Aufgaben für die Konsultationen – Mikrocontroller- und Signalprozessortechnik 1

Hendrik Fehr

9. April 2024

1 Einrichten der Umgebung

1.1 Verwendete Software

In den ersten Konsultationen werden Textdateien behandelt. Die verwendete Software ist Git sowie ein Texteditor mit folgenden Eigenschaften:

- Der Editor gestattet das Überschreiben geöffneter Dokumente und lädt diese bei Aktualisierung neu.
- Der Editor unterstützt die Zeilenenden von Windows und Unix und passt sich der vorgefunden Kodierung der Zeilenenden an.

1.2 Installation von Notepad++ (Windows)

Notepad++ steht in der Regel auf den PC-Pool-Rechnern zur Verfügung, muss also nicht installiert werden. Zur Installation kann es von notepad-plus-plus.org/downloads bezogen werden.

ohne Administratorrechte Bitte das portable Archiv verwenden, der Dateiname lautet beispielsweise `npp.8.6.4.portable.x64.zip`, wobei die Version ggf. abweicht.

Installer (benötigt Administratorrechte) Dateiname: `npp.8.6.4.Installer.x64.exe`, wobei die Version ggf. abweicht.

1.3 Installation von Git (Linux)

Einspielen über den Paketmanager der verwendeten Distribution. Unter git-scm.com/-download/linux sind die Schritte für einige Distributionen aufgeführt. Manche Distributionen stellen die grafischen Git-Komponenten in einem separaten Paket bereit, sodass die Befehle `git gui` und `gitk` nicht sofort funktionieren, sondern erst nach Einspielen

der entsprechende Pakete, die man über die Suchfunktion des Paketmanagers finden kann.

1.4 Installation von Git (Windows)

Unter Windows soll *Git for Windows* eingesetzt werden. Das steht in der Regel auf den PC-Pool-Rechnern zur Verfügung, muss also nicht installiert werden. Ein vorhandenes Git kann an den Einträgen Git GUI Here sowie Git Bash Here im Kontextmenu (Rechtsklick) des Explorers erkannt werden. Oder an dem Startmenü-Eintrag `Git Bash`.

Für den eigenen Rechner kann man Git for Windows über die Seite git-scm.com/-download/win als Installer herunterladen. Der Installer heißt 64-bit Git for Windows Setup oder ähnlich. Und der Dateiname lautet beispielsweise `Git-2.35.1.2-64-bit.exe`, wobei die Version ggf. abweicht. Der Installer kann als Benutzer ohne Administratorrechte ausgeführt werden. Bitte die Einstellungen so vornehmen, wie in den Screenshots in Abbildung 1 dargestellt. Manchmal kann der Installer den Ordner von Notepad++ nicht finden, dann wählt man zunächst die Einstellung `Use Notpad as Git`s default editor` um fortzufahren. Die Umstellung auf Notepad++ wird im Abschnitt **??** gezeigt.

1.5 Konfiguration von Git

Autorenname und Mailadresse Starten Sie die Git-Umgebung durch Ausführen von `Git Bash` im Startmenü, oder durch den Klick auf den Kontextmenü-Eintrag `Git Bash Here` im Explorer. Jetzt müssen Sie ihren Benutzernamen und Mailadresse einstellen:

```
$ git config --global user.name "Vorname Nachname"
$ git config --global user.email "vorname.nachname@tu-ilmenau.de"
```

Bitte tragen Sie den gleichen Namen wie in Moodle ein.

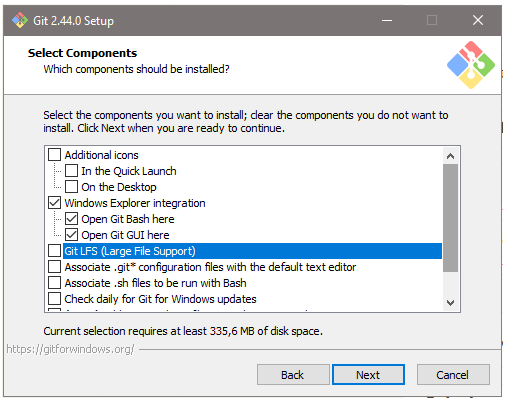
Editor Man kann den Editor nachträglich von Notepad auf Notepad++ umstellen. Das geht am einfachsten durch ausführen von

```
$ git config --global -e
```

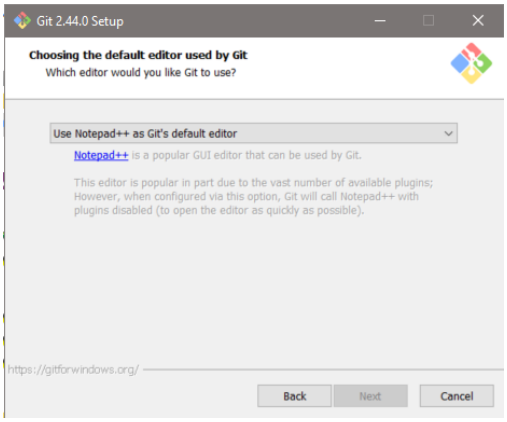
wodurch sich die globale Konfigurationsdatei in Notepad öffnet. Der entscheidende Abschnitt sieht so aus, wobei dort schon die Umstellung auf Notepad++ vorgenommen wurde:

```
[core]
  editor = \"C:\\\\Program Files\\\\Notepad++\\\\notepad++.exe\" -
    multiInst -notabbar -nosession -noPlugin
```

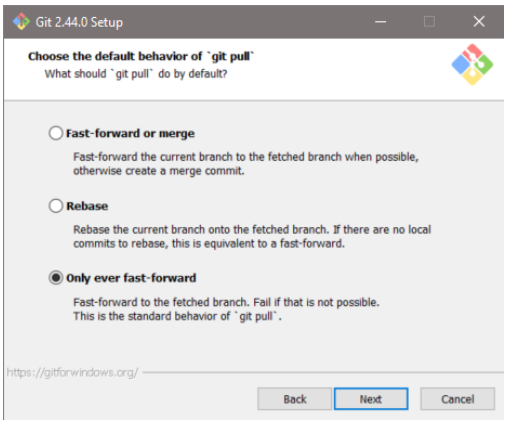
Die `editor`-Einstellung muss ohne die hier sichtbaren Zeilenumbrüche eingetragen werden. Wenn der Git-for-Windows-Installer den Ordner von Notepad++ nicht gefunden hat, müssen Sie den Pfad zur `notepad++.exe` entsprechend anpassen. Nach dem Speichern



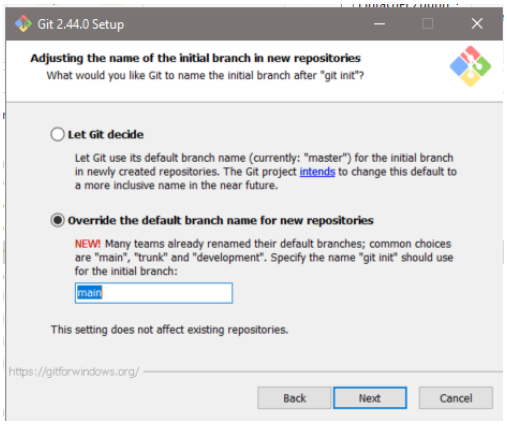
(a) Komponenten (nicht gezeigtes deaktivieren)



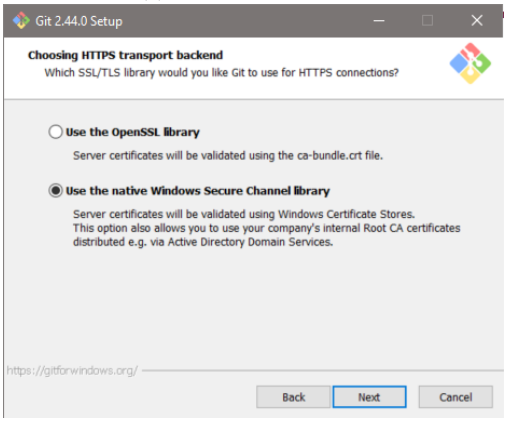
(b) verwendeter Editor



(c) Verhalten von `pull`



(d) Anfangszweig



(e) SSL/TLS-Implementierung

Abbildung 1: Screenshots der geänderten Einstellungen während der Installation von Git for Windows. Die nicht gezeigten Fenster können in der Default-Einstellung bleiben.

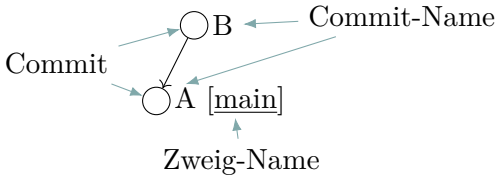


Abbildung 2: Notation von Commits und deren Abhängigkeiten: Kreise sind Commits, Pfeile weisen auf die Elternversionen, Zweignamen stehen in eckigen Klammern, der Zweigname des aktiven Zweigs ist unterstrichen.

und Überschreiben der bestehenden Datei sollte sich Editor jetzt Notepad++ öffnen. Das kann man testen durch nochmalige Eingabe von:

```
$ git config --global -e
```

2 Lokales Arbeiten

Aufgabe 1: Repository Erstellen

(a) Erstellen Sie ein Repository mit dem Namen `repo1`. Das Verzeichnis, in dem Sie `$ git init` ausgeführt haben wird auch als Repository oder Projekt bezeichnet. Standardmäßig speichert Git die Versionen und einige Konfigurationen im Repository in dem Unterverzeichnis `.git`. Dateien im Repository, die sich nicht in `.git` befinden, stellen das Arbeitsverzeichnis dar. Der Stand des Arbeitsverzeichnisses kann festgehalten, also versioniert werden. Die versionierten Dateien werden als *getrackte* Dateien bezeichnet. Erzeugen Sie im Arbeitsverzeichnis von dem Repository `repo1` die Textdatei dat.txt mit etwa 15 bis 20 kurzen Zeilen Inhalt und erstellen Sie einen Commit, um diesen Stand festzuhalten.

(b) Erstellen Sie zwei weitere Commits in denen sich die Datei dat.txt ein bisschen ändert und zeichnen Sie den Versionsgraph vor und nach dem Commit mit Hilfe der in Abb. 2 gezeigten Notation. Auf welches Commit verweist der aktive (d. h. ausgecheckte) Zweig vor dem Commit und nach dem Commit?

(c) Checken Sie jeden einzelnen der Stände aus, durch Angabe der ersten Stellen des jeweiligen Hashs, während dat.txt im Texteditor geöffnet bleibt. Beachten Sie das Verhalten des Editors, wenn beim Auschecken die Datei überschrieben wird.

(d) In der Default-Konfiguration von Git erscheint beim Auschecken eines Commits eine Nachricht. Erklären Sie, auf welche Eigenschaften von Git diese Nachricht zurückgeht. Hilfsfrage: Was kann man tun, damit nach einem `checkout <Hash>` die folgenden Commits dauerhaft erhalten bleiben?

Für die Aufgabe relevante Befehle:[1]

```
$ git init
$ git status
$ git add <Datei>
$ git commit
$ git log
$ git log --all --oneline
$ git checkout <Hash>
```

Aufgabe 2: Neuen Zweig erstellen und weiterführen

Verwenden Sie ein bestehendes Repository, z. B. das der vorangehenden Aufgabe. Zeichnen Sie wieder den Versionsgraph vor und nach den ausgeführten Aktionen auf (nur wenn sich der Graph ändert).

(a) Erstellen Sie den Zweig [A], sodass er sich auf das Commit an der Wurzel bezieht, also das in der vorigen Aufgabe als erstes erstellte Commit.

(b) Stellen Sie sicher, dass der erstellte Zweig [A] der aktive Zweig ist.

(c) Bearbeiten Sie die Datei dat.txt ein bisschen und committen die Änderungen.

(d) Wechseln Sie in den ursprünglichen Zweig zurück.

(e) Kennen Sie schon alternative Möglichkeiten, einen Zweig zu erstellen?

Für die Aufgabe relevante neue Befehle:

```
$ git checkout <Hash> -b <Zweigname>
$ git checkout <Zweigname>
```

Aufgabe 3: Korrigieren des letzten Commits mit `git commit --amend`

Manchmal bemerkt man direkt nach dem Ausführen einen Fehler im Commit. Zum Beispiel einen Tippfehler in der Commit-Nachricht oder einen Fehler im Inhalt. Wenn das Commit gerade eben erst passiert ist, kann man das Versehen mit `git commit --amend` beheben.

(a) Erstellen Sie einen Commit mit einem Tippfehler in der Nachricht.

(b) Korrigieren Sie den Tippfehler.

(c) Was passiert mit dem ursprünglichen Commit?

[1] Syntax (www.tfug.org/helpdesk/general/man.html): `<Pflicht>`, `[optional]`, `{entweder, oder}`.

(d) Warum sollte `git commit --amend` nur auf gerade eben erstelle Commits angewendet werden?

Für die Aufgabe relevante Befehle:

```
$ git commit --amend
```

Aufgabe 4: Tools mit graphischer Benutzeroberfläche

Wiederholen Sie die vorherigen Aufgaben und starten statt `git add`, `git commit` und `git commit --amend` die graphische Benutzeroberfläche mit `git gui &`. Visualisieren Sie die Geschichte mit `gitk --all &` statt mit `git log`.

Die Programmfenster dieser graphischen Benutzeroberflächen erscheinen unter Windows gelegentlich nicht. Dann kann man durch die Tastenkombination *<Alt>-<Leertaste>* das Fenster-Menü aufrufen und *Maxmieren* auswählen, um das Fenster sichtbar zu machen.

Für die Aufgabe relevante neue Befehle:

```
$ git gui &
$ gitk --all &
```

Aufgabe 5: Zweige zusammenführen

(a) Führen Sie den Zweig [A] mit dem ursprünglichen Zweig (z. B. [main] oder [master]) zusammen. Wenn sich Konflikte ergeben, analysieren sie diese im Editor und brechen danach den Merge ab und machen mit Teil (b) weiter.

(b) Versuchen Sie den Inhalt von Zweig [A] so anzupassen, dass die Konflikte vermieden werden: Bearbeiten Sie dazu die Datei dat.txt und committen die Änderungen. Probieren Sie wieder Teil (a). In hoffnungslosen Fällen machen Sie mit c weiter.

(c) Lösen Sie die Konflikte während der Zusammenführung. Dazu bearbeiten Sie die betroffene Datei und fügen das Ergebnis vor dem Merge-Commit hinzu.

```
$ git config [--global] merge.conflictStyle diff3
$ git merge <Quellzweig>
$ git merge --abort
```

Aufgabe 6: Umgang mit stash

Üben Sie den Umgang mit stash.

(a) Suchen Sie zwei Stände, deren Inhalt fast gleich ist. Erstellen Sie für die Stände Zweignamen: Der erste erhält den Zweignamen [B1], der zweite den Namen [B2]. Einer der beiden Zweige muss aktiv sein.

(b) Bearbeiten Sie den Inhalt des aktiven Zweigs, z.B. dat.txt. Aber halt, diese Arbeit sollte eigentlich in den anderen Zweig comittet werden. Versuchen Sie, den anderen Zweig auszuchecken, das wird fehlschlagen, wegen der Änderungen.

(c) Nutzen Sie `stash` um die Änderungen zwischenzuspeichern.

(d) Jetzt können Sie den anderen Zweig auschecken.

(e) Übertragen Sie die gestashten Änderungen in den aktiven Zweig und comitten.

Für die Aufgabe relevante neue Befehle:

```
$ git stash
$ git stash pop
```

Aufgabe 7: Lokales Klonen mit clone

Ermitteln Sie den Unterschied zwischen lokalem Kopieren und lokalem Klonen:

(a) Kopieren Sie den Ordner des oben verwendeten Projekts (repo1). Die Kopie heißt z. B. repo1_kopie.

(b) Erstellen Sie einen Klon des oben verwendeten Projekts. Der Name des Klons ist z. B. repo1_klon.

(c) Nutzen Sie die Kommandozeile oder ein GUI um Unterschiede zu finden.

Beispiele für relevante Befehle:

```
$ git clone <repository> [directory]
$ gitk --all &
$ git branch A --track origin/A
$ git checkout A
$ git checkout -b A
```

Aufgabe 8: Übernehmen von Änderungen in den Klon

Weitere Änderungen im Zweig [A] von repo1 sollen in das geklonte Projekt repo1_klon geholt werden.

- Wechseln Sie in das Projekt repo1 und führen den Zweig [A] weiter.
- Wechseln Sie in das Projekt repo1_klon und fetchen die Änderungen, sodass der Zweig [origin/A] aktualisiert wird.
- Führen Sie den Zweig [A] nach, sodass er denselben Stand hat wie [origin/A].

- Mit `$ git pull` wird das Aktualisieren und Weiterführen in einem Befehl erledigt. Probieren Sie das für eine weitere Änderung im ursprünglichen Repository.

Beispiele für neue relevante Befehle:

```
$ git fetch origin A
$ git merge origin/A
$ git log --oneline A origin/A
```

Aufgabe 9: Bereitstellen von Änderungen aus dem Klon

Änderungen im Zweig [A] von repo1_klon sollen in das Projekt repo1 geholt werden.

- Wechseln Sie in das Projekt repo1_klon und führen den Zweig [A] weiter.
- Wechseln Sie in das Projekt repo1 und Erstellen mit `$ git remote` einen Verweis auf das Projekt repo1_klon unter dem Verweisnamen klon.
- Konfigurieren Sie den Zweig [A] von repo1 so, dass er dem Zweig [klon/A] folgt, also dem Zweig [A] von repo1_klon.
- Führen Sie den Zweig [A] nach, sodass er denselben Stand hat wie [klon/A].

Beispiele für neue relevante Befehle:

```
$ git remote add klon <Pfad zu repo1_klon>
$ git fetch klon A
$ git branch A --set-upstream-to klon/A
$ git pull
```

3 Arbeiten mit Projekten der TU-Ilmenau GitLab-Instanz

Aufgabe 1: Bereitstellen von repo1 als persönliches Projekt

(a) Erstellen sie ein leeres persönliches Projekt in Ihrem GitLab-Bereich. Normalerweise ist eine README.md-Datei sinnvoll zur Dokumentation, aber diesmal deaktivieren wir den Haken zum automatischen Erzeugen dieser Datei. Sie kann bei Bedarf nachträglich erstellt werden.

(b) Nach dem Erstellen wird eine Anleitung angezeigt, wie man ein bestehendes Git-Repository in das leere Projekt kopiert. Folgen sie der Anleitung. Ändern Sie die Form der Adresse aus der Anleitung von

```
git@gitlab.tu-ilmenau.de:<...>.git
```

nach

```
https://gitlab.tu-ilmenau.de/<...>.git
```

wenn Sie über HTTPS und nicht über SSH zugreifen möchten.

(c) Gewähren Sie einem oder mehreren Partnern Zugriff auf das persönliche GitLab-Projekt indem Sie die jeweiligen Benutzerkonten in das Projekt einladen.

Aufgabe 2: An einem Projekt mitarbeiten

(a) Klonen sie das Projekt https://gitlab.tu-ilmenau.de/FakEI/ees-lse/lehre/msp1/material-msp-1/einfache-textdatei-und-mehrere-zweige.

(b) Es soll ein neuer Zweig, namens [D] entstehen aus der Zusammenführung der Zweige [A] und [B].

(c) Es soll ein neuer Zweig, namens [D2] entstehen aus der Zusammenführung der Zweige [A] und [B2].

(d) Stellen Sie die Arbeitsergebnisse, also die Zweige [D] und [D2] in einem persönlichen Projekt bereit.

Aufgabe 3: Gegenseitiges Bereitstellen von Arbeitsergebnissen

Organisieren Sie Gruppen von mindestens zwei TeilnehmerInnen und setzen persönliche Projekte auf:

- Jeder gewährt den anderen (mindestens) lesenden Zugriff auf sein persönliches GitLab-Repository.
- Jeder hat eine lokale Instanz indem sein persönliches Repository und die der anderen jeweils als sinnvoller `remote`-Verweis eingetragen sind.

Typischerweise stellt ein Teilnehmer ein Ausgangsprojekt bereit, welches die anderen klonen und als ihr persönliches Projekt wieder hochladen. Die Bereitstellungen werden auf dem jeweiligen persönlichen Projekt getätigt und können durch die anderen mit `fetch` oder `pull` bezogen werden.