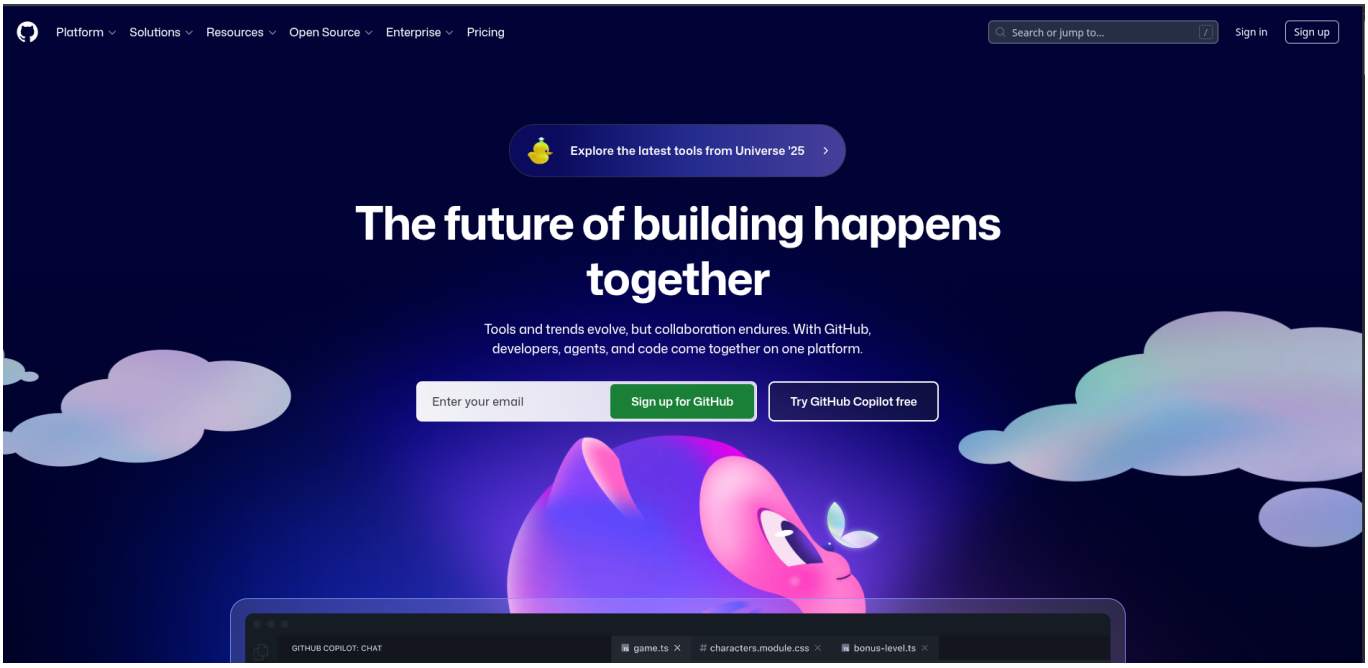
Einführung in die Versionskontrolle mit Git und GitHub

Weitere Informationen findest du in der offiziellen GitHub-Dokumentation.



GitHub Landing Page

Was wir behandeln werden:

- Grundlagen der Versionskontrolle
    - Prinzipien von Versionskontrollsystemen
    - Unterschiede zwischen zentralisierter und verteilter Versionskontrolle
- Einführung in Git und GitHub
    - Git-Einführung
        - Was ist Git?
        - Voraussetzungen und Installation von Git auf verschiedenen Betriebssystemen
        - Grundlegende Konfiguration von Git (user.name, user.email)
    - GitHub-Einführung
        - Was ist GitHub und warum solltest du es verwenden?
        - Erstellen eines GitHub-Kontos
    - Git mit GitHub verbinden
        - Einrichten von SSH-Schlüsseln für die sichere Kommunikation mit GitHub

        - Generieren von SSH-Schlüsseln
        - Hinzufügen von SSH-Schlüsseln zu deinem GitHub-Konto
    - Arbeiten mit persönlichen Repositories
        - Initialisierung eines neuen Git-Repositorys (git init)
        - Überprüfen des Status deines Repositorys (git status)
        - Hinzufügen von Änderungen zum Staging-Bereich (git add)
        - Committen von Änderungen in das Repository (git commit)
        - Speichern von Änderungen in einem Remote-Repository auf GitHub (git push)
        - Aufrufen der Git-Hilfe und Selbsthilfeoptionen

Grundlagen der Versionskontrolle

- Stell dir vor, du arbeitest an einem Dokument und möchtest:
    - Änderungen, die du vornimmst, nachverfolgen
    - Bei Bedarf zu früheren Versionen zurückkehren
    - Andere einladen, mit dir zusammenzuarbeiten
    - Änderungen von anderen annehmen, ohne deine Arbeit zu überschreiben
- Versionskontrollsysteme (VCS) helfen dir, dies und mehr zu erreichen.

Was ist Versionskontrolle?

- Ein System, das Änderungen an einer Datei oder einer Reihe von Dateien im Laufe der Zeit aufzeichnet.
- Stell es dir wie ein intelligentes Notizbuch vor, das jede von dir vorgenommene Änderung festhält.
- Ermöglicht es dir, zu früheren Versionen zurückzukehren, Änderungen zu vergleichen und mit anderen zusammenzuarbeiten.
- Ein Versionskontrollsystem verfolgt vier Hauptaspekte:
    - **WAS wurde geändert** (der Inhalt): *Zeile 5 in Dokument X wurde gelöscht, ein Absatz in Datei Y wurde geändert, Bild Z wurde hinzugefügt*
    - **WER hat die Änderung vorgenommen** (der Autor): *der Benutzername der Person, die die Änderung vorgenommen hat*
    - **WANN wurde die Änderung vorgenommen** (der Zeitstempel): *Datum und Uhrzeit der Änderung*
    - **WARUM wurde die Änderung vorgenommen** (die Commit-Nachricht): *eine kurze Beschreibung des Grundes für die Änderung, die vom Autor verfasst wurde*

Unterschiede zwischen zentralisierter und verteilter Versionskontrolle

- Es gibt zwei grundlegende Architekturen, die Versionskontrollsysteme haben können.
    - Zentralisiertes Versionskontrollsystem (CVCS)
    - Verteiltes Versionskontrollsystem (DVCS)
- Beide folgen den oben genannten Prinzipien, aber auf unterschiedliche Weise.

Zentralisiertes Versionskontrollsystem (CVCS)

- Stell dir eine Bibliothek mit einem einzigen, großen Bücherregal in der Mitte des Raumes vor
- das ist der **zentrale Server**.
- In diesem Regal befindet sich die **EINZIGE** Originalkopie der gesamten Projekthistorie.
- Wenn du an etwas arbeiten möchtest,
    - gehst du zu diesem Regal,
    - leihst dir eine Kopie der neuesten Version einer Datei aus (ein "Checkout").
    - und arbeitest an deiner Kopie an deinem Schreibtisch.
- Wenn du fertig bist,
    - gibst du die Datei zurück
    - reichst deine Änderungen ein ("commit").
    - Deine Änderungen werden direkt auf dem zentralen Server gespeichert.
- **Vorteile:**
    - Sehr einfach zu verstehen und zu verwalten.
    - Es gibt immer ein klares Verständnis davon, was die "offizielle" Version ist.
- **Nachteile:**
    - **Single Point of Failure:**
        - Wenn der zentrale Server ausfällt (z. B. Stromausfall, Festplattencrash), kann niemand arbeiten.
        - Niemand kann Commits durchführen, Versionen vergleichen oder Änderungen abrufen.
        - die gesamte Projekthistorie geht verloren.
    - **Online-Anforderung:**
        - Für fast alle Aktionen (Committen, Verlauf anzeigen, Branches erstellen) benötigst du eine Verbindung zum zentralen Server.
        - Offline-Arbeit (z. B. im Zug) ist sehr eingeschränkt.
- **Bekannte Beispiele:** Subversion (SVN), CVS, Perforce.

Verteiltes Versionskontrollsystem (DVCS)

- Stell dir nun ein anderes System vor.
- Anstelle einer zentralen Bibliothek gibt es einen Hauptserver, aber jeder Entwickler erhält beim Auschecken nicht nur die neueste Version, sondern eine **vollständige 1:1-Kopie der gesamten Bibliothek** für den eigenen Schreibtisch.
- Jeder Entwickler hat also die vollständige Projekthistorie lokal auf seinem eigenen Computer.
- Du arbeitest an deinem Projekt und machst Commits.
    - Diese Commits werden in deiner *lokalen* Kopie des Repositorys gespeichert.
    - Du kannst Branches erstellen, den Verlauf anzeigen, Versionen vergleichen – alles blitzschnell und ohne Internetverbindung.

- Nur wenn du deine Arbeit mit anderen teilen möchtest, kannst du
    - dich mit dem Hauptserver verbinden und
    - deine Änderungen dorthin pushen ("push").
    - Um die Änderungen anderer zu erhalten, holst du sie vom Server ab ("pull").
- **Vorteile:**
    - **Kein Single Point of Failure:**
        - Wenn der Hauptserver ausfällt, ist das ärgerlich, aber nicht tragisch.
        - Jeder Entwickler hat eine vollständige Kopie der Projekthistorie auf seinem Computer.
        - Das Projekt kann einfach aus einer dieser Kopien wiederhergestellt werden.
    - **Hervorragende Offline-Fähigkeit:**
        - Du kannst im Flugzeug sitzen und Dutzende von Commits machen, neue Branches erstellen und deine gesamte Arbeit organisieren.
        - Erst wenn du wieder eine Verbindung hast, teilst du deine Arbeit.
    - **Geschwindigkeit:**
        - Da die meisten Operationen (Commit, Verlauf anzeigen) lokal stattfinden, sind sie extrem schnell.
        - Es gibt keine Netzwerkverzögerung.
    - **Flexibles Branching & Merging:**
        - Das Erstellen von Branches in einem DVCS ist extrem einfach und schnell, was experimentelle und parallele Entwicklungsworkflows stark fördert.
- **Nachteile:**
    - Die Lernkurve ist anfangs etwas steiler, da du die Unterscheidung zwischen dem lokalen und dem Remote-Repository verstehen musst (z. B. Konzepte wie Push & Pull).
- **Bekannte Beispiele: Git** (mit Abstand das beliebteste), Mercurial.

Einführung in Git und GitHub

Git-Einführung

- **Was ist Git?**
    - Git ist ein verteiltes Versionskontrollsystem (DVCS), das 2005 von Linus Torvalds entwickelt wurde.
    - Es ermöglicht mehreren Entwicklern, gleichzeitig am selben Projekt zu arbeiten, ohne die Änderungen der anderen zu überschreiben.
    - Git verfolgt Änderungen an Dateien, sodass du zu früheren Versionen zurückkehren, Änderungen vergleichen und effektiv zusammenarbeiten kannst.

Voraussetzungen und Installation von Git

- Git ist eine kostenlose und quelloffene Software.

- Es kann auf verschiedenen Betriebssystemen installiert werden, einschließlich Windows, macOS und Linux.
- Hier ist die offizielle Git-Website für Download- und Installationsanweisungen: https://git-scm.com/install/
- **Installation von Git unter Linux:(Link zur offiziellen Git-Installationsseite)**
    - Öffne dein Terminal (Strg + Alt + T)
    - Überprüfe, ob du Git installiert hast mit:

```
git --version
```

    - Wenn Git nicht installiert ist, kannst du es mit deinem Paketmanager installieren.
    - Aktualisiere und upgrade nun deine Paketlisten:

```
sudo apt update
sudo apt upgrade
```

    - Installiere dann Git mit:

```
sudo apt-get install git
```

    - Überprüfe die Installation mit:

```
git --version
```

- **Installation von Git unter Windows:**
    - Lade den Git- (und Git Bash-) Installer von https://gitforwindows.org/ herunter.
    - Führe den Installer aus und folge den Einrichtungsanweisungen.

Grundlegende Konfiguration von Git

- Nach der Installation von Git ist es wichtig, deine Benutzerinformationen zu konfigurieren (Das WER, das die Versionskontrolle benötigt).
- Diese Informationen werden mit deinen Commits (dem WARUM-Teil) verknüpft.
- Dies ist eine **EINMALIGE Einrichtung**, wenn du Git zum ersten Mal auf einem neuen Computer installierst.
- Du musst **DIESELBE E-MAIL-ADRESSE** angeben, die du für dein GitHub-Konto verwendest.
- Öffne dein Terminal (oder Git Bash unter Windows) und führe die folgenden Befehle aus:

```

```

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

- Du kannst deine Konfiguration überprüfen mit:

```
git config --list
```

- Falls erforderlich, setze deinen Standard-Branch-Namen auf "main":

```
git config --global init.defaultBranch main
```

  - Dies stellt sicher, dass neue von dir erstellte Repositories "main" als Standard-Branch-Namen anstelle von "master" verwenden.
  - Diese Änderung wurde vorgenommen, um eine inklusive Sprache in der Softwareentwicklung zu fördern.
  - Viele Plattformen, einschließlich GitHub, haben "main" als Standard-Branch-Namen für neue Repositories übernommen.
- Überprüfe deine Konfiguration erneut mit:

```
git config --list
```

- Du solltest deinen Benutzernamen und deine E-Mail-Adresse in den Konfigurationseinstellungen sehen. Meine sieht zum Beispiel so aus:

```
user.email=meineprivate@gmail.com
user.name=Ndimofor
init.defaultbranch=main
```

GitHub-Einführung

- **Was ist GitHub und warum solltest du es verwenden?**
  - GitHub ist eine cloudbasierte Plattform, auf der du
    - Code oder andere textbasierte Dateien mit Git speichern,
    - teilen,
    - und gemeinsam mit anderen daran arbeiten kannst.
  - Es bietet eine Weboberfläche für Git-Repositories, die die Verwaltung und Zusammenarbeit bei Projekten erleichtert.

    - GitHub bietet zusätzliche Funktionen wie Issue-Tracking, Projektmanagement-Tools und Integrationen mit anderen Diensten.
    - Stell es dir wie einen Speicherort für deinen Code, Markdown-Dateien und andere projektbezogene Dokumente vor, auf die du von überall zugreifen und sie mit anderen teilen kannst.

Erstellen eines GitHub-Kontos

- Um GitHub zu nutzen, musst du ein Konto erstellen.
- BITTE VERWENDE DEINE PERSÖNLICHE E-MAIL-ADRESSE DAFÜR!
- Öffne deinen persönlichen E-Mail-Posteingang in einem neuen Tab, um dein Konto später zu verifizieren.
- Stelle sicher, dass du dir deinen Benutzernamen und dein Passwort merkst.
- Folge diesem Link, um dich anzumelden

Git mit GitHub verbinden

- **Wie Git und GitHub zusammenarbeiten**
    - Git verfolgt Änderungen an deinen Dateien lokal auf deinem Computer (dein *lokales* Repository).
    - GitHub ist die Online-Plattform, auf der du dieses Git-Repository speichern kannst (dein _Remote-_Repository).
    - Wenn du Dateien auf GitHub hochlädst, speicherst du sie in einem "Git-Repository".
    - Wenn du lokal Änderungen an deinen Dateien vornimmst, kannst du Git verwenden, um diese Änderungen zu verfolgen.
    - Um dein Remote-Repository auf GitHub mit deinen lokalen Änderungen zu aktualisieren, "pushst" du diese Änderungen auf GitHub.
    - Für all dies musst du eine sichere Verbindung zwischen deiner lokalen Git-Installation und deinem GitHub-Konto einrichten.
    - Wir werden dies mit SSH-Schlüsseln tun.

Einrichten von SSH-Schlüsseln für die sichere Kommunikation mit GitHub

- **Was sind SSH-Schlüssel?**
    - SSH- (Secure Shell) Schlüssel sind ein Paar kryptografischer Schlüssel, die zur sicheren Authentifizierung deines Computers bei GitHub verwendet werden.
    - Sie bestehen aus einem öffentlichen Schlüssel (den du mit GitHub teilst) und einem privaten Schlüssel (den du auf deinem Computer geheim hältst).
    - Die Verwendung von SSH-Schlüsseln ermöglicht es dir, dich mit GitHub zu verbinden, ohne bei jedem Push oder Pull von Änderungen deinen Benutzernamen und dein Passwort eingeben zu müssen.
- **Generieren von SSH-Schlüsseln (Linux Ubuntu)**
    - Öffne dein Terminal (oder Git Bash unter Windows).
    - Überprüfe, ob du bereits SSH-Schlüssel hast, indem du Folgendes ausführst:

```
ls -al ~/.ssh
```

- Wenn du Dateien mit den Namen `id_rsa` und `id_rsa.pub` siehst, hast du bereits SSH-Schlüssel.
- Du kannst zum Hinzufügen des Schlüssels zu GitHub springen.
- Wenn nicht, generiere ein neues SSH-Schlüsselpaar mit dem folgenden Befehl (ersetze "deine_email@example.com" durch deine tatsächliche E-Mail-Adresse):

```
ssh-keygen -t ed25519 -C "deine_email@example.com"
```

  - Wenn du aufgefordert wirst, "Enter a file in which to save the key," drücke die Eingabetaste, um den Standardspeicherort zu akzeptieren.
  - Wenn du nach einer Passphrase gefragt wirst, drücke die Eingabetaste.
  - Wenn du aufgefordert wirst, die Passphrase zu bestätigen, drücke erneut die Eingabetaste.
- Dies erstellt ein neues SSH-Schlüsselpaar im `~/.ssh`-Verzeichnis.

- **Hinzufügen von SSH-Schlüsseln zu deinem GitHub-Konto**
  - Starte den SSH-Agenten im Hintergrund, indem du Folgendes ausführst:

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

    - du solltest eine Nachricht wie `Agent pid 59566` sehen.
  - Füge deinen privaten SSH-Schlüssel zum SSH-Agenten hinzu mit:

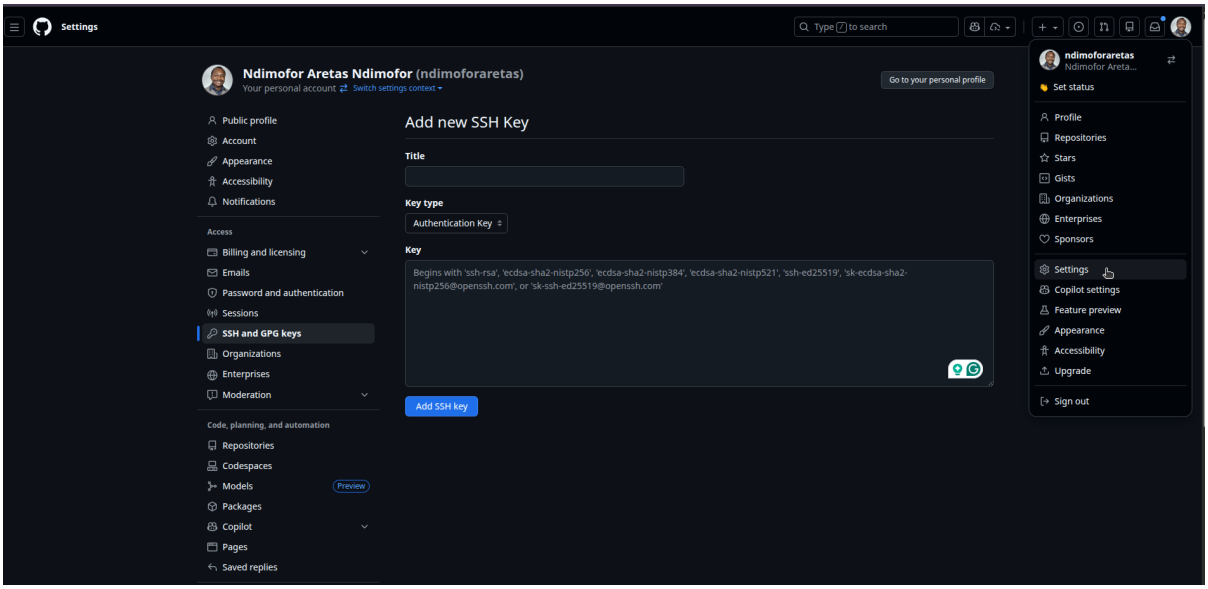
```
ssh-add ~/.ssh/id_ed25519
```

- **Hinzufügen des SSH-Schlüssels zu deinem GitHub-Konto**
  - Kopiere den Inhalt deines öffentlichen SSH-Schlüssels in deine Zwischenablage, indem du Folgendes ausführst:

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

    - Wähle und kopiere die gesamte Ausgabe (sie beginnt mit `ssh-ed25519` und endet mit deiner E-Mail-Adresse).

- Melde dich bei deinem GitHub-Konto an.



Schritte zum Hinzufügen des SSH-Schlüssels

  - Klicke auf dein Profilbild in der oberen rechten Ecke.
  - Gehe zu **Settings** > **SSH and GPG keys** > **New SSH key**.
  - Gib ihm einen aussagekräftigen Titel, wie "DCI Linux Laptop SSH Key".
  - Füge den kopierten Schlüssel in das "Key"-Feld ein.
  - Klicke auf **Add SSH key**, um ihn zu speichern.
- **Testen der SSH-Verbindung**
  - Um zu testen, ob alles korrekt eingerichtet ist, führe Folgendes aus:

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

  - Du solltest eine Nachricht wie diese sehen:

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

  - Das bedeutet, dein SSH-Schlüssel ist korrekt eingerichtet, und du kannst dich jetzt sicher mit Git bei GitHub verbinden.

Arbeiten mit persönlichen Repositories

- Es gibt zwei Arten von Repositories, mit denen du auf GitHub arbeiten kannst:
  - **Lokales Repository:**
    - Dies ist das Repository, das du auf deinem eigenen Computer erstellst.
    - Du musst dieses lokale Repository dann auf GitHub pushen, um ein Remote-Repository zu erstellen.
  - **Remote-Repository:**
    - Dies ist das Repository, das du direkt auf GitHub erstellst.

        - Du musst dieses Remote-Repository dann auf deinen Computer klonen, um ein lokales Repository zu erstellen.
- Wir werden uns darauf konzentrieren, zuerst ein lokales Repository zu erstellen und es dann auf GitHub zu pushen, um ein Remote-Repository zu erstellen.
- Heute werden wir dies über die VS Code Version Control-Oberfläche tun.