Git für Maker

Git ist nicht nur ein Tool für Software-Entwickler – es ist auch ein praktisches Werkzeug für Maker. Ob bei Software- oder Hardware-Projekten: Git organisiert Änderungen, speichert Versionen und erleichtert die Zusammenarbeit in einer Gruppe. Perfekt für Maker, die ihren Code übersichtlich verwalten möchten.

von Daniel Schwabe

Fleißige Make-Leser finden in Kurzinfos zu verschiedenen Artikeln regelmäßig Links zum Projektcode auf GitHub. GitHub ist eine von vielen Plattformen, die einen Service für Entwickler bereitstellen, um Code online speichern, teilen, strukturieren und verwalten zu können. Dabei wird auf die Standardsoftware Git zurückgegriffen, die jeder dieser Plattformen das Grundgerüst für ihre Services stellt. Doch was genau ist Git eigentlich und was kann es? Der folgende Artikel erläutert grundlegende Git-Befehle für Git-Einsteiger. Mit diesen Befehlen lassen sich lokale Projekte versionieren, unterschiedliche Code-Versionen anlegen und bei Bedarf bestimmte Änderungen im Projektverlauf rückgängig machen.

Was ist Git?

Git ist ein Versionskontrollsystem für Code. Dabei ist eine mit Git gespeicherte Version speziell: Es speichert nämlich nur Änderungen und sichert diese über den gesamten Projektverlauf. Das bedeutet zum einen, dass man keine kompletten Kopien desselben Projektes hat, die immer nur einen ganz speziellen Stand des Codes abbilden. Zum anderen kann man einfach durch die verschiedenen Stadien des Projektes „durchblättern", um ältere Versionen wiederherzustellen.

Dieses System funktioniert für unformatierte Arten von Textdateien. Für Maker beispielsweise mit einer .ino-Datei aus der Arduino IDE, einer .md-Markdown-Datei oder einer simplen .txt. Überwachen und sichern kann man aber alle Dateien – auch PNGs, PDFs oder 3D-Dateien, an denen man kontinuierlich arbeitet. Bei solchen komplexeren Formaten werden aber nicht die Änderungen gesichert, sondern immer die gesamte Datei.

Ein Beispiel: Man sitzt seit mehreren Tagen an der Implementierung einer bestimmten Funktion im Code. Man merkt aber, dass man in die falsche Richtung gearbeitet hat. Durch Git kann man mit nur einem Befehl die letzte funktionierende Version des Codes wiederherstellen. Man hat also permanent alle Stadien des Codes zur Verfügung.

In diesem Artikel wird auf die Verwendung von Git in der Kommandozeile eingegangen. Ein Onlineartikel behandelt die Nutzung grafischer Benutzeroberflächen. Der Link dazu befindet sich in der Kurzinfo.

Git unter Windows installieren

Git lässt sich unter Windows ganz einfach installieren, indem man die entsprechende .exe-Datei von der offiziellen Git-Website herunterlädt und mit Doppelklick ausführt. Während der Installation muss man im Installationsassistenten Entscheidungen treffen. Bei der Frage, welchen Text-Editor Git verwenden soll (Abbildung 1), kann man aus einer Liste nach eigener Präferenz entscheiden. Den voreingestellten VIM-Editor sollte man nicht auswählen, weil die Bedienung dieses Programms sehr umständlich ist. Bei den anderen Fragen ist standardmäßig jeweils die empfohlene Option vorausgewählt, für erste Schritte mit Git kann man diese Einstellungen so belassen. Nachdem der Installationsassistent durchgelaufen ist, kann man unter Windows in den Programmen Terminal, CMD und PowerShell mit dem Befehl `git` die Git-Funktionen nutzen.

Um zu überprüfen, ob alles geklappt hat, kann man im Programm Terminal den `git`-Befehl ohne weitere Attribute ausführen und bekommt eine Übersicht über alle Funktionen, die man nutzen kann.

Hat alles geklappt, legt man noch mit den beiden Befehlen `git config --global user.email "<e-Mail>"` und `git config --global user.name "<Name>"` eine Git-Identität fest. Diese Daten werden später mit Git-Datensätzen verknüpft.

Ein Git-Repository einrichten und verwalten

Git verwaltet nicht automatisch alle Projekte, Ordner und Dateien auf dem Computer. Um alle Funktionen von Git nutzen zu können, muss man als Erstes ein lokales „Repository" anlegen. Dabei handelt es sich um einen Ordner, in dem alle Dateien eines Projektes abgelegt werden und der für Git als zu überwachen markiert ist.

Kurzinfo

- Git für Windows installieren
- Code versionieren
- Unterschiedliche Projektversionen anlegen

Mehr zum Thema

- Florian Schäffer, AVR-Programme debuggen, Teil 1, Make 4/24, S. 104
- Daniel Schwabe, Private Maker-Tools auf dem Pi, Make 2/24, S. 110
- Daniel Schwabe, Eigene Serverdienste mit einem Klick, Make 3/24, S. 98

Alles zum Artikel im Web unter *make-magazin.de/xmh3*

Um ein Repository anzulegen, wird zunächst ein Zielordner benötigt. Ist er angelegt, navigiert man im Explorer in diesen Ordner und macht einen Rechtsklick. Im Kontextmenü dann „Weitere Optionen anzeigen/In Terminal öffnen" anklicken. Es öffnet sich ein schwarzes Eingabefenster. Hier kann man jetzt mit dem Befehl `git init` ein Repository anlegen und den Ordner mit Git überwachen. Ob es funktioniert hat, kann mit dem Befehl `git status` überprüft werden. Als Ausgabe muss „On branch main No commits yet nothing to com-

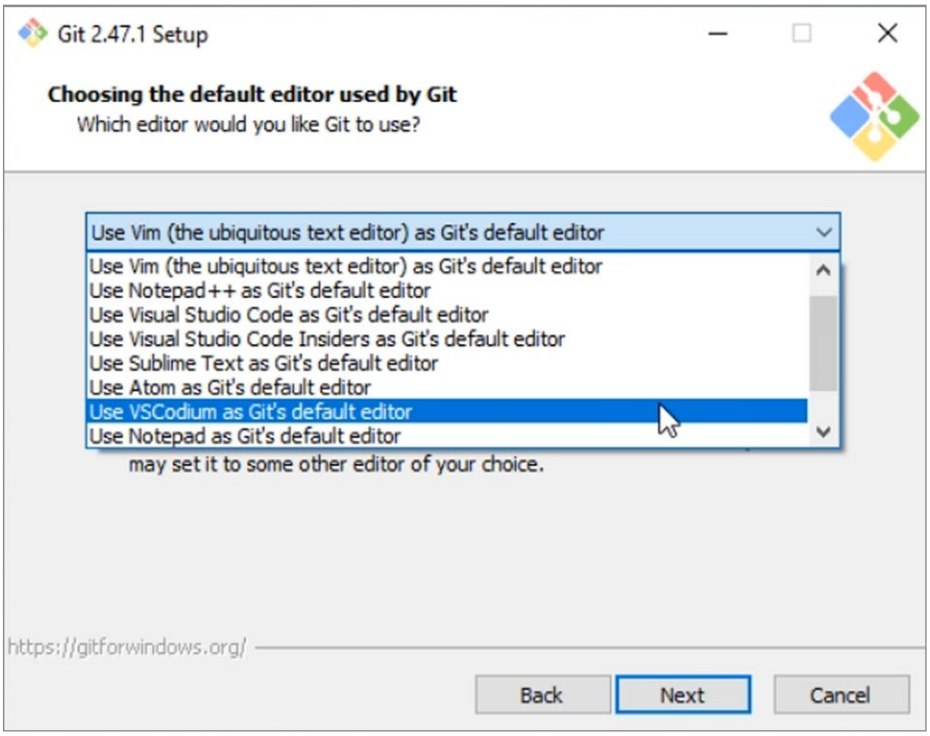

Abbildung 1: Bei der Installation von Git kann man verschiedene Texteditoren für die Git-Nutzung einrichten.

```
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Gespeicherte-TXT-Die-veraendert-wurde.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Neue-Datei.txt
```

Abbildung 2: Veränderte und nicht versionierte Dateien werden mit `git status` in Rot angezeigt.

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Gespeicherte-TXT-Die-veraendert-wurde.txt
        new file:   Neue-Datei.txt
```

Abbildung 3: Dateien und Änderungen in Grün werden zu einem neuen Rücksetzpunkt hinzugefügt.

mit (create/copy files and use "git add" to track)" in drei Zeilen erscheinen.

Legt man jetzt neue Ordner oder Dateien an bzw. verändert man deren Inhalte, gibt der Befel `git status` den aktuellen Stand des Repositorys an (Abbildung 2).

Will man einen aktuellen Ordner-Status in Gits Versionsverwaltung als Rücksetzpunkt (in der Fachsprache Commit genannt) speichern, muss man als Erstes festlegen, welche veränderten Dateien und Ordner zu diesem Rücksetzpunkt hinzugefügt werden sollen. Das legt man mit dem Befehl `git add <Datei-/Ordnername>` fest. Wenn man mit einem Befehl mehrere Dateien hinzufügen möchte, kann man diesen Befehl auch mit einer Liste von Dateinamen ausführen, z.B. `Git add ./Datei-1.txt ./Datei-2.ino`. Möchte man alle veränderten und neuen Dateien hinzufügen, reicht auch ein `git add *`. Führt man jetzt den Befehl `git status` aus, werden alle zum Sichern vorgemerkten Dateien in grünem Text angezeigt (Abbildung 3).

Noch sind diese Änderungen aber nicht in die Versionshistorie des Repositorys geschrieben worden. Dafür muss man noch den Befehl `git commit` ausführen. Bestätigt man den Befehl mit der Entertaste, öffnet sich der Texteditor, den man während der Installation ausgewählt hat, und verlangt nach einer Commit-Message. Eine Commit-Message beschreibt die Änderungen, die seit dem letzten Speicherpunkt gemacht wurden, z.B. „Funktion für blinkende LEDs eingefügt". Diese Nachricht gibt man im geöffneten Editor in der obersten Zeile ein, speichert und schließt den Editor. Dann wird der `git`-Befehl fertig ausgeführt. Benutzt man beim Befehl `git commit` das Zusatzargument `-m`, kann man die Commit-Message direkt im Terminal zusammen mit dem Commit-Befehl eingeben: `git commit -m „Funktion für blinkende LEDs eingefügt"`.

Mit dem Befehl `git log` wird eine Übersicht aller Rücksetzpunkte mit den Commit-Messages angezeigt (Abbildung 4).

```
commit d69c05d181275f35f54766b129560f7725fa4d62 (HEAD -> main)
Author: [redacted]
Date:   Fri Jan 10 17:55:34 2025 +0100

    Mein neuer Commit

commit feec11fb77385da05dd0bb3185320195d83c0411
Author: [redacted]
Date:   Fri Jan 10 16:19:51 2025 +0100

    Update NeueTXT.txt

commit 738a9e29563b25b1b29e7aab1f5ab104842c9ed4
Author: [redacted]
Date:   Fri Jan 10 16:19:31 2025 +0100

    Update NeueTXT.txt

commit cf6c25fab6286a7f110df6af332c1b34b1cbc81b
Author: [redacted]
Date:   Fri Jan 10 16:14:22 2025 +0100

    Update Unbenannt.png

commit 05d2f316d1f8598ff37c70de72be3dfc5dd72e32
```

Abbildung 4: Ausgabe des Befehls `git log`, in Gelb: die Commit-ID

Auf alten Projektstand zurücksetzen

Wenn jetzt das Szenario aus der Einleitung eintritt und man die Veränderungen seit dem letzten Commit verwerfen möchte, geschieht das mit dem Befehl `git reset --hard`. Dieser Befehl löscht alle Änderungen in Dateien, die schon einmal mit `git commit` gespeichert wurden.

Will man auch neu angelegte Dateien löschen, nutzt man den Befehl `git clean -f`. Diese Befehle sind aber sehr gefährlich, denn sie können nicht rückgängig gemacht werden.

Hat man Änderungen mit `git commit` bereits zu einem Rücksetzpunkt gespeichert, möchte aber auf einen älteren Punkt zurückspringen, braucht man die Commit-ID dieses Rücksetzpunktes. Die bekommt man mit dem Befehl `git log`. Es ist eine lange Zeichenkolonne. Hat man die richtige ID, kann man den Befehl `git reset --hard <Commit-ID>` ausführen und zu dem angegebenen Commit zurückspringen.

Was zur Hölle ist ein Branch?

Wenn man die bisher gezeigten Befehle nutzt, um ein Projekt zu sichern und zu versionieren, arbeitet man immer im sogenannten „main Branch". Neben der bisherigen linearen Versionsverwaltung bietet Git jedoch ein hilfreiches Feature: das Branching. Ein Branch (engl. Zweig) ist wie eine Abzweigung innerhalb eines Projekts. Dabei wird der aktuelle Stand des Codes an einer bestimmten Stelle kopiert, um unabhängig und parallel weiterentwickelt zu werden.

Wenn man einen neuen Branch erstellt, erzeugt man also eine zweite Version des Projekts. Es gibt dann die Hauptversion (den „main Branch") und die abgezweigte Version. Diese zweite Version kann beliebig benannt werden. Zum Beispiel kann man in einem Projekt, das ein LC-Display verwendet, einen„OLED-Branch" anlegen, um den passenden Code einzubauen.

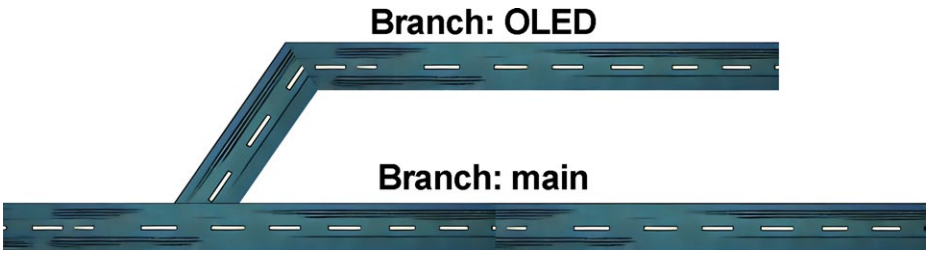

Abbildung 5: Branches laufen ab einem gewissen Projektstatus parallel.

Der Hauptstrang des Projekts arbeitet weiter mit einem LC-Display (Abbildung 5).

Wenn man später die Änderungen aus einem Branch auch in der Hauptversion haben möchte, können die Branches wieder zusammengeführt werden. Das nennt man „mergen".

Mit dem Befehl `git branch <Name-für-den-Branch>` erstellt man einen neuen Branch. Um dann in diesen Branch zu wechseln, nutzt man `git checkout <Branch-Name>`. Dadurch wird der Inhalt des Projektordners so geändert, dass er den Stand des ausgewählten Branches zeigt.

Der Befehl `git status` zeigt an, in welchem Branch man gerade arbeitet. Jeder Commit, den man macht, wird nur in diesem Branch gespeichert. Die anderen Branches bleiben unverändert.

Das Verbinden von zwei Branches funktioniert mit dem Befehl `git merge`. Um z.B. den OLED-Branch aus dem obigen Beispiel in den main Branch zu integrieren, wechselt man zunächst mit `git checkout main` in den Haupt-Branch und fügt dann mit `git merge OLED-Branch` die Änderungen aus der Abzweigung ein.

Git revert

Neben den einfachen Befehlen, um Commits zu löschen und das Repository auf einen bestimmten Stand zurückzusetzen, gibt es noch den git revert Befehl.

git revert <Commit-ID> entfernt die Inhalte eines bestimmten Commits aus dem aktuellen Dateistatus und erstellt dann einen neuen Commit.

Wenn man z.B. drei Dateien angelegt hat und für jede Datei einen eigenen Commit erstellt hat, kann man mit git revert den Commit, in dem die zweite Datei dem Repository hinzugefügt wurde, rückgängig machen. Im aktuellen Dateistatus sind dann nur noch Datei eins und drei. Der rückgängig gemachte Commit bleibt aber in der Historie erhalten.

Das Backup der Versionsverwaltung

Wenn man Git für die Versionsverwaltung eines Projekts nutzt, erstellt man durch seine Commits automatisch Rücksetzpunkte mit verschiedenen Codeversionen. Sie sind jederzeit wiederherstellbar. Aber Git allein ist noch kein vollständiges Backup. Verliert man den Projektordner durch einen Festplatten-Crash oder Ähnliches, kann Git nicht mehr helfen. Daher sollte man immer Kopien von Projekten sichern (an die 3-2-1-Regel denken).

Das Gute ist: Wenn man den Ordner eines Git-Projekts kopiert, sichert man nicht nur die Dateien, sondern auch die gesamte Historie aller Commits. Git speichert nämlich alles in einem versteckten Ordner namens .git, der sich im Projektordner befindet. Kopiert man das Projekt also auf einen anderen Computer, wird dieser Ordner mit übertragen. Dadurch kann man genauso wie zuvor auf alle alten Commits zugreifen, so als wäre es der Originalordner.

Was ist ein GitHub?

Dieser Artikel beschreibt, wie man Git als lokale Versionsverwaltung von Projekten benutzt. Aber was ist denn nun eigentlich Github? Github ist einer von vielen Services, die einem die Möglichkeit bieten, über Git verwalteten Code online zur Verfügung zu stellen. Über diese Plattformen können Entwickler, die Zugriff auf ein Projekt haben, ihre Commits online hochladen etc. Neben GitHub gibt es noch viele weitere Anbieter für einen solchen Service. Wie man Git online nutzt, behandelt ein Artikel in der kommenden Make-Ausgabe.

Mehr Infos online

Online gibt es weitere Begleitartikel zu Git. In ihnen lernt man z. B. einen Spickzettel mit wichtigen Git-Befehlen kennen und es werden Programme vorgestellt, welche die Git-Verwaltung aus der Kommandozeile herausholen und eine grafische Oberfläche für die verschiedenen Befehle bieten. Die Links dazu befinden sich in der Kurzinfo. *—das*