Den ersten Commit erstellen

Vom Index zum Commit

Nachdem du Dateien mit `git add` in die *Staging Area* aufgenommen hast, liegt der nächste Schritt nahe: Aus den vorgemerkten Änderungen wird ein **Commit** – ein unveränderlicher Schnappschuss deines Projekts zu einem bestimmten Zeitpunkt. Anders als der Name „Speichern“ vermuten lässt, erzeugt ein Commit keine einzelne Momentaufnahme einer Datei, sondern einen vollständigen, in sich geschlossenen Zustand des gesamten Repositorys.

Technisch betrachtet geschieht dabei Folgendes:

1. Git erstellt aus dem Inhalt der Staging Area einen sogenannten *Tree*, der die aktuelle Verzeichnisstruktur abbildet.
2. Ein neues *Commit-Objekt* verweist auf diesen Tree, auf den vorherigen Commit (den *Parent*) sowie auf Autor, Zeitstempel und Nachricht.
3. Der aktuelle Branch-Zeiger – meist `main` – wird auf dieses neue Commit-Objekt verschoben.

Diese internen Zusammenhänge vertiefst du später in Kapitel 22 „Git unter der Haube“. Für den praktischen Einstieg reicht es, den Commit als bewussten, benannten Kontrollpunkt in der Projektgeschichte zu verstehen.

Voraussetzung: Ein saubere Staging Area

Bevor du committest, lohnt sich ein kurzer Blick mit `git status`:

```
git status
```

```
Auf Branch main


Zum Commit vorgemerkte Änderungen:
```

```
  (benutze "git restore --staged <Datei>..." zum Entfernen aus der Staging Area)
        neue Datei:     README.md
```

Nur was hier unter *„Zum Commit vorgemerkte Änderungen“* aufgeführt ist, wird tatsächlich Teil des Commits. Dateien im Arbeitsverzeichnis, die noch nicht mit `git add` vorgemerkt wurden, bleiben unberührt – ein wichtiger Sicherheitsmechanismus, den du bereits in Kapitel 5 kennengelernt hast.

Den Commit erstellen

Mit einer kurzen Nachricht direkt in der Kommandozeile

Der schnellste und im Alltag häufigste Weg verwendet die Option `-m`:

```
git commit -m "Initiale Projektstruktur hinzufügen"
```

Git bestätigt den Vorgang mit einer kurzen Zusammenfassung:

```
[main (Root-Commit) a1b2c3d] Initiale Projektstruktur hinzufügen
 1 file changed, 12 insertions(+)
 create mode 100644 README.md
```

Diese Ausgabe enthält bereits drei wichtige Informationen:

- den **Branch**, auf dem der Commit liegt (`main`),
- den Hinweis **Root-Commit**, da es sich um den allerersten Commit im Repository handelt,
- die ersten Zeichen des **Commit-Hashes** (`a1b2c3d`), über den sich der Commit später eindeutig identifizieren lässt.

Mit dem konfigurierten Editor

Lässt du `-m` weg, öffnet Git den in Kapitel 2 festgelegten Standard-Editor:

```
git commit
```

Das ist besonders bei umfangreicheren Änderungen sinnvoll, da sich im Editor eine mehrzeilige Nachricht mit Betreff, Leerzeile und ausführlicher Beschreibung formulieren lässt:

```
Formular zur Benutzerregistrierung hinzufügen


Implementiert Validierung für E-Mail und Passwort.
Bezieht sich auf Issue #12.
```

Speicherst du die Datei und schließt den Editor, wird der Commit erstellt. Ein leerer Editorinhalt bricht den Vorgang dagegen ab – eine bewusste Schutzmaßnahme gegen Commits ohne Nachricht.

> **Hinweis für Windows 11:** Falls sich kein Editor öffnet oder Git Bash mit einer Fehlermeldung wie `error: cannot spawn` reagiert, wurde der Editor in `core.editor` vermutlich falsch konfiguriert. Kapitel 2 zeigt, wie du das korrigierst.

Anforderungen an eine gute Commit-Nachricht

Eine ausführliche Behandlung von Commit-Konventionen folgt in Kapitel 6. Für den Einstieg gelten drei einfache Regeln:

Regel	Beispiel
Im Imperativ formulieren	„Login-Validierung hinzufügen“ statt „Habe Validierung hinzugefügt“
Kurz und aussagekräftig	Erste Zeile idealerweise unter 50–72 Zeichen
Eine Änderung, eine Absicht	Nicht mehrere unabhängige Themen in einem Commit vermischen

Eine unpräzise Nachricht wie `"update"` oder `"fix"` mag im Moment praktisch erscheinen, erschwert aber jede spätere Auswertung mit `git log` oder `git blame` erheblich.

Wer steckt hinter einem Commit?

Jeder Commit speichert automatisch Autor und Zeitstempel – basierend auf den Werten, die du in Kapitel 2 mit `git config` gesetzt hast:

```
git config user.name
git config user.email
```

Diese Angaben lassen sich für einen einzelnen Commit auch gezielt überschreiben, etwa bei Pair-Programming oder Testcommits:

```
git commit -m "Testcommit" --author="Testperson <test@example.com>"
```

Der Unterschied zwischen *Autor* und *Committer* – relevant etwa bei Rebase oder Cherry-Pick – wird in Kapitel 6 vertieft.

Den Commit überprüfen

Direkt nach dem Erstellen solltest du dir zur Gewohnheit machen, das Ergebnis zu kontrollieren.

Historie ansehen:

```
git log
```

```
commit a1b2c3d4e5f67890abcdef1234567890abcdef1
Author: Max Mustermann <max@example.com>
Date:   Mon Jan 8 10:15:32 2025 +0100

    Initiale Projektstruktur hinzufügen
```

Inhalt des Commits im Detail prüfen:

```
git show a1b2c3d
```

Dieser Befehl zeigt sowohl die Metadaten als auch den vollständigen Diff der im Commit enthaltenen Änderungen – nützlich, um zu bestätigen, dass tatsächlich nur die gewünschten Anpassungen erfasst wurden.

Arbeitsverzeichnis erneut prüfen:

```
git status
```

```
Auf Branch main
nichts zu committen, Arbeitsverzeichnis unverändert
```

Diese Meldung zeigt an, dass alle vorgemerkten Änderungen erfolgreich in die Historie übernommen wurden.

Typische Stolperfallen

- **„Nothing to commit“ trotz Änderungen:** Die Dateien wurden verändert, aber nicht mit `git add` vorgemerkt. Ein Blick auf `git status` klärt sofort, ob Änderungen noch als *„nicht vorgemerkt“* geführt werden.
- **Leerer Commit gewünscht:** In seltenen Fällen – etwa um einen CI-Lauf auszulösen – lässt sich ein Commit ohne Inhaltsänderung mit `git commit --allow-empty -m "..."` erzwingen.
- **Falsche Nachricht direkt nach dem Commit:** Kein Grund zur Sorge – Kapitel 6 zeigt, wie sich der letzte Commit mit `git commit --amend` sauber korrigieren lässt, solange er noch nicht veröffentlicht wurde.
- **Zeilenenden-Warnungen unter Windows:** Meldungen wie `warning: LF will be replaced by CRLF` sind normal und resultieren aus der in Kapitel 2 behandelten `core.autocrlf`-Konfiguration. Sie verhindern den Commit nicht.

Ausblick

Damit besitzt dein Repository seinen ersten dokumentierten Zustand. Im weiteren Verlauf dieses Kapitels lernst du, wie sich diese Historie mit `git log` gezielt lesen lässt, wie Änderungen zwischen Commits mit `git diff` sichtbar werden – und wie sich exakt derselbe Ablauf komfortabel über die Commit-Oberfläche von PhpStorm abbilden lässt, ohne die Kommandozeile zu verlassen.

Revision #1
Created 2026-07-26 17:12:31 UTC by art10m
Updated 2026-07-26 17:12:41 UTC by art10m