Git-Guideline

-

Einführung in das verwendete Versionsmanagement

Version 1.1

Inhalt

0.1 Ziel des Dokumentes

Das Dokument „Git-Guideline“ dient der Aufklärung über unseren Umgang mit dem Versionsmanagement, der durch verschiedene im folgenden Regeln und Richtlinien erläutert ist.

0.2 Changelog

Version	Datum	Änderung	Geändert von
1.0	25.10.2018	Initial	Tim Dahm
1.1	10.11.2018	Formatierung	Steffen Sassalla
1.2	16.12.2018	Bennenung Feature Branches	Tim Dahm

1. Einführung Git

Das von uns benutzte Versionsmanagementtool „GitLab" läuft auf den Servern des „Informatik Rechner Betrieb", kurz IRB, und ist eine Alternative zu GitHub. Es ist ein so genanntes webbasiertes Repository, welches eine Kollaboration mehrerer Entwickler eines einzigen Projektes vereinfacht, Tools zum versionierten Testen bietet und Issue-Tracking vereinfacht und übersichtlich hält. GitLab ist dabei eine Alternative zu Github und wurde auf einer anderen Programmiersprache (Ruby) geschrieben und bietet Komponenten wie ein Berechtigungssystem oder ein, gegenüber Github überarbeitetes und verbessertes, Ticketsystem.

2. Nützliche Tools und Programme

Git kann sowohl mittels einer Kommandozeilen-Applikation als auch durch eine oberflächengesteuerte Anwendung verwendet werden. Die Funktionalität unterscheidet sich bei den meisten verfügbaren oder auch unten aufgelisteten Anwendungen kaum. Die Wahl ist hauptsächlich eine präferenzielle.
Im Folgenden ist eine Auflistung einiger weit verbreiteten Tools zu finden:

GitLab
Git mit Oberfläche:

- Windows Git Applikation
- GitHub Desktop
- Git for Windows
- GitKraken

Git mit Konsole:

- Windows Git Applikation
- Git for Windows

Git mit Windows-Integration:

- Tortoise
- Git for Windows

3. Master und weitere Branches

Achtung: Branches werden immer klein in der englischen Sprache geschrieben und enthalten keine Leerzeichen! Commits in den Master werden immer mit dem jeweiligen Teamleiter abgesprochen. Dokumente gilt es in den develop-Branch unter /docs zu pushen.

3.1 Master

Der master-Branch ist eine besondere Art des Branches, da es der wichtigste und erste in jedem Repository ist. Dieser enthält zu jedem Zeitpunkt eine laufende und die zuletzt veröffentlichte Version. Außerdem werden, je nach Dringlichkeit des Fehlers, Hotfixes sogar direkt auf den Master gemergt. Versionen werden ausschließlich von dem Teamleiter getaggt.
Erstellt wird dieser als initialer Branch.

3.2 Develop

Der develop-Branch enthält immer die aktuelle Entwicklungsversion und erhält alle Fixes die keine hohe Priorität haben. Außerdem ist dieser Branch der, von dem alle Feature-Branches ableiten und zu dem eben diese auch wieder zusammengeführt werden. Auch dieser sollte immer eine laufende Version enthalten und enthält alle Dokumente zuerst bevor diese in den master-Branch gemergt werden.

Erstellt wird dieser als initialer Branch zusammen mit dem master.

3.3 Feature

Der feature-Branch leitet sich in den meisten Fällen vom develop-Branch ab und unterstützt einem Entwickler oder eine Entwicklergruppe bei der Entwicklung eines Teilproblems oder auch der Lösung eines Issues, bspw. “feature-fixPlayButton”, mit niedriger Priorität. Es kann jedoch auch vorkommen, dass der feature-Branch auch von anderen feature-Branches ableitet. Erstellt wird der feature-Branch falls es eine neue Komponente geben soll, also die Entwicklung einer neuen Oberfläche oder einer neuen Funktionalität.
Ein feature für eine bestimmte Komponente wird mit der Abkürzung dieser, gemäß der Tabelle, zwischen feature und dem feature Namen kenntlich gemacht: feature-abkürzung-featureName.

Komponenten Name	**Abkürzung**
Desktop Beobachter	db
Android Client	ac
Spielserver	se
Künstliche Intelligenz	ki

3.4 Hotfix

Der hotfix-Branch kommt bei einem dringenden Problem im master-Branch zum Einsatz, damit das Problem schnellstmöglich behoben werden kann. Ein hotfix-Branch leitet sich somit direkt vom master-Branch ab. Ein Beispiel ist eine Sicherheitslücke oder ein gravierender Bug, der die Benutzung der Anwendung stark beeinträchtigt. Es gilt hierbei einen Fix von einem Hotfix zu unterscheiden, da ein Problem mit niedriger Wichtigkeit auch durch ein feature Branch behoben und dann in einem späteren Update erst hinzugefügt werden kann. Nach Behebung des Problems wird dieser Branch **sowohl** in den master-Branch **als auch** in den develop-Branch gemergt.
Erstellt wird dieser bei einem dringenden Problem im master-Branch, welches sofortige Behebung erfordert, sofern es die Stabilität, Nutzbarkeit oder Sicherheit stark einschränkt oder auch gefährdet und erfordert Absprache mit dem jeweiligen Teamleiter.

3.5 Release

Der release-Branch wird bei der Vorbereitung einer Veröffentlichung einer bestimmten Version verwendet und leitet sich direkt vom develop-Branch ab. Der release-Branch wird vor einem bevorstehenden Release erstellt, getestet und ergänzt, sodass er zur Deadline oder bei Fertigstellung in den master-Branch **und** in den develop-Branch gemergt werden kann.
Erstellt wird der Branch sofern eine Veröffentlichung, wie beispielsweise die Messe, bevorsteht, so dass dieser zum exzessiven Testen zur Verfügung steht und die allgemeine Entwicklung nicht beeinflusst.

4. Commits

Ein Commit verbindet Veränderung im Repository mit unter anderem einer Commit-Nummer, einem Autor, dem Datum und bietet die Möglichkeit einer Beschreibung.
Eben diese Beschreibungen können sehr schnell unübersichtlich werden und helfen der Entwicklung nur bedingt.
Ein Commit unterteilt sich in ein benötigtes „Summary” und eine optionale „Description”.
Dabei sollte ein Summary aus weniger als 50 Charakteren bestehen und das Thema sehr knackig und kurz darstellen - die Aussage sollte atomisiert sein.

Außerdem kann mittels der Commit-Nachricht auch der Status einer User-Story oder eines Tasks geändert werden. Dies muss durch die Phrase „TG-REF #STATUS-slug“ am Ende der Nachricht geschehen. „REF“ muss durch die entsprechende Identifikationsnummer und „STATUS-slug“ durch einen, in Taiga, definierten Status ersetzt werden.
Ein Commit ist in Englisch zu schreiben. Zudem ist es erlaubt Abkürzungen zu nutzen.

Im folgenden ist ein Beispiel von https://chris.beams.io/posts/git-commit/#why-not-how zu finden:

```
$ git log
commit 42e769bdf4894310333942ffc5a15151222a87be
Author: Kevin Flynn <kevin@flynnsarcade.com>
Date:   Fri Jan 01 00:00:00 1982 -0200

    Derezz the master control program

    MCP turned out to be evil and had become intent on world domination.
    This commit throws Tron's disc into MCP (causing its deresolution)
    and turns it back into a chess game.
```

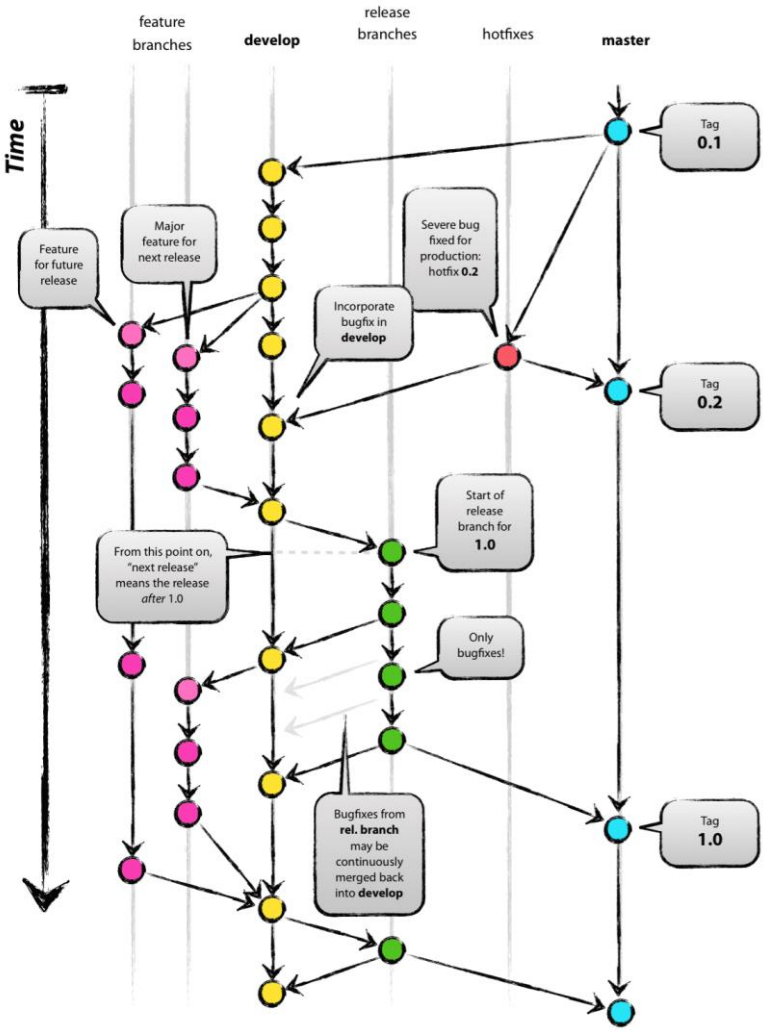
Auf dieser Website werden 7 Regeln für Commits angesprochen, an die wir uns auch halten werden. Eine genauere Beschreibung der Regeln ist auf der oben genannten Website zu finden:

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters

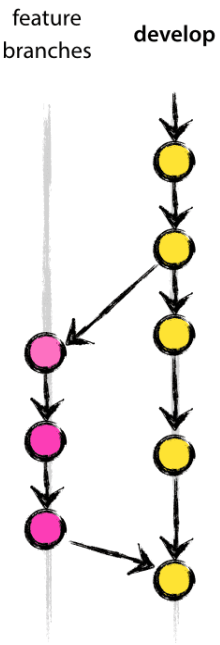
Use the body to explain *what* and *why* vs. *how*

5. Erläuterung am Beispiel

Zum Nachlesen des gesamten Artikels: https://nvie.com/posts/a-successful-git-branching-model/



In diesem Beispiel sieht man den Verlauf eines Repositories mit den oben gelisteten Branches. Dabei leitet sich der develop-Branch von dem Master ab, von dem sich darauf 2 weitere feature Branches ableiten, die Komponenten für den nächsten Release vorbereiten. In der Zwischenzeit ist ein extremer Fehler aufgetreten, der eine sofortige Behebung durch einen Hotfix bedarf. Dabei leitet sich eben dieser Branch direkt von dem master-Branch ab und wird nach Behebung in den Master und in den develop-Branch gemergt. Das markiert somit Version 2.0 für den master-Branch. Darauf anstehend ist der 1.0 Release, dessen „major feature" für den nächsten Release in den develop-Branch gemergt wird. Darauffolgend wird der release-Branch für die 1.0 von dem develop-Branch abgeleitet und unterzieht sich in dieser Zeit Tests des Testmanagers und der Quality Assurance. Bei größeren Veränderungen oder unter Umständen auch bei Bugfixes werden diese auch von dem release-Branch in den develop-Branch zurück gemergt. Bei Fertigstellung des Releases wird dieser sowohl in den develop-Branch als auch in den master-Branch als nächste Version, in diesem Falle die 1.0, gemergt.



6. Allgemeine Tipps

- Branching Simulator: https://learngitbranching.js.org/
- Falls feinere Möglichkeiten bei der Arbeit mit Git gefordert sind, gibt es die Möglichkeit die Command-Line zu nutzen
- „Commit early and often“