Git Merkblatt-1/2

www.datamics.com

Grundlagen

Git ist ein Werkzeug für das Versionsmanagement von Software mit einem Fokus auf kollaborativer Softwareentwicklung.

Zusammengehörige Änderungen commiten

Ein Commit sollte nur zusammenhängende Codeänderungen beinhalten. Zwei verschiedene Bugs sollten beispielsweise mit zwei seperaten Commits behandelt werden.

Häufig commiten

Kleine, häufige Commits mit klaren Beschreibungen halten die Entwicklung grosser Projekte übersichtlich und nachvollziehbar.

Vor dem Commit testen

Vor dem Commit ist zu sicherzustellen, dass die Änderungen ihrem Zweck entsprechen.

Nur abgeschlossene Änderungen commiten

Grössere Änderungen sollten in möglichst viele, kleine, nachvollziehbare Zwischenschritte unterteilt werden, anstatt nicht-funktionalen Code zu commiten.

Gute Commit-Beschreibungen sind wesentlich

Die Commit-Beschreibung sollte nicht mehr als 50 Zeichen umfassen und klar die Änderungen im Präsens beschreiben.

Branches

Branches dienen der parallelen Entwicklung verschiedener Versionen einer Software. Schon bei einfachen Projekten ist es sehr empfehlenswert, zumindest einen stabilen Branch und einen Branch für Weiterentwicklungen zu verwenden, der bei Erfolg dann in den stabilen Branch gemergt werden kann.

Tags

Tags können verwendetet werden, um spezifischen Commits Versionsnummern und Beschreibungen anzuhängen, um dann schnell zwischen diesen wechseln zu können.

Workflow

Um effektiv kollaborativ arbeiten zu können, ist es wichtig, sich mit allen beteiligten Entwicklern auf einen Workflow zu verständigen, beispielsweise welche Branches langfristig unverändert belassen werden.

Pull Requests

Mit einem Pull Request können (auch unabhängige) Entwickler die Betreiber eines Projektes davon in Kenntnis setzen, dass eine spezifische Änderung implementiert wurde. Diese muss dann zuerst getestet werden und wird dann bei Erfolg in das Projekt integriert.

Repositories erzeugen

Ein neues lokales Repository erzeugen

```
git init
```

Ein existierendes Repository klonen

```
git clone \
ssh://nutzer@domain.com/repository.git
```

Lokale Änderungen

Geänderte Dateien anzeigen

```
git status
```

Änderungen anzeigen

```
git diff
```

Alle Änderungen dem nächsten Commit hinzufügen

```
git add .
```

Nur eine bestimmte Datei dem nächsten Commit hinzufügen

```
git add . -p DATEI
```

Commite alle lokalen Änderungen mit der gegebenen Beschreibung

```
git commit -am”Beschreibung”
```

Alle lokalen Änderungen dem letzten Commit hinzufügen

```
git commit –amend
```

(Nur für nicht bereits publizierte Commits!)

Alle lokalen Änderungen dem letzten Commit mit gleicher Beschreibung hinzufügen

```
git commit –amend –no-edit
```

Lokale Änderungen verwerfen

```
git stash
```

Commitverlauf

Alle Commits beginnend mit dem ersten anzeigen

```
git log
```

Die letzten 5 Commits anzeigen

```
git log -n5
```

Alle Änderungen einer bestimmten Datei anzeigen

```
git log -p DATEI
```

Wer hat was und wann an der Datei geändert?

```
git blame DATEI
```

Branches und Tags

Alle Branches anzeigen

```
git branch -av
```

Branch wechseln

```
git checkout BRANCH
```

Neuen Branch aus dem aktuellen HEAD erzeugen

```
git branch BRANCH
```

Lokalen Branch löschen

```
git branch -d BRANCH
```

Neuen Tag erzeugen

```
git tag TAG
```

Tags anzeigen

```
git tag
```

Tag löschen

```
git tag -d TAG
```

Zu einem Tag wechseln

```
git checkout TAG
```

Aktualisieren und Publizieren

Alle Remotes anzeigen

```
git remote -v
```

Informationen zu einer Remote anzeigen

```
git remote show REMOTE
```

Ein neues Remote Repository anlegen

```
git remote add REMOTE URL
```

Remote Repository herunterladen, ohne es in HEAD zu integrieren

```
git fetch REMOTE
```

Remote Repository herunterladen und in HEAD integrieren

```
git pull REMOTE BRANCH
```

Lokale Änderungen auf Remote publizieren

```
git push REMOTE BRANCH
```

Branch im Remote Repository lōschen

```
git branch -dr REMOTE/BRANCH
```

Tags publizieren

```
git push –tags
```

Merge und Rebase

Branch in den aktuellen HEAD mergen

```
git merge BRANCH
```

Commits des aktuellen HEAD auf ein Branch anwenden (Rebase)

```
git rebase BRANCH
```

(Nur für nicht publizierte Commits!)

Rebase abbrechen

```
git rebase –abort
```

Rebase nach dem Auflösen von Konflikten fortsetzen

```
git rebase –continue
```

Mergetool zum Auflösen von Konflikten starten

```
git mergetool
```

Dateien mit Konflikten als gelöst markieren

```
git add DATEI
```

Änderungen rückgängig machen

Alle lokalen Änderungen im Arbeitsverzeichnis verwerfen

```
git reset –hard HEAD
```

(Kann nicht rückgängig gemacht werden)

Lokale Änderungen einer spezifischen Datei verwerfen

```
git checkout HEAD DATEI
```

Commit rückgängig machen mit einem neuen Commit, der alle Änderungen verwirft

```
git revert COMMIT
```

Zurücksetzen auf ein Commit und alle Änderungen verwerfen

```
git reset –hard COMMIT
```

Zurücksetzen auf ein Commit, ohne die Äenderungen anzuwenden

```
git reset COMMIT
```

Zurücksetzen auf ein Commit ohne Löschen lokaler Änderungen

```
git reset –keep COMMIT
```

DATAMICS
machine intelligence consulting services

Git Merkblatt-2/2

Glossar

Versionskontrollsystem (Concurrent Version System, CVS)

Versionskontrollsysteme dienen der Verwaltung verschiedener Versionen einer Software. Bekannte CVS neben Git sind Subversion und CVS. Ein CVS speichert die gesamte Entwicklungshistorie einer Software und ermöglicht die Zusammenarbeitet mehrerer Entwickler in kollaborativen Projekten.

Repository (Ablage)

Ein Repository beinhaltet ein Projekt mitsamt seiner Entwicklungshistorie. Das heisst, alle Zwischenschritte (in Form von Commits) der Software können wiederhergestellt oder betrachtet werden. Im Fall von Git wird unterschieden zwischen einem meißt online verfügbaren Master-Repository und lokalen Klonen bei allen beteiligten Entwicklern, die so unabhängig voneinander an dem Projekt arbeiten können.

Working Tree (Arbeitsverzeichnis)

Der Working Tree beinhaltet alle Dateien des Projektes im derzeitigen Zustand und kann bei Bedarf vom CVS auf einen früheren Zustand zurückgesetzt werden.

Commit (Übergabe)

Ein Commit speichert den aktuellen Stand des Working Directories, also die jüngst vorgenommenen Modifikationen, zusammen mit einer beschreibenden Nachricht und weiteren Metadaten wie Datum, Uhrzeit und Autor als den nächsten Arbeitsschritt in der Historie.

Push (Stoßen)

Mit einem Push werden alle zuletzt vorgenommenen Commits dem Repository des Projektes hinzugefügt, entweder auf einem Server wie beispielsweise GitHub oder in einem lokalen Repository auf der Festplatte.

Clone (Klonen)

Der Befehl Clone erlaubt das Herunterladen eines gesamten Repositories auf einen Computer, auf dem ein Entwickler dann lokale Modifikationen vornehmen kann, die anschließend in das Master-Repository auf dem Server integriert werden.

Pull (Ziehen)

Durch Pull werden alle Änderungen, die im Master-Repository von anderen Entwicklern vorgenommen wurden, aber nicht im lokalen Klon enthalten sind, auf das lokale Repository angewendet.

Glossar

Branch (Zweig)

Bei der Entwicklung einer Software kann es erforderlich sein, verschiedene Versionen in der Form von Branches parallel zu entwickeln und zur Verfügung zu stellen. Beispielsweise kann ein Branch zum sicheren Experimentieren mit neuen Komponenten angelegt werden (häufig **dev** genannt), oder es können parallel Versionen für Python2 und Python3 angelegt werden.

HEAD (Kopf)

Ein Zeiger auf den aktuellen Commit des derzeitigen Branches. Üblicherweise der jüngste Commit des Branches, kann aber mit Git-Befehlen versetzt werden, um auf ältere Commits zuzugreifen.

Master (hier: Original)

Master bezeichnet den ursprünglich von Git angelegten, ersten Branch eines Repositories. Dieser kann bei Bedarf umbenannt werden und unterscheidet sich nicht von anderen Branches.

Tag (Markierung)

Tags sind vom Entwickler angelegte Markierungen, auf die einfach zugegriffen werden kann. Dadurch wird die Verwaltung der komplexen Entwicklungshistorie eines Projektes vereinfacht. Tags werden dazu mit Bezeichnern versehen, wie beispielsweise **v1.01** oder **Django statt Flask**.

Merge (Vereinigung)

Im üblichen Arbeitszyklus wird zuerst ein Pull ausgeführt, damit der lokale Klon des Repositories auf dem neusten Stand ist. Anschliessend werden lokale Modifikationen vorgenommen, die dann anschliessend mit einem Commit und Push in das Online-Repository integriert werden. Dabei wird ein Merge durchgeführt - die neuen Programmteile werden integriert. Wenn allerdings zwei Entwickler an den gleichen Programmteilen gleichzeitig lokal gearbeitet haben, kann dabei ein Merge-Konflikt auftreten, da das Online-Repository nicht weiß, welche Teile übernommen werden sollen. Üblicherweise müssen derartige Merge-Konflikte manuell aufgelöst werden.

SHA-1

Mit SHA (Secure Hash Algorithmus) werden Prüfsummen für digitale Informationen erstellt. Die Prüfsumme hat immer 40 Hexadezimale Zeichen. In Git wird SHA verwendet, um Commits zu benennen. Somit kann die Validität eines Commits anhand seines Namens überprüft werden.

Glossar

Index

Index bezeichnet die Vorbereitung des Working Directory vor einem Commit. Dabei wird ausgewählt, welche geänderten oder neu erstellten Datein dem Commit hinzugefügt werden.

Forking (Gabelung)

Forking bezeichnet das Klonen eines Online-Repositories (z.B. auf GitHub) für das eigene Benutzerkonto. Damit können Entwickler unabhängig an einem Projekt arbeiten, ohne Teil der ursprünglichen Entwicklergruppe zu sein, beispielsweise um die Software für ein anderes Betriebssystem zu portieren.

Blame (hier: Verantwortung)

Mit dem Befehl Blame wird für jede Zeile einer Datei angezeigt, wer wann die letzte Veränderung daran vorgenommen hat.

Collaborator (Mitarbeiter)

Alle Entwickler eines Repositories, die über Les- und Schreibrechte verfügen.

Contributor (Beitragender)

Ein Entwickler, dessen Pull Request in das Repository integriert wurde, der aber kein Collaborator ist.

Diff (Differenz)

Die Anzeige der Unterschiede zwischen zwei Dateien. Wird benutzt, um Bugs zwischen verschiedenen Commits aufzuspüren.

Issue (Angelegenheit)

Finden Nutzer eines Repositories einen Bug oder wünschen eine Erweiterung der Software, können sie auf GitHub eine Issue eingeben, die dann von den Entwicklern entweder bearbeitet oder zurückgewiesen wird.

Markdown

Markdown ist eine einfache Formatierungssprache für Textdateien, mit der häufig README-Dateien formatiert werden.

Upstream (Flussauf)

Synonym für das Master-Repository (meißt online auf GitHub). Der lokale Klon eines Repositories wird auch Downstream genannt.

Checkout (etwa: Wechsel)

Der Befehl zum Wechsel auf einen anderen Branch. HEAD und Index werden dabei auf den gewählten Branch gesetzt.

Glossar

Clean (sauber)

Ein Working Directory ist Clean, wenn keinerlei Abweichungen zum letzten Commit vorliegen, das heisst keine Dateien verändert wurden.

Parent (Elter)

Parent ist eine Liste mit allen logischen Vorgängern eines Commits.

Resolve (Auflösen)

Resolve bezeichnet das manuelle Auflösen von Merge-Konflikten.

Rewind (Zurückspulen)

Mit einem Rewind wird der HEAD auf einen früheren Commit zurückgesetzt und die Arbeit bis zu diesem Commit verworfen.

Rebase

Bei einem Rebase werden die Änderungen eines Branches auf einen anderen Branch angewendet, beispielsweise von einem experimentiellem Branch auf Master. Anschließend kann der experimentielle Branch in Master gemerged werden, wodurch im Gegensatz zu einem direkten Merge eine lineare Entwicklungshistorie ensteht.

Cherry-pick (etwa: Rosinen rauspicken)

Mit einem Cherry-pick wird ein spezifischer Commit eines Branches auf einen anderen Branch angewendet. Dies kann unter anderem zum Einbringen eines Commits in den Ziel-Branch genutzt werden, nachdem der Commit auf einen falschen Branch ausgeführt wurde.

Log

Log dient der Darstellung der Commit-Historie des aktuellen Branches.