Warum Software versioniert wird

Lernziele

Nach dieser Lektion kannst du erklären,

- warum Softwareprojekte eine nachvollziehbare Historie benötigen,
- welche Risiken ohne Versionskontrolle entstehen,
- warum Kopien wie `projekt_final_neu_wirklichfinal` keine verlässliche Lösung sind,
- wie Versionierung Zusammenarbeit, Qualität und Sicherheit unterstützt,
- weshalb Git nicht nur für große Teams, sondern auch für Einzelentwickler wertvoll ist.

Software verändert sich ständig

Software ist kein statisches Dokument. Schon ein kleines Projekt entwickelt sich fortlaufend weiter:

- Funktionen kommen hinzu.
- Fehler werden korrigiert.
- Abhängigkeiten werden aktualisiert.
- Konfigurationen ändern sich.
- Dateien werden umbenannt, verschoben oder gelöscht.
- Anforderungen ändern sich nach Gesprächen mit Kunden, dem Team oder dem Fachbereich.

Eine einzelne Datei kann dabei dutzende oder hunderte Änderungen durchlaufen. Bei einem PHP-Projekt betrifft das häufig nicht nur den Anwendungscode, sondern auch Tests, Composer-Konfiguration, Datenbankmigrationen, Dokumentation und CI-Konfiguration.

Die entscheidende Frage lautet daher nicht, *ob* sich ein Projekt verändert, sondern:

```
```

> **Wie lässt sich nachvollziehen, was sich wann, warum und durch wen verändert hat?**

Genau dafür wird Software versioniert.

Das Problem ohne Versionskontrolle

Stell dir vor, du entwickelst eine kleine PHP-Anwendung für eine Terminverwaltung. Zu Beginn existiert nur eine Datei:

```
terminverwaltung/
└── index.php
```

Im Laufe der Zeit ergänzt du Validierungen, einen Datenbankzugriff und ein Login. Vor einer riskanten Änderung erstellst du vorsichtshalber eine Kopie des gesamten Ordners:

```
terminverwaltung/
terminverwaltung_backup/
terminverwaltung_backup_neu/
terminverwaltung_final/
terminverwaltung_final2/
terminverwaltung_final_wirklich/
```

Dieses Vorgehen wirkt zunächst sicher, führt aber schnell zu Problemen.

Unklare Bedeutung von Kopien

An den Ordnernamen ist kaum zu erkennen:

- Welche Version läuft produktiv?
- Welche Kopie enthält den letzten funktionierenden Stand?
- Welche Änderung wurde zwischen `final` und `final2` vorgenommen?
- Welche Version enthält den Fehler?
- Darf ein alter Ordner gelöscht werden?

Der Dateiname oder Ordnername wird zur improvisierten Versionshistorie. Diese Historie ist jedoch unvollständig, fehleranfällig und schwer durchsuchbar.

Änderungen gehen verloren

Wenn du eine Funktion umbaust und später feststellst, dass die frühere Variante besser war, brauchst du den alten Zustand. Ohne Versionskontrolle musst du hoffen, dass eine passende Sicherheitskopie existiert.

Fehlt sie, bleibt nur:

- Änderungen mühsam zurückbauen,
- Code aus Erinnerungen rekonstruieren,
- Dateien vergleichen,
- oder Arbeit erneut erledigen.

Zusammenarbeit überschreibt Arbeit

Arbeiten zwei Personen gleichzeitig an derselben Datei, entsteht ein besonders gefährliches Szenario:

1. Anna bearbeitet `UserService.php`.
2. Ben bearbeitet ebenfalls `UserService.php`.
3. Anna speichert ihre Datei.
4. Ben speichert später seine Datei über Annas Version.

Je nach Arbeitsweise kann Annas Änderung vollständig verschwinden. Selbst wenn beide vorher Kopien angelegt haben, ist später oft unklar, welche Teile aus welcher Datei übernommen werden müssen.

Fehler sind schwer einzugrenzen

Ein Fehler wird am Montag entdeckt. Am Freitag funktionierte noch alles. Inzwischen gab es viele Änderungen.

Ohne Historie ist die Fehlersuche mühsam:

- Welche Dateien wurden seit Freitag verändert?
- Welche konkrete Zeile war die Ursache?
- Wer kennt die fachliche Entscheidung hinter der Änderung?
- Lässt sich der funktionierende Zustand kurzfristig wiederherstellen?

Ohne belastbare Antworten steigt der Zeitaufwand – und das Risiko, beim Reparieren neue Fehler einzubauen.

Versionierung schafft eine überprüfbare Projektgeschichte

Versionskontrolle speichert nicht einfach nur Dateikopien. Sie hält eine **Geschichte bewusst festgehaltener Projektzustände** fest.

Ein solcher gespeicherter Zustand wird in Git später *Commit* genannt. Ein Commit dokumentiert typischerweise:

- welche Dateien geändert wurden,
- welche Inhalte sich geändert haben,
- wann die Änderung gespeichert wurde,
- wer sie erstellt hat,
- und warum sie vorgenommen wurde.

Statt vieler unklarer Projektordner entsteht eine nachvollziehbare Abfolge:

```
Projektbeginn
    ↓
Grundstruktur angelegt
    ↓
Benutzerregistrierung ergänzt
    ↓
Passwortvalidierung korrigiert
    ↓
Tests für Login hinzugefügt
    ↓
Sicherheitslücke geschlossen
```

Die Historie wird damit zu einem technischen Gedächtnis des Projekts.

Die zentralen Gründe für Versionskontrolle

Änderungen nachvollziehen

Eine gute Projektgeschichte beantwortet wichtige Fragen schnell:

- Was wurde verändert?
- Wann wurde es verändert?
- Wer hat es verändert?
- Warum wurde es verändert?
- Zu welchem Ticket, Issue oder Fehlerbericht gehört die Änderung?

Ein Commit kann beispielsweise die Nachricht tragen:

```
Fix: Passwort-Hash vor dem Speichern erzeugen
```

Diese Nachricht ist wesentlich hilfreicher als eine Ordnerkopie namens `backup_17`.

> **Grundsatz:** Eine Änderung ohne nachvollziehbaren Zweck wird später teuer – spätestens bei Wartung, Fehlersuche oder Review.

Frühere Zustände wiederherstellen

Nicht jede Änderung ist erfolgreich. Ein neues Feature kann Fehler verursachen, eine Konfigurationsänderung kann den Build beschädigen oder ein Refactoring kann unerwartete Seiteneffekte haben.

Versionskontrolle ermöglicht es, gezielt zu einem früheren Zustand zurückzukehren oder einzelne ältere Dateiversionen wiederherzustellen.

Das bedeutet nicht, dass man sorglos arbeiten sollte. Es bedeutet aber, dass experimentelles Arbeiten kontrollierbar wird:

- Du kannst eine Idee ausprobieren.
- Du kannst Änderungen vergleichen.
- Du kannst einen Fehler isolieren.

- Du kannst einen funktionierenden Stand wiederherstellen.
- Du kannst eine falsche Entscheidung transparent korrigieren.

Damit wird das Projekt widerstandsfähiger gegenüber menschlichen Fehlern.

Parallel arbeiten

In professionellen Projekten arbeiten mehrere Personen gleichzeitig:

- Entwicklerinnen und Entwickler implementieren Features.
- Tester prüfen Fehlerkorrekturen.
- DevOps-Teams ändern Deployment-Konfigurationen.
- Technische Redaktionen aktualisieren Dokumentation.
- Sicherheitsverantwortliche prüfen Abhängigkeiten und Richtlinien.

Versionskontrolle koordiniert diese parallele Arbeit. Sie erkennt, wenn Änderungen unabhängig voneinander kombiniert werden können, und macht sichtbar, wenn zwei Personen denselben Bereich unterschiedlich verändert haben.

Solche Überschneidungen heißen später **Konflikte**. Ein Konflikt ist keine Beschädigung des Projekts, sondern ein Hinweis:

> Git kann nicht zuverlässig entscheiden, welche von zwei konkurrierenden Änderungen fachlich richtig ist.

Menschen treffen diese Entscheidung bewusst. Das ist deutlich sicherer, als Änderungen unbemerkt zu überschreiben.

Fehler schneller finden

Eine Projektgeschichte hilft nicht nur beim Zurücksetzen. Sie ist auch ein Diagnosewerkzeug.

Angenommen, eine API-Anfrage liefert seit kurzem falsche Daten. Mit einer Historie kannst du untersuchen:

1. Wann trat der Fehler erstmals auf?
2. Welche Änderungen wurden kurz davor integriert?
3. Welche Datei oder Funktion wurde verändert?
4. Welche Person kennt den fachlichen Hintergrund?
5. Welcher Commit führte die fehlerhafte Änderung ein?

Git bietet dafür später Funktionen wie Log-Ansichten, Diffs, Dateihistorien und `git bisect`. Sie helfen dabei, Fehler systematisch einzugrenzen, statt nur Vermutungen anzustellen.

Code-Reviews ermöglichen

Bevor eine Änderung in den Hauptzweig eines Projekts gelangt, sollte sie oft von einer anderen Person geprüft werden. Dieser Prozess heißt **Code Review**.

Ein Review funktioniert nur, wenn klar erkennbar ist:

- Welche Änderung wird vorgeschlagen?
- Welche Dateien sind betroffen?
- Was war das Ziel?
- Welche Tests wurden ergänzt oder ausgeführt?
- Welche Auswirkungen sind zu erwarten?

Versionskontrolle liefert dafür eine präzise Änderungsmenge. Statt ein gesamtes Projekt manuell zu vergleichen, prüfen Reviewer gezielt die Unterschiede zwischen zwei definierten Ständen.

Das verbessert nicht nur die Codequalität. Reviews fördern auch Wissenstransfer, gemeinsame Standards und frühzeitige Sicherheitsprüfungen.

Releases reproduzierbar machen

Software wird häufig in klaren Versionen veröffentlicht, etwa als `1.0.0`, `1.1.0` oder `2.0.0`.

Ohne Versionskontrolle ist es schwierig, sicher zu beantworten:

- Welcher Quellcode wurde für Version `1.4.2` ausgeliefert?
- Welche Konfiguration gehörte zu diesem Release?
- Welche Fehlerkorrektungen sind darin enthalten?
- Kann derselbe Stand erneut gebaut werden?
- Lässt sich ein Hotfix auf genau dieser Basis entwickeln?

Mit Versionierung kann ein Release eindeutig an einen bestimmten Projektzustand gebunden werden. Später werden dafür Git-Tags verwendet.

Das ist besonders wichtig, wenn eine ältere Produktversion weiterhin gewartet werden muss.

Wissen dauerhaft im Projekt festhalten

Teams verändern sich. Personen wechseln Aufgaben, verlassen ein Unternehmen oder erinnern sich nach Monaten nicht mehr an jede Entscheidung.

Eine gute Historie ergänzt Dokumentation. Sie kann beispielsweise zeigen:

- warum eine Validierung absichtlich streng ist,
- weshalb eine Abhängigkeit aktualisiert wurde,
- warum eine scheinbar unnötige Sonderbehandlung existiert,
- welches Issue einen Fehler beschrieben hat,
- welche Sicherheitslücke geschlossen wurde.

Natürlich ersetzt eine Commit-Nachricht keine vollständige Architektur- oder Fachkonzeptdokumentation. Sie hält aber den Zusammenhang zwischen einer konkreten Codeänderung und ihrer Entscheidung fest.

Versionskontrolle ist mehr als ein Backup

Backups und Versionskontrolle haben unterschiedliche Aufgaben.

Aspekt	Backup	Versionskontrolle
Hauptziel	Datenverlust verhindern	Änderungen nachvollziehbar verwalten
Typischer Umfang	Gesamter Datenbestand	Projektdateien und ihre Historie
Zeitpunkt	Meist automatisch oder periodisch	Bewusst bei fachlich sinnvollen Änderungen
Vergleich einzelner Änderungen	Oft umständlich	Zentrale Funktion
Zusammenarbeit	Nicht dafür ausgelegt	Zentrale Funktion
Rückkehr zu einer bestimmten Änderung	Häufig schwierig	Gezielt möglich
Dokumentation des Zwecks	Meist nicht vorhanden	Commit-Nachrichten und Referenzen

Ein Backup schützt beispielsweise vor einem defekten Datenträger, Ransomware oder versehentlich gelöschten Daten. Git schützt nicht zuverlässig vor allen diesen Risiken, insbesondere nicht allein auf einem einzelnen Computer.

Umgekehrt beantwortet ein Backup normalerweise nicht die Frage:

> „Welche Zeilen wurden geändert, als die Passwortvalidierung angepasst wurde?“

Die professionelle Praxis lautet deshalb:

> **Versionskontrolle und Backups ergänzen einander – keines ersetzt das andere.**

Ein Beispiel aus dem PHP-Alltag

Nehmen wir an, eine Methode zur E-Mail-Prüfung wird geändert.

Vorher:

```
function isValidEmail(string $email): bool
{
    return str_contains($email, '@');
}
```

Nachher:

```
function isValidEmail(string $email): bool
{
    return filter_var($email, FILTER_VALIDATE_EMAIL) !== false;
}
```

Ohne Versionskontrolle ist später nur sichtbar, wie die Datei *jetzt* aussieht. Der frühere Ansatz, der Zeitpunkt der Änderung und die Motivation gehen verloren.

Mit Versionskontrolle kann die Änderung als eigener Commit festgehalten werden:

```
Fix: E-Mail-Adressen mit PHP-Filter validieren
```

Zusätzlich können Tests im selben Commit dokumentieren, welche Fälle berücksichtigt wurden. Später lässt sich eindeutig nachvollziehen:

- welche Implementierung ersetzt wurde,
- warum die Änderung erfolgte,

- welche Dateien dazugehören,
- und ob die Änderung mit einem Bug-Report verknüpft war.

Versionierung unterstützt kleine und große Projekte

Ein verbreiteter Irrtum lautet:

> „Git brauche ich erst, wenn mehrere Personen am Projekt arbeiten.“

Tatsächlich profitieren Einzelentwickler besonders früh von Versionierung.

Für Einzelentwickler

Git hilft dir,

- Experimente sicher auszuprobieren,
- Änderungen in kleinen Schritten festzuhalten,
- Fehler auf frühere Änderungen zurückzuführen,
- zwischen mehreren Aufgaben zu wechseln,
- ältere Zustände wiederzufinden,
- und ein Portfolio mit nachvollziehbarer Entwicklung zu pflegen.

Für Teams

Zusätzlich unterstützt Git,

- parallele Feature-Entwicklung,
- Pull Requests und Code Reviews,
- geregelte Freigaben,
- Release-Management,
- CI-Prüfungen,
- Branch-Schutz,
- und eine gemeinsame, transparente Projektgeschichte.

Der Einstieg lohnt sich daher bereits beim ersten ernsthaften Projekt.

Was genau sollte versioniert werden?

Grundsätzlich gehören Dateien in die Versionskontrolle, wenn sie nötig sind, um das Projekt zu verstehen, zu entwickeln, zu testen oder reproduzierbar zu bauen.

Für ein PHP-Projekt sind das häufig:

```
src/
tests/
config/
public/
composer.json
composer.lock
phpunit.xml
README.md
.github/
```

Nicht versioniert werden sollten typischerweise Dateien, die lokal erzeugt werden, vertrauliche Daten enthalten oder sich jederzeit reproduzieren lassen, beispielsweise:

```
vendor/
.idea/workspace.xml
.env
var/cache/
var/log/
```

Welche Dateien ignoriert werden sollen, hängt vom Projekt ab. Die Regeln dafür werden später mit `.gitignore` systematisch behandelt.

Wichtig ist zunächst das Prinzip:

> **Versioniere den gemeinsamen, nachvollziehbaren Projektzustand – nicht persönliche Arbeitsreste oder geheime Zugangsdaten.**

Gute Versionierung beginnt mit kleinen, sinnvollen Schritten

Versionskontrolle entfaltet ihren größten Nutzen, wenn Änderungen klar abgegrenzt werden.

Ungünstig wäre ein riesiger Sammel-Commit wie:

```
Diverse Änderungen
```

Darin könnten sich gleichzeitig befinden:

- ein neues Feature,
- eine Fehlerkorrektur,
- Formatierungsänderungen,
- aktualisierte Abhängigkeiten,
- und eine versehentlich eingecheckte Konfigurationsdatei.

Besser sind kleine, logisch zusammenhängende Schritte:

```
Feat: Terminvalidierung ergänzen
Test: Ungültige Endzeiten abdecken
Fix: Zeitzone beim Speichern berücksichtigen
Docs: API-Beispiel für Terminanlage ergänzen
```

Dadurch wird die Historie lesbar, überprüfbar und leichter rückgängig zu machen.

Git als Werkzeug für diese Aufgabe

Git ist ein verteiltes Versionskontrollsystem. Es speichert die Projektgeschichte lokal auf deinem Computer und kann sie mit anderen Repositories austauschen, etwa über GitHub.

Dabei ist Git nicht einfach ein Cloud-Speicher und auch nicht nur ein Werkzeug für Befehlszeilenprofis. Es bildet die Grundlage für viele alltägliche Entwicklungsabläufe:

- lokale Änderungen sicher festhalten,
- neue Entwicklungszweige erstellen,
- Änderungen anderer abrufen,
- Pull Requests prüfen,
- Releases markieren,

- und Fehler bis zu ihrer Ursache zurückverfolgen.

PhpStorm stellt viele dieser Funktionen grafisch bereit. Dennoch ist es wichtig, die zugrunde liegenden Konzepte zu verstehen. Dann kannst du sicher entscheiden, welche Aktion sinnvoll ist – unabhängig davon, ob du die IDE oder die Kommandozeile verwendest.

Merksätze

- **Softwareentwicklung ist Veränderung.** Ohne Historie werden Veränderungen schnell unübersichtlich.
- **Versionskontrolle beantwortet Fragen.** Was, wann, wer und warum sind zentrale Informationen.
- **Git ist kein Ersatz für Backups.** Beide Schutzmechanismen werden benötigt.
- **Auch Einzelentwickler profitieren.** Git macht Experimente, Korrekturen und Fehlersuche sicherer.
- **Kleine, zusammenhängende Änderungen erzeugen eine wertvolle Historie.**
- **Versionskontrolle macht Zusammenarbeit kontrollierbar, nicht automatisch konfliktfrei.**
- **Ein Commit ist ein bewusst dokumentierter Projektzustand**, nicht bloß ein beliebiger Speichervorgang.

Selbstcheck

Prüfe dein Verständnis mit diesen Fragen:

1. Warum sind Ordnerkopien mit Namen wie `projekt_final2` keine zuverlässige Versionsstrategie?
2. Welche Fragen sollte eine gute Projektgeschichte beantworten können?
3. Worin unterscheidet sich Versionskontrolle von einem Backup?
4. Warum ist Git auch bei einem Solo-Projekt sinnvoll?
5. Weshalb sollten unabhängige Änderungen nicht in einem einzigen großen Commit zusammengefasst werden?
6. Welche Arten von Dateien gehören in einem PHP-Projekt normalerweise *nicht* ins Repository?

Im nächsten Abschnitt wird betrachtet, wie Änderungen, Versionen und Projektgeschichte konkret zusammenhängen.

Revision #1
Created 2026-07-25 11:13:38 UTC by art10m
Updated 2026-07-25 11:13:38 UTC by art10m