Git & GitHub – Ultra-Schnellstart Cheat Sheet □□

Dieses Cheat Sheet gibt dir **alle wichtigen Befehle und Konzepte** auf einen Blick, damit du sofort mit Git und GitHub loslegen kannst. Es ist als **Nachschlagewerk** gedacht – wenn du die Hintergründe verstehen willst, schau in die ausführlichen Kapitel deines Kurses.

Die drei Bereiche von Git verstehen

Bevor du mit den Befehlen loslegst, solltest du das grundlegende Konzept verinnerlichen:

```
flowchart LR
    WD["□□□Working Directory\nDein Arbeitsordner"] -->|git add| SA["□□Staging Area\nVorbereitungsbereich"]
    SA -->|git commit| REPO["□□Repository\nVersionshistorie"]
    REPO -->|git push| REMOTE["☁□ Remote\nz.B. GitHub"]
    REMOTE -->|git pull| WD
```

Bereich	Beschreibung
Working Directory	Dein aktueller Projektordner mit allen Dateien, an denen du arbeitest
Staging Area	Der „Vorbereitungsraum" – hier sammelst du Änderungen für den nächsten Commit
Repository	Die lokale Datenbank mit der kompletten Versionshistorie
Remote	Ein entferntes Repository (z.B. auf GitHub), um Code zu teilen oder zu sichern

---

Ersteinrichtung

Diese Befehle führst du **einmalig** nach der Git-Installation aus:

```
# Deinen Namen setzen (erscheint in jedem Commit)
git config --global user.name "Dein Name"

# Deine E-Mail setzen (sollte mit GitHub übereinstimmen)
git config --global user.email "deine@email.de"

# Standardbranch auf "main" setzen (moderne Konvention)
git config --global init.defaultBranch main

# Konfiguration überprüfen
git config --list
```

> **Tipp:** Die `--global`-Option setzt die Einstellung für *alle* Repositories auf deinem Computer. Ohne diese Option gilt sie nur für das aktuelle Projekt.

Repository erstellen und klonen

Aktion	Befehl
Neues Repository im aktuellen Ordner erstellen	`git init`
Bestehendes Repository von GitHub klonen	`git clone https://github.com/user/repo.git`
Repository mit eigenem Ordnernamen klonen	`git clone https://github.com/user/repo.git mein-ordner`

Der tägliche Workflow

Dies sind die Befehle, die du **am häufigsten** verwenden wirst:

Status und Änderungen prüfen

```
# Aktuellen Status anzeigen (welche Dateien geändert, gestaged, etc.)
git status
```

```
# Kurzversion des Status
git status -s

# Änderungen im Working Directory anzeigen (noch nicht gestaged)
git diff

# Änderungen in der Staging Area anzeigen (bereit für Commit)
git diff --staged
```

Dateien stagen und committen

```
# Einzelne Datei zur Staging Area hinzufügen
git add dateiname.php

# Mehrere Dateien hinzufügen
git add datei1.php datei2.php ordner/

# ALLE geänderten und neuen Dateien hinzufügen
git add .

# Commit erstellen mit Nachricht
git commit -m "Kurze, aussagekräftige Beschreibung"

# Staging und Commit in einem Schritt (nur für bereits getrackte Dateien!)
git commit -am "Beschreibung"
```

Letzten Commit korrigieren

```
# Nachricht des letzten Commits ändern
git commit --amend -m "Neue, korrigierte Nachricht"

# Vergessene Dateien zum letzten Commit hinzufügen
git add vergessene-datei.php
git commit --amend --no-edit
```

> ⚠ **Wichtig:** Verwende `--amend` nur für Commits, die du **noch nicht gepusht** hast! Andernfalls überschreibst du die Historie, die andere bereits haben könnten.

Mit GitHub arbeiten

Remote-Verbindung einrichten

```
# Remote-Repository hinzufügen (einmalig nach git init)
git remote add origin https://github.com/dein-user/dein-repo.git

# Alle konfigurierten Remotes anzeigen
git remote -v

# Remote-URL ändern
git remote set-url origin https://github.com/user/neues-repo.git
```

Push und Pull

```
# Lokale Commits zum Remote-Repository hochladen
git push origin main

# Beim ersten Push den Upstream-Branch setzen (danach reicht "git push")
git push -u origin main

# Änderungen vom Remote-Repository holen und mergen
git pull origin main

# Nur Informationen holen, ohne zu mergen
git fetch origin
```

Branches – parallele Entwicklungslinien

Branches sind eines der mächtigsten Features von Git. Nutze sie, um Features zu entwickeln, ohne den Hauptcode zu gefährden.

Branch-Übersicht

```
# Alle lokalen Branches anzeigen (aktueller Branch mit * markiert)
git branch

# Alle Branches inkl. Remote-Branches anzeigen
git branch -a

# Neuen Branch erstellen
git branch feature-name

# Branch erstellen UND direkt wechseln
git checkout -b feature-name
# oder moderner:
git switch -c feature-name
```

Zwischen Branches wechseln

```
# Zu bestehendem Branch wechseln
git checkout feature-name
# oder moderner:
git switch feature-name

# Zurück zum main-Branch
git switch main
```

Branches zusammenführen und löschen

```
# Zuerst in den Zielbranch wechseln (z.B. main)
git switch main

# Anderen Branch in aktuellen Branch mergen
git merge feature-name

# Branch löschen (nach erfolgreichem Merge)
git branch -d feature-name

# Branch löschen (erzwingen, auch wenn nicht gemergt)
git branch -D feature-name

# Remote-Branch löschen
git push origin --delete feature-name
```

Historie und Logs

```
# Commit-Historie anzeigen
git log

# Kompakte einzeilige Darstellung
git log --oneline

# Mit grafischer Branch-Darstellung
git log --oneline --graph --all

# Die letzten n Commits anzeigen
git log -5

# Historie einer bestimmten Datei
git log -- dateiname.php

# Wer hat welche Zeile geschrieben?
git blame dateiname.php
```

Änderungen rückgängig machen

Situation	Befehl
Änderungen im Working Directory verwerfen (Datei auf letzten Commit zurücksetzen)	`git checkout -- datei.php` oder `git restore datei.php`
Datei aus Staging Area entfernen (aber Änderungen behalten)	`git reset HEAD datei.php` oder `git restore --staged datei.php`
Letzten Commit rückgängig machen (Änderungen bleiben im Working Directory)	`git reset --soft HEAD~1`
Letzten Commit komplett verwerfen (Änderungen werden gelöscht!)	`git reset --hard HEAD~1`
Einen früheren Commit „umkehren" (neuer Commit, der Änderungen rückgängig macht)	`git revert <commit-hash>`

> ⚠ **Vorsicht mit `--hard`!** Dieser Befehl löscht Änderungen unwiderruflich. Nutze ihn nur, wenn du dir sicher bist.

Die `.gitignore`-Datei

Erstelle im Projektroot eine Datei namens `.gitignore`, um bestimmte Dateien vom Tracking auszuschließen:

```
# Abhängigkeiten (werden mit Composer/npm installiert)
/vendor/
/node_modules/

# IDE-Einstellungen
/.idea/
/.vscode/

# Umgebungsvariablen und Secrets
.env
.env.local
*.log
```

```
# Betriebssystem-Dateien
.DS_Store
Thumbs.db

# Kompilierte/generierte Dateien
/dist/
/build/
*.cache
```

> 💡 **Tipp:** Füge die `.gitignore` **früh** zu deinem Projekt hinzu – am besten direkt nach `git init`. Bereits getrackte Dateien werden nicht automatisch ignoriert.

Gute Commit-Nachrichten schreiben

Eine gute Commit-Nachricht folgt diesem Muster:

```
<Typ>: <Was wurde geändert> (kurz, im Imperativ)

[Optional: Ausführlichere Beschreibung nach einer Leerzeile]
```

Beispiele für gute Nachrichten:

✅ Gut	❌ Schlecht
`feat: Kontaktformular mit Validierung hinzugefügt`	`update`
`fix: Null-Pointer-Exception bei leerem Warenkorb behoben`	`bug gefixt`
`refactor: Datenbankabfragen in eigene Klasse ausgelagert`	`änderungen`
`docs: README mit Installationsanleitung aktualisiert`	`asdf`

Gängige Präfixe:

- **feat:** Neues Feature
- **fix:** Bugfix
- **refactor:** Code-Umbau ohne Funktionsänderung
- **docs:** Dokumentation
- **style:** Formatierung, keine Code-Änderung

- **test:** Tests hinzugefügt oder geändert
- **chore:** Wartungsarbeiten (Dependencies, Config, etc.)

PhpStorm-Shortcuts □□

Wenn du PhpStorm nutzt, kannst du die meisten Git-Operationen direkt in der IDE ausführen:

Aktion	Shortcut (Windows/Linux)	Shortcut (macOS)
Commit-Dialog öffnen	`Ctrl + K`	`Cmd + K`
Push	`Ctrl + Shift + K`	`Cmd + Shift + K`
Pull/Update	`Ctrl + T`	`Cmd + T`
Git-Log anzeigen	`Alt + 9`	`Cmd + 9`
Branches verwalten	`Ctrl + Shift + ´`	`Ctrl + V` dann Branches
VCS-Popup	`Alt + ´` (Backtick)	`Ctrl + V`

Typischer Workflow auf einen Blick

```
flowchart TD
    A["1. Branch erstellen\ngit switch -c feature-xy"] --> B["2. Code schreiben\nund Dateien ändern"]
    B --> C["3. Änderungen prüfen\ngit status"]
    C --> D["4. Dateien stagen\ngit add ."]
    D --> E["5. Commit erstellen\ngit commit -m '...'"]
    E --> F{"Fertig mit\nFeature?"}
    F -->|Nein| B
    F -->|Ja| G["6. Zurück zu main\ngit switch main"]
    G --> H["7. Updates holen\ngit pull origin main"]
    H --> I["8. Feature mergen\ngit merge feature-xy"]
    I --> J["9. Hochladen\ngit push origin main"]
    J --> K["10. Branch löschen\ngit branch -d feature-xy"]
```

Schnellreferenz – die 15 wichtigsten Befehle

#	Befehl	Beschreibung
1	`git init`	Neues Repository erstellen
2	`git clone <url>`	Repository von Remote klonen
3	`git status`	Aktuellen Zustand anzeigen
4	`git add .`	Alle Änderungen stagen
5	`git commit -m "..."`	Commit erstellen
6	`git push`	Commits hochladen
7	`git pull`	Änderungen herunterladen und mergen
8	`git log --oneline`	Historie anzeigen
9	`git branch`	Branches auflisten
10	`git switch -c <name>`	Neuen Branch erstellen und wechseln
11	`git switch <name>`	Zu Branch wechseln
12	`git merge <branch>`	Branch zusammenführen
13	`git diff`	Änderungen anzeigen
14	`git restore <datei>`	Änderungen verwerfen
15	`git remote -v`	Remote-Verbindungen anzeigen

> □□ **Merke dir:** Mit nur **fünf Befehlen** (`status`, `add`, `commit`, `push`, `pull`) kommst du im Alltag schon sehr weit. Alles andere lernst du nach und nach, wenn du es brauchst!

Revision #1
Created 2026-06-17 01:21:06 UTC by art10m
Updated 2026-06-17 01:21:23 UTC by art10m