Textdateien

Effizienter arbeiten unter Linux

Patrick Bucher

07.04.2017

Windows- und Mac-User schreiben Texte meistens mit *Microsoft Word*. Als Dateiformat verwenden sie *Office Open XML* (OOXML) und als Endung des Dateinamens `.docx`.

Mit *OpenOffice.org Writer* und *Libre Office Writer* gibt es eine kostenlose Alternative zu *Microsoft Word*. Diese Software verwendet ein eigenes Dateiformat: *OpenDocument Text* (ODT) mit der Dateiendung `.odt`.

Beide Programme – *Word* und *Writer* – untersützen mittlerweile beide Formate. Im Gegensatz zu *Word* funktioniert aber *Writer* auch auf Linux. Hartgesottene Linux-User verwenden aber lieber einfache Textdateien (*plain text*) zum Speichern und Bearbeiten ihrer Texte. Warum ist das so?

Unterschiede zwischen Textdateien und *OOXML/ODT*

OOXML und ODT speichern Texte in einer komprimierten XML-Struktur ab. Einfache Textdateien haben keine solche Struktur, sondern bestehen nur aus einer Reihe von Zeichen. Dadurch ergeben sich einige Unterschiede in der Handhabung von OOXML/ODT-Dateien einerseits und Textdateien andererseits:

1. OOXML/ODT-Dateien lassen sich nur mit einigen wenigen Programmen (*Word*, *Writer*) zuverlässig bearbeiten. Textdateien lassen sich mit einem beliebigen Texteditor, wovon es hunderte gibt, bearbeiten.
2. OOXML/ODT-Dateien enthalten viele Zusatzinformationen für Formatierung, Struktur und zusätzliche Einstellungen. Textdateien haben diesen *Overhead* nicht, da sie nur die eigentlichen Nutzdaten enthalten.
3. OOXML/ODT-Dateien können nur so lange gelesen werden, wie die enstprechenden Programme dazu verfügbar sind. (Wer schon einmal eine alte `.wps`-Datei öffnen wollte, kennt das Problem.) Textdateien, die nur aus einer Reihe von Zeichen bestehen, können immer gelesen werden.

4. OOXML/ODT-Dateien lassen sich nur im „WYSIWYG“-Modus („what you see is what you get“) bearbeiten. Textdateien lassen sich auch als *Textstrom* bearbeiten.

Aus den ersten drei Punkten ergeben sich drei offensichtliche Vorteile für Textdateien. Doch was hat man als Benutzer davon, wenn man eine Datei als *Textstrom* bearbeiten kann?

Anwendungsbeispiel: Wörter in mehreren Artikeln zählen

Angenommen, wir haben eine Reihe von Artikeln; einmal im *OpenDocument Text*-Format (`.odt`) und einmal im Textformat (`.txt`). Nun wollen wir herausfinden, welcher Artikel in Wörtern gemessen der längste ist. Mit unseren *OpenDocument Text*-Dateien verfahren wir folgendermassen:

1. Wir öffnen den ersten Artikel mit *Writer*.
2. Wir gehen auf das Menü *Tools* und wählen den Eintrag *Word Count*.
3. Wir notieren uns den Dateinamen und die Anzahl Wörter dazu.
4. Wir schliessen den Artikel.
5. Wir wiederholen den Vorgang für den nächsten Artikel.

Dieses Vorgehen ist sehr aufwändig. Zudem muss der ganze Vorgang zu einem späteren Zeitpunkt wiederholt werden, falls die Dateien in Zwischenzeit bearbeitet wurden. Schliesslich könnte sich dadurch die Anzahl der Wörter verändert haben.

Mit Textdateien funktioniert das einfacher. Man verwendet einfach folgenden Konsolenbefehl bzw. folgende zwei, durch eine sogenannte *Pipe* verbundenen Befehle:

```
$ wc -w *.txt | sort -n -r
```

Dadurch erhält man folgende Ausgabe:

```
2220 eigenes-bier-brauen.txt
1739 berlinreise.txt
1231 neues-aquarium.txt
 893 im-stau.txt
```

Doch was hat das ganze zu bedeuten? Schauen wir uns die Befehlszeile genauer an:

Der `wc`-Befehl

- Das `wc` zählt die Wörter in einer Datei. Der Programmname ist eine Abkürzung für „word count“.
- Standardmässig gibt `wc` die Anzahl Zeilen, Wörter und Zeichen einer Datei aus. Wir interessieren uns aber nur für die Wörter. Darum geben wir den Parameter `-w` (für „words“) mit.
- Mit `*.txt` geben wir dem Programm sämtliche Textdateien im aktuellen Arbeitsverzeichnis zum Zählen.

Das (`wc -w *.txt`) ist der erste Teil des Befehls. Führte man ihn aus, erhielte man folgende Ausgabe:

```
1739 berlinreise.txt
2220 eigenes-bier-brauen.txt
 893 im-stau.txt
1231 neues-aquarium.txt
```

Die Ausgabe an `sort` weiterleiten

Die Dateien sind alphabetisch und nicht nach Nummern sortiert. Das liegt daran, dass `wc` die Dateien über die Wildcard `*.txt` in alphabetischer Reihenfolge zur Bearbeitung erhält. Darum kommt jetzt der zweite Teil der Befehlszeile zum Zug:

- Zwischen den Befehlen steht das Zeichen `|`. Das ist eine sogenannte *Pipe*, zu Deutsch etwa „Röhre“. Eine Pipe nimmt die Ausgabe eines Programmes entgegen und leitet sie als Eingabe zum nächsten Programm weiter.
- Das nächste Programm ist in diesem Fall `sort`, das Textzeilen in alphabetisch und in aufsteigender Reihenfolge sortiert.
- Da wir aber keine alphabetische, sondern numerische Sortierung wollen (“100“ wäre gemäss alphabetischer Sortierung kleiner als „9“), geben wir den Parameter `-n` an.
- Zudem soll die Reihenfolge nicht aufsteigend (die kleinste Zahl am Anfang) sondern absteigend (die grösste Zahl am Anfang) sein, was wir mit dem Parameter `-r` machen.

Die Ausgabe kürzen

Je mehr Artikel sich in unserem Verzeichnis befinden, desto länger wird die Ausgabe. Bei hunderten Artikel müssten wir bald nach oben scrollen, um zu sehen, welcher am meisten Wörter enthält. Nun könnte man natürlich die Sortierreihenfolge anpassen, sodass der Artikel mit den meisten Wörtern in der letzten Zeile steht. Dazu kann man einfach den Parameter `-r` beim `sort`-Befehl weglassen.

Eleganter ist es, die Ausgabe auf eine bestimmte Länge zu kürzen. Der `head`-Befehl nimmt beliebig viele Zeilen entgegen und gibt die per Parameter definierte Anzahl an Zeilen aus. Um die drei Artikel mit den meisten Wörtern zu ermitteln, kann man den Befehl `head -3` verwenden. Dieser Befehl wird wiederum über eine Pipe mit unserer Befehlszeile verbunden:

```
$ wc -w *.txt | sort -n -r | head -3
```

Die Ausgabe könnte dann folgendermassen aussehen:

```
9431 der-syrienkrieg.txt
8943 jahresrückblick-2016.txt
7131 tour-de-france_doping.txt
```