git

Dienstag, 3. Oktober 2017 11:49

Erfunden von Linus Torvalds im Jahr 2005, als Werkzeug für die Entwicklung des Linux Kernels. Heute das wahrscheinlich am meisten verbreitete Version Control System (VCS).

Distributed VCS => die Versionen werden auf jedem beteiligten Rechner lokal gespeichert, es gibt keinen zentralen Speicherpunkt (oder zumindest nicht zwangsläufig). Auf jedem PC liegt ein voll funktionierendes **Repository**.
Gearbeitet wird in einer **Working Copy.**

Bevor Änderungen „committed“, also in der history „gespeichert“ werden, müssen sie dafür vorgemerkt (**staged**) werden -> sie kommen in die **Stagin Area** oder **Index**.
Committed wird dann auf einen lokalen **Branch**, der auch auf anderen PCs (oder dem Repo-Server) existieren kann, aber nicht muss. Der letzte commit auf dem aktuellen branch heißt **HEAD**.
Ein oder mehrere commits können dann an einen remote **gepusht** werden, vorausgesetzt niemand anderes hat inzwischen seine eigenen Änderungen gepusht.

Gibt es hingegen schon remote changes, so müssen diese **gepullt** und zusammengeführt – **merged** – werden. Dabei werden Änderungen – auch in der gleichen Datei! – einfach kombiniert. Verändern zwei commits die gleichen Zeilen, kommt es zu einem merge conflict, der dann gelöst werden muss.

Ein branch ist ein Zweig in der Versionsgeschichte (version history), der default branch heißt **master**. Grundsätzlich kommt man auch mit einem branch aus, es gibt aber aus gutem Grund git workflows, die mehrere branches benutzen.

Grundlegende Befehle

git init	initialisiert ein git repo im aktuellen Ordner, erzeugt einen .git Ordner
git clone <repository> [target]	klont ein existierendes Repository von einem remote
git status	gibt den Status dieses git repos aus (aktueller branch, Änderungen ...)
git add <paths>	fügt Ordner und Dateien den „tracked files“ hinzu, oder staged sie
git rm <path>	löscht eine Datei, und führt git add aus, um die Änderung zu stagen
git commit -m "Lognachricht“	erstellt einen 'commit', also einen Speicherpunkt
git push <remote> <branch>	'pusht' den aktuellen branch auf einen remote (Server) branch
git pull <remote> <branch>	holt die letzten Änderungen von remote branch
git log [-p]	zeigt die Versionsgeschichte des aktuellen branches an
git reset --soft <sha>	macht commits rückgängig bis <sha>
git reset [--mixed] <sha>	macht commits und stages rückgängig bis <sha>
git reset --hard <sha>	macht commits, stages und Änderungen rückgängig bis <sha>
git reset --hard	macht working copy Änderungen (in **getrackten** files) rückgängig
git checkout <path>	macht Änderungen in einer Datei rückgängig (git reset --hard <path>)

git revert <sha>	erzeugt ein komplementäres (undo) commit zu den angegebenen

<sha> ist der hash (id) eines commits (auch nur ein Teil davon), man findet ihn mit git log oder git reflog. HEAD ist der letzte commit, HEAD~1 ist ein commit vorher, HEAD~2 zwei commits vorher usw.

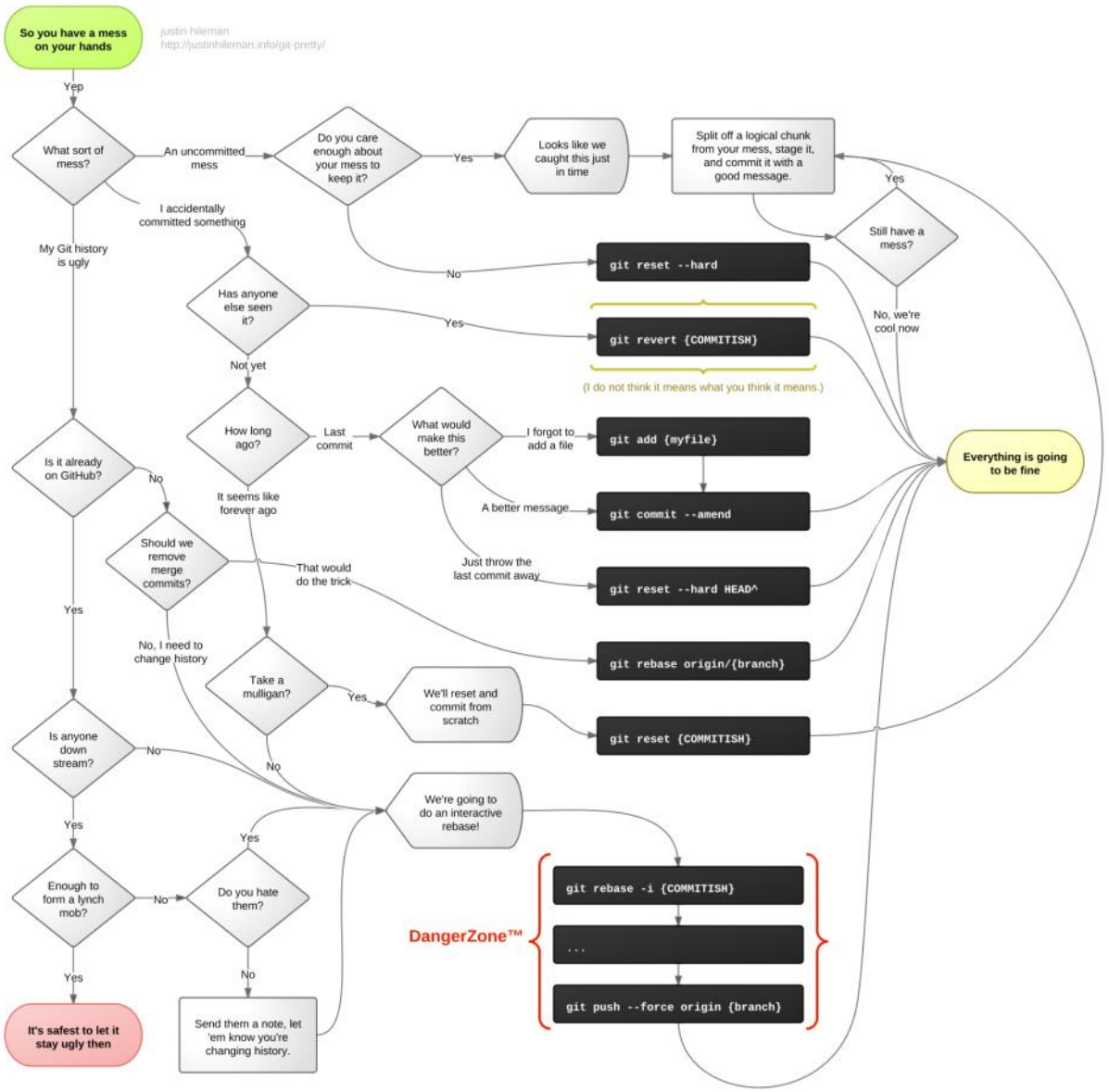
Bevor man Änderungen von einem remote pullt, sollte man dafür sorgen, dass das Arbeitsverzeichnis 'clean' ist, also keine Änderungen enthält, die noch nicht committed wurden:

a. Änderungen adden und committen
b. Unfertige Änderungen eventuell stashen (git statsh)
c. Dateien, die man nie committen will, ignorieren (per .gitignore Datei, die man auch committen sollte!)

Damit ist sichergestellt, dass man die Working Copy im Falle von merge conflicts nicht in einen unübersichtlichen Zustand bringt!

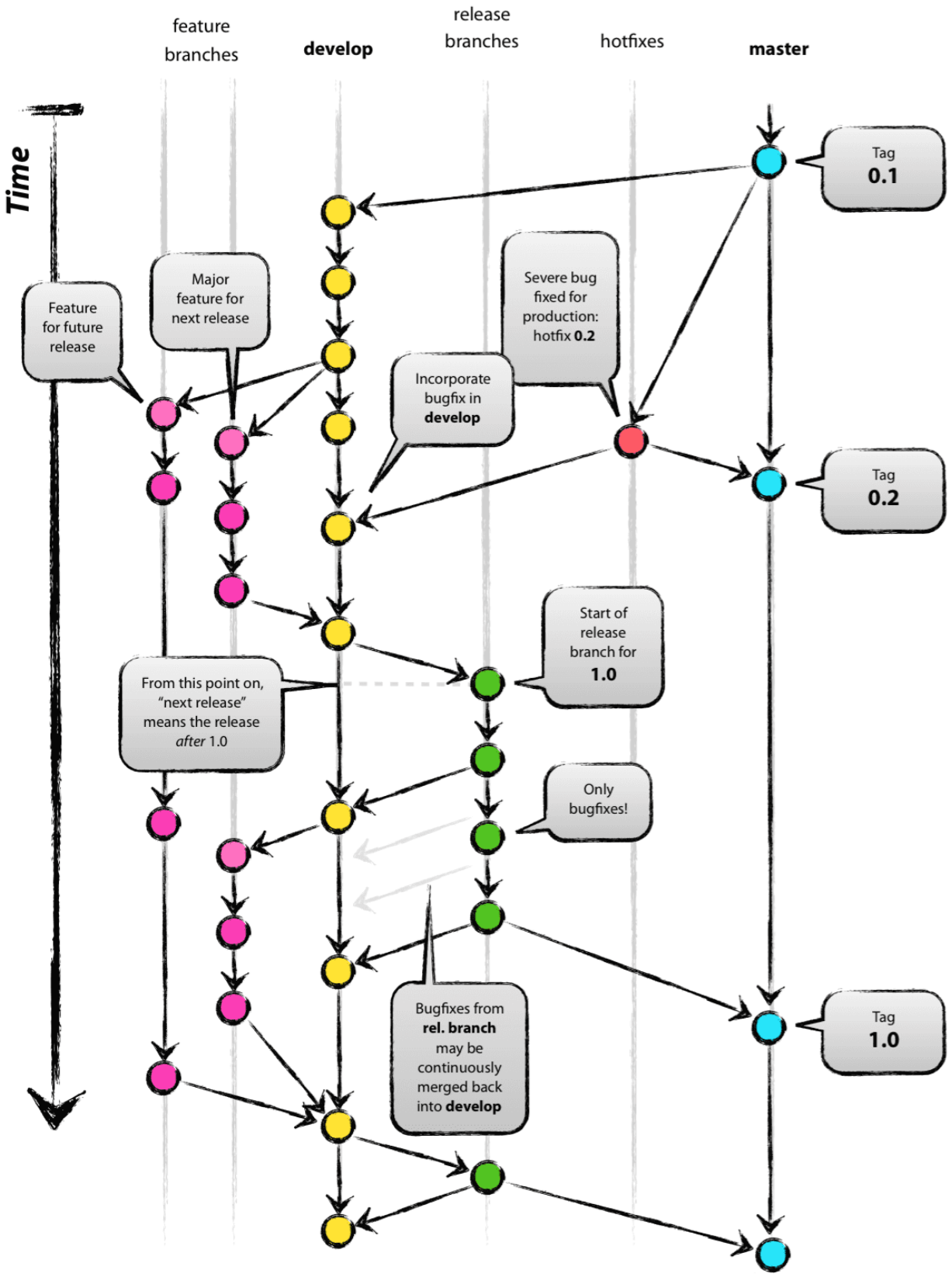
Erste Hilfe bei Chaos

(Quelle: http://justinhileman.info/article/git-pretty/)



Gitflow

(Quelle: http://nvie.com/posts/a-successful-git-branching-model/, Vincent Driessen)



Ein Hauptgedanke von git ist, dass mehrere Entwickler gleichzeitig am Projekt arbeiten können. Dafür empfiehlt es sich, sich einen workflow wie z.b. Gitflow anzueignen. In groben Zügen sieht das so aus:

Der **master** branch wird nur für Releases (also fertige Versionen, die möglichst bug-frei sind)

verwendet. Der „Arbeitsbranch“ ist der develop oder **dev** branch. Wenn ein Entwickler mit der Implementierung eines neuen Features beginnt, erstellt er von dev ausgehend einen feature-branch (meist heißen diese **feature/featureName**). Dieser branch existier meist nur lokal, er kann aber auch gepusht werden, z.B. um es anderen Entwicklern zu ermöglichen, daran zu arbeiten oder einen Blick darauf zu werfen.

Bevor in den master branch gemerged wird, kann ein **release-branch** hilfreich sein. Dabei wird einfach der aktuelle Stand des dev branches in den release-branch gemerged, dann wird getestet und eventuelle bugs gefixt. Es werden aber keine neuen features mehr implementiert! Fixes auf dem release-branch werden in den dev branch zurück gemerged, damit sie auch dort ankommen.

Ist man mit dem release-branch zufrieden, wird auf **master** gemerged, und mit der Versionsnummer getagged. Sollten in production schwerwiegende Fehler auftreten, werden sie auf einem **hotfixes** branch gefixt, und in master und dev gemerged.

CMD und GUIs

Auf Linux/Mac ist das Programm 'git' normalerweise schon vorinstalliert, für Windows kann es von der Webseite git-scm.com heruntergeladen werden, unter anderem auch als portable Version (z.B. für die Schulcomputer).
Vor allem für Anfänger (und auch allgemein, wenn man größere Änderungen vor hat) ist eine GUI aber von Vorteil. Für nicht-kommerzielle Zwecke eignet sich die cross-platform GUI „GitKraken“ wohl am besten (gratis, für kommerzielle Projekte ist eine subscription notwendig). Von SmartGit (ebenfalls gratis für nicht-kommerzielle Nutzung) gibt es auch eine portable Version.

Alternativen

Natürlich ist git nicht das einzige SCM tool, für viele aber das Beste – aus mehreren Gründen (siehe http://web.archive.org/web/20090210020404id_/http://whygitisbetterthanx.com).

Kommerzielle Projekte verwenden oft Perforce (p4), unter anderem wegen der Möglichkeit, Code- von Content-Commits trennen zu können, und auch weil es mit großen Binärdateien besser umgehen kann (git kann keine binary-diffs erstellen, speichert also bei jedem commit eines binary files eine neue Kopie!).
Eine weitere Alternative ist Subversion (SVN), welches zwar einfacher zu bedienen ist (es gibt keine Staging Area, commit und push sind in einem Befehl vereint), allerdings auch viele Nachteile gegenüber git hat (fehlende Staging Area, keine lokalen commits, nicht distributed, mit branches zu arbeiten ist umständlich, usw…)