

Skript zur Vorlesung Programmierung II : HS2021

Einstieg in die Geo-Programmierung mit Python Version 1.9 HS2021



Prof. Martin Christen
martin.christen@fhnw.ch
Hochschule für Architektur, Bau und Geomatik
Institut Geomatik

Einleitung

Die Vorlesung Programmierung II besteht aus drei Teilen, welche bestehendes Wissen über Python Programmierung vertiefen soll. Der erste Teil ist die Objektorientierung. Der zweite Teil die GUI-Programmierung und der dritte Teil ist ein Einstieg in die Programmierung mit Geo-Bibliotheken.

Die Kapitel entsprechen normalerweise den Vorlesungstagen.

Das Skript basiert unter anderem auf folgender Literatur

- Theis T. (2014) Einstieg in Python, 4. Auflage, Galileo Computing, ISBN 978-3-8362-2861-9
- Ernesti J, Kaiser P. (2012) Python 3 - Das Umfassende Handbuch, Galileo Computing, ISBN 978-3-8362-1925-9
- Downey B.D. (2012) Programmieren lernen mit Python, O'Reilly Verlag, ISBN 978-3-86899-946-4
- Bouda P. (2012) PyQt und PySide, GUI-Anwendungsentwicklung mit Python und Qt, Open Source Press München, ISBN 978-3941841505
- Klein B. (2013) Einführung in Python 3 - In einer Woche Programmieren lernen, Hanser Verlag, ISBN 978-3-446-43547-6
- Woyand H.-B. (2012) Python - Einführung in die Programmierung und mathematische Anwendungen, J. Schlembach Fachverlag 2012, ISBN 978-3-935340-73-1
- Python Referenz 2021 (<http://www.python.org/3>)

1 Grundlagen der Versionsverwaltung mit git

Die gemeinsame Entwicklung von Software-Projekten findet nicht nur innerhalb von Firmen statt: Auch im Open-Source-Sektor sind – je nach Projektgrösse – mehrere hunderte bis tausende Freiwillige und Ehrenamtliche an der **Instandhaltung, Optimierung, Weiterentwicklung** oder **Modifizierung** eines Programms beteiligt. Ohne ein passendes System zur Aufzeichnung und Kontrolle der zahlreichen Änderungen durch die verschiedenen Entwickler wären derartige Projekte kaum zu realisieren.

Eine der beliebtesten Lösungen zur Versionsverwaltung ist das lizenzfreie Tool Git, das einfach zu lernen und gänzlich kostenfrei nutzbar ist. In diesem Tutorial vermitteln wir Ihnen alle wichtigen Git-Grundlagen, die Sie für den Einstieg in das Versionsverwaltungsprogramm benötigen.

1.1 Was ist Git?

Git ist ein verteiltes Versionsverwaltungssystem, das 2005 von Linux-Schöpfer Linus Torvalds entwickelt und unter der freien GNU-GPLv2-Lizenz veröffentlicht wurde. Die Besonderheit des Tools besteht darin, dass zwar ein **zentrales Repository** für jedes Projekt existiert, alle beteiligten Nutzer sich aber eine **lokale Arbeitskopie** dieses Verzeichnisses auf das eigene Arbeitsgerät herunterladen. Jede dieser Kopien stellt ein vollständiges Back-up des Repositories dar, weshalb eine durchgehende Netzwerkverbindung für den grundlegenden Arbeitsprozess nicht erforderlich ist. Zudem dienen die Kopien als Absicherung, falls das Haupt-Verzeichnis ausfallen sollte oder beschädigt ist.

Vorgenommene **Änderungen** können jederzeit **mit allen anderen Projektteilnehmern ausgetauscht** und – sofern relevant – in das Repository aufgenommen werden.

1.2 Grundbegriffe

Quelle: <http://gitbu.ch/>

1.2.1 Versionskontrollsystem (Version Control System, VCS)

Ein System zur Verwaltung und Versionierung von Software oder anderer digitaler Information

1.2.2 Repository

Das Repository / Repo ist eine Datenbank, in der Git die verschiedenen Zustände jeder Datei eines Projekts über die Zeit hinweg ablegt. Insbesondere wird jede Änderung als «Commit» verpackt und abgespeichert.

1.2.3 Working Tree

Das Arbeitsverzeichnis von Git. Hier werden alle Modifikationen am Quellcode vorgenommen,

1.2.4 Commit

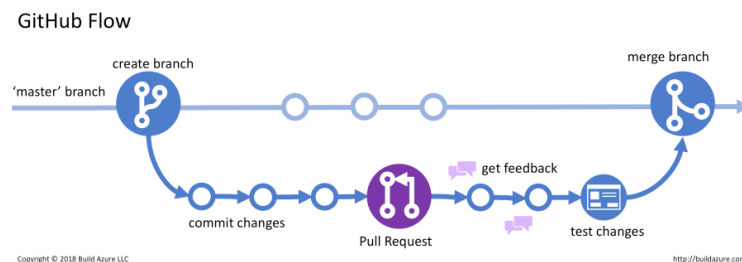
Ein Commit sind die Veränderungen am Working Tree, also z.B. modifizierte oder neue Dateien. Ein Commit referenziert immer den Zustand aller verwalteten Dateien zu einem bestimmten Zeitpunkt. Die verschiedenen Git-Kommandos dienen dazu, Commits zu erstellen, zu manipulieren, einzusehen oder die Beziehungen

1.2.5 HEAD

Head ist eine Symbolische Referenz auf den neusten Commit im aktuellen Branch. Von dieser Referenz hängt ab, welche Dateien Sie im Working Tree zur Bearbeitung vorfinden.

1.2.6 Branch

Eine Abzweigung in der Entwicklung. Branches werden in der Praxis verwendet, um beispielsweise neue Features zu entwickeln, Releases vorzubereiten oder um Alte Versionen mit Bugfixes zu versorgen. Branches können mittels «merge» zusammengeführt werden.



1.2.7 Clone

Wird ein Git-Repository aus dem Internet heruntergeladen, so erzeugen Sie automatisch einen Klon (Clone) dieses Repositories. Der Klon enthält alle Informationen, die im Ursprungsrepository enthalten sind, auch die gesamte Versionsgeschichte.

1.2.8 Tag

Tags sind symbolische Namen für schwer zu merkende SHA Prüfsummen. Wichtige Commits, wie z.B. Releases können mit «Tags» gekennzeichnet werden. Ein Tag kann zum Beispiel ein Bezeichner wie «v1.2» sein oder sonst irgendwelche Beschreibungen enthalten.

1.2.9 SHA

Der Secure Hash Algorithm (SHA) erstellt eine Prüfsumme für beliebige digitale Informationen. Alle Commits in git werden nach ihrer SHA Prüfsumme benannt (Commit-ID)

1.3 Git Installieren

1.3.1 Installation unter Windows

Git kann für Windows heruntergeladen werden:

<https://git-scm.com/>

Auch nützlich ist GitHub Desktop Client für git. Dieser kann für Windows und Mac hier heruntergeladen werden:

<https://desktop.github.com/>

1.3.2 Installation unter MacOS X

Für MacOS gibt man im Terminal

```
git --version
```

ein. Falls es noch nicht installiert ist, erscheint eine Abfrage wie man es installieren will.

Die Installation erfolgt mit allen Standard-Einstellungen.

1.4 Git konfigurieren

Nach der Installation müssen wir einmalig eine Konfiguration Durchführen. Dazu öffnen wir die «Git Bash». Es gibt einige Optionen, aber die wichtigsten sind Benutzername und E-Mail.

```
git config --global user.name "Vorname Nachname"
git config --global user.email name@email.com
```

1.5 Einen SSH-Key erstellen und auf GitHub stellen

In der Git Bash (oder Konsole) mit ssh-keygen einen neuen Schlüssel erzeugen:

```
ssh-keygen
```

Es wird ein Pfad und Dateiname für den Schlüssel vorgeschlagen, wir belassen das.

Eine Passphrase wird bei Versionssverwaltung in der Regel **nicht** verwendet, wir wollen mit dem Schlüssel eine sichere Verbindung um genau das Passwort nicht immer einzutippen müssen.

Im .ssh Directory befinden sich nun der Private und der öffentliche Schlüssel. Den öffentlichen Schlüssel kopieren wir auf GitHub. Dazu gehen wir auf «Settings» (Klick auf

das Profilbild oben rechts). Wir wählen das Menu auf der linken Seite «SSH and GPG keys) und fügen den key mit «New ssh key» hinzu.

1.6 Verwenden von git

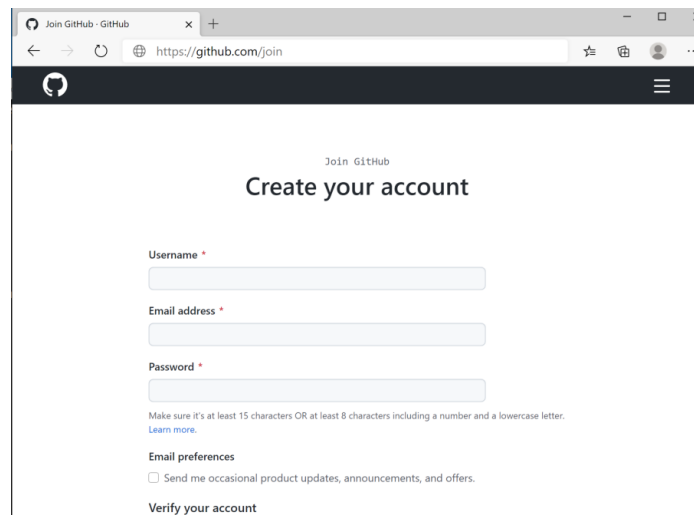
1.6.1 GitHub

GitHub ist eine Onlineplattform, welche Speicher zur Verfügung stellt und ein Host für Git-Projekte ist. GitHub ist kostenlos und ein Tochterunternehmen von Microsoft. Es gibt öffentliche und Private Projekte. Somit ist es möglich, dass jede Person (weltweit) an öffentlichen Projekten mitarbeitet. GitHub ist heute ein wesentlicher Player in der OpenSource Entwicklung.

Die Unterschiede zwischen git und GitHub sind:

- Git ist das System, welches eine nicht-lineare Entwicklung ermöglicht und das Protokoll für den Datentransfer etc. zur Verfügung stellt.
- GitHub ist ein Online-Dienst welcher auf git-Basis arbeitet
- Git wird lokal installiert, während GitHub eine online-Lösung ist.
- GitHub bietet eine Grafische Oberfläche, während git nur über die Kommandozeile aufgerufen wird
- Git ist Open Source und kostenlos, GitHub verlangt für gewisse Erweiterungen eine monatliche Gebühr.

Wir legen einen GitHub Account an: <https://github.com/join>



Screenshot: GitHub Account anlegen

1.6.2 Git-Repository anlegen/klonen

Das Git-Repository ist das zentrale Verzeichnis eines verwalteten Projekts und somit auch die zentrale Anlaufstelle für alle Teilnehmer, über die die komplette Versionskontrolle reguliert wird. Ihr erster Schritt in Git besteht dementsprechend darin, ein solches Hauptverzeichnis zu erstellen – oder zu klonen (in Form einer Arbeitskopie),

sofern Sie sich einem Projekt anschliessen, dessen gemeinsame Entwicklung bereits mithilfe von Git kontrolliert wird.

Wollen Sie die Versionskontrolle neu einrichten oder haben Sie das Tool zunächst einfach nur installiert, um die Arbeitsweise mit Git zu lernen, steht zunächst die Neuerstellung eines Repositorys an. Zu diesem Zweck wechseln Sie per „cd“ (change directory) in das gewünschte lokale Verzeichnis auf Ihrem Gerät:

```
mkdir programmierung2  
cd programmierung2  
  
git init
```

1.6.3 Repository-Status überprüfen und neue Dateien zur Versionsverwaltung hinzufügen

Zu den wichtigsten Git-Grundlagen zählt eine gute Organisation des eigenen Arbeitsverzeichnisses. Über dieses schlagen Sie nicht nur **eigene Änderungen und Neuerungen** an einem Projekt vor, die anschliessend per Commit („Freischaltung“) übernommen werden, sondern beziehen auch **Informationen über die Aktivitäten anderer Nutzer**. Die Aktualität Ihrer Arbeitskopie können Sie über folgende Eingabe überprüfen:

```
git status
```

Bei einem gerade neu angelegten Repository bzw. einer absoluten Übereinstimmung von Hauptverzeichnis und Arbeitskopie erhalten Sie in der Regel die Information, dass es keinerlei neue Anpassungen an dem Projekt gab („*No commits yet*“). Zudem teilt Git mit, dass Sie aktuell keine eigenen Änderungen für den nächsten Commit geteilt haben („*nothing to commit*“).

Um eine eigene, neue **Datei zur Versionsverwaltung hinzuzufügen** oder eine überarbeitete Datei **für den nächsten Commit anzumelden**, wenden Sie den Befehl „**git add**“ auf diese Datei an (diese muss sich hierfür im Arbeitsverzeichnis befinden). In unserem Git-Tutorial fügen wir exemplarisch ein Textdokument mit dem Namen „Test“ hinzu. Dies müssen wir natürlich z.B. im Explorer zunächst erstellen.

```
git add Test.txt
```

Überprüft man nun anschliessend den Repository-Status erneut, wird das Beispiel-Dokument als potenzieller Anwärter **für den nächsten offiziellen Änderungsschritt** des Projekts („*Changes to be committed*“) präsentiert.

1.6.4 Änderungen via Commit bestätigen und in den HEAD aufnehmen

Sämtliche Änderungen, die Sie (wie im vorherigen Abschnitt beschrieben) für die Versionsverwaltung angemeldet haben, müssen immer per Commit bestätigt werden, damit sie **in den HEAD aufgenommen** werden. Beim HEAD handelt es sich um eine Art Index, der auf den letzten wirksam gewordenen Commit in Ihrer aktuellen Git-Arbeitsumgebung (auch als „Branch“ bezeichnet) verweist. Das Kommando für diesen Schritt lautet:

```
git commit -m "Beschreibung der Änderung"
```


Sie erhalten nach der Ausführung von „git commit“ eine zusammenfassende Meldung zu dem Commit: In den voranstehenden eckigen Klammern steht zum einen der **Name der Branch** (Projektzweig; hier „master“, da unser Arbeitsverzeichnis gleichzeitig auch das Hauptverzeichnis ist), in den die Änderungen übertragen wurden, zum anderen die **SHA-1-Prüfsumme** des Commits (z.B. „c0fdf55“). Es folgen der frei gewählte Kommentar (hier „Test“) und konkrete Informationen über die vorgenommenen Anpassungen.

1.6.5 Commit-Historie anzeigen lassen

Das Projektmanagement mit Git zu lernen, lohnt sich insbesondere aufgrund der elementaren Versionierungs-Features. Ein grosses Plus des Open-Source-Systems besteht darin, dass Sie sich jederzeit anzeigen lassen können, welche **Änderungen zuletzt am Repository vorgenommen** worden sind.

Der hierfür benötigte Git-Befehl lautet:

```
git log
```

Der Befehl „git log“ listet die erzeugten Commits in umgekehrter, chronologischer Reihenfolge auf, wobei standardmässig die **SHA-1-Prüfsumme**, der **Autor** (Name und Mailadresse) sowie das **Datum des jeweiligen Commits** angezeigt werden. Zudem ist natürlich auch **die individuelle Meldung** zu sehen, die Ihnen bzw. anderen Nutzern als entscheidender Hinweis dient, die einzelnen Änderungen schnell einordnen zu können. In unserem Git-Tutorial haben wir zuvor einen einzelnen Commit mit der Nachricht „Test“ generiert, den uns das Kommando bei der Ausführung auch wie gewünscht anzeigt.

Das log-Kommando lässt sich ausserdem mithilfe verschiedener Parameter modifizieren. Einige nützliche Optionen sind in der nachfolgenden Tabelle aufgeführt:

Option für den Befehl „git log“	Beschreibung
-p	zeigt zusätzlich auch die Änderungen an, die in einem Commit enthalten sind
-2	listet nur die letzten beiden ausgeführten Commits auf
--stat	fügt jedem Eintrag eine kleine Statistik hinzu, die zeigt, welche Dateien geändert und wie viele Zeilen hinzugefügt oder entfernt wurden
--pretty	ändert das Format der Ausgabe, wobei verschiedene Formate zur Verfügung stehen; --pretty=oneline ist z. B. ein mögliches Format, das alle Commits in einer einzelnen Zeile auflistet
--abbrev-commit	zeigt nur die ersten Zeichen einer SHA-1-Prüfsumme an
--relative-date	zeigt das Datum einer Änderung in relativem Format an (z. B. „vor zwei Wochen“)

1.6.6 Tagging: Tags in Git erstellen, löschen und auflisten

Wie viele andere Versionskontrollsysteme verfügt auch Git über eine Tagging-Funktion, mit der sich **ausgewählte Punkte in der Historie eines Repositories als wichtig markieren** lassen. Typischerweise werden solche Tags dazu verwendet, Releases einer Software wie Version 1.0, 2.0 usw. zu kennzeichnen, damit diese auch bei grossen Projekten jederzeit leicht abrufbar bleiben. Git unterstützt dabei zwei Arten von Tags:

- **Annotierte Tags** („annotated“) werden als eigenständige Objekte in der Datenbank gespeichert, inklusive eigener Prüfsumme, Tagging-Meldung, Datum, Name und Mail-Adresse des Tag-Erstellers sowie optionaler GNU-Privacy-Guard-Signatur (GPG-Signatur).

- **Nicht-annotierte Tags** („*lightweight*“) fungieren – ähnlich wie Branches – ausschliesslich als Verweis auf einen Commit. Dieser Typ bietet sich dann an, wenn Sie lediglich temporäre Tags benötigen oder die erweiterten Informationen nicht speichern wollen.

Annotierte Tags erstellen Sie in Git, indem Sie den Befehl „**git tag -a**“ auf den gewünschten Commit anwenden. Hängen Sie ausserdem den Parameter „**-m**“ an, können Sie – in gerade Anführungszeichen gesetzt – auch direkt in der Kommandozeile die gewünschte **Tagging-Meldung** formulieren. In diesem Git-Tutorial haben wir den Commit „Test“ generiert, den wir zu diesem Zweck auch mit einem Tag inklusive der Meldung „example tag“ verknüpfen:

```
git tag -a Test -m "example tag"
```

Für **nicht-annotierte Tags** gehen Sie ähnlich vor: Allerdings wenden Sie ausschließlich den Grundbefehl „**git tag**“ auf den gewünschten Commit an und verzichten auf weitere Parameter. Für unser Git-Tutorial-Beispiel würde dies folgendes Kommando bedeuten:

```
git tag Test
```

Sobald Tags für Ihr Repository existieren, können Sie sich diese mit dem bereits bekannten „**git tag**“ und den optionalen Parametern „**-l**“ bzw. „**--list**“ anzeigen lassen:

```
git tag
git tag -l
git tag --list
```

Um einen **Tag aus dem lokalen Arbeitsverzeichnis zu löschen**, wenden Sie die Befehlskette „**git tag -d**“ auf diesen an. Unseren Verweis auf „Test“ entfernt man also folgendermassen:

```
git tag -d Test
```

1.6.7 Branches erstellen, verwalten und löschen

Die in diesem Git-Tutorial bereits erwähnten Branches stellen im Prinzip nichts Anderes als einzelne **Arbeitsversionen des Haupt-Respositories** dar, das selbst als Branch – mit dem Namen „master“ – eingestuft wird. Mit diesen Arbeitszweigen bietet Git die perfekte Grundlage dafür, Features und Funktionen isoliert voneinander zu entwickeln und erst zu einem späteren Zeitpunkt zusammenzuführen. Letzteres bezeichnet man auch als „**mergen**“ (von engl. merge „verschmelzen“).

Einen **neuen Branch zu erstellen**, ist nicht schwer: Sie benötigen lediglich die Anweisung „**git branch**“ und hängen dieser die gewünschte Bezeichnung für den Zweig an. Einen Beispiel-Branch mit dem Namen „**test_branch**“ erstellen Sie beispielsweise auf folgende Weise:

```
git branch test_branch
```

Anschliessend können Sie jederzeit mit der Anweisung „**git checkout**“ in diesen Branch wechseln:

```
git checkout test_branch
```

Wollen Sie Branches zusammenführen, gelingt dies mit dem Kommando „**git merge**“. Wechseln Sie zunächst via „**checkout**“ in das Verzeichnis, das einen anderen Zweig aufnehmen soll und führen Sie dort dann den genannten Befehl inklusive des Namens des aufzunehmenden Branchs aus.

Unsere Arbeitsversion „test_branch“ lässt sich beispielsweise folgendermassen **in das Haupt-Repository mergen**:

```
git checkout master
git merge test_branch
```

Haben Sie Arbeitszweige zusammengeführt und benötigen daher einen **bestimmten Branch** nicht mehr, können Sie diesen ganz einfach löschen. Zu diesem Zweck wenden Sie die Befehlsfolge „**git branch -d**“ auf den nicht mehr benötigten Versionszweig an. Unser Git-Tutorial-Beispiel „test_branch“ lässt sich durch folgende Eingabe entfernen:

```
git branch -d test_branch
```

Die einzige **Voraussetzung für den Löschprozess**: Sie müssen sich in einem anderen Branch befinden. Wir sind daher vor der Ausführung des Befehls in das Haupt-Repository gewechselt.

1.7 Git Repository auf GitHub erstellen

Zunächst muss auf GitHub ein neues Repo angelegt werden. Nennen wir das mal «programmierung2». Auf GitHub wäre das nun unter <https://github.com/username/programmierung2.git> verfügbar

Wir können unserem bestehenden repository nun eine neue Destination geben:

```
git remote add origin git@github.com:USERNAME/programmierung2.git
git push -u origin master
```

Beim ersten Verbinen mit «yes» bestätigen. Nun ist der gesamte Inhalt auf GitHub

1.7.1 Weitere Commits in das Haupt-Repository aufnehmen

Damit weitere Commits auch **in das Haupt-Repository** aufgenommen werden, ist die Eingabe folgenden Befehls erforderlich:

```
git push
```

Durch ihn **überträgt Git automatisch alle erstellten Commits**, die bisher nur in der Arbeitskopie vorhanden waren, **in das Hauptverzeichnis**, das auch die Bezeichnung „master“ hat. Ersetzen Sie diesen Namen in dem aufgeführten Code durch den eines anderen Branches (Projektzweigs), werden die Dateien stattdessen direkt dorthin gesendet.

1.7.2 Repository auf den neusten Stand bringen

Falls in der Zwischenzeit jemand anderes commits auf das Hauptrepository gestellt hat, kann man das lokale Repository mit «git pull» jederzeit wieder auf den neusten Stand bringen

```
git pull
```

1.7.3 Ein bestehendes Repository klonen

Ein bestehendes Repo kann jederzeit mit «git clone» geklont werden.

```
git clone git@github.com:martinchristen/programmierung2.git
```

1.8 Git in Visual Studio Code verwenden

Visual Studio Code hat git integriert und man kann direkt von dort aus git Kommandos senden. Ctrl+Shift+G