

GIT FÜR CLAUDE CODE: DIE VIER ORTE, AN DENEN DEIN CODE LEBT

7 PROMPTS

TOOL: CLAUDE CODE

STAND 2026-10-07

Mit Git bekommst du die Rückgängig-Taste für alles, was du mit Claude Code baust. Der Guide zeigt dir die vier Orte, an denen eine Version von deinem Code leben kann, vom Working Directory auf deinem Rechner bis zum Remote Repository auf GitHub, und welcher Befehl sie verbindet. Dazu bekommst du fertige Sätze für Claude Code, mit denen du speicherst, hochlädst und zu einem alten Stand zurückspringst, ohne einen einzigen Befehl auswendig zu lernen.

4

ORTE, AN DENEN DEIN CODE LEBT

3

BEFEHLE: ADD, COMMIT, PUSH

0 €

FÜR GIT UND GITHUB

WAS DU BRAUCHST

Claude Code: der Programmier-Assistent von Anthropic, der direkt in deinem Projektordner arbeitet. Ursprünglich läuft er im Terminal, dem Textfenster deines Computers. Du schreibst ihm aber einfach auf Deutsch, was er tun soll. Claude Code steckt im Claude-Pro-Abo: 18 € im Monat zzgl. MwSt., für Privatpersonen also 21,42 € im Monat.

Ein GitHub-Konto: kostenlos auf github.com. Der Gratis-Plan enthält unbegrenzt viele private Repositories.

Git: kostenlos. Fehlt es auf deinem Rechner, installiert Claude Code es in Schritt 1 für dich.

Ein Projektordner: zum Beispiel der Ordner mit der Website, die Claude Code für dich gebaut hat.

WARUM DU GIT BRAUCHST

Stell dir vor, du sagst Claude Code: „Bau mir eine Website für mein Business.“ Claude Code legt dann einen Ordner an und schreibt lauter Dateien rein: eine für das Aussehen, eine für die Texte, eine für die Buttons und Funktionen.

Dieser Ordner mit den Dateien heißt **Repository**, kurz **Repo**. Und genau diese Dateien sind deine Website. Sie steuern, wie sie aussieht und was sie kann.

Jetzt sagst du morgen: „Mach alles blau.“ Claude ändert zwanzig Dateien, und plötzlich ist dein Menü kaputt. Und genau dafür gibt es **Git**. Es merkt sich jeden Stand von deinem Projekt, und du kannst jederzeit zurück.

Das Ganze sieht nur kompliziert aus, weil dir keiner sagt, wo dein Code gerade liegt. Es gibt nämlich nur vier Orte, an denen eine Version von deinem Code leben kann. Sobald du die kennst, macht plötzlich jeder Befehl Sinn, den

Claude Code dir hinwirft.

DIE VIER ORTE AUF EINEN BLICK

ORT	WAS DAS IST	BEFEHL, DER DEINEN CODE ZUM NÄCHSTEN ORT BRINGT
1. Working Directory	Der Ordner, in dem du gerade arbeitest	<code>git add</code>
2. Staging Area	Die Vorschau auf deinen nächsten Speicherpunkt	<code>git commit</code>
3. Local Repository	Der Aktenschrank auf deinem Rechner mit allen Commits	<code>git push</code>
4. Remote Repository	Die Kopie von deinem Repository auf GitHub	Endstation: hier ist dein Projekt sicher

Ein **Commit** ist ein Speicherpunkt: ein fester Schnappschuss deines Projekts, zu dem du zurückkannst. Die Befehle in diesem Guide tippst du nicht selbst. Du schreibst Claude Code auf Deutsch, was passieren soll, und Claude führt den passenden Befehl aus. Fragt Claude Code, ob es einen Befehl ausführen darf, bestätigst du mit Ja.

01

GIT IN DEINEM PROJEKT EINSCHALTEN

Öffne Claude Code in dem Ordner, in dem dein Projekt liegt, so wie du es sonst auch startest. Kopier den Prompt unten in den Claude Code Chat und schick ihn ab. Claude prüft, ob Git installiert ist, installiert es bei Bedarf und macht aus deinem Ordner ein Repository.

Ersetze vorher `[Dein Name]` und `[Deine E-Mail]`. Git schreibt beides an jeden Speicherpunkt, damit man sieht, von wem er stammt. Nimm am besten die E-Mail, mit der du dich gleich bei GitHub anmeldest.

Die **.gitignore** ist eine kleine Liste mit Dateien, die Git nie speichern soll. Da gehören Passwörter und API-Schlüssel rein, damit sie nie auf GitHub landen.

GIT EINSCHALTEN

```
Prüf, ob Git auf meinem Rechner installiert ist, und installier es, falls es fehlt.
Mach danach aus diesem Ordner ein Git-Repository.
Trag als Absender für Git meinen Namen [Dein Name] und meine E-Mail [Deine E-Mail] ein.
Leg eine .gitignore an, damit Passwörter, API-Schlüssel, .env-Dateien und unnötige
Systemdateien nie gespeichert werden.
Erklär mir am Ende in zwei Sätzen, was du gemacht hast.
```

01

ORT EINS: DAS WORKING DIRECTORY

Das ist der Ordner, in dem du gerade arbeitest. Also ganz normal Datei öffnen, was ändern, speichern, wie in jedem anderen Programm auch. Nur hat Git davon noch nichts gespeichert.

Stell dir vor, Claude Code ändert dir gerade zwanzig Dateien, und in der Git-Historie steht davon noch nichts. Deswegen sagst du mit `git add`, welche Änderungen beim nächsten Speicherpunkt dabei sein sollen.

Bevor du das machst, lass dir zeigen, was sich überhaupt geändert hat. Dafür nutzt Claude den Befehl `git status`.

ÄNDERUNGEN ANZEIGEN

Zeig mir mit `git status`, welche Dateien sich seit dem letzten Speicherpunkt geändert haben.

Erklär mir zu jeder Datei in einem einfachen Satz, was sich geändert hat und ob die Änderung fertig wirkt oder noch eine halbe Baustelle ist.

01

ORT ZWEI: DIE STAGING AREA

Das ist die Vorschau auf deinen nächsten Speicherpunkt.

Sagen wir, Claude hat fünf Dateien angefasst, aber nur drei davon sind wirklich fertig. Die zwei anderen sind noch halbe Baustellen. Dann packst du nur die drei rein, und die zwei anderen bleiben erstmal draußen.

Genau das macht `git add`.

Trag im Prompt die Dateinamen ein, die Claude dir in Schritt 2 als fertig gezeigt hat. Sind alle Änderungen fertig, schreib stattdessen einfach: „Pack alle geänderten Dateien in die Staging Area.“

NUR FERTIGE DATEIEN AUSWÄHLEN

Pack nur diese Dateien mit `git add` in die Staging Area:

[Datei 1]

[Datei 2]

[Datei 3]

Alle anderen Änderungen bleiben draußen.

Zeig mir danach, was in der Staging Area liegt und was nicht.

01

SPEICHERPUNKT SETZEN MIT GIT COMMIT

Und dann kommt `git commit`. Das macht aus genau diesen drei Dateien einen festen Snapshot, zu dem du jederzeit zurück kannst.

Zu jedem Commit gehört eine kurze Nachricht, was sich geändert hat, zum Beispiel „Menü auf Blau umgestellt“. Die Nachricht ist später dein Wegweiser, wenn du einen alten Stand suchst. Lass Claude sie schreiben, auf Deutsch und konkret.

COMMIT MACHEN

```
Mach einen Commit mit allem, was gerade in der Staging Area liegt.  
Schreib eine kurze, konkrete Commit-Nachricht auf Deutsch, die sagt, was sich geändert  
hat.  
Zeig mir danach die Nachricht und welche Dateien drin sind.
```

01

ORT DREI: DAS LOCAL REPOSITORY

Aber wo wird der Schnappschuss abgelegt? Im Local Repository. Das ist wie ein Aktenschrank auf deinem Rechner. Da drin liegen alle deine Commits, deine Branches, die komplette Geschichte von deinem Projekt.

Jeder Commit ist wie ein Foto von deinem Projekt an dem Tag, an dem du gespeichert hast. Macht Claude Code dir morgen was kaputt, gehst du einfach zum Snapshot von heute zurück. Wie das geht, steht in Schritt 7.

Ein **Branch** ist ein Nebenzweig deines Projekts, auf dem du etwas ausprobieren kannst, ohne die Hauptversion anzufassen. Für den Anfang reicht dir der eine Hauptzweig, den Git automatisch anlegt.

Der Haken: Der Aktenschrank steht nur auf deinem Rechner. Geht dein Laptop kaputt, ist alles weg. Deswegen gibt es `git push`, und der schickt alles online.

SPEICHERPUNKTE ANSEHEN

```
Zeig mir mit git log meine letzten zehn Speicherpunkte.  
Für jeden: Datum, Commit-Nachricht und eine kurze Erklärung in einfachen Worten.
```

ORT VIER: DAS REMOTE REPOSITORY

Das ist eine Kopie von deinem Repository, nur liegt die auf GitHub statt bei dir. Stell dir das vor wie Google Drive für deinen Code. Dein Laptop kann aus dem zehnten Stock fallen, dein Projekt ist trotzdem noch da.

Mit `git push` schickst du deine Commits dorthin. Beim ersten Mal legt Claude dafür auf GitHub ein neues Repository an und verbindet es mit deinem Ordner. Wähle **privat**: Dann sehen den Code nur du und die Leute, die du einlädst.

Muss Claude dich bei GitHub anmelden, sagt es dir, was zu tun ist. Meist öffnet sich ein Browserfenster mit der GitHub-Anmeldung. Melde dich dort an, bestätige den Zugriff und geh zurück zu Claude Code.

Und wenn Claude Code dir ab jetzt sagt „committen und pushen“, weißt du genau, was da passiert: Speicherpunkt in Ort drei setzen, dann ab damit nach Ort vier.

AUF GITHUB HOCHLADEN

```
Leg auf GitHub ein privates Repository mit dem Namen [Projektname] an.  
Verbinde es mit diesem Ordner und schick alle Commits mit git push hoch.  
Wenn ich mich bei GitHub anmelden muss, sag mir Schritt für Schritt, was ich klicken  
soll.  
Gib mir am Ende den Link zu meinem Repository auf GitHub.
```

WENN DAS NICHT KLAPPT

Falls Claude das Repository auf GitHub nicht selbst anlegen kann, mach es so:

1. Geh auf github.com und melde dich an.
2. Klick oben rechts auf das + und dann auf **New repository**.
3. Gib bei **Repository name** einen Namen ein, zum Beispiel `meine-website`. Erlaubt sind Buchstaben ohne Umlaute, Zahlen, Bindestrich, Punkt und Unterstrich.
4. Wähle **Private**. Lass die Optionen für README, .gitignore und License aus, die bringt dein Ordner schon mit.
5. Klick unten auf **Create repository**.
6. Kopier die Adresse, die GitHub dir danach anzeigt (sie endet auf `.git`), und schreib Claude Code: „Verbinde diesen Ordner mit [Adresse] und schick alle Commits mit git push hoch.“

ZURÜCK ZU EINEM ALTEN STAND

Jetzt kommt der Moment, für den du das alles machst: Claude Code hat dir was kaputt gemacht, und du willst zum Stand von gestern zurück.

Sag Claude, welcher Speicherpunkt noch in Ordnung war, also das Datum oder die Commit-Nachricht aus Schritt 5. Der Prompt sorgt dafür, dass Claude mit `git revert` arbeitet. Die Fehler werden mit einem neuen Speicherpunkt rückgängig gemacht, und kein alter Speicherpunkt geht verloren.

Am Ende pusht Claude den reparierten Stand wieder, damit auch die Kopie auf GitHub stimmt.

ZUM ALTEN STAND ZURÜCK

```
Mein Projekt war beim Speicherpunkt [Datum oder Commit-Nachricht] noch in Ordnung.  
Bring den Code auf genau diesen Stand zurück.  
Nutz dafür git revert, damit kein Speicherpunkt gelöscht wird.  
Wenn es noch Änderungen gibt, die nicht committet sind, sag mir das vorher und frag, was  
damit passieren soll.  
Lösch nichts endgültig, ohne mich zu fragen.  
Schick das Ergebnis danach mit git push zu GitHub.
```

DER EHRliche TEIL

Git sichert nur, was du committest. Änderungen im Working Directory oder in der Staging Area sind noch kein Speicherpunkt. Gewöhn dir an, nach jedem fertigen Arbeitsschritt zu committen.

Ohne Push liegt alles nur auf deinem Rechner. Der Aktenschrank aus Ort drei hilft dir nichts, wenn der Laptop weg ist.

Passwörter und API-Schlüssel gehören nie in ein Repository, auch nicht in ein `privates`. Was einmal gepusht ist, bleibt in der Historie. Deshalb die `.gitignore` aus Schritt 1.

Claude Code hat mit `/rewind` eine eigene Rückgängig-Funktion. Die ist für schnelle Korrekturen innerhalb einer Sitzung gedacht. Für eine dauerhafte Versionsgeschichte empfiehlt Anthropic selbst weiter Git.

Kosten: Git und der Gratis-Plan von GitHub kosten nichts. Bezahlen musst du nur Claude Code über das Pro-Abo.

WENN DAS NICHT KLAPPT

„**fatal: not a git repository**“: Claude Code ist im falschen Ordner geöffnet oder Schritt 1 fehlt. Schließ Claude Code, öffne es im richtigen Projektordner und schick den Prompt aus Schritt 1 noch einmal.

„**Author identity unknown**“ oder „**Please tell me who you are**“: Name und E-Mail fehlen. Schreib Claude Code: „Trag als Absender für Git meinen Namen [Dein Name] und meine E-Mail [Deine E-Mail] ein und versuch den Commit noch einmal.“

Der Push wird abgelehnt oder fragt nach einem Passwort: Schreib Claude Code: „Der Push hat nicht geklappt. Lies die Fehlermeldung und führ mich Schritt für Schritt durch die Lösung.“ Gib dabei nie dein GitHub-Passwort in den Chat ein. Die Anmeldung läuft über den Browser.

Claude Code hängt oder reagiert komisch: Claude Code schließen, neu starten und den letzten Prompt noch einmal schicken. Deine Commits bleiben dabei erhalten.

HÄUFIGE FRAGEN

WAS IST DER UNTERSCHIED ZWISCHEN GIT UND GITHUB?

Git ist das Programm auf deinem Rechner, das die Speicherpunkte macht. Das sind die Orte eins bis drei. GitHub ist der Online-Dienst, auf dem die Kopie liegt, also Ort vier.

MUSS ICH DIE GIT-BEFEHLE AUSWENDIG LERNEN?

Nein. Claude Code tippt die Befehle für dich. Wichtig ist nur, dass du weißt, an welchem der vier Orte dein Code gerade liegt. Dann verstehst du jede Rückfrage von Claude.

WIE OFT SOLLTE ICH COMMITTEN?

Immer dann, wenn etwas funktioniert. Ein neuer Abschnitt auf der Website steht, ein Fehler ist behoben: Commit. Lieber viele kleine Speicherpunkte als einen großen am Ende des Tages.

SIEHT JEDER MEINEN CODE AUF GITHUB?

Nur wenn du das Repository öffentlich machst. Ein privates Repository sehen nur du und die Leute, die du einlädst.

SPICKZETTEL

git status — Zeigt, was sich in Ort eins, dem Working Directory, geändert hat.

git add — Schiebt fertige Änderungen in Ort zwei, die Staging Area.

git commit — Macht daraus einen festen Speicherpunkt in Ort drei, dem Local Repository.

git push — Schickt deine Commits nach Ort vier, ins Remote Repository auf GitHub.

git log — Listet alle Speicherpunkte aus deinem Aktenschränk.

git revert — Macht einen alten Commit mit einem neuen Commit rückgängig, ohne etwas zu löschen.

HINWEIS

Hinweis: Dieser Guide dient zu Informations- und Bildungszwecken. Alle Inhalte spiegeln den Stand bei Erstellung wider, Oberflächen, Funktionen und Preise der genannten Tools können sich ändern. Prüfe Ergebnisse von KI-Tools immer selbst nach. @GPTMarlon übernimmt keine Haftung für Entscheidungen auf Grundlage dieser Inhalte. Dieser Guide stellt keine Anlage-, Steuer-, Rechts- oder Gesundheitsberatung dar. Lass Ergebnisse bei Bedarf fachlich prüfen.