Der Feature-Branch-Workflow – sauber entwickeln in PhpStorm □□

Direkt auf dem `main`-Branch zu arbeiten ist **keine gute Idee** – selbst wenn du alleine arbeitest. Der sogenannte **Feature-Branch-Workflow** ist eine einfache, aber wirkungsvolle Methode, um dein Projekt sauber und sicher zu halten. Hier erfährst du, warum das so ist und wie du diesen Workflow in PhpStorm praktisch umsetzt.

Warum nicht direkt auf `main` arbeiten?

Der `main`-Branch (früher oft `master` genannt) sollte immer den **stabilen, funktionierenden Zustand** deines Projekts repräsentieren. Wenn du direkt darauf arbeitest, riskierst du mehrere Probleme:

- **Halbfertiger Code im Hauptbranch:** Du fängst ein Feature an, bist mittendrin – und plötzlich musst du einen dringenden Bug fixen. Jetzt ist dein `main` in einem unbrauchbaren Zwischenzustand.
- **Schwierige Fehlersuche:** Wenn alles in einem langen Strang von Commits liegt, ist es schwerer nachzuvollziehen, welche Änderungen zu welchem Feature gehören.
- **Kein einfaches Verwerfen:** Stellst du fest, dass ein Experiment nicht funktioniert, musst du mühsam einzelne Commits rückgängig machen, statt einfach einen Branch zu löschen.

Der Feature-Branch-Workflow löst all diese Probleme elegant.

Das Prinzip des Feature-Branch-Workflows

Die Idee ist simpel:

> **Für jedes neue Feature, jeden Bugfix oder jedes Experiment erstellst du einen eigenen Branch.** Erst wenn die Arbeit fertig und getestet ist, führst du den Branch in `main` zusammen.

```
flowchart TD
    MAIN["main-Branch\nImmer stabil und funktionsfaehig"]
    FEATURE1["feature/login-formular\nNeues Feature entwickeln"]
    FEATURE2["bugfix/navbar-fehler\nBug beheben"]

    MAIN -->|"Branch erstellen"| FEATURE1
    MAIN -->|"Branch erstellen"| FEATURE2
    FEATURE1 -->|"Merge nach Fertigstellung"| MAIN
    FEATURE2 -->|"Merge nach Fertigstellung"| MAIN

    style MAIN fill:#c8e6c9,color:#000000
    style FEATURE1 fill:#bbdefb,color:#000000
    style FEATURE2 fill:#ffe0b2,color:#000000
```

Dadurch bleibt `main` immer in einem Zustand, den du jederzeit deployen oder jemandem zeigen könntest.

Der Workflow Schritt für Schritt

1. Vor dem Start: Aktuellen Stand holen

Bevor du einen neuen Branch erstellst, solltest du sicherstellen, dass dein lokaler `main`-Branch auf dem neuesten Stand ist. In PhpStorm:

1. Wechsle zum `main`-Branch (falls nicht schon dort) – klicke dafür auf den **Branch-Namen unten rechts** und wähle `main`.
2. Führe einen **Pull** durch: **Git → Pull** oder `Ctrl+T` (Windows) / `Cmd+T` (Mac).

2. Neuen Feature-Branch erstellen

Jetzt erstellst du einen Branch für dein Vorhaben:

1. Klicke auf den **Branch-Namen unten rechts** in PhpStorm.
2. Wähle **New Branch** aus dem Popup-Menü.
3. Gib einen **aussagekräftigen Namen** ein, zum Beispiel:
   - `feature/kontaktformular` – für ein neues Feature
   - `bugfix/login-fehler` – für eine Fehlerbehebung
   - `experiment/neues-design` – für etwas, das du ausprobieren möchtest
4. Stelle sicher, dass **Checkout branch** aktiviert ist (damit du direkt auf den neuen Branch wechselst).
5. Klicke auf **Create**.

> 🧠 **Tipp zur Benennung:** Verwende Prefixe wie `feature/`, `bugfix/` oder `experiment/`, gefolgt von einer kurzen Beschreibung. Das macht deine Branch-Liste übersichtlich.

3. Arbeiten und regelmäßig committen

Jetzt arbeitest du ganz normal auf deinem neuen Branch:

- Schreibe Code, teste, verbessere.
- **Committe regelmäßig** kleine, in sich abgeschlossene Schritte – nicht erst am Ende alles auf einmal.
- Schreibe aussagekräftige Commit-Nachrichten, wie du es gelernt hast.

Das Schöne daran: Alles, was du hier commitest, landet **nur in diesem Branch**. Dein `main` bleibt davon unberührt.

4. Feature fertig? Zurück zu `main` wechseln

Wenn du mit deiner Arbeit fertig bist und alles getestet hast:

1. Wechsle zurück zum `main`-Branch – klicke auf den Branch-Namen unten rechts und wähle `main` → **Checkout**.

2. Optional, aber empfohlen: Hole dir den neuesten Stand mit **Pull**, falls sich zwischenzeitlich etwas geändert hat.

5. Feature-Branch in `main` mergen

Jetzt führst du deinen Feature-Branch in `main` zusammen:

1. Klicke wieder auf den **Branch-Namen unten rechts**.
2. Finde deinen Feature-Branch in der Liste (z. B. `feature/kontaktformular`).
3. Klicke darauf und wähle **Merge into Current** (oder „In aktuellen Branch mergen").
4. PhpStorm führt den Merge durch. Wenn es keine Konflikte gibt, ist alles sofort erledigt.
5. Bei einem **Merge-Konflikt** öffnet PhpStorm automatisch das Konflikt-Tool – löse die Konflikte wie gewohnt.

6. Änderungen pushen

Nach dem Merge solltest du deinen aktualisierten `main`-Branch auf GitHub pushen:

- **Git → Push** oder `Ctrl+Shift+K` (Windows) / `Cmd+Shift+K` (Mac).

7. Feature-Branch aufräumen (optional, aber empfohlen)

Der Feature-Branch hat seinen Zweck erfüllt. Du kannst ihn jetzt löschen, um deine Branch-Liste sauber zu halten:

1. Klicke auf den Branch-Namen unten rechts.
2. Finde den gemergten Feature-Branch.
3. Klicke darauf und wähle **Delete** – PhpStorm fragt dich, ob du ihn auch remote löschen möchtest (falls du ihn gepusht hattest).

Der komplette Ablauf visualisiert

```
flowchart TD
    A["1. Auf main wechseln\nund Pull durchfuehren"] --> B["2. Neuen Branch erstellen\nz.B. feature/warenkorb"]
    B --> C["3. Entwickeln und\nregelmaessig committen"]
```

```
    C --> D{"4. Feature fertig\nund getestet?"}
    D -->|"Nein"| C
    D -->|"Ja"| E["5. Zu main wechseln\nund Pull durchfuehren"]
    E --> F["6. Feature-Branch\nin main mergen"]
    F --> G["7. main auf\nGitHub pushen"]
    G --> H["8. Feature-Branch\nloeschen"]
    H --> I["Fertig!"]

    style A fill:#e3f2fd,color:#000000
    style B fill:#e3f2fd,color:#000000
    style C fill:#fff3e0,color:#000000
    style D fill:#fce4ec,color:#000000
    style E fill:#e3f2fd,color:#000000
    style F fill:#c8e6c9,color:#000000
    style G fill:#c8e6c9,color:#000000
    style H fill:#f3e5f5,color:#000000
    style I fill:#c8e6c9,color:#000000
```

Praktisches Beispiel: Ein Kontaktformular entwickeln

Angenommen, du möchtest ein Kontaktformular zu deinem PHP-Projekt hinzufügen:

1. **Pull auf** `main` – sicherstellen, dass du aktuell bist.
2. **Branch erstellen:** `feature/kontaktformular`
3. **Arbeiten:**
   - Commit 1: „HTML-Struktur für Kontaktformular erstellt"
   - Commit 2: „CSS-Styling hinzugefügt"
   - Commit 3: „PHP-Verarbeitung implementiert"
   - Commit 4: „E-Mail-Validierung ergänzt"
4. **Testen** – alles funktioniert wie gewünscht.
5. **Zu** `main` **wechseln** und Pull durchführen.
6. **Merge** von `feature/kontaktformular` in `main`.
7. **Push** des aktualisierten `main` auf GitHub.
8. **Branch löschen:** `feature/kontaktformular` wird nicht mehr benötigt.

Falls du mitten in der Arbeit einen dringenden Bug im Live-System fixen müsstest, wäre das kein Problem: Du könntest einfach zu `main` wechseln, einen neuen `bugfix/`-Branch erstellen, den Fehler

beheben, mergen und dann zu deinem Kontaktformular-Branch zurückkehren.

Zusammenfassung: Die goldenen Regeln

Regel	Warum?
`main` **ist immer stabil**	Du kannst jederzeit deployen oder demonstrieren
Ein Branch pro Aufgabe	Klare Trennung, einfaches Verwerfen
Aussagekräftige Branch-Namen	Du findest dich auch in einer Woche noch zurecht
Regelmäßig committen	Kleine Schritte sind leichter nachzuvollziehen
Nach dem Merge aufräumen	Keine Verwirrung durch alte Branches

Dieser Workflow mag anfangs wie ein Umweg erscheinen, aber nach kurzer Zeit wird er zur Gewohnheit – und du wirst dich fragen, wie du jemals ohne Branches gearbeitet hast. □□

Revision #2
Created 2026-06-17 04:47:01 UTC by art10m
Updated 2026-06-17 04:47:48 UTC by art10m