

Continuous Integration mit L^AT_EX und Jenkins

Martin Kraetke, Uwe Ziegenhagen

Angeregt durch einen englischen Blog-Artikel¹ möchten wir den Begriff »Continuous Integration« vorstellen und zeigen, wie man dieses Konzept auf L^AT_EX-Workflows anwenden kann.

Einführung

Da nicht vorauszusetzen ist, dass jeder Leser mit dem Stichwort aus dem Titel etwas anfangen kann, soll an dieser Stelle eine kurze Einführung gegeben werden. Die Wikipedia² beschreibt »Continuous Integration« – im folgenden durch »CI« abgekürzt – wie folgt:

Kontinuierliche Integration (auch fortlaufende oder permanente Integration; englisch *continuous integration*) ist ein Begriff aus der Software-Entwicklung, der den Prozess des fortlaufenden Zusammenfügens von Komponenten zu einer Anwendung beschreibt. Das Ziel der kontinuierlichen Integration ist die Steigerung der Softwarequalität. Typische Aktionen sind das Übersetzen und Linken der Anwendungsteile, prinzipiell können aber auch beliebige andere Operationen zur Erzeugung abgeleiteter Informationen durchgeführt werden. Üblicherweise wird dafür nicht nur das Gesamtsystem neu gebaut, sondern es werden auch automatisierte Tests durchgeführt und Softwagemetriken zur Messung der Softwarequalität erstellt. Der gesamte Vorgang wird automatisch ausgelöst durch Einchecken in die Versionsverwaltung.

Eine vereinfachte Variante der kontinuierlichen Integration – und häufig ihre Vorstufe – ist der Nightly Build (nächtlicher Erstellungsprozess). In der Praxis trifft man auch auf kontinuierliche Integration, gepaart mit einem Nightly Build, um die Vorteile beider Systeme zu kombinieren.

Zusammenfassend können wir sagen, dass CI einen Prozess beschreibt, der kontinuierlich prüft, ob Bestandteile einer Software die erforderliche Qualität besitzen. Was hat das nun mit L^AT_EX zu tun?

Größere L^AT_EX-Projekte, insbesondere dann wenn mehrere Autoren daran arbeiten, haben ähnliche Probleme wie Softwareentwickler: nicht sofort ist klar, was eine bestimmte Änderung im Code (das Laden eines neuen Paketes) bewirkt und ob das

¹ <https://www.pentapie.com/latex-meets-cicd/>

² https://de.wikipedia.org/wiki/Kontinuierliche_Integration

Gesamtprojekt noch erfolgreich in ein PDF kompiliert werden kann. Hier kann CI beispielsweise dafür sorgen, dass in der finalen Versionskontrolle stets eine geprüfte und damit fehlerfreie Version des Projekts abgelegt wird.

Jenkins bei le-tex

Hello World

Jenkins – Installation

In diesem Abschnitt möchte ich kurz zeigen, wie man Jenkins installieren und konfigurieren muss, um eigene L^AT_EX-Dokumente damit zu übersetzen.

Jenkins bildet die Softwarebasis, die wir zur Steuerung des Build-Prozesses nutzen werden und wurde ursprünglich unter dem Namen »Hudson« bei Sun Microsystems entwickelt. Jenkins benötigt Java – es basiert auf den sogenannten »Enterprise Java Beans« und wird über den Webbrowser bedient.

Im folgenden beschreibe ich das Setup unter Ubuntu, es sind aber auch Installationspakete für Windows, Mac OS X und diverse Linux-Varianten verfügbar.

Die eigentliche Installation verläuft unspektakulär, siehe das folgende Listing. Bei den meisten Debian-basierten Linuxen dürfte ein `apt-get update` mit anschließendem `apt-get install jenkins` ausreichen, für meine Linux-Version Xubuntu »Yakkety Yak« gab es jedoch noch kein Paket, sodass ich das Debian Repository einbinden musste, siehe <https://pkg.jenkins.io/debian/>.

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -  
# deb https://pkg.jenkins.io/debian binary/ in  
# die Datei /etc/apt/sources.list mit aufnehmen  
sudo apt-get update  
sudo apt-get install jenkins
```

Nach der Installation – ein Neustart ist nicht notwendig – kann dann über Port 8080 des Installationsrechners auf die jenkins-Oberfläche zugegriffen werden. Im ersten Schritt müssen wir das Initial-Passwort aus einer Datei der jenkins-Installation kopieren, siehe Abbildung 1. Ein Hinweis noch dazu: Wer keine explizite Security benötigt, kann `<useSecurity>true</useSecurity>` in der `/var/lib/jenkins/config.xml` auf `false` setzen.

Im nächsten Schritt (siehe Abbildung 2) werden die Plugins ausgewählt, die wir benutzen möchten. Hier belasse ich es bei den Standard-Plugins, weitere lassen sich später nachinstallieren.

Nach der Erstellung eines Administratorkontos, siehe Abbildung 4, können wir auf die Oberfläche zugreifen und neue Aufträge erstellen (siehe Abbildung 5).

Jenkins in der praktischen Anwendung

Angeregt durch eine Diskussion auf der internen Dante-Mailingliste hat sich in den letzten Monaten eine Gruppe Enthusiasten gefunden, die an einer Einführung zum Thema »L^AT_EX für Geisteswissenschaftler« arbeitet. Aktuell sieben Mitstreiter haben sich zusammengetan, die verschiedene Themen bearbeiten. Die Zusammenarbeit



Abb. 1: Festlegung des Initial-Passworts

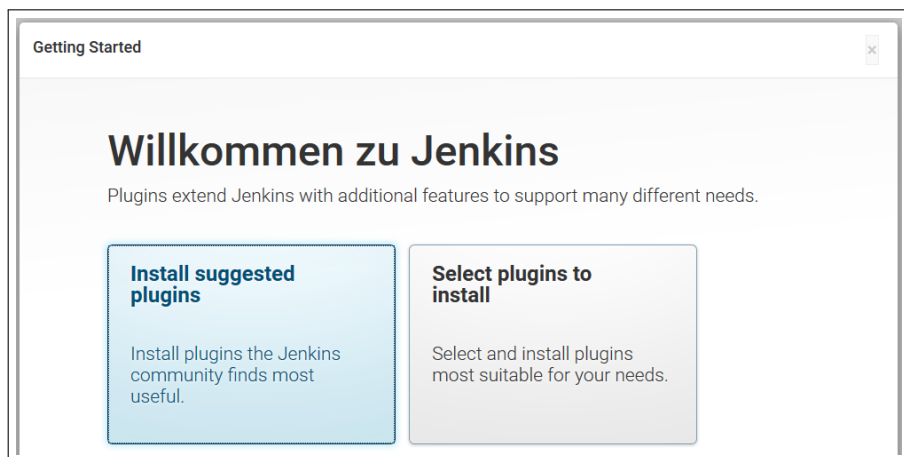


Abb. 2: Auswahl der Plugins

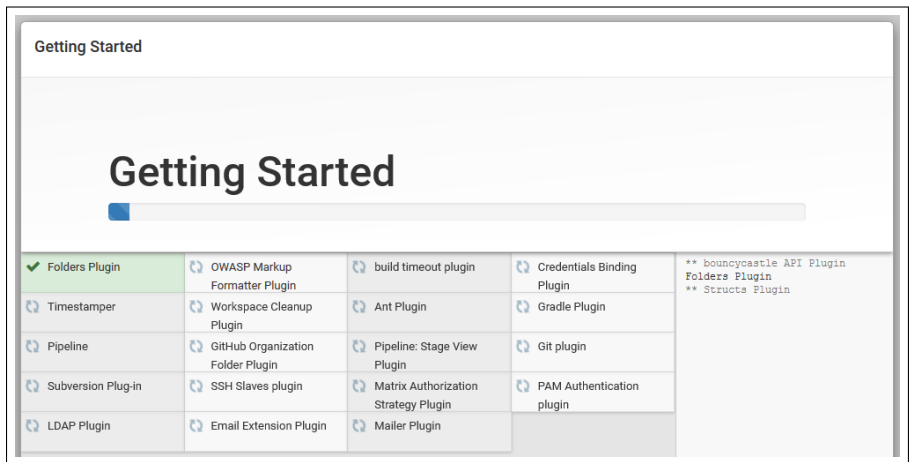


Abb. 3: Auswahl der Plugins

The screenshot shows the 'Create First Admin User' form in Jenkins. It has a heading 'Create First Admin User' and several input fields for user creation. At the bottom, there's a 'Jenkins 2.48' label, a 'Continue as admin' link, and a 'Save and Finish' button.

Benutzer:

Passwort:

Passwort-Wiederholung:

Voller Name:

E-Mail-Adresse:

Jenkins 2.48 [Continue as admin](#) [Save and Finish](#)

Abb. 4: Erstellung des Administrator-Kontos

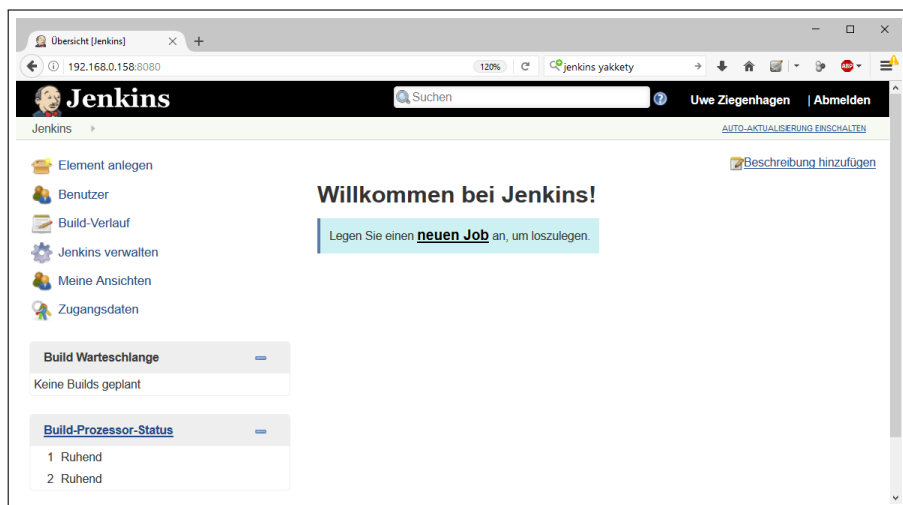


Abb. 5: Die jenkins-Oberfläche

wird über GitHub organisiert, auch die Projektdateien werden auf dieser Plattform (erreichbar unter <https://github.com/thomas-hilarius-meyer/LaTeX-fuer-Geisteswissenschaftler>) gespeichert.

Bei der Zusammenarbeit hat sich gezeigt, dass Tippfehler bei der Eingabe oder unterschiedliche Versionsstände der $\text{T}_{\text{E}}\text{X}$ -Installation dazu führten, dass das Gesamtprojekt nicht mehr kompilierbar war.

Hier bietet sich Jenkins an, um die Kompilierung automatisch durchführen zu lassen. Als Hardware dient ein Intel NUC mit relativ schwachem Celeron N2820 und acht Gigabyte RAM, der unter Ubuntu-Linux läuft und über ein aktuelles $\text{T}_{\text{E}}\text{X}$ Live 2016 verfügt.

In den allgemeinen Einstellungen, siehe Abbildung 6, legt man den Projektnamen fest sowie den Pfad zum GitHub-Projekt.

Unter »Source-Code-Management« (Abbildung 7) wird dann nochmal der Zugang zum Projekt definiert, neben `git` unterstützt GitHub auch `Subversion` für den Zugriff auf die `git`-Repositories.

Um den Build-Job auszulösen gibt es verschiedene Möglichkeiten. Wir haben uns für das zeitgestützte Anstoßen entschieden (Details siehe Abbildung 8), über eine CRON Syntax legen wir fest, dass täglich um Mitternacht der Build erfolgen soll.

The screenshot shows the 'General' tab of the Jenkins configuration interface. The 'Projectname' field is set to 'lfgw'. The 'Beschreibung' field is empty. Below the description, there is a link '[Plain text] [Vorschau](#)'. A list of checkboxes includes 'Alle Builds verwerfen', 'Dieser Build ist parametrisiert.', 'GitHub-Projekt' (checked), 'Throttle builds', 'Projekt deaktivieren', and 'Parallele Builds ausführen, wenn notwendig'. The 'Project url' field contains 'https://github.com/thomas-hilarius-meyer/LaTeX-fuer-Geisteswissenschaftler/'. There are two 'Erweitert...' buttons on the right side of the configuration.

Abb. 6: Allgemeine Einstellungen

The screenshot shows the 'Source-Code-Management' tab of the Jenkins configuration interface. The 'Git' option is selected under the 'Source-Code-Management' section. The 'Repositories' section contains a 'Repository URL' field with 'https://github.com/thomas-hilarius-meyer/LaTeX-fuer-Geisteswissenschaftler/' and a 'Credentials' dropdown menu showing 'ziegenhagen@gmail.com/******'. There is an 'Add' button next to the credentials. The 'Branches to build' section has a 'Branch Specifier (blank for 'any')' field with '*/*master'. The 'Repository Browser' dropdown menu is set to '(Auto)'. There are several 'Erweitert...' and 'Add' buttons throughout the configuration.

Abb. 7: Zugang zum Repository

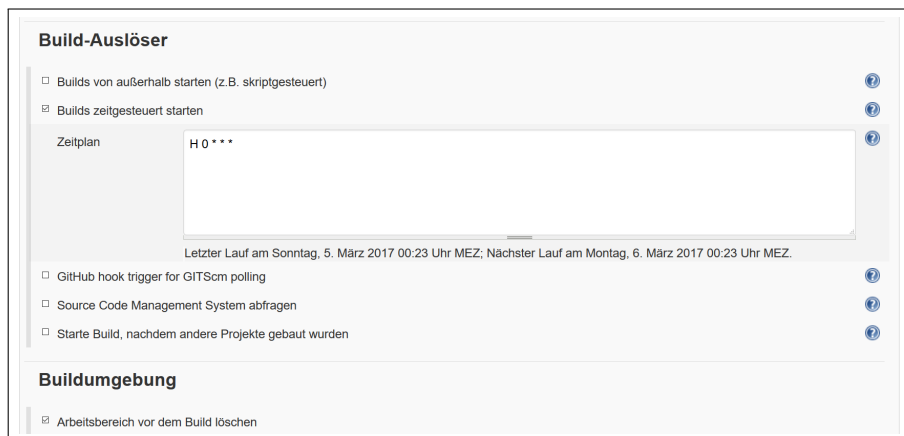


Abb. 8: Zeitgesteuerte Auslösung

Unter »Buildverfahren« schließlich wird definiert, was für die Übersetzung geschehen muss, in unserem Fall $\text{luaL}^{\text{A}}\text{T}_{\text{E}}\text{X}$ gefolgt von `biber` und anschließendem erneuten $\text{luaL}^{\text{A}}\text{T}_{\text{E}}\text{X}$. Alle drei Aufrufe schreiben ihre Ergebnisse in ein separates Verzeichnis, um nicht Probleme mit den Kompilierungen der einzelnen Co-Autorinnen und -Autoren zu verursachen. Abbildung 9 zeigt die entsprechenden Aufrufe.

Im Anschluss daran werden über den `git commit` Befehl die Änderungen (nur am PDF, da die $\text{T}_{\text{E}}\text{X}$ -Dateien hier nicht verändert werden) in das Repository geschrieben.

Der finale Schritt (siehe Abbildung 10) besteht dann darin, das geänderte PDF wieder zurück nach GitHub zu spielen.

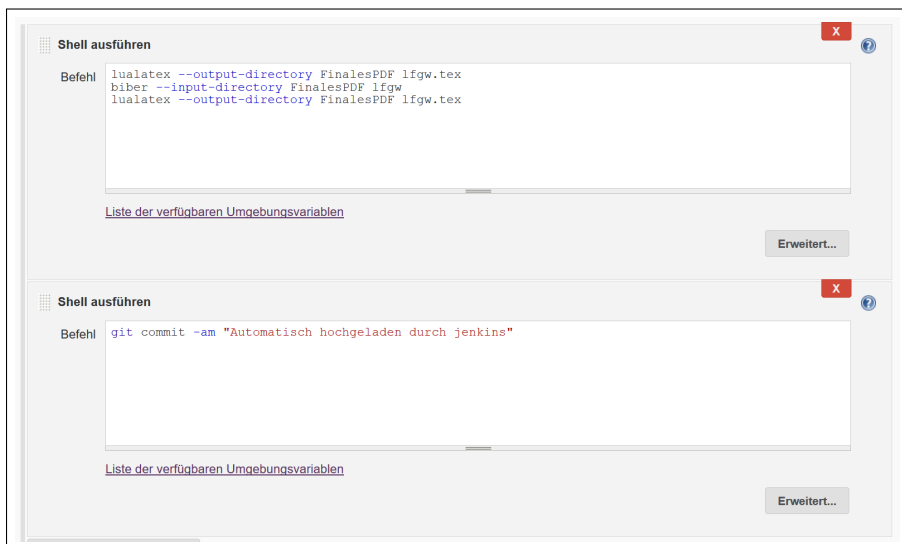


Abb. 9: Die eigentliche Übersetzung

Post-Build-Aktionen

Git Publisher

Push Only If Build Succeeds

☒

Merge Results

☐

If pre-build merging is configured, push the result back to the origin

Force Push

☐

Add force option to git push

Tags

Add Tag

Tags to push to remote repositories

Branches

Branch to push

master

Target remote name

origin

Add Branch

Branches to push to remote repositories

Notes

Add Note

Notes to push to remote repositories

Abb. 10: Der Weg zurück zu GitHub

Fazit