# ANT

WIN 7:

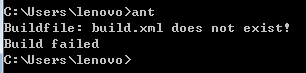
下载apache-ant-1.8.4-bin.zip，解压到目标目录（如D:\apache-ant-1.7.0）

在我的电脑->环境变量中增加

ANT\_HOME= D:\apache-ant-1.7.0

PATH=…; %ANT\_HOME%\bin;

命令行输入ant



则安装成功

在命令行中输入ant，ant则会自动在当前目录中寻找build.xml

或者在eclipse中，build.xml上右键弹出菜单中Run as->ant build

## Eclipse安装ANT

Google搜eclipse SDK 3.6和eclipse sourceBuild srcIncluded 3.6.2可找到源码

源码下载：

http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.6.2-201102101200/eclipse-sourceBuild-srcIncluded-3.6.2.zip

Ant与makefile类似，用于编译eclipse的build.xml。

Windows：

下载apache-ant-1.7.0-bin.zip，解压后设置ANT\_HOME为D:\apache-ant-1.7.0，Path增加D:\apache-ant-1.7.0\bin

打开cmd，输入D:\apache-ant-1.7.0\ant

如果出现

Buildfile: build.xml does not exist!

Build failed

则证明安装成功。

Linux：

apache-ant-1.7.0-bin.tar.gz解压后，vi　~/.bashrc

export ANT\_HOME=/path

export PATH=/path/bin

# maven

## 安装

### 安装maven

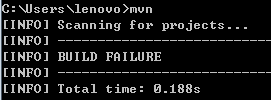
下载apache-maven-3.0.4-bin.zip，解压到目标目录（如D:\apache-maven-3.0.4）

在我的电脑->环境变量中增加

M2\_HOME= D:\\apache-maven-3.0.4

PATH=…; %M2\_HOME%\bin;

命令行输入mvn



则安装成功

### 修改本地仓库路径

\apache-maven-3.0.4\conf\setting.xml（该文件对全局有效，最好不要修改），修改maven的本地仓库路径为E:/maven/repo

<!-- localRepository

| The path to the local repository maven will use to store artifacts.

|

| Default: ~/.m2/repository

-->

<localRepository>E:/maven/repo</localRepository>

cmd中运行mvn help:system ，将文件从maven中央仓库下载到E:/maven/repo本地仓库中。

### 配置eclipse

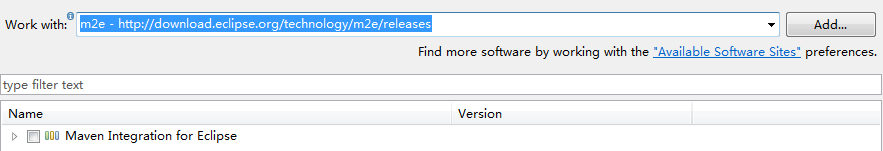
使用eclipse开发maven项目，需要先安装m2eclipse这个eclipse的插件。

Help->Install New Software

在Install框中，选择Add...

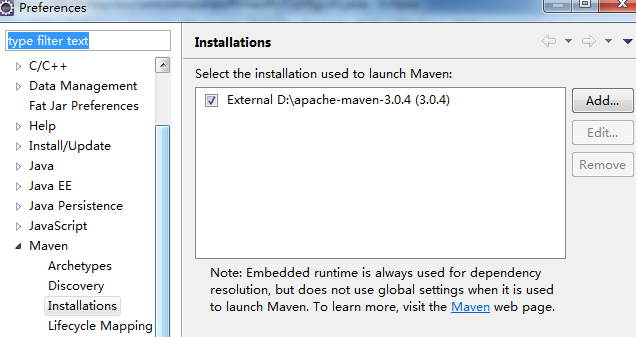
Name：m2e

Location：http://download.eclipse.org/technology/m2e/releases



选择Maven Integration for Eclipse，Next，Accept，Install

安装m2eclipse后，会内嵌一个maven，与当前maven容易出现版本冲突，

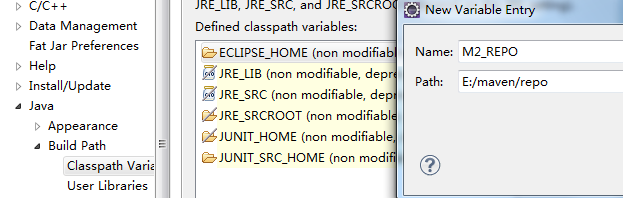


Window->Perferences->Java->Build Path->Classpath Variables

New一个变量的类路径。

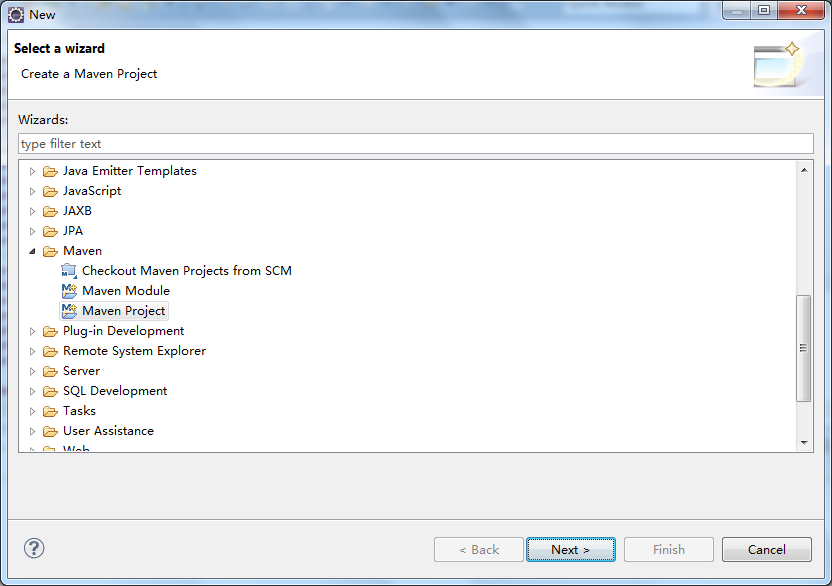
Name ：M2\_REPO   注意这个名字必须要大写。

Path ：E:/maven/repo  点击“Folder…”找到有本地仓库的位置。

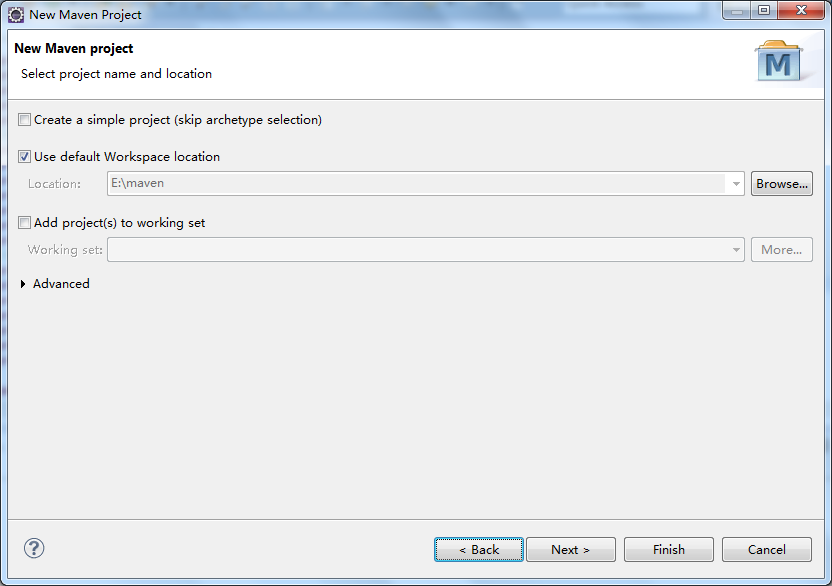


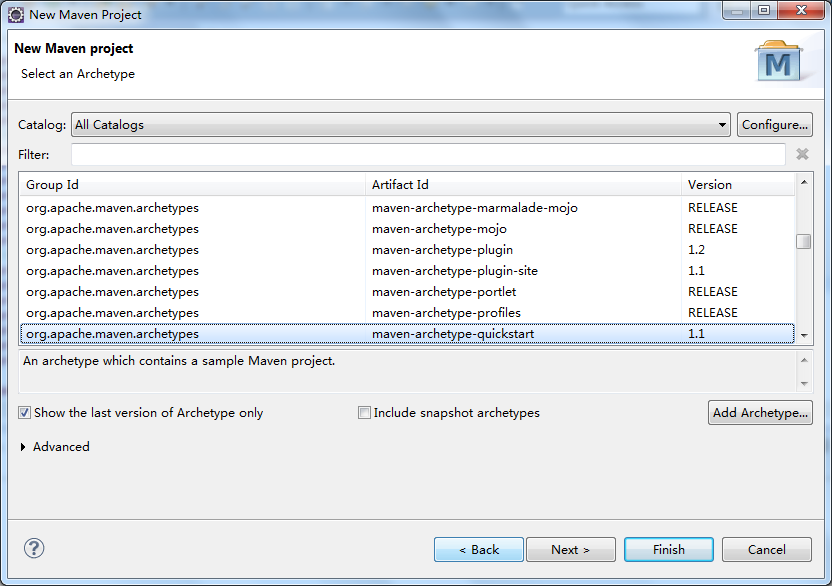
### Eclipse创建maven项目

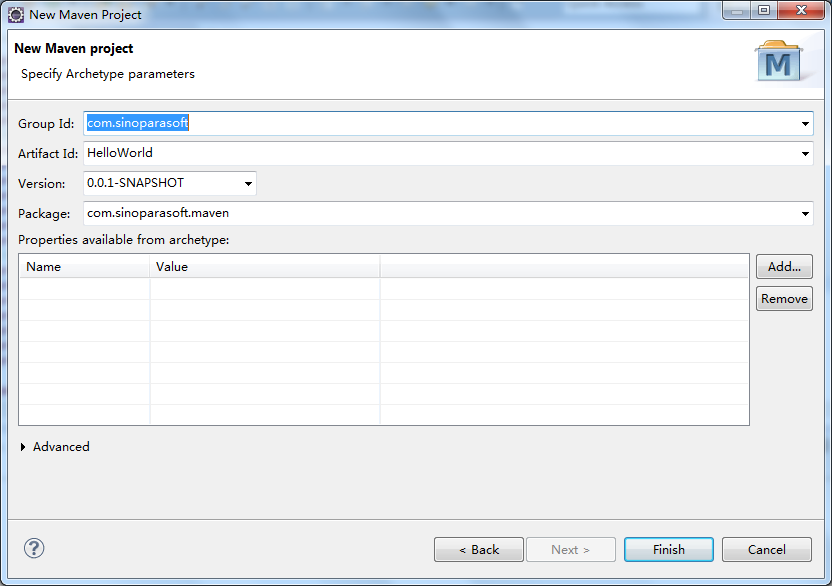
File->New->Other

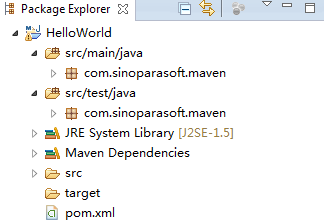


不要选Create a simple project，而是利用Maven Archetype生成项目。



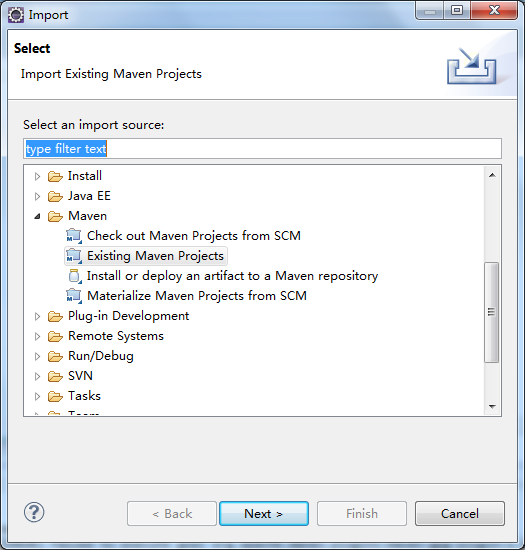






### Eclipse导入maven项目

File->Import->Maven



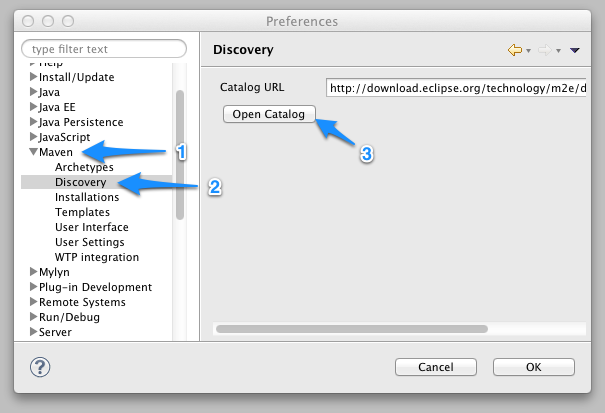
Eclipse四种导入maven项目方式：

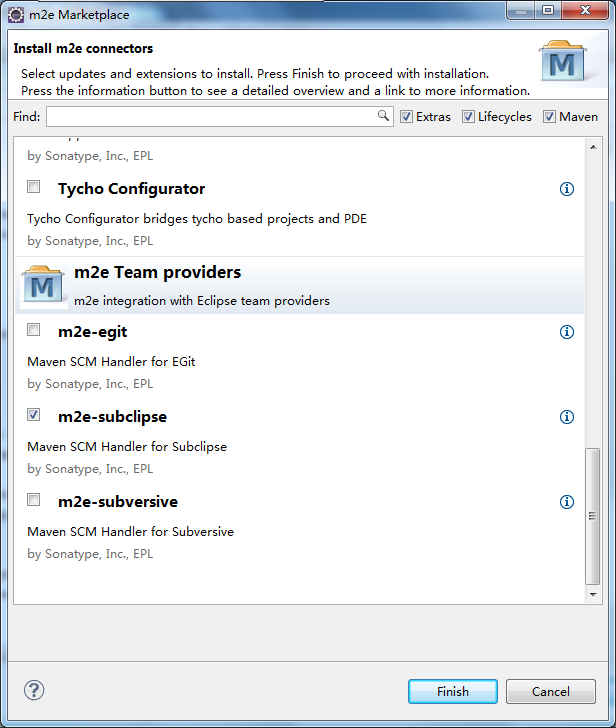
Existing Maven Project：导入本地的maven项目

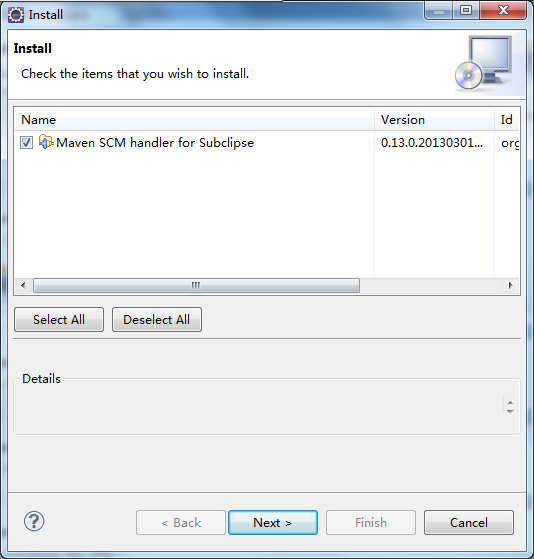
Check out Maven Project from SCM：以csv，sub，git检出maven 项目

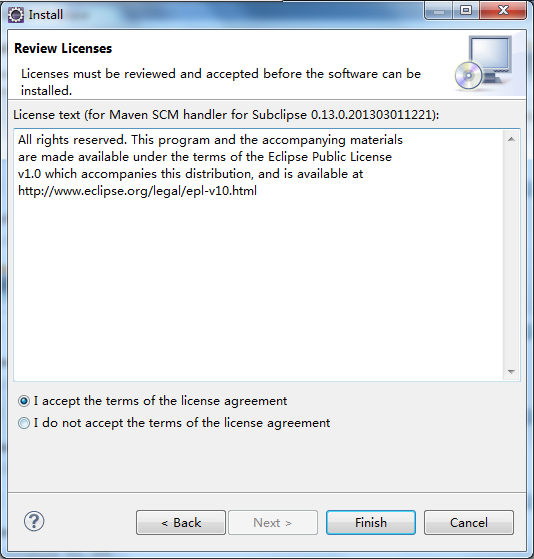
采用svn方式必须首先安装subclipse，即eclipse的svn插件，同时还需安装Maven SCM Handler for Subclipse，

Window->Preference->Maven->Discovery->Open Catalog，选中m2e-subclipse，Finish







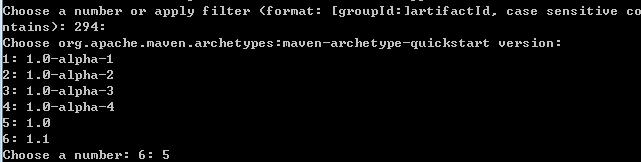


### 命令行创建maven项目

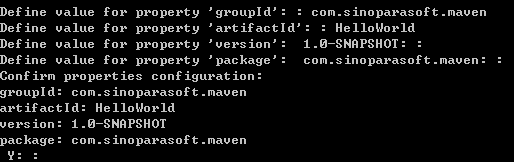
可以用mvn archetype:create 直接指定参数，也可以用mvn archetype:generate，通过向导设置参数。groupId与packageName保持一致。

mvn archetype:create -DgroupId=com.sinaparasoft.maven -DartifactId=HelloWorld -DpackageName=com. sinaparasoft.maven -Dversion=1.0 -DarchetypeArtifactId=maven-archetype-quickstart

mvn archetype:generate



选择archetypeArtifactId号，回车选择默认maven-archetype-quickstart，输入5，选择1.0版本



输入groupId和artifactId，三个回车

进入项目，运行mvn clean compile，生成target目录。



运行mvn eclipse:eclipse，生成.classpath和.project文件

运行mvn package，target目录中生成HelloWorld-1.0-SNAPSHOT.jar

运行java –cp target/HelloWorld-1.0-SNAPSHOT.jar com.sinoparasoft.maven.App



运行mvn install，把项目安装到本地库

1. **包更新**

修改项目中pom.xml的Junit的版本号，然后重新执行：mvn eclipse:eclipse

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.8.1</version>

<scope>test</scope>

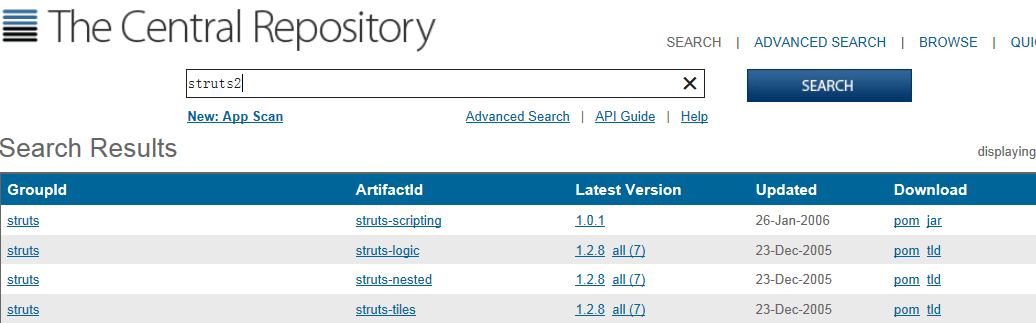
</dependency>

</dependencies>

1. **包下载**

Maven 中央仓库地址：[http://search.maven.org](http://search.maven.org/)

 假如，我们想下载一个struts 的jar包。在搜索框内搜索struts2 ，会要列出中央仓库中的所有struts版本。列表的格式与pom.xml配置文件的格式是对应的。



在pom.xml中加入：

<groupId>stuts</groupId>

 <artifactId>struts-scripting</artifactId>

 <version>1.0.1</version>

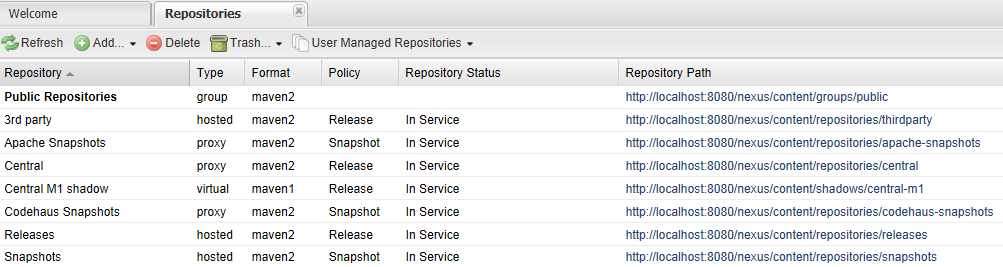
然后更新项目就可从中央仓库下载我们想要的任意jar包（必须是开源的包）

### 安装Nexus

<http://www.sonatype.org/nexus/go>下载war包nexus-2.6.0-05.zip，解压，目录名改为nexus，放到tomcat的webapps下，bin/startup.bat启动tomcat

<http://localhost:8080/nexus>访问nexus，单击右上角log in，默认管理员帐号admin，密码admin123。

单击左侧的Repositories



### 安装Hudson

<http://hudson-ci.org下载war>包，修改文件名为hudson.war，放入tomcat/webapps中启动tomcat，tomcat/bin/startup.bat<http://localhost:8080/hudson/>访问

## 基础知识

### maven目录

|  |  |
| --- | --- |
| 目录 | 目的 |
| ${basedir} | 存放 pom.xml和所有的子目录 |
| ${basedir}/src/main/java | 项目的 java源代码 |
| ${basedir}/src/main/resources | 项目的资源，比如说 property文件 |
| ${basedir}/src/test/java | 项目的测试类，比如说 JUnit代码 |
| ${basedir}/src/test/resources | 测试使用的资源 |
| ${basedir}/target | 产生的JAR 文件 |
| ${basedir}/target/classes | 编译后的classes |

### maven坐标

在当前目录下生成名为HelloWorld 的maven项目，其中包含一个Project Object Model文件（pom.xml），这个文件用于描述项目，配置插件和管理依赖关系。

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.sinoparasoft</groupId>

<artifactId>HelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>HelloWorld</name>

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

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

#### 坐标定义

groupId：必须，定义当前Maven项目隶属的实际项目。定义到项目级别。

artifactId：必须，定义实际项目中的一个Maven项目。建议以项目名做前缀。

version：必须，定义Maven项目当前版本。包括SNAPSHOT。

packaging：可选，定义Maven项目的打包方式。默认为jar。

classifier：不能直接定义，定义构建输出的附属构建。

文件名为artifactId-version[-classifier].packaging。

### 依赖配置

groupId,artifactId,version：依赖的基本坐标。

type：依赖类型，等同于坐标定义的packaging，默认为jar。

scope：依赖范围，控制依赖与（编译|测试|运行classpath）的关系。

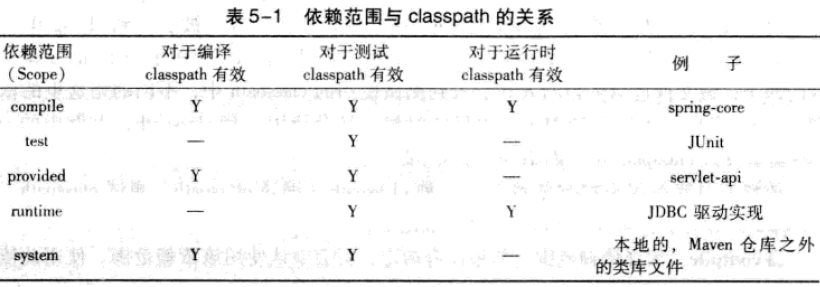
compile：编译依赖范围，默认值，对三种classpath都有效。如spring-core。

test：测试依赖范围，只对测试classpath有效。如JUnit。

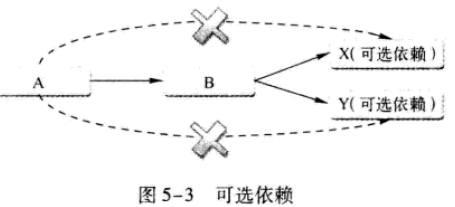
provided：已提供依赖范围，对编译|测试classpath有效，对运行时无效。如servlet-api。

runtime：运行时依赖范围，对测试和运行classpath有效，对编译无效。如JDBC。

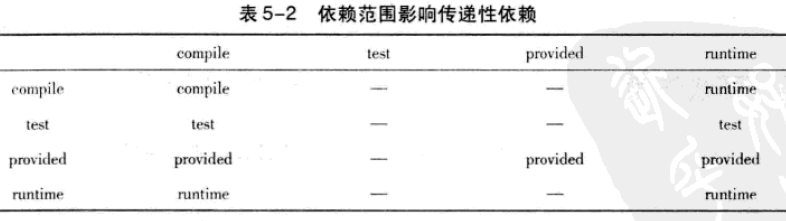
system：系统依赖范围，与provided一致。但必须通过systemPath显示知道依赖文件路径。



optional：依赖是否可选。



exclusion：排除传递性依赖。



依赖调解第一原则：路径最近者优先。

依赖调解第二原则：第一声明者优先。

#### 排除依赖

<dependency>

<groupdId></groupId>

<artifactId></artifactId>

<version></version>

<exclusions>

<exclusion>

<groupId></groudId>

<artifactId></artifactId>

</exclusion>

</exclusions>

</dependency>

#### 归类依赖

<properties>

<springframework.version>2.5.6</ springframework.version >

</properties>

${ springframework.version }

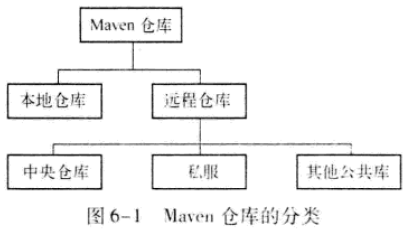
#### 优化依赖

mvn dependency:list：查看当前项目已解析依赖；

mvn dependency:tree：查看当前项目的依赖树；

mvn dependency:analyze：分析当前项目的依赖关系。

### 仓库



#### 配置远端仓库

pom.xml：

<project>

<repositories>

<repository>

<id>jboss</id> //id为center，则会覆盖中央仓库

<name>JBossRepository</name>

<url>[http://repository.jboss.com/maven2/</url](http://repository.jboss.com/maven2/%3c/url)>

<releases> //开启仓库的发布版本下载支持

<enabled>true</enabled>

</releases>

<snapshots>

<updatePolicy>daily</updatePolicy>//远程仓库检查更新频率

<enabled>false</enabled> //关闭仓库快照版本的下载支持

<checksumPolicy>ignore</checksumPolicy>//检查校验文件

</snapshots>

<layout>default</layout> //采用maven2及maven3的默认布局

</repository>

</repositories>

</project>

#### 部署至远程仓库

<project>

<distributionManagement>

<repository>

<id>proj-release</id>

<name>Proj Release Repository</name>

<url>http://192.168.1.100/content/repositories/proj-releases/<url>

</repository>

<snapshotRepository>

<id>proj-snapshots</id>

<name>Proj Snapshot Repository</name>

<url>http://192.168.1.100/content/repositories/proj-snapshots</url>

</snapshotRepository>

</distributionManagement>

</project>

mvn clean deploy输出到远程仓库

### 生命周期

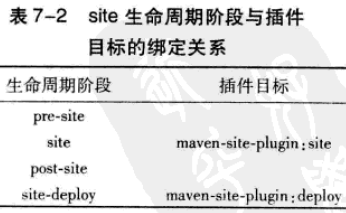
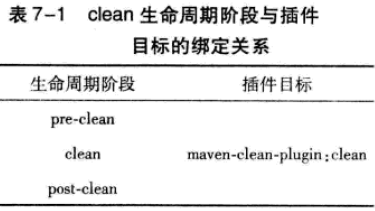
#### 三个生命周期

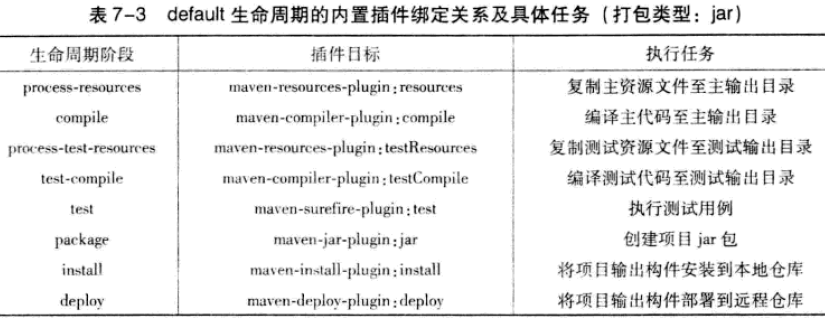
clean清理项目：pre-clean，clean，post-clean。

default构建项目：validate，initialize，generate-sources，process-sources，process-resources，compile，process-classes，generate-test-sources，process-test-sources，generate-test-resources，process-test-resources，test-compile，process-test-classes，test，prepare-package，pre-integration-test，integration-test，post-integration-test，verify，install，deploy。

site建立和发布项目站点：pre-site，site，post-site，site-deploy。

#### 插件与生命周期绑定





#### 自定义绑定

将maven-source-plugin的目标jar-no-fork绑定到verify阶段。

<build>

<plugins>

<plugin>

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

<artifactId>maven-source-plugin</artifactId>

<version>2.1.1</version>

<executions>

<execution>

<id>attach-source</id>

<phase>verify</phase>

<goals>

<goal>jar-no-fork</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

#### 插件配置

从命令行配置:mvn install -Dmaven.test.skip=true

在POM中通过在<plugin>标签下的<configuration>进行全局配置

<build>

<plugins>

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

在POM中通过在<execution>标签下的<configuration>进行任务配置

<build>

<plugins>

<plugin>

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

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

<version>1.3</version>

<executions>

<execution>

<id>ant-validate</id>

<phase>validate</phase>

<goals>

<goal>run</goal>

</goals>

<configurtion>

<tasks>

<echo>I’m bound to validate phase</echo>

</tasks>

</configuration>

</execution>

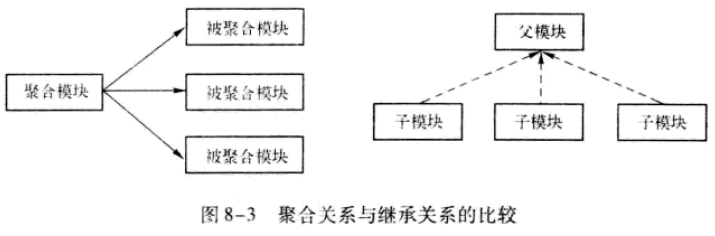
</executions>

</plugin>

</plugins>

</build>

### 聚合与继承



聚合：

1. <packaging>为pom；
2. <modules> <module>中包含各子项目id；
3. 没有src/main/java，src/test/java目录。

继承：

1. 子模块配置父模块

<parent>

<groupId></groupId>

<artifactId></artifactId>

<version></version>

<relativePath></relativePath>

</parent>

1. 父模块配置依赖

<dependencyManagement>

<dependencies>

<dependency>

<groupId></groupId>

<artifactId></artifactId>

<version></version>

</dependency>

</dependencies>

</dependencyManagement>

1. 父模块配置插件

<pluginManagement>

</pluginManagement>

1. 约定优于配置
2. 反应堆

反应堆的构建顺序：先构建模块依赖，再构建模块

### maven测试

maven-surefire-plugin默认路径为/src/test/java，执行以文件名Test开头和结尾或TestCase结尾的文件

mvn package –DskipTests：跳过测试

配置插件跳过测试：

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<version>2.5</version>

<configuration>

<skipTests>true</skipTests>

</configuration>

</plugin>

mvn test –Dtest=RandomGeneratorTest：测试某个用例

加入或排除测试用例：

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<version>2.5</version>

<configuration>

<includes>

<include>\*\*/\*Tests.java</include>

</includes>

<excludes>

<exclude>\*\*/\*ServiceTest.java</exclude>

</excludes>

</configuration>

</plugin>

在target/surefire-reports目录下生成两种格式的测试报告：

简单文本txt和与JUnit兼容的XML

打包测试：

<plugin>

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

<artifactId>maven-jar-plugin</artifactId>

<version>2.2</version>

<executions>

<execution>

<goals>

<goal>test-jar</goal>

</goals>

</execution>

</executions>

</plugin>

<dependency>

<groupId>com.juvenxu.mvn.account</groupId>

<artifactId>account-captcha</artifactId>

<version>1.0.0-SNAPSHOT</version>

<type>test-jar</type>

<scope>test</scope>

</dependency>

## 问题

**问题1：Compliation failue source1.3不支持注释（请使用-source5或更高版本）**

compiler插件默认只支持编译Java1.3，需要配置该插件使其支持Java5

<build>

<plugins>

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.5</source>

<target>1.5</target>

</configuration>

</plugin>

</plugins>

</build>

**问题2：生成可执行的jar文件**

<build>

<plugins>

<plugin>

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

<artifactId>maven-shade-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>com.juvenxu.mvnbook.helloworld.HelloWorld</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

**问题3：解决M2Eclipse的“Plugin execution not covered by lifecycle configuration”**

<http://wiki.eclipse.org/M2E_plugin_execution_not_covered>

<http://stackoverflow.com/questions/7391201/plugin-error-execution-not-covered-by-lifecycle-configuration>

在<build>标签内添加如下内容：

<pluginManagement>

<plugins>

<plugin>

<groupId>org.eclipse.m2e</groupId>

<artifactId>lifecycle-mapping</artifactId>

<version>1.0.0</version>

<configuration>

<lifecycleMappingMetadata>

<pluginExecutions>

<pluginExecution>

<pluginExecutionFilter>

<groupId>

org.codehaus.mojo

</groupId>

<artifactId>

findbugs-maven-plugin

</artifactId>

<versionRange>

[2.5.2,)

</versionRange>

<goals>

<goal>check</goal>

</goals>

</pluginExecutionFilter>

<action>

<ignore></ignore>

</action>

</pluginExecution>

</pluginExecutions>

</lifecycleMappingMetadata>

</configuration>

</plugin>

</plugins>

</pluginManagement>

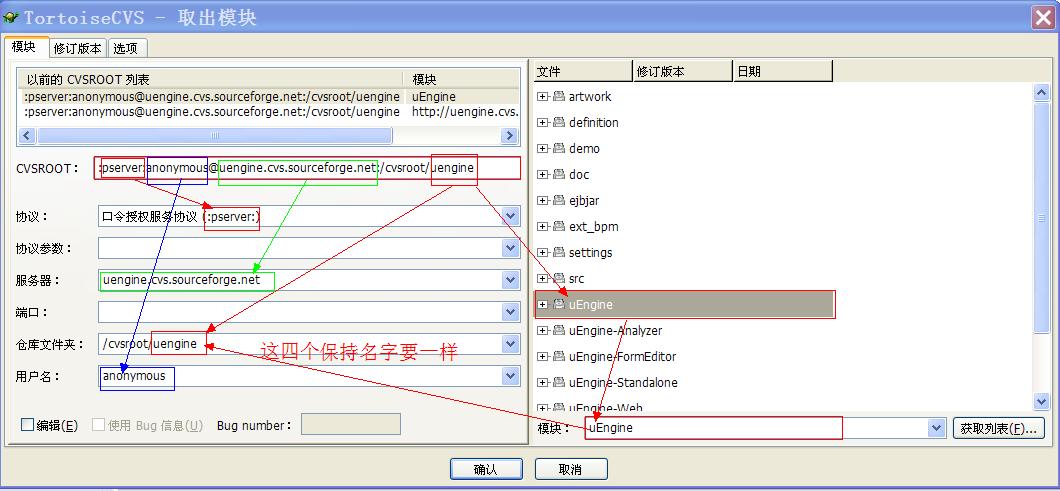
# CVS

WIN 7:

下载TortoiseCVS-1.12.5.exe，双击安装

新建目录（如cvs），在目录上单击右键，在弹出菜单选CVS Checkout，

以pserver:anonymous@uengine.cvs.sourceforge.net:/cvsroot/uengine 为例,粘贴到CVSROOT中，就会自动出现协议，服务器，仓库文件夹，用户名等。



CVSROOT这段代码在sourceforge.net网站的CVS里面可以找到，如果没有，那么应该会告诉你 repository （仓库），用户名，服务器等，可以把这些放到图二中对应的字段中，cvs会字段生成CVSROOT这段代码。最关键是要选择正确的模块module，在图二中已经表明，这个模块名在四处要保持一样，不然就会报错。

如果不知道这个模块名字，那么可以输入一个地址来获得这个列表，我这里的地址是<http://uengine.cvs.sourceforge.net/uengine/>。把这个地址粘贴到模块后面的空白处，然后点击获取列表，一会后就会出现图二中右边的这个区域，这个就是获取的列表。

确定下载。全部设置完成后，就点击确定，CVS就开始下载数据了，等待数据下载完，就完成了整个过程。

或者直接在cmd中运行命令：

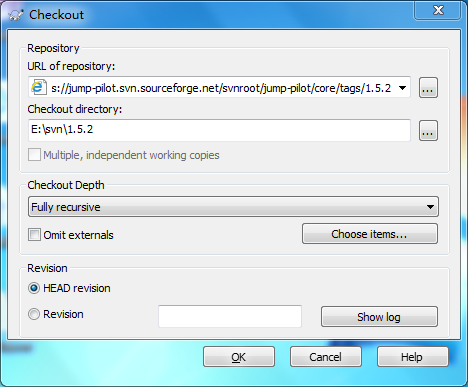
cvs -z3 -d:pserver:anonymous@mymaps.cvs.sourceforge.net:/cvsroot/mymaps co -P myMaps

# SVN

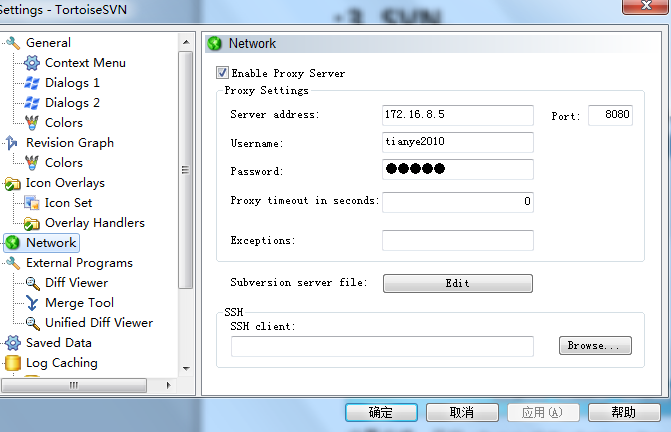
WIN 7:

下载TortoiseSVN-1.12.5.exe，双击安装

新建空文件夹（如svn），在目录上单击右键，在弹出菜单选SVN Checkout，repository自动填入复制内容



设置代理，开始->TortoiseSVN->Setting->Network，填入代理服务器地址，端口号，用户名，密码。



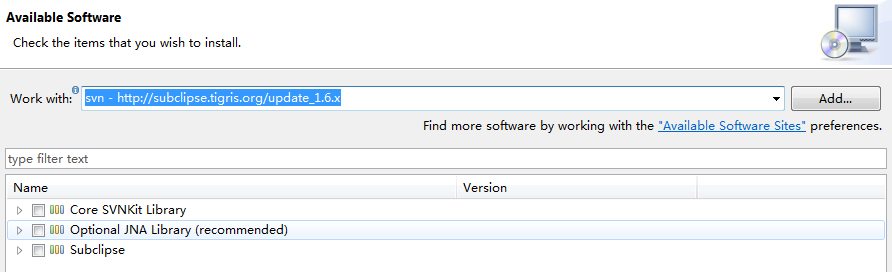
## Eclipse安装svn插件

Help->Install new software->Add

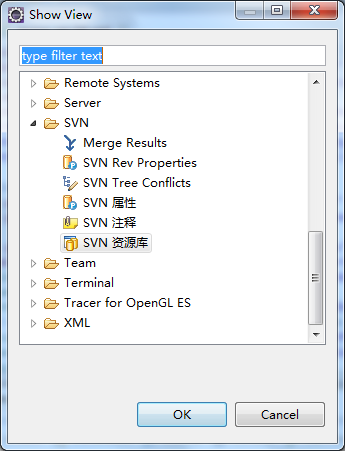
name:svn

<url:http://subclipse.tigris.org/update_1.6.x>

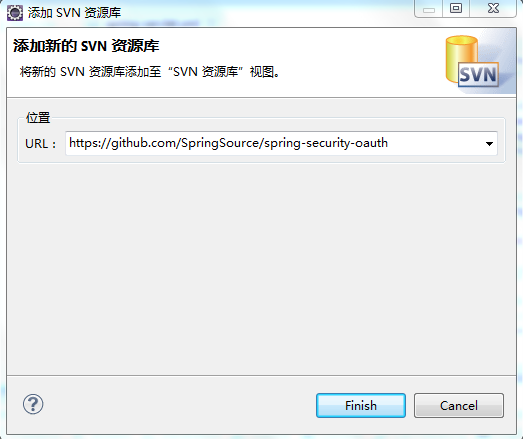
选中后Next，Install，重启eclipse

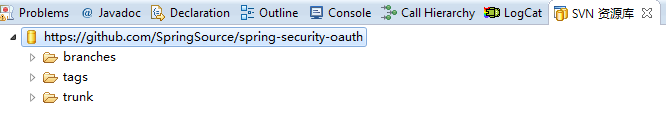


Window->Show View->Other打开资源库视图

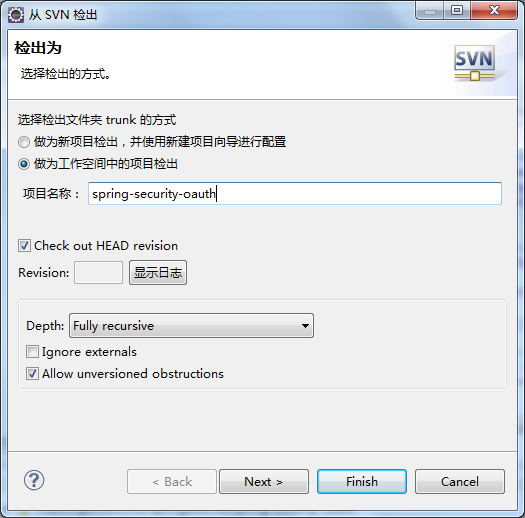


在视图中右键新建->资源库位置，输入网址提供的subversion的URL，Finish





在trunk上右键->检出为，设置检出项目的名称和位置，Finish

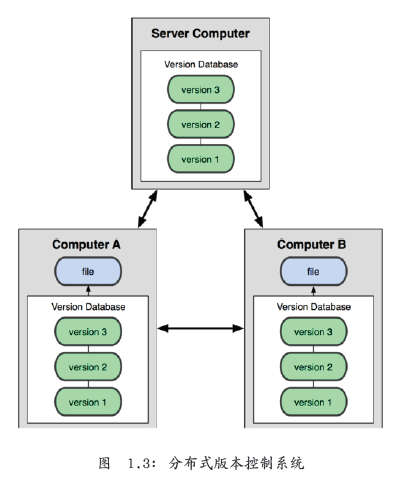
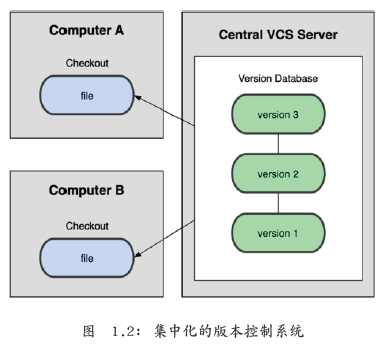


# GIT

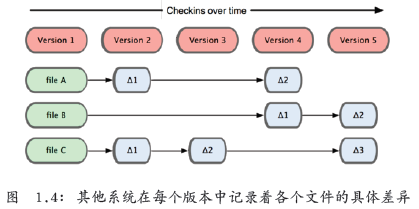
<http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

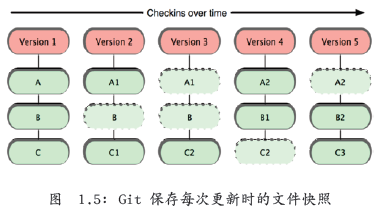
## 概述

相对于CVS，Subversion，Perforce等软件采用集中化版本控制，存在中央服务器的单点故障，容易丢失数据的缺点，GIT采用分布式版本控制。

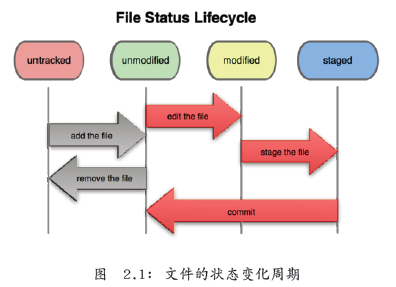
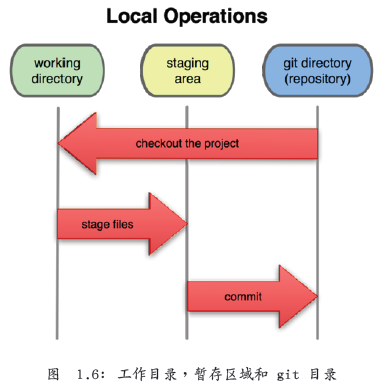


CVS等软件记录哪些文件作更新和更新哪些行的什么内容，Git把变化文件作快照，记录在微型文件系统中。





Git三种状态和文件状态生命周期



## 命令

git init

git add

git commit

git status

git diff

git log

git reflog

git reset

git checkout

git branch

git merge

git tag

git remote add origin [git@github.com:michaelliao/learngit.git](mailto:git@github.com:michaelliao/learngit.git)

git push -u origin master

git clone [git@github.com:michaelliao/gitskills.git](mailto:git@github.com:michaelliao/gitskills.git)

## github项目

在github上创建项目

git clone [https://github.com/ yourname / yourrepo.git](https://github.com/dongwenting/oa.git)

进入项目

git remote set-url origin git@github.com:yourname/yourrepo.git

ssh-keygen -t rsa

.ssh/ id\_rsa.pub 拷贝到github.com->setting->SSH keys中

git push试试

## 搭建git服务器

服务器端：

sudo apt-get install git

sudo adduser git

su - git

mkdir .ssh

chmod 700 .ssh

vi .ssh/authorized\_keys

将上传源码的公钥拷贝进去

chmod 600 .ssh/authorized\_keys

sudo git init --bare sample.git

sudo chown -R git:git sample.git

exit

cat /etc/shells

which git-shell

sudo vi /etc/shells

/usr/bin/git-shell

sudo vi /etc/passwd

git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell

或者

sudo chsh git

/usr/bin/git-shell

客户端：

mkdir sample

cd sample

git init

touch readme.md

git add .

git commit -m “test”

git remote add origin [git@192.168.1.3:sample.git](mailto:git@192.168.1.3:sample.git)

git push origin master

git clone [git@192.168.1.3:sample.git](mailto:git@192.168.1.3:sample.git)

cd sample

touch test

git add .

git commit -m “test”

git push

## 仓库迁移

git用户登录服务器

mkdir ~/mcm.git

cd ~/mcm.git

git init --bare

源代码所在项目

vi .git/config

[remote "origin"]

url = git@192.168.1.3:mcm.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

git push

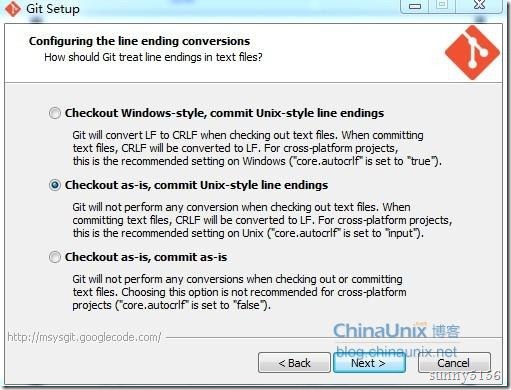
## TortoiseGit

### 安装

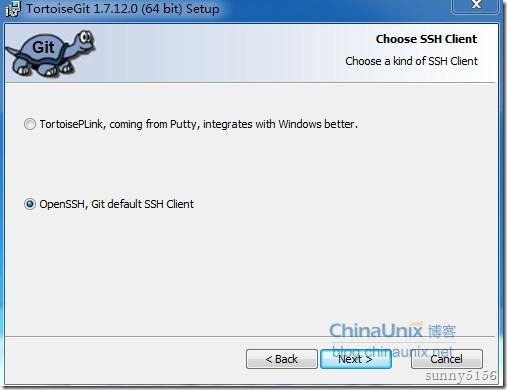
WIN 7:

安装TortoiseGit前须先安装msysGit.

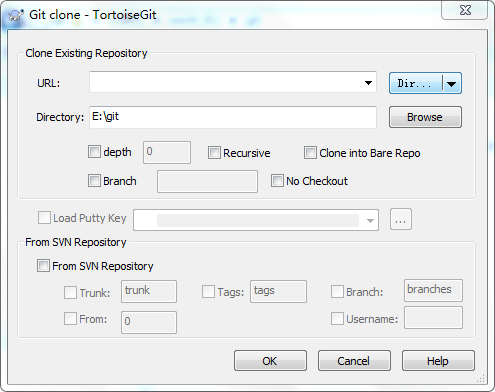
下载Git-1.8.0-preview20121022.exe，双击安装

[](http://blog.chinaunix.net/attachment/201208/20/25806493_134544868979mf.jpg)

下载TortoiseGit-1.7.15.0-32bit.msi,双击安装,选SSH

[](http://blog.chinaunix.net/attachment/201208/20/25806493_1345448734Iw1N.jpg)

新建目录（如git），右键弹出菜单，填入路径

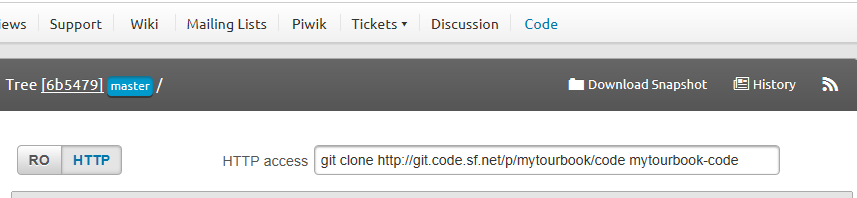


<http://www.haogongju.net/art/1691138>

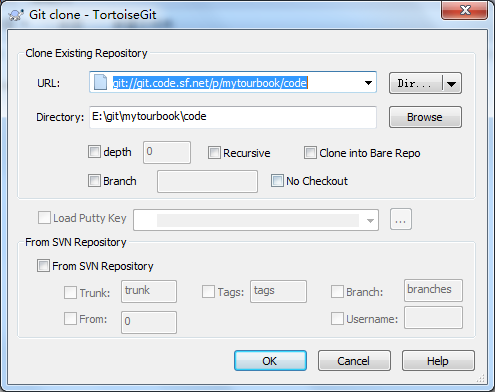
### 使用

#### 下载开源软件

选择某一开源软件如mytourbook，code中获得git地址

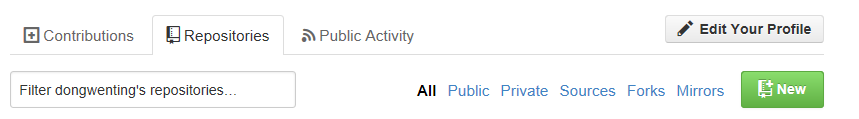


新建文件夹，右键Git Clone，填入地址，注意前面没有的git clone和后面的mytourbook-code

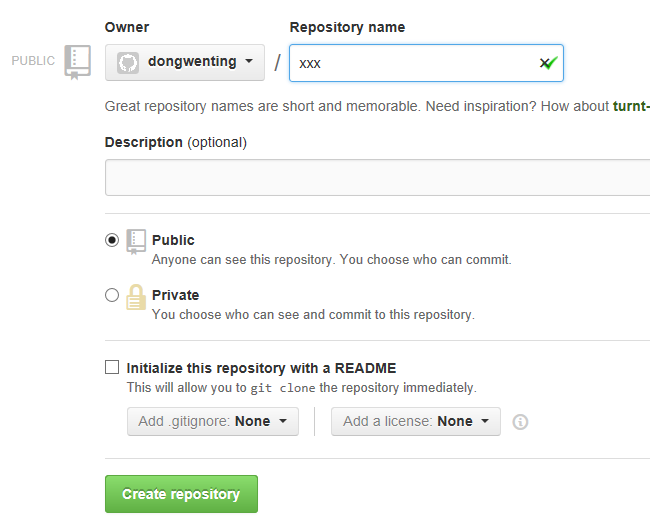


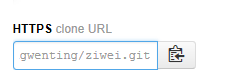
#### 建立自己的git

在<https://github.com>中注册帐号，Repositories->New



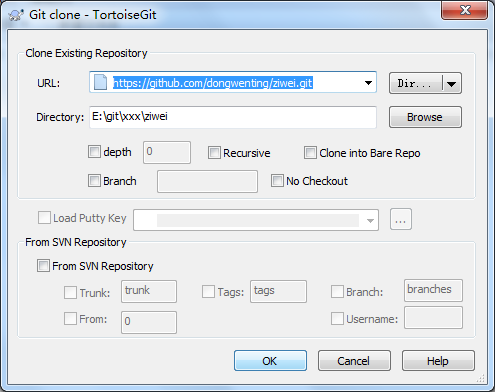
填入项目名称，选择是否公开



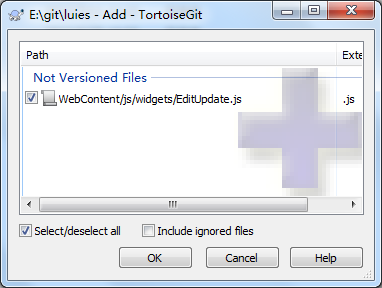
右侧小框中有git路径

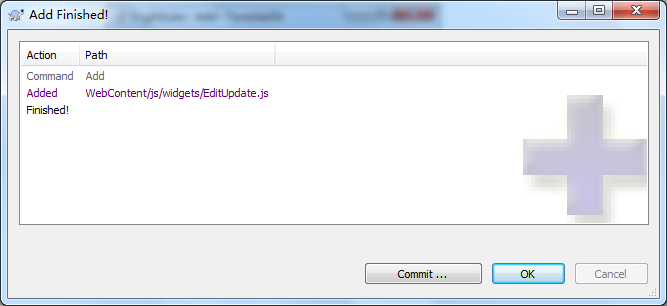
先要下载代码，再修改代码，上传

新建目录，右键Git Clone，填入git路径，将代码下载下来。

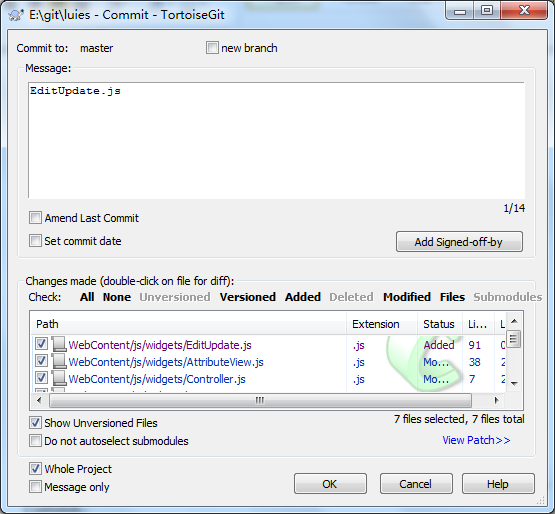


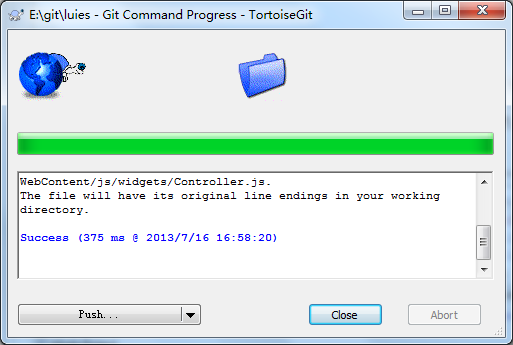
修改代码后，右键->TortoiseGit->Add，勾选->OK



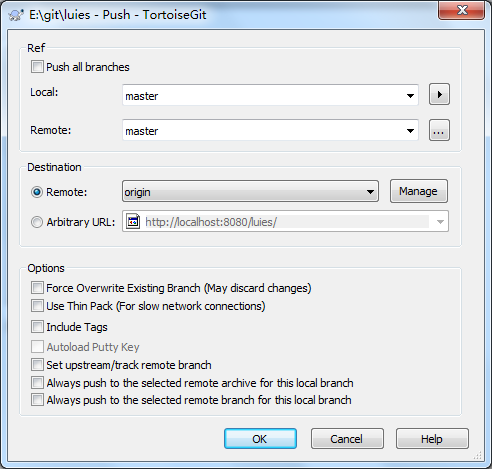


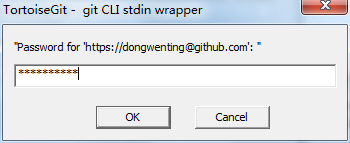
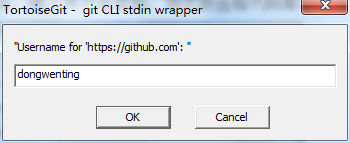
Commit，填入备注，同步到本地代码库中





Push，输入用户名密码，同步到远程代码库中





## 问题

**问题1：github remote permission to git denied to**

打开项目/.git/config文件，不要用https用ssh

[core]

repositoryformatversion = 0

filemode = false

bare = false

logallrefupdates = true

symlinks = false

ignorecase = true

hideDotFiles = dotGitOnly

[remote "origin"]

**url = git@github.com:sklois/sdn.git**

fetch = +refs/heads/\*:refs/remotes/origin/\*

[branch "master"]

remote = origin

merge = refs/heads/master

**问题2：恢复已删除的文件**

git log --diff-filter=D --summary 找到删除文件的commit id

git checkout commit-id~1 恢复到删除前的一个版本

git checkout commit-id~1 path/file 只恢复单个文件