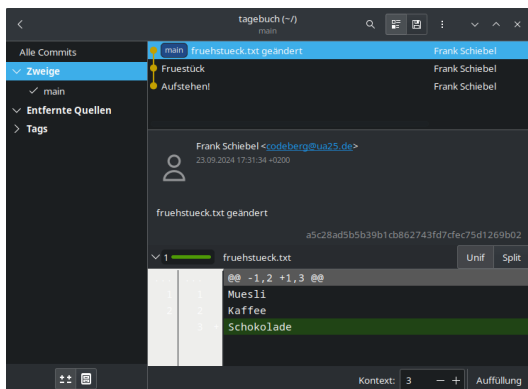


Szenario: Zurück in der Zeit und nochmal versuchen

Ein Szenario das man gelegentlich hat, ist, dass man in der Zeit zurückgehen möchte, dann aber einen Fehler macht, den man gerne korrigieren möchte.

Noch einmal das Tagebuch

Wir starten mit dem folgenden Zustand - im Commit a5c28ad wurde dem Frühstück Schokolade hinzugefügt.



```
frank@pike:~/tagebuch$ git lg
* a5c28ad - (HEAD -> main) fruehstueck.txt geändert (vor 22 Stunden)
* 2c70b75 - Fruestück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Wir wollen nun zu dem Zeitpunkt zurückgehen, zu dem das Frühstück nur aus zwei Speisen bestand, um dort anstelle der Schokolade eine Banane zu verspeisen.

Dazu gibt es mehrere Möglichkeiten, die verschiedene Vor- und Nachteile haben.

Einfach zurück und neu starten - bisherige Änderungen gehen verloren

Dieses Vorgehen ist eine der (wenigen) Möglichkeiten in git tatsächlich **Informationen unwiederbringlich zu verlieren**. Man sollte sich also wirklich sicher sein, dass man die Änderungen, die man nach dem Zeitpunkt, zu dem man zurückkehren möchte nicht mehr benötigt.

Zur Verdeutlichung dieses Situation habe ich jetzt noch eine weitere neue Datei angelegt - diese heißt wieder mittagessen.txt, steht aber noch nicht unter Versionsverwaltung.

```
frank@pike:~/tagebuch$ git status
Auf Branch main
Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
```

mittagessen.txt

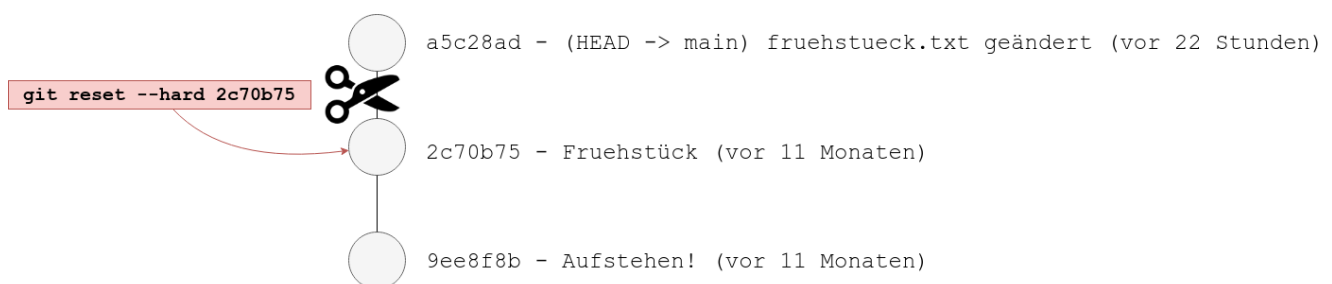
```
nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
frank@pike:~/tagebuch$ git lg
* a5c28ad - (HEAD -> main) fruehstueck.txt geändert (vor 23 Stunden)
* 2c70b75 - Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Mit dem Befehl `git reset --hard 2c70b75` bringt man alle Dateien im Verzeichnis, die unter Versionskontrolle stehen auf den Stand, die diese zum Zeitpunkt des Commits 2c70b75 hatten. Alle späteren Änderungen und die spätere Versionsgeschichte gehen unwiderruflich verloren! Dateien, die *nicht* unter Versionskontrolle stehen, werden nicht beeinflusst.

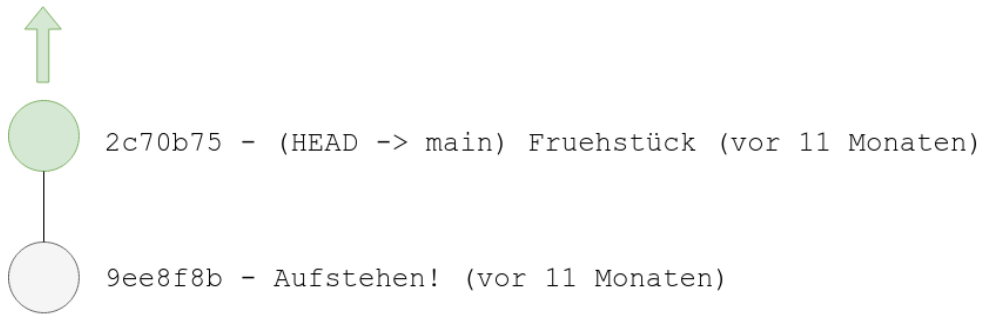
Das sieht dann so aus:

```
frank@pike:~/tagebuch$ git reset --hard 2c70b75
HEAD ist jetzt bei 2c70b75 Frühstück
frank@pike:~/tagebuch$ ls
aufstehen.txt fruehstueck.txt mittagessen.txt
frank@pike:~/tagebuch$ git lg
* 2c70b75 - (HEAD -> main) Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
frank@pike:~/tagebuch$ git lg --all
* 2c70b75 - (HEAD -> main) Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

- Die Dateien `aufstehen.txt` und `fruehstueck.txt` sehen jetzt so aus, wie sie zum Zeitpunkt des Commits aussahen
- Die Datei `mittagessen.txt` ist unverändert, weil sie nicht unter Versionskontrolle stand
- `git log` zeigt auch mit der Option `--all` die Commits, die nach 2c70b75 gemacht wurden nicht mehr an - diese Daten sind verloren.



Jetzt kann man sein Tagebuch ausgehend von 2c70b75 weiter führen, hat das alternative Frühstück mit Schokolade aber verloren.



Zurück und mit einem weiteren Branch weiterarbeiten - die bisherigen Änderung bleiben erhalten

Die Situation zu Beginn der Operation ist nun wieder so, dass wir drei Commits haben - im letzten wurde dem Frühstück Schokolade hinzugefügt.

```
frank@pike:~/tagebuch$ git lg
* a5c28ad - (HEAD -> main) fruehstueck.txt geändert (vor 23 Stunden)
* 2c70b75 - Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Schritt 1 - Checkout des letzten Commits, der beibehalten werden soll

Mit `git checkout 2c70b75` gelangt man zum Frühstück ohne Schokolade. Git warnt, dass man sich jetzt im Zustand des losgelösten HEAD befindet, gibt aber auch Ratschläge, wie man das beheben kann, wenn man Änderungen, die man jetzt macht behalten möchte. Wir erstellen also wie angeraten mit dem Befehl `git switch -c Neuer_Versuch` einen Branch mit dem Namen `Neuer_Versuch`:

```
frank@pike:~/tagebuch$ git checkout 2c70b75
Hinweis: Wechsle zu '2c70b75'.
[ ... Meldung zum losgelösten HEAD ... ]
HEAD ist jetzt bei 2c70b75 Frühstück

frank@pike:~/tagebuch$ git switch -c Neuer_Versuch
Zu neuem Branch 'Neuer_Versuch' gewechselt

frank@pike:~/tagebuch$ git lg --all
* a5c28ad - (main) fruehstueck.txt geändert (vor 23 Stunden)
* 2c70b75 - (HEAD -> Neuer_Versuch) Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Schritt 2 - Im neuen Branch weiterarbeiten

Jetzt kann man die Banane einfügen und einen weiteren Commit erstellen:

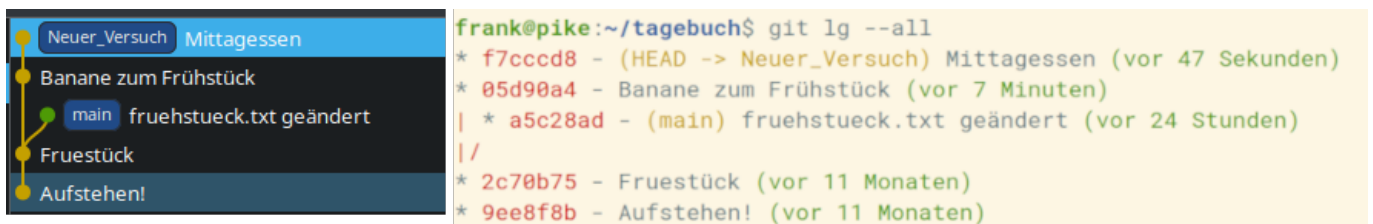
```
frank@pike:~/tagebuch$ vi fruehstueck.txt

frank@pike:~/tagebuch$ git add fruehstueck.txt

frank@pike:~/tagebuch$ git commit -m "Banane zum Frühstück"
[Neuer_Versuch 05d90a4] Banane zum Frühstück
1 file changed, 1 insertion(+)

frank@pike:~/tagebuch$ git lg --all
* 05d90a4 - (HEAD -> Neuer_Versuch) Banane zum Frühstück (vor 3 Sekunden)
| * a5c28ad - (main) fruehstueck.txt geändert (vor 24 Stunden)
|/
* 2c70b75 - Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Man sieht, dass HEAD auf den Branch Neuer_Versuch zeigt - wir arbeiten also derzeit nicht mehr auf main. Das macht zunächst nichts aus, wenn sich die neuen Änderungen bewähren, möchte man aber eventuell den neuen Branch als neuen main Branch verwenden - wie geht das?



The image shows a Git log output and a corresponding branch diagram. The log output is as follows:

```
frank@pike:~/tagebuch$ git lg --all
* f7cccd8 - (HEAD -> Neuer_Versuch) Mittagessen (vor 47 Sekunden)
* 05d90a4 - Banane zum Frühstück (vor 7 Minuten)
| * a5c28ad - (main) fruehstueck.txt geändert (vor 24 Stunden)
|/
* 2c70b75 - Frühstück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

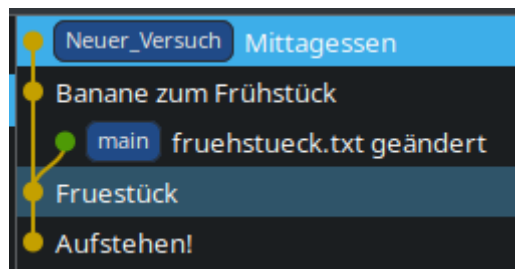
The branch diagram on the left shows the following structure:

- A vertical line of commits: Aufstehen! (yellow dot), Frühstück (yellow dot), main fruehstueck.txt geändert (blue dot), Banane zum Frühstück (yellow dot), and Mittagessen (yellow dot).
- A branch named 'Neuer_Versuch' (blue box) is shown at the top, pointing to the 'Mittagessen' commit.
- The 'main' branch (blue box) is shown pointing to the 'main fruehstueck.txt geändert' commit.

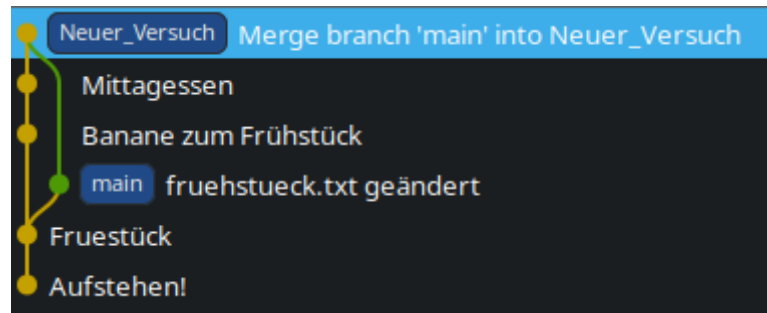
Schritt 3: Den neuen Branch zum main-Branch machen

Um zu erreichen, dass der neue Branch zum Hauptentwicklungszweig main wird, ohne die Informationen des "Fehlversuchs" zu verlieren, kann man folgendermaßen vorgehen:

- Man stellt sicher, dass man sich auf dem neuen Branch befindet: `git checkout Neuer_Versuch`, sehr wahrscheinlich erhält man die Meldung, dass man sich bereits auf dem neuen Branch befindet.
- Die Situation ist dann folgendermaßen - wir arbeiten auf dem Branch Neuer_Versuch:



- Nun führt man einen sogenannten "Merge" durch, der den main Branch mit unserem "Neuen_Versuch" zusammenführt. Weil wir bei dieser Zusammenführung unsere Änderungen auf jeden Fall behalten wollen, verwenden wir als "Zusammenführungsstrategie" die Option `-s strategy=ours`, also "unsere" Änderungen gewinnen immer. Der gesamte Befehl sieht dann so aus: `git merge--strategy=ours main`, Anschließend sieht das Repo so aus:



Man sieht, dass die Branches jetzt zwar zusammengeführt sind, aber main zeigt noch immer auf den alten Commit.

From:
<https://info-bw.de/> -

Permanent link:
<https://info-bw.de/faecher:informatik:oberstufe:git:neuanfang:start?rev=1727192455>

Last update: **24.09.2024 15:40**

