Workshop: Git und Social Coding (Blatt 01)

Bearbeiten Sie diese Aufgaben in Gruppen von ca. 2 Personen (abhängig davon, wieviele PCs verfügbar sind). Sie haben 45 Minuten für die Bearbeitung.

Es ist nicht weiter schlimm, wenn Sie mit den Übungen in der Zeit nicht fertig werden. Das Wichtigste ist, dass Sie ein grundlegendes Verständnis entwickeln, auf dessen Basis Sie dann nochmal in Ruhe zu Hause aufbauen können.

Viel Erfolg!

Lernziele
Nach der Bearbeitung dieses Arbeitsblattes haben Sie gelernt

- ✓ grundlegende Git-Befehle anzuwenden
- ✓ nachzuvollziehen, welchen Einfluss grundlegende Befehle auf den Versionierungsgraphen haben
- ✓ parallel an mehreren Versionen zu arbeiten und diese wieder zusammenzufügen

Aufgabe 1: Umgebung einrichten (ca. 5 min)

Sie haben eine neue Idee für ein Programm und wollen direkt anfangen zu programmieren. Ihnen fällt ein, dass es sich lohnt auch für private Projekte Git zu verwenden. Deshalb installieren Sie Git und richten es sich ein, bevor Sie mit der Arbeit anfangen:

1. Überprüfen Sie in der Kommandozeile mit `git --version` ob Git vorhanden ist.
2. Konfigurieren Sie Git und setzen Sie den Namen und die E-Mail Adresse unter der Sie künftig arbeiten werden:

```
$ git config --global user.name "Peter Lustig"
$ git config --global user.name "peter.lustig@uni-tuebingen.de"
$ git config --global color.ui auto
```

3. Erstellen sie eine leeren Ordner für diese Übung (z.B. `exercise01/`) und wechseln sie mit der Kommandozeile in diesen Ordner
4. Rufen Sie `git init` auf, um den Ordner in ein Git-Repository um zu wandeln.

Jetzt kann es endlich mit der Entwicklung losgehen.

Wichtige git-Befehle für diese Übung

```
$ git --version
```

Zeigt die installierte Version von Git an.

```
$ git init
```

Erstellt ein lokales Git-Repository im aktuellen Verzeichnis

Aufgabe 2: Änderungen machen und merken (ca. 10min)

Morgens bei einer Tasse Kaffee fangen Sie nun an den ersten Beitrag zu Ihrem Projekt zu leisten.

In dieser Aufgabe sollen Sie eine Datei anlegen, diese in die Versionierung aufnehmen und den Versionierungsgraphen zeichnen.

1. Erstellen Sie eine einfache Java-Datei in Ihrem Arbeitsverzeichnis, z.B. die Datei `hello.java` mit dem Inhalt:

```java
class Hello {
  public static void main(String[] args) {
    System.out.println("Hello Git");
  }
}
```

Wichtige git-Befehle für diese Übung

`$ git status`

Index und geänderte Dateien anzeigen.

`$ git commit`

Speichert den aktuellen Index als Snapshot in der Versionshistorie

`$ git log`

Zeigt die Versionshistorie für den aktuellen Branch

2. Überprüfen Sie den aktuellen Status Ihrer Arbeitskopie mit `git status`.
3. Fügen Sie die eben erstellte Java-Datei mit `git add` zum Index hinzu und überprüfen Sie erneut den Status Ihrer Arbeitskopie.
4. Erstellen Sie einen Snapshot, basierend auf dem aktuellen Index (`git commit`) und überprüfen Sie erneut den Status Ihrer Arbeitskopie.
5. Zeichnen Sie den Versionierungsgraph, der den aktuellen Zustand des Respositories darstellt. Annotieren Sie jeden Commit in dem Graphen mit vier Stellen des Commit-Hashs.

   Tipp: Nutzen Sie `git log --oneline --decorate`, um eine übersichtliche Darstellung der Versionshistorie zu erhalten.

Aufgabe 3: Vorübergehende Änderungen und paralleles Arbeiten (10min)

Nachdem Sie noch ein bisschen weiterarbeiten, kommt Ihnen plötzlich eine Idee. Sie sind sich aber nicht sicher, ob diese Idee wirklich funktioniert. Deshalb arbeiten Sie an der Idee in einem neuen Branch.

1. Machen Sie eine Änderung an `hello.java`, überprüfen Sie den Status, fügen Sie die Änderung zum Index hinzu und erstellen Sie einen Commit.

   Tipp: Achten Sie auch immer darauf, dass die Commit-Nachricht Ihre Änderung kurz und treffend beschreibt (Tipps zu Commits `(https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project)`). Nur so können Sie diese später wieder schnell nachvollziehen.

Wichtige git-Befehle für diese Übung

`$ git branch [new branch name]`

Erstellt einen neuen Branch durch Kopieren des `HEAD`-Pointers.

`$ git checkout [branch or commit-id]`

Setzt `HEAD` auf den angegebenen Branch und kopiert die Inhalte aus dem Snapshot in das Arbeitsverzeichnis.

2. Erstellen Sie nun einen Branch, z.B. mit `git branch my-experiment` und wechseln Sie zu diesem Branch, um an Ihrer Idee zu arbeiten.

3. Erstellen Sie eine neue Datei mit etwas Inhalt, z.B. `greeter.java` mit

```java
class Greeter {
  String name;
  Greeter(String name) { this.name = name; }
  public String greet() {
    return "Hello" + name + ", what a wonderful day.";
  }
}
```

4. Sie sind zufrieden mit Ihrer Idee? Dann erstellen Sie nun einen Commit und speichern den Zustand der neuen Datei in ihrem Repository (im Branch `my-experiment`).
5. Zeichnen Sie erneut den Versionierungsgraph, der den aktuellen Zustand des Respositories darstellt.

Aufgabe 4: Änderungen übernehmen (ca. 10min)

Sie sind zunächst zufrieden mit den Änderungen im Branch `my-experiment`. Nun wollen Sie diese Änderungen auch in den `master`-Branch übernehmen.

> **Wichtige git-Befehle für diese Übung**
>
> `$ git merge [branch name]`
>
> Erzeugt einen Merge-Commit mit zwei "parent"-Referenzen (HEAD und angegebener Ziel-Branch). Der aktuelle Branch wird auf den erzeugten Merge-Commit gesetzt.

1. Nutzen Sie `git checkout`, um in den `master`-Branch zu wechseln. Überprüfen Sie mit `git status`, ob sie im richtigen Branch sind.
2. Untersuchen Sie Ihr Arbeitsverzeichnis. In welchem Zustand befindet sich dies nun?
3. Übernehmen Sie nun die Änderungen aus `my-experiment` in `master` durch Aufrufen von `git merge my-experiment`.
4. Untersuchen Sie das Ergebnis von `merge` in Ihrem Arbeitsverzeichnis.
5. Zeichnen Sie erneut den Versionierungsgraph, der den aktuellen Zustand des Respositories darstellt.

Aufgabe 5: Änderungen zusammenführen (ca. 10min)

Zufrieden mit der aktuellen Version in `master` arbeiten Sie weiter und erzeugen einen weiteren Commit in `master`. Nun fällt Ihnen auf, was Sie in `my-experiment` noch besser machen könnten - Sie sind sich aber wieder nicht sicher, ob die Änderung auch in `master` sollte.

Da sie weiter auf `my-experiment` arbeiten wollen, aber auch die neusten Änderungen von `master` brauchen mergen Sie nun `master` in `my-experiment`.

1. Führen Sie eine Änderung in `master` durch und committen Sie diese.
2. Wechseln Sie zu `my-experiment` und mergen Sie `master` in `my-experiment`, um an der aktuellen Version weiterzuarbeiten.
3. Zeichnen Sie erneut den Versionierungsgraph, der den aktuellen Zustand des Respositories darstellt.

Optional: Machen Sie noch eine Änderung in `my-experiment` und mergen Sie nun diese Änderungen zurück in `master`. Zeichnen Sie den Versionierungsgraph.

Aufgabe 6: Hausaufgaben

1. Finden Sie heraus was eine `.gitignore` Datei macht. Kompilieren Sie hierzu die Java-Dateien aus dem obigen Projekt in einen Ordner `build/`. Warum sollten Sie die dadurch entstandenen `*.class`-Dateien nicht in Ihrem Repository speichern?
2. Finden Sie heraus was `git rebase` macht und wofür Sie es einsetzen könnten. In welchem Fall hätte es Sinn ergeben zu rebasen, statt zu mergen?