Ein einfacher Git-Workflow für Einzelentwickler □□

Als Einzelentwickler brauchst du keinen komplizierten Workflow mit Pull Requests, Code Reviews oder Release-Branches. Trotzdem lohnt es sich, ein paar **klare Gewohnheiten** zu etablieren – sie halten dein Projekt übersichtlich, schützen dich vor Datenverlust und machen es leichter, Fehler zu finden oder rückgängig zu machen.

Der empfohlene Workflow im Überblick

```
flowchart TD
    START["□□Arbeitstag beginnen"] --> PULL["git pull\nAktuellen Stand holen"]
    PULL --> BRANCH{"Neues Feature\noder Experiment?"}
    BRANCH -->|Ja| CREATE["Neuen Branch erstellen\nz.B. feature/kontaktformular"]
    BRANCH -->|Nein| WORK["Direkt auf main arbeiten\nbei kleinen Aenderungen"]
    CREATE --> WORK2["Im Feature-Branch arbeiten"]
    WORK --> COMMIT["Regelmaessig committen\nbei logischen Einheiten"]
    WORK2 --> COMMIT
    COMMIT --> MORE{"Weitere\nAenderungen?"}
    MORE -->|Ja| COMMIT
    MORE -->|Nein| MERGE["Feature-Branch mergen\nfalls vorhanden"]
    MERGE --> PUSH["git push\nAendeurngen sichern"]
    PUSH --> END["□□Feierabend"]
```

Dein typischer Arbeitstag mit Git

1. Arbeitsbeginn: Aktuellen Stand holen □□

Bevor du mit der Arbeit beginnst, solltest du sicherstellen, dass dein lokales Repository auf dem neuesten Stand ist – besonders wenn du an mehreren Geräten arbeitest oder gelegentlich direkt auf GitHub kleine Änderungen machst.

```
git pull
```

In **PhpStorm** geht das über **Git → Pull** oder mit dem blauen Pfeil-nach-unten-Button in der Toolbar. Dieser kurze Schritt verhindert, dass du später mit Push-Konflikten kämpfen musst.

2. Entscheiden: Branch oder nicht? □□

Hier kommt die wichtigste Frage des Tages:

Situation	Empfehlung
Kleiner Bugfix, Tippfehler, CSS-Anpassung	Direkt auf `main` arbeiten
Neues Feature, größere Änderung, Experiment	Eigenen Branch erstellen
Du bist unsicher, ob die Änderung funktioniert	Eigenen Branch erstellen

Faustregel: Wenn du dir vorstellst, dass du die Änderung möglicherweise komplett verwerfen willst, gehört sie in einen eigenen Branch.

Einen neuen Branch erstellst du in PhpStorm über die **Branch-Anzeige** unten rechts → **New Branch**. Benenne ihn sprechend, z. B.:

- `feature/newsletter-anmeldung`
- `fix/login-redirect`
- `experiment/neues-layout`

3. Arbeiten und Committen: Die goldene Regel ✍

Committe früh und oft – aber nicht wahllos. Ein Commit sollte eine **logische Einheit** darstellen, also eine abgeschlossene kleine Änderung, die für sich allein Sinn ergibt.

Gute Zeitpunkte für einen Commit:

- Du hast eine neue Funktion fertiggestellt (auch wenn sie klein ist)
- Du hast einen Bug behoben
- Du hast Refactoring abgeschlossen
- Du machst eine Pause oder wechselst das Thema
- Du hast etwas zum Laufen gebracht, das vorher nicht funktionierte

Schlechte Zeitpunkte für einen Commit:

- Der Code kompiliert/funktioniert nicht
- Du bist „mittendrin" in einer Änderung
- Du hast zehn verschiedene Dinge gleichzeitig geändert

> □□ **Tipp:** Lieber fünf kleine Commits mit klaren Nachrichten als ein riesiger Commit mit „diverse Änderungen".

Beispiel für einen typischen Vormittag:

```
09:15  Commit: "Kontaktformular HTML-Struktur erstellt"
09:45  Commit: "Validierung für E-Mail-Feld hinzugefügt"
10:30  Commit: "Formular-Styling angepasst"
11:00  Commit: "E-Mail-Versand implementiert"
```

4. Feature fertig: Mergen □□

Wenn du in einem Feature-Branch gearbeitet hast und zufrieden bist:

1. **Wechsle zurück zu** `main` (in PhpStorm: Branch-Anzeige → `main` → Checkout)
2. **Merge den Feature-Branch** (in PhpStorm: Branch-Anzeige → deinen Feature-Branch → Merge into Current)
3. **Lösche den Feature-Branch** (optional, aber hält die Liste sauber)

Bei einem Einzelentwickler-Projekt entstehen hier selten Konflikte, weil niemand anders parallel an `main` arbeitet.

5. Arbeitsende: Pushen und sichern □□

Am Ende jedes Arbeitstages solltest du pushen – egal ob du „fertig" bist oder nicht. Der Push ist dein Backup in der Cloud. Wenn dein Laptop morgen kaputtgeht, ist deine Arbeit sicher.

```
git push
```

Oder in PhpStorm: **Git → Push** bzw. der grüne Pfeil-nach-oben-Button.

>

> ⚠ **Wichtig:** Pushe nur Branches, mit denen du aktiv arbeitest. Unfertige Feature-Branches kannst du auch pushen – sie liegen dann sicher auf GitHub, ohne `main` zu beeinflussen.

Zusammenfassung: Die wichtigsten Gewohnheiten

Wann	Was	Warum
Arbeitsbeginn	`git pull`	Aktuellen Stand holen, Konflikte vermeiden
Vor größeren Änderungen	Neuen Branch erstellen	Saubere Trennung, einfaches Verwerfen möglich
Nach jeder logischen Einheit	Committen mit guter Nachricht	Nachvollziehbare Historie, einfaches Zurückrollen
Feature abgeschlossen	Branch mergen	Änderungen in `main` integrieren
Arbeitsende	`git push`	Backup in der Cloud, Zugriff von überall

Zusätzliche Tipps für den Alltag □□

Commit-Nachrichten: Schreibe so, dass du in drei Monaten noch verstehst, was du getan hast. „Login gefixt" ist schlecht, „Redirect-Schleife beim Login behoben, wenn Session abgelaufen" ist gut.

Nicht committen: Halte dich an deine `.gitignore` – Dependencies (`vendor/`, `node_modules/`), IDE-Einstellungen und sensible Daten gehören nicht ins Repository.

Regelmäßigkeit schlägt Perfektion: Es ist besser, jeden Tag etwas zu pushen, als alle zwei Wochen einen perfekten Mega-Commit zu machen. Git ist ein Werkzeug für deinen Alltag, nicht für besondere Anlässe.

Revision #1
Created 2026-06-17 04:43:56 UTC by art10m
Updated 2026-06-17 04:45:30 UTC by art10m