Git und Git Flow

Warum git?

Es gibt im Web viele, wenn nicht sogar tausende, sogenannte „Flame Wars“ über die Pros und Contras zu Git. Aus diesem Grund möchte ich hierauf auch nicht weiter eingehen. Es sollte jedoch erwähnt werden dass git die Entwicklung von Software in einem Team stark vereinfacht. Neben den Vorteilen die jedes Versionierungstool beinhaltet sticht git vor Allem mit einer Besonderheit ins Auge: Während Merging und Branching bei klassischen Tools wie CVS oder Subversion teilweise ein Drahtseilakt war, ist es bei git so simpel dass jeder Neuling damit umgehen kann.

Arbeiten mit Git

Der wohl größte Vorteil an der Arbeit mit git ist, dass ein ganzes Team an einem Projekt arbeiten kann ohne Angst haben zu müssen die Änderungen anderer Teammitglieder zu überschreiben. Da es gute, tiefgehende Dokumentationen zu git gibt, möchte ich hier allerdings nur oberflächlich ansetzen.

Die Funktionsweise der Teamarbeit mit git kann in einem theoretischen Aufbau auf ein simples Beispiel runtergebrochen werden:

Die Entwicklungsabteilung erhält einen neuen Auftrag und bildet daraufhin das Team „neuerAuftrag“. Auf dem Server wird ein leeres git Verzeichnis erstellt (ein sogenanntes „git repository“) und anschließend mit den Grunddaten (z.B. die Grundinstallation der Firmensoftware) gefüttert. Nun zieht der Projektleiter „Mark“ sich via SSH und dem Befehl „git clone“ eine Kopie des Verzeichnisses auf seinen Computer. Während Mark seine ersten Aufgaben erledigt zieht auch das Teammitglied „Alex“ eine Kopie auf seinen lokalen Computer. Durch das Branching und Merging innerhalb von git können beide Entwickler nun lokal an den Daten arbeiten und diese anschließend wieder in das git repository auf dem Server migrieren. Zusätzlich kann jeder Mitarbeiter sind zu jeder Zeit den aktuellsten Stand aus dem git repository auf dem Server holen und die lokalen Daten damit auf den neusten Stand bringen.

Die wichtigsten git Befehle sind…

```
git init – initialisiert ein neues git repository
git clone – kopiert das git repository vom Server auf einen lokalen Computer
git pull – aktualisiert das lokale repository mit den Daten vom Server
git add – fügt geänderte, gelöschte oder neue Dateien zum repository hinzu
git commit – erstellt eine „commit Nachricht“ zur Beschreibung was getan / geändert wurde
git checkout – wechselt auf einen bestimmten Branch
git branch – erstellt, löscht und listet Branches
```

Was sind Branches?

Git verfügt über die Möglichkeit sogenannte „Branches“ anzulegen. Diese agieren als Bezeichner für einen gewissen Versionsstand. Für die Entwicklung mit git unterscheidet man im Regelfall zwischen master, develop und release. In dem Branch „release“ befindet sich ein Versionsstand der Software, der als Vorbereitung für ein Release bestimmt ist. In dem Branch „master“ befindet sich immer NUR der Versionsstand der aktuellsten Veröffentlichung. Der Branch „develop“ ist für die Entwicklung gedacht. Man ist jedoch nicht auf diese Branches limitiert. Da git keine Begrenzung für die Anzahl der

Branches kennt, ist es möglich so viele anzulegen wie man benötigt. Es ist auch immer sinnvoll lokal einen neuen Branch zu erstellen statt auf dem „develop“ zu arbeiten. So hat man den Vorteil, dass man jederzeit den neusten Stand des „develop“ (z.B. mit Änderungen von Teammitgliedern) vom Server holen kann und dort die eigenen Anpassungen hinein mergen kann, sobald man soweit ist.

Was bedeutet Tagging?

Tagging ist grob gesagt eine Art der Vergabe von Release-Nummern an bestimmte Commits. Auf diese Weise ist es möglich später auf bestimmte Tags bzw. Release-Nummern zu referenzieren.

Was ist git flow?

Git flow ist eine Sammlung von Erweiterungen für git, die das Branching nach dem Modell von Vincent Driessen vereinfacht. Da git flow nur die Funktionalität von git erweitert, ist eine vorherige Installation von git nötig. Die Installation der Erweiterungen ist simpel über verschiedene Tools wie z.B. apt-get, brew oder port möglich. Unter Windows wird das Tool npm benötigt.

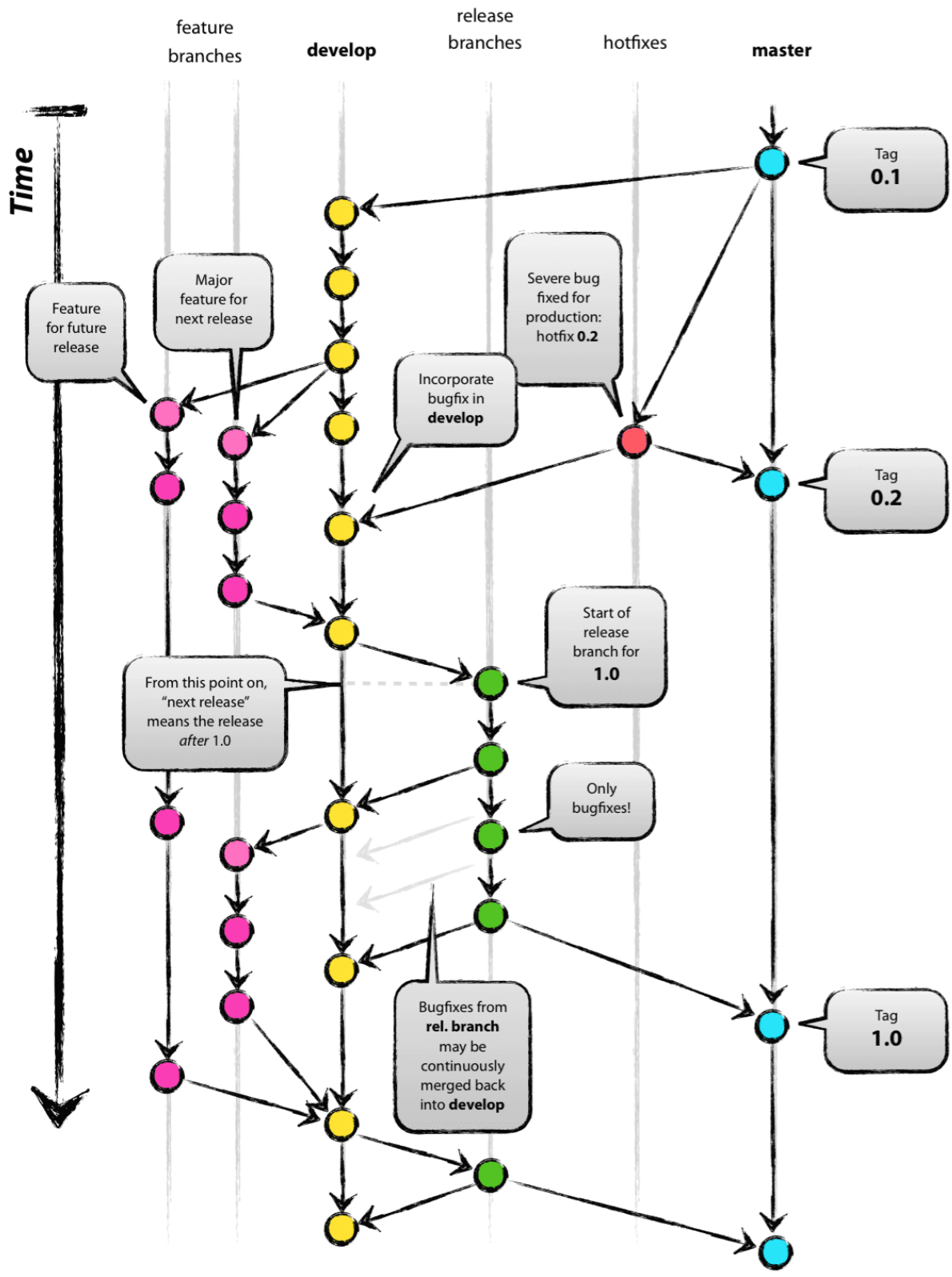
Ist git flow installiert muss es in dem gewünschten Projekt initialisiert werden. Hierfür muss man einfach den Befehl „git flow init“ in dem bestehenden lokalen git repository ausführen.

In git flow unterscheidet man, nach dem Modell von Vincent Driessen, zwischen den folgenden Branches:

- master: Die veröffentlichten Versionen. Der Branch sollte immer den Stand der aktuellsten Veröffentlichung beinhalten. (Versionierung-Bsp. v1.0.0)
- release: Hier wird die Veröffentlichung vorbereitet.
- hotfix: Ein hotfix ist ein Bugfix der in einer Kopie des Master erfolgt und anschließend in den release und den master Branch eingespielt wird. (Versionierung-Bsp. V1.0.1)
- develop: Der aktuelle Haupt-Entwicklungsstand.
- feature: In diesen Branches werden Erweiterungen für das aktuelle Projekt entwickelt und anschließend in den develop Branch eingespielt. (Versionierungs-Bsp: v1.1.0)

Strategie und Management

Das Branching Modell von Vincent Driessen basiert auf der Logik, dass für Entwicklungsprozesse entsprechende Branches angelegt werden. Zur Veranschaulichung dient folgende Grafik:



Grafik: © by Vincent Driessen

Naming Conventions

Es empfiehlt sich die verschiedenen Branches nach einem gewissen Muster zu erstellen:

```
- release: release-[Versionsnummer]
- hotfix: hotfix-[Bezeichnung]
- feature: feature-[Bezeichnung]
```

Quellen

A successful Git branching model by Vincent Driessen

Git Flow Cheatsheet by Daniel Kummer