

---

# Übungsprotokoll A04

## Continuous Integration mit Jenkins

---

Softwareentwicklung  
5BHIT 2017/18

Martin Wölfer

Note:  
Betreuer: DOLD & RAFW

Version 0.1  
Begonnen am 5. Oktober 2017  
Beendet am 13. Oktober 2017

# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabenstellung</b>	<b>1</b>
1.1	Grundanforderungen (70%) . . . . .	1
1.2	Erweiterungen (30%) . . . . .	1
<b>2</b>	<b>Installation und Konfiguration</b>	<b>2</b>
2.1	Erstes Problem mit VM . . . . .	2
2.2	Installation in Windows . . . . .	2
2.3	Github-User Konfiguration . . . . .	2
<b>3</b>	<b>Ersten Job erstellen</b>	<b>3</b>
3.1	Job konfigurieren . . . . .	3
3.1.1	Woher Job Quelle bezieht . . . . .	3
3.1.2	Wann Job ausgeführt wird . . . . .	4
3.1.3	Was ausgeführt wird . . . . .	4
3.1.4	Was passiert mit Ergebnissen . . . . .	4
3.2	Job ausführen . . . . .	5
<b>4</b>	<b>Warum es nicht funktioniert</b>	<b>5</b>
4.1	Sh kann nicht ausgeführt werden . . . . .	6
4.2	Reports werden nicht erzeugt . . . . .	7

# 1 Aufgabenstellung

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage." M.Fowler

Lass das Bruch-Projekt mithilfe von Jenkins automatisch bei jedem Build testen!

## 1.1 Grundanforderungen (70%)

- Installiere auf deinem Rechner bzw. einer virtuellen Instanz das Continuous Integration System Jenkins
- Installiere die notwendigen Plugins für Jenkins (Violations, Cobertura)
- Installiere Nose, Coverage und Pylint (mithilfe von pip)
- Integriere dein Bruch-Projekt in Jenkins, indem du es mit Git verbindest
- Überlege dir und argumentiere eine sinnvolle Pull-Strategie
- Konfiguriere Jenkins so, dass deine Unit Tests automatisch bei jedem Build durchgeführt werden inkl. Berichte über erfolgreiche / fehlgeschlagene Tests und Coverage
- Protokolliere deine Vorgehensweise (inkl. Zeitaufwand, Konfiguration, Probleme) und die Ergebnisse (viele Screenshots!)

## 1.2 Erweiterungen (30%)

- Konfiguriere und teste eine Git-Hook, sodass Änderungen auf GitHub automatisch einen Build auslösen! Dokumentiere deine Vorgangsweise (mit Screenshots)!
- Recherchiere, wie mithilfe von Jenkins GUI-Tests durchgeführt werden können und baue selbstständig einen Beispiel-GUI-Test ein! Dokumentiere deine Vorgangsweise (mit Screenshots)!
- Lass deine Sphinx-Dokumentation automatisch mitbuilden und veröffentlichen! Dokumentiere deine Vorgangsweise (mit Screenshots)!

## 2 Installation und Konfiguration

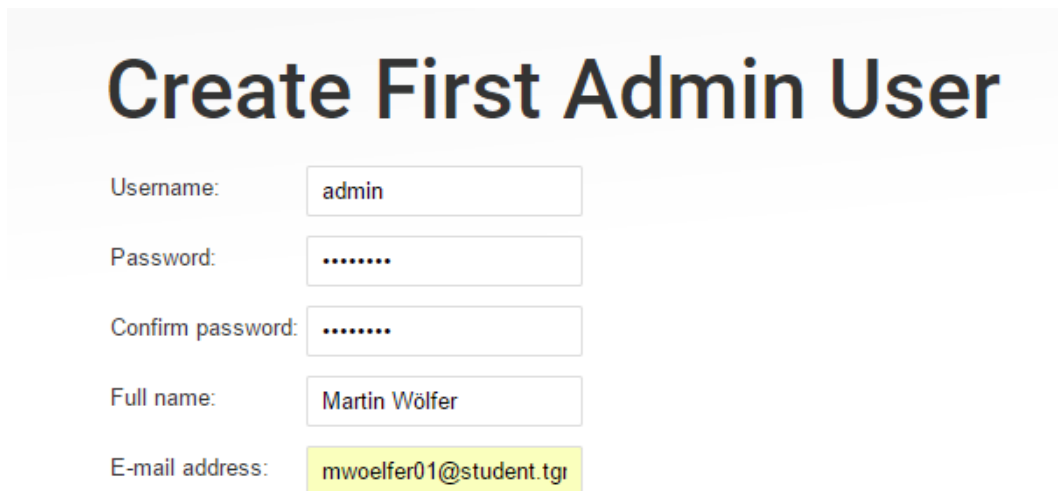
Der erste Schritt war es Jenkins zu installieren.

### 2.1 Erstes Problem mit VM

Bei der Installation ist mein erstes Problem aufgetreten, da ich dachte dass es einfacher wäre Jenkins in einer virtuellen Maschine laufen zu lassen. Dies bedeutet es wurde Jenkins installiert, was recht einfach lief dadurch dass Jenkins im apt-get Repository liegt, und die Konfiguration funktionierte dank Tutorial auch flüssig. Nun ist mir aufgefallen, vor allem beim Plugins installieren, dass die VM sehr langsam ist bzw. bei manchen Plugins garnicht funktioniert - noch dazu kommt dass ich das gesamte Git-Repository auf die virtuelle Maschine kopieren musste da Jenkins die Dateien lokal bezieht. Nach längerer Frustration bin ich schlussendlich auf Windows umgestiegen.

### 2.2 Installation in Windows

Die Installation in Windows läuft auch einfach ab, man installiert das .msi file, öffnet `localhost:8080` und geht den Installations-Wizard durch. Es wurden die richtigen Plugins angewählt bei der Erstkonfiguration (Außer Jenkins Violations da dieses erst im Dashboard installiert werden kann) und ein Admin user wurde angelegt:



The screenshot shows the 'Create First Admin User' form in Jenkins. The form has the following fields and values:

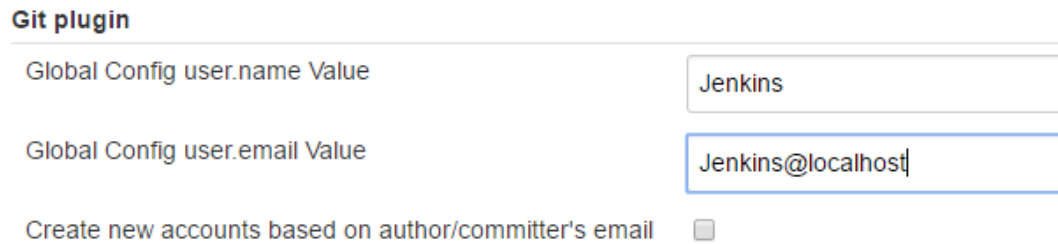
Field	Value
Username:	admin
Password:	.....
Confirm password:	.....
Full name:	Martin Wölfer
E-mail address:	mwoelfer01@student.tgr

Abbildung 1: Ein Admin User wurde erstellt

Anschließend wurde unter **Manage Jenkins** → **Manage plugins** → **Available** auch noch das letzte fehlende Plugin installiert (Jenkins Violations).

### 2.3 Github-User Konfiguration

Unter **Manage Jenkins** → **Configure System** → **Git Plugin** werden die Daten eingegeben.



**Git plugin**

Global Config user.name Value

Global Config user.email Value

Create new accounts based on author/commmitter's email ☐

Abbildung 2: Daten werden eingegeben für Git

### 3 Ersten Job erstellen

Im Dashboard wird einem gleich angeboten einen Job zu erstellen, danach wird ein Name für den Job eingegeben und es wird **Freestyle Project** ausgewählt



**Enter an item name**

[» Required field](#)

 **Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, co

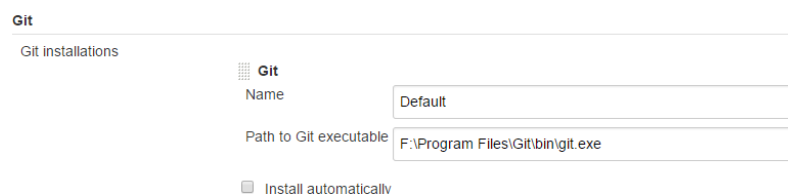
Abbildung 3: Neuen Job erstellen

#### 3.1 Job konfigurieren

##### 3.1.1 Woher Job Quelle bezieht


Zuerst muss Jenkins mitgeteilt werden wo die Quelle des Projektes überhaupt liegt, es wird zuerst unter der Section **Source Code Management** der Button **Git** angewählt, und anschließend der lokale Repository URL angegeben werden.

Hier bin ich auf ein weiteres Problem gestoßen, und zwar wurde die Fehlermeldung ausgegeben: **jenkins failed to connect to repository**. Dies lag daran dass Jenkins nicht wusste wo sich lokal mein **git.exe** befindet, daher musste ich in **Manage Jenkins** → **Global Tool Configuration** → **Git** → **Git Installations** den Pfad zu **git.exe** angeben:



**Git**

Git installations

 **Git**

Name

Path to Git executable

☐ Install automatically

Nun konnte der lokale Pfad zum Repository angegeben werden ohne dass ein Problem besteht:

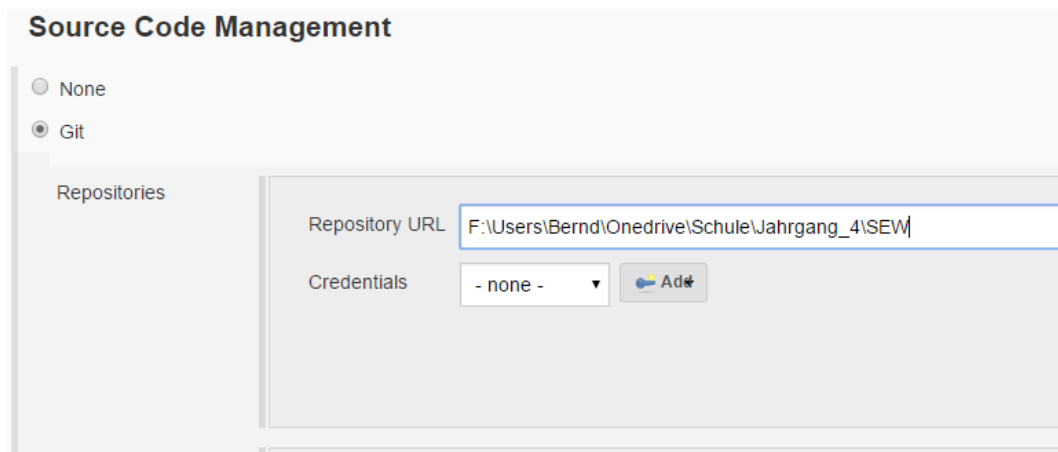
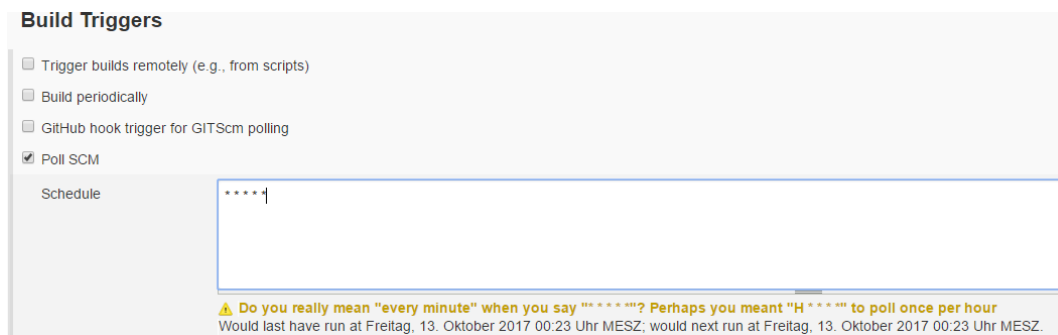


Abbildung 4: Pfad konnte angegeben werden ohne Fehlermeldung

### 3.1.2 Wann Job ausgeführt wird

Nun wird unter Build Triggers → Poll SCM angegeben wann der Job ausgeführt wird, indem 5 \* angegeben werden:



### 3.1.3 Was ausgeführt wird

Unter Build → Add build step → Execute Shell wird nun folgender Code angegeben:

```
1 PYTHONPATH=''
  nosetests --with-xunit --all-modules --traverse-namespace --with-coverage --cover-package=project1 --
    cover-inclusive
3 python -m coverage xml --include=project1*
  pylint -f parseable -d I0011,R0801 project1 | tee pylint.out
```

### 3.1.4 Was passiert mit Ergebnissen

In diesem Schritt wird nun entschieden wie die Ergebnisse analysiert werden. In diesem Fall werden sie interpretiert durch

- Coverage

- JUnit testing
- Report Violations

Unter **Post-build Actions** → **Publish Cobertura Coverage Report** wird der Link zum Coverage File angegeben. Es wird lediglich `coverage.xml` angegeben, weil dieses durch den Code im Schritt davor automatisch erzeugt wird.

Als nächstes werden die JUnit Test results hinzugefügt. Dazu unter **Post-build Actions** → **Publish JUnit test result report** wieder den Filenamen angegeben vom Code erzeugten File, `nosetest.xml`.

Der letzte Schritt ist es Report Violations zur Interpretation hinzuzufügen. Unter **Post-build Actions** → **Report Violations to GitHub** → **PYLINT** im Feld **Pattern** folgendes eingeben: `**/pylint.out`. Dieses File wird auch durch den bereits erwähnten Code erstellt.

Der Job muss nun nur mehr gespeichert werden.

## 3.2 Job ausführen

Dazu wird im Dashboard auf **Build Now** gedrückt.

## 4 Warum es nicht funktioniert

Wenn der Job gestartet wird, ist der Punkt nicht Grün sondern Rot. Dies indiziert schon mal etwas Schlechtes. Wenn man sich nun den Status ansieht, sieht man nichts. Das bedeutet dass es nicht funktioniert hat weder einen Coverage report, einen JUnit testing report oder einen Report violations report zu erzeugen.

Warum es nicht funktioniert hat, kann man sich im Console Output ansehen:

```

Gestartet durch Benutzer Martin Woelfer
2 Baue in Arbeitsbereich F:\Program Files (x86)\Jenkins\workspace\Bruch
> F:\Program Files\Git\bin\git.exe rev-parse --is-inside-work-tree # timeout=10
4 Fetching changes from the remote Git repository
> F:\Program Files\Git\bin\git.exe config remote.origin.url F:\Users\Bernd\Onedrive\Schule\Jahrgang_4\SEW # timeout=10
6 Fetching upstream changes from F:\Users\Bernd\Onedrive\Schule\Jahrgang_4\SEW

```

```

> F:\Program Files\Git\bin\git.exe --version # timeout=10
8 > F:\Program Files\Git\bin\git.exe fetch --tags --progress F:\Users\Bernd\Onedrive\Schule\Jahrgang_4\SEW
+refs/heads/*:refs/remotes/origin/*
> F:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
10 > F:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/origin/master^{commit}" # timeout=10
Checking out Revision efc0ff6d1cb8bc6793162b6a8d8b11047cc90b54 (refs/remotes/origin/master)
12 > F:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
> F:\Program Files\Git\bin\git.exe checkout -f efc0ff6d1cb8bc6793162b6a8d8b11047cc90b54
14 Commit message: "added .pdf"
> F:\Program Files\Git\bin\git.exe rev-list efc0ff6d1cb8bc6793162b6a8d8b11047cc90b54 # timeout=10
16 [Bruch] $ sh -xe C:\WINDOWS\TEMP\jenkins7976359631185991900.sh
The system cannot find the file specified
18 FATAL: Befehlsausfuehrung fehlgeschlagen
java.io.IOException: CreateProcess error=2, Das System kann die angegebene Datei nicht finden
20 at java.lang.ProcessImpl.create(Native Method)
at java.lang.ProcessImpl.<init>(Unknown Source)
22 at java.lang.ProcessImpl.start(Unknown Source)
Caused: java.io.IOException: Cannot run program "sh" (in directory "F:\Program Files (x86)\Jenkins\
workspace\Bruch"): CreateProcess error=2, Das System kann die angegebene Datei nicht finden
24 at java.lang.ProcessBuilder.start(Unknown Source)
at hudson.Proc$LocalProc.<init>(Proc.java:249)
26 at hudson.Proc$LocalProc.<init>(Proc.java:218)
at hudson.Launcher$LocalLauncher.launch(Launcher.java:930)
28 at hudson.Launcher$ProcStarter.start(Launcher.java:450)
at hudson.tasks.CommandInterpreter.perform(CommandInterpreter.java:109)
30 at hudson.tasks.CommandInterpreter.perform(CommandInterpreter.java:66)
at hudson.tasks.BuildStepMonitor$1.perform(BuildStepMonitor.java:20)
32 at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:736)
at hudson.model.Build$BuildExecution.build(Build.java:206)
34 at hudson.model.Build$BuildExecution.doRun(Build.java:163)
at hudson.model.AbstractBuild$AbstractBuildExecution.run(AbstractBuild.java:496)
36 at hudson.model.Run.execute(Run.java:1724)
at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:43)
38 at hudson.model.ResourceController.execute(ResourceController.java:97)
at hudson.model.Executor.run(Executor.java:421)
40 Build step Shell ausfuehren marked build as failure
[Cobertura] Publishing Cobertura coverage report...
42 Zeichne Testergebnisse auf.
ERROR: Step Veroeffentliche JUnit-Testergebnisse. failed: Keine JUnit-Testergebnisse gefunden. Liegt
vielleicht ein Konfigurationsfehler vor?
44
--- Jenkins Violation Comments to GitHub ---
46
gitHubUrl:
48 repositoryOwner:
repositoryName:
50 pullRequestId:
usernamePasswordCredentialsId: false
52 username: false
password: false
54 useOAuth2TokenCredentials: false
oAuth2Token: false
56 createSingleFileComments: false
createCommentWithAllSingleFileComments: false
58 commentOnlyChangedContent: false
minSeverity: INFO
60 keepOldComments: false
PYLINT with pattern **pylint.out
62 Running Jenkins Violation Comments To GitHub
PR
64 Workspace: F:\Program Files (x86)\Jenkins\workspace\Bruch
No pull request id defined, will not send violation comments to GitHub.
66 Finished: FAILURE

```

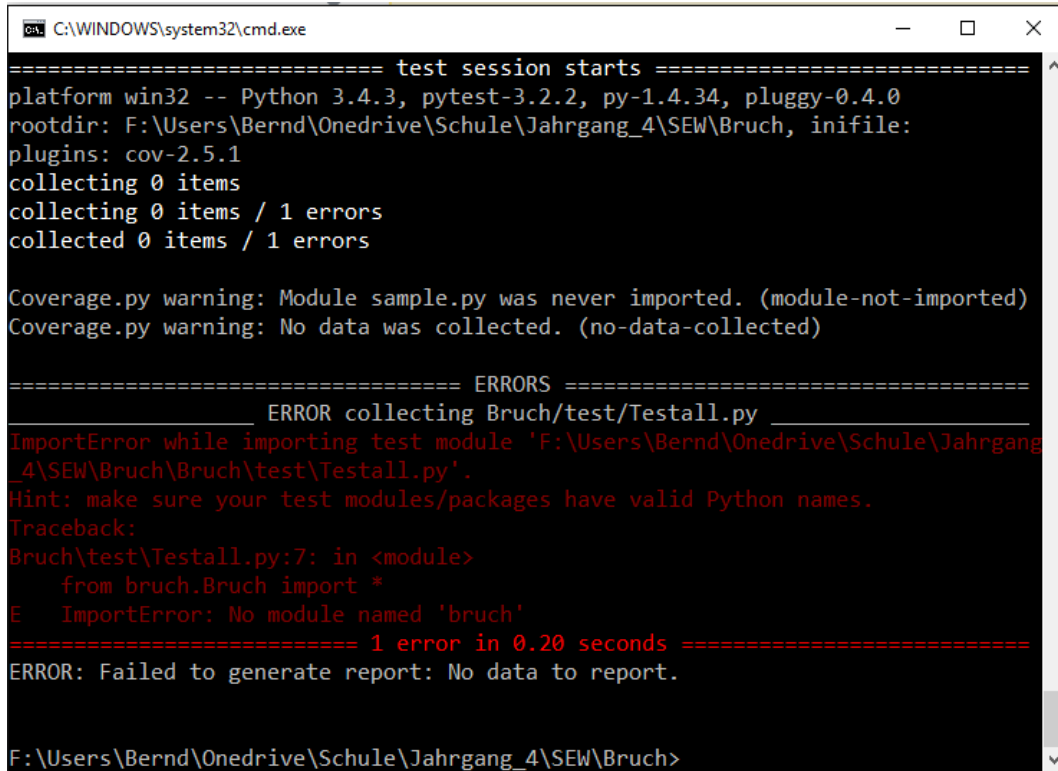
## 4.1 Sh kann nicht ausgeführt werden

Diese Problem konnte gelöst werden indem in den Settings von Jenkins der Pfad zu CMD.exe angegeben wird.



## 4.2 Reports werden nicht erzeugt

Das Problem liegt darin, dass die Befehle welche im Tutorial stehen in meinem System nicht richtig funktionieren. Ich habe probiert wenigstens lokal einen coverage report zu erzeugen, doch bin selbst daran gescheitert da der relative Import mit dem coverage tool nicht funktioniert:



```
C:\WINDOWS\system32\cmd.exe

===== test session starts =====
platform win32 -- Python 3.4.3, pytest-3.2.2, py-1.4.34, pluggy-0.4.0
rootdir: F:\Users\Bernd\Onedrive\Schule\Jahrgang_4\SEW\Bruch, inifile:
plugins: cov-2.5.1
collecting 0 items
collecting 0 items / 1 errors
collected 0 items / 1 errors

Coverage.py warning: Module sample.py was never imported. (module-not-imported)
Coverage.py warning: No data was collected. (no-data-collected)

===== ERRORS =====
_____ ERROR collecting Bruch/test/Testall.py _____
ImportError while importing test module 'F:\Users\Bernd\Onedrive\Schule\Jahrgang_4\SEW\Bruch\Bruch\test\Testall.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
Bruch\test\Testall.py:7: in <module>
    from bruch.Bruch import *
E   ImportError: No module named 'bruch'
===== 1 error in 0.20 seconds =====
ERROR: Failed to generate report: No data to report.

F:\Users\Bernd\Onedrive\Schule\Jahrgang_4\SEW\Bruch>
```

## Abbildungsverzeichnis

1	Ein Admin User wurde erstellt . . . . .	2
2	Daten werden eingegeben für Git . . . . .	3
3	Neuen Job erstellen . . . . .	3
4	Pfad konnte angegeben werden ohne Fehlermeldung . . . . .	4