Interactive Rebase meistern – Commits aufräumen wie ein Profi □□

Der **Interactive Rebase** ist eines der mächtigsten Werkzeuge in Git, um deine Commit-Historie nachträglich zu bearbeiten. Stell dir vor, du hättest eine Zeitmaschine für deine Commits: Du kannst sie umbenennen, zusammenfassen, aufteilen, neu ordnen oder komplett entfernen. Das Ergebnis ist eine saubere, verständliche Historie, die anderen (und deinem zukünftigen Ich) das Leben leichter macht.

Was ist Interactive Rebase und wann nutze ich ihn?

Beim normalen `git rebase` wird dein Branch auf einen anderen Basis-Commit „umgepflanzt". Der **Interactive Rebase** (`git rebase -i`) geht einen Schritt weiter: Er öffnet einen Editor, in dem du für *jeden einzelnen Commit* entscheiden kannst, was damit passieren soll.

Typische Anwendungsfälle:

- Du hast mehrere kleine „WIP"-Commits gemacht und möchtest sie vor dem Push zu einem sauberen Commit zusammenfassen
- Eine Commit-Message enthält einen Tippfehler oder ist nicht aussagekräftig genug
- Du hast versehentlich etwas committed, das nicht in die Historie gehört
- Die Reihenfolge deiner Commits ergibt logisch keinen Sinn
- Du möchtest einen großen Commit nachträglich in kleinere, thematisch getrennte Commits aufteilen

> ⚠ **Wichtig:** Interactive Rebase *schreibt die Historie um*. Das bedeutet, dass alle betroffenen Commits neue SHA-Hashes bekommen. **Nutze Interactive Rebase nur für Commits, die du noch nicht gepusht hast**, oder sei dir der

> Konsequenzen bewusst (Force-Push erforderlich, Probleme für andere Team-Mitglieder).

Die Grundlagen: Interactive Rebase auf der Kommandozeile

Bevor wir zu PhpStorm kommen, ist es wichtig, die Grundlagen auf der Kommandozeile zu verstehen. Das hilft dir, zu verstehen, was PhpStorm im Hintergrund macht.

Den Interactive Rebase starten

Um die letzten *n* Commits zu bearbeiten, verwendest du:

```
git rebase -i HEAD~n
```

Zum Beispiel, um die letzten 4 Commits zu bearbeiten:

```
git rebase -i HEAD~4
```

Alternativ kannst du auch einen spezifischen Commit-Hash angeben – dann werden alle Commits *nach* diesem Commit bearbeitet:

```
git rebase -i abc1234
```

Was passiert nach dem Start?

Git öffnet deinen konfigurierten Editor (z.B. Vim, VS Code, oder den von PhpStorm) mit einer Liste aller betroffenen Commits. Diese Liste sieht ungefähr so aus:

```
pick a1b2c3d Feature: Login-Formular hinzugefügt
pick e4f5g6h WIP: Validierung angefangen
pick i7j8k9l Tippfehler korrigiert
pick m0n1o2p Validierung fertiggestellt

# Rebase abc1234..m0n1o2p onto abc1234 (4 commands)
```

```
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's message
# d, drop = remove commit
```

Wichtig zu verstehen: Die Commits werden in *chronologischer Reihenfolge* angezeigt (ältester oben, neuester unten) – das ist umgekehrt zur Anzeige in `git log`!

Die sechs Rebase-Befehle im Detail

1. `pick` (p) – Commit unverändert übernehmen

```
pick a1b2c3d Feature: Login-Formular hinzugefügt
```

Was es macht: Der Commit wird genau so übernommen, wie er ist – gleicher Inhalt, gleiche Message. Das ist die Standardoption.

Wann verwenden: Wenn ein Commit bereits perfekt ist und keine Änderung braucht.

2. `reword` (r) – Commit-Message ändern ✏

```
reword a1b2c3d Feature: Login-Formular hinzugefügt
```

Was es macht: Der Commit-Inhalt bleibt unverändert, aber nach dem Speichern der Rebase-Liste öffnet Git einen Editor, in dem du die Commit-Message bearbeiten kannst.

Wann verwenden:

- Tippfehler in der Commit-Message korrigieren

- Eine aussagekräftigere Beschreibung nachträglich hinzufügen
- Ticket-Nummern oder Issue-Referenzen ergänzen

Beispiel-Workflow:

1. Du änderst `pick` zu `reword` in der Liste
2. Du speicherst und schließt die Liste
3. Git öffnet einen neuen Editor mit der alten Commit-Message
4. Du bearbeitest die Message und speicherst
5. Git setzt den Rebase fort

3. `edit` (e) – Commit anhalten und bearbeiten □□

```
edit a1b2c3d Feature: Login-Formular hinzugefügt
```

Was es macht: Git wendet den Commit an und *pausiert dann*. Du befindest dich in einem Zustand, in dem du:

- Dateien ändern kannst
- Weitere Commits hinzufügen kannst
- Den bestehenden Commit mit `git commit --amend` modifizieren kannst
- Den Commit sogar in mehrere Commits aufteilen kannst

Nach deinen Änderungen setzt du den Rebase mit `git rebase --continue` fort.

Wann verwenden:

- Einen großen Commit in mehrere kleinere aufteilen
- Eine vergessene Datei nachträglich zu einem Commit hinzufügen
- Eine Datei aus einem Commit entfernen (aber behalten)

Beispiel: Einen Commit in zwei aufteilen:

```
# Nach dem Pausieren beim edit-Commit:

# Schritt 1: Den Commit "aufmachen", aber Änderungen behalten
git reset HEAD~1

# Schritt 2: Selektiv committen
git add src/Login/Form.php
git commit -m "Feature: Login-Formular HTML-Struktur"
```

```
git add src/Login/Validation.php
git commit -m "Feature: Login-Formular Validierung"

# Schritt 3: Rebase fortsetzen
git rebase --continue
```

4. `squash` (s) – Mit vorherigem Commit verschmelzen □□

```
pick a1b2c3d Feature: Login-Formular hinzugefügt
squash e4f5g6h WIP: Validierung angefangen
squash i7j8k9l Validierung fertiggestellt
```

Was es macht: Der Commit wird mit dem *vorherigen* Commit (dem darüber in der Liste) verschmolzen. Die Änderungen beider Commits werden kombiniert. Git öffnet dann einen Editor, in dem du eine neue, kombinierte Commit-Message schreiben kannst – dabei siehst du die Messages aller beteiligten Commits.

Wann verwenden:

- Mehrere „Work in Progress"-Commits zu einem sauberen Commit zusammenfassen
- Commits, die logisch zusammengehören, vereinen
- Die Historie vor dem Push aufräumen

Der kombinierte Message-Editor sieht so aus:

```
# This is a combination of 3 commits.
# This is the 1st commit message:

Feature: Login-Formular hinzugefügt

# This is the commit message #2:

WIP: Validierung angefangen

# This is the commit message #3:

Validierung fertiggestellt
```

```
# Please enter the commit message for your changes.
```

Du löschst oder kommentierst die nicht benötigten Zeilen aus und schreibst eine saubere, neue Message:

```
Feature: Login-Formular mit Validierung hinzugefügt


- Formular-HTML-Struktur erstellt
- Client-seitige Validierung implementiert
- Server-seitige Validierung hinzugefügt
```

5. `fixup` (f) – Verschmelzen ohne Message □□

```
pick a1b2c3d Feature: Login-Formular hinzugefügt
fixup e4f5g6h WIP
fixup i7j8k9l Tippfehler
```

Was es macht: Genau wie `squash`, aber die Commit-Message des fixup-Commits wird *automatisch verworfen*. Es wird nur die Message des Basis-Commits (des `pick`-Commits darüber) verwendet. Kein Editor öffnet sich für die Message-Bearbeitung.

Wann verwenden:

- Kleine Korrekturen, die zu einem vorherigen Commit gehören
- „Oops, das habe ich vergessen"-Commits
- Commits mit wenig aussagekräftigen Messages wie „WIP", „fix", „typo"

Tipp: Du kannst im Voraus Commits erstellen, die für Fixup gedacht sind, mit:

```
git commit --fixup=abc1234
```

Git erstellt dann automatisch einen Commit mit der Message `fixup! <Original-Message>`. Später kannst du mit `git rebase -i --autosquash` diese Fixup-Commits automatisch an die richtige Stelle sortieren lassen!

6. `drop` (d) – Commit komplett entfernen □□

```
pick a1b2c3d Feature: Login-Formular hinzugefügt
drop e4f5g6h Debug-Code versehentlich committed
pick i7j8k9l Validierung fertiggestellt
```

Was es macht: Der Commit wird komplett aus der Historie entfernt. Die Änderungen dieses Commits sind danach *nicht mehr vorhanden* – weder die Dateien noch die Commit-Message.

Wann verwenden:

- Ein Commit enthält Testcode oder Debug-Ausgaben, die nicht in die Historie gehören
- Ein Commit war ein Fehler und sollte nie existiert haben
- Du möchtest Rauschen aus der Historie entfernen

Alternativ: Du kannst auch einfach die Zeile mit dem Commit aus der Liste löschen – das hat denselben Effekt wie `drop`.

> ⚠ **Vorsicht:** Wenn spätere Commits von den Änderungen des gelöschten Commits abhängen, wirst du Konflikte bekommen!

Commits umsortieren

Eine oft übersehene Funktion: Du kannst die Reihenfolge der Zeilen in der Rebase-Liste einfach ändern, um Commits umzusortieren!

Vorher:

```
pick a1b2c3d Feature A
pick e4f5g6h Feature C
pick i7j8k9l Feature B
```

Nachher:

```
pick a1b2c3d Feature A
pick i7j8k9l Feature B
pick e4f5g6h Feature C
```

Wann verwenden:

- Commits in eine logischere Reihenfolge bringen

- Verwandte Commits zusammenbringen, bevor du sie squashst
- Die Historie für Code-Reviews verständlicher machen

> ⚠ **Vorsicht:** Wenn Commits voneinander abhängen (Commit B ändert Code, den Commit A eingeführt hat), kann das Umsortieren zu Konflikten führen.

Übersicht: Alle Befehle auf einen Blick

Befehl	Kurzform	Änderungen	Message	Wann verwenden
`pick`	`p`	behalten	behalten	Commit ist perfekt
`reword`	`r`	behalten	ändern	Message korrigieren
`edit`	`e`	pausieren & ändern	ändern möglich	Commit aufteilen/erweitern
`squash`	`s`	mit vorherigem vereinen	kombinieren	Commits zusammenfassen
`fixup`	`f`	mit vorherigem vereinen	verwerfen	Kleine Korrekturen
`drop`	`d`	entfernen	entfernen	Commit löschen

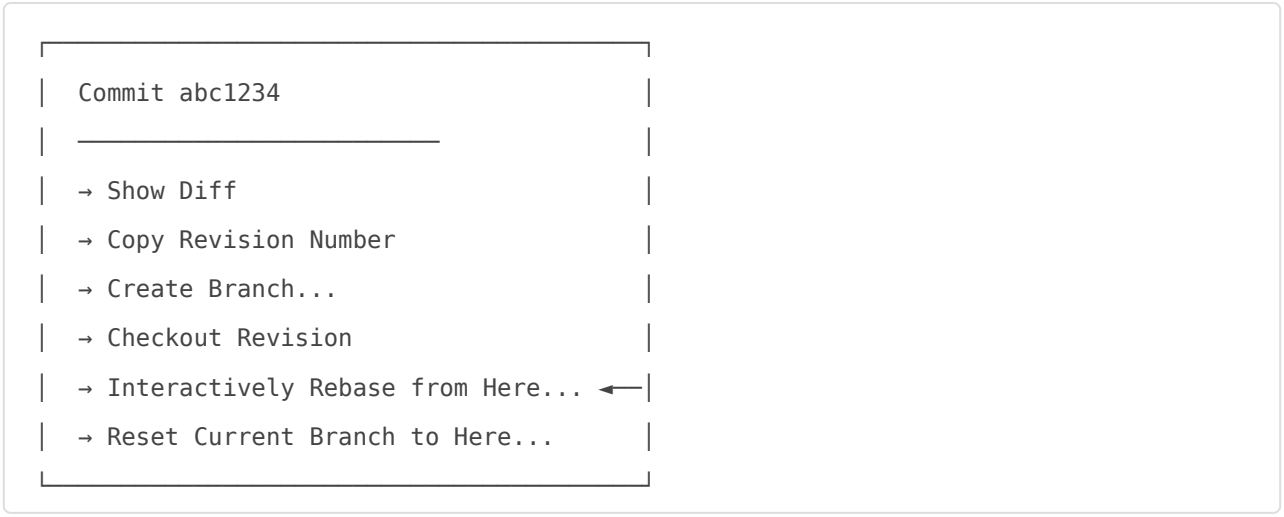
Interactive Rebase in PhpStorm □□

PhpStorm bietet eine komfortable grafische Oberfläche für Interactive Rebase, die besonders für Einsteiger viel zugänglicher ist als die Kommandozeile.

Methode 1: Über das Git-Log-Fenster

1. **Git-Log öffnen:** Gehe zu **View → Tool Windows → Git** oder klicke unten auf den Reiter „Git"
2. **Den Basis-Commit finden:** Scrolle im Log zu dem Commit, *ab dem* du die Historie bearbeiten möchtest (der Commit selbst wird nicht bearbeitet, nur alle danach)
3. **Kontextmenü öffnen:** Rechtsklick auf diesen Commit

4. **Rebase starten:** Wähle **„Interactively Rebase from Here..."**



Methode 2: Über das Hauptmenü

1. Gehe zu **Git → Rebase...**
2. Wähle im Dialog die Option **„Interactive"**
3. Wähle den Basis-Commit oder Branch aus

Methode 3: Über den aktuellen Branch

1. Klicke unten rechts auf den Branch-Namen in der Statusleiste
2. Rechtsklick auf deinen aktuellen Branch
3. Wähle **„Rebase Current onto Selected..."** und dann die Interactive-Option

Der PhpStorm Interactive Rebase Dialog

Nach dem Starten öffnet sich ein übersichtlicher Dialog:

```
│  │ fixup   ▼ │ i7j8k9l         │ typo                          │   │
│  │ pick    ▼ │ m0n1o2p         │ Validierung fertig            │   │
│  └───────────┴─────────────────┴───────────────────────────────┘   │
│                                                                    │
│  [↑ Move Up]  [↓ Move Down]  [Show Diff]                           │
│                                                                    │
├────────────────────────────────────────────────────────────────────┤
│                              [Cancel]  [Start Rebasing]            │
└────────────────────────────────────────────────────────────────────┘
```

Aktionen in PhpStorm ausführen

Action ändern: Klicke auf das Dropdown-Menü in der „Action"-Spalte neben jedem Commit. Du siehst alle verfügbaren Optionen:

- pick
- edit
- reword (wird in PhpStorm manchmal als „Edit Message" bezeichnet)
- squash
- fixup
- drop (oder „Skip")

Commits umsortieren: Wähle einen Commit aus und nutze die Buttons **„Move Up"** und **„Move Down"** oder ziehe die Commits per Drag & Drop an die gewünschte Position

Commit-Diff ansehen: Wähle einen Commit und klicke auf **„Show Diff"**, um zu sehen, welche Änderungen dieser Commit enthält – sehr hilfreich, um zu entscheiden, was damit passieren soll

Commit-Message vorab bearbeiten: Bei `reword` kannst du in manchen PhpStorm-Versionen die neue Message direkt im Dialog eingeben, ohne dass später ein separater Editor öffnet

Schritt-für-Schritt: Commits in PhpStorm squashen

Hier ein konkretes Beispiel – du hast diese Commits und möchtest sie aufräumen:

```
pick a1b2c3d Feature: User-Profil-Seite erstellt
pick e4f5g6h WIP
pick i7j8k9l noch mehr WIP
pick m0n1o2p Profil-Seite fertig
```

Ziel: Alle vier Commits zu einem einzigen, sauberen Commit zusammenfassen.

1. **Rebase starten:** Rechtsklick auf den Commit *vor* `a1b2c3d` → „Interactively Rebase from Here..."
2. **Actions konfigurieren:**

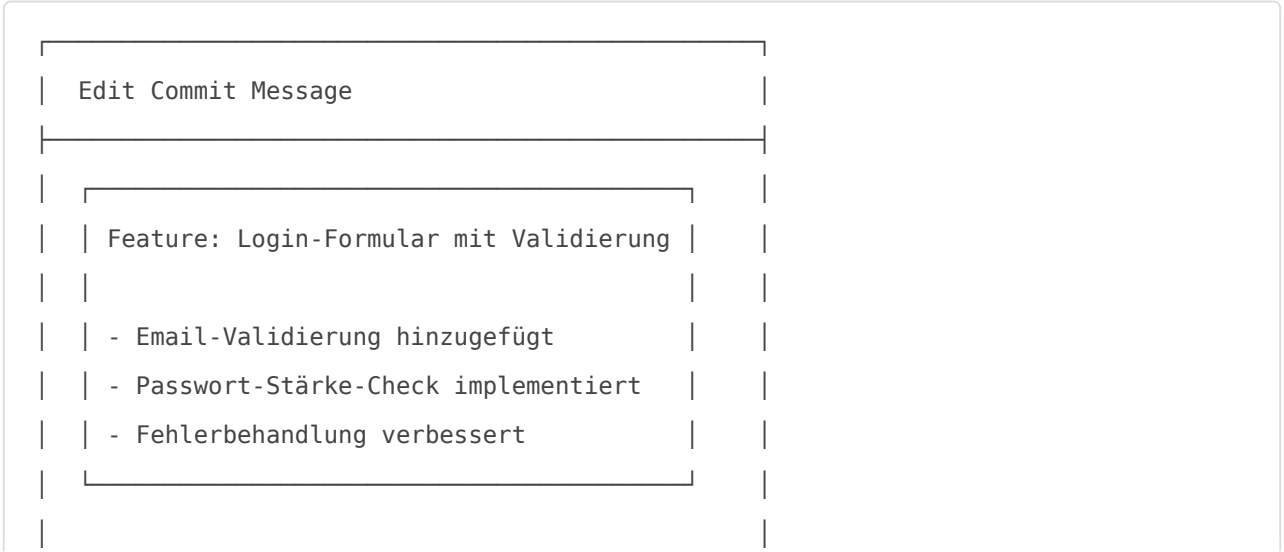
Commit	Aktion	Warum
a1b2c3d	`pick`	Basis-Commit, behält die Message
e4f5g6h	`fixup`	Soll zu a1b2c3d hinzugefügt werden, Message verwerfen
i7j8k9l	`fixup`	Soll auch hinzugefügt werden, Message verwerfen
m0n1o2p	`fixup`	Letzter Teil, Message verwerfen

3. **„Start Rebasing" klicken**
4. **Ergebnis:** Ein einziger Commit `a1b2c3d` mit der Message „Feature: User-Profil-Seite erstellt", der alle Änderungen enthält

Wenn du `squash` statt `fixup` verwendest: PhpStorm öffnet nach dem Rebase einen Editor, in dem du eine neue, kombinierte Message schreiben kannst.

Schritt-für-Schritt: Commit-Message in PhpStorm ändern

1. **Rebase starten**
2. **Action auf `reword` setzen:** Ändere die Action für den gewünschten Commit zu „reword"
3. **„Start Rebasing" klicken**
4. **Message bearbeiten:** PhpStorm öffnet einen Commit-Message-Editor für jeden `reword`-Commit



```
│                    [Cancel]  [Save Message]      │
└──────────────────────────────────────────────────┘
```

5. **Speichern:** Klicke auf „Save Message" oder drücke Ctrl+Enter

Den `edit`-Befehl in PhpStorm nutzen

Der `edit`-Befehl ist etwas spezieller, weil er den Rebase *pausiert* und dir Kontrolle gibt:

1. **Action auf `edit` setzen und Rebase starten**
2. **PhpStorm-Benachrichtigung:** Du siehst eine Notification:

   > „Interactive rebase stopped. You can amend the commit or continue rebasing."

3. **Änderungen vornehmen:**
   - Dateien bearbeiten
   - Neue Dateien zur Staging Area hinzufügen
   - `Commit` (ohne Push!) um den aktuellen Commit zu erweitern
   - Oder im Terminal: `git commit --amend`
4. **Rebase fortsetzen:**
   - Über die Notification: Klicke auf **„Continue Rebasing"**
   - Über das Menü: **Git → Continue Rebasing**
   - Im Terminal: `git rebase --continue`

Konflikte während des Interactive Rebase

Beim Umsortieren oder Squashen kann es zu Konflikten kommen, besonders wenn Commits voneinander abhängen.

Konflikte erkennen

PhpStorm zeigt dir eine klare Meldung:

```
┌──────────────────────────────────────────────────┐
│  ⚠ Rebasing conflict                             │
```

```
┌──────────────────────────────────────────────────────────┐
│                                                          │
│  Conflicts in the following files:                       │
│  • src/Login/Form.php                                    │
│  • src/Login/Validation.php                              │
│                                                          │
│  [Resolve Conflicts]  [Abort Rebase]                     │
└──────────────────────────────────────────────────────────┘
```

Konflikte lösen

1. **„Resolve Conflicts" klicken:** Der Merge-Dialog öffnet sich
2. **3-Way-Merge nutzen:** PhpStorm zeigt dir:
   - **Left (Yours/HEAD):** Der aktuelle Stand nach den bisherigen Rebase-Schritten
   - **Right (Theirs):** Die Änderungen aus dem Commit, der gerade angewendet wird
   - **Center (Result):** Das Ergebnis, das du zusammenbaust
3. **Konflikte einzeln lösen:** Nutze die Buttons „Accept Left", „Accept Right" oder bearbeite manuell
4. **Alle Konflikte gelöst:** Klicke auf **„Apply"**
5. **Rebase fortsetzen:** Klicke auf **„Continue Rebasing"** in der Notification

Rebase abbrechen

Wenn du merkst, dass der Rebase in eine Sackgasse führt:

- **In PhpStorm:** Klicke auf „Abort Rebase" oder gehe zu **Git → Abort Rebasing**
- **Im Terminal:** `git rebase --abort`

Das stellt den exakten Zustand vor dem Rebase wieder her – als wäre nichts passiert.

Praktisches Beispiel: Kompletter Workflow

Hier ein realistisches Szenario, das alles zusammenbringt:

Ausgangssituation

Du hast an einem Feature gearbeitet und folgende Commits erstellt:

```
1. abc1234 - WIP: Angefangen mit der API
2. def5678 - API Endpunkt erstellt
3. ghi9012 - FIXME: Debug-Ausgaben drin!!!
4. jkl3456 - API fertig, aber Typo in Message
5. mno7890 - Dokumentation hinzugefügt
```

Ziel

Bevor du pushst, möchtest du:

- Die ersten zwei Commits zusammenfassen
- Den Debug-Commit entfernen
- Den Typo in Commit 4 korrigieren
- Die Dokumentation nach oben verschieben (vor die API-Commits)

Schritt für Schritt in PhpStorm

1. **Rebase starten:** Rechtsklick auf den Commit *vor* abc1234 → „Interactively Rebase from Here..."
2. **Dialog konfigurieren:**

Original	Aktion	Neue Position	Ergebnis
mno7890	pick	↑ nach oben verschoben	bleibt
abc1234	pick	—	wird Basis für squash
def5678	squash	—	verschmilzt mit abc1234
ghi9012	drop	—	wird entfernt
jkl3456	reword	—	Message wird korrigiert

   Die neu sortierte Liste sieht so aus:

   ```
   pick    mno7890  Dokumentation hinzugefügt
   pick    abc1234  WIP: Angefangen mit der API
   squash  def5678  API Endpunkt erstellt
   reword  jkl3456  API fertig, aber Typo in Message
   ```

3. **„Start Rebasing" klicken**
4. **Squash-Message bearbeiten:** PhpStorm öffnet einen Editor für die kombinierte Message:

```
Feature: REST-API implementiert


- Endpunkt /api/users erstellt
- Authentifizierung hinzugefügt
- Response-Format definiert
```

5. **Reword-Message bearbeiten:** Danach öffnet sich der Editor für den Typo-Fix
6. **Ergebnis:** Saubere Historie mit drei aussagekräftigen Commits:

```
1. Dokumentation hinzugefügt
2. Feature: REST-API implementiert
3. API-Tests und finale Korrekturen
```

Best Practices für Interactive Rebase □□

Vor dem Rebase:

- Stelle sicher, dass dein Working Directory „clean" ist (keine uncommitteten Änderungen)
- Erstelle ggf. einen Backup-Branch: `git branch backup-vor-rebase`
- Nutze Interactive Rebase *vor* dem Push, nicht danach

Während des Rebase:

- Arbeite von oben nach unten durch die Liste
- Squashe immer in den älteren Commit (der oben steht)
- Bei Unsicherheit: lieber abbrechen und neu starten
- Nutze `edit` sparsam – er ist mächtig, aber auch komplex

Commit-Messages nach dem Squash:

- Schreibe eine neue, zusammenfassende Message
- Lösche die alten WIP-Messages komplett
- Folge dem Conventional Commits Format, wenn dein Team es nutzt

Wenn etwas schiefgeht:

- `git rebase --abort` ist dein Freund
- Das Reflog hat alles gespeichert: `git reflog` zeigt dir alle Zustände
- Mit `git reset --hard HEAD@{n}` kannst du zu jedem vorherigen Zustand zurück

Zusammenfassung

Interactive Rebase ist ein unverzichtbares Werkzeug für professionelle Git-Nutzung. Mit den sechs Befehlen `pick`, `reword`, `edit`, `squash`, `fixup` und `drop` kannst du deine Commit-Historie perfekt aufräumen, bevor du sie mit der Welt teilst. PhpStorm macht diesen Prozess durch seine grafische Oberfläche besonders zugänglich – du siehst auf einen Blick alle Commits, kannst Actions per Dropdown wählen und Commits per Drag & Drop umsortieren.

Die wichtigsten Takeaways:

- □□ **Nutze Interactive Rebase nur für lokale, noch nicht gepushte Commits**
- □□ **Squash/Fixup** sind deine Werkzeuge für saubere, atomare Commits
- ✏ **Reword** ermöglicht nachträgliche Message-Korrekturen ohne Aufwand
- □□ **Drop** entfernt ungewollte Commits komplett aus der Historie
- □□ **Edit** gibt dir volle Kontrolle, ist aber auch am komplexesten
- ↩ **Im Zweifel:** `git rebase --abort` und von vorne beginnen

Revision #1
Created 2026-06-29 15:43:07 UTC by art10m
Updated 2026-06-29 15:44:48 UTC by art10m