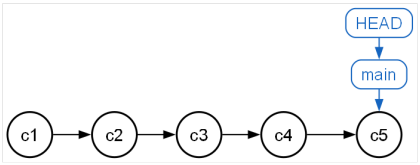
1. (5 Punkte) In einem Repository steht in der Datei `f.txt` nur der Buchstabe A. Arbeitsverzeichnis, Staging Area und letzter Commit stimmen überein. Gib nach jedem Schritt an, was in `f.txt` im Arbeitsverzeichnis, in der Staging Area und im letzten Commit steht.

	Schritt	Arbeitsverzeichnis	Staging Area	letzter Commit
	Anfang	A	A	A
1.	Datei auf B ändern			
2.	`git add f.txt`			
3.	Datei auf C ändern			
4.	`git commit -m x`			
5.	`git restore f.txt`			

2. (4 Punkte) Gib jeweils einen Git-Befehl an.
   (a) Du hast `programm.py` geändert und willst die Änderung verwerfen (noch nicht mit `add` gestaged).
   (b) Du hast `programm.py` versehentlich mit `git add` in die Staging Area gelegt und willst sie dort wieder herausnehmen, ohne die Änderung zu verlieren.
   (c) Du willst kurz auf einen anderen Branch wechseln, hast aber noch ungesicherte Änderungen, die du später weiterbearbeiten willst.
   (d) Du willst einen neuen Branch `grafik` anlegen und gleich dorthin wechseln.
   (e) Du willst sehen, was du seit dem letzten Commit geändert hast.
   (f) Du willst sehen, wie die Datei `programm.py` vor zwei Commits aussah, ohne etwas zu verändern.

3. (4 Punkte) Gegeben ist die folgende Geschichte eines Repositorys.



   (a) Welche Commits sind `HEAD~2` und `HEAD~4`?

   (b) Nach `git switch --detach HEAD~3`: Auf welchen Commit zeigt HEAD? Was bedeutet „detached“?

   (c) Wie kommst du danach wieder auf den neuesten Stand?

4. (5 Punkte) In einem neuen Repository werden nacheinander diese Befehle ausgeführt. Bei jedem Commit wurde vorher eine Datei geändert, die Commit-Nachricht nennen wir als Namen des Commits.

```
git commit -m c1   git commit -m c2   git switch -c neu   git commit -m n1
  git commit -m n2   git switch main   git commit -m c3   git merge neu
```

   (a) Zeichne den Commit-Graphen mit den Branches und HEAD.

   (b) Welche Art von Merge findet statt? Warum?

   (c) Wie viele Vorgänger hat der letzte Commit?

5. (4 Punkte) Auf `main` wird `git merge design` ausgeführt. Git meldet einen Konflikt, die Datei `farben.txt` sieht danach so aus:

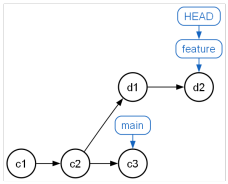
```
rot
<<<<<<< HEAD
weiß
=======
gelb
>>>>>>> design
blau
```

   (a) Wie kam es zu dem Konflikt? Was stand in der Datei, bevor sich die Branches getrennt haben (soweit man es erkennen kann)?

   (b) Welche Fassung stammt aus `main`, welche aus `design`?

   (c) Der Konflikt soll so gelöst werden, dass beide Farben in der Datei stehen. Gib die Datei und die Befehle an, mit denen der Merge abgeschlossen wird.

6. (5 Punkte) Gegeben ist die folgende Situation, HEAD steht auf `feature`.

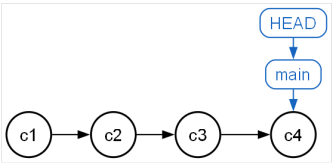


(a) Zeichne den Graphen nach `git rebase main`.

(b) Danach werden `git switch main` und `git merge feature` ausgeführt. Welche Art von Merge ist das? Zeichne das Ergebnis.

(c) Warum soll man `rebase` nicht auf Commits anwenden, die andere schon haben?

7. (5 Punkte) Der Commit c4 enthält einen Fehler und soll rückgängig gemacht werden.



(a) Zeichne den Graphen nach `git reset --hard HEAD~1` und (ausgehend von der Abbildung) nach `git revert HEAD`.

(b) Welchen Befehl nimmst du, wenn c4 schon auf GitHub liegt und andere damit arbeiten? Begründe.

(c) Was ist bei `git reset HEAD~1` (ohne `--hard`) anders?

8. (4 Punkte) Die Datei `.gitignore` eines Repositorys enthält:

```
*.log
geheim.txt
notizen/
*.tmp   # temporäre Dateien
```

(a) Welche der folgenden Dateien werden von Git ignoriert?

`fehler.log`, `logs/alt.log`, `geheim.txt`, `privat/geheim.txt`, `notizen/idee.txt`, `test.tmp`, `programm.py`

(b) `programm.py` ist schon in einem Commit. Jetzt wird `programm.py` in `.gitignore` eingetragen. Was passiert bei der nächsten Änderung an der Datei? Was muss man tun?