Versionsverwaltung mit Git

Urs Lautebach <urs.lautebach@zsl-rsfr.de>

Inhalt

1. Unterrichtsgang

Diese Fassung des Dokuments enthält Hinweise für Lehrkräfte.

Lehrkräfte:

Zur Entlastung können einige Dinge in einer vorbereitenden Hausaufgabe erledigt werden: Erstellung und Aktivierung des Accounts beim Git-Hoster, Anpassen der Privatsphären-Einstellungen, Erstellung des Access-Tokens, Installation und erster Start des Git-Clients.

Beim allerersten Start eines Git-Clients muss man zwei Details der Git-Konfiguration (Name und Mailadresse) angeben; das sollte vorab erledigt werden.

1.1. Allein arbeiten — mit Versionsarchiv

Hinweis für Lehrkräfte

In den ersten beiden Nutzungsszenarien werden zunächst Fachbegriffe etabliert und die Handhabung des Git-Clients ausprobiert. Sie entlasten damit die folgenden Szenarien, die zunehmend komplexere Abläufe verlangen. Die letztlich angestrebte Zusammenarbeit der Schüler wird erst ab Section 1.3, “Zusammenarbeiten an verschiedenen Dateien”] geübt.

Nach dem Ende der Git-Einführung können die zu Übungszwecken auf dem Server angelegten Repos gelöscht werden.

Git wird zunächst lediglich eingesetzt, um ältere Versionen eines Projektes rekonstruieren zu können.

a. Erstellen Sie ein neues **Arbeitsverzeichnis** und lassen Sie Git darin ein Git-**Repository** anlegen (Kommando `git init` bzw. Repository/Create).
b. Erstellen Sie im Arbeitsverzeichnis eine neue, leere Textdatei `erste-datei.txt`
c. **Commit**-ten Sie die nun neue Version (`git -a commit` bzw. Kontextmenu der Datei/Commit). Dabei muss ein Kommentar angegeben werden, der diesen Versionsstand nachvollziehbar beschreibt (hier z.B. "Datei erster-versuch.txt angelegt").
d. Ändern Sie die Datei und speichern Sie erneut.
e. Prüfen Sie, ob das Arbeitsverzeichnis un-committete Änderungen enthält, oder ob es **clean** ist (Kommando `git status` bzw. farbige Markierung im Git-Client),
f. commit-ten Sie die letzte Änderung.
g. Fügen Sie eine weitere Datei hinzu; stagen und committen Sie diese Änderung.
h. Löschen Sie eine der Dateien und committen Sie die Löschung.

i. Probieren Sie nun, ältere Stände des Projekts durch **auschecken** wiederherzustellen:
   - Vor dem Auschecken sollte das Arbeitsverzeichnis jeweils clean sein. Schauen Sie nach (ebenfalls am jüngsten Commit), wie der Branch heißt, in dem Sie momentan sind (meistens "main").
   - Doppelklicken Sie den gewünschten Commit; wählen Sie dann "just work read-only".
   - Schauen Sie die älteren Fassungen an, *ohne etwas zu ändern!*
   - Checken Sie anschließend wieder den ursprünglichen Branch aus (in SmartGit links unten in der Liste der Branches).

j. Wie könnten Sie diesen "Workflow" ohne Git nachbilden? Bewerten Sie Vor- und Nachteile.

Antwort

- Man müsste für jede Version ein Backup des Arbeitsverzeichnisses machen und mit Datum benennen. Dann hat man aber noch keine Commit-Kommentare.
- Mit Git kann man viel einfacher in alten Versionsständen nachschauen. Allerdings muss man sich in das Werkzeug auch erstmal einarbeiten.

k. Tragen Sie neue Fachbegriffe ins Glossar ein.

1.2. Allein arbeiten — mit Backup auf Server

Hinweis für Lehrkräfte

Führen Sie die Begriffe **Remote-Repo**, **clone** und **push** anhand der Folien ein und demonstrieren Sie die neuen Arbeitsschritte a) Repo auf dem Server anlegen, b) Repo vom Server auf den eigenen Rechner klonen.
Wenn Ihre Lerngruppe sehr sicher agiert, können Sie auch gleich noch **pull** vorführen und zum Arbeitsauftrag des nächsten Abschnitts übergehen.

a. Legen Sie ein Repo auf dem Server an.

b. Klonen Sie es in ein lokales Repo (`git clone` bzw. Menu Repository/Clone).

c. Erzeugen Sie dort 2-3 Commits; das Arbeitsverzeichnis sollte dann clean sein.

d. Push-en Sie die neuen Commits. Nehmen Sie weitere Änderungen vor, pushen Sie erneut.

e. Öffnen Sie im Webinterface des Servers die Übersicht des Projekts (Menu/Projects/Your projects, dann links Repository/Commits öffnen) und überzeugen Sie sich, dass alle Commits angekommen sind.

f. Was müssten Sie tun, wenn das Projekt auf Ihrem Rechner versehentlich gelöscht oder der Rechner beschädigt wird?

Antwort

Man muss einfach das Repo vom Server neu klonen.

g. Tragen Sie neue Fachbegriffe ins Glossar ein.

1.3. Zusammenarbeiten an verschiedenen Dateien

a. Ernennen Sie ein Mitglied der Gruppe zum "Sprecher".

b. **Nur Sprecher:** Nehmen Sie auf dem Server den Rest der Gruppe in ihr existierendes Projekt auf (Project information/Members/Invite members), und zwar mit *Developer*-Rechten.

c. **Jeder:** Sie finden das Projekt des Sprechers anschließend unter Menu/Projects/Your projects; klonen Sie es wie vorher.

d. **Jeder:** Fügen Sie dem Projekt eine neue Datei hinzu, die Ihren Namen trägt (damit es keine Dopplungen gibt). Ändern Sie sonst nichts. Committen Sie die Änderung, aber **warten** Sie mit dem Push.

Im nächsten Schritt kommen zunächst Fehlermeldungen, die nicht ganz selbsterklärend sind. Führen Sie daher die nächsten beiden Arbeitsschritte **gemeinsam und reihum** an den einzelnen Rechnern durch und **besprechen** Sie, was Sie sehen und tun.

e. **Die Gruppe** ermittelt mit Schere-Stein-Papier eine Gewinnerin.

f. **Die Gewinnerin** ("Alice") pusht als erste ihren neuen Commit.

g. **Die anderen** versuchen zu ebenfalls zu pushen.

Der Push gelingt nur Alice. Alle anderen (z.B. "Bob") dürfen danach zunächst nicht pushen, weil bei ihnen das lokale Repo nicht aktuell ist: Dort fehlt der Commit von Alice, der in der Zwischenzeit auf dem Server dazugekommen sind. Die Versionen sind also "auseinandergelaufen". Das bemerkt der Git-Client bei Bobs Push-Versuch und meldet das Problem. Um pushen zu können, muss Bob erst seine eigenen Änderungen mit denen von Alice zusammenführen. Das macht er mit einem **Pull**; der besteht aus **Fetch** (vom Server holen) und **Merge** (zusammenführen).
Hinterher darf er dann pushen — falls ihm nicht schon wieder jemand zuvorgekommen ist.

h. **Gemeinsam:** Beobachten Sie gemeinsam, wie Bob schrittweise den Pull durchführt. Der wählt bei der Auswahl "Rebase oder Merge?" zunächst Merge.

i. **Nur Bob:** committet und pusht das Ergebnis.

j. **Gemeinsam:** Beobachten Sie gemeinsam das Pull-Push bei den restlichen Mitgliedern.

k. **Gemeinsam:** Schließlich müssen sich alle noch mit einem Pull die Änderungen der anderen holen.

l. **Jeder:** Führen Sie eine zweite Runde "ändern-commit-push-pull-push" durch, für die jeder nur in "seiner" Datei etwas ändert.

m. **Gemeinsam:** Erläutern Sie, warum in dieser Übung keine inhaltlichen Konflikte zwischen den Beiträgen der einzelnen Personen auftreten.

Antwort

Die Änderungen betreffen unterschiedliche Dateien, die völlig unabhängig voneinander hin und her kopiert werden können.

1.4. Zusammenarbeiten in einer Datei

a. **Sprecher:** Fork-en Sie auf dem Server das von der Lehrkraft bereitgestellte Repo `git-kennenlernen`. Nehmen Sie Ihre Gruppe in den Fork auf (als *Developer*). Geben Sie jedem Mitglied eine Nummer.

b. **Jeder für sich:** Klonen Sie den Fork vom Server in ein lokales Repo auf Ihren Rechner. Sie öffnen darin die Datei `gituebung-namen.txt`, schreiben Ihren Steckbrief in Ihren Abschnitt und **ändern sonst nichts**.

c. **Jeder** speichert und committet.

d. **Gemeinsam:** Führen Sie das Push-Pull-Merge-Push durch wie vorher. Überzeugen Sie sich, dass am Schluss alle Steckbriefe bei allen angekommen sind.

e. **Gemeinsam:**: Erläutern Sie, warum hier immer noch keine Konflikte auftreten.

Antwort

Die Änderungen sind zwar in der gleichen Datei, darin aber in unterschiedlichen Bereichen. Git kann selbständig die richtigen Änderungen auswählen, wenn zwischen diesen Bereichen noch genügend unveränderte Zeilen stehen.

1.5. Zusammenarbeiten — Konflikt von Hand lösen

a. **Jeder** synchronisiert sein lokales `git-kennenlernen`-Repo mit dem Server, damit wirklich alles clean ist.

b. **Alice** ändert auf ihrem Rechner in der Datei `gituebung-namen.txt` ganz oben den Gruppennamen auf "**Alice-Gruppe**", committet und pusht.

c. **Bob** nennt die Gruppe auf seinem Rechner "Bob-Gruppe", committet und versucht zu pushen.

d. **Ganze Gruppe:** Welches Problem tritt auf? Wer bemerkt es, wer muss es lösen?
Antwort: Weil hier die *gleichen* Zeilen geändert wurden, müsste Git eine der Möglichkeiten wählen. Das ist aber eine *inhaltliche* Entscheidung, die ein Computer nicht treffen kann. Diese Situation heißt **Konflikt** und wird vom Werkzeug erkannt.
Aktiv werden muss Bob selber, der (alleine oder mit Alice) eine Lösung finden muss: Das kann Alice' Version, seine eigene oder auch etwas ganz anderes sein ("ALBO-Gruppe").

e. **Bob** studiert den problematischen Bereich der Datei (in einem Editor, oder in SmartGits Conflict Solver"), entscheidet sich und bereinigt die Datei. Dann muss er **stage**-n (die Änderung für den nächsten Commit vormerken), committen und schließlich noch pushen.

f. **Jeder** muss das Ergebnis wieder pullen.

1.6. Übung: Zusammenarbeit (geschickt)

a. **Jeder** öffnet die Datei `lehrerwitze.txt` in einem Texteditor. Leider ist sie voller Rehctscheibfeler.

b. **Gemeinsam** verteilt die Gruppen die Arbeit: Jeder bekommt einen Witz zugewiesen, korrigiert darin Rechtschreibung, Grammatik und Zeichensetzung, ändert aber **nichts** außerhalb der Markierungen.

c. **Alle** commit-ten, push-en und müssen dann wie gewohnt mergen (und am Schluss nochmal pullen).

1.7. Übung: Zusammenarbeit (weniger geschickt)

a. **Jeder** öffnet die Datei `erster-schultag.txt` im Editor. Die Datei ist voller Fhler.

b. **Gemeinsam** verteilen Sie die Korrektur: Alice ist ist nur für Rechtschreibung zuständig, Bob nur für Grammatik, einer für Interpunktion.

c. **Jeder** commit-tet und versucht die Änderungen zu push-en.

d. **Gemeinsam:** Welche Absprachen würden die Sache erleichtern?

Antwort

Das Mergen ist sehr kompliziert, weil alle Personen in jeder Zeile Änderungen vornehmen. Ein Werkzeug wie Git ersetzt weder klare und sinnvolle Absprachen, noch die nötige Disziplin. Ein sinnvolles Vorgehen wäre beispielsweise, dass *erst* Alice die Rechtschreibung korrigiert, committet und pusht; *dann* pullt Bob neu und kümmert sich um die Grammatik.

1.8. Zusatzübung: Zusammenarbeit

a. **Jeder** öffnet die Datei `letzte-worte.txt` im Editor.

b. Sprechen Sie eine Vorgehensweise ab und korrigieren Sie alle Fehler in den "letzten Worten".

2. Vorbereitung

2.1. `edugit`-Account einrichten

Der Verein `teckids.org` betreibt unter `edugit.org` eine GitLab-Instanz. Diese Anleitung bezieht sich auf GitLab CE 15.3.3, das im Oktober 2022 bei `edugit.org` installiert war.

Die Nutzung von `edugit.org` ist kostenlos; im Herbst 2022 verlangte der Dienst vom Nutzer außer einer E-Mailadresse (die nicht öffentlich sichtbar wird) keine weiteren persönlichen Daten. Das Webinterface bindet keinerlei Fremdinhalte oder Fremdskripte ein (z.B. Google Analytics oder Facebook-Buttons); das Datenschutzniveau ist damit *weit höher als auf praktisch jeder anderen Webseite*. Anders als viele kommerzielle Betreiber ist der Verein wegen seiner erkennbar pädagogischen Ausrichtung auch durchaus glaubwürdig.

- Eröffnen Sie einen Account auf `edugit.org`; derzeit (Herbst 2022) müssen Sie dabei nur Ihre Mailadresse angeben. Rechnen Sie mit einigen Tagen Bearbeitungszeit, aber prüfen Sie regelmäßig Ihren Maileingang und reagieren Sie schnell; Sie erhalten einen Bestätigungslink, der aber bald wieder verfällt.
- Loggen Sie sich im Webinterface ein.
- Öffnen Sie im rot-weißen "Blumen-Menü" rechts oben Ihr Profil und ändern Sie darin
  - Ihre Zeitzone auf Berlin;
  - Ihren "Full name" so, dass Mitschüler und Lehrkräfte ihn zuordnen können (z.B. "HaMe" für Hans Meier), Fremde jedoch nicht;
  - Ihre E-Mail können Sie angeben, aber
  - wählen Sie "Public email — do not show on profile";
  - bei "Commit email" wählen Sie "Use a private email" und notieren die hier angebotene Adresse der Form `123-kuerzel@users.noreply.edugit.org` (Sie brauchen Sie später für die Konfiguration Ihres Git-Clients);
  - setzen Sie den Haken bei "Private profile".
- Erzeugen Sie ein "access token" (eine Art Passwort für Ihren Gitlab-Account, aber mit flexibel konfigurierbaren Rechten):
  - Bleiben Sie dazu in "Settings" und öffnen Sie "Access Tokens";
  - wählen Sie für das Token einen Namen, ein Ablaufdatum (mehrere Jahre), seine Berechtigungen ("write_repository" ist sinnvoll) und erzeugen Sie das Token.
  - Bei Verlust können Sie das Token zwar leicht widerrufen und ein neues erzeugen. Sie sollten es trotzdem sorgfältig verwahren, denn Gitlab zeigt das Token nur dieses eine Mal an. Speichern Sie es in maschinenlesbarer Form (etwa in Ihrem Passwortmanager), denn das Abtippen ist mühsam.

2.2. Smartgit installieren

Mit Smartgit ist ein sehr ausgereifter Git-Client für Linux, MacOS und Windows verfügbar.

Smartgit ist keine freie Software, darf aber für nicht-kommerzielle Zwecke kostenlos genutzt werden. Im Oktober 2022 hat der Autor dieser Anleitung mit dem Hersteller abgesprochen, dass Schüler und Lehrkräfte baden-württembergischer Schulen Smartgit für schulische Zwecke kostenlos nutzen dürfen, Schüler sogar ohne persönliche Registrierung. Dafür ist eine Schullizenz erforderlich, die eine Lehrkraft kostenlos beantragen und allen Schülerinnen der Schule zur Verfügung stellen darf.

Im Dezember 2022 hat Syntevo allerdings darum gebeten, dass Lehrkräfte sich (ebenfalls kostenlos) eine persönliche Lizenz ausstellen lassen.

Eine Lehrkraft (pro Schule) muss eine Schullizenz besorgen:

Als Lehrkraft müssen Sie zunächst für Ihre Schule eine Smartgit-Lizenz beantragen, indem Sie:

- auf `https://www.syntevo.com/register-non-commercial/#academic` eine non-commercial-Lizenz beantragen:
  Dabei müssen Sie drei Häkchen setzen und eine gültige Mailadresse angeben. Das kann Ihre persönliche Schulmail sein; praktischer ist vermutlich, sich von der Schule einen Mail-Alias der Art
  `smartgit-schullizenz@domain.der.schule.de`
  einrichten zu lassen und den anzugeben, damit die Lizenz an die Schule und nicht an Sie persönlich gebunden wird: Dann kann später auch ein Kollege den Alias übernehmen und die Lizenz weiter betreuen, falls Sie z.B. die Schule wechseln.
- Rechnen Sie mit einigen Tagen Bearbeitungszeit.
- Syntevo schickt die Lizenz an diese Adresse und fragt hier ggf. auch nach, wenn die Lizenz missbraucht wird.
- die Lizenzdatei an beliebig viele Schüler (Ihrer Schule!) ausgeben und ihnen mitteilen, welche Mailadresse dazu gehört.

Dieses Vorgehen wurde im Herbst 2022 mit Syntevo abgesprochen; es erlaubt den Einsatz im Unterricht *ohne* dass Schülerinnen sich in irgendeiner Form beim Hersteller registrieren müssen. Syntevo kommt damit der Finanz- und Datenschutzsituation an Schulen weit entgegen.

Weisen Sie Ihre Schüler bitte darauf hin, dass die Lizenz nur den Unterrichtsbetrieb abdeckt. Spätestens nach dem Abitur brauchen sie eine eigene; in keinem Fall ist eine Nutzung durch Eltern o.ä. zulässig!

Um Smartgit auf Ihrem Rechner einsatzbereit zu machen, müssen Sie…

- es hier
  `https://www.syntevo.com/smartgit/download/`
  für Ihr Betriebssystem herunterladen;
- Smartgit installieren (dazu sind normalerweise Administratorrechte erforderlich wie bei

den meisten Installationen);

- beim ersten Start die Lizenzvereinbarung bestätigen und unter "Licence file" die Lizenzdatei Ihrer Schule zur Verfügung stellen;
- das Dialogfenster mit dem Namen der Schule und der Lizenz-Mailadresse *ohne Änderung* bestätigen;
- Ihren Commit-Namen und eine Commit-Email eingeben. Beide sind im Prinzip beliebig; für den Namen bietet sich das Kürzel aus dem `edugit`-Account an ("HaMe" für Hans Meier) und als Mailadresse die oben notierte `noreply`-Adresse;
- im nächsten Fenster "Use Smartgit as SSH client" wählen;
- unter "Style" den "Log Graph" wählen (empfohlen);
- unter "Privacy" je nach Datenschutzbedürfnis die drei Haken entfernen (empfohlen);
- kurz abwarten, während Smartgit diverse Downloads tätigt.

Ihre Smartgit-Installation ist damit einsatzbereit. Wiederholen Sie sie auf allen Computern bzw. Accounts, auf denen Sie Smartgit nutzen wollen. Spätere Starts brauchen dann keine Konfiguration mehr.

3. Glossar

Arbeitsverzeichnis

Ordner, der mit Hilfe eines Git-Repository verwaltet werden soll.

repository

Git-eigene Datenbank, die entweder direkt im `.git`-Unterordner des Arbeitsverzeichnisses liegt (lokales Repo), oder auch auf einem anderen Rechner.

staging area (manchmal auch "Index")

Vorbereitungsschritt, der geänderte Dateien für den nächsten Commit sammelt.

commit

Ein Versionsstand des Projekts, der mit Datum, Kommentar und weiteren Angaben festgehalten wurde; als Verb **committen**: einen Commit vornehmen.

clean

Das Arbeitsverzeichnis ist clean, wenn es mit dem letzten Commit exakt übereinstimmt.

checkout

bringt das Arbeitsverzeichnis auf den Stand eines bestimmten Commit (oder Branch). Es sollte dafür vor dem Checkout clean sein.

merge

auseinander gelaufene Versionen zusammenführen; das können Differenzen *zwischen zwei lokalen Branches* sein, aber auch Änderungen im gleichen Branch zwischen zwei

Teammitgliedern.

push
: neue Commits vom lokalen Repo ins Server-Repo schieben.

fetch
: neue Commits vom Server ins lokale Repo holen.

pull
: fetch und dann merge.

Konflikt
: tritt auf, wenn zwei "auseinandergelaufene" Versionen Änderungen in gleichen Zeilen enthalten, und muss von Hand aufgelöst werden.

branch
: Absichtlich eingerichtete Verzweigung der Versionsgeschichte.

4. Links, Literatur, Software

4.1. Git-Hoster

Für die meisten sinnvollen Workflows braucht man ein zentrales, online zugängliches Git-Hosting. Leider gibt es hier derzeit (Oktober 2022) keine perfekte Lösung:

- ~~`GitHub.com`~~ ist leistungsfähig und beliebt. Weil der Dienst außerdem Sonderkonditionen und eigene Werkzeuge für Lehrer bereithält (und weil manche Schüler danach fragen) sei hier betont, dass ein kommerzieller Dienst aus USA für den unterrichtlichen Einsatz natürlich grundsätzlich nicht in Frage kommt. Damit scheiden auch die meisten anderen Anbieter leider aus.
  Normale Nutzer (also Schüler) dürfen bei GitHub ohnehin nur wenige private Repos anlegen; vom unterrichtlichen Einsatz öffentlicher Repos ist abzuraten.
- `edugit.org` bietet auf deutschem Boden bei glaubwürdig hervorragendem Datenschutz unbegrenzt private Repos — ohne dafür Geld zu verlangen.
  Die Nutzungsbedingungen passen auf eine A4-Seite und sind für Kinder verständlich. Der Service ist für Bildungseinrichtungen gedacht und wird von ehrenamtlichen Kräften eines gemeinnützigen Vereins betrieben. Auch wegen der erkennbar pädagogischen Ausrichtung dürfte das Datenschutzniveau *weit* höher sein als bei anderen kostenlosen Git-Hostern, und auch höher als bei praktisch allen Webseiten. Ein zweites derartiges Angebot ist dem Autor nicht bekannt.
  Wer `edugit.org` nutzt und sich erkenntlich zeigen möchte, findet auf https://teckids.org/pages/spenden.html die nötigen Angaben.

WARNING	Eine Rahmenvereinbarung zwischen dem Träger von `edugit.org` und dem Kultusministerium Stuttgart war in Vorbereitung, liegt im Moment aber leider nicht vor. `edugit.org` verlangt bei der Registrierung zwar lediglich eine Mailadresse; weil aber auch das ein personenbezogenes Datum ist, kann die Benutzung im Unterricht von Schülern streng genommen nicht gefordert werden. Die Lehrkraft kann sich mit einer Einverständniserklärung der Eltern absichern.

- `selbst-hosten.macht-freu.de`: Wer sich Installation und Betrieb zutraut, kann die seinen Schüler selbst anbieten. Dann ist man aber Admin, muss das System pflegen und vermutlich auch ein gewisses Maß an inhaltlicher Überwachung leisten (Beschimpfungen in Commit-Kommentaren ahnden usw.).
- Langfristig wünscht man sich eine datenschutzkonform und verlässlich auf landeseigener Infrastruktur betriebene GitLab- oder Gitea-Instanz, wie bei Moodle.

4.2. Git-Clients

Git "ist" eigentlich eine Datenstruktur innerhalb des Repo (das Sie als `.git`-Ordner sehen), und ein Protokoll für den Austausch zwischen Repos. Die Verwaltung dieser Repos, und der Austausch zwischen ihnen, können mit unterschiedlichen Werkzeugen ("Clients") erfolgen, die sich stark unterscheiden. Hier werden drei Clients genannt, die sich für den Unterricht eignen. Der Autor freut sich über Ihre Rückmeldung zu diesen oder anderen Git-Clients!

4.2.1. BlueJ mit Git

In BlueJ ist ein Git-Client eingebaut, dessen Funktionsumfang sehr stark eingeschränkt wurde. Dafür ist er mit nur drei Buttons, drei Dialogen und zwei Menüeinträgen ausgesprochen übersichtlich.

TODO: Konfiguration, Bedienung

4.2.2. SmartGit

ist ein sehr leistungsfähiger Client, der für nicht-kommerzielle Zwecke kostenlos genutzt werden darf.

TODO: Download, Konfiguration

4.2.3. Kommandozeile

Git kann im vollen Funktionsumfang auf der Kommandozeile bedient werden; wer das beherrscht, kann Git auf jedem System nutzen und ist nicht auf Smartgit oder andere grafische Clients angewiesen. Die Einarbeitung ist allerdings etwas schwieriger, vor allem wenn man die kommandoorientierte Arbeitsweise nicht gewohnt ist, und man hat auch weniger Information im Überblick. Diese Version des Tutorials stellt dafür keine Anleitung bereit.

4.3. Bücher und Tutorials

- knappe Einführung, geeignet als Überblick oder zum Auffrischen: https://lerneprogrammieren.de/git/
- Pro-Git: Sehr ausführliches, präzises Buch, auf Englisch und Deutsch (die Übersetzung ist gar nicht schlecht) kostenlos verfügbar als PDF, epub u.s.w.: https://git-scm.com/book/en/v2 https://git-scm.com/book/de/v2
- Sehr gut gemachte, stufenweise aufgebaute Übungen in deutscher Sprache, vor allem zum Arbeiten mit Branches; verwendet ausschließlich Kommandozeilenbefehle: https://learngitbranching.js.org/?locale=de_DE
- Gamification: "Oh my git" (Spiel, muss installiert werden): https://blinry.itch.io/oh-my-git