**Thema, Ziele:** Git branch, merge und remote Repos, Code Dokumentieren

## Aufgabe 1: clone Repo

Erstellen sie auf dem HSR Git-Portal ein entferntes Repo. Öffnen Sie einen Internetbrowser und geben Sie die URL https://git.hsr.ch ein. Loggen Sie sich mit dem HSR Login ein. Wählen im linken Menü den Eintrag „Repositories“ aus. Nun können Sie mittels des „Hinzufügen“-Buttons ein neues Repo auf dem HSR Git Server erstellen. Folgende Informationen können/müssen angegeben werden:

| Feld | Beschreibung |
|---|---|
| Name | Name des Git Repos. Mit `git clone` wird automatisch ein Ordner mit diesem Namen im aktuellen Verzeichnis erstellt. |
| Typ | Hier soll Git ausgewählt werden. |
| Kontakt | Geben Sie hier Ihre HSR E-Mail Adresse an. |
| Beschreibung | Hier wäre eine kurze und prägnante Beschreibung ihres Projekts / Repos sinnvoll. |
| Öffentlich | Ist „Öffentlich“ markiert, kann dieses Repo von jeder Person mit Zugriff auf git.hsr.ch gelesen werden. |

Nachdem ein Repository auf dem HSR Git-Server erstellt wurde, kann dieses mittels `git clone` mit einem lokalen Repo verknüpft werden. Dieses lokale Repo wird ebenfalls durch den `git clone` Befehl erstellt.

Hinweis:
Gemäss den Vorgaben der IT-Servicedesks müssen die Repos nach Beendigung des Projekts gelöscht werden. In diesem Falle wäre es sinnvoll, wenn Sie die in diesem Praktikum erstellten Repos nach Beendigung des Praktikums löschen würden.

## Aufgabe 2: push

Für diese Aufgabe wird Aufgabe 1 vorausgesetzt.

Erstellen Sie einige Textdateien und comitten Sie diese. Anschliessend soll dieser Commit auf des remote Repo hochgeladen werden.

## Aufgabe 3: neuen Branch erstellen

Für diese Aufgabe wird Aufgabe 2 vorausgesetzt.

Auf welchem Branch haben Sie die letzten Änderungen in Aufgabe 2 committet?
Erstellen Sie einen neuen Branch mit dem Namen *featureA* und wechseln Sie auf diesen Branch.
Ändern Sie einige getrackte Dateien. Committen und pushen Sie diese anschliessend.

## Aufgabe 4: zwei verschiedene Branches zusammenführen

Für diese Aufgabe wird Aufgabe 3 vorausgesetzt.

Schritt 1:
Fügen Sie die Branches *master* und *featureA* wieder zusammen. Dabei soll der *featureA* Branch in den *master* Branch eingefügt werden.

Schritt 2:
Wechseln Sie wieder auf den featureA Branch und machen Sie ein Commit. Achten Sie darauf, dass Sie vor diesem Commit nur eine Datei verändern. Wechseln Sie auf den *master* Branch. Führen Sie nun auch auf dem *master* Branch ein Commit aus. Achten Sie darauf, dass die Änderungen nicht im gleichen Bereich gemacht werden wie auf dem *featureA* Branch (einige Zeilen Abstand). Mergen Sie anschliessend den *featureA* Brach wieder in den *master* Branch. Das Mergen der Braches sollte ohne Probleme funktionieren.

Falls nicht, sind die Änderungen der Datei bei den beiden Commits (*master* und *featureA* Branch) zu ähnlich. Brechen Sie entweder den Merge mit `git merge --abort` ab oder gehen Sie direkt zu Schritt 3 ohne den zweiten Schritt nochmals zu wiederholen.

Schritt 3:
Wiederholen Sie Schritt 2. Achten nun darauf, dass bei den jeweiligen Commits auf dem *master* und *featureA* Branch sehr ähnliche Änderungen gemacht werden, um ein Mergekonflikt zu provozieren. Beim Mergen des *featureA* Branches in den *master* Branch wird es entsprechend ein Konflikt geben. Lösen Sie den Konflikt mittels dem Git Mergetool `git mergetool`.

## Aufgabe 5: Merge Konflikt von binären Dateien

Erstellen Sie eine binäre Datei und fügen Sie diese zur Versionsverwaltung hinzu. Unter Linux können Sie z.B. mit folgendem Befehl drei Bytes mit den Werten 0x3, 0x2 und 0x1 in eine Datei schreiben:

echo -n $'\x03\x02\x01' > binary.dat

Weitere Schritte:

1. Committen Sie diese Datei.
2. Gehen Sie um den gerade erstellten zurück und erstellen Sie einen neuen Branch.
3. Erstellen Sie ebenfalls eine binäre Datei mit demselben Namen, jedoch mit anderem Inhalt als die bereits erstellte.
4. Committen Sie diese Datei.
5. Wechseln Sie auf den ursprünglichen Branch zurück und mergen Sie mit dem eben neu erstellten Branch.
6. Der Merge wird fehlschlagen. Lösen Sie diesen Konflikt, in dem Sie eine von beiden Versionen übernehmen (*ours* oder *theirs*).

## Aufgabe 6: Stash

In dieser Aufgabe sollen Änderungen zwischengespeichert werden und zu einem späteren Zeitpunkt (z.B. nach einem commit) wieder geladen werden. Die Theorie dazu wurde nicht in der Vorlesung behandelt. Informationen zum Stashen finden Sie im "Pro Git" Buch auf Seite 266.

Die folgenden Schritte geben ein mögliches Vorgehen vor:

1. Erstellen Sie zwei Dateien mit dem Namen Hallo.txt und Welt.txt.
2. Schreiben Sie irgendetwas in die Datei und committen Sie die beiden Files
3. Ändern Sie den Inhalt der Dateien Hallo.txt und Welt.txt etwas ab.
4. Stagen Sie die Datei Welt.txt
5. Zwischenspeichern Sie nun die Änderungen mittels dem *stash* Befehl
6. Erstellen Sie eine neue Datei mit dem Namen PMSwEng.txt und schreiben Sie irgendein Text in die Datei.
7. Ändern Sie die Datei Hallo.txt erneut
8. Holen Sie die *gestashten* Änderungen aus dem Zwischenspeicher zurück
   ➔ das wird nicht funktionieren
9. Sie müssen zuerst die aktuellen Änderungen committen oder verwerfen bevor die *gestashten* Änderungen zurückgeholt werden können. In diesem Beispiel sollen die Änderungen verworfen werden. Dies Betrifft nur die Datei Hallo.txt, da die Änderungen im Working Directory überschrieben würden. PMSwEng.txt ist nicht von Git getrackt.
10. Anschliessend können Sie nun die Änderungen wieder aus dem Stash-Speicher holen

## Aufgabe 7: commit - pull - push

Klonen Sie ein Repo eines anderen Praktikum-Teilnehmers. Dazu muss dieser die Rechte für Ihren Zugriff auf das entsprechende Repo hinzufügen. Dies kann ebenfalls über das HSR Git-Portal erfolgen.
Arbeiten Sie nun gemeinsam im gleichen Branch (typischerweise *master* Branch). Dieser Workflow wird in der Praxis oft angewendet und ähnelt dem Workflow mit einem zentralisierten Repository. Beachten Sie dabei die folgenden Punkte:

1. Wenn Sie Änderungen im Working Directory haben, die Sie auf das remote Repo legen möchten, ist es wichtig, dass Sie die Änderungen zuerst comitten.
2. Danach müssen allfällige Commits des remote/master Branches zuerst gefetched und gemerged werden. Dies geht am schnellsten mit dem `git pull` Befehl.
3. Anschliessend kann der lokale Commit mit dem `git push` Befehl auf das remote Repo übertragen werden.

Sprechen Sie sich mit Ihrem Arbeitspartner ab und spielen Sie die oben genannten Schritte ein paar Mal durch. Was passiert, wenn ein pull vor einem commit gemacht wird, wenn bereits Änderungen im Working Directory vorhanden sind?

## Aufgabe 8: VoltageDivider Anwendung um Dokumentation erweitern

Dokumentieren Sie Ihr Programm mit Doxygen inklusive Mainpage. Verwenden Sie dazu die VoltageDivider Anwendungs-Lösung vom letzten Praktikum.