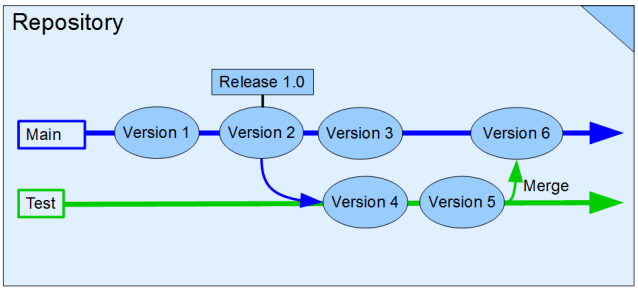
Versionskontrolle

Was ist Versionskontrolle und wieso braucht man sie?

Beim Arbeiten mit Dateien wäre man manchmal gerne in der Lage, etwas rückgängig zu machen, um einen anderen Weg einzuschlagen. Wenn wir zum Beispiel an einem Programm arbeiten und merken, dass der eingeschlagene Weg nicht zum Erfolg führt, würden wir gerne zu einem früheren Standpunkt des Projektes zurückkehren. Viele Texteditoren ermöglichen dies. Doch was ist, wenn ich das Projekt auf den Stand von Gestern oder von vor einer Woche zurücksetzen möchte? Was ist, wenn ich einen Bug, bzw. ein Fehlverhalten bemerke und herausfinden will, wann sich dieses eingeschlichen hat? Wie können mehrere Entwickler am gleichen Programm arbeiten? Bei solchen Problemen können Programme für die Versionskontrolle helfen.

Was muss ein Programm für die Versionskontrolle können?

Ein *VCS* (Ein Programm zur Versionskontrolle: Version Control System) muss ermöglichen ältere Zustände des Projektes wiederherzustellen. Ein solcher gespeicherter Zustand wird *Version* genannt. Es sollte möglich sein zwischen allen Versionen im *Repository* hin und her zu springen. Das Repository ist der Ort, an welchem die Versionen gespeichert werden. Zusätzlich sollte mit jeder Version eine Notiz gespeichert werden, die sogenannten *commit-messages*. Diese ermöglichen es, Änderungen an einem Projekt nachzuvollziehen und dienen gleichzeitig als Protokoll. Da ein Projekt meistens von mehreren Personen bearbeitet wird, muss ein VCS dies ermöglichen und regeln: das beinhaltet z.B das Zuordnen einer Änderung zu einem Entwickler, sowie das Management der Lese- und Schreibrechte. Ein VCS sollte ausserdem das Arbeiten an verschiedenen *Branches* unterstützen. Unter Branching versteht man eine Duplikation eines Objektes unter Versionskontrolle. Diese Funktion erlaubt es, dasselbe Projekt gleichzeitig an mehreren Bereichen unabhängig oder isoliert voneinander zu bearbeiten. Wenn zwei Branches oder Dateien mit unterschiedlichen Versionsnummern zusammengeführt werden, heisst dieser Vorgang *merging*. Spezielle Versionen können mit einem Namen, dem *Tag* versehen werden. Das Tag erlaubt es einem bestimmte, meist wichtige Version, schnell zu lokalisieren und diese Version wieder herzustellen. Der Vorgang, bei welchem der aktuelle Zustand des Projektes ins Repository kopiert und unter Versionskontrolle gestellt wird, heisst *commit*. Die lokale Kopie, an welcher man arbeitet, heisst *working copy*.



Darstellung eines Repositories mit zwei Branches: "Main" und "Test". Version 2 ist als "Release 1.0" getagged. Nachdem die Entwicklung auf dem Testbranch erfolgreich war wird er wieder in den Hauptzweig gemerged.

Zentrale Versionskontrolle

Die ersten VCS (z.B CVS, SVN) waren zentralisiert. Das heisst, es gab *ein* Repository, auf welches alle Entwickler an einem Projekt zugriffen. Der normale Arbeitsablauf mit einem zentralen CVS sieht folgendermassen aus: Als erstes holt man sich aus dem Repository die neuste Version. Dieser Vorgang wird *checkout* genannt. Danach arbeitet man daran, und bevor die Änderungen committed werden können, sollte zuerst nochmal ein checkout erfolgen. Denn es könnte sein, dass jemand anderes gleichzeitig dieselben Dateien bearbeitet hat. Dies würde zu Konflikten im Repository führen. Der Konflikt wird mit dem zweiten checkout auf den Rechner verlagert und dort lokal behoben, bevor die Änderungen in das Repository geschrieben werden können.

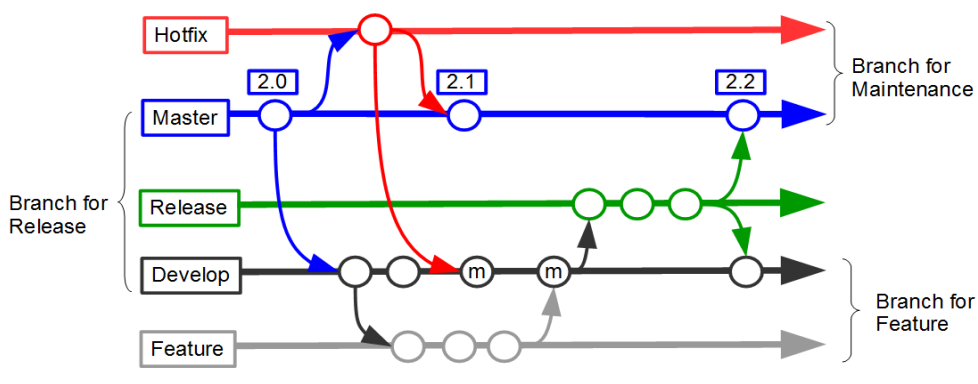
Zentrale VCS haben jedoch einige unangenehme Eigenschaften. Z.B. darf Code nur committed werden, falls er vollständig ist und funktioniert. Wenn unvollständiger oder fehlerhafter Code in das Repository gelangt, beeinflusst das alle anderen, die am Projekt arbeiten. Ein zweites Problem ist, dass man für lokale Änderungen keine Versionskontrolle hat. Falls man etwas ändern oder testen möchte, ist man den Möglichkeiten und Grenzen der verwendeten Bearbeitungsprogrammen unterworfen. Es ist jedoch praktisch, auch ein lokales Repository zu haben um solche Änderungen und Tests lokal unter Versionskontrolle durchzuführen.

Verteilte Versionskontrolle

Aus diesen Gründen wurden in den letzten Jahren verteilte Versionskontrollsysteme entwickelt, Distributed VCS oder *DVCS* (z.B Git, Mercurial). Es ist immer noch möglich ein zentrales Repository zu führen. Jedoch hat man zusätzlich auch noch lokal ein vollwertiges Repository. Die Vorteile eines lokalen Repository haben sich in der Praxis durchgesetzt. Der Vorgang bei welchem ein Repository kopiert wird, wird *clone* genannt. Synchronisiert werden die Repositories über *pull* und *push*. Mit pull holt man die Änderungen und über push schreibt man sie in ein anderes Repository. Ein weiterer Vorteil verteilter VCS ist, dass man nicht immer Zugriff auf den zentralen Server haben muss.

Branchingstrategien

Darüber wann und wieso ein neuer Branch in einem Repository erzeugt werden soll, gibt es verschiedene Philosophien und Strategien:
No Branches: Es wird nie gebranched. Für kleine und mittlere Projekte geeignet, welche keine Isolation für Features oder Releases benötigen.
Branch for Release: Es wird ein Branch erzeugt um ein Release vorzubereiten. Das Release kann dann isoliert und unbetroffen von Entwicklungen am Hauptstrang getestet und verbessert werden. Änderungen, welche in dem Release Branch vorgenommen werden, müssen zurück in den Hauptstrang gemerged werden.
Branch for Maintenance: Um eine ältere Version des Programms zu unterhalten, wird ein Branch von diesem Release erzeugt. Dies dient dazu die momentane Entwicklung nicht zu destabilisieren. Änderungen an diesem Branch werden je nach Fall zurück in den Mainbranch gemerged oder nicht. Ein kleiner Fix für eine alte Version ist zum Beispiel in der momentan entwickelten Version nicht nötig, da sich dies durch Umstrukturierung des Codes oder anderen Änderungen erübrigt hat.
Branch for Feature: Branches werden aufgrund von Features erzeugt. Für jedes Feature wird ein neuer Branch generiert, in welchem das Feature isoliert von anderen Entwicklungen und Features bearbeitet werden kann.
Branch for Team: Jede Gruppe erhält ein Branch um unabhängig von Änderungen am Hauptzweig arbeiten zu können, oder um parallel auf Milestones zuzuarbeiten.
Gitflow: Der Gitflow kombiniert die vorgestellten Branching Strategien.



Beispiel für eine Gitflow Branching Strategie, welche 3 Branching Strategien vereinigt.

Quellen:
Atlassiaon: Comparing Workflows (https://www.atlassian.com/git/tutorials/comparing-workflows)
Beanstalk Guides: An introduction to version control (http://guides.beanstalkapp.com/version-control/intro-to-version-control.html)