Erste Schritte mit git

Initialisieren

Um die Abläufe und die Funktionsweise zu erproben, wollen wir zunächst ein Verzeichnis unter Versionskontrolle stellen, in dem wir ein **Tagebuch** anlegen. Wir erstellen also ein Verzeichnis `tagebuch` und initialisieren dort ein Git-Repository:

```
max@pc:~$ mkdir tagebuch
max@pc:~$ cd tagebuch/
max@pc:~$ git init
Leeres Git-Repository in /home/max/tagebuch/.git/ initialisiert
```

Nun steht das Verzeichnis `tagebuch` unter Versionskontrolle. Das lokale Git-Repository befindet sich im Unterverzeichnis `.git`:

```
$ ls -la
insgesamt 132
drwxr-xr-x  3 max max   4096 24. Okt 13:32 .
drwxr-xr-x 21 max max 122880 24. Okt 13:32 ..
drwxr-xr-x  7 max max   4096 24. Okt 13:32 .git
```

Repository Status anzeigen lassen

Das Verzeichnis `tagebuch` ist jetzt ein "git-Repository" - es wird von git "beobachtet", so dass man von nun an Änderungen in diesem Verzeichnis und seinen Unterverzeichnissen nachverfolgen kann.

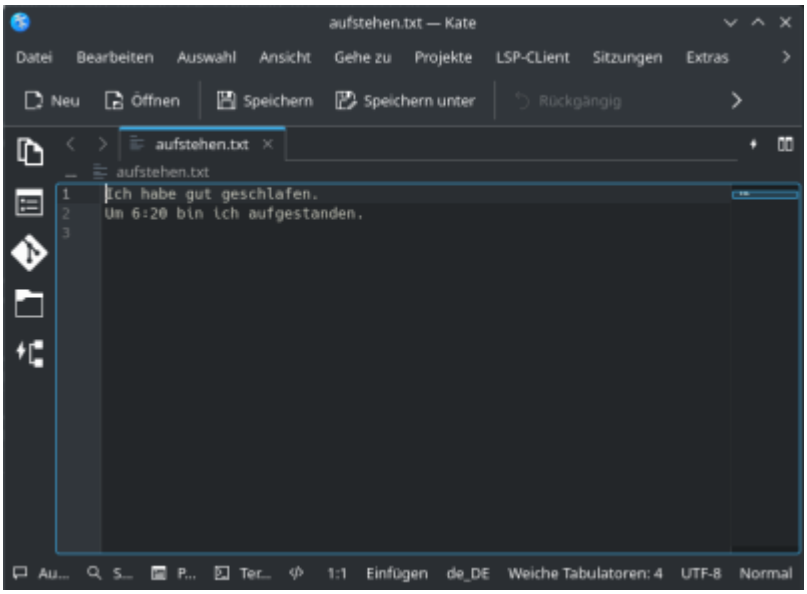
Mit dem Befehl `git status` kann man sich den aktuellen Status des "Repos" anzeigen lassen:

```
max@pc:~/tagebuch$ git status
Auf Branch main

Noch keine Commits

nichts zu committen (erstellen/kopieren Sie Dateien und benutzen
Sie "git add" zum Versionieren)
```

[1)]

Ein erster Tagebucheintrag

Lege mit einem Texteditor[2] eine Datei `aufstehen.txt` an. Du kannst in diese Datei z.B. hineinschreiben, wie du geschlafen hast und wann du aufgestanden bist. Das folgende Beispiel verwendet den Editor `nano` unter Linux, du kannst aber auch Notepad++ unter Windows oder Kate unter Linux verwenden, diese Editoren haben eine GUI. Wichtig ist, dass du die Datei im Verzeichnis `tagebuch` abspeicherst.




```
max@pc:~/tagebuch$ nano aufstehen.txt
max@pc:~/tagebuch$ cat aufstehen.txt
Ich habe gut geschlafen.
Um 6:20 bin ich aufgestanden.
```

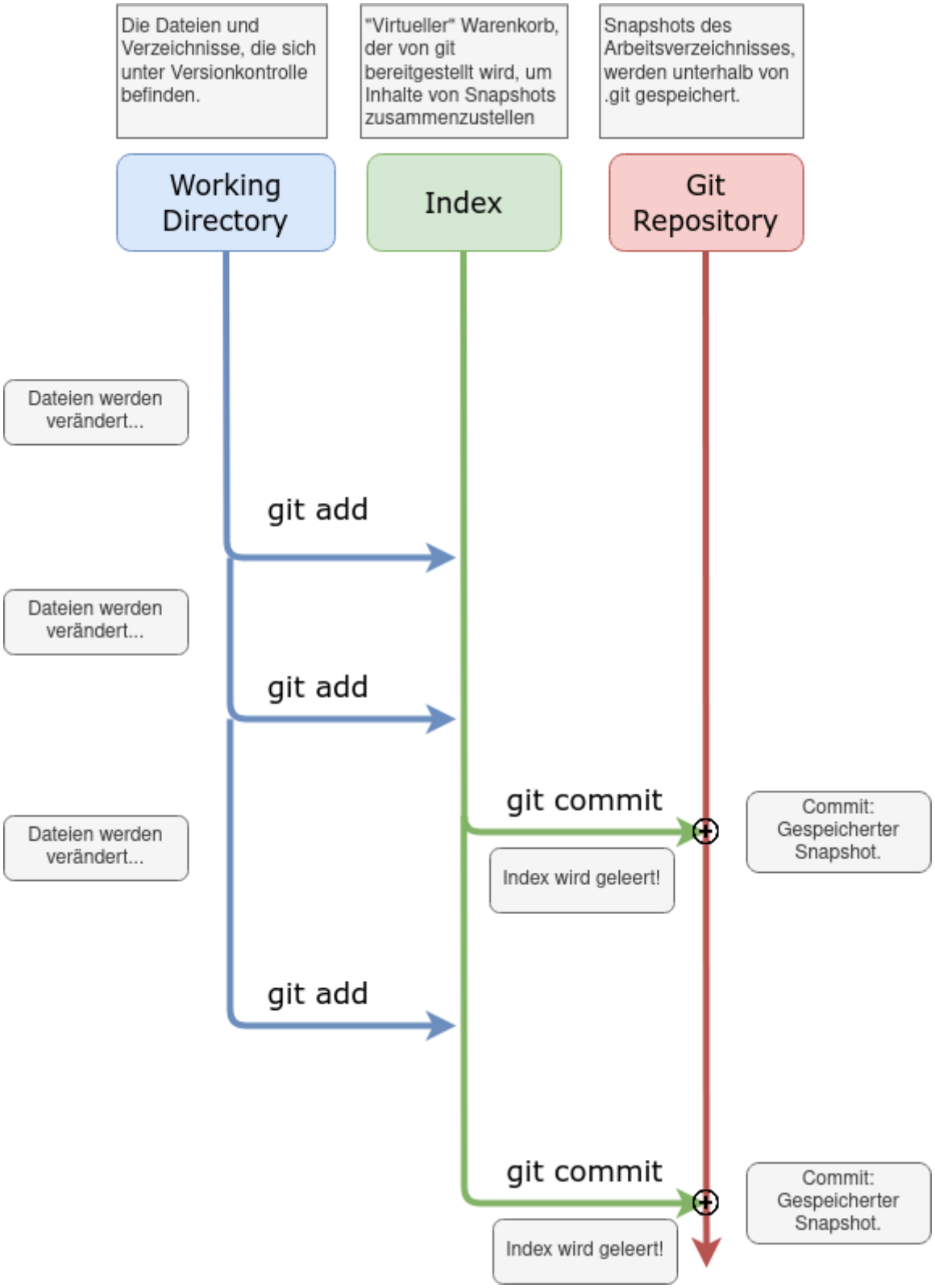
Unser Tagebuch enthält nun einen Eintrag in `aufstehen.txt`. Wir wollen jetzt den Zustand des Tagebuchs an dieser Stelle so in unserer Versionsverwaltung festhalten, dass wir ihn später wieder verwenden können.

Ein erster Commit

Um den git-Workflow zu verstehen, muss man drei Begriffe unterscheiden: Das Arbeitsverzeichnis ("Working Directory") den Index ("Staging Area") und das eigentliche Repository.

- **Arbeitsverzeichnis (Working Directory):** Das ist Verzeichnis, welches man zuvor mit `git init` unter Versionskontrolle gestellt hat mit allen seinen Dateien und Unterverzeichnissen, so wie man es auf der Festplatte vorfindet. Das "spezielle" Verzeichnis `.git` wird dabei ignoriert, es dient der internen Verwaltung der Abläufe durch git.
- **Index ("Staging Area")**: Im Index werden zunächst alle Dateien eingetragen, die in einem nächsten Schritt zu einem Snapshot zusammengefasst und im Repository gespeichert werden sollen. Der Sinn des Index erschließt sich nicht unmittelbar, da man dazu neigt, sich vorzustellen, dass man nacheinander Änderungen in deinem Arbeitsverzeichnis vornimmt und dabei von Zeit zu Zeit einfach Snapshots des gesamten Arbeitsverzeichnisses anlegt - das trifft aber nicht zu. Es gibt zahlreiche Anwendungsfälle, bei denen man nicht alle Änderungen des Arbeitsverzeichnisses in einem Snapshot festhalten möchte, sondern z.B. auf mehrere Snapshots aufteilen will. Außerdem kommt es häufig vor, dass sich im Arbeitsverzeichnis Dateien befinden, die man gar nicht unter Versionskontrolle stellen möchte, beispielsweise Compilate von Java Programmen (class-Dateien).
- **Repository:** Wenn man im Index alle Dateien für den nächsten Snapshot zusammengestellt hat, kann man einen neuen Snapshot erstellen. Ein solcher Snapshot heißt **Commit** und wird durch eine Hashsumme identifiziert, außerdem werden Metainformationen wie Zeit und Name des Commiters festgehalten. Ein Commit wird mit dem Befehl `git commit` durchgeführt. Nach

einem Commit ist der Index stets leer, da ja alle Änderungen, die dort vorgemerkt waren, in den Snapshot überführt wurden.



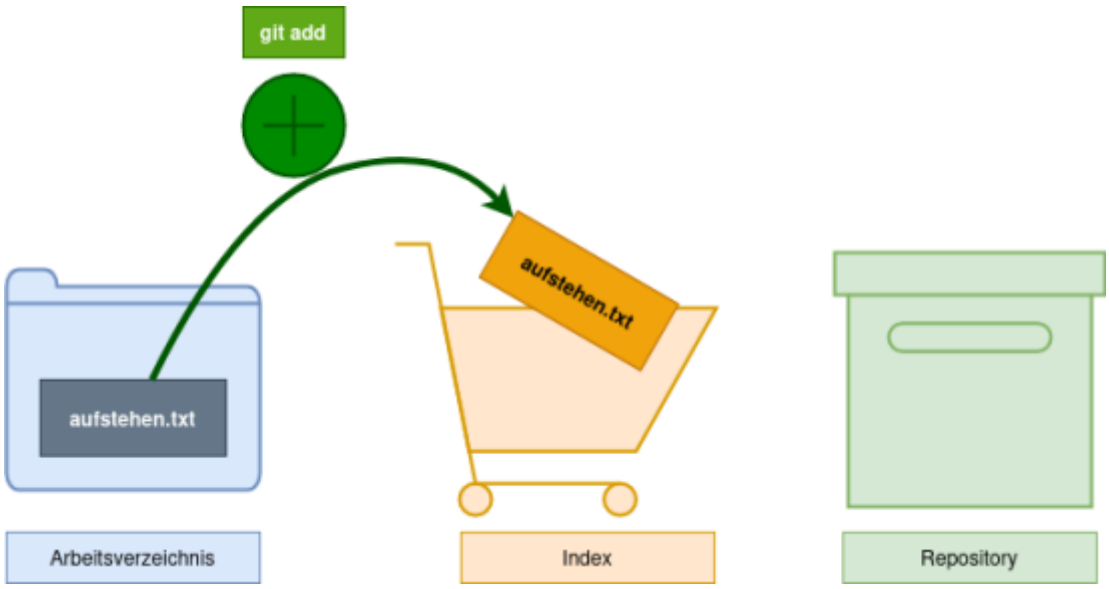
Schritt für Schritt

Neue Dateien befinden sich zunächst "nur" im Arbeitsverzeichnis und werden von git ignoriert. Mit `git status` kann man das überprüfen, solche Dateien tauchen dort in der Liste der "Unversionierten Dateien" auf, für unser Tagebuch sieht das so aus:

```
max@pc:~/tagebuch$ git status
Auf Branch main

Noch keine Commits

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
        aufstehen.txt

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
```

Mit dem Befehl `git add` wird eine Datei im Index vorgemerkt - das kann man sich vorstellen wie ein Einkaufswagen, in dem neue Dateien und Änderungen gesammelt werden, bis man zu einem Punkt kommt, den man sich "merken" möchte. Im Folgenden habe ich die einzige Datei `aufstehen.txt` zum Index hinzugefügt:

```
max@pc:~/tagebuch$ git add aufstehen.txt
max@pc:~/tagebuch$ git status
Auf Branch main

Noch keine Commits

Zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-
Area)
        neue Datei:     aufstehen.txt
```



Wenn man mit den im Index vorgemerkten Änderungen zufrieden ist, macht man einen "Commit" und merkt sich dabei den Zustand aller im Index befindlichen Dateien.

Mit dem Befehl `git commit -m "Erster Commit: aufstehen.txt angelegt"` legt man einen Commit mit einer Commit-Message an (Paramter `-m`). Wenn man die Commit-Message nicht mit `-m`

angibt, öffnet sich ein Editor, in dem man diese bearbeiten muss.

```
max@pc:~/tagebuch$ git commit -m "Erster Commit: aufstehen.txt angelegt"
[main (Root-Commit) 28ec5a7] Erster Commit: aufstehen.txt angelegt
 1 file changed, 2 insertions(+)
 create mode 100644 aufstehen.txt
```

Wenn man den Status des Arbeitsverzeichnisses jetzt erneut abfragt, erhält man folgende Ausgabe:

```
max@pc:~/tagebuch$ git status
Auf Branch main
nichts zu committen, Arbeitsverzeichnis unverändert
```

Man erkennt, dass der Index wieder leer ist ("nichts zum Commit vorgemerkt").

Nun kann man weitere Änderungen im Tagebuch vornehmen und sich zu allen wichtigen Zeitpunkten den Zustand der Dateien in einem Commit merken.

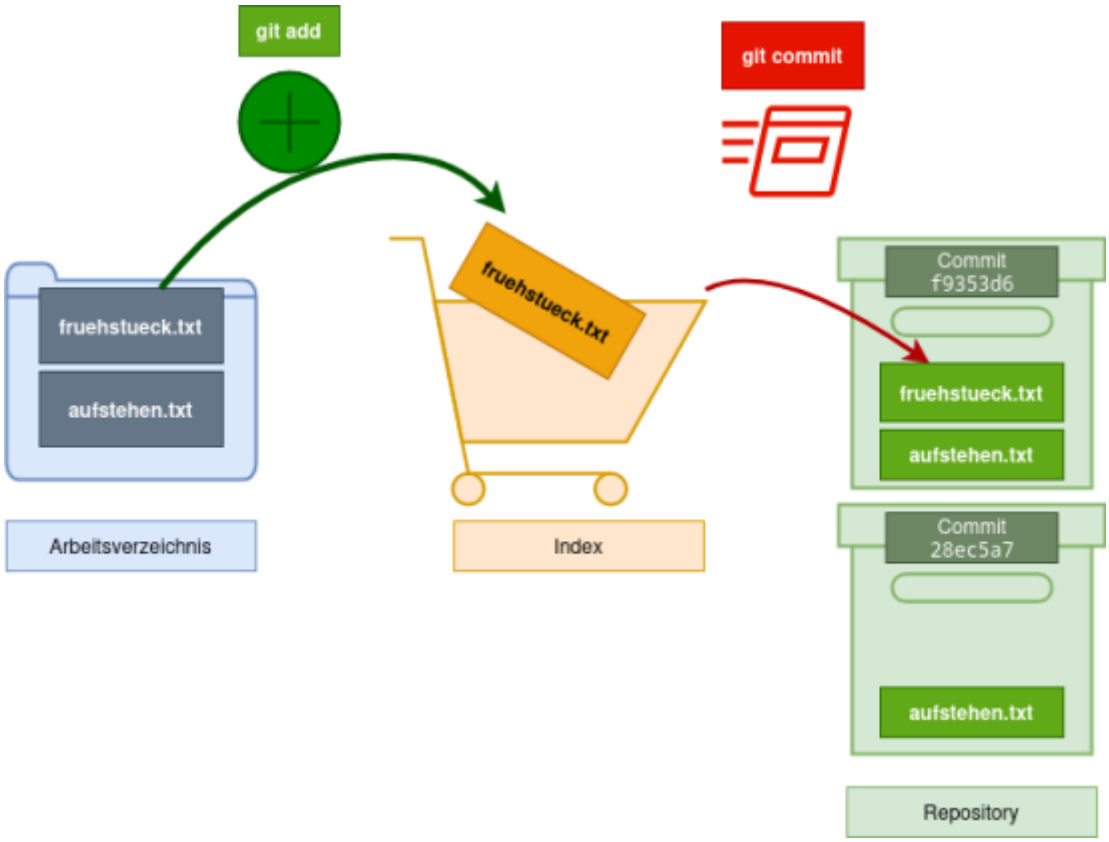
Wir frühstücken

(A1)

- Halte in der Datei `fruehstueck.txt` fest, was es zum Frühstück gab.
- Kontrolliere mit `git status`, dass es die Datei jetzt gibt, sie aber nicht unter Versionskontrolle steht.
- Füge die Datei `fruehstueck.txt` mit dem Befehl `git add fuehstuck.txt` zum Index hinzu.
- Erstelle einen Commit für das Frühstück. Vergiss nicht die Commit-Message nach der Option `-m`.
- Überprüfe den Zustand deines Repositorys.

Lösung

Wir haben nun einen zweiten Commit erstellt:



> **Wichtig**
>
> Wir haben zwar nur die Datei `fruehstueck.txt` zum Commit vorgemerkt und anschließend mit `git commit` "commited", **ein Commit beinhaltet jedoch stets den Zustand aller unter Versionskontrolle stehender Dateien im Arbeitsverzeichnis**, also in diesem Fall ist in unserem zweiten Commit auch die (unveränderte) Datei `aufstehen.txt` enthalten!

Man kann sich einen Commit also wie im Bild dargestellt als Archivbox vorstellen, in dem jeweils der Zustand aller versionierten Dateien festgehalten ist. Ein Commit wird durch einen Hexadezimalen "Hashwert" identifiziert, das ist gewissermaßen die eindeutige Nummer eines Commits, z.B. 28ec5a7.

Mit dem Befehl `git log` kann man sich die Commits auflisten lassen:

```
max@pc:~/tagebuch$ git log
commit f9353d6278296bd173d3760e3b95e743208650e7 (HEAD -> main)
Author: Max Mustermann <max@example.org>
Date:   Wed Oct 2 08:58:45 2024 +0200

    Frühstück in Datei 'fruehstueck.txt' hinzugefügt

commit 28ec5a71bc2bce75bbbbfe5f2d336fe49b3642fa
Author: Max Mustermann <max@example.org>
```

```
Date:   Wed Oct 2 08:35:53 2024 +0200

    Erster Commit: aufstehen.txt angelegt
```

Man erkennt hier auch, dass die eigentlichen Commit-Hashes sehr viel länger sind, als das Beispiel oben vermuten lässt, für die Identifizierung eines Commits reichen die ersten 7 Stellen des Hashes aus.

Mittagessen

(A2)

- Füge deinem Tagebuch einen Eintrag `mittagessen.txt` hinzu, zunächst ohne diese zu versionieren.
- Jetzt fällt dir ein, dass du zum Frühstück ein Stück Schokolade hattest, das du nicht notiert hattest. Ändere die Datei `fruehstueck.txt` ab, so dass die Schokolade dort vermerkt ist.
- Überprüfe mit `git status` den Zustand deines Repositorys.

Dein Repo sollte ungefähr so aussehen:

```
max@pc:~/tagebuch$ nano mittagessen.txt
max@pc:~/tagebuch$ nano fruehstueck.txt
max@pc:~/tagebuch$ git status
Auf Branch main
Änderungen, die nicht zum Commit vorgemerkt sind:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
  (benutzen Sie "git restore <Datei>...", um die Änderungen im
Arbeitsverzeichnis zu verwerfen)
        geändert:       fruehstueck.txt

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
        mittagessen.txt

keine Änderungen zum Commit vorgemerkt (benutzen Sie "git add" und/oder "git
commit -a")
```

Wir haben jetzt **zwei** Dinge geändert:

- In der Datei `fruehstueck.txt` haben wir eine Änderung vorgenommen.
- Die Datei `mittagessen.txt` wurde neu hinzugefügt.

Wenn man nun den nächsten Commit vorbereitet, kann man mit dem Befehl `git add` wieder auswählen, welche Änderungen in den nächsten Commit übernommen werden. Um das zu

demonstrieren, teilen wir die beiden vorgenommenen Änderungen im Folgenden auf zwei Commits auf.

```
max@pc:~/tagebuch$ git add fruehstueck.txt
max@pc:~/tagebuch$ git status
Auf Branch main
Zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git restore --staged <Datei>..." zum Entfernen aus der
Staging-Area)
    geändert:       fruehstueck.txt

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
    mittagessen.txt
```

Jetzt haben wir die Änderungen von `fruehstueck.txt` für den nächsten Commit vorgemerkt, die neue Datei `mittagessen.txt` wird allerdings nicht in diesen übernommen. Mit `git commit -m "fruehstueck.txt geändert"` wird der Commit ausgeführt.

Der Status ist jetzt:

```
max@pc:~/tagebuch$ git status
Auf Branch main
Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
    mittagessen.txt

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
```

Für den nächsten Commit übernehmen wir jetzt die Datei `mittagessen.txt`:

```
git add mittagessen.txt
git commit -m "Mittagessen hinzugefügt"
```

> **Zwischenergebnis**
>
> Wir haben jetzt gelernt, wie wir selektiv **Dateien** in einem Verzeichnis **unter Versionskontrolle** stellen können. Mit jedem Commit erzeugen wir einen **Snapshot** des Zustands, den die versionierten Dateien zum Zeitpunkt des Commits haben. Dateien, die man nicht mit `git add` unter Versionskontrolle gestellt hat, werden von git nicht beeinflusst.
>
> Als nächstes wollen wir uns ansehen, wie wir uns die **Versionsgeschichte** genauer ansehen können und in der Zeit zurückreisen und **ältere Versionen betrachten** können.

Material

02-erstes-repo.odp	1.2 MiB	28.04.2021 18:18
02-erstes-repo.pdf	431.1 KiB	28.04.2021 18:18
2023-10-29_19-56.png	32.5 KiB	29.10.2023 18:57
2023-10-29_20-02.png	33.0 KiB	29.10.2023 19:02
commit.drawio.png	35.5 KiB	02.10.2024 06:49
ff.svg	1.9 KiB	02.10.2024 07:01
git_add.drawio.png	39.9 KiB	29.10.2023 19:12
gitstagingcommit.png	61.2 KiB	28.04.2021 13:14
zweitercommit.drawio.png	63.6 KiB	02.10.2024 07:05

[1)]

Bei älteren git-Versionen heißt der Hauptbranch, der hier angezeigt wird gelegentlich `master`, in diesem Fall muss man sich für das vorliegende Tutorial stets `main` durch `master` ersetzt denken – und in den Befehlen schreiben.

[2)]

Nicht mit Word oder Writer!