FAQ

Johannes Merkert

Versionsverwaltung mit Git

Antworten auf die häufigsten Fragen

Warum Git?

Es gibt doch verschiedene Versionsverwaltungen. Wo liegen die Vorteile von Git?

Git wurde von Linus Torvalds und „einigen Hackern aus dem Netz“ ursprünglich für die Entwicklung des Linux-Kernels geschaffen. Mit der verteilten Versionsverwaltung archivieren Sie Dateiversionen lokal, gleichen Speicherpunkte im Team ab oder synchronisieren Ihre Änderungen mit einem Git-Server.

Der populäre Speicherdienst GitHub bietet kostenlose Git-Repositories für Open-Source-Projekte und hat sich durch dieses Angebot zur Standard-Plattform für quelloffene Software gemausert. Aber auch andere Plattformen wie Bitbucket oder Assembla bieten vergleichbare Angebote. Ihren eigenen Server machen Sie mit GitLab zum Git-Server samt Weboberfläche.

Alleine die lebhafte Community rund um GitHub rechtfertigt schon die Einarbeitung in das Programm. Die Möglichkeit, offline zu arbeiten, und raffinierte Mechanismen zum Zusammenführen der Änderungen verschiedener Entwickler machen das Tool zu Recht zum Standard, an dem sich alle Versionsverwaltungen messen müssen.

Repositories überall

Was bedeutet „verteilte Versionsverwaltung“? Bringt das wirklich Vorteile gegenüber einem zentralen Server?

Git verwaltet auf allen Rechnern, auf denen es läuft, lokale Kopien der Repositories. Sie können Git lokal ohne zentralen Server benutzen oder zeitweise lokal arbeiten und die Änderungen später auf einen Server übertragen.

Git synchronisiert per HTTPS, SSH oder einem eigenen Protokoll Daten zwischen dem lokalen und einem Repository auf einem beliebigen anderen Rechner. Im Unterschied etwa zu Subversion benötigt Git keinen zentralen Server.

Durch die verteilte Struktur kommen Git-Nutzer einander nur dann in die Quere, wenn sie eine Sammlung von Commits aus ihrem privaten Repository auf einen Server pushen. Git erlaubt einen Push nur, wenn auf dem Server kein neuerer Commit eines anderen Nutzers liegt. Ist das der Fall, muss der Nutzer die Änderungen vom Server erst pullen und mit seinem lokalen Repository zusammenführen (merge). Das stellt sicher, dass auf dem Server nichts landet, was nicht zusammenpasst.

Migration

Bisher habe ich Subversion eingesetzt. kann ich meine SVN-Repositories zu Git migrieren?

Dafür gibt es ein zusätzliches Tool namens `git svn`, das Sie möglicherweise nachinstallieren müssen. Git kann damit SVN-Repositories lesen und beschreiben. Ein lokales SVN-Repository unter /tmp/test-svn können Sie mit diesem Befehl in ein neues Git-Repository kopieren:

```
git svn clone file:///tmp/test-svn ↗
                    -T trunk -b branches -t tags
```

Tags in Subversion konvertiert Git zu Branches mit speziellem Namensschema. Prinzipiell können Sie mit diesem Schema weiterarbeiten und auch Änderungen von ihrem Git-Repository ins SVN integrieren. Wir empfehlen Ihnen aber, das SVN-Repository nur einmal zu klonen und anschließend ausschließlich Git zu nutzen. Danach haben Sie alle Freiheiten bei der Struktur und Benennung des Git-Repositories.

Git importiert aus CVS bis Version 2 mit `git-cvsimport`, `cvs2git` konvertiert auch aus neueren CVS-Repositories. Mercurial-Nutzer können mit dem Plug-in Hg-Git in Git-Repositories pushen.

Zeitreise

Ein Projekt hat auf GitHub die Lizenz geändert. Ich wünschte, ich hätte kurz vorher einen Fork erstellt. Geht das im Nachhinein?

Dafür bietet es sich an, zuerst auf GitHub die aktuelle Version zu forken. Anschließend klonen Sie den Fork auf Ihren Rechner. In der lokalen Version suchen Sie dann den Commit unmittelbar vor der Lizenzänderung. Das geht mit dem Befehl `git log` oder über das Web-Interface auf GitHub. Wenn der Commit beispielsweise den Hash `b58aaed` trägt, können Sie das Repository mit `git reset --hard b58aaed` auf diesen Commit zurücksetzen. Die Option `--hard` sorgt dafür, dass Git alle neueren Commits vergisst. Mit `git push origin master --force` zwingen Sie auch dem Repository auf GitHub die Zeitreise in die Vergangenheit auf. Die auf diese Weise vergessenen Commits zeigt GitHub in seinem Interface auch nicht mehr an.

Passwortabfrage bei GitHub

Ich habe ein Repository bei GitHub geklont. Jetzt muss ich für `git push` ein Passwort eingeben, obwohl ich meinen SSH-Schlüssel hinterlegt habe. Warum kommt diese Abfrage?

Wenn Sie auf GitHub Ihren öffentlichen SSH-Schlüssel hinterlegt haben, können Sie ohne Passworteingabe per SSH in ein Repository pushen. Falls Sie beim Klonen des Repositories jedoch statt der SSH-URL, die mit `git@` beginnt, die HTTPS-URL angegeben haben, werden Sie beim Pushen nach einem Passwort gefragt. Sie können die URL aber ändern:

```
git remote set-url origin git@...
```

Fehlersuche

Ich habe heute festgestellt, dass mein Programm einen Fehler hat, der vor einem Jahr noch nicht da war. In dieser Zeit habe ich Hunderte von Commits gemacht. Wie finde ich den Commit mit dem Fehler?

Git hilft in diesem Fall mit einer Automatik für eine binäre Suche nach dem Fehler. `git bisect start` startet die Fehlersuche. Danach müssen Sie zuerst angeben, ob der aktuelle Commit gut oder schlecht ist. Da Sie einen Fehler suchen, wird der neueste Commit den Fehler enthalten. Das teilen Sie git mit `git bisect bad` mit. Anschließend benennen Sie einen Commit, bei dem der Fehler sicher noch nicht bestand:

```
git bisect good 6905111
```

Git checkt anschließend einen Commit in der Mitte zwischen diesen Commits aus. Sie prüfen, ob der Fehler bei diesem Commit bereits bestand, und teilen Git das Ergebnis Ihres Tests mit: `git bisect bad`, wenn der Fehler besteht, sonst `git bisect good`. Mit diesem Befehl wird Git wieder einen Commit aus dem Zeitraum auschecken, in dem sich der Fehler eingeschlichen haben könnte, und der Prozess beginnt von vorne.

Die Suche ist beendet, wenn Git den ersten schlechten Commit ausgecheckt hat. Sie können nun in Ruhe prüfen, was das Problem ausgelöst hat. Mit `git bisect reset` verlassen Sie den Suchmodus und kehren zum aktuellen Stand zurück. (jme@ct.de)