Projektmanagement Übung

Konfigurationsmanagement mit Git in Eclipse (EGit)

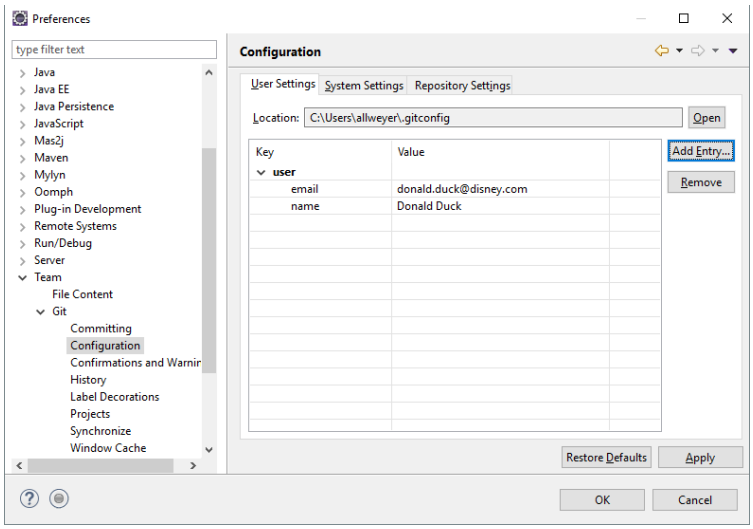
1. Konfiguration von EGit

Tragen Sie Ihren Namen und Ihre E-Mail-Adresse in die EGit-Konfiguration ein. Zu finden unter:

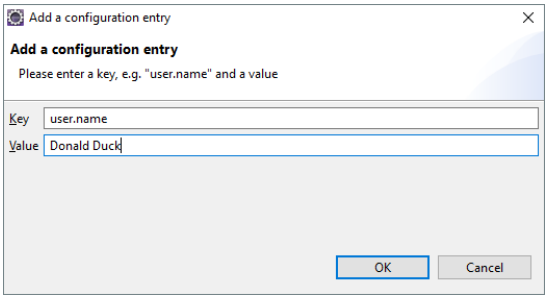
Window→Preferences→Team→Git→Configuration

oder (in neueren Eclipse-Versionen) unter

Window→Preferences→Version Control (Team)→Git→Configuration

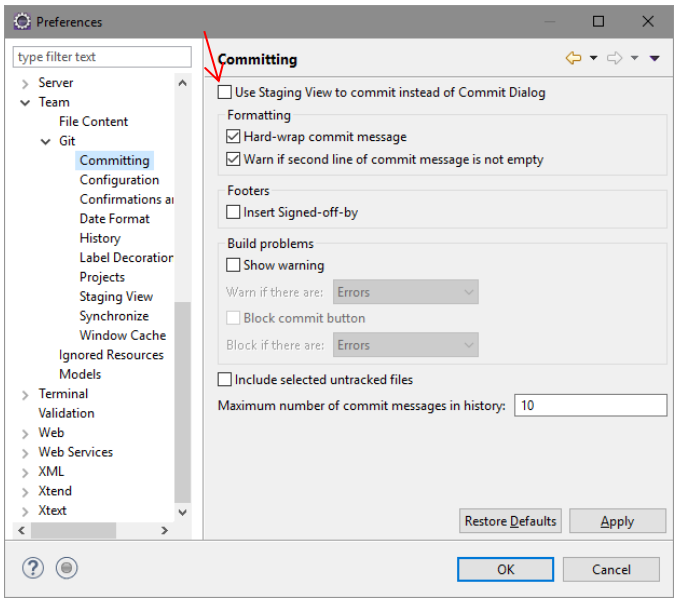


Die Einträge werden mit „*Add Entry*“ erzeugt:



Analog wird der Eintrag für „user.email“ erzeugt.

Entfernen Sie unter *Window→Preferences→Team→Git→Commiting* bzw. *Window→Preferences→Version Control (Team)→Git→Committing* das Häkchen bei dem obersten Eintrag „Use Staging View to commit instead of Commit Dialog“, falls es gesetzt ist.



2. Lokales Repository anlegen

Ein lokales Repository hilft noch nicht bei der Zusammenarbeit mit anderen, aber man kann z. B. Änderungen im Code rückgängig machen und Branches erzeugen, um verschiedene Ideen auszuprobieren ohne den Hauptentwicklungszweig zu beeinflussen (oder auch um verschiedene Varianten der Software zu erstellen), und diese später wieder zusammenführen.

Hierzu wird zunächst wie gewöhnlich ein Java-Projekt in Eclipse angelegt. Sie können als Grundlage das in der Datei „ZooApp.zip“ enthaltene Projekt importieren.

Legen Sie nun ein lokales Git-Repository an. Selektieren Sie das Projekt und wählen Sie im Kontextmenü *Team→Share Project*. Falls Sie nach einem Repository Type gefragt werden, wählen Sie „Git“.

Wählen Sie im nächsten Dialog („Configure Git Repository“) den Button „*Create*“ und geben Sie einen Pfad mit einem Verzeichnisnamen für das zu erzeugende Repository an (in der Regel legt man das Repository außerhalb des Eclipse Workspace an). Da das Repository nur ein Projekt enthalten soll, können Sie das Verzeichnis so nennen wie das Projekt (z. B. C:\users*username*\git*projektname*).

Im Workspace sieht man nun, dass die Symbole der Java-Klassen etc. mit kleinen Fragezeichen versehen sind.

Öffnen Sie die folgenden Views:

- *Window→Show View→Other→Git→Git Repositories*
- *Window→Show View→Other→Git→Git Staging*

In der Repository View sehen Sie das neu angelegte Repository. Die Verzeichnisse unter „Branches“ sind noch leer, d. h. es sind noch keine Dateien ins Repository commited worden. Das Working Directory ist das Arbeitsverzeichnis des Java-Projekts.

Die Staging View zeigt die Staging Area oder den „Index“, d. h. alle Änderungen, die für einen gemeinsamen Commit gesammelt wurden. Wenn Sie das Projekt selektieren, sehen Sie sämtliche Dateien des Projektes unter „Unstaged Changes“, d. h. sie wurden geändert (bzw. in diesem Fall alle neu angelegt), aber noch nicht für einen Commit ins Repository vorgesehen.

Die vom Compiler erzeugten .class-Dateien sollten hier nicht mit angezeigt werden. Da diese automatisch aus dem Quellcode erzeugt werden, brauchen sie nicht in das Repository aufgenommen zu werden.

Manchmal passiert es allerdings, dass sie doch mit angezeigt werden. Legen Sie in diesem Fall in dem Projekt eine Datei namens '.gitignore' an (Projekt selektieren, Kontextmenu→ New File), tragen Sie darin in die erste Zeile '/bin/' ein und speichern Sie sie. Git ignoriert nun alles, was sich im Ordner 'bin' befindet (die .gitignore-Datei selbst wird in der Java-Perspektive nicht angezeigt – um sie zu sehen muss man in den über das Symbol erreichbaren Filtern das Häkchen bei .* resources entfernen).

Beachten Sie bei allen weiteren Schritten, wie sich die Markierungen der Symbole im Project Explorer ändern. Sie zeigen die verschiedenen Zustände der einzelnen Dateien bzgl. des Repository an.

3. Projektinhalte in das Repository übernehmen

Wählen Sie im Kontextmenü des Projekts *Team→Add to index* oder klicken Sie auf das Symbol mit den zwei Plus-Zeichen in der Staging View. Hierdurch werden alle geänderten bzw. neuen Dateien in die Staging Area übernommen.

Um die Änderungen aus der Staging Area in das Repository zu übernehmen, muss man in der Staging View eine „Commit Message“ eintragen (z. B. „Projekt neu angelegt“) und „*Commit*“ drücken.

Hierdurch verschwinden die „Staged Changes“ aus der Staging View. In der Repository View sehen Sie in Ihrem Repository unter Branches / Local einen Eintrag für den zuletzt durchgeführten Commit.

Wozu legt man die Änderungen zuerst in die Staging Area um sie erst in einem zweiten Schritt in das Repository zu committen? Hierdurch kann man genauer kontrollieren, was in den einzelnen Commits (d. h. Schnappschüssen des Projektes) enthalten sein soll. Z. B. kann man, wenn man eine Zeitlang gearbeitet und mehrere Änderungen gemacht hat, nur einen Teil der Änderungen in einem Commit zusammenfassen und andere Änderungen, die vielleicht einen anderen Aspekt betreffen, in einem zweiten Commit zusammenfassen.

Häufig benötigt man dieses in Git vorgesehene zweistufige Verfahren aber nicht, weil man einfach den aktuellen Stand als Schnappschuss ins Repository aufnehmen möchte. Daher bietet EGit die Möglichkeit, beide Aktionen (Staging und Commit) in einen einzelnen Befehl zusammenzufassen.

Fügen Sie eine weitere Klasse hinzu und zu einer bestehenden Klasse eine neue Methode. Wählen Sie – ohne die gewünschten Änderungen erst ins Staging Area aufzunehmen – im Kontextmenü des Projekts *Team→Commit*.

Es geht ein separater Commit-Dialog auf. Neu hinzugefügte Dateien müssen u. U. in der unteren Hälfte dieses Dialogs erst markiert werden, damit sie in das Repository mit aufgenommen werden. Sie müssen wiederum einen Kommentar angeben (z. B. „Klasse A neu angelegt“) und mit „*Commit*“ beenden.
Falls der Commit-Dialog nicht aufgeht, ist möglicherweise in den Preferences noch das Häkchen bei „Use Staging View to commit instead of Commit Dialog“ gesetzt (s.o. bei Schritt 1).

Führen Sie weitere Änderungen in verschiedenen Klassen durch und committen Sie Ihren Code zwei bis drei Mal.

4. Änderungen einzelner Dateien rückgängig machen

Führen Sie einige Änderungen in einer der Klassen durch. Fügen Sie z. B. zwei Methoden hinzu und löschen Sie eine andere. Führen Sie zunächst keinen Commit durch.

Wählen Sie im Kontextmenü des Projekts *Compare With→HEAD Revision*. Beantworten Sie ggf. die Frage ob Sie in die Synchronize View wechseln wollen, mit *yes*. Selektieren Sie links im Tab „Synchronize“ eine der geänderten Klassen.

Es wird ein Vergleich der geänderten Klasse im Workspace (links) mit dem letzten Stand der Klasse im Repository (rechts) angezeigt. Falls der Vergleich nicht angezeigt wird: Über das Kontextmenü auf der selektierten Klasse „Open in Compare Editor“ auswählen.

Rechts über dem Vergleichseditor finden sich verschiedene Icons zum Auswählen einzelner Änderungen sowie zum Zusammenführen der Änderungen. Man kann aber auch nur für einzelne Änderungen den alten Stand wiederherstellen. Wenn man eine bestimmte Änderung rückgängig machen will, selektiert man sie und wählt „*Copy Current Change from Right to Left*“. Nachdem die links dargestellte Datei auf dem gewünschten Stand ist, speichern und den Vergleichseditor schließen.

Zurück in die Java-Perspektive wechseln und das Projekt erneut committen.

5. Änderungsverlauf ansehen

Führen sie nun mehrere Änderungen durch. Z. B. könnten Sie eine weitere Klasse für Futter anlegen oder Methoden zur Auflistung aller Tiere einer Tierart oder aller Tiere eines Geheges.

Wählen Sie im Kontextmenü des Projekts *Team→Show in History*. Es wird der Verlauf der verschiedenen Commits angezeigt. Durch Auswahl eines Commits kann man sich ansehen, welche Dateien geändert wurden. Man kann den damaligen Stand dieser Dateien auch (nur lesend) öffnen und z. B. zwischenzeitlich gelöschte Inhalte, die man doch noch verwenden möchte, herauskopieren und in die aktuelle Version übernehmen.

Es ist auch möglich, die aktuelle Version des Projektes oder einer einzelnen Datei komplett durch einen früheren Stand zu ersetzen. Dies wird aber eher selten gemacht, da man zumeist nicht komplett auf einen alten Stand zurück will, sondern nur manche Änderungen in

der aktuellen Version auf den alten Stand zurücksetzen will. Hierfür checkt man den früheren Stand als eigenen Branch aus und merged diesen mit dem aktuellen HEAD (siehe unten).

6. Branch erstellen

Wählen Sie im Kontextmenü des Projekts *Team→Switch to…→New Branch.* Geben Sie einen Namen für den Branch ein (z. B. „myFirstBranch“). Ihr Projekt im Workspace befindet sich nun in dem neu angelegten Branch.

Arbeiten Sie in dem Branch weiter, erstellen Sie mindestens zwei Commits. Erstellen Sie u. a. in einer ausgewählten Klasse eine neue Methode.

7. Branch wechseln

Wenn Sie in einem anderen Branch arbeiten wollen, müssen Sie diesen Auschecken. Einziger anderer Branch ist bislang der Hauptzweig „master“. Selektieren Sie ihn in der Repository View (unter *Branches / Local*) und wählen Sie im Kontext-Menü „Checkout“. Dadurch arbeiten Sie mit dem Projekt im Workspace nun wieder im master-Branch . Der Name des jeweiligen Branches wird hinter dem Projektnamen in eckigen Klammern angezeigt. Arbeiten Sie nun wieder im Master weiter und erstellen Sie mindestens zwei Commits. Führen Sie auch Änderungen in Klassen durch, die Sie im Branch ebenfalls geändert haben. Erstellen Sie u. a. in der oben gewählten Klasse eine andere neue Methode als im Branch.

8. Branch mit einem anderen zusammenführen (mergen)

Bleiben Sie mit Ihrem Projekt im master-Branch. Wählen Sie im Kontextmenü des Projekts *Team→Merge…* und selektieren Sie anschließend den oben angelegten Branch. Da in beiden Branches verschiedene Änderungen – z. T. in denselben Klassen – durchgeführt wurden, wird ein Konflikt angezeigt, der erst gelöst werden muss, bevor eine erfolgreiche Zusammenführung erfolgen kann.

Schließen Sie den Dialog, der auf den Konflikt hinweist. Die Konflikte werden nun in den Dateien angezeigt, die sich unterscheiden. EGit fügt zusätzliche Zeilen mit Anmerkungen in die Datei ein um zu markieren, welche Teile aus welchem Branch stammen. Diese zusätzlichen Zeilen muss man nach dem Anpassen der Datei wieder löschen.

Alternativ kann man das Merge-Tool verwenden, in dem die beiden Datei-Versionen mit ihren Unterschieden nebeneinander dargestellt werden (*Team→MergeTool*).

Lösen Sie den Konflikt manuell, z. B. indem Sie beide Methoden in der Klasse belassen. Entfernen Sie ggf. die von EGit eingefügten Zeilen (bei Benutzung des Merge-Tools nicht notwendig) und speichern Sie die betreffende Klasse.

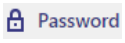
Fügen Sie die nun korrigierten Dateien mittels *Team→Add to index* in das Staging Area ein (ein direkter Commit ist in diesem Fall nicht möglich). Führen Sie anschließend einen Commit durch. Sehen Sie sich die Historie des Masters an, wo die Verzweigung und Zusammenführung grafisch dargestellt werden.

9. Zusammenarbeit: Ein Team-Repository verwenden.

Für diese Übung braucht das gesamte Team Zugriff auf einen Git-Server, der z. B. auf einem anderen Rechner im Netzwerk installiert sein kann, oder von einem Cloud-Anbieter gehostet werden kann. Im Folgenden wird die Arbeit mit dem Gitlab-Server der rheinland-pfälzischen Hochschulen beschrieben.

Öffnen Sie einen Browser, gehen Sie auf die Seite https://gitlab.rlp.net/ und loggen sich über den Button login.rlp.net mit Ihrem Hochschulaccount ein.

Da git keinen Login über den Shibboleth-Server der Hochschule vornehmen kann, muss in Gitlab noch einmal ein separates Passwort vergeben werden, das Sie in git für den Zugriff auf den Gitlab-Server nutzen können.

Hierzu oben ganz rechts unter dem Benutzer-Icon  „Preferences“ auswählen und im Menü links unter 

ein Passwort eintragen.

Ein Mitglied Ihres Teams richtet im GitLab-Portal ein neues Repository ein (*„New Project→Create blank project“*)

- Geben Sie einen Namen ein, belassen Sie „Visibility Level“ auf „Private“ und klicken Sie auf „*Create project“*.
- Klicken Sie auf „Clone“ und kopieren Sie die URL unter „Clone with HTTPS“ in die Zwischenablage. Sie hat die Form https://gitlab.rlp.net/*username*/*projektname*.git.
- Wechseln Sie zu Eclipse. Öffnen Sie das in Punkt 2 angelegte Projekt (zu dem Sie auch in Eclipse das lokale Repository angelegt haben).
- Bennen Sie das Projekt um! Da die anderen Teammitglieder schon ein Projekt mit dem Namen „ZooProjekt“ haben, würde es einen Konflikt geben, wenn das gemeinsame Projekt genauso heißen würde.
- Wählen Sie im Kontextmenü des Projektes „*Team→Push branch master*“ (Falls Sie nicht im master-Branch sind, können Sie diesen vorher auschecken, oder Sie verwenden den aktuellen Branch).
- Im folgenden Dialog muss nur noch Ihr oben vergebenes GitLab-Passwort eingegeben werden. Wenn Sie „Store in Secure Store“ markieren, müssen Sie das Passwort bei künftigen Aktionen mit dem GitLab-Server nicht mehr eingeben.
- Bei den folgenden Dialogen können Sie jeweils einfach weiterklicken, und im letzten Dialog auf „Push“ klicken.
- Im GitLab-Portal müssten nun die Inhalte des Projektes in dem Repository vorhanden sein. Wechseln Sie hierzu in den Branch „master“.

- Laden Sie nun Ihre Team-Kollegen in das Repository ein. Hierzu im linken Menü unter „*Project information*“ den Punkt „*Members*“ wählen, anschließend „*Invite members*“.
- Fügen Sie unter „*Username or email address*“ Ihre Team-Kollegen hinzu. Wenn sich diese bereits in GitLab angemeldet hatten, lassen sie sich über die Eingabe der E-Mail-Adresse finden und auswählen.
- Wählen Sie unter „*Select a role*“ „Developer“ oder „Maintainer“ aus. Beide können Code in das Repository pushen, Maintainer können z. B. auch weitere Mitglieder hinzufügen etc.
- Die Team-Kollegen werden per E-Mail über die Einladung informiert und können nun auch auf das GitLab-Repository zugreifen.

Die eingeladenen Team-Mitglieder klonen nun das Projekt.

- Schließen Sie in Eclipse die Editor-Fenster der Klassen, an denen Sie bisher gearbeitet haben, da Sie anschließend an dem gemeinsamen Projekt, das Ihr Kollege bereitgestellt hat, weiterarbeiten sollen.
- Folgen Sie dem Link aus der Mail und öffnen Sie das Repository im Browser.
- Klicken Sie auf „Clone“ und kopieren Sie die URL unter „Clone with HTTPS“ in die Zwischenablage. Sie hat die Form https://gitlab.rlp.net/username/projektname.git.
- Wechseln Sie zu Eclipse und klonen Sie das Repository. Hierzu in der Git Repository View in der Menüleiste das dritte Icon von links wählen  („*Clone a Git Repository …*“). Geben Sie Ihren User-Namen und das Passwort ein („Store in Secure Store“ anklicken).
- Auf der nächsten Seite den gewünschten Branch auswählen (hier: „master“). Auf der folgenden Seite ein Verzeichnis für das geklonte Repository angeben und „Import all existing Eclipse projects after clone finishes“ ankreuzen.

Nun können Sie alle an dem gemeinsamen Projekt weiterarbeiten.

- Jeder im Team legt eine neue Klasse an (die Namen sollten sich unterscheiden) und commited die Änderung. Hierdurch wird sie in das lokale Repository übernommen.
- Wählen Sie im Kontextmenü des Projekts *Team→Push to origin*. Dadurch werden die Inhalte, die gerade commited wurden, in das Repository auf dem GitLab-Server übernommen.
- Falls es Probleme gibt (weil schon jemand anders das Server-Repository geändert hat): Führen Sie zunächst über das Kontext-Menü des Projektes einen *Pull* durch. Dieser holt sich die aktuellen Inhalte vom Server und merged sie mit dem lokalen Projekt (*Pull* fasst zwei Befehle zusammen, die man auch einzeln ausführen könnte: „*Fetch*“ und „*Merge*“). In vielen Fällen funktioniert das ganz gut automatisch. Dann kann man die zusammengeführte Version wieder committen und pushen.
- Falls es beim Mergen Probleme gibt, muss man diese manuell lösen (s. o. unter Punkt 9). Anschließend die manuell zusammengeführten Dateien über „*Team → Add to index*“ in die Staging Area überführen, bevor man *Commit* und *Push* durchführt.

Bevor man Änderungen durchführt ist es sinnvoll, zunächst einen Pull vorzunehmen, damit das eigene Projekt mit dem entsprechenden Branch auf dem Server übereinstimmt, bevor man loslegt.

Hinweis: Für größere Erweiterungen empfiehlt es sich meist, diese in einem eigenen Branch durchzuführen (der ebenfalls in das Repository auf dem Server gepushed werden kann). Mit regelmäßigem Mergen sorgt man dafür, dass Änderungen des Hauptentwicklungszweigs in den Branch übernommen werden. Ist die Erweiterung dann fertig und getestet, kann sie wiederum in den Hauptentwicklungszweig gemerged werden. Mit diesem Vorgehen kommt es zu sehr wenigen Konflikten.

10. Einen bestimmten Entwicklungsstand markieren

Soll eine besondere Konfiguration markiert werden (z. B. wenn man ein für den Kunden freigegebenes Release mit einer Versionsnummer versehen möchte), kann man sie mit einem „Tag“ (einer Markierung) versehen.

Gehen Sie in die History View, selektieren Sie einen früheren Commit und versehen Sie ihn mit dem Tag „Version1.0“ (über das Kontextmenü mittels „*Create Tag*“). Tragen Sie auch etwas unter „Tag message“ ein und beenden Sie mit „*Create Tag and Start Push…*“ (Tags werden in den normalen Pushs nicht berücksichtigt. Falls man sie nicht direkt beim Anlegen pusht, kann man dies später in der Repository View durchführen).

Versehen Sie den aktuellen Stand mit dem Tag „Version2.0“. Hierzu im Kontextmenü des Projekts „*Team→Advanced→Tag…*“ wählen.

11. Änderungen von Entwicklern, die keine Schreibrechte haben.

In größeren Projekten soll vielleicht nicht jeder Mitarbeiter beliebig Änderungen im Hauptrepository vornehmen können. In diesem Fall erstellt der Mitarbeiter einen Klon des Repositories, führt dort in einem Branch seine Änderungen durch. Diesen Klon stellt der Entwickler dem Administrator des Hauptrepositories zur Verfügung, damit dieser ihn – nach einer Überprüfung der Änderung – in einen geeigneten Branch des Hauptrepositories mergen kann.

In der GitLab-Weboberfläche kann man hierfür sogenannte „Merge Requests“ erstellen (bei anderen Git-Repository-Hostern z. T. auch „Pull Request“ genannt), über die der jeweilige Repository-Eigentümer informiert wird. Mit diesem Mechanismus arbeiten auch viele Open Source-Projekte, die ihren Code als öffentliches Repository auf Github o. ä. zur Verfügung stellen, das jeder klonen kann. Beiträge der Community können dann kontrolliert in das Projekt eingebracht werden.