Übungsblatt 7: Versionskontrolle mit Git

Wissenschaftliches Schreiben und Arbeiten, WiSe 2023/'24

Aufgabe A: Git installieren

1. Installiere Git auf deinem PC (falls noch nicht vorhanden).

- **Unter Windows:**
  Download von **https://git-scm.com/**, dann Installationsprogramm ausführen
  (empfohlen: Git Bash mitinstallieren!)
- **Unter Mac OS:**
  in neueren Versionen wohl oft vorinstalliert; wenn nicht, dann entweder:
  `> brew install git` (im Terminal)
  oder Download & Installation von **https://git-scm.com/**
- **Unter Linux:**
  meistens ohnehin vorinstalliert; wenn nicht, dann z. B.
  `> sudo apt-get install git` (Ubuntu/Debian)

2. Prüfe, ob die Installation von Git erfolgreich war, indem du deine Bash-Konsole öffnest und den Befehl
   `> git help`
   eingibst. Jetzt sollte als Ausgabe eine Hilfsübersicht zu verschiedenen Git-Befehlen erscheinen.

Aufgabe B: GitHub-Account einrichten

1. Wenn du noch keinen GitHub-Account hast, dann navigiere zu **https://github.com/signup**.
   Folge dort den Anweisungen, um dir einen GitHub-Account zu erstellen.
2. Sobald der Account erstellt ist, logge dich ein und navigiere anschließend zur Einstellungsseite **https://github.com/settings/tokens**.
3. Klicke dort auf „Generate new token", dann „Generate new token (classic) *For general use*".
4. Hier kannst du jetzt ein sog. Personal Access Token (PAT) erstellen. Ein PAT ist im Grunde genommen wie ein zusätzliches Passwort, welches benötigt wird, um sich per Git mit GitHub zu verbinden und Änderungen aus einem lokalen Repository mit einer Remote-Kopie auf GitHub zu synchronisieren.
   a. Tippe im Textfeld unter dem Punkt „Note" irgendeine Notiz / einen Titel ein, bspw. „mein PAT".
   b. Wähle unter dem Punkt „Expiration" eine Gültigkeitsdauer für dein PAT aus, also z. B. 30, 60 oder 90 Tage. (Nach Ablauf der Gültigkeitsdauer kannst du dein PAT regenerieren lassen.) Aus Sicherheitsgründen wird empfohlen, nicht von der Option eines unbegrenzt gültigen PATs Gebrauch zu machen.
   c. Im Weiteren kannst du unter dem Punkt „Select scopes" festlegen, welche Zugriffsrechte mit dem PAT verbunden sein sollen. Hier reicht es für unsere Zwecke, die gesamte Überkategorie „repo" mit Häkchen auszuwählen (und nichts anderes).
   d. Scrolle ganz nach unten und klicke dann auf den Button „Generate token".
   e. Das PAT ist erstellt! Achtung: Schließe die Webseite noch nicht!
   Kopiere zunächst das jetzt angezeigte PAT, welches aus einer Folge von 40 Zeichen (Buchstaben, Zahlen, Sonderzeichen) besteht, in die Zwischenablage.

Bewahre das PAT gut auf, denn du bekommst es nur dieses eine Mal angezeigt!

Du wirst das kopierte PAT erst in Aufgabe F weiter unten erneut brauchen.

(Aber falls du die Webseite versehentlich schon geschlossen hast, bevor du das PAT kopieren konntest, dann kannst du dir auch einfach ein gänzlich neues generieren lassen.)

Aufgabe C: Autorennamen und E-Mail-Adresse konfigurieren

1. Öffne deine Bash-Konsole (auf Windows: Git Bash – auf Linux / Mac OS: normale Terminal-App).
2. Konfiguriere in Git deinen Vor- und Nachnamen und deine E-Mail-Adresse, indem du folgende zwei Befehle ausführst:
   ```
   > git config --global user.name "<Vorname> <Nachname>"
   > git config --global user.email "<deine E-Mail-Adresse>"
   ```

   Achtung: Also gib es bitte genau so ein wie hier, ***ohne*** *„=“-Zeichen nach* `user.name` *bzw.* `user.email` *– leider hat sich diesbezüglich im Vorlesungsvideo ein kleiner Fehler eingeschlichen.*

Aufgabe D: Repository lokal initialisieren

1. Bleibe weiterhin in deiner Bash-Konsole.
2. Navigiere mit
   `> cd Desktop`
   in dein Desktop-Verzeichnis.
   (Vorausgesetzt, dass du dich gerade in deinem Home- bzw. Benutzer-Verzeichnis befindest.)
3. Erstelle mit
   `> mkdir wsa-u7-abgabe`
   einen Ordner namens `wsa-u7-abgabe` auf deinem Desktop.
4. Navigiere mit
   `> cd wsa-u7-abgabe`
   in diesen neuen Ordner.
5. Initialisiere ein neues Git-Repository in diesem Ordner.
   (Weißt du noch, mit welchem Befehl das geht?)
6. Überprüfe mit
   `> ls -a`
   ob sich nun tatsächlich ein verstecktes Unterverzeichnis namens `.git` im Ordner befindet.
7. Führe zuletzt noch folgenden Befehl aus:
   `> git branch -M main`
   (Dieser stellt lediglich sicher, dass dein Haupt-Branch `main` heißt. In manchen früheren Git-Versionen ist der Standardname gemäß einer veralteten Konvention stattdessen nämlich `master`.)

Aufgabe E: Datei hinzufügen und ein paar lineare Commits

1. Erstelle im Ordner `wsa-u7-abgabe` eine neue Datei namens `skript.R`, also ein einfaches R-Skript.

   Hinweis: Um den Ordner `wsa-u7-abgabe` *schnell in deinem gewohnten, grafischen Dateiexplorer anzeigen zu lassen, kannst du in Bash einfach einen der folgenden drei Befehle,*
   `> explorer` *(auf Windows) oder*
   `> open .` *(auf Mac OS) oder*
   `> xdg-open .` *(auf Linux),*
   ausführen lassen.
2. Schreibe erst einmal nur folgende einzige Code-Zeile in `skript.R` auf und speichere die Datei dann:

```
data(airquality)
```

3. Überprüfe den Status deines Repositorys. Du solltest die Information erhalten, dass sich `skript.R` gerade noch unter den sog. „Untracked files“ befindet.
   (Weißt du noch den Befehl, mit dem du den Status eines Git-Repositorys überprüfen kannst?)
4. Merke die Datei `skript.R` für den nächsten Commit vor.
   (Findest du den passenden Befehl?)
5. Erstelle schließlich einen Commit, dem du die Nachricht „Datei skript.R hinzugefügt“ oder so ähnlich beifügen kannst.
6. Lasse dir deine aktuelle Versionsgeschichte mit
   `> git log --all --graph --decorate`
   anzeigen. Du solltest darin jetzt einen einzigen Commit vorfinden (den soeben erstellten).
7. Füge in `skript.R` als zweite Code-Zeile folgende hinzu:

```
summary(airquality)
```

8. Lasse nun diese letzte Änderung wieder zum Committen vormerken und erstelle anschließend tatsächlich einen weiteren Commit. Diesem zweiten Commit solltest du eine sinnvolle Nachricht beifügen, die die erfolgte Änderung kurz und knapp erklärt.
9. Füge nun **zwischen** die beiden bisherigen Zeilen `data(...)` und `summary(...)` einen weitere R-Befehlszeile ein, und zwar folgende:

```
airquality <- na.omit(airquality)
```

- (Hier werden nun also erst einmal alle Beobachtungen entfernt, denen für mindestens eine der Variablen ein Wert fehlt – Stichwort: `NA` – und erst danach wird auf dem so reduzierten `airquality`-Datensatz eine entsprechende Zusammenfassung mit `summary(...)` generiert.)

10. Merke diese Änderung wieder fürs Committen vor und erstelle direkt im Anschluss einen dritten Commit, mit geeigneter Commit-Nachricht.

Aufgabe F: Zwischenstand als Remote-Kopie auf GitHub speichern

1. Navigiere auf **https://github.com/new**, um mit der Erstellung eines Remote-Repositorys auf GitHub zu beginnen. (Du musst weiterhin mit deinem Account eingeloggt sein.)
2. Trage unter „Repository name“ wieder `wsa-u7-abgabe` ein.
3. Wähle als Sichtbarkeitstyp „Private“ (anstatt von „Public“) aus.
   So werden nur du und spezifisch von dir ausgewählte Personen auf das Repository zugreifen können.
4. Lasse alle weiteren Auswahlmöglichkeiten unverändert und klicke dann unten rechts auf den Button „Create repository“.
5. In deiner Bash-Konsole, im Verzeichnis deines lokalen Git-Repositorys:

   Führe folgenden Befehl aus,

   `> git remote add origin https://<PAT>@github.com/<GitHub-Username>/wsa-u7-abgabe.git`

   wobei du den Platzhalter `<PAT>` durch dein 40 Zeichen langes Personal Access Token ersetzen solltest sowie den Platzhalter `<GitHub-Username>` durch deinen Benutzernamen auf GitHub.
6. Verknüpfe mit folgendem Befehl deinen lokalen `main`-Branch mit deinem Remote-`main`-Branch, wobei zugleich eine erste Synchronisation (d. h. ein Upload deiner noch rein lokalen Dateien samt Commits) erfolgen wird:

   `> git push -u origin main`

Aufgabe G: Checkout, Branch und Merge

1. Versetze nun mit einem passenden Git-Befehl den Zustand deines lokalen Repositorys zurück auf den vergangenen Stand, als du noch nicht die Zeile mit dem `na.omit(...)`-Befehl hinzugefügt hattest.
2. Schaue dir nun erneut die volle Versionsgeschichte mit dem Befehl
   `> git log --all --graph --decorate`
   an und überprüfe, ob das Zurückversetzen geklappt hat.
   Hinweis: Achte darauf, wo sich das Label `HEAD` *befindet.*
3. Schaue dir den aktuellen Inhalt der Datei `skript.R` und überzeuge dich zusätzlich auch so, dass das Zurückversetzen der Datei auf den früheren Zustand tatsächlich funktioniert hat.
4. Erstelle in diesem Zustand einen neuen Branch, dem du das Label `diagramme` geben kannst.
5. Mache den Branch `diagramme` zu deinem aktuell aktiven Branch (anstatt `main`).
6. Öffne nun wieder die Datei `skript.R` und füge ganz unten noch folgende zwei Zeilen hinzu:

```
hist(airquality$Ozone)
plot(airquality$Wind, airquality$Temp)
```

   - (Bemerkung: Die vorinstallierten R-Grafik-Funktionen `hist` und `plot` erstellen hier jeweils ein einfaches Histogramm bzw. Streudiagramm auf Grundlage der gegebenen Daten. Wir haben sie bisher noch nicht kennengelernt. Sie sind im Prinzip „faule“ Alternativen zu ihren komplizierteren Pendants `geom_hist` und `geom_point` aus dem `ggplot2`-Paket, mit denen wir bisher ausschließlich gearbeitet haben. Während `ggplot2` modernere und hochgradig anpassbare Grafiken in druckreifer Qualität generieren kann, eignen sich vorinstallierte R-Grafik-Funktionen wie `hist` und `plot` eher für die Erstellung schneller Übersichtsplots, die aber nicht unbedingt zum Veröffentlichen gedacht sind.)
7. Committe die in Schritt 6 erfolgten Änderungen.
8. Schaue dir nun wieder die volle Versionsgeschichte an. Du solltest erkennen können, dass sie jetzt verzweigt ist.
9. Wechsle wieder zurück zum `main`-Branch, mache ihn also zu deinem aktuell aktiven Branch.
10. Vereinige die beiden Branches `main` und `diagramme` schließlich wieder zu einem einzigen Branch, indem der `diagramme`-Branch aufgelöst wird und seine Änderungen automatisch in den `main`-Branch eingearbeitet werden.
11. (Nur für dich selbst zum besseren Verständnis:)
    Vergleiche gedanklich den in Schritt 10 erfolgten „Merge“ mit dem Beispiel aus der Vorlesung. Gab es diesmal wieder einen Merge-Konflikt? Oder kam es dazu diesmal nicht? Und warum?
12. Führe den Befehl
    `> git push`
    aus, um die bei dir lokal committeten Änderungen auch wieder in die Remote-Kopie auf GitHub hinein-zu-synchronisieren.

Aufgabe H: Kompression des Repositorys zu einem ZIP-Archiv

1. Komprimiere jetzt den ganzen Ordner des Repositorys, also `wsa-u7-abgabe`, zu einem ZIP-Archiv.

   Hinweis: Hier findest du Videotutorials zum Erstellen eines ZIP-Archivs aus einem Ordner auf Windows (**https://youtu.be/-f1pEpRu31E**)*, Mac OS* (**https://youtu.be/SUlI3-3mkU0**) *oder Linux* (**https://youtu.be/7UocaXEScao**).
2. Benenne das ZIP-Archiv um in `wsa-u7-abgabe_<Vorname>_<Nachname>.zip`.

Aufgabe I: Kursevaluation (freiwillig, anonym)

1. (Nur falls du es nicht ohnehin schon ausgefüllt hast:)
   Das offizielle Evaluations-Formular für diesen Kurs, also „Wissenschaftliches Schreiben und Arbeiten“, findest du hier: **https://pep.uni-potsdam.de/ce/c4a2fad6/de.html**
   Hier hast du die Gelegenheit, komplett anonymisiert zu bewerten, wie dir die Lehre in diesem Kurs gefallen hat bzw. welche Aspekte deiner Meinung nach verbessert werden könnten.
   Die Teilnahme an der Evaluation ist freiwillig. Bitte nimm dir aber, wenn du es dir einrichten kannst, die paar Minuten Zeit, um das Formular auszufüllen und so dein ehrliches Feedback abzugeben.

Abgabe:

Lade **bis zum 15.02.24** um 23:59 Uhr folgende einzige Datei in der Abgabemaske im Kurs-Moodle hoch:

- `wsa-u7-abgabe_<Vorname>_<Nachname>.zip`