Der Stream-Editor Sed
Einführung, Tips und Tricks

Version 1.18 — 29.4.2003

Schutzgebühr: 5 Euro

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
eMail: 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 Anweisungen nicht wie z.B. der *Vi* direkt über Benutzereingaben, sondern aus einem **Sed-Skript** und führt diese Anweisungen dann auf den Eingabedaten durch.

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

1.1.1 Vorteile

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

1.1.2 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).

1.1.3 Verwendung

- Suchen und Ersetzen von Texten.
- Automatisiertes Editieren einer oder mehrerer Textdateien.
- Vereinfachte Wiederholung der gleichen Bearbeitung mehrerer Textdateien.
- Erstellung von Konvertierungsprogrammen (z.B. `dos2unix`, `unix2dos`).

1.2 Literatur

- Dougherty, *Sed & Awk, 2. Edition*, O'Reilly & Associates.
- Mortice Kern Systems, *MKS Toolkit Reference Manual - Sed manpage*.
- Gilly, *UNIX in a Nutshell, 2. Edition*, O'Reilly & Associates.

2 Beschreibung

2.1 Aufruf

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

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

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

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

Die zu verarbeitenden Daten werden von der Datei `FILE` eingelesen. Wird keine Datei `FILE...` angegeben, dann liest der *Sed* von der Standard-Eingabe ein. Das Ergebnis wird auf der Standard-Ausgabe ausgegeben. Als Optionen `OPTIONS` sind möglich:

Option	Bedeutung
`-f FILE`	Kommandos von Datei `FILE` einlesen [**file**]
`-e CMD`	Ein *Sed*-Kommando `CMD` (nützlich bei Angabe mehrerer Kommandos) [**execute**]
`-n`	Ausgabe nur bei Kommando `p/P` oder bei Kommando `s` mit Option `p` [**noprint**]

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

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 Anweisungen ist wichtig):

```
s/^  *//
s/  *$//
/^$/d
```

3. Die 3 Kommandos in diesem *Sed*-Skript können auch direkt auf der Kommandozeile in folgender Form angegeben werden:

```
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
s/^  *//
s/  *$//
/^$/d
```

Das Skript ist dann folgendermaßen aufzurufen:

```
rmempty.sed text > newtext
```

Ablauf: Die Shee-Bang-Zeile sorgt dafür, daß der **Unix-Kernel** bei der Ausführung des Skriptes automatisch den *Sed* aus `/bin` startet und das Skript mit Hilfe der Option `-f` an ihn übergibt. Diese Zeile wird vom aufgerufenen *Sed* anschließend ignoriert, da sie von ihm als Kommentar interpretiert wird (fängt im *Sed* mit `#` an).

2.2 Kommandosyntax

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

```
[ADDR [,ADDR]] [!] CMD [ARG]
```

d.h. *Sed*-Kommandos bestehen aus Adressen `ADDR`, der Negation `!`, einem Editier-Befehl `CMD` und einem Argument `ARG`:

- Die **Editier-Befehle** bestehen aus einem einzelnen Zeichen, das als Abkürzung für den Befehlsnamen steht (z.B. `d` [**delete**]).
- Die **Argumente** umfassen die Optionen des Kommandos `s` [**substitute**], die Dateinamen der Kommandos `r` [**replace**] oder `w` [**write**] und die Labels (Marken) der Kommandos `b` [**break**] oder `t` [**test**].

- Eine **Adresse** ADDR kann sein:

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

2.2.1 Adreßangaben

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

Form	Anwendung auf ...
—	Jede Eingabezeile (keine Adresse)
ADDR	Jede Zeile, auf die Adresse zutrifft (die Kommandos a, i, r, q und = akzeptieren nur eine Adresse)
ADDR,ADDR	Die erste Zeile, auf die die erste Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die zweite Adresse zutrifft (*wiederholt sich beliebig oft*)
ADDR!	Alle Zeilen, auf die Adresse *nicht* zutrifft
ADDR,ADDR!	Alle Zeilen *außer* der ersten Zeile, auf die die erste Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die zweite Adresse zutrifft (*wiederholt sich beliebig oft*)

Beispiele für Adreßangaben sind:

–	Alle Zeilen
/BSD/	Alle Zeilen, die den Text BSD enthalten
/[Uu][Nn][Ii][Xx]/	... Unix enthalten (Groß/Kleinschreibung egal)
/^BEGIN/,/^END/	Alle Zeilen zwischen Zeilen, die BEGIN und END am Zeilenanfang enthalten (einschließlich)
/error:/!	Alle Zeilen die *nicht* error: enthalten
/BEGIN/,/END/!	Alle Zeilen *außer* denjenigen zwischen Zeilen, die BEGIN und END irgendwo enthalten (einschließlich)

2.2.2 Trennzeichen

Anstelle des Standard-Trennzeichens / um Reguläre Ausdrücke (siehe Abschnitt 2.2.3 auf Seite 7) in Adressen oder Such- und Ersetzungsmustern kann jedes beliebige andere Zeichen verwendet werden; kommt das Trennzeichen im Regulären Ausdruck vor, so muß es durch \ quotiert werden. Dieses Verhalten ist nützlich, wenn das Trennzeichen / selbst im Regulären Ausdruck mehrfach vorkommt und die dann notwendige Schreibweise \/ ihn schwer lesbar macht.

Beispiel: C++-Kommentare der Form //... durch C-Kommentare der Form /*...*/ ersetzen, einmal unter Verwendung des Standard-Trennzeichens / und einmal der Trennzeichen @ bzw. #.

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

2.2.3 Reguläre Ausdrücke

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

Symbol	Bedeutung
c	Zeichen c (kein Metazeichen)
.	Ein beliebiges Zeichen
.*	0–∞ Wiederholungen des Zeichens . davor
^	Zeilenanfang
$	Zeilenende
$\backslash x$	Metazeichen x *quotieren*
[abc], [a-c]	Menge von Zeichen ([a-z] = Zeichenbereich)
[^abc], [^a-c]	Negierte Menge von Zeichen
\(...\)	Zeichenkette merken (in \1..\9)
\n	Zeilenvorschub [**newline**]
\{m,n\}	m–n Wiederholungen des Teils davor
\{m,\}	m–∞ Wiederholungen des Teils davor
\{m\}	Genau m Wiederholungen des Teils davor

Folgende Metazeichen sind im Ersetzungsteil des Substitute-Kommandos möglich:

Symbol	Bedeutung
$\backslash n$	n-te per \(...\) gemerkte Zeichenkette (n=1..9)
&	Suchmuster einsetzen (& selbst per \& möglich)

Durch \(...\) können auch *mehrere* Musterteile gemerkt und im Ersetzungsteil wiederverwendet werden, diese Klammern können sogar verschachtelt werden. Um die Nummer für die Referenz $\backslash n$ auf den mit einem Klammernpaar gemerkten Musterteil zu ermitteln, sind die öffnenden Klammern \(von links nach rechts durchzunumerieren.

2.2.4 Blöcke

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

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

Achtung: Nach den Klammern dürfen keine Leerzeichen stehen!

2.2.5 Kommentare

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

2.3 Ablauf

2.3.1 Vereinfachte Version

Die prinzipielle Arbeitsweise des *Sed* ist wie folgt:

1. Jede Eingabezeile wird *einzeln* in einen Zeilenspeicher, den sogenannten **Pattern Space** (oder **Puffer**) eingelesen und dort bearbeitet.
2. Die Editierkommandos im *Sed*-Skript werden pro Eingabezeile in ihrer Reihenfolge im Skript auf den Inhalt des Pattern Space angewendet.
3. Sie wirken standardmäßig auf *alle* Eingabezeilen [**global**], durch **Zeilenadressierung** kann ihre Anwendung auf bestimmte Zeilen eingeschränkt werden.
4. Ändert ein *Sed*-Kommando den Inhalt des Pattern Space, dann werden alle noch folgenden *Sed*-Kommandos auf die geänderte Zeile im Pattern Space angewendet, nicht mehr auf die Originalzeile.
5. Nach dem Abarbeiten des letzten Kommandos wird das Ergebnis auf der Standard-Ausgabe ausgegeben (kann in eine Datei umgelenkt werden).

2.3.2 Ergänzungen

1. Die Originaldatei bleibt unverändert, da der *Sed* von der Standard-Eingabe liest und auf die Standard-Ausgabe ausgibt.
2. Bei Angabe des Schalters `-n` [**noprint**] erfolgt nach dem letzten Kommando keine automatische Ausgabe mehr, sondern nur bei den Kommandos `p/P` [**print**] oder beim Kommando `s` [**substitute**] mit der Option `p` [**print**] .

2.4 Kommandos

2.4.1 Grundbefehle

Die Grundbefehle umfassen Einfügen, Anhängen, Ändern und Löschen von Zeilen sowie Ersetzen von Teilen einer Zeile (die durch Reguläre Ausdrücke gematcht werden):

Befehl	Bedeutung
`i\`	Nachfolgende Zeilen *vor* aktueller Zeile einfügen [**insert**] ; alle Zeilen außer der letzten sind mit `\` abzuschließen
`a\`	Nachfolgende Zeilen *nach* aktueller Zeile einfügen [**append**] ; alle Zeilen außer der letzten sind mit `\` abzuschließen
`c\`	Aktuelle Zeile durch nachfolgende Zeilen ersetzen [**change**] ; alle Zeilen außer der letzten sind mit `\` abzuschließen
`d`	Aktuelle Zeile löschen [**delete**]
`s/REGEX/SUBST/`	In aktueller Zeile `REGEX` durch `SUBST` ersetzen [**substitute**]
`y/abc/ABC/`	In aktueller Zeile Zeichen `a` durch `A`, `b` durch `B` [**yield/yank**] und `c` durch `C` ersetzen (analog dem UNIX-Kommando `tr`)

- Ein von den Kommandos `i` und `a` eingefügter Text ist nicht im Pattern Space vorhanden (und durch Kommandos bearbeitbar). Das Ergebnis dieses Kommandos wird erst ausgegeben, wenn die Anweisungsliste vollständig abgearbeitet ist, ganz egal was mit der aktuellen Zeile passiert.
- Das 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.
- Das Kommando `d` liest eine neue Eingabezeile ein und die Verarbeitung beginnt erneut am Anfang der Anweisungsliste.

2.4.2 Das Substitute-Kommando

Die allgemeine Form des *Sed*-Ersetzungs-Kommandos lautet (die Leerzeichen sind nur zu Darstellungszwecken eingefügt, in echten Kommandos sind sie weggelassen):

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

`REGEX` ist ein Regulärer Ausdruck, zu dem eine (oder mehrere) passende Zeichenketten in den durch den Adreßbereich `[ADDR [,ADDR]]` 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 $\backslash n$ auf den mit einem Klammernpaar gemerkten Musterteil zu ermitteln, sind die öffnenden Klammern `\(` von links nach rechts durchzunumerieren.

- `\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, Defaultwert ist 1.
- Maximal 10 Dateien können gleichzeitig zum Schreiben geöffnet werden.

2.4.3 Zeilenausgabe/information

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

Befehl	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 Befehle 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:

Befehl	Bedeutung
`n`	Aktuelle Zeile ausgeben und die 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 Anweisungsliste gesprungen, sondern beim nächsten Kommando weitergearbeitet.
- Das Kommando `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 Sprungbefehle 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.

Befehl	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 das Kommando `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 bei der nächsten Anweisung 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:

Befehl	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 1. `\n`.

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

- Nach den Kommandos `G`, `H`, `N` und `P` wird nicht wieder an den Anfang der Anweisungsliste gesprungen, sondern beim nächsten Kommando weitergearbeitet.
- Nach dem Kommando `D` wird die Verarbeitung wieder am Anfang der Anweisungsliste gestartet. Wird die ganze Zeile gelöscht (weil sie keinen Zeilenvorschub enthält), dann wird analog zum Kommando `d` eine neue Eingabezeile eingelesen.
- `H`- bzw. `G`-Kommando 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: Der Substitutionsbefehl `s/␣*/␣/g` funktioniert nicht! Er fügt vor jedem Zeichen ungleich einem Leerzeichen eines 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]*//
s/[ ^I]*$//
/^$/d
```

- 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\

/^1/s/^1/1^L/
s/^.//
```

3.3 Teil 3

- Aus einer Datei mit positionsorientiertem 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, Anrede, Anrede2 und Bemerkung sind 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 benötigt Form um:

```
1/i
Nachname;Vorname;Anrede;Anrede2;Bemerkung
s/^\(.\)\(.\{20\}\(.\{15\}\)/\2;\3;\1;/
s/  *;/;/g
s/  *$//
s/^\([^;]*;[^;]*;\)1;/\1Herr;Herrn;/
s/^\([^;]*;[^;]*;\)2;/\1Frau;Frau;/
```

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

```
1/i
Nachname;Vorname;Anrede;Anrede2;Bemerkung
s/^1;/HerrHerrn/
s/^2;/FrauFrau /
s/^\(.\{4\}\)\(.\{5\}\)\(.\{20\}\)\(.\{15\}\)/\3;\4;\1;\2;/
s/  *;/;/g
s/  *$//
```

- Die Zeilen der Eingabedatei sind paarweise zu vertauschen, d.h. in der Reihenfolge 2, 1, 4, 3, 6, 5, ... auszugeben (mit `sed -n` ausführen!):

```
h
n
p
g
p
```

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 (Tip: Zwischenpuffer verwenden):

```
/[A-Z]/!d
h
y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz/
H
g
s/\n/ /
s/^/mv /
```

- 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
s/^us/sh/
H
g
s/\n/ /
s/^/mv /
```

- 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 Standardtrennzeichen `/` sehr häufig im Suchmuster vorkommt, wird `@` als Trennzeichen verwendet:

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

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

```
/^#/d                   # Zeile beginnend mit # löschen
:loop
/".*"/!d                # Zeile ohne "..." löschen
h
s/^[^"]*//
s/\([^\\]\)".*/\1/
s/"//
s/\\"/"/g
p
g
s/^[^"]*"//
:rmquote
s/^[^"]*\\"//
trmquote
s/^[^"]"//
bloop
```

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 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 et 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 zu diesem Muster passende Zeile angewendet.
- Folgt auf ein `REGEX`-Muster das Zeichen `!` [**Negation**], so wird die Aktion auf alle Zeilen angewendet, die *nicht* zum Muster passen.
- Sind zwei durch Komma getrennte `REGEX`-Muster angegeben (durch ein Komma getrennt), dann wird die Aktion auf die erste zum ersten Muster passende Zeile und alle folgenden Zeilen bis einschließlich der ersten zum zweiten Muster passenden Zeile durchgeführt (*wiederholt*).
- Folgt auf zwei durch Komma getrennte `REGEX`-Muster das Zeichen `!`, dann wird die Aktion auf allen Zeilen durchgeführt *außer* von der ersten zum ersten Muster passenden Zeile und allen folgenden Zeilen bis einschließlich der ersten zum zweiten 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 zu zwei Mustern passenden *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 ersten 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 das zweite der beiden Muster und entfernt den Vermerk wieder, wenn die Zeile 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:

- Das Kommando `a` [**append**] veranlaßt ihn, *nach* der Abarbeitung aller Anweisungen die zugehörigen Zeilen auszugeben. Das Kommando `i` [**insert**] läßt sie ihn *vor* der Abarbeitung ausgeben und das Kommando `c` [**change**] läßt ihn die Zeile auf seinem Schmierzettel durch die zugehörigen Zeilen ersetzen und dann ausgeben.
- Das Kommando `y` [**yield**/**yank**] läßt ihn einzelne Buchstaben auf seinem Schmierzettel gegen andere auszutauschen. Das Kommando `s` [**substitute**] läßt ihn bestimmte Teile der Zeile auf seinem Schmierzettel suchen und durch andere Teile ersetzen.
- Das Kommando `r` [**read**] veranlaßt ihn, den angegebenen Buchband aus der Bibliothek zu holen und seinen Inhalt in den Folianten einzufügen. Das Kommando `w` [**write**] veranlaßt ihn, die Zeile auf seinem Schmierzettel an den Inhalt des angegebenen Buchbandes in der Bibliothek anzuhängen.
- Das Kommando `n` [**next**] läßt den Kopisten den Schmierzettel wegwerfen und eine neue Zeile holen. Das Kommando `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 das Kommando `h` [**hold**] kann er angewiesen werden, seinen Schmierzettel auf einen weiteren Schmierzettel zu „kopieren“ und diese Kopie in seine Tasche zu stecken. Durch das Kommando `H` [**Hold**] hängt er den Schmierzettelinhalt in seiner Hand an den in seiner Tasche an.
- Das Kommando `x` [**exchange**] läßt ihn den Schmierzettel in seiner Tasche mit dem in seiner Hand vertauschen.
- Das Kommando `g` [**get**] läßt ihn den Schmierzettel in seiner Hand wegwerfen und den aus seiner Tasche wieder in die Hand nehmen. Das Kommando `G` [**Get**] läßt ihn die Zeile auf dem Schmierzettel in seiner Tasche an die Zeile auf dem Schmierzettel in seiner Hand anhängen. Das Kommando `P` [**Put**] läßt ihn die erste Zeile auf seinem Schmierzettel in den Folianten schreiben und dann ausradieren.
- Findet er das Kommando `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.
- Das Kommando `b` [**branch**] sagt ihm, er soll die Anweisungsliste bis zum angegebenen Lesezeichen durchsuchen und dort weitermachen. Das Kommando `t` [**test**] sagt ihm, er soll dies nur dann tun, wenn er kurz vorher eine erfolgreiche Textersetzung durchgeführt hat.

- Das Kommando `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 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 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). 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` < `~`

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

Hinweis:

- `^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.

6 Kurzübersicht

Sed-Optionen	
Option	**Bedeutung**
`-f FILE`	Kommandos von Datei `FILE` einlesen [**file**]
`-e CMD`	Ein *Sed*-Kommando `CMD` (bei Angabe mehrerer Kommandos) [**execute**]
`-n`	Ausgabe nur bei Kommando `p/P` oder bei Kommando `s` mit Option `p` [**noprint**]

Sed-Kommandos	
Befehl	**Bedeutung**
`a\`	Nachfolgende Zeilen *nach* aktueller Zeile einfügen [**append**]; 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**]; 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**]; alle Zeilen außer der letzten sind mit `\` abzuschließen
`l`	Aktuelle Zeile ausgeben (Control-Zeichen im ASCII-Code) [**list**]
`n`	Aktuelle Zeile ausgeben und die 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 ersten 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 ersten 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

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**]

Sed-Adreßarten	
Adresse	**Bedeutung**
n	Die Zeilennummer n
`$`	Die letzte Zeile
`/REGEX/`	Regulärer Ausdruck, der mit dem Zeileninhalt verglichen wird

Sed-Adreßformen	
Form	**Anwendung auf . . .**
—	Jede Eingabezeile
`ADDR`	Jede Zeile, auf die Adresse zutrifft (die Kommandos `a`, `i`, `r`, `q` und = akzeptieren nur eine Adresse)
`ADDR,ADDR`	Die erste Zeile, auf die die erste Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die zweite Adresse zutrifft (wiederholt sich beliebig oft)
`ADDR!`	Alle Zeilen, auf die Adresse *nicht* zutrifft
`ADDR,ADDR!`	Alle Zeilen *außer* der ersten Zeile, auf die die erste Adresse zutrifft bis einschließlich der nächsten Zeile, auf die die zweite Adresse zutrifft (wiederholt sich beliebig oft)

Sed-Metazeichen	
Symbol	**Bedeutung**
c	Zeichen c (kein Metazeichen)
`.`	Ein beliebiges Zeichen
`.*`	0–∞ Wiederholungen des Zeichens `.` davor
`^`	Zeilenanfang
`$`	Zeilenende
`\`x	Metazeichen x *quoten*
`[`abc`]`, `[`a–c`]`	Menge von Zeichen (`[`a–z`]` = Zeichenbereich)
`[^`abc`]`, `[^`a–c`]`	Negierte Menge von Zeichen
`\(...\)`	Zeichenkette merken (in `\1..\9`)
`\n`	Zeilenvorschub [**newline**]
`\{`m`,`n`\}`	m–n Wiederholungen des Teils davor
`\{`m`,\}`	m–∞ Wiederholungen des Teils davor
`\{`m`\}`	Genau m Wiederholungen des Teils davor
`\`n	n-te per `\(...\)` gemerkte Zeichenkette (n=1..9)
`&`	Suchmuster einsetzen (`&` selbst per `\&` möglich)