Versionsgeschichte und Zeitmaschine

Die Abfolge der Commits betrachten

Nachdem wir einige Commits gemacht haben, können wir unser Repository untersuchen:

```
frank@pike:~/tagebuch$ git log

commit 022bdbc3574686f655aaadfefc04a79cf6b5a1a8 (HEAD -> main)
Author: Frank Schiebel <codeberg@ua25.de>
Date:   Mon Sep 23 17:32:56 2024 +0200

    Mittagessen hinzugefügt

commit a5c28ad5b5b39b1cb862743fd7cfec75d1269b02
Author: Frank Schiebel <codeberg@ua25.de>
Date:   Mon Sep 23 17:31:34 2024 +0200

    fruehstueck.txt geändert

commit 2c70b7517bcf0217c62b93336de038f166225c6a
Author: Frank Schiebel <codeberg@ua25.de>
Date:   Sun Oct 29 20:32:50 2023 +0100

    Fruestück

commit 9ee8f8bfdd6c532fee7d693c9d4431e22f455f0d
Author: Frank Schiebel <codeberg@ua25.de>
Date:   Sun Oct 29 20:14:11 2023 +0100

    Aufstehen!
```

Man sieht – recht ausführlich – einige Informationen zu jedem Zeitpunkt, an dem ein Snapshot der unter Versionskontrolle stehenden Dateien in unserem Arbeitsverzeichnis erstellt wurde:

- Die Hashsumme des Commits (die ersten 7 Zeichen identifizieren einen Commit eindeutig)
- Der Autor
- Das Datum, an dem der Snapshot erstellt wurde

Das ganz benötigt recht viel Platz, man kann die Ausgabe des Logs umfangreich anpassen, zum Beispiel so:

```
frank@pike:~/tagebuch$ git log --graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %Creset' --abbrev-commit
* 022bdbc - (HEAD -> main) Mittagessen hinzugefügt (vor 2 Stunden)
* a5c28ad - fruehstueck.txt geändert (vor 2 Stunden)
* 2c70b75 - Fruestück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Das sieht zwar nett aus iist aber ein wenig schwer zu merken, git ermöglicht es darum, für solche individuellen Befehle soganennte **Aliase** anzulegen, auf folgende Weise kann man einen eigenen "git-Befehl" anlegen, der die Versionsgeschichte hübsch ausgibt.

```
frank@pike:~/tagebuch$ git config alias.lg "log --graph --
pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %Creset'
--abbrev-commit"
frank@pike:~/tagebuch$ git lg
* 022bdbc - (HEAD -> main) Mittagessen hinzugefügt (vor 2 Stunden)
* a5c28ad - fruehstueck.txt geändert (vor 2 Stunden)
* 2c70b75 - Fruestück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Zeitmaschine

Go Back in Time - es gibt offene Fragen...

> **Wichtig**
>
> Zunächst ist es sehr empfehlenswert, dass man sich nur dann zu einem älteren Stand der Dateien zurückkehrt, wenn das Arbeitsverzeichnis **keine nicht committeten Änderungen** beinhaltet. ("Sauberes Arbeitsverzeichnis")

Wir befinden uns aktuell also in einem sauberen Arbeitsverzeichnis, `git status` zeigt also keine unbearbeitetn Änderungen:

```
frank@pike:~/tagebuch$ git status
Auf Branch main
nichts zu committen, Arbeitsverzeichnis unverändert
```

Jetzt können wir mit unserem neuen Alias `git lg` die Versionsgeschichte ansehen:

```
frank@pike:~/tagebuch$ git lg
* 022bdbc - (HEAD -> main) Mittagessen hinzugefügt (vor 16 Stunden)
* a5c28ad - fruehstueck.txt geändert (vor 16 Stunden)
* 2c70b75 - Fruestück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Man kann den Zustand des Arbeitsverzeichnisses aus den Commits wieder herstellen. Es ist also möglich, das Arbeitsverzeichnis in genau den Zustand zurückzuversetzen, in dem es beim Commit `2c70b75 - Fruestück` vor 11 Monaten war – oder eben zu jedem anderen Zeitpunkt, an dem ich zuvor den Zustand des Arbeitsverzeichnisses als Snapshot commitet habe. Der Befehl dazu ist `git checkout <Commit-Hash>`:

```
frank@pike:~/tagebuch$ git checkout 2c70b75
```

```
Hinweis: Wechsle zu '2c70b75'.

Sie befinden sich im Zustand eines 'losgelösten HEAD'. Sie können sich
umschauen, experimentelle Änderungen vornehmen und diese committen, und
Sie können alle möglichen Commits, die Sie in diesem Zustand machen,
ohne Auswirkungen auf irgendeinen Branch verwerfen, indem Sie zu einem
anderen Branch wechseln.

Wenn Sie einen neuen Branch erstellen möchten, um Ihre erstellten Commits
zu behalten, können Sie das (jetzt oder später) durch Nutzung von
'switch' mit der Option -c tun. Beispiel:

  git switch -c <neuer-Branchname>

Oder um diese Operation rückgängig zu machen:
  git switch -

Sie können diesen Hinweis ausschalten, indem Sie die Konfigurationsvariable
'advice.detachedHead' auf 'false' setzen.

HEAD ist jetzt bei 2c70b75 Fruestück
```

Was ist denn jetzt passiert? Was bedeutet diese komische Meldung? Was ist ein `lösgelöster HEAD`[1)]?

(1) Zunächst einmal befinden sich die Dateien im Verzeichnis jetzt in dem Zustand, in dem sie waren, als der Commit `2c70b75` erstellt wurde. Überprüfe das!

Lösung

Man erkennt, dass die Datei mitagessen.txt noch nicht vorhanden war, und auch der Inhalt der anderen Dateien entspricht dem Stand zu dem Zeitpunkt als der Commit gemacht wurde.

```
frank@pike:~/tagebuch$ ls
aufstehen.txt  fruehstueck.txt  mittagessen.txt
frank@pike:~/tagebuch$ git checkout 2c70b75
Hinweis: Wechsle zu '2c70b75'.
[... Meldunden zum losgelösten HEAD ausgelassen ...]
HEAD ist jetzt bei 2c70b75 Fruestück
frank@pike:~/tagebuch$ ls
aufstehen.txt  fruehstueck.txt
```

(2) Untersuche die Versionsgeschichte – wie sieht diese jetzt aus? Was fällt auf?

Lösung

```
frank@pike:~/tagebuch$ git lg
* 2c70b75 - (HEAD) Fruestück (vor 11 Monaten)
```

```
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Es fällt auf, dass die Commits die neuer sind als `2c70b75` irgendwie "verschwunden sind"??!!

Git zeigt mit `git log` standardmäßg die Versionsgeschichte an, die zu dem Commit geführt hatm den man im Arbeitsverzeichnis gerade angezeigt bekommt. Um git zu veranlassen, die **gesamte** Versionsgeschichte anzuzeigen, muss man dem Befehl `git log` die Option `--all` mitgeben.

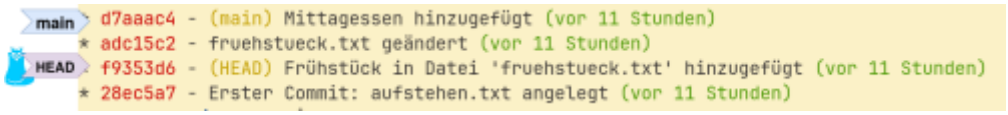
```
frank@pike:~/tagebuch$ git lg --all
* 022bdbc - (main) Mittagessen hinzugefügt (vor 16 Stunden)
* a5c28ad - fruehstueck.txt geändert (vor 16 Stunden)
* 2c70b75 - (HEAD) Fruestück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Was ist HEAD?

Man muss sich die Versionsgeschichte als (verzweigte) Abfolge von Commits vorstellen, die jeweils durch ihre Commit-ID identifiziert werden. Manche Commit-IDs bekommen Namen: **Diese benannten Commits sind "Branches" und "Tags".* Man spricht von "Referenzen auf Commits".

Eine besondere Referenz ist HEAD. HEAD springt in gewisser Weise in der Versiongeschichte herum, und zeigt immer auf denjenigen Commit, der im Arbeitsverzeichniss gerade "ausgecheckt" ist.

Der Zustand in unserem Beispiel ist "in bunt" folgender:



Es gibt insgesamt also 4 Commits, das Arbeitsverzeichnis ist im Zustand von Commit `2c70b75` darum zeigt HEAD auf diesem Commit. Außerdem gibt es den Branch `main`, das ist eine benannte Referenz auf den neuesten Commit mit der ID `022bdbc`

Um die Situation des "Lösgelösten HEAD" zu verstehen, muss man sich nun klar machen, dass der Commit `a5c28ad` auf dem Commit `2c70b75` (auf den derzeit HEAD zeigt) basiert. Wenn man im derzeitigen Zustand des Arbeitsverzeichnisses etwas ändert, bekomme man ein Problem, da diese Beziehung `a5c28ad` basiert auf `2c70b75` nicht mehr eindeutig interpretierbar ist: Wo sortiert sich das ausgehend von `2c70b75` veränderte Arbeitsverzeichnis in die Versionsgeschichte ein?

Diese Situation wird als "detached HEAD" oder losgelöster HEAD bezeichnet. Man kann sich umsehen und Dinge ändern, wenn man anschließend jedoch wieder in der Versiongeschichte "springt" gehen diese Änderungen verloren oder landen in einem Commit der "losgelöst" ist, also nicht in einem Branch innerhalb der Versionsgeschichte enthalten ist - und das sollte man vermeiden, da man solche Commits später nur sehr schwer wiederfinden kann.

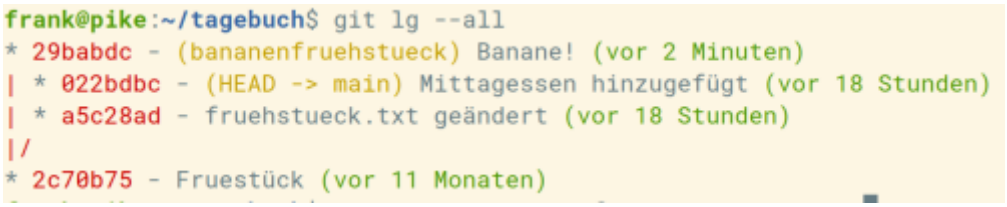
(A1)

- Wechsle mit dem Befehl `git checkout <Commit-ID>` zu unterschiedlichen Versionen deines Tagebuchs und schau dich um, ohne etwas zu ändern. Lasse dir jeweils die vollständige Versionsgeschichte anzeigen und beobachte, wie der HEAD in der Versionsgeschichte springt.
- Wechsle zur Commit ID des neuesten Commits. Wechsle dann zum Branch `main` indem du `main` auscheckst. Verhält sich git in beiden Fällen gleich?
- Gehe in der Zeit zurück zu einem älteren Commit. Verändere eine Datei. Versuche wieder in der Zeit zu springen. Was passiert?
- Mit der Option `-f` kann man git zwingen, einen Checkout zu machen, bei dem Änderungen verloren gehen. Springe von deinem geänderten Zeitpunkt unter Datenverlust zurück zu `main` und dann wieder zu dem Commit den du zuvor geändert hattest. Überprüfe, dass deine Änderungen tatsächlich verloren gegangen sind.
- Wir wissen ja, dass man sich Änderungen am einfachsten merkt, wenn man einen Commit erstellt. Ändere erneut eine Datei und erstelle einen neuen Commit, springe dann zurück zu `main`. Was passiert jetzt? Warum ist das ungünstig?

Demonstration der Aufgabenstellung

https://tube.schule.social/w/noCdL4HaygxW9Z9YCFU5du

Mit dem letzten Befehl `git branch bananenfruehstueck 29babdc` im Video wurde ein neuer Branch angelegt. In der Ausgabe von `git log --all` kann man das erkennen - es gibt im Zeitverlauf jetzt eine Verzweigung.



1)

Auf Englisch `detached HEAD`