Git und GitHub unterscheiden

Zwei Begriffe, zwei Aufgaben

Git und **GitHub** werden oft in einem Atemzug genannt, sind aber grundverschiedene Dinge:

- **Git** ist ein Versionskontrollsystem. Es läuft primär lokal auf deinem Computer und verwaltet die Geschichte deiner Dateien.
- **GitHub** ist eine Online-Plattform, die Git-Repositories hostet und Zusammenarbeit, Reviews, Automatisierung und Projektorganisation ergänzt.

Die Kurzform lautet:

> **Git verwaltet Versionen. GitHub verbindet Menschen und Repositories.**

Du kannst Git vollständig ohne GitHub einsetzen. Umgekehrt basiert ein GitHub-Repository in der Regel auf Git.

Git: das lokale Versionskontrollsystem

Git ist freie Software, die von Linus Torvalds entwickelt wurde. Es speichert die Entwicklung eines Projekts als Folge von **Commits**. Jeder Commit beschreibt einen nachvollziehbaren Stand des Projekts.

Git arbeitet in einem lokalen Repository auf deinem Rechner. Sobald du ein Repository mit `git init` anlegst oder mit `git clone` kopierst, enthält dein Projekt einen versteckten Ordner namens `.git`. Darin liegen unter anderem:

- die Commit-Historie,
- Branches und Tags,
- die Staging Area,
- lokale Konfigurationen,
- Referenzen auf entfernte Repositories,
- Informationen zur Wiederherstellung über Reflogs.

Ein Repository kann deshalb auch ohne Internetverbindung sinnvoll genutzt werden.

Typische Git-Aufgaben

Mit Git erledigst du beispielsweise diese Arbeitsschritte:

```
git status
git add src/Calculator.php
git commit -m "Füge Addition hinzu"
git switch -c feature/subtraction
git merge feature/subtraction
git log --oneline
```

Diese Befehle funktionieren **lokal**. Sie benötigen weder ein GitHub-Konto noch eine Netzwerkverbindung.

Git ist kein Server und keine Website

Git selbst stellt keine zentrale Website, keine Benutzerverwaltung und keine Pull-Request-Oberfläche bereit. Es kennt zwar entfernte Repositories, sogenannte **Remotes**, aber Git schreibt nicht vor, wo diese liegen.

Ein Remote kann zum Beispiel sein:

- ein Repository auf GitHub,
- ein Repository auf GitLab oder Bitbucket,
- ein Git-Server im Unternehmen,
- ein Repository auf einem eigenen Server,
- ein Verzeichnis in einem lokalen Netzwerk,
- theoretisch sogar ein anderer Ordner auf demselben Computer.

GitHub: eine Plattform rund um Git

GitHub ist ein kommerzieller Cloud-Dienst von GitHub, Inc., das zu Microsoft gehört. Die Plattform speichert Git-Repositories auf Servern und ergänzt Git um Funktionen für Zusammenarbeit und Projektmanagement.

Ein Repository auf GitHub ist also ein **entferntes Git-Repository**, das über das Internet erreichbar ist. Üblicherweise erhält es den Remote-Namen `origin`.

```
git remote -v
```

Eine typische Ausgabe sieht so aus:

```
origin  git@github.com:beispielkonto/mein-projekt.git (fetch)
origin  git@github.com:beispielkonto/mein-projekt.git (push)
```

Die URL zeigt: Das lokale Repository ist mit einem Repository auf GitHub verbunden.

Funktionen, die GitHub zusätzlich liefert

GitHub erweitert reines Git um eine umfangreiche Web-Plattform:

- **Repository-Hosting** für öffentliche und private Projekte
- **Pull Requests** für vorgeschlagene Änderungen und Code Reviews
- **Issues** für Fehler, Aufgaben und Feature-Wünsche
- **GitHub Actions** für Tests, Builds, Deployments und Releases
- **Projects** für Planung und Projektübersichten
- **Discussions** für langfristige technische oder Community-Diskussionen
- **Wiki** für ergänzende Dokumentation
- **Teams, Rollen und Berechtigungen** für Organisationen
- **Branch-Schutzregeln** und verpflichtende Prüfungen
- **Security-Funktionen**, etwa Secret Scanning, Dependabot und Code Scanning
- **Releases** mit Versionshinweisen und Dateien zum Herunterladen

Keine dieser Funktionen ist Bestandteil von Git selbst.

Der zentrale Vergleich

Aspekt	Git	GitHub
Art	Versionskontrollsystem	Hosting- und Kollaborationsplattform
Läuft primär	Lokal auf deinem Rechner	Auf Servern und im Webbrowser

Aspekt	Git	GitHub
Internet erforderlich	Nein, für lokale Arbeit nicht	Ja, normalerweise für die Nutzung
Kernaufgabe	Dateien, Commits, Branches und Historie verwalten	Repositories teilen und Zusammenarbeit organisieren
Wird installiert	Ja, beispielsweise über Git for Windows	Nein, Nutzung per Browser, API oder Client
Kennt Pull Requests	Nein	Ja
Kennt Issues und Projektboards	Nein	Ja
Kann allein verwendet werden	Ja	Nein, Git ist die technische Grundlage
Beispiele für Alternativen	Mercurial, Subversion	GitLab, Bitbucket, Azure Repos, Gitea

---

Ein praktisches Bild: Notizbuch und gemeinsamer Arbeitsraum

Eine hilfreiche Analogie ist ein persönliches Notizbuch:

- **Git** ist dein Notizbuch mit einer lückenlosen Versionsgeschichte. Du kannst Seiten ergänzen, Fehler korrigieren, Kapitel verzweigen und frühere Stände nachschlagen.
- **GitHub** ist ein gemeinsamer Arbeitsraum mit Kopien dieser Notizbücher. Dort können andere Personen Änderungen ansehen, kommentieren, prüfen, zusammenführen und automatisiert testen lassen.

Das Notizbuch funktioniert auch dann, wenn du allein arbeitest oder keine Internetverbindung hast. Der gemeinsame Arbeitsraum wird wichtig, sobald du dein Projekt sichern, veröffentlichen oder mit anderen entwickeln möchtest.

Diese Analogie hat allerdings eine Grenze: GitHub speichert nicht nur eine einfache Sicherungskopie. Es ist ein vollwertiges, ebenfalls Git-basiertes Repository mit zusätzlicher Plattformfunktionalität.

Ein typischer Ablauf mit Git und GitHub

In der Praxis greifen beide Werkzeuge ineinander.

1. Du erstellst oder klonst ein **lokales Git-Repository**.
2. Du änderst Dateien in deinem Arbeitsverzeichnis.
3. Du nimmst ausgewählte Änderungen mit `git add` in die Staging Area auf.
4. Du erzeugst lokal einen Commit mit `git commit`.
5. Du überträgst den Commit mit `git push` zu GitHub.
6. Auf GitHub erstellst du bei Bedarf einen Pull Request.
7. Teammitglieder prüfen die Änderungen, hinterlassen Kommentare und geben sie frei.
8. GitHub führt den Branch nach den Teamregeln zusammen oder löst automatisierte Prüfungen aus.
9. Du holst die neue gemeinsame Historie mit `git fetch` oder `git pull` zurück auf deinen Computer.

Wichtig ist die Reihenfolge: **Ein Commit entsteht zunächst lokal in Git.** Erst ein Push macht ihn für das Remote-Repository auf GitHub verfügbar.

Lokale und entfernte Historie unterscheiden

Ein häufiger Anfängerfehler besteht darin, die lokale und die entfernte Historie als identisch zu betrachten. Tatsächlich können sie voneinander abweichen.

Angenommen, dein aktueller lokaler Branch `main` enthält einen Commit, den du noch nicht veröffentlicht hast:

```
Lokal:    A — B — C
GitHub:   A — B
```

Commit `C` existiert bereits vollständig in deinem lokalen Git-Repository. Erst mit diesem Befehl wird er nach GitHub übertragen:

```
git push origin main
```

Danach entspricht die Historie wieder einander:

```
Lokal:    A — B — C
GitHub:   A — B — C
```

Andersherum können Teammitglieder Commits auf GitHub veröffentlicht haben, die du noch nicht abgerufen hast. Dann hilft:

```
git fetch origin
```

GitHub ist daher nicht „Git in der Cloud“, sondern ein **entferntes Repository mit einer Kollaborationsplattform darum herum**.

GitHub ist nicht die einzige Git-Plattform

Git ist ein offener Standard und nicht an GitHub gebunden. Ein lokales Git-Repository kann mit unterschiedlichen Anbietern verbunden werden.

Bekannte Alternativen sind:

Plattform	Typischer Einsatz
GitLab	Cloud-Angebot oder selbst betriebene Unternehmensinstanz
Bitbucket	Häufig in Teams mit Atlassian-Werkzeugen wie Jira
Azure Repos	Integration in Microsoft Azure DevOps
Gitea	Schlanke, selbst hostbare Open-Source-Lösung
Forgejo	Community-orientierte, selbst hostbare Git-Plattform

Die grundlegenden Git-Befehle bleiben dabei gleich:

```
git clone
git fetch
git pull
git push
```

Was sich ändert, sind vor allem die Weboberfläche, Rechteverwaltung, Automatisierungen und Begriffe der Plattform. GitHub verwendet beispielsweise den Begriff **Pull Request**. Bei GitLab heißt eine vergleichbare Funktion **Merge Request**.

GitHub ist auch nicht GitHub Desktop

Zusätzlich kann der Name **GitHub Desktop** verwirren.

GitHub Desktop ist eine grafische Anwendung für Git und GitHub. Sie vereinfacht häufige Git-Aktionen, etwa Commits, Branch-Wechsel und Pushes. Sie ist jedoch weder Git selbst noch die GitHub-Webplattform.

Die drei Begriffe lassen sich klar einordnen:

Begriff	Bedeutung
Git	Technisches Versionskontrollsystem
GitHub	Web-Plattform für Git-Hosting und Zusammenarbeit
GitHub Desktop	Grafische Anwendung zur Bedienung von Git und GitHub

Auch PhpStorm ist in diesem Sinn ein Client: Die IDE ruft im Hintergrund Git-Funktionen auf, zeigt ihre Ergebnisse grafisch an und kann zusätzlich mit GitHub verbunden werden.

Was PhpStorm dabei übernimmt

PhpStorm kann sowohl mit lokalem Git als auch mit GitHub arbeiten.

Lokale Git-Funktionen in PhpStorm

Ohne GitHub-Anmeldung kannst du in PhpStorm bereits:

- ein Repository initialisieren,
- Änderungen vergleichen,
- Dateien zur Staging Area hinzufügen,
- Commits erstellen,
- Branches anlegen und wechseln,
- Merges und Rebases durchführen,
- die lokale Historie untersuchen.

Dafür muss Git auf Windows installiert und in PhpStorm konfiguriert sein.

GitHub-Funktionen in PhpStorm

Nach der Verbindung mit einem GitHub-Konto kann PhpStorm zusätzlich:

- Projekte auf GitHub veröffentlichen,
- Repositories klonen,
- Pull Requests erstellen und prüfen,
- Pull-Request-Kommentare lesen und beantworten,
- Issues anzeigen,
- Remotes komfortabel verwalten.

Die IDE ersetzt dabei nicht das Verständnis der zugrunde liegenden Git-Abläufe. Sie bietet lediglich eine andere Bedienoberfläche für viele davon.

Häufige Missverständnisse

„Ich habe GitHub installiert.“

GitHub ist hauptsächlich ein Webdienst und wird nicht wie Git installiert. Möglich ist, dass du stattdessen eines dieser Werkzeuge installiert hast:

- Git for Windows,
- GitHub Desktop,
- die GitHub CLI `gh`,
- eine IDE-Erweiterung oder eine Anmeldung in PhpStorm.

Präziser wäre zum Beispiel: „Ich habe Git for Windows installiert und mein Repository mit GitHub verbunden.“

„Meine Dateien sind sicher auf GitHub, weil ich einen Commit erstellt habe.“

Ein lokaler Commit ist noch nicht auf GitHub gesichert. Erst `git push` überträgt ihn zu einem Remote.

Prüfe den Zustand mit:

```
git status
```

Git meldet häufig ausdrücklich, wenn dein lokaler Branch Commits enthält, die noch nicht übertragen wurden.

„GitHub erstellt meine Commits.“

GitHub kann über die Weboberfläche Commits erzeugen, etwa beim Bearbeiten einer Datei im Browser oder beim Zusammenführen eines Pull Requests. Die meisten Commits erstellst du jedoch lokal mit Git oder über PhpStorm.

„Ohne GitHub kann ich Git nicht nutzen.“

Git funktioniert vollständig ohne GitHub. Das ist nützlich für:

- private Entwürfe,
- Projekte ohne Internetzugang,
- lokale Experimente,
- interne Server,
- vertrauliche Quelltexte,
- Schulungs- und Übungsprojekte.

„Ein Branch auf GitHub ist dasselbe wie mein lokaler Branch.“

Sie sind verwandt, aber nicht identisch. Beispielsweise sind `main` und `origin/main` unterschiedliche Referenzen:

- `main` bezeichnet deinen lokalen Branch.
- `origin/main` repräsentiert den zuletzt bekannten Stand des Branches auf dem Remote `origin`.

Git aktualisiert `origin/main` erst durch Kommunikation mit dem Remote, etwa per `git fetch`.

Die richtige Begriffswahl im Alltag

Eine präzise Sprache verhindert Missverständnisse im Team:

Ungenau	Präziser

„Ich habe es in GitHub committed.“	„Ich habe lokal committed und nach GitHub gepusht.“
„GitHub hat einen Konflikt.“	„Der Pull Request kann nicht automatisch gemergt werden.“
„Ich ziehe GitHub.“	„Ich hole Änderungen vom Remote mit `git pull`.“
„Der Branch ist auf GitHub.“	„Der lokale Branch wurde zu `origin/feature/login` gepusht.“
„Git ist kaputt.“	„Mein lokaler Branch und der Remote-Branch sind auseinander gelaufen.“

Diese Unterscheidung wird besonders wichtig, sobald mehrere Personen gleichzeitig an einem Projekt arbeiten.

Merksatz

> **Git ist das Werkzeug für Versionsgeschichte. GitHub ist ein Ort und eine Plattform für gemeinsame Git-Projekte.**

Wenn du lokal Dateien änderst, stagen, committen, branchen oder zurücksetzen möchtest, arbeitest du primär mit **Git**. Wenn du Änderungen veröffentlichst, Pull Requests prüfst, Issues planst oder automatisierte Tests auf einem Server ausführst, nutzt du Funktionen von **GitHub**.

Im nächsten Schritt lernst du typische Git-Workflows kennen und kannst Git sowie GitHub gezielt für Einzelarbeit und Teamarbeit einordnen.

Revision #1
Created 2026-07-25 11:13:40 UTC by art10m
Updated 2026-07-25 11:13:40 UTC by art10m