Kapitel 8: Fehler rückgängig machen □□

Einleitung: Fehler passieren – und genau dafür ist Git gemacht! Die Fähigkeit, Änderungen rückgängig zu machen, ist einer der größten Vorteile der Versionsverwaltung. Allerdings gibt es verschiedene Arten von „Rückgängig", abhängig davon, in welchem Stadium sich deine Änderungen befinden: noch nicht gestaged, gestaged aber nicht committed, committed aber nicht gepusht, oder bereits gepusht. Jede Situation erfordert einen anderen Ansatz. Besonders heikel wird es, wenn du versehentlich sensible Daten wie Passwörter committed hast – hier reicht einfaches Löschen nicht aus, da Git die Historie aufbewahrt. Dieses Kapitel gibt dir das Werkzeug, um aus jeder Situation wieder herauszukommen.

- Änderungen rückgängig machen in PhpStorm – vor dem Commit □□
- Commits rückgängig machen: reset vs. revert □□
- Sensible Daten aus Git entfernen – warum einfaches Löschen nicht reicht □□

Änderungen rückgängig machen in PhpStorm – vor dem Commit □□

Du hast an einer oder mehreren Dateien gearbeitet, aber die Änderungen gefallen dir nicht oder du hast dich verrannt? Kein Problem – solange du noch **nicht committed** hast, kannst du deine Dateien ganz einfach auf den letzten gespeicherten Stand im Repository zurücksetzen. PhpStorm bietet dafür eine komfortable Funktion namens **Rollback**.

Das Konzept verstehen

Wenn du Änderungen an Dateien vornimmst, befinden sich diese zunächst im **Working Directory** – also deinem Arbeitsbereich. Git weiß, dass die Dateien verändert wurden (sie erscheinen als „modified"), aber diese Änderungen sind noch nicht Teil der Versionshistorie.

Der Befehl, den du im Terminal verwenden würdest, heißt `git checkout -- <datei>` oder in neueren Git-Versionen `git restore <datei>`. In PhpStorm musst du dir diese Befehle aber nicht merken – du nutzt einfach die **Rollback**-Funktion.

```
flowchart LR
    A["Datei im Working Directory\ngeaendert"] -->|Rollback| B["Datei zurueckgesetzt\nauf letzten Commit-Stand"]
    A -->|Ohne Rollback| C["Aenderung bleibt\nbis zum naechsten Commit"]
```

> ⚠ **Wichtig:** Wenn du eine Datei zurücksetzt, sind die Änderungen **unwiderruflich verloren** – es gibt kein „Undo" für den Rollback selbst. Überlege also kurz, ob du die Änderungen wirklich verwerfen möchtest.

Eine einzelne Datei zurücksetzen

Wenn du nur eine bestimmte Datei auf den letzten Commit-Stand zurücksetzen möchtest, hast du in PhpStorm mehrere Möglichkeiten:

Über das Kontextmenü im Projektbaum

1. Navigiere im **Project-Panel** (links) zu der Datei, deren Änderungen du verwerfen möchtest. Geänderte Dateien sind farblich markiert – typischerweise in **Blau** für modifizierte Dateien.
2. **Rechtsklick** auf die Datei und wähle im Kontextmenü **Git → Rollback…** (in älteren Versionen eventuell unter **Local History → Revert**).
3. Es öffnet sich ein Dialog, der dir die Änderungen anzeigt, die zurückgesetzt werden. Bestätige mit **Rollback**.

Über das Commit-Fenster

1. Öffne das **Commit-Tool-Fenster** mit `Ctrl + K` (Windows/Linux) bzw. `Cmd + K` (macOS) oder über **Git → Commit…**
2. Im Commit-Fenster siehst du links eine Liste aller geänderten Dateien. **Rechtsklick** auf die Datei, die du zurücksetzen möchtest.
3. Wähle **Rollback…** aus dem Kontextmenü.
4. Bestätige den Dialog – die Datei wird auf den Stand des letzten Commits zurückgesetzt.

Direkt im Editor

Wenn du die betreffende Datei gerade geöffnet hast, kannst du auch über das Menü gehen:

1. Gehe zu **Git → Rollback…** (oder **VCS → Git → Rollback…** in älteren Versionen)
2. PhpStorm zeigt dir einen Dialog mit allen geänderten Dateien. Wähle nur die Datei aus, die du zurücksetzen möchtest (andere Häkchen entfernen).
3. Klicke auf **Rollback**.

Alle Änderungen auf einmal zurücksetzen

Manchmal möchtest du einen kompletten „Reset" machen und **alle** nicht-committeten Änderungen verwerfen. Auch das ist in PhpStorm schnell erledigt:

1. Öffne das Commit-Fenster mit `Ctrl + K` / `Cmd + K`.
2. Du siehst die Liste aller geänderten Dateien. **Markiere alle Dateien**, die du zurücksetzen möchtest (mit `Ctrl + A` / `Cmd + A` kannst du alle auswählen).
3. **Rechtsklick** auf die Auswahl und wähle **Rollback…**
4. Bestätige den Dialog – alle ausgewählten Dateien werden auf den letzten Commit-Stand zurückgesetzt.

Alternativ kannst du auch direkt über das Menü gehen:

1. Gehe zu **Git → Rollback…**
2. Im Dialog sind standardmäßig alle geänderten Dateien ausgewählt
3. Klicke auf **Rollback**, um alles zurückzusetzen

Was ist mit bereits gestageten Dateien?

Wenn du Dateien bereits zur **Staging Area** hinzugefügt hast (also mit `git add` oder in PhpStorm durch Anhaken im Commit-Dialog), funktioniert der Rollback trotzdem – PhpStorm entfernt die Datei aus der Staging Area **und** setzt sie auf den letzten Commit-Stand zurück.

Falls du eine Datei nur aus der Staging Area entfernen möchtest, aber die Änderungen im Working Directory **behalten** willst, ist das ein anderer Fall: Dann entfernst du einfach das Häkchen bei der Datei im Commit-Dialog, ohne einen Rollback zu machen.

Zusammenfassung der Schritte

Was du tun möchtest	So geht's in PhpStorm
Eine Datei zurücksetzen	Rechtsklick auf Datei → Git → Rollback…
Alle Änderungen zurücksetzen	`Ctrl/Cmd + K` → alle auswählen → Rechtsklick → Rollback…
Aus Staging entfernen (Änderungen behalten)	Im Commit-Dialog das Häkchen bei der Datei entfernen

---

Der Unterschied zu „Revert Commit"

Verwechsle **Rollback** nicht mit **Revert**:

- **Rollback** verwirft Änderungen, die du gemacht hast, aber **noch nicht committed** hast
- **Revert** erstellt einen neuen Commit, der die Änderungen eines **bereits existierenden Commits** rückgängig macht

Wenn du also Änderungen rückgängig machen willst, die du schon committed hast, brauchst du einen anderen Ansatz – aber das ist ein Thema für ein anderes Kapitel. □□

Commits rückgängig machen: reset vs. revert □□

Du hast also einen oder mehrere Commits gemacht, die du loswerden möchtest – vielleicht war ein Fehler dabei, oder du hast dich in eine falsche Richtung entwickelt. Git bietet dafür zwei grundlegend verschiedene Strategien: **Reset** und **Revert**. Welche du wählen solltest, hängt entscheidend davon ab, ob du die Commits **bereits gepusht** hast oder nicht.

Das Grundprinzip verstehen

Bevor wir in die Details gehen, hier die Kernfrage, die du dir immer stellen solltest:

> **Haben andere Menschen (oder ich selbst auf einem anderen Gerät) bereits Zugriff auf diese Commits?**

Diese Frage bestimmt, welchen Weg du gehen solltest:

```
flowchart TD
    A["Commit rueckgaengig machen?"] --> B{"Bereits gepusht?"}
    B -->|Nein| C["git reset\nHistorie umschreiben"]
    B -->|Ja| D["git revert\nNeuen Commit erstellen"]

    C --> E["Commits verschwinden\naus der Historie"]
    D --> F["Alter Commit bleibt,\nwird durch neuen rueckgaengig gemacht"]

    style C fill:#90EE90,color:#000000
    style D fill:#87CEEB,color:#000000
```

Methode 1: `git reset` – die Historie umschreiben ✂

Mit `reset` **entfernst** du Commits aus deiner lokalen Historie, als hätten sie nie existiert. Das ist sauber und elegant – aber nur, solange niemand sonst diese Commits bereits hat.

Die drei Varianten von Reset

Git reset gibt es in drei „Härtegraden", die bestimmen, was mit deinen Änderungen passiert:

1. `--soft` – der sanfte Reset
   Die Commits werden entfernt, aber alle Änderungen bleiben in der **Staging Area** erhalten. Du kannst sie direkt neu committen (z. B. mit einer besseren Nachricht oder anders aufgeteilt).

   ```
   git reset --soft HEAD~1    # Letzten Commit entfernen, Änderungen bleiben staged
   ```

2. `--mixed` (Standard) – der mittlere Weg
   Die Commits werden entfernt, die Änderungen landen im **Working Directory** (also nicht mehr staged, aber noch vorhanden). Du kannst entscheiden, was du davon behalten möchtest.

   ```
   git reset HEAD~1           # Letzten Commit entfernen, Änderungen im Working
   Directory
   git reset --mixed HEAD~1   # Identisch zur Zeile darüber
   ```

3. `--hard` – der radikale Reset ⚠
   Die Commits werden entfernt **und** alle Änderungen werden **unwiderruflich gelöscht**. Hier ist Vorsicht geboten!

   ```
   git reset --hard HEAD~1    # Letzten Commit UND alle Änderungen komplett löschen
   ```

Was bedeutet `HEAD~1`?

- `HEAD` zeigt auf deinen aktuellen Commit
- `HEAD~1` bedeutet „ein Commit vor HEAD"
- `HEAD~3` würde drei Commits zurückgehen
- Du kannst auch einen konkreten Commit-Hash angeben: `git reset --soft abc1234`

Reset in PhpStorm durchführen

1. Öffne das **Git-Log** (unten im Git-Tab)
2. Finde den Commit, zu dem du **zurückkehren** möchtest (also den letzten „guten" Commit)
3. **Rechtsklick** auf diesen Commit
4. Wähle **Reset Current Branch to Here...**
5. Im Dialog wählst du zwischen **Soft**, **Mixed** oder **Hard**

> □□ **Tipp:** In PhpStorm siehst du direkt eine Erklärung zu jeder Option, was sehr hilfreich ist.

Methode 2: `git revert` – sicher rückgängig machen □□

Mit `revert` erstellst du einen **neuen Commit**, der die Änderungen eines früheren Commits rückgängig macht. Die ursprünglichen Commits bleiben in der Historie erhalten – es wird nur ein „Gegengift" hinzugefügt.

Warum Revert statt Reset bei gepushten Commits?

Wenn du Commits gepusht hast und dann mit `reset` deine lokale Historie umschreibst, entsteht ein Problem: Dein lokales Repository und das Remote-Repository haben dann **unterschiedliche Historien**. Git wird sich weigern, deinen Push zu akzeptieren, und du müsstest einen **Force Push** machen – was die Historie für alle anderen zerstört, die vielleicht schon mit diesen Commits arbeiten.

`revert` vermeidet dieses Problem elegant:

```
flowchart LR
    subgraph "Nach Reset - GEFÄHRLICH"
        A1["Commit A"] --> A2["Commit B"] --> A3["Commit C"]
        A1 -.->|"reset"| A4["Commit C\nverschwunden"]
```

```
    end

    subgraph "Nach Revert - SICHER"
        B1["Commit A"] --> B2["Commit B"] --> B3["Commit C"] --> B4["Revert C\nmacht C
rueckgaengig"]
    end
```

Revert im Terminal

```
# Den letzten Commit rückgängig machen
git revert HEAD

# Einen bestimmten Commit rückgängig machen (per Hash)
git revert abc1234

# Mehrere Commits auf einmal revert (ältester..neuester)
git revert abc1234..def5678
```

Nach dem Revert öffnet sich normalerweise ein Editor für die Commit-Nachricht. Git schlägt automatisch etwas wie „Revert ‚Feature XY hinzugefügt'" vor, was du anpassen oder übernehmen kannst.

Revert in PhpStorm durchführen

1. Öffne das **Git-Log**
2. Finde den Commit, den du rückgängig machen möchtest
3. **Rechtsklick** auf den Commit
4. Wähle **Revert Commit**
5. PhpStorm erstellt automatisch einen neuen Commit, der die Änderungen umkehrt

---

Wann welche Methode? – Die Entscheidungshilfe

Situation	Empfohlene Methode	Begründung

Commit noch nicht gepusht, du möchtest neu committen	`reset --soft`	Änderungen bleiben staged, du kannst sie direkt verbessern
Commit noch nicht gepusht, du möchtest alles überdenken	`reset --mixed`	Änderungen im Working Directory, du wählst neu aus
Commit noch nicht gepusht, Änderungen sollen weg	`reset --hard`	Schnell und sauber, aber **unwiderruflich**
Commit **bereits gepusht**	`revert`	Sicher, keine Probleme mit Remote
Alte Commits mitten in der Historie korrigieren	`revert`	Historie bleibt intakt
Gemeinsames Projekt mit anderen	**Immer** `revert`	Teamkollegen werden es dir danken

---

Der Sonderfall: Force Push ⚠

Falls du **trotzdem** nach einem Reset pushen möchtest (weil du z. B. allein am Projekt arbeitest und sicher bist, dass niemand sonst betroffen ist), kannst du einen **Force Push** machen:

```
git push --force
# oder etwas sicherer:
git push --force-with-lease
```

In **PhpStorm** findest du diese Option im Push-Dialog unter **Force Push**.

> ⚠ **Wichtig:** Force Push überschreibt die Remote-Historie unwiderruflich. Nutze das nur, wenn du **absolut sicher** bist, dass niemand anderes mit diesen Commits arbeitet. In Team-Projekten ist Force Push auf den Main-Branch oft sogar verboten.

Die Option `--force-with-lease` ist etwas sicherer: Sie verweigert den Push, falls jemand anderes in der Zwischenzeit gepusht hat.

Praktisches Beispiel: Der typische „Oh nein"-Moment

Szenario: Du hast gerade einen Commit gemacht und gepusht, der versehentlich Debug-Code enthält.

1. **Prüfen, was passiert ist:**
   Schau im Git-Log, welcher Commit das Problem verursacht hat (notiere dir den Hash, z. B. `a1b2c3d`).
2. **Revert durchführen:**
   ```
   git revert a1b2c3d
   ```
   Oder in PhpStorm: Rechtsklick auf den Commit → **Revert Commit**
3. **Den Revert-Commit pushen:**
   ```
   git push
   ```
4. **Fertig!** Die Debug-Änderungen sind rückgängig gemacht, und die Historie zeigt transparent, was passiert ist.

Zusammenfassung □□

- `reset` schreibt die Historie um – ideal für lokale Commits, die noch niemand gesehen hat
- `revert` erstellt einen neuen „Rückgängig-Commit" – sicher für bereits gepushte Commits
- **Faustregel:** Sobald ein Commit gepusht wurde, nutze `revert`
- **Force Push** nur im Notfall und nur bei Solo-Projekten
- PhpStorm bietet für beide Methoden komfortable Menüoptionen im Git-Log

Sensible Daten aus Git entfernen – warum einfaches Löschen nicht reicht □□

Du hast versehentlich ein Passwort, einen API-Key oder andere sensible Daten committed und vielleicht sogar auf GitHub gepusht. Das ist ein ernstes Problem, aber es lässt sich beheben – allerdings **nicht** durch einfaches Löschen der Datei in einem neuen Commit.

Warum reicht es nicht, die Datei einfach zu löschen?

Hier liegt ein fundamentales Missverständnis über Git vor, das vielen Anfängern passiert: **Git vergisst nichts**. Wenn du eine Datei mit einem Passwort commitest und sie dann im nächsten Commit löschst, ist das Passwort nicht weg – es steckt immer noch im *alten* Commit.

Jeder, der Zugriff auf dein Repository hat (oder hatte), kann:

- Die **gesamte Commit-Historie** durchsuchen
- Jeden **alten Commit auschecken** und die Datei dort finden
- Mit Befehlen wie `git log -p` oder `git show` den **kompletten Inhalt** jeder jemals existierenden Dateiversion sehen

> □□ **Merke:** Ein Commit ist wie ein Foto deines gesamten Projekts zu einem bestimmten Zeitpunkt. Auch wenn du die Datei später löschst, existiert das alte „Foto" mit der Datei noch immer in der Historie.

Das bedeutet: **Sensible Daten sind kompromittiert, sobald sie gepusht wurden** – selbst wenn du sie Sekunden später „löschst".

Was du *wirklich* tun musst

Die Lösung erfordert mehrere Schritte, und die Reihenfolge ist wichtig:

1. Sofort das kompromittierte Geheimnis ungültig machen □□

Das ist der **allerwichtigste Schritt** und sollte *sofort* passieren:

- **Passwort ändern** – logge dich ein und ändere das Passwort
- **API-Key rotieren** – erstelle einen neuen Key und deaktiviere den alten
- **Token widerrufen** – bei OAuth-Tokens, Personal Access Tokens etc.

> ⚠ **Geh davon aus, dass das Geheimnis bereits kompromittiert ist!** Bots scannen GitHub kontinuierlich nach Zugangsdaten. Innerhalb von Minuten nach dem Push können deine Daten bereits missbraucht worden sein.

2. Die Git-Historie bereinigen

Nachdem du das Geheimnis ungültig gemacht hast, solltest du es auch aus der Git-Historie entfernen – nicht weil es dadurch wieder sicher wird (das Geheimnis ist bereits kompromittiert), sondern um zu verhindern, dass es bei zukünftigen Clones oder Forks weiter verbreitet wird.

Methode A: Mit `git filter-repo` (empfohlen)

Das Tool `git filter-repo` ist der moderne, sichere Weg, um Dateien oder Inhalte aus der gesamten Git-Historie zu entfernen:

1. **Tool installieren**

```
# Mit pip (Python)
pip install git-filter-repo

# Oder über Paketmanager (z.B. Homebrew auf macOS)
brew install git-filter-repo
```

2. **Datei aus der gesamten Historie entfernen**

```
# Entfernt die Datei "config/secrets.php" aus ALLEN Commits
git filter-repo --path config/secrets.php --invert-paths
```

3. **Änderungen auf GitHub pushen**

```
# Force-Push, weil du die Historie umgeschrieben hast
git push origin --force --all
```

Methode B: Mit BFG Repo-Cleaner (einfacher für Anfänger)

Der BFG Repo-Cleaner ist speziell dafür entwickelt, sensible Daten zu entfernen:

1. **BFG herunterladen** (eine `.jar`-Datei)
2. **Passwörter aus der Historie entfernen**

```
# Erstelle eine Datei mit den zu entfernenden Texten
echo "meinGeheimesPasswort123" > passwords.txt

# BFG ausführen
java -jar bfg.jar --replace-text passwords.txt mein-repo.git
```

3. **Bereinigung abschließen und pushen**

```
cd mein-repo.git
git reflog expire --expire=now --all
git gc --prune=now --aggressive
git push origin --force --all
```

3. GitHub-Cache invalidieren

Auch nach dem Force-Push können alte Commits noch über ihre **SHA-Hashes** erreichbar sein, wenn jemand die URL kennt. GitHub bietet eine Möglichkeit, diese zu entfernen:

1. Kontaktiere den **GitHub-Support** über support.github.com
2. Bitte um die **Entfernung gecachter Ansichten** der betroffenen Commits
3. Wenn das Repository geforkt wurde, müssen auch die **Forks** bereinigt werden

4. Alle Mitwirkenden informieren

Wenn andere Personen das Repository geklont haben, müssen sie:

- Ihren lokalen Klon **löschen und neu klonen**, oder
- Mit `git fetch --all` und `git reset --hard origin/main` synchronisieren

> ⚠ Wenn jemand einen alten Klon hat und pusht, könnten die sensiblen Daten wieder ins Repository gelangen!

Zusammenfassung als Checkliste

```
flowchart TD
    A["Sensible Daten\ngepusht!"] --> B["1. Geheimnis SOFORT\nungueltigt machen"]
    B --> C["Passwort aendern\nAPI-Key rotieren\nToken widerrufen"]
    C --> D["2. Git-Historie\nbereinigen"]
    D --> E["git filter-repo\noder BFG verwenden"]
    E --> F["3. Force-Push\nauf GitHub"]
    F --> G["4. GitHub-Support\nkontaktieren"]
    G --> H["5. Team informieren\nNeu-Clone erforderlich"]

    style A fill:#ff6b6b,color:#000
    style B fill:#ffd93d,color:#000
    style C fill:#ffd93d,color:#000
```

Schritt	Aktion	Priorität
1	Geheimnis ungültig machen	◫ **Sofort**
2	Historie mit `filter-repo` oder BFG bereinigen	◫ Schnell
3	Force-Push durchführen	◫ Schnell
4	GitHub-Support kontaktieren	◫ Zeitnah
5	Team/Mitwirkende informieren	◫ Zeitnah

Wie du das in Zukunft verhinderst

Damit dir das nicht wieder passiert, solltest du einige Vorkehrungen treffen:

1. `.gitignore` **richtig konfigurieren** – sensible Dateien wie `.env`, `config/secrets.php` oder `credentials.json` sollten *niemals* getrackt werden:

```
# Umgebungsvariablen und Secrets
.env
.env.local
*.pem
*.key
config/secrets.php
```

2. **Umgebungsvariablen nutzen** – speichere Passwörter und API-Keys in Umgebungsvariablen statt direkt im Code:

```
// □ Schlecht
$password = "geheim123";

// □ Gut
$password = getenv('DB_PASSWORD');
```

3. **Pre-Commit-Hooks einrichten** – Tools wie git-secrets oder pre-commit können automatisch nach Secrets suchen, *bevor* du commitest
4. **GitHub Secret Scanning aktivieren** – GitHub scannt öffentliche Repositories automatisch nach bekannten Secret-Formaten und warnt dich

---

Fazit

Das versehentliche Pushen von Passwörtern ist ein häufiger und potenziell schwerwiegender Fehler. Die wichtigste Erkenntnis ist: **Einfaches Löschen reicht nicht, weil Git die komplette Historie aufbewahrt.** Deine erste Reaktion muss immer sein, das kompromittierte Geheimnis *sofort* ungültig zu machen – erst danach kümmere dich um die Bereinigung der Historie.