Der Hotfix-Workflow: Kritische Bugfixes sicher in alle Branches bringen □□

Ein Hotfix ist eine der stressigsten Situationen in der Softwareentwicklung: Ein kritischer Bug ist in der Produktion aufgetaucht und muss *sofort* behoben werden – ohne dabei die laufende Entwicklungsarbeit zu gefährden oder den Fix irgendwo zu „vergessen". In dieser Lektion zeige ich dir, wie du einen professionellen Hotfix-Workflow umsetzt und dabei sicherstellst, dass der Fix in allen relevanten Branches ankommt.

Das Grundproblem verstehen

Stell dir folgendes Szenario vor:

```
main (v1.2.0) ──●──●──●  ← Produktionsversion, hier ist der Bug!
                    \
develop ─────────────●──●──●──●  ← Hier wird an v1.3.0 gearbeitet
                         \
feature/neues-login ──────●──●  ← Feature in Entwicklung
```

Der Bug existiert in `main` (der Produktionsversion), aber auch in `develop` und allen Feature-Branches, die von `main` oder `develop` abgezweigt wurden. Wenn du den Fix nur in `main` einspielst, wird er bei der nächsten Entwicklung wieder überschrieben. Spielst du ihn nur in `develop` ein, dauert es bis zum nächsten Release, bis er in Produktion kommt.

Die Lösung: Ein dedizierter Hotfix-Branch, der von `main` abzweigt und nach der Behebung in *alle* relevanten Branches gemergt wird.

Der klassische Hotfix-Workflow (Schritt für Schritt)

1. Hotfix-Branch von `main` erstellen

Der Hotfix-Branch wird **immer von** `main` (oder dem Branch, der die Produktion repräsentiert) erstellt – niemals von `develop`! So stellst du sicher, dass du exakt den Code-Stand der Produktion als Basis hast.

In PhpStorm:

1. Öffne das **Git-Tool-Window** (`Alt+9` / `Cmd+9`)
2. Stelle sicher, dass du auf `main` bist (Rechtsklick auf `main` → *Checkout*)
3. Führe einen **Pull** durch, um sicherzustellen, dass du den aktuellen Stand hast
4. Klicke auf **New Branch** (oder `Ctrl+Shift+`` / `Cmd+Shift+``)
5. Benenne den Branch nach dem Schema: `hotfix/kurze-beschreibung` oder `hotfix/v1.2.1`

Auf der Kommandozeile:

```
# Sicherstellen, dass main aktuell ist
git checkout main
git pull origin main

# Hotfix-Branch erstellen
git checkout -b hotfix/kritischer-login-bug
```

Wichtige Namenskonventionen:

- `hotfix/beschreibung` – z.B. `hotfix/sql-injection-fix`
- `hotfix/vX.Y.Z` – z.B. `hotfix/v1.2.1` (wenn du semantische Versionierung nutzt)
- `hotfix/issue-123` – wenn du ein Issue-Tracking-System verwendest

---

2. Den Bug beheben und committen

Jetzt behebst du den Bug. Dabei gelten einige wichtige Regeln:

Nur den Bug fixen – nichts anderes! Ein Hotfix sollte so minimal wie möglich sein. Jede zusätzliche Änderung erhöht das Risiko, neue Probleme einzuführen. Widerstehe der Versuchung,

„schnell noch" andere kleine Dinge zu korrigieren.

Commit-Message mit Kontext:

```
git add src/Auth/LoginController.php
git commit -m "fix: SQL-Injection-Schwachstelle im Login behoben

- Prepared Statements statt String-Konkatenation
- Betrifft Login und Passwort-Reset
- Fixes #247"
```

In PhpStorm:

1. Mache deine Änderungen im Code
2. Öffne das **Commit-Tool-Window** (`Ctrl+K` / `Cmd+K`)
3. Wähle nur die relevanten Dateien aus
4. Schreibe eine aussagekräftige Commit-Message
5. Nutze optional *Amend* wenn du nachbessern musst

---

3. Hotfix testen

Bevor du den Hotfix irgendwohin mergst, **teste ihn gründlich**:

- Führe deine automatisierten Tests aus
- Teste den spezifischen Fix manuell
- Wenn möglich, deploye auf eine Staging-Umgebung

In PhpStorm kannst du Tests direkt ausführen:

- Rechtsklick auf den Test-Ordner → *Run Tests*
- Oder nutze die Run-Konfiguration für PHPUnit

---

4. Hotfix in `main` mergen und taggen

Jetzt kommt der kritische Teil: Der Fix muss zurück nach `main`, damit er in die Produktion deployed werden kann.

In PhpStorm:

1. Wechsle zu `main` (Rechtsklick → *Checkout*)
2. Rechtsklick auf deinen Hotfix-Branch → *Merge into Current*

3. PhpStorm zeigt dir den Merge-Dialog – überprüfe die Änderungen
4. Bestätige den Merge

Auf der Kommandozeile:

```
# Zu main wechseln
git checkout main

# Hotfix mergen (mit --no-ff für einen expliziten Merge-Commit)
git merge --no-ff hotfix/kritischer-login-bug -m "Merge hotfix/kritischer-login-bug: SQL-Injection behoben"
```

Warum `--no-ff`? Die Option `--no-ff` (*no fast-forward*) erzwingt einen Merge-Commit, auch wenn ein Fast-Forward möglich wäre. Das hat zwei Vorteile:

1. Der Hotfix bleibt in der Historie als eigenständiger „Block" erkennbar
2. Du hast einen klaren Merge-Commit, der dokumentiert, wann der Hotfix eingespielt wurde

Version-Tag erstellen:

Nach einem Hotfix solltest du einen neuen Version-Tag erstellen:

```
# Tag erstellen
git tag -a v1.2.1 -m "Hotfix: SQL-Injection-Schwachstelle behoben"

# Tag auf GitHub pushen
git push origin v1.2.1
```

In PhpStorm:

- Menü: *Git* → *New Tag*
- Oder im Git-Log: Rechtsklick auf den Commit → *New Tag*

---

5. Hotfix in `develop` mergen ⚠

Das ist der Schritt, der am häufigsten vergessen wird! Wenn du den Hotfix nicht auch in `develop` mergst, wird der Bug beim nächsten Release wieder auftauchen, weil `develop` dann ohne den Fix nach `main` gemergt wird.

In PhpStorm:

1. Wechsle zu `develop` (Rechtsklick → *Checkout*)

2. Rechtsklick auf `main` (oder den Hotfix-Branch) → *Merge into Current*
3. Löse eventuelle Konflikte im Merge-Tool

Auf der Kommandozeile:

```
git checkout develop
git pull origin develop  # Sicherstellen, dass develop aktuell ist
git merge --no-ff hotfix/kritischer-login-bug -m "Merge hotfix in develop: SQL-Injection behoben"
```

Oder alternativ – den Tag mergen:

```
git checkout develop
git merge --no-ff v1.2.1 -m "Merge v1.2.1 hotfix in develop"
```

---

6. Alles pushen

Jetzt müssen alle Änderungen auf GitHub landen:

```
# main mit dem Fix pushen
git push origin main

# develop mit dem Fix pushen
git push origin develop

# Tags pushen (falls noch nicht geschehen)
git push origin --tags
```

In PhpStorm:

- `Ctrl+Shift+K` / `Cmd+Shift+K` öffnet den Push-Dialog
- Stelle sicher, dass *Push Tags* aktiviert ist

---

7. Hotfix-Branch aufräumen

Nach erfolgreichem Merge in beide Branches kann der Hotfix-Branch gelöscht werden:

Lokal löschen:

```
git branch -d hotfix/kritischer-login-bug
```

Auf GitHub löschen:

```
git push origin --delete hotfix/kritischer-login-bug
```

In PhpStorm:

- Im Git-Tool-Window: Rechtsklick auf den Branch → *Delete*
- Häkchen setzen bei *Delete Tracking Branch* um ihn auch remote zu löschen

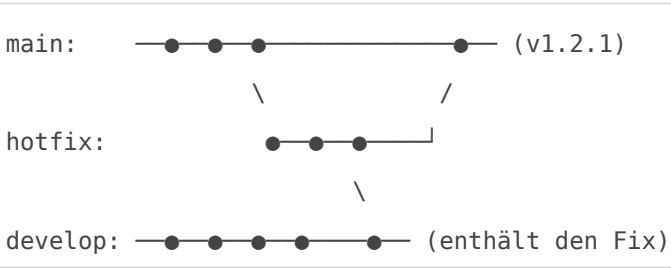
---

Visualisierung des Workflows

```
flowchart TD
    subgraph Ausgangssituation
        A["main\n v1.2.0 - Bug vorhanden"]
        B["develop\n Aktive Entwicklung"]
        A -.->|"abgezweigt"| B
    end

    subgraph Hotfix-Prozess
        C["1. Hotfix-Branch erstellen\n von main"]
        D["2. Bug beheben\n und committen"]
        E["3. Testen"]
        F["4. Merge in main\n Tag: v1.2.1"]
        G["5. Merge in develop"]
        H["6. Push und Cleanup"]
    end

    A --> C
    C --> D
    D --> E
    E --> F
    F --> G
    G --> H

    style C fill:#ffcccc,color:#000000
    style F fill:#ccffcc,color:#000000
```

```
    style G fill:#ccffcc,color:#000000
```

Die Commit-Historie nach dem Hotfix:



Umgang mit Konflikten beim Merge in `develop`

Es ist sehr wahrscheinlich, dass beim Merge des Hotfixes in `develop` Konflikte auftreten – schließlich wurde in `develop` weiterentwickelt, während der Hotfix auf dem älteren `main`-Stand basiert.

Typisches Konflikt-Szenario:

Der Hotfix hat eine Funktion in `LoginController.php` geändert, aber in `develop` wurde dieselbe Funktion für ein neues Feature erweitert.

Konfliktlösung in PhpStorm:

1. Nach dem Merge-Versuch zeigt PhpStorm die konfligierenden Dateien an
2. Doppelklick auf eine Datei öffnet den **3-Way-Merge-Editor**
3. Du siehst drei Spalten:
   - **Links (Yours):** Der aktuelle Stand von `develop`
   - **Rechts (Theirs):** Der Hotfix
   - **Mitte (Result):** Das gewünschte Ergebnis
4. Übernimm die Sicherheitsänderungen aus dem Hotfix
5. Behalte die neuen Features aus `develop`
6. Stelle sicher, dass beides zusammen funktioniert

Wichtig: Nach dem Lösen von Konflikten immer testen! Ein schlecht gelöster Konflikt kann den Fix unwirksam machen.

Was ist mit Feature-Branches?

Wenn zum Zeitpunkt des Hotfixes Feature-Branches existieren, die von `develop` (oder `main`) abgezweigt wurden, enthalten auch diese den Bug. Hier gibt es mehrere Strategien:

Option A: Feature-Branches auf `develop` rebasen

Nachdem der Hotfix in `develop` gemergt wurde, können Feature-Branch-Entwickler ihren Branch auf den neuen `develop`-Stand rebasen:

```
git checkout feature/neues-login
git fetch origin
git rebase origin/develop
```

Vorteil: Der Feature-Branch enthält automatisch den Fix.
Nachteil: Erfordert einen Force-Push, wenn der Branch bereits gepusht wurde.

Option B: `develop` in Feature-Branches mergen

Alternativ können die Feature-Branch-Entwickler `develop` in ihren Branch mergen:

```
git checkout feature/neues-login
git fetch origin
git merge origin/develop
```

Vorteil: Kein Force-Push nötig.
Nachteil: Zusätzliche Merge-Commits.

Option C: Nichts tun und beim PR lösen

Wenn der Feature-Branch ohnehin bald gemergt wird, kann der Konflikt auch beim finalen Merge/PR gelöst werden. Der Hotfix landet dann automatisch im Feature-Branch, sobald dieser nach `develop` gemergt wurde.

Empfehlung: Für sicherheitskritische Hotfixes ist Option A oder B besser, damit alle aktiv entwickelten Branches sofort geschützt sind.

Der Hotfix-Workflow mit Pull Requests

In professionellen Teams wird der Hotfix-Workflow oft mit Pull Requests kombiniert, um Code Reviews auch für kritische Fixes sicherzustellen:

Workflow mit PRs

1. **Hotfix-Branch erstellen und pushen:**

   ```
   git checkout -b hotfix/kritischer-bug main
   # Fix implementieren
   git push -u origin hotfix/kritischer-bug
   ```

2. **Ersten PR erstellen: Hotfix → main**
   - Auf GitHub: *New Pull Request*
   - Base: `main`, Compare: `hotfix/kritischer-bug`
   - Als *Critical* oder *Urgent* markieren
   - Reviewer zuweisen (idealerweise jemand, der sofort verfügbar ist)
3. **Review und Merge in main**
   - Schnelles, fokussiertes Review
   - Nach Approval: **Merge** (nicht Squash, damit der Commit-Hash erhalten bleibt)
   - Tag erstellen
4. **Zweiten PR erstellen: main → develop** (oder Hotfix → develop)
   - Base: `develop`, Compare: `main`
   - Dieser PR dokumentiert, dass der Hotfix auch in `develop` übernommen wurde

Automatisierung mit GitHub Actions

Du kannst einen GitHub Actions Workflow erstellen, der automatisch einen PR von `main` nach `develop` erstellt, sobald ein Hotfix-Tag gepusht wird:

```
name: Create Hotfix Backport PR

on:
```

```
  push:
    tags:
      - 'v*.*.*'  # Triggert bei Version-Tags

jobs:
  create-backport-pr:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: Create Pull Request to develop
        uses: peter-evans/create-pull-request@v5
        with:
          token: ${{ secrets.GITHUB_TOKEN }}
          branch: backport/${{ github.ref_name }}-to-develop
          base: develop
          title: "Backport ${{ github.ref_name }} to develop"
          body: |
            Automatischer Backport-PR für Hotfix ${{ github.ref_name }}.

            **Bitte prüfen und mergen, damit der Fix in develop landet!**
          labels: hotfix, backport
```

Checkliste für Hotfixes

Hier ist eine praktische Checkliste, die du bei jedem Hotfix abarbeiten solltest:

Vor dem Hotfix

- ☐ Bug ist verifiziert und reproduzierbar
- ☐ Priorität ist klar (ist es wirklich ein *kritischer* Hotfix?)
- ☐ Alle Stakeholder sind informiert

Während des Hotfixes

- [ ] Hotfix-Branch von `main` erstellt (nicht von `develop`!)
- [ ] Nur den Bug behoben, keine anderen Änderungen
- [ ] Aussagekräftige Commit-Message mit Issue-Referenz
- [ ] Tests geschrieben oder angepasst
- [ ] Alle Tests laufen durch

Nach dem Hotfix

- [ ] In `main` gemergt
- [ ] Version-Tag erstellt
- [ ] **In `develop` gemergt** ← *Wird am häufigsten vergessen!*
- [ ] Alle Branches gepusht
- [ ] Tag gepusht
- [ ] Hotfix-Branch gelöscht (lokal und remote)
- [ ] Deployment in Produktion durchgeführt
- [ ] Fix in Produktion verifiziert
- [ ] Team informiert (welche Feature-Branches sollten aktualisiert werden?)

Häufige Fehler und wie du sie vermeidest

Fehler 1: Hotfix von `develop` statt `main` erstellen

Problem: Du erstellst den Hotfix von `develop`, das bereits neue, ungetestete Features enthält. Wenn du jetzt nach `main` mergst, landen diese Features ungewollt in der Produktion.

Lösung: Immer zuerst `git checkout main` und dann den Hotfix-Branch erstellen.

Fehler 2: Vergessen, den Hotfix in `develop` zu mergen

Problem: Der Fix ist in Produktion, aber bei der nächsten großen Release-Integration wird der Bug wieder eingeführt, weil `develop` den Fix nicht enthält.

Lösung: Nutze die Checkliste oder automatisiere den Backport mit GitHub Actions.

Fehler 3: Zu viele Änderungen im Hotfix

Problem: Du nutzt den Hotfix als Gelegenheit, „schnell noch" andere Dinge zu fixen. Der Hotfix wird groß und schwer zu reviewen, das Risiko für neue Bugs steigt.

Lösung: Disziplin! Andere Fixes kommen in reguläre Feature-Branches. Ein Hotfix ist *nur* für den kritischen Bug.

Fehler 4: Keinen Tag erstellen

Problem: Ohne Tag ist später schwer nachzuvollziehen, welcher exakte Code-Stand in Produktion deployed wurde.

Lösung: Immer einen semantischen Version-Tag erstellen (`v1.2.1` für einen Hotfix auf `v1.2.0`).

Hotfixes in PhpStorm – Die wichtigsten Shortcuts

Aktion	Windows/Linux	macOS
Git-Tool-Window öffnen	`Alt+9`	`Cmd+9`
Neuen Branch erstellen	`Ctrl+Shift+```	`Cmd+Shift+```
Commit-Dialog öffnen	`Ctrl+K`	`Cmd+K`
Push-Dialog öffnen	`Ctrl+Shift+K`	`Cmd+Shift+K`
Pull (Update Project)	`Ctrl+T`	`Cmd+T`
Branches anzeigen	`Ctrl+Shift+```	`Cmd+Shift+```

Aktion	Windows/Linux	macOS
Git-Log anzeigen	`Alt+9`, dann Tab *Log*	`Cmd+9`, dann Tab *Log*

Zusammenfassung

Der Hotfix-Workflow folgt einem klaren Muster:

1. **Branch von `main`** – Nicht von `develop`!
2. **Minimal fixen** – Nur den Bug, nichts anderes
3. **Testen** – Automatisiert und manuell
4. **Merge in `main`** – Mit `--no-ff` und Version-Tag
5. **Merge in `develop`** – Der kritische Schritt, der oft vergessen wird
6. **Aufräumen** – Branch löschen, Team informieren

Mit diesem Workflow stellst du sicher, dass kritische Bugfixes schnell in die Produktion gelangen *und* in allen Entwicklungszweigen ankommen, sodass der Bug nicht bei der nächsten Release-Integration wieder auftaucht.

Revision #1
Created 2026-06-29 15:34:34 UTC by art10m
Updated 2026-06-29 15:34:52 UTC by art10m