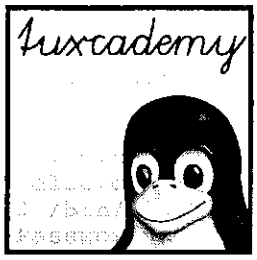

6

Der Stromeditor sed

Inhalt

Lernziele

- Funktion und Einsatzgebiete von sed kennen
- Einfache sed-Skripte formulieren können
- sed in Shellskripten verwenden können

Vorkenntnisse

- Kenntnisse der Shellprogrammierung (etwa aus den vorigen Kapiteln)
- Reguläre Ausdrücke

6.1 Einsatzgebiete

Linux verfügt über ein reiches Angebot an simplen Textbearbeitungswerkzeugen – von `cut` und `grep` über `tr`, `sort` bis zu `join` und `paste`. Werden die Aufgaben der Text-Manipulation jedoch anspruchsvoller, so reichen die klassischen Werkzeuge wie `cut` und `tr` nicht mehr aus. Beispielsweise kann `tr` Ihnen ein »X« für ein »U« vormachen, aber am alten Alchimisten-Traum, »Blei« in »Gold« zu verwandeln, scheitert es wie diese.

Normalerweise würden Sie jetzt einen hinreichend komfortablen Editor starten, um jedes Vorkommen von »Blei« in Ihrer Datei in »Gold« zu transformieren. Es gibt jedoch zwei Gründe, die es nahelegen könnten, einen anderen Weg einzuschlagen.

Erstens könnte es sein, dass Sie die Datei unbeaufsichtigt automatisch verändern wollen, beispielsweise in einem Shell-Skript. Zweitens gibt es Fälle, wo Sie gar keine *Datei* bearbeiten, sondern einen (potenziell unendlichen) Textstrom, etwa in der Mitte einer Pipeline.

sed

Hier bietet sich `sed` (für engl. *stream editor*) an, der es ermöglicht, einen Textstrom nach vorgegebenen Anweisungen zu bearbeiten. In Fällen, wo auch `sed` nicht ausreicht, können Sie weiter aufrüsten und `awk` nehmen (Kapitel 7), oder Sie steigen gleich auf eine Skript-Sprache wie Perl um.

Programmiermodell

Das Programmiermodell von `sed` ist einfach: Das Programm bekommt eine Menge von Anweisungen übergeben, liest seine Eingabe zeilenweise und wendet die Anweisungen da, wo angebracht, auf die Eingabezeilen an. Anschließend werden die Zeilen in gegebenenfalls manipulierter Form ausgegeben (oder auch nicht). Wichtig ist, dass `sed` seine Eingabedatei (falls es eine gibt) nur liest und in keinem Fall verändert; allenfalls werden veränderte *Daten* ausgegeben.

Anweisungen

`sed` akzeptiert Anweisungen entweder eine nach der anderen mit der mehrmals auf der Kommandozeile zulässigen `-e`-Option oder gebündelt in einer Datei, deren Name mit der `-f`-Option übergeben wird. Wenn Sie nur eine einzige Anweisung auf der Kommandozeile übergeben wollen, können Sie sich das `-e` auch sparen.

Innerhalb einer `-e`-Option können Sie mehrere `sed`-Kommandos angeben, wenn Sie sie mit Semikolons trennen.

Grundsätzlich spricht natürlich nichts gegen ausführbare »`sed`-Skripte« der Form

```
#!/bin/sed -f
⟨sed-Kommandos⟩
```

Zeilenadresse

Jede `sed`-Anweisung besteht aus einer Zeilenadresse, die die Zeilen bestimmt, auf die die Anweisung sich bezieht, und der eigentlichen Anweisung, zum Beispiel

```
$ sed -e '1,15y/uU/xX/'
```

`sed`-Anweisungen sind genau ein Zeichen lang, können aber weitere Parameter nach sich ziehen (je nach Anweisung).

6.2 Adressierung

Es gibt verschiedene Möglichkeiten, Zeilen in `sed` zu »adressieren«. Manche Kommandos wirken auf eine Zeile, manche auch auf Bereiche von Zeilen. Einzelne Zeilen können Sie wie folgt auswählen:

Zeilennummern Eine Zahl als Zeilenadresse wird als Zeilennummer interpretiert. Die erste Zeile der Eingabe hat die Nummer 1, und dann wird fortlaufend weitergezählt (auch wenn die Eingabe aus mehreren Dateien besteht).


Mit der -s-Option können Sie sed dazu bringen, jede Eingabedatei getrennt zu betrachten. In diesem Fall beginnt jede Eingabedatei mit einer Zeile Nr. 1.

Eine Zeilenspezifikation der Form *i*~*j* steht für »beginnend bei Zeile *i* jede *j*-te Zeile«. Mit »2~2« würden Sie also jede gerade numerierte Zeile auswählen.

Reguläre Ausdrücke Ein regulärer Ausdruck der Form »/⟨*Ausdruck*⟩/« selektiert alle Zeilen, die auf den Ausdruck passen. »/a.*a.*a/« wählt zum Beispiel alle Zeilen aus, die mindestens drei »a« enthalten.

Letzte Zeile Das Dollarzeichen (»$«) steht für die letzte Zeile der letzten Eingabedatei (auch hier betrachtet -s jede Eingabedatei separat).

Bereiche von Zeilen können Sie adressieren, indem Sie die Einzeladressen beliebig kombinieren, mit einem Komma dazwischen. Der Bereich beginnt mit einer Zeile, die auf die erste Adresse passt, und erstreckt sich von da bis zur ersten Zeile, auf die die zweite Adresse passt. Bereiche können auch in einer Datei anfangen und in einer späteren enden, es sei denn, die Option -s wurde angegeben. Hier sind einige Beispiele für Bereichsadressierung:

`1,10` Dies selektiert die ersten zehn Zeilen der Eingabe

`1,/^$/` Hiermit werden alle Zeilen bis zur ersten Leerzeile ausgewählt. Dieses Idiom ist zum Beispiel nützlich, um den »Kopf« einer E-Mail-Nachricht zu extrahieren (der per Definition immer mit einer Leerzeile aufhört, aber keine Leerzeilen enthalten darf)

`1,$` Dies beschreibt »alle Zeilen der Eingabe«, kann aber meistens wegfallen

`/^BEGIN/,/^END/` Dies beschreibt alle Zeilen von einer, die mit »BEGIN« anfängt, bis zu einer, die mit »END« anfängt (einschließlich).


Ist die zweite Adresse ein regulärer Ausdruck, dann wird nach ihm erst ab der Zeile *unmittelbar nach* der Zeile gesucht wird, bei der der Bereich beginnt. Wenn auch die erste Adresse ein regulärer Ausdruck ist, dann wird, nachdem eine für den zweiten Ausdruck passende Zeile gefunden wurde, wieder nach einer Zeile Ausschau gehalten, die auf den ersten Ausdruck passt – es könnte ja noch eine passende Folge von Zeilen geben.


Wenn die zweite Adresse eine Zahl ist, die eine Zeile *vor* derjenigen Zeile beschreibt, auf die die erste Adresse passt, wird nur die erste passende Zeile ausgegeben.

Sie können alle Zeilen auswählen, die von einer Adresse *nicht* beschrieben werden, indem Sie an die Adresse ein »!« anhängen:

`5!` adressiert alle Zeilen der Eingabe außer der fünften.

`/^BEGIN/,/^END/!` adressiert alle Zeilen der Eingabe, die *nicht* Teil eines BEGIN-END-Blocks sind.

Übungen


6.1 [!1] Betrachten Sie die folgende Datei:

```
ABCDEFG
123 ABC
EFG
456 ABC
123 EFG
789
```

Tabelle 6.1: Von sed unterstützte reguläre Ausdrücke und ihre Bedeutung (Auswahl)

Regulärer Ausdruck	Bedeutung
`[a-d]`	ein Zeichen aus der Menge a, b, c oder d
`[^abc]`	kein Zeichen aus der Menge a, b oder c
`.`	genau ein beliebiges Zeichen (auch Leerzeichen oder Zeilenumbruch)
`*`	Platzhalter für das vorherige Zeichen; mögliche Anzahl 0 bis ?
`?`	Platzhalter für das vorherige Zeichen; mögliche Anzahl 0 bis 1
`^`	Anfang einer Zeichenkette
`$`	Ende einer Zeichenkette
`\<`	Anfang eines Wortes
`\>`	Ende eines Wortes

Welche Zeilen beschreiben die folgenden Adressen? (a) `4`; (b) `2,/ABC/`; (c) `/ABC/,/EFG/`; (d) `$`; (e) `/^EFG/,2`; (f) `/ABC/!`

6.2 [2] Lesen Sie die GNU-sed-Dokumentation und erklären Sie die Bedeutung der (GNU-spezifischen) Adresse »`0,/`⟨*Ausdruck*⟩`/`«.

6.3 sed-Anweisungen

6.3.1 Ausgeben und Löschen von Zeilen

Normalerweise gibt sed jede eingelesene Zeile auf seiner Standardausgabe aus. Mit der Anweisung `d` (engl. *delete*, »löschen«) können Sie Zeilen unterdrücken, so dass sie nicht ausgegeben werden. Beispielsweise wäre

```
sed -e '11,$d'
```

äquivalent zum Kommando head: Nur die ersten 10 Zeilen der Eingabe werden durchgelassen.

Mit der Option `-n` wird das automatische Ausgeben unterdrückt. sed gibt dann nur noch Zeilen aus, wenn Sie die Anweisung `p` (engl. *print*, »drucken«) angeben. Sie können head also auch durch

```
sed -ne '1,10p'
```

nachbilden. Mit einem regulären Ausdruck statt des Zeilenbereichs so können wir grep nachbauen: Dem grep-Aufruf

```
grep '^[^#]' /etc/ssh/sshd_config
```

entspricht

```
sed -ne '/^[^#]/p' /etc/ssh/sshd_config
```

Nochmal zurück zu head: Unsere bisherigen Alternativen haben beide das Problem, dass sie die Eingabe von vorne bis hinten lesen. Wenn man überlegt, dass wir eigentlich nach der zehnten Zeile fertig sind, ist es natürlich irgendwo zwecklos, noch hunderttausend weitere Zeilen zu lesen, nur um sie entweder zu löschen oder nicht auszugeben. Die effizienteste head-Simulation ist tatsächlich

```
sed -e 10q
```

– mit der q-Anweisung (engl. *quit*) wird die sed-Ausführung einfach komplett abgebrochen.

Übungen

6.3 [!1] Wie würden Sie alle Leerzeilen aus der sed-Eingabe löschen?

6.4 [!1] Der populäre Web-Server Apache verwendet zur Konfiguration des Zugriffs auf bestimmte Verzeichnisse in der Datei `httpd.conf` (Verzeichnis kann von Distribution zu Distribution wechseln) Blöcke der Form

```
<Directory /var/www/htdocs>
  …
</Directory>
```

Geben Sie ein sed-Kommando an, das alle diese Blöcke aus der Datei `httpd.conf` extrahiert.

6.5 [2] `head` haben Sie ja nun gesehen. Wie können Sie `tail` mit sed nachbilden?

6.3.2 Einfügen und Verändern

Wirklich seine Muskeln spielen lässt sed aber erst, wenn Sie ihm erlauben, den Textstrom nicht nur zu filtern, sondern auch zu verändern. Um dies zeilenweise zu erledigen, stehen Ihnen die drei Anweisungen »a« (engl. *append*), »i« (engl. *insert*) und »c« (engl. *change*) für das Anhängen nach einer Zeile, das Einfügen vor einer Zeile bzw. den Austausch einer Zeile durch eine andere zur Verfügung:

```
$ fortune | sed -e '1 i >>>' -e '$ a <<<'
>>>
"So here's a picture of reality: (picture of circle with ▷
 ◁ lots of squiggles in it) As we all know, reality is a mess."

  -- Larry Wall (Open Sources, 1999 O'Reilly and Associates)
<<<
```

Dabei erlaubt Ihnen der bei Linux übliche GNU-sed das Schreiben in einer Zeile. Die traditionelle Form ist etwas umständlicher und besser für sed-Skripte geeignet. Im folgenden Beispiel sind die »$« nicht Teil der Datei, sondern werden von cat an das Zeilenende angefügt, um dieses kenntlich zu machen.

```
$ cat -E sed-skript
1i\$
Erste eingefügte Zeile\$
Zweite eingefügte Zeile$
```

sed erwartet (bei der traditionellen Form) nach dem Kommando `a`, `i` oder `c` einen Rückstrich *direkt vor dem Zeilenende*. Sollen mehr als eine Zeile eingefügt werden, so müssen Sie jede Zeile *bis auf die letzte* ebenfalls mit einem Rückstrich direkt vor dem Zeilenende beenden. Aufgerufen wird dieses Skript durch:

```
$ sed -f sed-skript datei
```

Die Kommandos a und i sind »Ein-Adress-Kommandos«: Sie erlauben nur Adressen, die auf eine Zeile passen – keine Bereichsangaben (wobei es durchaus mehrere Zeilen in der Eingabe geben kann, die auf die eine Adresse von a oder i passen und die dann auch alle bearbeitet werden). Diese eine Adresse darf aber alle oben angegebenen Tricks inklusive regulärer Ausdrücke usw. enthalten. Bei c bewirkt eine Bereichsangabe, dass der Bereich als Ganzes ersetzt wird.

Übungen

6.6 [!2] Geben Sie ein sed-Kommando an, das nach jeder Zeile der Eingabe, die nur aus Großbuchstaben und Leerzeichen besteht, eine Leerzeile einfügt. (Stellen Sie sich vor, Sie wollen Überschriften hervorheben.)

6.3.3 Zeichen-Transformationen

Durch das Kommando y können Sie sed dazu bringen, einzelne Zeichen durch andere zu ersetzen. So werden durch

```
$ echo 'unschöner Dateiname' | sed -e 'y/ ö/_?/'
unsch?ner_Dateiname
```

Dateinamen mit Leer- und Sonderzeichen »repariert«. Leider erlaubt y keine Bereichsangaben der Form a-z, so dass es nur ein schwacher Ersatz für tr ist.

Übungen

6.7 [1] Geben Sie ein sed-Kommando an, das in jeder »ungeraden« Zeile alle Kleinbuchstaben zu Großbuchstaben umwandelt.

6.3.4 Suchen und Ersetzen

Mit dem Kommando s (engl. *substitute*) haben Sie die mächtigsten Möglichkeiten. Es erlaubt die Ersetzung eines regulären Ausdrucks durch eine Zeichenkette, deren Zusammensetzung dynamisch verändert werden kann.

Dabei wird der zu ersetzende reguläre Ausdruck wie bei der Zeilenspezifikation in »/.../« eingeschlossen, gefolgt vom Ersetzungstext und »/«, also beispielsweise

```
$ sed -e 's/\<Blei\>/Gold/'
```

Dabei verhindert das Wortendezeichen die versehentliche Erfindung von Wörtern wie »Goldbe«, »Goldchmittel« oder »Goldvergiftung«.

Beachten Sie, dass »\<«, »\>«, «^« und »$« für *leere Zeichenketten* stehen. Insbesondere ist »s« *nicht* das Zeilenende, sondern die leere Zeichenkette »« direkt vor dem Zeilenende; durch »s/$/|/« wird daher am Zeilenende ein » | « eingefügt und nicht etwa das Zeilenende dadurch ersetzt.

Der Ersatztext kann auch vom zu ersetzenden Text abhängen: Neben dem Rückbezug auf geklammerte Teilstücke durch »\1« usw. steht auch »&« zur Verfügung, das für den ganzen Text steht, der vom Suchmuster erfasst wird. Beispiele:

```
$ echo Jedes Wort in Anführungszeichen. | sed -e 's/\([A-Za-z]\+\)/"\1"/g'
"Jedes" "Wort" "in" "Anführungszeichen".
$ echo Jedes Wort in Anführungszeichen. | sed -e 's/[A-Za-z]\+/"&"/g'
"Jedes" "Wort" "in" "Anführungszeichen".
```

Normalerweise ersetzt s nur den ersten Treffer in jeder Zeile. Wird ans Ende der s-Anweisung noch ein »g« (wie »global«) angehängt – so wie hier –, wirkt sie auf alle Treffer in der Zeile. Ein anderer nützlicher Modifikator ist »p« (*print*), der die Zeile nach dem Ersetzen ausgibt (analog zum »p«-Kommando).

Wenn Sie eine Zahl n anhängen, wird nur der n-te Treffer ersetzt: Eine Eingabe der Form

```
Spalte 1    Spalte 2    Spalte 3    Spalte 4
```

mit Tabulatoren zwischen den Spalten wird durch die sed-Anweisung

```
s/[Tab]/\
/2
```

(wobei [Tab] in Wirklichkeit ein »echtes« Tabulatorzeichen[1] sein muss; der Rückstrich am Zeilenende versteckt einen Zeilentrenner) zu

```
Spalte 1    Spalte 2
Spalte 3    Spalte 4
```

Manchmal ist der Schrägstrich als Trennzeichen von Such- und Ersatztext eher lästig, etwa wenn es um Dateinamen geht. Tatsächlich können Sie sich das Trennzeichen praktisch frei aussuchen; Sie müssen nur konsequent sein und dreimal dasselbe benutzen. Nur Leerzeichen und Zeilentrenner sind als Trennzeichen nicht erlaubt.

```
sed 's,/var/spool/mail,/var/mail,'
```

(Bei regulären Ausdrücken als Adressen sind die Schrägstriche leider vorgeschrieben.)

Übungen

6.8 [!1] Geben Sie ein sed-Kommando an, das in der kompletten Eingabe das Wort `gelb` durch das Wort `blau` ersetzt.

6.9 [!2] Geben Sie ein sed-Kommando an, das in allen Zeilen, die mit »A« anfangen, das erste Wort entfernt. (Ein Wort ist für die Zwecke dieser Aufgabe eine Folge aus Groß- und Kleinbuchstaben.)

6.4 sed in der Praxis

Hier noch einige Beispiele dafür, wie Sie sed in komplexeren Shellskripten einsetzen können:

Dateien umbenennen In Abschnitt 4.3 hatten wir uns mit dem Ändern von Dateiendungen bei »vielen« Dateien beschäftigt. Mit sed verfügen wir über ein Werkzeug, das uns das beliebige Umbenennen von vielen Dateien ermöglicht. Ein entsprechendes Kommando könnte etwa so aussehen:

```
$ multi-mv 's/pqr/xyz/' abc-*.txt
```

Unser (einstweilen hypothetisches) `multi-mv`-Kommando bekommt also als erstes Argument ein sed-Kommando übergeben, das dann auf jeden der folgenden Dateinamen angewendet wird.

Als Shellskript formuliert könnte `multi-mv` ungefähr so aussehen:

[1] In der Bash schwierig einzutippen; alle Steuerzeichen können Sie eingeben, indem Sie davor [Strg]+[q] oder [Strg]+[v] tippen.

```
#!/bin/bash
# multi-mv -- benennt mehrere Dateien um

sedcmd="$1"
shift

for f
do
    mv "$f" "$(echo \"$f\" | sed \"$sedcmd\")"
done
```

Dateien überschreiben Wenn Sie beim Thema »Ein- und Ausgabeumlenkung der Shell« aufgepasst haben, wissen Sie längst, dass Konstruktionen wie

```
$ sed … datei.txt >datei.txt
```

nicht das tun, was man naiverweise erwarten könnte: Die Datei `datei.txt` wird gründlich ruiniert. Wenn Sie also eine Datei mit `sed` editieren und das Ergebnis wieder unter dem ursprünglichen Dateinamen abspeichern möchten, müssen Sie zusätzlich arbeiten: Schreiben Sie das Ergebnis zuerst in eine temporäre Datei und benennen Sie diese dann anschließend um, etwa so:

```
$ sed … datei.txt >datei.tmp
$ mv datei.tmp datei.txt
```

Diese recht umständliche Vorgehensweise bietet sich dafür an, über ein Shellskript automatisiert zu werden:

```
$ oversed … datei.txt
```

Dabei beschränken wir uns zunächst auf den einfachen Fall, dass das erste Argument von `oversed` die kompletten Instruktionen für `sed` enthält. Außerdem gehen wir davon aus, dass in einem Aufruf wie

```
$ oversed … datei1.txt datei2.txt datei3.txt
```

die benannten Dateien jeweils einzeln betrachtet und mit ihrem neuen Inhalt überschrieben werden sollen (alles andere ergibt eigentlich keinen Sinn).

Das Skript könnte ungefähr so aussehen:

```
#!/bin/bash
# oversed -- Dateien mit sed "am Platz" editieren

out=/tmp/oversed.$$
sedcmd="$1"
shift

for f
do
    sed "$sedcmd" "$f" >$out
    mv $out "$f"
done
```

Das einzige Bemerkenswerte an diesem Skript ist vielleicht die Bestimmung des Namens für die temporäre Datei (in `out`). Wir achten darauf, dass es keine Kollision mit einem etwaigen anderen, gleichzeitigen `oversed`-Aufruf geben kann, indem wir die aktuelle Prozessnummer (in `$$`) in den Dateinamen einbauen.

Wenn Sicherheit Ihnen wichtig ist, sollten Sie von der »/tmp/oversed.$$«-Methode Abstand nehmen, da der von der Prozessnummer abgeleitete Dateiname vorhersehbar ist. Ein Angreifer könnte das ausnutzen, um Ihrem Programm eine Datei unterzuschieben, die es dann überschreibt. Um sicher zu gehen, könnten Sie zum Beispiel das Programm mktemp verwenden, das statt der Prozessnummer einen garantiert nicht existierenden Dateinamen aus zufälligen Zeichen konstruiert. Es legt die Datei dann auch gleich mit restriktiven Rechten an.

```
$ TMP=$(mktemp -t oversed.XXXXXX)
                                    Erzeugt zum Beispiel /tmp/oversed.z19516
$ ls -l $TMP
-rw------ 1 anselm anselm 0 2006-12-21 17:42 /tmp/oversed.z19516
```

Die nützliche Option »-t« von mktemp tut die Datei in ein »temporäres Verzeichnis«, namentlich zuerst das durch die Umgebungsvariable TMPDIR benannte Verzeichnis, ersatzweise ein Verzeichnis, das Sie mit der Option »-p« benennen können, ersatzweise /tmp.

Wir können das Skript auch noch weiter verfeinern. Zum Beispiel könnten wir prüfen, ob sed überhaupt etwas an der Datei geändert hat, ob also die Ausgabedatei und die Eingabedatei gleich sind. In diesem Fall können wir uns das Umbenennen sparen. Auch wenn die Ausgabedatei leer ist, ist vermutlich etwas schief gegangen, und wir sollten die Eingabedatei nicht überschreiben. In diesem Fall könnte die Schleife aussehen wie

```
for f
do
    sed "$sedcmd" "$f" >$out
    if [ test -s $out ]
    then
        if cmp -s "$f" $out
        then
            echo >&2 "$0: file $f not changed, not overwriting"
        else
            mv "$f" $out
        fi
    else
        echo >&2 "$0: file $f's output empty, not overwriting"
    fi
done
rm -f $out
```

Hier verwenden wir den Dateitestoperator -s, der Erfolg meldet, wenn die angegebene Datei existiert und länger als 0 Bytes ist. Das Kommando cmp vergleicht zwei Dateien Byte für Byte und liefert Erfolg zurück, wenn die beiden Dateien identisch sind; die Option -s unterdrückt die ansonsten unter Umständen produzierte Ausgabe der ersten unterschiedlichen Stelle (die wir hier nicht gebrauchen können).

Seien Sie ein bisschen vorsichtig mit oversed – Sie sollten sicher sein, dass Ihre sed-Kommandos das Richtige tun, bevor Sie sie auf wichtige Dateien loslassen. Beachten Sie auch Übung 6.10.

Im vorigen Beispiel hatten wir angenommen, dass nur das erste Argument des Skripts alles enthält, was Sie sed sagen wollen. Was können Sie tun, damit Sie sed mehrere -e-Optionen oder andere Optionen wie -n oder -r übergeben wollen? Hierfür müssen Sie die Kommandozeile genauer anschauen:

```
sedargs=""
while [ "${1:0:1}" = "-" ]
do
    case "$1" in
      -*[ef]) sedargs="$sedargs $1 $2"
           shift 2 ;;
      -*) sedargs="$sedargs $1"
           shift ;;
      *)   break ;;
    esac
done
```

Diese Schleife prüft die Kommandozeilenargumente: Alles, was mit »-« anfängt und auf »e« oder »f« endet, ist mutmaßlich eine Option der Form »-f datei« oder »-ne '...'«. Die Option nebst dem folgenden Dateinamen oder sed-Kommandoargument wird in sedargs gespeichert und aus der Kommandozeile entfernt (»shift 2«). Entsprechend wird alles, was dann noch übrig ist und mit »-« anfängt, als »gewöhnliche« Option ohne nachfolgendes Argument betrachtet und ebenfalls in sedargs untergebracht. Der sed-Aufruf in der Schleife wird also zu

```
sed $sedargs "$f"
```

Auch dies ist leider noch nicht perfekt (siehe Übung 6.11).

Der unter Linux übliche GNU-sed unterstützt in neueren Versionen als Erweiterung eine Option »-i«, die in etwa die Arbeit von oversed tut. Sie können auch etwas angeben wie »-i.bak«; in diesem Fall wird alles hinter dem »-i« als neue Endung für die Originaldatei angesehen. Bei

```
$ sed -i~ … bla
```

heißt die Eingabedatei später bla~ und die Ausgabe steht in bla.) Sie sollten auf die Anwendung dieser Option allerdings verzichten, wenn das Risiko besteht, dass Ihre Skripte auch auf anderen Unixen laufen sollen als Linux.

Übungen

6.10 [!2] Sorgen Sie dafür, dass in oversed vor dem Umbenennen der sed-Ausgabedatei die ursprüngliche Eingabedatei unter einem geeigneten Namen gesichert wird (Sie könnten zum Beispiel die zusätzliche Endung ».bak« an den Namen anhängen).

6.11 [3] sed erlaubt noch einige weitere Kommandozeilenargumente. Beispielsweise steht »--« (wie auch anderswo) für »Ende der Optionen – was jetzt kommt, ist ein Dateiname, selbst wenn es aussieht wie eine Option«, und es gibt »lange« Optionen der Form »--expression=...« (als Synonym für -e). Passen Sie oversed so an, dass es auch für diese Argumente das »Richtige« tut.

6.12 [3] Mit Ihren neuerworbenen Kenntnissen über sed können Sie das wwm-Skript aus Kapitel 5 etwas weniger umständlich machen, was das Format der Fragendatei angeht. Geben Sie eine Implementierung der question-Funktion an, die eine Fragendatei im Format

```
question 0
?Was hat die Morgenstund im Sprichwort?
-Blei im Hintern
-Eisen im Bauch
```

```
+Gold im Mund
-Silber im Kopf
>50
end
question 50
?Wovor sollten Sie sich beim Badeurlaub in Acht nehmen?
-Hallowal
<<<<<
```

verarbeitet.

Kommandos in diesem Kapitel

`cmp`	Vergleicht zwei Dateien byteweise	`cmp`(1)	101
`mktemp`	Erzeugt einen eindeutigen temporären Dateinamen (sicher)	`mktemp`(1)	100

Zusammenfassung

- `sed` ist ein »Stromeditor«, der seine Standardeingabe liest und modifiziert auf die Standardausgabe schreibt.
- `sed` erlaubt die flexible Adressierung von Eingabezeilen über ihre Position oder ihren Inhalt sowie die Beschreibung von Zeilenbereichen in der Eingabe.
- Diverse Kommandos zur Textmodifikation stehen zur Verfügung.

Literaturverzeichnis

DR97 Dale Dougherty, Arnold Robbins. *sed & awk*. Sebastopol, CA: O'Reilly & Associates, 1997, 2. Auflage. ISBN 1-56592-225-5.
`http://www.oreilly.de/catalog/sed2/`

Rob02 Arnold Robbins. *sed & awk – kurz & gut*. Sebastopol, CA: O'Reilly & Associates, 2002, 2. Auflage. ISBN 3-89721-246-3.
`http://www.oreilly.de/catalog/sedawkrepr2ger/`