Development Tools

Content

- Project management and build, Maven
- Version control, Git
- Code coverage, JaCoCo
- Profiling, NetBeans
- Static Analyzer, NetBeans
- Continuous integration, Hudson

Project Management and Build With Maven

Maven Home page:
http://maven.apache.org/

Tutorial, user guide and reference manual:
http://maven.apache.org/guides/index.html

Maven, Content

- Why do we need a build tool? What is a project management tool?
- Maven architecture
- Pom file
- Using Maven in NetBeans

Why do we need a build tool?

- All actions (compile, deploy, run, create documentation, etc.) must be well-defined and reproducible.
  - We need to store commands, switches, command line arguments, environment variables (like classpath), etc.
- All IDEs use a build tool.
  - Configured via IDE dialogs instead of editing the build tool's script text files.
- NetBeans, and many other IDEs, use Ant for building.

Maven vs Ant

- An ant file is like a program, you specify what to do, when to do it and where (in which directory) the used files are found.
- With maven you only specify *what* to do, e.g., compile, not *how* to do it.
  - Many common tasks, like compiling, are done by default, you do not even need to specify them.
- As a result, a maven script becomes shorter and easier to understand than an ant script.
- In particular, the NetBeans ant scripts are *very* long and complex.

Maven vs Ant (Cont'd)

- Ant forces you to manage all files yourself.
  - Manually download all third-party jars your code depends on and place them in correct directory.
  - Often very cumbersome and time-consuming.
- Maven defines your project's directory structure and manages all files in the project.
  - Just specify the dependencies and maven downloads needed jars and uses them as required.

What is a project management tool?

- Maven not only builds the project.
- Also defines project directory structure, which tasks to perform and in what order.

Maven Philosophy

- Maven defines project directory structure.
  - Always the same, no configuration needed.
- Maven defines what to do and in which order.
  - Always the same, no configuration needed.
- User only defines a unique project name, package format (jar, war...) and dependencies (third-party products used).
  - Much more configuration is of course possible.

Lifecycles, Phases and Goals

- Maven projects consist of lifecycles, which are divided in phases, which are divided in goals.
- There are three lifecycles: default, clean and site.
  - Default lifecycle creates the application.
  - Clean lifecycle removes all files generated by maven.
  - Site lifecycle generates a web site with project documentation.

Major Phases in Default Lifecycle

- process-resources, Copy resources directory into destination directory.
- compile, Compile the source code.
- process-test-resources, Copy resources directory into the test destination directory.
- test-compile, Compile the test source code.
- test, Run tests using a suitable unit testing framework.
- package, Package compiled code in JAR (or other).
- install, Install the package into the local repository.
- deploy, Copy the package to the remote repository.

Clean and Site lifecycle

- These lifecycles have only one important phase each.
  - Clean lifecycle has clean phase.
  - Site lifecycle has site phase.

Execution

- To run maven, you specify a phase.
- Phases in the lifecycle of the specified phase are executed in order.
- The lifecycle starts from the beginning and stops after the specified phase.

What Happens in Each Phase?

- Each phase executes a set of goals.
- The packaging type decides which goals belong to which phase.
  - Additional goals can be added manually.
- A goal is actually a piece of Java code, which is packaged in a plugin.
  - One plugin can define many goals

Typical Goals

The jar and war packaging types use (at least) the following goals.

Phase	**`plugin:goal`**
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar or war:war
install	install:install
deploy	deploy:deploy

Packages are Stored in Repositories

- Maven stores all packaged products (mainly jars) in repositories.
  - Enables Maven to handle all dependencies since needed jars can be downloaded from repositories where they are stored.
  - Promotes code reuse since local repositories can be used to share jars between projects.
- There is a default central repository and a local repository will be created.
  - Other repositories must be specified as a dependency.

Project Object Model

- A Maven project is described in a **P**roject **O**bject **M**odel file, `pom.xml`.
- The pom describes, at least, a unique name for the product and which dependencies it has (that is, which third-party products it uses).
- All POMs inherit a parent POM. If the parent is not specified it inherits the default POM.
  - The used POM, including inherited data is shown with the command `mvn help:effective-pom`
  - In NetBeans, open the POM and click **`Effective.`**

POM Generated by NetBeans

- `groupId`, `artifactId` and `version` together defines a unique name for the created archive.
- `packaging` decides the archive format for the product.
- `name` is the project's display name. It is used mainly in generated documentation.

```
<groupId>se.kth.iv1201</groupId>

<artifactId>MyProject</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<name>MyProject</name>
```

POM Generated by NetBeans (Cont'd)

- A `dependency` defines a package our product uses.
- `scope` decides when the included package should be available. Default is `compile`, which means it is always available. Here we have specified `provided`, which means it will not be available at runtime (since it is provided by the server).

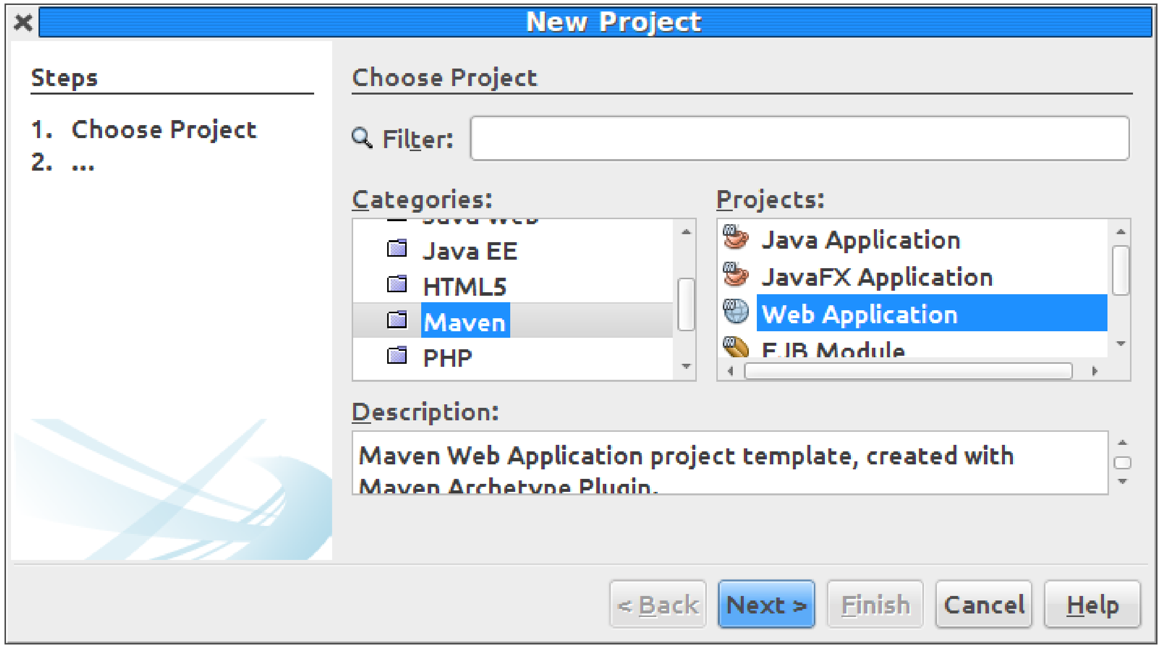
```
<dependencies>
  <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
  </dependency>
</dependencies>
```

POM Generated by NetBeans (Cont'd)

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
    <configuration>
      <source>1.7</source>
      <target>1.7</target>
    </configuration>
  </plugin>
</plugins>
```

- It is not required to specify plugins. Here, the compiler plugin is specified because we want to configure it (to use JDK1.7 for Java files and class files).

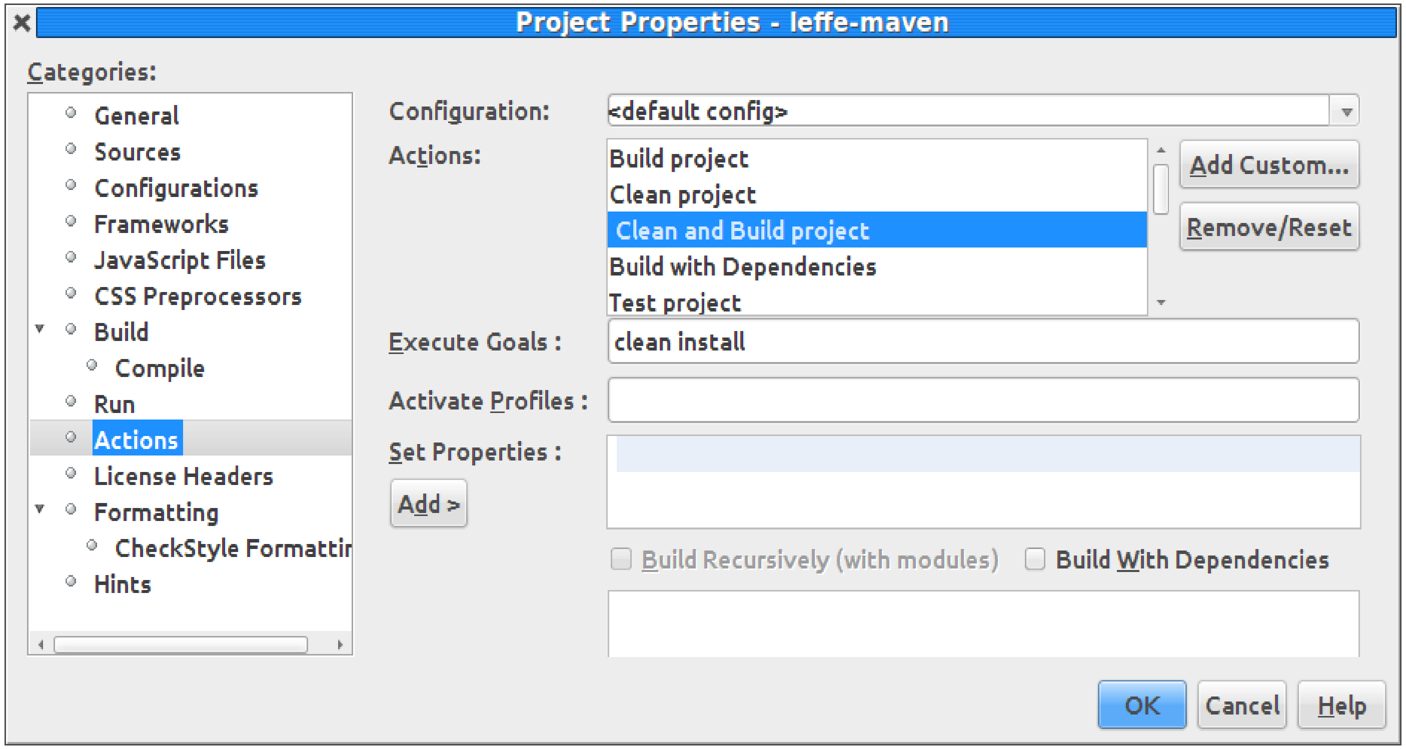
Using Maven in NetBeans



Maven Web Application project template, created with Maven Archetype Plugin.

- To create a Maven project you should choose Maven in the `Categories` list.
- Choose correct type of project in the `Projects` list.

Using Maven in NetBeans (Cont'd)



Project Properties - leffe-maven

Categories:
- General
- Sources
- Configurations
- Frameworks
- JavaScript Files
- CSS Preprocessors
- Build
  - Compile
- Run
- Actions
- License Headers
- Formatting
  - CheckStyle Formattir
- Hints

Configuration: <default config>

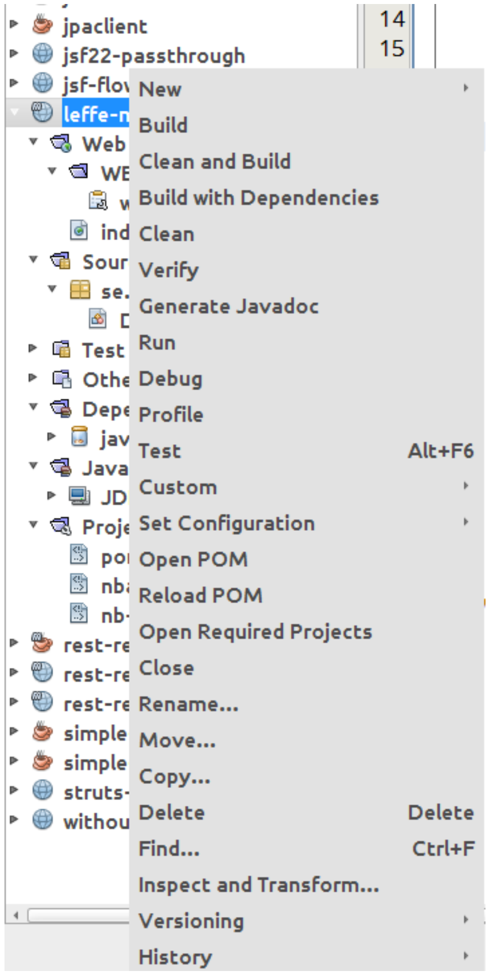
Actions:
- Build project
- Clean project
- Clean and Build project
- Build with Dependencies
- Test project

Execute Goals : clean install

Activate Profiles :

Set Properties :

Build Recursively (with modules) [ ] Build With Dependencies [ ]



- You can update the POM by editing project properties.
- In particular, The **`Actions`** category allows you to configure the items on the right-click menu of the project.
- It is also possible to manually edit the POM.

Version Control With Git

Git Home page:
http://gitscm.com/

Tutorial, book and reference manual:
http://git-scm.com/documentation

Git, Content

- Why do we need version control?
- Git architecture
- Important commands
- Using Git in NetBeans

Why Do We Need a Version Control System (VCS)?

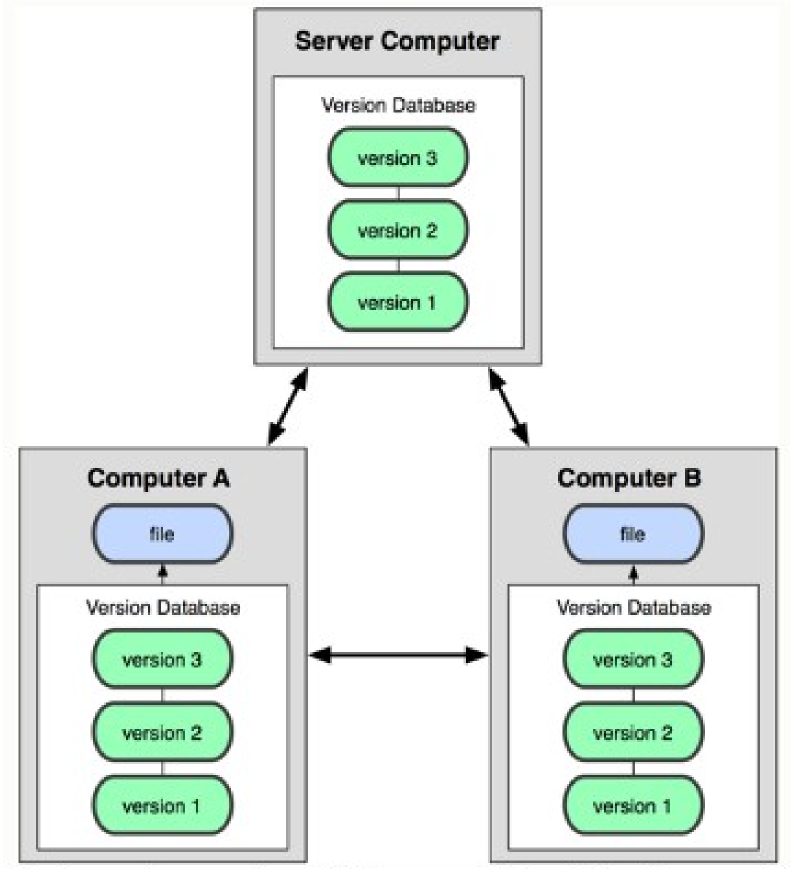
- An unlimited number of people may edit the same files at the same time.
  - The VCS records concurrent edits and helps resolve conflicts.
- No extra work to share or upload files.
  - Files are committed to a shared repository.
- Revert to previous versions if something fails.
  - The VCS stores a history with all states of all files.
- Stable versions can be tagged.
  - A particular snapshot of the entire repository can be named.
- Different branches of the same code base can be maintained without duplicated code.
  - Shared parts of files are not duplicated.

Different Version Control Systems

- CVS (1986) was the system that made version control popular.
  - Still used, quite easy to learn, has some serious drawbacks.
- Subversion, SVN (2000) is probably the most used system.
  - Used by for example SourceForge, Apache, Google Code.
- Git (2005) is becoming rapidly more adopted.
  - Faster than CVS and SVN, no central repository.
- Mercurial, Bazaar and Monotone are other examples.
  - All are distributed, like Git but unlike CVS and SVN.

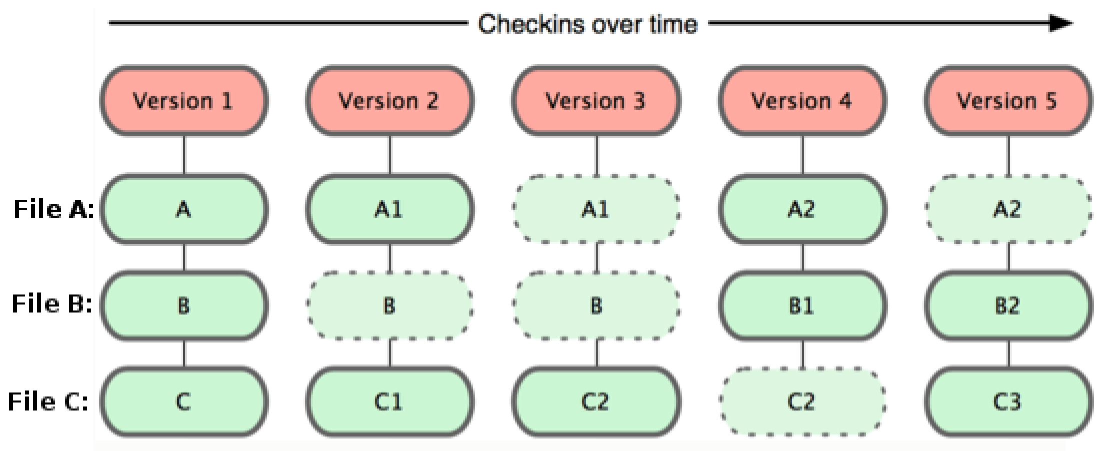
Git is a Distributed Version Control System, DVCS

- All clients fully mirror the entire repository.



Picture from Scott Chacon: Pro Git (Apress)

Git Stores Files, Not File Updates

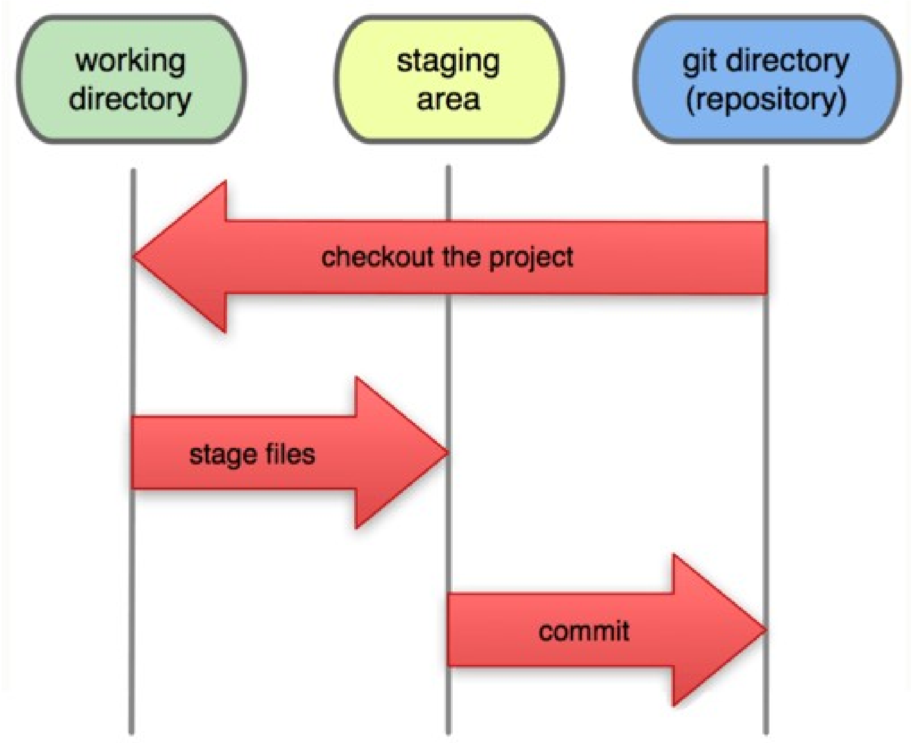


Picture from Scott Chacon: Pro Git (Apress)

- Git stores all content of all files at every commit.
  - Stores only a link to last version if a file is not updated .
- This means that nearly all operations are local and also that it is possible to work offline.
- Git almost only adds data, it very seldom deletes.
  - Nothing is lost.

Three States

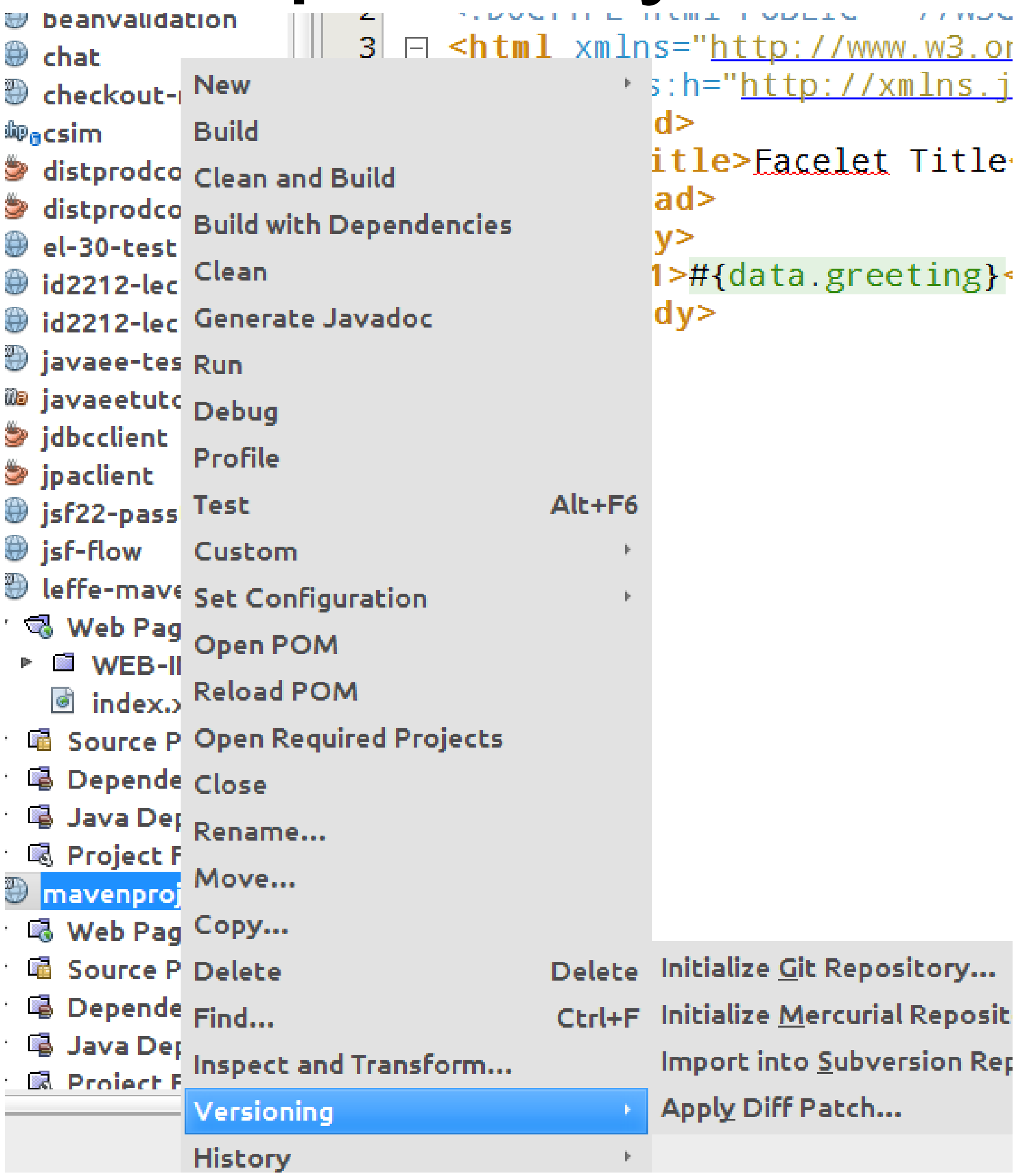
- The Git directory is the file repository, including metadata like tags and versions.
- The working directory is where you edit the files.
- The staging area is a file (in the git directory) that tells what will be in the next commit.
- The daily workflow is edit, stage, commit.
  - First you must (once) create the repository and check out files.



Picture from Scott Chacon: Pro Git (Apress)

Create a New Repository

- Without IDE, type `git init` in the project's root directory
  - Each project will have its own repository.
- Using NetBeans you right-click on the project and choose `Versioning> Initialize Git Repository…` , the proposed repository location is almost always OK.

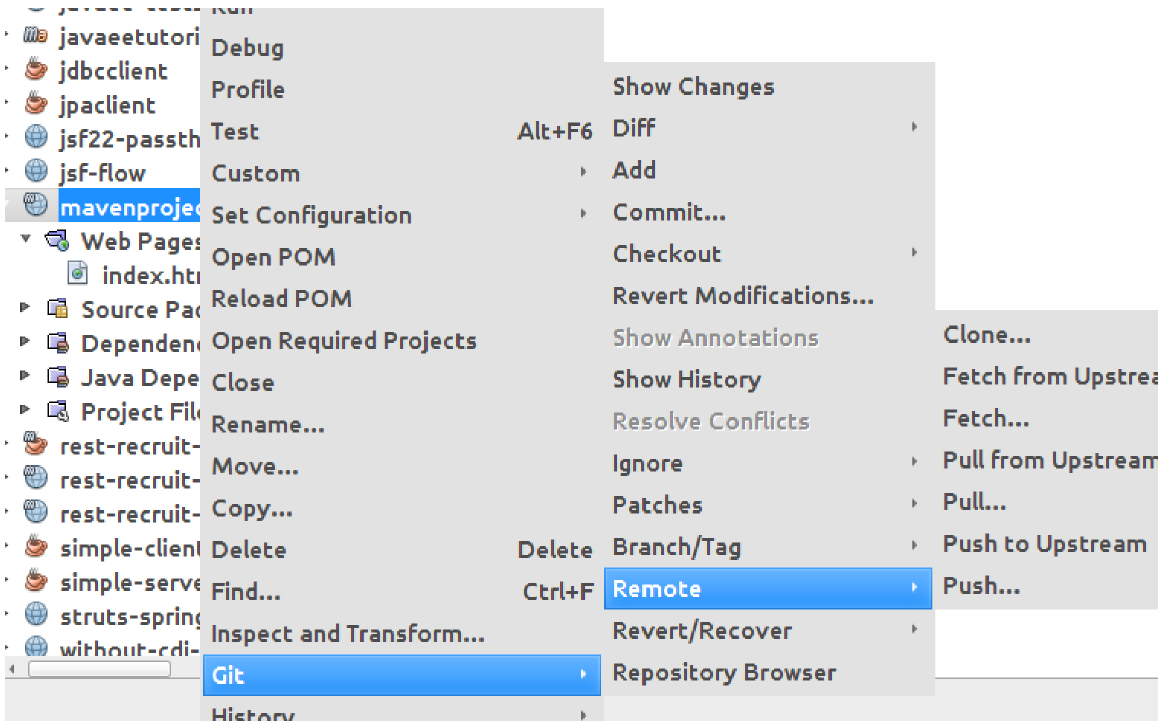


Add Files to the New Repository

- Without IDE, add files with the add command, which accepts wildcards, e.g., `git add *.java`
- In NetBeans, all files in the project are added when the Git repository is created.
  - To add files manually, right-click the file (or directory) and choose `Git> Add`.
- To commit the added files to the repository without using an IDE, type `git commit`.
- To commit in NetBeans, right-click the project and choose `Git> Commit…`.

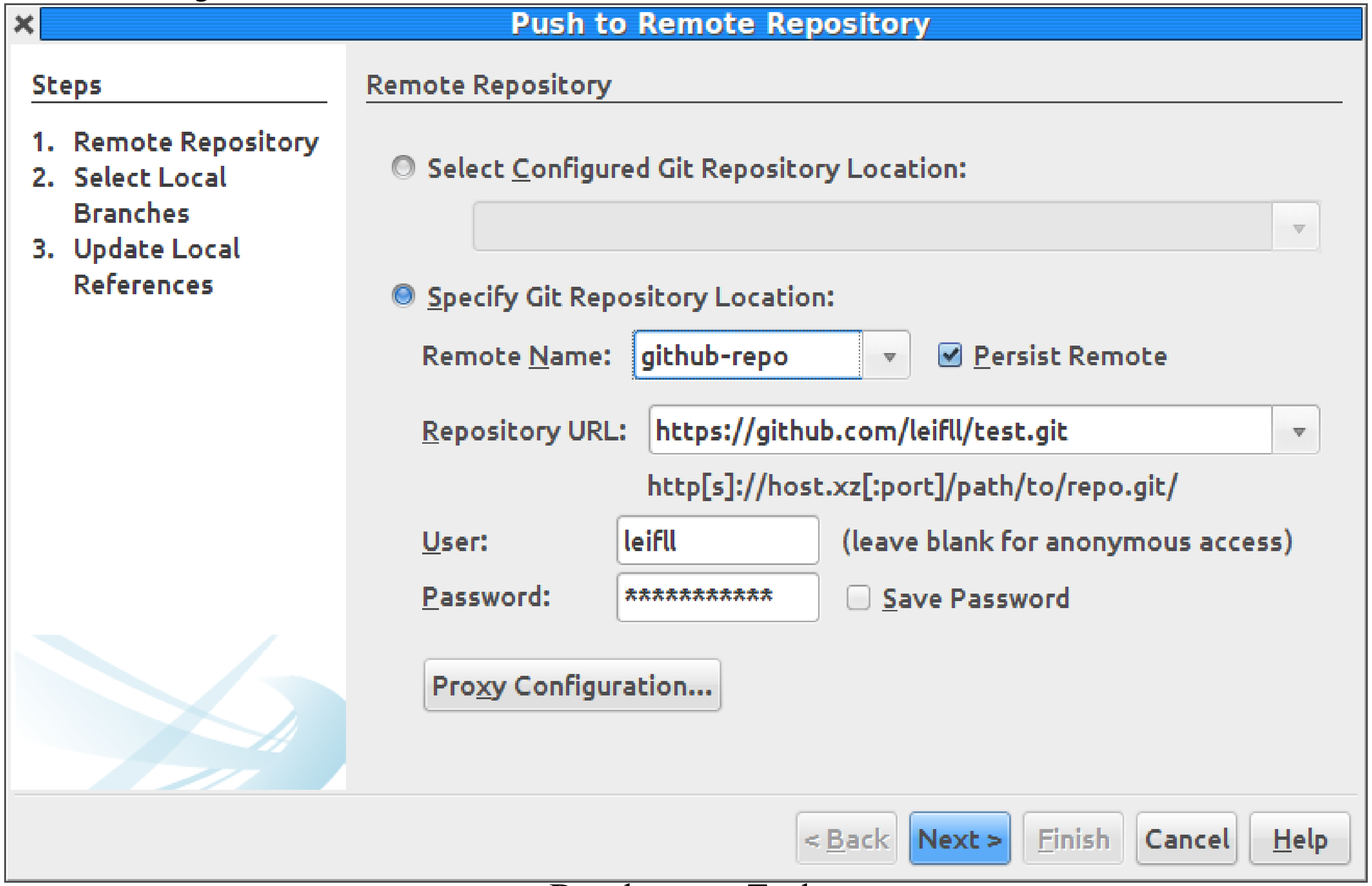
Push to a Remote Repository

- To share files you need a remote, empty, repository.
  - Can be hosted for free at for example `github.com`.
- To push to the remote directory using NetBeans, right-click the project and choose `Git> Remote> Push`



Push to a Remote Repository (Cont'd)

- Then specify the location of your remote repository.



Push to Remote Repository

Steps

1. Remote Repository
2. Select Local Branches
3. Update Local References

Remote Repository

Select Configured Git Repository Location:

[✓ Specify Git Repository Location:]

Remote Name: github-repo

Persist Remote ✓

Repository URL: https://github.com/leifll/test.git

http[s]://host.xz[:port]/path/to/repo.git/

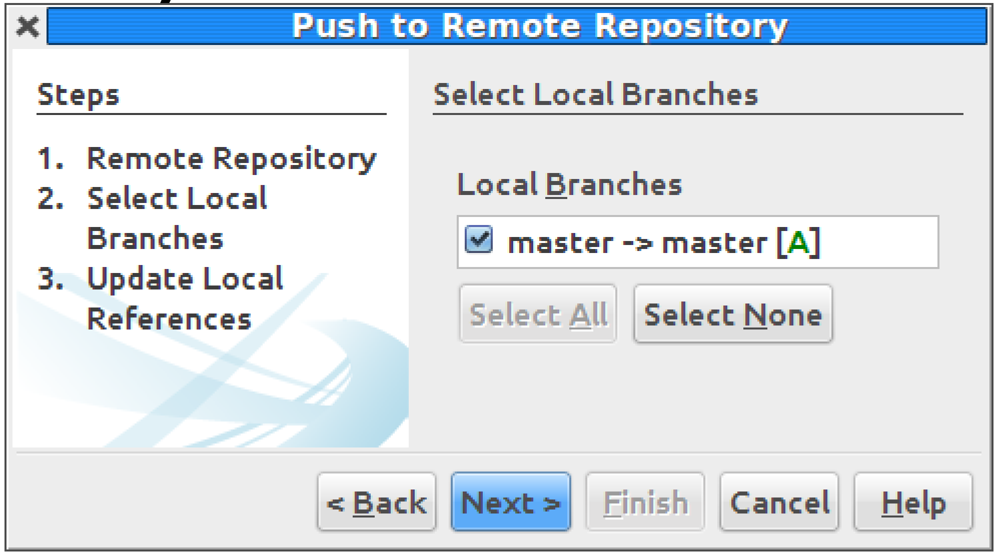
User: leifll (leave blank for anonymous access)

Password: ************

Save Password ☐

Push to a Remote Repository (Cont'd)

- Finally, specify that the local master shall be pushed to the remote master.



Push to Remote Repository

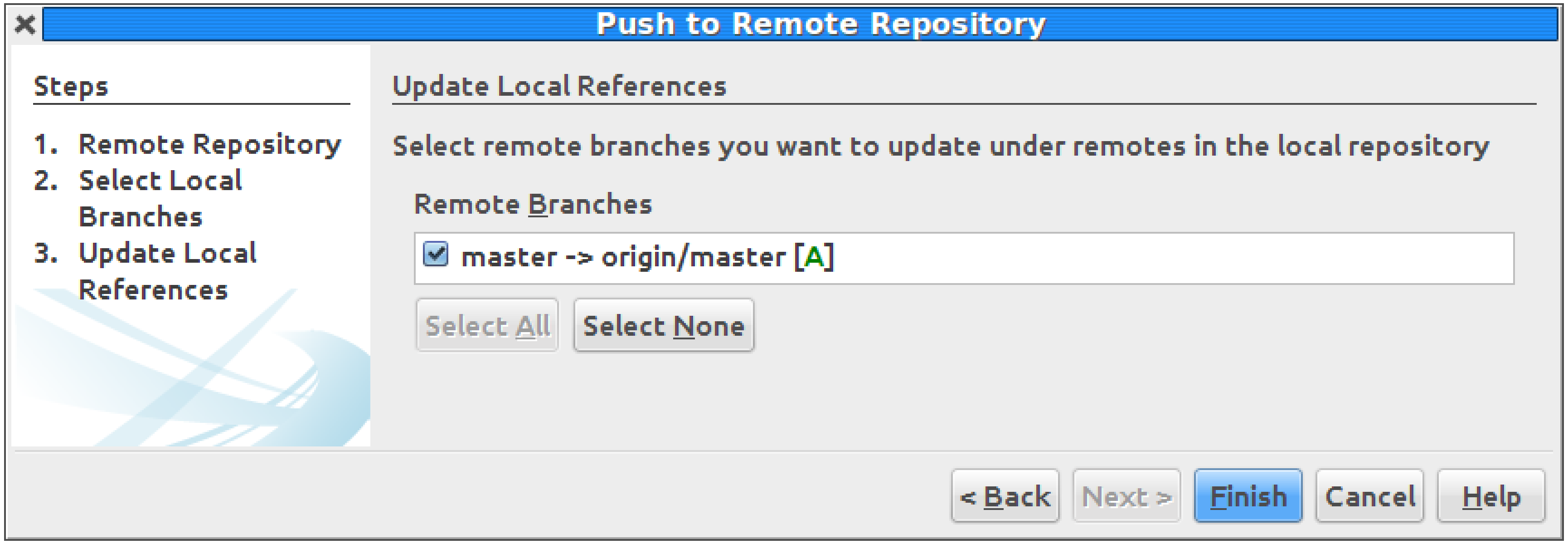
Steps

1. Remote Repository
2. Select Local Branches
3. Update Local References

Select Local Branches

Local Branches

☑ master -> master [A]



Push to Remote Repository

Steps

1. Remote Repository
2. Select Local Branches
3. Update Local References

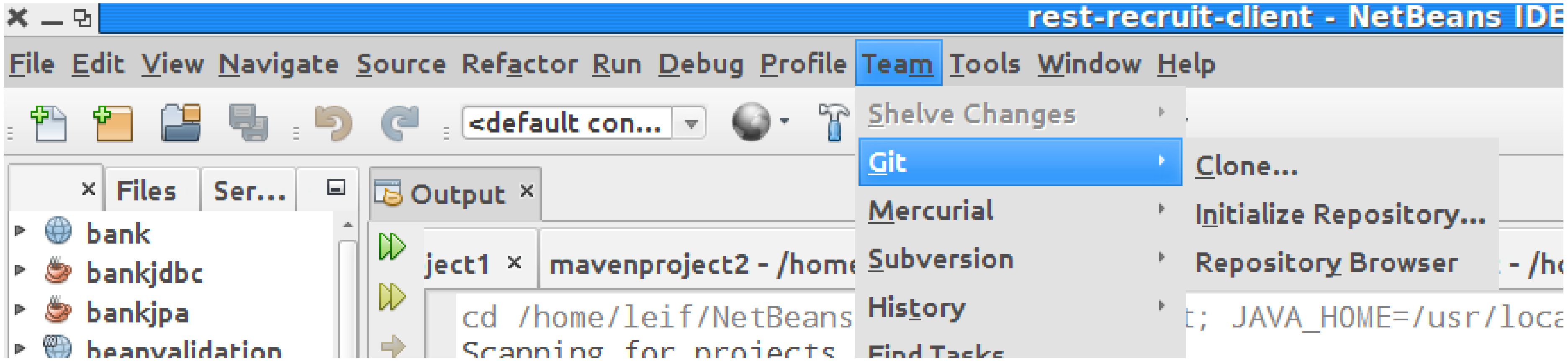
Update Local References

Select remote branches you want to update under remotes in the local repository

Remote Branches

☑ master -> origin/master [A]

Use an Existing Remote Repository



```
cd /home/leif/NetBeans
Scanning for projects
t; JAVA_HOME=/usr/loca
```

- Other team members can now download contents of the created repository.
- To do that, choose the menu item
  `Team> Git> Clone…`
  - Specify remote repository, remote branch and local directory in the dialogs that follow.

Git Daily Workflow

- Now that all team members have the same remote repository, the workflow will be as follows.
  - Edit/add/delete files.
  - Commit changes to local repository. In NetBeans, right click the project and choose `Git> Commit`. Note that this both stages and commits all changes. To do this at the command prompt type either `git commit -a` or `git add` and then `git commit`.
  - Push/pull to/from remote repository. In NetBeans, right-click the project and choose `Git> Remote> Push…`/`Pull…`.

There is Much More...

- This was only a tiny part of Git's functionality.
- Git provides a lot of help to branch, tag and merge snapshots.
- A good source for further inspiration is the Git online book at `http://git-scm.com/book`.

Code Coverage With JaCoCo

JaCoCo Home Page:
http://www.eclemma.org/jacoco/

Documentation:
http://www.eclemma.org/jacoco/trunk/doc/

JaCoCo, Content

- What is a Code Coverage Tool?
- JaCoCo Basics

What is a Code Coverage Tool?

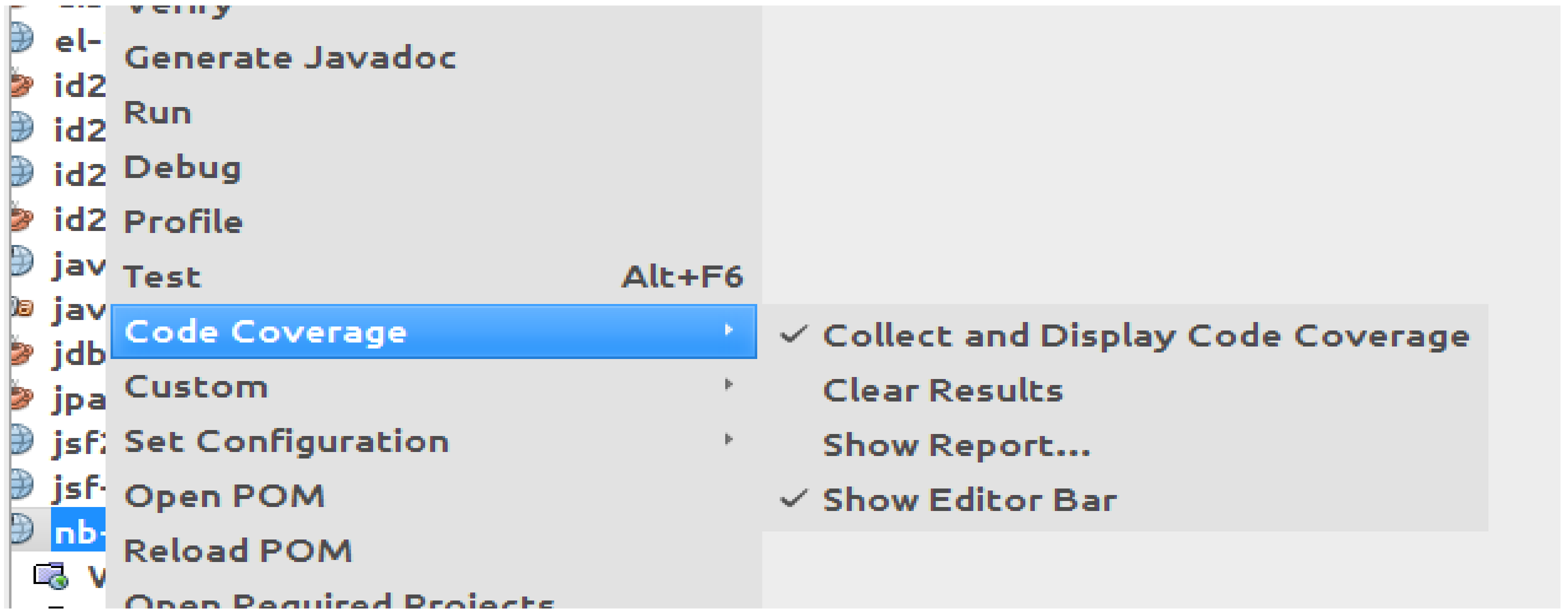
- Records which parts of the program have been executed during a test and generates a report of the coverage.
- Visualizes how complete the tests are.
- It is normally not meaningful to strive for 100% coverage, getters and setters may be omitted from the test.
- We shall, however, make sure all important parts of the code are tested.

JaCoCo Basics

Add this configuration to the POM to enable JaCoCo.

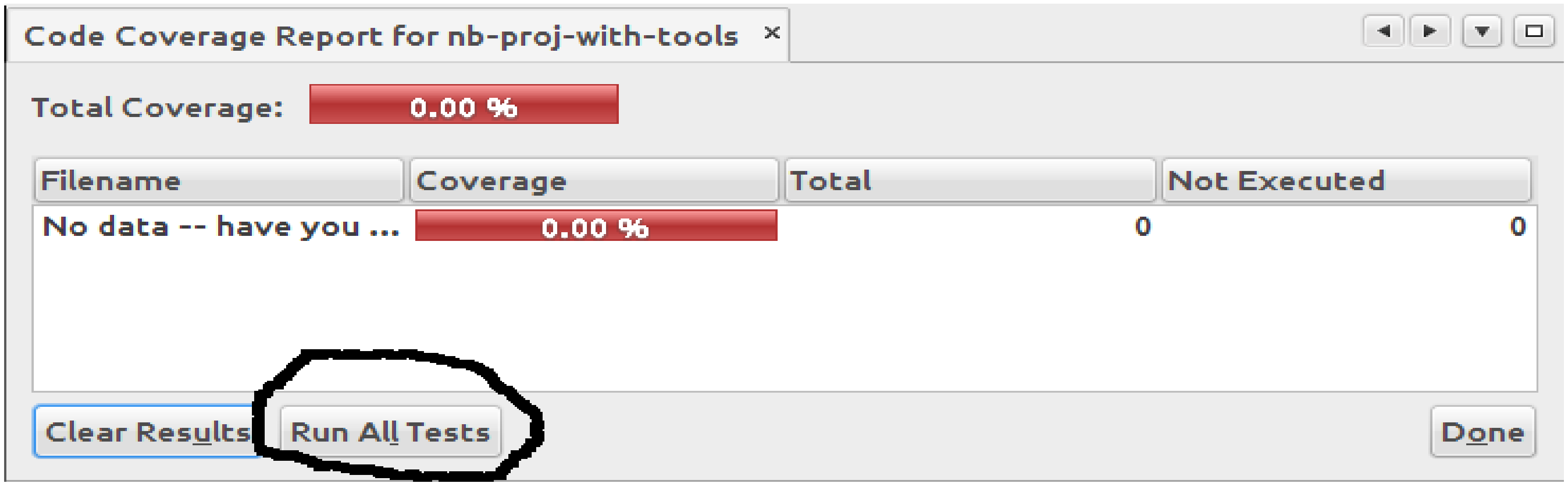
```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.6.0.201210061924</version>
  <executions>
    <execution>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>report</id>
      <phase>prepare-package</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

JaCoCo Basics



- JaCoCo is included in NetBeans.
  - Right-click the project and choose **`Code Coverage`**.
  - Make sure **`Collect and Display Code Coverage`** is checked.
  - To generate the code coverage report, choose **`Show Report...`**

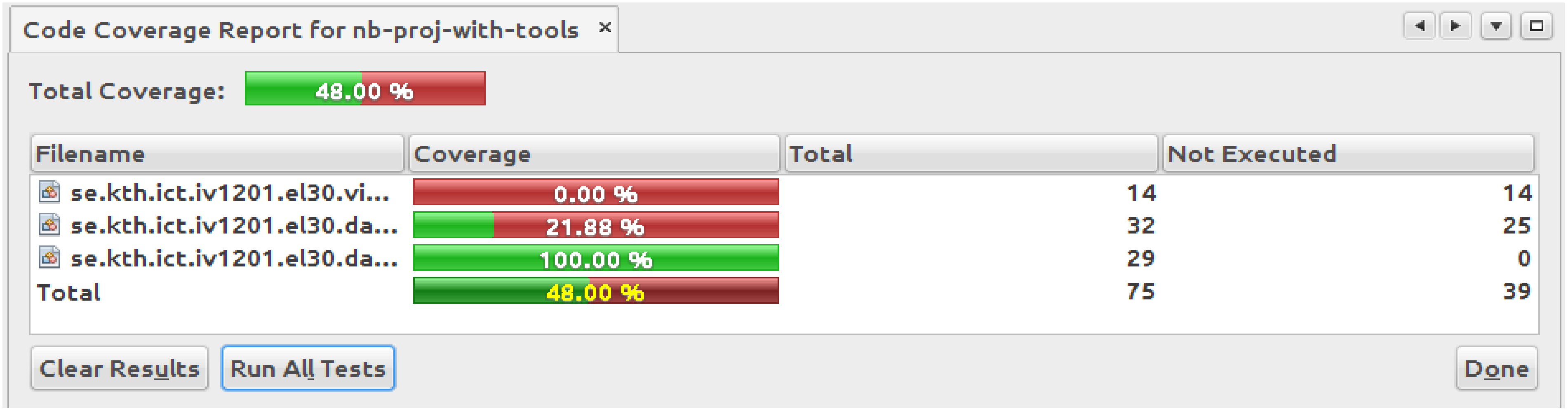
JaCoCo Basics (Cont'd)


Code Coverage Report for nb-proj-with-tools

Total Coverage: 0.00 %

Filename	Coverage	Total	Not Executed
No data -- have you ...	0.00 %	0	0

- Initially there is nothing to report, click `Run All Tests` to generate a report.
  - For this to work, you must have created tests.

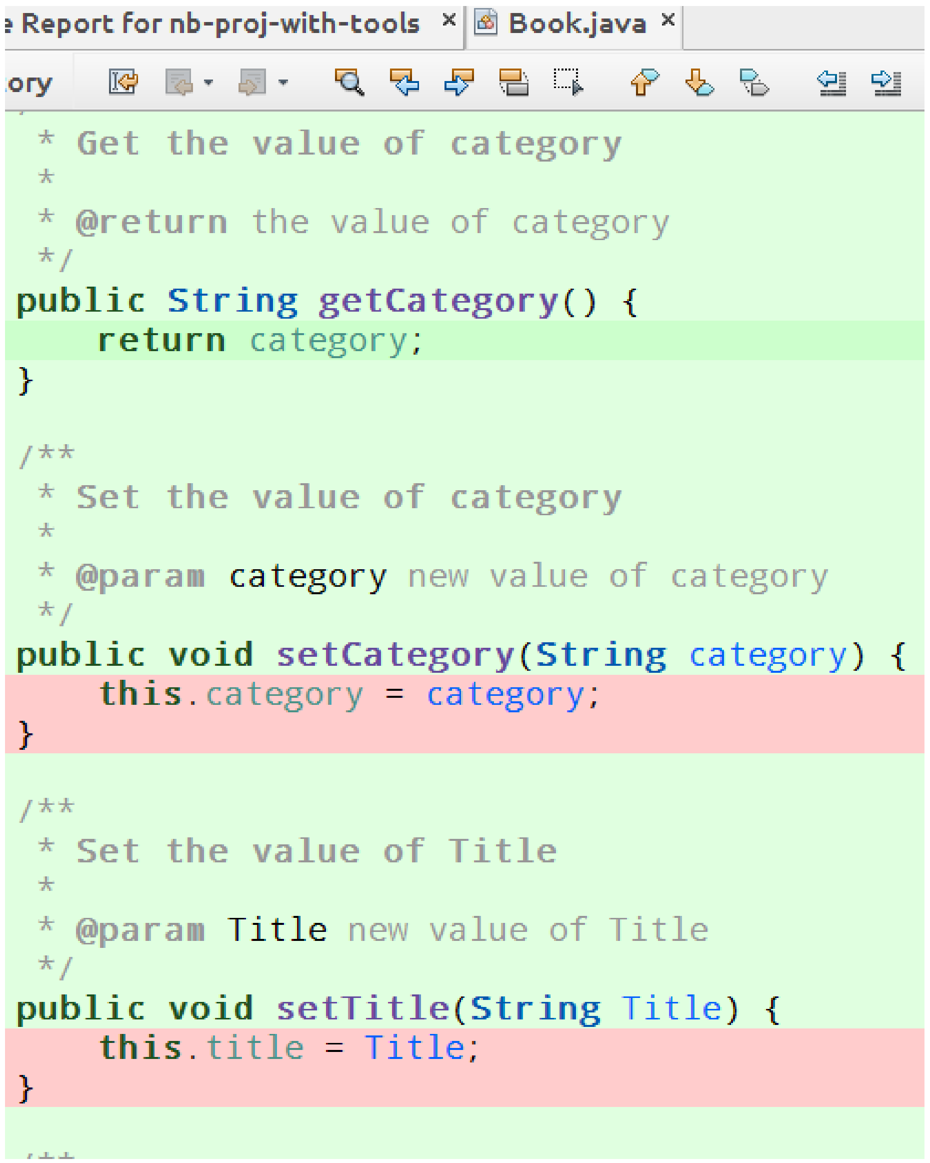

Code Coverage Report for nb-proj-with-tools

Total Coverage: 48.00 %

Filename	Coverage	Total	Not Executed
se.kth.ict.iv1201.el30.vi...	0.00 %	14	14
se.kth.ict.iv1201.el30.da...	21.88 %	32	25
se.kth.ict.iv1201.el30.da...	100.00 %	29	0
Total	48.00 %	75	39

JaCoCo Basics (Cont'd)

If you open a Java source file you will see the executed lines marked in green and those not executed in pink.



```
 * Get the value of category
 *
 * @return the value of category
 */
public String getCategory() {
    return category;
}

/**
 * Set the value of category
 *
 * @param category new value of category
 */
public void setCategory(String category) {
    this.category = category;
}

/**
 * Set the value of Title
 *
 * @param Title new value of Title
 */
public void setTitle(String Title) {
    this.title = Title;
}
```

Profiling With NetBeans

Profiling, Content

- Why Profiling?
- How to Profile Using NetBeans.

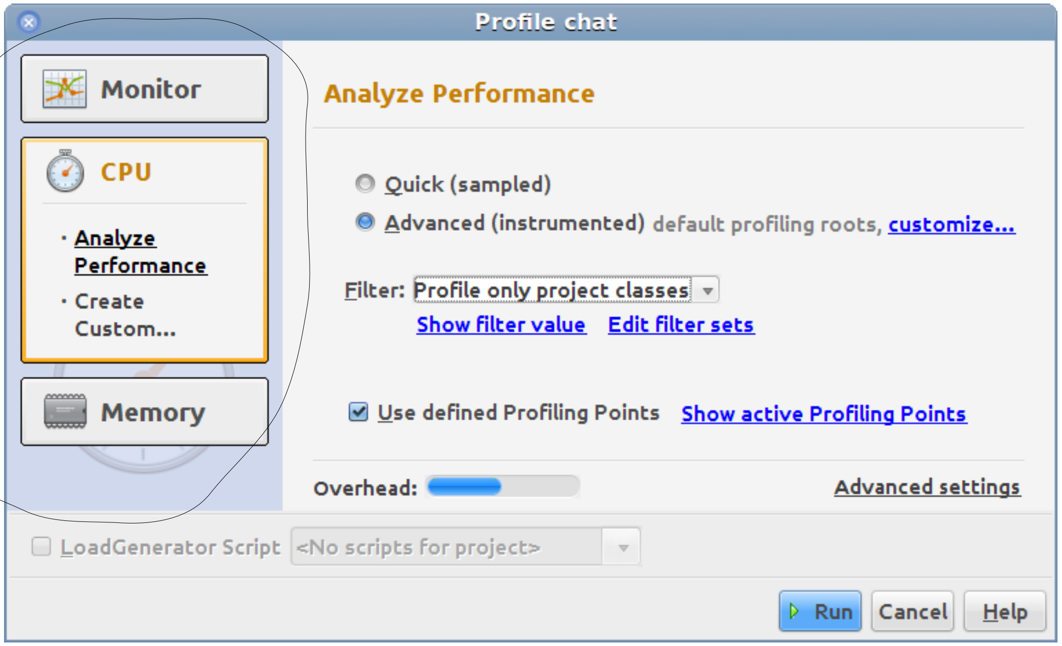
Why Profiling?

- A profiler reports memory usage, CPU time, thread state and other information about program execution.
  - Report can be per package, per class, per method, etc.
  - Either updated live as a running total or a snapshot at a specific time.
- This is very important if you want to optimize your code.
  - Never optimize without knowing what and where is the problem.

How to Profile Using NetBeans

- NetBeans comes with a bundled profiler.
- Before profiling the JDK must be calibrated, the profiler must know how long time different Java operations, e.g., method call, takes.
  - In NetBeans, choose the menu item **`Profile → Advanced Commands → Run Profiler Calibration`**
  - Switch off CPU frequency scaling when doing this.
  - Only needed once per JDK.

How to Profile (Cont'd)



Analyze Performance

- Analyze Performance
- Create Custom...

Quick (sampled)
Advanced (instrumented) default profiling roots, customize...

Filter: Profile only project classes
Show filter value Edit filter sets

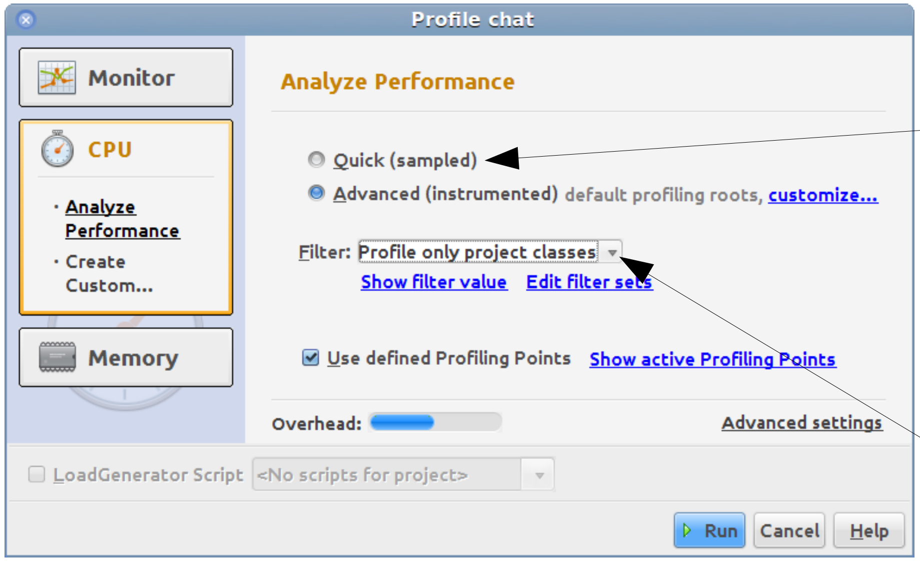
[✓] Use defined Profiling Points Show active Profiling Points

Overhead: Advanced settings

LoadGenerator Script <No scripts for project>

- Right-click the project and choose **`Profile`** to display the dialog box above.
- To the left you can choose to monitor CPU, memory or threads.

How to Profile (Cont'd)



Analyze Performance

Quick (sampled)

[✓ Advanced (instrumented)] default profiling roots, customize...

Filter: Profile only project classes

Show filter value Edit filter sets

[✓ Use defined Profiling Points] Show active Profiling Points

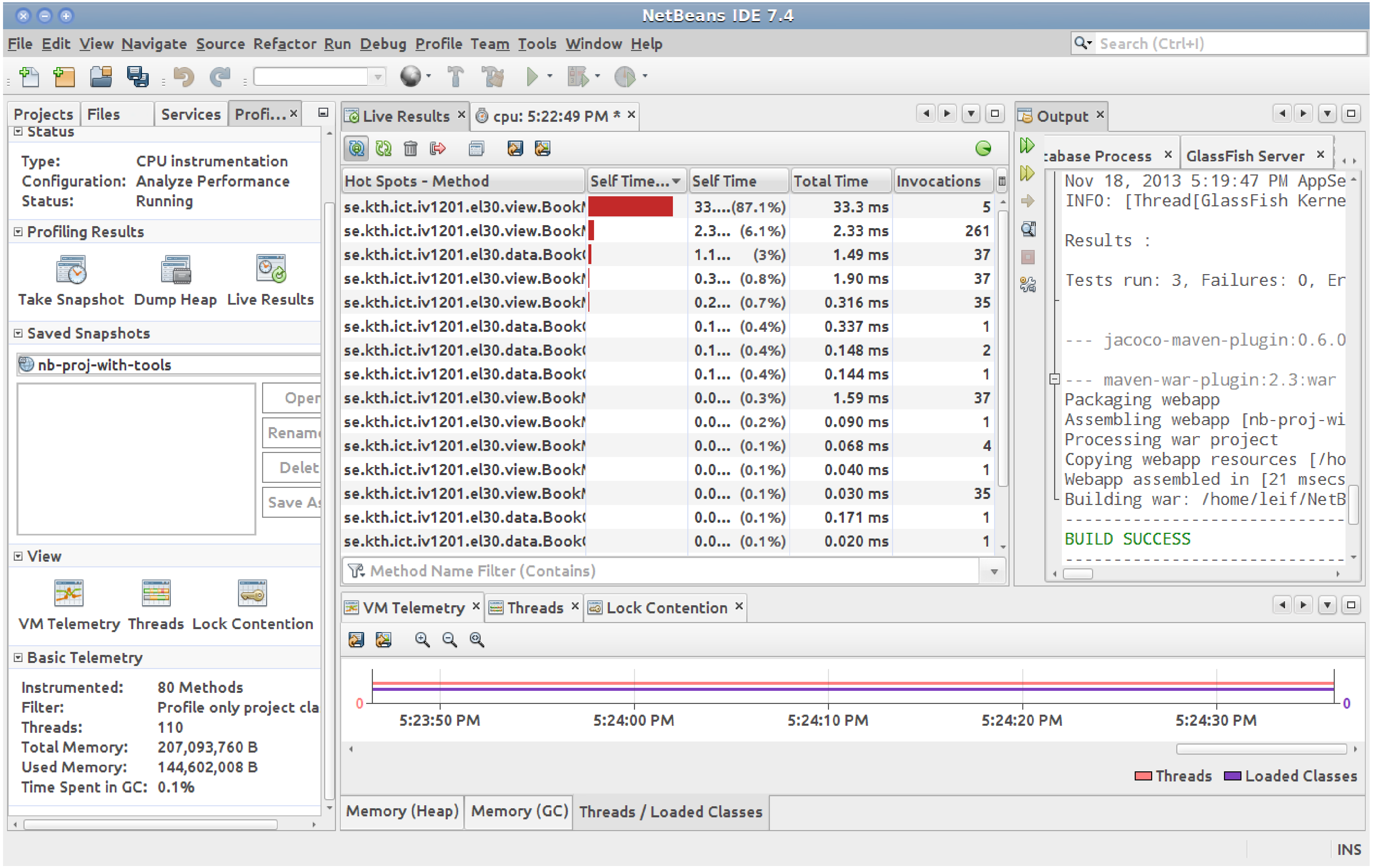
Overhead:

Advanced settings

LoadGenerator Script <No scripts for project>

- The profiler is configured in the main area.
- Choose sampled or instrumented profiling depending on how exact results you need.
- Select which classes to profile, preferably only project classes.

A Sample Profiler Session



NetBeans IDE 7.4

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Status

Type:	CPU instrumentation
Configuration:	Analyze Performance
Status:	Running

Profiling Results

Take Snapshot · Dump Heap · Live Results

Saved Snapshots

nb-proj-with-tools

View

VM Telemetry · Threads · Lock Contention

Basic Telemetry

Instrumented:	80 Methods
Filter:	Profile only project cla
Threads:	110
Total Memory:	207,093,760 B
Used Memory:	144,602,008 B
Time Spent in GC:	0.1%

Live Results · cpu: 5:22:49 PM *

Hot Spots - Method	Self Time...	Self Time	Total Time	Invocations
se.kth.ict.iv1201.el30.view.BookI		33....(87.1%)	33.3 ms	5
se.kth.ict.iv1201.el30.view.BookI		2.3... (6.1%)	2.33 ms	261
se.kth.ict.iv1201.el30.data.Book(		1.1... (3%)	1.49 ms	37
se.kth.ict.iv1201.el30.view.BookI		0.3... (0.8%)	1.90 ms	37
se.kth.ict.iv1201.el30.view.BookI		0.2... (0.7%)	0.316 ms	35
se.kth.ict.iv1201.el30.data.Book(		0.1... (0.4%)	0.337 ms	1
se.kth.ict.iv1201.el30.data.Book(		0.1... (0.4%)	0.148 ms	2
se.kth.ict.iv1201.el30.data.Book(		0.1... (0.4%)	0.144 ms	1
se.kth.ict.iv1201.el30.view.BookI		0.0... (0.3%)	1.59 ms	37
se.kth.ict.iv1201.el30.view.BookI		0.0... (0.2%)	0.090 ms	1
se.kth.ict.iv1201.el30.view.BookI		0.0... (0.1%)	0.068 ms	4
se.kth.ict.iv1201.el30.view.BookI		0.0... (0.1%)	0.040 ms	1
se.kth.ict.iv1201.el30.view.BookI		0.0... (0.1%)	0.030 ms	35
se.kth.ict.iv1201.el30.data.Book(		0.0... (0.1%)	0.171 ms	1
se.kth.ict.iv1201.el30.data.Book(		0.0... (0.1%)	0.020 ms	1

Method Name Filter (Contains)

Output — :abase Process · GlassFish Server

```
Nov 18, 2013 5:19:47 PM AppSe
INFO: [Thread[GlassFish Kerne

Results :

Tests run: 3, Failures: 0, Er


--- jacoco-maven-plugin:0.6.0

--- maven-war-plugin:2.3:war
Packaging webapp
Assembling webapp [nb-proj-wi
Processing war project
Copying webapp resources [/ho
Webapp assembled in [21 msecs
Building war: /home/leif/NetB
------------------------------
BUILD SUCCESS
------------------------------
```

VM Telemetry · Threads · Lock Contention

Memory (Heap) · Memory (GC) · Threads / Loaded Classes

INS

Static Analyzing With NetBeans

Static Analyzing, Content

- Why Static Analyzer?
- How to Use NetBeans' Static Analyzer.

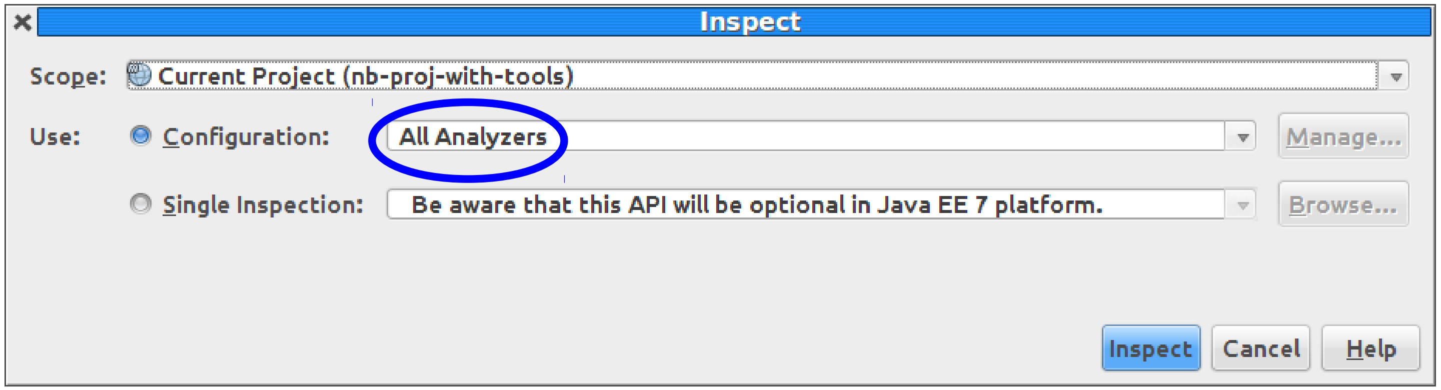
Why Do Static Analysis?

- Static analysis means the code is analyzed without executing the program.
- A static analyzer checks for coding mistakes. It can be for example bugs, unneccessary code or badly formatted code.
- Like the compiler, such a tool helps find coding errors.
- In particular, since a static analyzer checks coding style, many of the misstakes it finds will not be found by executing the program.

Static Analysis With NetBeans

- Choose `Source → Inspect`... from the menu to start the static analyzer.

- Choose which analyzes to perform. In the example below, all analyzers will be executed.


Inspect

Scope: Current Project (nb-proj-with-tools)

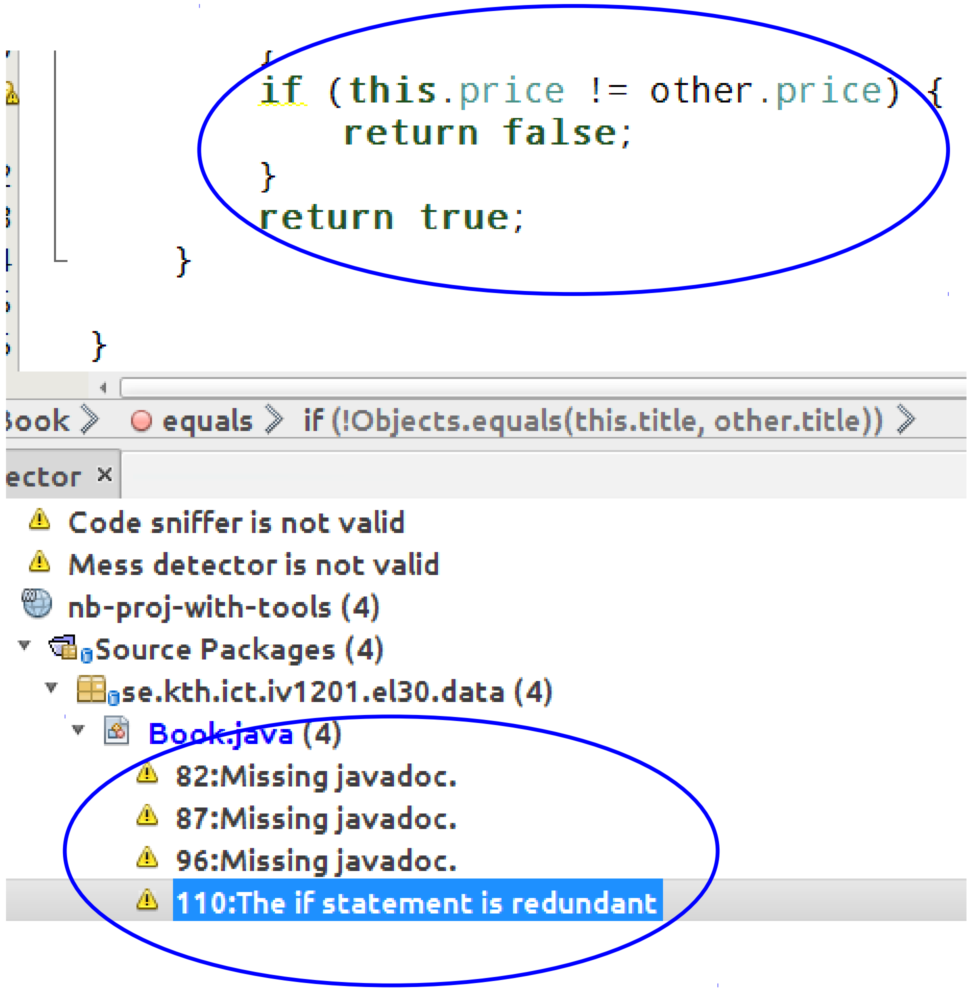
Use: ● Configuration: All Analyzers

○ Single Inspection: Be aware that this API will be optional in Java EE 7 platform.

Static Analysis With NetBeans (Cont'd)

- Here, the analyzer found missing javadoc and an `if` statement that can be rewritten as below.

```
return this.price == other.price;
```



```
if (this.price != other.price) {
    return false;
}
return true;
    }

}
```

ook > equals > if (!Objects.equals(this.title, other.title)) >

ector ×

- Code sniffer is not valid
- Mess detector is not valid
- nb-proj-with-tools (4)
  - Source Packages (4)
    - se.kth.ict.iv1201.el30.data (4)
      - Book.java (4)
        - 82:Missing javadoc.
        - 87:Missing javadoc.
        - 96:Missing javadoc.
        - 110:The if statement is redundant

Continuous Integration With Hudson

Hudson Home Page:
http://www.hudson-ci.org/

Documentation, including online book:
http://wiki.eclipse.org/Hudson-ci/documentation

Continuous Integration, Content

- What is Continuous Integration?
- An Introduction to the Hudson Continuous Integration Server.

What is Continuous Integration (CI)?

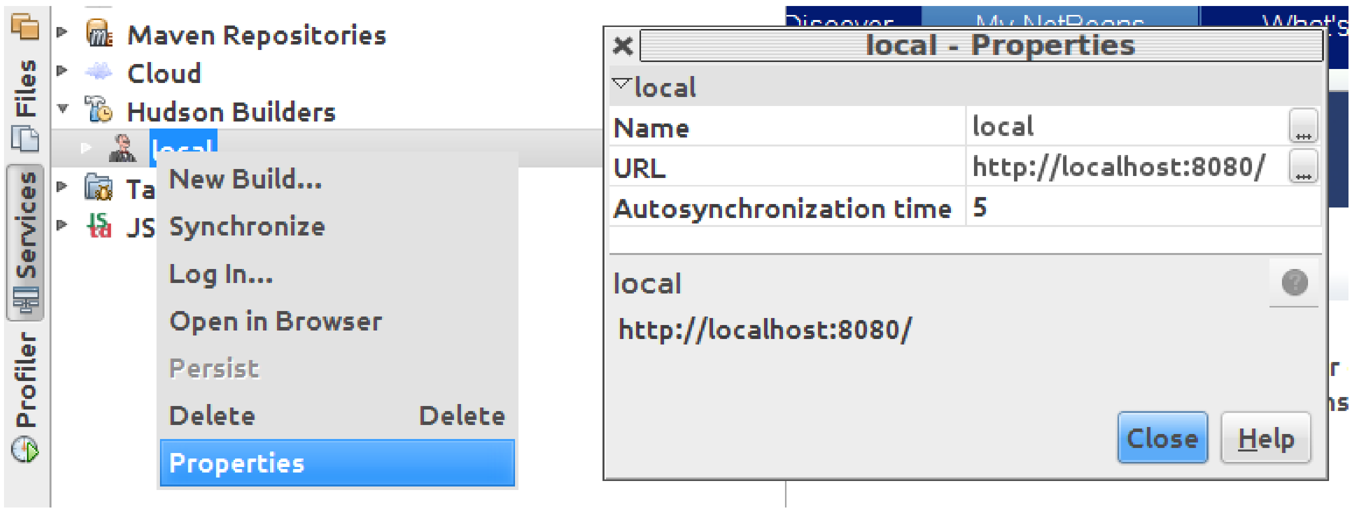
- A software development practice where all developers frequently integrate new code with existing code.
- Each integration is verified by an automated build, to detect integration errors as quickly as possible.
- The automated build includes tests, code coverage reports, repository updates, etc.

What is a CI Server?

- A continuous integration (CI) server manages all parts included in a build.
- Using a CI server all team members perform exactly the same tasks on each submit.
- While developing, team members perform all tests locally, in the IDE.
- Immediately when a piece of code is finished, it is submitted to the CI server. The CI server runs all checks on the entire codebase, publishes the result, updates the code repository and deploys the new version.

Hudson CI Server

- The Hudson CI server can manage all tools and perform all tasks covered in this presentation, and many more.
- A Hudson plugin is available for NetBeans.
  - The plugin is only for communication with the Hudson server. Hudson itself is installed separately.
- The image below shows a possible configuration of the Hudson plugin.



local - Properties

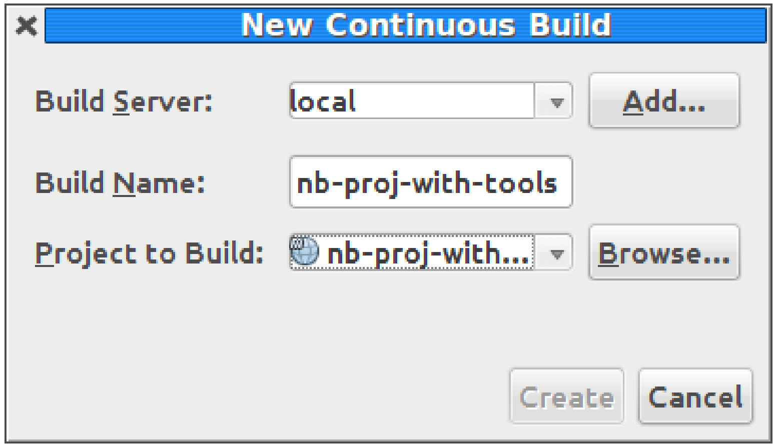
local

Name	local
URL	http://localhost:8080/
Autosynchronization time	5

local

http://localhost:8080/

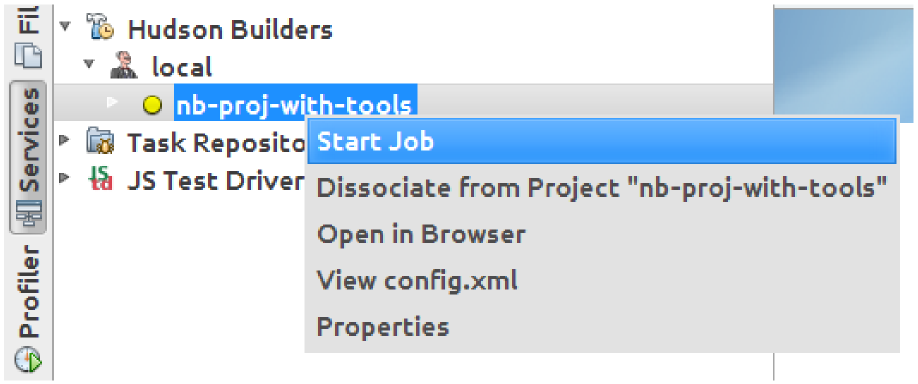
Make Hudson Build the Project



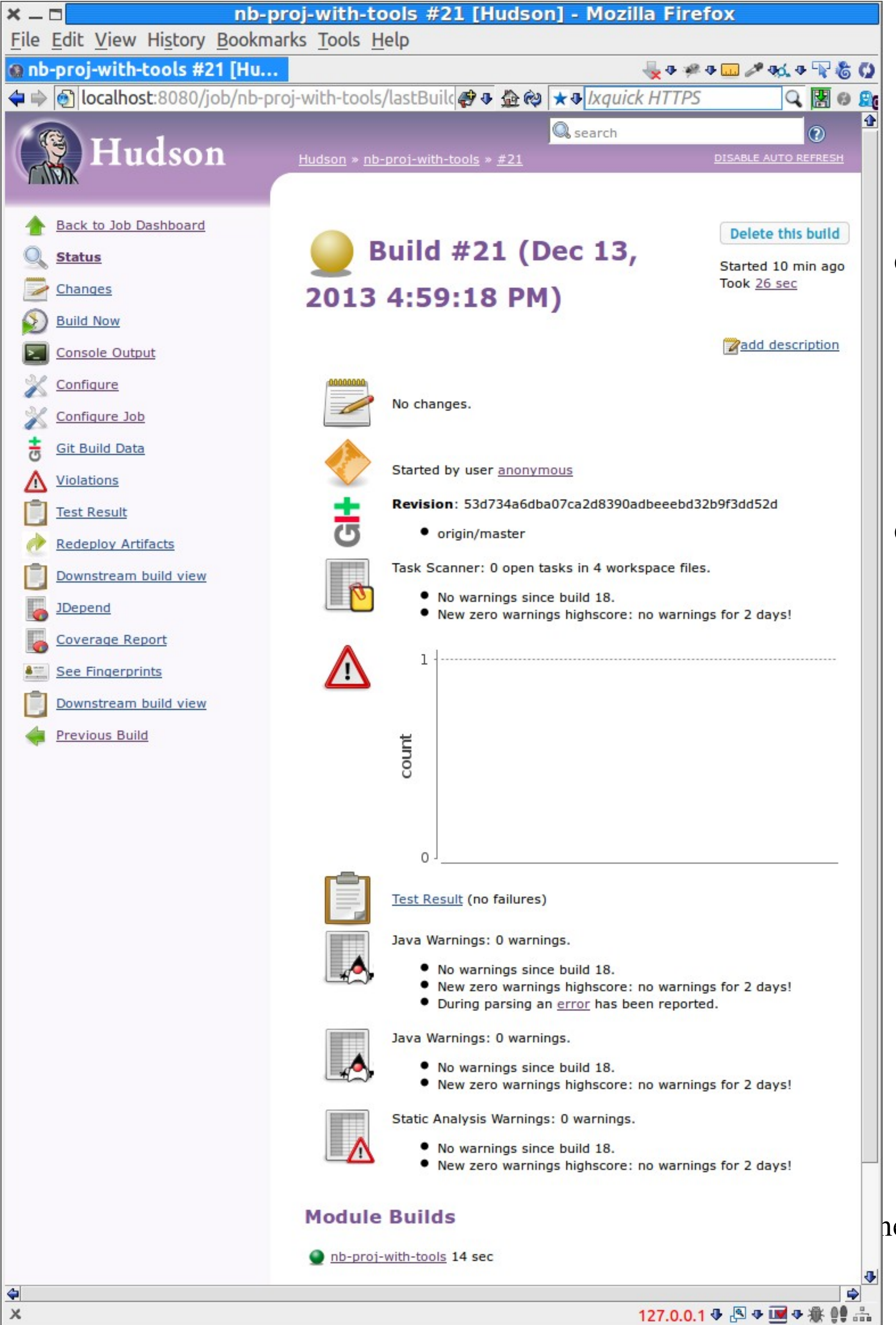
Build Server: local

Build Name: nb-proj-with-tools

Project to Build: nb-proj-with...



- Right-click the new Hudson instance (**`local`** in this example) and choose **`New Build...`** to show the **`New Continuous Build`** Dialog. (upper image)
- Right-click the new build and choose **`Start Job`** to start the build. (lower image)
- Again right-click the build and choose **`Open in Browser`** to show the result.


nb-proj-with-tools #21 [Hudson] - Mozilla Firefox

File Edit View History Bookmarks Tools Help

localhost:8080/job/nb-proj-with-tools/lastBuild

Hudson » nb-proj-with-tools » #21 DISABLE AUTO REFRESH

- Back to Job Dashboard
- Status
- Changes
- Build Now
- Console Output
- Configure
- Configure Job
- Git Build Data
- Violations
- Test Result
- Redeploy Artifacts
- Downstream build view
- JDepend
- Coverage Report
- See Fingerprints
- Downstream build view
- Previous Build

Build #21 (Dec 13, 2013 4:59:18 PM)

Delete this build

Started 10 min ago
Took 26 sec

add description

No changes.

Started by user anonymous

Revision: 53d734a6dba07ca2d8390adbeeebd32b9f3dd52d
- origin/master

Task Scanner: 0 open tasks in 4 workspace files.
- No warnings since build 18.
- New zero warnings highscore: no warnings for 2 days!

Test Result (no failures)

Java Warnings: 0 warnings.
- No warnings since build 18.
- New zero warnings highscore: no warnings for 2 days!
- During parsing an error has been reported.

Java Warnings: 0 warnings.
- No warnings since build 18.
- New zero warnings highscore: no warnings for 2 days!

Static Analysis Warnings: 0 warnings.
- No warnings since build 18.
- New zero warnings highscore: no warnings for 2 days!

Module Builds

nb-proj-with-tools 14 sec

127.0.0.1

Build Result

- Hudson displays a lot of information about to the build.
- We can see for example
  - Test result (no failures)
  - Compiler warnings (none)
  - Static analysis results
  - Code coverage reports
  - Repository status
  - Built modules (one stable)