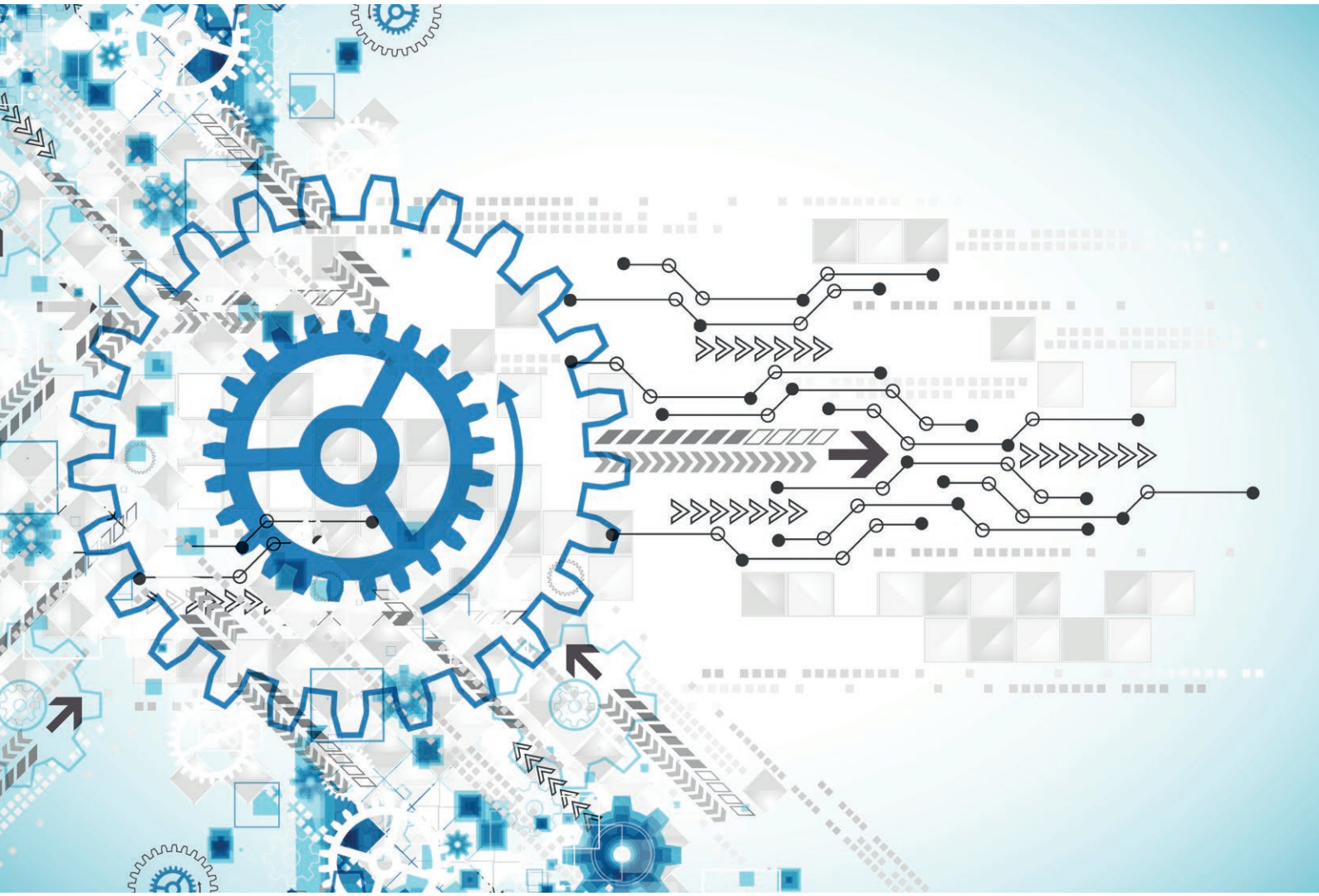

Automatisierung und Durchblick mit (Git-)Hub und (Git-)Lab

Kai Schmidt, selbstständig

GitHub und GitLab bieten Funktionen, die über das Versionsverwaltungssystem Git hinausgehen. Während Git eine gute Unterstützung für die Kommandozeile bietet und dessen Nutzung weit verbreitet scheint, werden GitHub und GitLab weitestgehend über das Webinterface bedient. Eine Automatisierbarkeit des Webinterface wäre beispielsweise über Selenium oder TestCafé möglich, jedoch umständlich und zeit- sowie wartungsintensiv. Glücklicherweise bieten beide Portale jeweils ein umfangreiches REST-API, über das viel möglich ist. Außerdem ist es weniger inkompatiblen Änderungen unterworfen als das Webinterface. Gleichzeitig ist die Nutzung des API aber mit einer erhöhten Einarbeitungszeit verbunden. Erleichterungen schaffen hierbei die Kommandozeilen-Tools Hub und Lab, die ein leicht erlernbares CLI zur Verfügung stellen und wesentliche Funktionen der jeweiligen Portale abdecken. Dieser Artikel soll einen Überblick über den Funktionsumfang der beiden Tools geben.

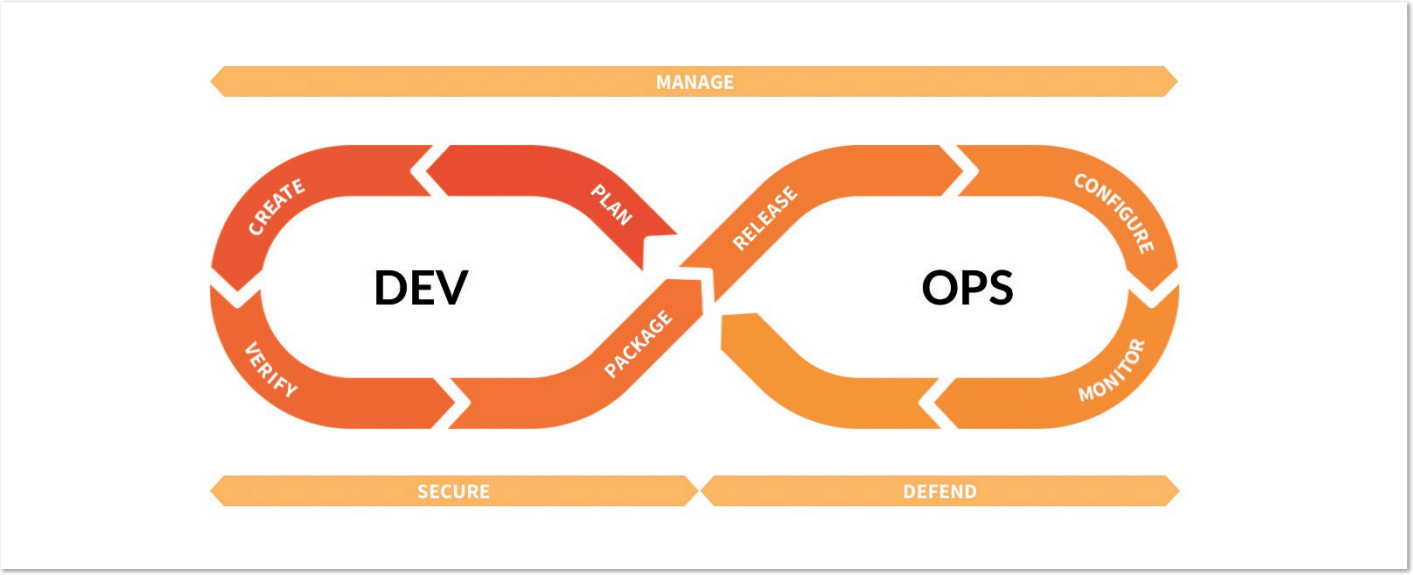

Abbildung 1: Der DevOps Lifecycle mit GitLab [5] (Quelle: Kai Schmidt)

Für uns sehr selbstverständlich gewordene Funktionen wie Pull- beziehungsweise Merge-Requests sind keine nativen Bestandteile von Git und daher nicht im Git-CLI enthalten. Die in GitHub *[1]* verfügbaren zusätzlichen Funktionen werden über das Kommandozeilen-Tool Hub *[2]* erweitert.

Insbesondere GitLab *[3]* führt viele weitere Funktionen ein, die über Git hinausgehen, um eine Abdeckung ihres DevOps Lifecycle *(siehe Abbildung 1)* zu erreichen. Diese umfassen die Planungsphase neuer Feature-Ideen bis hin zum Monitoring, und sie stehen zum Teil nur in kommerziellen Versionen zur Verfügung. Sehr bekannte und in der kostenfreien Version verfügbare Funktionen von GitLab sind das Erstellen von Issues. Sie versehen dieselbigen mit Labeln und Meilensteinen wie zum Beispiel die in GitLab eingebaute CI-Funktionalität. An Hub angelehnt, wird ein gewisser Anteil der GitLab-Funktionen über das Kommandozeilen-Tool Lab *[4]* zur Verfügung gestellt.

Am Anfang steht die Installation

Die Installation von Hub gestaltet sich durch die Paketmanager der unterschiedlichen Betriebssysteme (beispielsweise Homebrew oder Apt) sehr einfach und ist der Readme des Projekts zu entnehmen. Für Windows stehen ZIP-Dateien für den Download von Hub bereit, über Scoop oder Chocolatey können sie installiert werden. Damit lässt sich Hub direkt verwenden. Hub authentifiziert sich dabei auf dem gleichen Weg wie Git (HTTPS mit Benutzername/Passwort oder SSH).

Wer die gleiche Paketmanager-Abdeckung für Lab erwartet, wird etwas enttäuscht. Dennoch ist die Installation laut Readme ein Einzeiler. Unter Ubuntu muss der Einzeiler für eine erfolgreiche Installation leicht umgebaut werden *(siehe Listing 1)*.

Im Gegensatz zu Hub authentifiziert sich Lab über einen API Key, der in GitLab hinterlegt wird. Die zugehörige Konfiguration gestaltet sich durch den dargestellten Link nach dem Start des ersten Lab-Befehls recht einfach. Sollte der GitLab-Host oder der Token falsch sein, funktioniert kein einziger Lab-Befehl mehr. Beide Informationen lassen sich in der Datei `~/.config/lab.hcl` anpassen. Dies ist insbesondere dann praktisch, wenn man sich einen neuen Token generieren oder die gleiche Datei auf einem weiteren Rechner verwenden möchte.

Falls Windows-Veteranen die Nutzung von Scoop scheuen, lässt sich Lab (wie auch Hub) alternativ unter der WSL (Windows Subsystem for Linux) nutzen. In einer Ubuntu-WSL gestaltet sich die Installation wie unter einem Ubuntu-Betriebssystem. Beschreibungen zur Einrichtung der WSL finden sich unter *[6]* und *[7]*.

Erleichterte Git-Funktionalitäten

Durch das Wissen, mit welcher Repository-Plattform das jeweilige Projekt verbunden ist, wird die Nutzung einiger bekannter Git-Befehle erleichtert. Hat man eines der Tools installiert, lässt sich ein Repository durch die Kombination aus Benutzer- und Projektnamen auf den jeweiligen Rechner klonen. Dies macht die komplette URL auf das Repository überflüssig. *Listing 2* zeigt das Klonen der Projekte mit den entsprechenden Tools.

Die vollständige Liste der Funktionen sind bei Hub in der Readme verlinkt. Bei Lab führt ein Aufruf von Lab in der Kommandozeile weiter. Während Hub Man Pages (Befehl `man hub` oder `man hub-clone`) unterstützt, erhält man bei Lab detaillierte Informationen der einzelnen Befehle über das Flag `-h`.

```
$ cd /tmp/
$ wget https://raw.githubusercontent.com/zaquestion/lab/master/install.sh
$ chmod u+x ./install.sh
# Falls curl noch nicht installiert sein sollte
$ sudo apt install curl
$ ./install.sh
```

Listing 1: Installation von Lab auf Ubuntu

```
# “git clone git://github.com/github/hub.git” wird zu
$ hub clone github/hub

# “git clone git@gitlab.com:gitlab-org/manage.git” wird zu
$ lab clone gitlab-org/manage
```

Listing 2: Klonen eines Repository mit Hub beziehungsweise Lab

```
1 function localBranchToMR {
2   if git rev-parse --is-inside-work-tree > /dev/null 2>&1; then
3     git checkout $1
4     git push -u origin $1
5     lab mr create origin develop -d
6   else
7     echo "Not a git repository"
8  fi;
9 }
```

Listing 3: Funktion zur Erstellung eines Merge-Requests aus einem lokalen Branch

Die aus Git bekannten, erweiterten Formen lassen sich sowohl für Hub als auch für Lab grob in die folgenden Kategorien einteilen:

- Leichtere Adressierbarkeit des Remote (wie das bereits beschriebene `clone`)
- Einfachere Unterstützung verschiedener Remotes
- Arbeiten mit Forks, Pull- beziehungsweise Merge-Requests und Submodules (auf diese Weise kann ein Check-out direkt auf einem Pull-Request erfolgen: `hub checkout https://github.com/myuser/myproject/pull/1`)

Auch wenn die Funktionsweisen der Git-Erweiterungen ähnlich sind, ist der Funktionsumfang unterschiedlich. Ebenso unterscheiden sich die Anwendung der Befehle beziehungsweise der Kontroll-Flags zwischen Hub und Lab häufig. Für die konkrete Nutzungsweise der einzelnen Befehle sei auf die jeweilige Dokumentation verwiesen.

Mit Pull-Requests arbeiten

Es gibt bereits einige Artikel – zum Beispiel den Blog Post von Phillip Krüger *[8]* – , die beschreiben, wie komplexere Use Cases mit mehreren Git-Kommandos abgedeckt oder die Nutzung von Git-Kommandos vereinfacht werden können. Je nach Arbeitsprozess decken ebenso andere Tools wie Git Flow *[9]* entsprechende Bedürfnisse ab. Dank Hub und Lab können nun mit Leichtigkeit und sehr flexibel die Zusatzfunktionen der entsprechenden Repository-Plattformen genutzt und ebenso in Skripte eingebunden werden. Wiederkehrende Abläufe sind nun leicht automatisierbar. Je nach Persönlichkeit stellt man fest, dass der Bedarf, auf die jeweiligen Webseiten des Portals zu wechseln, hierdurch sehr stark begrenzt ist und die Kommandozeile immer mehr zum Cockpit für die Portale mutiert. Falls man bestimmte Tätigkeiten dennoch über die Webseite durchführen möchte, ist diese nun über `hub browse` beziehungsweise `lab project browse` leicht erreichbar.

Hub und Lab bieten Möglichkeiten der Verwaltung von Repositories (zum Beispiel Erstellen und Löschen derselbigen) über die Erstellung von Issues bis hin zum Arbeiten mit Pull-Requests an. Die genauen Aufrufe sind der jeweiligen Dokumentation entnehmbar. Um im Rahmen dieses Artikels ein Gefühl dafür zu geben, wie mit den zusätzlich angebotenen Funktionen der Repository-Plattformen mit Hub und Lab gearbeitet wird, beschränke ich mich im Folgenden auf die Funktionalität für Pull-Requests. Pull- und Merge-Requests sind weitestgehend synonym. Während GitHub diesen Prozess nach dem initialen Schritt benennt (Pull), hat sich GitLab entschieden, sich an der finalen Aktion zu orientieren (Merge) *[10]*.

Falls man in einem Projekt mit Feature Branches und Pull-Requests arbeitet, lässt sich leicht ein lokaler Branch in einen Pull-Request umwandeln und beispielsweise als `function` in die `.bashrc` einbinden. *Listing 3* zeigt ein Beispiel, um aus einem lokalen Branch in GitLab einen Merge-Request für den Develop Branch zu erstellen und den Quell-Branch beim Mergen automatisch wieder zu löschen (Flag -d). Bei Angabe eines Parameters wird für den benannten Branch ein Merge-Request erzeugt. Ohne Parameter wird für den gerade ausgecheckten Branch der Merge-Request erstellt. Hub-Nutzer müssen den Lab-Befehl in Zeile 5 wie folgt anpassen: `hub pull-request -b develop -m “merge branch to develop”`

Nach dem erfolgreichen Aufruf von `localBranchToMR` wird die URL und damit auch die ID des Pull-Requests angezeigt, die gegebenenfalls in ein Issue-Tracking-Tool übertragen werden könnte. Alternativ lässt sich jederzeit die Liste der Pull-Requests über den Aufruf `hub pr list` beziehungsweise `lab mr list` anzeigen.

Wäre die ID des Pull-Requests 1, würde der Merge – beispielsweise nach dem Abschluss eines Reviews – mit dem Befehl `hub merge https://github.com/myuser/myproject/pull/1` beziehungsweise `lab mr merge origin 1` ausgeführt werden. Der Unterschied ist, dass bei Hub der Merge lokal geschieht und entsprechend mit einem `git push` abgeschlossen werden muss. Bei Lab geschieht der Merge bereits remote. Läuft die GitLab-Pipeline des Quell-Branch noch, wird der Merge-Befehl an GitLab gesendet. GitLab führt den Merge – wie über das Webinterface – erst nach erfolgreichem Pipeline-Lauf durch. Über den abschließenden Status des Merge wird man von GitLab per E-Mail benachrichtigt.

Die Unterstützung für Merge-Requests bei GitLab-Repositories lässt sich ebenso über Plug-ins in IntelliJ IDEA einbinden. Eine Suche nach GitLab im Marketplace gibt Aufschluss darüber. Für GitHub ist dem Autor selbiges jedoch nicht bekannt. Dank Hub und Lab erhält

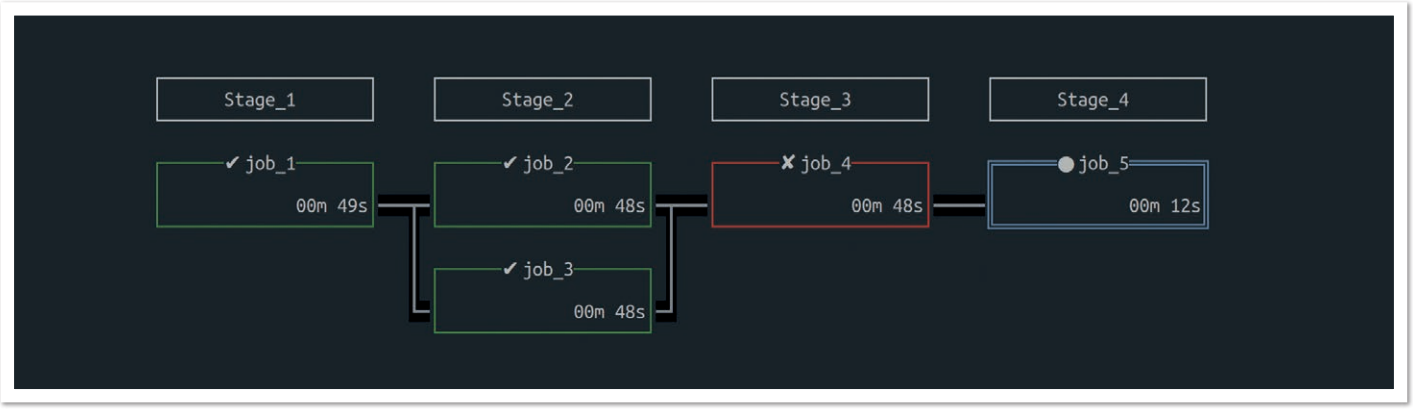

Abbildung 2: Die CI-Pipeline in der Kommandozeile (Quelle: Kai Schmidt)

man eine prächtige Unterstützung für jede beliebige IDE (beziehungsweise der dort eingeblendeten Konsole), um entsprechende Tätigkeiten verrichten zu können – unabhängig von einem Support durch Plug-ins.

Die CI-Pipeline in Textform

Mein persönliches Lieblingsfeature von Lab ist die Darstellung der CI-Pipeline. Nach einem Aufruf von `lab ci view origin develop` wird der aktuelle beziehungsweise letzte Pipeline-Lauf des Develop Branch angezeigt *(siehe Abbildung 2)*.

Wie gerade angedeutet, lässt sich diese Sicht praktischerweise in der Konsole der IDE einblenden, was *Abbildung 3* veranschaulichen soll. Durch IDE-Plug-ins erhält man diese Möglichkeit derzeit nicht. Jeder Job der Pipeline erhält ein Kästchen, in dem sowohl der Jobname als auch die verstrichene Zeit für den Job dargestellt wird. Bereits erfolgreich gelaufene Jobs werden in Grün dargestellt und erhalten einen Haken vor dem Jobnamen. Fehlgeschlagene Jobs sind rot gekennzeichnet und bekommen ein X. Noch nicht gelaufene Jobs bleiben schwarz. Laufende Jobs sind blau und erhalten einen ausgefüllten Kreis. Parallel laufende Jobs werden untereinander angezeigt.

Über die Cursor-Tasten kann man einen beliebigen Job auswählen. Welcher der Jobs gerade den Fokus erhalten hat, wird über eine doppelte Umrahmung der Box dargestellt. Den Job unter dem Fokus kann man durch einen Tastendruck auf `r` starten beziehungsweise erneut versuchen. Durch den Druck auf `c` wird der Job abgebrochen. Meist wird man sich jedoch für die Logs interessieren, auf die man über `t` Einblick erhält *(siehe Abbildung 4)*. Ein erneuter Druck geleitet uns wieder zurück zur Pipeline-Übersicht. Mit `T` werden die bisherigen Job-Ausgaben auf der Konsole wiedergegeben, sodass die Ausgaben auch nach Verlassen der Pipeline-Ansicht noch verfügbar sind.

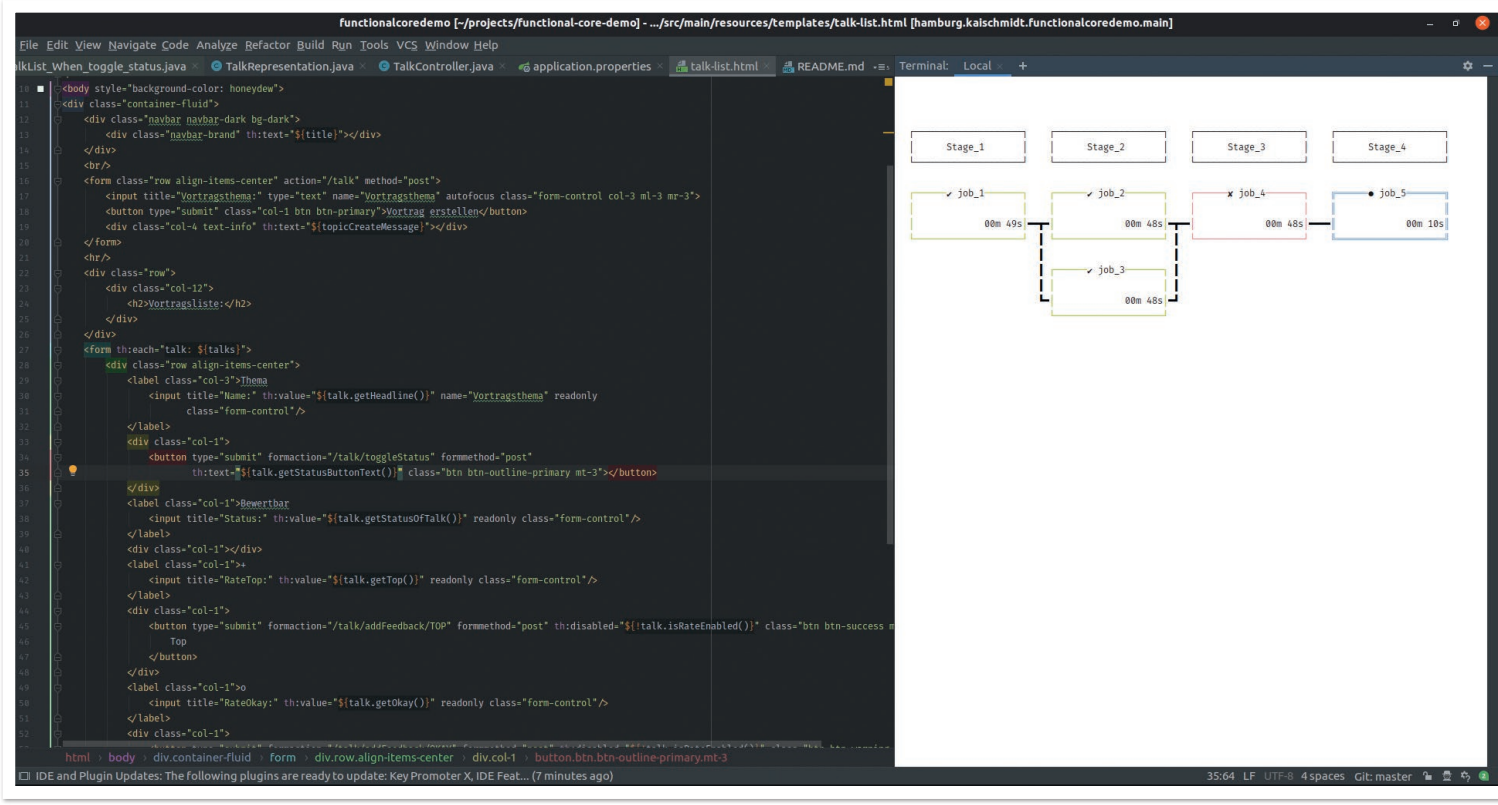

Abbildung 3: Die Pipeline in der Konsole einer IDE (hier IntelliJ IDEA) (Quelle: Kai Schmidt)

Abbildung 4: Beispielhaftes Log eines Jobs (Quelle: Kai Schmidt)

```
$ alias git=hub

# "hub clone github/hub" wird zu
$ git clone github/hub

# Hub-Funktionalitäten laufen nun augenscheinlich mit git
$ git pull-request -b develop

# Funktionen ohne direkte Hub-Unterstützung funktionieren weiter wie gewohnt
$ git branch -a
```

Listing 4: Aliasing am Beispiel von Hub

Falls man mit der WSL arbeitet, beschreiben die Referenzen *[11]* und *[12]* die Einbindung derselben in IntelliJ und in Eclipse.

Erleichterung durch Aliasing und Tab Completion

Je nach persönlichem Geschmack ist es ratsam, für das jeweilige CLI-Tool ein Aliasing auf `git` zu definieren. Sämtliche Befehle, die von Hub beziehungsweise Lab nicht direkt interpretiert werden können, werden weiterhin zu Git durchgeschleust, sodass sich ein Gefühl einstellt, als seien die Funktionen bereits immer direkt in Git enthalten gewesen *(siehe Listing 4)*.

Den Funktionsumfang von Hub und Lab hat man schnell gelernt. Beide Tools beschreiben in ihrer Readme, wie das Aliasing auf UNIX-basierten Systemen eingerichtet werden kann. Selbstverständlich kann man je nach Vorliebe den Alias wechseln, sobald man auf die andere Plattform wechselt. Oder man stellt den Alias für die Plattform ein, die man zum Großteil verwendet. Lab erkennt aber auch Hub, sodass es ebenfalls möglich ist, Lab als alleinige Aufrufschnittstelle über alle drei CLIs zu verwenden. Der beispielhafte Aufruf von

```
$ lab version
git version 2.20.1
hub version 2.7.0
lab version 0.16.0
```

Listing 5: Lab verträgt sich sowohl mit Hub als auch mit Git

```
# Für ein GitLab-Repository
$ lab mr create origin develop

# Für ein GitHub-Repository unter Verwendung von Lab
$ lab pull-request -b develop -m "merge branch to develop"
```

Listing 6: Die Befehle zwischen Lab und Hub sind nicht kompatibel

`lab version` resultiert dann in der Ausgabe von *Listing 5*. Dennoch müssen die Befehle von Hub genutzt werden, um erfolgreich mit einem GitHub-Repository umgehen zu können – wie *Listing 6* zeigt.

Als weitere Erleichterung beim Tippen lassen sich für beide Tools bash- und zsh-Tab-Completions einrichten, deren Installation ebenfalls der Readme-Seite des jeweiligen Projekts zu entnehmen ist.

Fazit

Hub und Lab portieren die Zusatzfunktionen der beliebten Portale GitHub und GitLab auf die Kommandozeile. Die Befehle der beiden Tools sind eingängig und stehen – zumindest indirekt dank WSL oder Scoop – auch in Windows-Umgebungen zur Verfügung. Das einzige festgestellte Manko der WSL ist der ohne Wirkung verbleibende Absprung in den Browser über die jeweiligen browse-Befehle. Der Workaround über `lab mr show` und der damit angezeigten und klickbaren URL genügt jedoch meist als Workaround.

Auch wenn Lab noch keine 1.0-Version erreicht hat, machen beide Tools einen sehr ausgereiften Eindruck und verfügen über ein Issue-Tracking auf den jeweiligen GitHub-Seiten. Falls Befehle oder deren Flags etwas schwer zu merken sind oder sich wiederkehrende Befehlsfolgen erkennen lassen, kann man diese nun sehr leicht mit einem Bash-Skript automatisieren. Durch „Sourcing" in der .bashrc beziehungsweise .zsh sind sie automatisch als entsprechender Kommandozeilen-Aufruf verfügbar.

Man wird sich entweder GitHub oder GitLab als heiligen Gral auserkoren haben und somit auf das jeweilige Tool stürzen. Für Nutzer beider Portale sind Hub und Lab weiterhin einzeln nutzbar. Bei Bedarf bringt man über Lab nicht nur Lab selbst, sondern – wenn auch nicht ideal – ebenso Hub und Git unter eine „CLI-Haube".

Quellen

[1] *https://github.com*
[2] *https://github.com/github/hub*
[3] *https://gitlab.com*
[4] *https://github.com/zaquestion/lab*
[5] *https://about.gitlab.com/stages-devops-lifecycle/*
[6] *https://docs.microsoft.com/de-de/windows/wsl/install-win10*
[7] *https://docs.microsoft.com/de-de/windows/wsl/install-manual*
[8] *https://www.phillip-kruger.com/post/some_bash_functions_for_git/*
[9] *https://github.com/nvie/gitflow*
[10] *https://docs.gitlab.com/ee/workflow/gitlab_flow.html#mergepull-requests-with-gitlab-flow*
[11] *https://medium.com/@thomas.schmidt.0001/how-to-use-ubuntu-bash-on-windows-10-as-the-intellij-idea-terminal-334fd9a10d8c*
[12] *https://superuser.com/questions/1416461/opening-linux-subsystem-for-windows-shell-from-eclipse-photon-ide*

Kai Schmidt
selbstständig
mail@kai-schmidt.hamburg

Kai Schmidt ist freiberuflicher Software-Entwickler und -Architekt. Zuvor war er bei den IT-Beratungsunternehmen .msg systems ag und Capgemini angestellt und in seiner über 10-jährigen Projekterfahrung größtenteils in Java- und C#-Projekten in den Bereichen Logistik, Flugzeugbau sowie Handel tätig. Hub und Lab haben ihn sofort begeistert, nachdem er vor einigen Wochen über Twitter-Tweets Kenntnis davon gewonnen hatte. Heute berät und beteiligt er sich gerne an betrieblichen Anwendungssystemen und ist in der JUG sowie für Kids4IT engagiert. Kai ist auch als Speaker auf Konferenzen und Meetups aktiv.