AUFGABE 1: QUELLCODEVERWALTUNG MIT GIT

Ziel dieser Aufgabe ist es, ein Gefühl für die dezentrale Versionsverwaltung mit `git` zu bekommen. In dieser Aufgabe erhalten Sie von uns ein bereits mit einer Versionshistorie befülltes `git`-Repository. An dieser Versionsgeschichte werden Sie eine Reihe von Änderungen vornehmen.

Nach Bearbeitung dieser Übungsaufgabe sollten Sie verstanden haben wie man Inhalte zu einem `git`-Repository hinzufügt und entfernt, wie man mit einem entfernten Repository kommuniziert, wie man Geschichte umschreibt, und vor allem was man machen muss, wenn mal etwas schiefgeht.

Halten Sie für die Abgabe fest, welche Befehle Sie in den einzelnen Teilaufgaben verwendet haben, sowie den Zustand Ihres Repositorys nach jeder Teilaufgabe (ab Teilaufgabe 4). Als Hilfestellung stellen wir Ihnen hierfür begleitend die Funktion `vezslog` bereit. Um diese verwenden zu können, führen Sie einmal **`source`** `vezslog.sh` aus (Dies ist erforderlich wannimmer Sie ein neues Terminalfenster öffnen). Zum Erstellen des Logs rufen Sie dann `vezslog` nach jeder Teilaufgabe auf:

```
vezslog <aufgabennummer>
```

Stellen Sie diese Datei mit Ihren Aufzeichnungen *am Ende*, d.h. unmittelbar vor Abgabe, unter Versionskontrolle, sodass sie Teil Ihrer Abgabe ist.

Hinweis: Stellen Sie vor der Abgabe **unbedingt** sicher, dass Ihre Logdatei **keine** sensiblen Informationen aus Ihrer Shell-History enthält!

Hinweis: Sie werden in dieser Aufgabe den Zustand des Repositories in einer Weise zurücksetzen, der lokale Änderungne verwirft. Dies würde natürlich auch Änderungen an Ihrer `answers.md` verwerfen. Kopieren Sie die Datei also zur Beantwortung der Fragen an eine Stelle außerhalb des Repositorys und kopieren Sie sie erst am Ende zurück und checken Sie ein, um einen unbeabsichtigten Verlust Ihrer Antworten zu vermeiden. Ähnliches gilt auch für die `abgabedatei`, welche von `vezslog` erzeugt wird. Stellen Sie diese aus diesem Grund erst unmittelbar vor Abgabe unter Versionskontrolle.

Hinweis: Sie müssen den Fork in Aufgabe 2 nur einmal pro Gruppe ausführen und können anschließend allen Gruppenmitgliedern entsprechend Zugriff auf das so erstellte Arbeitsrepository geben. Sie können mit der Bearbeitung des Blattes auch beginnen, wenn Sie noch keine Gruppe gefunden haben, dies ist erst zur Abgabe (Aufgabe 18) nötig.

1 Aufgabenstellung

Vermerken Sie Ihre Antworten zu den Fragen der einzelnen Aufgaben an den vorgesehenen Stellen in der vorgegebenen `answers.md`.

Aufgabe 1 *SSH-Schlüssel*

Da das Vorgabenrepository nur nach Anmeldung zugänglich ist, müssen Sie sich gegenüber Gitlab, beispielsweise mittels eines SSH-Schlüssels, authentifizieren. Erzeugen Sie dazu ein Schlüsselpaar und hinterlegen Sie den öffentlichen Schlüssel in Ihrem Gitlab-Account.

☞ `ssh-keygen`
☞ `.ssh/config`

Aufgabe 2 *Klonen*

Forken Sie, wie in den Übungsfolien beschrieben, das persönliche Abgaberepository Ihrer Gruppe. Laden Sie anschließend Ihren Fork mittels `git clone` herunter. Nutzen Sie zur Bearbeitung der Aufgabe einen eigenen Branch „`aufgabe1`“.

☞ `git checkout`

Aufgabe 3 *Aufsetzen des Gruppenrepositories*

Legen Sie, wie in der Tafelübung besprochen, ein Gruppenrepository an. Bitte denken Sie daran, die Betreuenden als Entwickler hinzuzufügen (Suche nach den Namen im Gitos). Passen Sie die Konfigurationsdatei des Repositories so an, dass das Remote `origin` auf Ihr Gruppenrepository und das Remote `vorgabe` auf unser Vorgaberepository verweist (siehe Übungsfolien).

☞ `git remote rename`
☞ `git remote add`
☞ `.git/config`

Aufgabe 4 *Log*

Verschaffen Sie sich nun mittels `git log` einen Überblick über die Versionsgeschichte. Wenn Sie wollen, können Sie hierfür auch ein graphisches Werkzeug wie `git cola` oder `gitk` verwenden.

Aufgabe 5 *Hinzufügen*

Fügen Sie Ihren Namen und Ihre E-Mail-Adresse zur Datei `README` zum Abschnitt „Credits“ hinzu. Legen Sie anschließend einen Commit mit dieser Änderung an. *Wie sieht die Versionsgeschichte nun aus?* Dokumentieren Sie sie in Ihrer Abgabedatei.

☞ `git add`
☞ `git commit`

Aufgabe 6 *Rückgängig machen*

Verwenden Sie nun `git reset --hard HEAD^` um den letzten Commit rückgängig zu machen. Betrachten Sie die Versionsgeschichte. *Worin liegt der Unterschied zwischen dem Ausgangsrepository und dem aktuellen Zustand?*

Antwort:

Aufgabe 7 *Benennung*

Was beschreibt der Platzhalter `HEAD^`*?*

Antwort:

Aufgabe 8 *git reset (I)*

Lesen Sie die man-Page von `git reset`.
Was haben Sie soeben rückgesetzt*?*

☞ `man git-reset`

Antwort:

Aufgabe 9 *git reset (II)*

Wie verändert der Kommandozeilenparameter `--hard` *das Ergebnis des Befehls?*

Antwort:

Aufgabe 10 *Zeitreise (I)*

Betrachten Sie nun die Ausgabe von `git reflog`.
Was sehen Sie hier?

Antwort:

Aufgabe 11 *Zeitreise (II)*

Suchen Sie nach dem Zustand, in dem sich das Repository am Ende der Teilaufgabe 5 „Hinzufügen“ befand, und stellen Sie diesen mit Hilfe von `git reset --hard` wieder her!

Antwort:

Aufgabe 12 *Zusammenarbeit*

Ändern Sie in der Datei `modern.astimerc` den Wert des „handThickness“-Eintrags von 70 auf 50 und legen Sie einen Commit für diese Änderung an.

Ziehen Sie nun mit Hilfe von `git pull` die Commits aus dem Repository

```
git@gitos.rrze.fau.de:i4/teaching/vezs/vezs-vorgabe-dev.git
```

mit dem Argument `--no-rebase` nach.

Beheben Sie anschließend den dabei entstehenden Konflikt. Sie können `git status` verwenden, um in Erfahrung zu bringen, welche Dateien von dem Konflikt betroffen sind. Achten Sie dabei auf den **Erhalt Ihrer Änderung**. Betrachten Sie nun die Historie – *Wie sieht diese aus?*

☞ `git mergetool`

Antwort:

Aufgabe 13 *git rebase*

Verwenden Sie jetzt `git rebase` um den in Aufgabe 12 entstandenen Merge-Commit zu beseitigen und die beiden, in Konflikt stehenden Commits zu erhalten. Achten Sie wiederum darauf, dass am Ende Ihre Änderungen erhalten bleiben. *In welcher Reihenfolge sollten Sie sinnvollerweise die Entwicklungslinien aufeinander aufbauen, was bedeutet dies für die Wahl der Basis Ihres Rebase? Wie sieht die Historie jetzt im Vergleich zu vorher aus?*

Antwort:

Aufgabe 14 *Geschichte neu schreiben*

Führen Sie einen interaktiven Rebase durch, in dem Sie die beiden Commits, die den Konflikt betreffen, miteinander verschmelzen. *Welchen Commit müssen Sie als Basis für* `git rebase -i` *angeben?*

☞ `git rebase -i`

Antwort:

Aufgabe 15 *Umsortieren und verschmelzen*

Führen Sie nun einen interaktiven Rebase auf dem Commit vom Montag, den 27. Mai 2013 um 18:45:34 Uhr (+0200) durch und gruppieren Sie diejenigen Commits, die Warnungen des Übersetzers betreffen. Sie erkennen die entsprechenden Commits an dem Wort „warning“ in der Commitnachricht. Verschmelzen Sie diese Commits anschließend miteinander, so dass nur noch ein Commit übrig bleibt, der Übersetzerwarnungen in Ordnung bringt.

Notieren Sie sich jetzt die Ausgabe von `git reflog` *für die Abgabe.*

Aufgabe 16 *Hochladen*

Laden Sie nun Ihre Versionsgeschichte als Branch `aufgabe1` in Ihr Repository hoch.

Aufgabe 17 *Abschließende Frage*

Angenommen die beiden Repositories, aus denen Sie gezogen haben, wären öffentlich zugänglich – wieso wären die Rebase-Schritte, die Sie durchgeführt haben, dann eine schlechte Idee?

Antwort:

Aufgabe 18 *Abgabe*

Reichen Sie nun Ihre fertige Lösung im gitlab als Mergerequest gegen **Ihr persönliches Abgaberepository (`gitos.rrze.fau.de/i4/teaching/vezs/ws24/<id>`)** bis zum Abgabedatum (25.10.2024) ein. Fügen Sie hier auch Ihre mittels `vezslog` gepflegte `abgabedatei` hinzu, nachdem Sie überprüft haben, dass diese keine Passwörter enthält. Die eigentliche Abgabe erfolgt dann im Rahmen eines Abnahmegesprächs. Zum Gespräch müssen alle Gruppenmitglieder anwesend sein. Falls wir noch Nachbesserung einfordern, reichen Sie diese einfach durch Aktualisierung Ihres Mergerequests nach.

Hinweise

- Bearbeitung: Gruppenarbeit
- Abgabefrist: 25.10.2024
- Fragen bitte an `i4ezs@lists.cs.fau.de`