Vorkus
Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI

# Musterlösung 4
# Git & Unix

Daniel Weber
Jessica Schmidt,
Luise Puhl,
Pascal Lauer



## Hinweis

Die Vorlesung hat Ihnen eine Übersicht grundlegenden Unix-Befehle gezeigt, die einem Unix-Nutzer nahezu täglich begegnen. Trotzdem gibt es viele Tricks, die einem das Unix-Leben einfacher machen. Vermuten Sie also, dass es auch einen einfacheren Weg geben muss als die Vorlesung Ihnen verraten hat, ist es in jedem Fall ratsam, nach einer anderen Lösung zu googlen oder in die `man` page des von Ihnen verwendeten Befehls zu schauen. Die in der Vorlesung gezeigten Befehle reichen jedoch auch zum Lösen der Aufgaben aus, sodass auch die Vorlesungsfolien eine gute Quelle sind, um Befehle nochmals nachzulesen. Für weiterführende Fragen können Sie selbstverständlich auch das Forum verwenden.
Gleiches gilt auch für Git. Zusätzlich ist hier auch das Git-Buch eine sehr nützliche Informationsquelle.

# 1 Unix

## Start

Konrad Klug hat bereits begonnen, seine Dateien in einem Dateisystem zu strukturieren. Damit er Ihnen seinen bisherigen Stand bereitstellen kann, müssen Sie zuerst einVerzeichnis namens `Studium` in Ihrem Home-Verzeichnis (∼) erstellen. Geben Sie danach `init-tasks` auf der Kommandozeile ein, damit Konrad seine Dateien in das gerade erstellte Verzeichnis kopieren kann.
Sollten Sie später an einen Punkt kommen, von dem Sie glauben, dass die Aufgabennicht mehr zu lösen sind, können Sie erneut `init-tasks` auf der Kommandozeile eingeben um den initialen Zustand wiederherzustellen. Um den Initialzustand einer einzelnen Aufgabe wiederherzustellen folgen Sie der Anweisung zu Beginn jeder Aufgabe. Es empfiehlt sich also bereits benutzte Befehle auf einem Blatt Papier mitzuschreiben, damit Sie Ihren Fortschritt reproduzieren können. Um Ihren Fortschritt zu testen, geben Sie auf der Kommandozeile `init-check` ein.

**Hinweis:** Die `init`-Befehle sind keine Unix-Befehle. Diese wurden lediglich von uns für Sie hinzugefügt, um Ihnen das Bearbeiten der folgenden Aufgaben zu erleichtern.

### Aufgabe 4.0: Dateien finden und schreiben

Um nur den Fortschritt dieses Teils zurückzusetzen, geben Sie auf der Kommandozeile `init-task1` ein.

a) Lassen Sie sich nun die Inhalte von `Studium` anzeigen. Wechseln Sie in den Ordner `Studium`. Konrad garantiert Ihnen, dass er dort 3 Ordner abgelegt hat. Wie viele Ordner sehen Sie? Können Sie sich das erklären? Wie können Sie sich wirklich alles in einem Ordner anzeigen lassen? Benennen Sie nun den dritten Ordner in `Gefunden` um.

b) Wechseln Sie in das Verzeichnis `Verschiedenes`. Wenn Sie die Inhalte betrachten, werden Sie feststellen, dass sich hier sehr viele Dateien befinden. Konrad erklärt Ihnen, dass er die `.log` Dateien eigentlich gar nicht haben wollte. Deswegen sollten Sie diese zunächst einmal löschen. Überprüfen Sie, ob sich nur noch `.txt` Dateien oder Dateien in deren Namen `fehler` enthalten ist in diesem Verzeichnis befinden. Können Sie auf alle Dateien mit der Dateiendung `.txt` auf einmal zugreifen?

c) Konrad sagt zudem, dass Dateien in `Verschiedenes` mit `fehler` im Namen fehlerhaft sind und nicht existieren sollten. Prüfen Sie, ob es solche Dateien gibt und löschen Sie diese gegebenenfalls. Ihr Verzeichnis sollte zu

diesem Zeitpunkt nur noch Dateien enthalten, deren Namen ausschließlich aus Ziffern bestehen und die Endung `.txt` besitzen. Hängen Sie die Dateien `000.txt`, `001.txt`, ... aneinander, entsteht der Text, den Konrad erhalten wollte. Schreiben Sie diesen Text in eine Datei namens `ergebnis.txt` im Ordner `Verschiedenes`. Löschen Sie alle verbliebenen Dateien außer `ergebnis.txt`. Lassen Sie sich den Inhalt von `ergebnis.txt` anzeigen.

## Lösung

a) In `Studium` wechseln:

```
cd Studium
```

Gesamten Inhalt des Ordners anzeigen:

```
ls -a
```

Umbenennen:

```
mv .Versteckt Gefunden
```

b) Ordner wechseln:

```
cd Verschiedenes
```

Alle log-Dateien löschen:

```
rm *.log
```

Dateine zum Überprüfen anzeigen:

```
ls
```

fehlerhafte Dateien löschen:

```
rm *fehler*
```

c) Dateien aneinanderhängen:

```
cat *.txt > ergebnis.txt
```

Dateien außer Ergebnis löschen:

```
rm 0*.txt
```

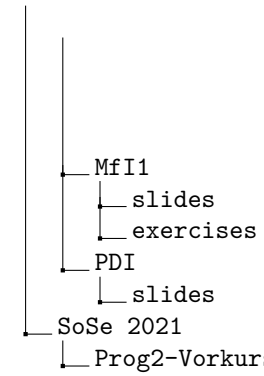
Inhalt zeigen:

```
cat ergebnis.txt
```

## Aufgabe 4.1: Ordner erstellen und Dateien verschieben

Um nur den Fortschritt dieses Teils zurückzusetzen, geben Sie auf der Kommandozeile `init-task2` ein.
Wechseln Sie nun in das Verzeichnis `Unterlagen`, das in `Studium` liegt. In diesem Verzeichnis möchte Konrad all seine Unterlagen zu den verschiedenen Vorlesungen, welche er im Laufe seines Studiums hören wird oder bereits gehört hat, abspeichern. Dafür hat er sich die folgende Ordnerhierachie ausgedacht:

```
Unterlagen
└── WiSe 2020
    └── Prog1
        ├── slides
        ├── exercises
        └── solutions
```




Erstellen Sie zunächst alle benötigten Ordner und verschieben Sie anschließend die Dateien, die momentan noch im Verzeichnis `Unterlagen` liegen, passend. Können Sie einfach die Leerzeichen in den Namen auch in der Kommandozeile übernehmen? Was passiert in diesem Fall?

**Beispiel:**

`~/Studium/Unterlagen/MfI1-slides-2.pdf`

soll verschoben werden nach

`~/Studium/Unterlagen/WiSe\ 2020/MfI1/slides/MfI1-slides-2.pdf`

### Challenge

Konrad Klug möchte nun alle Unterlagen so umbenennen, dass die Dateinamen keine Informationen seiner Verzeichnisstruktur wiederholen. Versuchen Sie die Dateien mit Hilfe von Schleifen umzubenennen.

**Beispiel:**

`~/Studium/Unterlagen/MfI1-slides-2.pdf`

soll verschoben werden nach

`~/Studium/Unterlagen/WiSe\ 2020/MfI1/slides/2.pdf`

### Lösung

Wörter (durch Leerzeichen getrennte Zeichenketten) werden als einzelne Argumente des Befehles interpretiert. `mkdir WiSe 2020` erstellt also 2 Ordner `WiSe` und `2020`. Um 1 Ordner mit diesem Namen zu erstellen, müssen wir entweder das Leerzeichen *escapen*, `mkdir WiSe\ 2020`, oder *Quotation* verwenden, `mkdir "WiSe 2020"`.
Die folgenden Befehle sind notwendig, um die Ordnerstruktur zu erstellen:

```
mkdir -p WiSe\ 2020/Prog1/{slides,exercises,solutions}
mkdir -p WiSe\ 2020/MfI1/{slides,exercises}
mkdir -p WiSe\ 2020/PDI/slides
mkdir -p SoSe\ 2021/Prog2-Vorkurs
mv Prog1-ex* WiSe\ 2020/Prog1/exercises
mv Prog1-sol* WiSe\ 2020/Prog1/solutions
mv Prog1-sl* WiSe\ 2020/Prog1/slides
mv MfI1-ex* WiSe\ 2020/MfI1/exercises
mv MfI1-sl* WiSe\ 2020/MfI1/slides
mv PDI-sl* WiSe\ 2020/PDI/slides
```

### Challenge

`TODO`

## 2 Git

### Start

Im Folgenden werden Sie auf einem kleinen Test-Repository arbeiten, um sich mit wichtigen Operationen zur Versionsverwaltung mit Git vertraut zu machen. Bevor Sie mit den Aufgaben beginnen, sollten Sie Git konfigurieren.

Dies ist wichtig, da sonst einige der Aufgaben nicht umzusetzen sind. Führen Sie zur Konfiguration die beiden folgenden Befehle aus:

```
git config --global user.name "Vorname Nachname"
git config --global user.email "kennung@stud.uni-saarland.de"
```

Dabei ersetzen Sie `Vorname` und `Nachname` durch Ihren Vor- bzw. Nachnamen und `kennung` durch Ihre studentische Kennung. Bei diesem Projekt arbeiten Sie unter den gleichen Bedingungen wie bei den Projekten in Programmierung 2. Das heißt: sobald Sie Ihre Änderungen über Git hochladen,werden automatisierte Tests laufen und Sie erhalten Feedback über den Stand Ihres Projektes. Nun werden Sie auf dem Test-Repository arbeiten.

#### Klon beziehen

Klonen Sie das Repository

```
https://prog2scm.cdl.uni-saarland.de/git/guss/$NAME
```

wobei `$NAME` durch Ihren Benutzernamen im *cms* zu ersetzen ist. Geben Sie an, dass der Klon in ein Verzeichnis namens `gittutorial` abgelegt wird.

#### Uni VPN

Der Server ist nur aus dem Uni VPN zu erreichen. Das heiSSt, dass Sie das Projekt nur klonen und abgeben können, wenn Sie mit dem Uninetzwerk verbunden sind. Gleiches gilt auch für die kommenden Prog2 Projekte. Eine Erklärung, wie Sie dies einrichten, finden Sie hier.

## Aufgabe 4.2: Versionsverwaltung mit Git

In dieser Aufgabe benötigen Sie die Grundlegenden Befehle in Git, die Sie auch auf den Vorlesungsslides finden.

a) **Zeile hinzufügen:** Das Repository enthält bereits eine Datei `hallo.txt`. Fügen Sie an diese eine weitere Zeile mit dem Inhalt `Ciao!` an.

b) **Änderungen anzeigen:** Lassen Sie sich die Änderungen anzeigen.

c) **Änderungen vormerken:** Merken Sie die Änderung zum Einbuchen, also dem Hochladen auf dem Server, vor.

d) **Weitere Zeile hinzufügen:** Fügen Sie eine weitere Zeile mit dem Inhalt `Test` an die Datei an.

e) **Änderungen erneut anzeigen:** Lassen Sie sich diese Änderung anzeigen. Können Sie sich diese Änderungen auch gemeinsam anzeigen lassen?

f) **Vorgemerkte Änderungen anzeigen:** Lassen Sie sich die bereits vorgemerkten Änderungen anzeigen.

g) **Zustand der Arbeitskopie:** Lassen Sie sich den Zustand der Arbeitskopie anzeigen.

h) **Letzte Änderung verwerfen:** Verwerfen Sie die letzte Änderung, jedoch nicht die zuvor vorgemerkten.

i) **Änderungen einbuchen:** Buchen Sie die vorgemerkten Änderung mit der Nachricht `'Sag auch "Ciao!"'` ein.

j) **Neue Datei:** Erzeugen Sie eine weitere Datei `test.txt` mit dem Inhalt `Test`. Stellen Sie diese Dateiunter Versionsverwaltung und buchen Sie die Änderung ein.

k) **Zeile ändern:** Ändern Sie die erste Zeile in der Datei `hallo.txt` zu `Hallo, Welt!`.

l) **Änderungen abgeben:** Buchen Sie die neuen Änderungen erneut ein. Geben Sie danach ihre Änderungen ab. Dies bedeutet, dass sie den jetzigen Zustand des Repositories auf unseren Server hochladen. Die zuletzt hochgeladene Version vor der Deadline entspricht Ihrer Abgabe, die getestet wird.

## Lösung

Das Projekt klonen können Sie durch:

```
git clone https://prog2scm.cdl.uni-saarland.de/git/guss/$NAME gittutorial
```

wobei `$NAME` durch Ihren Benutzernamen im *cms* zu ersetzen ist.

a) entweder mit Hilfe von *nano* oder

```
echo "Ciao!" >> hallo.txt
```

b)
```
git diff
```

c)
```
git add .
```

d)
```
echo 'Test' >> hallo.txt
```

e)
```
git diff @
```

f)
```
git diff --cached
```

g)
```
git status
```

h)
```
git checkout .
```

i)
```
git commit -m 'Sag auch "Ciao"!'
```

j)
```
echo Test > test.txt
git add .
git commit -m "Füge Test-Datei hinzu."
```

k)
```
echo "Hallo, Welt!" > hallo.txt
```

l)
```
git commit -a -m 'Begrüße die Welt.'
git push
```