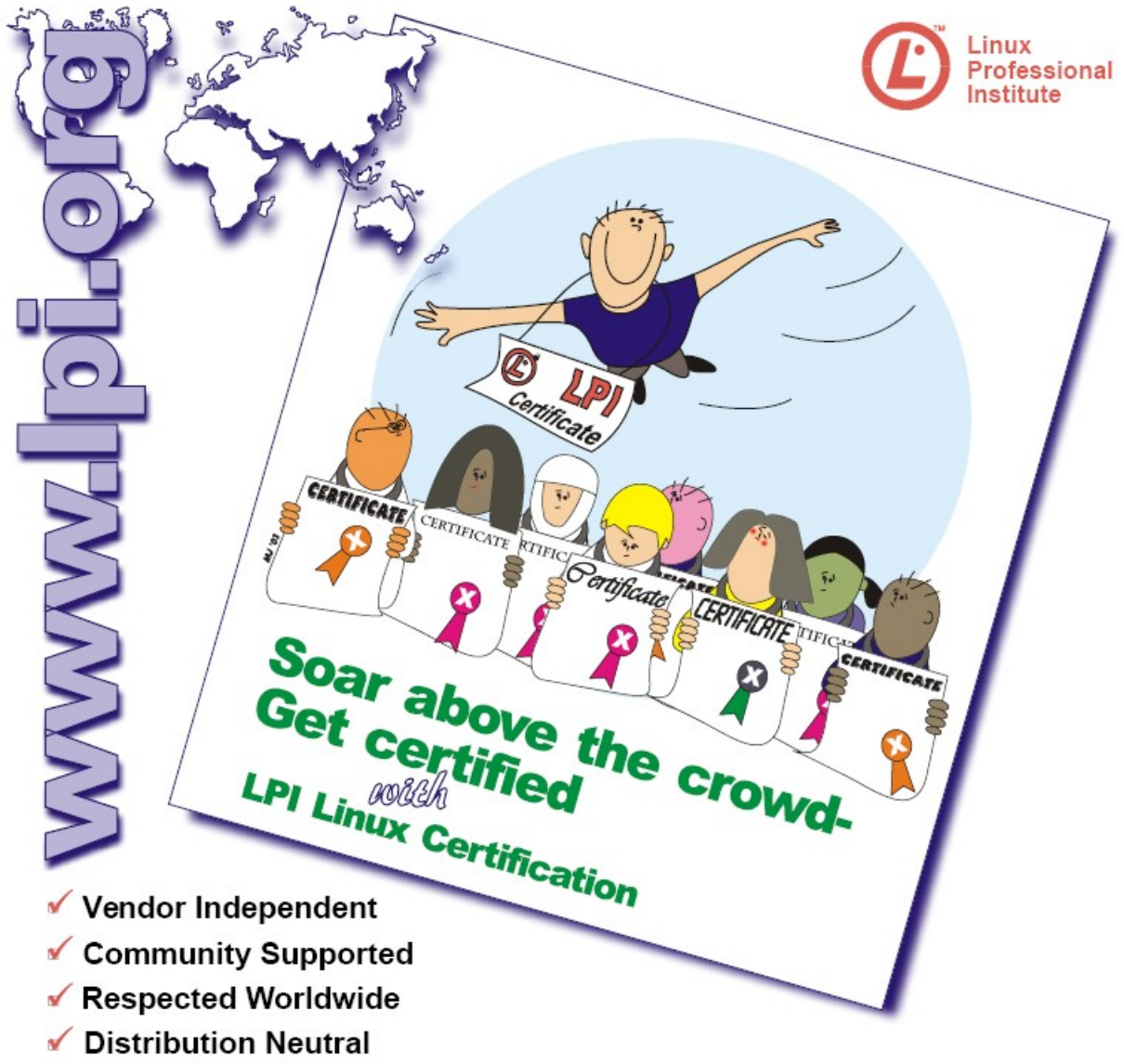

1.103.7 Reguläre Ausdrücke

Durchsuchen und manipulieren von Textdateien mit regulären Ausdrücken

Beschreibung: Prüfungskandidaten sollten in der Lage sein, (Text-)Dateien mittels regulärer Ausdrücke zu bearbeiten. Dieses Lernziel beinhaltet das Erzeugen einfacher regulärer Ausdrücke mit verschiedenen Elementen. Ebenfalls enthalten ist die Verwendung von Regex-Werkzeugen zum Durchführen von Suchen über ein Dateisystem oder Datei-Inhalte.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **grep (egrep, fgrep)**
- **sed**

Grundprinzip

- Reguläre Ausdrücke (Regular Expressions , Regexp, Regex, RE) sind verwandt mit den Wildcards, aber nicht das Selbe. Es handelt sich um Suchmuster, die auch eine Art Jokerzeichen beinhalten, jedoch wesentlich mächtiger sind, als die Wildcards der Shell.
- Die unterschiedlichsten Programme (grep, sed, awk, php, perl...) unter Linux können mit regulären Ausdrücken umgehen, allerdings unterscheiden sie sich manchmal im Detail in der Interpretation. Es gibt aber eine gemeinsame Grundmenge von Ausdrücken, die von allen Programmen verstanden werden.

  Varianten:

  - BRE (Basic Regular Expressions) - Einfache Reguläre Ausdrücke
  - ERE (Extended Regular Expressions) - Erweiterte Reguläre Ausdrücke

Muster und RE

- Unter einem regulären Ausdruck versteht man ein Muster, das eine bestimmte Menge von Zeichenketten beschreibt. Reguläre Ausdrücke werden ganz ähnlich aufgebaut wie arithmetische Ausdrücke, indem man sie mit Hilfe verschiedener Operatoren aus kleineren Ausdrücken zusammensetzt. **grep** versteht zwei verschiedene Klassen regulärer Ausdrücke: "gewöhnliche" und "erweiterte".
- Das Prinzip der RE ist von den Mustern (engl: Patterns) her bekannt. Einfache Beispiele:

  echo * (listet alle Dateien im aktuellen Verzeichnis auf)

  echo a* (alle Dateinamen, die mit kleinem „a“ beginnen)
- RE sind flexibler und mächtiger als die Muster.
- RE müssen nicht auf das ganze Wort bzw. die ganze Zeile passen, es reicht wenn eine passende Zeichenkette enthalten ist.

Syntax

- Reguläre Ausdrücke und Muster sind wie zwei verschiedene Sprachen zu behandeln. Viele Zeichen finden in beiden Sprachsystematiken Verwendung, allerdings mit unterschiedlichen Bedeutungen. Deshalb muß man aufpassen, um sie nicht zu verwechseln.
- Damit die Shell diese nicht verwechselt, sollte man ihr bei regulären Ausdrücken sagen, dass sie diese nicht (als Muster) interpretieren, sondern unverändert an das Programm übergeben soll (das RE beherrscht). Dazu maskiert man sie durch Einschließen in Hochkommata.
  - `grep '^root' /etc/passwd` (^Zeilenanfang)
  - `grep '^[^r].*' /etc/passwd` ([^Negation])

Syntax

- **Literale** normaler Text, das Zeichen **m** steht eben für ein **m**
- **Metazeichen** Zeichen besonderer Bedeutung, z.B. ? oder .
- **Positonszeiger** für die genaue Position, an der ein Zeichen gesucht wird **^,$**
- **Zeichenfolge** Anzahl von Zeichen, die Text findet [a-z] [0-9] [^Negation]
- **Quantifizierer** modifizieren die Anzahl an Zeichen, die gefunden wird.

?, ., +, *, {n,m}

Literale und Steuerzeichen

- Die Suchmuster bestehen aus **Literalen und Steuerzeichen**. Dabei sind Literale Zeichen, die genau das Zeichen erkennen, das sie selber darstellen. Die Steuerzeichen werden für verschiedene Aufgaben verwendet. Dazu gehören die Angabe von Zeichenklassen, Untermustern, Alternativen und sonstigen besonderen Funktionen.
- Das wohl wichtigste Steuerzeichen ist der **Backslash '\'.** Er dient als Escape-Zeichen, das einem anderen Steuerzeichen vorangestellt wird und bewirkt, dass dieses als Literal gewertet wird.
- Zur Angabe von mehreren Möglichkeiten verwendet man das **Alternativen-Steuerzeichen Pipe** '|'.
- Weitere Steuerzeichen sind die **runden Klammern '(' und ')'**. Diese verwendet man, um Untermuster anzulegen, bzw. Musterteile zu gruppieren.
- Mit **eckigen Klammern '[' und ']'** kann eine Menge von Zeichen angegeben werden, aus der das nächste gelesene Zeichen stammen muss. Es handelt sich hierbei also um eine vereinfachte Schreibweise einer Menge von Alternativen der Länge 1. (Zeichenklasse)
- Wenn man ein beliebiges Zeichen erkennen will, kann man den Punkt '.' verwenden. Dieser erkennt jedes beliebige Zeichen als gültig an (außer NewLine '\n').

Darstellung für RE

BRE	ERE	Bedeutung
xy	xy	Ein 'x' gefolgt von einem 'y'
.	.	Ein beliebiges Zeichen
[xyz]	[xyz]	Ein 'x' oder ein 'y' oder ein 'z' (Zeichenklasse)
[a-z]	[a-z]	Ein beliebiges kleines Zeichen (in diesem Fall das Alphabet)
[^xyz]	[^xyz]	Ein beliebiges Zeichen, außer 'x', 'y' und 'z'
a\{2,5\}	a{2,5}	zwei bis fünf mal 'a' (hintereinander)
a\{2,\}	a{2,}	zwei mal 'a' oder öfter
a\{2\}	a{2}	genau zwei mal 'a'
a*	a*	beliebig oft 'a' (also auch kein mal)
a\+	a+	mindestens ein mal 'a'
a\?	a?	höchstens ein mal 'a'
\(...\)	(...)	Klammern legen die Reihenfolge der Operationen fest
a|b	a|b	entweder 'a' oder 'b' (a und b können auch zusammengesetzte Ausdrücke sein)
^	^	Zeilenanfang (am Anfang des Ausdrucks)
$	$	Zeilenende (Am Ende des Ausdrucks)
\<	\<	Wortanfang
\>	\>	Wortende
\b	\b	Wortanfang oder -ende
\. * \[\] \+ \?		Jeweils das Zeichen '.', '*', '[', ']', '+' , '?'

Übung 1 (Postleitzahlen finden)

- Dieser Ausdruck sucht nach fünfstelligen Zahlen, wie zum Beispiel Postleitzahlen. [0-9] findet eine Ziffer zwischen 0 und 9. Durch den Quantifizierer {5} dahinter muss etwas Derartiges fünfmal hintereinander gefunden werden. Die Wortgrenzen \< und \> sorgen schließlich dafür, dass um diese 5 Ziffern nichts herum stehen darf.

Lösung:

```
egrep '\<[0-9]{5}\>' regexp.txt (Übungsdatei)
```

Übung 2 (E-Mailadressen finden)

- Ausdruck zum Aufspüren von E-Mail-Adressen. \b bedeutet Wortanfang oder -ende. Durch das ^ zu Beginn der eckigen Klammerung wird es verneint. So wird von obigem Ausdruck alles gefunden, was aus einem @ besteht, das links und rechts von mindestens einem Wortzeichen umrahmt wird. (der \ vor dem @ ist übrigens nicht Pflicht, aber in vielen Programmen hat das @ eine Sonderbedeutung)

```
Lösung:

egrep '[^\b]\@[^\b]' regexp.txt

(Allerdings findet die RE auch Begriffe wie,
  Intr@net, M@us die ein '@' enthalten) Optimieren!
```

Übung 3 (IP-Adressen finden)

- Ausdruck für einen einfachen IP-Adressen-Finder. Er sucht dreistellige Zahlen, die durch drei Punkte getrennt sind. Zur Verbesserung wäre noch eine Begrenzung auf ein Wort per \< und \> möglich.

Lösung:

```
egrep '([0-9]{3}\.){3}[0-9]{3}' regexp.txt
```

Problematik? 333.555.999.666 ist keine IP-Adresse!!!! 10.1.1.1 wird nicht gefunden!

Optimieren, dass wirklich nur gültige IP-Adressen gefunden werden? (Bereich 0.0.0.0 - 255.255.255.255)

Übung 4 (IP-Adressen optimiert)

- Dieser IP-Adressen-Finder ist eine optimierte Version des vorhergehenden Ausdrucks. Er könnte vielleicht in einem Skript eingesetzt werden, um Benutzereingaben zu überprüfen.

'\<(([01]?[0-9]{1,2}|2[0-4][0-9]|25[0-5])\.){3}([01]?[0-9]{1,2}|2[0-4][0-9]|25[0- 5])\>'

- Der Ausdruck lässt nur gültige IPv4-Adressen zu. Ausnahme: 0.0.0.0 ist keine gültige IP-Adresse, wird aber trotzdem anerkannt. Auch Sonderadressen (z.B. für Broadcasts) werden zunächst erkannt. Um dies zu verhindern kann der Ausdruck entsprechend angepasst werden.

- Die Optimierung: Es wurden \< und \> ergänzt, damit beispielsweise ag1234.122.33.4.012sd9 nicht gefunden wird. Anschließend wurde das [0-9]{3} durch eine präzisere Variante ersetzt. Sie besteht aus drei Alternativen: [01]?[0-9} {1,2} findet alle ein- und zweistelligen Zahlen und dreistellige, die mit 1 beginnen. Jeweils eingeschlossen sind auch die Entsprechungen mit führenden Nullen (auffüllend bis drei Stellen). Es sind jetzt also alle Zahlen von 0 bis 199 abgedeckt. 2[0-4][0-9} findet alle Zahlen von 200 bis 249. 25[0-5] alle von 250 bis 255. Damit ist die Suche komplett.

Übung 5 (Doppelte Worte finden)

- Oft übersieht man beim Tippen, dass ein Wort **doppelt doppelt** eingegeben wurde. Nun wollen wir Prüfen ob dies bei unserem Text der Fall ist!

  Lösung:

  egrep -i '\b([a-z]+) +\1\b' regexp.txt

  Mit dem Schalter „i“ sucht grep per „**ignore case“,** das heißt es findet beispielsweise die Wortfolge **Das das**

  grep „merkt“ sich den ersten gefundenen Ausdruck in der Klammer (hier: „Das“). Nach der Klammer kommt -wie zwischen Wörtern üblich- mindestens ein Leerzeichen (+) und dann die Metasequenz die auf das gefundene Wort passt \1 danach die Wortgrenze \b.

sed (stream editor)

- Wer viele Textdateien unter Linux bearbeitet, wird schnell die Vorteile der Skript-Sprachen SED (und AWK) kennen lernen. Es sind wertvolle Werkzeuge, um Texte effizient automatisch zu manipulieren.
  Vorteil: man spart Zeit und vor allem sichert der konsequente Einsatz auch eine weitestgehende Konsistenz der Daten.
- **sed** besitzt etwa 25 Befehle und ist im Gegensatz zu anderen Editoren nicht interaktiv, sondern wird von der Kommandozeile aufgerufen oder mit Skript-Dateien, die Editieranweisungen enthalten, genutzt.
- sed liest die Eingabedatei(en) Zeile für Zeile und prüft, welche Editieranweisungen für diese Zeile ausgeführt werden sollen. Das Ergebnis schreibt sed auf die Standadausgabe.

Adressen in sed

Adressen sind entweder Zeilennummern oder "regular expressions".

/test/	erfüllt durch test, test1, atest2, ...
/[0-9]/	erfüllt durch jeden String, der eine Zahl enthält
/[Uu]nix/	erfüllt durch einen String, der unix oder Unix enthält
/x.y/	der Punkt steht für einen beliebigen einzelnes Zeichen
/[^abcd]/	erfüllt durch einen String, der kein a,b,c oder d enthält
/^abc/	erfüllt durch einen String, der am Zeilenanfang mit abc beginnt
/abc$/	erfüllt durch einen String, der am Zeilenende mit abc aufhört
/^$/	leere Zeile (auch keine Leerzeichen)
/^ *$/	leere Zeile mit einer beliebigen Anzahl Leerschlägen (auch Null)
/^.$/	erfüllt durch eine Zeile mit genau einem Zeichen
(no address)	erfüllt durch alle Zeilen
n (n=integer)	n-te Zeile
$	letzte Zeile

Instruktionen in sed

Folgende Instruktionen werden von sed erkannt (unvollständig):

a\	Append. Fügt Text an die aktuelle Zeile an. Jede Zeile (außer die letzte) muss mit \ abgeschlossen werden.
c\	Change. Ersetzt die aktuelle Zeile durch neuen Text.
d	Delete. Die aktuelle Zeile wird gelöscht und die nächste Zeile eingelesen.
i\	Insert. Fügt Text vor der aktuellen Zeile ein.
n	Next. Nächste Zeile einlesen.
p	Print. Schreibt die aktuelle Zeile sofort auf stdout (ohne nachfolgende Instruktionen auzuführen).
q	Quit.
r	Read. Einlesen des Inhalts eines spezifizierten Files und anschliessendes Anfügen an die aktuelle Zeile.
s	Substitut. Die Substitutionsinstruktion hat dieselbe Form wie in vi. [Adresse] s/Muster/Ersatzmuster/[n][g][p][w file]
y	Ersetzt jeden Charakter in str1 durch den entsprechenden in str2. [Adresse] y/str1/str2/

sed: Ganze Zahl n und $

Testen Sie einige Funktionen von sed am Beispiel der Datei „staedte.txt“

```
(1)sed -n '2p' staedte.txt
(2)sed -n '$p' staedte.txt
(3)sed -n '3,6p' staedte.txt
(4)sed -n '5,$p' staedte.txt
(5)sed -n 'p' staedte.txt
```

Die Option **-n** unterdrückt die automatische Ausgabe jeder bearbeiteten Zeile. **p** (print) bewirkt die Ausgabe des Eingabepuffers

sed: Einfache Zeichen und Metazeichen

```
(1)sed -n '/o/p' staedte.txt
(2)sed -n '/o/,$p' staedte.txt
(3)sed -n '3,/o/p' staedte.txt
(4)sed -n '/o/,/b/p' staedte.txt
(5)sed -n '/g$/p' staedte.txt
(6)sed -n '/r.$/p' staedte.txt
(7)sed -n '/b[ue]rg$/p' staedte.txt
(8)sed -n '/^[^B-K][^o-z]/p' staedte.txt
```

sed mit einfachen Script-Dateien verwenden

- erzeugen und speichern Sie eine einfache Textdatei (Name: a1.sed)

```
/^[1-9][1-9]*\.[^1-9]/a\
==========\


/^[1-9][1-9]*\.[1-9]/a\
------------------
```

- Rufen Sie sed mit der Datei **a1.sed** auf. Wenden Sie die Datei auf die Textdatei „sed_ueb.txt“ an. Machen Sie sich klar was dabei geschieht!

Aufruf:

```
sed -f a1.sed sed_ueb.txt
```

sed: Bereiche der Datei löschen

```
(1)sed -e '1,7d' sed_ueb.txt
(2)sed -e '5,$d' sed_ueb.txt
(3)sed -e '2,/ie/d' sed_ueb.txt
(4)sed -e '/^$/d' /boot/grub/menu.lst
(5)sed -e '/^#/d' -e '/^$/d' /boot/grub/menu.lst
(6)...
```

sed: Ersetzungen

```
(1)sed -e 's/i[eo]/--/' sed_ueb.txt

(2)sed -e 's#i[eo]#--#' sed_ueb.txt
                        (selbe Wirkung!)

(3)sed -e '3,$s/[aeiou]/-/g' sed_ueb.txt
```

Aufgaben

(1) Überlegen Sie eine sed-Anweisung, die ihren Eintrag (Username) aus /etc/passwd auf dem Bildschirm ausgibt.

(2) Schreiben Sie eine sed-Anweisung, die allen Zeilen der Datei buch 5 Punkte voranstellt

(3) Ersetzen Sie mit sed die 2. Zeile in der Datei buch mit einem Text

```
zu (1) sed -n /`whoami`/p /etc/passwd

zu (2) sed -e 's/^/...../' sed_ueb.txt

zu (3) sed -e '2cZweite neue Zeile' sed_ueb.txt
```

Literatur

- Jeffrey Friedl, Reguläre Ausdrücke, O'Reilly 2000
- www.selflinux.org
- Helmut Herold, awk & sed, Addison-Wesley 2003