Universität Potsdam
Institut für Informatik
Wintersemester 2011–2012

Grundlagen von Betriebssystemen

Aufgabenblatt 6

9 Einige weitere UNIX-Werkzeuge

1. Kopieren Sie die Dateien `/home/rlehre/adressen` und `/home/rlehre/bdressen` in Ihr Arbeitsverzeichnis. Sehen Sie sich die Dateiinhalte an.

2. Das Kommando `sort` sortiert die Zeilen der als Argumente angegebenen Dateien, in der Standardeinstellung des Kommandos in alphabetischer Reiehnfolge. Ist kein Argument angegeben, so wird `stdin` sortiert.

   (a) Sortieren Sie Ihre Tastatureingaben: Rufen Sie `sort` ohne Argument auf und geben Sie einige Zeichenketten über die Tastatur ein. Beenden Sie die Eingabe mit `^D`.

   (b) Führen Sie `sort bdressen adressen` aus. Bezieht sich die alphabetische Sortierung auf die Dateinamen? _____

   (c) Sie wollen die Zeilen von `adressen` alphabetisch nach den Namen sortieren. Dazu können Sie mit den Optionen `-k`n oder `-k`n, m angeben, dass sich das Sortieren auf die Spalten n bzw. n bis m bezieht. Dabei bestehen Spalten aus Zeichenketten, die durch *Whitespaces*, d.h. Leerzeichen, Tabulator-Stops und/oder Zeilenumbrüchen, voneinander getrennt sind. Die Numerierung der Spalten beginnt mit 1.
   Probieren Sie folgende Kommandos und beschreiben Sie deren Wirkung mit eigenen Worten.

   `sort -k2 adressen` _____

   `sort -k2,2 adressen` _____

   Welchen Unterschied beobachten Sie in den Ausgaben dieser beiden Kommandos?

   Erklären Sie. _____

   _____

   (d) Erklären Sie mit Hilfe der Manual-Seite von `sort` die Ausgabe von

   `sort -n bdressen` _____

   (Welche Ausgabe erwarten Sie ohne die Option `-n`?)

   (e) Sortieren Sie die Ausgabe von `who` alphabetisch nach Benutzernamen.
   Hinweis: Benutzen Sie den Pipeline-Mechanismus.

   Kommando: _____

   Verwenden Sie die Option `-u`, um jeden Benutzer nur einmal aufzuführen.
   Lesen Sie die Bedeutung der Option in der Manual-Seite nach!
   Benutzen Sie dieses Kommando zusammen mit dem Pipeline-Mechanismus, um die Anzahl der zur Zeit an Ihrem System angemeldeten Benutzer zu bestimmen.

   Kommando: _____

(f) In `/etc/passwd` sind die Spalten durch Doppelpunkte voneinander getrennt. In der 6. (vorletzte) Spalte sind die Login-Verzeichnisse der in dieser Datei verwalteten Benutzer festgelegt. Lassen Sie sich den Inhalt dieser Datei auf das Terminal ausgeben, wobei nach den Login-Verzeichnissen alphabetisch zu sortieren ist.
Hinweis: Für die Einstellung auf den Spaltentrenner seperator *ist die Option* `-t` seperator *zu benutzen. Kontrollieren Sie anhand der Ausgabe Ihres Kommandos.*

Kommando: _____

3. Filtern Sie eine Ausgabe nach Spalten und Zeilen.
Vielleicht möchten Sie die Ausgabe des letzten Kommandos auf die Login-Verzeichnisse beschränken. Dazu müssten Sie angeben, welche Spalten zur Ausgabe gehören sollen und welche nicht. Ebenso können Sie mit Zeilen verfahren.

(a) Erklären Sie mit eigenen Worten die Wirkung folgender Kommandos:

`head -2 adressen` _____

`tail -2 adressen` _____

`cut -f 3 adressen` _____

`cut -f 2,3 adressen` _____

`cut -c 1-5 adressen` _____

`cut -c 1-5, 8-13 adressen` _____

(Probieren Sie ggf. auch andere Parameter der Optionen `-f` und `-c`.
Bechten Sie, dass der Tabulator-Stop als ein Character behandelt wird.)

(b) Lassen Sie sich mit einem Kommando genau die dritte und vierte Zeile von `adressen` ausgeben (*Pipeline?!*). Kommando: _____

(c) Finden Sie mit Hilfe der Manual-Seite von `cut` und durch Probieren ein Kommando, mit dem Sie die Spalte aus `/etc/passwd`, die die Login-Verzeichnisse enthält (6. Spalte), allein in dem Terminalfenster ausgeben können.

Kommando: _____

(d) Lassen Sie sich jetzt diese Ausgabe alphabetisch sortiert anzeigen, wobei kein Verzeichnis mehr als einmal aufgeführt wird.

Kommando: _____

4. Erschließen Sie mit Hilfe der Manual-Seite von `paste` die Funktionen dieses Kommandos.

(a) Fügen Sie die Dateiinhalte von `adressen` und `bdressen` mit der Standardeinstellung von `pate` zusammen.

Kommando: _____

Wie ist die Ausgabe formatiert (zeilen- oder spaletenweise Anordnung? Trennsymbole?)

_____

_____

(b) Jetzt sollen die Zeilen der ursprünglichen Dateien jeweils durch einen Doppelpunkt getrennt in zwei Spalten nebeneinander erscheinen.

Kommando: _____

(c) Jetzt sollen die ursprünglichen Dateiinhalte untereinander angeordnet werden.

Kommando: _____

(d) Fügen Sie eine neue erste Spalte mit `paste` an die Zeilen der Datei `adressen` an, die Sie über die Satndardeingabe eingeben, und fangen Sie die so modifizierten Zeilen in der Datei `list` auf.
Hinweis: *Benutzen Sie das Symbol* `-` *als erstes Argument von* `paste`.

Kommando: _____

Fügen Sie nun diese Zeilen als neue letzte Spalte an die Zeilen von `adressen` an.

Kommando: _____

(e) Erklären Sie die Funktion des Kommandos `ls -1 | paste - -`. Warum passiert das?

_____

_____

5. Das Kommando `tr` *folge1 folge2* übersetzt die Zeichen aus *folge1* in die entsprechenden Zeichen aus *folge2*. Das Kommando ist ein reiner **Filter**, d.h. es wird stets aus `stdin` gelesen und nach `stdout` geschrieben. Gegebenenfalls muss also mit `<` bzw. `>` umgelenkt werden. Probieren Sie an Zeilen, die Sie über die Tastatur eingeben:

```
tr aeiou iouae
```

Wenden Sie jetzt diese Transformation auf den Dateiinhalt von `adressen` and und fangen Sie die transformierte Ausgabe in einer neuen Datei `cdressen` auf.

Kommando: _____

Wichtige Optionen sind:

- `-d` löscht alle Zeichen, die in *folge 1* genannt sind (dann nur ein Argument von `tr`!)
- `-s` reduziert aufeinanderfolgende gleiche Zeichen auf ein einzelnes.

(a) Finden Sie heraus, dass die Ausgabe von `ls -l` mit Hilfe von Leerzeichen (`' '`) formatiert wird, indem Sie alle Leerzeichen in dieser Ausgabe durch ein Prozentzeichen ersetzen. (*Denken Sie daran, dass man bestimmte Zeichen in der Shell maskieren muss.*)

Kommando: _____

(b) Wiederholen Sie dieses Kommando, aber jetzt soll an jeder Trennstelle nur jeweils ein `%` erscheinen.

Kommando: _____

(c) Jetzt sollen alle Leerzeichen in dieser Ausgabe gelöscht werden.

Kommando: _____

10 Shell-Programmierung

Wir wollen die Funktionalität der Shell (möglichst gut) wie eine Programmiersprache gebrauchen. Eine Kommunikation mit dem Benutzer ist z.B. über das `echo`*-Kommando möglich. Ein typisches Merkmal von höheren Programmiersprachen ist die Verwendung von Variablen. Sie haben die Möglichkeit, (lokale) Shell-Variablen zu setzen und abzufragen, bereits kennen gelernt.*

6. Rechnen Sie mit ganzzahligen Variablenwerten (**Shell-Arithmetik**).

(a) Verwendet man beim Anlegen/Überschreiben einer Shell-Variablen anstelle von `set` das Kommando `@` so wird (falls möglich) die Zeichenkette rechts vom `=` als numerischer Ausdruck ausgewertet und dem Variablennamen links vom `=` der Wert des Ausdrucks zugewiesen. Probieren Sie (und achten Sie dabei auf die Leerzeichen!!):

`@ var = (3 + 4) * 5; echo $var` Ausgabe: _____

`@ var++; echo $var` Ausgabe: _____

`@ div = $var / 5; echo $div` Ausgabe: _____

`@ mod = $var % 5; echo $mod` Ausgabe: _____

(b) Was ist also die Bedeutung folgender Operatoren?

`++` _____ `/` _____ `%` _____

Probieren Sie diese Operatoren mit weiteren Zahlen aus.

7. Arbeiten Sie mit **Wortlisten**.

(a) Führen Sie folgende Kommandos nacheinander aus. Sehen Sie alle Ausgaben genau an.

```
> set inhaltsverzeichnis = (kapitel1 kapitel2 kapitel3 kapitel4)
> echo $bdir
> echo $bdir[2]              Ausgabe: _____
> echo $bdir[1-3]
> set bdir[2] = chapter2
> echo $bdir
> echo $#bdir                Ausgabe: _____
```

(b) Sehen Sie sich den Wert der lokalen Systemvariablen `path` in der Ausgabe von `set` an. Es handelt sich auch um eine Wortliste.

i. Lassen Sie sich das zweite Verzeichnis in Ihrem Suchpfad nach ausführbaren Dateien ausgeben. Kommando: _____

ii. Wieviele Verzeichnisse werden nach ausführbaren Dateien durchsucht? (*Hinweis: Verwenden Sie* `$#`). Kommando: _____

8. Arbeiten Sie mit Konstrukten zur Wiederholung von Anweisungen (**Schleifen**).

(a) Führen Sie aus und notieren Sie in eigenen Worten, was bewirkt wird.

`repeat 3 pwd` _____

```
foreach wortlistenwanderer (eins zwei drei vier)
foreach? echo $wortlistenwanderer
foreach? pwd
foreach? end
```
_____

```
foreach i (2 3 5 7 11)
foreach? @ quadrat = $i * $i
foreach? echo Das Quadrat von $i ist $quadrat.
foreach? end
```
_____

```
set c = 0; foreach i (alpha beta gamma delta epsilon)
foreach? @ c++
foreach? echo $c. Durchlauf:
foreach? echo $i
foreach? end
```
_____

```
set c = 0; foreach i (alpha beta gamma delta epsilon)
foreach? @ c++
foreach? echo -n $c. Durchlauf:
foreach? echo $i
foreach? end
```
_____

(b) Wozu dient die Option `-n` des `echo`-Kommandos?

(c) Können Sie die Ausgabe mit apostrophierten Leerzeichen noch schöner formatieren?!

(d) Lassen Sie sich mit einer `foreach`-Schleife die Verzeichnisse auflisten, die nach ausführbaren Dateien durchsucht werden. Beachten Sie, dass beim Zugriff auf eine Wortliste mit dem `$`-Mechanismus die Shell die runden Klammern entfernt. Kommando(stapel):

9. *Im Gegensatz zur* `foreach`*-Schleife prüft die* `while`*-Schleife vor jedem Schleifendurchlauf, ob eine bestimmte Bedingung (Boolescher Ausdruck) erfüllt ist. Die Anweisungen in der Schleife werden solang wiederholt, bis die Bedingung zu* `false` *evaluiert. Es kann auch eine Endlosschleife entstehen. Brechen Sie dann den Prozess mit* `^C` *ab.*

   (a) Setzen Sie die lokale Variable `i` auf den Wert `1`.
   Führen Sie dann aus (wobei die angegebenen Leerzeichen nicht weggelassen werden dürfen):

   ```
   while ($i < 10)
   while? echo $i
   while? @ i++
   while? end
   ```

   Erläutern Sie die Funktionsweise der `while`-Schleife an diesem Beispiel.

   (b) Welchen Wert hat die Shellvariable `i` jetzt? ___

   (c) Setzen Sie den Wert von `i` auf `1` zurück und wiederholen Sie dieses Beispiel, lassen Sie aber diesmal die Anweisung `@ i++` im Schleifenkörper weg.
   Was passiert und warum?

   (d) Lassen Sie sich mit einer `while`-Schleife die Verzeichnisse, die in der Wortliste der Shellvariablen `path` gespeichert sind, untereinander ausgeben.
   Hinweis: Benutzen Sie `$#path`. Kommando(stapel):

10. Führen Sie bedingte Anweisungen aus.

    (a) Führen Sie aus und notieren Sie in eigenen Worten, was bewirkt wird.

    `if (1 < 2) echo ja` ___

    `if (1 > 2) echo ja` ___

(b) Es gibt zur Realisierung von **Verzweigungen** auch eine `if-then-else`-Anweisung, die aber erst in Shell-Skripten ausprobiert werden soll. Die Syntax finden Sie in der Datei `Shell-Programmierung.pdf`, die die Inhalte zu diesem Thema zusammenfasst und ergänzt. Laden Sie sich diese Datei herunter und arbeiten Sie sie durch.

11. Sehen Sie sich **Bedingungen** etwas genauer an. Welche Ausgabe erhalten Sie und warum?

```
if (0) echo ja
if (1) echo ja
if (5) echo ja
if ($home == $HOME) echo ja
if ($home == $HOME && -f adressen) echo ja
if ($home == $HOME && -d adressen) echo ja
if ($home == $HOME || -d adressen) echo ja
if ({ pwd }) echo ja
```

12. Beim letzten Beispiel wird der **Exit-Status** des Kommandos als Bedinguing benutzt (s. `Shell-Programmierung.pdf`). Der Exit-Status des zuletzt ausgeführten Kommandos wird in der lokalen Variablen `status` gespeichert. Probieren Sie:

`grep root /etc/passwd; echo $status` Exit-Status: _____

`grep susu /etc/passwd; echo $status` Exit-Status: _____

`grep root /etc/shadow; echo $status` Exit-Status: _____

Verwenden Sie diese Kommandos als `if`-Bedingungen.

13. Man kann den Exit-Status auch benutzen, Kommandos mit UND- und ODER-Bedingungen zu verknüpfen. Unter welchen Bedingungen wird das zweite Kommando nach einer UND-Verknüpfung bzw. nach einer ODER-Verknüpfung ausgeführt? Probieren Sie dazu folgende Verknüpfungen:

```
grep root /etc/passwd && pwd
grep susu /etc/passwd && pwd
grep root /etc/passwd || pwd
grep susu /etc/passwd || pwd
```