htw saar

Studiengang Kommunikationsinformatik (Bachelor)
Studiengang Praktische Informatik (Bachelor)
Prof. Dr.–Ing. Damian Weber
Dipl.-Phys. Michael Meßner

Security Engineering

3. Übung

Aufgabe 1 (NTP)

Das Network-Time-Protocol (NTP) wird zum Setzen der Systemzeit verwendet. Typischerweise ist dies für das Booten eines Systems und zu periodisch wiederkehrenden Zeitpunkten konfiguriert.

Ein nützliches Tool ist in diesem Zusammenhang `ntpdate`. Gegebenenfalls müssen Sie `ntpdate` auf Ihrem Rechner installieren. Schauen sie sich die Manualpage von `ntpdate` an und finden Sie die beiden Optionen heraus, die verhindern, dass

- durch `ntpdate` die Systemzeit verändert wird und
- `ntpdate` Debug-Ausgaben der Kommunikation mit dem NTP-Server liefert

Kontaktieren Sie mit dem Tool alle NTP-Server von

- unserem Hochschulrechenzentrum, z.B. `ntp1.hiz-saarland.de`
- der physikalisch-technischen Bundesanstalt, z.B. `ptbtime1.ptb.de`

Wieviele offiziell erreichbare NTP-Server gibt es unter diesen Domains?

Finden Sie zehn weitere offiziell erreichbare NTP-Server.

Aufgabe 2 (Pipes in der Kommandozeile)

Wir untersuchen Filterprogramme, diese sind geeignet für Pipe–Operationen. Ein Programm ist ein Filterprogramm, wenn es eine Eingabe von Standardeingabe liest und die Ausgabe auf Standardausgabe schreibt. In dieser Aufgabe werden verschiedene Filter eingeübt.

Das Anwendungsbeispiel ist das Extrahieren von Daten aus einer HTML–Datei. Nun also die verwendeten Tools anhand von Beispielen (die nicht unbedingt genauso zum Lösen der Aufgabe eingesetzt werden müssen). Ausgangspunkt ist dieser Link:

```
https://www.fussballdaten.de/bundesliga/tabelle/
```

Wir speichern einen Ausschnitt davon, der hier bereitsteht

```
https://www-crypto.htwsaar.de/weber/download/
  2025_05_06_bundesliga-snippet-4a31ac318a.html
```

Speichern Sie diese lokal als `fussball-tabelle.html`

- **Stream–Editor sed:**

  kann Dateien automatisiert editieren. Er liest einen Datenstrom von Standardeingabe, verändert ihn und schreibt den veränderten Datenstrom auf Standardausgabe.

  Wir experimentieren hierzu mit einer Datei zur Fußball-Bundesliga Tabelle aus Wikipedia.

  Beispiel: um alle `<th>` Tags in `<td>` Tags zu verwandeln geben Sie ein (eine Zeile)

  ```
  sed -e "s:<th\>:<td\>:g" -e "s:</th>:</td>:g"
         <fussball-tabelle.html >fussball-tabelle2.html
  ```

  Schauen Sie sich die Datei `fussball-tabelle2.html` in einem Editor an.

- **Differenzen von Dateien: diff**

  Überprüfen können Sie die Ersetzungen mit dem `diff` Kommando.

  Geben Sie folgendes ein:

  ```
  diff -u fussball-tabelle.html fussball-tabelle2.html
  ```

  Die Differenz wird als entfernte(–)/hinzugefügte(+) Zeile angezeigt.

- **Suchen von Mustern: grep, fgrep, egrep**

  Finden Sie mit `egrep -n` heraus, in welcher Zeile der Verein *Bayern München* gelistet ist.

  Jetzt koppeln wir zwei Filterprogramme, nämlich `grep` und `sed`. Versuchen Sie durch Verwendung des Pipe–Symbols `|` die Ausgabe eines `fgrep` Kommandos mit der Eingabe von `sed` zu koppeln, sodass nur noch die Namen der Fußballvereine und ihre Platzierung sichtbar sind. Das Suchmuster für `sed` ist ein regulärer Ausdruck. Beispielsweise können Sie folgende Suchmuster benutzen

  ```
  .* für eine beliebige Zeichenkette
  [abc]* für eine beliebige aus a b c bestehende Zeichenkette
  [a-z]* für eine beliebige aus Kleinbuchstaben bestehende Zeichenkette
  ... (weiteres z.B.unter
  https://www.gnu.org/software/sed/manual/html_node/Regular-Expressions.html)
  ```

Die Aufgabe ist gelöst, wenn Sie eine Textdatei der Form

```
1. FC Bayern München
2. Borussia Dortmund
3. 1. FC Union Berlin
4. RB Leipzig
5. SC Freiburg
6. VfL Wolfsburg
...
```

produzieren können.

Aufgabe 3 (Shell–Programmierung)

Grundsätzlich ist bei den folgenden Shell–Skripten folgendes zu beachten:

- Beginn mit Hashbang und Bourne-Shell `#!/bin/sh`
- Ende mit Exit–Code
  - exit 0 falls erfolgreich
  - exit 1 falls Fehler
- Fehlerbehandlung, falls ein aufgerufenes Programm einen Fehler hatte
  (die Variable `$?` enthält den Exit-Code des aufgerufenen Programms)

a) Schreiben Sie ein Shell–Skript, das vor jedes Argument den String ,,Hallo" setzt. Beispiel:

```
./hallo Peter Stefan Michael
Hallo Peter
Hallo Stefan
Hallo Michael
```

b) Schreiben Sie ein Shell–Skript `viewer`, der abhängig vom Art des Inhalts einer angegebenen Datei ein entsprechendes Programm zum Anzeigen der Datei aufruft. Falls die Datei eine Grafikdatei ist, soll beispielsweise `/usr/local/bin/xv` aufgerufen werden.

Die Unterscheidung der Inhaltstypen von Dateien können Sie treffen, indem Sie `file` aufrufen, wie im folgenden Beispiel:

```
$ file tomate.jpg
tomate.jpg: JPEG image data, JFIF standard 1.01 ...
```

Unterscheiden Sie mindestens Bilddateien (`xv`), PDF-Dateien (`xpdf`), Textdateien (`less`) und Open-Document Texte (`libreoffice`).
Bemerkung: die Verwendung des Utilities `open(1)` ist hierfür nicht erlaubt (nicht verwechseln mit dem `open(2)` Systemcall). Da dieses Utility eine sehr ähnliche Funktionalität hat, wäre der Lerneffekt nicht gegeben.

c) Schreiben Sie ein Shell–Skript `wavtomp3`, das WAV-Dateien in MP3–Dateien umwandelt. Hierfür können Sie `ffmpeg` benutzen.

d) Schreiben Sie ein Shellskript, das das Kommando `which` emuliert, siehe Manualpage `which(1)`. Ihr Shellskript soll die in der Umgebungsvariable `PATH` genannten Pfade durchgehen und feststellen, ob das gesuchte Programm in einem der Directories ausführbar gespeichert ist.