dataport

anderScore
trust in competence

Tag 1: Einführung in Git und GitLab, Git-Workflow im Team

17.06.2024, Daniel Krämer & Malte Fischer

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Einführung in

GitLab

GitLab – Einführung

Inhalt

- Einführung
  - Was ist GitLab
  - Git vs GitLab
  - Versionen
- Live-Demo GitLab
  - Projekte erstellen
  - Gruppen
  - Arbeiten im Projekt
  - Planing Tools
  - Branches und Merge-Requests
- Übung zu GitLab

GitLab – Einführung

anderScore trust in competence

Was ist GitLab?

- 2011 von Dimitri Saparoschez und Valery Sizov entwickelt
- Service zur Softwareentwicklung und Versionsverwaltung
- Hosting von Git Remote Repositories als Projekte
- Zusätzliche Funktionen im Bereich:
  - Issue-Tracking: Verwaltung/Verfolugung von Aufgaben
  - Dokumentation
  - CI/CD (Continuous Integration/Continuous Deployment)

Git vs GitLab

Git

- Verteiltes Versionskontrollsystem zur lokalen Verwaltung von Dateien
- Verfolgung und Speicherung von Änderungen an Dateien

GitLab

- Umfassende DevOps-Plattform
- Erweitert Git um Tools und Funktionen für Projektmanagement, Zusammenarbeit und Automatisierung
- Zentrale Plattform zur Verwaltung von Projekten

Unterschiedliche GitLab Versionen

- Free
  - Kostenlos und Open Source
  - Konzipiert für persönliche Projekte
  - 400 compute minutes für CI/CD
  - Support nur über GitLab Forum
- Premium
  - Für mittlere bis große Teams
  - Fortgeschrittene Features im Bereich Projektmanagement, Code Reviews sowie CI/CD
  - 10.000 compute minutes für CI/CD
  - Support bei Problemen

GitLab – Einführung

- Ultimate
  - Kostenlos und Open Source
  - Für Unternehmen und große Organisationen
  - Umfangreiche Statistiken zur Analyse
  - Erweiterte Funktionen insbesondere im Bereich Security und Automatisierung
  - 50,000 compute minutes für CI/CD
- Versionen können als SaaS auf GitLab.com oder selbst gehostete Instanz im eigenen Netzwerk betrieben werden
- Zusätzliche Add-ons mit Fokus auf AI-Unterstützung für Premium und Ultimate verfügbar

GitLab

GitLab

Übung

Aufgabe 1: Neues Projekt erstellen

1. (Optional) Beim Arbeiten mit GitLab ist es sinnvoll mittels SSH Keys auf Remote Repositories zuzugreifen, statt immer wieder Username und Passwort zu verwenden.
   Falls Sie noch kein SSH Key in GitLab hinterlegt haben, so können Sie entweder die Anleitung aus dem GitLab nutzen oder dem Tutorial unter https://docs.gitlab.com/ee/user/ssh.html folgen.
2. Legen Sie ein neues privates Projekt Playground in Ihrem eigenen Bereich an, sodass nur Sie selbst Zugriff darauf haben. Achten Sie darauf, dass Sie das Repository mit einer Readme initialisieren, um dieses direkt clonen zu können.
3. Clonen Sie das Repository in ein lokales Verzeichnis auf Ihrem Rechner.

Hinweis

Man kann grundsätzlich viele Änderungen am Projekt sowohl lokal als auch in der GitLab Weboberfläche vornehmen. Für gewöhnlich bearbeitet man beispielweise eher selten Dateien in der GitLab Weboberfläche, sondern nimmt Änderungen lokal in einer IDE vor und pusht diese danach ins Remote Repository.

Um das Arbeiten mit mehreren Entwicklern im Projekt etwas nachzustellen, werden wir im Folgenden auch Änderungen über die Weboberfläche vornehmen, um damit Änderungen von anderen Entwicklern zu simulieren.

Dadurch lassen sich verschiedene Szenarien erzeugen, deren Behandlung wir an unserem lokalen Repository üben werden.

Aufgabe 2: Erste Schritte im Repository

1. Navigieren Sie in einem Terminal in das neue geklonte Projekt auf Ihrem Rechner.
2. Mit dem Befehl `git status` können Sie Ihren aktuellen Stand abfragen.
   Sie sollten sich auf dem Branch main befinden, der sich auf demselben Stand wie origin/main befindet.
3. `git status` vergleicht nur den lokalen Remote Tracking Bereich mit Ihrem aktuellen Branch. Es gibt keinen Aufschluss darüber, ob Remote neue Änderungen verfügbar sind.
   Nutzen Sie `git fetch`, um ihren lokalen Remote Tracking Bereich mit dem GitLab Repository zu synchronisieren.
   Es sollten keine neuen Änderungen verfügbar sein.

Aufgabe 2: Erste Datei hinzufügen

1. Legen Sie eine Datei random_numbers.sh an und füllen Sie diese mit dem folgenden Inhalt

```
#!/bin/bash

random_number=$(( RANDOM % 10 + 1 ))

if [ $random_number -lt 11 ]; then
    echo "Number is less than 11"
fi
```

2. Committen Sie Ihre Änderungen auf dem main Branch.
3. Pushen Sie Ihre Änderungen ins Remote Repository. Vergewissern Sie sich über die Web-GUI, dass Ihr Push erfolgreich war.

Aufgabe 3: Issue erstellen

1. Über das Issue Board in GitLab können Aufgaben definiert und an bestimmte Personen zugewiesen werden.
   Legen Sie im Issue Board ein neues Issue Add output if number greater than 5 an.
   Diese Aufgabe sollen Sie nun übernehmen, tragen Sie sich daher als Assignee ein.
2. GitLab bietet die Möglichkeit aus Issues direkt Merge-Requests und Branches zu erstellen.
   Klicken Sie auf das Dropdown beim Button „Create merge request and branch“ und wählen Sie nur „Create branch“ aus.
   Erstellen Sie nun eine Branch, um die Änderungen für das Feature vorzunehmen.

Aufgabe 4: Lokale Änderungen

1. Führen Sie in Ihrem lokalen Repository den Befehl `git status` sowie `git branch -a` aus.
   Ihnen sollte der neu angelegte Branch aus GitLab nicht angezeigt werden, da Sie diesen erst vom Remote abrufen müssen.
2. Fetchen Sie alle Änderungen vom Remote.
   `git branch -a` sollte nun den neuen Branch anzeigen.
3. Wechseln Sie in den Feature Branch.
4. Öffnen Sie die random_numbers.sh Datei und nehmen Sie die geforderten Änderungen vor, indem Sie die Datei um ein weiteres if ergänzen, indem geprüft wird, ob die Zahl größer als 5 ist.
5. Committen Sie Ihre Änderungen und pushen Sie diese ins Repository.

Aufgabe 5: Merge-Request

Nun wollen wir unsere Änderungen des Features in den main Branch übernehmen. Lokal würde man den Feature Branch in den main Branch mergen. Der Ablauf bei GitLab ist funktional sehr ähnlich, erweitert jedoch das reine Merging um einige Aspekte.

1. Navigieren Sie über die Sidebar in der GitLab Web-GUI auf den Punkt Merge-Requests.
2. Erstellen Sie einen neuen Merge-Request, um Ihren Feature Branch in den main Branch zu mergen.
   Weisen Sie sich als Assignee und Reviewer zu (normalerweise würde man logischerweise jemand anderes als Reviewer eintragen).
3. Im Merge-Request haben Sie oben die Option, sich Commits und Changes anzuschauen. Reviewen Sie Ihre Änderungen und approven Sie den Merge-Request.

Aufgabe 5: Merge-Request

4. Mergen Sie nun den Merge-Request mit der Option „Delete source branch“ und verifizieren Sie, dass die Änderungen auf dem main Branch vorhanden sind.
5. Löschen Sie Ihren lokalen Branch, da dieser abgeschlossen ist und keine Verwendung mehr besitzt.

Aufgabe 6: Rebase von lokalen Branches

Nun wollen wir ein Rebase nutzen, um Remote Änderungen in unseren lokalen Branch zu integrieren.

1. Erstellen Sie ein neues Issue Add output if number is maximum.
   Dabei soll eine Ausgabe erfolgen, wenn die Random Nummer die größtmögliche Zahl ist.
2. Man muss nicht zwingend den Branch aus einem Issue heraus erstellen.
   Legen Sie dieses Mal lokal einen Branch 2-add-output-if-max an und führen Sie die notwendigen Änderungen durch.
3. Committen Sie Ihre Änderungen und pushen Sie diese zum Remote Repository.

Aufgabe 6: Rebase von lokalen Branches

In der Zwischenzeit hat ein Kollege die Aufgabe bekommen, den Zahlenbereich von 10 auf 1000 zu erhöhen.
Um die Änderungen von Ihm zu simulieren, führen wir diese über die Web-GUI durch. Ein zugehöriges Issue bzw. einen Branch überspringen wir zur Simplifizierung.

Hinweis Ein gemeinsames Arbeiten über eine Datei sollte im Optimalfall vermieden werden, um Konflikten zu vermeiden.

4. Ändern Sie über GitLab den Random Number Bereich auf 1000 und committen Sie die Datei direkt aus dem Editor auf den main Branch.
5. Um im lokalen Repository die Änderungen zu übernehmen, müssen Sie den main Branch updaten. Wechseln Sie dazu in diesen und verwenden Sie `git pull`, um diese in Ihren lokalen main Branch zu übernehmen.

Aufgabe 6: Rebase von lokalen Branches

7. Um die Änderungen nun auch in den Feature Branch zu übernehmen, müssen Sie in den Feature Branch wechseln und dort auf den main Branch rebasen.
8. Rebasen Sie Ihren Feature Branch auf den main Branch. Dabei sollten keine Konflikte auftreten, da sich die betreffenden Zeilen bei Ihnen nicht geändert haben.
9. Passen Sie Ihre Ausgabe bezüglich der Random Number an und committen Sie Ihre Änderungen.
10. Versuchen Sie Ihren Branch zum Remote zu pushen. GitLab wird hierbei den Push wegen nicht zueinander passenden Commit-Historien ablehnen. Pushen Sie daher Ihren Branch mit der Option `--force`.

Aufgabe 6: Rebase von lokalen Branches

Wichtig `--force` bewirkt, dass der Remote Branch durch die lokale Version überschrieben wird.
Falls andere Entwickler auf diesem Branch arbeiten würden, so könnten Sie weder push noch pull auf diesem ausführen, da die Commit-Historien nicht vereinbar sind. Ihre Branches sind damit kaputt und müssen komplett neu vom Remote abgerufen werden.

Daher niemals auf Public-Branches rebasen!

Aufgabe 7: Merge-Konflikte

In dieser Aufgabe wollen wir uns das Auflösen von Merge-Konflikten anschauen.

Ein Kollege von Ihnen hat ebenfalls das Feature bezüglich Ausgabe bei maximaler Zahl umgesetzt und dieses schon in den main Branch integriert.

1. Simulieren Sie die Änderungen des Kollegen, indem Sie auch hier wieder über die Web-GUI direkt auf dem main Branch arbeiten. Kopieren Sie Ihre lokalen Änderungen bezüglich der Ausgabe und fügen sie diese über die Web-GUI in die random_numbers.sh Datei ein.
   Verändern Sie dabei die Konsolenausgabe, sodass sich die Änderungen unterscheiden.
2. Committen Sie Ihre Änderungen direkt auf main.

Aufgabe 7: Merge-Konflikte

3. Stellen Sie für Ihren Feature Branch ein Merge-Request. Nach der Erstellung sollte GitLab anzeigen, dass es Konflikte beim Mergen gibt.
4. Kleinere Konflikte lassen sich über die Web-GUI auflösen, größere Konflikte lassen sich nur lokal auflösen.
   Zur Übung lösen wir den Konflikt lokal auf.
5. Updaten Sie den main Branch auf den neusten Stand aus dem Remote Repository.
6. Lösen Sie die Konflikte nun, indem Sie entweder den main Branch in Ihren Feature Branch mergen oder indem Sie den Feature Branch auf den aktuellen Stand des main rebasen. Da Sie alleine auf ihrem Branch arbeiten, bietet sich auch hier ein Rebase an.
7. Lösen Sie den Konflikt, indem Sie sich für Ihre Ausgabe entscheiden.

Aufgabe 7: Merge-Konflikte

8. Pushen Sie nach Auflösen des Konflikts Ihre Änderungen ins GitLab.
9. Der Merge-Request sollte nun ohne Konflikte umsetzbar sein. Mergen Sie Ihr Feature in den main Branch, um die Änderungen abzuschließen.