1 Übungen zu Git

1.1 Git-Installation

1. Installiert euch Git:
   - Windows: `https://git-scm.com/download/win`
   - Linux: Ubuntu-/Debian-Pakete `git` und `gitk`
   - Mac: `https://git-scm.com/download/mac` oder das Paket `git` via MacPorts

1.2 Bash/Linux-Kommandozeile

1. Lasst euch anzeigen, in welchem Verzeichnis ihr gerade seid.
2. Lasst euch anzeigen, welche Verzeichnisse und Dateien es im aktuellen Verzeichnis gibt.
3. Wechselt in ein Unterverzeichnis.
4. Lasst euch wieder anzeigen, in welchem Verzeichnis ihr gerade seid.
5. Wechselt wieder ein Verzeichnis nach oben.
6. Wechselt mit `cd -` wieder in das Unterverzeichnis und wieder zurück.
7. Wechselt in das Verzeichnis, in dem ihr die Übungen für heute machen möchtet, und in dem ihr Dateien anlegen möchtet.
8. Legt eine leere Datei an (`touch`).
9. Schaut, dass die Datei jetzt tatsächlich existiert.
10. Benennt die Datei um.
11. Schaut, dass die Datei jetzt anders heißt.
12. Löscht die Datei.
13. Schaut, dass die Datei jetzt tatsächlich nicht mehr existiert.

1.3 Git-Konfiguration

1. Richtet euch Tab-Autovervollständigung für Git ein. Ihr könnt es testen, indem ihr `git check` tippt und dann Tab drückt. Wenn aus `check` ein `checkout` wird, tut's die Autovervollständigung.
   - Windows: sollte schon ab Werk gehen

- Linux und Mac: Folgt der Anleitung[1].

2. Richtet euch diese Sachen in Git ein und prüft die Einstellungen danach mit `git config --list`. Ersetzt dabei in den ersten beiden Zeilen die Platzhalter mit eurem echten vollen Namen und eurer Mailadresse.

```
git config --global user.name "Jane Doe"
git config --global user.email "jane.doe@example.com"
git config --global color.ui auto
git config --global branch.autosetupmerge always
git config --global rerere.enabled true
git config --global pull.rebase true
git config --global push.default simple
git config --global core.precomposeunicode true
git config --global core.quotepath false
git config --global branch.autosetuprebase always
git config --global rebase.autostash true
git config --global core.autocrlf false
git config --global core.eol lf
```

1.4 Lokales Arbeiten

1. Lasst euch die Git-Hilfe anzeigen.
2. Lasst euch die Git-Hilfe für `git init` anzeigen. Falls die Hilfe auf der Kommandozeile angezeigt wird, kommt ihr mit der `q`-Taste wieder raus.
3. Legt das Verzeichnis an, mit dem ihr lokal üben möchtet, und wechselt in das Verzeichnis.
4. Legt ein leeres Git-Repository an.
5. Lasst euch den Status des Repositories anzeigen.
6. Legt eine kleine Textdatei `hans.txt` an.
7. Lasst euch den Status anzeigen.
8. Fügt die Datei zur Staging-Area hinzu.
9. Lasst euch den Status anzeigen.
10. Lasst euch die Unterschiede zum letzten Commit anzeigen.
11. Lasst euch die Unterschiede der Staging-Area zum letzten Commit anzeigen.
12. Committet die Änderungen. Gebt dabei die Commit-Message direkt mit an.
13. Lasst euch den Status anzeigen.

[1] `https://git-scm.com/book/en/v1/Git-Basics-Tips-and-Tricks`

14. Lasst euch das Log anzeigen.
15. Lasst euch den letzten Commit anzeigen.
16. Lasst euch die Liste der im letzen Commit geänderten Dateien anzeigen.
17. Und jetzt noch einmal dasselbe mit der Commit-ID.
18. Bearbeitet die existierende Datei `hans.txt` und legt außerdem eine zweite Datei `wurst.txt` neu an.
19. Lasst euch die Unterschiede zum letzten Commit anzeigen. (Mit `q` kommt ihr da wieder raus.)
20. Fügt die Datei zur Staging-Area hinzu.
21. Lasst euch den Status anzeigen.
22. Lasst euch die Unterschiede zum letzten Commit anzeigen.
23. Lasst euch die Unterschiede der Staging-Area zum letzten Commit anzeigen.
24. Commited die Änderungen, benutzt aber diesmal für die Commit-Message einen separaten Editor, indem ihr sie nicht direkt schon auf der Kommandozeile angebt.
25. Lasst euch Status und Log anzeigen.
26. Bearbeitet `hans.txt`. Commit die Datei, ohne vorher ein explizites `add` zu machen.
27. Lasst euch das Log einzeilig anzeigen.

1.5 Schadensbegrenzung

1. Bearbeitet die `hans.txt` und fügt diese zum letzten Commit hinzu.
2. Lasst euch das Log anzeigen.
3. Bearbeitet `hans.txt` und setzt die Datei dann wieder die Version aus dem letzten Commit zurück.
4. Setzt `hans.txt` auf die Version auf dem vorletzten Commit zurück (ohne zu commiten).
5. Macht den kompletten letzten Commit rückgängig.
6. Ändert eine Datei. Bringt die Änderungen mit dem Stash in Sicherheit.
7. Holt die Änderungen wieder aus dem Stash.
8. Setzt die Datei wieder auf den Ursprungszustand zurück.

1.6 Branches

1. Lasst euch anzeigen, welche Branches es gibt und auf welchem Branch ihr gerade seid.
2. Legt einen Branch `pinguin` an.
3. Wechselt auf den Branch.
4. Legt in dem Branch eine Datei `watscheln.txt` an und committet sie.
5. Schaut euch das Log an.
6. Wechselt wieder auf den `master`-Branch und schaut euch wieder das Log an.
7. Lasst euch den Unterschied zum `pinguin`-Branch anzeigen.
8. Wechselt mit `git checkout -` zum `pinguin` und wieder zurück zum `master`.
9. Merget den `pinguin`-Branch in den master und schaut euch das Log an.
10. Löscht den `pinguin`-Branch.
11. Legt mit einem Kommando einen Branch `nacktmull` an und wechselt direkt in den Branch.
12. Legt eine Datei `frittiertes-mars.txt` an und committed sie.
13. Wechselt wieder in den `master`.
14. Löscht den nacktmull-Branch, ohne ihn zu mergen.

1.7 Rebase und Konflikte

1. Legt einen Branch `tee` mit ein paar Zeilen Text an, wechselt in den Branch, legt dort eine Datei `earl grey.txt` (mit Leerzeichen!) an und committed sie.
2. Wechselt in den `master`, legt dort eine Datei brot.txt an und committed sie.
3. Wechselt auf den `tee`-Branch und rebased von `master`.
4. Wechselt in den `master`, bearbeitet `brot.txt` und committed die Änderungen.
5. Wechselt nach `tee`, bearbeitet in `brot.txt` dieselben Zeilen und committed die Änderungen.
6. Rebased von `master` und lasst euch den Status der Konflikte anzeigen.
7. Brecht das Rebase ab.
8. Rebased noch einmal. Löst diesmal die Konflikte im Editor, added die Datei und führt den Rebase dann zu Ende.

1.8 RSA-Keys und Accounts

1. Erzeugt euch ein RSA-Schlüsselpaar (falls noch nicht geschehen).
2. Loggt euch bei GitHub ein und ladet euren öffentlichen RSA-Schlüssel hoch.
3. Loggt euch beim MHN-GitLab ein und ladet euren öffentlichen RSA-Schlüssel hoch (falls ihr einen MHN-Account habt).
4. Teilt Oli euren GitHub-Usernamen mit und bittet ihn, euch zum Übungs-Repository hinzuzufügen.

1.9 Verteiltes Arbeiten

1. Klont euch das Übungsrepository von GitHub. Achtet dabei darauf dass ihr es per SSH klont und nicht per HTTPS.
2. Wechselt in das Verzeichnis des geklonten Repositories.
3. Lasst euch die Remotes anzeigen.
4. Und jetzt mit den URLs.
5. Lasst euch die lokalen Branches anzeigen.
6. Lasst euch die Remote-Branches anzeigen.
7. Check den `gurkensalat`-Remote-Branch aus und wechselt direkt auf den Branch.
8. Wartet, bis Oli etwas geändert und gepusht hat.
9. Wechselt wieder auf den `master` und löscht den lokalen `gurkensalat`-Branch wieder.
10. Pullt euch die Änderungen.
11. Legt einen Branch an, legt dort einen Datei an und committed sie.
12. Wartet, bis Oli etwas geändert und gepusht hat.
13. Wechselt zum `master`, pullt, wechselt zu eurem Branch rebased, wechselt wieder zu `master`, merged euren Branch, pusht, und löscht euren Branch.

1.10 Noch mehr Schadensbegrenzung

1. Erzeugt einen Branch nicht von `master`, sondern vom vorletzten Commit von `master`. Löscht den Branch dann wieder.
2. Schaut euch das Reflog an.
3. Stellt den gelöschten Branch `nacktmull` wieder her.

4. Wechselt auf den Branch.
5. Vertauscht die letzten beiden Commits per interaktivem Rebase.
6. Löscht den vorletzten Commit aus dem Branch (per interaktivem Rebase).
7. Wechselt wieder auf den `master` und löscht den `nacktmull`-Branch.