

Textmanipulationen mit sed

– Eine Einführung –

Julia Stoll
Pieter van den Hombergh

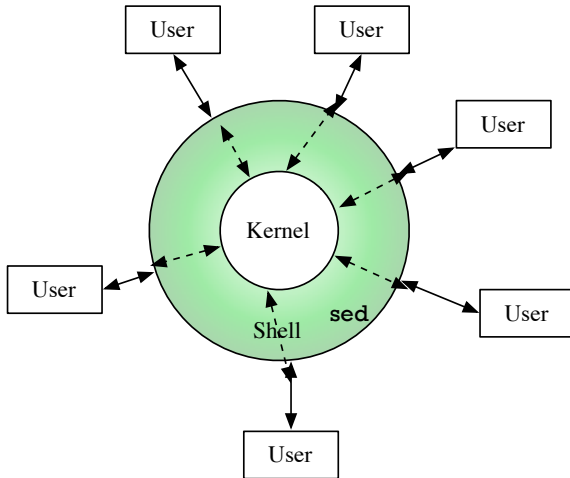
June 10, 2009

- 1 Stream Editor sed
 - Eigenschaften
 - Ein- und Ausgabe (`stdin` und `stdout`)
 - Verarbeitung in und mit sed
- 2 Aufrufsyntax des sed und ihre Bedeutung
- 3 Beispiele
 - Ersetzen mit sed
 - Reguläre Ausdrücke in sed
 - Leerzeichen entfernen
 - Ein komplexes Beispiel
 - Ein noch komplexeres Beispiel
- 4 Übungen
 - Aufgaben
 - Nützliche Quellen zum Arbeiten mit sed
 - Sehr kurze Zusammenfassung

Stream Editor (sed) - Eigenschaften

- sed ist ein nicht-interaktiver zeilenorientierter Editor (der vom Unix Editor ed abgeleitet wurde)
- sed wird zur Textmanipulationen eingesetzt.
- Mit sed lassen sich Daten in Dateien leicht analysieren.
- sed verändert dabei die (Quell-)Daten erstmal nicht.

- Ähnlich wie bei grep (oder auch egrep) lassen sich Daten in Dateien nach bestimmten vorgegebenen regulären Ausdrücken untersuchen.
- *Achtung*: sed kennt nur eine begrenzte Menge von regulären Ausdrücken (im Vergleich zu grep oder egrep).
- sed-Fehlermeldungen sind *wenig* aussagekräftig.
- sed ist aber sehr effizient (wenn auch in manchen Befehlen etwas unkomfortabel).
- Eine Alternative zu sed ist die vielseitige und leistungsfähige Sprache awk.



Eingabe (Editier-) Anweisungen werden

- 1 entweder aus einer Datei oder
- 2 aus der Kommandozeile (command line) und damit aus der Standardeingabe `stdin` eingelesen.

Ausgabe sed schreibt in die Standardausgabe `stdout` und somit werden die Originaldateien nicht verändert; d.h. der Inhalt einer Datei wird nach bestimmten Kriterien nur *temporär* verändert.

Funktionsweise von `sed`

- 1 `sed` liest
 - 1 die vorgegebene/n Eingabedatei/en oder
 - 2 die aktuelle Eingabezeile aus `stdin`.
- 2 Die aktuelle Eingabezeile wird in den Eingabepuffer (*space pattern*) übertragen.
- 3 `sed` prüft, welche Editieranweisungen für diese Zeile auszuführen sind.
- 4 Anweisungen werden ausgeführt.

- 5 Danach
 - 1 löscht sed den Inhalt des Eingabepuffers oder
 - 2 schiebt den Inhalt in den Zwischenspeicher (*hold buffer*), wenn z.B. mit einem Redirect oder einer Pipe gearbeitet werden.
- 6 Wenn sich in der zu manipulierenden Datei weitere (Eingabe-) Zeilen befinden, wird in Punkt 2 die Verarbeitung fortgesetzt.
- 7 sed beendet die Verarbeitung wenn
 - 1 entweder alle Befehlszeilen abgearbeitet sind oder
 - 2 eine explizite Beendigungsanweisung gegeben wird.

Erste Beispiele zur Funktionsweise von `sed`

Eingabedatei: `Steden.txt`

Amsterdam
Amersfoort
Den Haag
Eindhoven
Groningen
Maastricht
Nijmegen
Rotterdam
Utrecht
Tilborg
Venlo

Einfache Textmanipulationen

- 1 `sed -n '/aa/p' Steden.txt`
- 2 `sed -n '/a/p/' Steden.txt`
- 3 `sed -n '/[aa]/p' Steden.txt`
- 4 `sed -n '/A/p' Steden.txt`
- 5 `sed -n '/^A/p' Steden.txt`

```
sed -n '/aa/p' Steden.txt
```

Den Haag
Maastricht

Das sind die Städte, die in ihren Namen zwei aufeinanderfolgende *Kleinbuchstaben* 'a' enthalten.

sed -n '/a/p' Steden.txt

Amsterdam
Den Haag
Maastricht
Rotterdam

Das sind die Städte, die in ihren Namen einen *Kleinbuchstaben* 'a' enthalten.

Was ist mit dem folgenden Befehl?

```
sed -n '/[a*]/p' Steden.txt
```

```
sed -n '/[aa]/p' Steden.txt
```

Amsterdam
Den Haag
Maastricht
Rotterdam

Warum?

sed -n '/A/p' Steden.txt

Amsterdam
Amersfoort

Was ist mit?

```
sed -n '/^A/p' Steden.txt
```

Warum liefern in *diesem Fall* die beiden Befehle dasselbe Ergebnis?

Aufrufsyntax

```
sed [-n] [-e script] [-f scriptfile] [inputfile(s)]
```

Bedeutungen der Optionen

- Editieranweisungen für die Eingabedateien werden sed durch ein sed-Skript vorgegeben. Dieses Skript kann
 - ① entweder beim Aufruf des sed in Form von `-e script` (üblicherweise: `-e 'script'`) angegeben werden oder
 - ② in einer Skriptdatei stehen, deren Name mit `-f scriptfile` auf der Kommandozeile angegeben sind.
 - ③ Beim Aufruf von mehreren Skripten können die Optionen `-e` und `-f` auch kombiniert werden.
- Die Option `-n` unterdrückt die Ausgabe in `stdout`; mit einem expliziten `print`-Befehl `p` kann diese unterdrückte Ausgabe (aus dem Haltepuffer, `hold buffer`) wieder zu (einem späteren Zeitpunkt) ausgegeben werden.

sed -e 's/xyz/abc/' file.txt zum Ersetzen

Eingabedatei: Steden.txt

- `sed -e 's/Am/Bn/' Steden.txt`
 - `s` steht für *substitute*
 - `Am` wird durch `Bn` ausschließlich in der Ausgabe `stdout` ersetzt.
 - `/` dient als Trenner (Seperator) für die zu untersuchenden Zeichenketten.
- `sed -e 's/Am/Bn/' Steden.txt > GagSteden.txt`
 - Was wissen wir von letzter Woche?

Reguläre Ausdrücke in sed

x ein Zeichen, das vorkommen muss

. ein beliebiges Zeichen

[abc] eines der Zeichen a, b oder c

[^abc] ein Zeichen auer a, b und c

* vorheriger Ausdruck darf beliebig oft oder gar nicht vorkommen

^ Anfang der Zeile

\$ Ende der Zeile

(REGEXP) - gruppiert den inneren Ausdruck, kann in Substitutionen benutzt werden, um zu Präferenzen zu setzen.

Bedeutung von Flags beim s-Befehl

Flag	Bedeutung	Beispiel	Bemerkung
N	N=Zahl (1 ; 512) ; das N'te Vorkommen	s/AB/CD/2	wird N nicht angegeben, dann erste Vorkommen in der Zeile
g	global ; alle Vorkommen	s/AB/CD/g	es werden alle Vorkommen in der Zeile ersetzt
p	print ; Ausgabe bei Ersetzung	s/AB/CD/p	wurde eine Ersetzung vorgenommen, wird der Buffer an stdout ausgegeben
w FILE	write FILE bei Ersetzung	s/AB/CD/w FILE	bei Ersetzung Ausgabe des Buffers in die angegebene Datei
i	case-insensitive ; Gro-Kleinschreibung ignorieren	s/AB/CD/i	ist nicht kompatibel zu allen sed-Implementierungen

Leerzeichen entfernen

```
#!/bin/sed -nf
```

```
H  
$ {  
  x  
  s/\n//g  
  p  
}
```

Zeilennummerierung

```
# Number each line of a file  
# (simple left alignment)  
# Using a tab instead of space will  
# preserve margins.
```

```
sed = filename | sed 'N;s/\n/\t/'
```

Was ist das Problem mit diesem Beispiel?

Print directory tree starting at the current node

```
#!/bin/sh
# Create variable equal to selected directory
# (or current if none specified)
dir=${1-.}
# Change to the appropriate directory
(cd $dir; pwd)
find $dir -type d -print | sort -f | sed -e"
    s:^$1::
    /^$/d
    /^\.$/d
    s:[^/]*/\([^\/*\])$|-----\1:
    s:[^/]*/:|      :g"
```

Übungen

- 1 Studieren Sie die gegebenen Beispiele.
- 2 Ersetzen die Leerzeichen durch ein anderes Symbol, z. B. mit \$; was müssen Sie bzgl. sed beachten?
- 3 Entfernen Sie Leerzeichen aus einer Datei (anstatt sie zu verdoppeln).
- 4 Schreiben Sie Programm, das nur Zeilen nummeriert, die *nicht* leer sind.
- 5 Modifizieren Sie die Ausgabe des Verzeichnisbaumes, so dass ausschließlich Verzeichnisse (und keine anderen Dateien) angezeigt werden.
- 6 Nutzen Sie sed, um die aktuellen Werte der Umgebungsvariablen (Stichwort `env`) auszugeben und ggf. zu modifizieren.

Nützliche Quellen

Online manual <http://www.gnu.org/software/sed/manual/>

Many useful “little“ commands

<http://sed.sourceforge.net/sed1line.txt>

Introduction and Tutorial

<http://www.grymoire.com/Unix/Sed.html>

Script Archive <http://sed.sourceforge.net/grabbag/scripts/>

Sehr kurze Zusammenfassung

sed dient

der **Textmanipulation** ohne dabei per se die Daten dauerhaft zu verändern.

sed kann angewendet werden

direkt auf Dateien im Dateisystem des Betriebssystems (everything is file),

zur Substitution von Zeichenketten mittels regulärer Ausdrücke (es gibt natürlich weitere eingebaute Funktionen als s),

um Skripte zu erstellen, die (komplexere) Funktionen realisieren und

um Skripte aus Skriptdateien umzusetzen. Damit lassen sich z.B. make-Files und andere wiederkehrende (Kontroll-)Funktionen implementieren.