Kapitel 6: GitHub einrichten und verbinden

Einleitung: Bisher hast du nur lokal gearbeitet – dein Code existiert ausschließlich auf deinem Computer. Das ist riskant: Ein Festplattendefekt, und alles ist weg. GitHub löst dieses Problem, indem es dein Repository in der Cloud speichert. Aber GitHub ist mehr als nur ein Backup: Es ist eine Plattform, die Zusammenarbeit ermöglicht, dein Portfolio als Entwickler präsentiert und viele nützliche Werkzeuge bietet. Um sicher mit GitHub zu kommunizieren, wirst du SSH-Keys einrichten – eine Art digitaler Schlüssel, der deinen Computer bei GitHub identifiziert. Dieses Kapitel führt dich durch die Kontoerstellung, die Sicherheitseinrichtung und die Verknüpfung deines ersten Projekts.

- Einen GitHub-Account erstellen – Schritt für Schritt □□
- SSH-Keys für GitHub – Sichere Authentifizierung einrichten □□
- Lokales Git-Repository mit GitHub verbinden □□

Einen GitHub-Account erstellen – Schritt für Schritt

Du möchtest deinen Code online sichern, von überall darauf zugreifen oder später vielleicht sogar mit anderen zusammenarbeiten? Dann ist ein **GitHub-Account** der logische nächste Schritt nach der lokalen Git-Einrichtung. GitHub ist die weltweit größte Plattform für Code-Hosting und Versionsverwaltung – und die Registrierung ist kostenlos. Hier erfährst du, wie du deinen Account erstellst und welche Einstellungen du von Anfang an im Blick haben solltest.

Die Registrierung durchführen

1. **GitHub-Website aufrufen**

   Öffne deinen Browser und gehe zu github.com. Auf der Startseite siehst du prominent einen Button wie **„Sign up"** oder **„Registrieren"**.
2. **E-Mail-Adresse eingeben**
   GitHub fragt dich zuerst nach deiner E-Mail-Adresse. Verwende eine Adresse, auf die du **langfristig Zugriff** hast – idealerweise dieselbe, die du auch in deiner lokalen Git-Konfiguration eingetragen hast. Das sorgt dafür, dass deine Commits später korrekt mit deinem GitHub-Profil verknüpft werden.
3. **Passwort wählen**
   Wähle ein **starkes, einzigartiges Passwort**. GitHub enthält oft sensiblen Code und ist ein attraktives Ziel für Angreifer. Ein Passwort-Manager kann hier sehr hilfreich sein.
4. **Benutzernamen festlegen**
   Der Benutzername ist ein wichtiger Punkt, über den du dir **vorab Gedanken machen** solltest (siehe nächster Abschnitt).
5. **E-Mail-Verifizierung**
   GitHub schickt dir eine Bestätigungs-E-Mail. Klicke auf den Link darin, um deine Adresse zu verifizieren – erst dann ist dein Account vollständig aktiviert.
6. **Optionale Personalisierung**
   Nach der Registrierung führt dich GitHub durch ein paar optionale Fragen (z. B. „Wofür nutzt du GitHub?" oder „Wie groß ist dein Team?"). Diese dienen hauptsächlich dazu, dir passende Empfehlungen zu geben – du kannst sie überspringen, wenn du möchtest.

Den richtigen Benutzernamen wählen

Der Benutzername auf GitHub ist **öffentlich sichtbar** und Teil der URL zu deinen Repositories (z. B. `github.com/deinname/projekt`). Deshalb lohnt es sich, hier einen Moment nachzudenken:

- **Professionalität vs. Kreativität:** Wenn du GitHub auch für berufliche Zwecke nutzen möchtest (Bewerbungen, Portfolio, Open-Source-Beiträge), empfiehlt sich ein **seriöser, wiedererkennbarer Name**. Kombinationen aus Vor- und Nachname funktionieren gut, z. B. `maxmustermann` oder `m-mustermann`. Namen wie `xXx_CoderKing_xXx` wirken weniger professionell.
- **Einzigartigkeit:** Der Name muss auf GitHub einzigartig sein. Beliebte Namen sind oft schon vergeben – sei also kreativ, aber bleib lesbar.
- **Änderbarkeit:** Du kannst deinen Benutzernamen später ändern, aber das hat Konsequenzen: Alle URLs zu deinen Repositories ändern sich, und Links von außen führen ins Leere. Wähle also möglichst von Anfang an einen Namen, mit dem du langfristig zufrieden bist.

> □□ **Tipp:** Wenn du planst, GitHub als Portfolio zu nutzen, überlege, ob der Name zu deinem „Personal Branding" passt – also zu dem Eindruck, den du hinterlassen möchtest.

Wichtige Profileinstellungen nach der Registrierung

Sobald dein Account steht, solltest du einen Blick in die **Einstellungen** werfen. Du erreichst sie über dein Profilbild oben rechts → **Settings**. Hier die wichtigsten Bereiche:

Profil-Informationen

Unter **Public profile** kannst du grundlegende Informationen über dich hinterlegen:

- **Name:** Dein vollständiger oder angezeigter Name (unabhängig vom Benutzernamen).

- **Bio:** Eine kurze Beschreibung von dir – was du machst, welche Technologien dich interessieren.
- **Location:** Optional dein Standort.
- **Website/Social Links:** Hier kannst du auf dein Portfolio, LinkedIn oder andere Profile verlinken.

Diese Informationen sind **öffentlich** und helfen anderen, dich einzuordnen – besonders wenn du in Open-Source-Projekten aktiv wirst oder dein Profil als Portfolio nutzt.

E-Mail-Einstellungen

Unter **Emails** findest du wichtige Optionen:

- **Primary email:** Die Haupt-E-Mail-Adresse für Benachrichtigungen und Account-Wiederherstellung.
- **Keep my email addresses private:** Diese Option solltest du **aktivieren**, wenn du nicht möchtest, dass deine echte E-Mail-Adresse in Commits öffentlich sichtbar ist. GitHub stellt dir dann eine anonymisierte Adresse zur Verfügung (z. B. `123456+deinname@users.noreply.github.com`), die du in deiner lokalen Git-Konfiguration verwenden kannst.
- **Block command line pushes that expose my email:** Verhindert, dass du versehentlich Commits mit deiner echten E-Mail hochlädst.

> ⚠ **Wichtig:** Wenn du deine lokale Git-Konfiguration mit einer privaten E-Mail-Adresse eingerichtet hast und diese nicht öffentlich sein soll, aktiviere unbedingt die Datenschutzoptionen und verwende die von GitHub bereitgestellte No-Reply-Adresse.

Zwei-Faktor-Authentifizierung einrichten

Unter **Password and authentication** findest du die Option zur **Two-factor authentication (2FA)**. Diese solltest du **dringend aktivieren**:

- 2FA schützt deinen Account, selbst wenn dein Passwort kompromittiert wird.
- GitHub unterstützt Authenticator-Apps (z. B. Google Authenticator, Authy, 1Password) und Hardware-Keys.
- Ohne 2FA kannst du bei manchen Aktionen eingeschränkt sein (z. B. bei der Nutzung von GitHub Actions in bestimmten Organisationen).

Die Einrichtung dauert nur wenige Minuten und erhöht die Sicherheit deines Accounts erheblich.

SSH-Keys und Personal Access Tokens

Für die Verbindung zwischen deinem lokalen Rechner und GitHub gibt es zwei gängige Methoden:

- **SSH-Keys:** Eine sichere Methode, bei der du einen Schlüssel auf deinem Computer generierst und den öffentlichen Teil bei GitHub hinterlegst. Das ermöglicht passwortloses Pushen und Pullen.
- **Personal Access Tokens (PAT):** Wenn du HTTPS statt SSH nutzt, brauchst du ein Token anstelle deines Passworts. Diese Tokens kannst du unter **Developer settings → Personal access tokens** erstellen.

Für den Anfang kannst du diesen Schritt überspringen – PhpStorm fragt dich beim ersten Push nach deinen Zugangsdaten und hilft dir bei der Einrichtung. Aber es ist gut zu wissen, wo du diese Optionen findest.

Repository-Sichtbarkeit verstehen

Wenn du später ein Repository auf GitHub erstellst, wirst du nach der **Sichtbarkeit** gefragt. Es gibt zwei Optionen:

Sichtbarkeit	Bedeutung
Public	Jeder kann das Repository sehen, auch ohne GitHub-Account. Der Code ist vollständig öffentlich. Andere können ihn ansehen, forken und bei Erlaubnis auch beitragen.
Private	Nur du und Personen, denen du explizit Zugriff gibst, können das Repository sehen. Ideal für persönliche Projekte, Kundenprojekte oder Code, den du nicht teilen möchtest.

Für deine **persönlichen Lern- und Übungsprojekte** ist „Private" oft eine gute Wahl – du kannst sie jederzeit später öffentlich machen, wenn du sie zeigen möchtest. Für Open-Source-Projekte oder Portfolio-Stücke wählst du „Public".

> □□ **Gut zu wissen:** Mit einem kostenlosen GitHub-Account kannst du **unbegrenzt viele private Repositories** erstellen – früher war das ein kostenpflichtiges Feature.

Die GitHub-Oberfläche im Überblick

Nach der Anmeldung landest du auf deinem **Dashboard**. Hier ein kurzer Überblick über die wichtigsten Bereiche:

```
flowchart TB
    A["GitHub Dashboard"] --> B["Repositories\nDeine Projekte"]
    A --> C["Explore\nOpen-Source entdecken"]
    A --> D["Notifications\nBenachrichtigungen"]
    A --> E["Settings\nAccount-Einstellungen"]
    B --> F["Neues Repository erstellen"]
    B --> G["Bestehende Repos verwalten"]
```

- **Repositories:** Hier siehst du alle deine Projekte und kannst neue anlegen.
- **Your profile:** Deine öffentliche Profilseite mit deinen Repositories, Beiträgen und Aktivitäten.
- **Explore:** Entdecke interessante Open-Source-Projekte.
- **Settings:** Alle Account- und Sicherheitseinstellungen.

Zusammenfassung: Deine Checkliste nach der Registrierung

Nachdem du deinen GitHub-Account erstellt hast, solltest du folgende Punkte abhaken:

1. **E-Mail verifiziert** – damit alle Funktionen freigeschaltet sind
2. **Benutzername überprüft** – ist er professionell und langfristig passend?
3. **Profil ausgefüllt** – zumindest Name und eine kurze Bio
4. **E-Mail-Privatsphäre aktiviert** – wenn du deine echte Adresse schützen möchtest
5. **Zwei-Faktor-Authentifizierung eingerichtet** – für die Sicherheit deines Accounts
6. **Lokale Git-Konfiguration angepasst** – falls du die No-Reply-Adresse von GitHub nutzen möchtest

Mit diesen Grundlagen bist du bestens gerüstet, um im nächsten Schritt dein erstes Repository auf GitHub zu erstellen und es mit deinem lokalen Projekt zu verbinden! □□

SSH-Keys für GitHub – Sichere Authentifizierung einrichten □□

Wenn du regelmäßig mit GitHub arbeitest, wirst du schnell merken, dass die Eingabe von Benutzername und Passwort bei jedem `push` oder `pull` lästig wird. Außerdem hat GitHub die Passwort-Authentifizierung über HTTPS für Git-Operationen inzwischen eingeschränkt. Die elegante und sichere Lösung heißt **SSH-Key-Authentifizierung**. In diesem Kapitel erfährst du, was SSH-Keys sind, warum sie für GitHub so wichtig sind und wie du sie Schritt für Schritt einrichtest.

Was sind SSH-Keys überhaupt?

SSH steht für **Secure Shell** und ist ein Protokoll für verschlüsselte Verbindungen zwischen Computern. Ein SSH-Key ist ein **kryptografisches Schlüsselpaar**, das aus zwei Teilen besteht:

- **Privater Schlüssel** (*Private Key*): Dieser bleibt **ausschließlich auf deinem Computer** und wird niemals weitergegeben. Er ist wie der Schlüssel zu deiner Haustür – nur du besitzt ihn.
- **Öffentlicher Schlüssel** (*Public Key*): Diesen gibst du an Dienste wie GitHub weiter. Er ist wie ein spezielles Schloss, das nur mit deinem privaten Schlüssel geöffnet werden kann.

Das Geniale an diesem System: Wenn du dich mit GitHub verbindest, beweist dein Computer durch den privaten Schlüssel, dass er zu dem öffentlichen Schlüssel gehört, der bei GitHub hinterlegt ist – **ohne dass ein Passwort übertragen wird**. Selbst wenn jemand die Verbindung abhören würde, könnte er sich nicht als du ausgeben.

```
flowchart LR
    A["□□Privater Schluessel\nBleibt auf deinem PC"] --- B["□□
Kryptografisches\nSchluesselpaar"]
    B --- C["□□Oeffentlicher Schluessel\nWird bei GitHub hinterlegt"]

    D["□□Dein Computer"] -->|Authentifizierung\nohne Passwort| E["☁□ GitHub Server"]
```

Warum brauche ich SSH-Keys für GitHub?

Es gibt mehrere gute Gründe, warum du SSH-Keys einrichten solltest:

1. **Sicherheit**
   SSH-Keys sind deutlich sicherer als Passwörter. Ein typischer SSH-Key hat eine Länge von 256 bis 4096 Bit – das ist praktisch unknackbar. Selbst wenn jemand deinen Netzwerkverkehr mitliest, kann er deinen privaten Schlüssel nicht rekonstruieren.
2. **Komfort im Alltag**
   Nach der einmaligen Einrichtung musst du bei `git push` oder `git pull` **keine Zugangsdaten mehr eingeben**. Der SSH-Agent auf deinem Computer übernimmt die Authentifizierung automatisch im Hintergrund.
3. **GitHub hat HTTPS-Passwörter eingeschränkt**
   Seit August 2021 akzeptiert GitHub keine einfachen Passwörter mehr für Git-Operationen über HTTPS. Stattdessen müsstest du einen *Personal Access Token* verwenden – der ist allerdings lang, unpraktisch und muss regelmäßig erneuert werden. SSH-Keys sind die elegantere Alternative.
4. **Mehrere Geräte, eine Identität**
   Du kannst auf jedem deiner Computer einen eigenen SSH-Key erstellen und bei GitHub hinterlegen. So behältst du die Kontrolle darüber, welche Geräte Zugriff auf deine Repositories haben, und kannst einzelne Keys jederzeit widerrufen.

SSH-Key erstellen – Schritt für Schritt

Die Einrichtung unterscheidet sich je nach Betriebssystem nur minimal. Ich zeige dir den Prozess für **Windows**, **macOS** und **Linux** – die Befehle sind weitgehend identisch.

Voraussetzungen prüfen

Bevor du loslegst, stelle sicher, dass du ein Terminal öffnen kannst:

- **Windows**: Nutze **Git Bash** (wird mit Git installiert), **PowerShell** oder das **Windows Terminal**.
- **macOS**: Öffne die **Terminal**-App (findest du unter Programme → Dienstprogramme).

- **Linux**: Öffne dein bevorzugtes Terminal.

Schritt 1: Prüfen, ob bereits ein SSH-Key existiert

Es ist möglich, dass du schon einen SSH-Key hast, ohne es zu wissen. Prüfe das mit folgendem Befehl:

```
ls -la ~/.ssh
```

Wenn du Dateien wie `id_rsa`, `id_ed25519` oder ähnliche siehst (jeweils mit und ohne `.pub`-Endung), hast du bereits ein Schlüsselpaar. Du kannst dieses verwenden oder ein neues erstellen.

> **Tipp:** Wenn der Ordner `~/.ssh` nicht existiert, ist das kein Problem – er wird beim Erstellen eines Keys automatisch angelegt.

Schritt 2: Neuen SSH-Key generieren

GitHub empfiehlt den modernen **Ed25519-Algorithmus**, der sicherer und schneller ist als das ältere RSA. Führe folgenden Befehl aus und ersetze die E-Mail-Adresse durch **deine GitHub-E-Mail**:

```
ssh-keygen -t ed25519 -C "deine-email@beispiel.de"
```

Was passiert jetzt?

1. **Speicherort wählen**
   Du wirst gefragt, wo der Key gespeichert werden soll:
   ```
   Enter file in which to save the key (/home/deinname/.ssh/id_ed25519):
   ```
   Drücke einfach **Enter**, um den Standardpfad zu akzeptieren. Das ist in den allermeisten Fällen die richtige Wahl.
2. **Passphrase festlegen**
   Anschließend wirst du nach einer **Passphrase** gefragt:
   ```
   Enter passphrase (empty for no passphrase):
   ```
   Hier hast du zwei Optionen:

- **Mit Passphrase** *(empfohlen)*: Gib ein sicheres Passwort ein. Falls jemand Zugriff auf deinen Computer bekommt, ist der Key trotzdem geschützt.
- **Ohne Passphrase**: Drücke einfach Enter. Der Key funktioniert dann ohne zusätzliche Eingabe, ist aber weniger sicher.

> **Empfehlung:** Nutze eine Passphrase! Mit dem SSH-Agent (siehe weiter unten) musst du sie trotzdem nur einmal pro Sitzung eingeben.

3. **Bestätigung**
   Nach erfolgreicher Erstellung siehst du eine Ausgabe wie diese:

```
Your identification has been saved in /home/deinname/.ssh/id_ed25519
Your public key has been saved in /home/deinname/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:aBcDeFgHiJkLmNoPqRsTuVwXyZ1234567890 deine-email@beispiel.de
```

Du hast jetzt zwei neue Dateien:

- `~/.ssh/id_ed25519` – dein **privater Schlüssel** (niemals teilen!)
- `~/.ssh/id_ed25519.pub` – dein **öffentlicher Schlüssel** (kommt zu GitHub)

Schritt 3: SSH-Agent starten und Key hinzufügen

Der **SSH-Agent** ist ein Hintergrundprogramm, das deine SSH-Keys verwaltet und bei Bedarf automatisch verwendet. So musst du die Passphrase nicht bei jeder Git-Operation eingeben.

Auf macOS und Linux:

```
eval "$(ssh-agent -s)"
```

Du siehst eine Ausgabe wie `Agent pid 12345` – der Agent läuft jetzt.

Auf Windows (Git Bash):

```
eval "$(ssh-agent -s)"
```

Der Befehl ist identisch. In PowerShell kann es etwas anders sein – Git Bash ist hier die einfachere Wahl.

Key zum Agent hinzufügen:

```
ssh-add ~/.ssh/id_ed25519
```

Falls du eine Passphrase gesetzt hast, wirst du jetzt danach gefragt. Nach der Eingabe ist der Key geladen und du musst die Passphrase erst wieder eingeben, wenn du deinen Computer neu startest oder dich ab- und anmeldest.

Öffentlichen Schlüssel bei GitHub hinterlegen

Jetzt kommt der entscheidende Schritt: Du musst GitHub deinen öffentlichen Schlüssel mitteilen, damit GitHub weiß, dass Verbindungen mit deinem privaten Schlüssel vertrauenswürdig sind.

Schritt 1: Öffentlichen Schlüssel kopieren

Zeige den Inhalt deines öffentlichen Schlüssels an:

```
cat ~/.ssh/id_ed25519.pub
```

Du siehst eine lange Zeile, die ungefähr so aussieht:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIG... (viele Zeichen) ...deine-email@beispiel.de
```

Kopiere diese gesamte Zeile in die Zwischenablage – inklusive `ssh-ed25519` am Anfang und der E-Mail am Ende.

> □□ **Tipp für macOS:** Du kannst den Key direkt in die Zwischenablage kopieren mit:
>
> ```
> pbcopy < ~/.ssh/id_ed25519.pub
> ```
>
> **Auf Windows (Git Bash):**
>
> ```
> clip < ~/.ssh/id_ed25519.pub
> ```

Schritt 2: Key in GitHub einfügen

1. Öffne github.com und melde dich an.
2. Klicke auf dein **Profilbild** oben rechts und wähle **Settings**.
3. In der linken Seitenleiste findest du den Bereich **Access** – klicke dort auf **SSH and GPG keys**.
4. Klicke auf den grünen Button **New SSH key**.
5. Fülle die Felder aus:
   - **Title**: Gib einen beschreibenden Namen ein, z. B. „MacBook Pro Arbeit" oder „Windows Desktop". So weißt du später, welcher Key zu welchem Gerät gehört.
   - **Key type**: Lass die Auswahl auf **Authentication Key** (Standard).
   - **Key**: Füge hier den kopierten öffentlichen Schlüssel ein.
6. Klicke auf **Add SSH key**.
7. GitHub fragt dich möglicherweise nach deinem Passwort zur Bestätigung – gib es ein.

Dein SSH-Key ist jetzt bei GitHub registriert! □□

Verbindung testen

Bevor du loslegst, solltest du prüfen, ob alles funktioniert:

```
ssh -T git@github.com
```

Beim **ersten Mal** wirst du gefragt, ob du dem Server vertraust:

```
The authenticity of host 'github.com (IP-Adresse)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Gib **yes** ein und drücke Enter. Wenn alles klappt, siehst du:

```
Hi dein-username! You've successfully authenticated, but GitHub does not provide shell access.
```

Diese Meldung bedeutet: **Es funktioniert!** Du bist authentifiziert, aber GitHub erlaubt keinen direkten Shell-Zugriff – das ist normal und beabsichtigt.

Repositories mit SSH verwenden

Jetzt, wo SSH eingerichtet ist, musst du darauf achten, die **SSH-URL** statt der HTTPS-URL zu verwenden, wenn du Repositories klonst oder verbindest.

Beim Klonen eines Repositories

Wenn du auf GitHub ein Repository öffnest und auf den grünen **Code**-Button klickst, siehst du mehrere Optionen. Wähle **SSH** und kopiere die URL, die so aussieht:

```
git@github.com:dein-username/dein-repository.git
```

Dann klonst du mit:

```
git clone git@github.com:dein-username/dein-repository.git
```

Ein bestehendes Repository auf SSH umstellen

Falls du ein Repository bereits mit HTTPS geklont hast, kannst du die Remote-URL ändern:

```
git remote set-url origin git@github.com:dein-username/dein-repository.git
```

Prüfe die Änderung mit:

```
git remote -v
```

Du solltest jetzt SSH-URLs sehen:

```
origin  git@github.com:dein-username/dein-repository.git (fetch)
origin  git@github.com:dein-username/dein-repository.git (push)
```

SSH in PhpStorm konfigurieren

PhpStorm verwendet normalerweise automatisch die SSH-Konfiguration deines Systems. Trotzdem lohnt es sich, die Einstellungen zu prüfen:

1. Öffne **File → Settings** (auf macOS: **PhpStorm → Settings**).
2. Navigiere zu **Version Control → Git**.
3. Stelle sicher, dass **SSH executable** auf **Native** oder **Built-in** steht. Die Option **Native** nutzt die SSH-Installation deines Systems inklusive des SSH-Agents – das ist meist die beste Wahl.

4. Unter **Version Control → GitHub** kannst du deinen GitHub-Account hinzufügen. Wähle hier **Log In via GitHub** oder, falls du Token bevorzugst, die entsprechende Option. Für Git-Operationen über SSH ist dieser Schritt optional, aber für einige PhpStorm-Features wie das Erstellen von Pull Requests nützlich.

Wenn du jetzt in PhpStorm ein Repository mit SSH-URL klonst oder pushst, sollte alles reibungslos funktionieren – ohne Passworteingabe.

Häufige Probleme und Lösungen

„Permission denied (publickey)"

Diese Fehlermeldung bedeutet, dass GitHub deinen SSH-Key nicht erkennt. Mögliche Ursachen:

- **Key nicht zum SSH-Agent hinzugefügt**: Führe `ssh-add ~/.ssh/id_ed25519` erneut aus.
- **Falscher Key bei GitHub hinterlegt**: Überprüfe auf GitHub unter *Settings → SSH and GPG keys*, ob der richtige öffentliche Schlüssel eingetragen ist.
- **Falscher Dateiname**: Wenn du den Key unter einem anderen Namen gespeichert hast, musst du diesen explizit angeben.

SSH-Agent startet nicht automatisch

Auf Windows kann es sein, dass der SSH-Agent-Dienst nicht läuft. Öffne PowerShell als Administrator und führe aus:

```
Get-Service ssh-agent | Set-Service -StartupType Automatic
Start-Service ssh-agent
```

Auf macOS kannst du den Key dauerhaft im Agent speichern, indem du diese Zeilen zu `~/.ssh/config` hinzufügst (erstelle die Datei, falls sie nicht existiert):

```
Host github.com
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_ed25519
```

Mehrere GitHub-Accounts

Falls du mehrere GitHub-Accounts hast (z. B. privat und beruflich), kannst du für jeden einen eigenen Key erstellen und in `~/.ssh/config` konfigurieren:

```
# Privater Account
Host github.com-privat
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519_privat

# Beruflicher Account
Host github.com-arbeit
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519_arbeit
```

Beim Klonen verwendest du dann den jeweiligen Host:

```
git clone git@github.com-privat:username/repo.git
git clone git@github.com-arbeit:firmenname/repo.git
```

Zusammenfassung

SSH-Keys sind die **sichere und komfortable Methode**, um dich bei GitHub zu authentifizieren. Nach der einmaligen Einrichtung sparst du dir die ständige Eingabe von Zugangsdaten und profitierst von einer deutlich höheren Sicherheit. Hier nochmal die wichtigsten Schritte im Überblick:

```
flowchart TD
    A["1. SSH-Key generieren\nssh-keygen -t ed25519 -C ..."] --> B["2. SSH-Agent starten\nund Key hinzufuegen"]
    B --> C["3. Oeffentlichen Key kopieren\ncat ~/.ssh/id_ed25519.pub"]
    C --> D["4. Key bei GitHub einfuegen\nSettings - SSH Keys"]
    D --> E["5. Verbindung testen\nssh -T git@github.com"]
    E --> F["6. SSH-URLs verwenden\ngit@github.com:user/repo.git"]
```

Mit dieser Einrichtung bist du bestens vorbereitet, um deine Repositories sicher und effizient mit GitHub zu synchronisieren – sei es über die Kommandozeile oder direkt aus PhpStorm heraus. □□

Lokales Git-Repository mit GitHub verbinden □□

Du hast also schon fleißig lokal commitet und möchtest jetzt deinen Code sicher in der Cloud haben – sei es als Backup, um von mehreren Geräten darauf zuzugreifen oder um später mit anderen zusammenzuarbeiten. Die Verbindung zwischen deinem lokalen Repository und GitHub herzustellen ist ein wichtiger Meilenstein, und ich zeige dir hier ausführlich, wie das funktioniert.

Das Grundprinzip verstehen

Bevor wir loslegen, ist es hilfreich zu verstehen, was hier eigentlich passiert:

```
flowchart LR
    LOCAL["□□Lokales Repository\nauf deinem Computer"] -->|git push| REMOTE["☁□ Remote Repository\nauf GitHub"]
    REMOTE -->|git pull| LOCAL
```

Dein lokales Repository enthält bereits deine gesamte Commit-Historie. GitHub stellt dir einen **Remote** zur Verfügung – das ist im Grunde eine Kopie deines Repositories auf einem Server, mit der du dein lokales Repository synchronisieren kannst. Die Verbindung zwischen beiden nennt man **Remote-Verknüpfung**, und der Standard-Name für die Haupt-Remote ist `origin`.

Schritt 1: Ein neues Repository auf GitHub erstellen

Zunächst brauchst du ein „Zuhause" für deinen Code auf GitHub. Dieses Repository erstellst du direkt auf der GitHub-Website.

1. **Bei GitHub anmelden**

   Öffne github.com und melde dich mit deinem Account an. Falls du noch keinen hast, musst du zuerst einen erstellen.

2. **Neues Repository anlegen**
   Klicke oben rechts auf das **+**-Symbol und wähle **New repository** (oder gehe direkt zu github.com/new).
3. **Repository-Einstellungen festlegen**
   Jetzt siehst du ein Formular mit mehreren Optionen:
   - **Repository name:** Wähle einen aussagekräftigen Namen für dein Projekt. Am besten verwendest du den gleichen Namen wie dein lokaler Projektordner – das ist zwar nicht zwingend notwendig, macht es aber übersichtlicher. Beispiel: `mein-webprojekt`
   - **Description (optional):** Eine kurze Beschreibung deines Projekts. Das ist besonders nützlich, wenn du das Repository später öffentlich machst oder selbst nach Monaten noch wissen willst, worum es geht.
   - **Visibility – Public oder Private:**
     - **Public:** Jeder kann dein Repository sehen (aber nur du kannst Änderungen pushen, solange du niemanden einlädst).
     - **Private:** Nur du und eingeladene Personen können das Repository sehen.

     Für persönliche Projekte oder Lernprojekte ist „Private" oft die bessere Wahl – du kannst es später jederzeit öffentlich machen.
   - **Initialize this repository with… ⚠ Wichtig:**
     Da du bereits ein lokales Repository mit Commits hast, solltest du hier **nichts auswählen**:
     - ☐ Kein Häkchen bei „Add a README file"
     - ☐ Kein Häkchen bei „Add .gitignore"
     - ☐ Kein Häkchen bei „Choose a license"

     Warum? Wenn GitHub diese Dateien automatisch erstellt, hat das neue Remote-Repository bereits einen Commit. Dein lokales Repository hat aber eine andere Historie – und das führt zu Konflikten beim ersten Push. Ein **komplett leeres Repository** lässt sich dagegen problemlos mit deinem bestehenden lokalen Repository verbinden.
4. **Repository erstellen**
   Klicke auf **Create repository**. GitHub erstellt das Repository und zeigt dir eine Seite mit Anleitungen – genau diese werden wir im nächsten Schritt nutzen.

Schritt 2: Die Remote-URL kopieren

Nach der Erstellung zeigt GitHub dir eine Seite mit verschiedenen Optionen. Da du bereits ein lokales Repository hast, interessiert dich der Abschnitt **„…or push an existing repository from the command line"**.

Bevor du die Befehle ausführst, brauchst du die **URL deines Repositories**. GitHub bietet zwei Varianten an:

Protokoll	URL-Format	Wann verwenden
HTTPS	`https://github.com/benutzername/repo.git`	Einfacher Einstieg, aber du musst dich bei jedem Push authentifizieren (oder einen Credential Manager nutzen)
SSH	`git@github.com:benutzername/repo.git`	Empfohlen, wenn du SSH-Keys eingerichtet hast – keine Passwort-Eingabe nötig

Falls du SSH-Keys bereits eingerichtet hast (siehe Kapitel zu SSH-Keys), wähle **SSH**. Ansonsten funktioniert **HTTPS** genauso gut – du wirst dann beim ersten Push nach deinen GitHub-Zugangsdaten gefragt.

Klicke auf den entsprechenden Tab (HTTPS oder SSH) und kopiere die angezeigte URL.

Schritt 3: Remote in PhpStorm hinzufügen

Jetzt verbindest du dein lokales Repository mit dem frisch erstellten GitHub-Repository. Das machst du, indem du eine **Remote** hinzufügst.

Variante A: Über das PhpStorm-Menü (empfohlen für Einsteiger)

1. **Git-Menü öffnen**
   Gehe in PhpStorm zu **Git → Manage Remotes…** (in älteren Versionen: **VCS → Git → Remotes…**).
2. **Neue Remote hinzufügen**
   Es öffnet sich ein Dialog, der alle konfigurierten Remotes anzeigt – bei einem frischen lokalen Repository ist diese Liste leer.
   Klicke auf das **+**-Symbol, um eine neue Remote hinzuzufügen.
3. **Name und URL eingeben**
   - **Name:** Gib `origin` ein. Das ist der Standardname für die Haupt-Remote und wird von Git und den meisten Tools erwartet.
   - **URL:** Füge die kopierte GitHub-URL ein (HTTPS oder SSH).

   Klicke auf **OK**.

4. **Fertig!**
   Die Remote ist jetzt konfiguriert. PhpStorm weiß nun, wo dein Code auf GitHub „wohnt".

Variante B: Über das Terminal in PhpStorm

Falls du lieber mit Befehlen arbeitest, kannst du das integrierte Terminal nutzen:

1. Öffne das Terminal in PhpStorm (**View → Tool Windows → Terminal** oder `Alt + F12`).
2. Gib folgenden Befehl ein (ersetze die URL durch deine eigene):

   ```
   git remote add origin https://github.com/dein-benutzername/mein-webprojekt.git
   ```

   Oder mit SSH:

   ```
   git remote add origin git@github.com:dein-benutzername/mein-webprojekt.git
   ```

3. Überprüfe, ob die Remote korrekt hinzugefügt wurde:

   ```
   git remote -v
   ```

   Du solltest so etwas sehen:

   ```
   origin  https://github.com/dein-benutzername/mein-webprojekt.git (fetch)
   origin  https://github.com/dein-benutzername/mein-webprojekt.git (push)
   ```

Schritt 4: Den ersten Push durchführen

Jetzt kommt der spannende Moment: Du lädst deine gesamte lokale Commit-Historie auf GitHub hoch!

In PhpStorm (grafisch)

1. **Push-Dialog öffnen**
   Gehe zu **Git → Push…** (oder nutze das Tastenkürzel `Ctrl + Shift + K` auf Windows/Linux bzw. `Cmd + Shift + K` auf macOS).
2. **Push-Vorschau prüfen**

PhpStorm zeigt dir an, welche Commits gepusht werden – das sollten alle deine bisherigen lokalen Commits sein.
Du siehst auch, zu welchem Branch gepusht wird. Standardmäßig ist das `main` (oder `master`, je nach deiner Git-Konfiguration).

3. **Push ausführen**
   Klicke auf **Push**. Falls du HTTPS verwendest und noch keine Zugangsdaten gespeichert hast, wirst du nach deinem GitHub-Benutzernamen und Passwort gefragt (oder nach einem **Personal Access Token** – dazu gleich mehr).

Im Terminal

Alternativ kannst du den Push auch über das Terminal durchführen:

```
git push -u origin main
```

> **Was bedeutet `-u`?**
>
> Die Option `-u` (oder `--set-upstream`) sorgt dafür, dass Git sich merkt, dass dein lokaler `main`-Branch mit dem Remote-Branch `origin/main` verknüpft ist. Danach reicht ein einfaches `git push` ohne weitere Parameter.

Falls dein Hauptbranch `master` heißt (bei älteren Git-Versionen oder Projekten), ersetze `main` durch `master`.

Authentifizierung bei HTTPS: Personal Access Token (PAT)

Falls du HTTPS verwendest und GitHub nach einem Passwort fragt, **funktioniert dein normales GitHub-Passwort nicht mehr** für Git-Operationen. Stattdessen benötigst du einen **Personal Access Token (PAT)**.

Einen PAT erstellen

1. Gehe auf GitHub zu **Settings → Developer settings → Personal access tokens → Tokens (classic)** (oder direkt: github.com/settings/tokens).

2. Klicke auf **Generate new token (classic)**.
3. Gib dem Token einen beschreibenden Namen (z. B. „PhpStorm auf Laptop").
4. Wähle ein **Ablaufdatum** – für mehr Sicherheit empfiehlt sich ein begrenzter Zeitraum (z. B. 90 Tage), den du bei Bedarf verlängern kannst.
5. Setze mindestens folgende **Berechtigungen (Scopes)**:
   - `repo` – Vollzugriff auf Repositories (nötig für Push/Pull)
6. Klicke auf **Generate token** und **kopiere den Token sofort** – er wird nur einmal angezeigt!

Den Token verwenden

Wenn PhpStorm (oder Git) nach dem Passwort fragt, gibst du statt deines GitHub-Passworts den **Personal Access Token** ein.

> **Tipp:** PhpStorm kann den Token speichern, sodass du ihn nicht jedes Mal neu eingeben musst. Achte darauf, dass in den Einstellungen unter **Appearance & Behavior → System Settings → Passwords** ein sicherer Speicherort konfiguriert ist (z. B. der Systemschlüsselbund).

Nach dem Push: Überprüfen, ob alles geklappt hat

1. **Auf GitHub nachschauen**
   Öffne dein Repository auf GitHub (z. B. `https://github.com/dein-benutzername/mein-webprojekt`). Du solltest jetzt alle deine Dateien und die gesamte Commit-Historie sehen können.
2. **In PhpStorm**
   Öffne das Git-Log (**Git → Show Git Log**). Du siehst jetzt neben deinen lokalen Branches auch den Remote-Branch `origin/main`, der auf denselben Commit zeigt wie dein lokaler `main`.

Zusammenfassung des gesamten Ablaufs

Hier nochmal alle Schritte im Überblick:

```
flowchart TD
    A["1. Leeres Repository\nauf GitHub erstellen"] --> B["2. Remote-URL kopieren\nHTTPS oder SSH"]
    B --> C["3. Remote in PhpStorm\nhinzufuegen via\nGit > Manage Remotes"]
    C --> D["4. Ersten Push durchfuehren\nGit > Push"]
    D --> E["5. Auf GitHub pruefen\nob alles angekommen ist"]
```

Häufige Probleme und Lösungen

„Failed to push: rejected – non-fast-forward"

Dieser Fehler tritt auf, wenn das Remote-Repository bereits Commits enthält, die du lokal nicht hast (z. B. weil du bei der Erstellung versehentlich eine README-Datei hast anlegen lassen).

Lösung:

1. Ziehe zuerst die Remote-Änderungen mit `git pull origin main --allow-unrelated-histories`
2. Löse eventuelle Merge-Konflikte
3. Committe und pushe erneut

Besser: Erstelle das GitHub-Repository wirklich komplett leer (ohne README, ohne .gitignore, ohne Lizenz).

„Permission denied (publickey)"

Dieser Fehler erscheint, wenn du SSH verwendest, aber dein SSH-Key nicht korrekt eingerichtet ist.

Lösung:

- Überprüfe, ob dein SSH-Key bei GitHub hinterlegt ist (unter github.com/settings/keys)
- Teste die Verbindung mit `ssh -T git@github.com`
- Alternativ: Wechsle zu HTTPS, bis du SSH korrekt eingerichtet hast

„Repository not found"

Entweder existiert das Repository nicht, der Name ist falsch geschrieben, oder du hast keine Zugriffsrechte (bei privaten Repositories).

Lösung:

- Überprüfe die URL auf Tippfehler
- Stelle sicher, dass du bei GitHub angemeldet bist
- Bei privaten Repos: Prüfe, ob dein Account Zugriff hat

Wie es danach weitergeht

Nach dem ersten Push ändert sich dein Workflow nur minimal:

1. **Lokal arbeiten und committen** – wie bisher
2. **Regelmäßig pushen** – um deine Änderungen auf GitHub zu sichern (`Git → Push` oder `Ctrl + Shift + K`)
3. **Bei Bedarf pullen** – falls du von einem anderen Gerät aus gepusht hast oder mit anderen zusammenarbeitest (`Git → Pull`)

Die Verknüpfung zwischen lokalem und Remote-Repository bleibt dauerhaft bestehen – du musst die Remote-Konfiguration nur einmal pro Projekt durchführen. □□