Kickstart-Guide

A quick reference guide for developers

Ahmet Emirhan Göktaş

Table of contents

1. Kickstart-Guide

Willkommen zum Kickstart-Guide! Dieser Guide wird Ihnen den Einstieg mit verschiedenen Tools erleichtern. Aktuell beinhaltet der Guide die folgenden Tools:

- Git

Der Quellcode für den Guide ist auf GitHub verfügbar.

1.1 Feedback

Falls Sie Rückmeldungen oder Vorschläge zur Verbesserung haben, zögern Sie nicht, ein Issue im GitHub-Repository zu erstellen. Ihr Feedback ist sehr willkommen!

2. Anleitungen

2.1 Git

2.1.1 Git

Git ist ein verteiltes Versionskontrollsystem, das es mehreren Entwicklern ermöglicht, an einem Projekt zusammenzuarbeiten. Es bietet eine Möglichkeit, Änderungen zu verfolgen und Codeänderungen zu mergen. Git wird in der Softwareentwicklungsbranche weit verbreitet eingesetzt und ist ein unverzichtbares Werkzeug für jeden Entwickler. Es gibt bereits viel Dokumentation über Git, und eine der besten Ressourcen ist der GitHub Git Guide. Dieser Guide wird kurz die Grundlagen behandeln und dann in fortgeschrittenere Themen eintauchen.

Inhaltsverzeichnis:

2.1.2 Installation

Sie können überprüfen, ob Git auf Ihrem System bereits installiert ist, indem Sie den folgenden Befehl ausführen:

```
git version
```

Die Ausgabe sollte ähnlich wie folgt sein:

```
git version 2.43.1
```

Wenn Sie einen Fehler wie `git: Befehl nicht gefunden` erhalten, können Sie den folgenden Anweisungen folgen, um Git zu installieren.

Installation unter Windows

Die Installation von `git` unter Windows ist unkompliziert.

1. Laden Sie den Installer von der offiziellen Webseite herunter.
2. Führen Sie den Installer aus und folgen Sie den Anweisungen.
3. Nach der Installation können Sie das Terminal öffnen und `git version` ausführen, um zu überprüfen, ob die Installation erfolgreich war.

> **Tip**
>
> Unter Windows 10 oder später können Sie auch WSL nutzen, das Ihnen erlaubt, Linux auf Windows auszuführen. Dies ermöglicht es Ihnen, in einer Linux-Umgebung zu sein, während Sie Windows verwenden.

> **Note**
>
> Wählen Sie bei der `Anpassung Ihrer PATH-Umgebung` die Option `Git von der Kommandozeile und auch von Drittanbieter-Software`. Dies ermöglicht es Ihnen, `git` aus dem Windows-Terminal zu nutzen. Dieser Artikel geht davon aus, dass Sie dies getan haben. Andernfalls müssen Sie das Git Bash Terminal verwenden, das mit dem Installer geliefert wird, für die Befehle in diesem Guide.

Installation unter macOS

Normalerweise ist `git` bereits auf macOS installiert. Aber falls es aus irgendeinem Grund nicht installiert ist, können Sie es entweder mit Homebrew oder durch Herunterladen des Installers installieren.

MIT HOMEBREW

Homebrew ist ein Paketmanager für macOS. Wenn Sie Homebrew installiert haben, können Sie `git` einfach installieren.

1. Öffnen Sie das Terminal und führen Sie den folgenden Befehl aus, um `git` zu installieren:

   ```
   brew install git
   ```

2. Nach der Installation können Sie `git version` ausführen, um zu überprüfen, ob die Installation erfolgreich war.

MIT DEM INSTALLER

1. Laden Sie den neuesten Installer herunter.
2. Führen Sie den Installer aus und folgen Sie den Anweisungen.
3. Nach der Installation können Sie die Installation überprüfen, indem Sie `git version` im Terminal ausführen.

Installation unter Linux

Sie können `git` unter Linux mit dem Paketmanager Ihrer Distribution installieren.

DEBIAN/UBUNTU ODER DERIVATE

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

FEDORA ODER DERIVATE

```
sudo dnf install git-all
```

ARCH LINUX ODER DERIVATE

```
sudo pacman -S git
```

Nach der Installation können Sie die Installation überprüfen, indem Sie `git version` im Terminal ausführen.

Konfiguration

Die Installation von Git ermöglicht es Ihnen, Repositories zu klonen und Änderungen in Ihren Projekten zu verfolgen. Jedoch müssen Sie Git mit Ihren Anmeldedaten konfigurieren, bevor Sie Änderungen an ein entferntes Repository pushen können. Dies kann durch Ausführen der folgenden Befehle im Terminal geschehen:

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

Sie können auch die `--global`-Flagge weglassen, um die Konfiguration für ein einzelnes Repository zu setzen. Dafür müssen Sie zu dem Repository navigieren und die gleichen Befehle ohne die `--global`-Flagge ausführen.

Von diesem Punkt an können Sie Git nutzen, um Änderungen in Ihrem Projekt zu verfolgen. Git wird Sie nach Ihren Anmeldedaten fragen, wenn Sie versuchen, Ihre Änderungen an ein entferntes Repository zu pushen. Sie können auch SSH-Schlüssel verwenden, um sich bei dem entfernten Repository zu authentifizieren. Dies ist nicht nur sicherer, sondern auch bequemer.

> **Note**
>
> Wenn Sie GitHub verwenden, müssen Sie SSH-Schlüssel verwenden, um sich bei dem entfernten Repository zu authentifizieren. GitHub hat die Unterstützung für Passwortauthentifizierung am 13. August 2021 entfernt.

SSH-SCHLÜSSEL

Um einen SSH-Schlüssel zu erstellen, können Sie den folgenden Befehl im Terminal ausführen:

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

Dies erstellt einen neuen SSH-Schlüssel unter Verwendung der bereitgestellten E-Mail als Label. Sie können den Standarddateispeicherort bei der Frage akzeptieren. Sie können Ihrem SSH-Schlüssel auch ein Passwort hinzufügen, was für eine zusätzliche Sicherheitsebene empfohlen wird. Ohne Passwort wird der SSH-Schlüssel auf Ihrem Computer als Klartext gespeichert.

> **Tip**
>
> Wenn ssh-keygen Sie fragt, ob die Datei überschrieben werden soll, bedeutet dies, dass Sie bereits in der Vergangenheit einen SSH-Schlüssel erstellt haben. Sie können entweder die Datei überschreiben oder einen neuen Dateinamen angeben. Um einen benutzerdefinierten SSH-Schlüssel zu erstellen, können Sie den Standarddateispeicherort eingeben und dann den Namen der neuen Datei.

Nachdem der SSH-Schlüssel generiert wurde, ist der nächste Schritt, ihn zu Ihrem entfernten Repository hinzuzufügen. Anleitungen für GitHub und für GitLab sind in ihrer jeweiligen Dokumentation verfügbar.

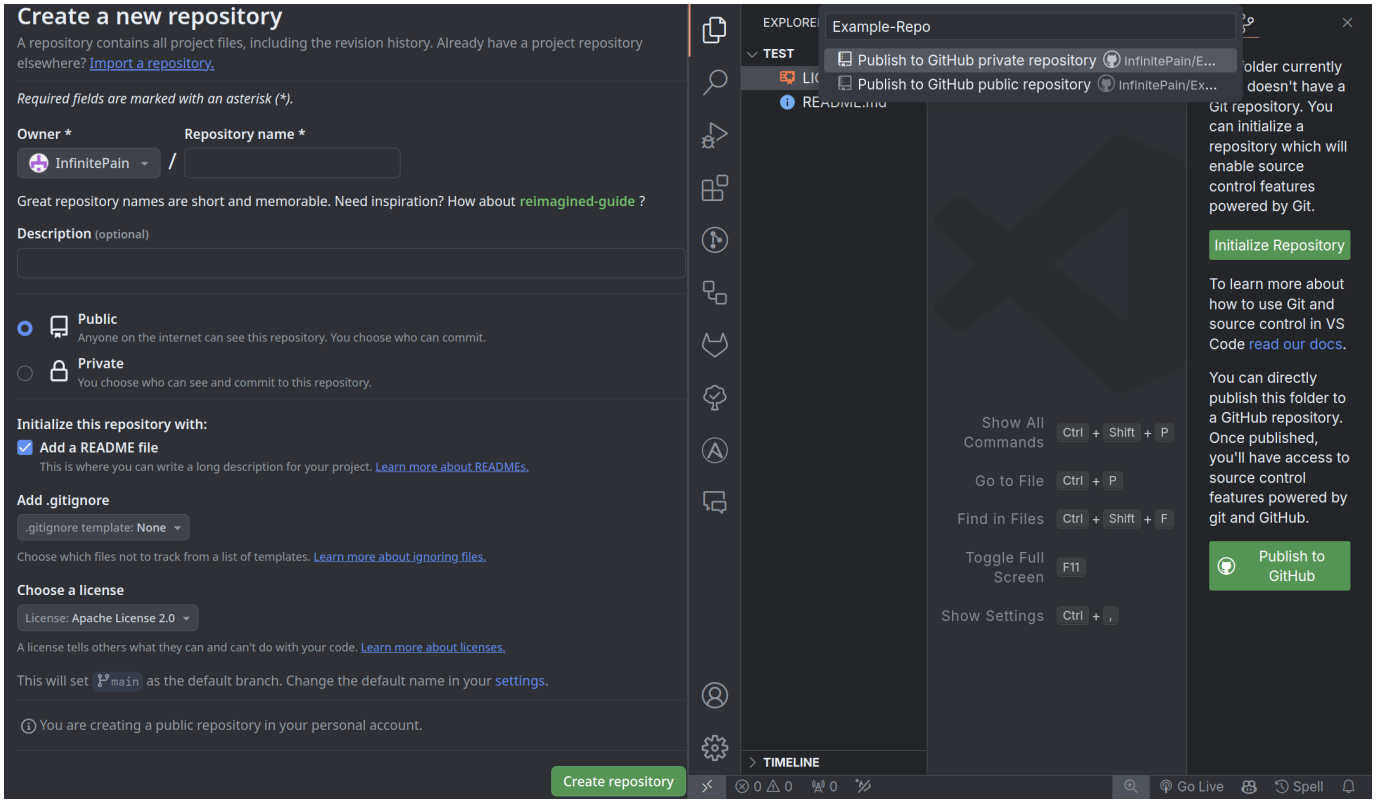
Sie können den SSH-Schlüssel auch Ihrem SSH-Agenten hinzufügen, was Ihnen erlaubt, den Schlüssel zu nutzen, ohne jedes Mal das Passwort eingeben zu müssen. Dafür hat GitHub eine gute Anleitung, wie man SSH-Schlüssel zum ssh-agent hinzufügt.

2.1.3 Grundlagen

In diesem Guide werden wir hauptsächlich die Kommandozeile verwenden, um mit Git zu interagieren. All dies kann jedoch auch mit einem GUI-Tool wie GitHub Desktop, GitKraken oder sogar VS Code durchgeführt werden.

Ein Repository erstellen

Es gibt mehrere Wege, dies zu tun. Sie können ein neues Repository auf GitHub mit einigen Anfangsdateien erstellen und es dann auf Ihren lokalen Rechner klonen. Oder Sie können ein neues Repository auf Ihrem lokalen Rechner erstellen und es dann auf GitHub veröffentlichen. In VS Code drücken Sie `^ Ctrl` + `⇧ Shift` + `P` oder `⌘ Cmd` + `⇧ Shift` + `P`, um die Befehlspalette zu öffnen und geben `Publish to GitHub` ein, um ein neues Repository auf GitHub zu erstellen.



Clone

Ein Repository zu clonen bedeutet, dass Sie eine Kopie des Repositories auf Ihrem lokalen Rechner anlegen. Dies kann durch Ausführen des folgenden Befehls im Terminal geschehen:

```
git clone <repository-url>
```

Tip

Wenn Sie Ihren SSH-Schlüssel zur Authentifizierung verwenden möchten, klonen Sie das Repository mit dem SSH-Pfad. Wenn Sie das Repository bereits mit HTTPS geklont haben, aber SSH zur Authentifizierung verwenden möchten, können Sie die Remote-URL mit dem folgenden Befehl ändern:

```
git remote set-url origin <new-repository-url>
```

Status

Der Befehl `git status` zeigt den Status des Arbeitsverzeichnisses und des Staging-Bereichs an. Er verändert nichts in Ihrem Repository, zeigt Ihnen nur den aktuellen Status an, wie welche Dateien gestaged, nicht gestaged oder nicht verfolgt werden.

Add

Der Befehl `git add` fügt Änderungen im Arbeitsverzeichnis zum Staging-Bereich hinzu und teilt Git effektiv mit, dass Sie Updates einer bestimmten Datei im nächsten Commit einschließen möchten.

Um spezifische Dateien hinzuzufügen, können Sie den folgenden Befehl ausführen:

```
git add <file>
```

Um alle Dateien hinzuzufügen, können Sie den folgenden Befehl ausführen:

```
git add .
```

> **Tip**
>
> Wenn Sie eine Datei aus einem Repository löschen, müssen Sie trotzdem `git add` ausführen, sonst wird git sich der Löschung nicht bewusst.

Commit

Der Befehl `git commit` wird verwendet, um Änderungen im lokalen Repository zu speichern. Es ist wie ein Snapshot Ihres Repositories zu einem bestimmten Zeitpunkt. Aber die Änderungen sind noch nicht im Remote-Repository.

Um Änderungen zu commiten, können Sie den folgenden Befehl ausführen:

```
git commit -m "commit message"
```

Es wird dringend empfohlen, sich über Conventional Commits zu informieren. Es ist eine Spezifikation, um Commit-Nachrichten menschen- und maschinenlesbare Bedeutung hinzuzufügen.

Push

Der Befehl `git push` wird verwendet, um Inhalte des lokalen Repositories auf ein Remote-Repository hochzuladen. Er wird verwendet, um die Commits, die Sie in Ihrem lokalen Repository gemacht haben, online zu stellen. Es wird empfohlen, die neuesten Änderungen aus dem Remote-Repository pullen, bevor Sie Ihre Änderungen pushen. Dadurch können Sie Konflikte lösen, bevor Sie Ihre Änderungen pushen und die Chancen auf Merge-Konflikte reduzieren.

Um Änderungen zu pushen, können Sie den folgenden Befehl ausführen:

```
git push
```

Pull

Der Befehl `git pull` wird verwendet, um die lokale Version eines Repositories von einem Remote zu aktualisieren. Dies kann durch Ausführen des folgenden Befehls geschehen:

```
git pull
```

2.1.4 Gitflow

Gitflow ist ein Branching-Modell, das für das Management der Entwicklung eines Projekts verwendet wird. Sie haben es wahrscheinlich in verschiedenen Open-Source-Projekten gesehen. Es ist im Grunde ein Satz von Regeln, der definiert, wie Branches erstellt und zusammengeführt werden, um die Entwicklung eines Projekts zu managen. Es besteht aus zwei Komponenten: den Hauptbranches und den unterstützenden Branches.

Die Hauptbranches

Die Hauptbranches sind die Branches, die verwendet werden, um die Veröffentlichung des Projekts zu managen. Sie sind:

1. `main`-Branch: Dies ist der Branch, der die offizielle Veröffentlichung des Projekts enthält.
2. `develop`-Branch: Dies ist der Branch, der die neueste Entwicklung des Projekts enthält und verwendet wird, um die Veröffentlichung des Projekts zu erstellen. Hier kommen alle neuen Features zusammen und werden getestet, bevor sie veröffentlicht werden.

Die unterstützenden Branches

Die unterstützenden Branches sind die Branches, die verwendet werden, um neue Features zu entwickeln und Bugs zu beheben. Sie sind:

1. `feature`-Branch: Dies ist der Branch, der zur Entwicklung neuer Features verwendet wird. Er wird vom `develop`-Branch abgezweigt und zurück in den `develop`-Branch gemerged, wenn das Feature fertig ist.
2. `release`-Branch: Dies ist der Branch, der zur Vorbereitung der Veröffentlichung des Projekts verwendet wird. Der Zweck dieses Branches ist es, die Entwicklung neuer Features zu stoppen und Bugs zu beheben. Er wird vom `develop`-Branch abgezweigt und zurück in den `main`-Branch gemerged, wenn die Veröffentlichung abgeschlossen ist.
3. `hotfix`-Branch: Dies ist der Branch, der zur Behebung von Bugs in der Veröffentlichung verwendet wird. Er wird vom `main`-Branch abgezweigt und zurück in den `main`-Branch gemerged, wenn der Bug behoben ist.

Der Workflow

Stellen wir uns ein Szenario vor, in dem wir ein Projekt haben, das entwickelt wird. Das Projekt hat die folgenden Branches:

1. `main`-Branch
2. `develop`-Branch

EIN FEATURE ERSTELLEN/BEENDEN

Wir möchten ein neues Feature für das Projekt entwickeln. Zuerst erstellen wir einen neuen Branch namens `feature-my-feature` vom `develop`-Branch.

```
# Erstelle einen neuen Branch namens feature-my-feature vom develop-Branch
git checkout -b feature-my-feature develop
```

Nachdem wir die Entwicklung des Features abgeschlossen haben, mergen wir den `feature-my-feature`-Branch zurück in den `develop`-Branch.

```
# Merge den feature-my-feature-Branch in den develop-Branch
git checkout develop
git merge --no-ff feature-my-feature
git branch -d feature-my-feature
```

> **Tip**
>
> Der letzte Befehl ist optional, aber empfohlen. Er löscht den `feature-my-feature`-Branch, nachdem er in den `develop`-Branch gemerged wurde. Keine Sorge, die Änderungen sind immer noch im `develop`-Branch. Wir möchten nur unsere Branches sauber halten.

EINE VERÖFFENTLICHUNG ERSTELLEN/BEENDEN

... Nach einiger Zeit erreichen wir einen Punkt, an dem wir das Projekt veröffentlichen möchten. In dieser Phase ist es entscheidend, wie wir unsere Veröffentlichungen versionieren. Hier kommt Semantic Versioning (SemVer) ins Spiel. SemVer ist ein Versionierungsschema für Software, das durch die Versionsnummer selbst Bedeutung über die zugrunde liegenden Änderungen in einer Veröffentlichung vermitteln soll. Es ist formatiert als `MAJOR.MINOR.PATCH`, wobei:

- **MAJOR**-Versionen inkompatible API-Änderungen anzeigen,
- **MINOR**-Versionen Funktionalität auf eine rückwärtskompatible Weise hinzufügen, und
- **PATCH**-Versionen rückwärtskompatible Bugfixes enthalten.

Für detailliertere Informationen zu SemVer und seinen Regeln besuchen Sie semver.org.

Nun, um unser Projekt für die Veröffentlichung vorzubereiten, erstellen wir

einen neuen Branch namens `release-1.0` vom `develop`-Branch. Die Versionsnummer `1.0` sollte den SemVer-Richtlinien folgen, was darauf hindeutet, dass dies unsere erste stabile Veröffentlichung mit einem Satz von fertiggestellten Features ist.

```
# Erstelle einen neuen Branch namens release-1.0 vom develop-Branch
git checkout -b release-1.0 develop
./bump-version.sh 1.0
git commit -a -m "chore(release): bump version number to 1.0"
```

Das `bump-version.sh`-Skript ist ein imaginäres Skript, das wir verwenden, um die Versionsnummer des Projekts zu erhöhen. Nachdem wir die Veröffentlichung vorbereitet haben, mergen wir den `release-1.0`-Branch zurück in den `main`-Branch.

```
git checkout main
git merge --no-ff release-1.0
# Tagge die Veröffentlichung
git tag -a 1.0
# Merge auch den develop-Branch, um ihn auf dem neuesten Stand mit der Veröffentlichung zu halten
git checkout develop
git merge --no-ff release-1.0
git branch -d release-1.0
```

> **Tip**
>
> Der letzte Befehl ist optional, aber empfohlen. Er löscht den `release-1.0`-Branch, nachdem er in den `main`-Branch gemerged wurde. Keine Sorge, die Änderungen sind immer noch im `main`-Branch. Wir möchten nur unsere Branches sauber halten.

EINEN BUG IN DER VERÖFFENTLICHUNG BEHEBEN

Nachdem wir die Version 1.0 unseres Projekts veröffentlicht haben, entdecken wir einen Bug, der sofort behoben werden muss. Es ist auch wichtig, sich an die SemVer-Prinzipien zu halten. Wenn der Bugfix rückwärtskompatibel ist und keine neuen Features einführt, sollte er die PATCH-Version erhöhen. Wenn wir beispielsweise einen Bug in Version `1.0` beheben, wäre die Hotfix-Version `1.0.1`.

```
# Erstelle einen neuen Branch namens hotfix-1.0.1 vom main-Branch
git checkout -b hotfix-1.0.1 main
./bump-version.sh 1.0.1
git commit -a -m "chore(release): bump version number to 1.0.1"
```

Nachdem wir den Bug behoben haben, mergen wir den `hotfix-1.0.1`-Branch zurück in den `main`-Branch.

```
git checkout main
git merge --no-ff hotfix-1.0.1
# Tagge die Veröffentlichung
git tag -a 1.0.1
# Merge auch den develop-Branch, um ihn auf dem neuesten Stand mit dem main zu halten
git checkout develop
git merge --no-ff hotfix-1.0.1
git branch -d hotfix-1.0.1
```

> **Tip**
>
> Der letzte Befehl ist optional, aber empfohlen. Er löscht den `hotfix-1.0.1`-Branch, nachdem er in den `main`-Branch gemerged wurde. Keine Sorge, die Änderungen sind immer noch im `main`-Branch. Wir möchten nur unsere Branches sauber halten.

2.1.5 References

- GitHub Git Guide
- GitHub Authentication Documentation
- Gitflow Medium
- Gitflow Nvie

3. Über den Kickstart-Guide

Der Kickstart-Guide ist eine direkte, praktische Ressource für Entwickler auf allen Stufen ihrer Laufbahn. Er ist darauf ausgelegt, schnellen Zugriff auf Informationen zu bieten. Dieser Guide dient als sofort verfügbare Referenz, um Ihre Arbeit, Ihr Lernen und Ihre Erkundungen im Bereich der Entwicklung zu unterstützen.

Für wen ist er gedacht?

- **Neueinsteiger:** Einführung in grundlegende Konzepte und Tools für diejenigen, die neu in der Entwicklung sind.
- **Erfahrene Entwickler:** Bietet schnelle Auffrischungen und fortgeschrittene Einblicke.
- **Teams:** Dient als gemeinsame Wissensbasis, um Konsistenz in den Praktiken zu gewährleisten.

3.1 Nutzung des Guides

Tauchen Sie in jeden Abschnitt ein, der Ihren aktuellen Bedürfnissen entspricht. Der Guide ist so konzipiert, dass Sie schnell finden, wonach Sie suchen, es verstehen und auf Ihre Projekte oder Ihren Lernpfad anwenden können.

3.2 Lizenz

Geteilt unter der Apache-Lizenz 2.0, fördert der Kickstart-Guide offene Nutzung und Beitrag. Sie können die vollständige Lizenz im LICENSE finden.

https://infinitepain.github.io/Kickstart-Guide/dev