Artemis und Git

Einführung in die Programmierung

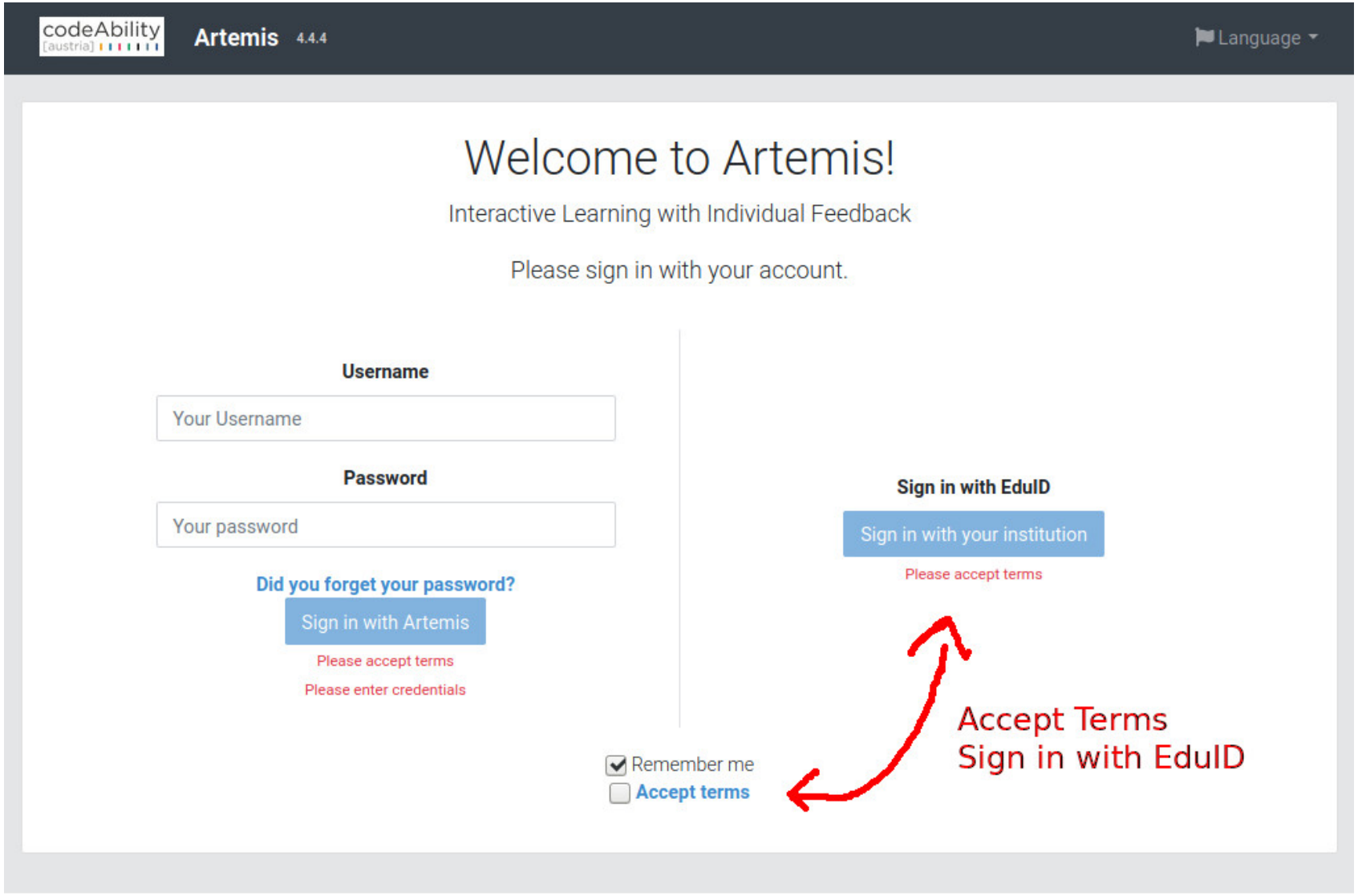
Paris Lodron Universität Salzburg

WS 2020

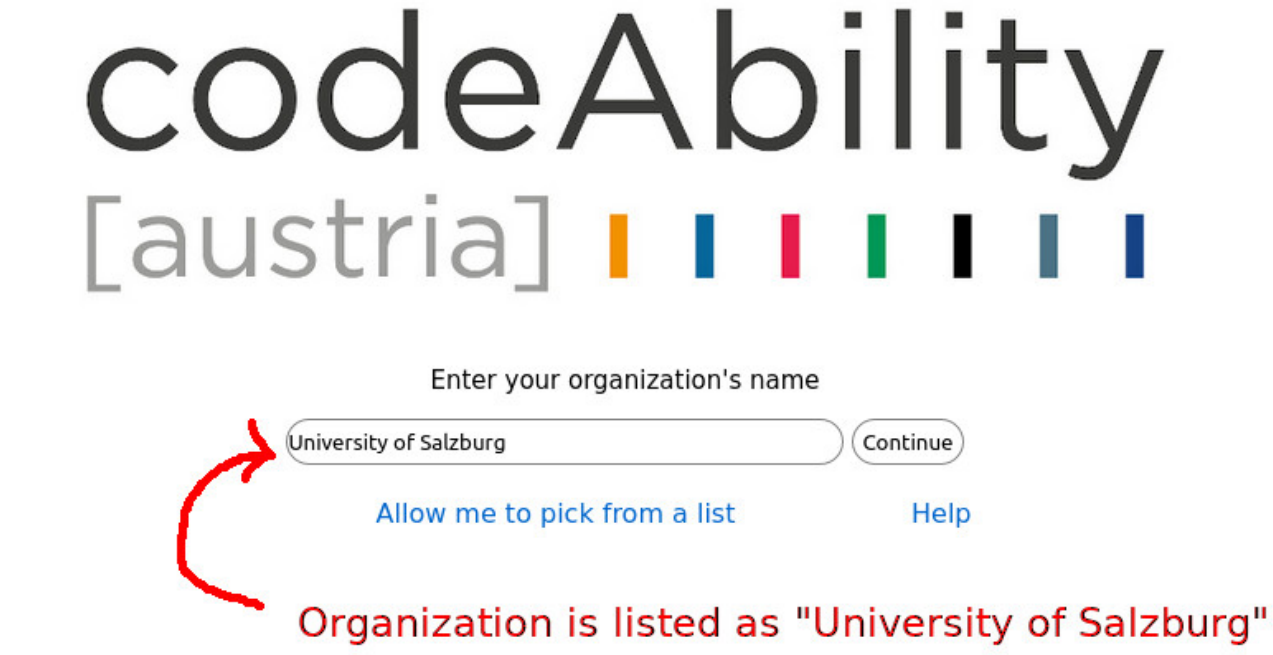
Abgabeserver

- Aufgaben werden am Artemis server abgegeben.
- Zuerst einloggen und den Kurs abonnieren.
- Abgaben werden automatisch ausgewertet und Probleme werden angezeigt.
- Zwei Möglichkeiten die Aufgabe zu lösen:
  - Online Editor im Artemis System.
    Direktes Arbeiten am Test Server, jedes Speichern führt Tests aus, dadurch langsamer.
  - Offline Arbeiten (mit Git) und nur fertige Aufgaben abgeben.
    Bessere Sicht auf Fehler und einfacheres/schnelleres Entwickeln.
    Extra Schritt mit Git (einfaches Tutorial folgt)

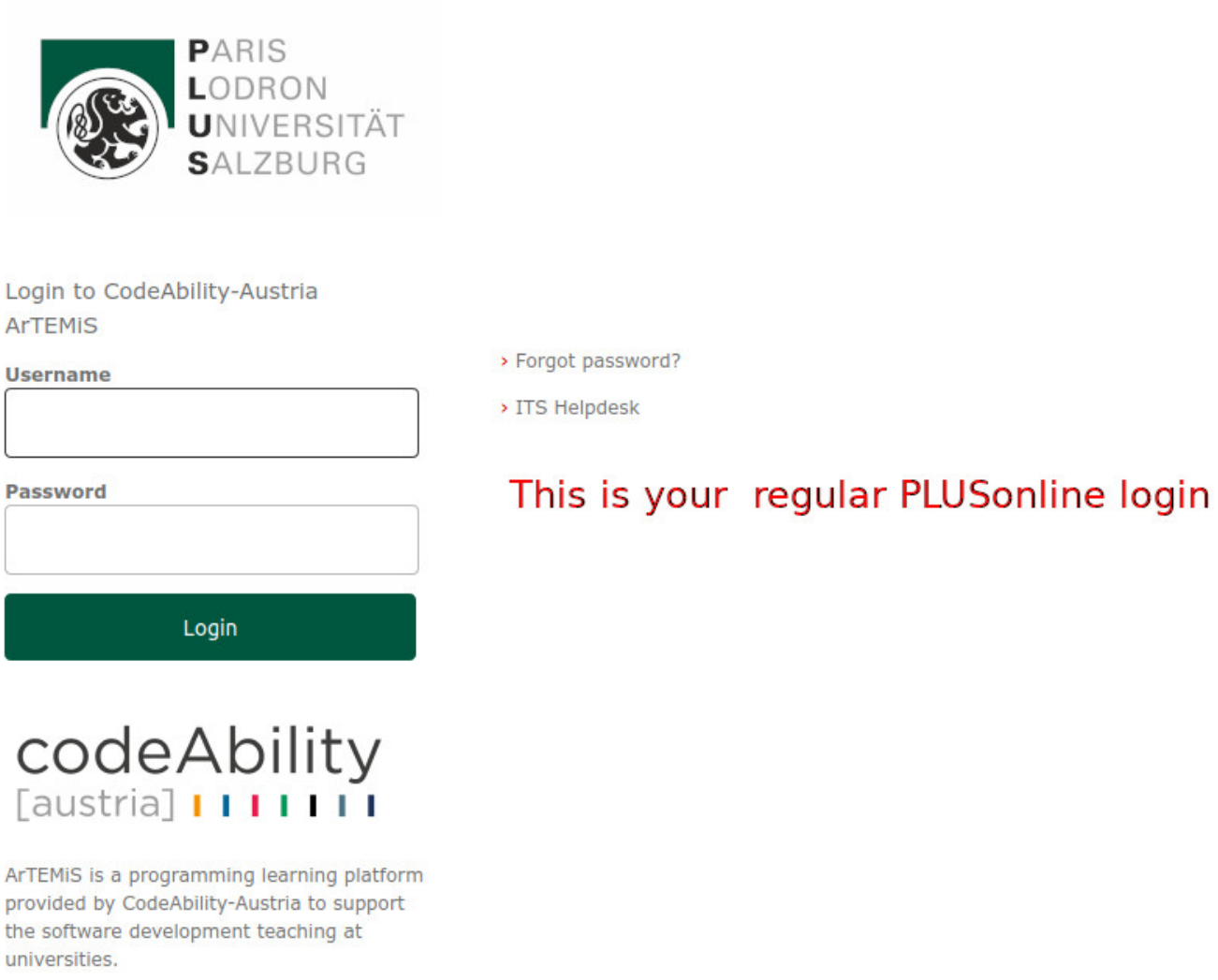
Artemis I


codeAbility [austria]
Artemis 4.4.4
Language
Welcome to Artemis!
Interactive Learning with Individual Feedback
Please sign in with your account.
Username
Your Username
Password
Your password
Did you forget your password?
Sign in with Artemis
Please accept terms
Please enter credentials
Remember me
Accept terms
Sign in with EduID
Sign in with your institution
Please accept terms
Accept Terms
Sign in with EduID
Report a bug
Request a feature
Contact us
Release Notes
Privacy Statement
Imprint

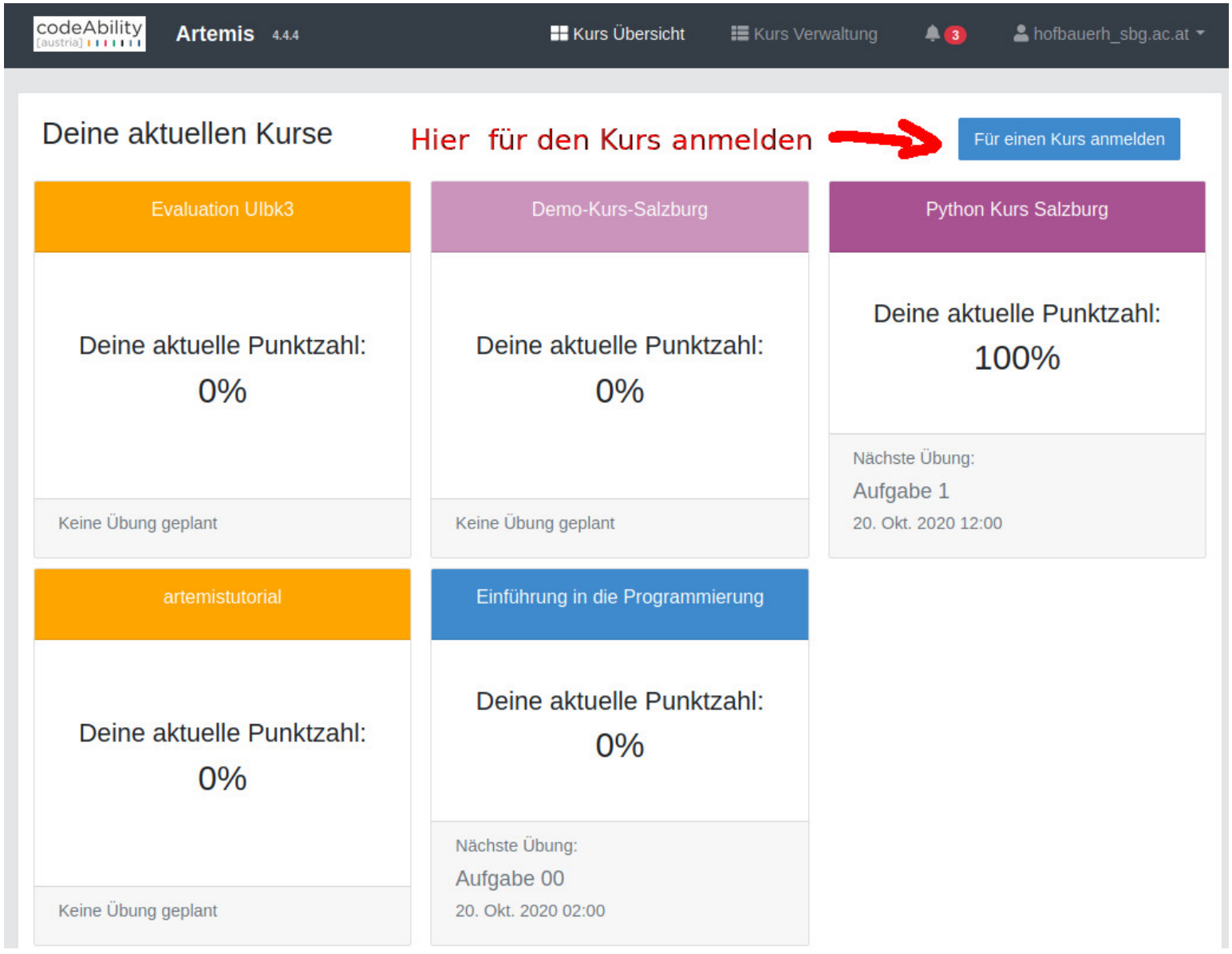
Artemis II


codeAbility
[austria]
Enter your organization's name
University of Salzburg
Continue
Allow me to pick from a list
Help
Organization is listed as "University of Salzburg"

Artemis III


PARIS
LODRON
UNIVERSITÄT
SALZBURG
Login to CodeAbility-Austria
ArTEMiS
Username
Password
Login
› Forgot password?
› ITS Helpdesk
This is your regular PLUSonline login
codeAbility
[austria]
ArTEMiS is a programming learning platform provided by CodeAbility-Austria to support the software development teaching at universities.

Artemis IV


codeAbility [austria]
Artemis 4.4.4
Kurs Übersicht
Kurs Verwaltung
3
hofbauerh_sbg.ac.at
Deine aktuellen Kurse
Hier für den Kurs anmelden
Für einen Kurs anmelden
Evaluation UIbk3
Deine aktuelle Punktzahl:
0%
Keine Übung geplant
Demo-Kurs-Salzburg
Deine aktuelle Punktzahl:
0%
Keine Übung geplant
Python Kurs Salzburg
Deine aktuelle Punktzahl:
100%
Nächste Übung:
Aufgabe 1
20. Okt. 2020 12:00
artemistutorial
Deine aktuelle Punktzahl:
0%
Keine Übung geplant
Einführung in die Programmierung
Deine aktuelle Punktzahl:
0%
Nächste Übung:
Aufgabe 00
20. Okt. 2020 02:00

Artemis V

Für die Arbeit mit Git die URL von dem Repository kopieren.

Im weiteren wird auf diese URL mit `REPO_URL` referenziert, also in Befehlen anstelle von `REPO_URL` die tatsächliche URL des Repositories kopieren!

Git I

Woher nehmen?

Windows 10 Download und Installation per Hand. Mehr Information dazu unter gitforwindows.org.

Linux Hat jedes Linux im package manager.
Debian basiert (Ubuntu etc.): `apt install git-all`
Fedora basiert (RHEL, CentOS): `dnf install git-all`

MacOS Sollte installiert sein, einfach in einer Shell `git --version` eingeben. Falls git nicht installiert ist sollte die Option zur Installation erscheinen. Alternativen sind

- In einer Shell: `brew install git`
- Download und Installation per Hand (sollte von allen Optionen in der neuesten Version von git resultieren).

Git II

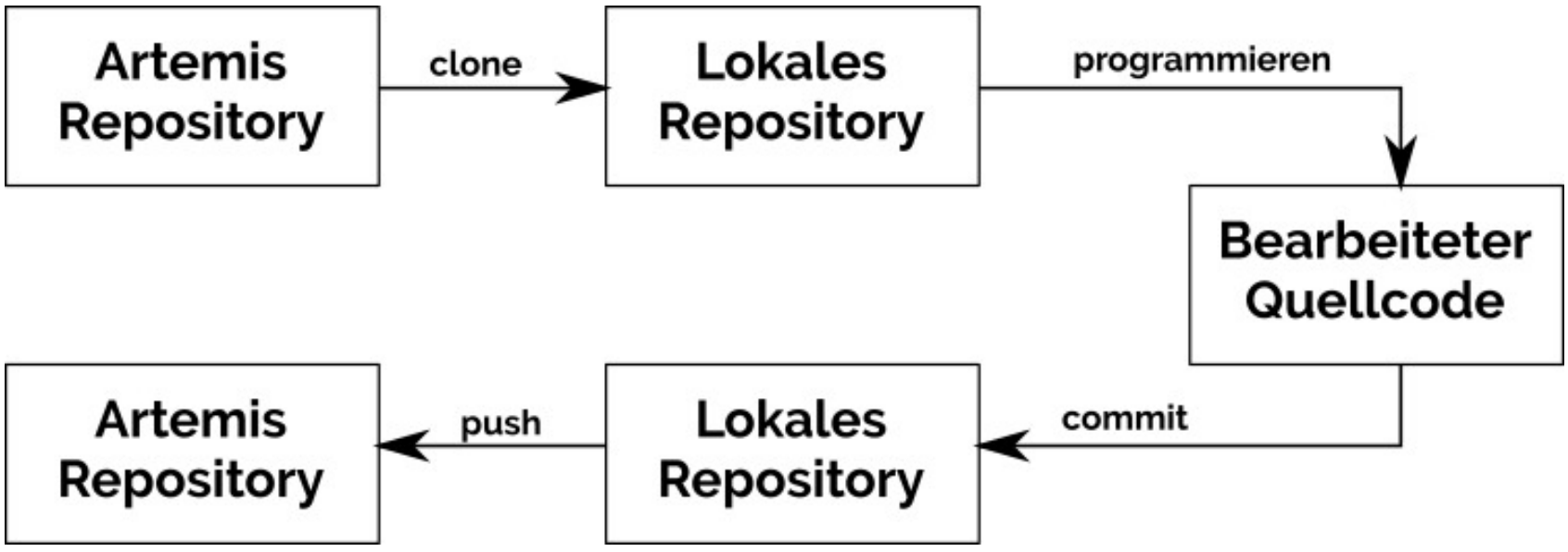
Was *ist* beziehungsweise *tut* git eigentlich?

Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien. Git kann recht viel, aber das meiste davon brauchen wir in diesem Kurs nicht. Es folgt also eine *sehr* kurze, *sehr* vereinfachte Einführung in die Befehle die wir brauchen.

Doch vorher kurz Idee hinter Git:

Der Quellcode liegt verteilt herum (z.B. am Artemis Server und lokal bei Ihnen auf der Festplatte), man kann in jedem dieser 'Repositories' arbeiten: Quellcode erstellen und löschen, Änderungen behalten oder verwerfen, und so weiter.

Git III



Für uns heißt das, wir holen (Klonen) uns das Repostiory vom Artemis Server. Dann arbeiten wir lokal, auf unserem Rechner, und können dort Änderungen einbringen. Wenn wir fertig sind können wir unser lokales Repository auf den Server schieben, dort werden dann automatisch die Abgabetests ausgeführt und Sie erhalten eine Rückmeldung.

Git IV

`git clone REPO_URL [Verzeichnis]`

Mittels `git clone` wird das Repository vom Server auf das lokale Gerät kopiert, und zwar in ein Verzeichnis das dem letzten Teil der URL entspricht (z.B.: `eidplusaufgabe1-hofbauerh_sbg.ac.at`).

Optional (der `[Verzeichnis]` Teil des obigen Befehls) kann man auch selber ein Verzeichnis angeben.

Beispielsweise:

```
git clone REPO_URL Aufgabe1
```

Würde das Repository in das Verzeichnis `Aufgabe1` klonen.

In diesem Verzeichnis liegt nun der Quellcode mit dem man arbeiten kann.

Wichtig!

Die folgenden git Befehle funktionieren nur in dem Verzeichnis in welches das Repository geklont wurde!

Git VI

Wie vermutlich aufgefallen ist wird beim Klonen nach einer Autorisierung gefragt. Diese ist jedes Mal notwendig wenn mit Server interagiert wird!

Damit man nicht jedes Mal Benutzernamen und Passwort eingeben muß gibt es folgenden Befehl:

```
git config credential.helper 'cache --timeout=3600'
```

Der Befehlt veranlasst das sich git die *credentials* (Benutzer/Passwort) merkt, und zwar für die in *timeout* angegeben Zahl in Sekunden (3600 Sekunden sind eine Stunde). Die Art wie sich git die *credentials* merken soll ist *cache*, also im Speicher.

Das Passwort wird also nie im Klartext auf die Festplatte geschrieben und nach einer Stunde auf jeden Fall verworfen.

Git VII

`git commit -a`

Wurden Änderungen am Quellcode gemacht und sollen jetzt in das Lokale Repository aufgenommen werden geht das mit `commit -a`. Dabei werden alle Änderungen übernommen (das ist die Bedeutung des `-a`).

Git ist ein Versionskontrollprogram und behält sich auch alte Änderungen auf. Damit man sich in der Vergangenheit des Quellcodes zurechtfindet wird nach einen 'Commit Message' gefragt. Das ist für uns unbedeutend und kann ignoriert werden (also einfach leer lassen).

Wer vermeiden will das jedesmal nach einer Beschreibung der Änderung gefragt wird kann einfach folgenden Befehl verwenden:

```
git commit -a -m '.'
```

Hier wird als Nachricht ein Punkt ('.') mitgegeben (`-m '.'`), leere `commit` Nachrichten sind eigentlich nicht elegant aber hier als Vereinfachung erlaubt.

Git VIII

`git push`

Ist man fertig und möchte eine Abgabe machen wird das lokale Repository (in das eventuelle Änderungen mit `git commit` übernommen wurden) auf den Server geschoben. Dafür verwenden wir `git push`. Es wird wieder nach den *credentials* gefragt, ebenso wie bei `git clone` und die Anmerkungen zum Speichern derselben gelten auch hier.

Nach dem `git push` werden am Server automatisch die Tests gestartet. **Die Ergebnisse der Tests sind am Server zu sehen**.

Wer bin ich?

`git push` wird beim ersten mal nichts tun, sondern sich aufregen das es nicht weis wer der Benutzer ist. Git ist ja eigentlich da um die Entwicklung zwischen mehreren Benutzern zu vereinfachen. Man muss also einen Benutzername und eine Emailadresse angeben:

```
git config --global user.name ”Dein Name”
git config --global user.email ”Deine EMail”
```

Das kann pro Repository gemacht werden (ohne `--global`) oder für global für alle (damit bei zukünftigen Aufgaben nicht wieder nachgefragt wird) wie oben.

Lokale überschreiben Globale Einstellungen, sollte das mal von Bedeutung sein.

`git mv Vorher Nachher`

Mit dem Befehl `git mv` können Dateien verschoben werden. Der wichtige Teil hier ist das damit auch die Umbenennung von Dateien möglich ist ohne `git` zu verwirren.

`git mv Welt.java Bsp00.java` benennt zum Beispiel die Datei `Welt.java` auf `Bsp00.java` um. Es ist wichtig dies mit `git mv` zu tun da git sonst nicht weis welche Datei bei einem `git commit -a` ins lokale Repository gesteckt werden soll.

Git XI

Zusammenfassend, und in kurzer Version:

1. Die Option zum Speichern der *credentials* Repository übergreifend (`--global`) aktivieren:
   `git config --global credential.helper 'cache --timeout=3600'`
2. Name und Email festlegen:
   `git config --global user.name "Dein Name"`
   `git config --global user.email "Deine EMail"`
3. Das Repository der Aufgabe klonen: `git clone REPO_URL`
4. Lokal programmieren und die Aufgabe lösen.
5. Das Programm in das lokale Repository übernehmen:
   `git commit -a -m '.'`
6. Das lokale Repository auf den Server kopieren:
   `git push`
7. Am Server die Fehlerfreiheit oder die Fehlermeldung nachlesen.

Hilfe was ist ein Terminal (eine Shell) und wie arbeitet man damit?

Das Terminal ist eine Programm das Zugriff auf die Kommandozeile erlaubt. Eine Shell ist das eigentliche Programm für das das Terminal quasi die ‘GUI’ ist.

Für das arbeiten mit der Shell gibt es verschiedene Tutorials auf die hier verwiesen sei:

- Windows DOS
- Linux BASH
- MacOS ist ein aufgebohrtes Linux also das sollte das BASH Tutorial passen.

Die GitBASH (windows git interface) ist an die BASH angelehnt (wie der Name schon sagt) und damit passt auch das Linux BASH Tutorial.

Mehr Information zu Git

Entweder ein Online Tutorial lesen (Beispiel) oder die Git interne Hilfe verwenden.

Mit `git help BEFEHL` bekommt man **alle** Information zu einem Befehl, z.B.: `git help push`. Das ganze ist sehr ausführlich und eventuell nicht für Anfänger zu empfehlen.

Git kommt allerdings mit einem Tutorial das man sich durchaus zu Gemüte führen kann:

```
git help tutorial
```

Online Editor *und* Git verwenden I

`git clone` dient dazu ein Repository das wir nicht haben zu uns zu holen. Wenn wir das Repo lokal haben können wir mit `git push` eingecheckte (via `git commit`) Änderungen auf das 'remote' Repository schieben.

Das ist der Anwendungsfall wir editieren bei uns und wollen die Änderungen am Server haben.

Was nun wenn man im Onlineeditor etwas verändert hat und drauf kommt da gibt es mehr zu tun. Lokal arbeiten ist da komfortabler (besserer Editor und so). Man kann mit `git pull` Änderungen aus dem 'remote' Repository nach lokal kopieren.

Kurz also:

`git push` Schiebt Änderungen von lokal nach remote.
`git pull` Holt Änderungen von remote nach lokal.

Online Editor *und* Git verwenden II

Achtung

Ändert man lokal etwas **und** remote und will dann einen Abgleich schaffen kann es zu Konflikten kommen (wenn etwa die gleiche Zeile geändert worden ist). Manchmal können diese automatisch repariert werden, etwa wenn die Änderungen nicht den gleichen Codeteil betreffen. Aber das ist nicht garantiert und im Zweifelsfall muss man das selber reparieren, dafür ist diese Minieinführung nicht gedacht, siehe `git help merge` und oder Git Tutorials.

Am besten, bevor etwas verändert wird einen Abgleich schaffen.

- Bevor der Onlineeditor gestartet wird lokal `git push`.
- Bevor lokal editiert wird `git pull`.