

***ma**ven*



git

Projektverwaltung

Sascha Just
Softwarepraktikum 2014



Projektverwaltung mit ***maven***

- ▶ Maven ist ein Werkzeug zur Verwaltung von Java-Projekten
- ▶ Konvention statt Konfiguration
- ▶ Abhängigkeiten (benötigte Bibliotheken) werden automatisch verwaltet

Project Object Model

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

pom.xml

*Das Project Object Model (**POM**) beschreibt das Projekt.*

Project Object Model

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

pom.xml

Project Object Model

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

pom.xml

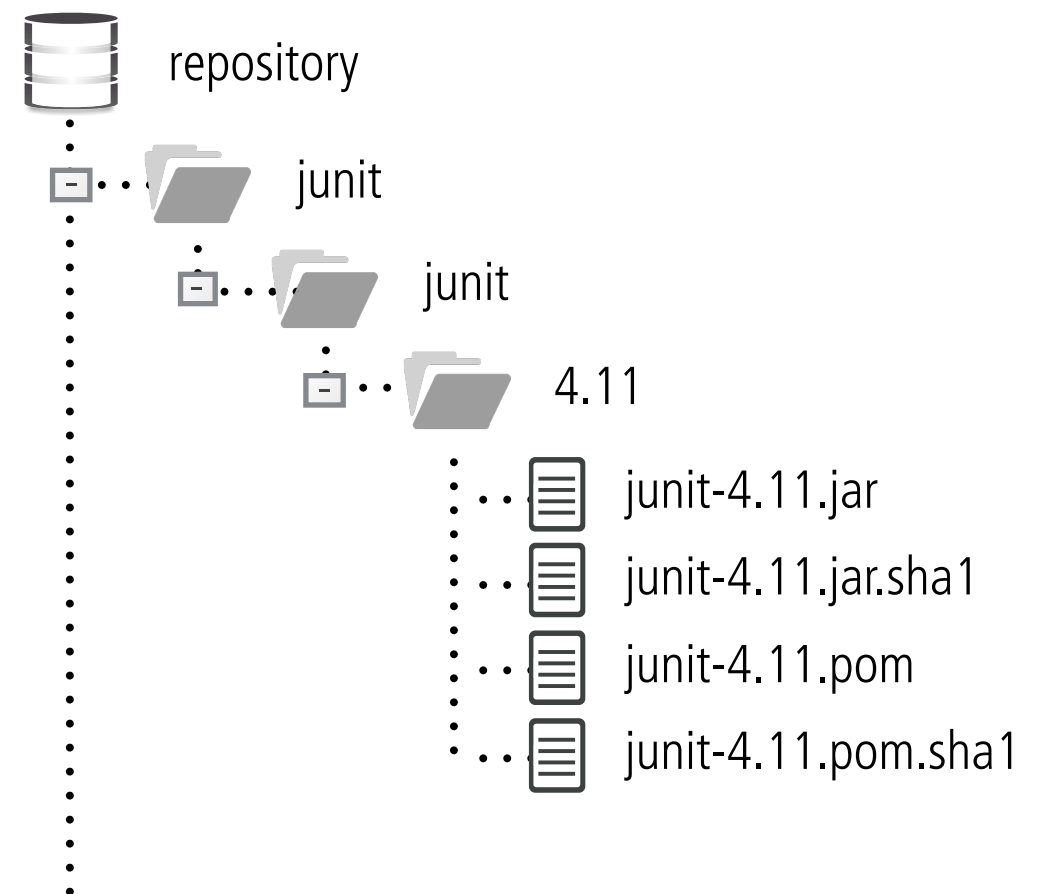
Project Object Model

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

pom.xml

Abhängigkeiten

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



Verzeichnisstruktur

pom.xml	POM Datei im Projekt-Verzeichnis.
src/main/java/	Java-Quelldateien.
src/main/resources/	Zur Laufzeit benötigte Ressourcen (<i>z.B. Bilder</i>).
src/test/java/	Java-Quelldateien für Unit-Tests.
src/test/resources/	Dateien welche zum Testen benötigt werden.
target/	Von Maven generierte Dateien.

Wichtige Goals

eclipse:eclipse

Erzeugt ein Eclipse-Projekt.

test

Führt alle Unit-Tests aus.

compile

Kompiliert das Projekt.

assembly:assembly

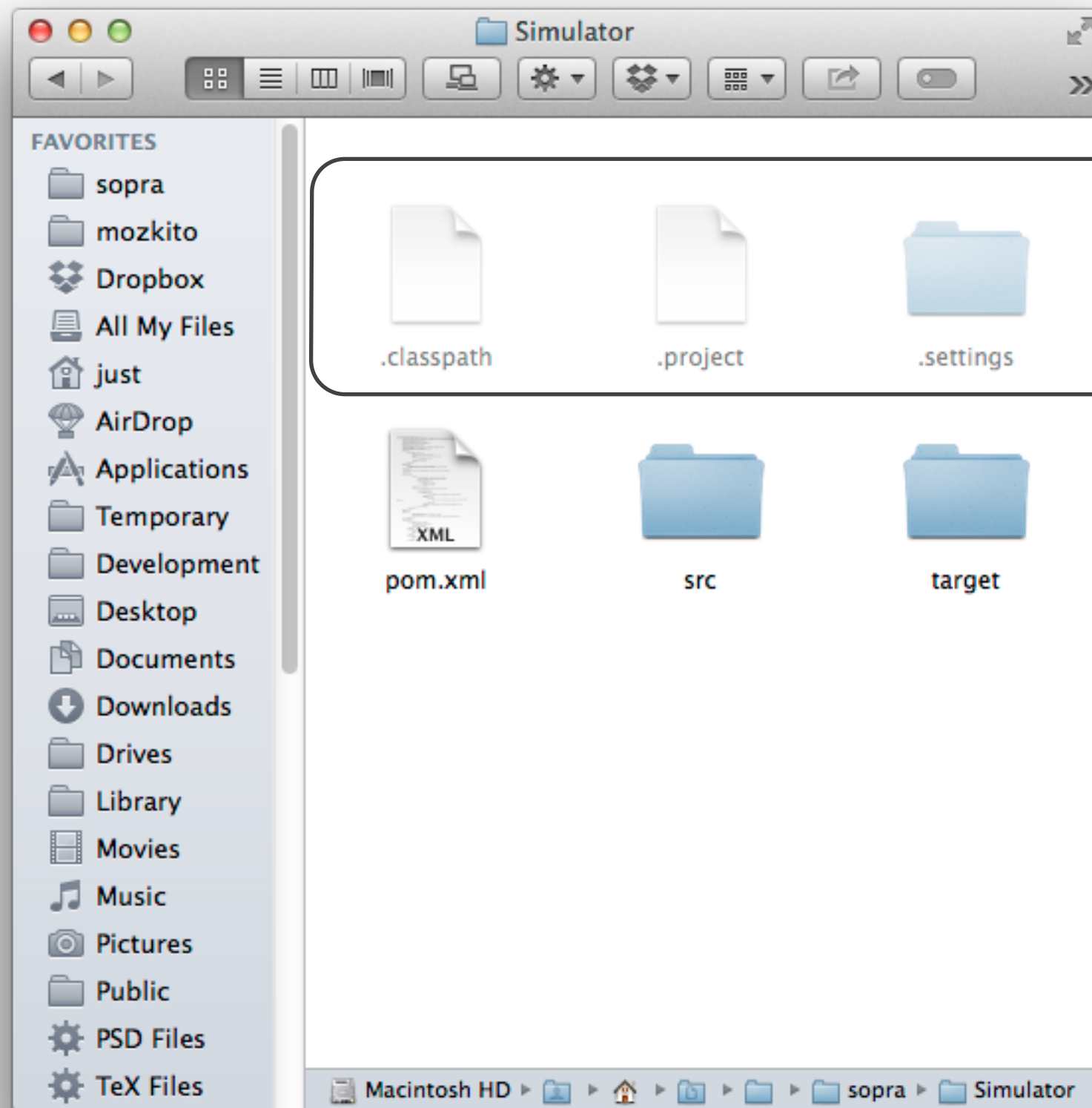
Packt ein Archiv für die Auslieferung.

install

Kopiert das Projekt in das lokale Repository.

site

Generiert eine HTML-Seite mit Projekt-Berichten.
(z.B. *JavaDoc*, *Checkstyle*, *FindBugs*, *PMD*, etc...)



Eclipse Projektdateien.

Wichtige Goals

eclipse:eclipse

Erzeugt ein Eclipse-Projekt.

test

Führt alle Unit-Tests aus.

compile

Kompiliert das Projekt.

assembly:assembly

Packt ein Archiv für die Auslieferung.

install

Kopiert das Projekt in das lokale Repository.

site

Generiert eine HTML-Seite mit Projekt-Berichten.
(z.B. *JavaDoc*, *Checkstyle*, *FindBugs*, *PMD*, etc...)

```

[INFO] Generating persistence data file versions-persistence.xml
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ mozkitto-versions ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources to META-INF
[INFO] Copying 5 resources
[INFO] Copying 1 resource
[INFO] --- maven-dependency-plugin:2.8:copy (copy) @ mozkitto-versions ---
[INFO] Configured Artifact: net.ownhero.dev:kanuni-agent:0.2:jar
[INFO] Copying kanuni-agent-0.2.jar to /Users/just/Development/mozkitto/mozkitto-modules/mozkitto-versions/target/kanuni-agent-0.2.jar
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ mozkitto-versions ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-checkstyle-plugin:2.10:check (compile) @ mozkitto-versions ---
[INFO] --- mozkitto-maven-plugin:0.4-SNAPSHOT:nls (compile) @ mozkitto-versions ---
[INFO] --- openjpa-maven-plugin:2.2.2:enhance (enhancer) @ mozkitto-versions ---
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ mozkitto-versions ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 8 resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ mozkitto-versions ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 15 source files to /Users/just/Development/mozkitto/mozkitto-modules/mozkitto-versions/target/test-classes
[WARNING] Supported source version 'RELEASE_6' from annotation processor 'org.apache.openjpa.persistence.meta.AnnotationProcessor6' less than -source '1.7'
[INFO] --- maven-surefire-plugin:2.16:test (default-test) @ mozkitto-versions ---
[WARNING] The parameter forkMode is deprecated since version 2.14. Use forkCount and reuseForks instead.
[INFO] Surefire report directory: /Users/just/Development/mozkitto/mozkitto-modules/mozkitto-versions/target/surefire-reports

```

T E S T S

T E S T S

Running org.mozkitto.persistence.Versions_PersistenceTest

```

2014-09-07 13:47:30,437 ( 4165) [main] ALWAYS deleting database directory '/Users/just/Development/mozkitto/mozkitto-modules/mozkitto-versions/target/test.db_38421_sunse
2014-09-07 13:47:31,515 ( 5243) [main] ALWAYS deleting database directory '/Users/just/Development/mozkitto/mozkitto-modules/mozkitto-versions/target/test.db_38421_sunse
2014-09-07 13:47:32,442 ( 6170) [main] ALWAYS deleting database directory '/Users/just/Development/mozkitto/mozkitto-modules/mozkitto-versions/target/test.db_38421_sunse
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.395 sec - in org.mozkitto.persistence.Versions_PersistenceTest

```

Running org.mozkitto.versions.git.GitLogParserTest

```

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.289 sec - in org.mozkitto.versions.git.GitLogParserTest

```

Running org.mozkitto.versions.git.GitRepositoryTest

```

Tests run: 12, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 3.734 sec <<< FAILURE! - in org.mozkitto.versions.git.GitRepositoryTest
testGetFormerPathName(org.mozkitto.versions.git.GitRepositoryTest) Time elapsed: 0.043 sec <<< FAILURE!

```

java.lang.AssertionError

```

    at org.junit.Assert.fail(Assert.java:86)
    at org.junit.Assert.assertTrue(Assert.java:41)
    at org.junit.Assert.assertNotNull(Assert.java:621)
    at org.junit.Assert.assertNotNull(Assert.java:631)
    at org.mozkitto.versions.git.GitRepositoryTest.testGetFormerPathName(GitRepositoryTest.java:204)
    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.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)

```



```
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ mozkitto-versions ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 15 source files to /Users/just/Development/mozkito/mozkito-modules/mozkito-versions/target/test-classes
[WARNING] Supported source version 'RELEASE_6' from annotation processor 'org.apache.openjpa.persistence.meta.AnnotationProcessor6' less than -source '1.7'
[INFO]
[INFO] --- maven-surefire-plugin:2.16:test (default-test) @ mozkitto-versions ---
[WARNING] The parameter forkMode is deprecated since version 2.14. Use forkCount and reuseForks instead.
[INFO] Surefire report directory: /Users/just/Development/mozkito/mozkito-modules/mozkito-versions/target/surefire-reports
```

TESTS

Tests run: 12, Failures: 1, Errors: 0, Skipped: 0

```
Tests run: 12, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 3.734 sec <<< FAILURE! - in org.mozkito.versions.git.GitRepositoryTest
testGetFormerPathName(org.mozkito.versions.git.GitRepositoryTest) Time elapsed: 0.043 sec <<< FAILURE!
java.lang.AssertionError
    at org.junit.Assert.fail(Assert.java:86)
    at org.junit.Assert.assertTrue(Assert.java:41)
    at org.junit.Assert.assertNotNull(Assert.java:621)
    at org.junit.Assert.assertNotNull(Assert.java:631)
    at org.mozkito.versions.git.GitRepositoryTest.testGetFormerPathName(GitRepositoryTest.java:204)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
```

Fehlgeschlagene Assertion in GitRepositoryTest.

```
at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:271)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:70)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:50)
at org.junit.runners.ParentRunner$3.run(ParentRunner.java:238)
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
at org.junit.runners.ParentRunner.access$000(ParentRunner.java:53)
at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:229)
at org.junit.runners.ParentRunner.run(ParentRunner.java:309)
at org.apache.maven.surefire.junit4.JUnit4Provider.execute(JUnit4Provider.java:264)
at org.apache.maven.surefire.junit4.JUnit4Provider.executeTestSet(JUnit4Provider.java:153)
at org.apache.maven.surefire.junit4.JUnit4Provider.invoke(JUnit4Provider.java:124)
at org.apache.maven.surefire.booter.ForkedBooter.invokeProviderInSameClassLoader(ForkedBooter.java:200)
at org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:153)
at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:103)
```

```
Running org.mozkito.versions.git.GitTransactionIteratorTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.471 sec - in org.mozkito.versions.git.GitTransactionIteratorTest
Running org.mozkito.versions.mercurial.MercurialLogParserTest
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.315 sec - in org.mozkito.versions.mercurial.MercurialLogParserTest
Running org.mozkito.versions.mercurial.MercurialRepositoryTest
```

Maven Testergebnisse

./target/surefire-reports/

•----- **org.mozkito.versions.git.GitRepositoryTest.txt**

Maven Ausgabe des Testlaufs.

•----- **TEST-org.mozkito.versions.git.GitRepositoryTest.xml**

Konfigurationsbeschreibung des Testlaufs in XML.

Wichtige Goals

eclipse:eclipse

Erzeugt ein Eclipse-Projekt.

test

Führt alle Unit-Tests aus.

compile

Kompiliert das Projekt.

assembly:assembly

Packt ein Archiv für die Auslieferung.

install

Kopiert das Projekt in das lokale Repository.

site

Generiert eine HTML-Seite mit Projekt-Berichten.
(z.B. *JavaDoc*, *Checkstyle*, *FindBugs*, *PMD*, etc...)

Wichtige Goals

eclipse:eclipse

Erzeugt ein Eclipse-Projekt.

test

Führt alle Unit-Tests aus.

compile

Kompiliert das Projekt.

assembly:assembly

Packt ein Archiv für die Auslieferung.

install

Kopiert das Projekt in das lokale Repository.

site

Generiert eine HTML-Seite mit Projekt-Berichten.
(z.B. *JavaDoc*, *Checkstyle*, *FindBugs*, *PMD*, etc...)

Wichtige Goals

eclipse:eclipse

Erzeugt ein Eclipse-Projekt.

test

Führt alle Unit-Tests aus.

compile

Kompiliert das Projekt.

assembly:assembly

Packt ein Archiv für die Auslieferung.

install

Kopiert das Projekt in das lokale Repository.

site

Generiert eine HTML-Seite mit Projekt-Berichten.
(z.B. *JavaDoc*, *Checkstyle*, *FindBugs*, *PMD*, etc...)

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

maven install



\$HOME/.m2/repository/com.mycompany.app/my-app/1.0-SNAPSHOT/

- ▶ my-app-1.0-SNAPSHOT.jar
- ▶ my-app-1.0-SNAPSHOT.pom

Wichtige Goals

eclipse:eclipse

Erzeugt ein Eclipse-Projekt.

test

Führt alle Unit-Tests aus.

compile

Kompiliert das Projekt.

assembly:assembly

Packt ein Archiv für die Auslieferung.

install

Kopiert das Projekt in das lokale Repository.

site

Generiert eine HTML-Seite mit Projekt-Berichten.
(z.B. *JavaDoc*, *Checkstyle*, *FindBugs*, *PMD*, etc...)



Pirates '14 Simulator

Last Published: 2014-09-07 | Version: 1.0

Pirates '14 Simulator

Project Documentation


▼ Project Information

- [Continuous Integration](#)
- [Dependencies](#)
- [Project License](#)
- [About](#)
- [Project Summary](#)
- [Source Repository](#)
- [Project Team](#)

► Project Reports



Project Information

This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by [Maven](#)  on behalf of the project.

Overview

Document	Description
Continuous Integration	This is a link to the definitions of all continuous integration processes that builds and tests code on a frequent, regular basis.
Dependencies	This document lists the project's dependencies and provides information on each dependency.
Project License	This is a link to the definitions of project licenses.
About	Reference Implementation of Pirates '14
Project Summary	This document lists other related information of this project
Source Repository	This is a link to the online source repository that can be viewed via a web browser.
Project Team	This document provides information on the members of this project. These are the individuals who have contributed to the project in one form or another.

Pirates '14 Simulator

Last Published: 2014-09-07 | Version: 1.0

Pirates '14 Simulator

Project Documentation

Project Information

Project Reports

- Checkstyle
- JavaDocs
- Source Xref
- Surefire Report
- Cobertura Test Coverage
- PMD



Generated Reports

This document provides an overview of the various reports that are automatically generated by [Maven](#) . Each report is briefly described below.

Overview

Document	Description
Checkstyle	Report on coding style conventions.
JavaDocs	JavaDoc API documentation.
Source Xref	HTML based, cross-reference version of Java source code.
Surefire Report	Report on the test results of the project.
Cobertura Test Coverage	Cobertura Test Coverage Report.
PMD	Verification of coding rules.

Copyright © 2014 [Software Engineering Group at Saarland University](#). All Rights Reserved.

Versionsverwaltung mit git

Warum Versionsverwaltung?

- ▶ Nachverfolgen von Änderungen (Projektgeschichte)
- ▶ Zusammenführen von Änderungen
- ▶ Dokumentation / Archivierung von Zwischenzuständen

Herkömmliche Versionsverwaltungssysteme

- ▶ CVS, Subversion (SVN)
- ▶ Ein zentraler Repositoryserver
- ▶ Jeder Commit wird zentral gespeichert
- ▶ Lineare Revisionskette
- ▶ Probleme:
 - kein Netzwerkzugriff?
 - kein Schreibrecht?
 - Branch&Merge komplex

Verteilte Versions-verwaltungssysteme

- ▶ Bazaar, Mercurial, Git
- ▶ jeder Entwickler hat eine Kopie des gesamten Repositories (*Historie des Projekts*)
- ▶ ein oder mehrere zentrale Repositories sind üblich zur Synchronisation
- ▶ schnelle Analysen möglich (log, bisect)
- ▶ verteiltes Arbeiten (github)

Commits

- ▶ die wichtigsten Objekte in Git
- ▶ bestehen aus:
 - Autor-Informationen
 - Datum des Commits
 - aktueller Stand des Repositories
 - Parent-Commits
- ▶ identifiziert durch SHA1-Hash über diese Informationen

f67fd31b

Changeset

Tree

Author Sascha Just <sascha.just@own-hero.net>
Author Date 7 Oct 2013 15:40:45 GMT+2
Committer Sascha Just <sascha.just@own-hero.net>
Committer Date 7 Oct 2013 15:40:45 GMT+2
Commit Hash f67fd31b29ec523525ec275ea4d9763ebd1eb614
Parent Hash 50ffbb5449e73871af6fa509346fe840a8e0fb74
Tree Hash 0132365954a915123c6ef6037934a2f2d6591ece



fixing infozilla impl

Expand All

Showing 21 changed files with 436 additions and 142 deletions.

- ▶ added A mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/AttachmentProvider.java
- ▶ added A mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/IAttachmentProvider.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/Filter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/IFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/enumeration/AdaptiveListingFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/enumeration/EnumerationFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/link/LinkFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/log/LogFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/patch/PatchFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/patch/PatchParser.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/patch/UnifiedDiffPatchFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/sourcecode/JavaSourceCodeFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/sourcecode/SourceCodeFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/stacktrace/JavaStackTraceFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/filters/stacktrace/StackTraceFilter.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/managers/AttachmentManager.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/managers/InlineManager.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/model/archive/Archive.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/model/attachment/Attachment.java
- ▶ modified M mozkitomozkito-infozilla/src/main/java/org/mozkito/infozilla/model/patch/PatchTextElement.java
- ▶ modified M mozkitomozkito-mappings/src/main/java/org/mozkito/mappings/engines/PatchEngine.java

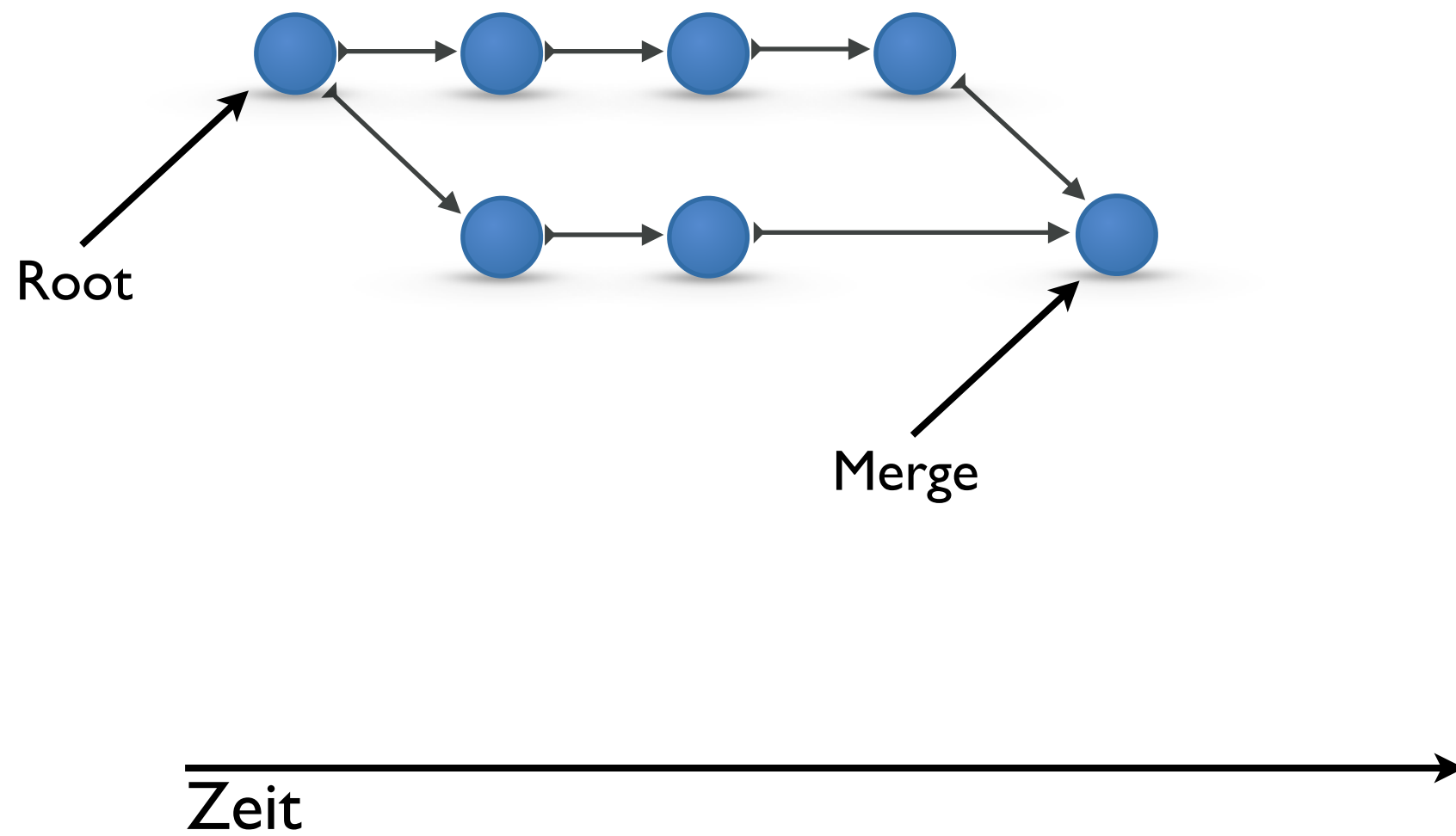
```
54 50 // body
55 51 // final SimpleEditor editor = new SimpleEditor();
56 52 // final Thread t = new Thread(editor);
57 53 // t.start();
58 54 final SimpleEditor editor = null;
-59 final Regex regex = new Regex("201\\d.*");
60 55
-61 new ProcessHook<Report, EnhancedReport>(this) {
+56 new ProcessHook<EnhancedReport, EnhancedReport>(this) {
62 57
63 58 /**
64 59 * {@inheritDoc}
65 60 *
66 61 * @see net.ownhero.dev.andama.threads.ProcessHook#process()
67 62 */
68 63 @Override
69 64 public void process() {
70 65     PRECONDITIONS: {
71 66         // none
72 67     }
73 68
74 69     try {
-75         final Report data = getInputData();
-76         final Match match = regex.find(data.getDescription());
-77         if (match != null) {
-78             if (Logger.logAlways()) {
-79                 Logger.always(match.getFullMatch().getMatch());
-80             }
-81         }
-82         EnhancedReport enhancedReport = null;
+70         final EnhancedReport data = getInputData();
83 71
84 72         if (data != null) {
85 73             final IFilterManager chain = new InlineFilterManager(data, editor);
-86             enhancedReport = chain.parse();
+74             chain.parse();
87 75         }
88 76
-89             provideOutputData(enhancedReport);
+77             provideOutputData(data);
90 78         } finally {
91 79             POSTCONDITIONS: {
92 80                 // none
93 81             }
94 82         }
95 83     }
96 84 };
```

Commits

Drei verschiedene Typen von Commits:

- ▶ Root-Commits (kein Parent)
- ▶ normale Commits (ein Parent)
- ▶ Merge-Commits (zwei oder mehr Parents)

Commits



Commits

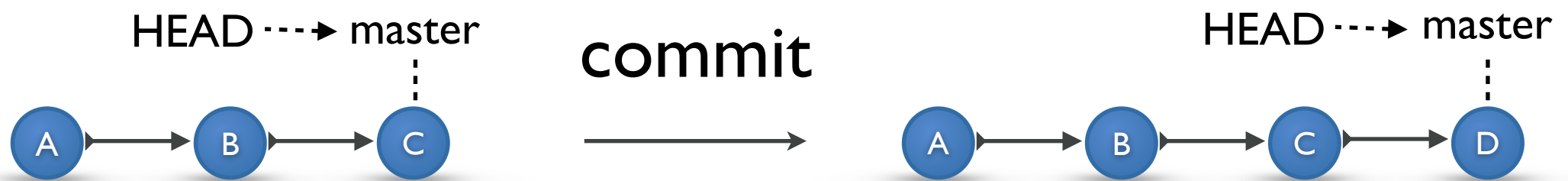
Commits können benannt sein

- ▶ Branches
- ▶ Tags
- ▶ spezielle Referenzen wie HEAD

Branches

- ▶ Branches kennzeichnen Commits
- ▶ Unterscheidung zwischen lokalen Branches ("master" oder "featureX") und remote-tracking Branches ("origin/master" oder "sopra/anotherFeature")
- ▶ genau ein Branch ist "ausgecheckt", d.h. das aktuelle Arbeitsverzeichnis enthält den Stand dieses Branches
 - dieser Branch ist durch HEAD referenziert
 - neuer Commit wird automatisch HEAD

Branches



Branches verwalten

- neuen Branch aus aktuellem HEAD:

```
$ git branch newFeature
```



Branches verwalten

- in den neuen Branch wechseln (auschecken):

```
$ git checkout newFeature
```



Branches verwalten

- beides in einem Schritt:

```
$ git checkout -b newFeature
```



Branches verwalten

- neuer Commit:

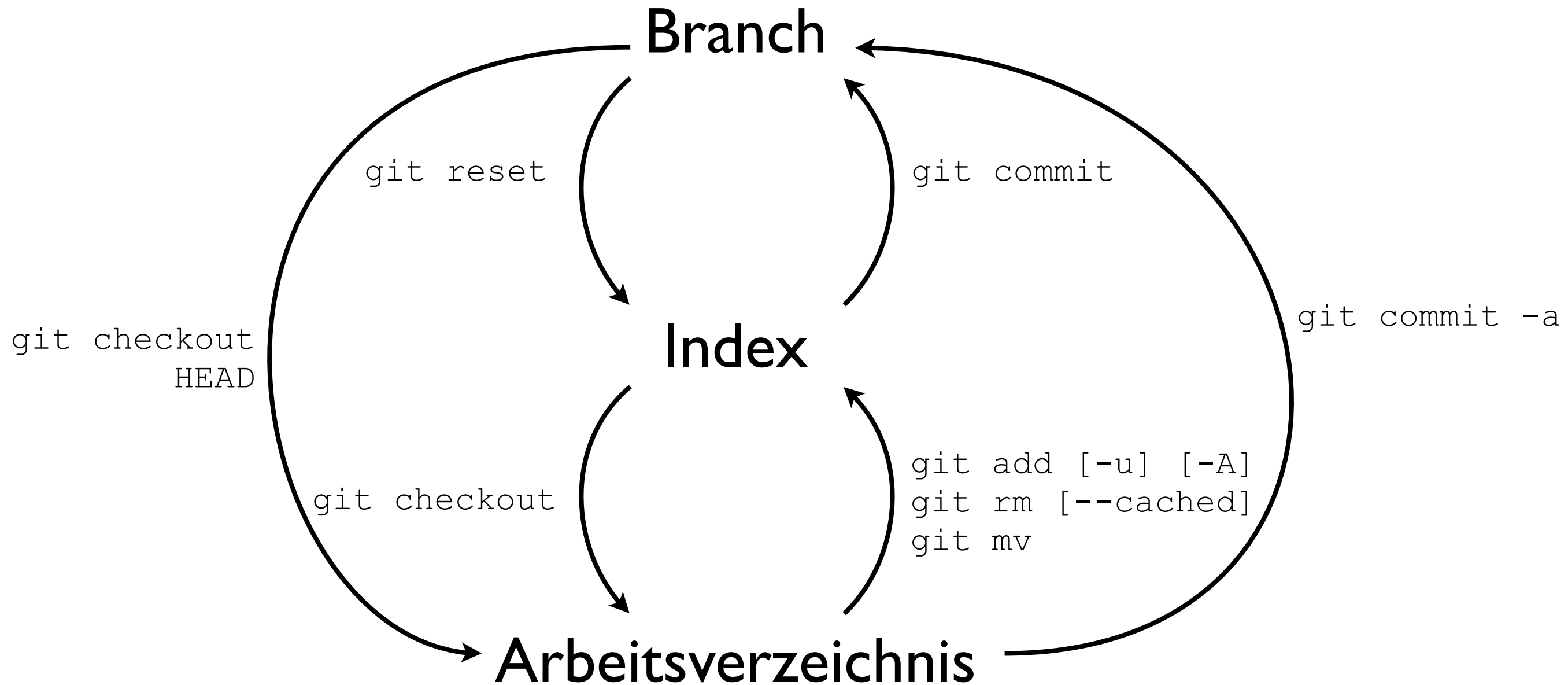
```
$ git commit -m "starting new feature"
```



Commits vorbereiten

- zu committende Änderungen werden in der *Staging Area* (in git auch “Index” genannt) vorbereitet
- wichtiger Unterschied zu SVN: nicht Dateien, sondern einzelne Änderungen werden gestaged!
- Zustand des Index überprüfen:
`$ git status`
- vorbereitete Änderungen committen:
`$ git commit -m "fixed bug #23"`

Commits vorbereiten



Commits vorbereiten

- Zustand des Index überprüfen:

```
$ git status
# On branch master
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   changedFile1
#   new file:   newFile1
#   deleted:    removedFile1
#
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   changedFile1
#   deleted:    removedFile2
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   newFile2
```


Commits vorbereiten

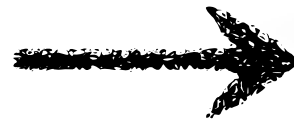
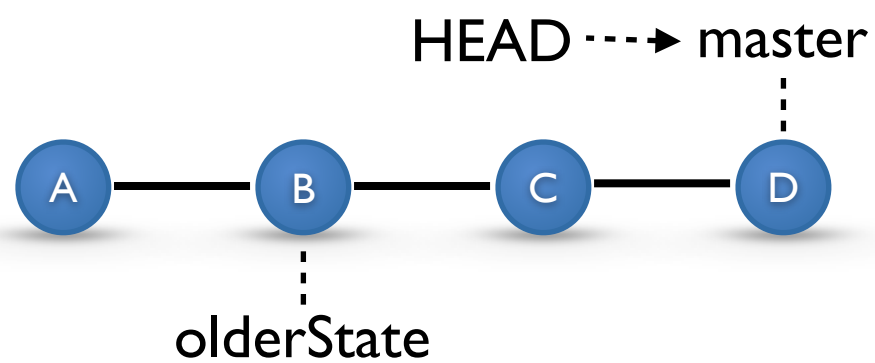
- Falls Index und/oder Arbeitsverzeichnis total durcheinander geraten:
`$ git reset --hard HEAD`
- Setzt *sowohl Index als auch Arbeitsverzeichnis* zurück auf den Stand aus HEAD
- Jede nicht committete Änderung ist verloren!

Lokales Repository anlegen

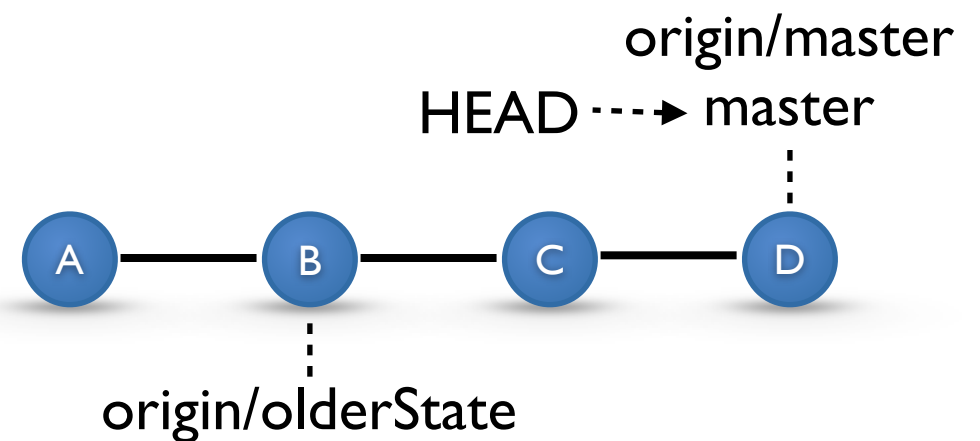
- erste Möglichkeit:
 - leeres Repository erzeugen:
`$ git init`
 - Root-Commit erzeugen:
`$ git add file1`
`$ git commit -m "initial commit"`
- zweite Möglichkeit:
 - Repository klonen:
`$ git clone sopra:gruppe8`

Repository Klonen

Remote



Lokal



Remotes

- beliebig viele Remotes definierbar
- Standardname nach dem Klonen: origin

- Remote hinzufügen:

```
$ git remote add sopra sopra:gruppe8
```

```
$ git remote add backup github:mySopraBackup
```

- Remote löschen:

```
$ git remote rm backup
```

- Remote umbenennen:

```
$ git remote rename backup github
```

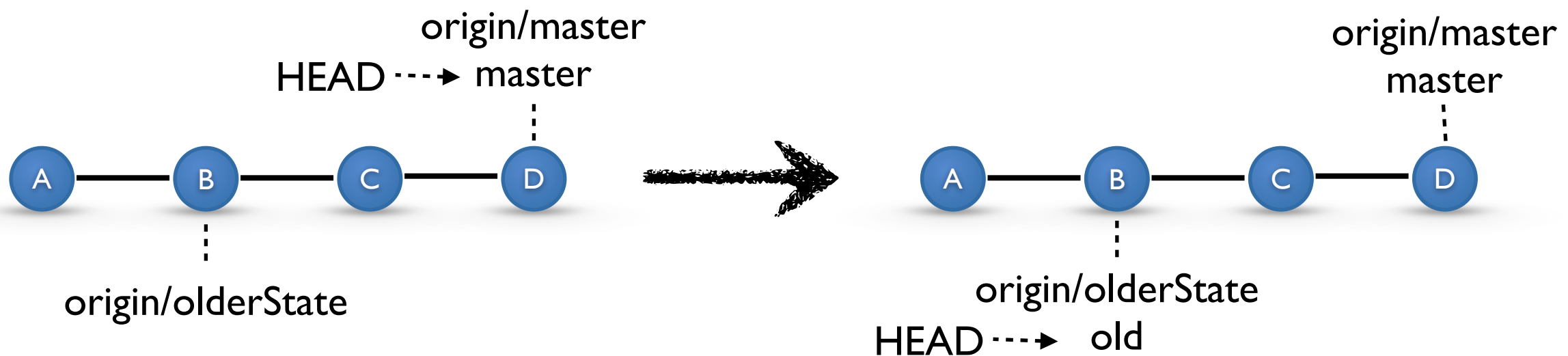
Remotes

- git unterstützt viele Protokolle, um Remotes anzusprechen:
 - ssh
 - git
 - http / https
 - ftp / ftps
 - rsync
 - file
- alle Details in den git manpages:
`$ git help clone`

Remote-Tracking Branches

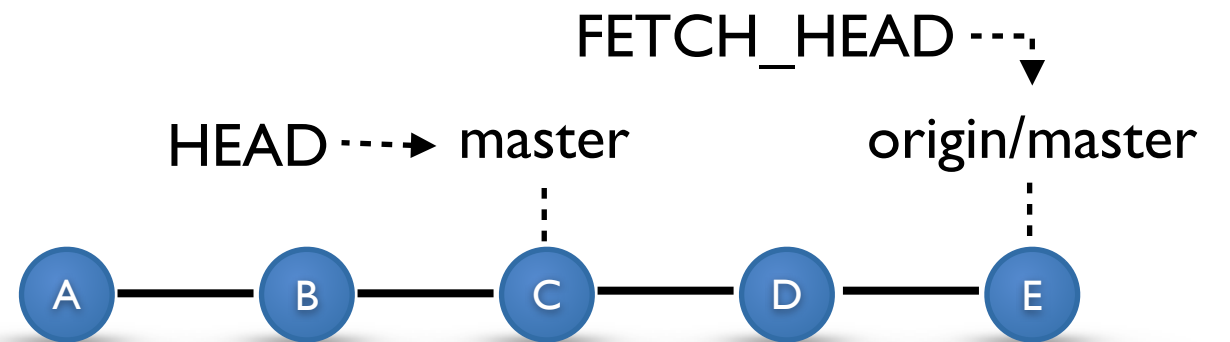
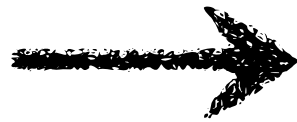
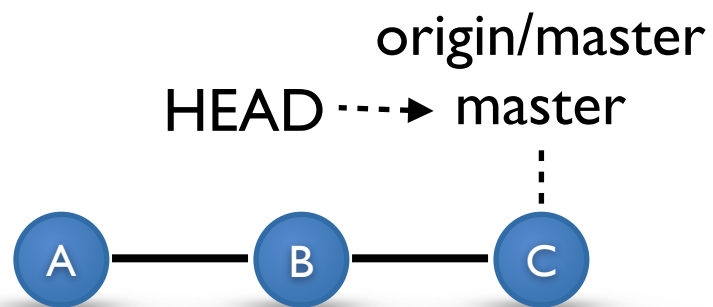
- Remote-Tracking Branch auschecken:

```
$ git checkout -t origin/olderState old
```



Änderungen übertragen

- Remote updaten:
`$ git fetch origin`



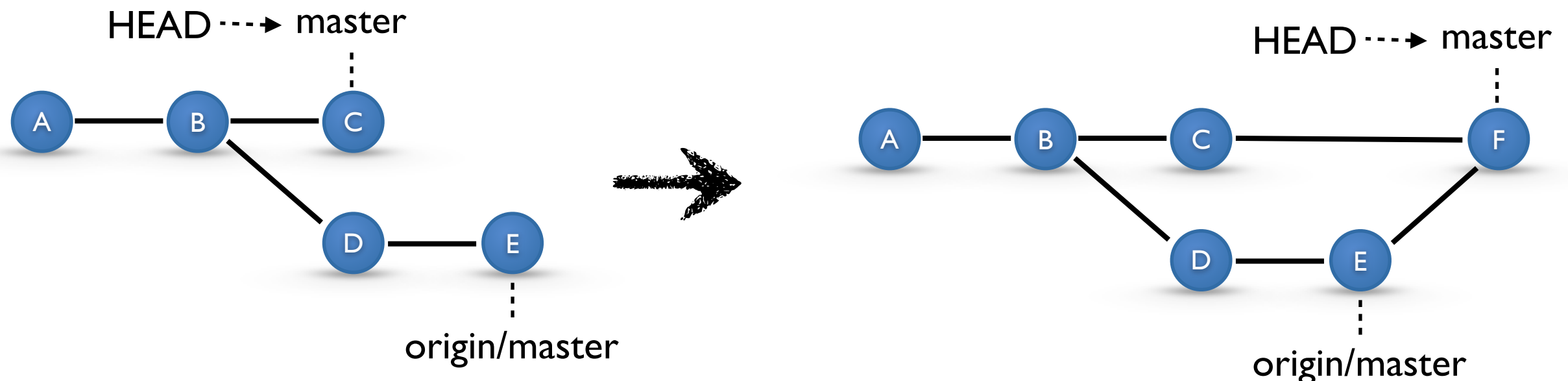
Änderungen übertragen

- Änderungen übernehmen (mergen):
`$ git merge origin/master`
- einfacher Fall: fast-forward
aktueller HEAD ist Vorfahre des zu übernehmenden Commits



Änderungen übertragen

- Änderungen übernehmen (mergen):
`$ git merge origin/master`
- schwieriger: echter merge
aktueller HEAD ist *kein* Vorfahre des zu übernehmenden Commits



Änderungen übertragen

- Remote updaten und Änderungen übernehmen in einem Schritt:

```
$ git pull origin/master
```

- äquivalent zu:

```
$ git fetch origin
```

```
$ git merge origin/master
```

Upstream Branches

- Fehler vermeiden und Tipparbeit sparen:
“upstream” branches setzen
- gilt anschließend als Standard für pull,
merge, ...

```
$ git merge
```

anstatt

```
$ git merge origin/master
```

Upstream Branches

- **durch viele Befehle automatisch gesetzt:**

```
$ git checkout -t origin/olderState old
```

```
$ git checkout olderState
```

```
$ git clone sopra:gruppe8
```

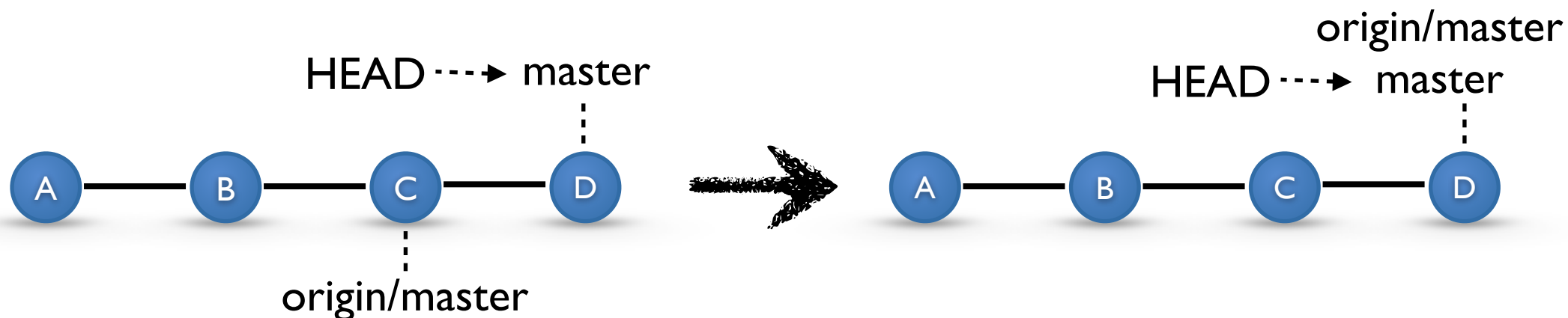
- **manuelle Konfiguration:**

```
$ git config branch.myFeature.remote github
```

```
$ git config branch.myFeature.merge myFeature
```

Änderungen übertragen

- HEAD in ein Remote übertragen:
`$ git push origin master:master`
- wenn upstream entsprechend gesetzt ist:
`$ git push`



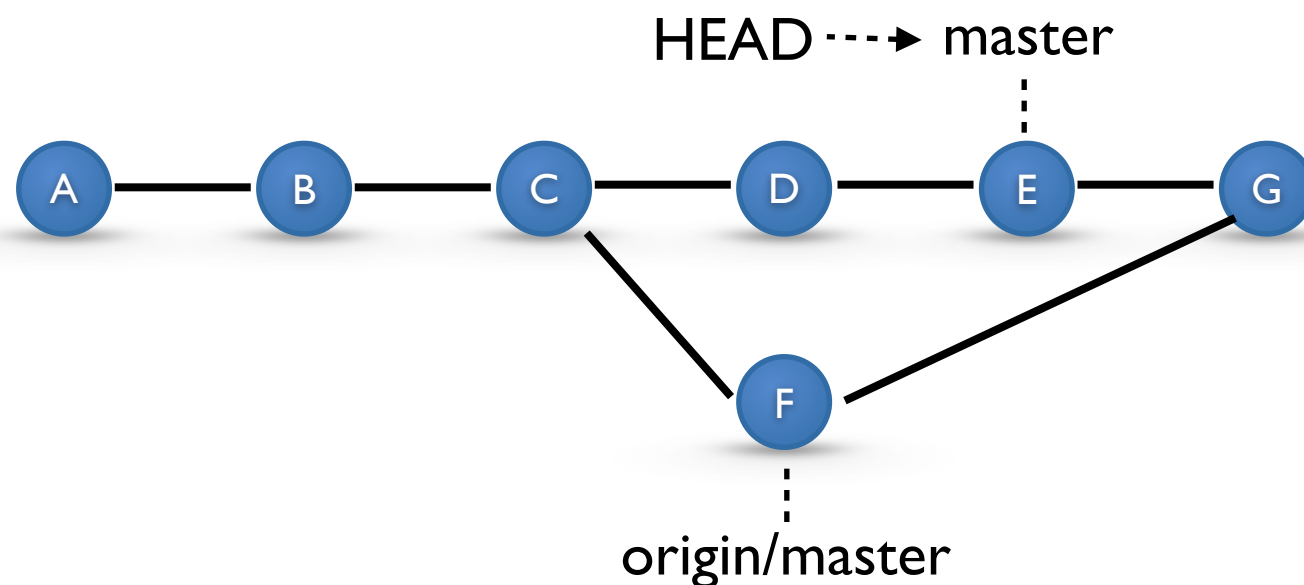
Änderungen übertragen

- abgewiesen, wenn lokaler Branch kein Nachfahre des Remote Branches:

```
$ git push
```

```
! [rejected]
```

```
master -> master  
(non-fast-forward)
```

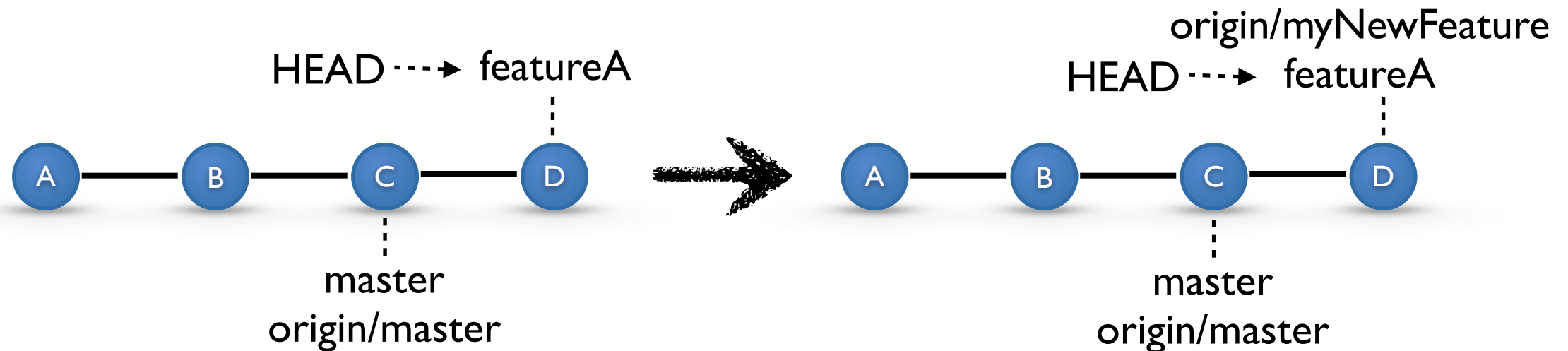


- in diesem Fall: erst pull (erzeugt merge)

Änderungen übertragen

- bestimmte Branches übertragen:

```
$ git push origin featureA:myNewFeature
```



Änderungen übertragen

- Remote Branches löschen:
`$ git push origin :myNewFeature`

Konflikte auflösen

- bei jedem Merge können Konflikte auftreten
 - in beiden Branches wurde eine Datei an der gleichen Stelle geändert
 - eine Datei wurde in einem Branch modifiziert, aber im anderen gelöscht
 - beide Branches haben eine Datei mit dem gleichen Namen angelegt

Konflikte auflösen

- **git meldet Konflikte:**

```
$ git pull
[...]  
Auto-merging file1  
CONFLICT (add/add): Merge conflict in file1  
Automatic merge failed; fix conflicts and then  
commit the result.  
$
```

- **Konflikte von Hand beheben und committen:**

```
$ vi file1  
$ git add file1  
$ git commit
```

Tags

- sehr ähnlich zu Branches
- kennzeichnen beliebige Commits
- werden i.d.R. nicht mehr verändert
- potenziell GPG-signiert werden
- potenziell mit Nachricht versehen
- `git tag r1.0 -m "release 1.0"`

Commits wiederfinden

- ▶ Branch-Namen: master, origin/featureA
- ▶ Tag-Namen: r0.8
- ▶ SHA1-Hash, oder einen eindeutigen Präfix davon: 3c4dd2
- ▶ Datums-Angabe: master@{1 month ago}
- ▶ letzte Referenzen eines Branches: master@{5}
- ▶ Parents eines Branches/Tags/Commits:
HEAD^, master^^, master~4

Mehr dazu in der Git manpage.

Änderungen anzeigen

- zwischen Arbeitsverzeichnis und Index:

```
$ git diff
```

- zwischen Index und HEAD:

```
$ git diff --cached
```

- zwischen Arbeitsverzeichnis und beliebigem Commit:

```
$ git diff master@{yesterday}
```

- zwischen beliebigen Commits:

```
$ git diff HEAD HEAD^^
```

Stammbaum anzeigen

- listet Vorfahren von HEAD mit Autor, Datum und Nachricht:

```
$ git log
```

- Stammbaum eines beliebigen Commits:

```
$ git log origin/featureA
```

Commits wiederherstellen

- gewünschten Commit auschecken:

```
$ git checkout 2d3c44
```

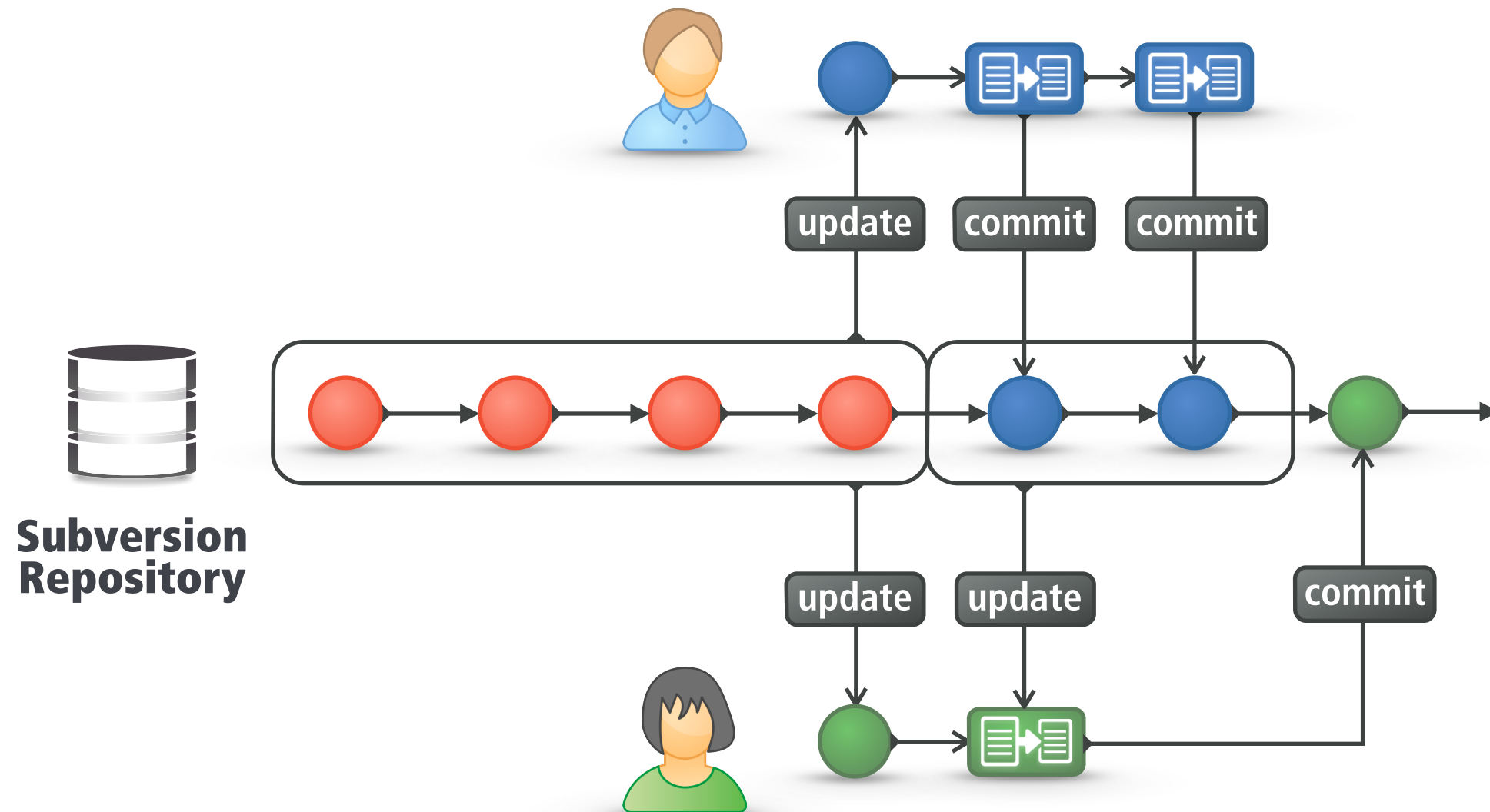
- neuen Branch anlegen:

```
$ git branch lostState
```

- alten Branch überschreiben:

```
$ git branch -f master
```

Zentrale Versionskontrolle



Dezentrale Versionskontrolle

