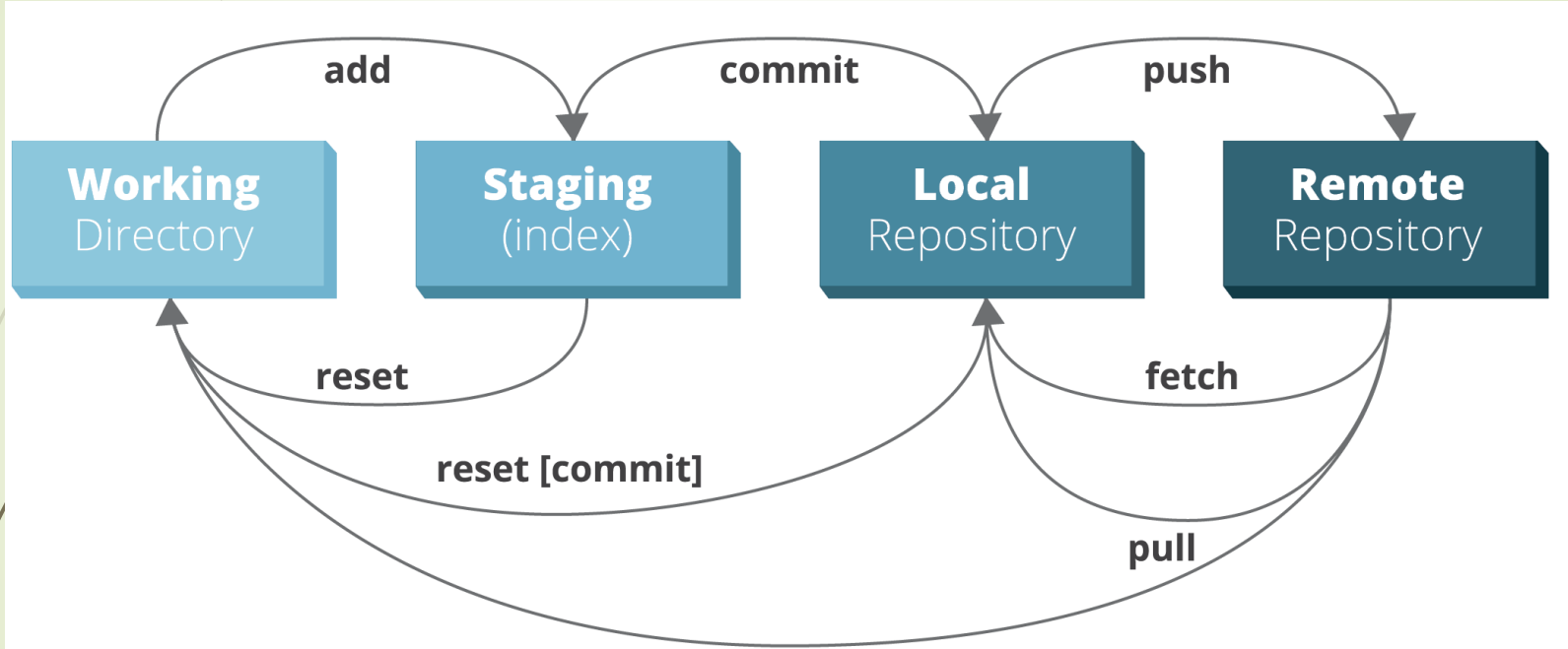
Workshop zum SoPra

Kai Mindermann M.Sc.

Ziele und Inhalte des Workshops: **Git** Basics und **GitLab** Basics

- Wichtigste Arbeitsabläufe von git kennenlernen
  - Änderungen im lokalen Repository eintragen
  - Änderungen in ein entferntes Repository übertragen
  - Konflikte auflösen
  - git tags erstellen
- GitLab einrichten und am Team-Projekt arbeiten
  - SSH-Schlüssel erstellen
  - Projekt klonen
  - Webeditor verwenden
  - Arbeiten mit Issues

Git Funktionsweise



Simon Maple/RebelLabs Git Cheatsheet

Vorbereitung

Installation von Werkzeugen

Benötigte Werkzeuge
Bitte spätestens jetzt installieren!

- Aktuelle Version (2.14.2) von **git**: https://git-scm.com/download
  - Inklusive Git Bash (mit Standardeinstellungen) installieren
- Aktuelle Version von **Puttygen zur Erstellung eines SSH-Schlüssels**: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Markdown-Editor **Haroopad**: http://pad.haroopress.com/user.html
- TortoiseGit: https://tortoisegit.org/

- ***Sie können auch andere Programme als die hier genannten verwenden.***

- *Aktueller Browser: z. B. Mozilla Firefox, Google Chrome, Microsoft Edge)*
- *Packprogramm: z. B. PeaZip (http://www.peazip.org/)*
- *Text-Editor: z. B. Notepad++ (https://notepad-plus-plus.org/)*

Arbeitsumgebung vorbereiten

- Im **Ilias** ist eine **ZIP-Datei** bereitgestellt worden, in der sich der Ordner „Git-Workshop“ befindet
- Laden Sie die Datei herunter: **Git-Workshop.ZIP** (verlinkt auf der Webseite)
- **Entpacken Sie die ZIP-Datei auf Ihrem Desktop**.
- Öffnen Sie die Folien, damit Sie unabhängig vom Vortragenden und der anderen weiterarbeiten können.
- Klicken Sie mit der **rechten Maustaste** auf den Ordner „Git-Workshop“ auf Ihrem Desktop
  - Klicken Sie im erscheinenden Menü auf „**Git Bash Here**“.
- Öffnen Sie den Ordner „Git-Workshop“ auf Ihrem Desktop (mit dem Windows Explorer)

Erster Teil: Arbeiten mit git

Erster Teil: Arbeiten mit git

- Ziel des ersten Teils ist das Kennenlernen der wichtigsten Befehle beim Arbeiten mit git.
- Dafür wird zunächst ein Repository geklont, im lokalen Repository(-Klon) gearbeitet. Danach werden die Änderungen im lokalen Repository eingetragen und abschließend in das entfernte Repository übertragen.
- Das entfernte Repository wird in dieser Aufgabe simuliert. Es liegt einfach in einem anderen Ordner. Die Funktionsweise ist aber genau gleich.

Vorhandenes Repository klonen

- Im Fenster „Git Bash“ wird Ihnen eine Eingabeaufforderung mit dem aktuellen Verzeichnis innerhalb des Ordners „Git-Workshop“ angezeigt.
- Geben Sie den Befehl **`git clone remoteRepo.git localRepo`** ein und bestätigen Sie mit der Taste Enter ↵

- Git klont das vorhandene Repository für Sie in den neuen Ordner „localRepo“
- Wechseln Sie in den Ordner „localRepo“ mit dem Befehl **`cd localRepo`**
- Wechseln Sie auch im Explorer in den Ordner „localRepo“
  - Sie müssten einen Ordner „doc“ und eine Datei „Readme.md“ sehen.

Arbeiten im localRepo vorbereiten

- Im Verzeichnis localRepo in Git Bash
- Git die Informationen zum Autor geben:
  - Befehl: `git config user.name "Vorname Nachname"`
  - Befehl: `git config user.email "st000000@stud.uni-stuttgart.de"`
  - *Verwenden Sie hier Ihre eigenen Daten. Diese Daten sind erforderlich und werden von git in den Commit-Daten gespeichert!*
- Legen Sie einen neuen Zweig an:
  - Befehl: `git branch newAppNameAndDescription`
- Checken Sie den neuen Zweig aus:
  - Befehl: `git checkout newAppNameAndDescription`

Arbeiten im localRepo

- Im Explorer Doppelklick auf **Readme.md** machen bzw. diese in Haroopad öffnen
  - Ändern Sie „**App-Name**“ in „**Awesome SoPra Guide**“
  - Fügen Sie unterhalb des Bildschirmfotos folgende **Beschreibung** hinzu: „**The Awesome SoPra Guide helps lost students to easily find what they are looking for. It does this by simply taking a selfie **automatically** and uploading it to Facebook with the following annotation: Comment if you know what I‘m looking for, thanks ☺**“
- Tragen Sie die gemachten Änderungen im Repository ein
  - Befehl: `git add Readme.md`
  - Befehl: `git commit -m "updates app name and description"`

Änderungen in den Hauptentwicklungszweig übernehmen

- Wechseln Sie vom Feature-Zweig zum Hauptentwicklungszweig
  - Befehl: `git checkout master`
- Übernehmen Sie die Änderungen im Feature-Zweig in den Hauptentwicklungszweig
  - Befehl: `git merge newAppNameAndDescription`

Änderungen in entferntes Repository übertragen

- Um die Änderungen vom lokalen Repository „localRepo“ in das entfernte Repository „remoteRepo.git“ zu übertragen reicht folgender Befehl
  - Befehl: `git push`
  - (Überträgt nur den aktuell ausgecheckten Zweig)

Konflikte lösen

Wechsel in „otherRepo“

- Wechseln Sie in der Git Bash das Verzeichnis „otherRepo“ wie folgt:
  - Befehl: `cd ..`
  - Befehl: `cd otherRepo`
- Stellen Sie sich vor Sie sind einer der anderen Entwickler aus Ihrem Team.
- Jetzt befinden Sie sich im otherRepo. Dort wurde auch schon an der Readme.md-Datei [von dem/der Anderen] gearbeitet.
- Die Änderungen wurden auch schon im lokalen Repository eingetragen.

Änderungen aus „otherRepo“ in das „remoteRepo“ übertragen

- Übertragen Sie die Änderungen aus otherRepo zum remoteRepo:
  - Befehl: `git push`
- Der Befehl liefert eine **Fehlermeldung**, die sagt, dass im entfernten Repository schon Änderungen eingetragen wurden, die in Ihrem lokalen Repository fehlen! (Klar, da wir vorhin ja Änderungen dorthin übertragen haben)
- Holen Sie sich diese Änderungen:
  - Befehl: `git fetch`
- Übertragen Sie nun die Änderungen aus otherRepo erneut zum remoteRepo:
  - Befehl: `git push`
- Es kommt **wieder zu einem Fehler**. Die geholten Änderungen stehen im Konflikt zu Ihren lokalen Änderungen. Sie müssen die Konflikte auflösen, um einen Stand zu erreichen, der eingetragen werden kann.
  - Befehl: `git merge`
- Die Änderungen konnten nicht automatisch zusammengeführt werden. **Manuelles Eingreifen erforderlich!**

Konflikte auflösen

- Sie können Konflikte auf 3 Arten lösen:
  - Sie verwerfen Ihre lokalen Änderungen (zumindest an den im Konflikt stehenden Dateien) oder
  - Sie verwerfen die Änderungen der Anderen und übernehmen nur Ihre Änderungen oder
  - **Sie bearbeiten die im Konflikt stehenden Dateien einzeln, um die konkreten Konflikte zu beheben**.
- Öffnen Sie die Datei Readme.md wieder in Haroopad
- Entscheiden Sie sich, welcher App-Name und welche Beschreibung die geeignete für Ihre App ist. Sie können sich auch komplett neue Inhalte überlegen.
- Entfernen Sie die Konflikt-Markierungen in der Datei.
  - Konfliktmarkierungen sind ganze Zeilen beginnend mit „<<<<<<<„, „=======„ und „>>>>>>>“.
- Wenn alle solche Konfliktmarkierungen entfernt wurden. Speichern Sie die Datei.

Neue Eintragung mit zusammengeführten Änderungen erstellen und übertragen

- Tragen Sie die Änderungen im Repository ein
  - Befehl: `git add Readme.md`
  - Befehl: `git commit -m „merges conflicting app name and description"`
- Übertragen Sie die Änderungen wieder in das remoteRepo
  - Befehl: `git push`

Abgabe/Release/Tag erstellen

git tag mit allen aktuellen Änderungen

- Sie sind wieder Sie selbst und wechseln dazu in das Verzeichnis localRepo
  - Befehl: `cd ..`
  - Befehl: `cd localRepo`
- Holen Sie sich die aktuellsten Eintragungen aus dem remoteRepo
  - Befehl: `git pull`
  - (git pull führt git fetch und git merge aus)
- Erstellen Sie einen neuen tag mit dem Namen „alpha-1“:
  - Befehl: `git tag –a alpha-1 –m "first alpha release“`
- Übertragen Sie den tag in das entfernte Repository
  - Befehl: `git push origin alpha-1`

Änderungen rückgängig machen (LOKAL bevor sie übertragen wurden)

(Bereits übertragene Änderungen können durch neue Commits rückgängig gemacht werden, indem sie die vorhergehenden Änderungen mit neuen Änderungen überschreiben)

Lokale Änderungen verwerfen

- Wechseln Sie in den Ordner „advancedOtherRepo“
- Versuchen Sie Ihr Glück dort (`cd ..; cd advancedOtherRepo`) mit `git pull`
- Verwenden Sie `git status`, um Hilfe zum Status Ihres Repositories zu bekommen.

- Für die Eintragung vorbereitete Änderungen (staging-Area) zurücksetzen:
  - `git reset HEAD filename`
  - `git reset is only` DANGEROUS `with --hard`
  - `git reset --hard`
  - `git revert COMMITID`

- Änderungen an der Datei verwerfen:
  - `git checkout -- filename` (DANGEROUS)

- Änderungen temporär sichern:
  - `git stash`
  - `git stash pop`
- https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things

Zweiter Teil: Arbeiten mit GitLab

GitLab Basic Features

- Social Coding Web-Plattform
- Git Repository Hosting
  - File-Browser
  - Commit-View
- Document View (Rendered Markdown)
- Issue-/Bug-Tracker inkl. Meilensteinen, Zuordnung von Issues und Kommentaren (mit Markdown)
- Merge-Request/Code-Review-Tool
- Einfache Forks
- Continuous Integration
- User/Project/Group Avatars und Emojis
- Activity-Stream/Notifications/RSS für Projekte/Gruppen

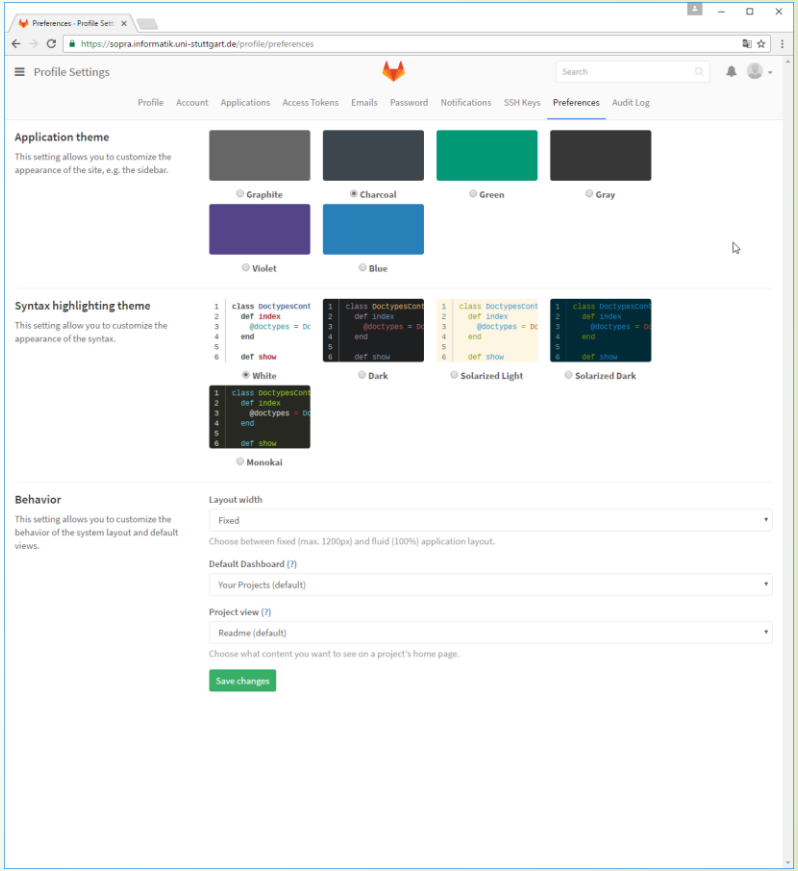
Konto beim SoPra GitLab

- Wird von Betreuern erstellt. Keine Selbst-Registrierung möglich!
- Man bekommt an seine **studentische E-Mail** einen **Link zum Zurücksetzen des Passworts**!

- URL von GitLab ist: https://sopra.informatik.uni-stuttgart.de

GitLab Profile

- Überblick verschaffen
- GitLab Anzeige konfigurieren



Git und GitLab für Zugriff mit SSH-Schlüssel einrichten

- SSH-Schlüssel hinterlegen
  - **Puttygen öffnen und geöffnet lassen (oder anderes Programm verwenden)**
  - SSH-2 RSA Schlüssel erstellen (Generate)
  - Den **GESAMTEN Inhalt** (beim Markieren nach unten scrollen) des Textfelds kopieren (STRG+C)
  - In GitLab in das entsprechende Eingabefeld im Profil unter SSH Keys einfügen (STRG+V)
  - Add Key anklicken
- Speichern Sie den Schlüssel für TortoiseGit in Puttygen als **ppk-Datei** („Save private key“)
- **Erstellen Sie mit Git Bash den Ordner „.ssh“ mit dem Befehl „`mkdir .ssh`“ im Benutzerverzeichnis**
- Exportieren Sie den Schlüssel im OpenSSH-Format in den Ordner „.ssh“(Conversions->Export OpenSSH Key) mit dem Namen „id_rsa“ (falls schon vorhanden, anderern Namen wählen, siehe nächste Folie)

  - http://serverfault.com/a/198691/118167
  - https://sopra.informatik.uni-stuttgart.de/help/ssh/README

Auswahl des SSH-Schlüssels für git konfigurieren (falls nicht als `id_rsa` gespeichert)

- Erstellen Sie die neue Datei „%USERPROFILE%\.ssh\config“
- Fügen Sie folgenden (zusätzlichen) Inhalt in diese Datei ein:

```
Host sopra.informatik.uni-stuttgart.de
  HostName sopra.informatik.uni-stuttgart.de
  IdentityFile ~/.ssh/sopra-key
```

  - Verwenden Sie den gerade eben gewählten Namen der exportieren Schlüssel-Datei

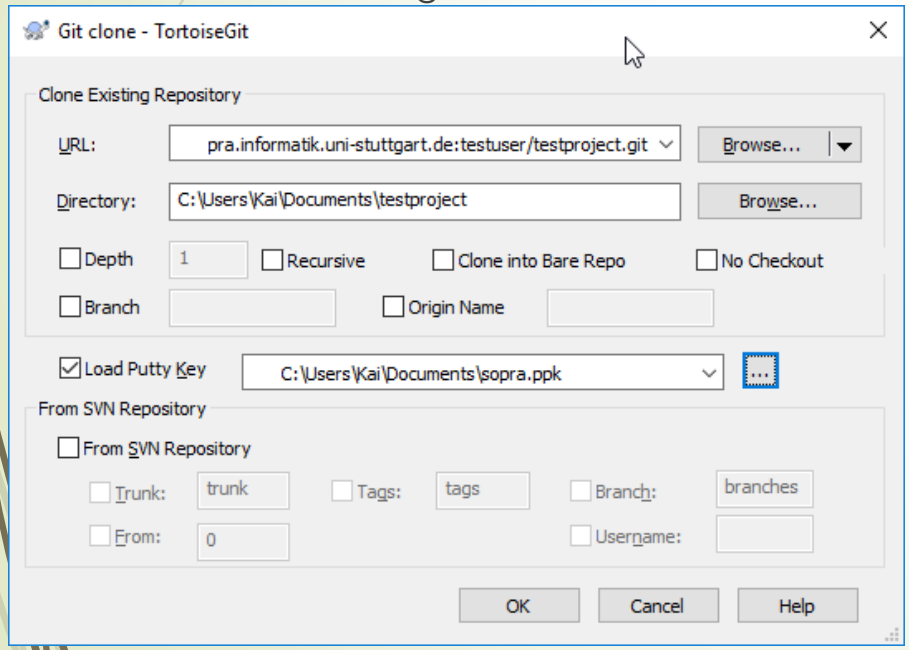
- http://stackoverflow.com/a/2419609/455578

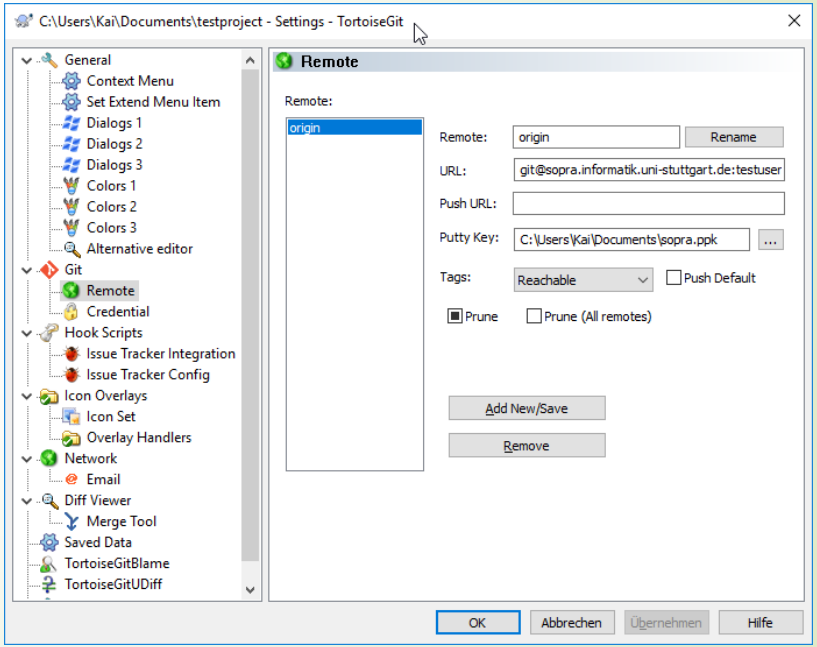
Team-Projekt/Repository klonen

- Holen Sie sich das für Ihr Team erstellte git-Repository auf Ihren Rechner
- Öffnen Sie dazu in GitLab das Projekt
  - Über Ihre Einladungs-E-Mail
  - Über die Liste Ihrer Projekte in Ihrem Profil in GitLab
- Dort finden Sie unterhalb des Projektnamens ein Feld mit einer Adresse wie dieser hier:
  - git@sopra.informatik.uni-stuttgart.de:sopra-ws1617/sopra-team-x.git
  - **Kopieren Sie die dort angezeigte Adresse (STRG+C)**
- Gehen Sie in auf Ihrem PC in den Ordner in dem Sie später an dem SoPra-Projekt arbeiten möchten (Explorer oder Git Bash)
  - Öffnen Sie mit Rechtsklick auf den Ordner das **TortoiseGit**-Menü
  - **Oder** öffnen Sie die **Git Bash** für diesen Ordner

SSH-Key für TortoiseGit angeben

- Geben Sie beim Klonen direkt ihren .ppk-Schlüssel an
- Alternativ können Sie später mit Rechtsklick auf den Ordner mit dem git Repository, dann TortoiseGit->Settings unter Git->Remote den Putty Key angeben





Arbeiten mit GitLab

Jeder Entwickler Ihres Teams macht jetzt etwas anderes! ① **oder** ② **oder** ③

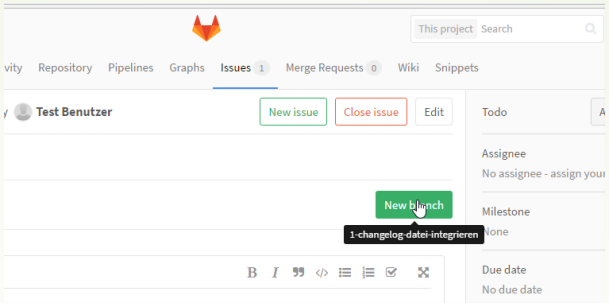
Die Änderungen sind eine Vorbereitung auf das SoPra und schaden Ihrem Repository bzw. GitLab-Projekt nicht!

Issue im Team-Projekt in GitLab erstellen

- Erstellen Sie einen neuen Issue
- Klicken Sie dazu auf „Issues“ und dann auf „New Issue“

- Geben Sie nur einen Titel an:
  - „Neues Definition of Done Kriterium“ ①
  - „Critical Feature 1 ins Product Backlog kopieren“ ②
- Erstellen Sie den Issue mit einem Klick auf „Submit Issue“

Neuer Zweig für Arbeit an dem Issue



- Erstellen Sie einen neuen Zweig für den Issue
  - Klicken Sie **im Issue** dazu auf den Knopf „New branch“
- Der neue Zweig wird für Sie online **direkt ausgecheckt**
- Führen Sie die Änderung für **IHREN** Issue entsprechend **IHRER nachfolgenden Folie** aus
  - „Neues Definition of Done Kriterium“ ①
  - „Critical Feature 1 ins Product Backlog kopieren“ ②

Neues Definition of Done Kriterium ①

- Öffnen Sie im Online-Editor die Datei Definition.of.Done.md
  - Klicken Sie im GitLab-Projekt auf „Files“
  - Wechseln Sie in den Unterordner „doc“ in dem Sie darauf klicken
  - Klicken Sie nun auf die Datei „Definition.of.Done.md“
  - Klicken Sie nun auf „Edit“ oberhalb der Anzeige des Inhalts der Datei
- Fügen Sie das Kriterium „Changelog-Eintrag hinzugefügt“ hinzu
- Ändern Sie die Commit-Nachricht zu „adds new criterion in DoD to add changelog entry“
- Tragen Sie die Änderungen mit der neuen Datei durch Klick auf „Commit Changes“ ein

Merge Request für „Neues Definition of Done Kriterium“①

- Sie sind nun der Meinung, dass Sie alles erforderliche für diesen Issue getan haben.
- Erstellen Sie deshalb einen Merge Request dafür indem Sie auf den „Create Merge Request“ Knopf drücken.
- Nun werden Ihnen die Änderungen angezeigt und Sie können den neuen Merge Request mit Klick auf „Sumit merge request“ erstellen.

Helfen Sie jetzt den anderen beiden Entwicklern bevor Sie weitermachen

Dann gemeinsam hier weitermachen

Critical Feature 1 ins Product Backlog kopieren ②

- Öffnen Sie (in einem neuen Tab) das GitLab-Projekt zur SoPra-Dokumentation
  - https://sopra.informatik.uni-stuttgart.de/sopra-ws1718/SoPra-Doku-Entwickler
- Navigieren Sie dort zu den vorgegebenen Critical Features wie folgt:
  - Klicken Sie in diesem GitLab-Projekt auf „Files“
  - Dann auf den Ordner „Aufgabe“
  - Klicken Sie auf „Readme.md“
  - Lassen Sie sich den unverarbeiteten Dateiinhalt anzeigen, klicken Sie auf „Raw“
- Kopieren Sie den Text des ersten Features bis zum Beginn der Überschrift vom zweiten Feature (STRG+C)
- Wechseln Sie zurück zu Ihrem Team-Projekt (anderer Tab)

Critical Feature 1 ins Product Backlog kopieren ②

- Navigieren Sie dort wie gerade in den Ordner „doc“
- Und klicken Sie auf die Datei „Product.Backlog.md“
- Klicken Sie auf „Edit“ um das Product Backlog zu bearbeiten
  - Fügen Sie den kopierten Text an der richtigen Stelle ein.
  - Da hier noch keine anderen sinnvollen Änderungen gemacht wurden, fügen Sie es einfach **unterhalb der User Story des Epic 1** ein (STRG+V)
- Ändern Sie die Commit-Nachricht zu „adds critical feature 1 to product backlog (no implementable stories/tasks yet!)“
- Tragen Sie die Änderungen durch Klick auf „Commit Changes“ ein

Merge Request für „Critical Feature 1 ins Product Backlog kopieren“②

- Sie sind nun der Meinung, dass Sie alles erforderliche für diesen Issue getan haben.
- Erstellen Sie deshalb einen Merge Request dafür indem Sie auf den „Create Merge Request“ Knopf drücken.
- Nun werden Ihnen die Änderungen angezeigt und Sie können den neuen Merge Request mit Klick auf „Sumit merge request“ erstellen.

Helfen Sie jetzt den anderen beiden Entwicklern bevor Sie weitermachen

Dann gemeinsam hier weitermachen

Merge Requests bearbeiten

Gemeinsam als Team

Bearbeiten: Merge Request für „Neues Definition of Done Kriterium“ ①

- Sehen Sie sich alle Änderungen im „Changes“-Tab unten auf der Seite an
- Jetzt können Sie den Merge Request akzeptieren und mit ihrem Hauptentwicklungszweig „master“ zusammenführen.
  - Klicken Sie auf „Accept Merge Request“

Bearbeiten: „Critical Feature 1 ins Product Backlog kopieren“②

- Sehen Sie sich alle Änderungen im „Changes“-Tab unten auf der Seite an
- Ihnen fällt auf, dass Sie die Änderung noch im Changelog erwähnen sollten
- Editieren Sie die Datei „Changelog.md“ während Sie auf dem Feature Zweig bleiben
- Fügen Sie folgendes unterhalb von „Unreleased“ hinzu:

```
### Added

- Added new product backlog entry for critical feature
```

- Tragen Sie die Änderungen durch Klick auf „Commit Changes“ auf dem Zweig ein
- Jetzt können Sie den Merge Request akzeptieren und mit ihrem Hauptentwicklungszweig „master“ zusammenführen.
  - Klicken Sie auf „Accept Merge Request“

Hinweise zur Interaktion zwischen git und GitLab

- Sie können in der Commit-Nachricht einen Issue referenzieren:
  - „ref #xxx“ (xxx ist die Issue-Nummer)
  - Dann wird im Issue ein Link zu diesem Commit angezeigt
- Sie können mit einer speziellen Commit-Nachricht einen Issue schließen:
  - „closes #xxx“ (xxx ist die Issue-Nummer)
- Dies funktioniert auch bei den Commit-Nachrichten die Online verwendet werden (z. B. wie bei den Merge Requests vorgegeben)

Team Development Workflow

- Machen Sie sich Gedanken dazu, wie Sie arbeiten wollen.
- Sie können die GitLab Issues zur Koordination von Aufgaben verwenden, in dem Sie einen Issue einem Entwickler direkt zuweisen
- Die Zuweisung kann auch später geändert werden
- Sie können einen Workflow mit zusätzlichen Labels (neben denen die für die User Stories später benötigt werden) im **Issue Board** etablieren, um die Übersicht zu behalten, welche Aufgabe sich in welchem Status befindet.

Abschluss: Zusammenfassung

Thanks for your participation

- git clone, git branch, git add, git commit, git checkout, git merge, git push, git fetch, git pull, git **tag**

- Einrichten mit SSH-Schlüssel
- Erstellen von Issues
- Benutzen des Online-Editors

Quellenangaben

- The **GitLab logo** and wordmark artwork are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
- **Git Logo** by Jason Long is licensed under the Creative Commons Attribution 3.0 Unported License.