Betriebssysteme

Labor 4

Regulärer Ausdruck

- = eine endliche Zeichenfolge, die ein Suchmuster definiert (ist eine Folge von Zeichen, in dem einige Zeichen besondere Bedeutung haben)

- **`eine Übereinstimmung (a match)`** = ein einzelnes Zeichen, eine Folge von Zeichen, eine Folge von Bytes, ein Stück Text

- sehr seltsame und hässliche, aber kompakte und flexible Sprache für passenden Text

Regulärer Ausdruck

- **`die Sonderzeichen`** (Metazeichen/ meta-characters):

`. period (dot)`	`$ dollar sign`	`* asterix (star)`	`) closing parenthesis`
`\ backslash`	`| vertical bar`	`+ plus sign`	`[ square bracket`
`^ caret`	`? question mark`	`( opening parenthesis`	`{ curly brace`

- diese Sonderzeichen haben in regulären Ausdrücken eine andere Bedeutung
- man muss diese maskieren (mit **`\ Backslash`**), um ihre eigene reguläre Bedeutung wiederherzustellen

Regeln für reguläre Ausdrücke

- **`.`**: entspricht einem einzelnen Zeichen
- **`\`**: Escape ändert die Bedeutung des darauf folgenden Zeichens zwischen normal und speziell
- **`\.`**: der . (Punkt) Zeichen
- **`[abc]`**: entspricht jedem einzelnen Zeichen, das in der Liste angezeigt wird (in diesem Fall `a` oder `b` oder `c`)
- **`[^abc]`**: ein einzelnes Zeichen **AUßER** denen in eckigen Klammern (`d, e, ..., z`)
- **`[a-z]`**: entspricht einem einzelnen Zeichen, das zum Bereich gehört (in diesem Fall einem Kleinbuchstaben)

Regeln für reguläre Ausdrücke

- **`[A-Z]`**: ein einzelner Großbuchstabe von A bis Z (beliebiger Großbuchstabe)
- **`[a-zA-Z]`**: ein einzelner Klein- oder Großbuchstabe
- **`[0-9]`**: eine einzelne Ziffer von `0` bis `9`
- **`[^0-9]`**: ein einzelnes Zeichen, das **KEINE Ziffer** ist
- **`\d`**: eine einzelne Ziffer von `0` bis `9` (entspricht `[0-9]`)
- **`\s`**: ein einzelnes Leerzeichen (einschließlich `SPACE`, `TAB`, `CR`, `LF`)
- **`\w`**: ein einzelnes alphanumerisches Zeichen oder `_` (Unterstrich)
- **`\(\)`**: eine Gruppe einfangen

Anker (anchors)

- **^**: der Zeilenanfang
- **$**: das Zeilenende
- **\<**: die leere Zeichenfolge am Anfang eines Wortes
- **\>**: die leere Zeichenfolge am Ende eines Wortes
- **\b \b**: Äquivalent zu \< \>

Wiederholungsoperatoren:

- **?**: entweder null oder einmal
- *****: null oder mehrmals
- **+**: ein- oder mehrmals
- **{n}**: genau n mal
- **{n,}**: n mal oder mehr
- **{,m}**: höchstens m mal
- **{n,m}**: mindestens n mal, aber höchstens m mal

Aufgabe

Geben Sie ausgehend von den oben dargestellten Aspekten die unten dargestellten Ausdrücke an:

1. `.*`
2. `[a-zA-Z02468]`
3. `[ ,]`
4. `^[^0-9]+$`

GREP

- steht für **g**lobal **r**egular **e**xpression **p**rint
- sucht in einer oder mehreren Dateien nach einer bestimmten Muster und zeigt das Ergebnis in der Standardausgabe an.

```
grep [ -[[AB] ]num ] [ -[CEFGVBchilnsvwx] ] [ -e ] regulären_Ausdruck
[-f Name-Szenario] [ Dateiliste ]
```

```
    grep [OPTIONS] PATTERN [FILE...]
    grep [OPTIONS] [-e PATTERN] [-f FILE...] [FILE...
```

Optionen

- **`-c (--count):`**
  - Trefferanzahl anzeigen
  - die Anzahl aller Zeilen, die das Suchmuster beinhalten, kann man mit `-c` oder `--count` ausgeben lassen.
  - damit liefert `grep` die reine Anzahl, also z.B. “`4`” für `4` Treffer zurück, sonst nichts.

Beispiel:

```
grep -c 'abc' datei.txt

grep --count 'abc' datei.txt
```

Optionen

- **`-i (ignore case)`**
  - Groß- und Kleinschreibung ignorieren
  - standardmässig sucht der **grep-Befehl** *case-sensitive*. Das heisst, die Groß- und Kleinschreibung wird beachtet. Will man dies, also *case-insensitive*, suchen, muss man die Option -i hinzufügen.

Beispiel:

```
grep -i 'abc' datei.txt
```

Optionen

- **`-l:`**
  - nur Dateinamen ausgeben lassen
  - manchmal braucht man nur den/die Dateinamen, in denen der Suchstring gefunden wurde. Das kann man über die Option `-l` (kleines “`L`”) erreichen.

Beispiel:

```
grep -l 'a' *.txt
```

Optionen

- **`-L:`**
  - zeige gefundene, NICHT matchende Dateinamen
  - auch gibt es Anwendungsfälle, in denen man nur die Dateinamen aufgelistet haben will, in denen der Suchstring NICHT gefunden wurde. Dafür nutzt man die Option `-L`.

Beispiel:

```
grep -L 'a' *.txt
```

Optionen

- **`-n:`**
  - Zeilennummer für jedes Match
  - wird vor jeder Match die Zeilennummer des Matches geschrieben

Beispiel:

```
grep -n 'abc' datei.txt
```

Optionen

- **`-v`:**
  - NICHT betroffene Zeilen anzeigen
  - zeigt alle Zeilen an, die die angegebene Zeichenfolge nicht enthalten

Beispiel:

```
grep -v 'abc' datei.txt
```

Optionen

- **`-w`:**
  - nur ganze Wörter finden
  - per default findet **grep** alle Vorkommen des Suchstrings, sowohl in Wortteilen als auch in ganzen Wörtern. Nur auf Vorkommen in ganzen Wörtern beschränkt man per `-w`.

Beispiel:

```
grep -w 'abc' datei.txt
```

Optionen

- **-h (hide):**
  - zeigt den Dateinamen nicht an

Beispiel:

```
grep -h 'a' *
```

Optionen

- **`-E`:**
  - verwenden erweiterte reguläre Ausdrücke
  - ist wie grep, nur mit dem Unterschied, dass man auch `+`, `?`, `|` und `()` als Regular Expression nutzen kann.

Beispiele

- Alle nicht-leeren Zeilen anzeigen:

```
grep "." datei.txt
```

- Alle leeren Zeilen anzeigen:

```
grep "^$" datei.txt
```

- Alle Zeilen anzeigen, die eine ungerade Anzahl von Zeichen enthalten:

```
grep "^\(..\)*.$" datei.txt
```

etc/passwd Datei

In Unix und Unix-ähnlichen Betriebssystemen ist die Datei `/etc/passwd` eine Textdatei, die Informationen über die Benutzerkonten im System enthält. Die Datei hat eine bestimmte Struktur, wobei jede Zeile Informationen über ein einzelnes Benutzerkonto enthält. Die Felder in jeder Zeile sind durch Doppelpunkte (`:`) getrennt und die Reihenfolge der Felder ist festgelegt.

Die Struktur der einzelnen Zeilen in der Datei `/etc/passwd` sieht folgendermaßen aus:

```
username:password:uid:gid:full_name:home_dir:shell
```

etc/passwd Datei

Wo:

- `username`: der Name des Benutzerkontos.
- `password`: ein verschlüsseltes Passwort oder ein `x`, wenn das Passwort in einer separaten Datei (z. B. /etc/shadow) gespeichert ist.
- `uid`: die Benutzer-ID (eine Zahl), die mit dem Benutzerkonto verbunden ist.
- `gid`: die primäre Gruppenkennung (eine Zahl), die mit dem Benutzerkonto verbunden ist.
- `full_name`: zusätzliche Informationen über den Benutzer, wie z. B. sein vollständiger Name.
- `home_dir`: das Heimatverzeichnis des Benutzers.
- `shell`: Das Standard-Shell-Programm des Benutzers.

Beispiele

Wir beabsichtigen, die Datei `users.txt` zu nutzen, welche eine Struktur aufweist, die derjenigen der Datei `/etc/passwd` entspricht.

- Alle Zeilen anzeigen, die "`David`" enthalten

```
grep "David" users.txt
```

- Zeigt die Zeilen aller Benutzer an, die keine Ziffern in ihrem Benutzernamen haben.

```
grep "^[^0-9]\+:" users.txt
```

- Alle Zeilen anzeigen, die in ihrem Benutzernamen mindestens zwei Vokale aufweisen. Diese Aufgabe ist ein wenig tricky, da die Vokale nicht aufeinanderfolgen müssen. Daher müssen wir alle Zeichen zwischen den Vokalen (einschließlich keiner) zulassen. Gleichzeitig dürfen jedoch keine Doppelpunkte zwischen den Vokalen erscheinen, da wir uns ansonsten außerhalb des Benutzernamens befinden würden.

```
grep -i "^[^:]*[aeiou][^:]*[aeiou][^:]*:" users.txt
grep -i "^[^:]*\([aeiou][^:]*\)\{2,\}:" users.txt
```

SED (**S**tream **ED**itor)

- ist ein nicht interaktiver Texteditor, der verwendet wird, um grundlegende Texttransformationen an einem Eingabestrom durchzuführen
- anstatt Änderungen direkt an der Datei vorzunehmen, erzeugt man zunächst eine temporäre Datei, deren Inhalt anschließend an die Ausgangsdatei übergeben wird. Jede Zeile einer Datei wird einzeln eingelesen, bearbeitet und dann wieder ausgegeben. (Standardmäßig wird die Datei nicht geändert, sondern das Ergebnis der Verarbeitung der Eingabedatei angezeigt)

- *Die wichtigste Funktion:* bestimmte Zeichenketten in der Datei zu suchen und dann durch andere Zeichen zu ersetzen.

SED (Stream EDitor)

```
sed [-n] [-e] '[/pattern/]command' [input-file]
sed [-n] -f script-file [input-file]
```

-n: automatisches Drucken des internen Puffers (Patternspace) unterdrücken

-e *script*: Skript zu den auszuführenden Befehlen hinzufügen

-f *script-file*: Fügt den Inhalt der Skriptdatei den auszuführenden Befehlen hinzu

- Der Eingabestrom kann sein: der Standard-Eingabestrom (Tastatur), eine durch Eingabedatei bezeichnete Datei oder das Ergebnis einer anderen Befehlsausführung
- wenn kein Muster, eine bestimmte Zeile oder mehrere Zeilen angegeben sind, wird der Befehl für alle Zeilen des Eingabestreams ausgeführt

Bedingungen

- Ein sed-Szenario besteht aus Linien der Form:

```
Bedingung Anweisung
```

  - **`keine Bedingung`:** wahr für alle Zeilen in der Datei
  - **`n`:** wahr für die Zeilen mit dem Zeilennummer n (Zeilen sind in der Dateiliste kumulativ nummeriert)
  - **`$`:** wahr für die letzte Zeile in der Datei
  - **`/ regulären Ausdruck /`** : wahr für die Zeilen, die mindestens einen Teilstring enthalten, der dem regulären Ausdruck entspricht
  - **`expr1,expr2`:** wahr für die Zeilen, die zwischen der Zeile, die mit Ausdruck1 übereinstimmt, und der Zeile, die mit Ausdruck2 übereinstimmt

Beispiele

```
$ sed 1,10 Anweisung fis1
```

(*führt die Anweisung in den Zeilen 1 bis 10 aus*)

```
$ sed 10,$ Anweisung fis1 fis2
```

(*führt die Anweisung in den Zeilen von 10 bis zum Ende der Datei aus, die durch Verketten von fis1 mit fis2 erhalten wird*)

Beispiele

Sei “`phone.txt`”-Datei:

```
(555)555-1212
(555)555-1213
(555)555-1214
(666)555-1215
(666)555-1216
(777)555-1217
```

```
cat phone.txt | sed 's/\(.*\)\(.*-\)\(.*$\)/Area code: \1 Second: \2 Third: \3/'
```

```
Area code: (555) Second: 555- Third: 1212
Area code: (555) Second: 555- Third: 1213
Area code: (555) Second: 555- Third: 1214
Area code: (666) Second: 555- Third: 1215
Area code: (666) Second: 555- Third: 1216
Area code: (777) Second: 555- Third: 1217
```

Suchen / Ersetzen

```
sed -E "s/regex/replacement/flags" fis.txt
```

- **`s`** - ist der Such- / Ersetzungsbefehl
- **`/`** ist das Trennzeichen und kann ein beliebiges anderes Zeichen sein. Das erste Zeichen nach dem Befehl s wird als Trennzeichen betrachtet
- Die Flags am Ende können g, i oder beides sein
  - **`g`** - führt den Ersatz überall in der Linie. Ohne sie wird nur das erste Erscheinungsbild ersetzt
  - **`i`** - Führt eine Suche ohne Berücksichtigung der Groß- und Kleinschreibung durch
- Die Ersetzung kann Verweise auf die Ausdrücke enthalten, die in der Regex als `\1`, `\2` usw. gruppiert sind, wobei die Nummer die Reihenfolge ist, in der die Gruppen in der Regex erscheinen

Transliterate

```
sed -E "y/characters/replacement/" fis.txt
```

- **`y`** - ist der Transliterationsbefehl
- **`/`** ist das Trennzeichen und kann ein beliebiges anderes Zeichen sein. Das erste Zeichen nach dem Befehl y wird als Trennzeichen betrachtet
- Die *characters* und der *replacement* müssen gleich lang sein

Zeilen löschen, die einem regulären Ausdruck entsprechen

```
sed -E "/regex/d" fis.txt
```

- **/** ist das Trennzeichen
- **d** - ist der Befehl zum Löschen von Zeilen

Aufgaben

Lassen Sie uns mit dem Inhalt von `users.txt` arbeiten:

1. Zeigen Sie alle Zeilen an und ersetzen Sie alle Selbstlaute (Vokale) durch Leerzeichen.
2. Zeigen Sie alle Zeilen an und konvertieren Sie alle Selbstlaute (Vokale) in Großbuchstaben
3. Zeigen Sie alle Zeilen an und löschen Sie die Zeilen mit fünf oder mehr Ziffern
4. Zeigen Sie alle Zeilen an und vertauschen Sie alle Buchstabenpaare
5. Zeigen Sie alle Zeilen an und duplizieren Sie alle Vokale