Versionskontrolle: Git verwenden (Teil 2)

BBZW, Modul 426

Patrick Bucher

07.03.2024

Zur Erinnerung: Zustände und Zustandsübergänge

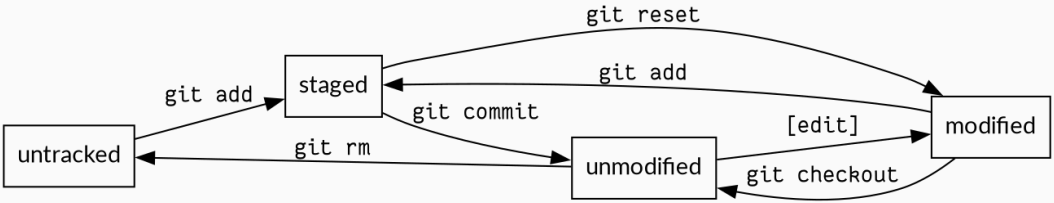

Abbildung 1: Lebenszyklus von Dateizuständen

Zustand zurücksetzen: `git reset`

Wurde eine Datei fälschlicherweise mit `git add` in den Staging-Bereich aufgenommen, kann sie mit `git reset` wieder daraus entfernt werden:

```
$ git add *
$ git status --short
A  params.exe
M  params.c
$ git reset params.exe
$ git status --short
M  params.c
?? params.exe
```

Vorsicht: Mit `git reset --hard` werden Änderungen auch im Arbeitsverzeichnis rückgängig gemacht (d.h. möglicherweise gelöscht)!

Änderungen verwerfen: `git checkout`

Wurden Änderungen im Arbeitsverzeichnis vorgenommen, die verworfen werden sollen, kann der Zustand mit `git checkout` wiederhergestellt werden:

```
$ git status --short
 M helloworld.c
 M params.c
$ git checkout -- helloworld.c
$ git status --short
 M params.c
```

Wichtig: `git checkout` arbeitet normalerweise mit Commit-Hashes. Mit `--` wird signalisiert, dass die Parameterliste fertig ist und nun Dateinamen folgen!

Zustand wiederherstellen: `git checkout`

Mithilfe von `git checkout` kann der frühere Zustand einer Datei oder des ganzen Repositories wiederhergestellt werden. Hierzu wird ein Commit Hash benötigt.

```
$ git checkout 7b6c08aa -- params.c
$ git status --short
M  params.c
```

Die Datei `params.c` wurde auf den Stand vom Commit `7b6c08aa` und im Staging-Bereich vorgemerkt. Ohne Angabe einer Datei werden alle Dateien zurückgesetzt:

```
$ git checkout 7b6c08aa
```

Mit `git checkout -` gelangt man zurück zum neuesten Stand.

Aufgabe 9 (5 Minuten, Einzelarbeit)

Verwenden Sie wieder das Repository `git-exercises`.

1. Nehmen Sie eine Änderung an einer Datei vor, fügen Sie diese Änderung dem Staging-Bereich hinzu (`git add`) und kontrollieren Sie dies mit `git status`.
2. Machen Sie diese Änderung nun mit `git reset` wieder rückgängig und kontrollieren Sie mit `git status`, ob das wie erwünscht funktioniert hat. (Die Änderung sollte nicht mehr gestaged sein.)
3. Machen Sie nun auch die Änderung am Arbeitsverzeichnis rückgängig, indem Sie `git checkout` darauf anwenden. Kontrollieren Sie das wiederum mit `git status`.
4. Nehmen Sie noch einmal eine Änderung an einer Datei vor und stagen Sie diese Änderung.
5. Verwenden Sie nun `git reset --hard`, um diese Änderung sowohl im Staging-Bereich als auch im Arbeitsverzeichnis rückgängig zu machen.

Dateien entfernen (I): `git rm`

```
$ git rm hello.c
$ rm params.c
$ git status --short
D  hello.c
 D params.c
```

- `rm` löscht eine Datei im Arbeitsverzeichnis.
- `git rm` löscht eine Datei im Arbeitsverzeichnis *und* merkt diese Löschung im Staging-Bereich vor.

`git rm` ist für Löschungen meistens vorzuziehen.

Dateien entfernen (II): `git rm -f` und `git rm --cached`

Dateien im Staging-Bereich können nicht einfach gelöscht werden:

```
$ git add newprog.c
$ git rm newprog.c
error: the following file has changes staged in the index:
newprog.c
(use --cached to keep the file, or -f to force removal)
```

Die Datei kann entweder aus dem Staging-Bereich entfernt werden:

```
$ git rm --cached newprog.c
```

Oder aus dem Staging-Bereich und dem Arbeitsverzeichnis gelöscht werden:

```
$ git rm -f newprog.c
```

Dateien umbenennen : `git mv`

Vorsicht beim Umbenennen und Verschieben von Dateien, die History kann verlorengehen:

```
$ mv hello.c helloworld.c
$ git status --short
 D hello.c
?? helloworld.c
```

Mit `git mv` wird die History beibehalten und die Umbenennung als solche festgehalten:

```
$ git mv hello.c helloworld.c
$ git status --short
R  hello.c -> helloworld.c
```

Aufgabe 10 (3 Minuten)

Führen Sie folgende Schritte auf, und halten Sie Ihre Antworten in der Datei `aufgabe-10.txt` fest, welche Sie im Repository ablegen.

1. Löschen Sie zwei Dateien: Einmal mit `rm` und einmal mit `git rm`. Was fällt Ihnen in der Ausgabe von `git status` auf?
2. Machen Sie nun beide Änderungen wieder rückgängig. Welche Befehle müssen Sie verwenden, um den Ursprungszustand wiederherstellen zu können?
3. Benennen Sie nun zwei Dateien um: Einmal mit `mv` und einmal mit `git mv`. Welche Variante ist vorzuziehen, und warum?

Versionsgeschichte einsehen (I): `git log`

Mit `git log` wird die Versionsgeschichte des Repositories angezeigt:

```
$ git log
commit aef74c545202548853bc09bf7afa1860f4262323
Author: Patrick Bucher <patrick.bucher@mailbox.org>
Date:   Sat Jul 10 11:34:23 2021 +0200

    new hello world program in C

commit e97e48b1067060cb77f743143c3a17fc082dd6cc
Author: Patrick Bucher <patrick.bucher@mailbox.org>
Date:   Sat Jul 10 11:34:04 2021 +0200

    initial commit: README
```

Versionsgeschichte einsehen (II): `git log` (Ausgabe)

Die Ausgabe enthält alle Commits (von oben nach unten: von neu zu alt).

Standardmässig werden folgende Informationen angezeigt:

- `commit`: der 40-stellige SHA-1 Hash des Commits
- `Author`: der Autor mit Name und E-Mail-Adresse
- `Date`: das Datum des Commits
- Kommentar: die Commit-Message (ein- oder mehrzeilig)

Versionsgeschichte einsehen (III): `git log` (Parameter)

Die Ausgabe kann mit verschiedenen Parametern gesteuert werden:

- `-[n]`: nur die neuesten `n` Commits anzeigen, z.B. `git log -3`
- `-p`: die Änderungen jedes Commits anzeigen
- `--stat`: Statistiken zu Änderungen anzeigen (Zeilen +/- pro Datei und total)
- `--pretty`: "schönere" Ausgabe, mit verschiedenen Varianten (`oneline`, `short`, `medium`, `full`, `fuller`, etc.), z.B. `git log --pretty=oneline`
  - benutzerdefiniertes Format: `--pretty=format:"%h %cn"`
  - siehe `git help log` (nach `--pretty` suchen)
- `--graph`: ASCII-Graph (v.a. bei Branches interessant)
- `--since`/`--after` und `--until`/`--before`: Eingrenzung auf Zeitraum
- `--name-only`: Zeigt Namen betroffener Dateien an

Grafische Alternative: `gitk` (`git help gitk`)

Aufgabe 11 (Einzelarbeit, 5 Minuten)

1. Führen Sie `git log` mit den Parametern von der vorherigen Folie aus. Sie können die Parameter auch miteinander kombinieren.
2. Denken Sie sich zwei, drei konkrete Anwendungsfälle für `git log` aus und finden Sie die passenden Kommandozeilenparameter, um den Anwedungsfall möglichst elegant abdecken zu können.
    - Dokumentieren Sie die Anwendungsfälle mit dem entsprechenden Befehl und den verwendeten Parametern in einer Datei `aufgabe-11.txt`.

Versionen markieren/benennen: `git tag`

Wichtige Zwischenstände können mittels `git tag` markiert werden (z.B. `v0.0.1`, siehe Semver.org):

```
$ git tag v0.0.1
```

Tags können mit annotiert werden, z.B. mit einer *Tag Message* (`-m`):

```
$ git tag -a v0.0.1 -m 'release with major bug fixes'
```

Tags können auch nachträglich für frühere Commits (hier: `e97e48`) erstellt werden:

```
$ git tag -a v0.0.0 e97e48 -m 'tag initial version'
```

Versionen anzeigen: `git tag`, `git show`

Bestehende Tags können mit `git tag` aufgelistet werden:

```
$ git tag
v0.0.1
v0.0.2
```

Informationen zu einem bestimmten Tag können mit `git show [tag]` angezeigt werden:

```
$ git show v0.0.2
tag v0.0.2
Tagger: Patrick Bucher <patrick.bucher@mailbox.org>
Date:   Sat Jul 10 16:43:00 2021 +0200

major bugfixes
```

Versionen teilen

Mit `git push` werden Tags nicht automatisch auf das Remote übertragen.

Ein einzelner Tag kann mit `git push [remote] [tag]` auf das Remote übertragen werden:

```
$ git push origin v0.0.1
```

Mit dem Parameter `--tags` werden *alle* lokalen Tags übertragen:

```
$ git push origin --tags
```

Aufgabe 12 (Einzelarbeit, 3 Minuten)

1. Pushen Sie den Zwischenstand Ihres Repositories auf den Server.
2. Erstellen Sie einen Tag namens `after-ex11`.
3. Pushen Sie diesen Tag nun auf den Server.
4. Verwenden Sie `git log`, um den Hash eines früheren Commits zu erhalten.
5. Erstellen Sie nun einen Tag für diesen früheren Commit und pushen Sie diesen ebenfalls auf den Server.

Branches

Bisher wurde nur *linear* gearbeitet.

In der Softwareentwicklung wird aber oft an mehreren Sachen gleichzeitig gearbeitet.

Mit *Branches* kann dem besser Rechnung getragen werden.

Beispiel: lange andauerndes *Refactoring*

Branch erstellen: `git checkout -b`

Ein Branch kann folgendermassen erzeugt und gleichzeitig aktiviert werden:

```
$ git checkout -b [name]
$ git checkout -b refactoring
```

Mit `git branch` kann der derzeit aktive Branch angezeigt werden:

```
$ git branch
  master
* refactoring
```

Branches wechseln und zusammenführen

Wurden die Änderung am Branch vorgenommen, kann mit `git checkout` wieder auf den ursprünglichen Branch gewechselt werden:

```
$ git checkout master
Switched to branch 'master'
```

Ein anderer Branch kann nun mit `git merge` mit dem derzeit aktiven Branch zusammengeführt werden:

```
$ git merge refactoring
```

Der Branch `master` enthält nun alle Änderungen aus dem Branch `refactoring`. (Jedoch nicht zwingend umgekehrt, falls `master` in der Zwischenzeit verändert worden ist.)

Aufgabe 13 (Einzelarbeit, 5 Minuten)

1. Erstellen Sie einen neuen Branch namens `experiment`.
2. Nehmen Sie eine Änderung daran vor, welche Sie auch committen.
3. Wechseln Sie nun in dem Branch `master` zurück.
4. Fügen Sie die Änderungen aus dem Branch `experiment` nun mit dem `master` zusammen.
5. Finden Sie den Commit auf `experiment` in der Ausgabe von `git log` wieder?

Abkürzungen definieren: `git config alias.*`

Befehle, die man häufig braucht, und die viel Tipparbeit erfordern, können mithilfe eines *Alias* abgekürzt werden.

Durch folgendes Alias wird `git log --pretty=oneline` durch `git lol` abgekürzt:

```
$ git config alias.lol 'log --pretty=oneline'
```

Was wesentlich weniger Tipparbeit erfordert:

```
$ git lol
```

Diese Änderungen gelten nur für das jeweilige Repository. Mit `--global` und `--system` können sie benutzer- oder systemweit gesetzt werden.

Aufgabe 14 (Einzelarbeit, 3 Minuten)

1. Definieren Sie ein eigenes Alias für einen der `git log`-Befehle von Aufgabe 11 und vergeben Sie einen passenden Namen.
2. Testen Sie diesen Befehl nun.

Befehle im Überblick (III)

- `git reset`: Datei(en) aus Staging-Bereich entfernen
- `git checkout`: Änderungen rückgängig machen
- `git rm`: Datei(en) entfernen
- `git mv`: Datei(en) umbenennen
- `git log`: Versionsgeschichte einsehen
- `gitk`: Versionsgeschichte in GUI anzeigen
- `git tag`: Versionen markieren/benennen
- `git checkout -b`: Branch erstellen und aktivieren
- `git branch`: Branches anzeigen und bearbeiten