**Git & Spike Solutions**

Von Marco Egner, Dominik Thirmeyer

1. Das Versionskontrollsystem Git

Git ist ein Open Source Versionskontrollsystem, das unter der General Public License in der Version 2 veröffentlicht wurde.



Abbildung 1: Git Logo

Der Software Client steht für folgende Systeme bereit:

* Windows, Linux, Solaris, Mac OS X

2. Funktionsweise von Git

2.1 Die drei Zustände des lokalen Git-Repository

|  |  |
| --- | --- |
| **Working directory:** | Eine Kopie (clone) einer spezifischen Version des Projektes |
| **Staging Area:** | Enthält die für den nächsten Commit vorgemerkten Dateien |
| **Git directory** | Enthält die Speicherabbilder (entstehen bei commit) des Projekts (lokal), noch nicht im entfernten Repository |

|  |  |
| --- | --- |
| **Entferntes git repository** | Dateien auf dem Server |

2.2 Branches

Branches werde in Git angelegt, um eigene Entwicklungsstränge nach einem Ticket System bereitzustellen.

|  |  |
| --- | --- |
| **Branch anlegen** | Git branch <branchname> |
| **Branch wechseln** | Git checkout <branchname> |
| **Branch löschen** | Git branch –d <branchname> |
| **Branch zusammenführen** | Git merge <branchname> |

3. Git Grundkonfiguration

|  |  |
| --- | --- |
| **Name definieren** | git config –global user.name „<Username>“ |
| **E-Mail-Adresse definieren** | git config –global user.email <user@email.com> |
| **Standard Push Logik definieren (Für kürzeren push-Befehl, nur noch git push)**  **Benötigt git push --set-upstream origin <branchName>** | git config push.default simple </pfad/zum/repository> |
| **Kopieren des Projekts incl. Erzeugen des working directory:** | git clone https://github.com/daHaimi/2dgame-thi-2016.git [<Ordnername>] |
| **Geänderte Dateien zur Staging Area hinzufügen und commiten:** | git commit -a -m „<Commit Nachricht>“ |

3. Die wichtigsten Befehle in Git

|  |  |
| --- | --- |
| **Git init** | Erstellen eines neuen Repository |
| **Git clone /pfad/zum/repository oder**  **/pfad/zum/repository <Ordnername>** | Erstellt eine Arbeitskopie des Git Repository, gegebenenfalls in extra angegebenen Ordner |
| **Git add <dateiname> oder git add \*** | Fügt eine, mehrere oder alle noch nicht aufgeführten Dateien zum Repository hinzu oder merkt bereits vorhandene und modifizierte Dateien für den nächsten Commit vor |
| **Git rm <dateiname>** | Löscht die Angegebene Datei aus dem Repository (bei ganzen Verzeichnissen wird der Parameter –r benötigt) |
| **Git commit –m „Commit Nachricht“** | Bestätigen der Änderungen (modifizierte, hinzugefügte und gelöschte Dateien) gegenüber des lokalen Git Repository |
| **Git status** | Zeigt die getätigten Änderungen (nicht commited) an |
| **Git push** | Übertragen der commits an master repository |
| **Git pull** | Aktualisieren der lokalen repository vom master |
| **Git diff <quell\_branch><ziel\_branch>** | Zeigt Differenzen zweier Branches an |

4. Spike Solution

Unter dem Begriff Spike Solution versteht man kurz gesagt ein Code Experiment.

Ziel hierbei ist es, Wissen durch einfaches Ausprobieren möglichst schnell zu gewinnen. Dabei achtet man nicht auf Clean-Code-Constrains sowie Unit-Tests und schreibt dabei nur so viel Code wie nötig um seine definierte Fragestellung zu beantworten.

Eine gängige Praxis ist es, Spikes in eigenen Branches zu kapseln. Der darin erzeugte Quelltext wird aber keinesfalls in das Projekt übernommen, das heißt der Branch wird nicht per merge-Befehl mit dem master-Branch zusammengeführt.

Ist die Frage beantwortet, wird der Code nicht verbessert, sondern weggeworfen. Das aus dem Experiment gewonnenen Wissen wird dazu genutzt, um eine saubere, den Richtlinien entsprechende Implementierung zu erzeugen.

Nützliche Quellen:

<https://git-scm.com/book/de/v1>

<https://git-scm.com/docs>

<http://qualitycoding.org/spike-solution/>