Änderungen mit git diff untersuchen

Änderungen mit `git diff` untersuchen

Ein Commit fasst nur zusammen, *was* sich verändert hat – doch bevor du überhaupt vormerkst oder committest, willst du genau wissen, *welche* Zeilen betroffen sind. Genau dafür ist `git diff` gedacht: Es macht Unterschiede zwischen zwei Zuständen deines Projekts sichtbar, ohne selbst irgendetwas zu verändern. `git diff` ist damit ein reines *Lesewerkzeug* – ideal, um vor jedem `git add` und jedem Commit einen letzten Kontrollblick zu werfen.

Drei Bereiche, drei Vergleiche

Erinnere dich an das Drei-Bereiche-Modell aus dem vorherigen Kapitel: Arbeitsverzeichnis, Index (Staging Area) und Repository. `git diff` vergleicht standardmäßig genau zwei dieser Bereiche – je nach Aufruf unterschiedliche.

Befehl	Vergleicht	Beantwortet die Frage
`git diff`	Arbeitsverzeichnis ↔ Index	„Was habe ich verändert, aber noch nicht vorgemerkt?“
`git diff --staged` (oder `--cached`)	Index ↔ letzter Commit	„Was würde ein Commit jetzt genau festhalten?“
`git diff HEAD`	Arbeitsverzeichnis ↔ letzter Commit	„Was unterscheidet sich insgesamt vom letzten Commit – egal ob vorgemerkt oder nicht?“

Diese Unterscheidung ist der Schlüssel zum Verständnis: `git status` sagt dir nur, *dass* etwas verändert wurde, `git diff` zeigt dir *wie*.

Der Standardfall: Arbeitsverzeichnis gegen Index

Nimm an, du hast eine Datei `index.php` bearbeitet, aber noch nichts mit `git add` vorgemerkt:

```
git diff
```

Die Ausgabe könnte so aussehen:

```
diff --git a/index.php b/index.php
index e69de29..f2c1a3d 100644
--- a/index.php
+++ b/index.php
@@ -1,3 +1,5 @@
 <?php
-echo "Hallo Welt";
+echo "Hallo, Git!";
+
+// Neue Zeile für die Begrüßung
```

Diese Ausgabe folgt dem sogenannten **Unified-Diff-Format**, das nicht nur Git, sondern auch viele andere Werkzeuge verwenden. Es lohnt sich, jeden Bestandteil einmal genau zu lesen:

- **`diff --git a/... b/...`** – kennzeichnet die verglichenen Dateien; `a` steht für die alte, `b` für die neue Version.
- **`index e69de29..f2c1a3d`** – zeigt die betroffenen Objekt-Hashes (dazu mehr im Kapitel über Git-Interna).
- **`---`** / **`+++`** – markieren die alte bzw. neue Dateiversion.
- **`@@ -1,3 +1,5 @@`** – der sogenannte *Hunk-Header*. Er bedeutet: „Ab Zeile 1 wurden in der alten Datei 3 Zeilen, in der neuen Datei 5 Zeilen betrachtet.“
- **Zeilen mit `-`** wurden entfernt, Zeilen mit `+` wurden hinzugefügt, unveränderte Zeilen erscheinen ohne Präfix als Kontext.

Diese Kontextzeilen sind wichtig: Sie helfen dir, die Änderung im umgebenden Code einzuordnen, ohne die ganze Datei öffnen zu müssen.

Vorgemerkte Änderungen prüfen: `--staged`

Sobald du `git add` ausgeführt hast, zeigt der einfache `git diff` nichts mehr an – die Änderung liegt jetzt im Index, nicht mehr nur im Arbeitsverzeichnis. Um zu sehen, was genau ein Commit übernehmen würde, verwendest du:

```
git diff --staged
```

`--cached` ist ein reines Synonym und liefert dasselbe Ergebnis:

```
git diff --cached
```

Diese Unterscheidung zwischen *unstaged* und *staged* Diffs ist besonders bei selektivem Staging wertvoll (siehe Kapitel 5), wenn du bewusst nur einen Teil deiner Änderungen committen möchtest.

Alles auf einen Blick: `git diff HEAD`

Willst du unabhängig vom Staging-Status wissen, was sich seit dem letzten Commit insgesamt verändert hat – egal ob vorgemerkt oder nicht – hilft:

```
git diff HEAD
```

`HEAD` ist dabei ein Zeiger auf den aktuellen Commit, mit dem du später noch viel intensiver arbeiten wirst.

Einzelne Dateien oder Verzeichnisse vergleichen

`git diff` lässt sich immer auf konkrete Pfade einschränken. Das ist besonders bei größeren Projekten hilfreich, um den Fokus zu behalten:

```
git diff index.php
git diff src/
```

Mehrere Pfade lassen sich einfach aneinanderhängen:

```
git diff README.md composer.json
```

Kompakte Übersichten statt vollständiger Diffs

Manchmal willst du nicht jede Zeile sehen, sondern nur wissen, *welche* Dateien betroffen sind und *wie stark*:

```
git diff --stat
```

```
 index.php | 4 ++--
 README.md | 2 ++
 2 files changed, 4 insertions(+), 2 deletions(-)
```

Für eine reine Auflistung der geänderten Dateinamen eignen sich:

```
git diff --name-only
git diff --name-status
```

`--name-status` ergänzt zusätzlich ein Kürzel wie `M` (modified), `A` (added) oder `D` (deleted) – praktisch für Skripte oder einen schnellen Überblick.

Wortweise statt zeilenweise vergleichen

Bei Prosatexten, Dokumentation oder Konfigurationsdateien ist eine zeilenbasierte Darstellung oft unübersichtlich, wenn nur ein einzelnes Wort geändert wurde. Hier hilft der Wortmodus:

```
git diff --word-diff
```

Er markiert nicht ganze Zeilen, sondern einzelne geänderte Wortfolgen mit `[-alt-]` und `{+neu+}` – deutlich präziser für kleinteilige Textänderungen.

Vorausblick: Commits und Branches vergleichen

`git diff` beschränkt sich nicht auf die drei lokalen Bereiche. Es lassen sich ebenso zwei beliebige Commits, Branches oder Tags gegenüberstellen, etwa mit:

```
git diff main feature/login
```

Diese Fähigkeit wird im Kapitel „Historie lesen, durchsuchen und vergleichen“ ausführlich vertieft – an dieser Stelle reicht es, zu wissen, dass `git diff` dafür dieselbe Syntax und dieselbe Ausgabe-Logik verwendet wie beim Vergleich von Arbeitsverzeichnis und Index.

Diff-Ausgaben besser lesbar machen

Auf vielen Windows-Terminals lohnt es sich, farbige Ausgaben zu aktivieren, falls sie nicht bereits standardmäßig aktiv sind:

```
git config --global color.diff auto
```

Für sehr lange Diffs kann zusätzlich ein Pager wie `less` hilfreich sein, den Git unter Windows über Git Bash automatisch mitbringt. Falls die Ausgabe im Terminal umbricht oder unübersichtlich wirkt, ist das ein guter Moment, um zur grafischen Diff-Ansicht in PhpStorm zu wechseln – dort erscheinen alte und neue Version übersichtlich nebeneinander, farblich hervorgehoben und mit Navigationspfeilen zwischen den Änderungsblöcken. Wie dieser Workflow konkret aussieht, zeigt der letzte Abschnitt dieses Kapitels.

Zusammenfassung

`git diff` ist dein wichtigstes Werkzeug, um Änderungen *vor* dem Staging und *vor* dem Commit kritisch zu prüfen. Die zentrale Faustregel lautet:

- **Kein Argument** → Arbeitsverzeichnis gegen Index (unstaged Änderungen).
- **`--staged`** / **`--cached`** → Index gegen letzten Commit (was der nächste Commit enthält).
- **`HEAD`** → Arbeitsverzeichnis gegen letzten Commit (alles zusammen).

Wer diese drei Varianten sicher unterscheidet, vermeidet eine der häufigsten Anfängerverwirrungen in Git: die Frage, *warum* eine erwartete Änderung im Diff plötzlich nicht mehr auftaucht, obwohl die Datei sichtbar verändert wurde. □□

Revision #1
Created 2026-07-26 17:13:03 UTC by art10m
Updated 2026-07-26 17:13:17 UTC by art10m