



Aufgabenblatt 10

Ausgabe: 30.10.2023 14:00

Abgabe: 10.11.2023 20:00

Thema: Erste Schritte mit Git

Abgabemodalitäten

1. Die Aufgaben des Programmier-Kurses bauen aufeinander auf. Versuchen Sie daher die Aufgaben zeitnah zu bearbeiten bzw. hochzuladen.
2. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf Ihrem Rechner mit dem Befehl `clang -std=c11 -Wall -g` kompilieren.
3. Die Abgabe für den Quellcode erfolgt ausschließlich über unser Git im entsprechenden Branch. Nur wenn ein Ergebnis im **ISIS-Kurs** angezeigt wird, ist sichergestellt, dass die Abgabe erfolgt ist. Die Abgabe ist bestanden, wenn Sie an Ihrem Test einen grünen Haken sehen.
4. Sie können bis zur Abgabefrist beliebig oft neue Versionen abgeben. Lesen Sie die Hinweise der Tests genau durch, denn diese helfen Ihnen, Ihre Abgabe zu korrigieren.
Bitte beachten Sie, dass ausschließlich die letzte Abgabe gewertet wird.
5. Die Abgabe erfolgt, sofern nicht anders angegeben, in folgendem Branch: `pkurs-b<xx>`, wobei `<xx>` durch die zweistellige Nummer des Aufgabenblattes zu ersetzen ist.
6. Geben Sie für jede Aufgabe die Quelldatei(en) gemäß der Vorgabe ab. Im **ISIS-Kurs** werden zum Teil Vorgabedateien bereitgestellt. Nutzen Sie diese zur Lösung der Aufgaben.
7. Die Abgabefristen werden vom Server überwacht. Versuchen Sie, Ihre Abgabe so früh wie möglich zu bearbeiten und hochzuladen. Sie minimieren so auch das Risiko, die Abgabefrist auf Grund von „technischen Schwierigkeiten“ zu versäumen. Eine Programmieraufgabe gilt als bestanden, wenn alle bewerteten Teilaufgaben bestanden sind.

Erste Schritte mit Git

Git ist ein Open Source Versionsverwaltungssystem und kommt hauptsächlich bei der Entwicklung von Software zum Einsatz. Es protokolliert Änderungen an Dateien (z. B. Programmcode) in einem *Repository* und erlaubt dabei jederzeit den Zugriff auf jeden der protokollierten Zustände (*Commits*). Änderungen am Programmcode sind somit immer nachvollziehbar. SoftwareentwicklerInnen ist es dadurch möglich, parallel und koordiniert an einem gemeinsamen Projekt zu arbeiten.

In dieser Lehrveranstaltung werden Sie Git benutzen, um den Programmcode Ihrer Hausaufgaben zu verwalten. Im Folgenden sind die dafür notwendigen Konzepte, Abläufe und Git-Befehle erklärt. Um die Inhalte nachvollziehen zu können, ist es hilfreich, bereits mit den grundlegenden Befehlen in einem Terminal (Bewegen im Dateisystem: `cd`, `ls`, `pwd`; Dateioperationen: `cp`, `mv`, `touch`, `mkdir`) und der Verwendung eines Texteditors vertraut zu sein. Weiterführende Informationen zu Git finden Sie z.B. online im **Pro Git**-Buch oder in den Handbüchern zu den einzelnen Git-Befehlen (`man git <befehl>`).

Git gibt es für alle gängigen Betriebssysteme. Eine ausführliche Anleitung, um einen Git-Client für Ihr jeweiliges Betriebssystem zu installieren, finden Sie ebenfalls im **Pro Git**-Buch.

Falls Sie zum ersten Mal mit Git arbeiten, müssen nach der Installation noch zwei Einstellungen vorgenommen werden, um Git Ihren Namen und Ihre E-Mail-Adresse mitzuteilen. Führen Sie dazu die folgenden zwei Befehle aus:

```
git config --global user.name '<Ihr Name>'
git config --global user.email '<ihre@mail.adresse>'
```

Der Name und die E-Mail-Adresse können an dieser Stelle frei gewählt werden. Wir empfehlen, dass Sie Ihre TU-E-Mail-Adresse verwenden.

Initialisieren eines Git-Repository

Dieser Schritt ist nur nötig, wenn Sie noch kein Repository geklont haben. Nach dem Check-In auf ISIS können Sie Ihr Abgaberepository auch klonen (vgl. Abschnitt: Arbeiten mit verteilten Repositories).

Erstellen Sie ein Verzeichnis mit beliebigem Namen (`mkdir <name>`) und wechseln Sie in dieses Verzeichnis (`cd <name>`). Mit dem Befehl `git init` können Sie nun in diesem Verzeichnis ein lokales Repository initialisieren. Dabei wird ein Unterverzeichnis¹ `.git` angelegt, welches das Metadaten und Parameter des Repositories beinhaltet. Die Dateien in diesem Unterverzeichnis werden durch Git selbst verwaltet und sollten im Regelfall nicht angepasst werden.

¹Bei Unix-artigen Betriebssystemen sind Verzeichnisse, welche mit einem Punkt beginnen, versteckte Verzeichnisse. Diese können mit dem Befehl `ls -a` angezeigt werden.

Der Status des Repositories kann mit dem Befehl `git status` abgefragt werden. Dieser gibt unter anderem an, in welchem Branch Sie sich im Repository befinden. Für das eben erstellte Repository sollte das der `main`-Branch ² sein. Ebenso zeigt der Status, ob geänderte Dateien im Repository sind. Da dieses Repository noch leer ist, werden keine Änderungen angezeigt.

Versionierung von Dateien

Im initialisierten Verzeichnis des Repository können Ordner und Dateien angelegt werden, um diese später in einem zweiten Schritt mit der Versionskontrolle zu verwalten. Erzeugen Sie dazu zuerst eine leere Textdatei im Verzeichnis des Repositories (z.B. mit `touch datei.txt`). Die eben erstellte Datei befindet sich zwar im Verzeichnis des Repository, jedoch noch nicht mit `git` verwaltet. Sie wurde noch nicht dem Versionsverwaltungssystem hinzugefügt. Der Befehl `git status` sollte deshalb auch die Datei unter `Untracked files` auflisten.

Um die Datei nun tatsächlich dem Repository hinzuzufügen, führen Sie `git add datei.txt` aus. Der Befehl `git status` listet daraufhin die neu hinzugefügte Datei als Änderung am Repository auf (`new file: datei.txt`). Für diese Änderung soll nun ein Commit angelegt werden, welcher die Änderungen im Repository protokolliert. Jedem Commit muss mit dem Parameter `-m` eine benutzerdefinierte Nachricht angehängt werden, welche die Änderung beschreibt. Führen Sie den Befehl `git commit -m "Neue Datei datei.txt hinzugefügt"` aus, um einen Commit anzulegen.

Durch den mehrfachen Aufruf von `git add` können einzelne Änderungen auch zu einem Änderungssatz zusammengefasst werden. Der Befehl `git add <datei>` fügt dabei nicht nur neue Dateien, sondern auch Änderungen an bestehenden Dateien zu einem Änderungssatz hinzu.

Führen Sie die unten stehenden Schritte durch, um sich mit den Befehlen und Abläufen vertraut zu machen. Vergleichen Sie nach jedem Schritt die Ausgabe von `git status` mit Ihren Erwartungen.

1. Erstellen Sie eine neue Datei `datei.txt` im Verzeichnis Ihres Repositories.
2. Fügen Sie die neue Datei dem Repository hinzu (`git add <datei>`).
3. Ändern Sie den Inhalt der bisher leeren Datei `datei.txt` und schauen Sie sich die Änderungen an (`git diff`).
4. Fügen Sie die Änderung von `datei.txt` zum Änderungssatz hinzu (`git add datei.txt`).
5. Erstellen Sie einen Commit, der alle bisherigen Änderungen (neue Datei und Änderungen an `datei.txt`) enthält (`git commit -m "<Nachricht>"`).

Der Zustand (Snapshot) des Repositories zum Zeitpunkt des Commits ist durch eine Prüfsumme (Hash³) eindeutig identifizierbar. Anhand dieser Prüfsumme kann Git den Zustand wiederherstellen oder mit dem anderer Commits vergleichen. Der Befehl `git log` zeigt Ihnen die Historie der Commits im aktuell aktiven Branch Ihres Repositories.

Verwalten von Branches

Mit Hilfe von Branches können verschiedene Entwicklungen parallel betrieben und später wieder zusammengefügt werden (*Merge*). In der Lehrveranstaltung nutzen wir Branches jedoch ausschließlich zur Trennung der einzelnen Abgaben. Ein neuer Branch wird immer vom letzten Commit (Zustand) des aktiven Branches abgezweigt. Mit dem Befehl `git branch <branch-name>` oder `git switch -c <branch-name>` kann ein Branch erzeugt und mit `git branch -d <branch-name>` wieder gelöscht werden. Eine Übersicht aller (auch remote) verfügbaren Branches erhält man mit `git branch -a`. Den jeweils aktiven Branch kann man mit dem Befehl `git checkout <branch-name>` oder `git switch <branch-name>` wechseln.

Führen Sie folgende Schritte aus, um sich mit den Befehlen und Abläufen vertraut zu machen. Prüfen Sie nach jedem Schritt mit `git status`, in welchem Branch Sie sich befinden.

1. Stellen Sie sicher, dass Sie sich im `master` oder `main`-Branch Ihres Repositories befinden (`git status`).
2. Erstellen Sie einen neuen Branch (`git branch <branch-name>` oder mit gleichzeitigem Wechsel `git switch -c <branch-name>`).
3. Wechseln Sie in den vorhandenen Branch (`git checkout <branch-name>` oder `git switch <branch-name>`), falls Sie noch nicht gewechselt haben.
4. Führen Sie eine Änderung am Repository durch und erzeugen Sie einen Commit. Lassen Sie sich das Commit-Log ausgeben (`git log`). Sie beenden die Anzeige mit „q“ im Terminal.
5. Wechseln Sie zurück in den `master` oder `main`-Branch. Lassen Sie sich das Commit-Log ausgeben und vergleichen Sie es mit der vorherigen Ausgabe.

Arbeiten mit verteilten Repositories

Bisher haben wir nur in einem lokalen Repository gearbeitet. Git erlaubt aber auch das Teilen von Repositories und damit die Kollaboration in einem Projekt. Für diese Lehrveranstaltung nutzen wir persönliche Repositories, welche von der Gitlab-Plattform der TU-Berlin (<https://git.tu-berlin.de>) bereit gestellt werden. Dort finden Sie später auch Ihr eigenes Repository mit dem Namen Ihres TUB-Accounts in der `introprog-ws23` Gruppe. Sie müssen sich einmalig über die Webseite mit Ihrem TUB-Account am Gitlab anmelden, damit Ihr Gitlab abschließend provisioniert wird.

Ein Repository klonen: Mit dem Befehl `git clone <repository-url>` legen Sie eine lokale Kopie des Remote-Repositories an. Beim Aufruf des Befehls werden Sie gegebenenfalls nach Ihrem TUB-Nutzeraccount gefragt. Um den Accountnamen und das Passwort nicht immer eingeben zu müssen, kann der Zugang per SSH⁴ eingerichtet werden. Nach dem Klonen finden Sie das Repository in dem dabei neu erstellten Ordner `<TUB-Account>`. Dort können Sie nun Ihre Abgabendateien versionieren und Branches verwalten.

Bevor Sie das tun können, müssen Sie im ISIS Kurs ein Check-In <https://isis.tu-berlin.de/mod/lti/view.php?id=1707205> für Ihren Benutzeraccount durchführen. Damit verknüpfen Sie Ihren TUB-Account mit unserem automatischen Abgabensystem Osiris. Zusätzlich wird in diesem Schritt Ihr persönliches Jetzt können Sie in ISIS zur jeweiligen Abgabe Ihr Feedback ansehen und prüfen, ob die Abgabe bestanden ist.

²Je nach Git-Version und Servereinstellungen kann es auch `master`-Branch heißen

³Beispiel: 5303a671af5b28a9c095024ffd630aca3d6c9ce1, Kurzform: 5303a671

⁴<https://docs.gitlab.com/ee/ssh/>

Wenn Sie noch keinen TUB-Account haben, wenden Sie sich an das Campusmanagement ⁵. Dann können Sie weitermachen.

Änderungen übertragen: Bislang arbeiten Sie auf der lokalen Kopie Deines Repository. Um Commits und Branches an die Gitlab-Plattform zu übertragen, führen Sie den Befehl `git push` aus. Falls Sie lokal einen neuen Branch erzeugt hatten und dieser auf der Gitlab-Plattform noch nicht existiert, muss er zuerst mit `git push --set-upstream origin <branch-name>` auf der Plattform angelegt werden.

Führen Sie folgende Schritte durch, um sich mit den Befehlen und Abläufen vertraut zu machen. Gleichzeitig bereiten Sie dabei die Abgabe für die erste Aufgabe vor.

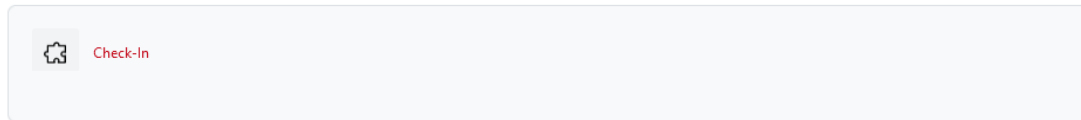


Abbildung 1: Hier auf Check-In klicken <https://isis.tu-berlin.de/mod/lti/view.php?id=1707205>

1. Verlassen Sie den bislang genutzten Ordner, da der nächste Schritt nur funktioniert, wenn Sie sich nicht in einem git verwalteten Ordner befinden.
2. Führen Sie den Check-In auf ISIS durch. Siehe Abbildung 1
3. Klonen Sie Ihr persönliches IntroProg Repository (`git clone https://git.tu-berlin.de/introprog-ws23/<TUB-Account>`).
4. Erstellen Sie und wechseln Sie in einen neuen Branch (z.B. `git switch -c pkurs-b10`).
5. Übertragen Sie den Branch an das Repository auf der Gitlab-Plattform (`git push --set-upstream origin pkurs-b10`).
6. Melden Sie sich auf <https://git.tu-berlin.de> an und machen Sie sich mit der Weboberfläche von Gitlab vertraut. Finden Sie Ihr persönliches IntroProg Repository und prüfen Sie, ob der Branch `pkurs-b10` angelegt wurde. ⁶
7. Bei Problemen wenden Sie sich unbedingt schnell an unsere TutorInnen im Tutorium oder dem betreuen Arbeiten und suchen Sie sich Hilfe.

⁵<https://www.tu.berlin/campusmanagement/angebot/tub-account>

⁶Unter <https://git.tu-berlin.de> können Sie auch weiter eigene (private) Projekte und Repositories verwalten.

Aufgabe 1 Teil 1: Hallo Osiris!

Die erste Aufgabe soll Sie mit dem Workflow unserer Abgabeplattform vertraut machen. Dieser Ablauf ist in jeder Programmierhausaufgabe ähnlich. Die Abgaben erfolgen immer in Ihrem persönlichen IntroProg Repository:

<https://git.tu-berlin.de/introprog-ws23/<TUB-Account>>

Jede Aufgabe muss in einem separaten Branch eingereicht werden. Das Schema der Branch-Bezeichnung finden Sie in den Abgabemodalitäten. Für diese Aufgabe lautet der Branch **pkurs-b10**. Stellen Sie für Ihre Abgaben sicher, dass Sie neue Branches immer vom (leeren) **main**-Branch abzweigen, damit dort keine veralteten Dateien liegen. Erstellen Sie Ihre Dateien immer im Root-Verzeichnis Ihres IntroProg Repositories. Abgaben in Unterverzeichnissen können nicht gewertet werden.

Eine Datei abgeben: Erstellen Sie im Abgabe-Branch dieser Aufgabe (**pkurs-b10**) in Ihrem persönlichen Repository eine Textdatei **pkurs_blatt10_hallo_osiris.txt** mit beliebigem Inhalt. Protokollieren Sie die Änderung mit einem Commit (**git add** und **git commit**). Übertragen Sie abschließend mit (**git push**) diese an das Repository der Gitlab-Plattform.

Für die erste Aufgabe ist hier noch mal der vollständige Ablauf beschrieben. Die Schritte 1-3 können übersprungen werden, wenn Sie bereits erfolgreich *Erste Schritte mit Git* durchgeführt haben.

1. Klonen Sie Ihr persönliches Repository (**git clone https://git.tu-berlin.de/introprog-ws23/<TUB-Account>**) und wechseln Sie in das erzeugte Verzeichnis.
2. Erstellen Sie einen neuen Branch (**git branch pkurs-b10**).
3. Wechseln Sie in den neuen Branch (**git checkout pkurs-b10**).⁷
4. Erstellen Sie eine Textdatei (**pkurs_blatt10_hallo_osiris.txt**) mit beliebigem Inhalt im Verzeichnis Deines Repository.
5. Fügen Sie diese Änderung zum Repository hinzu (**git add pkurs_blatt10_hallo_osiris.txt**).
6. Erstellen Sie einen Commit für diese Änderung (z.B. **git commit -m 'Leere Abgabedatei hinzugefügt'**).
7. Übertragen Sie die Änderungen auf dem neuen Branch an das Repository auf der Gitlab-Plattform (**git push --set-upstream origin pkurs-b10**).
8. Überprüfen Sie Ihre Abgabe in ISIS, Details folgen.

Schauen Sie sich in unserem **ISIS-Kurs (Abgabe Programmierkurs Blatt 10)** das Ergebnis und das Feedback zu Ihrer Abgabe an.

Hinweise für die Abgaben Je nach Auslastung der Abgabeplattform kann es wenige Sekunden dauern, bis das Ergebnis hier:<https://isis.tu-berlin.de/mod/lti/view.php?id=1708289> angezeigt wird. Lesen Sie sich das Feedback aufmerksam durch und folgen Sie den Hinweisen für notwendige Änderungen an der Textdatei **pkurs_blatt10_hallo_osiris.txt**, um die Abgabe zu bestehen. Dort gelangen Sie hin, wenn Sie im ISIS Kurs auf die entsprechende Aktivität „Externes Tool“ in Abbildung 2 klicken.

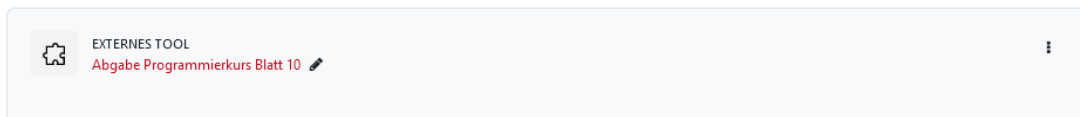


Abbildung 2: Hier auf Abgabe Programmierkurs Blatt 10 klicken.

Eine Abgabe erfüllt alle Anforderungen und gilt nur als bestanden, wenn alle obligatorischen Tests der Abgabe erfolgreich abgeschlossen werden und oben rechts in Abbildung 3 mit einem doppelten weißen Haken auf grünem Hintergrund zu sehen ist.

Abgabe Programmierkurs Blatt 10

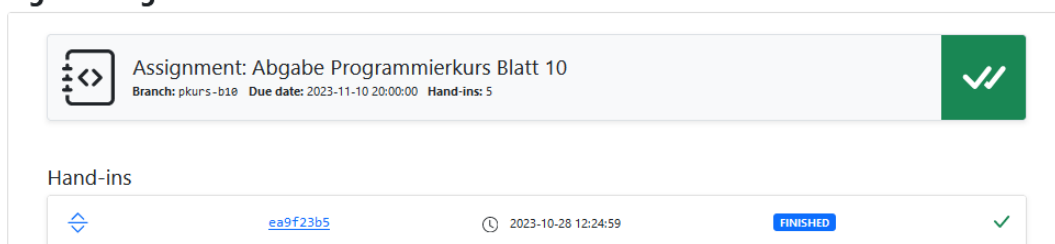


Abbildung 3: Ihre Abgabe, mit allen Untertests ist bestanden.

Wenn Sie zu einer Programmieraufgabe noch nicht **git push** im entsprechend erzeugten Abgabebranch ausgeführt haben, sehen Sie üblicherweise ein graues Fragezeichen wie in Abbildung 4. Dort finden Sie ebenso den Verweis auf den Abgabebranch und Ihre Abgabefrist. Daneben befindet sich eine Information, wie oft sie Ihre Abgabe schon hochgeladen haben.

Ist Ihre Aufgabe noch nicht vollständig gelöst oder enthält sie Fehler, dann wird dies durch ein weißes Kreuz auf rotem Hintergrund gezeigt. Dieses ist in Abbildung 5 veranschaulicht. Unterhalb von „Hand-Ins“ werden alle Abgaben aufgelistet, die Sie mit **git push** übertragen haben. Für eine Abgabe finden Sie in der linken Seite einen Button, mit dem Sie Details Ihrer Tests einsehen können. In der zweiten Spalte findet sich der letzte commit-hash, der Ihrem **git push** vorangegangen war. In der Mitte finden Sie den Zeitpunkt zu dem Sie **git push**. Dieser Zeitpunkt ist der für die Abgabefrist relevante Zeitpunkt. In der nächsten Spalte können Sie den Status des Tests sehen, also ob dieser beendet ist oder noch läuft. In der Tabelle 1 finden Sie die vollständige Übersicht dieser Status.

⁷Schritt 2 und 3 können auch mit **git switch -c pkurs-b10** zusammengefasst werden.

Abgabe Programmierkurs Blatt 10

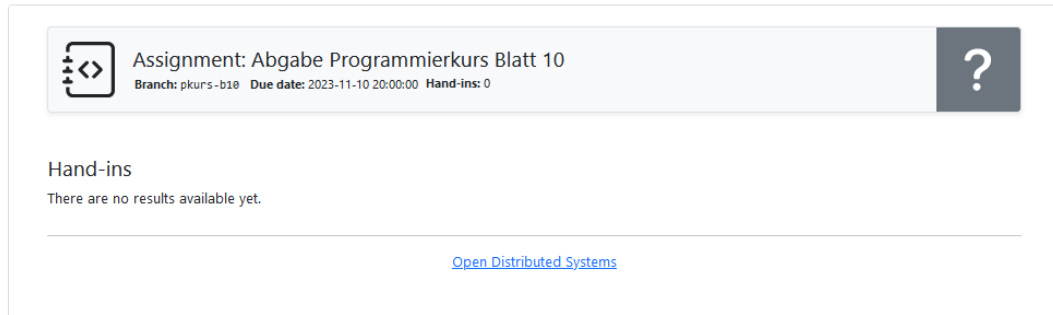


Abbildung 4: Ansicht, wenn noch keine Abgabe durchgeführt wurde.

Abgabe Programmierkurs Blatt 10

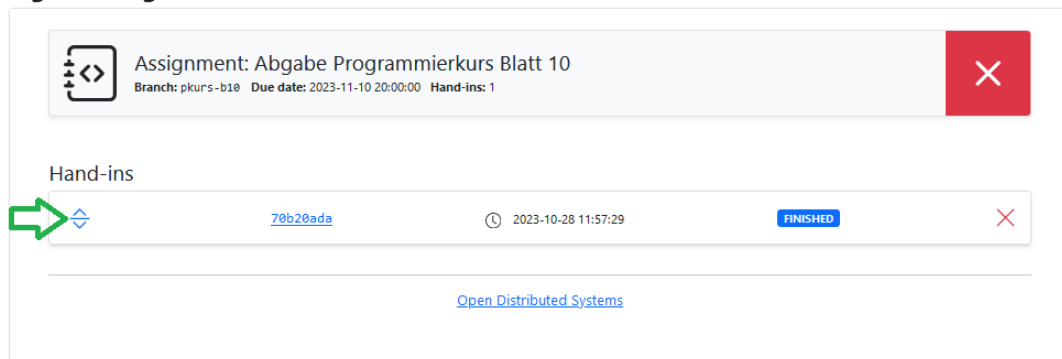


Abbildung 5: Ansicht, wenn noch keine Abgabe durchgeführt wurde, d.h. ein leerer Branch wurde mit git übertragen.

Um die Details Ihrer Abgabe einzusehen, klicken Sie entsprechend der Pfeilmarkierung in Abbildung 5 und kommen in die Detailansicht Abbildung 6. Hier können Sie sehen, warum ein Test fehlgeschlagen ist und erhalten ggf. Hinweise, die zur Lösung des Problems führen können.

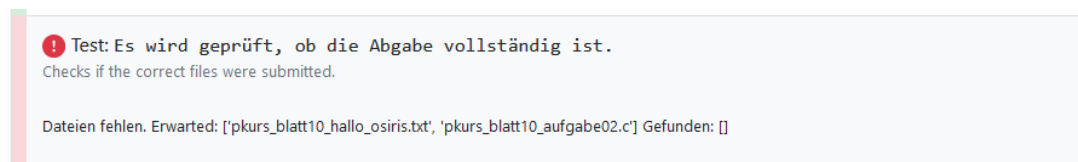


Abbildung 6: Ansicht, wenn noch keine Abgabe durchgeführt wurde.

Abbildung 7 zeigt eine Abgabe wo ein großer Teil der Tests schon erfolgreich ist, jedoch eine Teilabgabe noch fehlt.


Abbildung 8 zeigt dieselbe Abgabe, bei der die Abgabe auf dem Testsystem nicht kompilieren konnte. In dem Fall beachten Sie Compilerwarnungen und Fehlermeldungen auf Ihrem lokalen Rechner.

Sollten Sie eine Abgabe nach Ablauf der Abgabefrist einreichen, so wird diese nicht mehr gewertet. Es gilt dann das Ergebnis der letzten Abgabe, davor, bei der der Abgabeprozess fehlerfrei beendet werden konnte. Siehe auch die Status in Tabelle 1.


Im Falle der Abbildung 9 ist die Abgabe bestanden.

Im Beispiel Abbildung 10, hingegen, ist die Abgabe nicht bestanden.

Abgabe Programmierkurs Blatt 10

**Assignment: Abgabe Programmierkurs Blatt 10**

Branch: pkurs-b10 Due date: 2023-11-10 20:00:00 Hand-ins: 3



Hand-ins













| | | | | |
|---|--------------------------|---|-----------------|---|
|  | cf017bfc |  2023-10-28 12:10:40 | FINISHED |  |
| Test: File test: File Test Check 1 Prüfung ob 'Hallo Osiris!' im Text enthalten ist. | | | | |
| Test: File test: File Test Check 2 Prüfung auf Minimum an Zeilen | | | | |
| Test: File test: File Test Check 3 Prüfung auf Maximum an Zeilen | | | | |
| Test: File Test Check Zusammenfassung Es wurden die Anforderungen an die Abgabe der Textdatei überprüft. | | | | |
|  Test: Es wird geprüft, ob die Abgabe vollständig ist. Checks if the correct files were submitted. The submission is missing the following files: {pkurs_blatt10_aufgabe02.c} | | | | |
|  | 0da3ce88 |  2023-10-28 12:07:35 | FINISHED |  |
|  | 70b20ada |  2023-10-28 11:57:29 | FINISHED |  |

Abbildung 7: Ansicht, wenn noch keine Abgabe durchgeführt wurde.

Abgabe Programmierkurs Blatt 10

**Assignment: Abgabe Programmierkurs Blatt 10**

Branch: pkurs-b10 Due date: 2023-11-10 20:00:00 Hand-ins: 4



Hand-ins







| | | | | |
|--|--------------------------|---|-----------------|---|
|  | 156fd5a5 |  2023-10-28 12:18:37 | FINISHED |  |
| Test: File test: File Test Check 1 Prüfung ob 'Hallo Osiris!' im Text enthalten ist. | | | | |
| Test: File test: File Test Check 2 Prüfung auf Minimum an Zeilen | | | | |
| Test: File test: File Test Check 3 Prüfung auf Maximum an Zeilen | | | | |
| Test: File Test Check Zusammenfassung Es wurden die Anforderungen an die Abgabe der Textdatei überprüft. | | | | |
| Test: Es wird geprüft, ob die Abgabe vollständig ist. Checks if the correct files were submitted. | | | | |
|  Test: Compiler Check Checks the compilation step. Failed. | | | | |

Abbildung 8: Ansicht, wenn noch keine Abgabe durchgeführt wurde.

Abgabe Programmierkurs Blatt 10



Assignment: Abgabe Programmierkurs Blatt 10
 Branch: pkurs-b10 Due date: 2023-10-10 20:00:00 Hand-ins: 6



Hand-ins


Deadline passed!

For this assignment you cannot hand-in new solutions.


| | | | | |
|---|--------------------------|-----------------------|----------|---|
| — | d9a1c9c4 | 🕒 2023-10-28 12:32:23 | MISSED | ? |
| ⬇ | ea9f23b5 | 🕒 2023-10-28 12:24:59 | FINISHED | ✓ |

Abbildung 9: Ansicht, wenn noch keine Abgabe durchgeführt wurde.

Abgabe Programmierkurs Blatt 1



Assignment: Abgabe Programmierkurs Blatt 1
 Branch: pkurs-b01 Due date: 2023-10-10 20:00:00 Hand-ins: 0



Hand-ins

Deadline passed!

For this assignment you cannot hand-in new solutions.

There are no results available yet.

[Open Distributed Systems](#)

Abbildung 10: Ansicht, wenn noch keine Abgabe durchgeführt wurde.

| Status | Spalte 4 | Bedeutung |
|----------|----------|--|
| FINISHED | | Der gesamte Abgabeprozess wurde ohne Fehler beendet. zusätzlich in Spalte 5 grüner Haken, Abgabe ist bestanden |
| SKIPPED | | Der Test wurde übersprungen, da ein neuer Test abgegeben wurde. |
| MISSED | | Die Abgabe fand nach Ablauf der Abgabefrist statt und wird nicht mehr bewertet. |
| STALLED | | Das Testergebnis konnte nicht rechtzeitig ermittelt werden. |
| RUNNING | | Der Test wird gerade durchgeführt. |
| ERROR | | Der gesamte Abgabeprozess wurde mit Fehler beendet. Deutet auf ein technisches Problem im Abgabesystem hin. |

Tabelle 1: Status der automatischen Tests

Aufgabe 2 Teil 2: Hallo Osiris!

Schreiben Sie ein C-Programm, welches auf dem Terminal mittels `printf` den Text „Hallo Osiris!“ mit Zeilenumbruch ausgibt, d.h. die Ausgabe erfolgt auf einer separaten Zeile. Erweitern Sie das Programm mit Hilfe einer Funktion `prepare_repeated_message` so, dass die Ausgabe so oft wiederholt wird, wie in `repeat` angegeben. Diese Funktion soll nicht direkt auf `stdout` schreiben, sondern den Ausgabestring entsprechend vorbereiten. Sie können die Funktion in der `main`

Eine beispielhafte Benutzung des Programms liefе ab wie in Listing 1 gezeigt:

Listing 1: Kompilieren und Aufruf des Programms

```
> clang -std=c11 -Wall -g pkurs_blat10_aufgabe02.c -o pkurs_blat10_aufgabe02
> ./pkurs_blat10_aufgabe02
Hallo Osiris!
> _
```

Stellen Sie sicher, dass Ihre Aufgabe dabei die folgenden Bedingungen erfüllt:

- Die Bibliotheken `stdio.h`, `stdlib.h` und `string.h` können benutzt werden.
- Der Dateiname lautet `pkurs_blat10_aufgabe02.c`.
- Die Funktion `main` ist definiert, die Signatur spielt keine Rolle.
- Die Funktion mit der Signatur `char* prepare_repeated_message(char* message, int repeat)` ist implementiert und führt keine Ausgaben durch und benutzt z.B. `sprintf` zur Stringerzeugung.
- `printf` wird zur Ausgabe von Text auf dem Terminal nur in der Funktion `main` benutzt. „Hallo Osiris!“ darf nur hier „hard-codiert“ sein.
- Die Datei kompiliert ohne Fehler und Warnungen beim Aufruf von:
`clang -std=c11 -Wall -g pkurs_blat10_aufgabe02.c -o pkurs_blat10_aufgabe02`

Fügen Sie Ihre Lösung als Datei `pkurs_blat10_aufgabe02.c` im Abgabebereich `pkurs-b10` in Ihr persönliches Repository ein und übertragen Sie die Lösung an die Abgabepattform.

Aufgabe 3 Programmierkursabgaben ab dem 02.11.2023

Die Aufgaben Blatt01 bis Blatt09 haben Sie bisher nur lokal auf Ihrem Rechner bearbeitet und getestet. Wie bereits in der Ankündigung bekannt gegeben, müssen Sie diese Blätter nun mit Git an unser Abgabesystem, wie in den ersten beiden Teilen dieses Blattes, übergeben.

Verwenden Sie dabei die in der Tabelle 2 aufgelisteten Branchnamen und laden Sie nur die angegebene Datei hoch.

| Blatt | Branch | Datei |
|-------|-----------|--------|
| 01 | pkurs-b01 | 01ex.c |
| 02 | pkurs-b02 | 02ex.c |
| 03 | pkurs-b03 | 03ex.c |
| 04 | pkurs-b04 | 04ex.c |
| 05 | pkurs-b05 | 05ex.c |
| 06 | pkurs-b06 | 06ex.c |
| 07 | pkurs-b07 | 07ex.c |
| 08 | pkurs-b08 | 08ex.c |
| 09 | pkurs-b09 | 09ex.c |

Tabelle 2: Abgaben des Programmierkurses