Benutzername und E-Mail konfigurieren

Warum Git einen Namen und eine E-Mail-Adresse braucht

Jeder Git-Commit enthält Angaben darüber, **wer die Änderung erstellt** hat. Git verwendet dafür zwei Konfigurationswerte:

- `user.name` – der sichtbare Autorenname
- `user.email` – die E-Mail-Adresse des Autors

Diese Informationen werden dauerhaft in jeden neuen Commit geschrieben. Sie erscheinen später beispielsweise:

- in `git log`,
- in GitHub-Commitansichten,
- bei Pull Requests und Code Reviews,
- in Release- und Audit-Historien.

> Git prüft nicht, ob Name und E-Mail-Adresse „echt“ sind. Dennoch sollten sie bewusst gewählt werden: Sie sind Teil der nachvollziehbaren Projektgeschichte.

Ein Commit speichert dabei mindestens Autor, Zeitpunkt, Nachricht und Inhalt. Vereinfacht:

```
Autor: Max Mustermann <max@example.de>
Datum: 2025-03-08
Nachricht: Add validation for registration form
```

Globale Identität für alle Repositories festlegen

Unter Windows 11 wird die Identität normalerweise **global** eingerichtet. Dann gilt sie für alle Git-Repositories des aktuellen Windows-Benutzerkontos.

Öffne Git Bash, PowerShell oder das Terminal in PhpStorm und führe diese Befehle aus:

```
git config --global user.name "Max Mustermann"
git config --global user.email "max.mustermann@example.de"
```

Ersetze die Beispielwerte durch deine gewünschte Commit-Identität.

Ein typisches Beispiel mit beruflicher Adresse:

```
git config --global user.name "Maria Schneider"
git config --global user.email "maria.schneider@beispielunternehmen.de"
```

Für persönliche Open-Source-Projekte könnte die Konfiguration so aussehen:

```
git config --global user.name "Maria Schneider"
git config --global user.email "maria.schneider@example.com"
```

Die Anführungszeichen sind besonders beim Namen sinnvoll, weil er Leerzeichen enthält.

Konfiguration überprüfen

Lies die gerade gesetzten Werte mit folgenden Befehlen aus:

```
git config --global user.name
git config --global user.email
```

Die Ausgabe sollte beispielsweise so aussehen:

```
Maria Schneider
maria.schneider@example.com
```

Alternativ kannst du beide Werte gemeinsam anzeigen:

```
git config --global --get-regexp "^user\."
```

Beispielausgabe:

```
user.name Maria Schneider
user.email maria.schneider@example.com
```

Mit diesem Befehl lässt sich außerdem die gesamte globale Git-Konfiguration prüfen:

```
git config --global --list
```

Eine mögliche Ausgabe:

```
user.name=Maria Schneider
user.email=maria.schneider@example.com
init.defaultbranch=main
```

Wo Git die globale Konfiguration speichert

Git speichert die globale Konfiguration unter Windows in einer Datei im Benutzerprofil. Der Pfad ist üblicherweise:

```
C:\Users\DeinWindowsBenutzername\.gitconfig
```

Git Bash zeigt denselben Speicherort in Unix-Schreibweise an:

```
~/.gitconfig
```

Den tatsächlich verwendeten Ursprung eines Konfigurationswerts findest du mit:

```
git config --show-origin --get user.name
git config --show-origin --get user.email
```

Eine mögliche Ausgabe in Git Bash:

```
file:C:/Users/maria/.gitconfig    Maria Schneider
```

Das ist besonders nützlich, wenn mehrere Konfigurationsebenen vorhanden sind.

Die Konfigurationsebenen verstehen

Git kann Werte auf mehreren Ebenen speichern. Für Name und E-Mail sind vor allem diese drei Ebenen relevant:

1. **Systemweit**
   Gilt für alle Windows-Benutzer und alle Repositories auf dem Computer.
2. **Global**
   Gilt für alle Repositories eines Windows-Benutzerkontos.
3. **Lokal**
   Gilt nur für ein bestimmtes Repository.

Die Priorität lautet:

$$
\text{lokal} > \text{global} > \text{systemweit}
$$

Ein lokal gesetzter Wert überschreibt also die globale Einstellung.

In der Praxis ist dieses Vorgehen sinnvoll:

- Eine persönliche Standardidentität wird **global** eingerichtet.
- Für ein Kundenprojekt oder ein Arbeitsrepository wird bei Bedarf eine **lokale** Geschäftsidentität gesetzt.

Unterschiedliche Identitäten für private und berufliche Projekte

Viele Entwicklerinnen und Entwickler verwenden verschiedene E-Mail-Adressen:

- private Adresse für Lernprojekte oder Open Source,
- geschäftliche Adresse für Unternehmensprojekte,
- datenschutzfreundliche GitHub-`noreply`-Adresse für öffentliche Commits.

Angenommen, deine globale Konfiguration ist privat:

```
git config --global user.name "Maria Schneider"
git config --global user.email "maria.schneider@example.com"
```

Für ein berufliches Repository wechselst du zuerst in dessen Verzeichnis:

```
cd C:\Projekte\kundenportal
```

Dann setzt du die lokale Identität **ohne** `--global`:

```
git config user.name "Maria Schneider"
git config user.email "maria.schneider@beispielunternehmen.de"
```

Diese Einstellungen werden in der Datei `.git/config` innerhalb dieses einen Repositories gespeichert.

Prüfe die lokale Konfiguration:

```
git config --local --list
```

Oder frage gezielt ab, welche Werte Git für das aktuelle Repository tatsächlich verwendet:

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

Wichtig: Diese beiden Befehle zeigen den wirksamen Wert an. Git berücksichtigt dabei die Priorität aller Ebenen.

Die Herkunft eines Werts eindeutig prüfen

Wenn unklar ist, ob eine globale oder lokale Einstellung verwendet wird, hilft:

```
git config --show-origin --get-regexp "^user\."
```

Beispielausgabe:

```
file:C:/Users/maria/.gitconfig              user.name Maria Schneider
file:C:/Projekte/kundenportal/.git/config   user.email maria.schneider@beispielunternehmen.de
```

Daran erkennst du:

- Der Name stammt aus der globalen Konfiguration.
- Die E-Mail-Adresse wurde lokal für das Kundenprojekt überschrieben.

Für eine vollständige Übersicht inklusive aller Ebenen verwende:

```
git config --list --show-origin
```

GitHub-E-Mail-Adresse richtig zuordnen

Damit GitHub Commits deinem Konto zuordnen kann, muss die im Commit verwendete E-Mail-Adresse zu deinem GitHub-Konto passen.

In GitHub findest du deine hinterlegten E-Mail-Adressen unter:

Settings → Emails

Dort kannst du:

- eine E-Mail-Adresse hinzufügen und bestätigen,
- eine primäre Commit-Adresse auswählen,
- die Sichtbarkeit deiner E-Mail-Adresse steuern,
- eine GitHub-`noreply`-Adresse verwenden.

Wenn du eine bestätigte Adresse verwendest, kann GitHub den Commit normalerweise deinem Profil zuordnen. Ohne Zuordnung erscheint der Commit zwar im Repository, wird aber möglicherweise nicht in deinem GitHub-Profil berücksichtigt.

> **Wichtig:** Die Commit-E-Mail-Adresse und die Adresse für die Anmeldung bei GitHub müssen nicht identisch sein. Entscheidend ist, dass die Commit-Adresse im GitHub-Konto hinterlegt und verifiziert ist.

Private E-Mail-Adresse auf GitHub schützen

Bei öffentlichen Repositories ist die E-Mail-Adresse aus Commits grundsätzlich sichtbar. Wer seine private Adresse nicht in einer öffentlichen Git-Historie veröffentlichen möchte, kann die von GitHub bereitgestellte `noreply`-Adresse nutzen.

Sie hat typischerweise dieses Format:

```
12345678+benutzername@users.noreply.github.com
```

Oder, abhängig von den GitHub-Einstellungen:

```
benutzername@users.noreply.github.com
```

Die genaue Adresse zeigt GitHub im Bereich **Settings → Emails** an. Übernimm sie exakt in deine Git-Konfiguration:

```
git config --global user.email "12345678+benutzername@users.noreply.github.com"
```

Diese Variante bietet zwei Vorteile:

- GitHub kann Commits weiterhin deinem Konto zuordnen.
- Deine persönliche E-Mail-Adresse erscheint nicht in öffentlichen Commit-Daten.

Für berufliche oder interne Repositories können allerdings Unternehmensrichtlinien verlangen, dass die geschäftliche Adresse verwendet wird. Prüfe deshalb immer die Vorgaben des Teams.

Name und E-Mail nachträglich ändern

Eine globale Einstellung lässt sich jederzeit überschreiben:

```
git config --global user.name "Neuer Name"
git config --global user.email "neue.adresse@example.com"
```

Auch eine lokale Einstellung kann geändert werden:

```
git config user.name "Neuer Name"
git config user.email "neue.adresse@beispielunternehmen.de"
```

Die Änderung gilt jedoch nur für **zukünftige Commits**. Bereits vorhandene Commits behalten ihre gespeicherten Autorendaten.

Den letzten noch nicht veröffentlichten Commit korrigieren

Wenn nur der letzte Commit die falsche Identität enthält und noch nicht veröffentlicht wurde, kannst du ihn mit der aktuellen Identität neu erstellen:

```
git commit --amend --reset-author --no-edit
```

Dabei gilt:

- `--amend` ersetzt den letzten Commit.
- `--reset-author` übernimmt den aktuell konfigurierten Namen und die aktuelle E-Mail-Adresse.
- `--no-edit` behält die bisherige Commit-Nachricht.

> ⚠ Ändere veröffentlichte Commits nicht leichtfertig. Eine nachträgliche Änderung erzeugt neue Commit-Hashes und kann die Zusammenarbeit im Team stören.

Die systematische Korrektur von Autorendaten in mehreren oder bereits veröffentlichten Commits wird später beim Thema Historienbereinigung behandelt.

Typische Fehlermeldung: „Author identity unknown“

Ohne konfigurierte Identität verweigert Git das Erstellen eines Commits häufig mit einer Meldung wie:

```
Author identity unknown

Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
```

Die Lösung ist genau die globale Konfiguration:

```
git config --global user.name "Dein Name"
git config --global user.email "deine.adresse@example.com"
```

Prüfe danach die Werte:

```
git config --global --get-regexp "^user\."
```

Versuche anschließend erneut zu committen.

Commit-Identität ist keine Anmeldung

Ein häufiger Anfängerfehler besteht darin, `user.name` und `user.email` mit GitHub-Zugangsdaten zu verwechseln.

Die Git-Konfiguration beschreibt nur die **Autorenangabe eines Commits**:

```
git config --global user.name "Maria Schneider"
git config --global user.email "maria.schneider@example.com"
```

Sie authentifiziert dich **nicht** bei GitHub.

Für das Klonen, Pushen und Abrufen privater Repositories brauchst du zusätzlich eine Authentifizierung, beispielsweise über:

- HTTPS mit Personal Access Token und Git Credential Manager,
- SSH mit einem SSH-Schlüssel,
- die GitHub-Anmeldung in PhpStorm.

Diese Themen folgen im Kapitel zu GitHub-Konto, Remotes und Authentifizierung.

Konfiguration in PhpStorm kontrollieren

PhpStorm verwendet für Git-Commits in der Regel die Git-Konfiguration des aktiven Repositories. Deshalb ist die Konfiguration über das Terminal zuverlässig und transparent.

Du kannst die Werte auch im integrierten Terminal von PhpStorm prüfen:

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

Achte besonders darauf, wenn du mehrere Projekte gleichzeitig geöffnet hast. Jedes Repository kann eigene lokale Werte in `.git/config` besitzen.

Vor einem wichtigen Commit kannst du im Terminal kontrollieren, welche Identität Git wirklich verwendet:

```
git var GIT_AUTHOR_IDENT
```

Eine mögliche Ausgabe:

```
Maria Schneider <maria.schneider@beispielunternehmen.de> 1741435200 +0100
```

Das ist die Identität, die Git für einen neuen Commit als Autor verwenden würde.

Empfohlene Einrichtung für den Kurs

Für die Übungen dieses Kurses genügt eine globale Konfiguration. Verwende entweder eine bestätigte GitHub-Adresse oder deine GitHub-`noreply`-Adresse:

```
git config --global user.name "Dein Name"
git config --global user.email "deine-github-adresse@users.noreply.github.com"
```

Prüfe danach:

```
git config --global --get-regexp "^user\."
```

Wenn beide Werte korrekt ausgegeben werden, ist deine Commit-Identität eingerichtet. Jeder kommende Commit kann nun eindeutig einer Autorin oder einem Autor zugeordnet werden.

Revision #1
Created 2026-07-25 11:13:45 UTC by art10m
Updated 2026-07-25 11:13:45 UTC by art10m