WHEN FIGHTING APACHE MAVEN…

Robert Scholte (@rfscholte)

Developer vs. Maven?

Feels like Terminator 1

Should be Terminator 2

Developer with Maven!

Feels like Terminator 1

Should be Terminator 2

Reason for a fight? #fail

- Setup (no plug ‘n’ play)
- Adding/removing code/plugins/…
- Suddenly broken

The situation

- An unknown huge multilevel Maven Multimodule Project
- We suddenly have a FAILURE:
  - *CI Server*
  - *Maven Commandline*
  - *IDE*

Is it structural?

IF YOU CANNOT REPRODUCE THE ISSUE, THEN IT IS NOT A BUG

Analysis

Options	Description
-v,--version	Display version information
-V,--show-version	Display version information WITHOUT stopping build
-e,--errors	Produce execution error messages
-X,--debug	Produce execution debug output

Most likely causes

- Your project / code :P
- Maven Plugin
- External Tool (java, javac, javadoc, …)
- Maven

If message doesn’t help

- Google
- Stack Overflow
- Documentation
- Issue management system

Isolate the issue

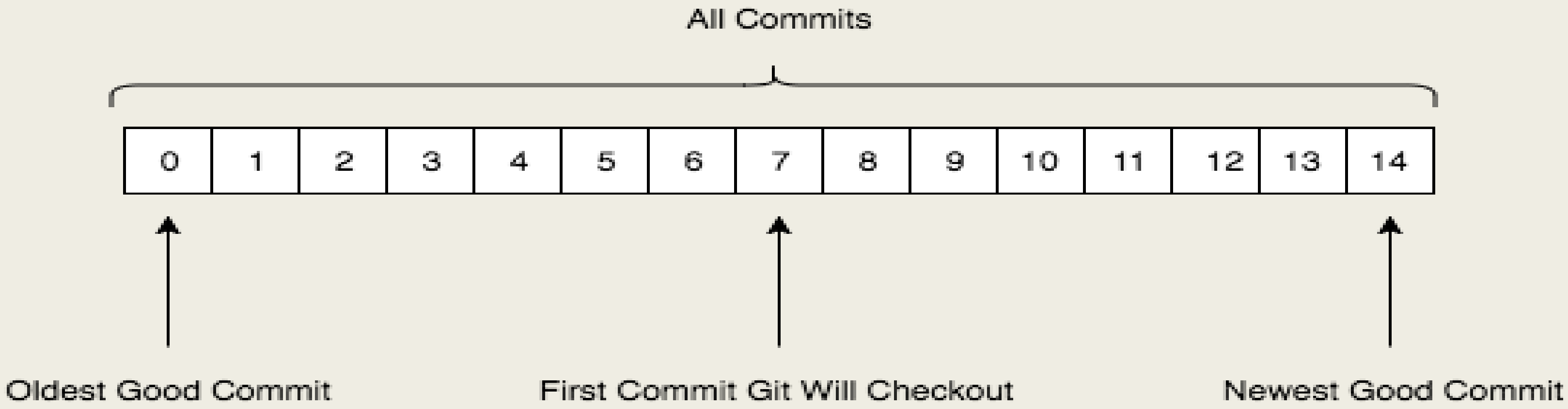
Options	Description
-pl,--projects <arg>	Comma-delimited list of specified reactor projects to build instead of all projects. A project can be specified by [groupId]:artifactId or by its relative path
-am,--also-make	If project list is specified, also build projects required by the list
-amd,--also-make-dependents	If project list is specified, also build projects that depend on projects on the list

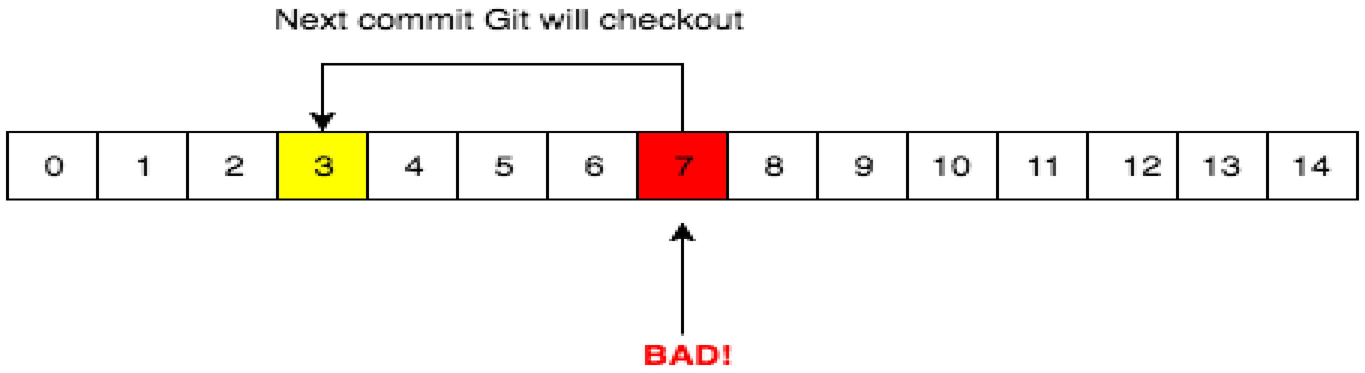
In case of external code:
Sometimes reading code is enough

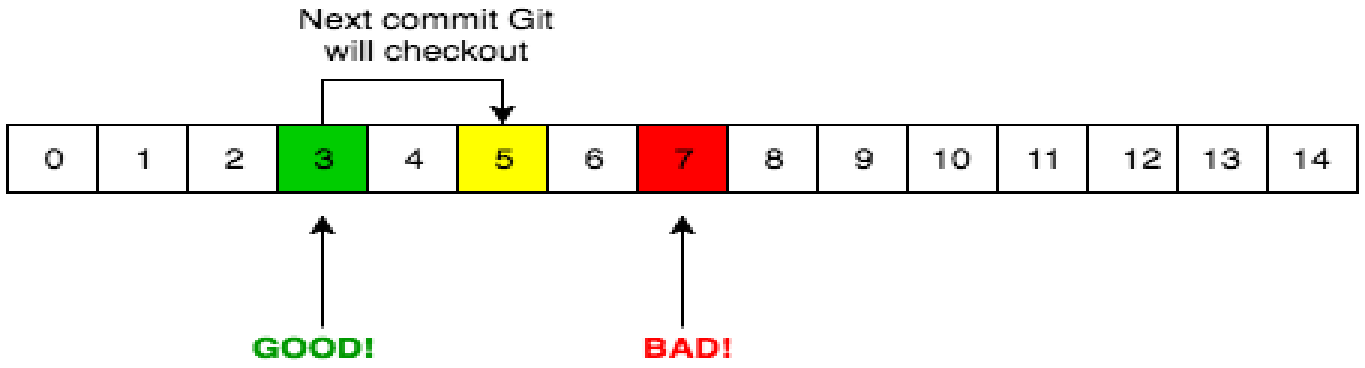
- Github
- JXR pages

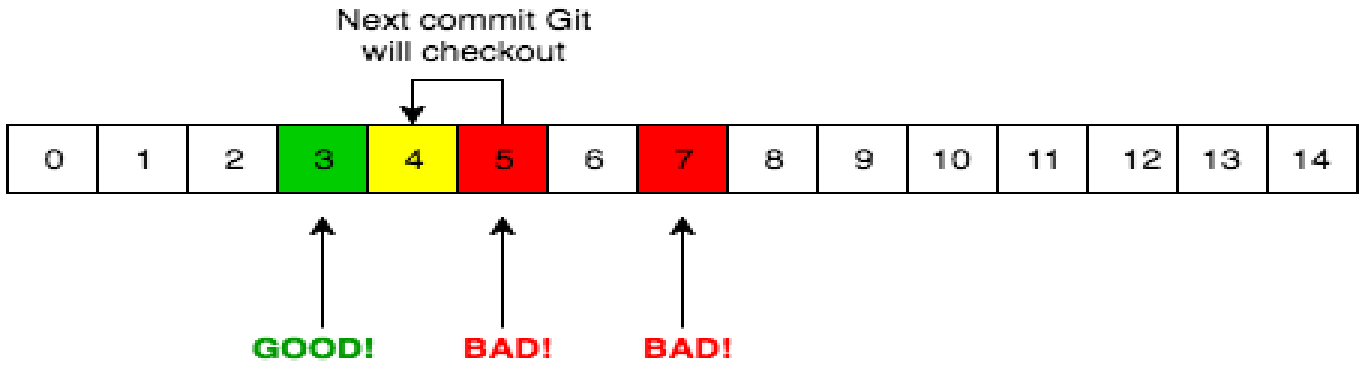
In search of regression with GIT

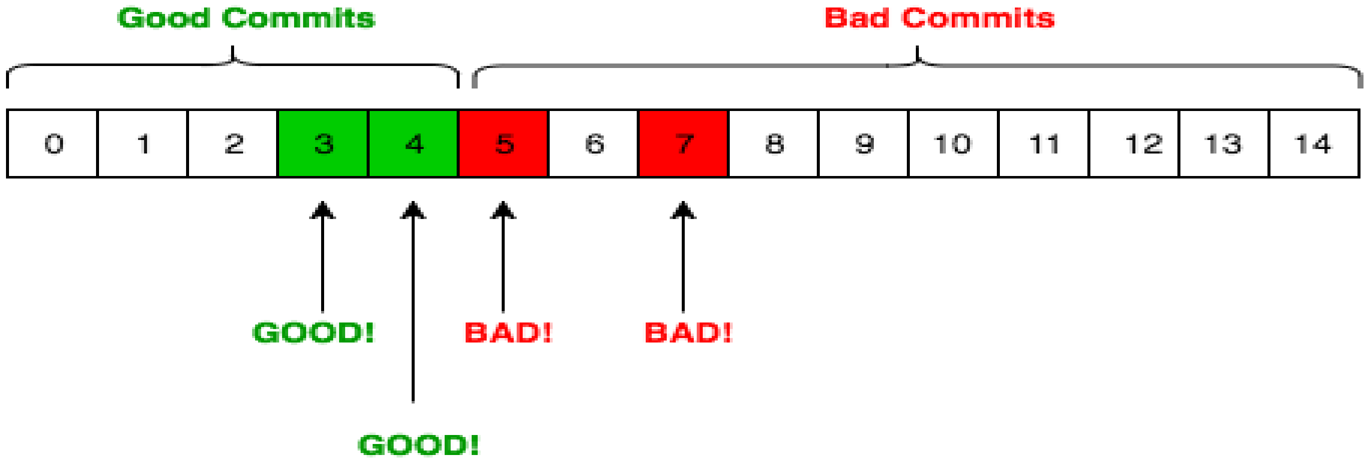
GIT bisect


All Commits
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Oldest Good Commit
First Commit Git Will Checkout
Newest Good Commit


Next commit Git will checkout
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
BAD!


Next commit Git will checkout
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
GOOD!
BAD!


Next commit Git will checkout
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
GOOD!
BAD!
BAD!


Good Commits
Bad Commits
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
GOOD!
GOOD!
BAD!
BAD!

Most likely solution

- Fix your code :P
- Upgrade to a more recent version
  - *Dependencies*
  - *Plugins*
- Patch / fix others code

No more fighting

Let’s go for quick and dirty

- The pom is a strict XML configuration file
- Still… people can be VERY creative

#1 Extending parent pattern

```
<parent>
    <groupId>com.acme.product</groupId>
    <artifactId>parent</artifactId>
    <version>9</version>
 </parent>
 <artifactId>parent</artifactId>
 <version>10</version>
```

Discovery

- ModelValidator
- Check for circular references

Why Dirty?

- Increase number of downloads
- Getting Effective Model is complex process

#2 Replacing pom.xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-deploy-plugin</artifactId>
  <version>2.7</version>
  <configuration>
    <pomFile>custom.pom</pomFile>
  </configuration>
</plugin>
```

Discovery

- Code refactoring to support
  - *installAtEnd*
  - *deployAtEnd*

Why Dirty?

- No guarantee pom is valid

Solution

- Introduction flatten-maven-plugin
  - *“transforms” original pom.xml*
  - *Can apply effective pom elements*
  - *Can remove elements*
- Experience will be used in Maven4
- Experimental feature likely in Maven 3.7.0
  - *maven.experimental.buildconsumer*

#3 Bind to none-phase

- Disable predefined or inherited plugin executions
- E.g. replace surefire with junit-platform-maven-plugin

```xml
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <executions>
    <execution>
      <id>default-test</id>
      <phase>none</phase>
    </execution>
  </executions>
</plugin>
```

Why dirty?

“You should always listen to your parent”

(that’s why I don’t like structural ‘skipping’)

#4 Sharing local repo

Slow wifi / connection

Workshop that requires more plugins/dependencies than expected

Solution: memorystick ?!?!

Mock Repository Manager

org.codehaus.mojo:mrm-maven-plugin:run

mrm:run

When going for quick and dirty…

- Commandline arguments
- Pom.xml

You have one custom requirement

.. But there’s no maven-plugin for it (and don’t want to write one…)

Inside pom execution:

- maven-antrun-plugin
  - *Executes Ant scripts*
- maven-scripting-plugin
  - *Uses the Scripting API (JSR223)*
- exec-maven-plugin
  - *Executes commandline or Java’s main(args)*

You need to manipulate Maven

- Maven Extensions
- Custom Maven Builder

Understanding properties

- Project
- Settings (via profile properties)
- System Properties
- Commandline (-Dkey=value)

testFailureIgnore

Name	Type	Since	Description
<testFailureIgnore>	boolean	-	Set this to "true" to ignore a failure during testing. Its use is NOT RECOMMENDED, but quite convenient on occasion. **Default value is:** false. **User property is:** maven.test.failure.ignore.

Replace default value

```
<properties>
    <maven.test.failure.ignore>true</maven.test.failure.ignore>
</properties>
```

Replace expression

```
<properties>
    <surefire.failureIgnore>false</surefire.failureIgnore>
</properties>
…
<configuration>
  <testFailureIgnore>
    ${surefire.failureIgnore}
  </ testFailureIgnore>
</configuration>
```

Replace with constant

```
<configuration>
  <testFailureIgnore>false</testFailureIgnore>
</configuration>
```

Making friends

```
<configuration>
  <skipTests>false</skipTests>
</configuration>
```

The ‘evil’ jenkinsci maven-plugin

- Why does it continue after a failed test???
- h.m.r.SurefireArchiver L87-L114

Commandline arguments

What will happen when you execute

‘mvn deploy -DjavaVersion=13’ ?

spring-boot-dependencies

What will happen when you execute

‘mvn deploy -Dspring.version=5.1.0.RELEASE’ ?

“ARGUMENTS SHOULD NEVER HAVE EFFECT ON THE CREATED ARTIFACTS”

Clean Install

- Maven is about convention of configuration
- If the convention was ‘clean install’,

  you should simply execute ‘mvn’

Clean Install is not quick, just dirty

Clean lifecycle

Phase	Binding
pre-clean	
clean	clean:clean (remove target directory)
post-clean	

Clean

- Delete and re-place (same) files is waste of resources
- ***Most*** maven-plugins are aware if they must execute their task

Avoid “clean”

Clean Install is not quick, just dirty

Build / default lifecycle

Phase	Binding (for every packaging)
…	
install	install:install (copy artifact to local repo)
deploy	deploy:deploy (upload to remote repo)

Install

Maven 2:

- Not aware of ‘reactor’
- Dependencies had to exist in local repo.

Maven 3:

- ‘reactor’ aware
- No need for ‘install’ anymore

Avoid Clean, Avoid Install

Introducing the Maven CI Extension

Ultimate hack:
just fork and re-version

Ensure no conflicts with official future versions

e.g. 3.6.3-rfscholte-SNAPSHOT

Up for Grabs

- ~60-80% of Java Project/Developers use Maven
- The Apache Maven Project holds ~95 (sub)projects
- Maintained by ~5-10 active volunteers (No Company!)

- Let’s restore the balance!
- https://s.apache.org/up-for-grabs_maven

- https://maven.apache.org/guides/development/guide-committer-school.html

THANK YOU AND HAPPY HACKING!