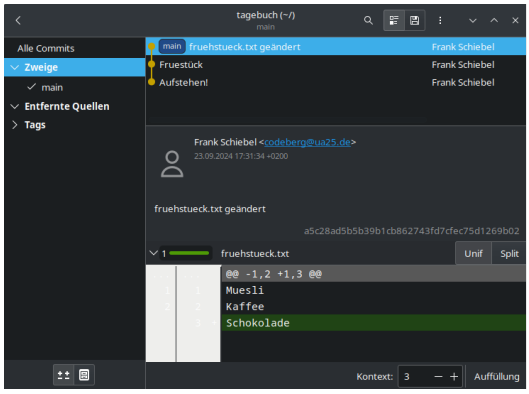
Szenario: Zurück in der Zeit und nochmal versuchen

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, die sich bereits in einem Commit befinden, 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)
```

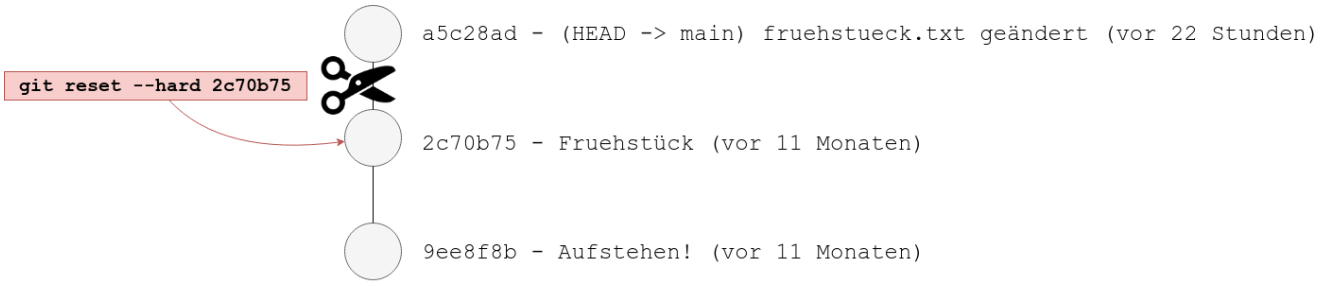
```
    mittagessen.txt

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 - Fruestück (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.

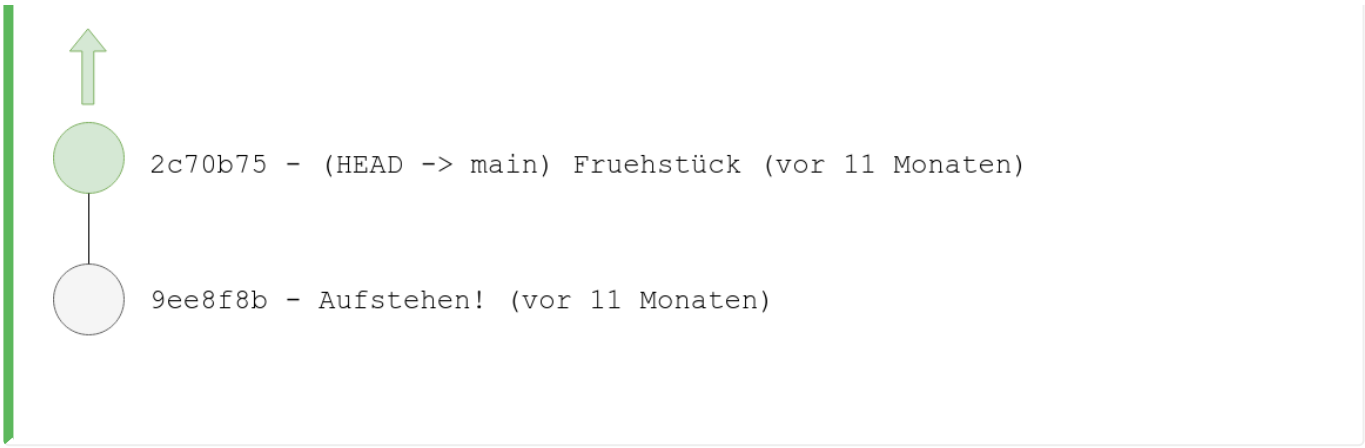
Das sieht dann so aus:

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

- Die Dateien `aufstehen.txt` und `fruehstueck.txt` sehen jetzt so aus, wie sie zum Zeitpunkt es 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 - Fruestück (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 lösgelösten 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 Fruestück

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

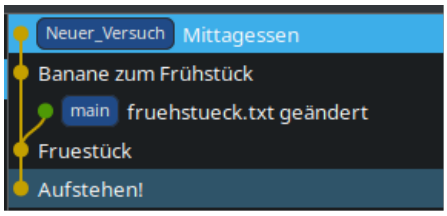
Schritt 2 - Im neuen Branch weiterarbeiten

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 - Fruestück (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?

Anmerkung: Man kann selbstverständlich auch einfach auf dem neuen Branch weiter arbeiten und den nächsten Schritt auslassen. Wenn man das Vorgehen dann aber später wiederholt, weil man wieder in der Zeit zurückgehen möchte bekommt man jedes mal einen neuen Branch, das ist "unschön".

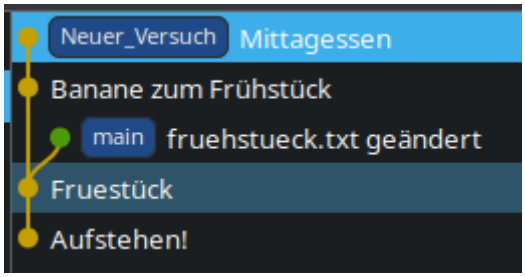



```
frank@pike:~/tagebuch$ git lg --all
* f7cccd8 - (HEAD -> Neuer_Versuch) Mittagessen (vor 47 Sekunden)
* 05d90a4 - Banane zum Frühstück (vor 7 Minuten)
| * a5c28ad - (main) fruehstueck.txt geändert (vor 24 Stunden)
|/
* 2c70b75 - Fruestück (vor 11 Monaten)
* 9ee8f8b - Aufstehen! (vor 11 Monaten)
```

Schritt 3: Den neuen Branch zum main-Branch machen

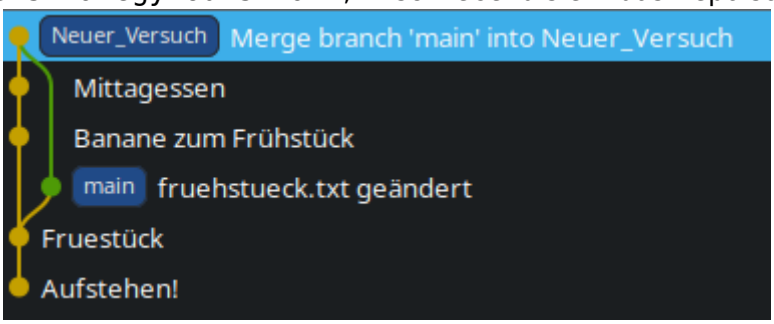
Um zu erreichen, dass der neue Branch zum Hauptentwicklungszweig `main` wird, ohne die Informationen des "Fehlversuchs" zu verlieren, kann man folgendermaßen vorgehen:

- 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`:



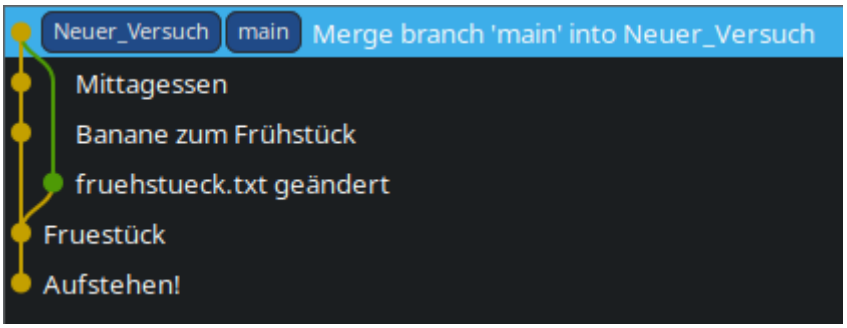
- Nun führt man einen sogenannten "Merge" durch, der den `main` Branch mit unserem

"Neuen_Versuch" zusammenführt. Weil wir bei dieser ZUsammenführung unsere Änderungen auf jeden Fall behalten wollen, verwenden wir als "Zusammenführungsstrategie" die Option `--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:

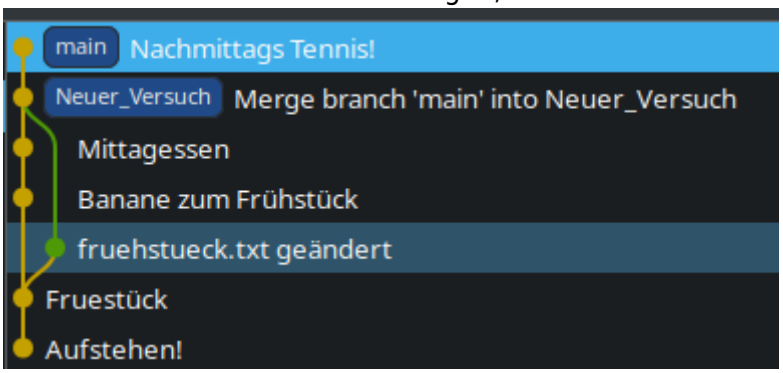


Man sieht, dass die Branches jetzt zwar zusammengeführt sind, aber `main` zeigt noch immer auf den alten Commit.

- Um das zu beheben checkt man jetzt `main` aus und führt dann den neuen Branch mit main zusammen: `git checkout main` – jetzt ist HEAD auf `main`. Jetzt mit `git merge Nuer_Versuch` den neuen Branch nach `main` zusammenführen, dann sieht das Ergebnis so aus:



- Wenn man jetzt neue Commits anfügt, finden diese im `main` Branch statt. Der Commit `a5c28ad5b`, bei dem es Schokolade zum Frühstück gab, bleibt erhalten.



From:
https://www.info-bw.de/ -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:git:neuanfang:start?rev=1727849257

Last update: **02.10.2024 06:07**