

Git und Maven

- ▶ Git
 - Installation
 - arbeiten mit Repositories
- ▶ Maven (mit Eclipse)
 - Abhängigkeiten einrichten
 - ausführbare jar-Archive erzeugen

Git

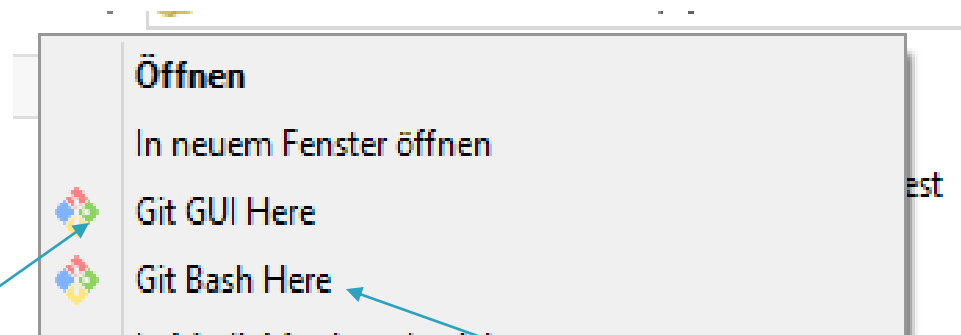
- ▶ Git ist ein Revision Control System:
 - Es speichert mehrere Entwicklungsstände eines größeren Software-Projektes, die jederzeit wiederhergestellt werden können.
 - Es kann dabei mehrere Entwicklungszweige gleichzeitig verwalten.
 - Es hilft, die Arbeit mehrerer Programmierer am gleichen Software-Projekt zu koordinieren.
- ▶ Git dient eigentlich nicht nur dem Austausch von Dateien (→ DropBox u.ä.)

Git-Installation

- ▶ Linux:
 - Git sollte in jeder Distribution vorhanden sein.
- ▶ Windows:
 - <https://git-for-windows.github.io/>
- ▶ Mac:
 - <https://git-scm.com/download/mac>
- ▶ Update-Seite für Eclipse-Plugin Egit:
 - <http://download.eclipse.org/egit/updates>

Git verwenden

- ▶ Nach der Installation von Git gibt es im Kontextmenü der Ordner im Windows Explorer die beiden Einträge
 - Git GUI Here
 - Git Bash Here



graphische
Oberfläche für die
häufigsten Befehle

Konsole mit
mehr
Möglichkeiten

"sich anmelden"

- ▶ Damit klar ist, von wem Dateien kommen/verändert wurden, muss man git seinen Namen mitteilen:
- ▶ `git config --global user.name`
`'Dein Name'`
- ▶ `git config --global user.email`
`'Deine Email'`

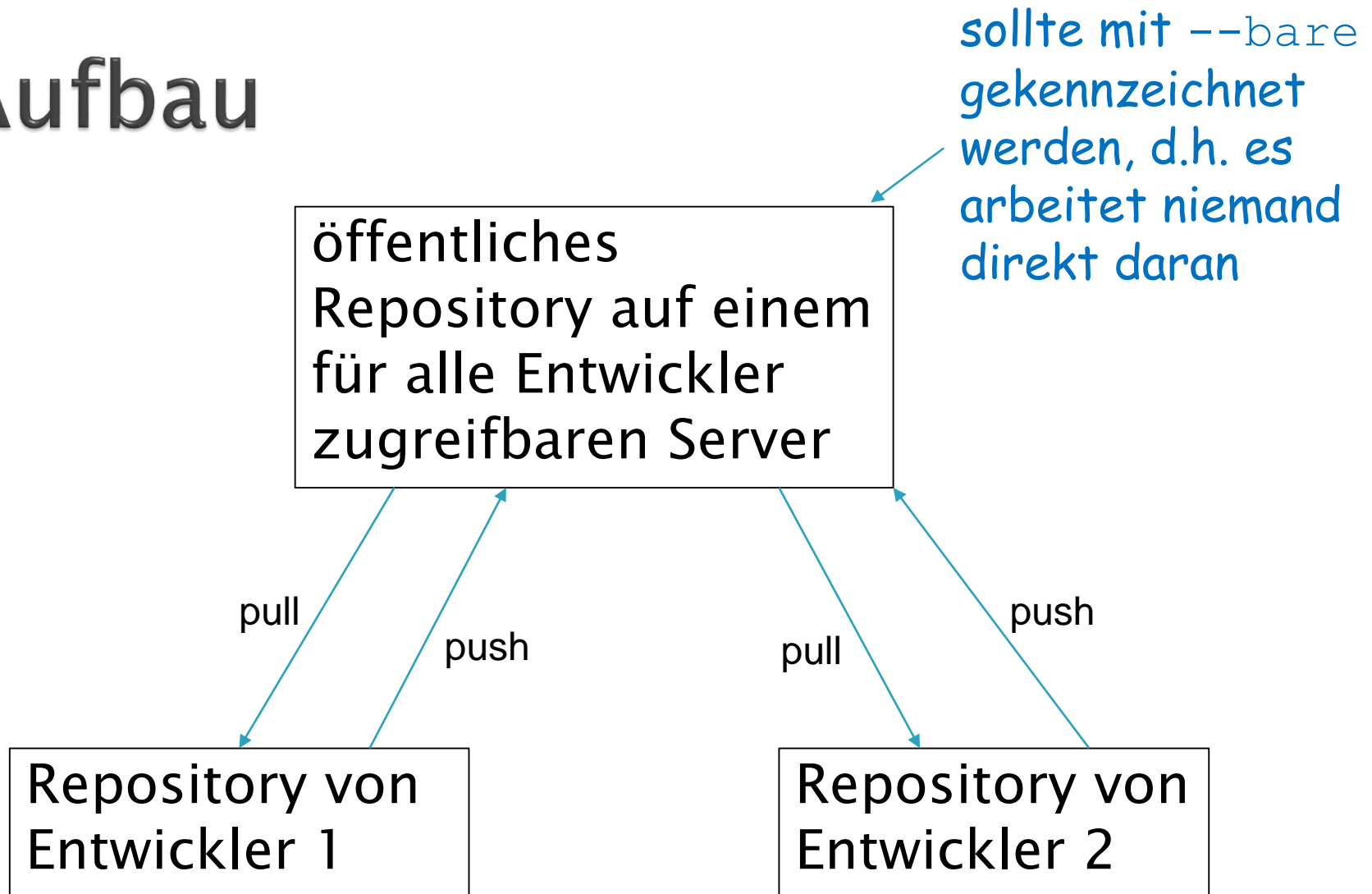


ohne `--global` gelten die
Angaben nur für das
aktuelle Repository

Fachbegriffe

- ▶ Repository = Ordner, den Git verwaltet.
 - gearbeitet wird grundsätzlich in einer lokalen Arbeitskopie des für alle Entwickler zugreifbaren Repositories
- ▶ Commit = Ein gespeicherter Stand der Dateien im Repository
 - Sinnvollerweise ist ein Commit eine in sich geschlossene, lauffähige Version eines Projektes.
- ▶ Branch = Entwicklungszweig, mehrere aufeinanderfolgende Commits
 - Der Hauptzweig trägt den Namen `master`.

Aufbau



Server-Repository erstellen

▶ In der Git Bash:

- ggf. kann man immer mit `cd "ordner"` vorher in den gewünschten Speicherordner für das Repository wechseln

- `git init --bare`

falls das Repository von anderen Rechnern aus zugreifbar sein soll, muss `git daemon` gestartet werden

- ▶ Es wird ein leeres Server-Repository erstellt.
- ▶ Bitte NICHT selber an den Ordnern und Dateien ändern!

auch in den anderen Repositories nicht...

Server vom FB4 der HTW nutzen

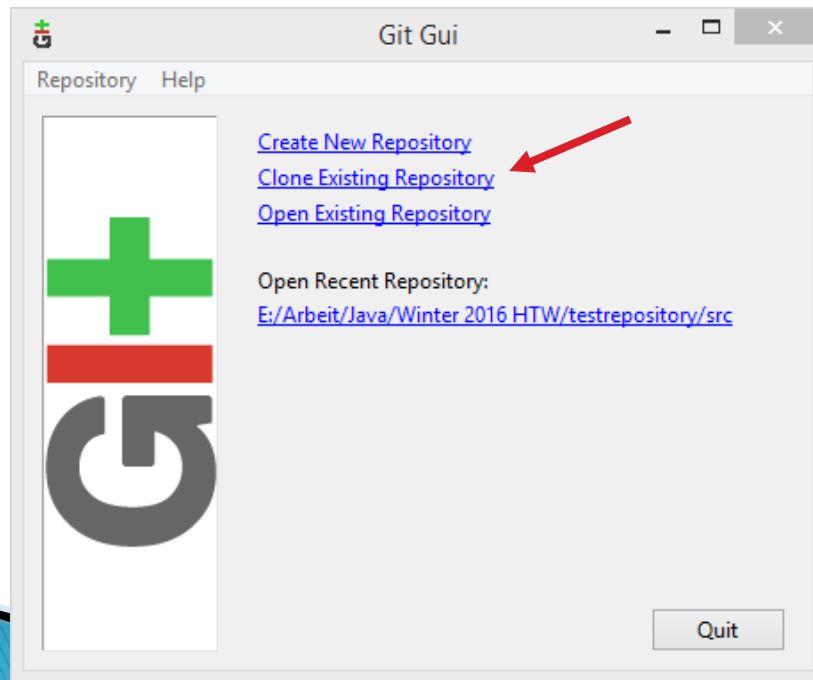
- ▶ Einloggen auf <https://studi.f4.htw-berlin.de/www/login/?next=/www/services/git/>
- ▶ Neues Repository erstellen
- ▶ Berechtigte Benutzer hinzufügen
 - damit auch andere Entwickler das Repository nutzen können und gemeinsam entwickelt werden kann

Eigenes Repository (Arbeitskopie)

► In der Git Bash

- `git clone "PfadZumServerrepository"`
zielordner

► Git GUI:




Das eigene Repository speichert den Pfad zum Server in der Variablen origin, was die späteren Aktionen leicht macht

Vorsicht!

- ▶ Bevor der zweite Entwickler das Serverrepository klonet, sollte es mindestens eine Datei enthalten, die der erste Entwickler hochlädt (gleich), sonst klappt es hinterher nicht!
 - Fehlermeldung: refusing to merge unrelated histories
- ▶ Beim Repository auf dem HTW-Server ist das automatisch der Fall.

oder mit Eclipse

- ▶ File->Import->Git->Projects from Git
 - Geben Sie in den Dialogen den Pfad zum Serverrepository an,
 - wählen Sie einen lokalen Speicherort für das lokale Repository aus
 - und achten Sie beim Speichern eines existierenden oder neuen Projektes darauf, dass es im lokalen Repository-Ordner liegt. 
- sonst ziemlich sinnlose Fehlermeldung...

im Repository arbeiten

- ▶ Im eigenen Repository-Ordner können Dateien hinzugefügt, geändert und gelöscht werden wie immer.
- ▶ Soll ein Entwicklungsstand veröffentlicht werden, sind folgende Schritte auszuführen:
 1. Index updaten
 2. Commit ausführen
 3. Zusammenführung verschiedener Entwicklungsstände
 4. Daten zum Server pushen

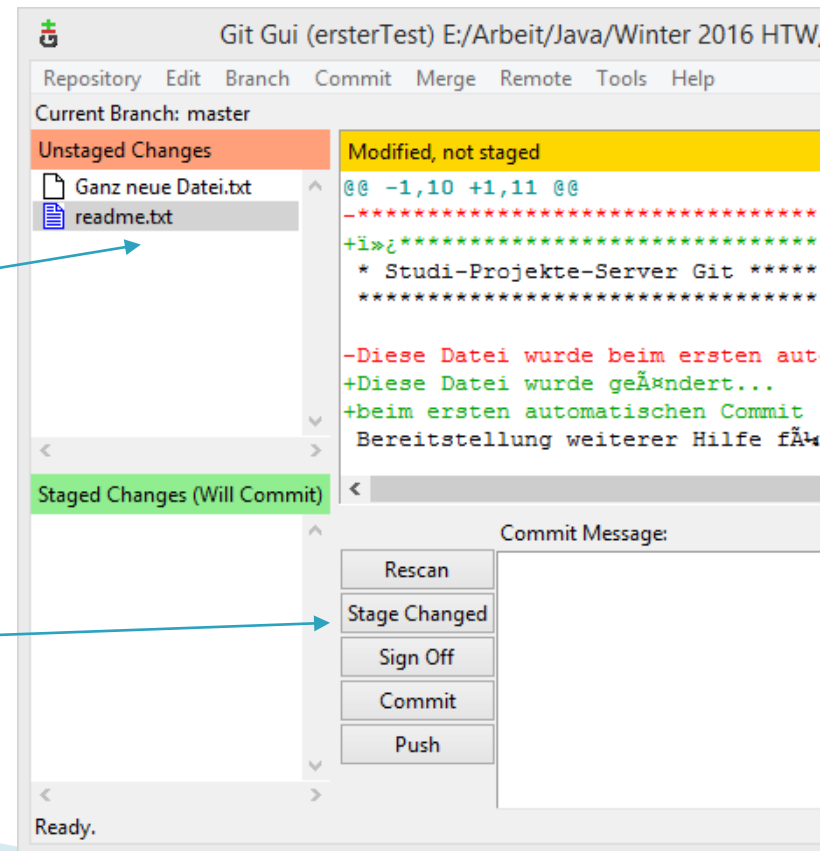
1. Index updaten

► Git Bash:

- `git add dateiname`
 - für jede neue oder geänderte Datei
- `git rm dateiname`
 - für jede gelöschte Datei

► Git GUI: alle Änderungen im Repository-Ordner, ggf. aktualisieren mit F5

Änderungen übernehmen = Index updaten



2. Commit

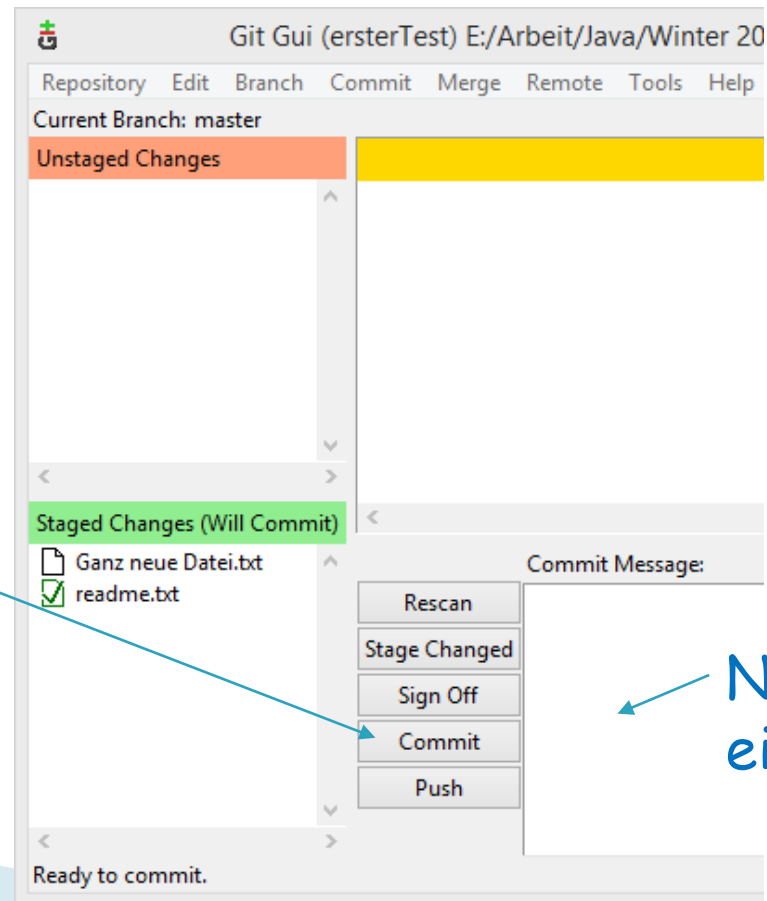
▶ Git Bash:

- `git commit --message "eine Nachricht"`

▶ Git GUI

dran denken: In
sich
abgeschlossene,
lauffähige neue
Version des
Projektes!

Beschreibung,
was neu ist



Nachricht
eintragen

2a Commit zurücknehmen

- ▶ Falls sich doch Fehler eingeschlichen haben, kann man den aktuellen Entwicklungsstand auf einen früheren Commit zurücksetzen:
- ▶ Git Bash:
 - `git reset -hard commitId`
- ▶ einfacher:
 - `gitk`
 - öffnet eine graphische Übersicht aller Branches und Commits
 - im Kontextmenü gibt es den Befehl *Reset master Branch to here*

mit

`git log`

erhält man eine Liste aller bisherigen Commits inklusive ihrer Id

3. Zusammenführung

- ▶ Es kann sein, dass die Dateien auf dem Server inzwischen von anderen Entwicklern verändert wurden.
 - spätestens beim Push kriegt man dann eine Fehlermeldung.

3. Zusammenführung

▶ Git GUI:

- Remote -> fetch From origin
 - holt die neuen Dateien vom Server in einen neuen Branch namens origin/master
- Branch -> Checkout... -> origin/master
 - ermöglicht es, die neuen Dateien zu betrachten. Mit dem gleichen Befehl kommt man zu seinem eigenen Branch zurück. -> für Merge nicht zwingend notwendig
- Merge -> Local Merge... -> origin/master -> Merge
 - führt den eigenen Branch mit dem auf dem Server zusammen. Konflikte werden angezeigt und müssen von Hand gelöst werden.

Visualize statt Merge zeigt die durchzuführenden Änderungen nur an

3. Zusammenführung

- ▶ Achtung beim Merge:
 - Wenn Sie Änderungen am Code vornehmen, müssen Sie das ganze Projekt noch einmal TESTEN, bevor sie die Änderungen durch einen Commit bestätigen.

Ganz in echt! Wirklich!
Immer!

- ▶ Danach noch einmal
Index updaten und commit
ausführen.

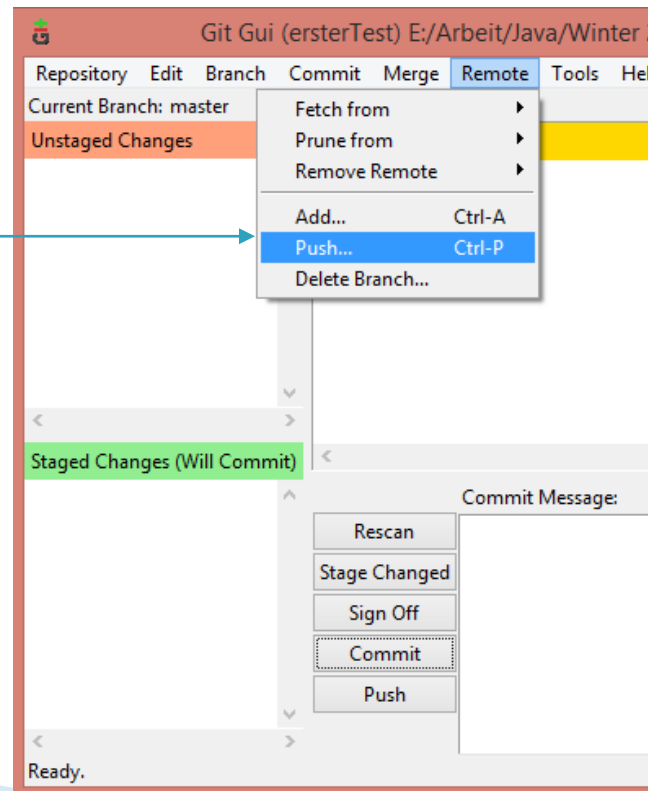
4. Daten zum Server übertragen

▶ Git Bash:

- `git push origin`

oder Adresse eines
anderen Server-
Repositorys

▶ Git GUI:



im nächsten
Schritt kann ggf.
ein anderes
Server-
Repository
gewählt werden

oder mit Eclipse

- ▶ Kontextmenü des Projektes:
 - Team->Commit
 - Im sich öffnenden Fenster kann man den Index updaten, den Commit ausführen und die Dateien zum Server pushen
- ▶ Wenn man noch kein Repository eingerichtet hat:
 - Team->Share Project

Daten vom Server holen

- ▶ Hat man keine eigenen Änderungen seit dem letzten push durchgeführt, kann man die eigenen Daten ohne komplizierten Merge aktualisieren:

- ▶ Git Bash:

- `git pull`

Immer machen,
bevor man
weiterarbeitet!

- ▶ Git GUI – siehe Zusammenführung:

- Remote -> fetch From origin
 - Merge -> Local Merge... -> origin/master -> Merge

oder mit Eclipse

- ▶ Team -> Pull
- ▶ Oder auch ein ganz neues Projekt:
 - New->Import...->Projects From Git

Branch

- ▶ Ein Entwicklungszweig (Branch) besteht aus mehreren aufeinanderfolgenden Commits.
- ▶ Es gibt immer einen ersten Branch namens `master`.
- ▶ Daneben kann man weitere Branches anlegen.
 - Irgendwann müssen die verschiedenen Branches mit Merge zusammengeführt werden.
- ▶ Einer der Entwicklungszweige ist immer aktuell.
 - Der letzte Commit des aktuellen Branches heißt `HEAD`.

Branches

- ▶ `git branch name`
 - Anlegen eines neuen Branches, der eine Kopie des aktuellen darstellt
- ▶ `git checkout name`
 - Wechsel in den Branch name; das Arbeitsverzeichnis enthält dann die Dateien dieses Branches
- ▶ `git merge name`
 - Zusammenführen des aktuellen Branches mit dem angegebenen
- ▶ `gitk`
 - graphische Darstellung der bisherigen Branches und Commits

Apache Maven


- ▶ Maven ist ein Build-Management-Tool. Es unterstützt beim
 - Anlegen, Kompilieren, Testen, Packen, Verteilen
- ▶ eine Softwareprojektes, indem möglichst viele Schritte dieses Build-Ablaufs automatisiert werden.

Man spricht vom
Lebenszyklus



- ▶ Idee: Konvention vor Konfiguration

halte dich an
Konventionen, dann sparst
du die extra
Konfiguration




standardisierte Projektstruktur

- ▶ `src/main/java`:
 - Quelltextdateien
- ▶ `src/main/resources`:
 - weitere benötigte Dateien
- ▶ `src/test/java`:
 - Testklassen für Unit-Tests
- ▶ `target`:
 - alle erzeugten Dateien, insbesondere das Ziel-jar-Archiv
 - `target/classes`: übersetzte `.class`-Dateien
- ▶ `pom.xml`: Konfigurationsdatei

kann man ändern, sollte man aber nicht...
Convention over configuration

Standard– Lebenszyklus 1

- ▶ archetype:  Projekt-Vorlage
 - Es wird eine Projektstruktur vorgegeben (Ordnerstruktur, zusätzliche jar-Bibliotheken)
 - ▶ validate:
 - Projektstruktur wird überprüft
 - ▶ compile:
 - Quelltext wird übersetzt
 - ▶ test:
 - Testcode wird ausgeführt
 - ▶ package:
 - Der übersetzte Code und ggf. Zusatzressourcen werden verpackt, z.B. in ein jar-Archiv
- erweiterbar
und/oder
konfigurierbar
durch Maven-
Plugins

Standard-Lebenszyklus 2

- ▶ integration-test:
 - Das Paket wird in eine Testumgebung geladen und ausgeführt
- ▶ verify:
 - Überprüfung weitere Qualitätskriterien
- ▶ install:
 - Paket ins lokale Maven-Repository verschieben
- ▶ deploy:
 - Paket ins öffentliche Maven-Repository verschieben und damit allen zugänglich machen

Management von Abhängigkeiten

- ▶ Benötigte zusätzliche Bibliotheken lädt man nicht von Hand herunter und fügt sie selbst dem Classpath hinzu, sondern teilt Maven mit, dass sie gebraucht werden (mit genauem Namen und Versionsnummer).
- ▶ Maven lädt die jar-Dateien dann aus einem öffentlichen Repository und setzt alle Pfade richtig.

Fachbegriffe

- ▶ **Artefakt:**

Das Ziel-Produkt, meist ein jar- oder ein war-Archiv

- ▶ **POM:**

- Project Object Model; Konfigurationsdatei eines Maven-Projektes; basiert auf der Super-POM

- ▶ **Maven-Plugin:**

- Ein Programm, das einen Teil des Build-Prozesses durchführt oder um zusätzliche Aktionen erweitert

- ▶ **Goal:**

- Kommando an ein Plugin

- ▶ **Dependency:**

Im Programm benötigte zusätzliche Ressourcen (Bibliotheken)

Maven-Projekt anlegen

- ▶ File->New->Other...->Maven->Maven Project
 - Häkchen bei "Create Simple Project" oder im nächsten Schritt Archetype wählen
 - Group Id: Hersteller-Bezeichnung, üblicherweise umgedrehter Firmen-Domainname
 - Artifact Id: Bezeichnung für das Produkt
 - Version: Versionsnummer; -SNAPSHOT sagt, dass diese Versionsnummer noch in der Entwicklung ist.

GAV - gemeinsam müssen diese 3 Angaben das Produkt eindeutig identifizieren, sonst kann man es im öffentlichen Repository nicht finden

Maven-Projekt builden

► Im Kontextmenü des Projektes:

- Run As...->Maven ...

Hier Phase auswählen, bis zu der der Build ausgeführt werden soll

- oder:

- Run As...->Maven build...


- im Dialog unter Goal Ziel-Phase bzw. Kommando für ein Maven-PlugIn eintragen

z.B. compile, test, package, install...

Abhängigkeiten

- ▶ pom.xml öffnen
 - Registerkarte Dependencies
 - Add
 - benötigte Bibliothek/PlugIn auswählen
- ▶ Projekt->Maven->Update Project
 - lädt Bibliotheken herunter und übernimmt sie in den Build-Path

meist
zusätzliche jar-
Archive; genauer
Name siehe
Dokumentation



Bibliotheken mit einpacken

► In pom.xml einfügen:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-
        plugin
      </artifactId>
      <version>2.6</version>
      <configuration>
        <descriptorRefs>
          <descriptorRef>
            jar-with-dependencies
          </descriptorRef>
        </descriptorRefs>
        <archive>
          <manifest>
            <mainClass>IhreKlasse
          </mainClass>
        </manifest>
      </configuration>
    </plugin>
  </plugins>
</build>
```

PlugIn maven-assembly-plugin
kann benötigte Bibliotheken
mit einpacken

```
</archive>
</configuration>
<executions>
  <execution>
    <id>make-assembly</id>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
</build>
```

ausführbares jar-
Archiv bauen mit
dieser Startklasse

Download Maven

- ▶ <http://maven.apache.org/download.cgi>
- ▶ Update-Site für M2Eclipse-Plugin:
 - <http://download.eclipse.org/technology/m2e/releases/>

Literatur

- ▶ Git-Tutorial auf den HTW-Servern:
 - <https://studi.f4.htw-berlin.de/www/help/tutorial/git/>
- ▶ Git-Man-Pages:
 - <https://www.kernel.org/pub/software/scm/git/docs/user-manual.html>

Literatur

- ▶ Offizielle Seite von Maven:
 - <https://maven.apache.org>
 - Tutorial und Referenzen:
 - <https://maven.apache.org/users/index.html>
- ▶ Maven-Tutorial:
 - <http://www.torsten-horn.de/techdocs/maven.htm>
 - <http://invidit.de/blog/mit-maven-raus-aus-der-abhaengigkeit/>
- ▶ Log4J
 - <http://logging.apache.org/log4j/2.x/>