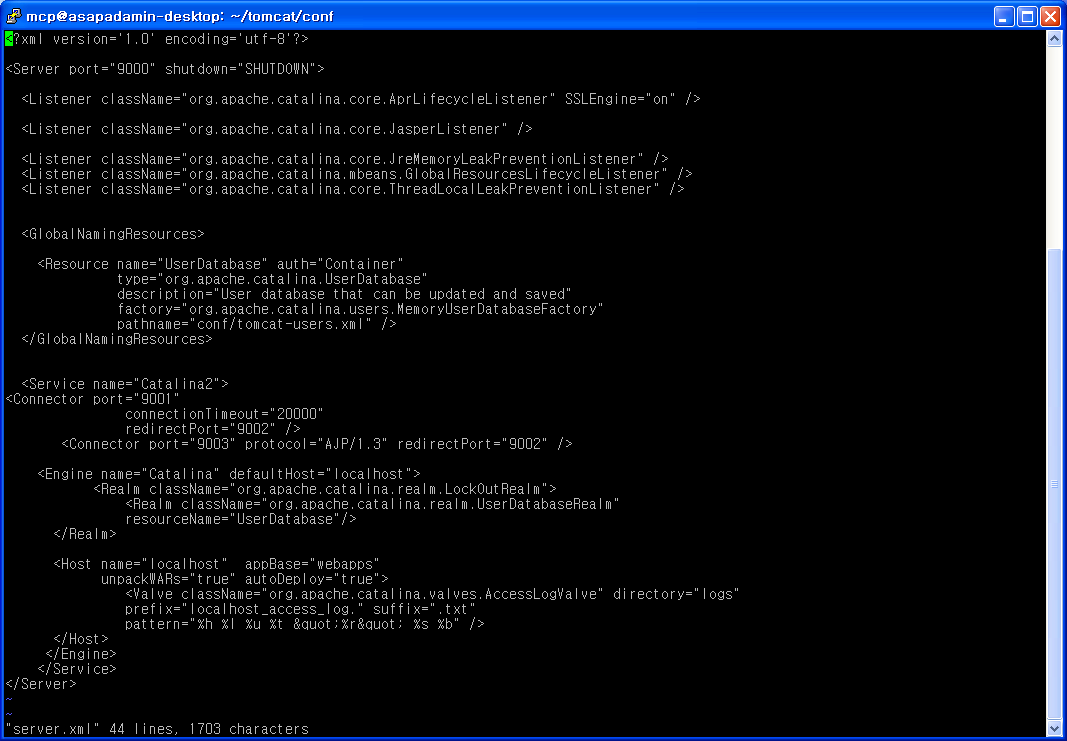
<AOT-USER> <MCP-USER>

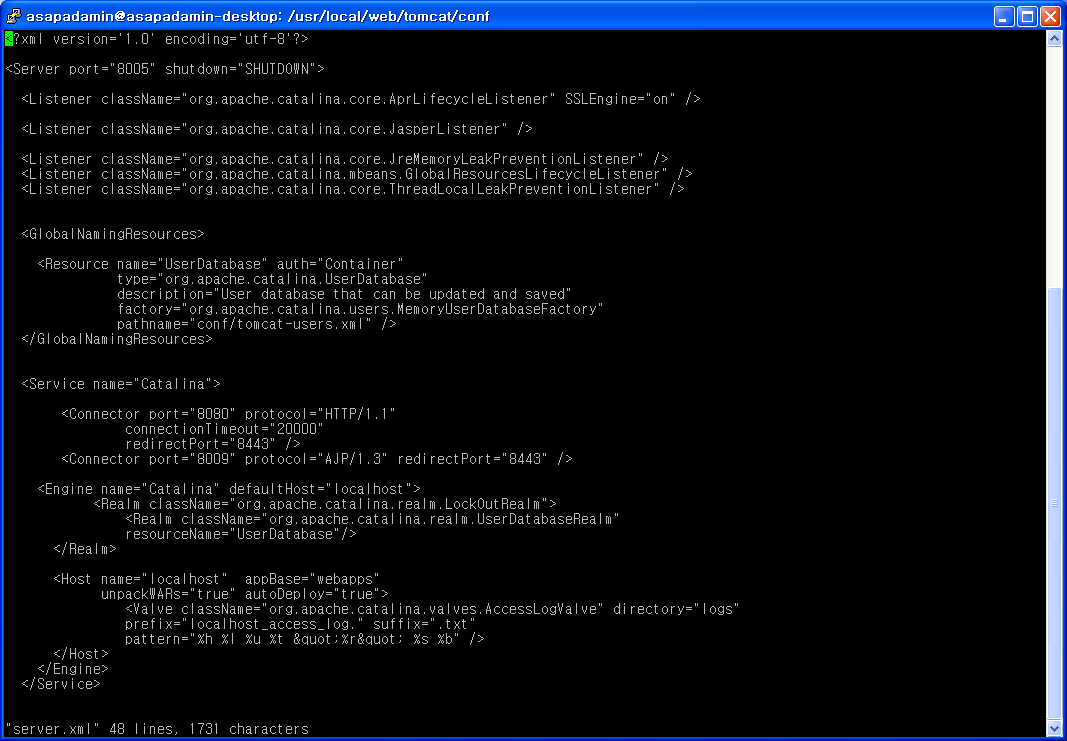


K-1.png 다수의 유저가 독립된 tomcat을 사용하기 위해서 1.3.5 절을 반복하여 수행한다.

K-1.png 유저별 -/conf/server.xml 설정 내용







K-1.png 다중 톰캣 서버를 사용하는 경우, /home/user/tomcat/webapps/으로 이동하여 Jenkins.war 파일을 다운받아 Jenkins를 설치해야한다.

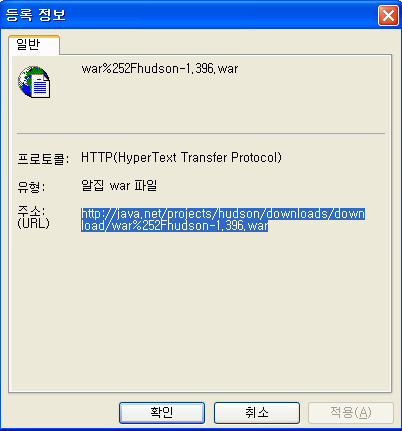
Jenkins가 설치되면 .jenkins 파일이 생성되는데 이 파일이 존재해야 독립적으로 User 별로 jenkins를 별도 관리 할 수있다.

### Jenkins

1. [http://jenkins-ci.org/](http://hudson-ci.org/) 에 접속하여 war 파일로 되어있는 jenkins.war를 최신버전으로 다운로드 한다.



여기서 Tomcat 설치와 마찬가지로 직접 리눅스 환경에 jenkins.war 파일을 다운로드 하는 방법으로 진행하겠다.



1. 만약에 GUI가 설치되어 있지 않은 linux 환경이라면 다른 pc에서 url을 위와 같이 확인한후 wget 명령어를 이용하여 jenkins.war 파일을 리눅스로 직접 다운로드 한다.

$ wget http://java.net/projects/hudson/downloads/download/war%252hudson-1.369.war



다운로드 완료된 화면이다.

1. ls 명령어를 이용하여 jenkins.war 파일이 다운로드 완료된 것을 확인할 수 있다.

$ l

Jenkins-1.369.war

1. 실제로 파일명은 jenkins-1.396.war 로 되어있는데 mv 명령어를 이용하여 간단하게 jenkins.war로 변경해준다.

$ sudo mv jenkins -1.369.war jenkins.war

1. 다운로드 이후에 jenkins.war 파일을 $TOMCAT\_HOME$/webapps/ 폴더로 이동시켜 준다.

$ sudo mv jenkins.war /usr/local/web/tomcat/webapps/

War 파일을 이동하게 되면 Tomcat 에서 jenkins.war 파일을 자동으로 인식하여 자동으로 프로젝트를 생성하게 된다.

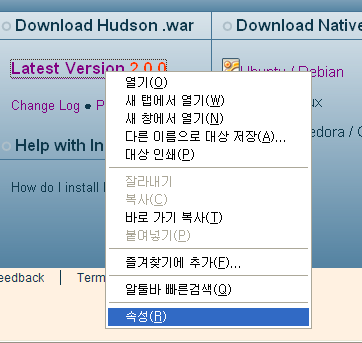
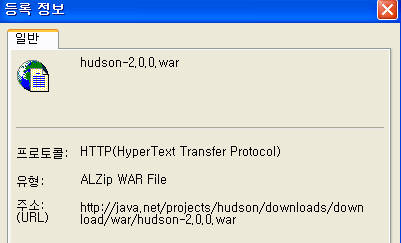
K-1.png 다중 톰캣 서버를 사용하는 경우, /home/user/tomcat/webapps/으로 이동하여 Jenkins.war 파일을 다운받아 Jenkins를 설치해야한다.

Jenkins가 설치되면 .jenkins 파일이 생성되는데 이 파일이 존재해야 독립적으로 User 별로 jenkins를 별도 관리 할 수있다. 따라서, 모든 user에 jenkins를 설치해야 한다.

[참고] Jenkins 버전 Upgrade 방법

사용자의 Jenkins의 버전이 현재 최신버전(2.0.0) 이하라면, 최신버전으로 upgrade 하도록 한다. 분산빌드 환경을 구축하기 위해 설치해야하는 plugin이 존재하기 때문이다. 그 뿐만아니라, 모든 plugin이 최신버전에 맞추어 항상 업데이트 되고 있으므로, 낮은 버전의Jenkins을 계속 사용하는 경우 설치할 수 없는 plugin이 늘어나기 때문에 최신버전을 유지하는 것이 바람직하다. Jenkins을 기본버전에서 최신버전으로 upgrade하는 방법은 아래와 같다.

1. Jenkins의 메인 홈페이지에 접속한다. ([http://jenkins-ci.org/](http://hudson-ci.org/))
2. 최신버전이 나온 링크(Latest Version 2.0.0)에 마우스를 올리고, 오른쪽 버튼을 눌러 속성을 선택하여 Jenkins.war파일을 받을 수 있는 경로를 복사한다.

1. Jenkins(CI서버)이 설치된 서버의 리눅스 환경에서 wget을 이용하여 최신 버전의 Jenkins.war파일을 얻는다.

$ sudo wget http://java.net/projects/hudson/downloads/download/war/hudson-2.0.0.war

1. 기존의 jenkins.war가 설치되어있는 경로를 찾아 들어간다. (대부분 ~/tomcat/webapps/ 이하에 존재할 것이다.) 그 경로에서 jenkins디렉토리와 jenkins.war파일을 지우거나 필요에 따라 다른위치로 백업해둔다. 단, 지우더라도 기존의 프로젝트 내용은 남아있다.

$ sudo rm –r ~/tomcat/webapps/hudson

$ sudo rm ~/tomcat/webapps/hudson.war

1. 과정(3)에서 받은 최신버전의 jenkins.war 파일을 기존jenkins.war파일이 존재하던 경로에 가져오도록한다.

$ sudo mv hudson.war ~/tomcat/webapps/

1. 마지막으로 ~/tomcat/bin/ 에서 tomcat을 껐다가 다시 켜주도록 한다. 1분후에 web에서 확인해보도록 한다.

$ sudo ./shutdown.sh

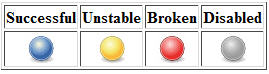
$ sudo ./startup.sh

K-1.png

1. 위의 모든과정을 수행했음에도 불구하고 web에서 실행되지 않는다면, 사용하던 포트(예:8080)가 열려있는지 확인하고 다른포트로 변경해줘보도록 한다. 포트가 열려있는지 확인은 sudo netstat –nap 등으로 하고, 포트를 바꾸어 줄 때는, /etc/ 아래에 있는 service 파일을 vi에디터로 열어 사용가능한 포트를 확인한 후 바꾸어준다. 사용하는 포트가 확인되면 ~/tomcat/conf/ 아래의 server.xml를 vi에티터로 열어 기존의 사용하던 포트를 지우고 새로운 포트번호를 대체하도록 한다. 포트를 변경한 후에는 과정(6)을 다시 진행해준다.
2. Jenskins이 열렸지만, 내부 UI에서 자바관련 error가 발생한다면 $sudo update-alternatives --config java 명령어를 이용하여 기존에 쓰던 것과 다른 자바버전을 선택해서 사용해본다. 다른 자바를 선택했다면 그에 맞는 이름으로 /etc/profile에 있는 JAVA\_HOME설정도 바꿔줘야한다.

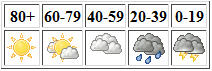
[참고]

* 빌드 업무 상태 (Job States)



* 성공 (Successful) : 빌드가 완료되었고 안정적으로 간주됨
* 불안정 (Unstable) : 빌드가 완료되었고 불안정한 것으로 간주됨
* 실패 (Failed) : 빌드가 실패함
* 사용불가 (Disabled) : 빌드 업무가 사용 불가임
* 빌드 업무 안정도 (Job stability)

빌드 업무가 완료되고 이슈없이 대상 아티팩트를 생성할지라도, 허드슨은 플러그인 형태로 구현된, 그러니까 암묵적으로 안정도를 평가하기 위해 만들어 놓은 후속 작업들에 기반하여 해당 빌드에 ( 0~100 사이의) 안정도 점수를 부여할 것이다. 이 암묵적인 작업들이란 단위 테스트들 (JUnit), 커버리지(Cobertura), 정적 코드분석(Findbugs)을 포함한다. 해당 점수가 높으면 높을수록 빌드는 더 안정적이다. 아래의 그림은 안정도 점수 범위에 따른 심볼들을 요약해서 보여준다.



### Ant

#### 소개

Ant(Another Neat Tool)는 make, Batch File, Shell 스크립트등의 기능을 통합한 것이다.

* Make : C계열 언어를 컴파일시 복잡한 컴파일 단계를 스크립트 파일로 묶어 컴파일을 쉽게 하도록 지원하는 툴
* Batch File: DOS, Window 계열 운영체제에서 사용하는 자동화 스크립트
* Shell Script: Linux 계열 운영체제에서 사용하는 자동화 스크립트

XML 기반이라 기초적인 XML 문법만 알면 배우기가 쉽고 어느 운영체제에서도 사용이 가능한 크로스 플랫폼을 지원한다. Ant 내에는 JVM이 별도로 운영이 되어 java 컴파일, jar 패키징 작업을 자체적으로 수행한다. EJB Compile과 패키징 작업 그리고 J2EE 개발을 지원하는 기능(XML 스크립트 자동생성)이 내장되어 있다. 또한 java 프로젝트들을 편리하게 배치할 수 있다. FTP, Telnet, Application Server, SQL 명령어들을 배치 자동화에 사용할 수 있다.

• Ant의 디렉터리 구조

+- README, LICENSE, fetch.xml, other text files

// README, 라이선스 정책, 부가적인 라이브러리 패치를 위한 빌드 스크립트 (fetch.xml)등의 기본 설정 파일

+-bin //Ant 실행 스크립트 모음(배치, 쉘, 파이썬 스크립트 등)

+-lib //Ant 수행을 위한 부가 라이브러리 모음

+-docs //가이드 문서 모음(웹 문서 형식으로 제공)

+-etc // 빌드하고 나서 결과를 리포팅하기 위한 스타일시트 모음(Jprobe, JUnit, JDepend 등에 대한 기본 스타일시트 제공), 사용자가 커스터마이징해서 사용 가능

#### 설치

1. 특정 version을 수동으로 설치하려면 아래의 사이트에서 apache-ant-1.7.1-bin.tar.gz를 다운로드 한다. **(다운로드 위치)**  <http://ant.apache.org/bindownload.cgi>
2. 다음의 명령어로 apache-ant-1.7.1-bin.tar.gz 파일을 압축 해제한다.

$ sudo tar xvfz apache-ant-1.7.1-bin.tar.gz

1. 마지막으로 아래의 명령어를 실행시켜준다. **(실행위치)**

$ sudo ./build.sh install

1. Apt-get을 사용해서도 ant 설치가 가능하다.

$ sudo apt-get install ant

#### 설정

build.xml은 하나의 project 요소를 가진다. project 요소는 target 요소를 포함하고, 각 target은 여러 task를 포함한다. Target 요소에는 task 요소로 세부 작업을 정의한다. 그 밖에 property 태그로 빌드와 관련된 기타 다양한 설정을 정의할 수 있다.

<project name="MyProject" default="dist" basedir=".">

<description>

simple example build file

</description>

<!-- set global properties for this build -->

<property name="src" location="src"/>

<property name="build" location="build"/>

<property name="dist" location="dist"/>

<target name="init">

<!-- Create the time stamp -->

<tstamp/>

<!-- Create the build directory structure used by compile -->

<mkdir dir="${build}"/>

</target>

<target name="compile" depends="init" description="compile the source " >

<!-- Compile the java code from ${src} into ${build} -->

<javac srcdir="${src}" destdir="${build}"/>

</target>

<target name="dist" depends="compile" description="generate the

distribution" >

<!-- Create the distribution directory -->

<mkdir dir="${dist}/lib"/>

<!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->

<jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>

</target>

<target name="clean" description="clean up" >

<!-- Delete the ${build} and ${dist} directory trees -->

<delete dir="${build}"/>

<delete dir="${dist}"/>

</target>

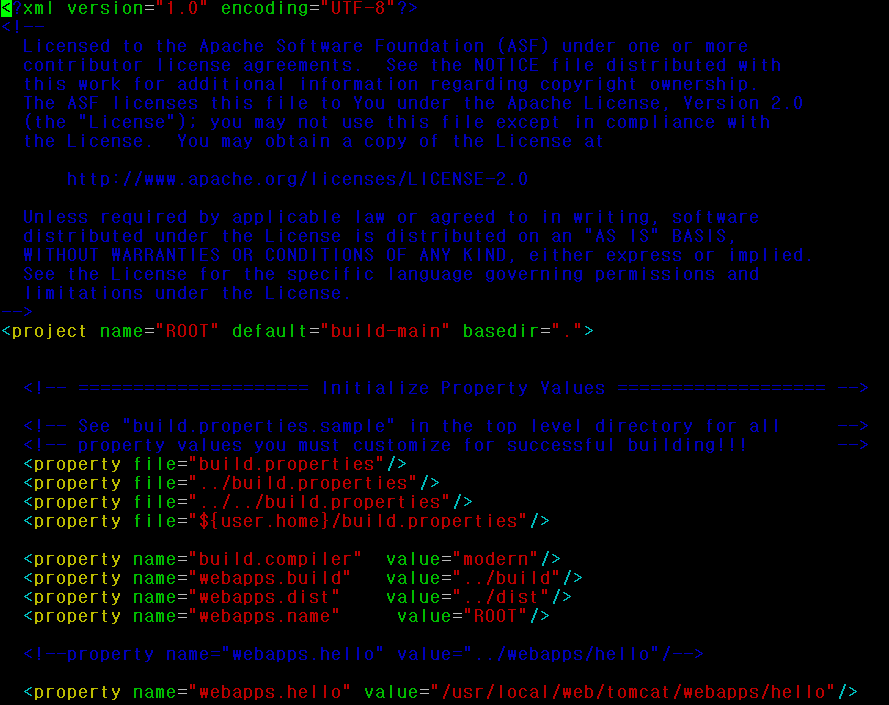
</project>

위 예제에서는 MyProject라는 자바 프로젝트의 빌드 자동화 스크립트를 정의 한다. project 태그의 default 속성은 Ant로 빌드를 수행할 때 target을 지정하지 않으면 기본으로 수행할 target을 의미한다. 먼저 property 태그로 빌드 스크립트에서 사용할 다양한 저장소 위치에 대한 심벌릭 명칭을 정의한다. 그리고 init, dist, clean 이라는 target을 정의한다. Target의 이름만으로도 어떤 작업을 의미하는지 짐작할 수 있을 것이다. Init은 빌드 결과를 저장할 build 디렉터리를 생성한다. Target 태그의 depends 속성은 해당 target을 수행하기 전에 수행하는 target의 이름을 의미한다. 위 예제에서는 compile을 수행하기 전에 init을 수행하고 dist를 수행하기 전에 compile을 수행한다.

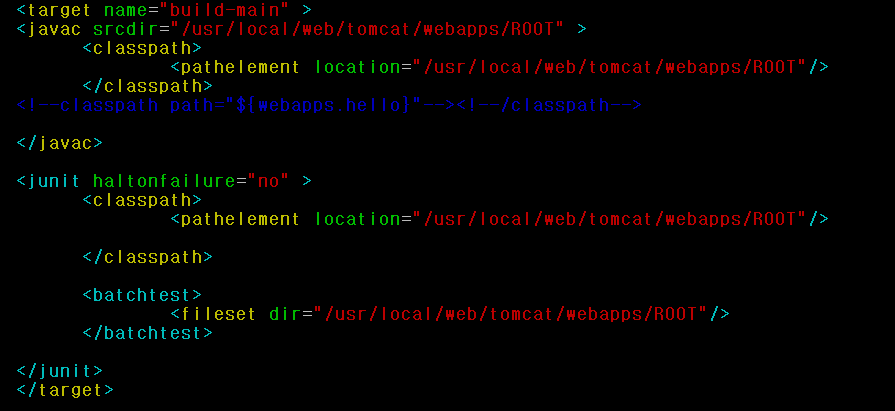
Compile은 javac라는 task를 정의해서 빌드를 수행한다. Ant는 빌드와 관련된 다양한 task를 제공한다. Compile을 성공적으로 수행하고 나면 dist가 lib경로를 생성하고 그 안에 jar task로 배포 파일을 생성한다. Clean은 다음 빌드 작업을 위해 앞에서 생성한 디렉터리를 제거한다.

리눅스에서 ant –version 명령어를 사용하여 설치된 Ant의 버전을 알아낸 뒤 입력한다. Targets을 지정해주기 위해 build.xml 파일을 열어본다.

/usr/local/web/tomcat/webapps/ROOT 에 디폴트로 지정된 build.xml 파일을 열어보면 다음과 같이 나온다.

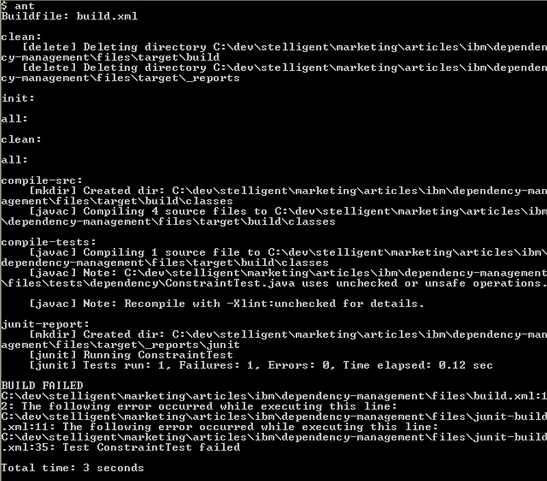


Project name을 ROOT로 지정해 주고 default로 build-main을 지정해준다. 실제로 빌드가 이루어지는 건 이 부분을 통해서 이기 때문에 이 부분의 설정을 변경해주었다.



#### 실행 결과

build.xml 설정이 끝난뒤 ant를 실행시키면 아래와 같은 결과가 나온다.

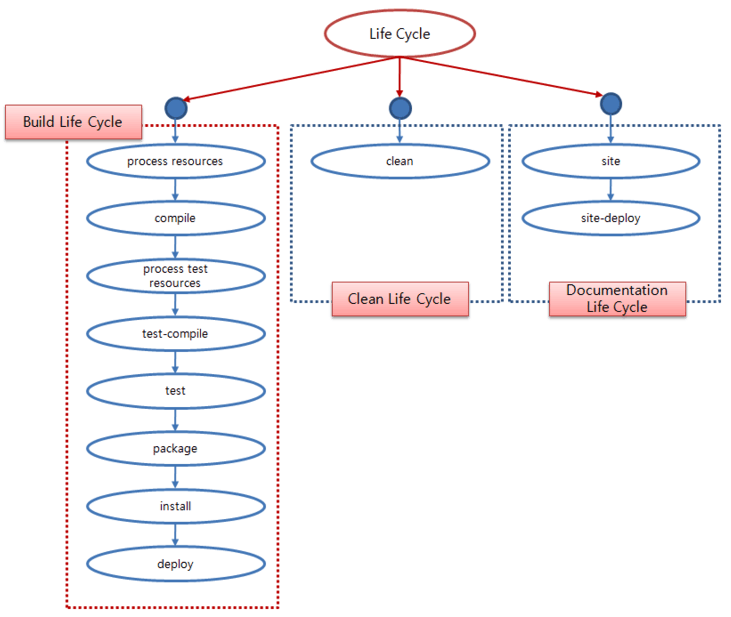


### Maven

소개

크게 개발자들이 프로젝트를 진행하면서 주로 하는 일은 프로젝트의 산출물인 어플리케이션을 Build하고, 전체를 한번 싹 지워주는 Clean 그리고 해당 어플리케이션에 대한 산출물을 문서화해서   
이를 배포해주는 Documentation 과정으로 나눠볼 수 있을 것 같다.

이 세가지 Life Cycle에 대해서 Maven이 정의하는 개별 단계는 아래와 같다.



각 라이브 사이클의 특정 단계를 지정하면 처음부터 해당 단계까지의 Life Cycle을 구성하는 Phase를 타고 실행이 된다. 예를 들어 test phase단계까지를 지정하면 "process-resources - compile - process-test-resources - test-compile - test"까지의 과정을 수행한다.  따라서 테스트를 수행하기 위해서 이전에 개별적으로 수행해줘야 했던 부분들을 Life Cycle과 지정 phase에 의해 수행하기 때문에 개발자는 고유 업무인 개발과 테스트에 집중하도록 Maven이 지원한다**[출처]** [Maven 사용 관련 정리](http://blog.naver.com/chidoo74/10098880659)|**작성자** [민혁아빠](http://blog.naver.com/chidoo74).

설치

• Maven 2 (option 1)

$ sudo apt-get update // 패키지리스트를 최신 버전으로 업데이트

$ apt-cache search maven // apt-get을 통해 maven을 설치할 수 있는지 검색

$ sudo apt-get install maven2 // maven 2 설치

$ mvn -version // 설치된 maven 버전 확인

[참고]

$ sudo apt-get autoremove maven2 // 설치된 패키지 제거

$ sudo apt-get -f purge maven2 // 오류가 나서 설치가 안된 프로그램 제거

• Maven3 (Option 2)

: Maven3는 apt-get을 통해 설치불가하며, wget을 통해 홈페이지에서 binary를 직접 다운받아야 함

$ wget http://apache.tt.co.kr//maven/binaries/apache-maven-3.0.3-bin.tar.gz

$ tar xvfz apache-maven-3.0.3-bin.tar.gz // 압축 풀기

설정

• 환경 변수 설정

: sudo vi /etc/environment 명령을 통해 아래와 같이 maven 홈디렉토리를 환경변수에 추가

PATH="/home/hyejinoh/apache-maven-3.0.2/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"

• Maven 설정 : Local Repository 설정

$ sudo vi /etc/maven2/conf/setting.xml

#! /bin/sh

<localRepository>/home/hyejinoh/maven-local-repo</localRepository>

• Maven 설정 : Mirror 설정

$ sudo vi /home/hyejinoh/apache-maven-3.0.2/conf/setting.xml

#! /bin/sh

<mirrors>

<mirror>

<id>Internal</id>

<mirrorOf>\*</mirrorOf>

<name>Hyejin's Internal Repository </name>

<url>http://10.177.211.66:8080/archiva/repository/internal/</url>

</mirror>

</mirrors>

• Maven 프로젝트 생성

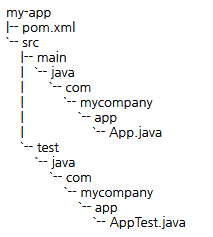
$ mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

[참고] 파라메터 설명

- agroupId : Java에서 생성시킬 Main package를 지정

- artifactId : artifact라는 단어가 의미하는 것처럼 산출물 혹은 성과물의 이름을 지정

- Archetype : ArtifactId 프로젝트의 성격을 나타내며 Architecture Type을 지정



• 프로젝트 POM 파일 설정

$ cd ~/my-app

$ vi pom.xml

----------------------------------------------------------------------------------------------------------------------

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.mycompany.app</groupId>

<artifactId>my-app</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>my-app</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

[참고] POM.xml 파일의 구조에 대해서는 Appendix의 Maven: project descriptor 부분을 참조하라.

실행 결과

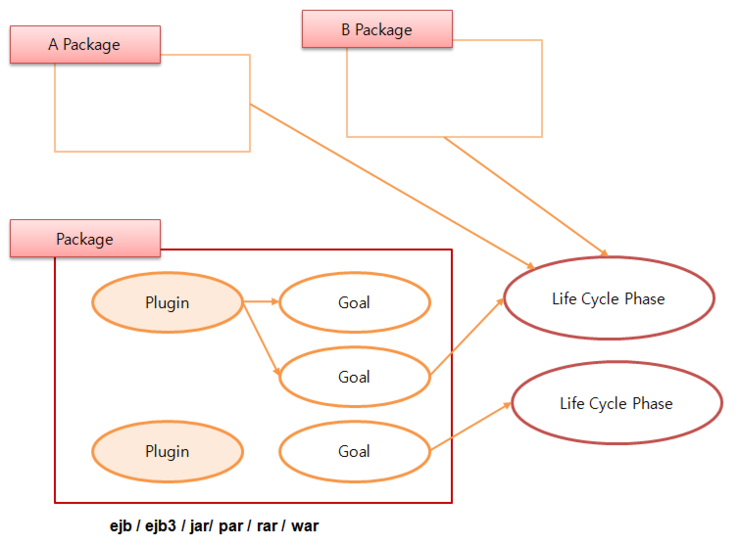
• maven 실행 방법

: Maven의 핵심 실행 단계는 plugin과 goal에 의해서 이루어진다.

하나의 plugin은 여러 goal을 정의할 수 있으며, 개별 goal은 특정 Life Cycle phase와 연관되어 있다.

$ mvn [plugin-name]:[goal-name]

[참고] Maven Plugin에서 제공하는 자세한 Plugin과 Goal 종류는 Appendix에서 참고하라.



**[그림] plugin, goal, life cycle phase의 관계**

• maven 컴파일

$ cd ~/my-app

$ mvn compiler:compile

--------------------------------------------------------------------------------------------------------------------

[INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'compiler'.

[INFO] ------------------------------------------------------------------------

[INFO] Building MyfirstMaven

[INFO] task-segment: [compiler:compile]

[INFO] ------------------------------------------------------------------------

[INFO] [compiler:compile]

[INFO] Nothing to compile - all classes are up to date

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: < 1 second

[INFO] Finished at: Fri Jul 15 10:14:10 KST 2011

[INFO] Final Memory: 5M/104M

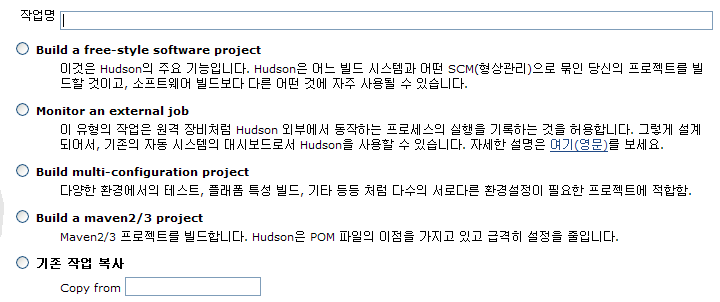
[INFO] ------------------------------------------------------------------------

## 설정

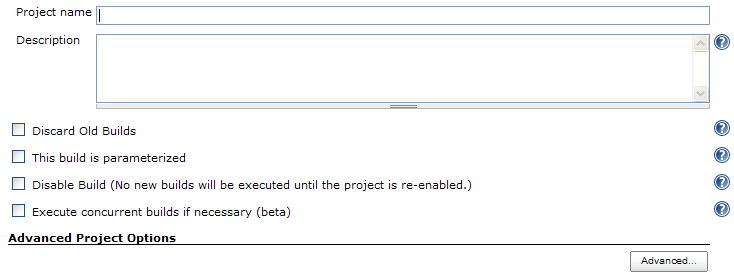
### 프로젝트 공통 설정

• Jenkins에서 빌드 업무 설정하기

1. 프로젝트명 입력 후 build a free-style software project 선택 후 ok 버튼 클릭

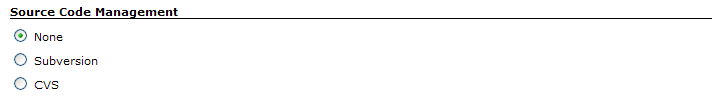


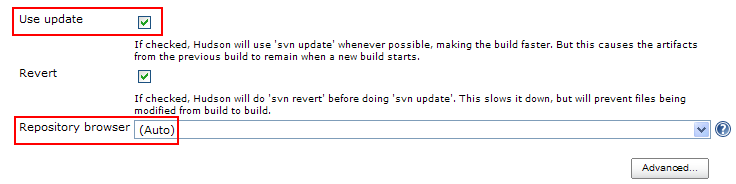
새로운 빌드 업무에 대한 다량의 설정 항목이 이어진다.



* Project name
* Description : 빌드 업무에 대해 기술할 수 있는 자유 형식 폼 필드
* Discard old builds : 이 체크박스를 선택하지 않는다면, 이전 기록을 보관할 것이다.
* This build is parameterized : 만일 이 옵션을 선택하면, 허드슨은 빌드 프로세스에 전달 가능한, 이름-값 쌍으로 이루어진 임의의 파라미터 세트 제공을 허락 할 것이다. 설정 파라미터들은 러닝 빌드 환경에서 환경 변수로 지정될 것이다.
* Disable build : 이 항목이 체크되면 옵션이 비 활성화 될 때까지 이 빌드 업무는 실행되지 않을 것이다.
* Advanced Project Options
  + Quiet Period : 빌드 수행이 예정되었을 때 실제 수행 이전에 발생하는 정숙기간(=휴지기간)을 설정할 수 있다. (즉, Delay time과 동일의미, 0이면 build now의 의미)
  + Use custom workspace : 기본적으로, 허드슨은 ${jboss-home}./jenkins/jobs.[project name]에 빌드 업무용 워크스페이스를 만든다. 이 옵션은 다른 장소를 지정하는 것을 가능케 해준다.

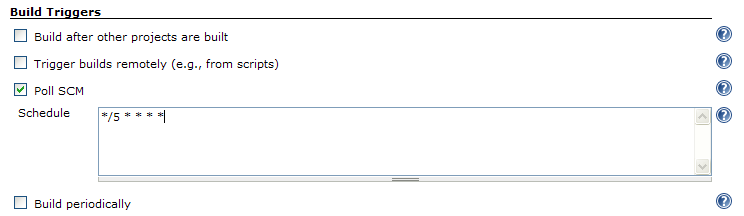
1. Source 관리 시스템과 연동





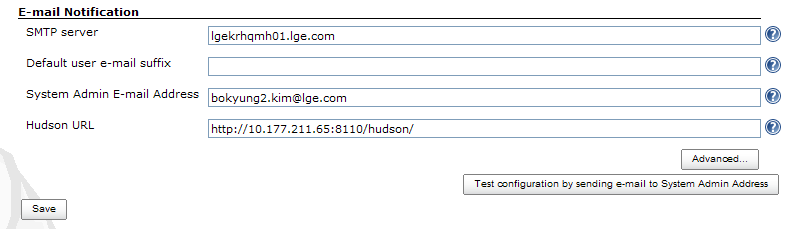
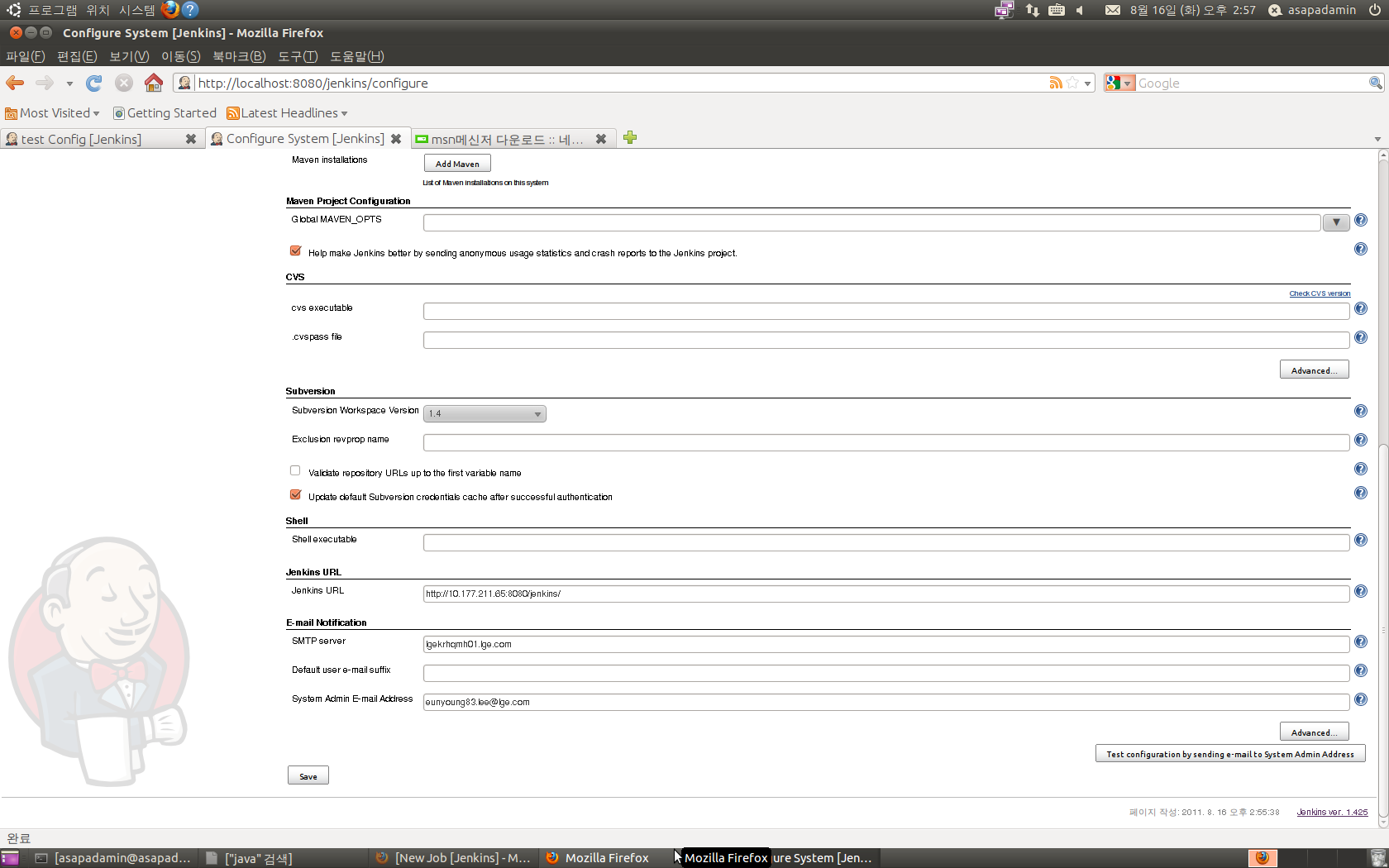
* Use Update : Source 관리 시스템에서는 Source를 내려 받을 때 default가 모든 Source를 매번 Download 받기 때문에 Use Update Option을 사용하면 svn update와 같은 action을 취한다.
* Repository Browser : Source 관리 시스템에 저장된 Source의 내용을 웹에서 browsing할 수 있는 도구이다. 도구에 따라서 이전 버전과 변경된 부분에 대한 비교 또는 처음 Source Code가 생성되었을 떄부터 매번 commit 되었을 때 변경 내용에 대한 Revision등에 대한 모든 history를 출력해준다.

1. Build Trigger (어느 시기에 build가 돌아갈지 정할 수 있는 option)



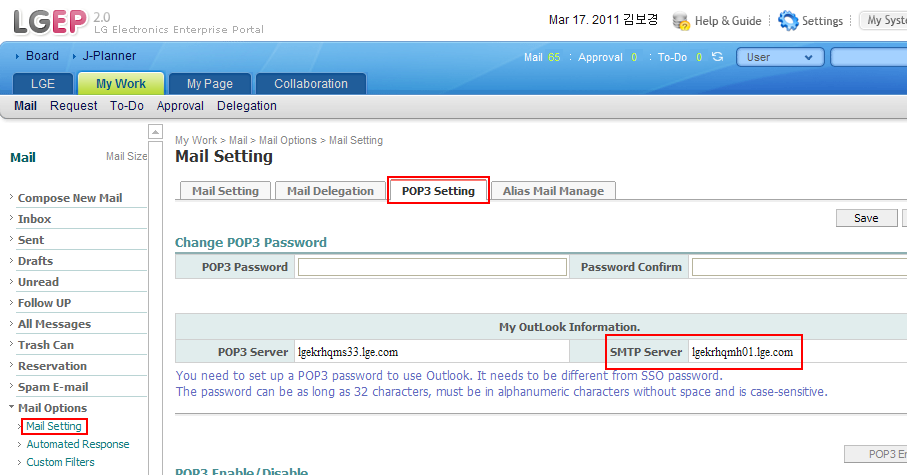
* Build after other projects are built : 이 option에는 다른 Job(Project)의 이름을 인자로 넣는다. 이렇게 하면 지정된 프로젝트의 build가 정상적으로 끝나면 자동으로 이 프로젝트가 Invoke된다.
* Poll SCM : 이 option을 사용하면 여기에 지정한 주기별로Source관리 시스템을 polling(check)하여 변경이 있을 경우에만 build를 수행한다. 시간 설정 방법은 unix의 crontab 명령과 같은 형식을 사용한다. (분 시간 날짜 월 요일) 예를 들어, 표현식이 0,15,30,45 \* \* \* \* 이라면 허드슨은 매 15분 마다 변경을 위해 저장소를 체크할 것이다.
* Build periodically : 정해진 시간 주기 별로 Source가 변경과 상관없이 무조건 주기적으로 build를 수행하며 Poll SCM과 마찬가지로 crontab과 같은 형식으로 schedule을 등록한다.

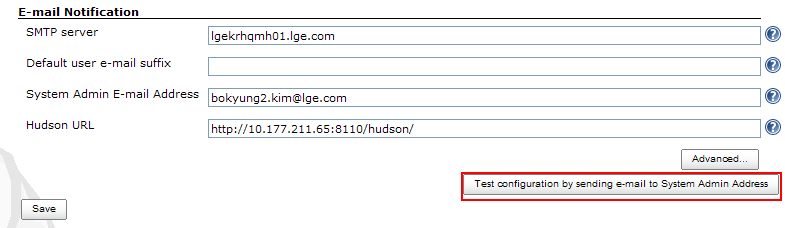
여기서 중요한 마지막 항목은, 허드슨이 빌드 실패 했을 경우와 같은 중요한 이벤트들을 이메일로 알려줄 수 있게 하는 SMTP설정이다. 만일 SMTP서버가 인증을 필요로 한다면, 고급 옵션(advanced)를 사용할 필요가 있다.

\*\*Jenkins 관리> 

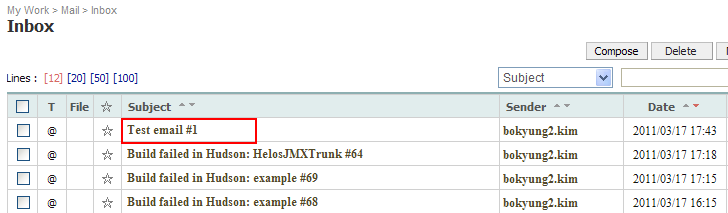
SMTP server는 LGEP 홈페이지에 가서 확인할 수 있다.

<http://sso.lge.com> 으로 접속하여 Mail Options에 있는 Mail Setting메뉴에서 SMTP Server 주소를 알아내도록 한다.





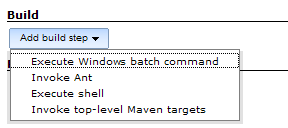
필요한 정보를 얻었다면 Test 이메일을 보내는 버튼을 누른다.



Test email이 온 것을 확인 해 볼 수 있다.

### 프로젝트 빌드 설정

Jenkins에서는 Unix/Windows shell, Ant, Maven 을 기반으로 정해진 schedule 정책에 따라 지정된 build 스크립트가 구동됨으로써 build Process가 수행된다. 선택할 수 있는 build Category 는 다음과 같으며, 이는 Jenkins > 프로젝트명 > configure 에서 아래와 같이 선택 가능하다.



[전제조건]

* 접근 가능한 소스 코드 저장소 (repository)가 있어야 한다.
* 저장소는 빌드를 필요로 하는 소스 코드를 담고 있어야 한다.
* 저장소는 소스를 빌드하는 빌드 스크립트를 반드시 포함하고 있어야 한다. 허드슨이 간단한 쉘을

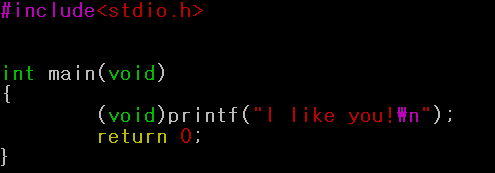
지원하지만, 여기서 말하는 스크립트는 보통 Ant나 Maven 스크립트, Ant나 MSBuild 같은 빌드 스크립트를 말한다.

일반 shell 빌드

1. 우분투에 gcc 설치하기

sudo apt-get install build-essential

1. 간단하게 C 소스를 만든다.



1. gcc 컴파일러를 이용하여 리눅스에서 예제 소스를 빌드할 차례다.

$ gcc test.c

$ ./a.out

I like you !

gcc test.c의 명령어를 실행하여 실행파일을 만든후 ./a.out을 통해 결과를 확인해보았다.

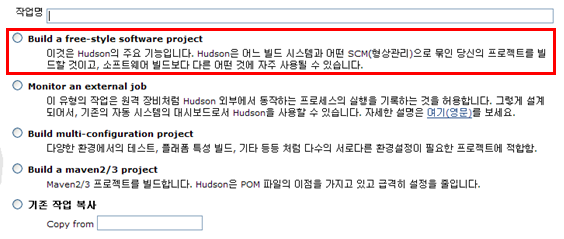
**• 빌드 스크립트: test.sh**

vi test.sh

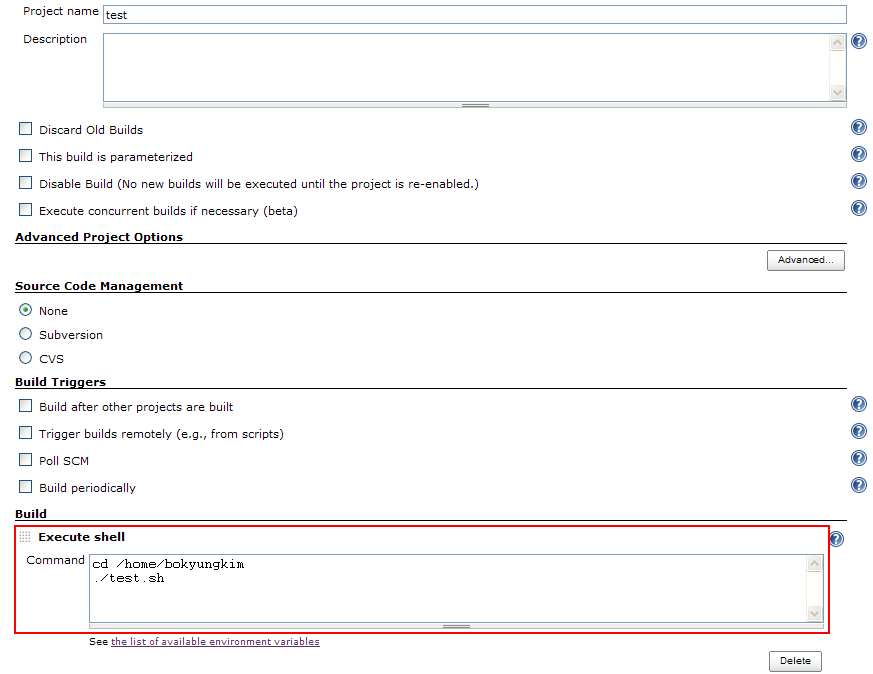
-------------------------------------------------------------

./a.out

**• Jenkins 프로젝트 설정**

****

Build a free-style software project를 선택한다.



Jenkins에서도 빌드결과를 확인 해 보기 위해 Shell Script를 작성해준 뒤 빌드해보면 다음과 같이 빌드가 성공된 것을 확인할 수 있다.

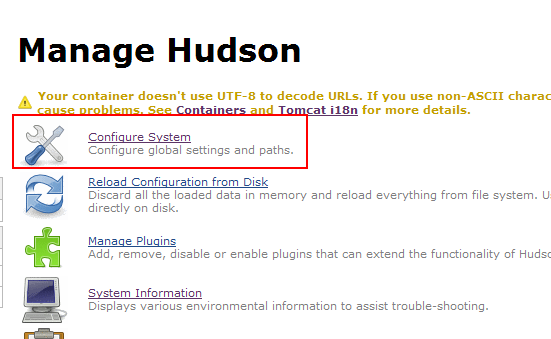
But!! Shell Script에서는 ./ 명령어가 동작하지 않기 때문에 command 창에

cd /home/asapadmin

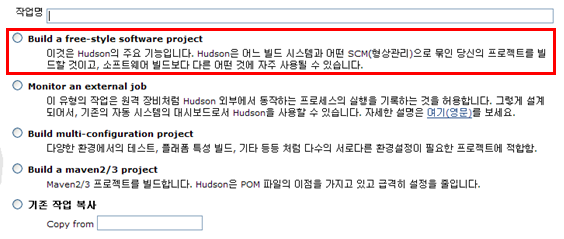
sh test.sh

#### Ant 빌드

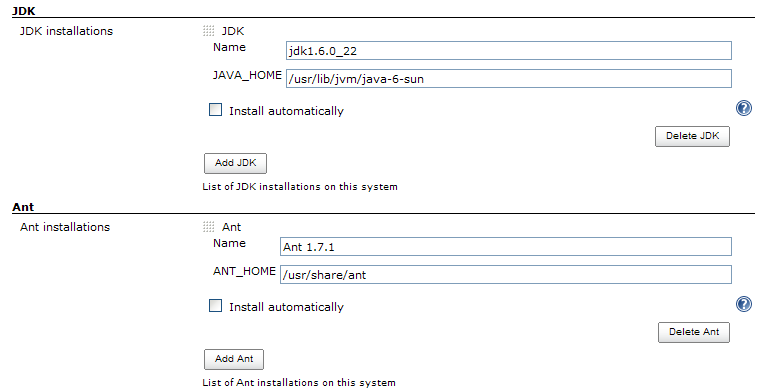
1. [http://localhost:8110/Jenkins](http://localhost:8110/Hudson)의 Jenkins 메인 페이지에서, Jenkins 관리를 클릭하여라. 이어지는 페이지에서 Configure System을 클릭한다.



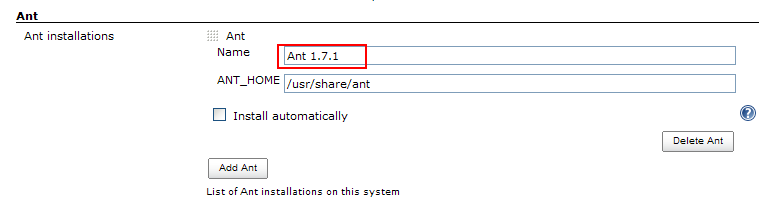
시스템 설정 페이지에서, 첫 째 항목인 Home directory 에서 Jenkins의 모든 설정이 저장되고 모든 작업이 이루어진다.



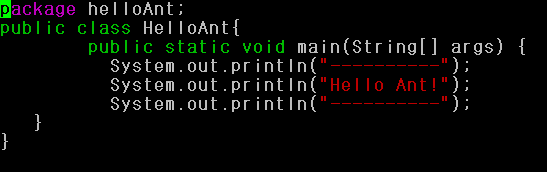
Build a free-style software project를 선택한다.

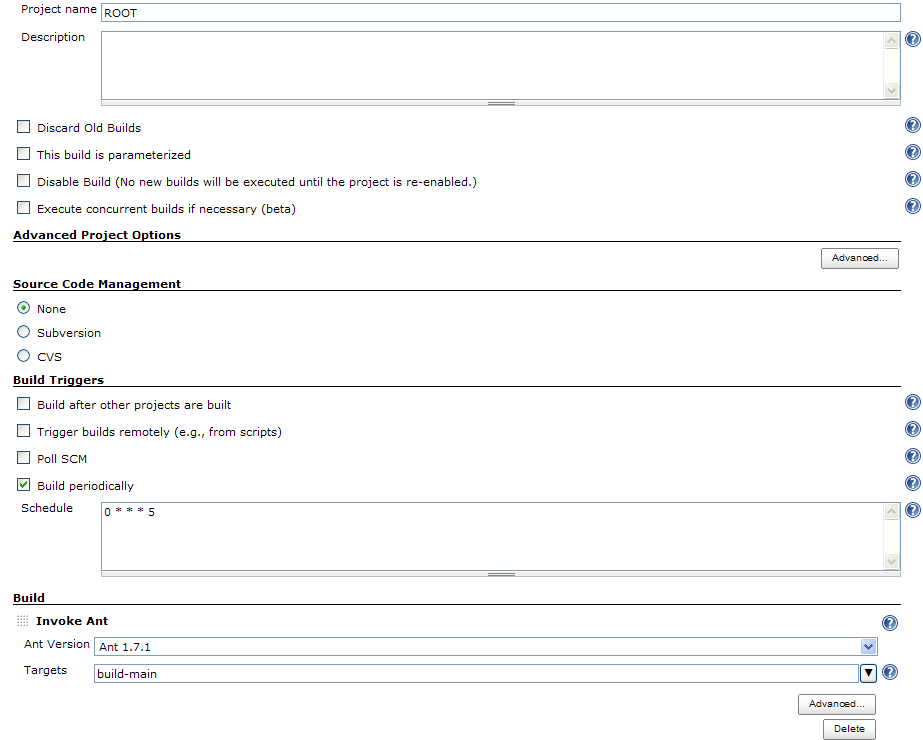


이 설정을 위해서 Ubuntu 에 Ant가 설치 되어있어야 한다.



1. Ubuntu환경에서 간단한 Java 소스 파일을 만든다.

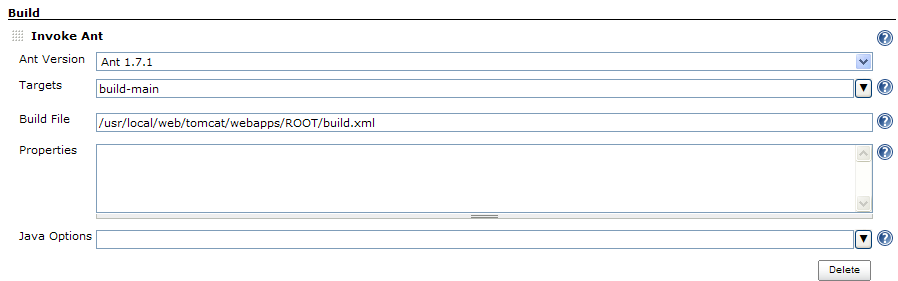




Java 소스를 빌드하기 위해 새로운 새로운 작업을 설정해준다.

먼저 허드슨 메인 페이지에 있는 새 작업 메뉴를 클릭한 후 빌드를 위한 프로젝트 설정을 해준다.

프로젝트 이름을 ROOT로 해주고 기본적인 설정을 해 준 뒤 build 도구로 Ant를 선택해준다.

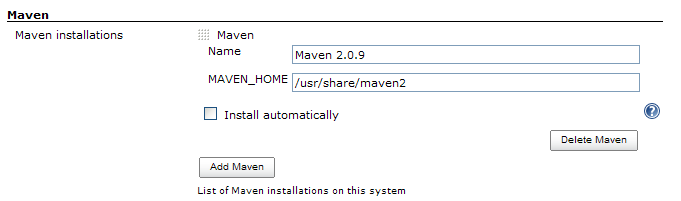


~~maven 빌드~~

1. Jenkins 새작업에서 maven2/3 Project를 선택

그림2.emf

1. Jenkins> Jenkins 관리 > Configure System



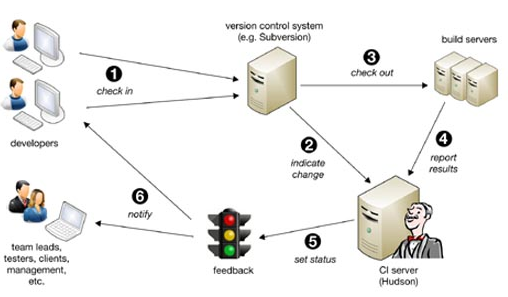
1. Jenkins> 프로젝트명 선택 > Configure



### 분산빌드 환경 구축

분산빌드 구조

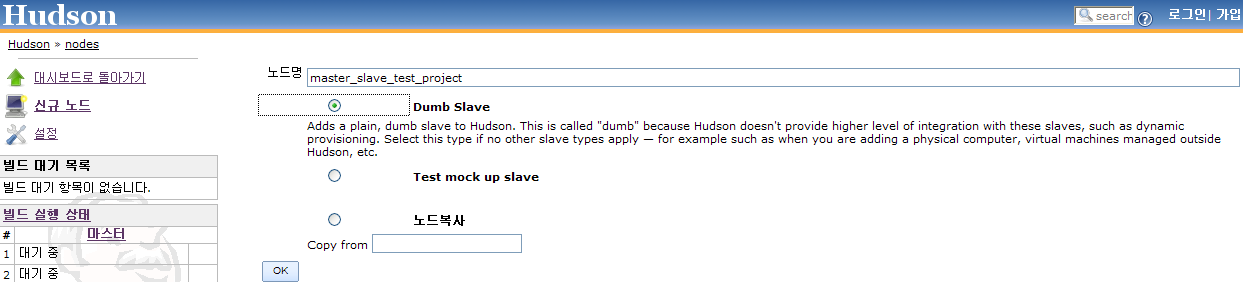
1.1.9버전 이상의 Jenkins에서는 master&slave 모드를 지원하여 분산빌드 기능을 제공하고 있다. 여러대의 컴퓨터에 빌드/테스트 부하를 분산시켜 진행시킬 수 있고, 여러 빌드서버에서 이루어지는 빌드작업을 제어할 수 있다. 또한 빌드결과를 Jenkins(CI서버)에 기록하고, 여러 플러그인 들을 이용하여 결과를 분석하고 관찰할 수 있다.



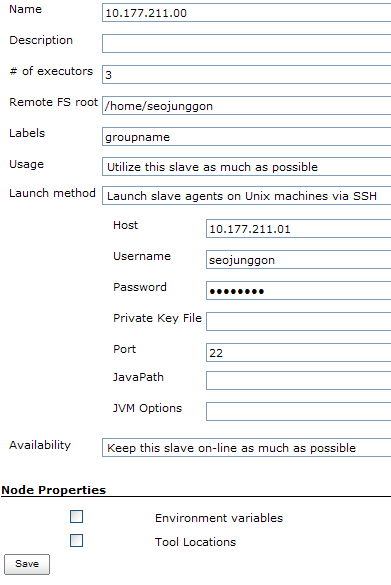
Jenkins에서 Linux master&Linux slave설정

유휴장비를 빌드서버로 사용하고 Jenkins(CI서버)에서 통합하여 관리하는 분산빌드 환경을 구축하기 위해, master&slave설정을 해보자. Jenkins서버가 master가 되고, 빌드서버가 slave가 된다. 설정에 앞서, Jenkins( master node)가 jdk 1.4 버전이상, ant 1.7 버전이상의 환경을 갖추어야만 한다. 마스터는 jenkins이 설치된 서버로 이미 설정이 되어 있으므로, slave를 아래의 방법으로 등록해보자.

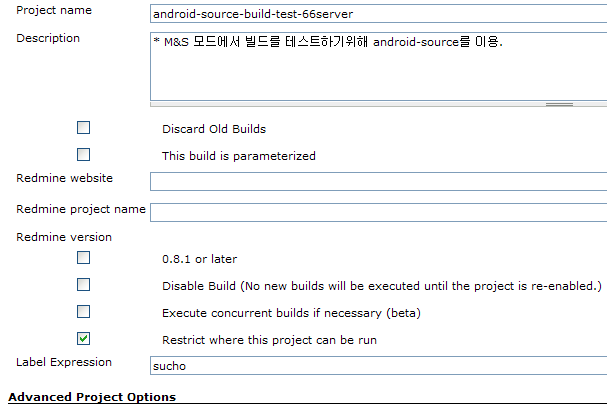
1. Jenkins master node에서 신규노드를 추가한다. ‘Jenkins관리->Manage Nodes->신규노드’ 에서 노드명을 적고, Dumb slave 옵션을 선택한 후 OK버튼을 누른다.



1. 필수적으로 적어주어야 하는 대상으로는, 노드명(slave hostname을 추천)을 적고 Remote FS root에는 slave의 작업 root절대경로를 적어준다. Labels에는 slave들을 논리적으로 grouping짓기 위한 명칭을 적는다. Launch method에는 자신이 원하는 방법을 선택하는데, ‘Launch slave agents on Unix machines via SSH’을 사용할 경우 Advanced를 눌러서, slave가 될 서버의 host와 계정, 비밀번호를 넣어주어야 한다. 반드시sudo 계정이 아니더라도, slave서버 내의 작업할 공간에 읽고,쓰고,실행할 수 있는 권한을 가진 계정이라면 가능하다. 다시말해, 위에서 설정한 Remote FS root경로에 접근할 수 있고, 그 경로 이하에 있는 폴더 및 디렉토리를 읽고,쓰고,실행할 수 있는 권한이 있는 계정을 넣어주면 되는것이다. Port는 22번을 이용한다. 이외의 선택은 default값으로 해도 무관하다. 설정이 끝나면 save버튼을 눌러 저장을 한다.



1. 2번과정에서 지정한 Labels이름에 여러가지 프로젝트를 묶어줄 수 있다. 이 작업을 하기 위해 필요한 Plugin을 Jenkins에 설치해보자. ‘Jenkins관리->Manage Plugins->설치가능’ 에서 ‘Matrix Tie Parent plugin’을 찾아 선택한 후, 오른쪽 하단에 있는 ‘설치’버튼을 누른다. 정상적인 환경에서는 플러그인이 자동으로 설치될 것이고, error가 발생한다면, Jenkins서버의 network설정을 확인하거나, 방화벽 설정을 확인해본다.
2. 이제 Jenkins(CI서버)에 작업(프로젝트)설정을 변경하여, slave node에서 실행되도록 해보자. slave node에서 돌리고자 하는 프로젝트의 configure에 들어간다. 3번과정에서 설치한 플러그인으로 인해 아래의 그림과 같이 ‘Restrict where this project can be run’ 라는 check박스가 보일 것이다. Jenkins의 버전이 낮을 경우 ‘Tie this project to a node’라고 보일 수 도 있지만 기능은 동일하다. 만약 Mantis plugin을 설치한 경우라면, Mantis목록 하단에 존재할 수 도 있다.



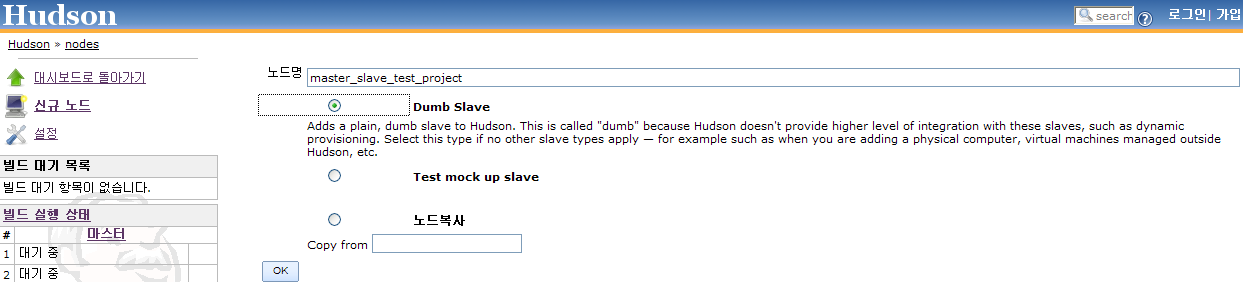
‘Restrict where this project can be run’ check박스를 선택하고, Label Expression 란에는 해당 프로젝트가 돌아가길 원하는 slave서버의 Label명을 적어준다. 예를들어, ‘android-source-build-test-66server’ 프로젝트가 10.177.211.01의 slave노드에서 빌드되길 원한다면, 10.177.211.01서버를 slave로 등록할 때(2번 과정) 적어준 Label명인 sucho를 동일하게 적어주면 된다.

1. 그 외에 설정은 자신의 소스를 빌드하는데 있어, 필요에 따라 설정한 후 오른쪽 하단의 ‘save’버튼을 눌러 저장한다. 그 프로젝트를 빌드해보면, Huson(CI서버)서버가 아닌 slave노드(빌드서버)에서 빌드되는 것을 확인할 수 있다.

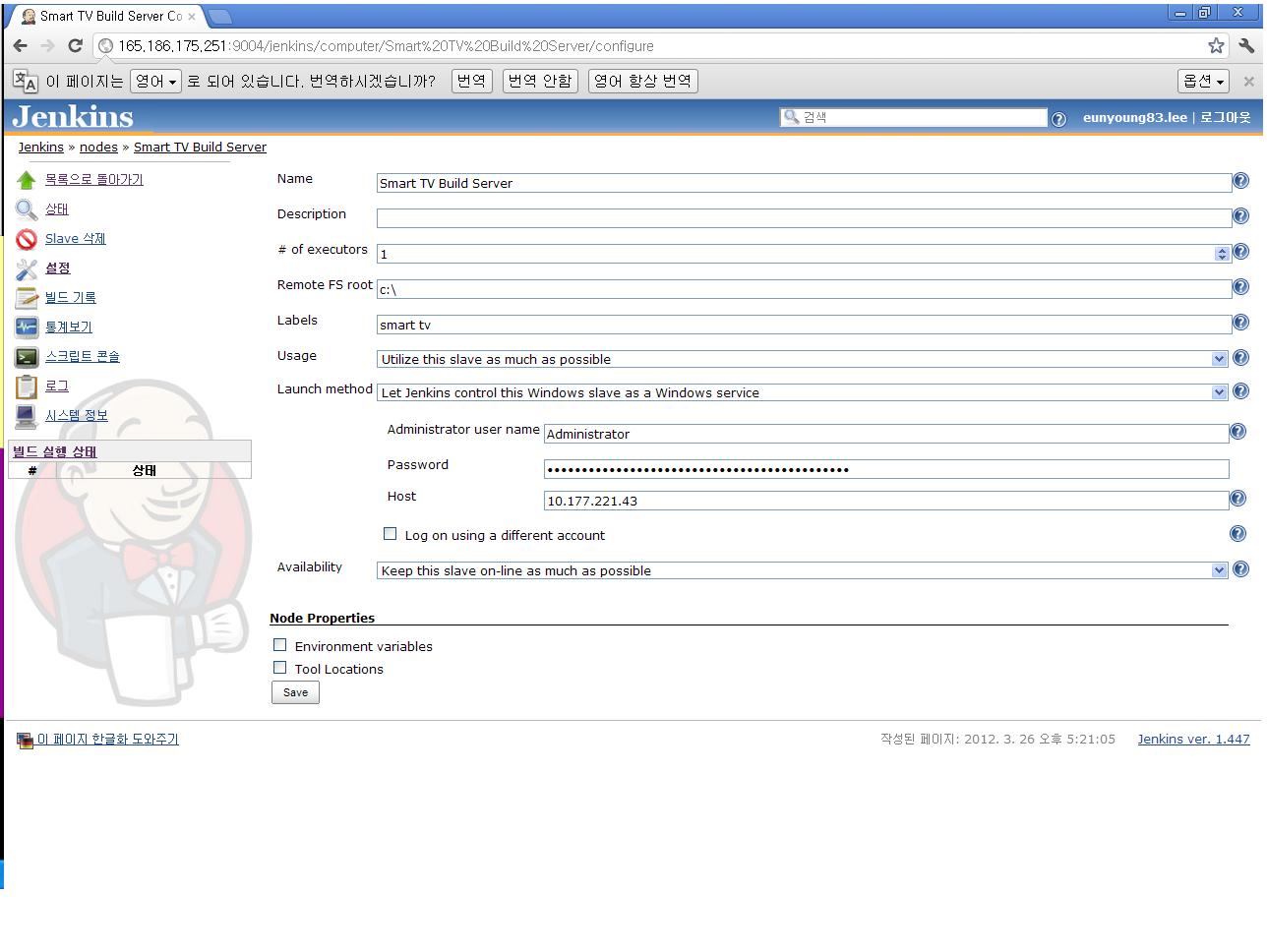
Jenkins에서 Linux master& Window slave설정

개발 팀의 특성에 따라 개발 서버를Linux로 사용할 수도 있지만 Windows 서버를 사용하기도 한다. CI가 운영되는 Master 환경이 Linux 이고 slave로 운영될 개발팀의build 서버 환경이 windows 서버일때 node 설정의 방법은 Linux master& Linux slave설치하는 방법과 유사하다.

1. Jenkins master node에서 신규노드를 추가한다. ‘Jenkins관리->Manage Nodes->신규노드’ 에서 노드명을 적고, Dumb slave 옵션을 선택한 후 OK버튼을 누른다.

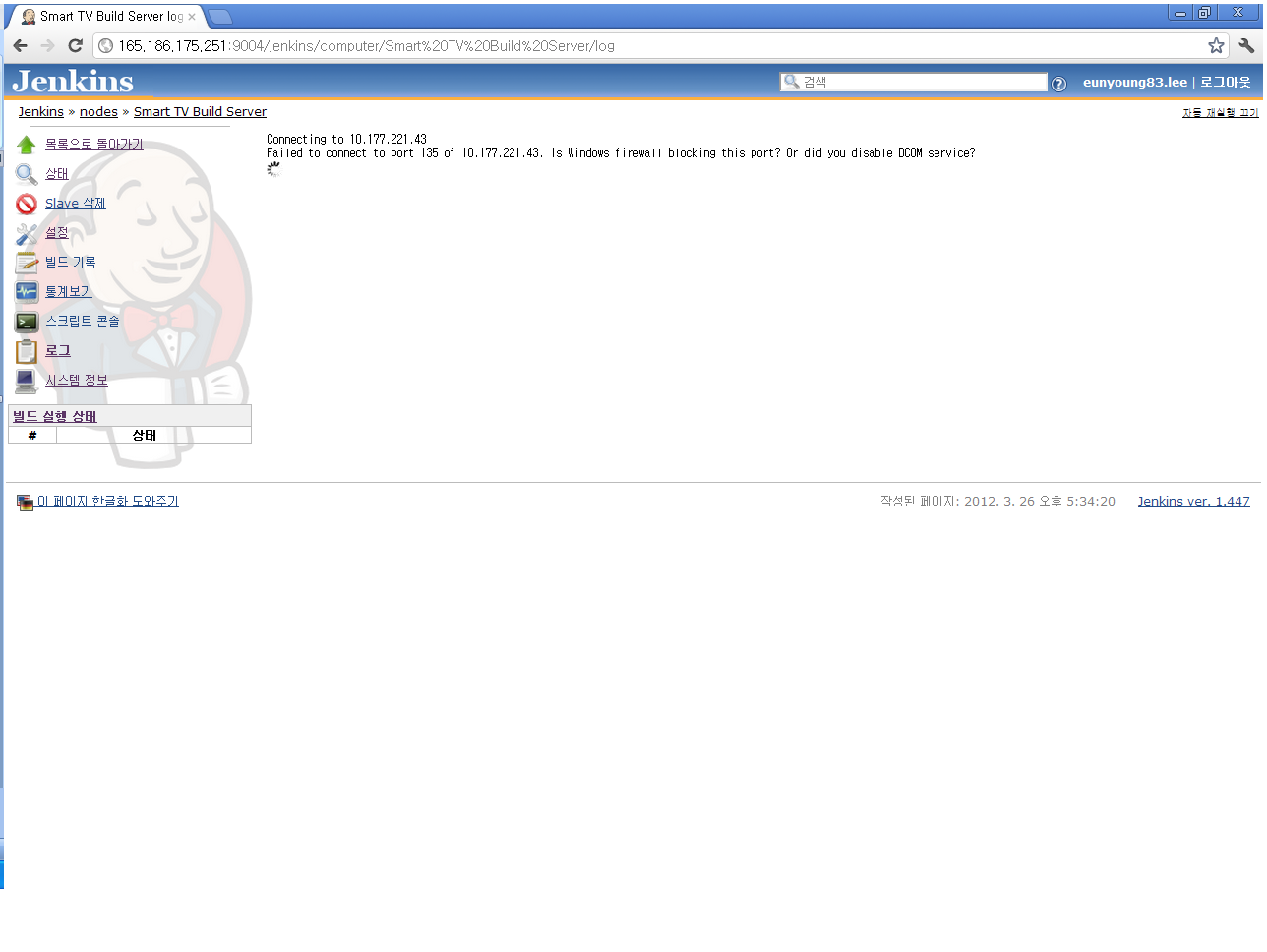


1. 필수적으로 적어주어야 하는 대상으로는 Linux slave와 유사하지만 Launch method 를 다음과 같이 “ Let Jenkins control this Windows slave as a Windows service” 를 선택한다. 나머지 설정은 Linux slave 설정 방법(1.3.3.2Jenkins에서 Linux master&Linux slave설정)을 참고한다.



K-1.png Window slave 로 설정할 때는 몇가지 제약 사항이 따른다. 반드시 135번 port 와 DCOM Service 활성화를 사전에 처리해야 한다. 미설정시 다음과 같은 오류가 뜰 수 있다.

Is Windows firewall blocking this port? Or did you disable DCOM service?



1. Port 135

Jenkins 에서 slave로 연결할 때 135번 port를 사용하는 것으로 설정되어 있다. 개발 팀에 요청하여 windows 서버의 방화벽을 해제해서 135번 port를 사용할 수 있도록 사전에 요청해야한다. Waterwall이 설치되었기 때문에 개발자 스스로가 활성화된 port 135를 강제로 종료할 수 없다. Windows 서버에 PC plus exception 신청하여 개발자 스스로가 방화벽 및 port 설정을 변경할 수 있도록 가이드 한다.

1. DCOM service

DCOM은 네트웍 상에서 클라이언트 프로그램 객체가 다른 컴퓨터에 있는 서버 프로그램 객체에 서비스를 요청할 수 있도록 해주는 마이크로소프트의 개념이자 프로그램 인터페이스들이다. COM은 같은 컴퓨터(윈도우95나 NT 시스템) 내에서 사용될 수 있도록 클라이언트와 서버에 인터페이스 집합을 제공한다.

예를 들어, 어떤 웹 사이트에 자신의 웹서버가 아닌 다른, 즉 네트웍 상의 보다 특정한 서버에서만 수행되는 스크립트나 프로그램을 가지도록 페이지를 만들 수 있을 것이다. 그 웹사이트의 프로그램(마치 클라이언트 객체처럼 동작하는)은 DCOM 인터페이스를 이용해, 필요한 절차를 수행하고 결과를 웹 서버 사이트에 돌려 주는 특정한 서버 객체에 RPC를 전달할 수 있으며, DCOM은 그 결과를 웹 페이지 뷰어에 넘긴다.

따라서 Windows slave를 설정하려면 DCOM sevices를 사용함으로 상태로 변경해야 한다.

(DCOM사용/사용안함 설정)

1. 구성 요소 서비스>콘솔 트리에서 컴퓨터 폴더를 클릭>

2. DCOM찾기 > 마우스 오른쪽 단추 클릭> 속성 클릭

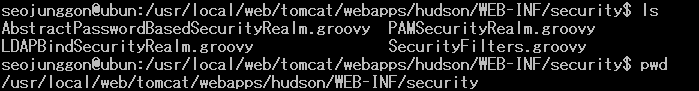
3. 기본속성 > DCOM 사용확인란 선택> 확인버튼 클릭

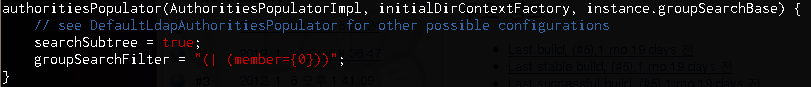
기본 속성 탭을 클릭합니다. >

### LDAP 설정

사전 작업

- {CATALINA\_HOME}/webapps/jenkins/WEB-INF/security/**LDAPBindSecurityRealm.groovy**의  
groupSearchFilter 부분을 하기와 같이 바꾼다.





[참고] Jenkins 로그인과 관련하여 작업이 원활하지 않을 경우,

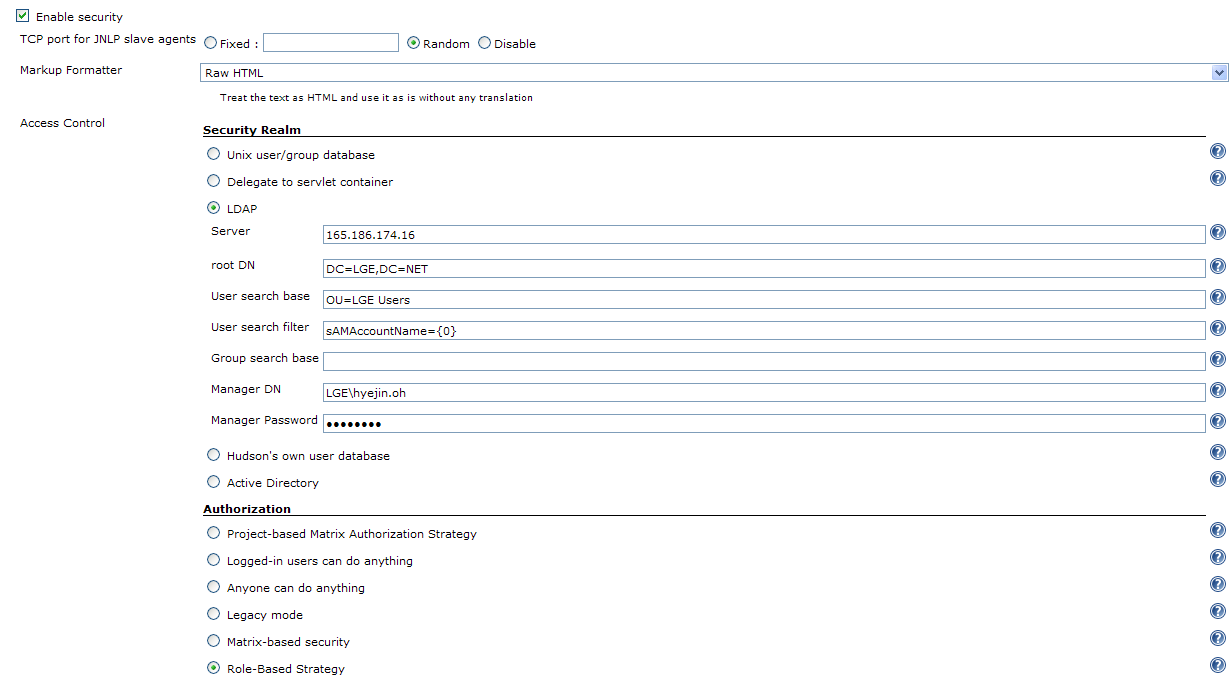
{CATALINA\_HOME}/logs의 내용을 tracking하여 debugging 한다.

설정

1. Jenkins 관리 > Manage Plugins 에서 “Role-based Authorization Strategy” Plugin을 설치한다.



1. Jenkins 관리 > Configure System에서 하기와 같이 설정한다.



K-1.png Configur System 에서 LDAP 설정을 save하게 되면 기존의 계정들은 없어지게 된다. 따라서 혹시 LDAP 설정이

잘못되어 login이 되지 않을 경우를 대비해 LDAP 설정을 하기 전에 anonymous에 admin 권한을 부여한다.

1. Jenkins 관리 > Manage Roles > Manage Roles 에서 Role을 할당한다.

3-1) Global roles: Jenkins 전체적인 부분에 대한 권한 설정 (Project roles 보다 상위 설정)

3-2) Project roles: Project 별로 각각 권한 설정

- Role to add: Project 설정에 대한 이름을 사용자가 임의의 이름으로 설정

- Pattern: Project 이름을 채우는 란

(비슷한 프로젝트의 이름일 경우 Pattern을 인식 못하는 경우가 발생하므로

그럴 경우 프로젝트 명을 확연히 다른 이름으로 변경 후 적용)



K-1.png - anonymous에게 주어진 권한 들은 모든 user에게 동일하게 적용된다.

예) anonymous에게 모든 Project read 권한 부여

user1에게는 project1 에 대한 read 권한만 부여

결과적으로 user1은 anonymous에 대한 권한인 모든 project를 read 할 수 있는 권한을 가짐.

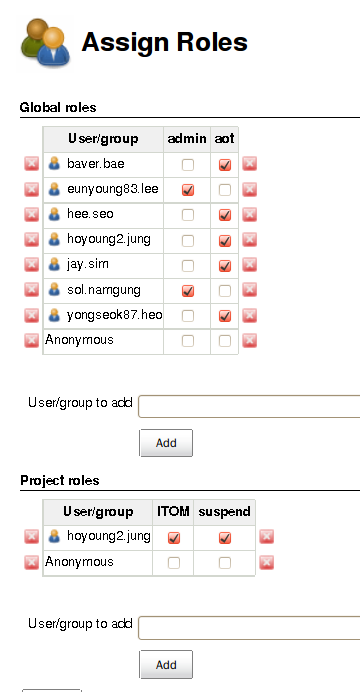
- Global roles ⊃ Project roles

Global roles

Project roles

1. Jenkins 관리 > Manage Roles > Assign Roles

Manage roles 에서 설정한 권한들을 user들에게 부여해준다.



1. Jenkins을 재가동 시키고, 로그인이 되는지 확인한다.

[참고] security 관련 작업을 수행하다, 로그인이 안되는 작업이 발생하면,

.jenkins/config.xml 파일을 하기와 같이 재설정하고, 재시작시킨다.

<useSecurity>false</useSecurity>

정상적으로 세팅되었을 때의 conf.xml 파일은 하기와 같다.

seojunggon@ubun:~/.hudson$ vi /home/seojunggon/.hudson/config.xml

----------------------------------------------------------------------------------------------------------------------------------

<?xml version='1.0' encoding='UTF-8'?>

<hudson>

<version>2.0.0</version>

<numExecutors>2</numExecutors>

<mode>NORMAL</mode>

<useSecurity>true</useSecurity>

<authorizationStrategy class="com.michelin.cio.hudson.plugins.rolestrategy.RoleBasedAuthorizationStrategy">

<roleMap type="globalRoles">

<role name="admin" pattern=".\*">

<permissions>

<permission>hudson.model.Hudson.Read</permission>

<permission>hudson.model.Item.Create</permission>

<permission>hudson.model.View.Create</permission>

<permission>hudson.model.Item.Build</permission>

<permission>hudson.model.Item.Read</permission>

<permission>hudson.model.Computer.Configure</permission>

<permission>hudson.model.Run.Update</permission>

<permission>hudson.model.Hudson.Administer</permission>

<permission>hudson.model.View.Configure</permission>

<permission>hudson.model.Computer.Delete</permission>

<permission>hudson.scm.SCM.Tag</permission>

<permission>hudson.model.Run.Delete</permission>

<permission>hudson.model.Item.Configure</permission>

<permission>hudson.model.View.Delete</permission>

<permission>hudson.model.Item.Delete</permission>

<permission>hudson.model.Item.Workspace</permission>

</permissions>

<assignedSIDs>

<sid>junggon.seo</sid>

<sid>hyeonjeong.shin</sid>

<sid>woojun.jung</sid>

<sid>bokyung2.kim</sid>

<sid>hyejin.oh</sid>

<sid>eunhee.choi</sid>

</assignedSIDs>

</role>

<role name="anonymous" pattern=".\*">

<permissions/>

<assignedSIDs>

<sid>anonymous</sid>

</assignedSIDs>

</role>

<role name="developer" pattern=".\*">

<permissions>

<permission>hudson.model.Hudson.Read</permission>

<permission>hudson.model.Item.Read</permission>

</permissions>

<assignedSIDs>

<sid>benjamin.choi</sid>

</assignedSIDs>

</role>

</roleMap>

<roleMap type="projectRoles"/>

</authorizationStrategy>

<securityRealm class="hudson.security.LDAPSecurityRealm">

<server>165.186.174.16</server>

<rootDN>DC=LGE,DC=NET</rootDN>

<userSearchBase>OU=LGE Users</userSearchBase>

<userSearch>sAMAccountName={0}</userSearch>

<managerDN>LGE\hyejin.oh</managerDN>

<managerPassword>amhrdzY3NTA=</managerPassword>

</securityRealm>

<markupFormatter class="hudson.markup.RawHtmlMarkupFormatter"/>

<jdks>

<jdk>

<name>/usr/lib/jvm/java-1.5.0-sun-1.5.0.19</name>

<home>/usr/lib/jvm/java-1.5.0-sun-1.5.0.19</home>

<properties/>

</jdk>

</jdks>

<viewsTabBar class="hudson.views.DefaultViewsTabBar"/>

<myViewsTabBar class="hudson.views.DefaultMyViewsTabBar"/>

<clouds/>

<slaves>

<slave>

<name>10.177.211.65</name>

<description></description>

<remoteFS>/home/asapadmin</remoteFS>

<numExecutors>3</numExecutors>

<mode>NORMAL</mode>

<retentionStrategy class="hudson.slaves.RetentionStrategy$Always"/>

<launcher class="hudson.slaves.JNLPLauncher"/>

<label>sucho</label>

<nodeProperties>

<hudson.tools.ToolLocationNodeProperty>

<locations>

<hudson.tools.ToolLocationNodeProperty\_-ToolLocation>

<type>hudson.model.JDK$DescriptorImpl</type>

<name>/usr/lib/jvm/java-1.5.0-sun-1.5.0.19</name>

<home>/usr/lib/jvm/java-6-sun</home>

</hudson.tools.ToolLocationNodeProperty\_-ToolLocation>

</locations>

</hudson.tools.ToolLocationNodeProperty>

</nodeProperties>

</slave>

</slaves>

<quietPeriod>5</quietPeriod>

<scmCheckoutRetryCount>0</scmCheckoutRetryCount>

<views>

<hudson.model.AllView>

<owner class="hudson" reference="../../.."/>

<name>All</name>

<filterExecutors>false</filterExecutors>

<filterQueue>false</filterQueue>

</hudson.model.AllView>

</views>

<primaryView>All</primaryView>

<slaveAgentPort>0</slaveAgentPort>

<label></label>

<nodeProperties/>

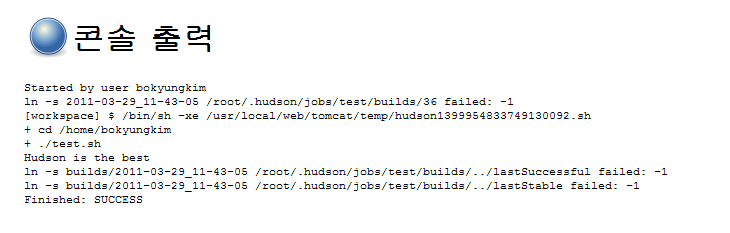
<globalNodeProperties/>

<disabledAdministrativeMonitors/>

</hudson>

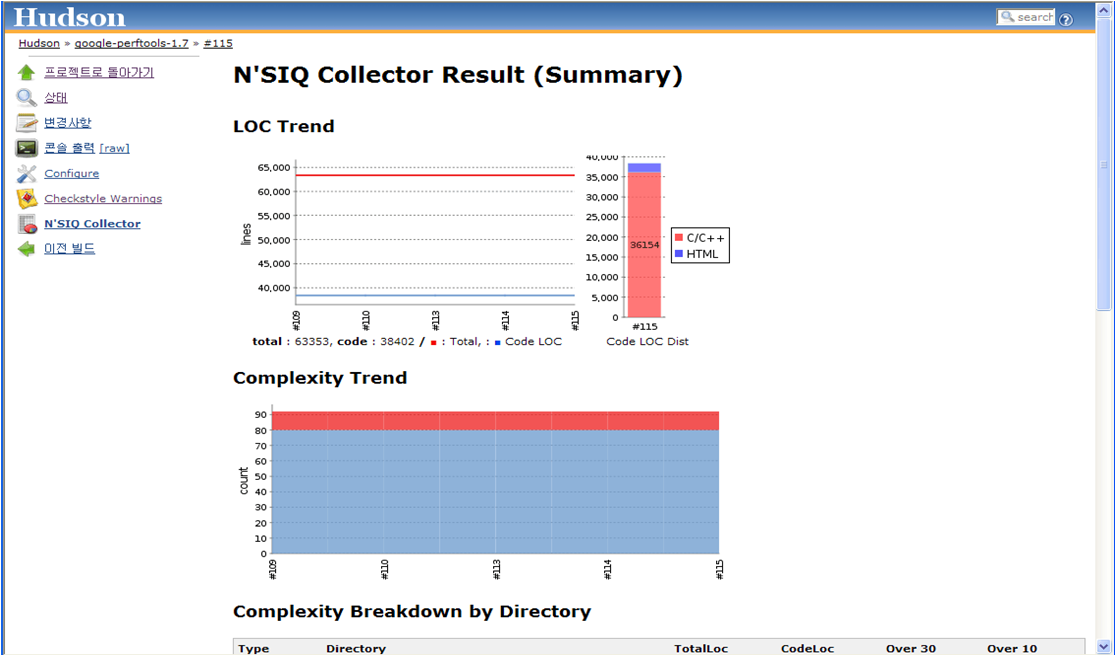
## 실행 결과

• 성공

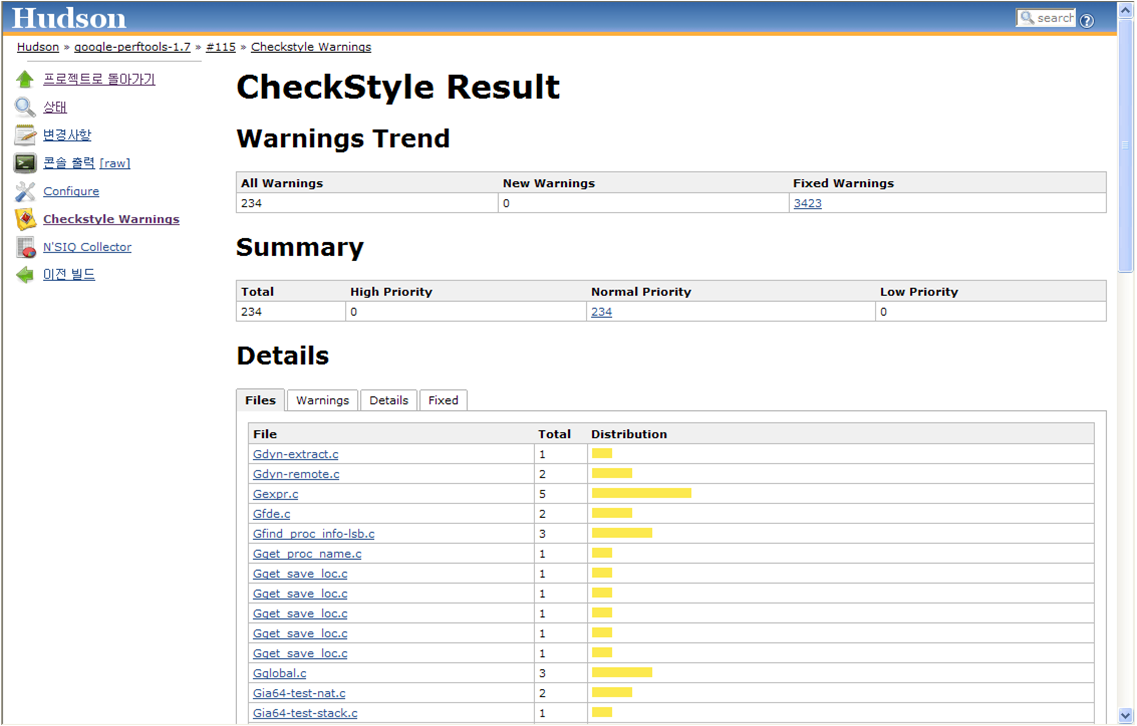


콘솔화면으로 빌드가 SUCCESS된 것을 확인 할 수있다. 성공하면 다음과 같은 품질 분석이 수행된다.

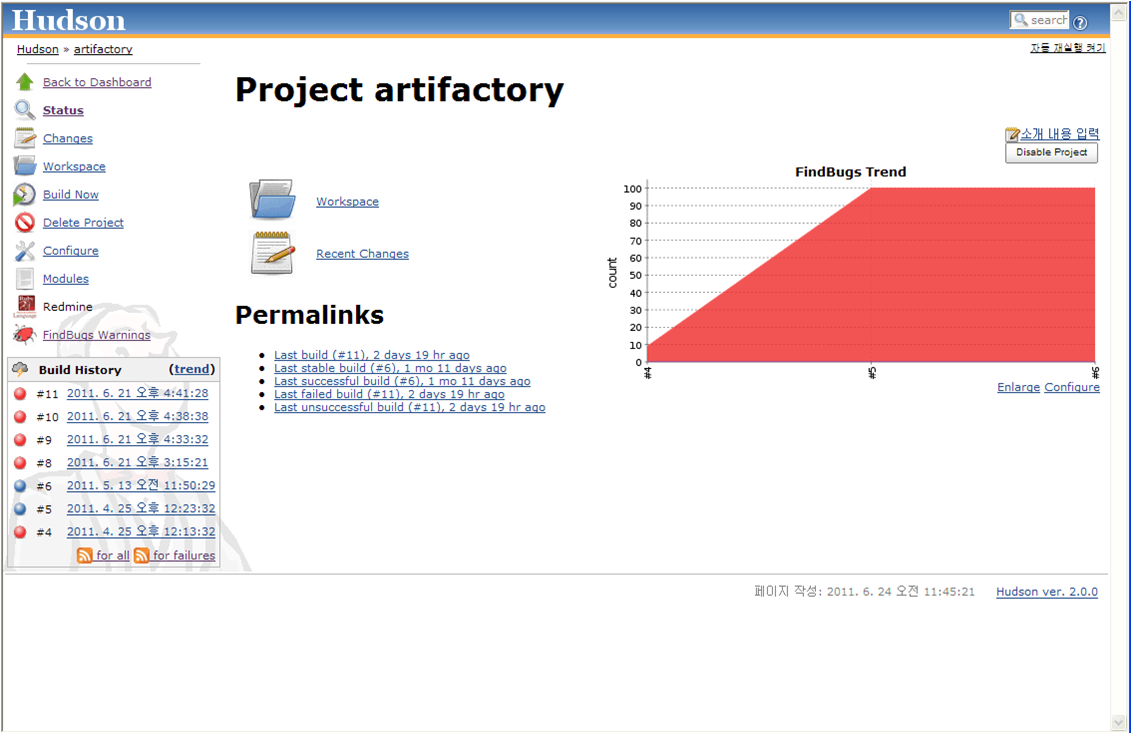
1. **복잡도 체크** (N’SIQ Collector) : SW복잡도/LOC 측정



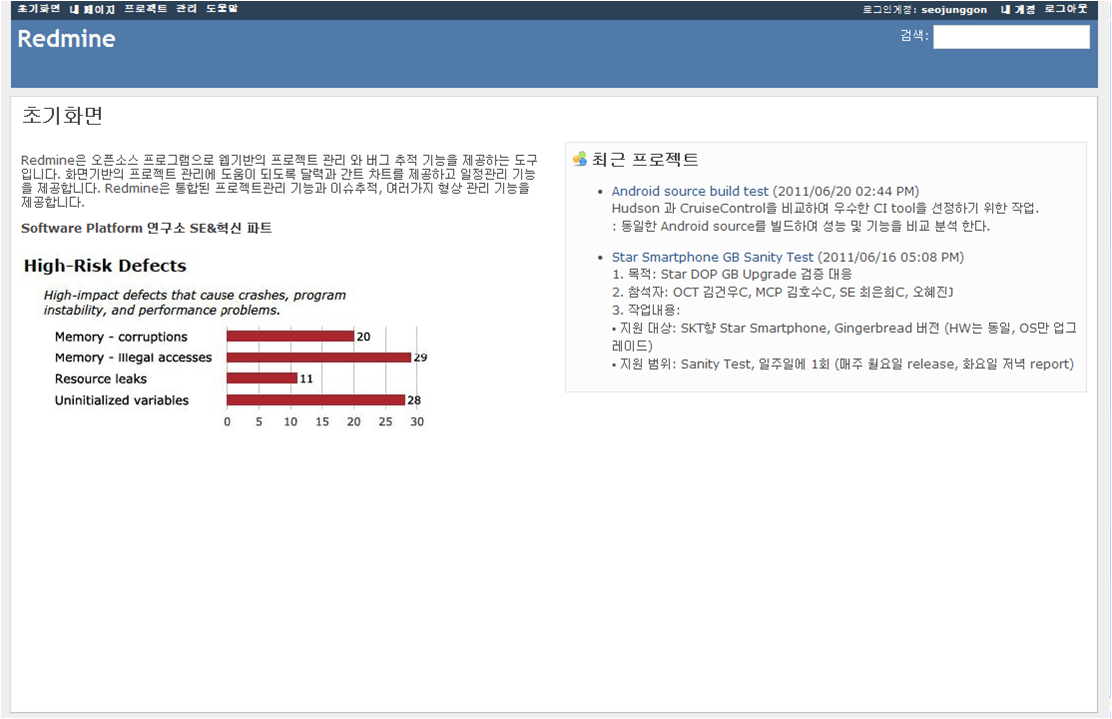
1. **코딩룰 체크** (N’SIQ CppStyle) : C/C++ 에 대한 45개 이상의 코딩 규칙 여부를 검사



1. **정적코드 분석** (Findbugs) : JAVA에 대한 **370**개 이상의 잠재적 코드 결함 유형을 분석



1. **결함관리**(Redmine) : 테스팅에서 탐지된 결함 추적/관리



• 실패

실패했을 경우 아래와 같은 붉은색 공으로 대시보드에 표시가 되며 실패 원인을 팀원들에게 이메일로 통보해준다.



## Trouble shooting

### Tomcat log 확인 방법

Jenkins의 오류 발생시 발생 메시지만으로 오류의 확인이 불가능하거나 아예 접속 자체가 안된다면 tomcat web engine catalina에 대한 log를 확인하는 방법을 통해서 오류가 일어나기 전까지의 기록을 확인해 볼수 있습니다. 이 방법은 효과적으로 error의 정확한 원인에 대해서 찾아내는 방법이 될수 있습니다.

1. Log 폴더로 이동

$cd $TOMCAT\_HOME/logs

\*$TOMCAT\_HOME : tomcat이 설치된 위치

1. Log 확인

$vi catalina.“날짜”.log

\*Jenkins는 Catalina를 통해서 실행되는 web application이기 때문에 catalina에 대한 로그를 확인하시면 됩니다.

1. Log 종류

Catalina : Catalina engine(webapp 폴더 안에 있는 web application을 실행시키는)에 대한 로그

Host-manager : Host-manager (Web 으로 접속해서 web application에 대해 설정)에 대한 로그

Manager : Manager(Web으로 접속해서 tomcat에 대해서 설정하는 web application )에 대한 로그

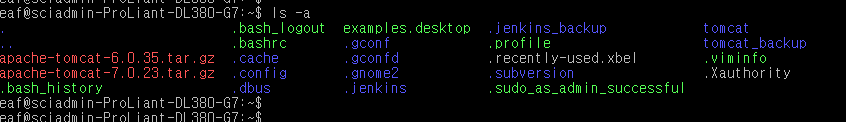
### Tomcat 재 설치 방법

Jenkins를 물리적으로 다른 서버로 이전하는 경우 혹은 tomcat에 어떠한 문제가 발생하여 재 설치 하는 경우에는 $HOME/.jenkins 에 대한 Data만 유지하게 된다면 기존 운영하던 Jenkins의 기록 및 plug-in, 설정값을 유지 할수 있습니다.

* 예제 화면 Tomcat shutdown/startup  => Jenkins 접속 불가



1. Tomcat 재설치를 결정
2. 문제가 발생한 그룹의 Jenkins 서버(251) ls –a 옵션 후 숨겨진 .jenkins 파일의 내용 유지(backup)



1. 문제가 발생한 그룹의 tomcat 삭제
2. Tomcat version 6.0 설치(정상적으로 동작하는 그룹의 tomcat을  copy)

/home/sciadmin/재설치용톰캣/tomcat copy

1. /home/그룹/tomcat/conf/ 이동
2. Vi server.xml 에서 포트번호를 변경함
3. Tomcat 실행

### Plug-in 충돌에 의한 지속적인 tomcat의 down 대책

[1/9~10 간 ODS team의 Jenkins server의 불안정 상태에 대한 trouble shooting 내용입니다.]

* 현상 : Tomcat server의 지속적인 down 현상, 재 부팅해도 tomcat을 다시 설치해도 지속적으로 문제 발생
* 원인 : 사용하지 않은 plug-in(sona, gerrit, Cobertura)의 충돌로 인한 catalina engine의 error 발생으로 추정
* 대책 : plug-in 설정  설치된 플러그인 목록  해당 plug-in 체크 해제

trouble shooting 과정

1. Tomcat log 확인

SEVERE: The web application [/jenkins] appears to have started a thread named [com.google.common.base.internal.Finalizer] but has failed to stop it. This is very likely to create a memory leak.

Jan 10, 2012 10:33:51 AM org.apache.catalina.loader.WebappClassLoader clearReferencesThreads

SEVERE: The web application [/jenkins] appears to have started a thread named [ViewJob reload thread] but has failed to stop it. This is very likely to create a memory leak.

Jan 10, 2012 10:33:51 AM org.apache.catalina.loader.WebappClassLoader clearReferencesThreads

SEVERE: The web application [/jenkins] appears to have started a thread named [TCP slave agent listener port=0] but has failed to stop it. This is very likely to create a memory leak.

WARNING: Failed to shut down gerrit-trigger

java.lang.NullPointerException

at hudson.PluginWrapper.stop(PluginWrapper.java:343)

at hudson.PluginManager.stop(PluginManager.java:545)

at jenkins.model.Jenkins.cleanUp(Jenkins.java:2458)

at hudson.WebAppMain.contextDestroyed(WebAppMain.java:350)

at org.apache.catalina.core.StandardContext.listenerStop(StandardContext.java:4244)

at org.apache.catalina.core.StandardContext.stop(StandardContext.java:4879)

at org.apache.catalina.core.ContainerBase.removeChild(ContainerBase.java:932)

at org.apache.catalina.startup.HostConfig.undeployApps(HostConfig.java:1357)

at org.apache.catalina.startup.HostConfig.stop(HostConfig.java:1328)

at org.apache.catalina.startup.HostConfig.lifecycleEvent(HostConfig.java:326)

at org.apache.catalina.util.LifecycleSupport.fireLifecycleEvent(LifecycleSupport.java:142)

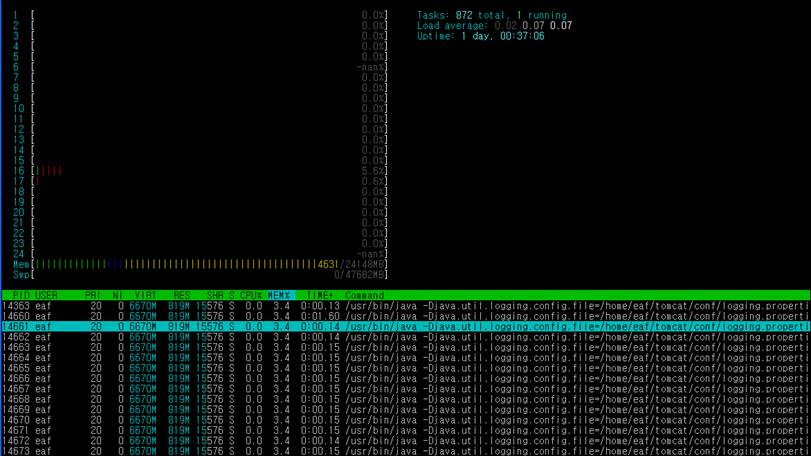
1. Log 분석 결과

Jenkins(web application)가 종료되는 과정에서 실행중 생성했던 thread들이 종료되지 못하고 memory leak을 발생시키고 있는 것으로 확인

사용하고 있지도 않은Gerrit-trigger가 지속적인 error를 발생

1. 실제 Mamory leak이 발생하고 있는지 확인

실제로 비 정상적으로 많은 thread들이 Jenkins server내에서 돌아가고 있음



1. 대처 방안

Jenkins 내에서 사용하고 있는 필요하지 않은 plug-in을 실행종료(gerrit trigger 포함)

Jenkins server의 재부팅

### AWT관련 Headless error 대책

[Error Massage]

AWT is not properly configured on this server. Perhaps you need to run your container with "-Djava.awt.headless=true"?

java.lang.NullPointerException

at sun.awt.X11FontManager.getDefaultPlatformFont(X11FontManager.java:779)

at sun.font.SunFontManager$2.run(SunFontManager.java:428)

at java.security.AccessController.doPrivileged(Native Method)

at sun.font.SunFontManager.<init>(SunFontManager.java:371)

at sun.awt.X11FontManager.<init>(X11FontManager.java:32)

at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)

at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)

at java.lang.reflect.Constructor.newInstance(Constructor.java:525)

at java.lang.Class.newInstance0(Class.java:372)

at java.lang.Class.newInstance(Class.java:325)

at sun.font.FontManagerFactory$1.run(FontManagerFactory.java:80)

at java.security.AccessController.doPrivileged(Native Method)

at sun.font.FontManagerFactory.getInstance(FontManagerFactory.java:71)

at java.awt.Font.getFont2D(Font.java:489)

at java.awt.Font.getFamily(Font.java:1186)

at java.awt.Font.getFamily\_NoClientCode(Font.java:1160)

at java.awt.Font.getFamily(Font.java:1152)

at java.awt.Font.toString(Font.java:1649)

at hudson.util.ChartUtil.<clinit>(ChartUtil.java:229)

at hudson.WebAppMain.contextInitialized(WebAppMain.java:175)

at winstone.WebAppConfiguration.<init>(WebAppConfiguration.java:897)

at winstone.HostConfiguration.initWebApp(HostConfiguration.java:131)

at winstone.HostConfiguration.<init>(HostConfiguration.java:73)

at winstone.HostGroup.initHost(HostGroup.java:85)

at winstone.HostGroup.<init>(HostGroup.java:45)

at winstone.Launcher.<init>(Launcher.java:197)

at winstone.Launcher.main(Launcher.java:398)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:601)

at Main.\_main(Main.java:268)

at Main.main(Main.java:96)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:601)

at org.tanukisoftware.wrapper.WrapperJarApp.run(WrapperJarApp.java:394)

at java.lang.Thread.run(Thread.java:722)

* 현황 : Jenkins 시작시 headless error massage를 나타내며 접속 불가
* 원인 : 기존 환경에 Open-6-jdk의 추가로 설치 및 default java compiler로 설정됨에 따라 기존에 사용되었던 Sun-6-java6과는 다른 환경설정이 필요함

필요한 환경( ttf-dejavu font, Catalina 엔진의 awt.headless에 대한 설정)

* 대책 : ttf-dejavu font 설치

       $TOMCAT\_HOME/conf/catalina.properties에 awt.headless=true 추가

1. 아래의 URL에서 문제 확인

Jenkins wiki <https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+got+java.awt.headless+problem>

Googling <http://stackoverflow.com/questions/8109607/headless-continuous-integration-with-jenkins>

1. ttf-dejavu 설치

$sudo apt-get install ttf-dejavu

1. awt.headless=true 속성값 추가

vi $TOMCAT\_HOME/conf/catalina.properties

.

.

.

.

마지막에 awt.headless=true 값 추가

1. tomcat 재실행 [Maven 사용 관련 정리](http://blog.naver.com/chidoo74/10098880659)**작성자**[민혁아빠](http://blog.naver.com/chidoo74)[Maven 사용 관련 정리](http://blog.naver.com/chidoo74/10098880659)**작성자**민혁

# Jenkins Plugin 설치 : N’SIQ Collector

## 소개

본 도구는 소스 코드를 정량적으로 측정하기 위한 메트릭 분석 툴이다. Freeware 코드 분석도구인 CodeAnalyzerPro를 내부적으로 호출하고, 그 결과를 정제하여 메트릭셋 정보를 추출한다.

본 도구가 제공하는 기능은 다음과 같다.

• 지정한 디렉토리 하위에 있는 코드를 대상으로 한번의 명령어 실행으로 다양한 언어에 대해 LOC 정보와

복잡도 정보 추출

• 지정한 디렉토리중 필요 없는 하위 디렉토리내의 코드 정보를 필터링 가능 초기 파일리스트를 지정하여,

변경 / 신규 생성된 코드만 측정 수행 가능

• Windows / Linux 지원

• CSV / HTML / EXCEL / XML 로 분석 결과 출력

본 도구를 사용하여 소스 코드로부터 추출 가능한 정보는 다음과 같다.

• 소스 코드 파일 개수 (C/C++, Java, JSP, PHP, Javascript, HTML 코드 파일 대상)

• 소스 코드 총 LOC (Lines of Code)

• 소스 코드의 코드 LOC (커멘트로만 구성된 라인과 빈줄을 제거한 나머지 줄수)

• 총 함수의 개수(C/C++, Java 코드 파일 대상)

• 각 함수의 총 LOC / 코드 LOC / Cyclomatic Complexity

• 총 함수중 Cyclomatic Complexity가 10 이상인 함수 개수 및 비율

• 총 함수중 Cyclomatic Complexity가 30 이상인 함수 개수 및 비율

## 설치

1. Jenkins Plugin 설치: Jenkins > Jenkins 관리 > Manage Plugins > 설치가능 Tab > Jenkins NSIQ Collector Plugin 체크 > 맨 하단 설치 버튼
2. Jenkins 재부팅

<http://dev.naver.com/projects/nsiqcollector/download>에서 N’SIQ Collector소스를 로컬 PC에 다운로드 한다.

**[참고]** GUI가 설치되어 있지 않은 환경이라면 아래와 같이 wget을 사용해서 다운로드 받는다.

Wget <http://dev.naver.com/frs/download.php/957/nsiqcollector_2.1.4.tar.gz>

$ wget <http://dev.naver.com/frs/download.php/957/nsiqcollector_2.1.4.tar.gz>

## 설정

1. N’SIQ Collector 실행 파일 위치 지정: Jenkins > Jenkins 관리 > Configure System

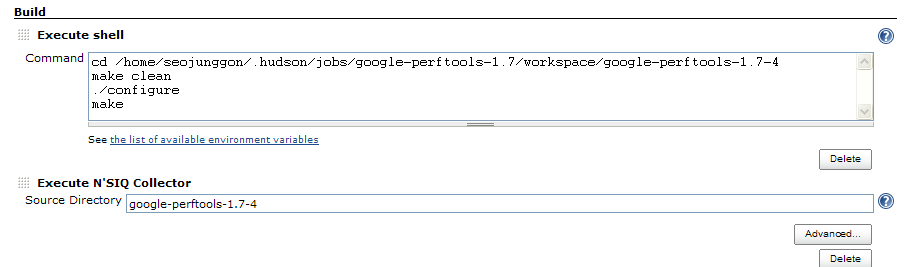
N’SIQ Collector 폴더를 입력해주는 것이 아니라 N’SIQ Collector 실행 파일을 입력해준다



\***주의** nsiqcollector의 폴더 위치를 설정해 주는 것이 아니라 폴더 안의 nsiqcollector 실행 파일을

정해주는 것입니다.

1. Build Script 및 디렉토리 설정(project->configure)



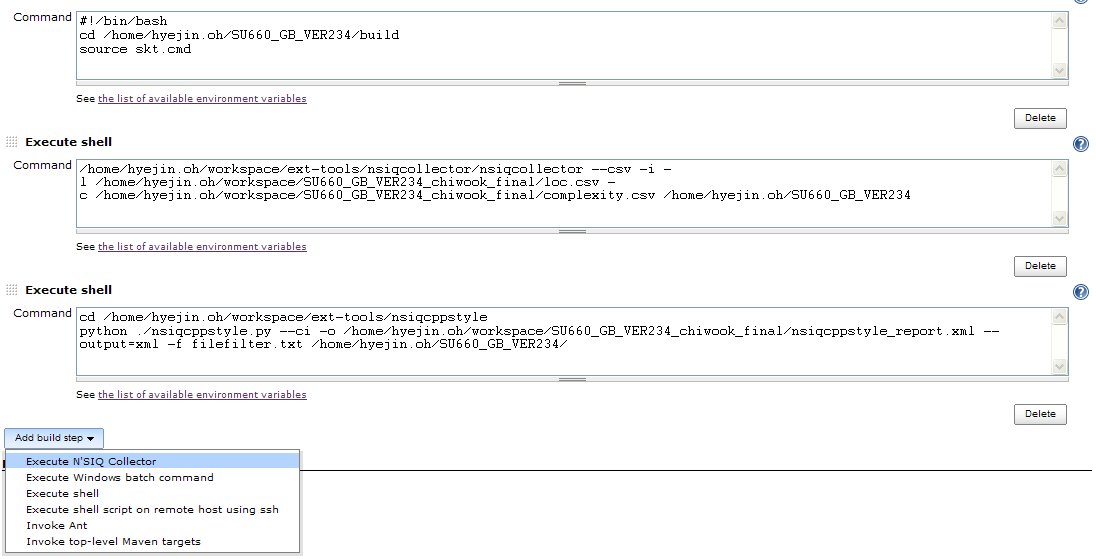
cd /home/asapadamin/.jenkins/jobs/eeeeee/workspace/google-perftools-1.7

./configure --enable-frame-pointers

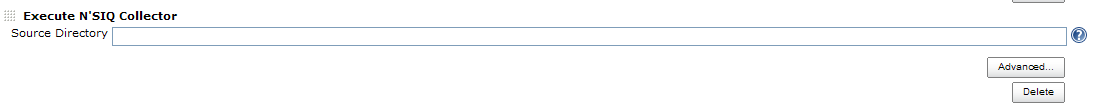
make clean

make

1. add build step에서 excute N’SIQ Collector를 추가시켜줌니다.



1. Source Directory에 Workspace안의 source 폴더의 위치를 적어줌니다.



1. Post-build Actions 설정



위의 그림과 같이 check해 주시면 됨니다.

## Window용

nsiqcollector.exe --csv -i -l c:\plugin\workspace\loc.csv -c c:\plugin\workspace\complexity.csv c:\plugin\ex\google

**명령어 설명**

nsiqcollector.exe : 실행파일

--csv : 출력파일 형식

-i : ci mode

-l "loc.csv가 설치될 디렉토리"\loc.csv : loc.csv 가 저장될 위치

-c "complexity.csv가 설치될 디렉토리" :\complexity.csv

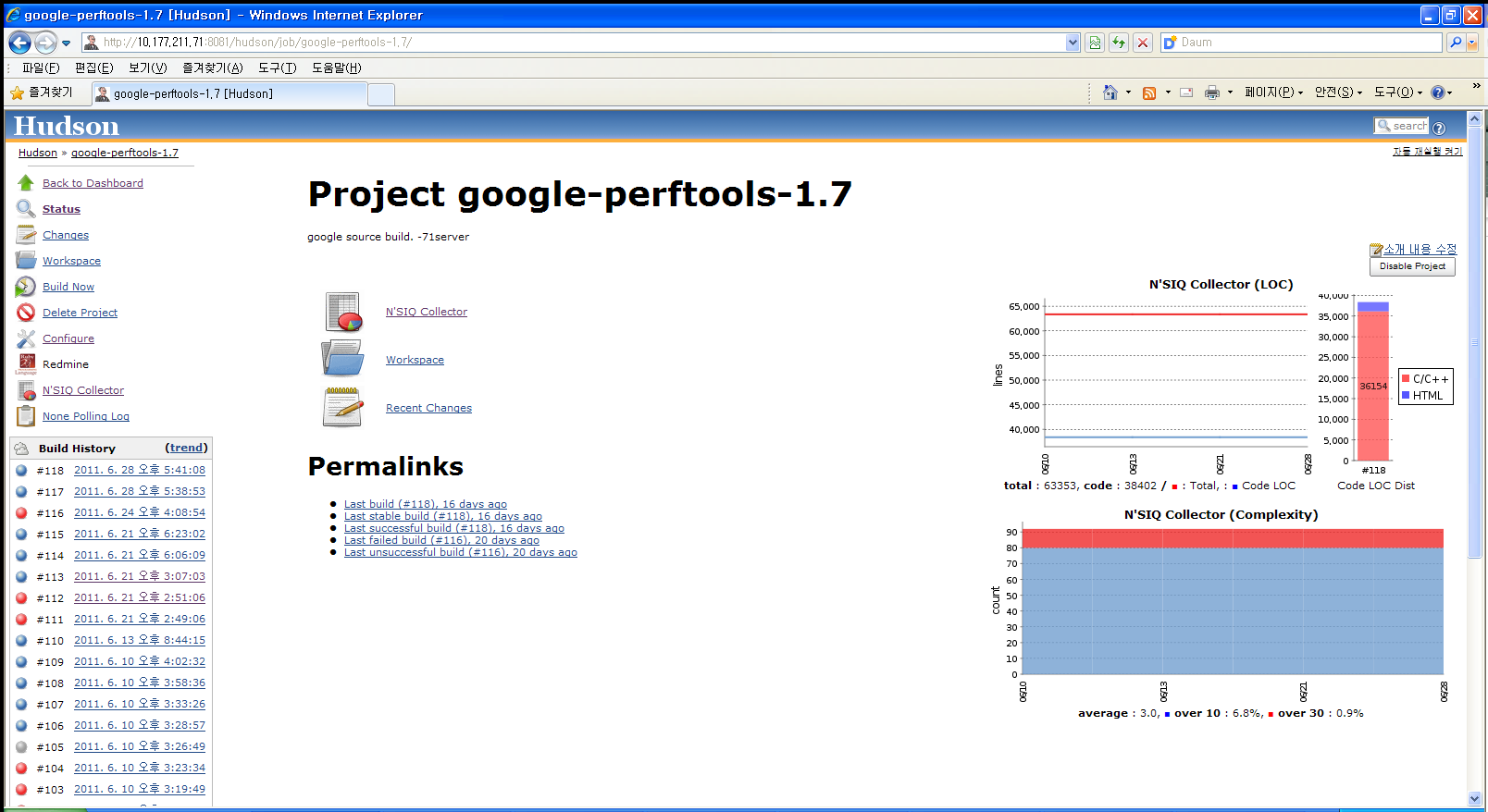
"source 대상 directory"

## 실행결과

1) 복잡도 결과 요약 확인

- 빌드 별 LOC, 복잡도 추이를 프로젝트 메인 페이지에서 확인 가능

- 좌측 하단 Build history에서 성공한 빌드를 누르면(빨간 박스) 빌드별 요약 페이지로 이동

**

**클릭!**

2) 빌드 별 결과 요약

- 성공한 빌드의 LOC, 복잡도 확인 가능함.

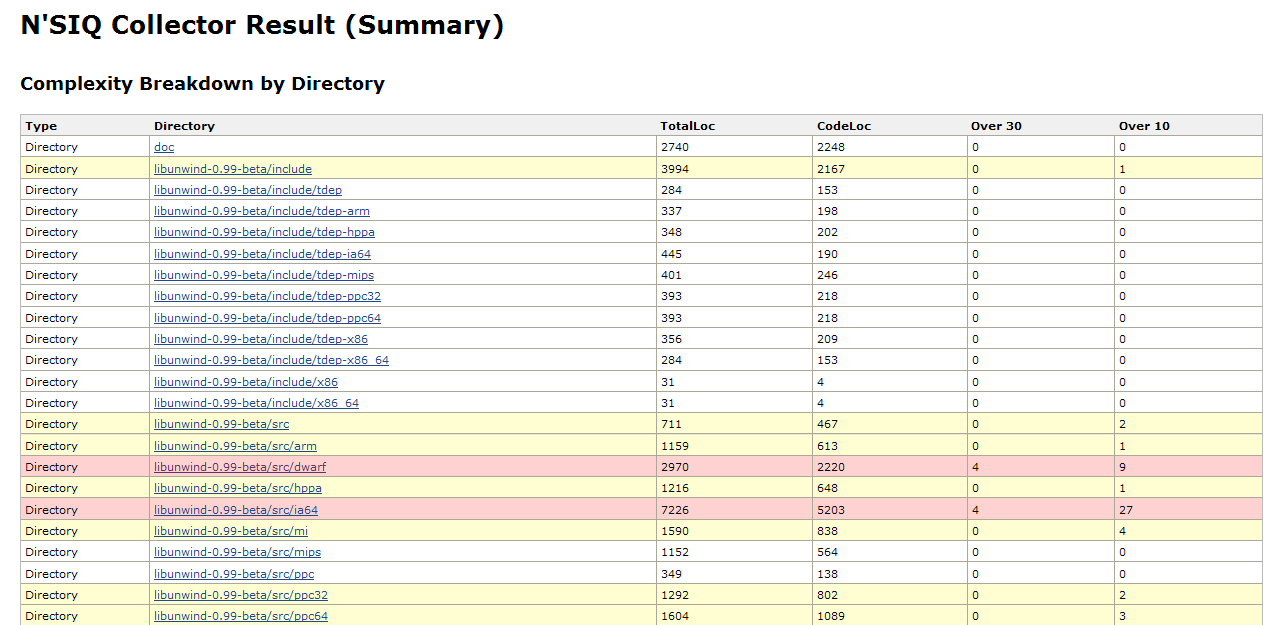


**클릭!**

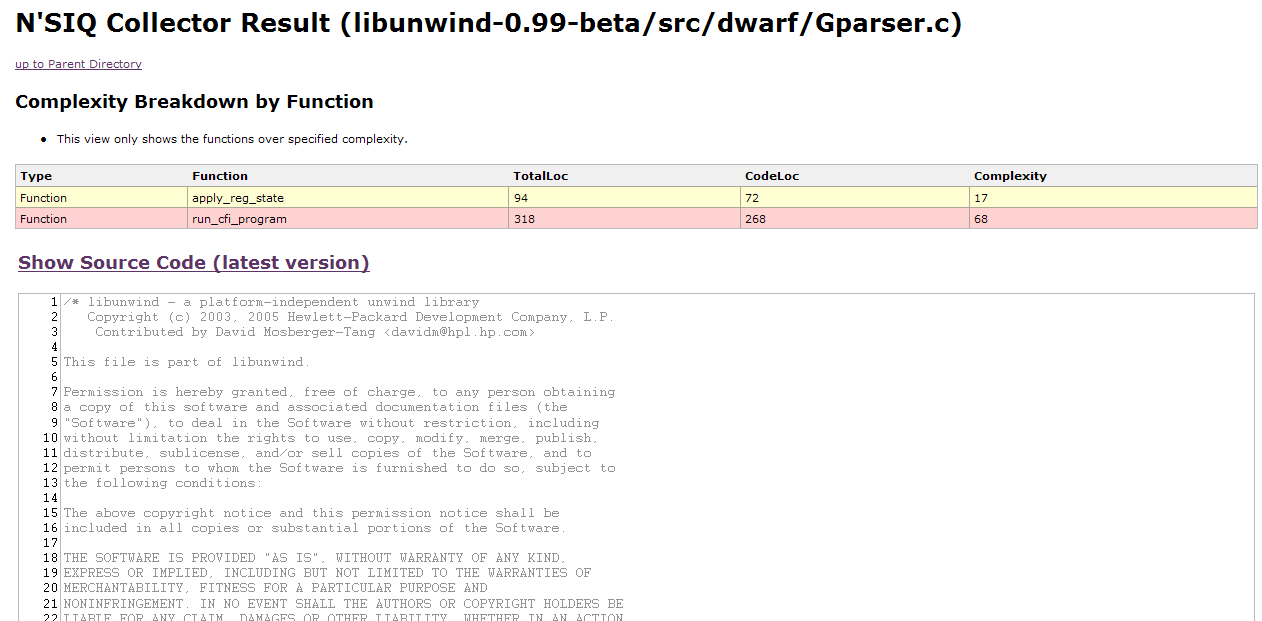
3) 디렉토리/파일/함수별 결과 확인

- 디렉토리/파일/함수별 LOC와 복잡도가 10, 30이상인 코드를 확인 가능

- 함수 단위 복잡도 확인 후, 관련 코드를 직접 확인하고 수정할 수 있음.



**클릭!**



**클릭!**

# Jenkins Plugin : N’SIQ CppStyle

## 소개

C/C++은 전처리(preprocess)라는 2단계 컴파일을 거치고 C/C++ 언어 자체도 컴파일러마다 조금씩 다르므로 Java에 비하면 파싱이 매우 어렵고 따라서 사용 파싱 도구도 자바에 비해 좀더 비싼 편이다 . N’SIQ CppStyle은 NHN이 개발한 휴리스틱(heuristic)기반 C/C++ 코딩 스타일 점검 도구다. 정식 C/C++ 파서를 사용하여 코딩 스타일을 점검하는 C++ Test, QAC 등의 상용 코딩 스타일 점검 도구와 비교하면 정확도나 검출 범위에 차이가 있다. 따라서 MISRA C와 같이 엄격한 코딩 컨벤션을 적용할 때는 상용 도구를 적용할지 고려해야겠지만 매우 엄격하게 적용할 경우가 아니라면 N’SIQ CppStyle만으로도 상당수의 코딩 컨벤션 위반 사항을 검출할 수 있다.

N’SIQ CppStyle 0.2.0.0 이상은 Java의 CheckStyle과 동일한 XML 출력 기능을 가지고 있다. 이 기능을 사용하면 Jenkins의 CheckStyle 플러그인을 사용하여 Jenkins 상에서 결과를 볼 수 있다.

## 설치

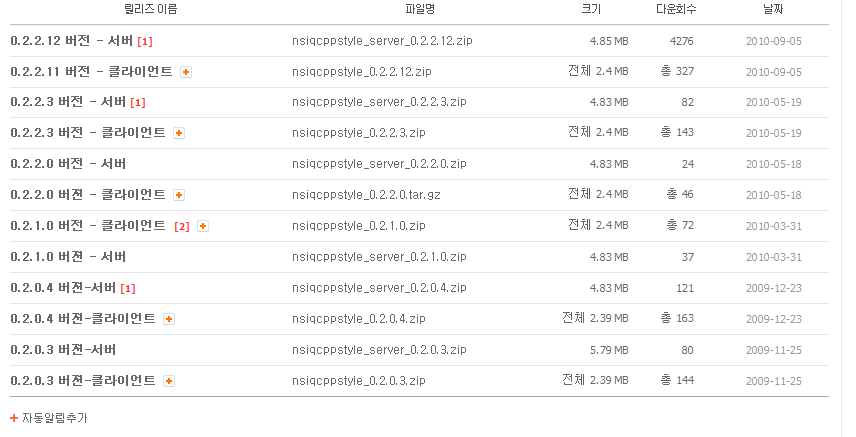
### 공통 설치

1. N’SIQ CppStyle 다운로드

아래의 사이트로 들어가서 nsiqcppstyle 서버와 클라이언트 최신 버전 두개를 모두 다운받는다.

<http://dev.naver.com/projects/nsiqcppstyle/download>

**\*주의** tar.gz형식의 파일은 아래 표시한 + 아이콘을 눌러야만 표시됨니다.

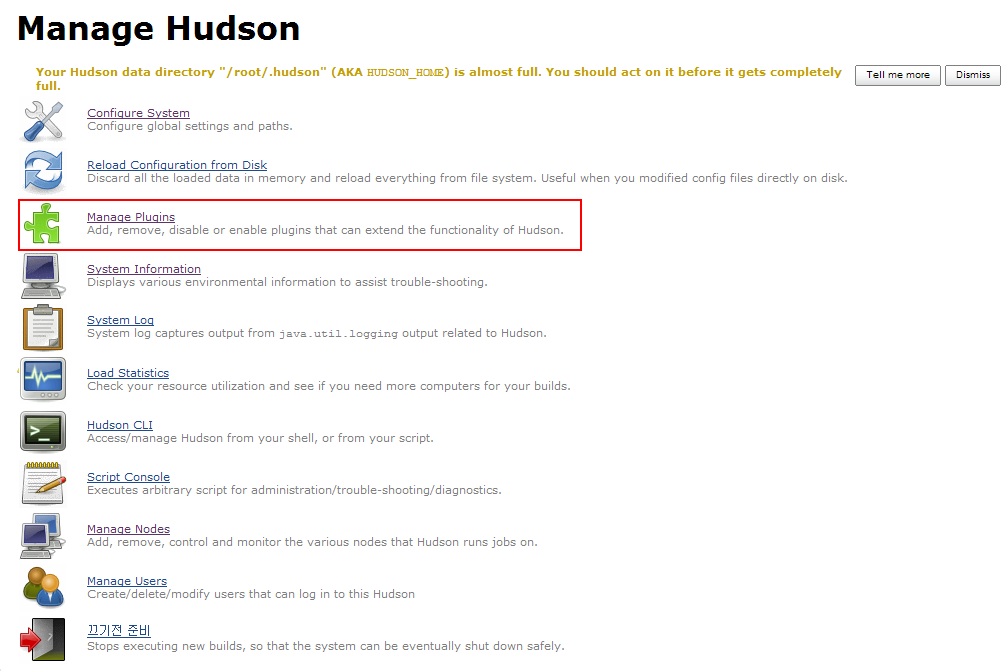




\*GUI가 설치되어 있지 않은 Linux에서는 아래와 같이 wget을 사용해서 다운로드 가능합니다.

|  |
| --- |
| $wget http://dev.naver.com/frs/download.php/1860/nsiqcppstyle\_0.2.2.12.tar.gz |

1. 다음으로 Jenkins에 CheckStyle Plugin을 설치해야한다.



Jenkins 웹 페이지에서 Jenkins 관리의 Manage Jenkins으로 들어가보면 다양한 플러그인들을 설치 할 수 있는 Manage Plugins가 보인다. 이를 클릭하면 Jenkins과 연동할 수 있는 다양한 종류의 플러그인들이 나타나는데 그 중에서 CheckStyle Plug-in을 설치해준다.

checkstyle plugin

이 플러그인으로 N’SIQ CppStyle의 결과를 볼 수있다.

## 설정

### Filefilter.txt 설정

47개의 룰을 filefilter.txt로 작성하여 nsiquniittest 폴더 폴더 안에 저장해둔다.

~ RULE\_10\_1\_A\_do\_not\_use\_bufferoverflow\_risky\_function\_for\_unix

~ RULE\_10\_1\_B\_do\_not\_use\_bufferoverflow\_risky\_function\_for\_windows

~ RULE\_3\_1\_A\_do\_not\_start\_filename\_with\_underbar

~ RULE\_3\_2\_B\_do\_not\_use\_same\_filename\_more\_than\_once

~ RULE\_3\_2\_CD\_do\_not\_use\_special\_characters\_in\_filename

~ RULE\_3\_2\_F\_use\_representitive\_classname\_for\_cpp\_filename

~ RULE\_3\_2\_H\_do\_not\_use\_underbars\_for\_cpp\_filename

~ RULE\_3\_2\_H\_do\_not\_use\_uppercase\_for\_c\_filename

~ RULE\_3\_3\_A\_start\_function\_name\_with\_is\_or\_has\_when\_return\_bool

~ RULE\_3\_3\_A\_start\_function\_name\_with\_lowercase\_unix

~ RULE\_3\_3\_A\_start\_function\_name\_with\_upperrcase\_windows

~ RULE\_3\_3\_B\_start\_private\_function\_name\_with\_underbar

~ RULE\_4\_1\_A\_A\_use\_tab\_for\_indentation

~ RULE\_4\_1\_A\_B\_use\_space\_for\_indentation

~ RULE\_4\_1\_B\_indent\_each\_enum\_item\_in\_enum\_block

~ RULE\_4\_1\_B\_locate\_each\_enum\_item\_in\_seperate\_line

~ RULE\_4\_1\_C\_align\_long\_function\_parameter\_list

~ RULE\_4\_1\_E\_align\_conditions

~ RULE\_4\_2\_A\_A\_space\_around\_operator

~ RULE\_4\_2\_A\_B\_space\_around\_word

~ RULE\_4\_4\_A\_do\_not\_write\_over\_120\_columns\_per\_line

~ RULE\_4\_5\_A\_brace\_for\_namespace\_should\_be\_located\_in\_seperate\_line

~ RULE\_4\_5\_A\_braces\_for\_function\_definition\_should\_be\_located\_in\_seperate\_line

~ RULE\_4\_5\_A\_braces\_for\_type\_definition\_should\_be\_located\_in\_seperate\_line

~ RULE\_4\_5\_A\_braces\_inside\_of\_function\_should\_be\_located\_in\_end\_of\_line

~ RULE\_4\_5\_A\_indent\_blocks\_inside\_of\_function

~ RULE\_4\_5\_A\_matching\_braces\_inside\_of\_function\_should\_be\_located\_same\_column

~ RULE\_4\_5\_B\_use\_braces\_even\_for\_one\_statement

~ RULE\_5\_2\_C\_provide\_doxygen\_class\_comment\_on\_class\_def

~ RULE\_5\_2\_C\_provide\_doxygen\_namespace\_comment\_on\_namespace\_def

~ RULE\_5\_2\_C\_provide\_doxygen\_struct\_comment\_on\_struct\_def

~ RULE\_5\_3\_A\_provide\_doxygen\_function\_comment\_on\_function\_in\_header

~ RULE\_5\_3\_A\_provide\_doxygen\_function\_comment\_on\_function\_in\_impl

~ RULE\_6\_1\_A\_do\_not\_omit\_function\_parameter\_names

~ RULE\_6\_1\_E\_do\_not\_use\_more\_than\_5\_paramters\_in\_function

~ RULE\_6\_1\_G\_write\_less\_than\_200\_lines\_for\_function

~ RULE\_6\_2\_A\_do\_not\_use\_system\_dependent\_type

~ RULE\_6\_4\_B\_initialize\_first\_item\_of\_enum

~ RULE\_6\_5\_B\_do\_not\_use\_lowercase\_for\_macro\_constants

~ RULE\_6\_5\_B\_do\_not\_use\_macro\_for\_constants

~ RULE\_7\_1\_B\_A\_do\_not\_use\_double\_assignment

~ RULE\_7\_1\_C\_do\_not\_use\_question\_keyword

~ RULE\_7\_2\_B\_do\_not\_use\_goto\_statement

~ RULE\_8\_1\_A\_provide\_file\_info\_comment

~ RULE\_9\_1\_A\_do\_not\_use\_hardcorded\_include\_path

~ RULE\_9\_2\_D\_use\_reentrant\_function

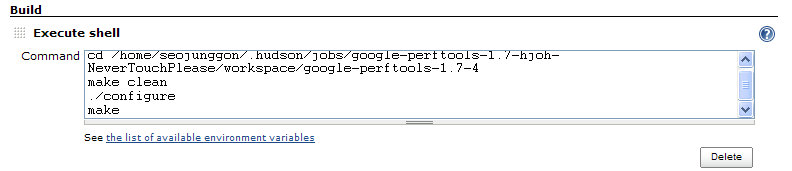
~ RULE\_A\_3\_avoid\_too\_deep\_blocks

### Linux 용

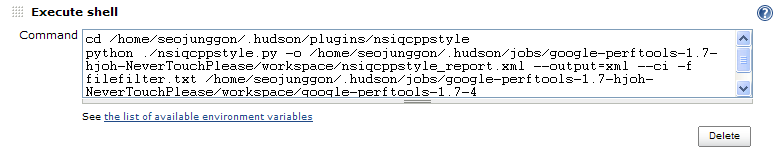
1. 개별 프로젝트 설정 화면으로 들어간다.

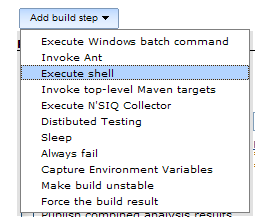


프로젝트 이름을 google-perftools-1.7.hjoh-NeverTouchPlease 로 설정해 주었다. 이는 뒤에 나올 빌드 스크립트 작성시 경로 설정 할 때 프로젝트 이름을 정확하게 지정해주어야 하므로 유의하도록 한다.



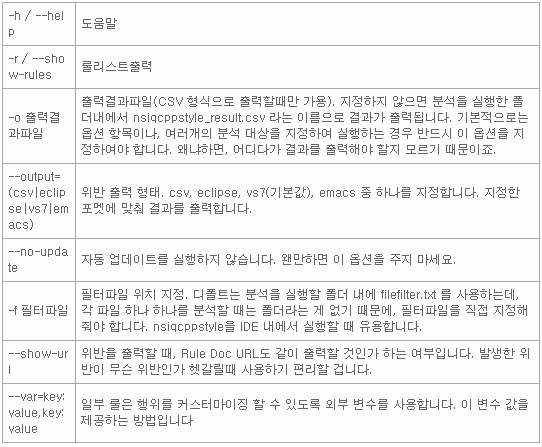
1. 빌드가 되도록 쉘 스크립트를 작성해준다.





|  |
| --- |
| Python ./nsiqcppstyle.py –o /home/seojunggon/.hudson/jobs/프로젝트명/workspace  /nsiqcppstyle\_report.xml –-output=xml –-ci –f filefilter.txt /home/seojunggon/ .hudson/jobs  /프로젝트명/workspace/ |

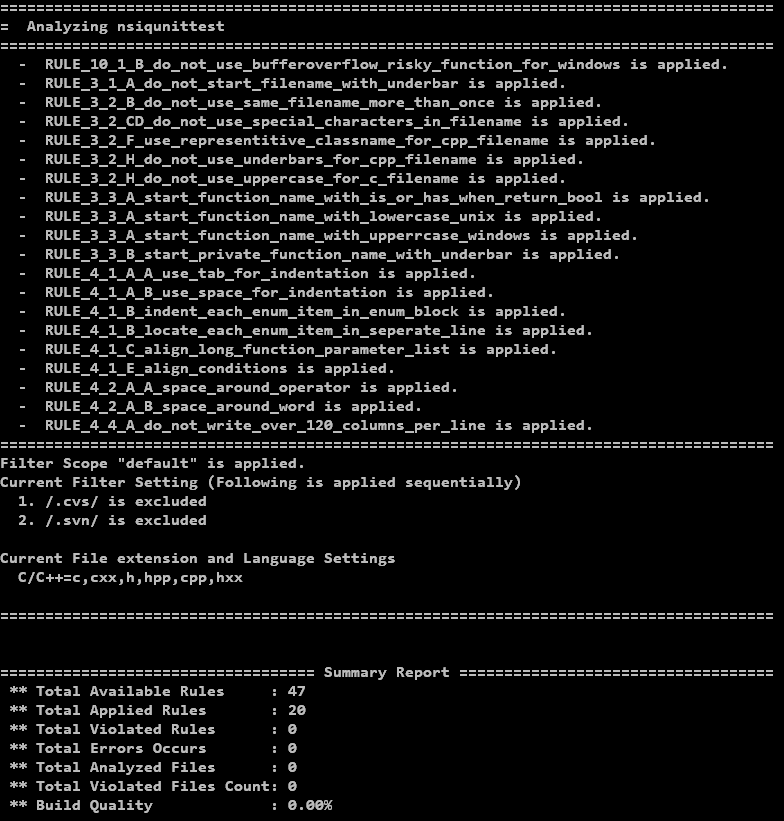
1. Script option



1. Nsiqcppstyle test방법

|  |
| --- |
| $ python./nsiqcppstyle\_exe.py./nsiqunittest |

리눅스 서버상에서 위와 같은 명령어를 입력하여 정상적으로 실행이 되는 것을 확인해보았다.



Summary Report로 결과 화면이 출력되는 것을 확인할 수 있다.

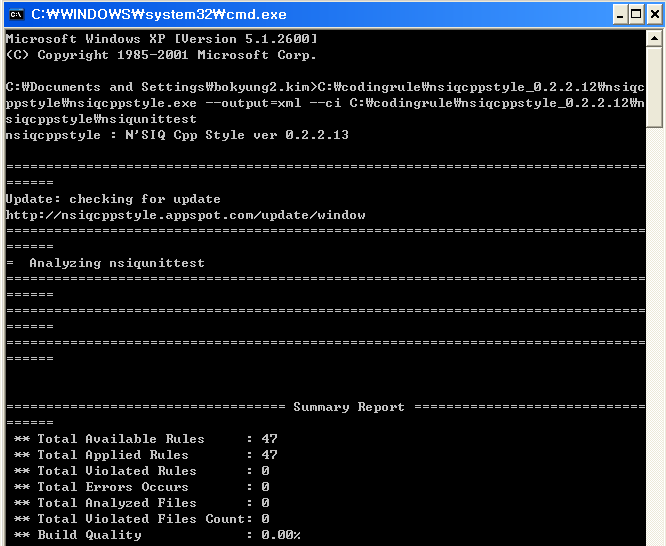
### Window용

1. Cmd창을 열어 다음과 같이 입력한다.

cㅡㅇ 창 명령어

|  |
| --- |
| C:\codingrule\nsiqcppstyle\_0.2.2.12\nsiqcppstyle\nsiqcppstyle.exe --output=xml --ci  C:\codingrule\nsiqcppstyle\_0.2.2.12\nsiqcppstyle\nsiqunittest |

다음과 같은 결과 화면을 얻을 수 있다.

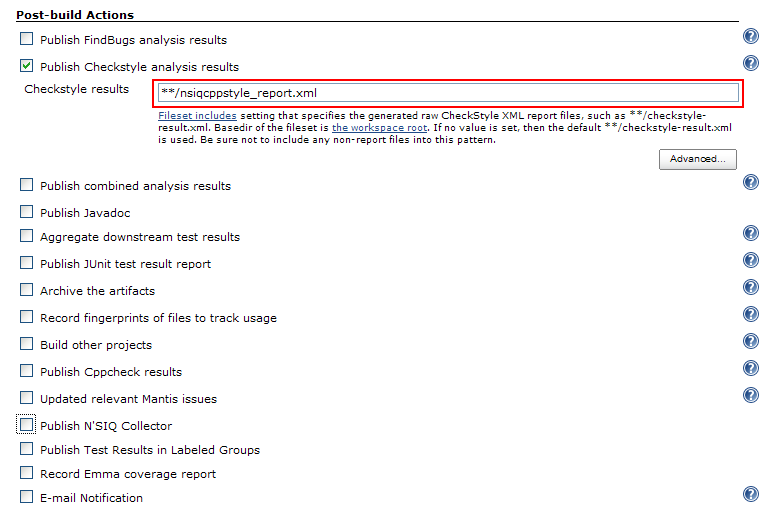


### 3.2.3 N’SIQ CppStyle 리포팅 설정

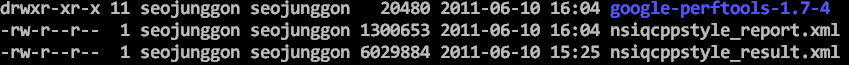
Jenkins 실행시 root 계정이 아닌 사용자 계정으로 login 해야 /home/사용자계정/.jenkins이 생성되고, 사용자 권한으로 작업을 진행할 수 있다.

분석할 파일을 ~/.jenkins/jobs/프로젝트명/workspace 경로 아래에 둔다.

Post-build Actions 에서 Publish CheckStyle anlaysis results를 선택한 뒤 Checkstyle results에 \*\*/nsiqcppstyle\_report.xml을 기재해 준다.



이때 Publsih Cppcheck results와 혼돈하지 않도록 주의한다. 반드시 Publish Checkstyle analysis results를 체크하도록 한다. N’SIQCppStyle0.2.0.0 이상은 Java의 Checkstyle과 동일한 XML Output을 출력하는 기능을 가지고 있다. 따라서 Jenkins의 CheckStyle 플러그인을 사용하여 Jenkins상에서 결과를 볼 수 있게 해준다.

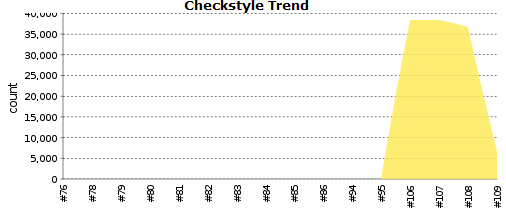


모든 설정을 완료한 뒤 저장을 하면 위와같이 nsiqcppstyle\_result.xml 파일이 생성된다. 이것을 vi 편집기를 이용하여 열어보면 아래와 같은 xml 파일이 생성된 것을 확인할 수 있다.

****

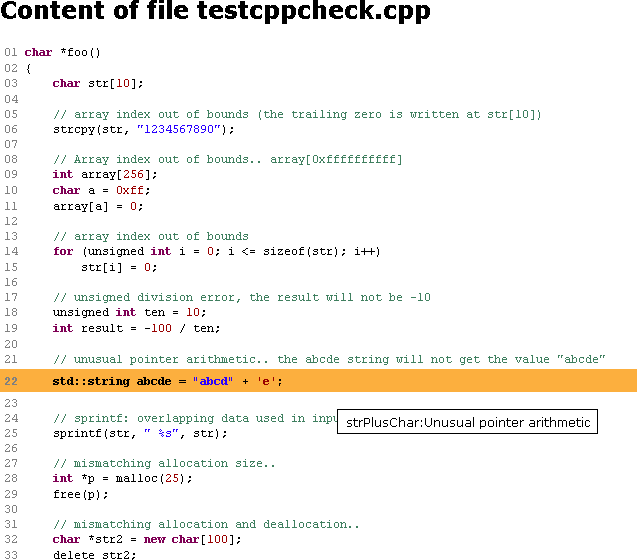
## 실행 결과

아래와 같이 규칙 위반 검출 개수에 따른 그래프 변화를 볼 수 있다.



**세로축 : 규칙 위반 검출 개수**

**가로축 : 빌드 번호**



코드에서 규칙 위반을 검출한 위치가 주황색으로 표시되며 마우스 포인터를 올리면 위반 항목에 대한 상세 정보를 확인할 수 있다. 안내에 따라 코드를 수정하고 커밋하면 CI 서버는 변경 사항을 확인하고 다시 빌드한다.

# Jenkins Plugin 설치 : ~~Findbugs~~

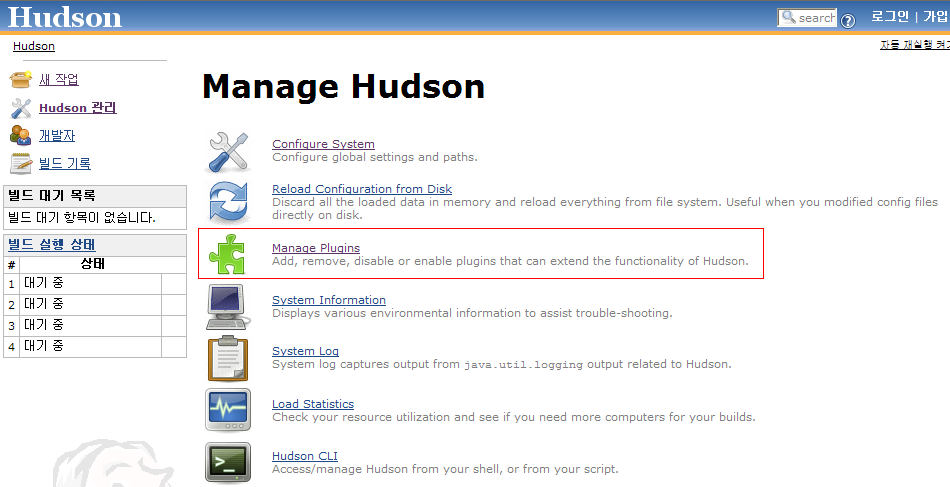
## 소개

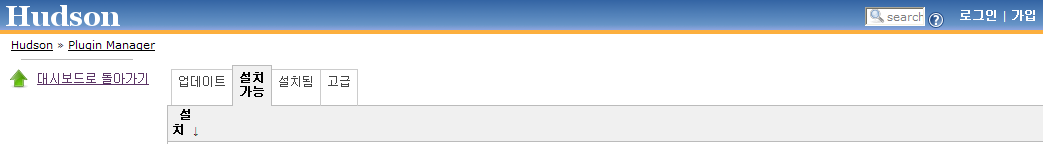
FindBugs는 자바 코드 내에서 버그들을 찾아내는 Static Analysis를 사용하는 프로그램이다. Findbugs는 JRE(or JDK)1.5.0 or 그 이상의 버전을 필요로 한다. Java의 어느 버전을 위해 컴파일되는 프로그램들을 분석 할 수 있다. **[출처]** [Maven 사용 관련 정리](http://blog.naver.com/chidoo74/10098880659)|**작성자** [민혁아빠](http://blog.naver.com/chidoo74)

## 설치

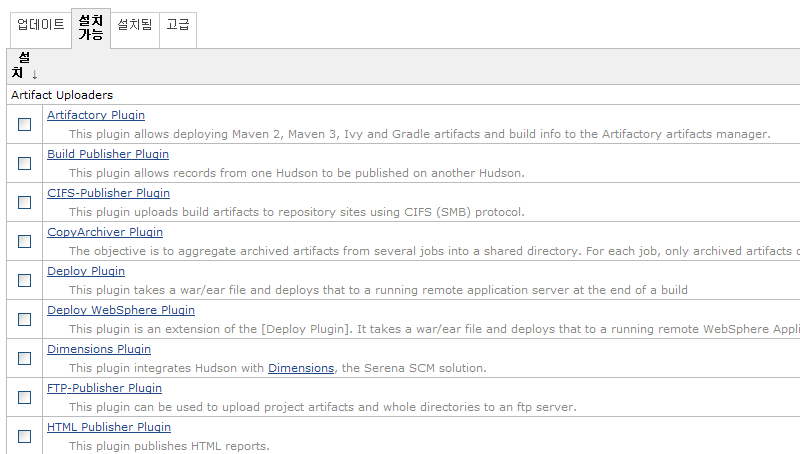
### 플러그인 설치

1. 왼쪽 상단에 있는 Jenkins 관리 메뉴 -> Manage Plugins를 선택한다.





1. 이 메뉴를 클릭하면 설치 가능한 플러그인들이 나타난다. 여기서 Findbugs Plugin과 이를 위한 Static Analysis Utilities를 체크한 뒤 아래에 있는 설치 버튼을 누른다.



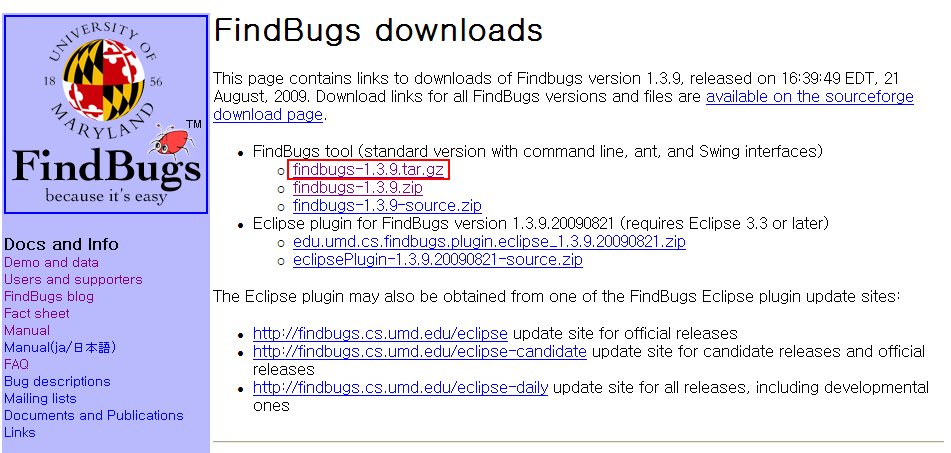
아래의 화면을 보면 Findbugs Plugin과 Static Analysis Utilities가 설치된 것을 확인 할 수 있다.

이로써 Jenkins 웹 사이트 상에서 Findbugs 플러그인 설치를 완료했다.



### Findbugs 설치

1. 먼저 <http://findbugs.sourceforge.net/downloads.html> 이 곳으로 접속한다.
2. 좌측 Downloads를 클릭한 뒤 Findbugs-1.3.9.tar.gz를 다운로드한다.



1. 먼저 처음 시도한 것과같이 <http://findbugs.sourceforge.net/> 사이트에 접속하여 findbugs-1.3.9.tar.gz를 다운로드 한다.

그 다음은 FileZilla라는 프로그램을 이용하여 윈도우상에 다운로드 되어있는 findbugs-1.3.9.tar.gz를 Ubuntu 서버로 복사한다.

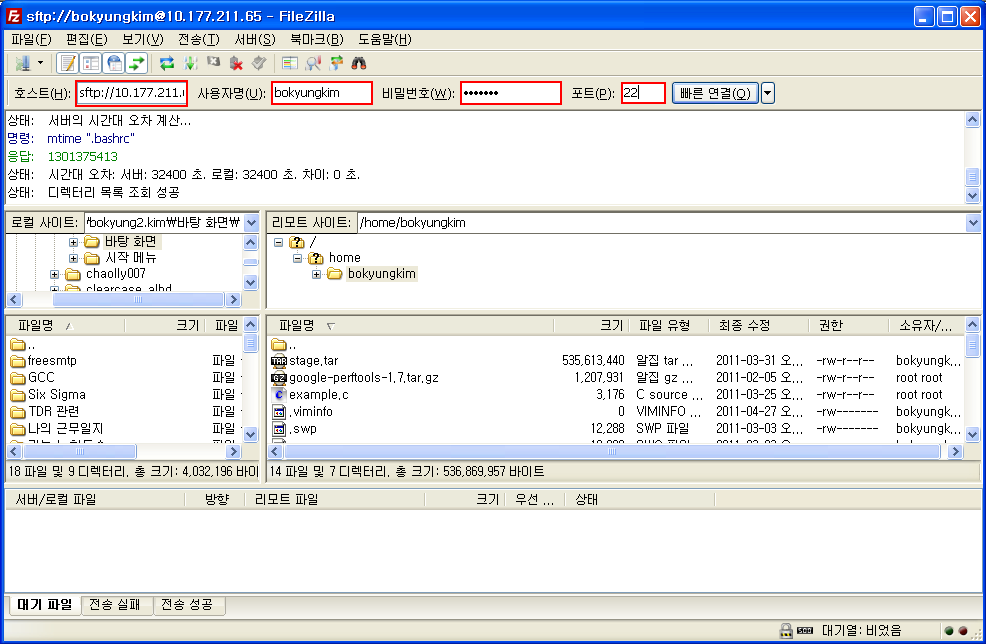
[참고**]** FileZilla는 FileZilla Client와 FileZilla Server로 구성된 **FTP** 소프트웨어이다. FTP, SFTP와 FTPS를 지원한다.

1. <http://filezilla-project.org/>로 접속하여 FileZilla Client를 다운로드 한다.



설치가 끝나면 다음과 같은 화면을 볼 수 있게 될 것이다.

1. 여기에 호스트, 사용자명, 비밀번호, 포트를 각자 자신의 PC 환경에 따라 입력을 해준 뒤 빠른 연결을 누른다.



1. 이렇게 윈도우와 Ubuntu를 연결시키고나면 간단하게 마우스로 드래그하여 다운로드한 findbugs-1.3.9.tar.gz를 리눅스 서버상의 /home/**자신의 디렉토리** 아래로 복사한다.



위의 화면에서 findbugs-1.3.9.tar.gz가 생긴 것을 확인할 수 있다.

1. 명령어를 사용하여 압축을 풀어준다. 이렇게 하면 Jenkins과 플러그인의 연동을 위한 Linux 서버상의 준비는 완료된다. 위의 화면에서 findbugs-1.3.9.tar.gz가 생긴 것을 확인할 수 있다.

|  |
| --- |
| $ sudo tar xvfz findbugs-1.3.9.tar.gz |

## 설정

### 설정build.xml 설정

여기서 이 프로젝트를 빌드하는 도구는 ant 이므로 빌드에 관하여 설정하는 build.xml 파일을 프로젝트에 맞게 수정한다.

<?xml version="1.0" encoding="UTF-8"?>

<project name="ROOT" default="" basedir=".">

<description>

simple example build file

</description>

<!-- set global properties for this build -->

<property name="src" location="src"/>

<property name="build" location="build"/>

<property name="dist" location="dist"/>

<!-- FindBugs configuration/directories -->

<property name="findbugs.home" value="/home/.hudson/jobs/ROOT/workspace/ROOT/findbugs/"/>

<property name="report.home" value="/home/.hudson/jobs/ROOT/workspace/ROOT/findbugs/"/>

<property name="publish.home" value="/home/.hudson/jobs/ROOT/workspace/ROOT/"/>

<path id="findbugs.classpath">

<pathelement location="${findbugs.home}/lib/findbugs-ant.jar" />

<pathelement location="${findbugs.home}/lib/findbugs.jar" />

</path>

<target name="init">

<!-- Create the time stamp -->

<tstamp/>

<!-- Create the build directory structure used by compile -->

<mkdir dir="${build}"/>

</target>

<target name="build-main">

<javac srcdir="/root/.hudson/jobs/ROOT/workspace/ROOT" includeantruntime="false" >

<classpath>

<pathelement location="/root/.hudson/jobs/ROOT/workspace/ROOT"/>

</classpath>

<classpath path="${workspace.hello}">

</classpath>

</javac>

</target>

<path id="findbugs\_jar">

<pathelement path="/root/.hudson/jobs/ROOT/workspace/ROOT/findbugs/lib/findbugs-ant.jar"/>

</path>

<taskdef name="findbugs" classpathref="findbugs\_jar" classname="edu.umd.cs.findbugs.anttask.FindBugsTask" classpath="/root/.hudson/jobs/ROOT/workspace/ROOT/findbugs/lib"/>

<mkdir dir="reports"/>

<target name="findbugs">

<findbugs home="/root/.hudson/jobs/ROOT/workspace/ROOT/findbugs/lib" output="xml:withMessages"

outputFile="findbugs.xml" jvmargs="-Xmx1024m">

<sourcePath path="${basedir}/src"/>

<class location="/usr/share/ant/lib/bcel.jar"/>

<auxClasspath path="${catalina.home}/common/lib/servlet-api.jar"/>

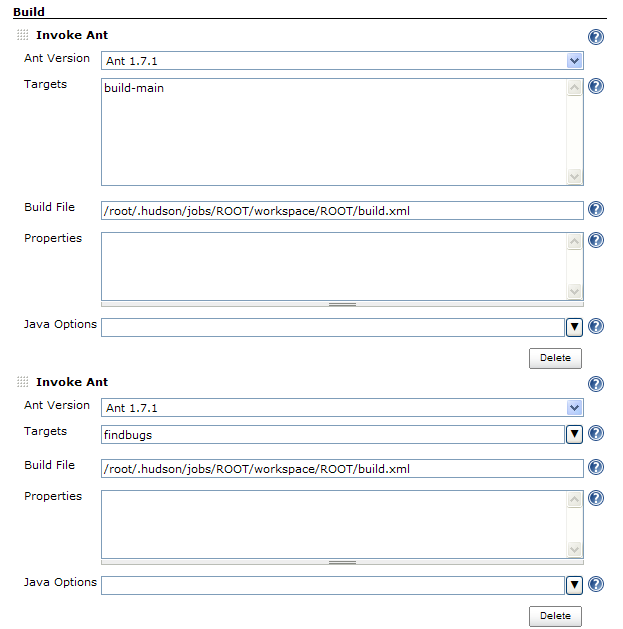
</findbugs>

</target>

</project>

### Jenkins설정

Findbugs 사용을 위해 Java 코드를 기반으로 한 프로젝트의 빌드 환경을 설정해 준다.

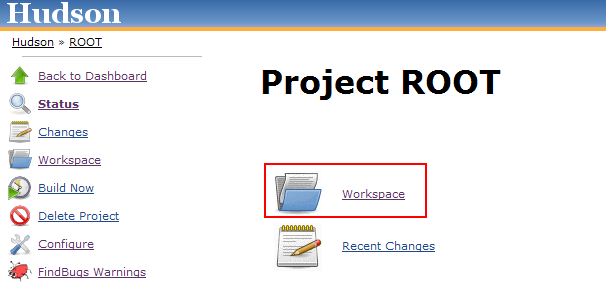


다음으로는 Post-build Actions 부분에 대한 설정이다. Findbugs results를 위해 \*\*/findbugs.xml과 같은 형식으로 입력해준다.



Java 예제 소스를 이용하여 직접 허드슨에서 Findbugs 결과를 얻어내 보자. 미리 코딩해 놓은 간단한 Java 소스 “ROOT”를 빌드하고 Findbugs 플러그인과 연동하여 결과를 얻어내 보자.

리눅스 서버상에서 /home/.jenkins/jobs/ROOT/workspace 디렉터리 밑으로 ROOT 소스 파일을 복사해 놓는다. 그러면 아래와 같이 Jenkins 웹 사이트에 ROOT 라는 Workspace 디렉터리 아래에 관련된 파일들이 생성되는 것을 확인할 수 있다.





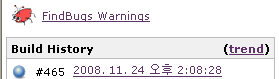
## 실행 결과

1. Findbugs에 관한 설정을 마친 뒤 빌드를 하면 다음과 같은 결과를 얻을 수 있다. 빌드가 성공하면 xml

형식으로 findbugs 플러그인 기능을 제공한다.

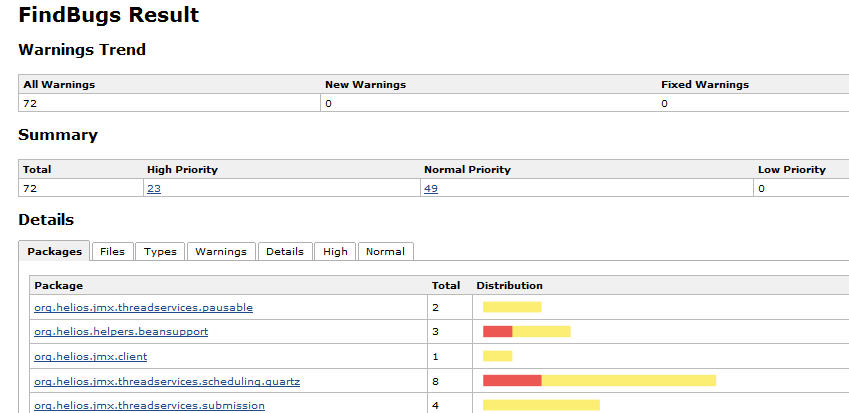


1. 버그의 자세한 리포트는 프로젝트 왼쪽 메뉴의 FindBugs Warnings 를 통해서 확인가능하다.



1. warning이 존재할 경우 건수를 클릭하면 해당하는 클래스들이 나오고, 클래스를 선택하면 소스에서

warning을 발생시키는 부분까지 보여준다.



# Jenkins Plugin : Redmine

## 소개

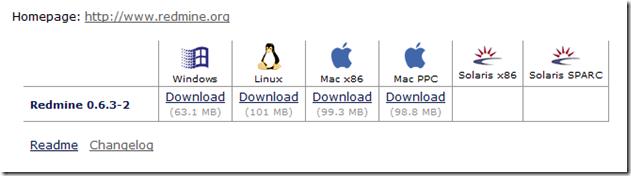
**Redmine**은 [오픈소스](http://ko.wikipedia.org/wiki/%EC%98%A4%ED%94%88_%EC%86%8C%EC%8A%A4) 프로그램으로 웹기반의 [프로젝트 관리](http://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8_%EA%B4%80%EB%A6%AC) 와 [버그](http://ko.wikipedia.org/wiki/%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EB%B2%84%EA%B7%B8) 추적 기능을 제공하는 도구이다. 화면기반의 프로젝트 관리에 도움이 되도록 달력과 [간트 차트](http://ko.wikipedia.org/wiki/%EA%B0%84%ED%8A%B8_%EC%B0%A8%ED%8A%B8)를 제공하고 일정관리 기능을 제공한다. Redmine은 통합된 프로젝트관리 기능과 [이슈추적](http://ko.wikipedia.org/w/index.php?title=%EC%9D%B4%EC%8A%88%EC%B6%94%EC%A0%81&action=edit&redlink=1), 여러가지 [형상 관리](http://ko.wikipedia.org/wiki/%ED%98%95%EC%83%81_%EA%B4%80%EB%A6%AC) 기능을 제공한다. Redmine을 설치하기에 앞서 Ruby, Rail, MySQL등 사전에 반드시 설치해야 한다.

## 설치

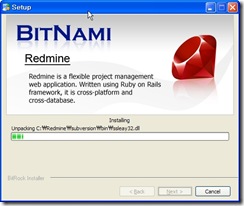
### Window 용

Window에서 수동으로 설치할 경우, 기반 프로그램들부터 하나씩 설치해야 하지만, 자동설치를 할 경우 한가지 파일만 다운받는 것으로 모든 설치가 끝난다. 이곳에서는 자동으로 설치하는 방법을 제공한다.

1. 자동설치 폴더는 필수프로그램(Ruby, Rails 등)을 모두 포함하고 있기 때문에, Window에 설치되어있다면 새로 설치할 프로그램과 포트번호를 다르게 해주어야 한다는 것을 명심하자.
2. <http://bitnami.org/stack/redmine> 에서 자신의 OS환경에 맞는 파일을 다운로드 한다.



1. 다운받은 파일의 압축을 풀어준다. 그리고 설치파일을 실행한다.

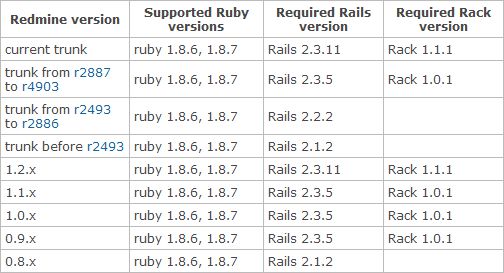


1. 관리에 필요한 모든 계정 정보를 영문으로 입력한다. 이 정보들은 Redmine 의 admin으로 로그하는데 이용된다. 그리고 웹포트 또는 MySQL포트번호를 따로 기록해둔다.
2. 설치가 끝나면, <http://localhost:웹포트번호/redmine> 을 열어본다. 웹포트번호는 4번과정에서 지정해준 번호이다.
3. 아래와 같은 화면이 나온다면, 정상적으로 설치된 것이다.

### Linux용

Ubuntu환경에서 Redmine을 설치하는 방법을 제공한다. Ubuntu에서 수동으로 설치한다면, wget명령어 등으로 기반 프로그램들부터 하나씩 설치해야 한다. 하지만 Ubuntu에서는 apt-get명령어로 Redmine을 쉽게 설치가 가능하다. 설치하기에 앞서 아래의 그림을 보고 서로 호환되는 버전인지 확인하도록 하자.



1. Ruby 설치

|  |
| --- |
| $ sudo apt-get install rubygems  $ sudo gem install rubygems-update |

1. Rails 설치

|  |
| --- |
| $ sudo gem install rails –v=2.3.11 |

1. Redmine 설치

|  |
| --- |
| $ svn co http://redmine.rubyforge.org/svn/branches/1.0-stable redmine-1.0  [출처] [Ubuntu 10.10] Redmine 설치|작성자 얼룩푸우 |

1. MySQL 설치

|  |
| --- |
| $ sudo apt-get install mysql-server mysql-admin mysql-query-browser |

1. Mongrel 설치

|  |
| --- |
| $ sudo apt-get install mongrel |

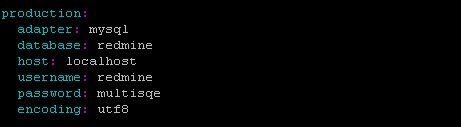
[참고]: <http://blog.naver.com/PostView.nhn?blogId=budget74&logNo=90103735211>

## 설정

1. MySQL 설정

|  |
| --- |
| $ mysql –u root -p  mysql>create database redmine character set utf8;  Query OK, 1 row affected (0.00 sec)  mysql> create user 'redmine'@'localhost' identified by '패스워드';  Query OK, 0 rows affected (0.00 sec)  mysql> grant all privileges on redmine.\* to 'redmine'@'localhost';  Query OK, 0 rows affected (0.00 sec)  mysql>quit  [출처] [Ubuntu 10.10] Redmine 설치|작성자 얼룩푸우 |

|  |
| --- |
| $ cp redmine-1.0/config/database.yml.example redmine-1.0/config/database.yml  $ vi redmine-1.0/config/database.yml |



1. rake generate\_session\_store

|  |
| --- |
| $ sudo rake generate\_session\_store |

* 1. 만약 아래와 같은 에러 메시지가 나온다면,

|  |
| --- |
| 'rake' 프로그램은 현재 설치되어 있지 않습니다. 다음을 입력하여 이를 설치할 수 있습니다:  sudo apt-get install rake |

아래의 명령어를 실행해준다.

|  |
| --- |
| $ sudo apt-get install rake |

* 1. 만약 아래와 같은 에러 메시지가 나온다면,

|  |
| --- |
| (in /var/lib/gems/1.8/gems/rake-0.9.2)  rake aborted!  uninitialized constant Rake::DSL  /var/lib/gems/1.8/gems/rake-0.9.2/Rakefile:19  (See full trace by running task with --trace) |

0.9.2버전의 rake를 삭제하고 0.8.7버전의 rake를 설치한다. rake는 /var/lib/gems/1.8/gems에 위치해 있을 것이다.

|  |
| --- |
| $ sudo rm –r rake-0.9.2  $ sudo apt-get install rake –v=0.8.7 |

1. redmine이 설치되어 있는 곳으로 이동한 뒤, 아래의 명령어를 실행한다.

|  |
| --- |
| $ sudo rake generate\_session\_store |

* 1. 만약 아래와 같은 에러 메시지가 나온다면,

|  |
| --- |
| (in /var/www/redmine-1.0)  Missing the i18n 0.4.2 gem. Please `gem install -v=0.4.2 i18n` |

아래의 명령어를 실행해 준다.

|  |
| --- |
| $ sudo gem install -v=0.4.2 i18n |

1. RAILS\_ENV 설정1

|  |
| --- |
| $ sudo rake generate\_session\_store |

* 1. 만약 아래와 같은 에러 메시지가 나온다면,

|  |
| --- |
| (in /var/www/redmine-1.0)  !!! The bundled mysql.rb driver has been removed from Rails 2.2. Please install the mysql gem and try again: gem install mysql.  rake aborted!  no such file to load -- mysql  (See full trace by running task with --trace) |

아래의 명령어를 실행해준다.

|  |
| --- |
| $ sudo apt-get install libmysql-ruby1.8 |

1. RAILS\_EVN 설정2

|  |
| --- |
| $ sudo RAILS\_ENV=production rake redmine:load\_default\_data  (in /var/www/redmine-1.0)  Select language: bg, bs, ca, cs, da, de, el, en, en-GB, es, eu, fi, fr, gl, he, hr, hu, id, it, ja, ko, lt, lv, mk, mn, nl, no, pl, pt, pt-BR, ro, ru, sk, sl, sr, sr-YU, sv, th, tr, uk, vi, zh, zh-TW [en] ko  ====================================  Default configuration data loaded.  [출처] [Ubuntu 10.10] Redmine 설치|작성자 얼룩푸우 |

1. Additional\_environment.rb 설정

|  |
| --- |
| $ cp config/additional\_environment.rb.example config/additional\_environment.rb  $ vi additional\_environment.rb  #Logger.new(PATH,NUM\_FILES\_TO\_ROTATE,FILE\_SIZE)  config.logger = Logger.new(config.log\_path, 2, 1000000)  config.logger.level = Logger::INFO    [출처] [Ubuntu 10.10] Redmine 설치|작성자 얼룩푸우 |

1. 구동하기 위해, redmine이 설치된 경로로 이동한뒤 아래의 명령어를 이용해 mongrel서버로 실행시킨다.

$ cd /var/rails/redmine-1.0

|  |
| --- |
| $ sudo mongrel\_rails start --environment=production -d |

이제 <http://ip주소:3000>으로 접속하면 redmine의 GUI를 확인할 수 있다.

[참고]: <http://blog.naver.com/PostView.nhn?blogId=budget74&logNo=90103735211>

K-1.png **레드마인에서 jenkins플러그인 설치**

1. <http://www.redmine.org/wiki/redmine/Plugin_List>에서 원하는 플러그인을 wget명령어로 다운로드 한 뒤 redmine폴더 내의 plugin 디렉토리 아래에옮긴다.

|  |
| --- |
| $ sudo wget labs.googlecode.com/files/redmine\_hudson-1.0.7.zip  $ sudo mv redmine\_hudson-1.0.7.zip ~/redmine/vender/plugins/ |

1. Redmine이 설치된 디렉토리 아래에서 다음명령어를 실행한다.

|  |
| --- |
| $ sudo rake db:migrate\_plugins RAILS\_ENV=production |

[참고]: <http://srue.net/92>

K-1.png **레드마인에서 Jenkins 연동**

K-1.png **레드마인에서 translate error fix**

* + - 1. **Redmine 1.2 version에서 Korean translate error fix**
      2. **Redmine 1.2 version을 사용할 수 없을시**
         1. **Redmine1.x/Vender/plugins/hudson/config/locale/en.yml 파일의 내용을 redmine/config/locale/ko.yml안으로 복사해 준후 redmine을 재부팅 시키면 된다.**

## 실행 결과

### 프로젝트 생성 및 일감등록

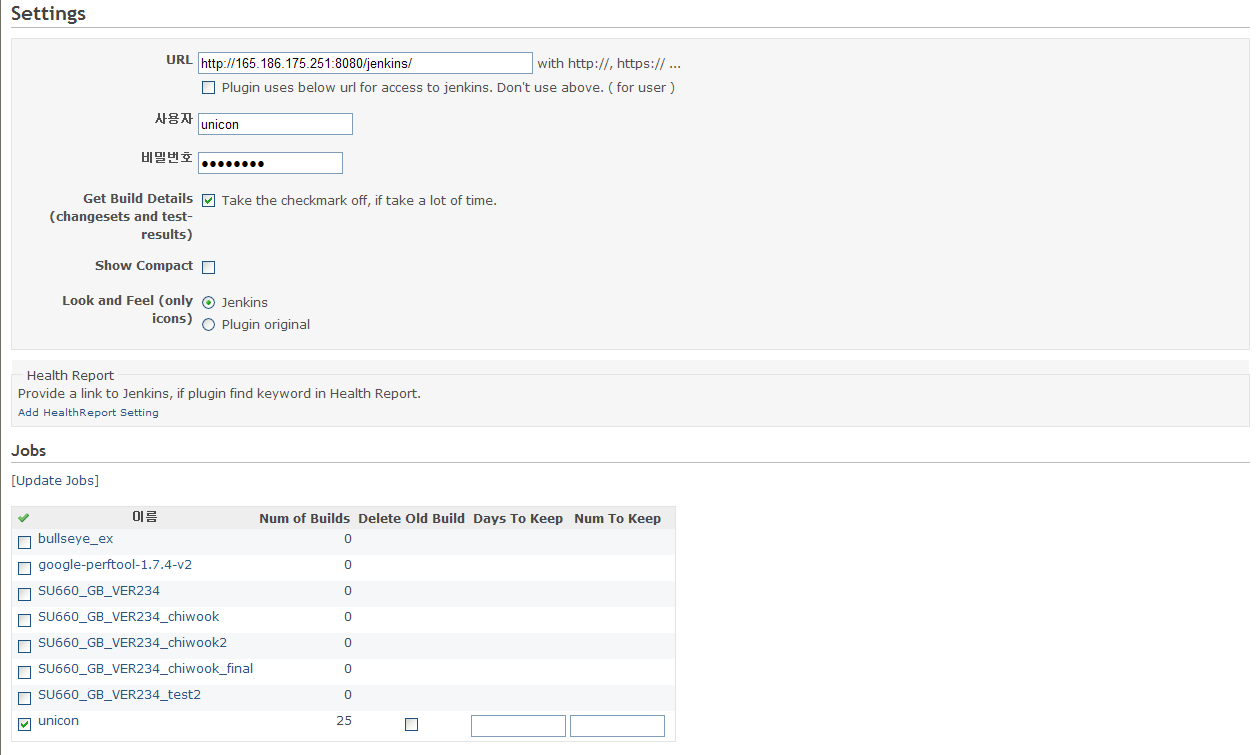
1. 프로젝트 생성
   1. 생성할 프로젝트의 이름을 기입한다.
   2. 식별자를 기입한다. 식별자는 redmine의 경로로 식별되는 것이다. http://localhost/redmine/projects/**test**/issues 와 같이 test 경로로 들어간다.
   3. 프로젝트를 공개 여부를 결정한다.
   4. 생성할 프로젝트의 일감유형을 선택한다.
   5. 필요한 모듈을 import하여 사용한다.
2. 프로젝트 관리
   1. 선택한 프로젝트의 Menu
   2. 선택한 프로젝트의 관리 Menu (관리 Menu를 보면 어떤 관리인지 쉽게 알 수 있다)
   3. 관리내용을 기입한다.
3. 프로젝트 저장소 설정
   1. 형상관리 시스템을 선택한다. 선택한 형상관리 시스템의 접속 URL을 입력하고 계정정보를 기입한 후 저장한다.
   2. 설정이 완료되었으면, 프로젝트 Menu에서 저장소를 클릭한다.
   3. 형상관리 시스템과 연계되어 다음과 같은 화면이 출력된다.
4. 프로젝트 일감 등록
   1. 일감의 구분을 선택한다. (버그, 새기능, 지원)
   2. 일감의 명칭을 입력한다.
   3. 기타 설정을 입력한다. 상태, 우선순위, 시작일, 종료일, 담당자 등을 입력하고 목표버전은 예를들어 0.5 내부오픈, 1.0 그랜드 오픈과 같이 이 기능이 속하는 버전을 말한다.
5. 프로젝트 Issue(일감) 및 일정 확인
   1. 상태가 진행중인 일감 List이다.
   2. 프로젝트 일정 관리 report View
   3. 달력으로 현재 선택된 필터로 일정을 확인한다.
   4. Gantt차트로 일정을 확인한다

### Jenkins 연동

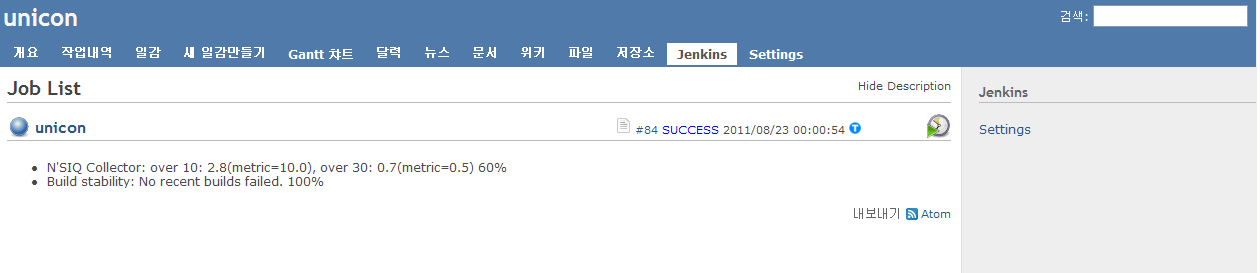
1. Jenkins tap에 Setting에 들어감니다.



1. Jenkins Setting 설정



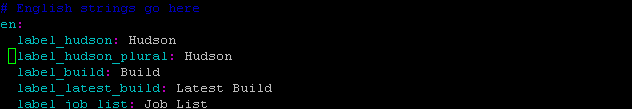
1. URL란 : Jenkins URL
2. ID,PASSWORD : Jenkins ID와 PASSWORD
3. 그 후에 Save한 다음 다시 Jenkins tap을 다시 클릭해줍니다.
4. 그럼 Jenkins에 등록되어있는 Job의 list가 위의 그림과 같이 뜸니다.
5. Job list중 불러드릴 Job을 클릭한 후 save하면 아래 그림과 같이Jenkins tap에 Job list가 불려와 진 것을 확인 할 수 있습니다.

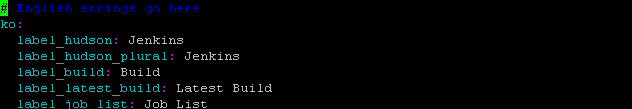


### Jenkins translate 오류 해결

1. Redmine1.2 version 에선 문제 fix 되어 있습니다.
2. Redmine 1.2 version을 사용할 수 없는 경우
   1. Redmine이 설치되어있는 폴더로 이동
   2. Vendor/plugins/redmine\_hudson/config/locales로 이동해줌니다.
   3. 아래의 명령어로 en.yml을 ko.yml로 복사해준후 아래 그림과 같이 en: 을 ko:로 바꿔줌니다.

|  |
| --- |
| $ cp en.yml ko.yml |





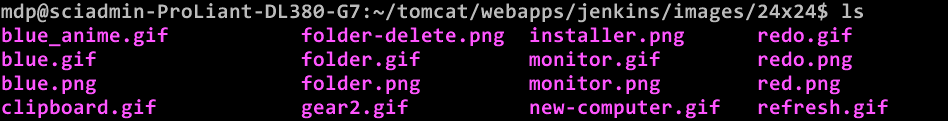
* 1. 그 후 remine을 재 실행 시켜주면 Korean translation문제는 해결됨니다.

# Jenkins Plugin : 기타

## Sidebar Links 설치

1. Jenkins 홈 > Jenkins 관리 > 플러그인 관리 > ‘설치 가능’ Tab > ‘Sidebar Links’ 설치   
   > 맨 하단의 ‘Download now and install after restart’ 버튼 클릭
2. Jenkins 서버(165.186.175.251) 접속 > id: 그룹명/pw:그룹명pw (예: id:mdp/pw:mdppw)

> ~/tomcat/webapps/jenkins/images/24x24에 Sidebar Links에 활용할 prevent-icon.gif을 복사



1. Jenkins 홈 > 프로젝트 선택 > 설정 > 하기와 같이 세팅

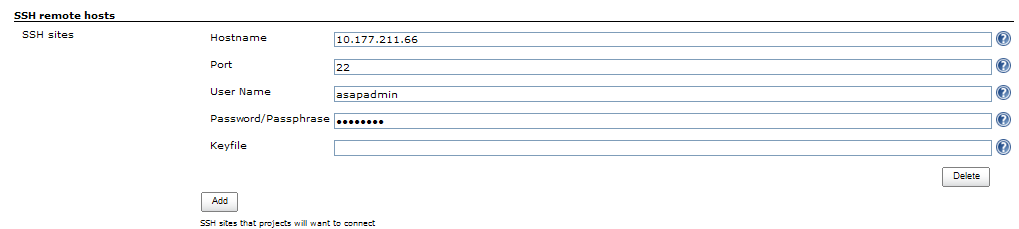
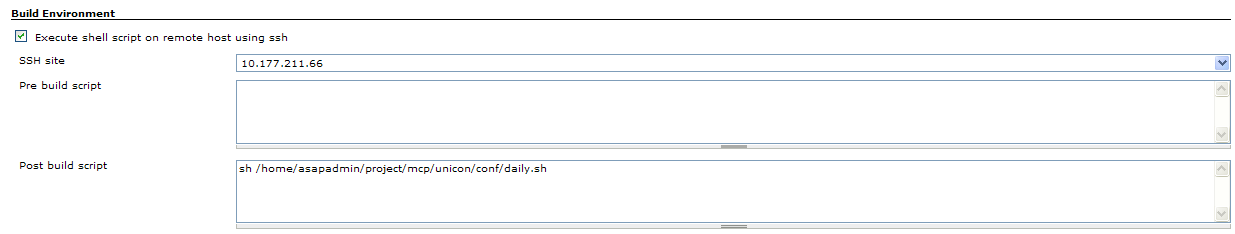


1. 하기와 같이 Prevent 아이콘을 통해 Prevent 서버(10.177.211.66)로 연결됨을 확인 가능



## Jenkins SSH plugin설치

1. Jenkins 홈 > Jenkins 관리 > 플러그인 관리 > ‘설치 가능’ Tab > ‘Jenkins SSH Plugin’ 설치   
   > 맨 하단의 ‘Download now and install after restart’ 버튼 클릭
2. Jenkins 홈 > 프로젝트 선택 > 설정 > 하기와 같이 세팅



# Appendix

## 리눅스 (Ubuntu)관련 Tip

### Ubuntu 네트워크 설정

1. 네트워크 설정을 위해 interface파일을 vi에디터로 열어준다.

|  |
| --- |
| $ sudo vi /etc/network/interface |

1. IP주소, netmask 등의 내용을 아래와 같이 추가해주도록 한다.

|  |
| --- |
| # This file describes the network interfaces available on your system  # and how to activate them. For more information, see interfaces(5).  # The loopback network interface |

auto lo

iface lo inet loopback

# The primary network interface

Auto eth0

Iface eth0 inet static

address 10.177.211.71

netmask 255.255.254.0

nameserver 156.147.135.180

broadcast 10.177.210.255

gateway 10.177.210.1

hostname ubun

domainname loge.com

1. 2번과정에서 수정한 내용을 적용하기 위해 아래의 명령어를 실행시킨다.

|  |
| --- |
| $ sudo etc/init.d/networking restart |

### Putty접속이 안될경우

리눅스에서는 ssh를 이용하여 원격접속을 가능하도록 한다. 새롭게 리눅스를 설치하였다면, ssh를 설치함으로써 putty로 원격접속이 가능하다. 이미 ssh가 설치되어있음에도 불구하고 putty접속이 안될경우 ssh를 재실행해줌으로써 문제를 해결할 수 있다.

1. 6.1.1장에서의 내용을 참고하여 네트워크 설정을 확인하고, 재설정해준다.
2. putty접속 시 사용했던 ssh(22 포트) 가 열려 있는지 아래의 명령어로 확인한다.

|  |
| --- |
| $ sudo nmap [ip 주소] |

만약 22번포트가 열려있지 않다면 아래의 명령어로 열어 준다.

|  |
| --- |
| $ sudo iptables-A INPUT –p tcp –dport 22 –j ACCEPT |

1. ssh가 설치되어있는지 확인한다.

|  |
| --- |
| $ ssh -V |

ssh가 설치되어있지 않다면, 아래의 명령어로 간단하게 ssh를 설치할 수 있다.

|  |
| --- |
| $ sudo apt-get install ssh |

1. 아래의 명령어로ssh를 점검 한다.

|  |
| --- |
| $ etc/init.d/ssh status |

만약 실행이 안되고 있다면,

|  |
| --- |
| $ etc/init.d/ssh restart |

위의 명령어로 다시 시작해준다. 만약 아래와 같은 에러 메시지가 나온다면 다시 설치해준다.

|  |
| --- |
| Could not load host key: /etc/ssh/ssh\_host\_rsa\_key  Could not load host key: /etc/ssh/ssh\_host\_dsa\_key |

|  |
| --- |
| $apt-get install ssh  $ssh-keygen –t rsa  Enter file in which to save the key (/root/.ssh/id\_rsa): /etc/ssh/ssh\_host\_rsa\_key  Enter passphrase (empty for no passphrase) : 엔터  Enter same passphrase again : 엔터  $ssh-key-getn –t dsa  Enter file in which to save the key (/root/.ssh/id\_dsa): /etc/ssh/ssh\_host\_dsa\_key  Enter passphrase (empty for no passphrase) : 엔터  Enter same passphrase again : 엔터 |

1. ssh를 다시 실행시켜 준다.

|  |
| --- |
| $ etc/init.d/ssh restart |

### MySQL의 database위치 변경 방법

1. 아래의 명령어를 수행하여, MySQL을 중지한다.

|  |
| --- |
| $ sudo service mysql stop |

1. mysql을 원하는 위치로 이동시킨다.

|  |
| --- |
| $ mv –R /var/lib/mysql /원하는경로/ |
|  |

1. 옮겨진mysql의 권한을 변경한다.

|  |
| --- |
| $ sudo chown -R mysql:mysql /mysql이 옮겨진경로/mysql |
|  |

1. /etc/mysql/my.cnf 파일을 열어서 아래와 같이 수정한다.

|  |
| --- |
| #datadir = /var/lib/mysql # 주석처리  datadir = /mysql이 옮겨진경로/mysql # 추가 |

1. /etc/apparmor.d/usr.sbin.mysqld 을 열어서 아래 항목을 수정합니다.

|  |
| --- |
| #/var/lib/mysql/ r, # 주석처리  #/var/lib/mysql/\*\* rwk, # 주석처리  / mysql이 옮겨진경로/mysql/ r, #추가  / mysql이 옮겨진경로/\*\* rwk, #추가 |

1. 용량이 0인 mysqld.sock파일을 만들고 권한을 변경해준다.

|  |
| --- |
| $ sudo touch /var/run/mysqld/mysqld.sock  $ sudo chown mysql:mysql mysqld.sock |
|  |

1. Apparmor 과 mysql을 재시작 한다.

|  |
| --- |
| $ sudo /etc/init.d/apparmor restart  $ sudo service mysql start |
|  |

[참고]: <http://floor.tistory.com/notice/159>

### 외부 서버로 파일 보내는 방법

scp 명령어를 사용하면 간단하게 파일을 보낼 수 있다. 보내고자하는 것이 파일이 아닌 디렉토리일 경우 tar나 zip으로 압축을 파일로 만들어 보내면 된다. 예를 들어 설명하기 위해 상황을 가정해보자.

**상대방IP: 10.177.211.65**

**상대방계정: hongkildong**

**보내고자 하는 파일명: sjg.tar**

**보내고자하는 위치: /home/hongkildong/source**

사용방법은 아래와 같다.

scp [파일명] [상대방계정]@[상대방IP]:[보내고자하는위치]

<예>

|  |
| --- |
| $ scp sjg.tar hongkildong@10.177.211.65:~/source |

## Maven

### Maven POM 파일의 구조

<project> element는 decriptor의 root이다. 다음 표는 가능한 child element를 모두 보여준다.

|  |  |  |
| --- | --- | --- |
| **element** | **타입** | **설명** |
| modelVersion | String | 이 POM 파일이 따르는 Project descriptor의 버전을 선언 |
| parent | Parent | 부모 project가 존재한다면, 그 위치를 나타냄. 만약 프로젝트 설정을 아무것도 하지 않는다면, 부모 프로젝트의 값(value)이 이 프로젝트의 default값이 된다. 위치는 groupID, artifact ID, version으로 주어진다. |
| groupID | String | 프로젝트를 식별하기 위한 보편적이고, unique identifie이다. 일반적으로, 자격있는 패키지 이름이 다른 프로젝트의 유사한 이름과 구분하기 위해 사용된다. |
| artifact ID | String | 이 산출물 identifier는 groupID가 주어진 특정 그룹 내에서 unique하다. Artifact란 한 프로젝트에 의해 만들어지거나 사용되는 어떤 것이다.  한 프로젝트를 위해 Maven에 의해서 만들어지는 Artifacts의 예는 다음과 같다 : JARs, 소스 및 binary 배포판, Wars |
| version | String | 이 프로젝트에 의해 만들어진 artifact의 현 버전을 가르킨다. |
| packaging | String | 이 프로젝트가 만드는 artifact의 타입을 의미한다. 예로 jar, war ear pom등이 있다. Plugins는 그들만의 packaging과 packaging 타입을 만들어낸다. 그러므로, 이 list는 어떠한 타입도 포함하고 있지 않다. Default 값은 jar이다. |
| name | String | 프로젝트의 이름이다. |
| description | String | 웹 싸이트와 같은 곳에서 프로젝트를 기술할 필요가 있을때마다, Maven에 의해서 사용되는 프로젝트의 상세화된 설명이다. 이 element는 CDATA로써 상세화되어 설명 내에 HTML tag 사용을 가능하게 하고, 일반적인 text 표현을 사용을 자제하도록 한다. 만약 생성된 웹 싸이트의 index page를 수정하고 싶다면, 이 text를 조정하는 대신, 원하는 내용을 기술할 수 있다. |
| url | String | 프로젝트 홈페이지의 url이다. |
| inceptionYear | String | 프로젝트 개시 년도를 4자리로 기술한다. Copyright 공지를 생성하거나 정보를 주기 위해 사용된다. |
| organization | Organization | 프로젝트가 속해있는 조직의 다양한 속성을 기술한다. Copyright 공지와 links위한 문서를 만들 때 사용된다. |
| licenses/  license\* | List<License> | (Many) 이 element는 프로젝트의 모든 license를 기술한다. 각 license는 license element에 의해 기술되며, license element는 추가적인 element들에 의해 기술횐다. 프로젝트에 적용되는 license(s)만을 기술해야하며, dependencies에 적용되는 license는 기술하지 않는다. 만약 여러 개의 license가 list-up되었다며느 사용자가 그 중 어떤 것이든 선택할 수 있다는 것을 의미하며, 그 모든 것을 다 수용하겠다는 것을 의미하는 건 아니다. |
| Developers/  developer\* | List<Developer> | (Many)프로젝트의 committer를 기술한다. |
| contributors/  contributor\* | List<Contributor> | (Many) 프로젝트의 contributor(아직 committer는 아님)를 기술한다 |
| mailingLists  /mailingList\* | List<MailingList> | (Many) 프로젝트 maling list에 대한 정보를 포함한다. |
| prerequisites | Prerequisites | 프로젝트의 빌드 환경의 사전 조건을 기술한다. |
| modules/  module\* | List<String> | 프로젝트의 일부로써 빌드되는 모듈(때때로 subproject라고도 불린다)을 기술한다. 열거된 각 모듈은 모듈을 포함하는 디렉토리에 대한 상대 경로이다. |
| scm | Scm | CVS, Subversion 등과 같이 프로젝트에 의해 사용되는 SCM을 기술한다. |
| issueManagement | IssueManagement | 프로젝트의 이슈 관리 시스템 정보 |
| ciManagement | CiManagement | 프로젝트 이슈 통합 정보 |
| distributionManagement | DistributionManagement | site와 artifact를 원격 웹서버와 저장소에 각각 배포하도록 하는 프로젝트 분산 정보 |
| properties/key=value\* | Properties | (Many) POM을 통해 대체(substitution)로써 사용가능한 속성들. enable된다면 resources 안에서 filter로써 사용될 수 있음. format은 <name> value </name> 임 |
| dependencyManagement | DependencyManagement | 이것으로부터 상속된 projects에 대한 기본 참조객체 정보. 이 section의 참조객체는 즉각적으로 가져와지지는 않는다. 이 것으로부터 파생된 POM이 groupId, artifactId, version 기술된 참조객체를 선언할 때, 버전과 다른 값들은 참조객체 정보를 위해 사용된다. (미리 설정되어 있지만 않았다면) |
| dependencies/dependency\* | List<Dependency | (Many) 이 element는 프로젝트와 관련된 모든 참조객체를 기술한다. 이러한 참조객체들은 Build 프로세스 동안 프로젝트의 classpath를 구축하는데 사용된다. 그들은 프로젝트에서 정의된 저장소로부터 자동으로 다온로드 된다. 보다 더 많은 정보를 보고 싶다면, [dependency mechanism](http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html) 를 참고하라. |
| repositories/repository\* | List<Repository> | (Many) depencencies 및 extensions를 찾아내기 위한 원격 저장소의 list |
| pluginRepositories/pluginRepository\* | List<Repository> | (Many) 빌드 및 보고용 플러그인을 찾아내기 위한 원격 저장소 list |
| build | Build | 프로젝트를 빌드하는데 필요한 정보 |
| reporting | Reporting | 이 element는 report plugin의 기술을 포함하여 Maven이 만들어 내는 site의 보고서를 생성하는데 사용된다. 이러한 보고서들은 사용자가 mvn site를 실행할 때 실행된다. 모든 보고서는 브라우징을 위한 탐색 bar를 포함하고 있다. |
| profiles/profile\* | List<Profile> | 활성화되었을 때, build process를 변경할 프로젝트의 지역 빌드 profile list. |

### Maven Plugins

* Core Plugins : 기본 핵심 단계(즉, clean, compile)와 상응하는 Plugin이며, 여러 개의 goal을 갖을 수 있음

|  |  |
| --- | --- |
| **Plugin** | **Description** |
| [clean](http://maven.apache.org/plugins/maven-clean-plugin/) | Build 후, Clean-up  **[goal]**  **1. clean:clean**  build 시간에 생성된 프로젝트의 working directory의 파일(들)을 지운다. default로는 project.build.directory, project.build.outputDirectory, project.build.testOutputDirectory, and project.reporting.outputDirectory로 설정된 디렉토리를 찾고, 삭제한다. |
| [compiler](http://maven.apache.org/plugins/maven-compiler-plugin/) | JAVA 소스를 Compile  **[goal]**  **1. compiler:compile**  compile 단계와 연결되어 있어 main 소스 파일들을 컴파일하는데 사용된다.  **2. compiler:testCompile**  test-compile단계와 연결되어 있어, test 소스 파일을 컴파일하는데 사용된다.  **3. compiler:help**  maven compiler 플러그인의 help 정보를 보여준다. 파라메터의 상세사항을 보여주기 위해서는 다음을 call하라🡺 mvn compiler:help -Ddetail=true -Dgoal=<goal-name> |
| [deploy](http://maven.apache.org/plugins/maven-deploy-plugin/) | 원격 저장소에 build 산출물을 배포  **[goal]**  **1. deploy:deploy**  특정 프로젝트에 의해 생성된 artifact를 자동으로 설치하는데 사용된다. 배포와 관련된 모든 정보(다는 아닐지라도)는 프로젝트의 pom 파일에 저장되어 있다.  **2. deploy:deploy-file**  pom 파일을 가지고 single artifact를 설치하는데 사용된다. 그 경우, artifact 정보는 특정 pom 파일에 의해서 선택적으로 추출될 수 있으나, command line을 사용하여 completed 되고, overridden 될 수 있다. |
| [failsafe](http://maven.apache.org/plugins/maven-failsafe-plugin/) | 고립된 classloader에서 Junit 통합 테스트를 실행  **[goal]**  **1. failsafe:integration-test**  application의 통합 테스트를 실행하는데 사용된다.  **2. failsafe:verify**  Pass 된 application의 통합 테스트를 검토하는데 사용된다. |
| [install](http://maven.apache.org/plugins/maven-install-plugin/) | 지역 저장소에 Build 산출물을 설치  **[goal]**  **1. install:install**  프로젝트의 main artifact (JAR, WAR, EAR)와 pom 파일, 특정 프로젝트에 의해 생성된 첨부 artifact (sources, javadoc 등) 을 자동으로 설치하는데 사용한다.  **2. install:install-file**  외부적으로 생성된 artifact를 POM 파일과 함께 지역 저장소에 설치하는데 사용한다. 그 경우, 프로젝트의 정보가 선택된 pom 파일에 의해 선택적으로 획득될 수는 있으나, 역시 command line 파라메터를 사용해 주어질 수도 있다.  **3. install:help**  Maven-install-plugin에 help 정보를 표시한다. |
| [resources](http://maven.apache.org/plugins/maven-resources-plugin/) | JAR에 포함시키기 위한 목적으로 자원을 output 디렉토리에 복사  **[goal]**  **1. resources:resources** Main 소스를 위한 리소스를 main output directory에 copy한다.  이 goal은 일반적으로 자동으로 실행되는데, 기본적으로 process-resources 생명주기 단계에 bound되기 때문이다. 리소스에 대한 상세사항을 적는데 project.build.resources element가 사용되며, 기본적으로 copy 목적지에 대한 상세 사항을 적기 위해 project.build.outputDirectory가 사용된다.  **2. resources:testResources** 테스트 소스를 위한 리소스를 test output directory에 copy한다.  이 goal은 일반적으로 자동으로 실행되는데, 기본적으로 process-test-resources 생명주기 단계에 bound되기 때문이다. 리소스에 대한 상세사항을 적는데 project.build.testResources element가 사용되며, 기본적으로 copy 목적지에 대한 상세 사항을 적기 위해 project.build.testOutputDirectory 가 사용된다.  **3. resources:copy-resources** output 디렉토리에 리소스를 copy한다.  이 goal은 copy된 리소스를 설정하고, outputDirectory를 기재할 것을 요구한다. |
| [site](http://maven.apache.org/plugins/maven-site-plugin/) | 현 프로젝트를 위한 site 생성  **[goal]**  **1. site:site** 단일 프로젝트를 위한 site를 생성하기 위해 사용된다. multi module 빌드에서 각 모듈 site 간의 연결(Link)이 효과 없음에 유의하라.  **2. site:deploy** scp나 파일 프로토콜을 사용하여 생성된 site를, POM 파일의 <distributionManagement> 부분에 적힌 URL로 배포하는데 사용된다.  **3. site:run** Site를 시작하고, 더 빠른 editing을 위해 요청된 문서들을 rendering한다. Jetty를 Web 서버로 사용한다.  **4. site:stage** POM 파일의 <distributionManagement> 부분에 기재된 site URL을 기반으로 local staging이나, mock directory에 site를 생성한다. Multi 모듈 빌드에서 module sites 간의 연결을 테스트하기 위해 사용될 수 있다.  **5. site:stage-deploy**  staging이나 mock 디렉토리에 생성된 site를 배포하여, POM 파일의 <distributionManagement>  부분에 기재된 site URL에 배포되도록 한다. 배포를 위해 scp와 파일 프로토콜을 지원한다.  **6. site:attach-descriptor**  설치/배포된 파일 리스트에 site descriptor(site.xml)을 추가한다. site descriptor에 대해 더 알고 싶다면, [다음 링크](http://maven.apache.org/plugins/maven-site-plugin/examples/sitedescriptor.html)를 참고하라.  **7. site:jar** JAR에 site output을 묶어서 repository에 배포할 수 있다.  **8. site:effective-site** 상속과 interpolation(보간법) 후에, 효과적인 site descriptor를 계산한다. |
| [surefire](http://maven.apache.org/plugins/maven-surefire-plugin/) | 고립된 classloader에 Junit 단위 테스트를 실행  **[goal]**  **1. surefire:test** 어플리케이션의 unit test를 실행한다. |
| [verifier](http://maven.apache.org/plugins/maven-verifier-plugin/) | 통합 테스트에 유용하며, 특정한 조건이 존재하는지 확인  **[goal]**  **1. verifier:verify**  파일/디렉토리의 존재 유무를 검증하고, 파일 내용이 정규식에 적합한지 추가적으로 체크한다. |

* Packaging Plugins : 관련된 산출물 타입을 패키지화하는 것과 관련 있는 Plugin

|  |  |
| --- | --- |
| **Plugin** | **Description** |
| [ear](http://maven.apache.org/plugins/maven-ear-plugin/) | 현 프로젝트로부터 EAR을 생성  **[goal]**  **1. ear:ear** J2EE Enterprise Archive (EAR) 파일을 생성한다.  **2. ear:generate-application-xml** 배포 descriptor 파일을 생성한다. |
| [ejb](http://maven.apache.org/plugins/maven-ejb-plugin/) | 현 프로젝트로부터 EJB (와 선택적 client)을 생성  **[goal]**  **1. ejb:ejb** ejb 패키지 타입을 사용하는 프로젝트를 위한 Maven에 의해 사용된다. |
| [jar](http://maven.apache.org/plugins/maven-jar-plugin/) | 현 프로젝트로부터 JAR을 생성  **[goal]**  **1. jar:jar** 프로젝트 소스를 위한 jar 파일을 생성한다.  **2. jar:test-jar** 프로젝트 테스트 클래스를 위한 jar 파일을 생성한다. |
| [rar](http://maven.apache.org/plugins/maven-rar-plugin/) | 현 프로젝트로부터 RAR을 생성  **[goal]**  **1. rar:rar** Resource Adapter Archive (RAR) 파일을 package하는데 사용된다. |
| [war](http://maven.apache.org/plugins/maven-war-plugin/) | 현 프로젝트로부터 WAR을 생성  **[goal]**  **1. war:war**  war 파일을 packaging 타입으로 한 프로젝트(들)의 패캐지 단계 동안 invoked되는 기본 goal이다. WAR 파일을 빌드한다.  **2. war:exploded**  일반적으로 개발 단계 동안 테스팅의 속도를 높이기 위해, 기재된 디렉토리에 exploded webapp을 생성함으로써 사용된다.  **3. war:inplace**  war:explode의 또다른 변형이며, 대신 webapp이 web application 소스 디렉토리(디폴트로는 src/main/webapp)에 생성된다.  **4. war:manifest**  이 webapp을 위한 manifest를 생성한다. manifest 파일은 웹 어플리키이션 소스 디렉토리에 생성된다. |
| [shade](http://maven.apache.org/plugins/maven-shade-plugin/) | 현 프로젝트로부터 참조객체를 포함한 Uber-JAR 을 생성  **[goal]**  **1. shade:shade** package 단계와 bound되어 있고, shaded jar 파일을 생성하는데 사용된다. |

* Reporting plugins: POM 파일에서 report로 설정되어, site 생성주기 하에 실행되는 리포트 제작 Plugins

|  |  |
| --- | --- |
| **Plugin** | **Description** |
| [changelog](http://maven.apache.org/plugins/maven-changelog-plugin/) | SCM로부터 최근 변경된 내역을 생성  **[goal]**  **1. changelog:changelog**  changelog 보고서를 생성한다. changelog 보고서는 지정된 change set 기간 동안 날짜, 파일, revision의 author를 포함하는 모든 SCM 활동에 대한 보고서를 보여준다.  **2. changelog:dev-activity**  개발자 활동 보고서를 생성한다. dev-activity 보고서는 프로젝트 descriptor에 list된 총 commit 수, 각 개발자에 의해 변경된 파일의 수에 대한 요약을 보여준다.  **3. changelog:file-activity**  파일 활동 보고서를 생성한다. 파일 활동 보고서는 지정된 changeset 동안 수정된 모든 파일의 list를 보고한다. 가장 최신의 revision을 가지는 파일이 처음에 list되는 방식으로 정렬된다. |
| [changes](http://maven.apache.org/plugins/maven-changes-plugin/) | 이슈 트래킹이나 변경 문서로부터 보고서를 생성  **[goal]**  **1. changes:announcement-mail** email을 통해 release 공고를 보낸다.  **2. changes:announcement-generate** release 공고를 생성한다.  **3. changes:changes-check** changes.xml 파일이 적합한 release 일자를 포함하는지 체크한다.  **4. changes:changes-report** 프로젝트의 각각 다른 release 사이에 무엇이 변경되었는지를 보여주는 리포트를 생성한다.  **5. changes:changes-validate** change.xml 파일의 유효성을 인증한다.  **6. changes:jira-report** JIRA로부터 다운로드 받은 이슈에 대한 리포트를 생성한다.  **7. changes:trac-report** Trac으로부터 다운로드 받은 이슈로부터 리포트를 생성한다. |
| [checkstyle](http://maven.apache.org/plugins/maven-checkstyle-plugin/) | Checkstyle 보고서 생성  **[goal]**  **1. checkstyle:checkstyle** 체크 스타일 분석을 수행하며, 위반하는 내용에 대한 리포트를 생성한다.  **2. checkstyle:check** 마지막 checkstyle 수행에 대한 위반 사항이 있는지 살핀다. Checkstyle 아웃풋 파일을 읽고, 발견된 위반 사항의 수를 세서, 콘솔 상에서 보여준다. |
| [doap](http://maven.apache.org/plugins/maven-doap-plugin/) | POM로부터 프로젝트(DOAP) 파일의 상세내역 생성  **[goal]**  **1. doap:generate** POM으로부터 DOAP 파일을 생성한다. |
| [docck](http://maven.apache.org/plugins/maven-docck-plugin/) | Checker Plugin 문서화  **[goal]**  **1. docck:check** 표준 최소 조건에 대한 플러그인의 문서화를 체크한다. |
| [javadoc](http://maven.apache.org/plugins/maven-javadoc-plugin/) | 프로젝트를 위한 Javadoc 생성  **[goal]**  **1. javadoc:javadoc** 프로젝트를 위한 Javadoc 파일을 생성한다. 표준 javadoc 도구를 실행시켜, 도구에 의해 사용되는 파라메터를 지원한다.  **2. javadoc:test-javadoc** 프로젝트를 위한 test JAVAdoc 파일을 생성한다. 표준 javadoc 도구를 실행시켜, 도구에 의해 사용되는 파라메터를 지원한다.  **3. javadoc:aggregate** aggregator 프로젝트를 위한 Javadoc 파일을 생성한다. 표준 javadoc 도구를 실행시켜, 도구에 의해 사용되는 파라메터를 지원한다.  **4. javadoc:test-aggregate** 프로젝트에 대한 테스트 Javadoc 파일을 생성한다. 표준 javadoc 도구를 실행시켜, 도구에 의해 사용되는 파라메터를 지원한다.  **5. javadoc:jar** 생성된 Javadoc에 대한 archive 파일을 생성한다. release 프로세스 동안 사용되어, 프로젝트 release를 위한 Javadoc artifact를 생성한다. 이 산출물은 컴파일된 바이너리, 소스 archive와 함께 원격 저장소에 업로드 된다.  **6. javadoc:test-jar** 생성된 Test Javadocs의 archive 파일을 생성한다.  **7. javadoc:aggregate-jar** aggregator 프로젝트에 대해 생성된 Javadocs 의 archive 파일을 생성한다.  **8. javadoc:test-aggregate-jar** aggregator 프로젝트에 대해 생성된 Test Javadocs 의 archive 파일을 생성한다.  **9. javadoc:fix** Javadoc 문서와 Java 파일을 위한 tags를 고치는 interactive goal이다.  **10. javadoc:test-fix** Javadoc 문서와 test Java 파일을 위한 tags를 고치는 interactive goal이다. |
| [jxr](http://maven.apache.org/plugins/maven-jxr-plugin/) | 소스 상호 참조 생성  **[goal]**  **1. jxr:jxr** 프로젝트의 메인 소스의 cross-reference 페이지를 생성하는데 사용된다. 생성된 JXR 파일은 프로젝트의 javadocs와 연결될 수 있다.  **2. jxr:test-jxr** 프로젝트의 테스트 소스의 cross-reference 페이지를 생성하는데 사용된다. |
| [linkcheck](http://maven.apache.org/plugins/maven-linkcheck-plugin/) | 프로젝트 문서의 Linkcheck 보고서 생성  **[goal]**  **1. linkcheck:linkcheck** Linkcheck 보고서를 생성한다. |
| [pmd](http://maven.apache.org/plugins/maven-pmd-plugin/) | PMD 보고서 생성  **[goal]**  **1. pmd:pmd**  플러그 인의 ruleset과 configuration set을 기반으로 PMD site 보고서를 생성한다. xml, csv, txt 파일과 같은 포맷으로 site report를 통해 pmd 산출 파일을 생성할 수도 있다.  **2. pmd:cpd**  PMD의 Copy/Paste Detector (CPD) 도구를 위한 리포트를 생성한다. xml, csv, txt 파일과 같은 포맷으로 cpd report를 통해 pmd 산출 파일을 생성할 수도 있다.  **3. pmd:check**  PMD 보고서가 비었는지를 검사하며, 비어있지 않다면 fail이 된다. 이 goal은 pmd:pmd를 실행할 때, 기본으로 실행된다.  **4. pmd:cpd-check**  CPD 보고서가 비었는지를 검사하여 비어있지 않다면, fail이 된다. 이 goal은 pmd:cpd 를 실행할 때, 기본으로 실행된다. |
| [project-info-reports](http://maven.apache.org/plugins/maven-project-info-reports-plugin/) | 표준 프로젝트 보고서 생성  **[goal]**  **1. project-info-reports:cim** 프로젝트의 지속적 통합 시스템 보고서 생성에 사용된다.  **2. project-info-reports:dependencies** 프로젝트 참조객체 리포트 생성에 사용된다.  **3. project-info-reports:dependency-convergence** reactor builds를 위한 프로젝트 참조객체 컨버전스 보고서 생성에 사용된다.  **4. project-info-reports:dependency-management** 프로젝트 DependencyManagement 보고서 생성에 사용된다.  **5. project-info-reports:distribution-management** 프로젝트 분산 관리 리포트 생성에 사용된다.  **6. project-info-reports:help** 프로젝트 Info 리포트 플러그 인에서 help 정보를 보여주기 위해 사용된다.  **7. project-info-reports:index** 프로젝트 인덱스 페이지를 생성하는데 사용된다.  **8. project-info-reports:issue-tracking** 프로젝트 이슈 추적 도구를 생성하는데 사용된다.  **9. project-info-reports:license** 프로젝트 라이센스 보고서를 생성하는데 사용된다.  **10. project-info-reports:mailing-list** 프로젝트 메일링 리스트 보고서를 생성하는데 사용된다.  **11. project-info-reports:modules** 프로젝트 모듈 보고서를 생성하는데 사용된다.  **12. project-info-reports:plugin-management** 프로젝트 PluginManagement 보고서를 생성하는데 사용된다.  **13. project-info-reports:plugins** 프로젝트 플러그인 보고서를 생성하는데 사용된다.  **14. project-info-reports:project-team** 프로젝트 팀 보고서를 생성하는데 사용된다.  **15. project-info-reports:scm** 프로젝트 소스 코드 관리 보고서를 생성하는데 사용된다.  **16. project-info-reports:summary** 프로젝트 정보 보고 요약을 생성하는데 사용된다. |
| [surefire-report](http://maven.apache.org/plugins/maven-surefire-report-plugin/) | 단위 테스트 결과에 기반한 보고서 생성  **[goal]**  **1. surefire-report:report** HTML 형식으로 test 결과 보고서를 생성한다.  **2. surefire-report:report-only**  이 goal은 테스트를 실행시키지 않으며, 리포트만을 build한다. SUREFIRE-257를 위한 work around로 제공되었다. |

* Tools : 기본적으로 Maven을 통해 활용할 수 있는 다양한 도구

|  |  |
| --- | --- |
| **Plugin** | **Description** |
| [ant](http://maven.apache.org/plugins/maven-ant-plugin/) | 프로젝트를 위한 Ant Build 파일 생성  **[goal]**  **1. ant:ant** Ant 빌드 파일을 생성한다.  **2. ant:clean** 모든 Ant 빌드 파일을 지운다. |
| [antrun](http://maven.apache.org/plugins/maven-antrun-plugin/) | Build의 한 단계 ant task 집합을 실행  **[goal]**  **1. antrun:run** Maven 2를 위한 Ant task를 실행시킨다. |
| [archetype](http://maven.apache.org/archetype/maven-archetype-plugin/) | archetype으로부터 프로젝트 구조 골격 생성  **[goal]**  **1. archetype:create** (삼가할 것) archetype으로부터 Maven 프로젝트를 생성한다. Archetype 플러그인 버전 1.0-alpha-7의 행동을 사용한다.  **2. archetype:generate** archetype으로부터 Maven 프로젝트를 생성한다:  archetype catalog로부터 사용자에게 atchetype을 선택하도록 묻고, 원격 저장소로부터 그것을 가져온다. 일단 가져오고 나면, 작동하는 Maven 프로젝트를 생성하기 위해 처리된다.  **3. archetype:create-from-project** 기존 프로젝트로부터 archetype을 생성한다.  **4. archetype:crawl** archetypes를 위한 저장소를 검색하고, catalog를 업데이트 한다. |
| [assembly](http://maven.apache.org/plugins/maven-assembly-plugin/) | 소스나 바이너리의 어셈블리 (분산) 빌드  **[goal]**  assembley plugin에서 main goal은 single goal이다. 이것은 모든 어셈블리를 생성하기 위해 사용된다. 다른 모든 goal은 사용이 자제되며, 향후 릴리즈에서는 제거될 것이다. |
| [dependency](http://maven.apache.org/plugins/maven-dependency-plugin/) | 참조객체 처리(copy, unpack)와 분석  **[goal]**  **1. dependency:analyze** 프로젝트의 참조객체를 분석하여 어느 것이; 사용되고 선언되고; 사용되고 선언되지 않고; 사용되지 않고 선언 되었는지를 결정한다.  **2. dependency:analyze-dep-mgt** 프로젝트의 참조객체를 분석하여, resolve된 참조객체와 dependencyManagement 부분에 list된 것 간의 mismatch를 list한다.  **3. dependency:analyze-only** analyze와 같으나, but is meant to be bound in a pom. build를 fork하지 않으며, test-compile을 실행시킨다.  **4. dependency:analyze-report** 프로젝트의 참조객체를 분석하여 어느 것이; 사용되고 선언되고; 사용되고 선언되지 않고; 사용되지 않고 선언 되었는지에 대한 요약 보고서를 생성한다.  **5. dependency:analyze-duplicate** pom.xml 파일에서 dependencies/ 와 dependencyManagement/ tags를 분석하여, 선언된 참조객체가 중복되었는지 결정한다.  **6. dependency:build-classpath** java –cp를 사용해서 classpath format 내의 지역 저장소로부터 참조객체 path를 산출하도록 한다. classpath 파일은 역시 첨부되며, main artifact와 함께 설치/배포된다.  **7. dependency:copy** 플러그인 설정에 정의된 artifacts의 list를 가져와서 특정 위치에 그것들을 copy, rename하고, 만약 바란다면, 그 버전을 stripping한다. 이 goal은 만약 local에 artifact가 존재하지 않는다면, 원격 저장소로부터 artifacts를 resolve할 수 있다.  **8. dependency:copy-dependencies** 프로젝트의 직접적인 참조객체와 선택적으로 상호적인 참조객체의 리스트를 가져와서 특정한 위치에 artifact를 복사하고, 바라는 버전을 stripping한다. 이 goal은 command line에서 실행될 수 있다.  **9. dependency:get** 특정 원격 저장소로부터 transitively한 single artifact를 다운받는다.  **10. dependency:go-offline** 오프라인으로 가기 위한 준비로써 이 프로젝트가 의존하는 (참조객체, 플러그인,보고서) 모든 것을 resolve하도록 한다.  **11. dependency:list** 프로젝트의 참조객체 list의 alias(별명)을 짓는다.  **12. dependency:properties** 파일 시스템에서 artifact를 포함하는 각 프로젝트 참조객체에 대한 속성을 설정한다.  **13. dependency:purge-local-repository** 모든 참조객체-artifact 파일을 지역 저장소로부터 clear하고, 선택적으로 다사 참조 객체를 찾아준다.  **14. dependency:resolve** 모든 참조 객체를 찾아주고, 버전을 보여준다.  **15. dependency:resolve-plugins** 플러그인과 참조객체를 찾아준다.  **16. dependency:sources** 모든 참조객체와 그 소스에 첨부된 것들을 찾아주고, 버전을 보여준다.  **17. dependency:tree** 프로젝트를 위한 참조객체 트리를 보여준다.  **18. dependency:unpack** 지역 저장소로부터 프로젝트 참조객체를 제거하고, 선택적으로 그것들을 re-resolve한다.  **19. dependency:unpack-dependencies** copy-dependencies와 같으나, unpack한다. |
| [enforcer](http://maven.apache.org/plugins/maven-enforcer-plugin/) | 환경적 제약 체크 (Maven 버전, JDK 등), 사용자 custom 룰 실행  **[goal]**  **1. enforcer:enforce** multi-project build에서 각 프로젝트를 위한 rules를 실행시킨다.  **2. enforcer:display-info** 표준 룰로써 탐지된 현 정보를 보여준다. |
| [gpg](http://maven.apache.org/plugins/maven-gpg-plugin/) | 산출물과 pom의 signature 생성  **[goal]**  **1. gpg:sign** 프로젝트 artifact, POM, 첨부된 artifact들을 배포를 위해 GnuPG로 sign한다.  **2. gpg:sign-and-deploy-file** 원격 저장소에 artifact를 sign하고, 설치한다. |
| [help](http://maven.apache.org/plugins/maven-help-plugin/) | 프로젝트의 실행 환경에 대한 정보 획득  **[goal]**  **1. help:active-profiles** 빌드를 위해 현재 active된 profile들을 list한다.  **2. help:all-profiles** 현 프로젝트 하에서 활용 가능한 profiles들을 list 한다.  **3. help:describe** Plugin과/나 Mojo(Maven plain Old Java Object)의 속성을 기술한다.  **4. help:effective-pom** 현 빌드를 위해 XML로써, active profiles가 고려된 효과적인 POM을 보여준다.  **5. help:effective-settings** profile 향상과 사용자 레벨 세팅으로 global setting을 상속이 주어진 상태에서, 프로젝트를 위한 계산된 세팅을 XML로 보여준다.  **6. help:evaluate** 대화형 모드로 사용자에 의해 주어지는 Maven expression을 평가한다.  **7. help:expressions** Maven이 지원하는 Plugin expression을 보여준다.  **8. help:system** 시스템 속성과 환경 변수와 같이 platform 상세사항을 list하여 보여준다. |
| [invoker](http://maven.apache.org/plugins/maven-invoker-plugin/) | Maven 프로젝트 집합 실행과 ouput 검증  **[goal]**  **1. invoker:install** 프로젝트 artifacts와 main build의 참조객체를dedicated된 지역 저장소로 복사하여, 고립된 환경에서 선택된 sub project에 대한 실행을 준비한다.  **2. invoker:integration-test** Maven 프로젝트 집합을 디렉토리에서 실행시킨다.  **3. invoker:verify** invoker:integration-test 의 결과를 검증한다.  **4. invoker:run** 디렉토리에서 Maven 프로젝트 집합을 실행시키고, 결과를 검증한다. invoker:integration-test 와 invoker:verify 모두를 실행시킨 것과 같다.  **5. invoker:report** 이전 빌드로부터의 결과를 site에 통합한다. |
| [jarsigner](http://maven.apache.org/plugins/maven-jarsigner-plugin/) | 프로젝트 산출물의 승인이나 검증  **[goal]**  **1. jarsigner:sign** 프로젝트 산출물과 첨부된 산출물을 싸인한다.  **2. jarsigner:verify** 프로젝트 산출물과 첨부된 산출물을 검증한다. |
| [one](http://maven.apache.org/plugins/maven-one-plugin/) | 기존 Maven 1.x 저장소 및 빌드와 상호작용하는 플러그인  **[goal]**  **1. one:convert** Maven1 project.xml (v3 pom) 을 Maven2 pom.xml (v4 pom) 로 변환시킨다.  **2. one:deploy-maven-one-repository** Maven1 원격 저정소로 산출물을 배포한다.  **3. one:install-maven-one-repository** Maven1 지역 저장소에 산출물을 설치한다.  **4. one:maven-one-plugin** Maven1 플러그인을 패키지화 한다. |
| [patch](http://maven.apache.org/plugins/maven-patch-plugin/) | 소스 코드에 patch 파일 적용하기 위한 gnu patch 도구 사용  **[goal]**  **1. patch:apply** 하나 이상의 patches를 프로젝트 소스에 적용한다. |
| [pdf](http://maven.apache.org/plugins/maven-pdf-plugin/) | PDF형식으로 프로젝트 문서화  **[goal]**  **1. pdf:pdf** 모든 프로젝트 문서를 포함하는 PDF 문서를 생성한다. |
| [plugin](http://maven.apache.org/plugins/maven-plugin-plugin/) | JAR에 포함되기 위해, 소스 트리에서 발견되는 mojos를 위한 Maven 플러그인 descriptor 생성  **[goal]**  **1. plugin:descriptor** 플러그인 descriptor를 생성한다.  **2. plugin:report** 플러그인 문서 보고서를 생성한다.  **3. plugin:updateRegistry** 설치된 버전에 반영하기 위하여, 사용자의 plugin registry를 (사용중이라면) 업데이트한다.  **4. plugin:xdoc** 프로젝트 mojos나 목표를 위한 Xdoc 파일을 생성한다.  **5. plugin:addPluginArtifactMetadata** 플러그인과 관련된 산출물 메타 데이터를 추후 설치, 배포를 위해 프로젝트의 산출물에 삽입한다. 이를 위한 첫 use-case는 가장 최신의 metadata를 플러그인의 산출물과 함께 shipping하기 위해 추가하는 것이다.  **6. plugin:helpmojo** 한 플러그인 내의 모든 mojos를 설명하는 help mojo를 생성한다. |
| [release](http://maven.apache.org/plugins/maven-release-plugin/) | 현재 프로젝트 릴리즈 – POM 파일 업데이트, SCM에 태그 만들기  **[goal]**  **1. release:clean** 릴리즈 준비 후에, clean-up 한다.  **2. release:prepare** SCM 내 release에 대한 준비를 한다.  **3. release:prepare-with-pom** SCM 내 릴리즈에 대한 준비를 하고, 사용된 fully resolved projects를 기록한 릴리즈 POMs 을 생성한다.  **4. release:rollback** 이전 릴리즈로 롤백한다.  **5. release:perform** SCM으로부터 릴리즈를 실행한다.  **6. release:stage** SCM으로부터 staging folder/repository로 릴리즈를 실행한다.  **7. release:branch** 업데이트된 모든 버전을 가진 현 프로젝트의 브랜치를 만든다.  **8. release:update-versions** POM(s)에 버전(들)을 업데이트한다. |
| [reactor](http://maven.apache.org/plugins/maven-reactor-plugin/) | Reactor 내에 상호의존적인 프로젝트의 부분집합을 Build  **[goal]**  **1. reactor:resume** 특정 시점에서의 reactor을 재개한다. (예를 들어, 중간에 fail되었을 때)  예) mvn reactor:resume -Dfrom=bar  **2. reactor:make** project X와 X가 의존하는 모든 reactor 프로젝트를 빌드한다.  예) mvn reactor:make -Dmake.folders=foo,bar  **3. reactor:make-dependents** 프로젝트 X와 X 에 의존하는 모든 reactor 프로젝트(들)을 빌드한다. (reactor:make와 반대)  예) mvn reactor:make-dependents -Dmake.folders=foo,bar  **4. reactor:make-scm-changes** 개인적으로 변경한 (SCM에 따라) 모든 reactor 프로젝트(들)와, 변화에 의존하는 모든 reactor 프로젝트를 빌드한다.  예) mvn reactor:make-scm-changes |
| [remote-resources](http://maven.apache.org/plugins/maven-remote-resources-plugin/) | 원격 리소스를 output 디렉토리에 복사하여, artifact 내에 포함  **[goal]**  **1. remote-resources:bundle** 원격 리소스 번들 처리에 의해 요구되는 리소스 번들 manifest를 생성한다. manifest 파일인, $basedir/target/classes/META-INF/maven/remote-resources.xml 는 src/main/resources 디렉토리의 컨텐츠로부터 생성된다.  **2. remote-resources:process** 특정 원격 리소스 번들을 검색하여, 처리하고, process-resources 단계에 활용가능하도록 만든다. |
| [repository](http://maven.apache.org/plugins/maven-repository-plugin/) | 저장소 기반 task들을 돕기 위한 Plugin  **[goal]**  **1. repository:bundle-create** Maven으로 프로젝트를 빌드하기 위한 업로드 번들을 생성한다.  **2. repository:bundle-pack** 지역 저장소에 존재하는 산출물을 위한 업로드 번들을 생성한다. |
| [scm](http://maven.apache.org/scm/maven-scm-plugin/) | 현 프로젝트를 위한 SCM 생성  **[goal]**  **1. scm:branch** 프로젝트 브랜치를 만든다.  **2. scm:validate** 파일 내의 scm 정보를 인증한다.  **3. scm:add** 파일을 추가하는 명령  **4. scm:unedit** 작동하는 copy 에 대한 편집을 멈추는 명령  **5. scm:export** 새로 exported된 copy를 얻는 명령  **6. scm:bootstrap** 프로젝트를 체크아웃하고, 빌드하기 위한 명령  **7. scm:changelog** 소스 코드 revisions을 보기 위한 명령  **8. scm:list** 프로젝트 파일의 리스트를 얻기 위한 명령  **9. scm:checkin** 변화(들)을 commit하기 위한 명령  **10. scm:checkout** 소스 코드를 얻기 위한 명령  **11. scm:status** 동작하는 copy에 대한 scm 현황을 보기 위한 명령  **12. scm:update** 최신 변화(들)이 있는 동작하는 copy를 업데이트하기 위한 명령  **13. scm:diff** 원격에 있는 것과 작동하는 copy의 차이를 보기 위한 명령  **14. scm:update-subprojects** 다중 프로젝트 빌드에서 모든 프로젝트를 업데이트하기 위한 명령  **15. scm:edit** 동작하는 copy에 대한 편집을 시작하는 명령  **16. scm:tag** 특정 revision에 대한 tagging을 하는 명령 |
| [source](http://maven.apache.org/plugins/maven-source-plugin/) | IDE에서 사용하고, 저장소에 배포하기 위해 소스들의 JAR를 Build  **[goal]**  **1. source:aggregate** aggregator 프로젝트에 있는 모든 모듈에 대한 소스를 모은다.  **2. source:jar** 프로젝트 main 소스(들)을 jar archive로 묶는데 사용된다.  **3. source:test-jar** 프로젝트 테스트 소스(들)을 jar archive로 묶는데 사용된다.  **4. source:jar-no-fork** jar와 비슷하지만 빌드 생명주기를 fork하지는 않는다.  **5. source:test-jar-no-fork** test-jar와 비슷하나 빌드 생명주기를 fork하지는 않는다. |
| [stage](http://maven.apache.org/plugins/maven-stage-plugin/) | release staging 와 promotion 지원  **[goal]**  **1. stage:copy** 한 저장소로부터 다른 저장소로 산출물을 복사한다. |
| [toolchains](http://maven.apache.org/plugins/maven-toolchains-plugin/) | Plugin간 설정을 공유하는 것을 허용  **[goal]**  **1. toolchain:toolchain**  이 goal은 전형적으로 기본 생명주기의 “validate” 단계에 attached된다. |

• IDE Plugins : IDE와 간편하게 통합해주는 Plugin

|  |  |
| --- | --- |
| ***Plugin*** | ***Description*** |
| [eclipse](http://maven.apache.org/plugins/maven-eclipse-plugin/) | 현 프로젝트를 위한 Eclipse 프로젝트 파일 생성  **[Standard eclipse goals]**  **1. eclipse:configure-workspace** classpath 변수 M2\_REPO 를 지역 저장소를 가르키고 있는 이클립스에 추가하는 것과, 선택적으로 다른 워크스페이스 특징을 설정하는데 사용된다.  **2. eclipse:eclipse** Eclipse 설정 파일들을 생성한다.  **3. eclipse:clean** Eclipse IDE에 의해 사용된 파일을 제거하는데 사용된다.  **[specific goals]**  **Specific goals for PDE developments**  **1. eclipse:to-maven** 이클립스 설치에서 나온 산출물을 지역 저장소에 추가한다.  이 mojo는 자동적으로 eclipse 디렉토리를 분석하여, 플러그인 jars를 지역 maven 저장소에 복사하고, 적합한 poms를 생성한다. 이것은 이클립스 플러그인을 위한 공식 중앙 저장소 빌더이다. 그래서 필요한 기본값이 있다. 맞춤된 저장소를 위해서는 eclipse:make-artifacts 을 참고하라.  **2. eclipse:install-plugins** 특정 타입(보통 'eclipse-plugin')의 모든 resolved 프로젝트 참조객체를 지정된 Eclipse 설치 디렉토리에 설치한다.  **3. eclipse:make-artifacts** 설치디렉토리로부터 모든 Eclipse 플러그인을 지역 저장소로 copy하고, 각각을 위한 POMs를 만든다.  **Specific goals to support myeclipse**  **1. eclipse:myeclipse** MyEclipse 설정파일을 생성한다.  **2. eclipse:myeclipse-clean** MyEclipse에 의해 사용된 설정 파일들을 지운다.  **Specific goals to support RAD v6**  **1. eclipse:rad** Rad-6 설정 파일을 생성한다.  **2. eclipse:rad-clean** Rad-6에 의해 사용된 모든 설정 파일들을 지운다. |
| [idea](http://maven.apache.org/plugins/maven-idea-plugin/) | 현 프로젝트(각 모듈이 IDEA 모듈로 생성됨)를 위한 IDEA workspace를 생성/업데이트  **[goal]**  **1. idea:idea** 이 플러그인의 다른 세가지 goal: project, module, workspace을 실행하는데 사용된다.  **2. idea:project** IntelliJ IDEA 프로젝트에 필요한 project 파일 (\*.ipr)을 생성하는데 사용된다.  **3. idea:module** IntelliJ IDEA 모듈에 필요한 module 파일 (\*.iml)을 생성하는데 사용된다.  **4. idea:workspace** IntelliJ IDEA 프로젝트에 필요한 workspace file (\*.iws)을 생성하는데 사용된다.  **5. idea:clean** IntelliJ IDEA와 관련있는 파일들을 지우는데 사용된다. |

### Maven POM 파일의 설정 방법

Maven Plugin 종류는 Build Plugin과 Report Plugin으로 나뉜다.

▪ Build Plugin : Build되는 동안 실행되며, <build/> element 안에서 설정되어야 한다.

▪ Report Plugin : site 생성 동안 실행되며, <reporting/> element 안에서 설정되어야 한다.

모든 Plugin은 최소한으로 요구되는 정보를 가져야 한다: groupId, artifactId, version

주의: Build시, Build 재현성을 보장하기 위해 Plugin의 각 버전을 항상 정의할 것을 권장한다.

좋은 사례로 <build><pluginManagement/></build> element를 모든 Build Plugin마다 상세화하는 것이다. (일반적으로는, parent POM 안에 <pluginManagement/> element를 정의할 것이다.)

Report 플러그인에서는 <reporting><plugins/></reporting> element 안에 모든 버전을 상세화해야하며, 역시 <build><pluginManagement/></build> element 안에도 상세화해야한다.

일반적 설정

Maven Plugins (build와 reporting)는 <configuration/> element를 상세화하여 설정되며, <configuration/> element의 자식 element들이 Mojo 내에서 필드나 setters로 map된다. (하나의 Plug-in은 하나 혹은 그 이상의 Mojos로 구성되며, 하나의 Mojo 는 하나의 목표로 map되는 것을 기억하라)

예를 들어, 우리가 timeout과 option list가 기재된 특정한 URL에 대한 쿼리를 수행하는 Mojo를 가지고 있다면, Mojo는 다음과 같을 것이다.

/\*\*

\* @goal query

\*/

public class MyQueryMojo extends AbstractMojo

{

/\*\*

\* @parameter expression="${query.url}"

\*/

private String url;

/\*\*

\* @parameter default-value="60"

\*/

private int timeout;

/\*\*

\* @parameter

\*/

private String[] options;

public void execute() throws MojoExecutionException

{ ... }

}

희망하는 URL, timeout, option을 담고 있는 POM으로부터 Mojo를 설정하기 위해 다음과 같이 작업할 수 있다.

<project>

...

<build>

<plugins>

<plugin>

<artifactId>maven-myquery-plugin</artifactId>

<version>1.0</version>

<configuration>

<url>http://www.foobar.com/query</url>

<timeout>10</timeout>

<options>

<option>one</option>

<option>two</option>

<option>three</option>

</options>

</configuration>

</plugin>

</plugins>

</build>

...

</project>

설정 내에 element들은 Mojo의 필드 명과 매치된다. Maven이 차용한 설정 매커니즘은 XML 내의 element들이 object와 map되는 Xstream의 방식과 매우 유사하다. 위의 예제는 매우 직관적이며, url element는 url field에, timeout element는 timeout field에, options element는 options filed에 map된다. Mapping 매커니즘은 Field의 유형을 검사하고, 적합한 mapping이 가능한지를 결정하는 방식으로 배열을 다룬다.

CLI와 파라메터로부터 직접 실행되도록 고안된 mojos는 POM 파일의 <configuration/> 부분 대신, system 속성을 통해 설정할 수 있는 수단을 제공한다.

이러한 파라메터에 대한 플러그인 문서화는 설정을 위한 시스템 속성을 나타내는 표현들을 열거할 것이다. 위의 mojo와 url 파라메터는 ${query.url} 표현과 연관되어 있으며, 이 값이 아래와 같은 시스템 속성, query.url 에 의해 설정될 것이라는 것을 의미한다.

mvn myquery:query -Dquery.url=http://maven.apache.org

시스템 속성의 이름이 mojo의 이름과 반드시 match되지는 않는다는 것을 주목하라. 이것이 일반적인 사례인 반면에, 다른 시스템 속성과 이름 clash가 나는 것을 피하기 위해 시스템 속성에 prefix를 차용하는 플러그인을 종종 발견할 것이다. 매우 드물긴 하지만, 전혀 관련없는 파라메터 이름을 가진 (역사적인 이유로) 시스템 속성을 차용하는 플러그인 파라메터도 있다. 그러므로 플러그인 문서를 매우 주의깊게 보아야 한다.

• **Help Goal**

최근 maven plugin은 일반적으로 command line에서 helpgoal을 사용하여 plugin의 설명과 parameter, type을 알아낸다. 예를 들어, javadoc goal을 이해하기 위해, 다음과 같이 부를것이다:

mvn javadoc:help -Ddetail -Dgoal=javadoc

• **Parameter 설정**

**- Simple Objects Mapping**

Boolean이나 Integer와 같은 Simple 타입의 Mapping은 정말 간단하다. <configuration/> element가 다음과 같을 것이다.

...

<configuration>

<myString>a string</myString>

<myBoolean>true</myBoolean>

<myInteger>10</myInteger>

<myDouble>1.0</myDouble>

<myFile>c:\temp</myFile>

<myURL>http://maven.apache.org</myURL>

</configuration>

...

**- Complex Objects Mapping**

Maven에서는 complex type에 대한 mapping도 매우 직관적이어서, Person Object에 대한 설정을 map하기 위한 간단한 예제를 보면, <configuration/> element가 다음과 같을 것이다.

...

<configuration>

<person>

<firstName>Jason</firstName>

<lastName>van Zyl</lastName>

</person>

</configuration>

...

Complex object를 mapping하는 법칙은 다음과 같다:

Element 이름에 상응하는 private field가 있어야 한다. 그래서 이 예제에서는 person element는 mojo의 person field에 map되어야만 한다. 예시된 Object는 Mojo 와 같은 package 내에 있어야 한다. 그래서 mojo가 com.mycompany.mojo.query 내에 에 있다면, Person Object를 위한 package 내에 mapping 매커니즘이 보여야 한다.

매커니즘은 element의 첫자는 대문자이며, object를 찾는 것을 예제로 사용할 것이다.

만약 다른 패키지에 있는 object를 사용하길 희망하거나, 복잡한 이름을 갖길 희망한다면, 다음과 같이 implementation attribute을 사용하여 설정해야 한다.

...

<configuration>

<person implementation="com.mycompany.mojo.query.SuperPerson">

<firstName>Jason</firstName>

<lastName>van Zyl</lastName>

</person>

</configuration>

...

• **Mapping Collections**

설정 mapping 매커니즘은 대부분의 collection을 쉽게 다룰 수 있다. 이것이 어떻게 이루어지는지 몇가지 예제를 살펴보자.

**- Mapping list**

List mapping은 List에 element의 list를 mapping하려고 할때, array mapping과 같은 방식으로 작동한다.

그래서, 만약 다음과 같은 mojo를 가지고 있다면,

public class MyAnimalMojo

extends AbstractMojo

{

/\*\*

\* @parameter

\*/

private List animals;

public void execute()

throws MojoExecutionException

{

...

}

}

animals라는 필드를 가지고 있고, plugin에 대한 설정은 다음과 같을것이다.

<project>

...

<build>

<plugins>

<plugin>

<artifactId>maven-myanimal-plugin</artifactId>

<version>1.0</version>

<configuration>

<animals>

<animal>cat</animal>

<animal>dog</animal>

<animal>aardvark</animal>

</animals>

</configuration>

</plugin>

</plugins>

</build>

...

</project>

List된 animal들이 animals field에 입력될 것이다. array와는 달리 collection은 지정된 component 타입이 없다. List된 항목의 타입을 도출하기 위해 다음과 같은 전략이 사용된다 :

만약 XML element가 implementation hint 속성을 가지고 있다면, 그것이 사용된다.

만약 XML 태그가 a.를 포함한다면, 자격있는 class이름으로 여겨라

(맨 앞글자가 대문자인) XML tag 를 mojo/objecct가 설정된 같은 package 내에 있는 class로 여겨라

만약 element가 children이 없다면, 이것을 String 타입으로 가정하라. 그렇지 않으면, 설정이 실패할 것이다.

**- Mapping Maps**

같은 방식으로, 다음과 같이 maps를 정의할 수 있다.

/\*\*

\* My Map.

\*

\* @parameter

\*/

private Map myMap;

<configuration>

<myMap>

<key1>value1</key1>

<key2>value2</key2>

</myMap>

</configuration>

**- Mapping Properties**

속성(properties)는 다음과 같이 정의되어야 한다.

/\*\*

\* My Properties.

\*

\* @parameter

\*/

private Properties myProperties;

<configuration>

<myProperties>

<property>

<name>propertyName1</name>

<value>propertyValue1</value>

<property>

<property>

<name>propertyName2</name>

<value>propertyValue2</value>

<property>

</myProperties>

</configuration>

Build Plugin 설정

다음은 <build/> element 내에서 Build Plugin을 설정하기 위해서 사용된다.

• **<executions/> Tag 사용**

<executions> tag를 사용하여 mojo를 설정할 수 있다. 이것은 mojos를 위해 사용되는 가장 일반적으로 사용되는 방법이며, Build 생명주기의 일부 단계에서 참여하기 위해 고안되었다. 예로, MyQueryMojo를 사용하여, 다음과 같이 보이도록 작업할 수 있다.

<project>

...

<build>

<plugins>

<plugin>

<artifactId>maven-myquery-plugin</artifactId>

<version>1.0</version>

<executions>

<execution>

<id>execution1</id>

<phase>test</phase>

<configuration>

<url>http://www.foo.com/query</url>

<timeout>10</timeout>

<options>

<option>one</option>

<option>two</option>

<option>three</option>

</options>

</configuration>

<goals>

<goal>query</goal>

</goals>

</execution>

<execution>

<id>execution2</id>

<configuration>

<url>http://www.bar.com/query</url>

<timeout>15</timeout>

<options>

<option>four</option>

<option>five</option>

<option>six</option>

</options>

</configuration>

<goals>

<goal>query</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

...

</project>

id "execution1”을 가지는 첫번째 실행은, 이 설정을 테스트 단계에 연결시켜준다.

두번째 실행은 <phase> tag를 가지지 못하는데, 이 실행이 어떻게 이루어질 것으로 생각하는가?

Goal은 앞으로 아래에서 논의될 것처럼 default 단계 binding을 가질 수 있다. 만약 목표가 default 단계 binding 을 가지면, 그 단계에서 실행될 것이다. 그러나, 만약 목표가 어떤 생명주기 단계에도 bound되어 있지 않다면, build 생명주기 동안 실행되지 못할 것이다.

실행 id가 POM 내에서 단일 plugin의 모든 실행 중에서 unique해야하지만, POM들의 상속 관계에까지 unique할 필요는 없다. 다른 POMs에서 나온 같은 id의 실행은 merge된다. 이와 같은 원리는 profiles에 의해 정의된 실행에도 똑같이 적용된다.

그렇다면, 각각 다른 단계를 가진 다중의 실행이 그것(?)에 bound된다면 어떻할 것인가? 이것이 어떻게 실행될 것인가? 위의 POM 예제를 다시 사용해보자. 그러나 이번에는 execution2를 phase에 bind할 것이다.

<project>

...

<build>

<plugins>

<plugin>

...

<executions>

<execution>

<id>execution1</id>

<phase>test</phase>

...

</execution>

<execution>

<id>execution2</id>

<phase>install</phase>

<configuration>

<url>http://www.bar.com/query</url>

<timeout>15</timeout>

<options>

<option>four</option>

<option>five</option>

<option>six</option>

</options>

</configuration>

<goals>

<goal>query</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

...

</project>

만약, 다중의 실행이 각각 다른 단계에 bound된다면, mojo는 언급된대로 각 단계를 위해 한번씩만 실행된다. 다시 말하자면, execution1은 build 단계가 test일 때 설정 setup을 위해 실행되며, execution2는 build 단계가 이미 설치되고 나서, 설정 setup을 위해 실행 될 것이다.

그럼 default 생명주기 단계 binding을 보여주는 mojo의 또 다른 예를 보자.

/\*\*

\* @goal query

\* @phase package

\*/

public class MyBindedQueryMojo

extends AbstractMojo

{

/\*\*

\* @parameter expression="${query.url}"

\*/

private String url;

/\*\*

\* @parameter default-value="60"

\*/

private int timeout;

/\*\*

\* @parameter

\*/

private String[] options;

public void execute()

throws MojoExecutionException

{

...

}

}

위의 mojo 예제를 보면, MyBindedQueryMojo는 package 단계에 기본으로 bound되어 있다. (@phase notation을 보라) 그러나, 만약 우리가 이 mojo를 package가 아닌 install 단계동안 실행하기 원한다면, 우리는 mojo를 <execution> tag 아래 <phase> tag를 이용하여 새로운 생명주기 단계에 다시 bind할 수 있다.

<project>

...

<build>

<plugins>

<plugin>

<artifactId>maven-myquery-plugin</artifactId>

<version>1.0</version>

<executions>

<execution>

<id>execution1</id>

**<phase>install</phase>**

<configuration>

<url>http://www.bar.com/query</url>

<timeout>15</timeout>

<options>

<option>four</option>

<option>five</option>

<option>six</option>

</options>

</configuration>

<goals>

<goal>query</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

...

</project>

이제, default 단계가 package였던 MyBindedQueryMojo 는 install 단계로 override되었다.

\*\*주의 : <executions> tag 안에 설정은 direct command line 호출로부터 사용될 수 없는 <executions> 외부에 있는 설정과 다르다. bound된 생명주기 단계가 호출될 때만 적용된다. 대안적으로, 만약 executions section밖으로 설정 부분을 옮긴다면, 이것은 플러그인의 모든 호출에 의해 범용적으로 적용될 수 있다.

• **<dependencies/> Tag 사용**

가장 최근에 나온 참조객체 버전을 사용하기 위해 Build Plugin의 dependency 를 설정할 수 있다.

예를 들어, Maven Antrun Plugin1.2는 Ant 버전 1.6.5를 사용한다. 만약 최신 Ant 버전을 사용하고 싶다면, 다음과 같이 <dependencies/> element를 추가해야한다.

<project>

...

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.2</version>

...

<dependencies>

<dependency>

<groupId>org.apache.ant</groupId>

<artifactId>ant</artifactId>

<version>1.7.1</version>

</dependency>

<dependency>

<groupId>org.apache.ant</groupId>

<artifactId>ant-launcher</artifactId>

<version>1.7.1</version>

</dependency>

</dependencies>

</plugin>

</plugins>

</build>

...

</project>

• **Build Plugin에서 <inherited/>Tag 사용**

기본적으로, 플러그인 설정은 child POM으로 확산되어야 한다. 그래서 상속을 깨기 위해서는 <inherited/> tag를 사용해야 한다.

<project>

...

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.2</version>

<inherited>false</inherited>

...

</plugin>

</plugins>

</build>

...

</project>

Reporting Plugin 설정

다음은 <reporting/> element 안에서 Reporting 플러그인을 설정하는데 사용된다.

• **<reporting/> Tag VS <build/> Tag**

Pom 내에 <reporting/> 나 <build/> element 에서 reporting 플러그인을 설정하는 것은 같은 동작이 아니다.

mvn site

🡺 <reporting/> 에서 기술된 각 reporting Plugin의 <configuration/> 안에서 정의된 파라메터만을 사용한다.

즉, site는 <build/>에서 기술된 각 플러그인의 <configuration/> element에 정의된 파라메터는 항상 무시한다.

mvn aplugin:areportgoal

🡺 <reporting/> 에서 기술된 각 reporting Plugin의 <configuration/> 안에서 정의된 파라메터를 우선적으로 사용한다; 만약 파라메터가 발견되지 않는다면, <build/>에서 기술된 각 plugin의 <configuration/>에 정의된 파라메터를 찾을 것이다.

• **<reportSets/>Tag 사용**

<reportSets> Tag를 사용하여 reporting plugin을 설정할 수 있다. 이것은 가장 일반적으로 mvn site를 수행할 때 선별적으로 report를 생성하기 위해 사용된다. 다음은 프로젝트 팀 보고서를 생성할 것이다.

<project>

...

<reporting>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>2.1.2</version>

<reportSets>

<reportSet>

<reports>

<report>project-team</report>

</reports>

</reportSet>

</reportSets>

</plugin>

</plugins>

</reporting>

...

</project>

\*\* 유의

1. 모든 report를 제외시키기 위해서, 다음과 같이 사용하라:

<reportSets>

<reportSet>

<reports/>

</reportSet>

</reportSets>

2. 활용가능한 report goal을 알고 싶다면, 각 Plugin 문서를 참고하라 (즉, plugin-info.html)

• **Reporting Plugin 안에서 <inherited/>Tag 사용**

Build Plugin과 유사하게, 상속을 깨기 위해, <inherited/> tag를 사용할 수 있다.

<project>

...

<reporting>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>2.1.2</version>

<inherited>false</inherited>

</plugin>

</plugins>

</reporting>

...

</project>