Den Zustand mit git status prüfen

Den Zustand mit `git status` prüfen

Bevor du irgendeine Git-Operation ausführst – einen Commit erstellst, einen Branch wechselst oder Änderungen verwirfst – solltest du dir eine Frage stellen: *„In welchem Zustand befindet sich mein Repository gerade?“* Der Befehl `git status` ist die Antwort darauf. Er ist der wichtigste diagnostische Befehl in Git und sollte zur Selbstverständlichkeit werden, ähnlich wie ein Blick in den Rückspiegel vor dem Ausparken.

Warum dieser Befehl so zentral ist

Git verändert nichts an deinen Dateien, ohne dass du es ausdrücklich anforderst. Das bedeutet aber auch: Es liegt an dir, jederzeit zu wissen, *was* sich im Arbeitsverzeichnis, in der Staging Area und im Repository befindet. `git status` liefert genau diese Momentaufnahme – ohne Nebenwirkungen, ohne Risiko. Er verändert niemals Dateien oder die Historie; du kannst ihn beliebig oft aufrufen.

> **Merksatz:** Im Zweifel `git status` ausführen – dieser Befehl kostet nichts und schützt vor teuren Fehlern.

Grundlegender Aufruf

Im Übungsprojekt aus dem vorherigen Abschnitt reicht ein einfacher Aufruf:

```
git status
```

Direkt nach `git init`, wenn das Repository noch leer ist, liefert Git etwa folgende Ausgabe:

```
Auf Branch main

Noch keine Commits

Nichts zu committen
```

Diese Ausgabe bestätigt drei wichtige Informationen gleichzeitig:

1. **Aktueller Branch** – hier `main`, der Standard-Branch.
2. **Commit-Historie** – es existiert noch kein einziger Commit.
3. **Arbeitsverzeichnis** – es gibt keine Änderungen, die Git registriert hat.

Der typische Aufbau der Ausgabe

Sobald Dateien im Projekt vorhanden sind, wird die Ausgabe deutlich informativer. Angenommen, du hast eine neue Datei `index.php` erstellt:

```
git status
```

```
Auf Branch main

Noch keine Commits

Unversionierte Dateien:
  (benutze "git add <Datei>..." zum Aufnehmen in das, was committet wird)
        index.php

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutze "git add" zum Aufnehmen)
```

Diese Struktur begleitet dich durch den gesamten Kurs. Git gliedert die Ausgabe grundsätzlich in folgende Abschnitte:

Abschnitt	Bedeutung
Änderungen, die committet werden	Inhalte in der Staging Area – bereit für den nächsten Commit
Änderungen, die nicht zum Commit vorgemerkt sind	Bereits verfolgte Dateien mit ungespeicherten Änderungen
Unversionierte Dateien	Dateien, die Git noch nicht kennt

Besonders hilfreich: Git formuliert unter jeder Kategorie direkt den passenden Befehl, um den Zustand zu ändern. Diese Hinweistexte lohnt es sich, in den ersten Wochen bewusst zu lesen, statt sie zu überspringen.

Den Zustand nach `git add` beobachten

Merke einen Teil der Datei nun mit `git add` vor:

```
git add index.php
git status
```

```
Auf Branch main


Noch keine Commits


Änderungen, die committet werden:
  (benutze "git rm --cached <Datei>..." zum Entfernen aus der Staging-Area)
        neue Datei:     index.php
```

Die Datei ist jetzt aus der Kategorie *unversioniert* in die Kategorie *committen* gewandert. Genau dieses Verschieben zwischen den drei Bereichen – Arbeitsverzeichnis, Index und Repository – ist der Kern dessen, was `git status` sichtbar macht. Kapitel 5 vertieft dieses Modell; hier lernst du bereits, es praktisch zu beobachten.

Zustand nach einem Commit und weiteren Änderungen

Nach dem ersten Commit (siehe folgender Abschnitt dieses Kapitels) ändert sich die Ausgabe erneut. Bearbeitest du `index.php` danach weiter, zeigt `git status` beispielsweise:

```
Auf Branch main
Änderungen, die nicht zum Commit vorgemerkt sind:
  (benutze "git add <Datei>..." zum Aktualisieren, was committet werden soll)
  (benutze "git restore <Datei>..." zum Verwerfen von Änderungen im Arbeitsverzeichnis)
        geändert:       index.php

keine Änderungen zum Commit vorgemerkt (benutze "git add" und/oder "git commit -a")
```

Beachte den Unterschied zur vorherigen Ausgabe: Es fehlt der Hinweis *„Noch keine Commits“*, da bereits ein Commit existiert. Zudem erscheint jetzt die Kategorie *geändert* statt *neue Datei*, weil Git die Datei bereits aus einem früheren Commit kennt.

Kurzform mit `git status -s`

Für den täglichen Gebrauch ist die ausführliche Ausgabe oft zu lang. Die Kurzform liefert dieselben Informationen kompakt:

```
git status -s
```

```
 M index.php
?? notizen.txt
```

Die beiden Buchstaben links stehen jeweils für den **Index-Status** und den **Arbeitsverzeichnis-Status** einer Datei. Die wichtigsten Codes im Überblick:

Code	Bedeutung
`??`	Unversionierte Datei
` M`	Geändert, aber nicht vorgemerkt
`M`	Geändert und vorgemerkt
`A`	Neu hinzugefügt und vorgemerkt
`D`	Gelöscht und vorgemerkt
`UU`	Merge-Konflikt in dieser Datei

Diese Kurzform ist besonders nützlich, wenn du `git status` in eigene Aliase oder Skripte einbindest (siehe Kapitel 26).

Branch- und Tracking-Informationen

`git status` zeigt zusätzlich an, wie dein lokaler Branch im Vergleich zu seinem Remote-Gegenstück steht – sobald ein Remote-Tracking existiert (Kapitel 13 und 14 behandeln dies ausführlich):

```
Auf Branch main
Ihr Branch ist auf demselben Stand wie 'origin/main'.
```

Oder, falls du bereits lokale Commits hast, die noch nicht veröffentlicht wurden:

```
Auf Branch main
Ihr Branch ist 2 Commits vor 'origin/main'.
  (benutze "git push", um lokale Commits zu veröffentlichen)
```

Diese Information bewahrt dich davor, versehentlich unveröffentlichte Arbeit zu verlieren oder zu überschreiben.

`git status` in PhpStorm

PhpStorm zeigt denselben Zustand grafisch an, ohne dass du die Kommandozeile öffnen musst:

- Im **Projektbaum** werden Dateien farblich markiert: Blau für geänderte, Grün für neue (unversionierte) und Grau für ignorierte Dateien.
- Das **Commit-Werkzeugfenster** (`Alt` + `0`) listet alle geänderten und unversionierten Dateien strukturiert nach Changelists auf.
- Die **Statusleiste** unten zeigt den aktuellen Branch sowie die Anzahl ausstehender Änderungen.

```
flowchart LR
    A["Arbeitsverzeichnis"] -->|unversioniert / geändert| B["git status Ausgabe"]
    C["Staging Area / Index"] -->|vorgemerkt| B
    D["Letzter Commit"] -->|Vergleichsbasis| B
    B --> E["Entscheidung: add, restore, commit"]
```

Die farbliche Kennzeichnung in PhpStorm ist letztlich nichts anderes als eine visuelle Aufbereitung derselben Informationen, die `git status` textuell liefert. Wer beide Darstellungen kennt, kann flexibel zwischen grafischer Übersicht und präziser Kommandozeilenkontrolle wechseln.

Praktische Gewohnheit etablieren

Für den weiteren Kursverlauf empfiehlt sich folgende Routine, die du ab jetzt konsequent anwenden solltest:

1. **Vor** jeder Aktion: `git status`, um den Ausgangszustand zu kennen.
2. **Nach** jeder Aktion (`add`, `commit`, `checkout`, `merge` etc.): erneut `git status`, um die Wirkung zu bestätigen.
3. Bei unklaren oder unerwarteten Meldungen: die Hinweistexte in der Ausgabe genau lesen, bevor du einen Befehl aus dem Gedächtnis ausführst.

Diese Disziplin verhindert die meisten Anfängerfehler, die in Abschnitt 1.8 bereits angesprochen wurden – etwa das versehentliche Committen unfertiger Dateien oder das Verwerfen von Änderungen, die eigentlich behalten werden sollten. `git status` ist damit nicht nur ein Informationsbefehl, sondern dein wichtigstes Sicherheitsnetz im gesamten Git-Alltag.

Revision #1
Created 2026-07-26 17:11:49 UTC by art10m
Updated 2026-07-26 17:12:04 UTC by art10m