Arbeitsverzeichnis, Index und Repository

Wer Git wirklich verstehen will, muss zunächst ein einziges Modell verinnerlichen: Git verwaltet Änderungen nicht in *einem* Speicherort, sondern in **drei klar getrennten Bereichen**. Fast jede Verwirrung im Umgang mit Git – „Warum sieht `git status` das nicht?“, „Wo ist meine Änderung hin?“, „Warum wird das committet, obwohl ich es nicht wollte?“ – lässt sich auf ein unklares Verständnis dieser drei Bereiche zurückführen. Dieses Kapitel schafft die gedankliche Grundlage für alles Weitere.

Die drei Bereiche im Überblick

Git unterscheidet zwischen:

1. **Arbeitsverzeichnis** *(Working Directory / Working Tree)* – die Dateien, wie du sie im Explorer oder in PhpStorm siehst und bearbeitest.
2. **Index** *(Staging Area, auch „Cache“ genannt)* – ein Zwischenspeicher, der festlegt, was in den *nächsten* Commit aufgenommen wird.
3. **Repository** *(genauer: die Objektdatenbank in `.git`)* – die dauerhafte, unveränderliche Historie aller bisherigen Commits.

$$
\text{Arbeitsverzeichnis} \xrightarrow{\text{git add}} \text{Index} \xrightarrow{\text{git commit}} \text{Repository}
$$

Diese drei Stationen bilden eine Art Förderband: Änderungen entstehen im Arbeitsverzeichnis, werden bewusst für den nächsten Commit ausgewählt (Index) und schließlich dauerhaft in der Historie verankert (Repository).

```
flowchart LR
    A["Arbeitsverzeichnis<br/>bearbeitete Dateien"] -->|git add| B["Index<br/>Staging Area"]
    B -->|git commit| C["Repository<br/>.git Objektdatenbank"]
    C -->|git restore --staged| B
    C -->|git checkout / restore| A
    B -->|git restore| A
```

Hinweis: In Mermaid-Labels sind keine `<br/>`-Zeilenumbrüche erlaubt – im tatsächlichen Diagramm werden Zeilenumbrüche daher durch Kommas oder kurze Beschriftungen ersetzt.

Das Arbeitsverzeichnis

Das Arbeitsverzeichnis ist der Ort, an dem die eigentliche Arbeit passiert. Es enthält die aktuelle Momentaufnahme deines Projekts als reale Dateien auf der Festplatte – genau das, was PhpStorm im Projektbaum anzeigt und was du mit einem Editor öffnest.

Wichtig ist: Das Arbeitsverzeichnis ist **nicht** die Historie. Es zeigt immer nur den Zustand, der aktuell ausgecheckt ist – meist der letzte Commit des aktiven Branches, plus alle Änderungen, die du seither vorgenommen hast.

```
# Zeigt den aktuellen Zustand des Arbeitsverzeichnisses
git status
```

Git vergleicht dabei den Inhalt der Dateien auf der Festplatte mit dem, was im Index bzw. im letzten Commit gespeichert ist. Aus diesem Vergleich ergeben sich die bekannten Kategorien: *unverändert*, *verändert*, *nicht verfolgt* (untracked) und *ignoriert*.

Der Index als Steuerzentrale

Der Index ist konzeptionell der wichtigste – und zugleich am häufigsten missverstandene – Bereich. Er ist **kein** Verzeichnis auf der Festplatte, sondern eine einzelne binäre Datei unter `.git/index`, die eine Liste von Dateipfaden mit zugehörigen Metadaten und Objekt-Hashes enthält.

Man kann sich den Index als **Entwurf des nächsten Commits** vorstellen. Erst wenn eine Änderung explizit mit `git add` in den Index übernommen wurde, „weiß“ Git, dass sie beim nächsten `git commit` berücksichtigt werden soll.

```
# Änderung im Arbeitsverzeichnis vornehmen
echo "Neue Zeile" >> beispiel.txt

# Änderung für den nächsten Commit vormerken
git add beispiel.txt

# Den Index einsehen
git status
```

Diese Zwischenstufe erlaubt eine entscheidende Fähigkeit, die viele andere Werkzeuge nicht bieten: **selektives Committen**. Du kannst zehn geänderte Dateien im Arbeitsverzeichnis haben und trotzdem nur drei davon – oder sogar nur einen Teil einer einzelnen Datei – in den nächsten Commit übernehmen. Diesem Thema widmet sich das weitere Kapitel im Detail (Abschnitte zu selektivem und interaktivem Staging).

> **Merksatz:** Der Index beschreibt nicht, was du geändert hast, sondern was du *committen willst*.

Das Repository als Objektdatenbank

Das Repository – technisch der Inhalt von `.git` – ist die dauerhafte Ablage aller jemals erstellten Commits, Bäume und Dateiinhalte. Sobald ein Commit erstellt wurde, ist er **unveränderlich**: Jede „Änderung“ eines bestehenden Commits erzeugt in Wahrheit einen neuen Commit mit neuer Prüfsumme.

```
# Den Index als neuen Commit im Repository verankern
git commit -m "Beispieländerung hinzufügen"

# Die Commit-Historie im Repository einsehen
git log --oneline
```

Nach einem `git commit` sind Arbeitsverzeichnis, Index und Repository für die betroffenen Dateien wieder **synchron** – bis die nächste Änderung im Arbeitsverzeichnis vorgenommen wird und der Zyklus erneut beginnt.

Zustände im Zusammenspiel

Die folgende Tabelle fasst zusammen, wie sich derselbe Dateiinhalt je nach Bereich unterscheiden kann:

Bereich	Enthält	Verändert sich durch
Arbeitsverzeichnis	reale Dateien auf der Festplatte	Bearbeiten, Speichern, Löschen
Index	Entwurf des nächsten Commits	`git add`, `git restore --staged`, `git reset`

Bereich	Enthält	Verändert sich durch
Repository	vollständige, unveränderliche Historie	`git commit`, `git merge`, `git rebase`

Ein praktisches Beispiel verdeutlicht das Zusammenspiel:

```
# 1. Datei im Arbeitsverzeichnis ändern
echo "Zeile A" >> datei.txt

# 2. Zustand prüfen: Änderung ist "not staged"
git status

# 3. Änderung in den Index übernehmen
git add datei.txt

# 4. Zustand prüfen: Änderung ist jetzt "staged"
git status

# 5. Änderung dauerhaft im Repository verankern
git commit -m "Zeile A hinzufügen"

# 6. Zustand prüfen: Arbeitsverzeichnis, Index und Repository sind wieder synchron
git status
```

Der letzte `git status`-Aufruf zeigt typischerweise „nothing to commit, working tree clean“ – ein klares Signal dafür, dass alle drei Bereiche denselben Stand widerspiegeln.

Warum dieses Modell zentral ist

Ohne dieses Drei-Bereiche-Modell wirken viele Git-Befehle willkürlich. Mit ihm werden sie **logisch vorhersehbar**:

- `git diff` vergleicht standardmäßig **Arbeitsverzeichnis mit Index**.
- `git diff --staged` vergleicht **Index mit dem letzten Commit**.
- `git restore` bewegt Inhalte **vom Repository oder Index zurück ins Arbeitsverzeichnis**.
- `git restore --staged` bewegt Inhalte **vom letzten Commit zurück in den Index**.
- `git reset` verändert, wie weit der Index und optional das Arbeitsverzeichnis dem Repository „nachgezogen“ werden.

Wer diese drei Bereiche als feste Landkarte im Kopf hat, kann jeden Git-Befehl anhand einer einzigen Frage einordnen: *„Zwischen welchen zwei Bereichen bewegt dieser Befehl etwas – und in*

welche Richtung?“

Die folgenden Abschnitte dieses Kapitels vertiefen genau diese Bewegungen: das Unterscheiden verfolgter und unverfolgter Dateien, selektives und interaktives Staging, das gezielte Verwerfen von Änderungen sowie die Darstellung dieser Zustände direkt in PhpStorm.

Revision #1
Created 2026-07-26 17:14:34 UTC by art10m
Updated 2026-07-26 17:14:42 UTC by art10m