# Teil 1

## 1.1. Checke dein Git-Projektarchiv aus

* git@gitlab.fh-ooe.at:sib/ebd/SXXXXXXXXXX.git

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* *git clone* *git@gitlab.fh-ooe.at:sib/ebd/S2310239008.git  
  git clone* *<URL>* ist ein Befehl, um eine lokale Kopie eines Remote Git-Repositories zu erstellen. Dabei wird ein neues Verzeichnis erstellt und die Git-Metadaten angelegt. Unterstützt werden die Protokolle SSH und HTTPS.

## 1.2. Lege eine neue Datei „archimedes.txt“ an und übernimm diese in das zentrale Projektarchiv

* Verwende die Commit-Nachricht: git-1.2

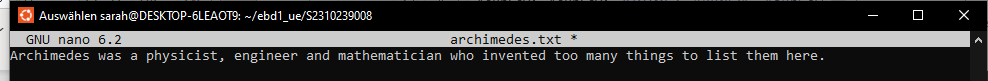
Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* *touch archimedes.txt*Der Befehl touch <neuer\_dateiname.endung> erstellt eine neue Datei, falls sie noch nicht existiert. Ansonsten wird einfach ihr Zeitstempel aktualisiert.
* *git status*  
  Zeigt den aktuellen Status von Arbeitsverzeichnis und Staging Area im Vergleich zum letzten Commit (HEAD). Er informiert darüber was gelöscht/modifiziert/hinzugefügt wurde und was sich in der Staging Area befindet und was nicht.
* *git add .*  
  Mittels *git add <Datei/Verzeichnisname>* werden die Änderungen an den angegebenen Dateien und Verzeichnissen in die Staging Area übernommen. Mit dem Punkt statt Datei/Verzeichnisnamen werden einfach alle Änderungen übernommen.
* *git commit -m „git-1.2“*  
  Der Befehl *git commit* wird verwendet, um Änderungen in einem lokalen Repository, welche sich in der Staging Area befinden, zu einem neuen Commit zu machen. Ein Commit repräsentiert eine bestimmte Version des Projekts zu einem bestimmten Zeitpunkt.   
  Auf diese Versionen kann jederzeit wieder zurückgegriffen werden.  
  Mit -m kann direkt eine Message übergeben werden, welche Informationen zur Version enthält. Wird dies nicht gemacht, öffnet sich ein Editor und die Nachricht muss nachträglich eingegeben werden.
* *git push*  
  Die Änderungen (Commits) des aktiven Branches werden auf das zentrale Projektarchiv hochgeladen. Dabei wird standardmäßig das Projektarchiv verwendet, von dem gecloned wurde (Origin). Möchte man nicht den aktiven Branch, in dem man sich befindet pushen, oder in ein anderes Archiv pushen, kann dies im Befehl noch genauer spezifiziert werden mittels *git push <remote-name> <branch-name>.*

## 1.3. Erweitere die Datei „archimedes.txt“ um folgenden Inhalt und übernehme die Änderung in das zentrale Projektarchiv

* “Archimedes was a physicist, engineer and mathematician who invented too many things to list them here.”
* Verwende die Commit-Nachricht: git-1.3



Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* *nano archimedes.txt*  
  Wenn man nano mit dem Namen einer Datei aufruft, wird die Datei im Nano Editor geöffnet. Wenn die Datei nicht vorhanden ist, wird sie erstellt und geöffnet.

# Teil 2

## 2.1. Erstelle einen Branch mit deiner S-Nummer (SXXXXXXXXXX) als Namen und wechsle hinein

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

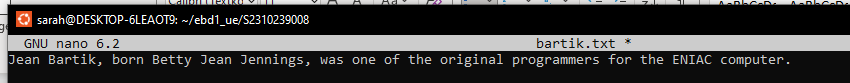
* *git checkout -b S2310239008*  
  Erstellt einen neuen Branch und wechselt sofort in diesen. Ebenso könnte der Branch mit dem Befehl *git branch <branchname>* erstellt werden, dann müsste aber manuell mit *git checkout <branchname>* in den Branch gewechselt werden.
* *git branch*Zeig eine Liste aller im lokalen Repository vorhandenen Branches an. Der aktive Branch (der in dem aktuell gearbeitet wird) wir mit einem Stern (\*) markiert.

## 2.2. In dem neuen Branch; erstelle eine neue Datei „bartik.txt“ mit dem unten stehenden Inhalt und übernimm es in das zentrale Projektarchiv

* Setze den neuen Branch als upstream Branch
* “Jean Bartik, born Betty Jean Jennings, was one of the original programmers for the ENIAC computer.”
* Verwende die Commit-Nachricht: git-2.2

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung



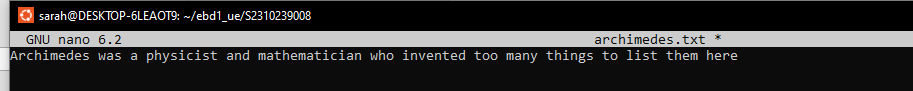
Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

* *git push –set-upstream origin S2310239008*- *set-upstream*.. "Upstream" wird beim ersten Mal pushen eines neuen Branches benötigt, damit Git weiß, welcher Remote-Branch mit diesem lokalen Branch in Verbindung steht.  
  - *origin*.. Das Remote Repository von dem gecloned wurde  
  - *S2310239008.*. Name des neuen Branches

## 2.3. Erweitere die Datei „archimedes.txt“ um folgenden Inhalt und übernehme die Änderung in das zentrale Projektarchiv

* “Archimedes was a **physicist and mathematician** who invented too many things to list them here.”
* Verwende die Commit-Nachricht: git-2.3



Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung

# Teil 3

* Wechsle zurück in den Main-Branch

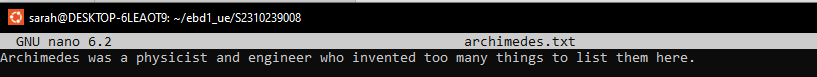
Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* *git checkout master*  
  Mit *checkout <branchname>* kann in einen anderen Branch gewechselt werden

## 3.1. Erweitere die Datei „archimedes.txt“ um folgenden Inhalt und übernehme die Änderung in das zentrale Projektarchiv

* “Archimedes was a **physicist and engineer** who invented too many things to list them here.”
* Verwende die Commit-Nachricht: git-3.1



Ein Bild, das Text, Elektronik, Screenshot, Schrift enthält.

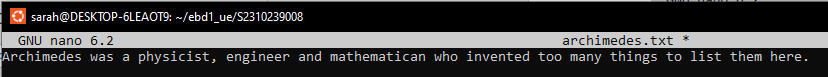
Automatisch generierte Beschreibung

## 3.2. Wechsle zurück in den Branch mit deiner S-Nummer und merge die Änderungen vom main-Branch in deinen Branch. Löse den auftreten Merge-Konflikt zu folgendem Text

* “Archimedes was a **physicist, engineer and mathematician** who invented too many things to list them here.”
* Verwende die Commit-Nachricht: git-3.2

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung



Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* *git merge master*  
  Gibt man im aktiven Branch *git merge <quelle-branch>* ein, so wird der aktuelle Stand des Quell-Branches in den aktiven Branch integriert. Sofern in den beiden Branches nur unterschiedliche Dateien, oder unterschiedliche Stellen in derselben Datei bearbeitet wurden, sollte dies im Normalfall automatisch ohne Probleme funktionieren. Wenn, wie in unserem Fall, dieselbe Datei an derselben Stelle bearbeitet wurde, so gibt es einen Merge Conflict und man muss sich händisch für eine der beiden Änderungen entscheiden – oder etwas anderes hinschreiben. Diese Manuelle Lösung wird dann commited und der Merge Conflict ist gelöst.

## 3.3. Lass dir die Log-Einträge anzeigen. Füge den Log-Output in die Textdatei „log.txt“ ein und lade diese in das zentrale Projektarchiv hoch (in dem Branch mit deiner S-Nummer)

* Verwende die Commit-Nachricht: git-3.3

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung

* *git log*  
  Gibt die Commit-Historie eines Git-Repositories aus, beginnend mit dem neuesten Commit und rückwärts in die Vergangenheit.  
  Folgende Informationen über jeden Commit werden ausgegeben:
  + Commit-ID (SHA-1-Hash)
  + Autor
  + Datum und Uhrzeit des Commits
  + Commit-Nachricht

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* *git log > log.txt*  
  Mit dem Größer-Zeichen wird der Output von git log zum Input für eine neue Datei namens log.txt
* *cat log.txt*  
  Schreibt den Inhalt von log.txt auf die Standardausgabe (Terminal)

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

Automatisch generierte Beschreibung