



Home About Archive Impressum Datenschutzerklärung

← String Deduplication zum Sparen von Speicherplatz in Java 8

Git Workflows Teil 2: Workflows meistern →

Search



OIO's Developer Blog
This is the publication playground for these guys from OIO, the Trivadis Group's Java experts.



Tag cloud

Agile Methods and development Atlassian Tools Build, config and deploy Did you know? Eclipse Universe Groovy and Grails Java and Quality Java Basics Java EE Java modularization Java Persistence Java Runtimes - VM, Appserver & Cloud Java Web Frameworks mod Open Source BI Other languages for the Java VM Politics Security ECA, WebServices Spring Universe Web as a Platform mod Universe

Email Subscription

Click to subscribe to this blog and receive notifications of new posts by email.

Join 134 other subscribers

Email Address

Top Posts & Pages

- High performance at low cost - choose the best JVM and the best Garbage Collector for your needs

Latest Comments

- Jonathan Kazmierczak on High performance at low cost - choose the best JVM and the best Garbage Collector for your needs
- Franta on High performance at low cost - choose the best JVM and the best Garbage Collector for your needs
- Cameron on How to use JUnit 5 @MethodSource-parameterized tests with Kotlin
- Kirsty Bon on How to use JUnit 5 @MethodSource-parameterized tests with Kotlin
- Les modules en TypeScript en 5 min | Blog .NET on Declaration Merging with TypeScript

RSS

RSS - Posts

Archiv

- January 2020 (3)
- December 2019 (4)
- November 2019 (2)
- October 2019 (2)
- September 2019 (2)
- August 2019 (1)
- July 2019 (1)
- June 2019 (5)
- May 2019 (1)
- April 2019 (3)
- March 2019 (3)
- February 2019 (8)
- January 2019 (9)
- November 2018 (1)
- October 2018 (3)
- September 2018 (2)
- August 2018 (2)
- July 2018 (3)
- June 2018 (4)
- May 2018 (10)
- April 2018 (1)
- March 2018 (6)
- February 2018 (7)
- January 2018 (3)
- November 2017 (3)
- October 2017 (1)
- September 2017 (6)
- August 2017 (5)
- July 2017 (2)
- March 2017 (1)
- February 2017 (4)
- December 2016 (1)

Git Workflows Teil 1: Warum wir Workflows brauchen

Posted on 22. September 2014 by Felix Bruckner

In dieser Artikelserie werden wir das in der Software-Entwicklung inzwischen omnipräsente Phänomen Git Workflows näher betrachten. Im ersten Teil betrachten wir das Wie und Warum hinter Git Workflows, bevor wir uns im zweiten Teil konkreten Workflow-Modellen widmen. Im dritten Teil beantworten wir die Frage, wie man eigene Workflows mithilfe von Werkzeugen geschickt umsetzt.

Ein typisches Szenario in einer Softwareschmiede könnte so aussehen: Eine gemeinsame Codebasis, mehrere Mitarbeiter oder sogar Teams und ein stabiler master-Zweig, von dem aus veröffentlicht werden soll. Ein "zweig-affines" Versionskontrollsysteem soll her, denn Zweige gehören zum guten Ton – also setzt man auf Git. Warum eigentlich? Benutzt jeder heutigezeit. Wie das im Detail funktioniert, ist ja auch nicht so wichtig.

Ein Team soll eine neue Funktionalität entwickeln und hat, wie es sich gehört, einen neuen Zweig erstellt und an diesem bereits einige Wochen fleißig gearbeitet. Trotz der Absichtsformulierung "ab und zu" die neusten Änderungen aus master in den Feature-Zweig zu integrieren, damit alles auch schön aktuell bleibt, ist die letzte Aktualisierung doch bereits eine ganze Weile her – keine Zeit für sowas! In der Zwischenzeit sind bereits unzählige Änderungen von anderen Teams eingeflossen und größere Refactoringarbeiten an der Codebasis vorgenommen worden, klar. Die Unterschiede zwischen Feature- und Mainline-Zweig sind inzwischen groß. Der sowieso unter Zeitdruck stehende Entwickler baut bzw. hackt die Mainline mit dem Feature-Zweig zusammen und pusht das Feature in den Hauptzweig. Was auf der lokalen Entwicklermaschine noch so schön funktioniert hat, baut der letzte Woche eiligst eingehängte CI-Server auf einmal nicht mehr. Die stundenlange Fehlersuche in der Commit-History beginnt...

Im nächsten Sprint Review einigt man sich darauf, endlich einen dieser Workflows zu implementieren – damit wird laut StackOverflow alles besser.

Alles auf Anfang: Die Motivation hinter Software

Es wirkt beinahe so, als würden manche Teams Workflows zum Selbstzweck einsetzen – weil man es eben so tut. Ebenfalls erschließt sich anderen Projektbeteiligten oder gar Entscheidern die Notwendigkeit dieser Arbeitsprozesse innerhalb eines Entwicklungsteams oftmals nicht direkt. Um die komplexe Thematik rund um Workflows zu verstehen, muss man sich auf die grundlegenden Gedanken, die hinter Versionsverwaltung, ja gar hinter Software allgemein stehen, besinnen.

Was ist der Zweck von Software? Menschen zu helfen. Software abstrahiert Probleme und soll Menschen täglich dabei helfen, die Probleme in ihrem Kontext besser, einfacher und schneller zu bewältigen.

Der Zweck eines Software-Entwicklungsprozesses ist es, Menschen zu helfen, Software einfacher, schneller, besser zu veröffentlichen. Software besteht aus Anforderungen z.B. in Form von User Stories, welche Geschäftsziele repräsentieren. Prozessnahmenmodelle wie Scrum helfen den Entwicklern, ihre Arbeit entsprechend diesen Zielen zu priorisieren.

Genauso gibt es Software für Entwickler, die Entwicklern dabei hilft, einfacher Software zu erstellen und zu veröffentlichen. Alltägliches Werkzeug ist zum Beispiel die Entwicklungsumgebung, der Issue-Tracker und die Versionskontrolle.

Von Zweigaktualisierungen und Tortwächtern: Pull Requests

Die Hauptaufgabe von Versionskontrolle ist, Entwicklern beim Schreiben von Software zu helfen und Änderungen zwischen Team-Mitgliedern zu synchronisieren und regelmäßig zu sichern. Die zweite Aufgabe ist Konfigurationsmanagement: Parallelie Entwicklungsstränge verwalten, alte Versionen pflegen, neue Funktionen prototypisch implementieren und so weiter.

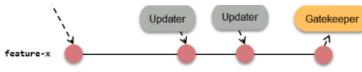
Das Erstellen von Zweigen (Branching) ist immer mit einem gewissen Integrationsaufwand verbunden. Arbeitet man im Team an einem Projekt, gibt es in der Regel ein zentralisiertes Repository als gemeinsame Ablageort. Vielleicht ist sogar eine Continuous Delivery-Pipeline aufgesetzt, sodass jede Änderung am master-Zweig in ein potenziell auslieferbares Artefakt mündet.

Auf einmal kommen Fragen auf, die die Diskussion um Branching anheizt:

- Ist jetzt jeder Entwickler selbst dafür verantwortlich, dass seine Änderungen in den Hauptzweig integrierbar sind? Falls ja, wie stellt man sicher, dass die Änderungen auch keine Regressionen hervorrufen?
- Wem vertraue ich soweit, dass die Änderungen auf dem master-Zweig nicht im Desaster enden?
- Wie verhindere ich, dass ein unachtsamer Entwickler einen force-push auf den master-Zweig macht?
- Habe ich Vorkehrungen getroffen, sodass ich im Fehlerfall schnell die Ursache herausfinden kann?
- Wer erteilt die Freigabe für die Integration eines neuen Features?
- Können Stakeholder eigentlich verstehen, was wir da warum tun?

Zwei Konzepte können helfen, etwas mehr Klarheit zu schaffen. Klar definierte Regeln für (periodisches) Branch Updating und die Einführung eines Gatekeepers.





Branch Updating beschreibt den Vorgang, bei dem die Änderungen des Zielzweiges – also beispielsweise master – regelmäßig in unseren aktuellen Entwicklungs Zweig integriert werden.

Ein **Gatekeeper** beschreibt eine Rolle oder einen Mechanismus, welcher festlegt, dass eine Person oder ein Programm vor dem Integrieren eines Zweiges die Änderungen vorab auf Korrektheit prüft. Das kann beispielsweise ein Code Review eines Entwicklers oder ein Build eines Continuous Integration-Servers sein.

Ein beliebter Prozess des Gatekeepings findet in Form von **Pull Requests** statt, bei denen ein Entwickler ohne Schreibrechte auf einen kritischen Zweig wie master die Anfrage zur Integration seiner Änderungen an einen berechtigten Integrator stellt. Dazu benötigt man wiederum ein Rollenkonzept, welches klar die Zuständigkeiten und Rechte regelt.

Und spätestens jetzt wird ersichtlich, dass eine Art Prozess her muss, der irgendwo dokumentiert werden sollte: Ein Workflow.

Die Definition eines DVCS (Git)- Workflows

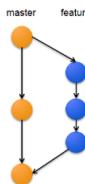
Git bzw. VCS-Workflows liefern einen strukturierten Ansatz für die Software-Entwicklung mit vielen Zweigen. Zusammengefasst sollte ein Workflow folgende Eigenschaften besitzen:

- Ein Workflow sollte Stakeholdern, Projektmanagern und IT-Betrieb dabei helfen, besser zu verstehen, was gerade in der Entwicklung vor sich geht.
- Zweige sollten dafür benutzt werden, ein einziges Problem/Feature abzubilden.
- Man sollte zwischen öffentlichen/permanenten und privaten/temporären Zweigen unterscheiden.
- Öffentliche Zweige müssen zu jeder Zeit von Projektverantwortlichen freigegeb- und auslieferbar sein.
- Vorgänge, wie der Entwicklungsbeginn eines neuen Features, Merges oder Deployments, sollten nicht ohne entsprechende Freigaben von Projektverantwortlichen stattfinden können, d.h. es sollte ein Rollenkonzept abzubilden sein.

Mit dieser Definition gerüstet, betrachten wir den einfachst möglichen Workflow.

Das einfachste Beispiel: master-only

Der Aufbau ist denkbar einfach; den einzigen zentralen Zweig stellt der Master-Zweig dar. Features werden in privaten Zweigen entwickelt. Die Grundregel lautet, öffentliche Zweige als unveränderlich, feststehend mit sauber gehaltener Historie zu behandeln. Private Feature-Zweige werden als veränderbares und lediglich temporär existierendes Konstrukt betrachtet.



- Erstelle einen temporären Zweig (feature) aus einem öffentlichen Zweig (master) heraus.
- Schreibe Änderungen regelmäßig in den temporären Zweig.
- Sobald die Arbeiten abgeschlossen sind, räume die Historie auf und sorge für eine saubere Integrierbarkeit.
- Führe den Zweig mit master zusammen.
- Lösche den nun überflüssigen Feature-Zweig.

Vorteil dieses Workflows ist seine Einfachheit. Der große Nachteil ist die fehlenden Zuständigkeiten, schlechte Skalierbarkeit für größere Projekte und die fehlende Möglichkeit, Legacy-Versionen weiterhin zu pflegen. Die Grundidee wird aber bestehen bleiben, und bei der Betrachtung weiterer Workflows werden auf dieser Grundlage weitere Ansätze betrachtet und schrittweise komplexer modelliert.

Diesen widmen wir uns [im zweiten Teil dieser Artikelserie](#), bevor wir uns mit dem Thema des Workflow-Designs und der Umsetzung mithilfe von Software befassen.

Share!



Like this:



Related

- | | | |
|--|---|---|
| Eine Einführung in OAuth 2 | Wann ist modellgetriebene Softwareentwicklung sinnvoll? | Welche neuen Features bringt Angular 8? |
| 20. August 2018
In "Security" | 2. August 2017
In "MDSD" | 7. June 2019
In "Web as a Platform" |

Short URL for this post: <https://wp.me/p4nxik-2f>

This entry was posted in [Agile Methods and development](#), [Build, config and deploy](#), [Politics](#) and tagged [git](#), [git workflow](#), [gitflow](#). Bookmark the [permalink](#).

← String Deduplication zum Sparen von Speicherplatz in Java 8

Git Workflows Teil 2: Workflows meistern →

Leave a Reply

Enter your comment here...

- November 2016 (5)
- October 2016 (4)
- September 2016 (4)
- August 2016 (5)
- July 2016 (3)
- June 2016 (3)
- May 2016 (11)
- April 2016 (4)
- February 2016 (4)
- January 2016 (2)
- December 2015 (5)
- November 2015 (9)
- October 2015 (1)
- September 2015 (2)
- August 2015 (5)
- July 2015 (3)
- June 2015 (4)
- May 2015 (4)
- April 2015 (3)
- March 2015 (4)
- February 2015 (4)
- January 2015 (4)
- December 2014 (3)
- November 2014 (3)
- October 2014 (3)
- September 2014 (10)
- August 2014 (2)
- July 2014 (9)
- June 2014 (1)
- May 2014 (5)
- April 2014 (5)
- March 2014 (16)
- February 2014 (20)
- January 2014 (6)
- December 2013 (6)
- November 2013 (2)
- October 2013 (3)
- September 2013 (1)
- August 2013 (7)
- July 2013 (15)
- June 2013 (11)
- May 2013 (17)
- April 2013 (9)
- March 2013 (8)
- February 2013 (16)
- January 2013 (8)
- December 2012 (9)
- November 2012 (8)
- October 2012 (12)
- September 2012 (7)
- August 2012 (9)
- July 2012 (9)
- June 2012 (11)
- May 2012 (9)
- April 2012 (8)
- March 2012 (6)
- February 2012 (16)
- January 2012 (4)
- December 2011 (24)
- November 2011 (2)
- October 2011 (3)
- September 2011 (7)
- August 2011 (6)
- July 2011 (10)
- June 2011 (5)
- May 2011 (26)
- April 2011 (13)
- March 2011 (19)
- February 2011 (11)
- January 2011 (7)
- December 2010 (8)
- November 2010 (19)
- October 2010 (33)
- September 2010 (2)
- August 2010 (5)
- July 2010 (7)
- June 2010 (1)
- May 2010 (21)
- April 2010 (9)
- March 2010 (9)
- February 2010 (17)
- January 2010 (14)
- June 2009 (1)
- May 2009 (2)

Recent Posts

- High performance at low cost – choose the best JVM and the best Garbage Collector for your needs
- Timeouts in JUnit 5
- Map Reduce in Hadoop Example
- Introduction to HttpClient
- What is Docker Compose? How to use

Categories

- Agile Methods and development
- Atlassian Tools
- Build, config and deploy
- Did you know?
- Eclipse Universe
- Groovy and Grails
- Java and Quality

Subscribe to Blog via Email

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 134 other subscribers

Email Address

it?

- Java Basics
- Java EE
- Java modularization
- Java Persistence
- Java Runtimes – VM, Appserver & Cloud
- Java Web Frameworks
- MDS
- Open Source BI
- Other languages for the Java VM
- Politics
- Security
- SOA / Webservices
- Someone thinks I fit nowhere else
- Spring Universe
- Uncategorized
- Web as a Platform
- XML Universe

[Subscribe](#)

[techscouting through the java news](#)

 Proudly powered by WordPress.