2. Praktikumstermin: Git-Anbindung an das zentrale Repository und Codeübergabe CocktailPro

Seit dem letzten Praktikumstermin haben Sie Ihren Rechner ‚lokal arbeitsfähig' gemacht: Git und CLion sind installiert und funktionieren. Sie haben sich vertraut gemacht mit der lokalen Nutzung von Git und können die Ergebnisse zu den Übungsaufgaben präsentieren und erklären.

Mit dem heutigen Termin beginnt das Arbeiten im Team. Dazu werden wir mit Ihnen zunächst die Anbindung Ihres Git an das zentrale Repository (mit dem Ausgangscode von CocktailPro) durchführen. Sie üben dann das Arbeiten im Team mit Git, mit individuellen Branches und die Zurückführung Ihrer Änderungen in die gemeinsame Codebasis, den "Master".

Bitte üben Sie zunächst in Ihrem Team die Schritte wie in Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.** beschrieben, bis Sie sich sicher fühlen, mit Git im Team zu arbeiten. Erst dann gehen Sie zum nächsten Abschnitt 2.2 und binden CLion an das zentrale Repository an. Damit können Sie dann die Aufgabe zur Nachbereitung erledigen.

Am Ende dieses Praktikumstermins haben Sie eine erste Vorstellung vom Arbeitszyklus zwischen Ihrer lokalen IDE mit einem individuellen Branch und dem zentralen Repository mit dem Masterbranch. Im Praktikum 3 werden wir diesen Arbeitszyklus erweitern zu einem vollständigen CI / CD- Zyklus (CI / CD = Continuous Integration / Continuous Deployment).

2.1. Arbeiten mit Git im Team

Git an ein zentrales Repository anbinden

Bisher haben Sie ein lokales Git-Repository verwendet. Das ist auch in Ordnung um grundlegende Befehle zu erlernen, aber für die Arbeit im Team muss man ein Repository verwenden, das für alle Teammitglieder verfügbar ist.

*Git nutzt verteilte Repositories, so dass **mehrere Personen** gemeinsam an einem Projekt arbeiten können. Man einigt sich zuerst darauf, dass es ein zentrales Repository gibt, aus dem sich die lokalen Repositories bedienen (der erste Zugriff auf ein entferntes Repository, bei dem man alles kopiert, heißt „clonen"). Anschließend werden Änderungen immer zuerst im lokalen Repository vorgenommen („commit") und dann an das zentrale Repository geschickt („push"). Um auch zwischenzeitlich gemachte Änderungen im zentralen Repository mitzubekommen, gibt es die Kommandos „fetch" bzw. „pull".*
Im Praktikum arbeiten wir mit einem Server des Fachbereichs Informatik, der die zentralen Repositories bereitstellt. Dieser Server ist unter https://code.fbi.h-da.de/ zu erreichen. Die verwendete Software ist gitlab, die Oberfläche ist ähnlich zu github.com, allerdings sind ihre Daten auf diesem Server in der Hochschule gespeichert und nicht durch andere einsehbar.
Nun sollen Sie einen Clone (d.h. eine lokale Kopie) ihres gemeinsamen Repositories auf ihrem lokalen Rechner erstellen. Diese Schritt können (und sollen) Sie auf jedem Computer durchführen an dem Sie mit dem Quellcode arbeiten möchten.

1. Loggen Sie sich unter https://code.fbi.h-da.de/ mit ihrem Hochschul-Account ein.
2. Sie sollten ein Projekt "Software Engineering Praktikum/21WS…/..." sehen. Öffnen Sie dieses Projekt und Sie sehen die Dateien und Ordner aus dem Legacy Projekt CocktailPro.
3. Sie finden die URL für das Clonen Ihres Projektes oben auf der Seite. Wählen Sie die HTTPS-Variante und kopieren Sie den Pfad:

4. Gehen Sie in Ihr „dev“ Verzeichnis[7] und clonen Sie das Git-Repository:
   `git clone «https://stxxx@code.fbi...» CocktailPro`
   «https://... » steht für den Pfad zum Git-Projekt; beachten Sie aber, dass Ihr User-Name (stxxxx) nach dem "//" mit einem angehängten @-Zeichen eingefügt werden muss. Das Projekt wird in einem lokalen Ordner CocktailPro erzeugt und es werden alle Dateien des Projektes aus dem Repo in das lokale Verzeichnis CocktailPro kopiert.

Änderungen vom lokalen Repository an das zentrale Repository übertragen

Ändern Sie lokal eine Datei in dem gerade erzeugten Projektverzeichnis und machen Sie einen commit. Finden Sie das Kommando heraus mit dem Sie Ihre Änderungen aus Ihrem lokalen Repository an das zentrale Repository übertragen und schreiben Sie es hier auf:

git push

Prüfen Sie in einem Browser auf code.fbi.h-da.de, ob die Änderungen tatsächlich angekommen sind.

Parallele Entwicklung im Team: "Branches"

Eine ganz wichtige Aufgabe von Versionskontrollsoftware ist die Verwaltung von unabhängigen Entwicklungs-Zweigen, den sogenannten Branches. Solange man in Git keine Branches anlegt, befindet man sich automatisch auf dem „master“ (=Haupt) Branch.
Ein neuer Branch kann einfach angelegt werden – und es gibt Möglichkeiten, Änderungen aus einem Branch in den „master“-Branch zu integrieren.

Hinweis: Sobald Sie mit dem zentralen Repository arbeiten, kann es zu unerwarteten Ergebnissen der Übung kommen, wenn Sie unterschiedlich schnell die Aufgaben erledigen. Deshalb sollten alle Teammitglieder ab sofort jede Teilaufgabe parallel abarbeiten und am Ende der Aufgabe aufeinander warten.

1. Legen Sie in ihrem lokalen Repository (Verzeichnis: CocktailPro) einen neuen Branch mit dem Namen „test_branch_<XY>“ an (wobei Sie <XY> durch Ihre Initialen ersetzen):
   `git branch test_branch_<XY>`
2. Mit dem Anlegen des Branches wird der gerade verwendete Branch nicht automatisch geändert! Prüfen Sie mit dem Befehl „`git status`“, auf welchem Branch Sie sich befinden.
3. Wechseln Sie nun zum neu angelegten Branch mit dem Befehl:
   `git checkout test_branch_<XY>`
4. Prüfen Sie erneut mit „`git status`“, ob der Wechsel auf den neuen Branch funktioniert hat.
5. Suchen Sie sich eine beliebige Quellcodedatei aus und fügen Sie einen Kommentar ein. Stellen Sie sicher, dass jedes Teammitglied eine andere Datei modifiziert und achten Sie darauf, dass Sie keinen Syntaxfehler einbauen.
6. Machen Sie einen commit um Ihre Änderung in das lokale Repository zu übernehmen
7. Wenn Sie Ihren Branch mit der Veränderung zum Server ("origin") schicken wollen, so müssen Sie Git mitteilen, dass es sich um einen neuen Branch handelt, und wie der Branch heißt:
   `git push -u origin test_branch_<XY>`
8. Sie erhalten eine Meldung, dass ein neuer Branch auf dem Server angelegt wurde.

[7] Der Ordnername kann auf Ihrem Privatrechner natürlich anders heißen.

9. Überprüfen Sie, ob Sie den neuen Branch auch auf der Web-Oberfläche unter https://code.fbi.h-da.de/ sehen können.
10. Wechseln Sie nun zurück zum „master“-Branch:
    `git checkout master`
11. Glauben Sie, dass die Quellcodedatei in ihrem Working-Directory noch Ihre Änderung enthält?

    ☒ ja oder ☐ nein

    Begründen Sie Ihre Vermutung:

    Der aktuellste Stand aus master wurde nicht gepullt.

Änderungen aus einem Branch in einen anderen Branch übernehmen: Merge

Jetzt lernen Sie, wie Sie die Veränderungen eines Branches in einen anderen Branch übernehmen. Dieser Vorgang wird bei git „merge“ genannt.

1. Ihr Branch heißt „test_branch_<XY>“ (wobei <XY> für Ihre Initialen steht).
   Stellen Sie sicher, dass das Sie sich im master-Branch befinden.
2. Sie möchten nun die Änderungen im Branch „test_branch_<XY>“ in den „master“-Branch übernehmen. Verwenden Sie dazu den Befehl:
   `git merge test_branch_<XY>`
3. Die Ausgabe zeigt an, dass es sich um einen sogenannten „Fast-forward“ Merge handelt; d.h. der Branch „master“ kann direkt um die Änderungen des Branches „test_branch_<XY>“ erweitert werden.
4. Glauben Sie, dass die Quellcodedatei in ihrem Working-Directory noch Ihre Änderung von oben enthält?

   ☒ ja oder ☐ nein

   Begründen Sie Ihre Vermutung:

   Die Änderung kann ohne Probleme mit merge übernommen werden.

5. Überprüfen Sie Ihre Vermutung mit einem Editor und mit „git log“
6. Führen Sie anschließend ein
   `git push`
   durch, damit der Merge auch auf dem Server gespeichert wird und für Ihre Teammitglieder sichtbar ist.

Änderungen (vor dem Push) synchronisieren - Pull

Wenn man mit mehreren Personen an einem Projekt arbeitet, dann kann es passieren, dass Änderungen im zentralen Repository vorgenommen wurden, die das lokale Repository noch nicht kennt. Spätestens beim "push" wird Git Ihnen deutlich machen, dass es Änderungen gibt, die Git nicht

so einfach zusammenführen kann. In diesem Fall müssen Sie Ihr lokales Repository aktualisieren und die Änderungen mit Ihrem lokalen Repository zusammenführen.
Diese Synchronisierung erfolgt mit dem Befehl „`git pull`“. (Intern werden dabei zunächst die Änderungen des Branches mit einem „git fetch“ in das lokale Git-Repository kopiert und dann ein „git merge“ der verschiedenen Versionsstände durchgeführt.)

Einen solchen Konflikt sollen Sie nun zusammen mit einem Teampartner verursachen und lösen.

1. Aktualisieren Sie Ihr lokales Repository auf den Stand des zentralen Repository mit:
   `git pull`
2. Wählen Sie einen Teampartner und eine Quellcodedatei aus, an der Sie beide in der gleichen Zeile unterschiedliche Änderungen an mehreren Stellen vornehmen (Der Merge-Algorithmus von Git soll keine Chance haben, die Änderungen automatisch zu vereinen).
3. Führen Sie beide ein Commit aus
4. Das „`git push`“ des ersten Teammitglieds sollte ohne Probleme funktionieren
5. Das „`git push`“ des zweiten Teammitglieds sollte einer Fehlermeldung liefern. Notieren Sie die Fehlermeldung:

```
! [rejected]
error: failed to push some refs to
'https://code.fbi.h-da.de/SE/21WS-Andelfinger/Di
```

6. Führen Sie das Kommando „`git pull`“ aus. Welche Meldungen werden ausgegeben?

```
Auto-merging src/main/Stampfer.cpp
CONFLICT (content): Merge conflict in src/main/Stampfer.cpp
Automatic merge failed; fix conflicts and then commit
the result.
```

7. Beheben Sie den Konflikt, in dem Sie die Datei mit dem Editor öffnen und die beiden Änderungen zu einer gemeinsamen Änderung zusammenführen.
8. Markieren Sie die Datei als bereinigt durch „`git add`“
9. Übertragen Sie die korrigierte Datei zuerst in Ihr lokales Repository (Commit) und dann in das zentrale Repository (Push).
10. Führen Sie Beide ein „git pull“ durch. Die Datei sollte jetzt in beiden Verzeichnissen gleich sein. Überprüfen Sie das.

2.2. Anbindung von CLion und Codeübergabe

Vergewissern Sie sich zunächst, dass Sie CLion gemäß den oben beschriebenen Schritten installiert haben und dass Sie CLion problemlos öffnen können. Empfohlen wird, ein ganz kleines Beispielprojekt zu erstellen und zu prüfen, dass es compiliert und ausgeführt werden kann.

Ab diesem Praktikum werden Sie den Code des CocktailPro schrittweise im Team bearbeiten, verbessern und erweitern. Dazu werden Sie jetzt die Anbindung von CLion an das zentrale Repository vornehmen. Danach können Sie immer direkt prüfen, ob Ihre Änderungen am Code (auf Ihrem Branch) erfolgreich sind. Sie müssen dazu lediglich Ihren (ergänzten) Code lokal compilieren und ausführen.

Um das CocktailPro-Projekt mit dem zugelieferten Code zu erstellen, wählen Sie in dem Start-Fenster von CLion die Option "Check Out from Version Control" und wählen "Git". Anschließend fragt CLion nach der URL Ihres Projekts. Kopieren Sie die URL aus Ihrem GitLab-Projekt. Wählen Sie noch einen Pfad in dem Ihre Kopie des Projekts abgelegt werden soll. Anschließend importiert CLion automatisch sämtliche Einstellungen.

Wichtig: Merken Sie sich bitte als Grundprinzip: Pushen Sie immer nur Code aufs zentrale Repository, wenn Sie ihn lokal fehlerfrei compilieren und ausführen können.

2.3. Nachbereitung: Kommentierung des Codes

Eine anerkannte Maßnahme zur Sicherung der Qualität ist das Kommentieren des Quellcodes. Im Rahmen Ihrer Einarbeitung in das Projekt sollen Sie den Code verstehen und ausführlich kommentieren. Nebenbei üben Sie dadurch den Umgang mit Git im Team.

2.3.1. Aufgabenaufteilung und Issues zur Kommentierung des Quellcodes

Als Nachbereitung sollen Sie den kompletten Quellcode kommentieren. Dazu sollten Sie in Ihrem Team festlegen, wer welche Klassen bearbeitet. Legen Sie für die Arbeitspakete in GitLab jeweils ein "Ticket" an[8] und weisen Sie einen Bearbeiter zu. Wenn Sie Ihre Aufgabe erledigt haben, markieren Sie das Ticket als "erledigt". Mit den Tickets wird sichtbar (und nachvollziehbar) wer welche Aufgabe zu erledigen hat. **Dieses Vorgehen setzen Sie bitte für das gesamte Praktikum ein.**

2.3.2. Kommentarrichtlinien

Lesen Sie die Google Kommentarrichtlinien und versuchen Sie, die Anregungen umzusetzen:
https://google.github.io/styleguide/cppguide.html#Comments

2.3.3. Kommentare für Doxygen

Die Kommentare für die Klassen, Methoden und Attribute sollen so erstellt werden, dass später automatisch eine ansprechende Dokumentation generiert werden kann.

- Kommentieren Sie alle Klassen gemäß der Konvention von Doxygen:
  www.doxygen.org/manual/docblocks.html (Verwenden Sie den JavaDoc style).
  Tipp: Die Datei "RecipeBook.h" beinhaltet die Kommentare in dem Format, das Doxygen fordert. Diese Datei können Sie als Vorlage verwenden:
  Alle Klassen werden in der h-Datei oberhalb der Klasse kommentiert in folgendem Format

```
/**
 * @class ExampleClass
```

[8] Falls Sie noch keine Tickets verwendet haben, müssen Sie diese eventuell noch in GitLab aktivieren unter Einstellungen->Allgemein->Issues. Anschließend müssen Sie runterscrollen und die Änderungen speichern!

```
 *
 * @brief Short explanation what the class is good for
 *
 * This is the long explanation.
 * This class is meant as ...
 * This explanation may be very long
 *
 */
class example...
```

- Alle Methoden werden mit Parametern und gegebenenfalls auch dem Rückgabewert in der h-Datei oberhalb der Methode kommentiert:

```
/** @brief A short description of the method
* @param [in] firstParam a boolean indicating that ...
* @return This int is ...
*
* This method does ...
* And returns ... results.
*/
int MyUsefulMethod(bool firstParam);
```

- Alle Attribute werden in der h-Datei oberhalb der Klasse kommentiert in folgendem Format:

```
/**
* my beloved variable.
* This is important because...
*/
int myVar;
```

2.3.4. Analysieren und kommentieren Sie den kompletten Quellcode!

- Kommentierung Sie auch die Methodenrümpfe! Erläutern Sie - unabhängig von Doxygen - schwer verständliche Codeteile in den cpp-Dateien durch Kommentare im Code!
  Es wird erwartet, dass Sie den Zweck und die Funktionsweise "Ihrer" Klassen im Praktikumstermin erläutern können. Jede cpp-Datei sollte eine angemessene Anzahl an (sinnvollen) Kommentarzeilen enthalten.

Hinweise zur Nachbereitung

Stellen Sie **bis zur Abgabefrist**[9] sicher, dass folgende Abnahmepunkte erfüllt sind:

- Der Code ist vollständig und sinnvoll gemäß obiger Regeln kommentiert.
- Alle Teammitglieder haben in einem eigenen Branch einen Teil des Codes kommentiert, den Branch auf den Git-Server gepushed, die Änderungen in den Master gemerged und auf dem zentralen Git-Server eingecheckt.
- Prüfen Sie über das Webfrontend des Git_Servers (https://code.fbi.h-da.de), dass der kommentierte Code von allen Teammitgliedern auch tatsächlich im Master-Branch des zentralen Repositories vorliegt.

Achtung! Der Inhalt des Master-Branches Ihres Team-Repositories im Git zum Abgabetermin ist die Grundlage für die Bewertung Ihres Meilensteins. Sie erhalten Feedback zur Qualität Ihrer Abgabe für diesen Stand. Verspätete Abgaben werden nicht mehr berücksichtigt!

[9] Die Abgabe findet 48h vor dem nächsten Praktikumstermin statt. Ihr Betreuer legt den genauen Zeitpunkt fest.