Der Stream-Editor Sed: Einführung, Tipps und Tricks

Version 1.26 — 9.12.2020

Hinweise auf inhaltliche Fehler, Schreibfehler und unklare Formulierungen sowie Ergänzungen, Kommentare, Wünsche und Fragen können Sie gerne an den Autor richten:

OSTC Open Source Training and Consulting GmbH
Thomas Birnthaler
E-Mail: tb@ostc.de
Web: www.ostc.de

Inhaltsverzeichnis

1 Einführung

1.1 Eigenschaften

Der *Sed* (*stream editor*) ist ein nicht interaktiver oder **stream-orientierter** Editor. Er erhält seine Kommandos nicht — wie z.B. der *Vi* — direkt über Benutzereingaben, sondern aus einem **Sed-Skript** und führt diese Kommandos dann auf den Eingabedaten durch.

Wie viele andere UNIX-Programme (z.B. `sort`) liest er **Text-Dateien** (zeilenorientierte ASCII-Dateien) zeilenweise von der Standard-Eingabe und gibt das Verarbeitungsergebnis zeilenweise auf der Standard-Ausgabe aus. D.h. er kann als **Filter-Programm** in Pipelines verwendet werden.

1.1.1 Einsatzgebiete

- Suchen und Ersetzen von Texten.
- Automatisiertes Editieren einer oder mehrerer Textdateien.
- Wiederholen der gleichen Bearbeitung auf mehreren Textdateien.
- Erstellen von Konvertierungsprogrammen (z.B. `dos2unix`, `unix2dos`).

1.1.2 Vorteile

- Erlaubt Reguläre Ausdrücke zur Texterkennung und -ersetzung.
- Auf jedem UNIX-System verfügbar.
- Standardisiert.
- Schnell und Speicherplatz sparend (nur eine Eingabezeile gleichzeitig im Speicher).

1.1.3 Nachteile

- Kryptische Syntax.
- Ungewöhnliches Programmiermodell.
- Nur geringe Möglichkeiten zur Ablaufsteuerung.
- Kennt keine Variablen.
- Merken und Wiederverwenden von Textteilen schwierig (nur eine Zeile gleichzeitig im Speicher).

1.2 Literatur

- Stephan Thesing, *sed&awk GE-PACKT*, mitp.
- Arnold Robbins, *sed&awk kurz&gut*, O'Reilly.
- Ulrich Cuber, *Linux Scripting*, Franzis'.
- Dale Dougherty, *Sed & Awk, 2. Edition*, O'Reilly.
- Mortice Kern Systems, *MKS Toolkit Reference Manual - Sed manpage*.
- Daniel Gilly, *UNIX in a Nutshell, 2. Edition*, O'Reilly.

2 Beschreibung

2.1 Aufruf

Entweder wird das *Sed*-Skript `SCRIPT` direkt auf der **Kommandozeile** angegeben (in Hochkommata, um Metazeichen darin vor der Shell zu schützen, mehrere *Sed*-Kommandos sind dabei durch `;` zu trennen):

```
sed [OPT...] 'SCRIPT' [FILE...]
```

Oder das *Sed*-Skript steht in einer **Datei** `SCRIPT` und wird über die **Option** `-f` ausgewählt (kann auch mehrfach angegeben werden):

```
sed [OPT...] -f SCRIPT [FILE...]
```

Der *Sed* arbeitet wie ein klassisches **Filterprogramm**: Die zu verarbeitenden Daten werden automatisch von den Dateien `FILE...` eingelesen. Ohne Datei liest der *Sed* von der **Standard-Eingabe**. Das Ergebnis wird auf die **Standard-Ausgabe** ausgegeben. Als Optionen `OPT...` sind möglich:

Gawk	Option	Bedeutung
	`-f FILE`	*Sed*-Kommandos von Datei `FILE` einlesen [**file**]
	`-e CMD`	Ein *Sed*-Kommando `CMD` (zur Angabe mehrerer Kommandos) [**execute**]
	`-n`	Ausgabe nur bei Kommando `p/P` oder Kommando `s` mit Option `p` [**noprint**]
*	`-E/-r`	Extended Regular Expressions erlauben [**ERE**]
*	`-i [EXT]`	Angebene Dateien direkt verändern [**inline**]
		(Backup mit Extension `EXT` erzeugen bzw. bei leerem `EXT` kein Backup)

Normalerweise gibt der *Sed* nach dem Ausführen des letzten Kommandos die aktuelle Zeile automatisch aus, dies wird durch die Option `-n` [**noprint**] verhindert.

2.1.1 Beispiele für Aufrufe

1. Das folgende Beispiel löscht alle Leerzeilen (Leerzeilen sind entweder völlig leer oder enthalten nur Leerzeichen) aus der Datei `text` und legt das Ergebnis in der Datei `newtext` ab:

   ```
   sed '/^ *$/d' text > newtext
   ```

2. Das folgende Beispiel löscht alle Leerzeichen am Zeilenanfang und -ende und alle Leerzeilen in der Datei `text` und legt das Ergebnis in der Datei `newtext` ab:

   ```
   sed -f rmempty.sed text > newtext
   ```

   Es verwendet dazu das *Sed*-Skript `rmempty.sed` mit folgendem Inhalt (die Reihenfolge der Kommandos ist wichtig):

   ```
   s/^  *//   # Leerzeichen am Zeilenanfang entfernen
   s/  *$//   # Leerzeichen am Zeilenende entfernen
   /^$/d      # Leerzeilen löschen
   ```

3. Die drei Kommandos in diesem *Sed*-Skript sind auch **direkt auf der Kommandozeile** in folgender Form angebbar (gemeinsam durch `;` getrennt oder einzeln mit Option `-e`):

   ```
   sed 's/^  *//;s/  *$//;/^$/d' text > newtext
   sed -e 's/^  *//' -e 's/  *$//' -e '/^$/d' text > newtext
   ```

4. Aus dem *Sed*-Skript `rmempty.sed` kann unter UNIX auch ein **direkt ausführbares Programm** erzeugt werden, indem man es mit einer sogenannten „**Shee-Bang-Zeile**“ versieht und per `chmod␣u+x␣rmempty.sed` ausführbar macht:

   ```
   #!/bin/sed -f
   # Leerzeichen am Zeilenanfang entfernen
   s/^  *//
   # Leerzeichen am Zeilenende entfernen
   s/  *$//
   # Leerzeilen löschen
   /^$/d
   ```

   Das Skript ist dann folgendermaßen aufzurufen:

   ```
   rmempty.sed text > newtext     # Suchpfad PATH enthält Sed-Verzeichnis
   ./rmempty.sed text > newtext   # Skript liegt im aktuellen Verzeichnis
   ```

   Ablauf: Die Shee-Bang-Zeile sorgt dafür, dass der **Unix-Kernel** bei der Ausführung des Skriptes automatisch den *Sed* aus dem Verzeichnis `/bin` startet und das Skript mit Hilfe der Option `-f` an ihn übergibt. Diese Zeile wird vom aufgerufenen *Sed* anschließend ignoriert, da Zeichen nach `#` vom *Sed* als **Kommentar** interpretiert werden (siehe Abschnitt 2.3.5 auf Seite 9).

2.1.2 Exit-Status

Der *Sed* gibt am Skriptende folgende Exit-Stati zurück:

Status	Bedeutung
0	Sed-Skript fehlerfrei ausgeführt
$\neq$0	Während Ausführung trat ein Fehler auf (Syntax)

2.2 Ablauf

2.2.1 Vereinfachte Version

Die prinzipielle Arbeitsweise des *Sed* ist wie folgt:

1. Die Eingabezeilen werden *einzeln* der Reihe nach in einen **Zeilenspeicher**, den sogenannten **Pattern Space** (oder **Puffer**) eingelesen.
2. **Alle Editierkommandos** im *Sed*-Skript werden in der Reihenfolge im Skript auf den Pufferinhalt (die aktuelle Eingabezeile) angewendet.
3. Sie wirken standardmäßig auf *jede* Eingabezeile [**global**], durch **Zeilenadressierung** kann ihre Anwendung aber auf bestimmte Zeilen beschränkt werden.
4. **Ändert** ein *Sed*-Kommando den Pufferinhalt, werden die folgenden *Sed*-Kommandos auf die geänderte Zeile im Puffer angewendet, nicht mehr auf die Originalzeile.
5. Nach dem Abarbeiten des letzten *Sed*-Kommandos wird der aktuelle Pufferinhalt auf der **Standard-Ausgabe** ausgegeben und mit der nächsten Eingabezeile erneut beim ersten *Sed*-Kommando begonnen.

2.2.2 Ergänzungen

1. Die Originaldatei bleibt **unverändert**, da der *Sed* von der Standard-Eingabe liest und auf die Standard-Ausgabe schreibt (außer die Option `-i` [**inline**] wird angegeben, nur im Gawk).
2. Bei Angabe des Schalters `-n` [**noprint**] erfolgt nach dem letzten *Sed*-Kommando keine automatische Ausgabe der aktuellen Zeile, sondern nur bei den Kommandos `p/P` [**print**] oder beim Kommando `s` [**substitute**] mit der Option `p` [**print**].

2.3 Kommandosyntax

Die allgemeine Form eines *Sed*-Kommandos lautet (die Leerzeichen sind nur zu Darstellungszwecken eingefügt, in echten Kommandos sind sie wegzulassen!):

```
[ADR [,ADR]] [!] CMD [ARG...]
```

D.h. sie bestehen aus **Adressen** `ADR`, der **Adress-Negation** `!`, einem (einbuchstabigen) **Editier-Kommando** `CMD` und evtl. **Argumenten** `ARG...` zum Kommando:

- Eine **Adresse** `ADR` kann sein:

Adresse	Bedeutung
n	Zeilennummer n
`$`	Letzte Zeile
`/REGEX/`	Regulärer Ausdruck, der mit dem Zeileninhalt verglichen wird

- Die **Negation** `!` ist Teil der Adresse (nicht des Kommandos) und negiert die Bedeutung der Adresse. D.h. das Kommando `CMD` ist auf alle Zeilen *außer* den durch die Adresse definierten anzuwenden.
- Der **Kommando** `CMD` besteht aus einem **einzelnen Zeichen**, das als Abkürzung für den Kommandonamen steht (z.B. `d` [**delete**]).
- **Argumente** `ARG` sind:
  - ▷ **Optionen** des Kommandos `s` [**substitute**].
  - ▷ **Dateinamen** der Kommandos `r` [**replace**] oder `w` [**write**].
  - ▷ **Labels** (Marken) der Kommandos `b` [**break**] oder `t` [**test**].
  - ▷ **Buchstabenlisten** des Kommandos `y` [**yank**].

2.3.1 Adressangaben

Durch die unterschiedlichen Formen der Adressangabe wird die Anwendung des zugehörigen Kommandos folgendermaßen eingeschränkt:

Form	Anwendung auf
—	Jede Eingabezeile (*keine Adresse*)
`ADR`	Jede Zeile, auf die Adresse zutrifft (*die Kommandos* `a`, `i`, `r`, `q` *und* `=` *akzeptieren nur diese Form!*)
`ADR,ADR`	Von der ersten Zeile, auf die die 1. Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die 2. Adresse zutrifft (*wiederholt sich beliebig oft*)
`ADR!`	Alle Zeilen, auf die Adresse *nicht* zutrifft
`ADR,ADR!`	Alle Zeilen *außer* der ersten Zeile, auf die die 1. Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die 2. Adresse zutrifft (*wiederholt sich beliebig oft*)

Beispiele:

–	Alle Zeilen
`100`	Zeile 100
`20,40`	Zeilen 20–40
`$`	Die letzte Zeile
`1,$`	Alle Zeilen (*nicht notwendig*)
`!100`	Alle Zeilen außer 100
`!20,40`	Alle Zeilen außer 20–40
`/BSD/`	Alle Zeilen, die den Text `BSD` enthalten
`/[Uu][Nn][Ii][Xx]/`	`...` `Unix` enthalten (*Groß/Kleinschreibung egal*)
`/^BEGIN/,/^END/`	Alle Zeilen zwischen jedem Zeilenpaar, das `BEGIN` und `END` am Zeilenanfang enthält (*einschließlich*)
`/error:/!`	Alle Zeilen die *nicht* `error:` enthalten
`/BEGIN/,/END/!`	Alle Zeilen *außerhalb* der Zeilenpaare, die `BEGIN` und `END` irgendwo enthalten

2.3.2 Reguläre Ausdrücke

Folgende Metazeichen sind im *Sed* in Regulären Ausdrücken (d.h. in Adressen und im Substitute-Kommando) möglich:

Symbol	Bedeutung
x	Zeichen x (*kein Metazeichen*)
`.`	1 beliebiges Zeichen
x`*`	0–∞ Wiederholungen des Teils x davor
`^`	Zeilenanfang
`$`	Zeilenende
`\`x	Metazeichen x *quotieren* (steht für sich selbst)
`[`abc`]` `[`a–z`]`	Menge von Zeichen (`[`a–z`]` = **Zeichenbereich**)
`[^`abc`]` `[^`a–z`]`	Negierte Menge von Zeichen (alle außer diesen)
`\(...\)`	Zeichenkette merken (in `\1`..`\9`)
`\n`	Zeilenvorschub [**newline**]
`\/`	Begrenzerzeichen
`\\`	Backslash
x`\{`m`,`n`\}`	m–n Wiederholungen des Teils x davor
x`\{`m`,\}`	m–∞ Wiederholungen des Teils x davor
x`\{`m`\}`	m Wiederholungen des Teils x davor (genau)

Folgende Metazeichen sind im Ersetzungsteil des Substitute-Kommandos möglich:

Symbol	Bedeutung
`\`n	n-te per `\(...\)` gemerkte Zeichenkette (n=1..9)
`&`	Suchmuster einsetzen (`&` selbst per `\&` angeben)

Durch `\(...\)` kann der aktuelle Treffertext von Musterteilen **gespeichert** und im Ersetzungsteil wiederverwendet werden, die Klammern dürfen auch verschachtelt sein. Für die **Nummer** einer Referenz `\`n auf den Treffertext eines Klammernpaars sind die öffnenden Klammern `\(` von links nach rechts beginnend mit 1 durchzuzählen.

2.3.3 Begrenzungszeichen

Kommt das **Standard-Begrenzungszeichen** `/` im Regulären Ausdruck selbst vor, so muss es durch `\` quotiert werden: `\/`. Anstelle des Standard-Begrenzungszeichen `/` um Reguläre Ausdrücke (siehe Abschnitt 2.3.2 auf Seite 8) in Adressen oder Such- und Ersetzungsmustern kann jedes beliebige andere Zeichen verwendet werden (z.B. `@ #`). Dieses Verhalten ist nützlich, wenn das Begrenzungszeichen `/` selbst im Regulären Ausdruck mehrfach vorkommt und die dann notwendige Schreibweise `\/` diesen schwer lesbar macht.

Beispiel: C++-Kommentare der Form `//...` durch C-Kommentare der Form `/*...*/` ersetzen, einmal unter Verwendung des Standard-Begrenzungszeichen `/` und einmal der Begrenzungszeichen `@` bzw. `#` (`s` [**substitute**]).

```
s/\/\/\(.*\)$/\/*\1*\//
s@//\(.*\)$@/*\1*/@
s#//\(.*\)$#/*\1*/#
```

Noch einfacher wird diese Ersetzung, wenn `&` als Platzhalter für das gesamte gefundene Muster verwendet wird:

```
s#//.*$#/*&*/#
```

Hinweis: Allerdings entsteht hierdurch als Ergebnis ein Kommentar der Form `/*//...*/`, d.h. der ursprüngliche Kommentar wird **komplett** in `/*...*/` eingebettet.

2.3.4 Blöcke

Um Adressangaben zu verschachteln oder zu einer Adresse mehrere Kommandos anzugeben, können geschweifte Klammern `{...}` verwendet werden (**Block**). Die öffnende Klammer muss am Zeilenende stehen, die schließende Klammer muss alleine auf einer Zeile stehen:

```
/ADR/,/ADR/ {
   CMD1
   CMD2
   ...
}
```

Achtung: Nach den Klammern dürfen *keine* Leerzeichen stehen!

2.3.5 Kommentare und Leerzeilen

Sed-Kommentare müssen auf einer Zeile für sich stehen, werden durch `#` eingeleitet und erstrecken sich bis zum Zeilenende. Es ist also keine Kommentar *nach* einem Kommando in der gleichen Zeile erlaubt.

Leerzeilen werden vom *Sed* ignoriert.

2.4 Kommandos

2.4.1 Zeilen einfügen, ändern und löschen

Folgendermaßen lassen sich Zeilen einfügen, anhängen, ändern und löschen:

Kommando	Bedeutung
`i\` `ZEILE...`	Nachfolgende Zeilen *vor* aktueller Zeile einfügen [**insert**] (*alle Zeilen außer der letzten sind mit* `\` *abzuschließen*)
`a\` `ZEILE...`	Nachfolgende Zeilen *nach* aktueller Zeile einfügen [**append**] (*alle Zeilen außer der letzten sind mit* `\` *abzuschließen*)
`c\` `ZEILE...`	Aktuelle Zeile durch nachfolgende Zeilen ersetzen [**change**] (*alle Zeilen außer der letzten sind mit* `\` *abzuschließen*)
`d`	Aktuelle Zeile löschen [**delete**]

- Von den Kommandos `i` und `a` eingefügter Text ist nicht im Pattern Space vorhanden (und daher nicht durch Kommandos bearbeitbar). Das Ergebnis dieses Kommandos wird erst ausgegeben, wenn die Kommandoliste vollständig abgearbeitet ist, ganz egal, was mit der aktuellen Zeile passiert.
- Kommando `c` ersetzt alle adressierten Zeilen durch den eingefügten Text. Der Inhalt des Pattern Space wird gelöscht, weitere Editierkommandos können nicht mehr darauf angewendet werden.
- Kommando `d` liest eine neue Eingabezeile ein und die Verarbeitung beginnt erneut am Anfang der Kommandoliste.

2.4.2 Ersetzen von Text

Teile einer Zeile (die durch Reguläre Ausdrücke gematcht werden) durch beliebigen Text oder einzelne Zeichen durch andere Zeichen ersetzen:

Kommando	Bedeutung
`s/REGEX/SUBST/` `y/abc/ABC/`	In aktueller Zeile `REGEX` durch `SUBST` ersetzen [**substitute**] In aktueller Zeile Zeichen `a` durch `A`, `b` durch `B` [**yield**/**yank**] und `c` durch `C` ersetzen (analog dem UNIX-Kommando `tr`)

Die allgemeine Form des *Sed*-Ersetzungs-Kommandos `s` [**substitute**] lautet (die Leerzeichen sind nur zu Darstellungszwecken eingefügt, in echten Kommandos sind sie wegzulassen):

```
[ADR [,ADR]] [!] s /REGEX/SUBST/ [OPTIONS]
```

`REGEX` ist ein Regulärer Ausdruck, zu dem passende Zeichenketten in den durch den **Adressbereich** `[ADR [,ADR]]` festgelegten Zeilen gesucht werden, `SUBST` ersetzt diese gefundenen Zeichenketten. Als Besonderheit kann in `REGEX` das Metazeichen `\n` zum Matchen von Zeilenvorschüben verwendet werden.

Das Substitute-Kommando kennt folgende Optionen (ohne eine der Optionen n oder `g` wird die *erste* zum Muster passende Zeichenkette ersetzt):

Option	Bedeutung
n	n-tes Auftreten von `REGEX` durch `SUBST` ersetzen [**number**]
`g`	*Jedes* Auftreten von `REGEX` durch `SUBST` ersetzen [**global**]
`p`	Zeile nach *erfolgreicher* Ersetzung ausgeben [**print**]
`w FILE`	Zeile nach der Ersetzung auf Datei `FILE` ausgeben [**write**]

- Durch die **Metaklammern** `\(` und `\)` in `REGEX` eingeklammerte Teilketten können in `SUBST` durch `\1`, `\2`, ... angesprochen werden, sie werden durch den Inhalt der korrespondierenden Metaklammer ersetzt. Um die Nummer für die Referenz `\`n auf den mit einem Klammernpaar gemerkten Musterteil zu ermitteln, sind die öffnenden Klammern `\(` von links nach rechts beginnend mit 1 durchzuzählen.
- `\n` in `REGEX` passt auf einen Zeilenvorschub (der nur durch die Kommandos `G` [**Get**], `H` [**Hold**] und `N` [**Next**] entstehen kann).
- `&` in `SUBST` steht für die gesamte von `REGEX` gematchte Teilkette.
- Ein Zeilenvorschub in `SUBST` muss durch Maskieren des echten Zeilenvorschubs per `\` erzeugt werden.
- Als Zahl n ist der Bereich 1–512 erlaubt (Default: 1).
- Maximal 10 Dateien können gleichzeitig zum Schreiben geöffnet werden.

2.4.3 Zeilenausgabe/information

Die Kommandos `p` und `l` dienen zum Vervielfachen von Zeilen oder zur Ausgabe von Zeilen, wenn die Option `-n` (*noprint*) angegeben wurde:

Kommando	Bedeutung
`p`	Aktuelle Zeile ausgeben [**print**]
`l`	Aktuelle Zeile ausgeben (*Control-Zeichen als ASCII-Code*) [**list**]
`=`	Nummer der aktuellen Zeile ausgeben

2.4.4 Ein/Ausgabe und Abbruch

Folgende Kommandos dienen zum Abbruch der Verarbeitung einer Zeile bzw. des kompletten *Sed*-Skripts, fügen den Inhalt einer externen Datei ein oder geben eine Zeile auf eine externe Datei aus:

Kommando	Bedeutung
`n`	Aktuelle Zeile ausgeben und nächste Zeile einlesen [**next**]
`q`	Aktuelle Zeile ausgeben und *Sed*-Skript abbrechen [**quit**]
`r FILE`	Inhalt der Datei `FILE` nach aktueller Zeile einfügen [**read**]
`w FILE`	Aktuelle Zeile auf Datei `FILE` ausgeben [**write**]

- Nach dem Kommando `n` wird nicht wieder an den Anfang der Kommandoliste gesprungen, sondern beim nächsten Kommando weitergearbeitet.
- `w` erstellt die Datei `FILE`, falls sie noch nicht existiert, sonst wird sie überschrieben. Es wird sofort ausgeführt, nicht erst wenn der Pattern Space ausgegeben wird.
- Zwischen den Kommandos `r` und `w` und dem Dateinamen `FILE` muss *genau ein* Leerzeichen stehen.
- Maximal 10 Dateien können gleichzeitig zum Schreiben geöffnet werden.

2.4.5 Test und Verzweigung

Über Test- und Sprungkommandos mit **Labels** (**Marken**) kann der Verarbeitungsfluss gesteuert werden. Mit `b` [**branch**] wird in jedem Fall gesprungen, mit `t` [**test**] nur nach einer erfolgreichen Substitution vorher.

Kommando	Bedeutung
`b[LABEL]`	Zu Marke `LABEL` (oder Skriptende) springen [**branch**]
`t[LABEL]`	Zu Marke `LABEL` (oder Skriptende) springen [**test**], wenn seit dem letzten Einlesen oder seit dem letzten `t`-Kommando eine Ersetzung erfolgte
`:LABEL`	Marke `LABEL` für `b`- oder `t`-Kommando (*max. 7 Zeichen*)

- Am besten vergleicht man `t` mit einem `case`-Statement in C. Hat eine Ersetzung stattgefunden (d.h. hat ein Fall zugetroffen), so wird der Sprung durchgeführt, ansonsten geht es beim nächsten Kommando weiter.

2.4.6 Zwischenpuffer

Neben dem Pattern Space kennt der *Sed* noch einen zweiten Zeilenspeicher, den **Hold Space** (oder **Zwischenpuffer**). Der Inhalt der beiden Speicher kann auf folgende Arten ausgetauscht werden:

Kommando	Bedeutung
`h`	Aktuelle Zeile in Zwischenpuffer kopieren [**hold**]
`g`	Aktuelle Zeile durch Zwischenpuffer ersetzen [**get**]
`x`	Aktuelle Zeile und Zwischenpuffer vertauschen [**exchange**]

- Im Hold Space kann eine **Kopie** der Originalzeile aufgehoben werden, während sie im Pattern Space editiert wird.

2.4.7 Mehrzeilenverarbeitung

Eine Besonderheit des *Sed* ist, dass er auch Zeilenvorschübe im Puffer enthalten, erkennen und ersetzen kann. Zeilenvorschübe können *nur* durch eines der Kommandos `G` [**Get**],

`H` [**Hold**] oder `N` [**Next**] entstehen, die Zeilen aneinanderhängen und dabei automatisch einen Zeilenvorschub dazwischen einfügen. Die beiden Kommandos `P` [**Print**] und `D` [**Delete**] löschen den Zeilenteil bis einschließlich dem ersten `\n`.

Kommando	Bedeutung
`G`	Zwischenpuffer an aktuelle Zeile anhängen [**Get**]
`H`	Aktuelle Zeile an Zwischenpuffer anhängen [**Hold**]
`N`	Nächste Zeile an aktuelle Zeile anhängen [**Next**]
`D`	Aktuelle Zeile bis zum 1. Newline löschen [**Delete**]
`P`	Aktuelle Zeile bis zum 1. Newline ausgeben (und löschen) [**Print**]

- Nach den Kommandos `G`, `H`, `N` und `P` wird nicht wieder an den Anfang der Kommandoliste gesprungen, sondern beim nächsten Kommando weitergearbeitet.
- Nach dem Kommando `D` wird die Verarbeitung wieder am Anfang der Kommandoliste gestartet. Wird die ganze Zeile gelöscht (weil sie keinen Zeilenvorschub mehr enthält), dann wird analog zum Kommando `d` eine neue Eingabezeile eingelesen.
- Die Kommandos `H` bzw. `G` fügen auch dann einen Zeilenvorschub ein, wenn der Hold Space bzw. Pattern Space leer ist.
- Das Metazeichen `\n` kann verwendet werden, um im Pattern-Space Zeilenvorschübe zu matchen (die aus den Kommandos `G`, `H` oder `N` resultieren können). Ein Zeilenvorschub am Zeilenende ist damit nicht matchbar, hierzu dient das Metazeichen `$`.

3 Beispielprogramme

3.1 Teil 1

- Mehrfache Leerzeichen in der ganzen Zeile (`g` [**global**]) durch ein einziges ersetzen:

```
sed 's/  */ /g'
```

  Achtung: Das Substitutionskommando `s/␣*/␣/g` funktioniert nicht! Er fügt vor jedem Zeichen (ungleich einem Leerzeichen) ein Leerzeichen ein, da das Muster `␣*` auch auf *Nichts* passt.

- Die ersten beiden Zeichen **vertauschen**:

```
sed 's/^\(.\)\(.\)/\2\1/'
```

- Zeilen, die mit `#` anfangen, mit sich selbst **verketten** (d.h. doppelt aneinanderhängen. `^` und `$` um `.*` sind nur der Klarheit halber angegeben, eigentlich sind sie überflüssig, da immer die längstmögliche Zeichenkette gematcht wird):

```
sed '/^#/s/^.*$/& &/'
```

- Leerzeichen und Tabulatoren am Zeilenanfang und -ende entfernen und alle Leerzeilen löschen (`^I` ist durch `Ctrl-V TAB` einzugeben):

```
s/^[ ^I]*//           # Leerraum am Zeilenanfang entfernen
s/[ ^I]*$//           # Leerraum am Zeilenende entfernen
/^$/d                 # Leerzeilen löschen
```

- In einer Zeile geschrieben sieht die gleiche Kommandofolge wie folgt aus:

```
sed 's/^[ ^I]*//;s/[ ^I]*$//;/^$/d'
```

- Alternativ kann die Kommandofolge auch so angegeben werden (`-e` [**execute**]):

```
sed -e 's/^[ ^I]*//' -e 's/[ ^I]*$//' -e '/^$/d'
```

3.2 Teil 2

- Alle Zeilen löschen, die in der 1. Spalte eine 1 oder in der 3. Spalte eine 3 enthalten:

```
/^1/d
/^..3/d
```

- Eine Textdatei enthält in der 1. Spalte jeder Zeile ein **Drucksteuerzeichen**, das eine der folgenden Aktionen auslösen soll und dann entfernt wird:

Code	Aktion
2	Vorher Leerzeile ausgeben
1	Vorher neue Seite beginnen (`Ctrl-L` drucken)
Sonst	Zeile normal ausgeben

Das *Sed*-Skript zur Realisierung dieser Vorgaben lautet:

```
/^2/i\              # "2" am Zl.anfang -> Leerzeile davor einfügen
                    # "2" am Zl.anfang -> Leerzeile davor einfügen
/^1/s/^1/1^L/       # "1" am Zl.Anfang -> durch 1 Ctrl-L ersetzen
s/^.//              # 1.Spalte jeder Zeile löschen
```

3.3 Teil 3

- Aus einer Datei mit spaltenorientiertem Aufbau soll eine **Word-Steuerdatei** erstellt werden. Der Satzaufbau der Eingabedatei lautet:

Spalte	Länge	Bedeutung
1- 1	1	Geschlecht (1=männlich, 2=weiblich)
2-21	20	Nachname
22-36	15	Vorname
37-..	..	Bemerkung

Das Word-Format hat folgenden Aufbau: 5 Felder `Nachname`, `Vorname`, `Anrede1` (Singular), `Anrede2` (Plural) und `Bemerkung` pro Zeile, nur so breit wie nötig und durch `;` getrennt, in der 1. Zeile stehen die Feldnamen. Folgendes *Sed*-Skript wandelt die spaltenorientierte Form in die von Word benötigte Form um:

```
1i\                                                 # Kopfzeile ergänzen
Nachname;Vorname;Anrede1;Anrede2;Bemerkung          # Kopfzeile ergänzen
s/^\(.\)\(.\{20\}\)\(.\{15\}\)/\2;\3;\1;/           # 3 Spalten vertauschen
s/  *;/;/g                                          # Leerraum vor ; weg
s/  *$//                                            # Leerraum am Zl.ende weg
s/^\([^;]*;[^;]*;\)1;/\1Herr;Herrn;/                # 3.Sp -> Anrede1+2
s/^\([^;]*;[^;]*;\)2;/\1Frau;Frau;//                # 3.Sp -> Anrede1+2
```

- Führt man erst die Ersetzung des Geschlechts durch die Anrede durch, so ergibt sich ein etwas einfacheres Skript:

```
1i\                                                 # Kopfzeile ergänzen
Nachname;Vorname;Anrede1;Anrede2;Bemerkung          # Kopfzeile ergänzen
s/^1;/HerrHerrn/                                    # 1.Sp -> Anrede1+2
s/^2;/FrauFrau /                                    # 1.Sp -> Anrede1+2
s/^\(.\{4\}\)\(.\{5\}\)\(.\{20\}\)\(.\{15\}\)/\3;\4;\1;\2;/   # 4 Sp vert.
s/  *;/;/g                                          # Leerraum vor ; weg
s/  *$//                                            # Leerraum am Zl.ende weg
```

- Die Zeilen einer Eingabedatei **paarweise vertauschen**, d.h. in der Reihenfolge 2, 1, 4, 3, 6, 5, ... ausgeben (*mit* `sed -n` *ausführen, die Anzahl der Zeilen muss gerade sein, sonst geht die letzte Zeile verloren!*):

```
h   # Aktuelle Zl. nach Hold-Space
n   # Nächste Zeile lesen
p   # Akt. Zeile ausgeben
g   # Zl. aus Hold-Space holen
p   # Akt. Zeile ausgeben
```

3.4 Teil 4

- Einen ASCII-Text **verschlüsseln**, indem jeder Buchstabe um 13 Stellen im Alphabet verschoben wird. Die nochmalige Anwendung dieses Skripts entschlüsselt den Text wieder (`rot13`-Verschlüsselung):

```
y/abcdefghijklmnopqrstuvwxyz/nopqrstuvwxyzabcdefghijklm/
y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/NOPQRSTUVWXYZABCDEFGHIJKLM/
```

- Dateinamen mit Großbuchstaben im Namen in Kleinschreibung **umsetzen**. Dazu aus einer Liste der Dateinamen (per `ls␣-1` erhält man pro Zeile einen Dateinamen) ein Shell-Skript mit `mv`-Befehlen der Form `mv␣OLDNAME␣newname` erzeugen (Tipp: Zwischenpuffer verwenden):

```
/[A-Z]/!d   # Kein Grossbuchstabe? -> ignorieren, nächste Zeile
h           # Dateinamen in Hold-Space kopieren
y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz/   # GROSS->klein
H           # Kleinen Namen mit Originalname im Hold-Space vertauschen
g           # Kleinen Namen aus Hold-Space anhängen (mit \n dazwischen!)
s/\n/ /     # "\n" in Leerzeichen umwandeln
s/^/mv /    # Vorne "mv " ergänzen
```

- Dateinamen der Form `us*.txt` **umwandeln** in `sh*.txt`. Dazu aus einer Liste der Dateinamen (per `ls␣-1` erhält man pro Zeile einen Dateinamen) ein Shell-Skript mit `mv`-Befehlen der Form `mv␣OLDNAME␣NEWNAME` erzeugen durch folgendes Kommando:

  ```
  ls -1 us*.txt | sed -n '...' > rename.sh
  ```

  Dieses Skript ist ausführen per `sh␣rename.sh`, der *Sed*-Teil lautet:

  ```
  h           # Dateinamen in Holdspace kopieren
  s/^us/sh/   # Zeilenanfang "us" gegen "sh" austauschen
  H           # Konvertierten Namen mit Originalname im Hold-Space vertauschen
  g           # Konvertierten Namen aus Hold-Space anhängen (mit \n dazwischen)
  s/\n/ /     # "\n" in Leerzeichen umwandeln
  s/^/mv /    # Vorne "mv " ergänzen
  ```

- Aus einem C++-Programm alle **Kommentare entfernen**. Kommentare beginnen entweder mit `//` und reichen bis zum Zeilenende oder sie beginnen mit `/*` und reichen bis zum nächsten `*/` (das nicht in der gleichen Zeile stehen muss). Da das Standard-Begrenzungszeichen `/` sehr häufig im Suchmuster vorkommt, wird `@` als Begrenzungszeichen verwendet:

  ```
  :loop
  s@/\*.*\*/@@g    # /*...*/ entfernen
  @/\*@ {          # Bei /* alleine (Blockanfang):
     s@/\*.*@/*@   #    /*... verkürzen zu /*
     N             #    Zeile anhängen
     bloop         #    zum Anfang springen
  }                # (Blockende)
  s@//.*@@         # //... entfernen
  ```

- Alle in einer C-Quelldatei in Anführungszeichen `"..."` stehenden Zeichenketten **extrahieren** und untereinander auflisten (ohne Anführungszeichen). In einer Zeile können mehrere Zeichenketten vorkommen, jede Zeichenkette beginnt und endet in derselben Zeile, innerhalb einer Zeichenkette kann die Folge `\"` für ein explizites `"` stehen:

  ```
  /^#/d               # Zeile beginnend mit # löschen (Präprozessoranw.)
  :loop               # Marke "loop"
  /".*"/!d            # Zeile ohne "..." löschen
  h                   # Zeile in Hold-Space kopieren
  s/^[^"]*//          # Links bis zum ersten " löschen
  s/\([^\\]\)".*/\1/  # Rechts ab einschl. letztem " löschen (\" aufheben)
  s/"//               # Erstes " von links ersetzen
  ```

```
s/\\"/"/g            # Alle \" -> " umwandeln
p                    # Zeile ausgeben
g                    # Zeile aus Hold-Space holen
s/^[^"]*"//          # Links bis einschließlich erstem " entfernen
:rmquote             # Marke "rmquote"
s/^[^"]*\\"//        # Links bis erstem \" entfernen
trmquote             # Zu Marke "rmquote" springen, wenn vorher Ersetzung
s/^[^"]*"//          # Links bis einschließlich erstem " entfernen
bloop                # Zu Marke "loop" springen
```

4 Epilog

4.1 Der mittelalterliche Kopist

Die Arbeitsweise des *Sed* kann — vielleicht etwas verschroben — mit der Arbeitsweise eines **Kopisten** in einem mittelalterlichen Kloster verglichen werden, der sich auf Anweisung seines Abtes mit dem Übertragen einer alten Handschrift in einen Folianten abmüht. Seine Vorgehensweise wird durch eine Reihe von Beschränkungen verkompliziert:

- Das Originalmanuskript liegt im **ersten Raum** auf, die Anweisungen des Abtes zum Kopieren der Handschrift sind im **zweiten Raum** zu finden und Feder, Tinte und Foliant sind nur in einem **dritten Raum** vorhanden.
- Originalmanuskript und Anweisungen sind in Stein gehauen und können daher nicht bewegt werden. Die Tinte trocknet sehr langsam, daher darf der Foliant ebenfalls nicht bewegt werden.

Der pflichtbewusste und geduldige Kopist kann die ihm aufgetragene Kopie nur durchführen, indem er von Raum zu Raum geht und dabei jeweils genau **eine Zeile** gleichzeitig bearbeitet. Sobald er den **ersten Raum** mit dem Originalmanuskript betritt, nimmt er aus seiner Kutte einen **Schmierzettel**, um die erste Zeile des Manuskripts abzuschreiben. Dann geht er in den **zweiten Raum** mit den Editieranweisungen des Abtes. Er liest jede der Anweisungen von oben nach unten, um festzustellen, ob sie auf die einzelne Zeile anzuwenden ist, die er auf seinen Schmierzettel gekritzelt hat.

Jede Anweisung ist in einer speziellen Notation (*sed* [**scripsit ex deus**]) geschrieben und besteht aus zwei Teilen: einem **Muster** und einer **Aktion**.

- Wie es sich für ein mittelalterliches Kloster gehört, sind die **Aktionen** in einer geheimnisvollen Symbolsprache geschrieben, sodass nur Eingeweihte wie der Mönch, aber kein Außenstehender sie verstehen können.
- Zur weiteren Verschleierung wird für die **Muster** eine uralte und archaische Notation namens `REGEX` verwendet, die angeblich von GOTTes Sohn selbst stammen soll (*regius ex crucis*). Um diese Notation überhaupt verstehen zu können, ist ein jahrelanges Studium alter Handschriften erforderlich. Hat man sie jedoch einmal verinnerlicht, dann erscheint sie (wie alles GÖTTliche) ganz selbstverständlich, wie zahlreiche Adepten vergangener (und heutiger Zeiten) nicht müde werden zu versichern.

Der Kopist liest die erste Anweisung und vergleicht ihr Muster mit der Zeile auf seinem Schmierzettel. Passt es nicht, so muss er nichts tun und kann zur nächsten Anweisung schreiten. Passt es, dann befolgt er die Aktion(en) dieser Anweisung.

Da sein Kopf sowieso schon mit so vielen Dingen angefüllt ist, führt er die Editieraktionen jedesmal sofort auf seinem Schmierzettel durch, bevor er zur nächsten Anweisung weitergeht. Er liest jedesmal gründlich die ganze Folge von Anweisungen durch, nicht nur bis zur ersten Anweisung, die passt. Da er die Editieraktionen sofort durchführt, vergleicht er immer die letzte Version der Zeile auf seinem Schmierzettel mit dem nächsten Muster, die Originalzeile versinkt in der Vergessenheit.

Kommt er schließlich zum Ende der Anweisungsliste und hat alle notwendigen Editieraktionen auf seinem Schmierzettel durchgeführt, so geht er in den **dritten Raum** und kopiert die Zeile darauf in den Folianten (dafür braucht er keine Anweisung, das macht er ganz von alleine). Danach kehrt er in den ersten Raum zurück und schreibt die nächste Zeile auf einen neuen Schmierzettel. Er geht wieder in den zweiten Raum, liest der Reihe nach alle Anweisungen und führt die passenden aus, bevor er wieder in den dritten Raum geht usw.

Dies ist seine übliche Vorgehensweise, solange ihm nichts anderes gesagt wird. Allerdings kann ihm — bevor er überhaupt mit dem Kopieren anfängt — gesagt werden, dass er *nicht* jede Zeile automatisch kopieren soll (per Option `-n` [**noprint**]). In diesem Fall muss er auf die Anweisung `p` [**print**] warten, die ihm sagt, er solle *jetzt* schreiben. Findet er bis zum Ende der Anweisungsliste keine derartige Anweisung, dann wirft er den Schmierzettel weg und fährt mit der nächsten Zeile aus dem Manuskript fort. Egal ob er die Zeile automatisch kopieren soll oder nicht, er geht die Anweisungsliste immer von der ersten bis zur letzten Anweisung durch (Geduld ist eine der 7 Haupttugenden eines Mönches).

Was für Anweisungen hat der Kopist zu befolgen? Zunächst einmal kann eine Anweisung kein, ein oder zwei `REGEX`-Muster enthalten:

- Ist keines angegeben, dann wird die Aktion auf **jede Zeile** angewendet.
- Ist ein `REGEX`-Muster angegeben, dann wird die Aktion auf **jede passende Zeile** angewendet.
- Folgt auf ein `REGEX`-Muster das Zeichen `!` (**Negation**], so wird die Aktion auf *alle nicht zum Muster passenden Zeilen* angewendet.
- Sind zwei durch Komma getrennte `REGEX`-Muster angegeben, dann wird die Aktion **innerhalb allen Zeilenbereichen** folgender Form durchgeführt: Von der ersten zum 1. Muster passenden Zeile bis einschließlich der ersten zum 2. Muster passenden Zeile (*wiederholt*).
- Folgt auf zwei durch Komma getrennte `REGEX`-Muster das Zeichen `!` (**Negation**), dann wird die Aktion **außerhalb allen Zeilenbereichen** folgender Form durchgeführt: Von der ersten zum 1. Muster passenden Zeile bis einschließlich der ersten zum 2. Muster passenden Zeile (*wiederholt*).

Zur inneren Kontemplation **numeriert** der Mönch die Zeilen auf seinem Schmierzettel im Kopf durch. Da in einem Kloster alles seinen tieferen Sinn hat, kann ein Muster auch aus

einer Zeilennummer statt einem `REGEX`-Muster bestehen. Auf die dazu passenden Zeilen sind dann ebenfalls die Aktionen anzuwenden.

Wie kann der Kopist die von zwei Mustern begrenzten *Zeilenbereiche* bearbeiten, obwohl er sich doch in tiefer Kontemplation befindet und zu jedem Zeitpunkt nur eine Zeile gleichzeitig bearbeitet?

- Bei jedem Durchgehen der Anweisungen vergleicht er das erste der beiden Muster. Hat er eine Zeile gefunden, die zum 1. Muster passt, so macht er neben diese Anweisung einen **Vermerk**.
- Die Aktionen einer Anweisung mit *Zeilenbereich* werden nur ausgeführt, wenn sie einen Vermerk tragen (oder keinen Vermerk, falls sie noch mit einem `!` gekennzeichnet sind).
- Bei jedem Durchgehen der Anweisungen vergleicht er die Zeile mit dem 2. Muster und entfernt den Vermerk wieder, wenn sie passt.
- Das Setzen bzw. Entfernen des Vermerks geschieht direkt vor bzw. direkt nach dem Durchführen der Aktionen.

Jede Aktion besteht aus einem oder mehreren **Kommandos**. Ist eine Aktion mit einem Muster verknüpft, so muss das Muster passen, bevor die Aktion durchgeführt werden kann. Folgende Kommandos hat der Abt in seiner Geheimschrift für den Mönch vorgeschrieben:

- `a` [**append**] veranlaßt ihn, *nach* der Abarbeitung aller Anweisungen die zugehörigen Zeilen auszugeben. `i` [**insert**] lässt sie ihn *vor* der Abarbeitung ausgeben und `c` [**change**] lässt ihn die Zeile auf seinem Schmierzettel durch die zugehörigen Zeilen ersetzen und dann ausgeben.
- `y` [**yield**/**yank**] lässt ihn einzelne Buchstaben auf seinem Schmierzettel gegen andere auszutauschen. `s` [**substitute**] lässt ihn bestimmte Teile der Zeile auf seinem Schmierzettel suchen und durch andere Teile ersetzen.
- `r` [**read**] veranlasst ihn, den angegebenen Buchband aus der Bibliothek zu holen und seinen Inhalt in den Folianten einzufügen. `w` [**write**] veranlasst ihn, die Zeile auf seinem Schmierzettel an den Inhalt des angegebenen Buchbandes in der Bibliothek anzuhängen.
- `n` [**next**] lässt den Kopisten den Schmierzettel wegwerfen und eine neue Zeile holen. `N` [**Next**] weist ihn an, sofort eine weitere Zeile zu holen und an die auf dem Schmierzettel stehende anzuhängen. In beiden Fällen macht er mit der nächsten Anweisung weiter.
- Durch `h` [**hold**] kann er angewiesen werden, seinen Schmierzettel auf einen weiteren Schmierzettel zu „kopieren“ und diese Kopie in seine Tasche zu stecken. Durch `H` [**Hold**] hängt er den Schmierzettelinhalt in seiner Hand an den in seiner Tasche an.
- `x` [**exchange**] lässt ihn den Schmierzettel in seiner Tasche mit dem in seiner Hand vertauschen.

- `g` [**get**] lässt ihn den Schmierzettel in seiner Hand wegwerfen und den aus seiner Tasche wieder in die Hand nehmen. `G` [**Get**] lässt ihn die Zeile auf dem Schmierzettel in seiner Tasche an die Zeile auf dem Schmierzettel in seiner Hand anhängen. `P` [**Put**] lässt ihn die erste Zeile auf seinem Schmierzettel in den Folianten schreiben und dann ausradieren.
- Findet er `d` [**delete**], so wirft er den Schmierzettel in seiner Hand weg, holt sich die nächste Zeile und fängt wieder bei der obersten Anweisung in der Liste an. Hat er mehrere Zeilen auf seinem Schmierzettel notiert, so sagt ihm das Kommando `D` [**Delete**], er möge die erste der Zeilen löschen und wieder mit der ersten Anweisung anfangen. War nur eine Zeile auf dem Schmierzettel, holt er sich die nächste Zeile aus dem Manuskript.
- `b` [**branch**] sagt ihm, er soll die Anweisungsliste bis zum angegebenen Lesezeichen durchsuchen und dort weitermachen. `t` [**test**] sagt ihm, er soll dies nur dann tun, wenn er kurz vorher eine erfolgreiche Textersetzung durchgeführt hat.
- `q` [**quit**] sagt ihm, sein Tagwerk ist getan und er kann sich in seine Zelle zum Ausruhen zurückziehen. Das Gleiche ist ihm auch gestattet, wenn er am Ende des zu kopierenden Manuskripts angekommen ist.

Das ist die ganze Geheimsprache und die gehüteten Praktiken, die den Abt und seinen Mönch in die Lage versetzen, mit einfachsten Mitteln eine unglaubliche Vielzahl an Dingen mit ihren alten Handschriften anzustellen. Angeblich sollen damit alle Weisheiten der Welt gefunden werden können, wie immer im Leben gilt jedoch auch hier: „Nur der Beharrliche erreicht sein Ziel“.

4.2 Erklärung

Wandelt man die Analogie wieder in die Sicht auf den Computer um, so ergeben sich folgende Zusammenhänge:

- Das mittelalterliche Kloster ist der Computer.
- Der Abt ist der Anwender.
- Der erste und der dritte Raum in diesem mittelalterlichen Kloster sind die Standard-Eingabe und die Standard-Ausgabe (die Originaldatei wird daher nie verändert).
- Der zweite Raum ist das *Sed*-Skript des Anwenders.
- Der geduldige und sorgfältige Mönch selbst ist das Programm *Sed*.
- Der Schmierzettel in seiner Hand ist der Pattern Space, der Schmierzettel in seiner Tasche ist der Hold Space (er ermöglicht das Speichern eines Zeilenduplikats, während das Original im Pattern Space verändert wird).
- Die geheimnisvollen Symbole (Anweisungen) entsprechen den *Sed*-Kommandos, die `REGEX`-Muster entsprechen den Regulären Ausdrücken und die Lesezeichen entsprechen den Labels (Marken).

- Die Bibliothek ist das Verzeichnissystem des Rechners und die Buchbände sind die Dateien darin.

5 Kurzübersicht Sed-Kommandos

Optionen		
Gawk	**Option**	**Bedeutung**
	`-f FILE`	*Sed*-Kommandos von Datei `FILE` einlesen [**file**]
	`-e CMD`	Ein *Sed*-Kommando `CMD` (zur Angabe mehrerer Kommandos) [**execute**]
	`-n`	Ausgabe nur bei Kommando `p/P` oder Kommando `s` mit Option `p` [**noprint**]
*	`-E/-r`	Extended Regular Expression erlauben [**ERE**]
*	`-i [EXT]`	Angegebene Dateien direkt verändern [**inline**]
		(Backup mit Extension `EXT` erzeugen bzw. bei leerem `EXT` kein Backup)

Exit-Status	
Status	**Bedeutung**
0	Sed-Skript fehlerfrei ausgeführt
$\neq 0$	Während Ausführung trat ein Fehler auf (Syntax)

Kommandos	
Kommando	**Bedeutung**
`a\`	Nachfolgende Zeilen *nach* aktueller Zeile einfügen [**append**]
`ZEILE...`	(*alle Zeilen außer der letzten sind mit* `\` *abzuschließen*)
`b[LABEL]`	Zu Marke `LABEL` (oder Skriptende) springen [**branch**]
`c\`	Aktuelle Zeile durch nachfolgende Zeilen ersetzen [**change**]
`ZEILE...`	(*alle Zeilen außer der letzten sind mit* `\` *abzuschließen*)
`d`	Aktuelle Zeile löschen [**delete**]
`g`	Aktuelle Zeile durch Zwischenpuffer ersetzen [**get**]
`h`	Aktuelle Zeile in Zwischenpuffer kopieren [**hold**]
`i\`	Nachfolgende Zeilen *vor* aktueller Zeile einfügen [**insert**]
`ZEILE...`	(*alle Zeilen außer der letzten sind mit* `\` *abzuschließen*)
`l`	Aktuelle Zeile ausgeben (*Control-Zeichen als ASCII-Code*) [**list**]
`n`	Aktuelle Zeile ausgeben und nächste Zeile einlesen [**next**]
`p`	Aktuelle Zeile ausgeben [**print**]
`q`	Aktuelle Zeile ausgeben und *Sed*-Skript abbrechen [**quit**]
`r FILE`	Inhalt der Datei `FILE` nach aktueller Zeile einfügen [**read**]
`s/REGEX/SUBST/`	In aktueller Zeile `REGEX` durch `SUBST` ersetzen [**substitute**]
`t[LABEL]`	Zu Marke `LABEL` (oder Skriptende) springen [**test**], wenn seit dem letzten Einlesen oder seit dem letzten `t`-Kommando eine Ersetzung erfolgte
`w FILE`	Aktuelle Zeile auf Datei `FILE` ausgeben [**write**]
`x`	Aktuelle Zeile und Zwischenpuffer vertauschen [**exchange**]
`y/abc/ABC/`	In aktueller Zeile Zeichen `a` durch `A`, `b` durch `B` [**yield**/**yank**] und `c` durch `C` ersetzen (analog dem Kommando `tr`)
`D`	Aktuelle Zeile bis zum 1. Newline löschen [**Delete**]
`G`	Zwischenpuffer an aktuelle Zeile anhängen [**Get**]
`H`	Aktuelle Zeile an Zwischenpuffer anhängen [**Hold**]
`N`	Nächste Zeile an aktuelle Zeile anhängen [**Next**]
`P`	Aktuelle Zeile bis zum 1. Newline ausgeben [**Print**] (und löschen)
`:LABEL`	Marke `LABEL` für `b`- oder `t`-Kommando (*max. 7 Zeichen*)
`=`	Nummer der aktuellen Zeile ausgeben
`{`	Kommandos bis zu `}` als Gruppe behandeln (**Block**)

Adressarten	
Adresse	**Bedeutung**
n	Zeilennummer n
`$`	Letzte Zeile
`/REGEX/`	Regulärer Ausdruck, der mit dem Zeileninhalt verglichen wird

Substitute-Optionen	
Option	**Bedeutung**
n	n-tes Auftreten von `REGEX` durch `SUBST` ersetzen [**number**]
`g`	*Jedes* Auftreten von `REGEX` durch `SUBST` ersetzen [**global**]
`p`	Zeile nach *erfolgreicher* Ersetzung ausgeben [**print**]
`w FILE`	Zeile nach der Ersetzung auf Datei `FILE` ausgeben [**write**]

Adressangaben	
Form	**Anwendung auf**
—	Jede Eingabezeile
`ADR`	Jede Zeile, auf die Adresse zutrifft (*die Kommandos* `a`, `i`, `r`, `q` *und* = *akzeptieren nur diese Form!*)
`ADR,ADR`	Von der ersten Zeile, auf die die 1. Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die 2. Adresse zutrifft (*wiederholt sich beliebig oft*)
`ADR!`	Alle Zeilen, auf die Adresse *nicht* zutrifft
`ADR,ADR!`	Alle Zeilen *außer* der ersten Zeile, auf die die 1. Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die 2. Adresse zutrifft (*wiederholt sich beliebig oft*)

BRE-Metazeichen	
Symbol	**Bedeutung**
x	Zeichen x (*kein Metazeichen*)
`.`	1 beliebiges Zeichen
$x*$	0–∞ Wiederholungen des Teils x davor
`^`	Zeilenanfang
`$`	Zeilenende
$\backslash x$	Metazeichen x *quotieren* (steht für sich selbst)
`[`abc`]` `[`a–z`]`	Menge von Zeichen (`[`a–z`]` = **Zeichenbereich**)
`[^`abc`]` `[^`a–z`]`	Negierte Menge von Zeichen (alle außer diesen)
`\(...\)`	Zeichenkette merken (in `\1`..`\9`)
`\n`	Zeilenvorschub [**newline**]
`\/`	Begrenzerzeichen
`\\`	Backslash
$x\backslash\{m,n\backslash\}$	m–n Wiederholungen des Teils x davor
$x\backslash\{m,\backslash\}$	m–∞ Wiederholungen des Teils x davor
$x\backslash\{m\backslash\}$	m Wiederholungen des Teils x davor (genau)

Metazeichen im Ersetzungsteil	
Symbol	**Bedeutung**
$\backslash n$	n-te per `\(...\)` gemerkte Zeichenkette (n=1..9)
`&`	Suchmuster einsetzen (`&` selbst per `\&` angeben)

6 ASCII Tabelle

Der ASCII-Zeichencode definiert die **Standardbelegung** der Codes `0-127` mit Zeichen (kennt keine landesspezifischen Sonderzeichen wie z.B. Umlaute). Die Codes `128-255` werden je nach Zeichensatz unterschiedlich belegt (mit Sonderzeichen wie z.B. Umlauten) und sind hier nicht dargestellt. Die wichtigsten ASCII-Zeichen und ihre Reihenfolge sind:

- **Steuer-Zeichen** (Control) (0–31, *zusammenhängend*)
- **Leerzeichen** (32)
- **Ziffern** `0-9` (48–57, *zusammenhängend*)
- **Großbuchstaben** `A-Z` (65–90, *zusammenhängend*)
- **Kleinbuchstaben** `a-z` (97–122, *zusammenhängend*)
- **Tilde** ~ (126)
- **Druckbare Zeichen** `SPACE-~` (32–127, *zusammenhängend*)

d.h. es gelten folgende **Beziehungen**: `SPACE` < `0-9` < `A-Z` < `a-z` < ~

Dez	0	16	32	48	64	80	96	112	
0	^@ [NUL]	^P	[SPACE]	0	@	P	`	p	**0**
1	^A	^Q	!	1	A	Q	a	q	**1**
2	^B	^R	"	2	B	R	b	r	**2**
3	^C	^S	#	3	C	S	c	s	**3**
4	^D	^T	$	4	D	T	d	t	**4**
5	^E	^U	%	5	E	U	e	u	**5**
6	^F	^V	&	6	F	V	f	v	**6**
7	^G [BEL]	^W	’	7	G	W	g	w	**7**
8	^H [BS]	^X	(	8	H	X	h	x	**8**
9	^I [TAB]	^Y	)	9	I	Y	i	y	**9**
10	^J [LF]	^Z	*	:	J	Z	j	z	**A**
11	^K [VTAB]	^[[ESC]	+	;	K	[	k	{	**B**
12	^L [FF]	^\	,	<	L	\	l	|	**C**
13	^M [CR]	^]	−	=	M	]	m	}	**D**
14	^N	^^	.	>	N	^	n	~	**E**
15	^O	^_	/	?	O	_	o	[DEL]	**F**
	00	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**Hex**

Hinweise:

- `^X` steht für `Ctrl-X` (Control) oder `Strg-X` (Steuerung) und beschreibt die Terminal-Steuerzeichen.
- **Zeichennamen**: `BEL` = Glocke, `BS` = Backspace, `CR` = Carriage Return, `DEL` = Delete, `ESC` = Escape, `FF` = Formfeed, `LF` = Linefeed, `SPACE` = Leerzeichen, `TAB` = Tabulator, `VTAB` = Vertikaler Tabulator.